



MASTER OF DATA SCIENCE (SEMESTER 1 – 2022/2023)

FACULTY OF COMPUTER SCIENCE & INFORMATION TECHNOLOGY

WQD7008 PARALLEL AND DISTRIBUTED COMPUTING

GROUP ASSIGNMENT

**DEPLOYMENT OF API MANAGEMENT PLATFORM ON AWS FOR STEM BONUS
PREDICTION USING MACHINE LEARNING**

INSTRUCTOR: DR LIEW CHEE SUN

GROUP MEMBERS	
LEOW JUN SHOU	17123313
THOO POOI LUEN	17218141
JINGFA TANG	S2141959
SIM LIN ZHENG	S2102170
ROSE TIONG	S2123103

Table of Contents:

1.0 Introduction.....	3
1.1 Project Objective	4
1.2 Scenario	4
1.3 Previous Case Study	5
2.0 Data.....	6
2.1 Dataset	6
2.2 Data Preparation	7
2.3 Modelling	8
3.0 Project Architecture.....	12
3.1 Overall Architecture	12
3.2 DAGMan Process	15
3.3 Enabling Technology.....	15
3.4 User Interface (UI)	17
4.0 Discussion	23
5.0 Conclusion	25
6.0 Future outlook	26
7.0 References.....	27

1.0 Introduction

A distributed system is a collection of computer systems built on commodity hardware that are physically separated but network-ly connected through a centralized computer network equipped with distributed system software (Lamport, 2019). Parallel distributed system is an example of distributed systems using multiple processors or computing nodes to perform a task simultaneously (Rashid et al., 2019). They work in parallel to solve a problem, leading to stronger computing power, improved performance, and shorter computation time. In fact, the data and tasks are split among multiple processors or nodes in which they will work independently on their assigned task respectively and communicate with other processors or nodes when needed. The outputs from all the processors are then combined to produce the outcome.

There are two types of parallel distributed systems ranging from shared memory systems (OpenMP) and distributed memory systems (MPI); the shared-memory processors share a common memory in shared memory systems whereas in distributed memory systems each processor has its own memory, and they communicate with each other through message passing (Barbera et al., 2019). Additionally, parallel distributed systems offer scalable, flexible, and highly available computing resources. They can be used in various applications such as scientific computing, data analytics, machine learning, or cloud computing. Different technologies such as clusters, grids, and cloud computing uphold the implementation of parallel distributed system by using several frameworks ranging from OpenMP, MPI, and Pthreads (Kumar and Sadashiv, 2011; Jin et al., 2011). This report will focus on HTCondor, the open-source high-throughput computing software framework used in managing cluster workload and cycle scavenging.

HTCondor and Network File System

HTCondor is a specialised workload management system for computation-intensive jobs like those in engineering, business analytics, and scientific research (Livny et al., 2006). HTCondor can manage and schedule jobs on different platforms, including desktops, clusters, and grids in which it allows users to schedule, prioritise, and match jobs to desired machines while managing the resources (HTCondor, 2021). Briefly speaking, HTCondor uses a matchmaking algorithm in which it is capable of matching jobs with available resources in the most efficient way possible (Liu et al., 2016). Users are given chances to specify requirements and preferences for their jobs in terms of the amount of memory or disk space needed, and the type of operating system or architecture required. Besides, HTCondor offers some advanced features ranging from automatic checkpointing, remote system call, and file transfer (Livny et al., 2006). In the case of file transferring, HTCondor can utilise Network File System (NFS) distributed file system protocol to transmit files to and from the processors (Callaghan, 2004). NFS uses a client-server architecture granting users the privileges to access the same input and output files regardless of which node the job is running on (Eisler and Labiaga, 2010). This is because NFS allows multiple users across the network to access the same files simultaneously through a remote server as if the files were stored locally (Tipton, 2015). Since NFS is supported on most operating systems, HTCondor can use it to share the same home directories over the processors such that the users can work in a consistent environment across all nodes (Eisler and Labiaga, 2010; Livny et al., 2006).

Even though HTCondor has proven effective in assisting researchers across different disciplines, some limitations happen to constrain its capacities on supporting applications, notably (Smith, 2018):

- Applications require Windows to operate.
- User applications can merely use third-party software that is either preinstalled on PCs (i.e., MATLAB) or that can be quickly and readily installed straight away (i.e., R).
- Resources are limited up to 4 GB of memory per job and 2 GB of disk storage.
- Jobs can usually operate for a maximum of 8-10 hours, and a hard restriction of 24 hours as PCs are restarted daily.

Therefore, Amazon Web Services (AWS) cloud instances were developed to overcome these limitations.

Amazon Web Services (AWS)

AWS is a collection of remote computing services that make up a cloud computing platform, offered by Amazon.com (AWS, 2023). These services operate through the respective data centers in more than 20 countries across the world (Zhang, 2022). AWS offers a wide range of services ranging from computing power, storage, and databases that can be used to build and run applications and services. Some of the popular services that AWS offers are:



Figure 1.1: AWS services (Source: <https://www.optisolbusiness.com/insight/top-10-aws-services-for-digital-transformation>)

Talking about the pros of using AWS, AWS is scalable as it allows users to scale their resources depending on their needs (AWS, 2023). In addition, AWS offers a cost-effective pricing model – the pay-as-you-go payment method that means users only pay for the resources they use (Intellipaat, 2022). It also offers a wide range of services that are designed to be highly available, fault-tolerant, and run different types of applications from multiple regions around the world. Nonetheless, AWS offers various services possessing complexity, making it difficult for users to navigate and understand options available (Mueller, 2016). Besides, there is high dependence on internet connectivity as AWS services should be accessed over the Internet (AWS, 2021).

1.1 Project Objective

This project aimed at setting up a distributed data processing platform on Amazon Web Services (AWS) that can effectively manage, schedule and process a large dataset for machine learning workflow (STEM bonus prediction). The platform would include the following systems:

- (a) Resource management task scheduler – HTCondor, this system would be used to manage and schedule tasks on different platforms and match jobs with available resources;
- (b) Distributed data management – NFS, this system would be used to share the same home directories over the processors such that the users can work in a consistent environment across all nodes;
- (c) Workflow management engine – HTCondor DAGMan, this system would be used to manage and schedule the execution of multiple jobs in a workflow;
- (d) User interface – a user-friendly interface was designed for data inputting and result viewing.

1.2 Scenario

As the world is stepping into the fourth paradigm, Science, Technology, Engineering, and Mathematics (STEM) occupations are rising in demand such as materials science engineer, software developer, data engineer, aerospace engineer et cetera. People are keen to know who is earning a great fortune before making a transition or submitting a university application if they wish to.

In addition, it has been a taboo to share about how much one is earning which often leaves people in the dark when figuring out how much is considered an acceptable salary or bonus range that is commensurate with their skills and experiences. The lack of wage transparency often leads to employers low-balling the salaries of the employees and paying them below salary or bonus ranges of what the employees are worth.

Fortunately, the younger working generation has been vocal about pay discrimination and they are striving for wage transparency and the elimination of the pay gap. For instance, there is recently a popular page on Instagram called “Malaysian Pay Gap” that provides the public a platform to share their job and pay detail anonymously so that every job seeker knows how much they are worth for during their job search and salary negotiations, regardless of their age, gender, and ethnicity.

The Wisteria group was moved by the noble cause that these young working adults were pursuing and realised the need to contribute to this greater cause in eradicating unjust pay gaps. Thus, the modus operandi was to build a machine learning workflow to predict STEM bonus based on several features through setting up a distributed data processing platform by using HTCondor, NFS, and AWS.

1.3 Previous Case Study

There are plenty of real-life applications of using parallel distributed systems in machine learning. For instance:

- (a) Google's TensorFlow: TensorFlow is an open-source machine learning framework developed by Google. It uses data parallelism and model parallelism to distribute the training process across multiple machines. TensorFlow has been used in a variety of real-world applications, including image recognition, natural language processing, and speech recognition. (Abadi et al., 2016);
- (b) Baidu's PaddlePaddle: PaddlePaddle is an open-source machine learning framework developed by Baidu. It uses data parallelism and model parallelism to distribute the training process across multiple machines. PaddlePaddle has been used in a variety of real-world applications, including image recognition, natural language processing, and speech recognition. (PaddlePaddle, 2020);
- (c) Hadoop's Mahout: Mahout is a machine learning library for Hadoop that is designed to scale machine learning algorithms to large data sets (Owen et al., 2010). Mahout uses a combination of data parallelism and model parallelism to train models in parallel, and it has been used to train models for applications such as collaborative filtering, clustering, and classification. However, Mahout is now an inactive project.
- (d) Facebook's Big Sur: Big Sur is a large-scale machine learning cluster developed by Facebook. It is used to train deep learning models on large amounts of data. Big Sur uses a combination of data parallelism and model parallelism to train models in parallel, and it has been used to train models for applications such as image recognition, natural language processing, and speech recognition. (Miller, 2016).

2.0 Data

2.1 Dataset

This dataset was obtained from Kaggle :

<https://www.kaggle.com/datasets/jackogozaly/data-science-and-stem-salaries>

which was originally scraped from levels.fyi with some additional cleaning. This dataset contained 29 attributes with over 60,000 records, focusing on the salaries of employees in a variety of technological companies. Below were the descriptions of the attributes contained within the dataset.

No.	Attributes	Description
1	timestamp	Time and date of the record
2	company	Company name
3	level	Level of the position
4	title	Job title
5	totalyearlycompensation	Annual remuneration
6	location	Location of the job
7	yearsofexperience	Years of experience of the employee
8	yearsatcompany	Years of the employee staying at the company
9	tag	Specialisation of the job
10	basesalary	Base salary of the job
11	stockgrantvalue	Stock grant value of the job
12	bonus	Bonus amount of the job
13	gender	Gender
14	otherdetails	Other details such as education level and ethnicity
15	cityid	Unique ID of the city in which the job is located
16	dmaid	Designated market area ID
17	rowNumber	Number ID of the records
18	Masters_Degree	Holder of a master's degree (in binary format)
19	Bachelors_Degree	Holder of a bachelor's degree (in binary format)
20	Doctorate_Degree	Holder of a doctorate degree (in binary format)
21	Highschool	Holder of a high school's certificate (in binary format)
22	Some_College	Holder of a college degree (in binary format)
23	Race_Asian	Asian ethnicity (in binary format)
24	Race_White	White ethnicity (in binary format)
25	Race_Two_Or_More	Mixed ethnicity (in binary format)
26	Race_Black	Black ethnicity (in binary format)
27	Race_Hispanic	Hispanic ethnicity (in binary format)
28	Race	Ethnicity of the employee
29	Education	Education level of the employee

Table 2.1: Description of the attributes in the dataset.

The dataset contained a lot of noisy data and further pre-processing was needed. The following Python modules were used for data cleaning:

- Pandas
- NumPy
- Scikit-Learn

2.2 Data Preparation

The steps involved in cleansing the dataset were as follows:

- a) **Remove Null Values.** Firstly, completely removed the records with null values in the dataset's important attributes. This reduced the number of records from 60,000 to around 21,000.
- b) **Remove Unnecessary Attributes.** Removed columns that would not help in the bonus prediction. The following were the columns removed based on initial understanding of the dataset with rationale behind the removals.

No.	Removed Attributes	Rationale
1	level	Level of the position
2	tag	Specilisation of the job
3	basesalary	'Bonus' would be used as the target attribute as it is generally regarded as the focus of the employees, and these attributes are a subset of the target attribute.
4	stockgrantvalue	
5	totalyearlycompensation	
6	otherdetails	Duplicated column.
7	cityid	They did not mean anything as they were unique ID.
8	dmaid	
9	rowNumber	
10	Masters_Degree	These binary attributes had already been concatenated into other columns which were 'Race' and 'Education', making these attributes redundant.
11	Bachelors_Degree	
12	Doctorate_Degree	
13	Highschool	
14	Some_College	
15	Race_Asian	
16	Race_White	
17	Race_Two_Or_More	
18	Race_Black	
19	Race_Hispanic	

Table 2.2: Summary of removed attributes.

- c) **Standardise Company Names.** There were substantial inconsistencies in the names of the companies. For instance, Google can have a lot of variations such as 'google', 'Google Inc', 'GOOGLE', etc. Standardised their names to avoid duplicated categories within the attribute.
- d) **Pre-process 'location' attribute.** The values within the attribute contained city, state and country but they were separated by commas which made them a string datatype. Separated them out and created additional attributes for the dataset.
- e) **Label-Encode Categorical Attributes for Modelling.** Converted these categorical attributes that would be used for the modelling into numeric attributes in order to fit them into machine learning models. Specifically for the categorical attribute 'company', labelled them based on the top 89 companies by market capitalisation, those companies that were not within these 89 companies were labelled as '89', similar to 'others' category as the dataset contained hundreds of companies.

2.3 Modelling

In this phase, Python 3 was used to train three different machine learning algorithms for the dataset. Since this project aimed at predicting the bonus (continuous data), only regression algorithms were chosen ranging from Multi Linear Regression (MLR), Random Forest, and K-Nearest Neighbour (KNN).

Upon training, different commands were run to obtain the summary statistics of the dataset.

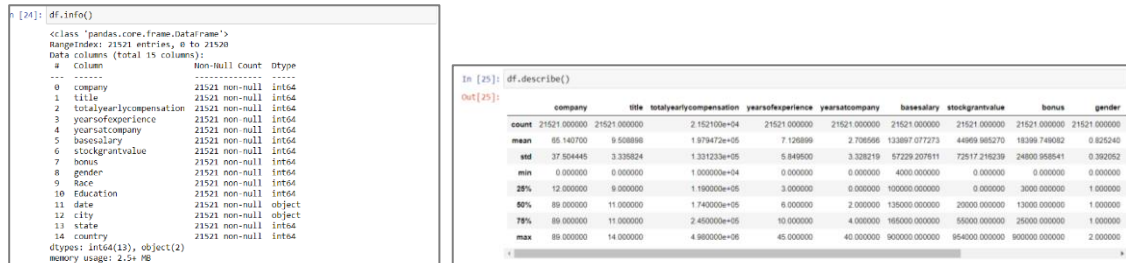


Figure 2.1: Summary statistics of the dataset.

Two variables, 'date' and 'city' were not selected for the modelling as they did not comprehend the machine learning objective.

Train-test Split

The dataset which contained 21,521 rows and 11 columns were split into training and test set in the ratio of 80:20. The variable 'bonus' was made target variable.

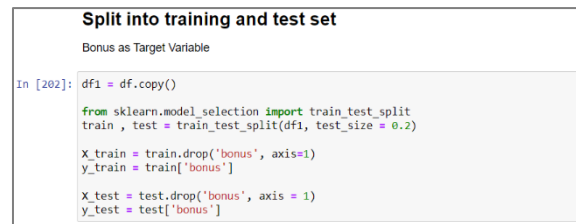


Figure 2.: Train-test split.

Multi Linear Regression (MLR)

Multiple linear regression is a statistical technique used to model the relationship between a dependent variable (also known as the response or outcome variable) and one or more independent variables (also known as predictor or explanatory variables) (Moore et al., 2006). The goal of multiple linear regression is to fit a linear model that can predict the value of the dependent variable based on the values of the independent variables.

The basic equation for a multiple linear regression model is:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_n X_n + \epsilon$$

where Y is the dependent variable, X_1, X_2, \dots, X_n are the independent variables, $\beta_0, \beta_1, \beta_2, \dots, \beta_n$ are the regression coefficients (also called parameters), and ϵ is the error term (also called residual) (Mapelu et al., 2013). The coefficients represent the change in the dependent variable for a one-unit change in the independent variable, while holding all other independent variables constant.


```
In [203]: # Import module:
from sklearn.linear_model import LinearRegression
LR = LinearRegression()

# Fitting the training data
LR.fit(X_train,y_train)

# Predicting the result
y_prediction = LR.predict(X_test)
```

Figure 2.3: Fit the Linear Regression on the training and test set.

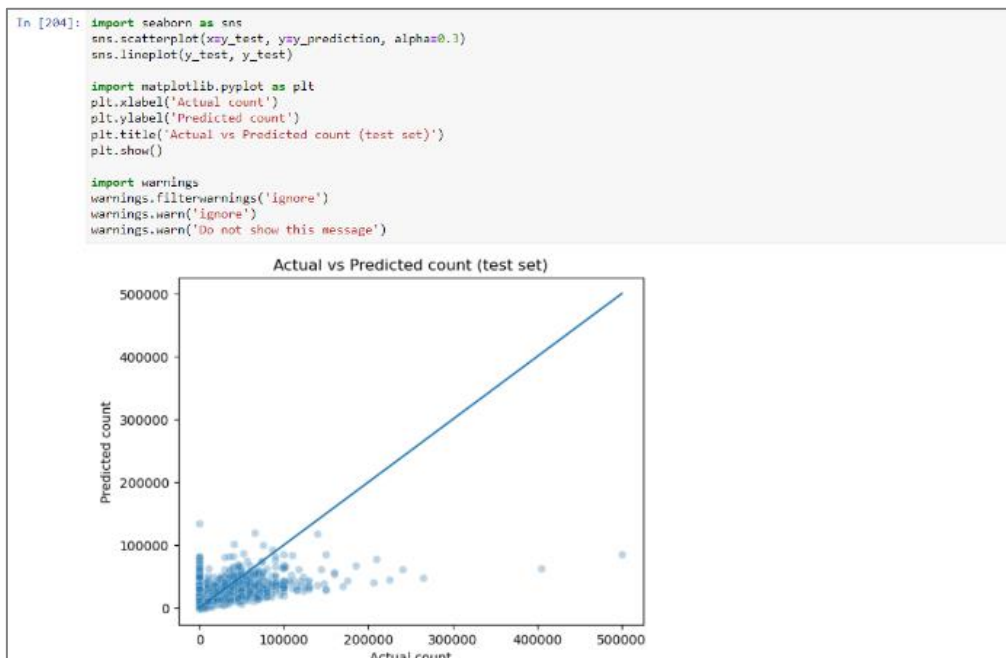


Figure 2.4: Plot the graph of actual vs predicted count of test set.

```
In [205]: from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
print("Mean absolute error (linear model): {:.2f}".format(mean_absolute_error(y_
print("Mean squared error (linear model): {:.2f}".format(mean_squared_error(y_te
print("r2_score (linear model): {:.2f}".format(r2_score(y_test, y_prediction)))

Mean absolute error (linear model): 10736.35
Mean squared error (linear model): 387797250.34
r2_score (linear model): 0.30
```

Figure 2.5: Evaluation metrics of MLP performance.

The advantage of MLP is it can be used to model the relationship between multiple independent variables and a single dependent variable, which can provide more detailed insights into the underlying data (Mata, 2011). However, it assumes that the relationship between the independent variables and the dependent variable is linear (Kayri et al., 2017), which may not always be the case. Therefore, it can be concluded that the MLP did not perform well on the dataset as the variables were not linear.

Random Forest

Random Forest is a type of ensemble machine learning algorithm that is used for both classification and regression tasks (Shaik and Srinivasan, 2018). It is an extension of decision trees, which are a type of algorithm that can be used to predict the value of a target variable based on the values of other variables.

The basic idea behind random forest is to build multiple decision trees and then combine their predictions to make a final prediction. Each decision tree is built using a random subset of the training data and a random subset of the features. By building multiple decision trees and combining their

predictions, random forest can reduce the variance and increase the accuracy of the predictions (Wang et al., 2018).

1 st Parameter	2 nd Parameter	3 rd Parameter
bootstrap = False, max_depth = 100, max_features = 'sqrt', min_samples_leaf = 1, min_samples_split = 2, n_estimators = 88	max_depth = 2, max_features = 'log2', min_samples_leaf = 1, min_samples_split = 2, n_estimators = 88	bootstrap = True, max_depth = None, max_features = 'auto', min_samples_leaf = 1, min_samples_split = 2, n_estimators = 100, verbose = 0
Evaluation Metrics		
Mean Absolute Error: 8453.67 Root Mean Squared Error: 17682.04 Mean squared error (linear model): 312654576.94 r2_score (linear model): 0.57	Mean Absolute Error: 11787.74 Root Mean Squared Error: 20499.51 Mean squared error (linear model): 420229981.35 r2_score (linear model): 0.23	Mean Absolute Error: 8475.14 Root Mean Squared Error: 17955.66 Mean squared error (linear model): 322405592.87 r2_score (linear model): 0.56

Table 2.3: Three sets of parameters for Random Forest.

1st Parameter (Best Model)
<pre>In [42]: from sklearn.ensemble import RandomForestRegressor regressor = RandomForestRegressor(bootstrap = False, max_depth = 100, max_features = 'sqrt', min_samples_leaf = 1, min_samples_split = 2, n_estimators = 88).fit(X1_train, y1_train) y1_pred = regressor.predict(X1_test)</pre>
<pre>In [43]: from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error print('Mean Absolute Error:', mean_absolute_error(y1_test, y1_pred).round(2)) print('Root Mean Squared Error:', np.sqrt(mean_squared_error(y1_test, y1_pred)).round(2)) print("Mean squared error (linear model): {:.2f}".format(mean_squared_error(y1_test, y1_pred))) print("r2_score (linear model): {:.2f}".format(r2_score(y1_test, y1_pred)))</pre> <p>Mean Absolute Error: 8453.67 Root Mean Squared Error: 17682.04 Mean squared error (linear model): 312654576.94 r2_score (linear model): 0.57</p>

Figure 2.6: Evaluation metrics of Random Forest performance.

Random forest can handle both categorical and continuous variables and it does not require a lot of data pre-processing; it is robust to outliers and can handle missing values (Jordanov et al., 2016; Maniruzzaman et al., 2018). Since Random Forest may not be suitable for very small datasets or datasets with only a few observations (Ho, 1998), the chosen dataset was big hence Random Forest can handle large number of features and high dimensionality data and performed better than the MLP.

K-Nearest Neighbours (KNN)

K-nearest neighbours (KNN) is a type of supervised learning algorithm that is used for both classification and regression tasks. It is a non-parametric algorithm, which means that it does not make any assumptions about the underlying data distribution (Ordiano et al., 2017).

The basic idea behind KNN is to classify or predict the value of a target variable based on the values of its k nearest neighbours. For a given test point, the algorithm finds the k training points that are closest to it in terms of a chosen distance metric (such as Euclidean distance) (Bundak et al., 2021). The algorithm then uses the majority class or average value of the k nearest neighbours to predict the class or value of the test point.

```

In [210]: from sklearn.preprocessing import MinMaxScaler
          scaler = MinMaxScaler(feature_range=(0, 1))

          x_train_scaled = scaler.fit_transform(X2_train)
          X2_train = pd.DataFrame(x_train_scaled)

          x_test_scaled = scaler.fit_transform(X2_test)
          X2_test = pd.DataFrame(x_test_scaled)

In [211]: #import required packages
          from sklearn import neighbors
          from sklearn.metrics import mean_squared_error
          from math import sqrt
          import matplotlib.pyplot as plt
          %matplotlib inline

```

Figure 2.7: Data normalisation using MinMaxScaler.

```

In [22]: rmse_val = [] #to store rmse values for different k
          for K in range(20):
              K = K+1
              model = neighbors.KNeighborsRegressor(n_neighbors = K)

              model.fit(X2_train, y_train) #fit the model
              pred=model.predict(X2_test) #make prediction on test set
              error = sqrt(mean_squared_error(y2_test,pred)) #calculate rmse
              rmse_val.append(error) #store rmse values
              print('RMSE value for k= ', K , 'is:', error)

RMSE value for k= 1 is: 23697.865941350417
RMSE value for k= 2 is: 22512.28597331873
RMSE value for k= 3 is: 21659.29950094408
RMSE value for k= 4 is: 21350.1993071016
RMSE value for k= 5 is: 21157.261502512898
RMSE value for k= 6 is: 21101.000747503305
RMSE value for k= 7 is: 20974.73504716789
RMSE value for k= 8 is: 20951.9207060302
RMSE value for k= 9 is: 20927.309170316013
RMSE value for k= 10 is: 20922.686174293383
RMSE value for k= 11 is: 20890.899386967947
RMSE value for k= 12 is: 20878.590061848186
RMSE value for k= 13 is: 20904.284422170276
RMSE value for k= 14 is: 20886.13737182229
RMSE value for k= 15 is: 20899.020878666237
RMSE value for k= 16 is: 20898.369769678146
RMSE value for k= 17 is: 20901.087103085167
RMSE value for k= 18 is: 20947.735703225397
RMSE value for k= 19 is: 20944.425018543225
RMSE value for k= 20 is: 20931.317027849087

```

Figure 2.8: Performance metrics (RMSE).

```

In [26]: from sklearn.metrics import mean_squared_error, r2_score, accuracy_score, mean_ab
          print("Mean ab error (linear model): {:.2f}".format(mean_absolute_error(y2_test,
          print("Mean squared error (linear model): {:.2f}".format(mean_squared_error(y2_t
          print("r2_score (linear model): {:.2f}".format(r2_score(y2_test, y2_pred)))
          #print("r2_score (linear model): {:.2f}".format(accuracy_score(y_test, y_pred)))

          Mean ab error (linear model): 15176.53
          Mean squared error (linear model): 530778175.26
          r2_score (linear model): 0.03

```

Figure 2.: Evaluation metrics of KNN performance.

KNN is simple to understand and implement since it does not require a lot of data processing (Hachclam, 2022). Nevertheless, it can be affected by the dimensionality, which means that as the number of features increases, the distance between points becomes less informative (Grant, 2019). Hence, KNN might not suit the chosen dataset.

In short, Random Forest was the best performing algorithm hence was chosen for the workflow.

3.0 Project Architecture

3.1 Overall Architecture

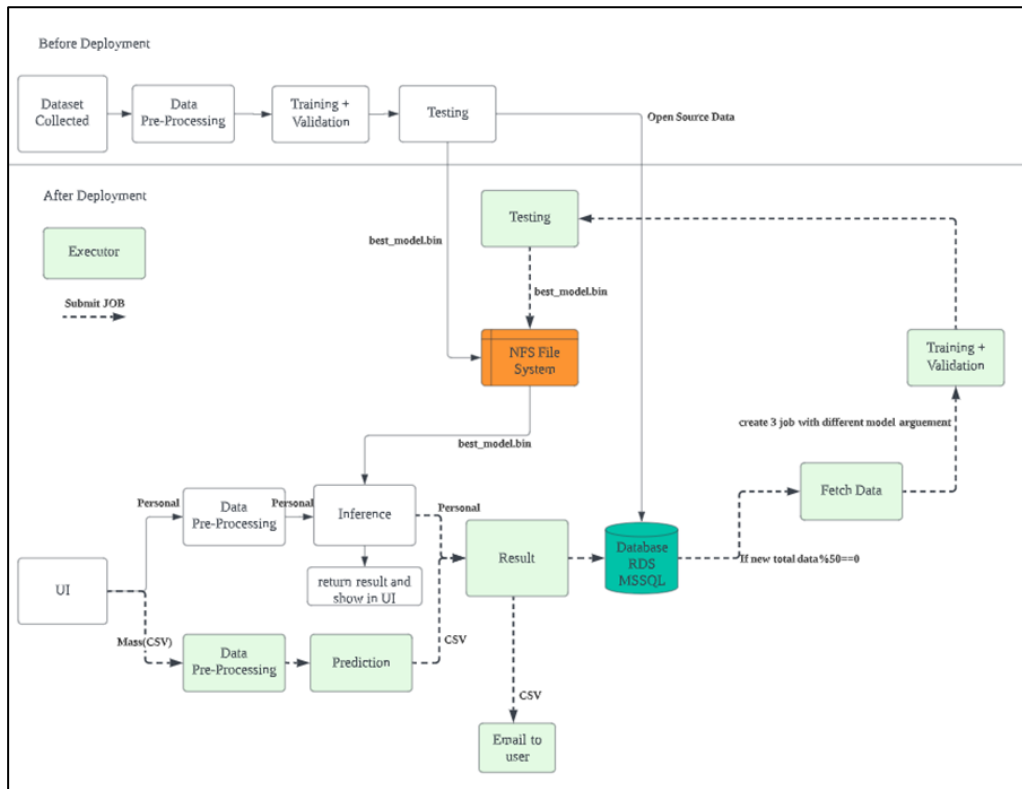


Figure 3.1: Architecture design of system set-up.

From the above figure, the process was separated into two parts, which were “before deployment” and “after deployment”. Deployment referred to deploying the code base into the EC2 server. The green colored boxes were distributed steps which would be executed by the executor.

Before deployment, the dataset was firstly pre-processed and fit into three different machine learning models which were Multi Linear Regression (MLR), Random Forest, and K-Nearest Neighbours (KNN). The best machine learning algorithm would be selected and used in the EC2 later. The Random Forest algorithm, which yielded the best result, was stored in the NFS file system. Also, the python script written in this stage was also used after deployment in the EC2 server.

Next, during the “after deployment” phase, in the EC2 submit host server, a flask API was hosted which accepted two POST method requests, which are used for single prediction and bulk CSV prediction.

For single prediction path, the received JSON string went through the data-preprocessing and the model prediction to generate the predicted bonus. The predicted bonus was returned to the user instantly. The data and predicted result were then stored in a relational database MSSQL. This database was hosted using AWS RDS.

For bulk CSV prediction, when a user submitted a CSV file, the user would receive an API message “wait for email” instantly. If the CSV file has more than 50 rows, then the server would submit two DAGMan workflows to the server, else it would only be bulk prediction workflow.

There were two DAGMan workflows:

1. Bulk prediction workflow

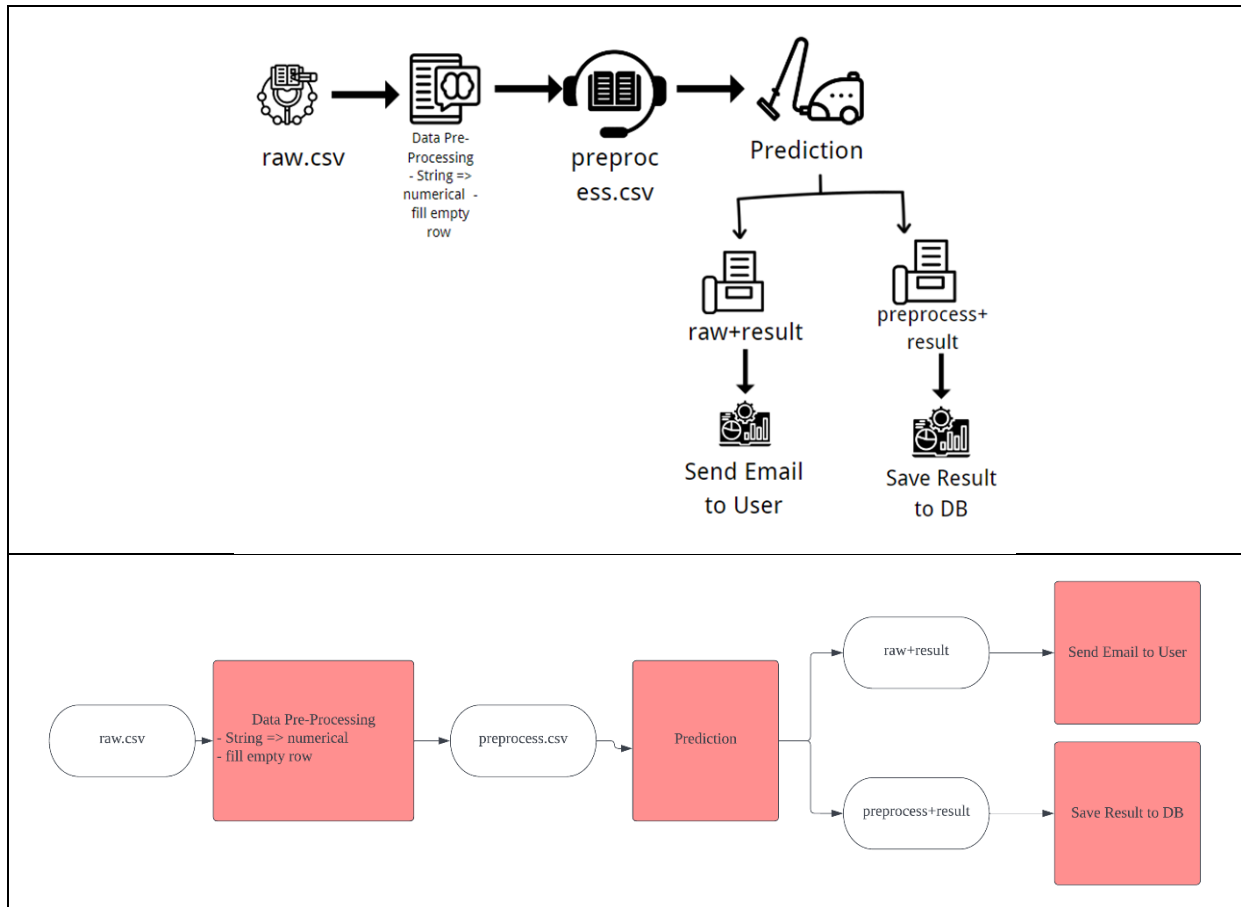


Figure 3.2: Bulk prediction workflow.

In this workflow, first, the submitting host received the raw CSV via the API. The commit host then submitted a DAGMan workflow to initiate the preprocessing of the raw CSV file into a preprocess.csv file. The preprocess.csv file was then used to make predictions using the Random Forest algorithm, once the prediction was complete, the workflow came to two sub-processes, the first process involved emailing the prediction results and the raw data file in CSV format to the user, the second the two processes involved preprocessing the CSV results and saving the prediction results to the database, which were crucial for modeling the training data.

2. Auto Retrain Workflow

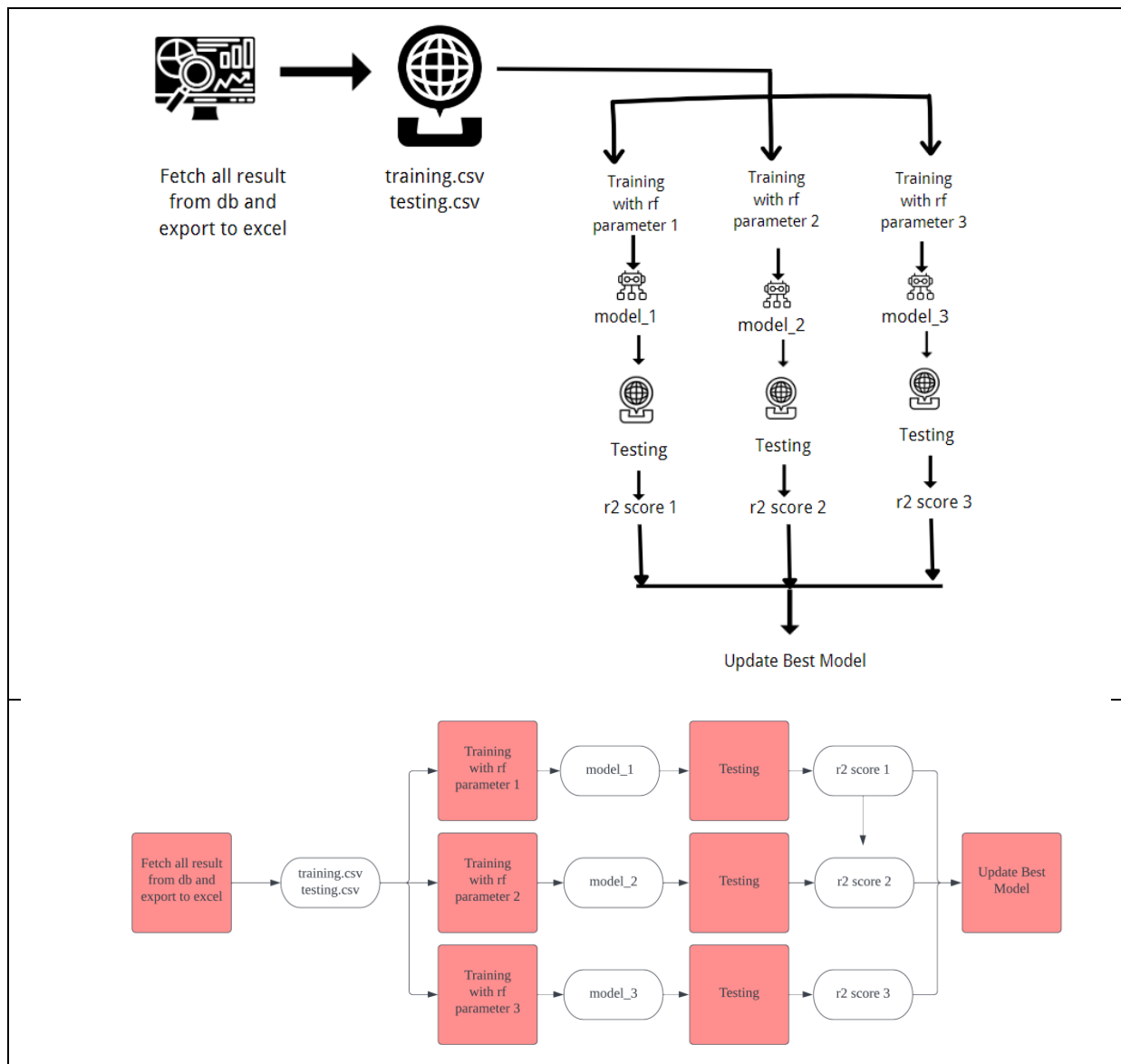


Figure 3.3: Auto Retrain Workflow.

The automatic workflow triggered a branching workflow when the original CSV file exceeded 50 rows. In this workflow, the submitting master submitted a DAGMan workflow to the central host at the beginning. The executor extracted all the data stored in the database and split it into training and testing CSV files.

Start by training the model: Trained the Random Forest algorithm using the training.csv file and trained three different Random Forest models (model_1, model_2, model_3) with three different parameters (Refer Table 2.3 at page 10).

Then, model testing and R2 score calculation: Tested each trained model using the testing.csv file and calculated the R2 score for each model (R2 score 1, R2 score 2, R2 score 3). At the same time, compared the R2 score with the best value stored in the database; if the R2 score was higher than the value in the database, updated the model by moving the weight file to the NFS file system.

Finally, best algorithm selection: Compared the R2 scores and chose the algorithm with the highest score as the best algorithm. The best algorithm was then updated in the database.

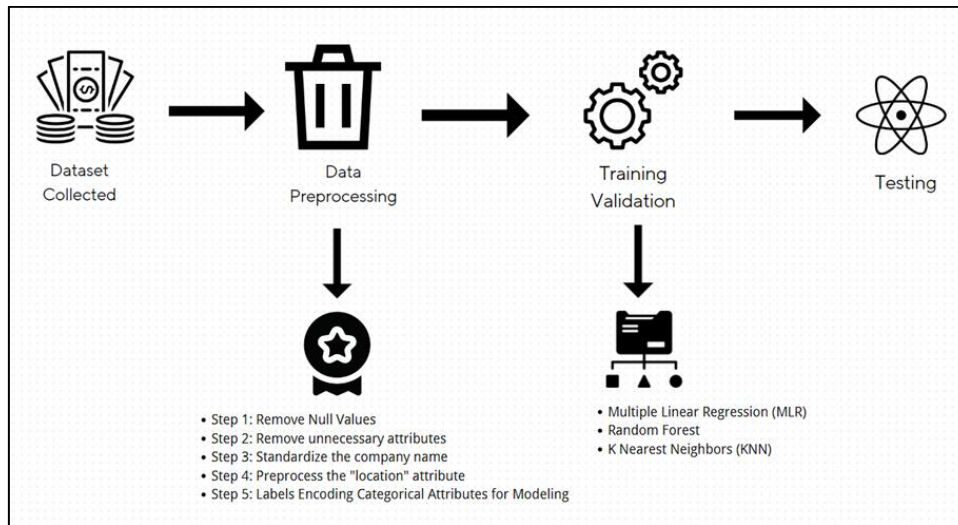


Figure 3.4: Focus HTCondor workflow.

3.2 DAGMan Process

Flask is a widely used powerful micro web framework for creating lightweight RESTful APIs on Python in which predictions will be calculated and returned when it receives input data for the bonus variables (Dholakia, 2020). RESTful API is used to exchange information securely that follow reliable and efficient software communication standards over the internet. Postman, an API platform to design and test APIs, were used to send API requests for carrying out the bonus predictions.

The first API was created for inferencing, where prediction results would be produced when input data such as years of experience, years at company, et cetera (variables that are central to the bonus prediction model) were sent over to the API through Postman.

The second API was created where DAGMAN workflow would run to produce prediction results that would be sent over to the input email address in the form of CSV file format when the API received two information - email address and a CSV file containing bonus information of job positions.

There were 4 DAGMAN workflows created for the distributed system.

3.3 Enabling Technology

1. Amazon Web Services (AWS) Cloud Service

This is the primary infrastructure used to deploy the API management platform and provide email services.

2. Flask API

This is a lightweight Python web framework that is used to quickly build web applications. It provides a simple templating system and routing mechanism to help developers quickly build web applications.

3. Network File System (NFS)

This is used to store models, allowing them to be accessed by multiple systems over a network.

4. Nginx

This is a high-performance HTTP server that is often used as a reverse proxy server, load balancer, and HTTP cache. Nginx can handle high concurrent requests and works well with various web application frameworks like Flask.

5. Python

This programming language is mainly used for all the process, including data pre-processing modelling, HTCondor DAGMan workflow, submit script.

6. Amazon Relational Database Service (RDS) with MSSQL

This is used to store and manage data.

7. User Interface (UI)

The UI is designed through Streamlit framework to serve as a communication channel between users and the data processing pipeline.

3.4 User Interface (UI)

Streamlit is a new, free, and open-source framework for rapid development and sharing of interactive data science web applications (Mhadhbi, 2021). It is a Python-based library that can assist data scientists or machine learning engineers in displaying data-related findings, collecting required parameters for machine learning modeling, and pretty much anything else without spending too much time coding user interface (Deliwala, 2019; Mhadhbi, 2021).

According to Mhadhbi, Streamlit enables developers to create an intuitive and user-friendly application with fewer lines of code. Some of its key features in this project include:

- a. It does not require developers to have front-end experience or knowledge, such as html, JavaScript, and CSS.
- b. Compatibility with majority of Python libraries, such as pandas, matplotlib, seaborn, plotly, Keras, Pytorch, Sympy (latex) etc.

By using Streamlit framework, a three-pages web application that was created in User Interface (UI) Server. Users can access this web application through configured Elastic IP address 54.163.132.24 via 8501 server port, or “54.163.132.24:8501”. The web application can be subdivided into a Personal Bonus Prediction page, Bulk Bonus Data Processing page, and About Us page along with hide able navigation menu, as shown in **Figure 3.5** and **3.6** below. Users can click or tap on the cross icon to close the navigation menu. Vice versa, users may click on or tap on the > icon to display the hidden navigation menu. This hide-able functionality can improve user friendliness for mobile web applications users.

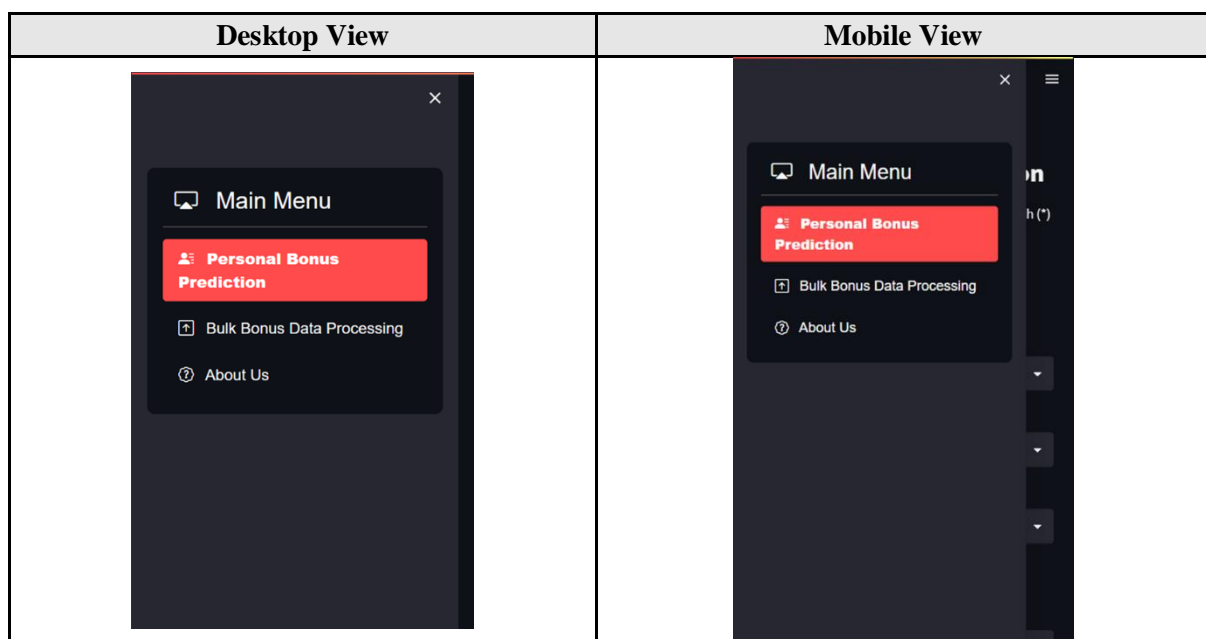


Figure 3.5: Displayed Navigation Menu for Desktop View and Mobile View

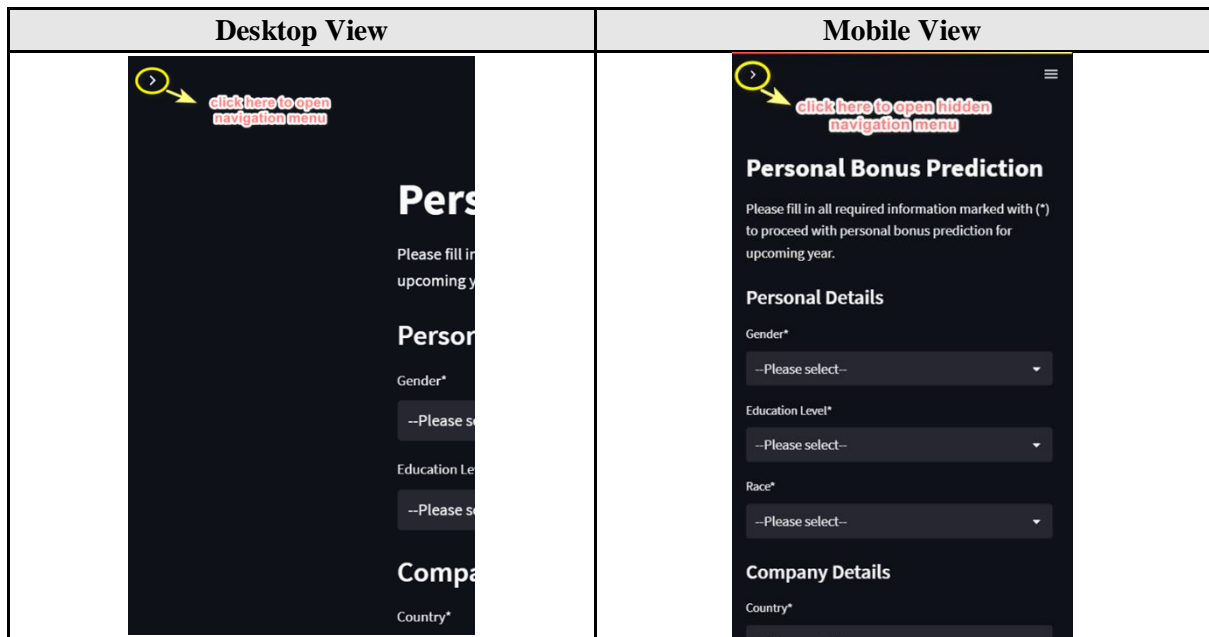


Figure 3.6: Hidden Navigation Menu for Desktop View and Mobile View

Before all mandatory input fields were filled in, the “Submit” button on related remains hidden. When users click on “Submit” button at bottom of page, web application will initiate Hypertext Transfer Protocol (HTTP) POST request to send client (browser)’s data to Amazon Web Service (AWS) server. Form-data was used as the data transmission method since people conventionally use it to support data file upload requirements (Jawahar, 2018).

3.4.1 Personal Bonus Prediction Page

Personal Bonus Prediction page can be subdivided into 4 sections, namely Personal Details, Company Details, Working Experience and Salary Package as shown in figure below. All mandatory input fields have (*) symbol at their end. Before all mandatory input fields were filled in, “Submit” button of the page will remain hidden.

Desktop View

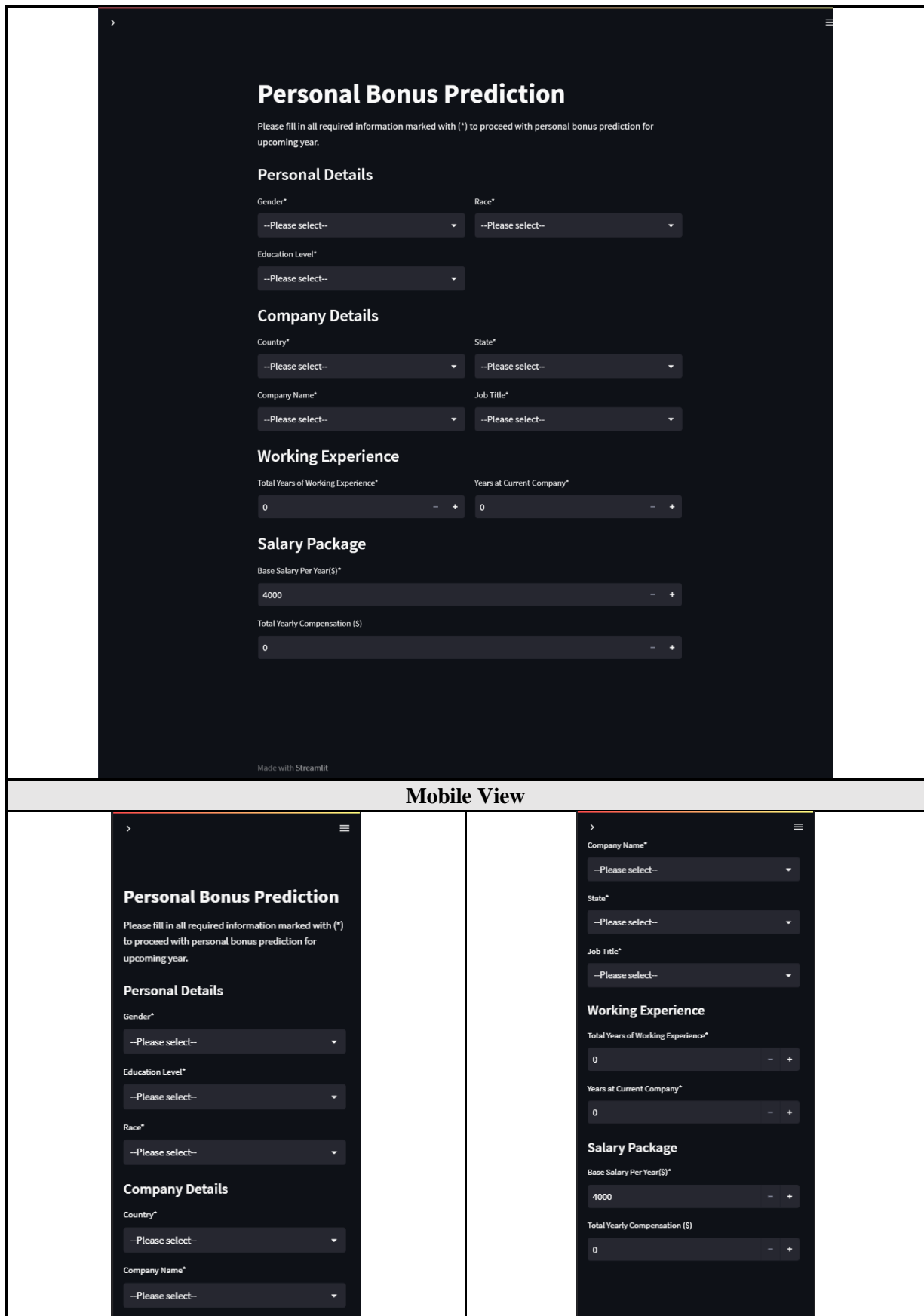


Figure 3.7: Personal Bonus Prediction Page for Desktop View and Mobile View

After all mandatory input fields were filled in, only then will the “Submit” button appear at bottom of page for user to click on. After users clicked on “Submit” button, related HTTP request will be sent to AWS server through <http://34.200.183.255:5000/predict> using form-data.

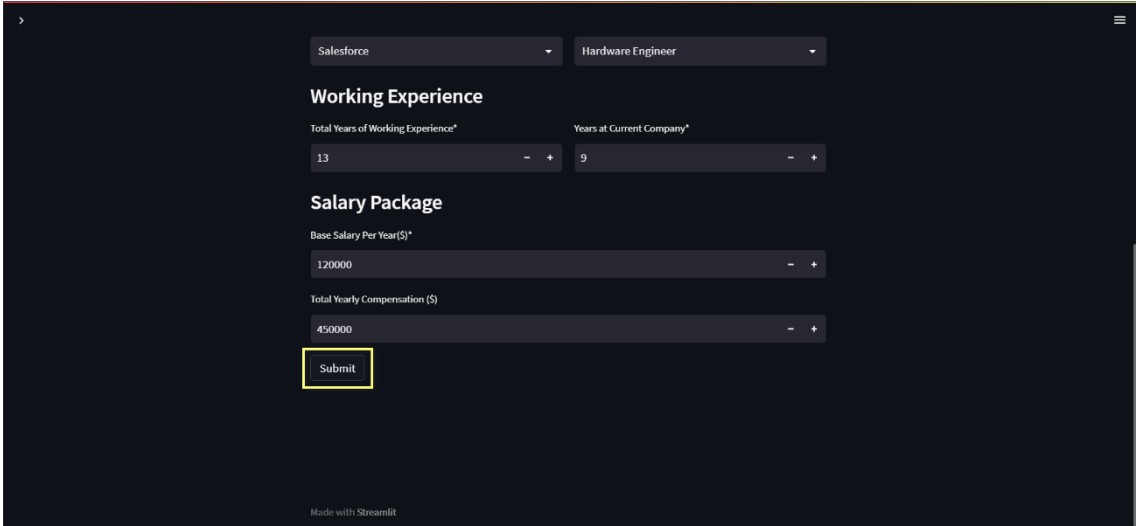
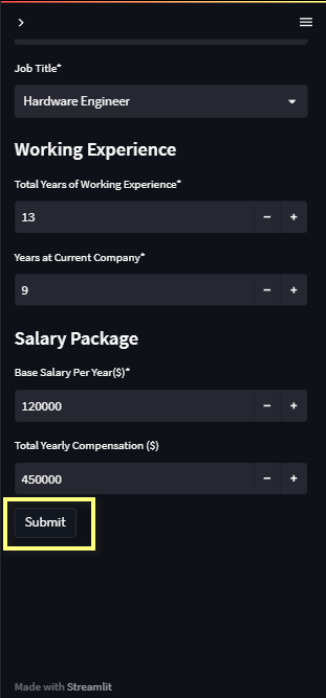
Desktop View	
	
Mobile View	
	

Figure 3.8: Personal Bonus Prediction Page with Submit button for Desktop View and Mobile View

3.4.2 Bulk Bonus Data Processing Page

The Bulk Bonus Data Processing page had 2 mandatory fields: “Email Address” Text Input field, and “Select your local CSV file” data uploader field. The “Submit” button will be hidden until all mandatory input fields were filled in.

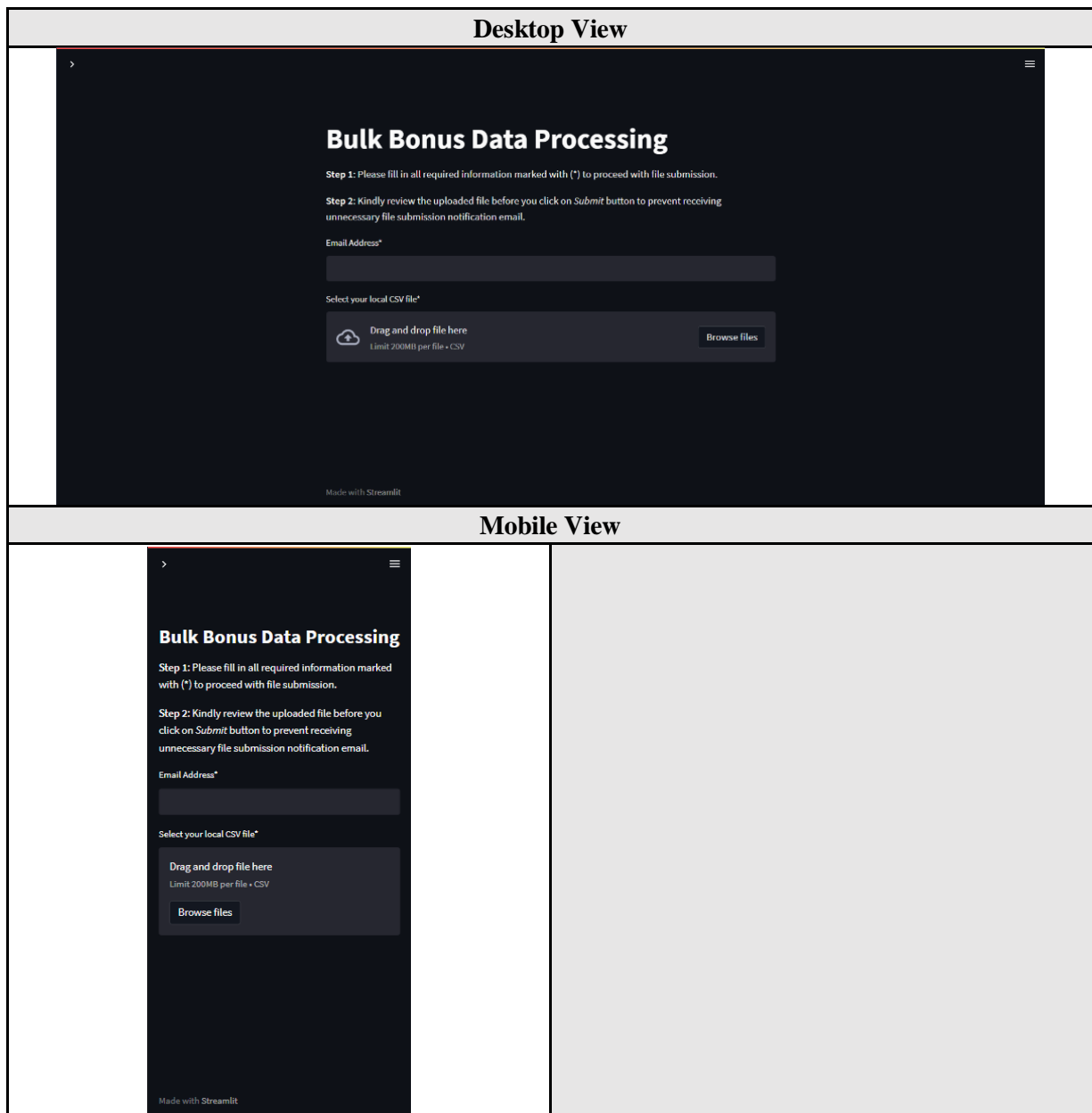
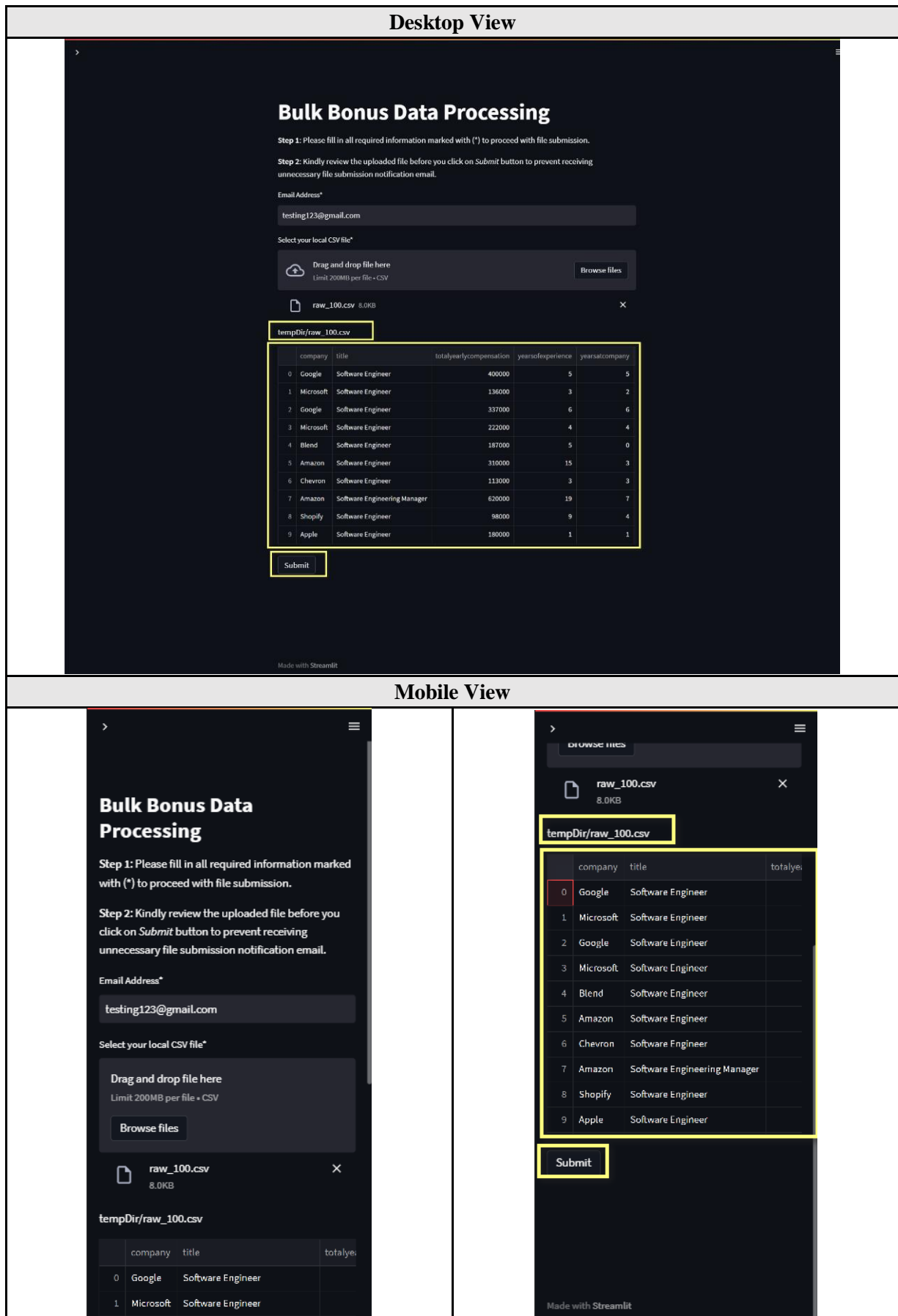


Figure 3.9: Bulk Bonus Data Processing Page for Desktop View and Mobile View

After all mandatory input fields were filled in, the web application will: (i) display the server’s data file path, (ii) display the top 10 rows of data for the user to review, and (iii) display a “Submit” button at bottom of page for users to click. After users clicked on Submit button, related HTTP request will be sent to the AWS server via form-data at http://34.200.183.255:5000/upload_CSV.



3.4.3 Basic Web Service Error Handling

It is normal to see HTTP errors when one needs to transmit data over the Internet. Since web services related failures are unavoidable, it's a standard practice for web application to degrade gracefully in the presence of failures with minimal impact on customers' experience (Fichman, 2018). There might be times when Nginx Web Server is down or even total failure on web services (Fichman, 2018). Thus, some codes were written to display error code and error messages in a user friendlier manner.

```
if response.status_code == 200:
    # modified code
    # 1. define prediction as response returned
    prediction = response.text
    # 2. place prediction amount into output subheader below
    st.text(f"The predicted amount of upcoming bonus is ${prediction}!")
    # clear existing session state
else:
    # display returned error code if not 200 is returned
    st.error("Error: {}".format(response.status_code))
    print(response.text.encode('utf8'))
```

Figure 3.11: Code Excerpt for Basic Web Service Error Handling

4.0 Discussion

Keeping up with rapid technological advancement, particularly through traditional architectural design, has become increasingly difficult. However, the well-known discoveries of Microservices and Containers have made the implementation of distributed systems much simpler and more significant. As a result, relevant studies were performed on these two relatively new architecture terms and their differences from the traditional monolith architecture system.

Figure 4.1 depicted a monolithic system that uses an all-in-one architecture for software to operate as a single unit. As compared to distributed systems, a monolith allows fast and uncomplicated deployment due to having an application based on a single codebase. Besides, one API within the monolith can perform the same function that numerous APIs perform with distributed system which saves a lot of hassles. Furthermore, since it has one codebase for the application to work, the testing is simplified, and debugging will be easier as all the codes are in one place (Harris, n.d.).

Nonetheless, monoliths are difficult to scale and modify. It does not provide the flexibility to scale only one segment of the system; instead, the entire system must be scaled at once (Integrate.io, n.d.). Because of the architecture of a monolithic system, it is more susceptible to slowdown issues as user load increases. The maintenance of the system's code base becomes more difficult as the system becomes more complex (Integrate.io, n.d.). Not to mention the difficulty in integrating monolithic systems with new technologies, as code rewrite is unavoidable prior to successful integration with other technologies (Integrate.io, n.d.).

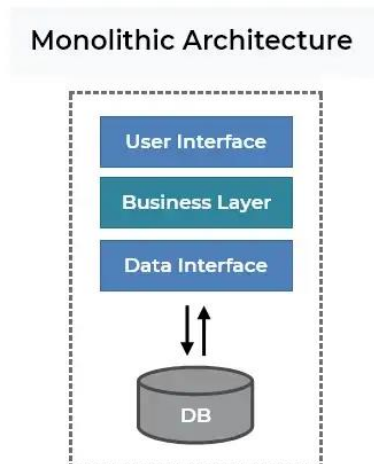


Figure 4.1: Monolithic Architecture (Source: Medium. <https://henriquesd.medium.com/monolithic-microservices-architecture-239e8799d3e1>)

Fortunately, there is a better alternative to monolithic architecture – distributed system, as shown in **Figure 4.2**. Despite their substantial implementation costs, distributed systems are more efficient. To recap, a distributed system is essentially a collection of linked machines that can perform a task in parallel remote access protocols. They are designed to be efficient and cost-effective in the long run thanks to the use of multiple computers to perform independent fast processing concurrently. Distributed systems are made to be scalable by default workload; they can be scaled to handle huge volumes of tasks reliably.

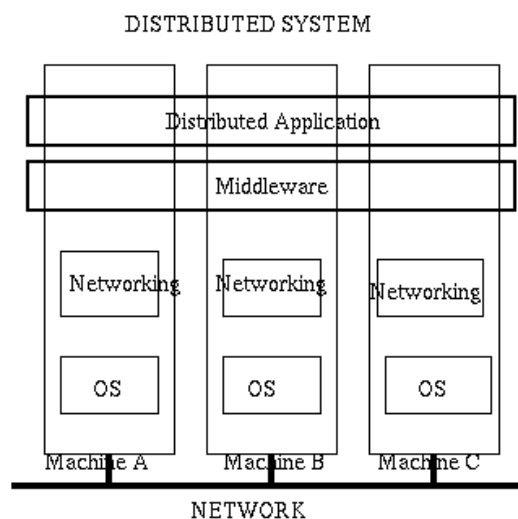


Figure 4.2: Distributed architecture (Source: <https://cis.temple.edu/~giorgio/cis307/readings/client-server.html>)

It is possible to break monolith applications into Microservices with AWS Lambda, which is one of the types of distributed system. With Microservices, an application is built on multiple components that run each process of the application as a service, where each component has their respective codebases. Microservices are way more scalable than monolithic architecture because each microservice can be scaled independently without affecting the other services within the application. This also eliminates the “single point of failure” in the application (Medium, 2021). It is also possible for each microservice to have its own database to cater for the needs of each microservice and there are many more advantages in using the Microservices architecture for large and long-term projects.

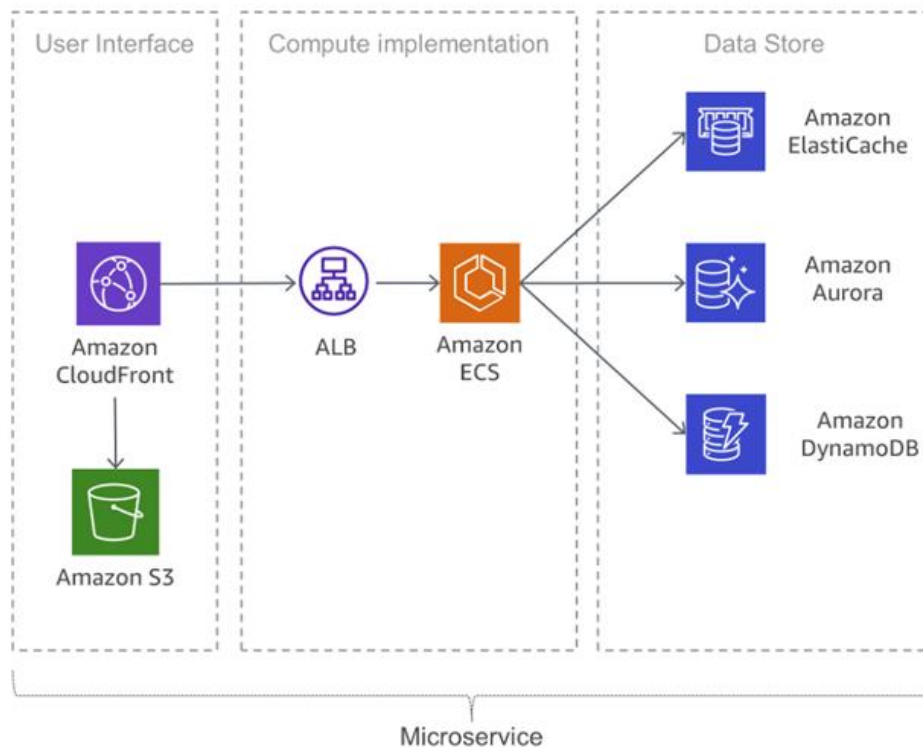


Figure 4.3: Microservices architecture on AWS (Source: <https://docs.aws.amazon.com/whitepapers/latest/microservices-on-aws/simple-microservices-architecture-on-aws.html>)

Another interesting term that is worth noting is a container, which is a form of operating system virtualisation that can run anything from a microservice to a larger application and within the container includes the necessary executables, binary code, configuration files, and dependencies. Containers are resources effective with increased portability which can be easily deployed to multiple different operating systems and hardware platforms (NetApp, n.d.).

5.0 Conclusion

The central focus for this project was to deploy a machine learning predictive model for STEM bonus prediction using Amazon Web Services (AWS). The model was trained using a dataset collected from various sources and pre-processed to remove null values, remove unnecessary attributes, and standardise the data.

A combination of different machine learning algorithms was applied such as Multi Linear Regression (MLR), Random Forest, and K-Nearest Neighbours (KNN) to model the data. Random Forest as the best algorithm was chosen for the subsequent workflow. A distributed system was then developed using HTCondor, NFS, and AWS.

A web application was also created using the Streamlit framework to provide users with an intuitive and user-friendly interface for entering data and receiving bonus predictions. A bulk data processing page was also available, where users could upload a CSV file and receive their predictions via email. Nonetheless, because of its monolithic architecture, this prototype web application was not truly scalable.

The project ultimately led to the development of a robust and scalable system that was able to predict STEM bonuses with a high degree of accuracy. However, it is important to note that there were some limitations that were encountered during the project. One of these limitations was the small size of the

dataset used to train and test the system. While the dataset was sufficient for the purpose of demonstrating the capabilities of the system, it may not be large enough to fully reflect the complexity and variability of real-world scenarios.

In short, deploying machine learning models on AWS provides a few advantages such as scalability, reliability, and cost-effectiveness. The use of distributed systems and web services further enhances the performance and usability of the system (“Machine Learning (ML) - Overview of Amazon Web Services,” 2023). In the future, it may be possible to expand the dataset and improve the model accuracy by incorporating more data and fine-tuning the parameters.

6.0 Future outlook

The outlook for this project is promising, as the use of distributed systems for machine learning and data processing is becoming increasingly important. As the volume and complexity of data continues to grow, it is becoming increasingly challenging to process and analyse data using traditional methods (OECD, 2021). Distributed systems offer a powerful solution to this problem by allowing multiple processors or nodes to work in parallel to solve a problem, leading to stronger computing power, improved performance, and shorter computation time.

In the future, this platform can be further optimised to improve its performance, scalability, and flexibility. For example, the platform could be enhanced to support more advanced machine learning algorithms, or to integrate with other data processing tools such as Apache Spark or TensorFlow. Additionally, the platform could be integrated with other cloud services to improve its scalability, security, and cost-effectiveness. As for the web application, it could be enhanced into a distributed application when its’ scalability needs arise (Lutkevich, 2022).

While developing the project, we encountered several difficulties such as architecture complexity, data management, and communication barriers between team members due to knowledge and expertise gaps. To overcome these obstacles, we consulted workforce experts and used specialized software to build the designed distributed system. Furthermore, we conducted extensive system analysis and design to ensure compatibility between soon-to-be integrated technologies and services without losing sight of the project's agreed-upon goals and objectives. Collaboration between team members is critical for the project's success, and we fostered it throughout the project.

Overall, the outlook for this project is positive, and it has the potential to make a significant impact in the field of machine learning and data processing. With the help of this platform, organisations and researchers will be able to process and analyse large amounts of data more efficiently and effectively, leading to new insights and discoveries.

7.0 References

- Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mane, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viegas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., & Zheng, X. (2016). TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. *Preliminary White Paper*. DOI: <https://arxiv.org/pdf/1603.04467.pdf> <http://arxiv.org/abs/1603.04467.pdf>
- AWS. (2021). *AWS Architecture Center*.
- AWS. (2023). About AWS. AWS. Retrieved from <https://aws.amazon.com/about-aws/>
- AWS. (2023). Benefits at a glance. AWS. Retrieved from <https://aws.amazon.com/application-hosting/benefits/>
- Barbera, P., Kozlov, A.M., Czech, L., Morel, B., Darriba, D., Flouri, T., & Stamatakis, A. EPA-ng: massively parallel evolutionary placement of genetic sequences. *Systematic Biology*, 68(2), 365-369. DOI: <https://doi.org/10.1093/sysbio/syy054>
- Bundak, C.E.A., Rahman, M.A.A., Karim, M.K.A., & Osman, N.H. (2021). Effect of different signal weighting function of magnetic field using KNN for indoor localization. *Recent Trends in Mechatronics Towards Industry 4.0*, 571-581. Retrieved from https://link.springer.com/chapter/10.1007/978-981-33-4597-3_52
- Callaghan, B. (2004). *NFS Illustrated*.
- Deliwala, S. (2019). Streamlit 101: An in-depth introduction. *Medium*. Retrieved from <https://towardsdatascience.com/streamlit-101-an-in-depth-introduction-fc8aad9492f2>
- Domareski, H. S. (2021). Monolithic & Microservices Architecture. *Medium*. <https://henriquesd.medium.com/monolithic-microservices-architecture-239e8799d3e1>
- Eisler, M., & Labiaga, R. (2010). *NFS and NIS*.
- Dholakia, J. (2020). Creating RESTful Web APIs using Flask and Python. *Towards Data Science*. Retrieved from <https://towardsdatascience.com/creating-restful-apis-using-flask-and-python-655bad51b24>
- Grant, P. (2019). k-Nearest Neighbors and the curse of dimensionality. *Towards Data Science*. Retrieved from <https://towardsdatascience.com/k-nearest-neighbors-and-the-curse-of-dimensionality-e39d10a6105d>
- Hachcham, A. (2022). The KNN algorithm – explanation, opportunities, limitations. *Neptune.AI*. Retrieved from <https://neptune.ai/blog/knn-algorithm-explanation-opportunities-limitations#:~:text=KNN%20is%20widely%20known%20as,a%20training%20phase%20at%20a%20>
- Harris, H. (n.d.) Microservices vs. Monolithic Architecture. *Atlassian*. <https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith#:~:text=A%20monolithic%20architecture%20is%20a%20singular%2C%20large%20computing%20network%20with,of%20the%20service%20Dside%20interface>
- Ho, T.K. (1998). The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8), 832-844. DOI: <https://doi.org/10.1109/34.709601>

- HTCondor. (2021). An overview of the HTCondor system. *HTCondor*. Retrieved from <https://research.cs.wisc.edu/htcondor/overview/>
- Integrate.io. (n.d.). What is Monolithic Architecture? *Integrate.io*. Retrieved from <https://www.integrate.io/glossary/what-is-monolithic-architecture/>
- IntelliPaat. (2022). Top 7 benefits of AWS - advantages and disadvantages of Amazon Web Services. *IntelliPaat*. Retrieved from <https://intellipaat.com/blog/aws-benefits-and-drawbacks/#no5>
- Jawahar, R. (2018). Content Type: x-www-form-urlencoded, form-data and json. *Medium*. <https://medium.com/@rajjawahar77/content-type-x-www-form-urlencoded-form-data-and-json-e17c15926c69>
- Jin, H., Jespersen, D.C., Mehrota, P., & Biswas, R. (2011). High performance computing using MPI and OpenMP on multi-core parallel systems. *Parallel Computing*, 37(9), 562-575. DOI: <http://dx.doi.org/10.1016/j.parco.2011.02.002>
- Jordanov, I., Petrov, N., & Petrozziello, A. (2016). Supervised radar signal classification. *International Joint Conference on Neural Networks (IJCNN)*, 1464-1471. DOI: 10.1109/IJCNN.2016.7727371
- Kayri, M., Kayri, I., & Gencoglu, M.T. (2017). The performance comparison of Multiple Linear Regression, Random Forest and Artificial Neural Network by using photovoltaic and atmospheric data. *14th International Conference on Engineering of Modern Electric Systems (EMES)*, 1-4. DOI: 10.1109/EMES.2017.798036
- Kumar, D.S.M., & Sadashiv, N. (2011). Cluster, grid and cloud computing: a detailed comparison. *6th IEEE International Conference on Computer Science and Education (ICCSE '11)*. DOI: <http://dx.doi.org/10.1109/ICCSE.2011.6028683>
- Lamport, L. (2019). Time, clocks, and the ordering of events in a distributed system. *Concurrency, the Works of Leslie Lamport*, 179-196. DOI: <https://doi.org/10.1145/3335772.3335934>
- Livny, M., Tannenbaum, T., Mutka, M.W., & Livny, M. (2006). *HTCondor: High-Throughput Computing on Compute Clusters*.
- Lutkevich, B. (2022). What is distributed applications? *TechTarget; IT Operation*. Retrieved from <https://www.techtarget.com/searchitoperations/definition/distributed-applications-distributed-apps>
- Machine Learning (ML) - Overview of Amazon Web Services. (2023). Retrieved January 26, 2023, from Amazon.com website: <https://docs.aws.amazon.com/whitepapers/latest/aws-overview/machine-learning.html>
- Maniruzzaman, M., Rahman, M.J., Al-MehediHasan, M., Suri, H.S., Abedin, M.M., El-Baz, A., & Suri, J.S. (2018). Accurate diabetes risk stratification using machine learning: role of missing values and outliers. *Journal of Medical Systems*, 42(92). Retrieved from <https://link.springer.com/article/10.1007/s10916-018-0940-7>
- Mapelu, I. C., Bor, T., Nthiga, R., Kamwea, J., & Yego, J. (2013). Determinants of the level of revenue of tourist enterprises within the north coastal region of Kenya. *Journal of Tourism, Hospitality and Sports*, 1, 23-27.
- Mata, J. (2011). Interpretation of concrete dam behaviour with artificial neural network and multiple linear regression models. *Engineering Structures*, 33(3), 903-910. Retrieved from <https://www.sciencedirect.com/science/article/pii/S0141029610004839>

- Mhadhbi, N. (2021). Python Tutorial: Streamlit. *Datacamp*. Retrieved from <https://www.datacamp.com/tutorial/streamlit>
- Miller, R. (2016). Big Sur: a closer look at the engine powering Facebook's AI. *Data Center Frontier*. Retrieved from <https://www.datacenterfrontier.com/cloud/article/11431168/big-sur-a-closer-look-at-the-engine-powering-facebook8217s-ai>
- Moore, A.W., Anderson, B., Das, K., & Wong, W. (2006). Chapter 15 – combining multiple signals for biosurveillance. *Handbook of Biosurveillance*, 234-242. Retrieved from <https://www.sciencedirect.com/science/article/pii/B978012369378550017X>
- Mueller, J.P. (2016). *AWS: Up and Running*.
- OECD. (2021). Artificial Intelligence, Machine Learning and Big Data in Finance Opportunities, Challenges and Implications for Policy Makers. Retrieved from <https://www.oecd.org/finance/financial-markets/Artificial-intelligence-machine-learning-big-data-in-finance.pdf>
- Ordiano, J.A.G., Doneit, W., Waczowicz, S., Groll, L., Mikut, R., & Hagenmeyer, V. (2017). Nearest-neighbor based non-parametric probabilistic forecasting with applications in photovoltaic systems. DOI: <https://doi.org/10.48550/arXiv.1701.06463>
- Owen, S., Anil, R., Dunning, T., & Friedman, E. (2010). *Mahout in Action*.
- PaddlePaddle. (2020). An open-source deep learning platform originated from industrial practice. *PaddlePaddle*. Retrieved from <https://www.paddlepaddle.org.cn/en>
- Rashid, Z.N., Zeebaree, S.R.M., & Shengul, A. (2019). Design and analysis of proposed remote controlling distributed parallel computing system over the cloud. *International Conference on Advanced Science and Engineering (ICOASE)*. DOI: <https://doi.org/10.1109/ICOASE.2019.8723695>
- Shaik, A.B., & Srinivasan, S. (2018). A brief survey on Random Forest Ensembles in classification model. *International Conference on Innovative Computing and Communications*, 253-260. Retrieved from https://link.springer.com/chapter/10.1007/978-981-13-2354-6_27
- Smith, I.C. (2018). Cloud computing with HTCondor. Retrieved from https://condor.liv.ac.uk/cloud/Condor_cloud.pdf
- Tipton, R.W. (2015). *Understanding Network File System (NFS)*.
- Wang, Z., Wang, Y., Z, R., Srinivasan, R.S., & Ahrentzen, S. (2018). Random Forest based hourly building energy prediction. *Energy and Buildings*, 171, 11-25. Retrieved from <https://www.sciencedirect.com/science/article/pii/S0378778818311290>
- NetApp. (n.d.) What are containers. *NetApp*. <https://www.netapp.com/devops-solutions/what-are-containers/#:~:text=Containers%20are%20a%20form%20of,%2C%20libraries%2C%20and%20configuration%20files>
- Zhang, M. (2022). Amazon Web Services (AWS) data center locations: regions and availability zones. Dgtl Infra. Retrieved from [https://dgtlinfra.com/amazon-web-services-aws-data-center-locations/#:~:text=Regions-,Within%20these%20markets%2C%20Amazon%20Web%20Services%20\(AWS\)%20operates%20data,%2C%20and%20China%20\(Ningxia](https://dgtlinfra.com/amazon-web-services-aws-data-center-locations/#:~:text=Regions-,Within%20these%20markets%2C%20Amazon%20Web%20Services%20(AWS)%20operates%20data,%2C%20and%20China%20(Ningxia)

END.