

Worcester Polytechnic Institute

## Automatic Data Analysis Library for ASSISTments

As Interactive Qualifying Project submitted to the Faculty of WORCESTER POLYTECHNIC INSTITUTE in partial fulfillment of requirements for the Degree of Bachelor of Science.

By

Junbong Jang

Date

March 9<sup>th</sup>, 2019

Advisors:

Korin S. Ostrow

Neil T. Heffernan

## Table Of Contents

Abstract .....	4
Acknowledgements.....	4
1. Introduction.....	5
1.1 ASSISTments Background.....	5
1.2 Purpose .....	6
1.3 Theoretical Framework.....	7
1.3.1 Programming with Python .....	7
1.3.2 Software Engineering.....	9
1.3.3. Database.....	10
2. Methods.....	11
2.1 Software Design .....	11
2.2 Data Analysis Examples.....	12
2.3 Design of the Library .....	13
2.4 Web Application.....	15
2.5 Setting up ALI project in your Local Machine.....	16
2.5.1 Install plugins on Eclipse.....	16
2.5.2 Download Repository using Subversion .....	16
2.5.3 Build Project using Gradle .....	17
2.5.4 ALI Doc generation.....	17
2.6 Python Library Integration to ALI.....	18
2.6.1 Analysis function.....	18
2.6.2 Troubleshooting MANOVA .....	19
2.7 ALI Project Structure .....	20
2.8 ALI Doc generation.....	22
2.8.1 Data Preparation.....	22

2.8.2 Population of data inside the ALI Doc Template.....	23
2.9 ASSISTments Template Tool.....	24
2.9.1 Template Explanation .....	24
2.9.2 Connecting to the Database .....	27
3. Results.....	30
3.1 Web Application Final Version .....	30
3.2 Python Data Analysis .....	32
3.2.1 Multiple Linear Regression Output.....	32
3.2.2 Multiple Linear Regression Output on HTML5 Webpage.....	33
3.2.3 Limiting the number of decimal points.....	35
3.2.4 Data Preprocessor .....	37
3.2.5 Generic Analysis Function.....	38
3.3 ALI Doc Report .....	40
4. Discussion.....	43
4.1 Hardship.....	43
4.2 Limitations & Future Improvements .....	45
4.2.1 Web Application.....	45
4.2.2 Python Data Analysis Library.....	46
4.2.3 ALI Document Report.....	47
4.2.4 Future.....	48
References .....	49
Appendix A. Original ALI Doc Sample.....	51
Appendix B. Web Application & Python Data Analysis Library source code.....	54
Python Code.....	54
HTML5 templates .....	70
Java Files.....	77

## Abstract

ASSISTments is a useful educational platform for students to do their homework and receive their grade promptly online. Also, it is an educational research tool to learn about improving student learning. The system called Assessment of Learning Infrastructure(ALI) sends analytics to researcher's emails containing raw-data and analysis. However, ALI currently lacks in analysis features so Python will be used to build data analysis library and integrate with ALI to offer researchers more analysis information from students' data.

## Acknowledgements

I would like to thank my advisors Korinn Ostrow and Neil T. Heffernan for guiding me through this process. I would also like to thank Anthony Bolteho for teaching me the infrastructure of ALI and guiding me with programming it, and Thanaporn "March" Patikorn for providing a template tool which enabled the automatic data analysis from my library.

# 1. Introduction

## 1.1 ASSISTments Background

With the spread of network infrastructure and growing trend of online learning, teachers and students start gaining benefits from improved performance as shown by Neil Heffernan's online learning platform, ASSISTments. It is an online learning platform in which many schools participate to assign homework and give immediate feedback to students. The efficacy trial conducted by SRI proved that the ASSISTment improve students' mathematics learning [7]. Moreover, this was created with sound educational research in mind unlike many other learning platforms such as Edx or Udacity. ASSISTment is a better place for learning science researchers to test their ideas and learn if certain intervention method affects students' performance [3].

ASSISTments is great as a collaborative research tool for sound science, and researchers to conduct randomized controlled trial on students [3]. One can access myriad meaningful data that will push forward the advancement of public-school education in the United States and worldwide. According to [2], "capacity to support collaborative research environments has the potential to lower the stakes by drastically reducing costs, promoting validated universal measures of achievement, and assisting researchers through the process of designing, implementing, and analyzing RCEs conducted at scale within real-world classrooms." Therefore, ASSISTments is a valuable environment that could benefit from further development.

With ASSISTments as a powerful environment for education research at scale, Assessment of Learning Infrastructure(ALI) strives to further optimize the online learning platform by providing an analysis tool for researchers, and bring ASSISTments "to the next level as a shared scientific instrument for educational research." [4] ALI started out as a mere data reporting tool for researchers who have trouble accessing the database using SQL, relational database language for retrieving data from a database. Then, more sophisticated feature such as automated data-preprocessing and data analysis tools were developed. Current capabilities of ALI include universal data reporting in 4

different file formats, timestamped links to every data analysis report ever provided to a researcher, Chi square test which can detect bias in selection and email notification to the researchers whenever there is a risk of not meeting internal validity.

Moreover, ALI builds upon the Pittsburgh Science of Learning Center's Data shop's way of shared datasets to "promote open, replicable, and sound science" [8]. In order to cope with inability to replicate research finding, and put an emphasis on data accountability, ALI provides timestamped links to every data analysis report ever provided to a researcher. Today, it has become an automated data reporting and analysis tool for learning at scale.

Further development of ALI will help students, teachers and researchers all together. As researchers conduct online learning research more efficiently and productively, they will be able to make more progress in finding the effective methods to educate students. Then, students can benefit from increased learning rate and improved performance at school and teachers benefit by learning the best possible teaching method.

## 1.2 Purpose

The primary focus in this project is automating the analysis of data from ALI and presenting the result with visually appealing tables and graphs in APA format. ALI used to do analysis on the data by sending analytic commands to Rserve which is the server that allows other programs to call R functions through TCP/IP connection. However, currently, the ASSISTments team switched to calling analysis functions in Python and discarded using functions in R. As a result, ALI is currently lacking in data analysis tools for researchers to utilize.

I plan to create a reliable and accurate data analysis tool by working closely with ASSISTments team at WPI and writing an elegant python code. I aspire to make a reliable, and easily usable data analysis library that can be utilized by many other educational learning platforms and extensible such that future developers can build upon my previous work, unlike the previous analysis tool in ALI which is no longer in use. The completion of this project will greatly aid learning science

researchers to be more productive and accumulate statistically sound information from data.

The development of analysis tools in ALI has more additional benefits, as mentioned in [4], “ability to analyze at scale will also benefit the platform, as it will help the ASSISTments team to quickly isolate and remove ineffective interventions.” The platform will become more useful and manageable. Moreover, finished ALI can lead to “personalize learning by better understanding why certain educational practices and interventions work for certain students but not for others” [4]. The above statement is supported in [3] that the experimental results will help educational researchers answer the question “What works best for whom?” ALI shifts the paradigm of Big Data to Big Experimentation, by allowing researchers to do experiments on the data, not just data mining it to obtain valuable information. By contributing to the development of ALI, I would like to make the online personalized learning happen and benefit millions of students worldwide.

One of the grand goals once the infrastructure for the analysis is completed is the usage of Artificial Intelligence techniques such as Deep Learning to further boost the capability of Ali, beyond simple statistical analysis tool, as a continuation of the work done in [1] which utilizes the big data and Deep Learning to analyze the data and improve in precision to estimate the effect estimates. Deep Learning is an artificial neural network is a form of model that learns from samples of data to predict other data in the population.

## 1.3 Theoretical Framework

### 1.3.1 Programming with Python

Python is chosen as a computer language to build the data analysis library because stable and widely-used python libraries such as pandas, NumPy, and Matplotlib to perform complex data manipulation, analysis and visualization easily [6]. There are python packages that do the data analysis for data scientists and statisticians and two popular options are StatsModels and Scikit-learn. StatsModels has many functions that computes complex statistics in the data, has similar syntax to and is validated against R, programming language. Scikit-learn is more often used in machine learning and data science [17]. According to Anthony Botelho who developed the ALI, ALI switched

from R to python because python supports machine learning library called TensorFlow.

The JetBrains PyCharm will be used for Intergrated Development Environment (IDE) and main dependencies of the project are Numpy, Pandas, Matplotlib, and StatsModels. Numpy stands for Numerical Python and is the “fundamental package for scientific computing” because it is useful for preprocessing and transforming the data and performing numerical operations and descriptive statistics such as calculating means and standard deviations [12]. Pandas stands for Python Data Analysis Library and it is dependent on the Numpy library. It will be used for sorting, and filtering data and cleaning data before analysis [13]. Matplotlib is the library for visually showing the data by drawing diagrams. Statsmodel and Scikit-Learn are the golden standard library for statistical analysis or machine learning in Python [16].

Python is a high level, object-oriented language that features simple syntax compared to Java or C++ and vast number of libraries and community support. According to [9], “Python was created in the early 1990s by Guido van Rossum at Stichting Mathematisch Centrum in the Netherlands as a successor of a language called ABC.” Worldwide, the trend of Python’s increasing popularity is evident. Based on the Stack Overflow question visit, “python is the fastest growing major programming language.” Python’s popularity surpassed java since 2016 according to the Figure 1. Even though Python is versatile in that it can be used for system administration, web development and more, “The fastest-growing use of Python is for data science, machine learning, and academic research” [5] as shown in Figure 2 below. To sum up, Python is a common choice for data science and machine learning because of its capacity for scientific computing and data analysis.



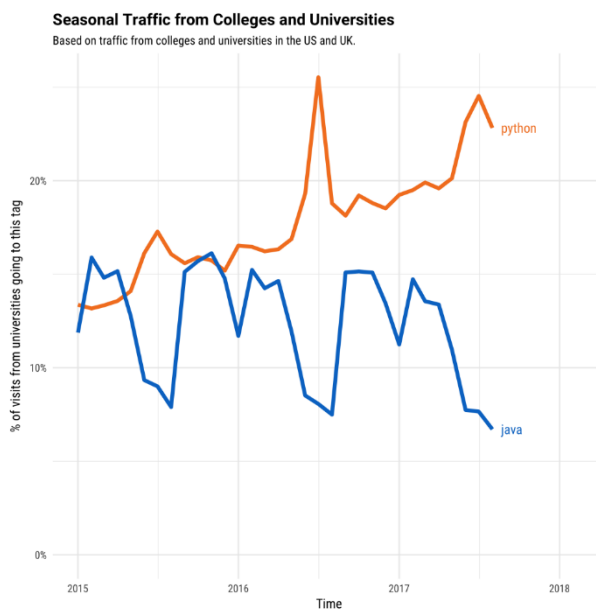


Figure 1 Python vs. Java Popularity

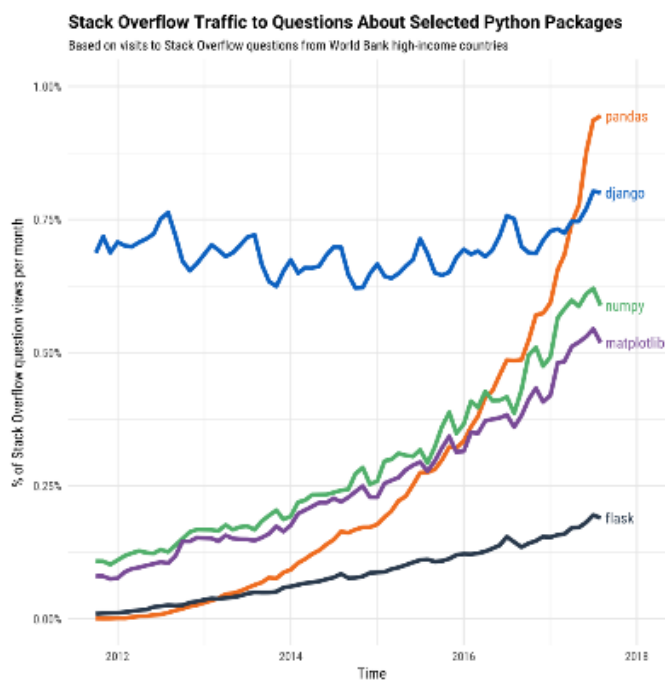


Figure 2 - Python Package Popularity

### 1.3.2 Software Engineering

The library will be built following the best methodologies in Software Engineering.

According to IEEE, Software Engineering is “the application of a systematic, disciplined, quantifiable

approach to the development, operation, and maintenance of software.” The library will be comprised of many Python modules. Module is a piece of code that has a specific functionality so that my code doesn’t interfere with operation or understanding of the rest of ALI code. Also, for future developers to understand my work, I will document my process by drawing class diagrams, ER diagram, and commenting source codes in detail.

### 1.3.3. Database

Compared to MySQL database which is a relational database management system (RDBMS), PostgreSQL which is an object-relational database management system (ORDBMS) that supports many NoSQL features is used to manage ASSISTment data. The queries used to save and retrieve the data are almost the same for both database systems. Community support are great for both systems and the list of programming language supported are also almost the same [30]. MySQL uses pluggable database engines, among which InnoDB is the mostly the used database engine. However, in PostgreSQL uses only one database engine designed for PostgreSQL. Database engine is the component of the system that performs storing and retrieving the data from SQL query. As a client software to manage and modify the PostgreSQL database, pgAdmin version 4 will be used. It is an open source administration and development platform for managing PostgreSQL database in the browser.

When comparing between other database systems, there is not a clear advantage for using PostgreSQL compared to MySQL. However, PostgreSQL is a newer database management system compared to MySQL and the current trend shows that PostgreSQL might be more used widely in the future. Figure 3 below shows that PostgreSQL database is increasing in popularity every year most quickly among the top 5 popular database worldwide [29].

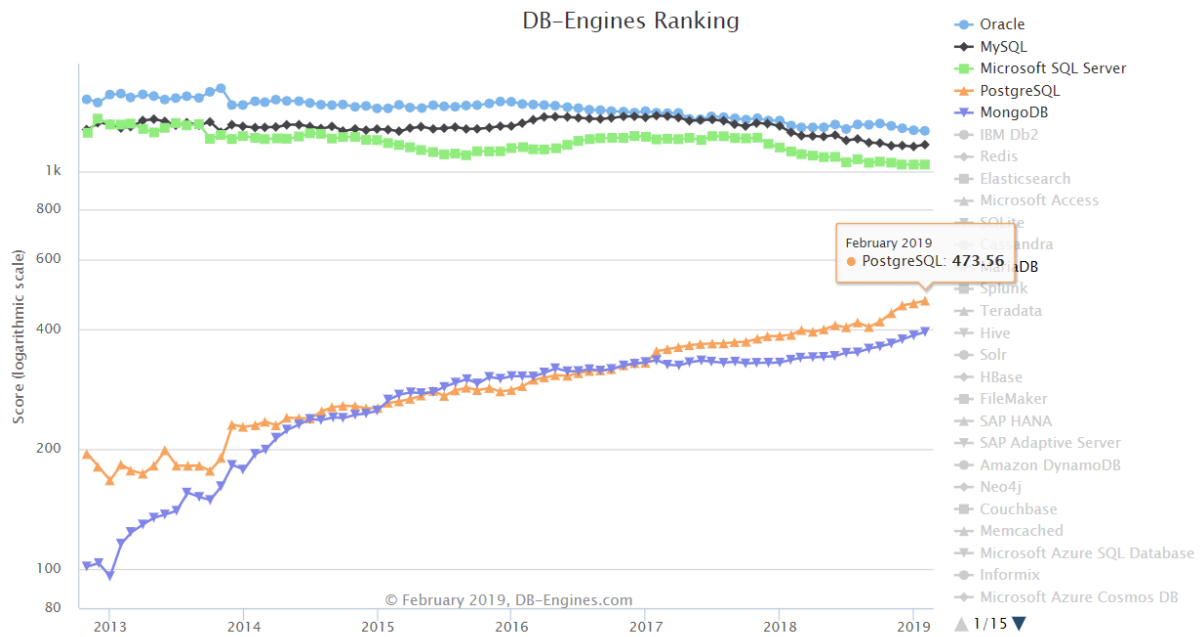


Figure 3 - Database Engine Popularity Rank

## 2. Methods

### 2.1 Software Design

The process first starts with learning the best technique and technology available today to productively make an effective library that can last. Due to the nature of fast changing and improving Computer Science Technology and Language features, the library will be built using the latest Python version which is Python 3.7.1 released in October 20th, 2018. Among the best practices that are provided in the Zen of Python[11], simplicity, usability, consistency, and reliability are the utmost goal in writing the data analysis library. Time and space complexity of my program is a secondary concern currently since ALI can perform data analysis periodically inside the server when the user is away, and report can be sent to the users through email. In the future, the optimization will be necessary because there will be a research tab in Assistments webpage where the researcher can choose to perform the data analysis anytime.

There are six basic Software Development Life Cycle (SDLC) methodologies known in Software Engineering field: Agile, Lean, Waterfall, Iterative, Spiral, DevOps [28]. I will take the combination of agile and lean approach, so the Python library will progress through the incremental change in the repetitive cycle of release, test and modification very often, in weekly basis. Also, time will be saved by reducing the unnecessary features and working on the most important goals. The design and features of the library will be improved frequently through communicating with Anthony Botelho and ASSISTments team every week.

## 2.2 Data Analysis Examples

To ensure that the statistical analysis library gives a valid output, I understood various concepts of statistical analysis methods first by doing six assignments in PSY503 Research Methods for Learning Sciences course. The class assignment dataset originates from a multivariate class at Clark University and has a first row comprised of header names and the rows below is comprised of data. It contains multiple predictors such as anger management, jealousy and abuse problems that affect the outcome variable, the conflict in a relationship between a couple.

The class assignment dataset1 is analyzed first using multiple linear regression in IBM SPSS and the output will be compared to the output from the Python data analysis library to test the validity of the output. IBM SPSS is a software platform with GUI for advanced statistical analysis. Dataset2 is a cleaned and screen version of dataset1 for ANCOVA, MANOVA, and factor analysis. Their analysis outputs are referenced from Homework 1, 2, 3, 4, 5 and 6 of PSY503 and are generated from analyzing dataset 1 and 2 using IBM. However, they are not shown in this paper to keep Homework solution hidden from the future PSY503 students.

The statistical analysis methods covered in six assignments are Multiple Regression, Logistic Regression, Moderation/Mediation, ANOVA/ANCOVA, MANOVA/MANCOVA, and Factor Analysis; However, implementing factor analysis in the library would need integration with Qualtrics because ALI currently do not store the data necessary for factor analysis. Other additional analysis

operations covered in [22] can be added so that the vast range of statistical analysis tools will be available in the Python library for ALI. At the end, an experimental study should be done so that the analysis tool can be tested by going through the same steps that a researcher using Ali would be going through.

After the outputs are validated, unit test will be performed to eradicate any bugs in the program. The Unit Testing is an essential component in software development process because bugs and errors in the program sometimes take several weeks or months to fix and this can cause detrimental effects to the customers and business. In the context of ALI, the incorrectly written Python library can undermine the effectiveness of ALI and even produce incorrect results to the researchers and invalidate their research; Therefore, the validity and reliability of the library will be verified by testing every code in the library through Unit Test Framework. The Python Unit Testing Framework “unittest” is inspired by Junit from Java and it supports test automation, and other important features in an object-oriented way [10].

### 2.3 Design of the Library

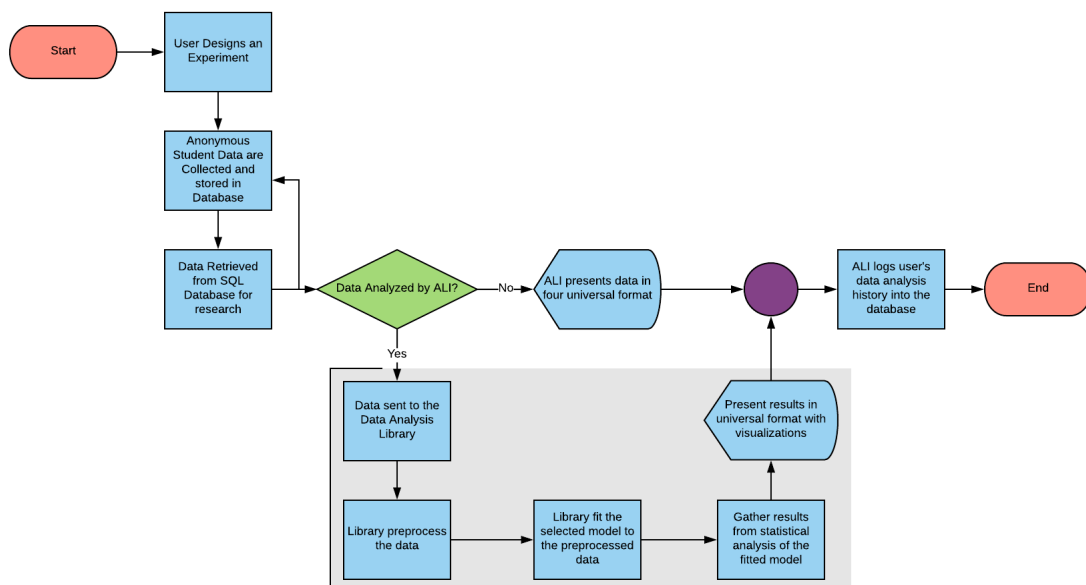


Figure 4 - Flowchart diagram of ALI and data analysis library

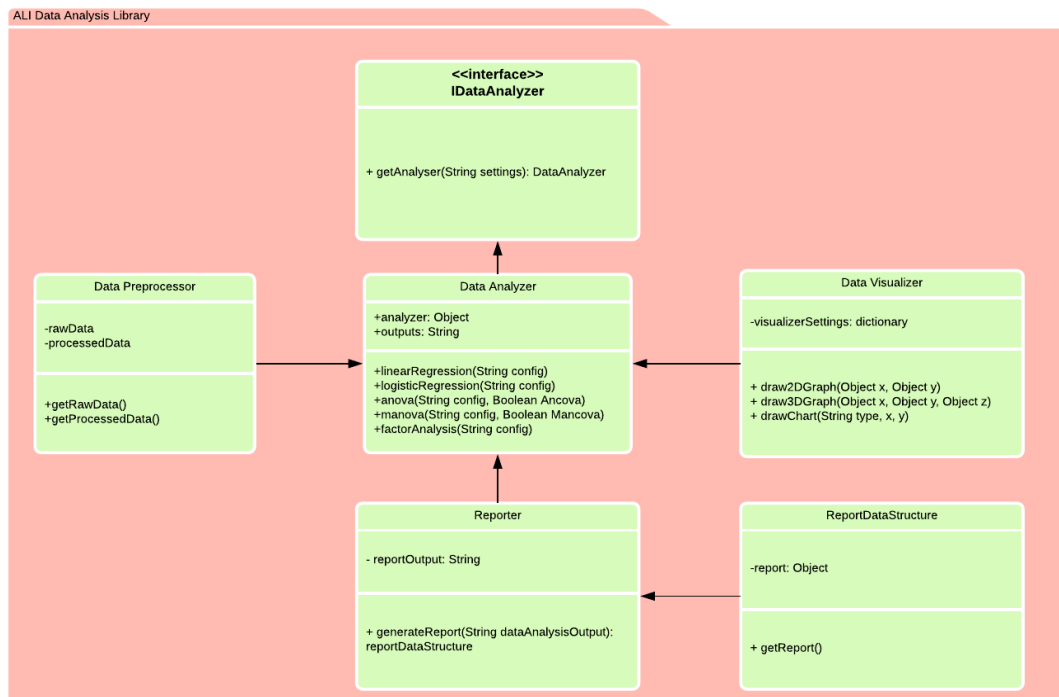


Figure 5 - Structural UML Class Diagram

Figure 4 is a flowchart or activity UML diagram to document an existing process of using ALI and demonstrate how the data analysis library will integrate with ALI from the perspective of a user using our program. The gray shaded part of the diagram is where my data analysis library fit in the overall process. Python library first receives the request from ALI and preprocess the data that can be inputted into the linear model function. The data will be fitted to the model and the result from the model is converted into tables and graphs which will be presented to the user in a consistent format. After the Python library finishes, the analysis history will be logged or recorded in ALI to have a transparent records of everything researcher did. This is inspired by the Open Science Framework that stores all info on preregistering experiments and collaborations.

UML Class diagram in Figure 5 shows the object-oriented design of the data analysis library and gives an overall top-level view of the system. These also serve as documentation for programmers so that they can build upon this project later. UML stands for unified modeling language and it is a general-purpose modeling language in the field of software engineering to represent the system

visually using diagrams[15]. The data analysis library package inside the red rectangular grouping will be utilized by ALI through IDataAnalyzer Interface. Data Analyzer Class is the central part of the library that calls methods from various helper classes. Data Preprocessor class is for preprocessing the raw input data, Data Visualizer class is dedicated only for representing the output from data analysis in a visual form such as 2D graph or a histogram. Reporter class take the output from data analysis functions and turn it into a universal format that is consistent with ALI's universal report format. ReportDataStructure class provides the Reporter Class with a data structure to put information into.

## 2.4 Web Application

Instead of having data analysis library hidden under the operation of the ALI, the data analysis library will be available for everyone not using ALI through web application. This adds the values to the library and will benefit anyone who wants to analyze the data online without using expensive statistical software. Also, the data analysis library can be tested easily when it is interfaced with the web application because the HTML5 web page of web application facilitates the user interaction. Since the analysis result will not be sent to ALI but it will be displayed on HTML5 webpage, ALI does not need to be developed or modified to process the result from Python library and present the result to the user. Python data analysis library will still work with ALI but users will also have an option to do data analysis for other datasets not from ASSISTments.

There are currently many statistical analysis webpages that let users input data and receive the result of a statistical analysis. However, none of them can take a raw csv file, parse it, and generate a result in APA format. For example, the website called VassarStats has the linear regression webpage shown above but it can take only up to 5 different independent variables, and users must manually input the data for each column [27]. It is more convenient for the user if the website takes any csv file, parses it and generates the result automatically.

The result will be presented on a static webpage in HTML5 which can be easily viewed on

many modern browsers such as Edge, Firefox, Chrome and Safari. There are two popular Python web frameworks to choose from: Flask and Django. Django framework has many features that are unnecessary for this project and would require investment of additional time to learn [26]. In contrast, Flask is a microservices web framework used to develop an application comprised of small related services that only complete one business function. This fits the current trend of rapid development and deployment of products to users according to the agile practice and continuous testing [25]. Since Python library will only work as a restAPI that responds to the request of the client by showing the data analysis result on the webpage, Flask is an appropriate option.

## 2.5 Setting up ALI project in your Local Machine

This section is formatted as a tutorial for a developer who wants to setup ASSISTments ALI project in his/her local machine to start programming ALI. This tutorial is made using Eclipse Version: 2018-12 (4.10.0). but should work for future Eclipse versions too. Eclipse IDE and Subversion software versioning and revision control system are used among ASSISTments developer so please download the latest version of Eclipse for Java before proceeding further.

### 2.5.1 Install plugins on Eclipse

In Eclipse IDE, download Subversive and Gradle plugins by going to

help → install Eclipse plugin in Eclipse Marketplace → Subclipse 4.3.0

help → install Eclipse plugin in Eclipse Marketplace → Buildship Gradle Integration 3.0

### 2.5.2 Download Repository using Subversion

In Eclipse IDE, download the ALI project from the repository by going to

File → New → Other → SVN → Check out projects from SVN



Link of the svn repository is <https://fusion.wpi.edu/svn/assitments-rep/assitments/theNextGeneration/branches/analytics/DataDumper>

If the project is successfully downloaded from the repository, you can pull the latest code from the repository by doing

Right Clicking on the root folder of the project → Team → Synchronize with Repository

### 2.5.3 Build Project using Gradle

Open the gradle.properties file located on the root folder of the project and change jdk directory to your local jdk directory. Please note that jdk version 1.8.0 should already be installed in your machine.

For my local machine, it is `org.gradle.java.home=C:/Program Files/Java/jdk1.8.0_181`

Finally, do the following to build the project using Gradle.

Right Clicking on the root folder of the project → Run As → Run Configurations → Gradle Project  
 → Inside Gradle Tasks text area, type in clean build → Set Working Directory to the root folder of your current ALI Project → Click “Run” Button

### 2.5.4 ALI Doc generation

If the project is built successfully, modify ALIDumper.java under alidumper package to get your first ALI Doc report to your email.

Put problem set ID you want to analyze in this line:

```
Int id = PsidCodec.decode("PSAYCFH");
```

And put your email address in this line:

```
aliParameters.put(ALIParameterKeys.EMAIL, "youremail@wpi.edu");
```

Finally, run the main function of the ALIDumper.java ☺

## 2.6 Python Library Integration to ALI

Python Data Analysis library was integrated into the ALI Project as shown in Figure 7 below. The file structure was organized in a way that resembles the common FLASK application structure. All the Python Analysis files are stored in the “analysis” folder under the “src/main/python” folder and the rest of the existing Python code was stored in “previous” folder. `post_req.py` file under “src/main/python/app” is the main Python server which need to be running for ALI to communicate with ALI data analysis library. The “src/main/python/app” folder also contains static folder which have css, and image files and templates folder have html5 template files on which new data are populated.

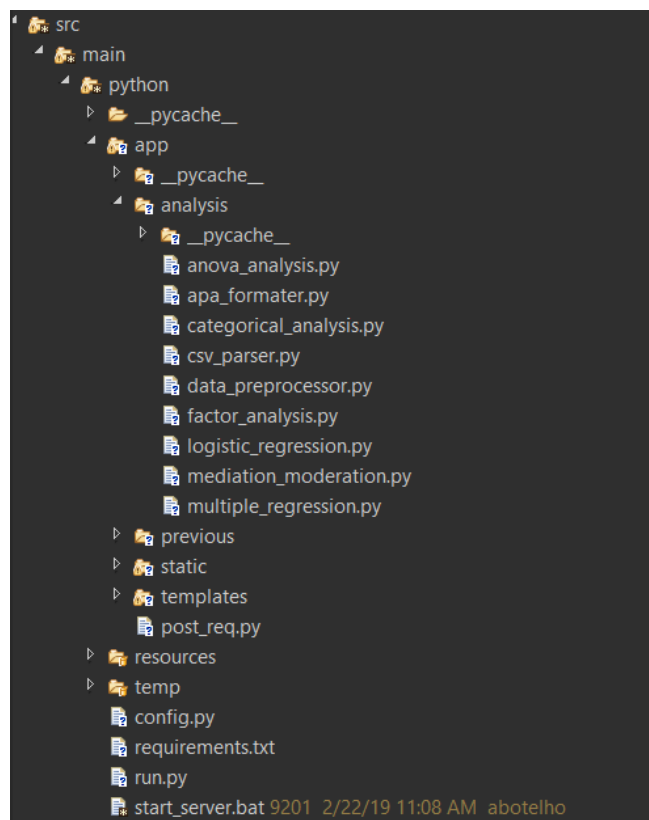


Figure 6 - New Python Server Structure

### 2.6.1 Analysis function

Server response functions of FLASK application were added to respond to post requests from ALI for multiple linear regression, ANOVA, and MANOVA inside Python Code as shown below. In

general, the code first starts by parsing the Json data in request object into the dictionary variable called “content”. Then, turning the dictionary variable is converted to Pandas dataframe object for it to be processed by my custom class called Data\_Preprocessor. This class takes the input dataframe, what to do for given missing value, header names of independent, dependent and covariate variables. It also needs to know which of them are categorical variables. Then, Data\_Preprocessor class processes the data in a uniform way such that other Python analysis classes such as Anova\_Analysis can be instantiated with the instance of Data\_Processor class. At the end, the call to run the analysis on the processed input data is made and the result of the analysis is returned to the ALI program which requested that result.

```
@app.route('/analysis/manova', methods=['POST'])
def manova_func():
    print('manova func')
    content = request.get_json()
    print(content)
    input_df = pd.DataFrame.from_dict(content)

    data_preprocessor = Data_Preprocessor(input_df=input_df,
                                        missing_data='drop',
                                        x_columns=["independent1"],
                                        y_columns=["dependent1", "dependent2"],
                                        categorical_columns=[])

    # print(data_preprocessor.input_df.to_string())

    manova_obj = Anova_Analysis(data_preprocessor)
    manova_result = manova_obj.run_manova()

    return manova_result
```

*Code 1 - MANOVA Server Response Function*

## 2.6.2 Troubleshooting MANOVA

When running MANOVA analysis with

independent variables: number of Correct Problems in Posttest, and number of All problems in Posttest

dependent variables: Prior Percent Correct, Z-Scored Mastery Speed

the program halted with "Covariance x is singular" error. This was solved by removing the independent variable "Number of total posttest problems" from the equation. This is because the homogeneity of covariance matrices is assumed for MANOVA. The correlation between any two dependent or independent variables is the same in all groups [31]. In other words, none of the variables should be highly correlated with each other. Since the number of correct problems in Posttest is highly correlated with total number of problems in Posttest, covariance matrix must have been singular.

As a reference, variance is the measure of the variation of one random variable and covariance is the measure of difference between two random variables.

$$\sigma_x^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$$

Formula for variance:

$$\sigma(x, y) = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})$$

Formula for covariance:

Covariance Matrix is square matrix given by  $C_{i,j} = \sigma(x_i, x_j)$

The diagonal entries of the covariance matrix are the variances and the other entries are the covariances [32]. As a side note, covariance matrix can be used to calculate the Mahalanobis distance which represents the correlated multivariate distances between two points.

$$D_M(x) = \sqrt{(x-\mu)^T C^{-1} (x-\mu)}$$

## 2.7 ALI Project Structure

In order to add features to the existing ALI project in Java, understanding the existing

structure of ALI is essential. Since there are not any proper documentations to refer to regarding the ALI project structure, I studied the packages and classes that are relevant to the integration of Python data analysis library and Template Tool and created a UML class diagram. The class diagram only describes a small portion of the entire ALI project but is helpful reference for the future developers and can be extended to include the entire ALI project structure in the future.

ALI already had ANOVA analysis feature, so I started with sending ANCOVA analysis request to Python library and generating ALI documentation. ALI is comprised of 24 Java packages each containing various classes. For the purpose of SQL connection, data analysis, object storage, and ALI documentation generation, only the following packages need to be learned: structure, dataset, persistence, statsobjects.oVa, rserve, functions, alidumper, and alidoc. Python Data Analysis Library was put in a separate package in orange color in the class diagram. The figure 6 below describes the packages and classes involved with data analysis and document generation. Please note that not every details are included for each package or classes for simplicity and time saving.

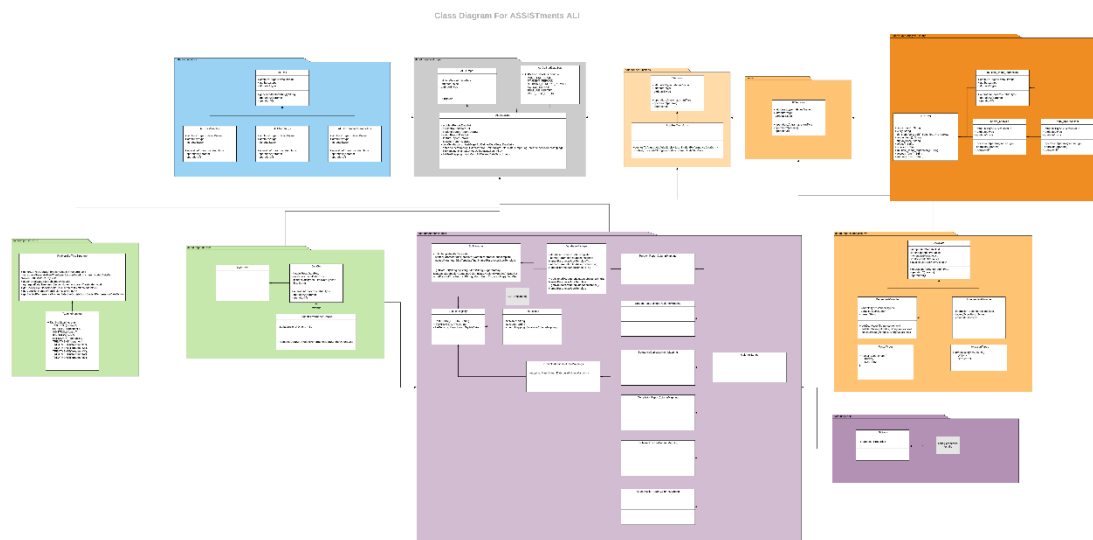


Figure 7 - ALI Class Diagram

## 2.8 ALI Doc generation

The packages involved with ALI Doc generation are alidoc, alidumper, functions, dataset and rserve. The existing code was left intact for stability reason and only comments and additional code was added inside the existing classes. The main program called ALIDoc.java is in alidoc package and runs the doc generation as the main class. Since ALIDoc.java has existing ANOVA analysis request function, studying this file is the starting point.

### 2.8.1 Data Preparation

In order to add the functions to request multiple linear regression, ANOVA, and MANOVA in ALI, a way to store rows of data for each independent variables, covariate variables, and dependent variables have to be found. Currently, the existing ANOVA function prepares the independent variable containing the values: “treatment1”, “treatment2”, etc. and the dependent variable containing the values: 1.00, 0.0333, 0, 1, etc. However, these input data are for analyzing the problem set’s characteristic, not how each individual student did in the problem set. Studying ANOVA function led me to the ALIDataSets class which defined DataSet variables storing the report data such as ProblemReport describing how each student does in a problem set and FeatureReport containing covariate data of each student from Postgres database. DataSet class is comprised of DataRow class objects so specified column and row of the dataset can be retrieved.

At first, effort was made to resemble ANOVA’s call to prepare independent and dependent variable and call analysis function with that prepared data as shown in code 3:

```
List<DataSet> studentCP = PostTestTransform.posttestToAggregateVariable(posttest, conditionName);
anovaAnalysisResultList = Statistics.calculateANOVA(studentCP);
```

*Code 2 - ANOVA data preparation*

However, understanding how PostTestTransform class and its related class ProblemTreeStructure store the data in the tree structure and retrieve only certain information it wants was too time-consuming without any guidance. Since ALI sends List of String data to Python server anyway, list of

String was directly inputted into the analysis call to Python Server. In the code 4 shown below, the index number of each column with the specified column names are found. Then, each index number is used to find rows of the specified column's data. Lastly, the data are inputted as parameters of the analysis function inside statsobject.ova package. which later requests Python Server for analysis.

```
independent_var1_index = aliDataSets.getStudentReport().getHeaderColumnIndexByString("Correct Problem 1");
dependent_var1_index = aliDataSets.getStudentReport().getHeaderColumnIndexByString("Correct Problem 8");
covariate_var1_index = aliDataSets.getFeatureReport().getHeaderColumnIndexByString("Prior Percent Correct");

List<String> independent_var = aliDataSets.getStudentReport().getColumnWithoutHeader(independent_var1_index);
List<String> dependent_var = aliDataSets.getStudentReport().getColumnWithoutHeader(dependent_var1_index);
List<String> covariate_var = aliDataSets.getFeatureReport().getColumnWithoutHeader(covariate_var1_index);
ancovaAnalysisResult = Statistics.calculateANCOVA(dependent_var, independent_var, covariate_var);
```

*Code 3 - ANCOVA data preparation*

## 2.8.2 Population of data inside the ALI Doc Template

After receiving the analysis result in String data format from Python Server, the result needs to be shown in the ALI Doc report. In order to do that, the following steps are taken:

1. Add static String variables inside ALIDocVariables.java.

```
static String ANALYSIS_REPORT_PYTHON = "ANALYSIS_REPORT_PYTHON";
static String ANALYSIS_REPORT_LINEAR_REGRESSION = "ANALYSIS_REPORT_LINEAR_REGRESSION";
static String ANALYSIS_REPORT_ANCOVA = "ANALYSIS_REPORT_ANCOVA";
static String ANALYSIS_REPORT_MANOVA = "ANALYSIS_REPORT_MANOVA";
static String ANALYSIS_REPORT_MANCOVA = "ANALYSIS_REPORT_MANCOVA";
static String TEMPLATE_REPORT = "TEMPLATE_REPORT";
```

2. Then, in the `generateALIDocString()` function inside ALIDoc.java, add the following function call: `emailParameters.addPair(String, String)`. Also in the same file, add two getter and setter functions for predefined templateResult String variable.

```
public String getTemplateResult() { return templateResult; }
public void setTemplateResult(String templateResult) {
    this.templateResult = templateResult; }
}
```

3. In ALIDumper.java, adding the following code `aliDoc.setTemplateResult(String)` will send the processed template result from SQL database to ALIDoc class instance.

4. Lastly, the html template that is in src/main/resources/AliDocTemplate.htm can be edited to append the analysis result to the middle of ALI doc report by adding the following HTML5 paragraph.

```
<p style='font-size:12.0pt;font-family:"Times New Roman",serif; color:black; margin-bottom: 12px;'>&lt;%TEMPLATE_REPORT&gt;</p>
```

## 2.9 ASSISTments Template Tool

### 2.9.1 Template Explanation

From the beginning of the C term, Web application was considered difficult to integrate with the current ALI system before the end of C term for my IQP project, so my advisors and Anthony suggested me to create automated analysis feature utilizing ASSISTments template tool. Therefore, instead of continuing the development of Web application, ASSISTments template tool was used as an interface to automate the data analysis.

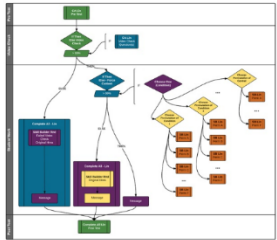


**ASSISTments Build-from-Pattern** Signed in as junbongj0313 ([Log out](#))

Standard Skill Builder | **Video VS Text Experiment** | Forced Early Content Experiment | Intervention Before First Problem | Intervention After First Problem

Intervention After N Problems

Template: **PSA2KAD**  
 Summary: a simple video vs text experiment.  
 This template forces altered contents early and uses permutations as cheating prevention.  
 Click [here](#) for more information about this design pattern



PSA2KAD  
([view full image here](#)).

New problem set name*	problem set by junbongj0313
Video check PRA ID* (?)	PRA4MUZ
Comma-separated list of PRA IDs of flat skill builder problems*	PRA86DP,PRA86DQ,PRA86DR,PRA86DS,PRA86DT,PRA86DU,PRA86DV,PRA86DW,
Comma-separated list of PRA IDs of problems in Control**	PRABAX9Y,PRABAX9Z,PRABAX92
Comma-separated list of PRA IDs of problems in Treatment**	PRABAX9P,PRABAX9Q,PRABAX9R
Comma-separated list of PRA IDs of problems in Pre Test	
Comma-separated list of PRA IDs of problems in Post Test	
Original problem set ID (?)	
PRA ID of the message before the main content	PRA8BX8W
PRA ID of the message after the main content	PRA8S2F

\*required  
 \*\*at least one condition is required

Status: **ERROR: ("message":"Internal Server Error: null")**

**Pattern Note:**

1. This builder will consider treatment2 only when treatment1 is provided, treatment3 only when treatment2 is provided, and so on. If treatment3 is provided without treatment2, problem IDs in treatment3 will be ignored and will not appear in the problem set.
2. Any conditions (control/treatment) must contain at least 3 items
3. Due to the structure of this design pattern, problems in any conditions (Control or Treatment) must not overlap with flat skill builder problems. However, any conditions may overlap with each other.

**Builder Note:**

1. One problem cannot be in a given section more than once.
2. The flat skill builder must contain at least 10 problems, and at most 200 problems.
3. Sections other than the flat skill builder can contain at most 25 items, unless it is specified otherwise.

Figure 8 - Template Tool Webpage

The link to the template tool website: <http://askeeper.cs.wpi.edu:8080/SBGenerator/>

For users who want the automated data analysis added to their ALI doc report, the template should be filled out. As shown in Figure 5, user first chooses which template their skill build problem fit into by clicking on tabs on top of the page. Then, user types in all problem IDs to be analyzed, distinguishing which problems are in control, treatment, pretest, and posttest in the empty input fields shown in the middle of the webpage. At the end, clicking on “Build Problem Set” button under the input fields sends the user input to database which ALI will use later to analyze the data. But currently as of March 1<sup>st</sup>, the website does not have a way to submit a new problem set to relate to one of the given templates so clicking a “Build Problem Set” button only prints out the error message.

The template called “video check, forced early content” is the most generic and complex template among all the templates in the Template Tool website so understanding this template will enlighten users to know how to use other templates. The experiment first starts with a pretest. Then, it checks if the user can view the video online by showing the video content that asks the user to type what the video says as an answer. Video check is implemented in case the student does not have access to view Youtube Video because some schools block the access to the Youtube. If the student can view the video, then the student is randomly given either control or treatment. There can be multiple treatments from which a student is assigned only one. If the student finished the problems with several consecutive right answers, they are put into another group. At the end of the problem set, every students receive the posttest. The figure below is provided as a reference.

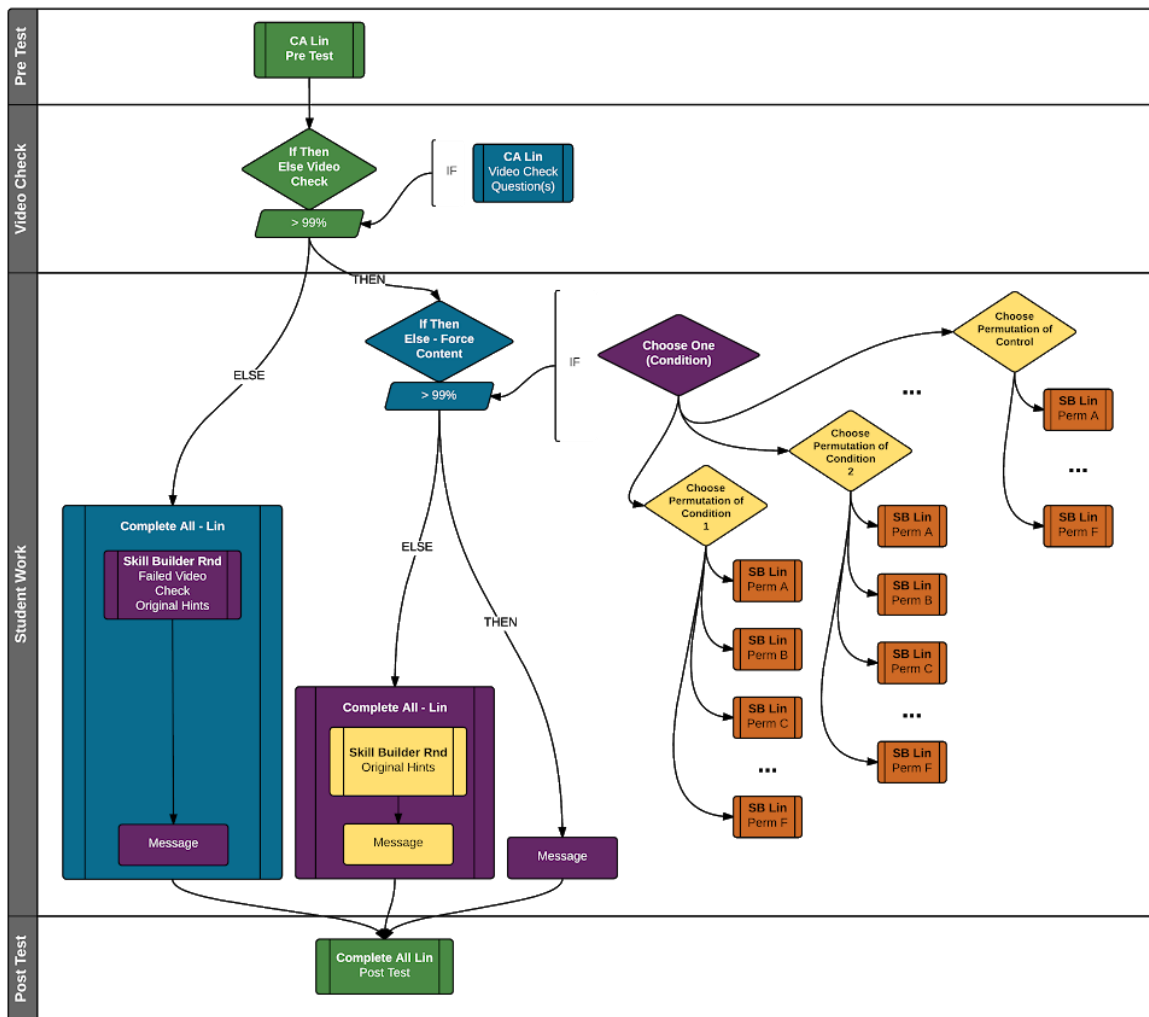


Figure 9 - Video Check, Forced Early Content Template

## 2.9.2 Connecting to the Database

By using template tool database that maps researcher’s Skill Builder Problem set to one of the predefined templates, analysis to do among T-Test, ANOVA, ANCOVA, MANOVA, MANCOVA, or multiple linear regression can be determined appropriately. In the ASSISTments database called “analytic\_db\_test” hosted at dev.tng.cs.wpi.edu, two tables “experiment\_templates” and “generated\_experiments” are used in the Template tool. Figures below show rows and columns of each table and dummy values populated in them.

	id [PK] integer	name text	description text	version integer	is_active boolean	created_at timestamp with time zone	updated_at timestamp with time zone	parameter_schema text
1	3	skill_...	This pattern u...	1	true	2019-02-22 11:09:52.322-05	2019-02-22 11:09:52.322-05	
2	4	skill_...	This pattern p...	1	true	2019-02-22 11:10:21.658-05	2019-02-22 11:10:21.658-05	
3	5	skill_...	This pattern i...	1	true	2019-02-22 11:10:41.823-05	2019-02-22 11:10:41.823-05	
4	6	skill_...	This pattern i...	1	true	2019-02-22 11:11:13.099-05	2019-02-22 11:11:13.099-05	

Figure 10 - Experiment\_Template table

	id [PK] integer	sequence_id integer	experiment_template_id integer	creator_user_id integer	created_at timestamp with time zone	updated_at timestamp with time zone	note text	parameters text
1	1	967828		6	337537	2019-02-22 11:13:53.73-05	2019-02-22 11:13:53.73-05	TODO
2	2	967829		5	337537	2019-02-22 11:13:58.3-05	2019-02-22 11:13:58.3-05	TODO
3	3	967830		4	337537	2019-02-22 11:14:01.437-05	2019-02-22 11:14:01.437-05	TODO
4	4	967831		3	337537	2019-02-22 11:14:04.572-05	2019-02-22 11:14:04.572-05	TODO
5	5	967832		3	337537	2019-02-22 11:14:05.48-05	2019-02-22 11:14:05.48-05	TODO
6	6	967833		3	337537	2019-02-22 11:14:08.409-05	2019-02-22 11:14:08.409-05	TODO
7	7	967834		3	337537	2019-02-22 11:14:09.363-05	2019-02-22 11:14:09.363-05	TODO
8	10	967837		6	337537	2019-02-22 11:14:23.761-05	2019-02-22 11:14:23.761-05	TODO
9	13	967841		6	337537	2019-02-22 11:22:11.589-05	2019-02-22 11:22:11.589-05	TODO

Figure 11 - Generated\_Experiments table

The following steps are taken to connect to Template Tool database in ALI:

1. Config.properties file under “src/main/resources” folder have all the credentials for connecting to the database. This file was is edited to include URL, username, and password obtained from Anthony.
2. SQL files “GetTemplate.sql” and “GetExperiemntWithTemplate.sql” are created under “src/main/resources/sql” folder. “GetTemplate.sql” is for retrieving the predefined templates with some description and “GetExperiemntWithTemplate.sql” is for retrieving registered experiments associated with a template.

“GetTemplate.sql” file have the following SQL query in it:

```
SELECT * FROM experiment_templates WHERE id = (:id);
```

“GetExperimentWithTemplate.sql” file have the following SQL query in it:

```
SELECT * FROM generated_experiments WHERE sequence_id = (:sequence_id);
```

3. Column names of two tables are added inside ColumnNames.java under the persistence package. These will be referenced by the ColumnMapping java files created in the next step.
4. Additional Java files called “ExperimentTemplateColumnMapping.java” and “TemplateColumnMapping.java” are added under the persistence package. They specify the column name and type for the Java Database Connectivity (JDBC) such that the data retrieved from SQL can be stored in the Java variables of appropriate data types.
5. Inside DatabaseManager.java, copy and paste the `getNamedParameterDatabaseInstance()` function to create `getNamedParameterDatabaseInstanceTng()` with different url, username, and password for connecting to the Template Tool database.
6. Define the following static variables inside SqlFileRegistry.java

```
public static String TEMPLATE = "GetTemplate.sql";
public static String EXPERIMENT_WITH_TEMPLATE = "GetExperimentWithTemplate.sql";
```

7. Add the following functions calls inside KnownDatabasesColumnMappings.java to map the sql file names with the corresponding column mappings in Java.

```
mapping.put(SqlFileRegistry.TEMPLATE, TemplateColumnMapping.getInstance());
mapping.put(SqlFileRegistry.EXPERIMENT_WITH_TEMPLATE, ExperimentTemplateColumnMapping.getInstance());
```

8. Since the existing “generated\_experiments” table did not have an entry for the problem set 566374, a row was manually inserted into the analytic\_db\_test.public.generated\_experiments table by the following SQL query:

```
INSERT INTO public.generated_experiments(
  id, sequence_id, experiment_template_id, creator_user_id, created_at, updated_at, note, parameters)
VALUES (14, 566374, 3, 337537, '2019-03-09 11:13:53.73-05', '2019-03-09 11:13:53.73-05', 'created manually by Junbong Jang', '');
```

As a result of the above step, the “generated\_experiemment” table has an additional row at the

end that maps the problem set 566374 with the template 3, which is a “video check, forced content early” template as shown below.

id	sequence_id	experiment_template_id	creator_user_id	created_at	updated_at	note	parameters
integer	integer	integer	integer	timestamp with time zone	timestamp with time zone	text	text
2	2	967829	5	2019-02-22 11:13:58.3-05	2019-02-22 11:13:58.3-05	TODO	TODO
3	3	967830	4	2019-02-22 11:14:01.437-05	2019-02-22 11:14:01.437-05	TODO	TODO
4	4	967831	3	2019-02-22 11:14:04.572-05	2019-02-22 11:14:04.572-05	TODO	TODO
5	5	967832	3	2019-02-22 11:14:05.48-05	2019-02-22 11:14:05.48-05	TODO	TODO
6	6	967833	3	2019-02-22 11:14:08.409-05	2019-02-22 11:14:08.409-05	TODO	TODO
7	7	967834	3	2019-02-22 11:14:09.363-05	2019-02-22 11:14:09.363-05	TODO	TODO
8	10	967837	6	2019-02-22 11:14:23.761-05	2019-02-22 11:14:23.761-05	TODO	TODO
9	13	967841	6	2019-02-22 11:22:11.589-05	2019-02-22 11:22:11.589-05	TODO	TODO
10	14	566374	3	2019-03-09 11:13:53.73-05	2019-03-09 11:13:53.73-05	creat...	

9. In ALIDumper.java, copy and paste the `getProblemSetStructure(Integer sequenceId)` function to create `getTemplate(Integer template_id)` and `getExperimentWithTemplate(Integer sequence_id)` which returns the SQL data parsed into the DataSet Java object.
10. Since ALIDumper implements Runnable interface, and a thread is started inside main function, the following code added inside `public void run()` function of ALIDumper.java will be ran. The code starts with getting parsed SQL data from the “experimentWithTemplate” table and finding the template ID associated with the specific problem set ID. Then, takes only the name and description of the specified template ID from the “template” table. At the end, two String variables are added together in a HTML5 String format.

```
DataSet dumpExperimentWithTemplate = getExperimentWithTemplate(566374);
int experiment_template_id_index = dumpExperimentWithTemplate.getHeaderColumnIndexByString("experiment_template_id");
String template_id = dumpExperimentWithTemplate.getColumnWithoutHeader(experiment_template_id_index).get(0);

DataSet dumpTemplate = getTemplate(Integer.parseInt(template_id));
int template_name_index = dumpTemplate.getHeaderColumnIndexByString("name");
int template_description_index = dumpTemplate.getHeaderColumnIndexByString("description");

String template_name = dumpTemplate.getColumnWithoutHeader(template_name_index).get(0);
String template_description = dumpTemplate.getColumnWithoutHeader(template_description_index).get(0);
String template_result = template_name + "<br>" + template_description;
```

## 3. Results

### 3.1 Web Application Final Version

The Python library application in conjunction with the Flask web framework is organized to follow best practices according to [23] and [24]. Run.py and config.py are needed to run Flask. `__init__.py` has the configurations of the Flask application. Views.py is used to generate the webpage through html template files. `Apa_formatter.py` is used to modify the data according to the APA format. `Multiple_regression.py` preprocesses and analyzes the data. All the code is provided in the appendix section.

Figure 10 below shows the current file structure of the Data analysis Flask application. Static folder have Html5, Javascript, Css and other image files that are used for displaying webpages. Templates folder contain base template which is extended by three child templates so there are three different webpages in total. The uploads folder will contain all the dataset ever uploaded by the users. App and ALI Analyzer folders contain the Python files which functions as a server that performs data analysis and send result to the web application.

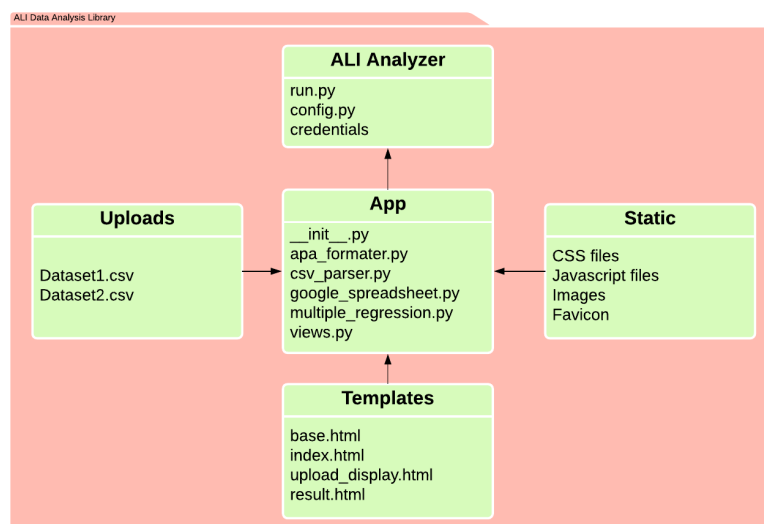


Figure 12 - Web Application File Structure

The web application is comprised of three pages as shown in Figure 11 below. The first page

asks the user to submit a CSV file and select the type of linear model to be used. The second page opens the submitted csv file in google spreadsheet for viewing and editing. It also asks the researcher to assign independent and dependent variables as inputs for data analysis. The third page displays the results. When multiple linear regression is performed, these results are comprised of the descriptive statistics, correlation matrix, multicollinearity statistics, coefficients statistics, ANOVA output, and APA formatted tables.

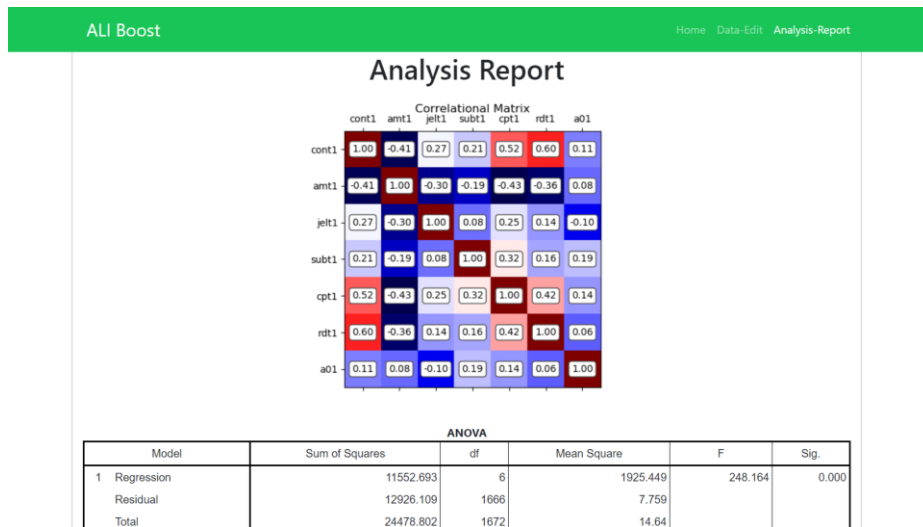
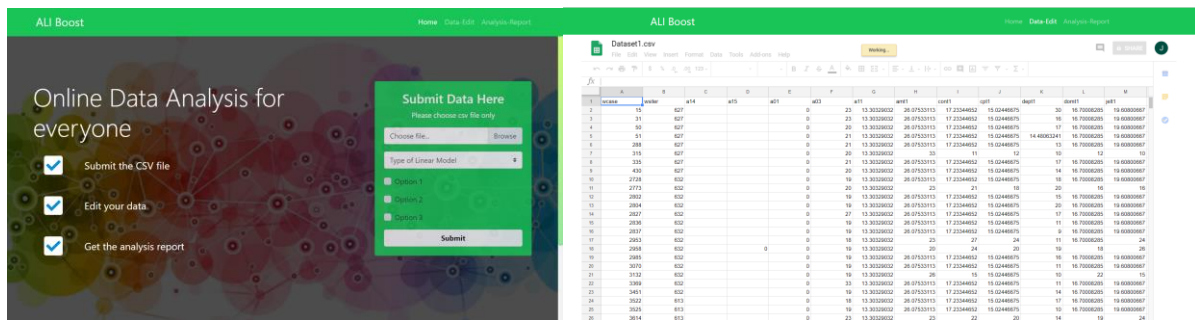


Figure 13 - Three Webpages of the Web Application

Inside the web application, Python library will analyze the dataset upon request from the webpage and present the result to the webpage. When working on producing the test statistics on the dataset, the multiple linear regression analysis was performed in three different ways to validate the result. The test statistic is a random variable that is calculated from sample data and used in a hypothesis test to determine whether to reject the null hypothesis [19]. Two results were obtained from StatsModels and Scikit-Learn, and the last result was obtained from the manual calculation in

Python.

## 3.2 Python Data Analysis

### 3.2.1 Multiple Linear Regression Output

In the context of linear regression, both `sklearn.linear_model` and `Statsmodel.api` modules provide estimation by ordinary least squares (OLS), which is a type of linear least squares method for estimating the unknown parameters in a linear regression, and weighted least squares (WLS). But only `Statsmodel.api` provides additional options such as generalized least squares (GLS), and feasible generalized least squares with autocorrelated AR(p) errors [18]. Two screenshots below represent the Python code using `StatsModels` package and `scikit-learn` package for linear regression.

```
def ols_linear_regression(self, X, y):
    print('-----Statsmodel Linear Regression-----')
    import statsmodels.api as sm # import statsmodels
    X = sm.add_constant(X) # statsmodels doesn't automatically fit a constant
    ols_model = sm.OLS(y, X).fit()

    print('-----OLS Statistics-----')
    # print(ols_model.summary())
    print(ols_model.mse_model)
    print(ols_model.mse_resid)
    print(ols_model.mse_total)
    print(ols_model.fvalue)
    print(ols_model.params)
```

*Code 4 - Multiple Linear Regression using Statsmodel*

```
def scikit_linear_regression(self, X, y):
    print('-----Scikit Linear Regression-----')
    # print(feature_selection.f_regression(X, y.values.ravel()))

    lm = linear_model.LinearRegression()
    scikit_model = lm.fit(X, y)
    scikit_predictions = lm.predict(X) # make a prediction
    # R^2 = SSR / SST, R^2 is a variance explained by the model
    print(lm.score(X, y))
    print(lm.coef_)
    print(lm.intercept_)
```

*Code 5 - Multiple Linear Regression using Scikit-Learn*



Computations used for manual calculation are show below. In one-way ANOVA, the F-statistic is this ratio [20]:

$$F = \text{Mean Square Error of the model} / \text{Mean Square Error of Residuals}$$

$$R^2 = \text{RegSS}/\text{TotSS} = (\text{TotSS}-\text{ResSS})/\text{TotSS} = 1 - \text{ResSS}/\text{TotSS}$$

$$\text{TotSS} = \text{RegSS} + \text{ResSS}$$

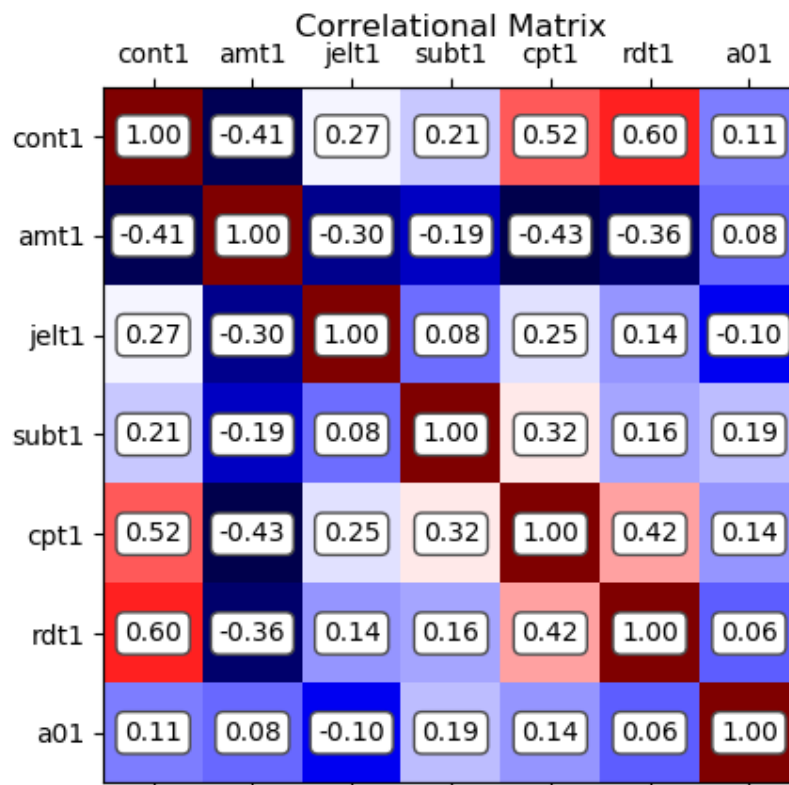
Where TotSS is the total sum of squares, RegSS is the regression sum of squares, and ResSS is the residual sum of squares. The problem with  $R^2$  is that it can only increase as predictors are added to the regression model. This is the problem when the added predictors do not improve the model's fit, so the adjusted  $R^2$  is computed also [21].

$$\text{adj } R^2 = 1 - (\text{ResSS}/\text{ResDF})/(\text{TotSS}/(n-1))$$

Output Comparison between python data analysis library and SPSS.

### 3.2.2 Multiple Linear Regression Output on HTML5 Webpage

Results generated from performing multiple linear regression on data are shown below. Advantage of Python data analysis library is that the precision ranges more than SPSS output. The result is different from example outputs in SPSS because the python library took  $N = 1673$  for total size of the data after dropping the missing rows in the data listwise, compared to  $N = 1677$  in SPSS dropping the missing data case wise. One drawback for python library is that when the dataset is converted from SPSS, the parameter labels are lost, and their actual parameter names are used instead, so amt1 instead of anger management. This can be fixed if the user manually renames the parameter names located in the header of the csv dataset to more readable names. Also, Java code in ALI can be modified such that ALI sends the dataset with appropriate names and description for each column of the dataset.



## ANOVA

Model	Sum of Squares	df	Mean Square	F	Sig.
1 Regression	11552.693	6	1925.449	248.164	0.000
Residual	12926.109	1666	7.759		
Total	24478.802	1672	14.64		

## Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	Change Statistics				
					R Square Change	F Change	df1	df2	Sig. F Change
1	0.687	0.472	0.47	0.687	0.472	248.164	6	1666	0.000

## Coefficients

Model	Unstandardized Coefficients		Standardized Coefficients	t	Sig.	Correlations			Collinearity Statistics	
	B	Std. Error	Beta			Zero-order	Partial	Part	Tolerance	VIF
1 const	7.627	1.097	0.000	6.952	0.000	1.0	1.000		0.004	259.560
amt1	-0.148	0.028	-0.111	-5.281	0.000	-0.407	-0.128		0.722	1.384
jelt1	0.126	0.019	0.124	6.532	0.000	0.273	0.158		0.885	1.130
subt1	0.018	0.017	0.020	1.059	0.290	0.209	0.026		0.870	1.149
cpt1	0.298	0.027	0.245	11.234	0.000	0.517	0.265		0.667	1.500
rdt1	0.399	0.019	0.429	21.303	0.000	0.596	0.463		0.782	1.279
a01	0.517	0.156	0.062	3.323	0.001	0.106	0.081		0.918	1.089

## APA formatted tables

Means, Standard Deviations, and Intercorrelations for cont1 and the Predictor Variables

Variable	<i>M</i>	<i>SD</i>	1	2	3	4	5	6
cont1	17.233	3.826	-0.407**	0.273**	0.209**	0.517**	0.596**	0.106**
Predictor Variable								
1. amt1	26.075	2.861	--					
2. jelt1	19.608	3.764	-0.296**	--				
3. subt1	13.999	4.28	-0.193**	0.084**	--			
4. cpt1	15.019	3.146	-0.428**	0.245**	0.316**	--		
5. rdt1	15.35	4.116	-0.362**	0.141**	0.159**	0.419**	--	
6. a01	0.296	0.457	0.077*	-0.095**	0.193**	0.14**	0.062*	--

Regression Analysis Summary for the Predictors of cont1

Variable	<i>B</i>	<i>SE B</i>	$\beta$	<i>t</i>	<i>p</i>	Squared Semi-partial Correlation	Structure Coefficient
amt1	-0.148	0.028	-0.111	-5.281	0.000		
jelt1	0.126	0.019	0.124	6.532	0.000		
subt1	0.018	0.017	0.020	1.059	0.290		
cpt1	0.298	0.027	0.245	11.234	0.000		
rdt1	0.399	0.019	0.429	21.303	0.000		
a01	0.517	0.156	0.062	3.323	0.001		

### 3.2.3 Limiting the number of decimal points

Generally, showing values such as Means, SDs, coefficients, SEs, and t statistics with two decimal places is enough. However, multiple linear regression analysis above gave every result in 3 decimal places. In order to reduce the decimal places to two, the snippet of code below was modified to round to 2 decimal places instead of 3. P value is kept at 3 decimal places because that is the norm.

```

model_stat_dict = {
    'model_ss': model_ss.round(2),
    'residual_ss': residual_ss.round(2),
    'total_ss': total_ss.round(2),
    'model_mse': ols_model.mse_model.round(2),
    'residual_mse': ols_model.mse_resid.round(2),
    'total_mse': ols_model.mse_total.round(2),
    'fvalue': ols_model.fvalue.round(2), # F-statistic of the fully specified model
    'pvalue': "{0:.3f}".format(ols_model.f_pvalue), # p-value of the F-statistic
    'model_df': "{0:.0f}".format(ols_model.df_model),
    'residual_df': "{0:.0f}".format(ols_model.df_resid),
    'total_df': "{0:.0f}".format(ols_model.df_model + ols_model.df_resid),
    'rvalue': "{0:.2f}".format(math.sqrt(ols_model.rsquared)),
    'rsquared': ols_model.rsquared.round(2),
    'rsquared_adj': ols_model.rsquared_adj.round(2),
}

```

*Code 6 - Multiple Linear Regression precision*

As a temporary measure to limit the number of floating points for other analysis such as ANCOVA and MANOVA, the `forg()` function located in “AppData/Local/Programs/Python/Python36/Lib/site-packages/statsmodel/iolib/summary.py” of the StatsModel library was edited to round all input x value to two decimal places. As shown in code 7 below.

```

def forg(x, prec=2):
    if prec == 3:
        #for 3 decimals
        if (abs(x) >= 1e4) or (abs(x) < 1e-4):
            return '%9.3g' % x
        else:
            return '%9.3f' % x
    elif prec == 4:
        if (abs(x) >= 1e4) or (abs(x) < 1e-4):
            return '%10.4g' % x
        else:
            return '%10.4f' % x
    elif prec == 2: # special case added by JJ
        return round(x, 2)
    else:
        raise NotImplementedError

```

*Code 7 - StatsModel Summary precision*

### 3.2.4 Data Preprocessor

Data Preprocessor class handles the missing data and assigns numerical data type if the column of data is continuous and map the string into nominal values 0, 1, 2, 3, etc. for categorical data. As shown in code 8 below, the code starts by storing input parameter values into local object variables. Then, the input data is preprocessed by converting all values representing null values in ALI database such as '-' into empty String, assigning generic object variables into categorical or continuous variables, and dropping all the rows with at least one column of empty String. At the end, preprocessed input dataframe is divided into independent dataframe variable, dependent dataframe variable. Dataframe is a word used by Pandas Python library to refer to the special data type that can be easily manipulated by the operations defined in Pandas.

```

class Data_Preprocessor(object):
    def __init__(self, input_df,
                 filename="Dataset1.csv",
                 covariate_columns=[],
                 x_columns=['rct1', 'rdt1', 'all'],
                 y_columns=['a14'],
                 categorical_columns=[],
                 missing_data='drop'):

        self.cov_columns = covariate_columns
        self.x_columns = x_columns
        self.y_columns = y_columns
        self.categorical_columns = categorical_columns
        self.all_columns = covariate_columns + x_columns + y_columns
        # ----- Data Preprocessing -----
        if input_df is None:
            dataset_path = '../resources/uploads/' + filename
            self.input_df = pd.read_csv(dataset_path)
        else:
            self.input_df = input_df

        self.handle_missing_data()
        self.handle_categorical_columns(categorical_columns)
        self.convert_to_correct_type()
        self.drop_missing_data(missing_data)

        # ----- Variables Defined -----
        self.X = self.input_df[self.x_columns]
        self.y = self.input_df[self.y_columns]
        self.feature_df = self.input_df[self.x_columns + self.y_columns + self.cov_columns]

```

*Code 8 - Data Preprocessor*

The current limitation of this class is that the names for independent, dependent and covariate columns need to be pre-specified in the Python source code to understand the analysis request from ALLI. Therefore, the analysis is limited to only predefined set of columns.

### 3.2.5 Generic Analysis Function

StatsModel library supports R-style equation as shown below. R is a programming language for statistical computing and has an intuitive way to represent an equation.

```
formula = 'subt1 ~ C(a01) + C(a08) + dept1'
```

Variable names on the left side of ~ represent dependent variables. Variable names on the right side of ~ represents independent and covariate variables. C() around the variable name indicates that the variable is a categorical variable. In special case such as mediation analysis, “variable1:variable2” is used to indicate the interaction term between those two variables. This is R-style notation is convenient for extending the current anova function’s capability because simply listing multiple dependent variables on the left side of the equation make the formula for MANOVA analysis. Also, the way covariate or independent variable are put in R-style equation is the same so the ANOVA function in code 7 can be used to do ANCOVA analysis.

The following ANOVA analysis function accepts different number of independent, dependent and covariate variables and dynamically create a R-style formula that can be used in analysis. Then, the formula and the input dataframe is passed to the ANOVA analysis function provided by StatsModel library and gets the ANOVA model that has been fitted to the data. At the end, the summary of the ANOVA analysis is obtained in HTML5 format. Snippet of the ANOVA analysis function is used to explain how the analysis is performed in Python but the same pattern is used inside the analysis function for multiple linear regression and MANOVA functions also.

```
def run_anova(self):
    formula_y = self.make_formula(self.data.y_columns)
    formula_x = self.make_formula(self.data.x_columns)
    formula_cov = self.make_formula(self.data.cov_columns)

    formula = 'subt1 ~ C(a01) + C(a08) + dept1'
    if formula_cov != '':
        formula = ("{y} ~ {x} + {cov}".format(y=formula_y, x=formula_x, cov=formula_cov))
    else:
        formula = ("{y} ~ {x}".format(y=formula_y, x=formula_x))

    print(formula)
    lm = ols(formula, self.data.feature_df).fit()
    print(lm.summary())

    return lm.summary().as_html()
```

Code 7 - ANOVA analysis function

### 3.3 ALI Doc Report

ALI Doc generation, connection to the Template Tool database, and Python Data Analysis Library working coherently together is proven through the ALI Doc generation. ALI is the main program that runs first. Then, it retrieves the template data associated with the current problem set from the template database. The template data is used to determine which analysis to perform and send appropriate analysis request to Python Flask Server. The server responds by performing requested analysis with the input data and sending the analysis result back to ALI. ALI then uses the result to populate the ALI Doc template file and send the generated ALI Doc Report to the user's email.

Name and description of the template related to given problem set 5663744 and Multiple Linear Regression, ANCOVA, and MANOVA analysis results are printed onto the ALI Doc report. Below is the analysis section added to the original ALI Doc Report in Appendix A. The style and format of the tables are less legible and neat compared to the analysis result presented on the Web Application in section 3.1. The variables for analysis are not chosen for analytical value but for demonstrating the capability of the automatic data analysis.

Multiple Linear Regression analysis is performed using independent variable: Prior Percent Correct and dependent variable: Z-Scored Mastery Speed. These two variables are chosen because they are continuous variables. ANOVA tests whether the model is significantly better at predicting the outcome than using the mean as a 'best guess', and the F value (26.324, 1) represents the ratio of the improvements in prediction that results from fitting the model. Since its significance p value is less than 0.001, the multiple linear regression model is significant. From the analysis result, Prior Percent Correct. Coefficients table shows whether variables in the analysis are significant predictors of conflict. First, the t-statistics column and its significance are checked to see if any predictor has significance greater than 0.05. Independent variable has the significance value less than 0.001 so Prior Percent Correct is a significant predictor for Z-Scored Mastery Speed.



ANCOVA is performed using independent variable: Correct Problem 1, dependent variable: Correct Problem 8, and covariate variable: Prior Percent Correct. Prior Percent Correct data is obtained from the featureReport variable which only stores the covariate data of students in ALIDefinedDataSets.java. Finally, MANOVA is performed using independent variable: Number Correct Problems in Posttest and dependent variables: Correct Problem 2, Hint Count Problem 2, Attempt Count Problem 2.

## Python Data Analysis Section

### Template Report

skill\_builder\_force\_content\_early

This pattern uses a trick to force the modified content to appear as soon as the student starts the skill builder

### Multiple Linear Regression

Independent Variable: Prior Percent Correct

Dependent Variable: Z-Scored Mastery Speed

#### ANOVA

Model	Sum of Squares	df	Mean Square	F	Sig.
1 Regression	5.68	1	5.68	26.32	0.000
Residual	126.03	584	0.22		
Total	131.71	585	0.23		

#### Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	Change Statistics				
					R Square Change	F Change	df1	df2	Sig. F Change
1	0.21	0.04	0.04	0.21	0.04	26.32	1	584	0.000

#### Coefficients

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.	Correlations			Collinearity Statistics	
		B	Std. Error	Beta			Zero-order	Partial	Part	Tolerance	VIF
1	const	0.54	0.11	-0.00	4.91	0.00	1.0	1.00		0.03	32.82
	independent1	-0.81	0.16	-0.21	-5.13	0.00	-0.208	-0.21		1.00	1.00

APA formatted tables

Means, Standard Deviations, and Intercorrelations for independent1 and the Predictor Variables

Variable	M	SD	1
independent1	0.689	0.122	-0.208**
Predictor Variable			
1. dependent1	-0.015	0.474	--

Regression Analysis Summary for the Predictors of independent1

Variable	B	SE B?	t	p	Squared Semi-partial Correlation	Structure Coefficient
independent1	-0.81	0.16	-0.21	-5.13	0.00	

## ANCOVA

Independent Variable: Correct Problem 1  
 Dependent Variable: Correct Problem 8  
 Covariate Variable: Prior Percent Correct

### OLS Regression Results

Dep. Variable:	dependent	R-squared:	0.010
Model:	OLS	Adj. R-squared:	0.006
Method:	Least Squares	F-statistic:	2.702
Date:	Mon, 11 Mar 2019	Prob (F-statistic):	0.0680
Time:	17:11:28	Log-Likelihood:	-390.50
No. Observations:	552	AIC:	787.0
Df Residuals:	549	BIC:	799.9
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P> t	[0.025	0.975]
Intercept	0.62	0.13	4.85	0.000	0.37	0.87
independent	-0.1	0.04	-2.31	0.021	-0.19	-0.02
covariate	0.04	0.18	0.21	0.831	-0.32	0.39

Omnibus:	9.473	Durbin-Watson:	1.997
Prob(Omnibus):	0.009	Jarque-Bera (JB):	88.817
Skew:	-0.324	Prob(JB):	5.17e-20
Kurtosis:	1.145	Cond. No.	14.8

Warnings:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

## MANOVA

Independent Variable: Number Correct Problems in Posttest

Dependent Variable: Correct Problem 2, Hint Count Problem 2, Attempt Count Problem 2

Intercept	Value	Num DF	Den DF	F Value	Pr > F
Wilks' lambda	0.0371	3.0000	81.0000	699.9491	0.0000
Pillai's trace	0.9629	3.0000	81.0000	699.9491	0.0000
Hotelling-Lawley trace	25.9240	3.0000	81.0000	699.9491	0.0000
Roy's greatest root	25.9240	3.0000	81.0000	699.9491	0.0000

independent1	Value	Num DF	Den DF	F Value	Pr > F
Wilks' lambda	0.9408	3.0000	81.0000	1.6984	0.1739
Pillai's trace	0.0592	3.0000	81.0000	1.6984	0.1739
Hotelling-Lawley trace	0.0629	3.0000	81.0000	1.6984	0.1739
Roy's greatest root	0.0629	3.0000	81.0000	1.6984	0.1739

## 4. Discussion

### 4.1 Hardship

The hardest part of this project was the beginning when I did not know fully understand what ALI is and where to start. I was hesitant to start to write any code because I wanted to design the architecture of the library carefully so that it can be easily maintained and used for many years. Refactoring and restructuring the already developed library because of its bad design or hidden programmatic errors would surely be time consuming for programmers and frustrating for ALI users working with the library. In order to overcome the hardship, I first became more familiar with the Python by reading a book called “Fluent Python” by Ramalho, Python online tutorials and learning about various Python libraries such as Numpy, Pandas, Scikit-learn and StatsModels.

Then, I started to work together with ASSISTments researchers, Anthony Botelho and Thanaporn “March” Patikorn. Anthony is the graduate student who oversees the development of ALI and he taught me how the ALI code is structured and the best way to integrate ALI with my python data analysis library. Thanaporn is a graduate student who built Template tool and database to store user’s individual template so that the data can be automatically analyzed when generating ALI Doc. One of the problem with ALI was that the source code barely had any comments and documentations such as java doc or class diagram that describe what each class, or functions do were nonexistent. Therefore, learning process was slow but they answered my questions to enhance my learning and helped me to overcome the hardest part of the project.

The second hardest part was the slow testing and development cycle when working with ALI Doc generation. Running ALIDumper.java file to test how ALI communicate with my Python library to generate ALI Doc was very slow. The first time sending the ALI Doc request and receiving ALI Doc in email takes about 15 minutes by average. And subsequent ALI Doc request and email reception takes about 1 minute and 50 seconds. The bottleneck of ALI Doc generation is shown in the snippet below. Two functions dumpProblemLevel(), and dumpFeatureLevel() retrieves the data from the Postgres database but since their long query asks for so much data at once, MySQL database can’t process them quickly. To improve this situation, one needs to send analysis request to Python library server with predefined local data in ALI code so that the SQL query does not need to be sent to MySQL database.

```
long previous_time = System.nanoTime();
DataSet dumpProblem = dumpProblemLevel(sequenceList);
System.out.println(System.nanoTime() - previous_time);
DataSet dumpFeature = dumpFeatureLevel(sequenceList);
System.out.println(System.nanoTime() - previous_time);
```

*Code 1 - ALI bottleneck*

The next hardship was figuring out how to get all the statistics that SPSS produces from various Python libraries. Out of many different choices, I narrowed down my selection to using the most popular libraries for statistical analysis: StatsModels and Scikit-learn. I tried to produce the same

statistics found in SPSS with three different methods: first using StatsModels, second using Scikit-Learn, and third without using them. I could obtain the same results from each of the three different methods but StatsModels was the best method because it produced most of the results found in SPSS and used convenient python dictionary data type, "StatsModels Instance". In summary, the best way to build the library was learned from trial and error of testing many ways.

## 4.2 Limitations & Future Improvements

Currently, the ALI project can request ANOVA, ANCOVA, MANOVA, MANCOVA, and multiple linear regression analysis to Python Server which in term analyze the input data and return the output in String. ALI project can also connect to Template Tool Database to check the type of template associated with the specified sequence ID. According to Anthony Botelho, current status of the project is in phase 1 and there are many ways it can be improved. Future phases will mainly be about improving or combining current Template Tool with my web application, adding more data analysis capability, making ALI Doc report look more professional and providing more individualized analysis for each type of templates.

### 4.2.1 Web Application

On the first page of the web application, more options should be provided for users to perform linear regression in various ways such as using Hierarchical, Forced Entry, or Stepwise methods. On the second page, data editing functionality should be improved so that the user can edit multiple values on the spreadsheet simultaneously. Possible improvements include having the user choose either to drop missing values or fill them with another value for the purposes of trimming or imputation. On the third page, 'copy to clipboard' buttons will be placed next to the tables and graphs so that they can easily be pasted onto documents in Microsoft Word to facilitate writing research papers. The tables and graphs will be also modifiable in the result page so that users can change them before copying them as images. The tables and graphs will be partitioned into related groups (e.g.,

coefficients statistics, correlation statistics, etc.) and they will be minimized to tabs on the result page, allowing users to choose to expand only the tabs that interest them.

#### 4.2.2 Python Data Analysis Library

In the Python data analysis library, multiple linear regression, logistic regression, mediation and moderation, ANOVA/ANCOVA, MANOVA/MANCOVA, and factor analysis are implemented. However, Python Server do not have functions to respond to the logistic regression, mediation, and factor analysis requests so those functions should be added. As shown below in Figure 14 (obtained from [22]), before fitting a model to the data, initial checks for linearity and unusual cases should be performed. Especially, Levene's test should be performed for ANOVA to decide whether to run ANOVA. Then, after fitting a model, linearity, homoscedasticity, independence, and normality should be checked to correctly interpret the result from the model. Therefore, these checks before and after linear regression will be added to the data analysis library to notify the user if customary assumptions have been violated.

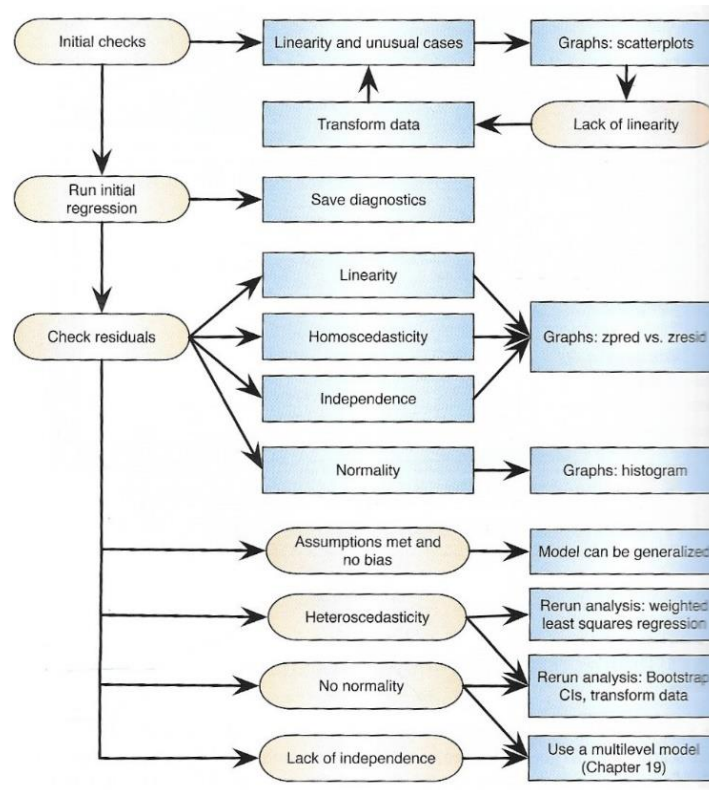


Figure 14 - Process of Performing a Linear Regression

### 4.2.3 ALI Document Report

The style and format of the tables look less aligned and neat compared to the analysis result presented in Webpage but it can be fixed by converting tables in text into image and adding that image to ALI Doc. The analysis result is not in APA format except for multiple linear regression, so other analysis can be formatted in APA in addition to the current analysis outputs. ALI Doc shown in section 3.3 shows that the floating point precision problem is resolved for Multiple Linear Regression analysis and partially for ANCOVA, but not at all for MANOVA. Also, modifying the StatsModel library code is not reliable way to fix problems since installing a new version of Statsmodel or other people who pulled from the repository will not have the modifications I made in StatsModel library in my local machine. Therefore, ANCOVA and MANOVA function will store each individual test statistics in the variables and round each of them separately, which is how Multiple Linear Regression function does to round every number to 2 decimal places.

#### 4.2.4 Future

As stated on section 1.2 of this paper, the goal was to make a reliable and usable Python data analysis library with longevity. So far, the source code was commented appropriately, and UML class diagram was created to describe the major part of the ALI project structure. Most of the steps I took to make this project work has been documented in this IQP report so anyone who wants to develop the ALI or Python Library further can do so with my work. I hope to have as many people as possible benefit from my work, so easy to use user interface should be made, unlike my web application which stopped in the middle of the development. In the future, it would be useful to add the automatic data analysis feature that takes the user's skill builder problem set and recommends the most suitable linear model type for that problem set. Even though considerable amount of time and effort have been devoted to this project, ALI and Python analysis library still have countless ways to be improved and have a potential to be an even better research tool.

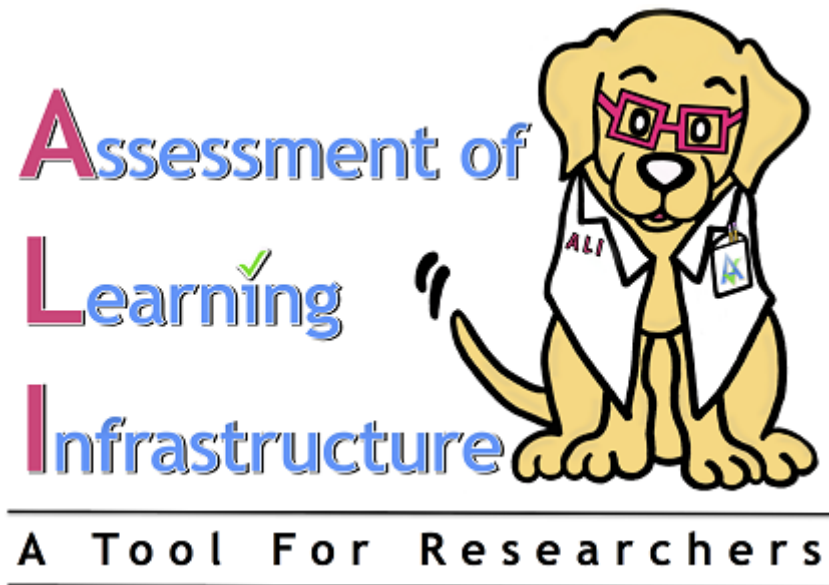


## References

- [1] Sales, A., Botelho, A. F., Patikorn, T., & Heffernan, N. T. (2018, July). Using Big Data to Sharpen Design-Based Inference in A/B Tests. In Proceedings of the Eleventh International Conference on Educational Data Mining, 479-485.
- [2] Ostrow, K.S., Heffernan, N.T., & Williams, J.J. (2017). Tomorrow's EdTech Today: Establishing a Learning Platform as a Collaborative Research Tool for Sound Science. Teachers College Record, 119 (3), 1-36.
- [3] Ostrow, K.S. (2016). Toward a Sound Environment for Robust Learning Analytics. Included in the Learning Analytics and Knowledge 2016 Doctoral Consortium
- [4] Ostrow, K., Selent, D., Wang, Y., Van Inwegen, E., Heffernan, N., & Williams, J.J. (2016). The Assessment of Learning Infrastructure (ALI) The Theory, Practice, and Scalability of Automated Assessment. In Proceedings of the 6th International Conference on Learning Analytics & Knowledge, 279-288. ACM.
- [5] Robinson, D. (2017, September 14). Why is Python Growing So Quickly? Retrieved from <https://stackoverflow.blog/2017/09/14/python-growing-quickly/>
- [6] Timo, R. (2017, September 27). Why Is Python Good For Research? Benefits of the Programming Language. Retrieved from <https://www.netguru.co/blog/why-is-python-good-for-research-benefits-of-the-programming-language>
- [7] Park, M. (2016, October 24). Rigorous SRI Study Shows Online Mathematics Homework Program Developed at Worcester Polytechnic Institute Increases Student Achievement. Retrieved from <https://www.sri.com/newsroom/press-releases/rigorous-sri-study-shows-online-mathematics-homework-program-developed>
- [8] Koedinger, K. R., Baker, R. S., Cunningham, K., Skogsholm, A., Leber, B., & Stamper, J. (2010). A data repository for the EDM community: The PSLC DataShop. Handbook of EDM, 43.
- [9] <https://docs.python.org/release/3.7.1/license.html>
- [10] Drake, F. L. (2018, October 20). Unittest - Unit testing framework. Retrieved October 25, 2018, from <https://docs.python.org/3/library/unittest.html>
- [11] Peters, T. (2004, August 19). PEP 20 -- The Zen of Python. Retrieved October 25, 2018, from <https://www.python.org/dev/peps/pep-0020/>
- [12] Bronshtein, A. (2017, April 26). A Quick Introduction to the NumPy Library – Towards Data Science. Retrieved October 25, 2018, from <https://towardsdatascience.com/a-quick-introduction-to-the-numpy-library-6f61b7dee4db>
- [13] Bronshtein, A. (2017, April 18). A Quick Introduction to the "Pandas" Python Library. Retrieved October 25, 2018, from <https://towardsdatascience.com/a-quick-introduction-to-the-pandas-python-library-f1b678f34673>
- [14] Mock, L. (2015, March 07). How to Flowchart, Four Most Common Flowchart Types (Part 3 of 3). Retrieved October 25, 2018, from <https://www.glimfy.com/blog/how-to-flowchart-four-most-common-flowchart-types-part-3-of-3>

- [15] Ceta, N. (2018, September 14). All You Need to Know About UML Diagrams: Types and 5 Examples. Retrieved October 25, 2018, from <https://tallyfy.com/uml-diagram/>
- [16] Bronshtein, A. (2017, May 08). Simple and Multiple Linear Regression in Python – Towards Data Science. Retrieved October 25, 2018, from <https://towardsdatascience.com/simple-and-multiple-linear-regression-in-python-c928425168f9>
- [17] Boland, S. (2018, July 18). Scikit-learn vs. StatsModels: Which, why, and how? Retrieved from <https://blog.thedataincubator.com/2017/11/scikit-learn-vs-statsmodels/>
- [18] Linear Regression. (n.d.). Retrieved from <https://www.statsmodels.org/stable/regression.html>
- [19] What is a test statistic? (n.d.). Retrieved from <http://support.minitab.com/en-us/minitab/17/topic-library/basic-statistics-and-graphs/hypothesis-tests/basics/what-is-a-test-statistic/>
- [20] How to Read the Output From Multiple Linear Regression Analyses. (n.d.). Retrieved from <http://www.jerrydallal.com/lhsp/regout.htm>
- [21] GRACE-MARTIN, K. (2018, May 10). Assessing the Fit of Regression Models. Retrieved from <https://www.theanalysisfactor.com/assessing-the-fit-of-regression-models/>
- [22] Field, A. P. (2013). Discovering statistics using IBM SPSS. London: Sage Publications.
- [23] Nzomo, M. (2018, November 20). Getting Started With Flask, A Python Microframework. Retrieved from <https://scotch.io/tutorials/getting-started-with-flask-a-python-microframework>
- [24] Picard, R. (n.d.). Organizing your project. Retrieved from <http://exploreflask.com/en/latest/organizing.html>
- [25] Vasudevan, K. (2017, April 18). Microservices, APIs, and Swagger: How They Fit Together. Retrieved from <https://swagger.io/blog/api-strategy/microservices-apis-and-swagger/>
- [26] Brown, M. (2016, May 25). Creating a RESTful API: Django REST Framework vs. Flask. Retrieved from <https://www.excella.com/insights/creating-a-restful-api-django-rest-framework-vs-flask>
- [27] Lowry, R. (n.d.). VassarStats: Statistical Computation Web Site. Retrieved from <http://vassarstats.net/>
- [28] Half, R. (2017, November 21). 6 Basic SDLC Methodologies: Which One is Best? Retrieved from <https://www.roberthalf.com/blog/salaries-and-skills/6-basic-sdlc-methodologies-which-one-is-best>
- [29] S. (n.d.). DB-Engines Ranking. Retrieved from [https://db-engines.com/en/ranking\\_trend](https://db-engines.com/en/ranking_trend)
- [30] Bui, A. (2018, December 05). PostgreSQL Vs. MySQL. Retrieved from <https://blog.panoply.io/postgresql-vs.-mysql>
- [31] Checking the Additional Assumptions of a MANOVA. (2015, May 11). Retrieved from <https://www.statisticssolutions.com/checking-the-additional-assumptions-of-a-manova/>
- [32] Janakiev, N. (2018, August 03). Understanding the Covariance Matrix. Retrieved from <https://datascienceplus.com/understanding-the-covariance-matrix/>

## Appendix A. Original ALI Doc Sample



Data Record for PSAWANE - Logs prior to January 6, 2019

Dear Researcher,

Welcome to the data record for problem set PSAWANE. You have received this record based on your recent data request. Automated data analysis is featured below, offering a preliminary overview of your sample and a selection of analyses for your consideration. The latter portion of this report contains the raw data files from which you can conduct your own thorough analyses. When publishing your work, please reference this report as a stable location for readers to access your data for review and replication.

By clicking any link to download content from this page, you are agreeing to our [Terms of Use](#).

### Automated Data Analysis

#### Completion Rates

Students that have started PSAWANE : 772

Students that have completed PSAWANE : 543

#### Bias Assessment

Before analyzing learning outcomes, we suggest first assessing potential bias introduced by your experimental conditions (i.e., examine differential dropout). The table below reports the number of students that have completed PSAWANE, split out by experimental condition.

Conditions	Students who started the problem set	Students who finished the problem set	Percent of students completed
treatment1	119	54	45

treatment2	102	59	58
N	221		
Degrees Of Freedom	1		
Chi-Squared	2.93		
P-Value	0.09		

### Mean and Standard Deviation of Posttest Score by Condition

To examine learning outcomes at posttest, an analysis of means was conducted across conditions. The tables below reports mean post-test score and standard deviation for each condition. This information was sourced from our automated [post-test subreport](#).

	Students who completed the problem set	Percent of students completing the problem set	PostTestScore*
treatment1	54	45	0.19 (0.28)
treatment2	59	58	0.23 (0.32)

\* Presented as Mean (SD)

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Control	1	0.08	0.08	0.89	0.35
Residual	219	19.29	0.09	NaN	NaN

## Raw Data Files

Raw data files contain the logged information for each student that has participated in your study (including those who have not completed the problem set and those who do not have available path information). We provide this data in a variety of formats, as explained below, to assist in your analytic efforts. We use Google Docs to share these files with you. If you would like to process these files manually, we recommend downloading the CSV file of your choice and resaving the file as an Excel spreadsheet or workbook to retain formatting and formulas. If you will be passing the file directly to a statistical package, downloading the CSV to a convenient location should suffice.

For a field glossary and tutorials on how to read each type of file, visit our [Data Dump Glossary page](#).

## Historical Data

[Covariate File](#) - A collection of useful covariates for the students who participated in your study. This file includes student level variables (i.e., gender), class level variables, (i.e., homework completion rates), and school level variables (i.e., urban city). Click [here](#) for a tutorial on how to link this file to your experimental data.

-

## Experimental Data

1. [Action Level](#) - One row per action per student; the finest granularity. Students participating in your study have performed 27513 actions (e.g., beginning problems, attempting to answer problems, asking for hints or tutoring, and eventually completing problems).
2. [Problem Level](#) - One row per problem per student. Students participating in your study have completed 6753 problems. The flow through a single problem incorporates many actions, resulting in a coarser data file (fewer rows).
3. [Student Level](#) - One row per student; the coarsest granularity. Columns are laid out in opportunity order to depict the student's progression through the problem set. Problem level information is expanded to one column per problem per field (column heavy).
4. [Student Level + Problem Level](#) - One row per field per student. Columns are laid out in opportunity order to depict the student's progression through the problem set. This is an alternative view of the student level information (row heavy).

If after consulting our glossary page you have trouble interpreting any of the above files, please feel free to email [assistments-data@wpi.edu](mailto:assistments-data@wpi.edu)

The ASSISTments Research Team

## Appendix B. Web Application & Python Data Analysis Library source code

### Python Code

```

# Author: Junbong Jang
# Date: 11/6/2018
# app/multiple_regression.py

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math

from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm # import statsmodels

from scipy import stats, linalg
from scipy.stats.mstats import zscore

from app.analysis.data_preprocessor import Data_Preprocessor

class Multiple_Regression(object):

    # when not running flask, change the path to 'uploads/Dataset1.csv'
    def __init__(self, data_preprocessor):
        self.data = data_preprocessor

    def calc_multiple_regression_stat(self):
        print('-----Correlation Matrix-----')
        print()
        partial_corr = self.calc_partial_corr(self.data.feature_df)[1]
        partial_corr.pop(0)
        # print('----- Multicollinearity Test -----')
        vif_df = self.calc_vif(self.data.X)
        # tolerance_df = vif_df['VIF Factor'].rdiv(1)
        multicol_list = []
        param_names = []
        for index, row in vif_df.iterrows():
            param_names.append(row['features'])
            multicol_list.append(("{0:.2f}".format(row['VIF Factor']), "{0:.2f}".format(1 / row['VIF Factor'])))

        # print('----- Linear Regression -----')
        # self.scikit_linear_regression(X, y)
        ols_model, model_ss, residual_ss, total_ss = self.ols_linear_regression(self.data.X, self.data.y)
        standardized_model, model_ss2, residual_ss2, total_ss2 = self.ols_linear_regression(zscore(self.data.X),
                                                                                       zscore(self.data.y))

        model_stat_dict = {
            'model_ss': model_ss.round(2),
            'residual_ss': residual_ss.round(2),
            'total_ss': total_ss.round(2),
            'model_mse': ols_model.mse_model.round(2),
            'residual_mse': ols_model.mse_resid.round(2),
            'total_mse': ols_model.mse_total.round(2),
            'fvalue': ols_model.fvalue.round(2), # F-statistic of the fully specified model
            'pvalue': "{0:.3f}".format(ols_model.f_pvalue), # p-value of the F-statistic
            'model_df': "{0:.0f}".format(ols_model.df_model),
            'residual_df': "{0:.0f}".format(ols_model.df_resid),
            'total_df': "{0:.0f}".format(ols_model.df_model + ols_model.df_resid),
            'rvalue': "{0:.2f}".format(math.sqrt(ols_model.rsquared)),
            'rsquared': ols_model.rsquared.round(2),
            'rsquared_adj': ols_model.rsquared_adj.round(2),
        }

        coefficients_dict = {
            'param_names': param_names,
            'unstandardized_beta': Multiple_Regression.round_list(ols_model.params.tolist()),
            'bse': Multiple_Regression.round_list(ols_model.bse.tolist()),
            'standardized_beta': Multiple_Regression.round_list(standardized_model.params.tolist()),

```

```

        'tvalues': Multiple_Regression.round_list(ols_model.tvalues.tolist()),
        'pvalues': Multiple_Regression.round_list(ols_model.pvalues.tolist()),
        'zero_order_corr': self.calc_zero_order_corr()[0][0],
        'partial_corr': Multiple_Regression.round_list(partial_corr),
        'semi_partial_corr': '',
        'multicol_list': multicol_list
    }
    return model_stat_dict, coefficients_dict

    @staticmethod
    def print_statistics(ols_model, standardized_model):
        # https://www.statsmodels.org/dev/generated/statsmodels.regression.linear_model.RegressionResults.html
        print('-----OLS Statistics-----')
        print(ols_model.summary())
        print(ols_model.mse_model)
        print(ols_model.mse_resid)
        print(ols_model.mse_total)
        print(ols_model.fvalue)
        print(ols_model.f_pvalue)
        print('----- Coefficients -----')
        print(ols_model.params) # coefficients
        print(ols_model.bse) # The standard errors of the parameter estimates.
        print(standardized_model.params) # standardized coefficients
        print(ols_model.tvalues) # t-test statistics on coefficients
        print(ols_model.pvalues) # coefficient's p-value

    @staticmethod
    def round_list(a_list):
        new_list = []
        for value in a_list:
            new_list.append("{0:.2f}".format(value))
        return new_list

    def ols_linear_regression(self, X, y):
        X = sm.add_constant(X)
        ols_model = sm.OLS(y, X).fit()
        model_ss, residual_ss, total_ss = self.calc_ss(ols_model, X, self.data.y)

        return ols_model, model_ss, residual_ss, total_ss

    def scikit_linear_regression(self, X, y):
        from sklearn import linear_model
        lm = linear_model.LinearRegression()
        scikit_model = lm.fit(X, y)
        scikit_predictions = lm.predict(X)

    def calc_ss(self, ols_model, X, y):
        ols_predictions = np.array(ols_model.predict(X).tolist())
        predictions = np.reshape(ols_predictions, (-1, 1))

        # sum of squared
        model_ss = sum(np.array
            ([[prediction - y.values.mean()] ** 2 for prediction in predictions]))
        residual_ss = sum((y.values - predictions) ** 2)
        total_ss = model_ss + residual_ss

        model_mse = model_ss / ols_model.df_model
        residual_mse = residual_ss / ols_model.df_resid

        # F is the ratio of the Model Mean Square to the Error Mean Square
        f_value = model_mse / residual_mse

        return model_ss[0], residual_ss[0], total_ss[0]

    def calc_vif(self, given_df):
        """
        https://etav.github.io/python/vif\_factor\_python.html
        The Variance Inflation Factor (VIF) is a measure of colinearity
        among predictor variables within a multiple regression.

        :param given_df:
        :return:
        """
        from statsmodels.tools.tools import add_constant
        given_df = add_constant(given_df)

```

```

vif = pd.DataFrame()
vif["features"] = given_df.columns
vif["VIF Factor"] = [variance_inflation_factor(given_df.values, i) for i in range(given_df.shape[1])]
return vif

def calc_partial_corr(self, C):
    """
    https://stats.stackexchange.com/questions/288273/partial-correlation-in-panda-dataframe-python
    Returns the sample linear partial correlation coefficients between pairs of variables in C, controlling
    for the remaining variables in C.
    Parameters
    -----
    C : array-like, shape (n, p)
        Array with the different variables. Each column of C is taken as a variable
    Returns
    -----
    P : array-like, shape (p, p)
        P[i, j] contains the partial correlation of C[:, i] and C[:, j] controlling
        for the remaining variables in C.
    """
    C = sm.add_constant(C)
    C = np.asarray(C)
    p = C.shape[1]
    P_corr = np.zeros((p, p), dtype=np.float)
    for i in range(p):
        P_corr[i, i] = 1
        for j in range(i + 1, p):
            idx = np.ones(p, dtype=np.bool)
            idx[i] = False
            idx[j] = False
            beta_i = linalg.lstsq(C[:, idx], C[:, j])[0]
            beta_j = linalg.lstsq(C[:, idx], C[:, i])[0]

            res_j = C[:, j] - C[:, idx].dot(beta_i)
            res_i = C[:, i] - C[:, idx].dot(beta_j)

            corr = stats.pearsonr(res_i, res_j)[0]
            P_corr[i, j] = corr
            P_corr[j, i] = corr

    return P_corr.tolist()

def draw_correlation_matrix(self, filename):
    """
    :return:
    """
    # get pearson's r and p values
    corr_matrix = self.data.feature_df.corr()

    # Plot Correlational Matrix heat map table
    fig, ax = plt.subplots()
    # Using imshow here just because it sets the ticks up nicely. imshow is faster.
    ax.imshow(corr_matrix, cmap='seismic')
    plt.xticks(range(len(self.data.feature_df.columns)), self.data.feature_df.columns)
    plt.yticks(range(len(self.data.feature_df.columns)), self.data.feature_df.columns)
    plt.title('Correlational Matrix')

    for (i, j), z in np.ndenumerate(corr_matrix):
        ax.text(j, i, '{:0.2f}'.format(z), ha='center', va='center',
                bbox=dict(boxstyle='round', facecolor='white', edgecolor='0.3'))
    # plt.show()
    fig.savefig('app/static/dynamic_img/correlation_'+filename+'.png') # save the figure to file
    plt.close(fig)

def calc_zero_order_corr(self):
    feature_df_rows, feature_df_cols = self.data.feature_df.shape
    import scipy.stats as ss

    corr_list = [ss.pearsonr(self.data.feature_df.values[:, i], self.data.feature_df.values[:, j])
                 for i in range(feature_df_cols)
                 for j in range(feature_df_cols)]
    correlation_values = np.transpose(np.array(corr_list))[0].round(3)
    correlation_p = np.transpose(np.array(corr_list))[1].round(5)

```



```

rows = correlation_values.shape
# I want correlation values in 2d array
corr_values_2d = correlation_values.reshape(int(math.sqrt(rows[0])), int(math.sqrt(rows[0])))
corr_p_2d = correlation_p.reshape(int(math.sqrt(rows[0])), int(math.sqrt(rows[0])))

return corr_values_2d.tolist(), corr_p_2d.tolist() # converted to list so that jinja2 iterate it in for loop

def calc_descriptive_dict(self):
    descriptive_dict = [{"name": column_name,
                        'mean': round(self.data.feature_df.mean().loc[column_name], 3),
                        'std': round(self.data.feature_df.std().loc[column_name], 3),
                        'count': round(self.data.feature_df.count().loc[column_name])} for column_name in
                        self.data.all_columns]

    return descriptive_dict

if __name__ == "__main__":
    data_preprocessor = Data_Preprocessor(missing_data='drop',
                                         x_columns = ['amt1', 'jelt1', 'subt1', 'cpt1', 'rdt1', 'a01'],
                                         predict_column = ['cont1'])
    multiple_regression_obj = Multiple_Regression(data_preprocessor)
    multiple_regression_obj.calc_multiple_regression_stat()

```

```

# Author: Junbong Jang
# Date: 1/23/2018
# mediation_moderation.py

import statsmodels.api as sm
import statsmodels.genmod.families.links as links
from statsmodels.stats.mediation import Mediation
import pandas as pd
from app.analysis.data_preprocessor import Data_Preprocessor

class Mediation_Moderation(object):

    def __init__(self, data_preprocessor_in):
        self.data = data_preprocessor_in

    def run_mediation(self, predict_col_categorical=False):
        if predict_col_categorical:
            probit = links.probit
            outcome_model = sm.GLM.from_formula("cont1 ~ rdt1 + jelt1", self.data.X, family=sm.families.Binomial(link=probit))
        else:
            outcome_model = sm.GLM.from_formula("cont1 ~ rdt1 + jelt1", self.data.X)

        mediator_model = sm.OLS.from_formula("rdt1 ~ jelt1", self.data.X)
        med = Mediation(outcome_model, mediator_model, "jelt1", mediator="rdt1").fit()
        with pd.option_context('display.max_rows', None, 'display.max_columns', None):
            print(med.summary())

if __name__ == "__main__":
    # homework 3
    data_preprocessor = Data_Preprocessor(missing_data='drop',
                                         filename="Dataset1.csv",
                                         predict_column=['rdt1'],
                                         x_columns=['rdt1', 'jelt1', 'cont1'])
    mediation_obj = Mediation_Moderation(data_preprocessor)
    mediation_obj.run_mediation()

```

```

# Author: Junbong Jang
# Date: 1/12/2018
# app/logistic_regression.py

import numpy as np
from app.analysis.data_preprocessor import Data_Preprocessor

```

```

class Logistic_Regression(object):

    def __init__(self, data_preprocessor):
        self.data = data_preprocessor

    # LogitResults object's attribute information obtained from
    # https://github.com/statsmodels/statsmodels/blob/master/statsmodels/discrete/discrete_model.py
    def calc_logistic_regression_stat(self):
        import statsmodels.api as sm
        predictor_with_constant = sm.add_constant(self.data.X)

        import statsmodels.discrete.discrete_model as sm
        logit = sm.Logit(self.data.y, predictor_with_constant)
        logit_model = logit.fit()
        print(type(logit_model))
        print(logit_model.summary())

        return logit_model.summary().as_html()
        # print(logit_model.params)
        # print(logit_model.conf_int())
        # print(np.exp(logit_model.params))
        # print(logit_model.llf)
        # print(logit_model.llnull)
        # print(logit_model.llr)
        # print(logit_model.llr_pvalue)
        # print(logit_model.df_model)
        # print(logit_model.df_resid)
        # print(logit_model.prsquared)
        # print(logit_model.aic)
        # print(logit_model.bic)
        # print(logit_model.bse)
        # McFadden's pseudo-R-squared. `1 - (llf / llnull)`

    def calc_freq_table(self, column_name):
        no_counter = 0
        yes_counter = 0
        missing_counter = 0
        for index, row in self.data.y.iterrows():
            column_value = row[column_name]
            if column_value == 0:
                no_counter += 1
            elif column_value == 1:
                yes_counter += 1
            elif np.isnan(column_value):
                missing_counter += 1

        valid_total = no_counter + yes_counter
        total_rows = valid_total + missing_counter

        no_percentage = no_counter / total_rows
        yes_percentage = yes_counter / total_rows
        missing_percentage = missing_counter / total_rows

        valid_no_percentage = no_counter / valid_total
        valid_yes_percentage = yes_counter / valid_total

        print(no_counter)
        print(yes_counter)
        print(missing_counter)
        print(valid_total)
        print(total_rows)

        print(no_percentage)
        print(yes_percentage)
        print(missing_percentage)

        print(valid_no_percentage)
        print(valid_yes_percentage)

if __name__ == "__main__":
    data_preprocessor = Data_Preprocessor(missing_data='drop')
    logistic_regression_obj = Logistic_Regression(data_preprocessor)
    logistic_regression_obj.calc_freq_table('a14')
    logistic_regression_obj.calc_logistic_regression_stat()

```

```

# Author: Junbong Jang
# Date: 1/12/2018
# app/anova_analysis.py
from __future__ import print_function
from statsmodels.formula.api import ols
from statsmodels.multivariate.manova import MANOVA
from scipy.stats import bartlett
from scipy.stats import levene
from app.analysis.data_preprocessor import Data_Preprocessor

class Anova_Analysis(object):

    def __init__(self, data_preprocessor):
        self.data = data_preprocessor

    def run_anova(self):
        formula_y = self.make_formula(self.data.y_columns)
        formula_x = self.make_formula(self.data.x_columns)
        formula_cov = self.make_formula(self.data.cov_columns)

        formula = 'subt1 ~ C(a01) + C(a08) + dept1'
        if formula_cov != '':
            formula = ("{y} ~ {x} + {cov}".format(y=formula_y, x=formula_x, cov=formula_cov))
        else:
            formula = ("{y} ~ {x}".format(y=formula_y, x=formula_x))

        print(formula)
        lm = ols(formula, self.data.feature_df).fit()
        print(lm.summary())

        return lm.summary().as_html()

    def run_manova(self):
        # https://stackoverflow.com/questions/51553355/how-to-get-p-value-from-statsmodels-manova
        formula_y = self.make_formula(self.data.y_columns)
        formula_x = self.make_formula(self.data.x_columns)
        formula_cov = self.make_formula(self.data.cov_columns)

        formula = 'cpt1 + dept1 + jelt1 ~ C(a01) + C(a08) + C(a01) * C(a08)'
        if formula_cov != '':
            formula = ("{y} ~ {x} + {cov}".format(y=formula_y, x=formula_x, cov=formula_cov))
        else:
            formula = ("{y} ~ {x}".format(y=formula_y, x=formula_x))

        print(formula)
        manova = MANOVA.from_formula(formula, self.data.feature_df)
        manova_model = manova.mv_test()
        print(manova_model.summary())

        return manova_model.summary().as_html()

    def calc_bartlett(self, column_names):
        bartlett_args = []
        for column_name in column_names:
            bartlett_args.append(self.data.feature_df[column_name].values)
        stat, pvalue = bartlett(*bartlett_args)
        print(stat, pvalue)

    def calc_levene(self, column_names):
        levene_args = []
        for column_name in column_names:
            levene_args.append(self.data.feature_df[column_name].values)
        stat, pvalue = levene(*levene_args)
        print(stat, pvalue)

    def make_formula(self, var_list, covariate_bool=False):
        formula = ''
        if covariate_bool:
            for index in range(len(var_list)):
                if index == len(var_list)-1:
                    formula = formula + "C{}".format(var_list[index])
                else:
                    formula = formula + "C{}".format(var_list[index]) + " + "
        else:

```

```

        for index in range(len(var_list)):
            if index == len(var_list)-1:
                formula = formula + var_list[index]
            else:
                formula = formula + var_list[index] + " + "

        return formula

if __name__ == "__main__":
    # homework 4
    data_preprocessor = Data_Preprocessor(None,
                                         missing_data='drop',
                                         filename="Dataset2.csv",
                                         y_columns=['subt1'],
                                         x_columns=['a01', 'a08', 'dept1'],
                                         covariate_columns=[],
                                         categorical_columns=['a01', 'a08'])

    anova_analysis_obj = Anova_Analysis(data_preprocessor)
    anova_analysis_obj.run_anova()
    anova_analysis_obj.calc_bartlett(['a01', 'a08'])
    anova_analysis_obj.calc_levene(['a01', 'a08'])

    # homework 5
    # data_preprocessor = Data_Preprocessor(None,
    #                                       missing_data='drop',
    #                                       filename="Dataset2.csv",
    #                                       y_columns=['a01', 'a08'],
    #                                       x_columns=['cpt1', 'dept1', 'jelt1'],
    #                                       categorical_columns=['a01', 'a08'])

    # anova_analysis_obj = Anova_Analysis(data_preprocessor)
    # anova_analysis_obj.run_manova()

```

```

# Author: Junbong Jang
# Date: 2/6/2018
# app/factor_analysis.py
from app.analysis.data_preprocessor import Data_Preprocessor
import statsmodels.multivariate.factor as sm
import numpy as np
import pandas as pd

class Factor_Analysis(object):

    def __init__(self, data_preprocessor_in):
        self.data = data_preprocessor_in

    def run_factor(self, rotate_option='varimax'):
        factor = sm.Factor(self.data.X, 35)
        factor_model = factor.fit()
        factor_model.rotate(rotate_option)
        print(factor_model.summary())
        factor_model.plot_scree()
        import matplotlib.pyplot as plt
        plt.show()

# https://factor-analyzer.readthedocs.io/en/latest/_modules/factor_analyzer/factor_analyzer.html
def covariance_to_correlation(self, m):
    """
    This is a port of the R `cov2cor` function.

    Parameters
    -----
    m : numpy array
        The covariance matrix.

    Returns
    -----
    retval : numpy array
        The cross-correlation matrix.

    Raises

```

```

ValueError
    If the input matrix is not square.
"""

# make sure the matrix is square
numrows, numcols = m.shape
if not numrows == numcols:
    raise ValueError('Input matrix must be square!')

ls = np.sqrt(1 / np.diag(m))
retval = ls * m * np.repeat(ls, numrows).reshape(numrows, numrows)
np.fill_diagonal(retval, 1.0)
return retval

def partial_correlations(self, data):
    """
    This is a python port of the `pcor` function implemented in
    the `ppcor` R package, which computes partial correlations
    of each pair of variables in the given data frame `data`,
    excluding all other variables.

    Parameters
    -----
    data : pd.DataFrame
        Data frame containing the feature values.

    Returns
    -----
    df_pcor : pd.DataFrame
        Data frame containing the partial correlations of each
        pair of variables in the given data frame `df`,
        excluding all other variables.
    """
    numrows, numcols = data.shape
    df_cov = data.cov()
    columns = df_cov.columns

    # return a matrix of nans if the number of columns is
    # greater than the number of rows. When the ncol == nrow
    # we get the degenerate matrix with 1 only. It is not meaningful
    # to compute partial correlations when ncol > nrow.

    # create empty array for when we cannot compute the
    # matrix inversion
    empty_array = np.empty((len(columns), len(columns)))
    empty_array[:] = np.nan
    if numcols > numrows:
        icvx = empty_array
    else:
        # we also return nans if there is singularity in the data
        # (e.g. all human scores are the same)
        try:
            icvx = np.linalg.inv(df_cov)
        except np.linalg.LinAlgError:
            icvx = empty_array
    pcor = -1 * self.covariance_to_correlation(icvx)
    np.fill_diagonal(pcor, 1.0)
    df_pcor = pd.DataFrame(pcor, columns=columns, index=columns)
    return df_pcor

def calculate_kmo(self, data):
    """
    Calculate the Kaiser-Meyer-Olkin criterion
    for items and overall. This statistic represents
    the degree to which each observed variable is
    predicted, without error, by the other variables
    in the dataset. In general, a KMO < 0.6 is considered
    inadequate.

    Parameters
    -----
    data : pd.DataFrame
        The data frame from which to calculate KMOs.
    """

```

```

Returns
-----
kmo_per_variable : pd.DataFrame
    The KMO score per item.
kmo_total : float
    The KMO score overall.
"""

# calculate the partial correlations
partial_corr = self.partial_correlations(data)
partial_corr = partial_corr.values

# calculate the pair-wise correlations
corr = data.corr()
corr = corr.values

# fill matrix diagonals with zeros
# and square all elements
np.fill_diagonal(corr, 0)
np.fill_diagonal(partial_corr, 0)

partial_corr = partial_corr ** 2
corr = corr ** 2

# calculate KMO per item
partial_corr_sum = partial_corr.sum(0)
corr_sum = corr.sum(0)
kmo_per_item = corr_sum / (corr_sum + partial_corr_sum)
kmo_per_item = pd.DataFrame(kmo_per_item,
                            index=data.columns,
                            columns=['KMO'])

# calculate KMO overall
corr_sum_total = corr.sum()
partial_corr_sum_total = partial_corr.sum()
kmo_total = corr_sum_total / (corr_sum_total + partial_corr_sum_total)
return kmo_per_item, kmo_total

if __name__ == "__main__":
    # homework 6
    data_preprocessor = Data_Preprocessor(missing_data='drop',
                                         filename="Dataset2.csv",
                                         predict_column=['subt1'],
                                         x_columns=['am01', 'am02', 'am03', 'am04', 'am05', 'am06', 'am07', 'am08', 'am09',
                                                    'con01', 'con03', 'con05', 'con09', 'con12', 'con13', 'con14', 'con15',
                                                    'con17',
                                                    'cp10', 'cp11', 'cp12', 'cp13', 'cp15', 'cp02', 'cp07', 'cp08',
                                                    'dom01', 'dom02', 'dom03', 'dom04', 'dom05', 'dom06', 'dom07', 'dom08',
                                                    'dom09'],
                                         categorical_columns=['a01', 'a08'])
    factor_analysis_obj = Factor_Analysis(data_preprocessor)
    print(factor_analysis_obj.calculate_kmo(factor_analysis_obj.data.X))
    factor_analysis_obj.run_factor('varimax')

```

```

# Author: Junbong Jang
# Date: 1/23/2019
# app/data_preprocessor.py

import pandas as pd

# note that when connected with ALL, columns of the data frame will be 'object'
class Data_Preprocessor(object):
    def __init__(self, input_df,
                 filename="Dataset1.csv",
                 covariate_columns=[],
                 x_columns=['rct1', 'rdt1', 'a11'],
                 y_columns=['a14'],
                 categorical_columns=[],
                 missing_data='drop'):

        self.cov_columns = covariate_columns
        self.x_columns = x_columns

```

```

self.y_columns = y_columns
self.categorical_columns = categorical_columns
self.all_columns = covariate_columns + x_columns + y_columns
# ----- Data Preprocessing -----
if input_df is None:
    dataset_path = '../resources/uploads/' + filename
    self.input_df = pd.read_csv(dataset_path)
else:
    self.input_df = input_df

self.handle_missing_data()
self.handle_categorical_columns(categorical_columns)
self.convert_to_correct_type()
self.drop_missing_data(missing_data)

# ----- Variables Defined -----
self.X = self.input_df[self.x_columns]
self.y = self.input_df[self.y_columns]
self.feature_df = self.input_df[self.x_columns + self.y_columns + self.cov_columns]

# print(self.input_df.shape)
# print(self.input_df[self.x_columns].shape)
# print(self.input_df[self.y_columns].shape)
# print(self.input_df[self.cov_columns].shape)
# for index, row in self.input_df.iterrows():
#     print(row[self.x_columns[0]], row[self.y_columns[0]])

# .loc for label indexing
# .iloc for integer indexing
# single bracket inside loc returns a value of the row for pandas series,
# double brackets return the pandas series or data frame

# print(feature_df.loc[:, 'a01'].to_string())
# Tutorial from https://towardsdatascience.com/simple-and-multiple-linear-regression-in-python-c928425168f9
# print(feature_df.describe().unstack().to_string())

def convert_to_correct_type(self):
    # all the items that are not categorical
    for column in [item for item in self.all_columns if item not in self.categorical_columns]:
        self.input_df[column] = pd.to_numeric(self.input_df[column])

def handle_categorical_columns(self, categorical_columns):
    for column_name in categorical_columns:
        self.input_df.loc[:, column_name] = self.input_df.loc[:, column_name].map({'3': 3, '2': 2, '1': 1, '0': 0})

def handle_missing_data(self):
    # retain only alphanumeric characters
    for a_column in self.all_columns:
        self.input_df[a_column].str.replace('WW', '')
        self.input_df = self.input_df[self.input_df[a_column] != '']

    # All doc report marks null value with -
    for a_column in self.all_columns:
        self.input_df = self.input_df[self.input_df[a_column] != '-']

def drop_missing_data(self, missing_data):
    # print(input_df.isnull().any()) # check for missing values
    if missing_data == 'drop':
        self.input_df.dropna(inplace=True)
    elif missing_data == 'fill':
        print('fill')

```

```

# Author: Unknown
# Modified by: Junbong Jang
# Date: 3/11/2019
# app/post_req.py

```

```

from flask import Flask, request, render_template
import json
from scipy.stats import chi2_contingency
import statsmodels.api as sm
from statsmodels.formula.api import ols
import math
import pandas as pd
import numpy as np

from app.analysis import apa_formatter
from app.analysis.anova_analysis import Anova_Analysis
from app.analysis.data_preprocessor import Data_Preprocessor
from app.analysis.multiple_regression import Multiple_Regression
from app.previous.affect_detectors import apply_affect_detectors

app = Flask(__name__)

@app.route('/', methods=['GET', 'POST'])
def default_output():
    return 'hello world'

@app.route('/post', methods=['POST'])
def chi():
    print("CHI SQUARED TEST")
    search = request.args.get("userName")
    print("SEARCH:", search)

    print(request.is_json)
    content = request.get_json()
    matrix = content['Matrix']

    a = [matrix[i:i + 2] for i in range(0, len(matrix), 2)]
    col0 = [a[i][0] for i in range(0, len(a))]
    col1 = [a[i][1] for i in range(0, len(a))]
    res = chi2_contingency(a)
    chi_step2(col0, col1)
    data = {}
    data['chi2'] = res[0]
    data['p'] = res[1]
    data['dof'] = res[2]

    js = json.dumps(data)
    return js

def chi_step2(col0, col1):
    new_combined_matrix = [[col0[i], col1[i]] for i in range(len(col0))]
    res = chi2_contingency(new_combined_matrix)
    return res

@app.route('/anova', methods=['POST'])
def anova_func():
    print("ANOVA TEST")

    print(request.is_json)
    content = request.get_json()
    print(content)
    col0 = content["Condition"]
    col1 = content["Posttest"]

    js = anova_func2(col0, col1)
    return js

def anova_func2(col0, col1):
    df = pd.DataFrame(columns=['Control', 'PostTest'])
    df['Control'] = col0
    df['PostTest'] = col1
    df['PostTest'] = df['PostTest'].astype("float")
    mod = ols('PostTest ~ Control', data=df).fit()
    aov_table = sm.stats.anova_lm(mod, typ=2)
    data = {}

```



```

data['columns'] = aov_table.axes[0].tolist()
data['sum_sq_c'] = format(aov_table['sum_sq']['Control'], )
data['sum_sq_r'] = format(aov_table['sum_sq']['Residual'], )
data['df_c'] = int(aov_table['df']['Control'])
data['df_r'] = int(aov_table['df']['Residual'])

data['mean_sq_c'] = format(aov_table['sum_sq']['Control'] / aov_table['df']['Control'], )
data['mean_sq_r'] = format(aov_table['sum_sq']['Residual'] / aov_table['df']['Residual'], )

data['F_c'] = format(aov_table['F']['Control'], )

if math.isnan(aov_table['F']['Control']):
    print("NaN found")
    data['F_c'] = "NaN"

data['F_r'] = aov_table['F']['Residual']

if math.isnan(aov_table['F']['Residual']):
    print("NaN found")
    data['F_r'] = "NaN"
data['PR_c'] = format(aov_table['PR(>F)']['Control'], )
if math.isnan(aov_table['PR(>F)']['Control']):
    print("NaN found")
    data['PR_c'] = "NaN"
data['PR_r'] = aov_table['PR(>F)']['Residual']
if math.isnan(aov_table['PR(>F)']['Residual']):
    print("NaN found")
    data['PR_r'] = "NaN"

js = json.dumps(data)
return js

@app.route('/affect', methods=['POST'])
def affect():
    print("AFFECT TEST")

    # confirm that request was sent and get json data
    print(request.is_json)

    # configure and format the json object into the format for the affect model arguments
    content = request.get_json()
    headers = content['header']
    np_headers = np.array(headers) # need a ndarray rather than list
    distilled_data = content['matrix']
    np_data = np.array(distilled_data) # need a ndarray rather than list

    # use the affect model
    data, headers = apply_affect_detectors(np_data, np_headers)

    # convert to list of list of floats (not strings)
    list_data = []
    for x in data:
        temp = []
        for y in x:
            temp.append(float(y))
        list_data.append(temp)

    # configure results of the affect model for the json to send back to the java
    out_data = {}
    out_data['header'] = headers
    out_data['matrix'] = list_data
    js = json.dumps(out_data)
    return js

@app.route('/sizetest', methods=['POST'])
def sizetest():
    # confirm that request was sent and get json data
    print()
    print("Is JSON? ", request.is_json)
    content = request.get_json()
    # modify the data
    data = {}
    for i in range(0, 100):

```

```

        listname = "list" + str(i)
        list = content[listname]
        list2 = []
        for r in list:
            list2.append(r.lower())
        data[listname] = list2
    # send data back to java
    js = json.dumps(data)
    return js

@app.route('/analysis/linear_regression', methods=['POST'])
def multiple_linear_regression():
    print('multiple linear regression')
    print(request.is_json)
    content = request.get_json()
    print(content)
    input_df = pd.DataFrame(columns=['independent1', 'dependent1'])
    input_df['independent1'] = content['independent1']
    input_df['dependent1'] = content['dependent1']
    # https://stackoverflow.com/questions/25065900/request-args-getkey-gives-null-flask
    data_preprocessor = Data_Preprocessor(input_df=input_df,
                                         missing_data='drop',
                                         x_columns=["independent1"],
                                         y_columns=["dependent1"])
    multiple_regression_obj = Multiple_Regression(data_preprocessor)
    model_stat_dict, coefficients_dict = multiple_regression_obj.calc_multiple_regression_stat()
    print(model_stat_dict)
    print(coefficients_dict)

    multiple_regression_obj.draw_correlation_matrix('multi_reg')
    with app.app_context():
        return render_template('result.html',
                               view_state='result',
                               filename='multi_reg',
                               descriptive_stat=multiple_regression_obj.calc_descriptive_dict(),
                               corr_2d=apa_formatter.corr_matrix_apa(multiple_regression_obj),
                               model_stat_dict=model_stat_dict,
                               coefficients_dict=coefficients_dict)

@app.route('/analysis/ancova', methods=['POST'])
def ancova_func():
    print('ancova func')
    content = request.get_json()
    print(content)
    input_df = pd.DataFrame.from_dict(content)
    data_preprocessor = Data_Preprocessor(input_df=input_df,
                                         missing_data='drop',
                                         covariate_columns=["covariate"],
                                         x_columns=["independent"],
                                         y_columns=["dependent"],
                                         categorical_columns=["independent", "dependent"])

    ancova_obj = Anova_Analysis(data_preprocessor)
    ancova_result = ancova_obj.run_anova()

    return ancova_result

@app.route('/analysis/manova', methods=['POST'])
def manova_func():
    print('manova func')
    content = request.get_json()
    print(content)
    input_df = pd.DataFrame.from_dict(content)

    data_preprocessor = Data_Preprocessor(input_df=input_df,
                                         missing_data='drop',
                                         x_columns=["independent1"],
                                         y_columns=["dependent1", "dependent2", "dependent3"],
                                         categorical_columns=["dependent2", "dependent3"])
    # print(data_preprocessor.input_df.to_string())

    manova_obj = Anova_Analysis(data_preprocessor)

```

```

manova_result = manova_obj.run_manova()

return manova_result

```

```

# Author: Junbong Jang
# Date: 12/4/2018
# app/google_spreadsheet.py

from __future__ import print_function
from googleapiclient.discovery import build
from httplib2 import Http
from oauth2client import file, client, tools
from pprint import pprint

# If modifying these scopes, delete the file token.json.
SCOPES = 'https://www.googleapis.com/auth/spreadsheets'

class Google_Spreadsheet(object):
    def __init__(self):
        self.spreadsheet_id = ''

    def user_authentication(self):
        """Shows basic usage of the Sheets API.
        Prints values from a sample spreadsheet.
        """
        # The file token.json stores the user's access and refresh tokens, and is
        # created automatically when the authorization flow completes for the first
        # time.
        store = file.Storage('token.json')
        creds = store.get()
        if not creds or creds.invalid:
            flow = client.flow_from_clientsecrets('credentials.json', SCOPES)
            creds = tools.run_flow(flow, store)
            service = build('sheets', 'v4', http=creds.authorize(Http()))

        return service

    def create_spreadsheet(self, service, title):
        spreadsheet_body = {
            'properties': {
                'title': title
            }
        }

        request = service.spreadsheets().create(body=spreadsheet_body)
        response = request.execute()

        pprint(response)
        self.spreadsheet_id = response['spreadsheetId']
        return response['spreadsheetUrl']

    def update_spreadsheet(self, service, list_of_values):
        body = {
            'values': list_of_values
        }
        result = service.spreadsheets().values().append(
            spreadsheetId=self.spreadsheet_id, range='Sheet1',
            valueInputOption='USER_ENTERED', body=body).execute()
        print('{0} cells appended.'.format(result
            .get('updates')
            .get('updatedCells')))

```

```

# Author: Junbong Jang
# Date: 11/30/2018
# app/views.py

from flask import render_template, send_from_directory, flash, request, redirect, url_for
from werkzeug.utils import secure_filename
import os

from app import app, multiple_regression, apa_formatter, google_spreadsheet, csv_parser

def allowed_file(filename):
    ALLOWED_EXTENSIONS = set(['txt', 'csv'])
    return '.' in filename and \
        filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

# http://flask.pocoo.org/docs/1.0/patterns/fileuploads/
@app.route('/', methods=['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        print('Post')
        # check if the post request has the file part
        if 'file' not in request.files:
            flash('No file part')
            return redirect(request.url)
        file = request.files['file']
        # if user does not select file, browser also
        # submit an empty part without filename
        if file.filename == '':
            flash('No selected file')
            return redirect(request.url)
        if file and allowed_file(file.filename):
            filename = secure_filename(file.filename)
            file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
            return redirect(url_for('display_upload',
                                    filename=filename)) # build a URL to a specific function
    elif request.method == 'GET':
        print('Get')
        return render_template("index.html", view_state='home')

@app.route('/display/<filename>', methods=['GET', 'POST'])
def display_upload(filename):
    spreadsheet_instance = google_spreadsheet.Google_Spreadsheet()
    service = spreadsheet_instance.user_authentication()
    spreadsheet_url = spreadsheet_instance.create_spreadsheet(service, filename)

    list_of_values = csv_parser.read_uploaded_csv(filename)
    spreadsheet_instance.update_spreadsheet(service, list_of_values)
    return render_template('upload_display.html',
                            view_state='data',
                            spreadsheet_url=spreadsheet_url,
                            filename=filename,
                            columns=list_of_values[0])

@app.route('/result/<filename>', methods=['GET', 'POST'])
def result(filename):
    print('result ~~~~~')
    # https://stackoverflow.com/questions/25065900/request-args-getkey-gives-null-flask
    parsed_independent_var = request.form.get('independent_var').split(", ")
    parsed_dependent_var = [request.form.get('dependent_var')]
    print(parsed_independent_var)
    print(parsed_dependent_var)
    print(filename)
    regression_model = multiple_regression.Multiple_Regression(filename,
                                                                parsed_independent_var,
                                                                parsed_dependent_var)

    model_stat_dict, coefficients_dict = regression_model.calc_multiple_regression_stat()
    print(model_stat_dict)
    print(coefficients_dict)
    regression_model.draw_correlation_matrix(filename)
    with app.app_context():
        return render_template('result.html',

```

```

view_state='result',
filename=filename,
descriptive_stat=regression_model.calc_descriptive_dict(),
corr_2d=apa_formatter.corr_matrix_apa(regression_model),
model_stat_dict=model_stat_dict,
coefficients_dict=coefficients_dict)

```

```

# Author: Junbong Jang
# Date: 11/29/2018
# app/apa_formatter.py

from app import multiple_regression

def corr_matrix_apa():
    multi_reg = multiple_regression.Multiple_Regression()
    corr_values_2d, corr_p_2d = multi_reg.calc_correlation_matrix()
    for row_index, row in enumerate(corr_values_2d):
        for col_index, elem in enumerate(row):
            if row_index == col_index and row_index != 0:
                corr_values_2d[row_index][col_index] = '---'
            elif col_index > row_index and row_index != 0:
                corr_values_2d[row_index][col_index] = ''
            else:
                if corr_p_2d[row_index][col_index] < 0.001:
                    corr_values_2d[row_index][col_index] = str(corr_values_2d[row_index][col_index]) + '***'
                elif corr_p_2d[row_index][col_index] < 0.05:
                    corr_values_2d[row_index][col_index] = str(corr_values_2d[row_index][col_index]) + '*'

    return corr_values_2d

corr_matrix_apa()

```

```

# Author: Junbong Jang
# Date: 12/3/2018
# app/csv_parser.py

import pandas as pd

def read_uploaded_csv(filename):
    uploaded_df = pd.read_csv("%s%s" % ('app/uploads/', filename))
    uploaded_df.dropna(inplace=True)
    return [uploaded_df.columns.tolist()] + uploaded_df.values.tolist()

```

```

# Author: Junbong Jang
# Date: 11/30/2018
# app/__init__.py

from flask import Flask

# Initialize the app
app = Flask(__name__,
            static_url_path='',
            static_folder='static',
            template_folder="templates",
            instance_relative_config=True)

# Load the views
from app import views

# Load the config file
app.config.from_object('config')
app.config['UPLOAD_FOLDER'] = 'app/uploads/'

```

```
app.config["MAX_CONTENT_LENGTH"] = 16 * 1024 * 1024 # 16MB
app.secret_key = 'super secret key'
```

```
# Author: Junbong Jang
# Date: 11/30/2018
# config.py

# Enable Flask's debugging features. Should be False in production
DEBUG = False
```

```
# Author: Junbong Jang
# Date: 11/30/2018
# run.py
from app import post_req

if __name__ == '__main__':
    post_req.app.run()
```

## HTML5 templates

```
<!-- base.html -->

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>{% block title %}{% endblock %}</title>
  <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.0.13/css/all.css" integrity="sha384-
DNOHZ68U8hZfIKX0rtjWvixusGo9WQnrNx2sqG0tfsghAvtVRW3tvkXWZh58N9jp"
  crossorigin="anonymous">
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css" integrity="sha384-
MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPM0" crossorigin="anonymous">
  <link rel="stylesheet" href="/css/base.css">
  <link rel="stylesheet" href="/css/index.css">
  <link rel="stylesheet" href="/css/result.css">

  <link rel="apple-touch-icon" sizes="57x57" href="/favicon/apple-icon-57x57.png">
  <link rel="apple-touch-icon" sizes="60x60" href="/favicon/apple-icon-60x60.png">
  <link rel="apple-touch-icon" sizes="72x72" href="/favicon/apple-icon-72x72.png">
  <link rel="apple-touch-icon" sizes="76x76" href="/favicon/apple-icon-76x76.png">
  <link rel="apple-touch-icon" sizes="114x114" href="/favicon/apple-icon-114x114.png">
  <link rel="apple-touch-icon" sizes="120x120" href="/favicon/apple-icon-120x120.png">
  <link rel="apple-touch-icon" sizes="144x144" href="/favicon/apple-icon-144x144.png">
  <link rel="apple-touch-icon" sizes="152x152" href="/favicon/apple-icon-152x152.png">
  <link rel="apple-touch-icon" sizes="180x180" href="/favicon/apple-icon-180x180.png">
  <link rel="icon" type="image/png" sizes="192x192" href="/favicon/android-icon-192x192.png">
  <link rel="icon" type="image/png" sizes="32x32" href="/favicon/favicon-32x32.png">
  <link rel="icon" type="image/png" sizes="96x96" href="/favicon/favicon-96x96.png">
  <link rel="icon" type="image/png" sizes="16x16" href="/favicon/favicon-16x16.png">
  <link rel="manifest" href="/favicon/manifest.json">
  <meta name="msapplication-TileColor" content="#ffffff">
  <meta name="msapplication-TileImage" content="/favicon/ms-icon-144x144.png">
  <meta name="theme-color" content="#ffffff">
</head>
<body>

{# <nav class="navbar navbar-expand-lg navbar-dark green_background">#}
{# <div class="container"><a class="navbar-brand" href="/" style="font-size:1.6rem;">ALI Boost</a>#}
{# <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent" aria-
controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">#}
{# <span class="navbar-toggler-icon"></span>#}
{# </button>#}
{##}
```

```

{#         <div class="collapse navbar-collapse" id="navbarSupportedContent">#}
{#         <ul class="navbar-nav mr-auto">#}
{##}
{#         </ul>#}
{#         <ul class="navbar-nav">#}
{#         <li class="nav-item {{'active' if view_state == 'home' }}">#}
{#         <a class="nav-link" href="/">Home <span class="sr-only">(current)</span></a>#}
{#         </li>#}
{#         <li class="nav-item">#}
{#         <a class="nav-link {{'active' if view_state == 'data' }}" href="#" onclick="alert('Please submit data
file at Home view.')">Data-Edit</a>#}
{#         </li>#}
{#         <li class="nav-item">#}
{#         <a class="nav-link {{'active' if view_state == 'result' }}" href="/result">Analysis-Report</a>#}
{#         </li>#}
{#         </ul>#}
{##}
{#         <form class="form-inline my-2 my-lg-0">#}
{#         <input class="form-control mr-sm-2" type="search" placeholder="Search" aria-label="Search">#}
{#         <button class="btn btn-success my-2 my-sm-0" type="submit">Search</button>#}
{#         </form>#}
{#     </div>#}
{##}
{# </div>#}
{# </nav>#}

{# <div class="mt-3 main-content scrollbar-dusty-grass">#}
{#     {% with messages = get_flashed_messages() %}#}
{#     {% if messages %}#}
{#         {% for message in messages %}#}
{#         <div class="alert alert-danger">#}
{#             {{ message }}#}
{#         </div>#}
{#     {% endfor %}#}
{#     {% endif %}#}
{#     {% endwith %}#}

{# block body %}
{# endblock %}

<!--[ footer ] -->
{#     <div id="footer" class="d-flex align-items-center mt-2">#}
{#         <div class="container text-center">#}
{#             <span class="align-self-center">ALI Boost created by Junbong Jang in 2018</span>#}
{#         </div>#}
{#     </div>#}

{# </div>#}

</body>
</html>

```

```

<!-- index.html -->

{% extends "base.html" %}
{% block title %}Welcome to ALI Data Analysis tool{% endblock %}
{% block body %}

<header id="home-section">
    <div class="dark-over lay">
        <div class="home-inner container">
            <div class="row">
                <div class="col-lg-8 d-none d-lg-block">
                    <h1 class="display-4" style="color:white;">
                        Online Data Analysis for everyone
                    </h1>
                <div class="d-flex">
                    <div class="p-4 align-self-start">
                        <i class="fas fa-check fa-2x"></i>

```

```

    </div>
    <div class="p-4 align-self-center white-text-style">
      Submit the CSV file
    </div>
  </div>

  <div class="d-flex">
    <div class="p-4 align-self-start">
      <i class="fas fa-check fa-2x"></i>
    </div>
    <div class="p-4 align-self-center white-text-style">
      Edit your data
    </div>
  </div>

  <div class="d-flex">
    <div class="p-4 align-self-start">
      <i class="fas fa-check fa-2x"></i>
    </div>
    <div class="p-4 align-self-center white-text-style">
      Get the analysis report
    </div>
  </div>
</div>

<div class="col-lg-4">
  <div class="card card-form green_background">
    <div class="card-body">
      <h3 class="gray-text-style">Submit Data Here</h3>
      <p class="gray-text-style">Please choose csv file only</p>

      <form id="csv_file_form" method="post" enctype="multipart/form-data">
        <div class="custom-file mb-3">
          <input id="csv_input_file" name="file"
            class="custom-file-input d-inline" type="file"
            onchange="handleFileInput(this.value)"
            accept=".csv, application/vnd.openxmlformats-officedocument.spreadsheetml.sheet,
application/vnd.ms-excel"
            required>
          <label class="custom-file-label"
            for="csv_input_file"
            id="csv_input_label">Choose file...</label>
          <div class="invalid-feedback">Example invalid custom file feedback</div>
        </div>

        <select class="custom-select form-group">
          <option selected>Type of Linear Model</option>
          <option value="1">Linear Regression</option>
          <option value="2">Logistic Regression</option>
          <option value="3">Mediation & Moderation</option>
          <option value="4">ANOVA</option>
          <option value="5">MANOVA</option>
          <option value="6">Factorial Analysis</option>
        </select>

        <div class="custom-control custom-checkbox form-group">
          <input type="checkbox" class="custom-control-input" id="customCheck1">
          <label class="custom-control-label" for="customCheck1">Option 1</label>
        </div>

        <div class="custom-control custom-checkbox form-group">
          <input type="checkbox" class="custom-control-input" id="customCheck2">
          <label class="custom-control-label" for="customCheck2">Option 2</label>
        </div>

        <div class="custom-control custom-checkbox form-group">
          <input type="checkbox" class="custom-control-input" id="customCheck3">
          <label class="custom-control-label" for="customCheck3">Option 3</label>
        </div>

        <button class="btn btn-light btn-block mt-2"
          style="color: black; font-weight: bold;"
          onclick="document.getElementById('csv_file_form').submit()">Submit</button>
      </form>

```





```

    </select>
  </div>

  <div class="input-group mb-3">
    <div class="input-group-prepend">
      <span class="input-group-text">Independent Variable</span>
    </div>
    <input type="text"
      name="independent_var"
      class="form-control"
      value="amt1, jelt1, subt1, cpt1, rdt1, a01">
  </div>

  <div class="text-center">
    <button type="submit" class="btn btn-outline-primary btn-lg mt-2 mb-4">
      Analyze
    </button>
  </div>
</form>
</div>
{% endblock %}

```

```

<!-- result.html -->

{% extends "base.html" %}
{% block title %}Analysis Result{% endblock %}
{% block body %}
<div class="container" style="border-left: 1px solid rgba(0,0,0,0.25); border-right: 1px solid rgba(0,0,0,0.25);">

  {# <h1 class="text-center">Analysis Report</h1>#}

  {# <div class="text-center">#}
  {# #}
  {# </div>#}

  <div class="text-center table-responsive scrollbar-lady-lips">
    <span class="font-weight-bold">ANOVA</span>
    <table class="regular_table">
      <thead>
        <tr>
          <th colspan="2">Model</th>
          <th>Sum of Squares</th>
          <th>df</th>
          <th>Mean Square</th>
          <th>F</th>
          <th>Sig.</th>
        </tr>
      </thead>
      <tbody>
        <tr>
          <td>1</td>
          <td style="text-align:left;" class="bold-border-right">Regression</td>
          <td>{{ model_stat_dict.model_ss }}</td>
          <td>{{ model_stat_dict.model_df }}</td>
          <td>{{ model_stat_dict.model_mse }}</td>
          <td>{{ model_stat_dict.fvalue }}</td>
          <td>{{ model_stat_dict.pvalue }}</td>
        </tr>
        <tr>
          <td></td>
          <td style="text-align:left;" class="bold-border-right">Residual</td>
          <td>{{ model_stat_dict.residual_ss }}</td>
          <td>{{ model_stat_dict.residual_df }}</td>
          <td>{{ model_stat_dict.residual_mse }}</td>
          <td></td>
          <td></td>
        </tr>
        <tr>
          <td></td>
          <td style="text-align:left;" class="bold-border-right">Total</td>
          <td>{{ model_stat_dict.total_ss }}</td>

```

```

        <td>{{ model_stat_dict.total_df }}</td>
        <td>{{ model_stat_dict.total_mse }}</td>
        <td></td>
        <td></td>
    </tr>
</tbody>
</table>
</div>

<div class="text-center table-responsive scrollbar-lady-lips mt-3">
    <span class="font-weight-bold">Model Summary</span>
    <table class="regular_table">
        <thead>
            <tr>
                <th rowspan="2">Model</th>
                <th rowspan="2">R</th>
                <th rowspan="2">R Square</th>
                <th rowspan="2">Adjusted R Square</th>
                <th rowspan="2">Std. Error of the Estimate</th>
                <th colspan="5">Change Statistics</th>
            </tr>
            <tr>
                <th class="border-right">R Square Change</th>
                <th>F Change</th>
                <th>df1</th>
                <th>df2</th>
                <th>Sig. F Change</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td class="bold-border-right">1</td>
                <td>{{ model_stat_dict.rvalue }}</td>
                <td>{{ model_stat_dict.rsquared }}</td>
                <td>{{ model_stat_dict.rsquared_adj }}</td>
                <td>{{ model_stat_dict.rvalue }}</td>
                <td>{{ model_stat_dict.rsquared }}</td>
                <td>{{ model_stat_dict.fvalue }}</td>
                <td>{{ model_stat_dict.model_df }}</td>
                <td>{{ model_stat_dict.residual_df }}</td>
                <td>{{ model_stat_dict.pvalue }}</td>
            </tr>
        </tbody>
    </table>
</div>

<div class="text-center table-responsive scrollbar-lady-lips mt-3">
    <span class="font-weight-bold">Coefficients</span>
    <table class="regular_table">
        <thead>
            <tr>
                <th rowspan="2">Model</th>
                <th colspan="2">Unstandardized Coefficients</th>
                <th colspan="2">Standardized Coefficients</th>
                <th rowspan="2">t</th>
                <th rowspan="2">Sig.</th>
                <th colspan="3">Correlations</th>
                <th colspan="2">Collinearity Statistics</th>
            </tr>
            <tr>
                <th style="border-right: 1px">B</th>
                <th>Std. Error</th>
                <th>Beta</th>
                <th>Zero-order</th>
                <th>Partial</th>
                <th>Part</th>
                <th>Tolerance</th>
                <th>VIF</th>
            </tr>
        </thead>
        <tbody>
            {% for variable in coefficients_dict.param_names %}
                <tr>

```

```

        <td>{% if loop.index == 1 %}
        1
        {% endif %}
        </td>
        <td style="text-align: left;" class="bold-border-right">{{ variable }}</td>
        <td>{{ coefficients_dict.unstandardized_beta[loop.index-1] }}</td>
        <td>{{ coefficients_dict.bse[loop.index-1] }}</td>
        <td>{{ coefficients_dict.standardized_beta[loop.index-1] }}</td>
        <td>{{ coefficients_dict.tvalues[loop.index-1] }}</td>
        <td>{{ coefficients_dict.pvalues[loop.index-1] }}</td>
        <td>{{ coefficients_dict.zero_order_corr[loop.index-1] }}</td>
        <td>{{ coefficients_dict.partial_corr[loop.index-1] }}</td>
        <td></td>
        <td>{{ coefficients_dict.multicol_list[loop.index-1][1] }}</td>
        <td>{{ coefficients_dict.multicol_list[loop.index-1][0] }}</td>
    </tr>
{% endfor %}
</tbody>

</table>
</div>

<h4 class="mt-3">APA formatted tables</h4>

<div class="table-responsive scrollbar-lady-lips">
    <span>Means, Standard Deviations, and Intercorrelations for {{ descriptive_stat[0].name }} and the Predictor
    Variables</span>
    <table class="apa_format_table">
        <thead>
            <tr>
                <th>Variable</th>
                <th style="text-align:center; font-style: italic;">M</th>
                <th style="text-align:center; font-style: italic;">SD</th>
                {% for variable_index in range(descriptive_stat|length - 1) %}
                <th style="text-align:center;">{{variable_index+1}}</th>
                {% endfor %}
            </tr>
        </thead>
        <tbody>

            {% for variable in descriptive_stat %}
            {% set outer_loop = loop %}
            <tr>
                <td style="{{ 'padding-left: 10px;' if outer_loop.index != 1 }}">
                    {% if loop.index != 1 %}
                    {{loop.index - 1}}.
                    {% endif %}
                    {{variable.name}}
                </td>
                <td style="text-align:center;">{{variable.mean}}</td>
                <td style="text-align:center;">{{variable.std}}</td>
                {% for elem in corr_2d[loop.index-1] %}
                {% if loop.index != 1 %}
                <td style="text-align:center;">{{elem}}</td>
                {% endif %}
                {% endfor %}
            </tr>
            {% if loop.index == 1 %}
            <tr>
                <td>
                    Predictor Variable
                </td>
            </tr>
            {% endif %}
            {% endfor %}
        </tbody>
    </table>
</div>

<div class="scrollbar-lady-lips mt-3">
    <span>Regression Analysis Summary for the Predictors of {{ descriptive_stat[0].name }} </span>
    <table class="apa_format_table">
        <thead>
            <tr>

```

```

        <th>Variable</th>
        <th style="font-style: italic;">B</th>
        <th style="font-style: italic;">SE B</th>
        <th> $\beta$ </th>
        <th style="font-style: italic;">t</th>
        <th style="font-style: italic;">p</th>
        <th>Squared Semi-partial<br>Correlation</th>
        <th>Structure<br>Coefficient</th>
    </tr>
</thead>
<tbody>
    <tr>
        {%- for variable in coefficients_dict.param_names %}
        {%- if loop.index != 1 %}
        <tr>
            <td style="text-align: left;">{{ variable }}</td>
            <td>{{ coefficients_dict.unstandardized_beta[loop.index-1] }}</td>
            <td>{{ coefficients_dict.bse[loop.index-1] }}</td>
            <td>{{ coefficients_dict.standardized_beta[loop.index-1] }}</td>
            <td>{{ coefficients_dict.tvalues[loop.index-1] }}</td>
            <td>{{ coefficients_dict.pvalues[loop.index-1] }}</td>
            {#
            <td>{{ coefficients_dict.zero_order_corr[loop.index-1] }}</td>#
            {#
            <td>{{ coefficients_dict.partial_corr[loop.index-1] }}</td>#
            <td></td>
            <td></td>
        </tr>
        {%- endif %}
        {%- endfor %}
    </tbody>
</table>
</div>
</div>
{% endblock %}

```

## Java Files

```

// Author: Junbong Jang
// Date: 3/9/2019
// persistence/TemplateColumnMapping.java
package org.assistments.service.datadumper.persistence;

import org.assistments.service.datadumper.dataset.DataTypes;

public class TemplateColumnMapping extends DatabaseColumnMapping
{
    private static TemplateColumnMapping singletonInstance;

    private static final DatabaseColumnData ID = new
DatabaseColumnData(ColumnNames.TEMPLATE_ID, "id", "id", DataTypes.INTEGER);
    private static final DatabaseColumnData NAME = new
DatabaseColumnData(ColumnNames.TEMPLATE_NAME, "name", "name", DataTypes.STRING);
    private static final DatabaseColumnData DESCRIPTION = new
DatabaseColumnData(ColumnNames.TEMPLATE_DESCRIPTION, "description", "description",
DataTypes.STRING);
    private static final DatabaseColumnData VERSION = new
DatabaseColumnData(ColumnNames.TEMPLATE_VERSION, "version", "version",
DataTypes.INTEGER);

```

```

        private static final DatabaseColumnData IS_ACTIVE = new
DatabaseColumnData(ColumnNames.TEMPLATE_IS_ACTIVE, "is_active", "is_active",
DataTypes.BOOLEAN);
        private static final DatabaseColumnData CREATED_AT = new
DatabaseColumnData(ColumnNames.TEMPLATE_CREATED_AT, "created_at", "created_at",
DataTypes.STRING);
        private static final DatabaseColumnData UPDATED_AT = new
DatabaseColumnData(ColumnNames.TEMPLATE_UPDATED_AT, "updated_at", "updated_at",
DataTypes.STRING);
        private static final DatabaseColumnData PARAMETER_SCHEMA = new
DatabaseColumnData(ColumnNames.TEMPLATE_PARAMETER_SCHEMA, "parameter_schema",
"parameter_schema", DataTypes.STRING);

```

```

private TemplateColumnMapping()
{
    super();
    columnDataMapping.put(ColumnNames.TEMPLATE_ID, ID);
    columnDataMapping.put(ColumnNames.TEMPLATE_NAME, NAME);
    columnDataMapping.put(ColumnNames.TEMPLATE_DESCRIPTION,
DESCRIPTION);
    columnDataMapping.put(ColumnNames.TEMPLATE_VERSION, VERSION);
    columnDataMapping.put(ColumnNames.TEMPLATE_IS_ACTIVE, IS_ACTIVE);
    columnDataMapping.put(ColumnNames.TEMPLATE_CREATED_AT, CREATED_AT);
    columnDataMapping.put(ColumnNames.TEMPLATE_UPDATED_AT, UPDATED_AT);
    columnDataMapping.put(ColumnNames.TEMPLATE_PARAMETER_SCHEMA,
PARAMETER_SCHEMA);
}

```

```

        order.add(ID);
        order.add(NAME);
        order.add(DESCRIPTION);
        order.add(VERSION);
        order.add(IS_ACTIVE);
        order.add(CREATED_AT);
        order.add(UPDATED_AT);
        order.add(PARAMETER_SCHEMA);
    }

    public static TemplateColumnMapping getInstance()
    {
        if (singletonInstance == null)
        {
            singletonInstance = new TemplateColumnMapping();
        }
        return singletonInstance;
    }
}

```

```

// Author: Junbong Jang
// Date: 3/9/2019
// persistence/ExperimentTemplateColumnMapping.java
package org.assistments.service.datadumper.persistence;

import org.assistments.service.datadumper.dataset.DataTypes;

public class ExperimentTemplateColumnMapping extends DatabaseColumnMapping
{
    private static ExperimentTemplateColumnMapping singletonInstance;

    private static final DatabaseColumnData ID = new
DatabaseColumnData(ColumnNames.E_TEMPLATE_ID, "id", "id", DataTypes.INTEGER);
    private static final DatabaseColumnData SEQUENCE_ID = new
DatabaseColumnData(ColumnNames.E_TEMPLATE_SEQUENCE_ID, "sequence_id",
"sequence_id", DataTypes.INTEGER);
    private static final DatabaseColumnData EXPERIMENT_TEMPLATE_ID = new
DatabaseColumnData(ColumnNames.E_TEMPLATE_EXPERIMENT_TEMPLATE_ID,
"experiment_template_id", "experiment_template_id", DataTypes.INTEGER);
    private static final DatabaseColumnData CREATOR_USER_ID = new
DatabaseColumnData(ColumnNames.E_TEMPLATE_CREATOR_USER_ID, "creator_user_id",
"creator_user_id", DataTypes.INTEGER);
    private static final DatabaseColumnData CREATED_AT = new
DatabaseColumnData(ColumnNames.E_TEMPLATE_CREATED_AT, "created_at", "created_at",
DataTypes.STRING);
    private static final DatabaseColumnData UPDATED_AT = new
DatabaseColumnData(ColumnNames.E_TEMPLATE_UPDATED_AT, "updated_at", "updated_at",
DataTypes.STRING);
    private static final DatabaseColumnData NOTE = new
DatabaseColumnData(ColumnNames.E_TEMPLATE_NOTE, "note", "note", DataTypes.STRING);
    private static final DatabaseColumnData PARAMETERS = new
DatabaseColumnData(ColumnNames.E_TEMPLATE_PARAMETERS, "parameters", "parameters",
DataTypes.STRING);

    private ExperimentTemplateColumnMapping()
    {
        super();
        columnDataMapping.put(ColumnNames.E_TEMPLATE_ID, ID);
        columnDataMapping.put(ColumnNames.E_TEMPLATE_SEQUENCE_ID,
SEQUENCE_ID);
        columnDataMapping.put(ColumnNames.E_TEMPLATE_EXPERIMENT_TEMPLATE_ID,
EXPERIMENT_TEMPLATE_ID);
        columnDataMapping.put(ColumnNames.E_TEMPLATE_CREATOR_USER_ID,
CREATOR_USER_ID);
        columnDataMapping.put(ColumnNames.E_TEMPLATE_CREATED_AT,
CREATED_AT);
        columnDataMapping.put(ColumnNames.E_TEMPLATE_UPDATED_AT,
UPDATED_AT);
        columnDataMapping.put(ColumnNames.E_TEMPLATE_NOTE, NOTE);
        columnDataMapping.put(ColumnNames.E_TEMPLATE_PARAMETERS,
PARAMETERS);

        order.add(ID);

```

```
        order.add(SEQUENCE_ID);
        order.add(EXPERIMENT_TEMPLATE_ID);
        order.add(CREATOR_USER_ID);
        order.add(CREATED_AT);
        order.add(UPDATED_AT);
        order.add(NOTE);
        order.add(PARAMETERS);
    }

    public static ExperimentTemplateColumnMapping getInstance()
    {
        if (singletonInstance == null)
        {
            singletonInstance = new ExperimentTemplateColumnMapping();
        }
        return singletonInstance;
    }
}
```