Configuration Manual

MSc Research Project
MSC Data Analytics

Devika Kulkarni
Student ID: X19202865

School of Computing
National College of Ireland

Supervisor: Dr. Catherine Mulwa

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Devika Kulkarni
Student ID:	X19202865
Programme:	MSC Data Analytics
Year:	2021
Module:	MSc Research Project
Supervisor:	Dr. Catherine Mulwa
Submission Due Date:	16/12/2021
Project Title:	Configuration Manual
Word Count:	970
Page Count:	18

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Devika Kulkarni
Date:	29th January 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Devika Kulkarni
X19202865

1 Introduction

This is a configuration manual document that outlines a step-by-step approach for putting the project on US visa analysis into action. The goal of this project was to analyze US visas.Five machine learning models were run to identify the best match model. The performance of all the five models were implemented, evaluated and the results, Details on the installation and overall implementation of the code are provided in the following sections.

2 Prerequisites

This section contains information about the software specs that were utilized to complete this project and the minimum requirements. It also walks you through the process of installing the program or application step by step.

2.1 Hardware Requirements

- Operating system: Windows 10
- Processor: Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz 1.99 GHz
- RAM: 8 GB
- System type: 64-bit operating system, x64-based processor
- HDD: 1 TB

2.2 Software Requirements

Python was utilized as the programming language for this project, while Jupyter Notebook and Google colab were both used to implement the code. However, in this document, Jupyter Notebook is used to guide through the implementation of the code. Python's seaborn and matplotlib libraries were used to create the visualizations. A step by step guide to download Python on windows is available at learning Lounge (2021). Following are the versions of these software:

- Python: 3.8.3
- Jupyter Notebook: 6.0.3

3 Environment Setup

This sections guides through the environment setup required for running the code.

- To use Jupyter notebook, Anaconda software can be downloaded here. Also, to install anaconda you can refer Anaconda (2021) [1]
  Figure 1 shows the website to download Anaconda software.

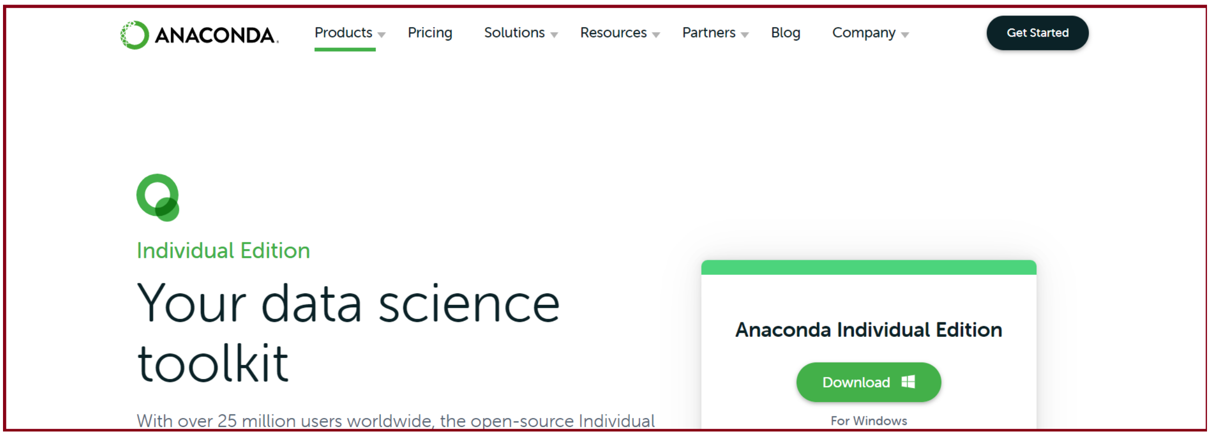

Figure 1: Website to download Anaconda

- Once Anaconda is installed, Jupyter notebook needs to be installed through Anaconda software. Open Anaconda software and refer to Figure 2 to install Jupyter Notebook.

[1] https://www.anaconda.com/products/individualwindows

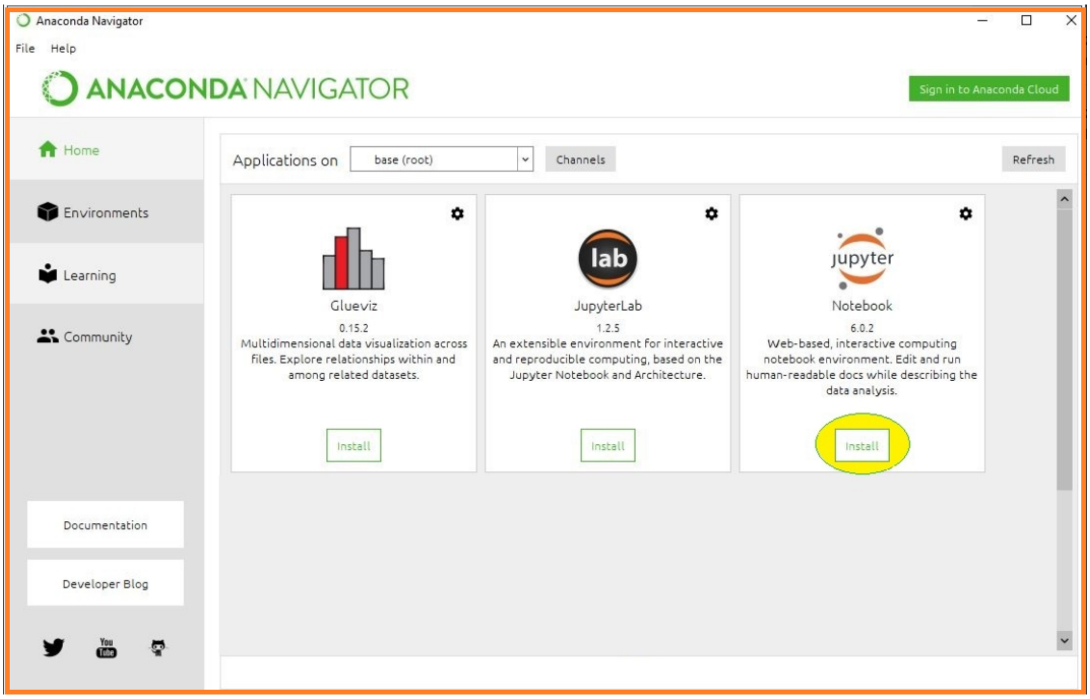

Figure 2: Installing Jupyter Notebook using Anaconda

- Figure 3 shows how to launch Jupyter Notebook.

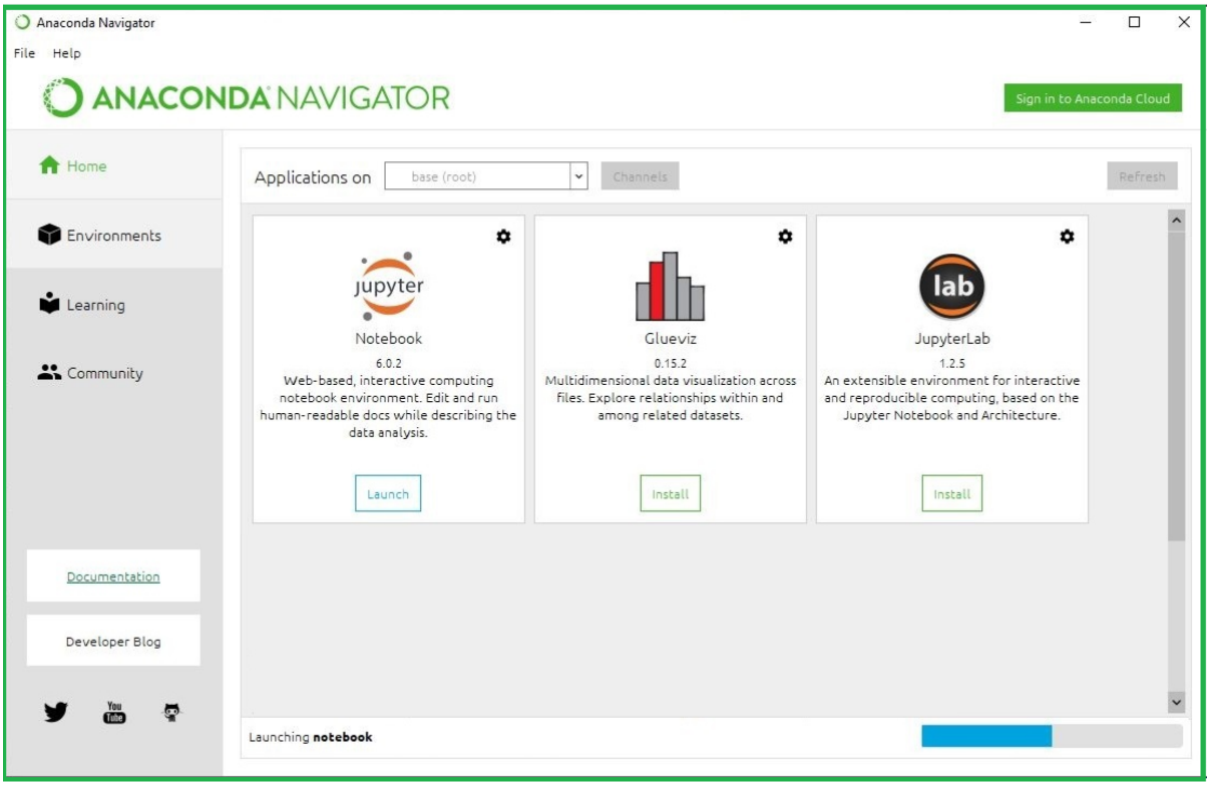

Figure 3: Launch Jupyter Notebook

- After Jupyter notebook is launched, click on the "New" button and then click on "Python 3" to create a notebook. Please refer Figure 4.

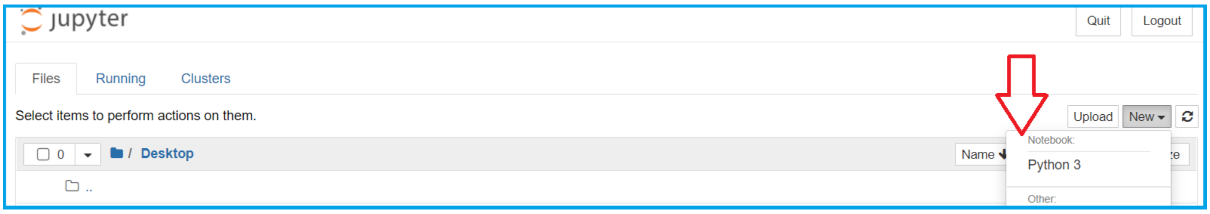

Figure 4: Creating a Notebook

- To start with the code, libraries needs to be imported first. To import the libraries refer the Figure 4 and run the cell by typing the code and press shift + enter together to run the cell.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Figure 5: Importing Libraries

4 Data Preprocessing and Data Cleaning

- The data set was acquired from Kaggle.com [2]
- To read the.csv file, set the file on the location or path, and then provide the path to read the data. Figure 6 shows the code to read the .csv file in the notebook.

```
#Reading the csv file
df = pd.read_csv(r'C:\Users\dkulk\Desktop\Research Project\us_perm_visa.csv')
df.head()
```

Figure 6: Reading .csv file

- To display the first 10 rows, following line of code can be used shown in Figure 7.

[2]https://www.kaggle.com/

```
In [4]: # Displaying 10 first rows
        df.head(10)
```

Out[4]:

	add_these_pw_job_title_9089	agent_city	agent_firm_name	agent_state	application_type	case_no	case_number	case_received_date	case_status	class_of_a
0	NaN	NaN	NaN	NaN	PERM	A-07323-97014	NaN	NaN	Certified	
1	NaN	NaN	NaN	NaN	PERM	A-07332-99439	NaN	NaN	Denied	
2	NaN	NaN	NaN	NaN	PERM	A-07333-99643	NaN	NaN	Certified	
3	NaN	NaN	NaN	NaN	PERM	A-07339-01930	NaN	NaN	Certified	
4	NaN	NaN	NaN	NaN	PERM	A-07345-03565	NaN	NaN	Certified	
5	NaN	NaN	NaN	NaN	PERM	A-07352-06288	NaN	NaN	Denied	
6	NaN	NaN	NaN	NaN	PERM	A-07354-06926	NaN	NaN	Certified-Expired	
7	NaN	NaN	NaN	NaN	PERM	A-08004-10147	NaN	NaN	Denied	
8	NaN	NaN	NaN	NaN	PERM	A-08004-10184	NaN	NaN	Certified	
9	NaN	NaN	NaN	NaN	PERM	A-08010-11785	NaN	NaN	Denied	

10 rows × 154 columns

Figure 7: Displaying first 10 rows in the data set

- To display the columns from the data set, refer Figure 8.

```
In [6]: print(df.columns.values)

['add_these_pw_job_title_9089' 'agent_city' 'agent_firm_name'
 'agent_state' 'application_type' 'case_no' 'case_number'
 'case_received_date' 'case_status' 'class_of_admission'
 'country_of_citizenship' 'country_of_citzenship' 'decision_date'
 'employer_address_1' 'employer_address_2' 'employer_city'
 'employer_country' 'employer_decl_info_title' 'employer_name'
 'employer_num_employees' 'employer_phone' 'employer_phone_ext'
 'employer_postal_code' 'employer_state' 'employer_yr_estab'
 'foreign_worker_info_alt_edu_experience'
 'foreign_worker_info_birth_country' 'foreign_worker_info_city'
 'foreign_worker_info_education' 'foreign_worker_info_education_other'
 'foreign_worker_info_inst' 'foreign_worker_info_major'
 'foreign_worker_info_postal_code' 'foreign_worker_info_rel_occup_exp'
 'foreign_worker_info_req_experience' 'foreign_worker_info_state'
 'foreign_worker_info_training_comp' 'foreign_worker_ownership_interest'
 'foreign_worker_yr_rel_edu_completed' 'fw_info_alt_edu_experience'
 'fw_info_birth_country' 'fw_info_education_other' 'fw_info_postal_code'
 'fw_info_rel_occup_exp' 'fw_info_req_experience' 'fw_info_training_comp'
 'fw_info_yr_rel_edu_completed' 'fw_ownership_interest'
 'ji_foreign_worker_live_on_premises' 'ji_fw_live_on_premises'
 'ji_live_in_dom_svc_contract' 'ji_live_in_domestic_service'
 'ji_offered_to_sec_j_foreign_worker' 'ji_offered_to_sec_j_fw'
 'job_info_alt_cmb_ed_oth_yrs' 'job_info_alt_combo_ed'
 'job_info_alt_combo_ed_exp' 'job_info_alt_combo_ed_other'
 'job_info_alt_field' 'job_info_alt_field_name' 'job_info_alt_occ'
 'job_info_alt_occ_job_title' 'job_info_alt_occ_num_months'
 'job_info_combo_occupation' 'job_info_education'
 'job_info_education_other' 'job_info_experience'
 'job_info_experience_num_months' 'job_info_foreign_ed'
 'job_info_foreign_lang_req' 'job_info_job_req_normal'
 'job_info_job_title' 'job_info_major' 'job_info_training'
 'job_info_training_field' 'job_info_training_num_months'
 'job_info_work_city' 'job_info_work_postal_code' 'job_info_work_state'
 'naics_2007_us_code' 'naics_2007_us_title' 'naics_code' 'naics_title'
 'naics_us_code' 'naics_us_code_2007' 'naics_us_title'
 'naics_us_title_2007' 'orig_case_no' 'orig_file_date'
 'preparer_info_emp_completed' 'preparer_info_title' 'pw_amount_9089'
 'pw_determ_date' 'pw_expire_date' 'pw_job_title_908' 'pw_job_title_9089'
```

Figure 8: Displaying columns in the data set

- All the null values were dropped from the data set. To execute the code please refer Figure 9 below.

```
In [11]: #Dropping all empty columns
         df = df.dropna(axis=1, how='all');

         #Dropping all empty rows
         df = df.dropna(axis=0, how='all');

         df.shape

Out[11]: (356168, 153)
```

Figure 9: Dropping null values from the data set

- Missing values must be eliminated or discarded from the data collection in order to erase the columns and clean the data. The code to display missing columns is shown in Figure 10.

```
In [12]: # Displaying number of missing values in each column
         for column in df.columns:
             print("Attribute '{}' contains ".format(column),  df[column].isnull().sum().sum(), " missing values")

         Attribute 'add_these_pw_job_title_9089' contains  317031  missing values
         Attribute 'agent_city' contains  153452  missing values
         Attribute 'agent_firm_name' contains  157646  missing values
         Attribute 'agent_state' contains  156544  missing values
         Attribute 'application_type' contains  229320  missing values
         Attribute 'case_received_date' contains  126848  missing values
         Attribute 'case_status' contains  0  missing values
         Attribute 'class_of_admission' contains  21085  missing values
         Attribute 'country_of_citizenship' contains  19272  missing values
         Attribute 'country_of_citzenship' contains  336951  missing values
         Attribute 'decision_date' contains  0  missing values
         Attribute 'employer_address_1' contains  37  missing values
         Attribute 'employer_address_2' contains  141026  missing values
         Attribute 'employer_city' contains  10  missing values
         Attribute 'employer_country' contains  126920  missing values
         Attribute 'employer_decl_info_title' contains  126885  missing values
         Attribute 'employer_name' contains  8  missing values
         Attribute 'employer_num_employees' contains  126925  missing values
         Attribute 'employer_phone' contains  126883  missing values
```

Figure 10: Number of missing columns in the data set

5 Exploratory Data Analysis

To study the data set, EDA is required. With the use of visuals, EDA allows to better examine and understand the data set. The data set employed was huge, and there were numerous fields that could be studied in a variety of ways. Only a few visualizations are given in this document to provide context and familiarity with the data set. Figure 11 shows 20 most popular cities for which visa applications were filed.

```
In [14]: # Displaying 15 most popular cities
         df['employer_city'] = df['employer_city'].str.lower()
         df['employer_city'].value_counts().head(20)

Out[14]: new york           17198
         college station    11985
         santa clara        10519
         san jose            9147
         redmond             8485
         mountain view       8121
         houston             6720
         san francisco       6352
         sunnyvale           6104
         plano               5607
         chicago             5561
         seattle             5051
         edison              4056
         los angeles         4045
         san diego           3702
         dallas              3693
         philadelphia        3526
         cupertino           3310
         palo alto           3229
         irving              3144
         Name: employer_city, dtype: int64
```

Figure 11: Most popular cities in the US for visa applications

- To set the plot parameters the two variables were chosen as x and y axis. On x axis, employer city was taken and on y axis total number of visa applications. Figure 12 shows the plot parameters to show the number of applications in the employer city.

```
# Setting plot parameters
fig, ax = plt.subplots()
fig.set_size_inches(13.7, 8.27)
sns.set_context("paper", rc={"font.size":12,"axes.titlesize":12,"axes.labelsize":12})
sns.countplot(x='employer_city', hue='year', data=df, order=df.employer_city.value_counts().iloc[:10].index)
plt.xticks(rotation=90)
ax.set(xlabel='Employer city', ylabel='Total number of visa applications')
```

```
[Text(0, 0.5, 'Total number of visa applications'),
 Text(0.5, 0, 'Employer city')]
```

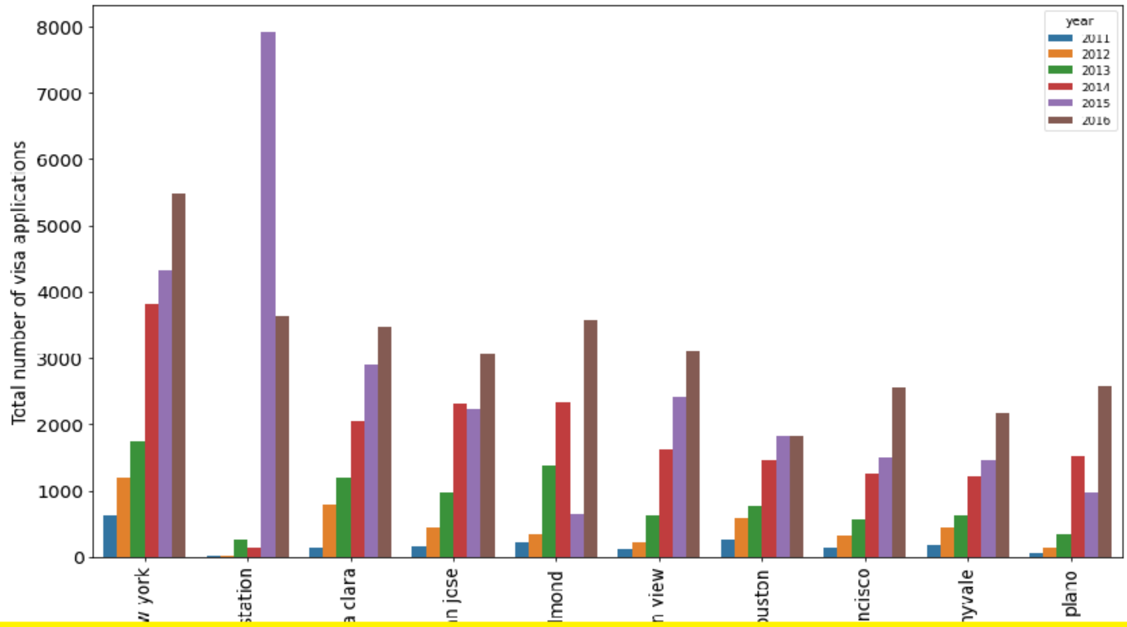

Figure 12: Total visa applications on employer city

- Another visual in Figure 13 depicting the number of applications for particular organization.

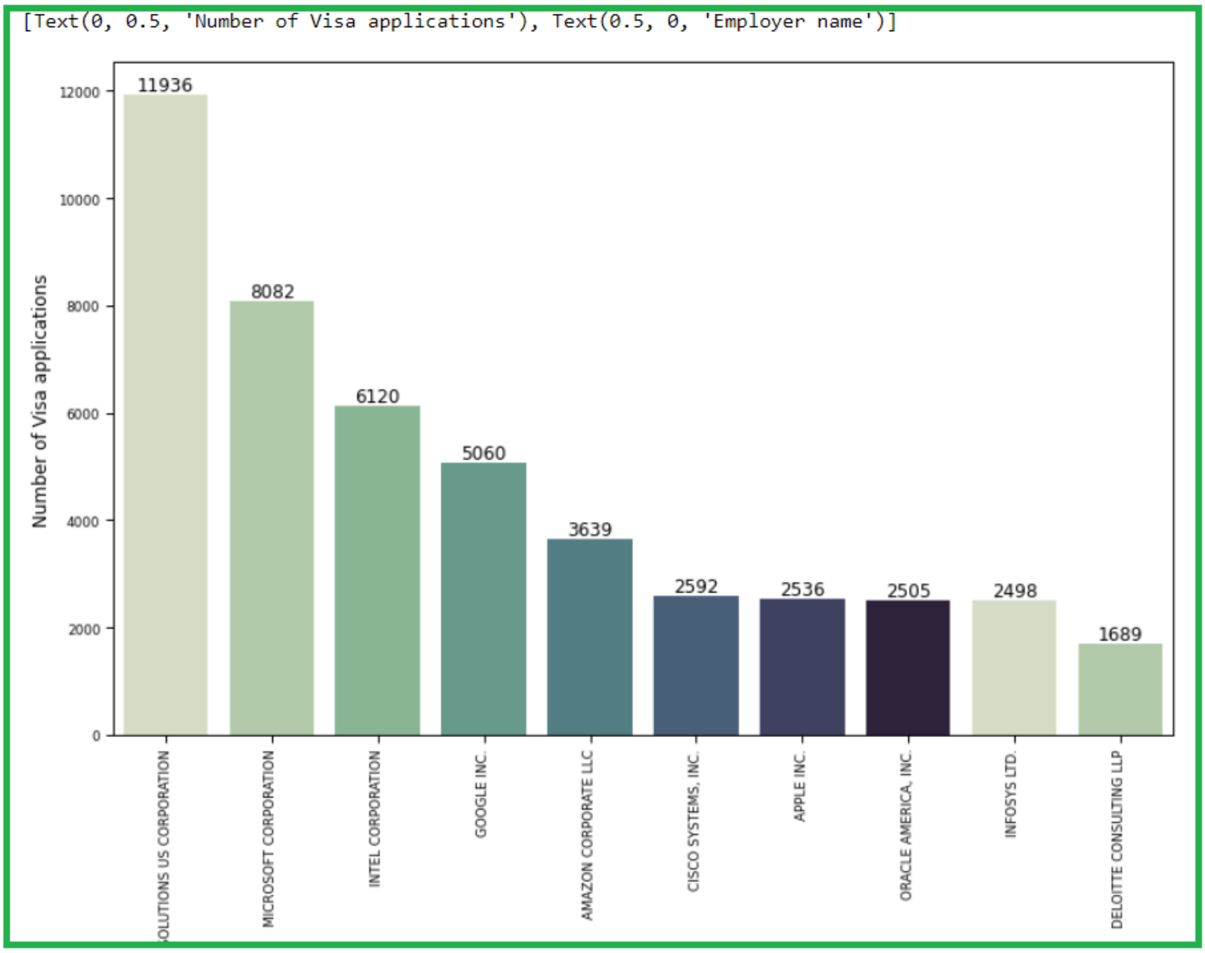

Figure 13: Total visa applications for different companies

- Converted all the values into lower case in order that the value count () method accurately calculate them. Figure 14 shows the code and output.

```
In [20]: #Converting values to lower case
         df['job_info_job_title'] = df['job_info_job_title'].str.lower()

         #Splitting job titles by '-'
         df['job_info_job_title'] = df['job_info_job_title'].astype(str).str.split('-').str[0]
         #Splitting job titles by 'ii'
         df['job_info_job_title'] = df['job_info_job_title'].astype(str).str.split('ii').str[0]
         #Splitting job titles by '/'
         df['job_info_job_title'] = df['job_info_job_title'].astype(str).str.split('/').str[0]
         #Removing leading and ending spaces
         df['job_info_job_title'] = df['job_info_job_title'].astype(str).str.strip()
         #Replacing "sr." values with "senior"
         df['job_info_job_title'] = df['job_info_job_title'].str.replace('sr.', 'senior')
         #Replacing "NaN", "NaT" and "nan" values with np.nan
         df['job_info_job_title'].replace(["NaN", 'NaT','nan'], np.nan, inplace = True)


         df['job_info_job_title'].value_counts(dropna=True)[:10]

Out[20]: software engineer               18582
         computer systems analyst        12054
         senior software engineer         5802
         software developer               4501
         programmer analyst               3763
         assistant professor              2869
         software development engineer    2766
         systems analyst                  2587
         senior programmer analyst        1884
         senior software developer        1625
         Name: job_info_job_title, dtype: int64
```

Figure 14: Converting values to lower case

6 Feature Selection

- Columns with more than 330000 non null values were displayed. In Figure 15 it can be seen that there are 19 columns with less than 12 percent missing values.

```
In [32]: #Leaving columns which have more than 330000 non-missing observations
         df = df.loc[:,df.count() >= 330000]
         df.info()

         <class 'pandas.core.frame.DataFrame'>
         Int64Index: 356168 entries, 0 to 374353
         Data columns (total 19 columns):
          #   Column                Non-Null Count   Dtype
         ---  ------                --------------   -----
          0   case_status           356168 non-null  object
          1   class_of_admission    335083 non-null  object
          2   country_of_citizenship 336896 non-null  object
          3   decision_date         356168 non-null  object
          4   employer_address_1    356131 non-null  object
          5   employer_city         356158 non-null  object
          6   employer_name         356160 non-null  object
          7   employer_postal_code  356135 non-null  object
          8   employer_state        356131 non-null  object
          9   job_info_work_city    356073 non-null  object
          10  job_info_work_state   356072 non-null  object
          11  pw_amount_9089        356168 non-null  float64
          12  pw_soc_code           355778 non-null  object
          13  pw_soc_title          353847 non-null  object
          14  pw_source_name_9089   354081 non-null  object
          15  pw_unit_of_pay_9089   354687 non-null  object
          16  casenumber            356168 non-null  object
          17  year                  356168 non-null  object
          18  remuneration          356168 non-null  category
         dtypes: category(1), float64(1), object(17)
         memory usage: 62.0+ MB
```

Figure 15: Displayed entities wih more than non null values

- In order to make it easier to read, state names in the data set were named with their abbreviations. Following Figure 16 and Figure 17 illustrates the same.

```
In [34]: #Assigning Labels to Case Status
df.loc[df.case_status == 'Certified', 'case_status'] = 1
df.loc[df.case_status == 'Denied', 'case_status'] = 0

#Filling missing values in "employer_state" column with mode
df['employer_state'] = df['employer_state'].fillna(df['employer_state'].mode()[0]);

#Mapping from state name to abbreviation
state_abbrevs = {
    'Alabama': 'AL',
    'Alaska': 'AK',
    'Arizona': 'AZ',
    'Arkansas': 'AR',
    'California': 'CA',
    'Colorado': 'CO',
    'Connecticut': 'CT',
    'Delaware': 'DE',
    'Florida': 'FL',
    'Georgia': 'GA',
    'Hawaii': 'HI',
    'Idaho': 'ID',
    'Illinois': 'IL',
    'Indiana': 'IN',
    'Iowa': 'IA',
    'Kansas': 'KS',
    'Kentucky': 'KY',
    'Louisiana': 'LA',
    'Maine': 'ME',
    'Maryland': 'MD',
    'Massachusetts': 'MA',
    'Michigan': 'MI',
    'Minnesota': 'MN',
    'Mississippi': 'MS',
    'Missouri': 'MO',
    'Montana': 'MT',
    'Nebraska': 'NE',
    'Nevada': 'NV',
```

Figure 16: Labels to states assigned

```
    'New Hampshire': 'NH',
    'New Jersey': 'NJ',
    'New Mexico': 'NM',
    'New York': 'NY',
    'North Carolina': 'NC',
    'North Dakota': 'ND',
    'Ohio': 'OH',
    'Oklahoma': 'OK',
    'Oregon': 'OR',
    'Pennsylvania': 'PA',
    'Rhode Island': 'RI',
    'South Carolina': 'SC',
    'South Dakota': 'SD',
    'Tennessee': 'TN',
    'Texas': 'TX',
    'Utah': 'UT',
    'Vermont': 'VT',
    'Virginia': 'VA',
    'Washington': 'WA',
    'West Virginia': 'WV',
    'Wisconsin': 'WI',
    'Wyoming': 'WY',
    'Northern Mariana Islands':'MP',
    'Palau': 'PW',
    'Puerto Rico': 'PR',
    'Virgin Islands': 'VI',
    'District of Columbia': 'DC'
}
```

Figure 17: ..contd Labels to states assigned

- Before running the models, feature variables were converted into categories. Figure 18 shows the converted categories.

```
In [38]: from sklearn.preprocessing import LabelEncoder
         categorical_variables = {}

         #Creating categories denoted by integers from column value
         for col in df.columns:
             cat_var_name = "cat_"+ col
             cat_var_name = LabelEncoder()
             cat_var_name.fit(df[col])
             df[col] = cat_var_name.transform(df[col])
             categorical_variables[col] = cat_var_name

         df.info()

         <class 'pandas.core.frame.DataFrame'>
         Int64Index: 356168 entries, 0 to 374353
         Data columns (total 10 columns):
          #   Column                Non-Null Count   Dtype
         ---  ------                --------------   -----
          0   case_status           356168 non-null  int64
          1   class_of_admission    356168 non-null  int32
          2   country_of_citizenship 356168 non-null  int32
          3   employer_city         356168 non-null  int32
          4   employer_name         356168 non-null  int32
          5   employer_state        356168 non-null  int32
          6   pw_soc_code           356168 non-null  int64
          7   pw_source_name_9089   356168 non-null  int32
          8   year                  356168 non-null  int32
          9   remuneration          356168 non-null  int32
         dtypes: int32(8), int64(2)
         memory usage: 29.0 MB
```

Figure 18: Converted feature variables into categories

7 Applying Machine Learning Models

- Random Forest model was applied which gave an accuracy of 93%. It was a best performing model. Figure 19 shows code for Random Forest Model.

```
In [41]: start = time.time()
         clf = RandomForestClassifier(min_samples_leaf=24)
         clf.fit(X, y)
         end = time.time()
         print ("Random Forest", end - start, clf.score(X,y))
         proba = clf.predict_proba(X)

         Random Forest 56.85677886009216 0.9390147346196177

In [42]: from sklearn import metrics

In [43]: print("Accuracy:",clf.score(X,y))

         Accuracy: 0.9390147346196177
```

Figure 19: Random Forest Model

- AdaBoost with Logistic Regression model was applied which gave an accuracy of 79%. Figure 20 shows the code for AdaBoost model.

```
In [45]: # evaluate adaboost algorithm with logistic regression weak learner for classification
         from numpy import mean
         from numpy import std
         from sklearn.datasets import make_classification
         from sklearn.model_selection import cross_val_score
         from sklearn.model_selection import RepeatedStratifiedKFold
         from sklearn.ensemble import AdaBoostClassifier
         from sklearn.linear_model import LogisticRegression
         # define dataset
         X, y = make_classification(n_samples=1000, n_features=20, n_informative=15, n_redundant=5, random_state=6)
         # define the model
         model = AdaBoostClassifier(base_estimator=LogisticRegression())
         # evaluate the model
         cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
         n_scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1, error_score='raise')
         # report performance
         print('Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))

         Accuracy: 0.794 (0.032)
```

Figure 20: AdaBoost with Logistic Regression

- Figure 21 shows the code to calculate MAE value for the model as part of evaluation.

```
In [46]:
# evaluate adaboost ensemble for regression
from numpy import mean
from numpy import std
from sklearn.datasets import make_regression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedKFold
from sklearn.ensemble import AdaBoostRegressor
# define dataset
X, y = make_regression(n_samples=1000, n_features=20, n_informative=15, noise=0.1, random_state=6)
# define the model
model = AdaBoostRegressor()
# evaluate the model
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
n_scores = cross_val_score(model, X, y, scoring='neg_mean_absolute_error', cv=cv, n_jobs=-1, error_score='raise')
# report performance
print('MAE: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))

MAE: -73.077 (4.120)
```

Figure 21: Calculating MAE for AdaBoost with Logistic Regression

- Multinomial Logistic Regression model was applied which gave an accuracy of 68%. Figure 22 shows the code for Multinomial Logistic Regression model.

```
In [47]: # evaluate multinomial logistic regression model
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.linear_model import LogisticRegression
# define dataset
X, y = make_classification(n_samples=1000, n_features=10, n_informative=5, n_redundant=5, n_classes=3, random_state=1)
# define the multinomial logistic regression model
model = LogisticRegression(multi_class='multinomial', solver='lbfgs')
# define the model evaluation procedure
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate the model and collect the scores
n_scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
# report the model performance
print('Mean Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))

Mean Accuracy: 0.681 (0.042)
```

Figure 22: Multinomial Logistic Regressiom

- Radius Neighbors classifier model was applied which gave an accuracy of 87%. Figure 23 shows the code for Radius Neighbors classifier model.

```
In [51]: # evaluate an radius neighbors classifier model on the dataset
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import RadiusNeighborsClassifier
# define dataset
x, y = make_classification(n_samples=1000, n_features=20, n_informative=15, n_redundant=5, random_state=1)
# define model
model = RadiusNeighborsClassifier()
# create pipeline
pipeline = Pipeline(steps=[('norm', MinMaxScaler()),('model',model)])
# define model evaluation method
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate model
scores = cross_val_score(pipeline, x, y, scoring='accuracy', cv=cv, n_jobs=-1)
# summarize result
print('Mean Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))

Mean Accuracy: 0.754 (0.042)
```

Figure 23: Radius Neighbors classifier model

- Figure 24 shows the code for Grid search radius for Radius Neighbors classifier model.

```
In [53]: # grid search radius for radius neighbors classifier
from numpy import arange
from sklearn.datasets import make_classification
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import RadiusNeighborsClassifier
# define dataset
X, y = make_classification(n_samples=1000, n_features=20, n_informative=15, n_redundant=5, random_state=1)
# define model
model = RadiusNeighborsClassifier()
# create pipeline
pipeline = Pipeline(steps=[('norm', MinMaxScaler()),('model',model)])
# define model evaluation method
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# define grid
grid = dict()
grid['model__radius'] = arange(0.8, 1.5, 0.01)
# define search
search = GridSearchCV(pipeline, grid, scoring='accuracy', cv=cv, n_jobs=-1)
# perform the search
results = search.fit(X, y)
# summarize
print('Mean Accuracy: %.3f' % results.best_score_)
print('Config: %s' % results.best_params_)

Mean Accuracy: 0.872
Config: {'model__radius': 0.8}
```

Figure 24: Grid Search radius for RN classifier

- XG Boost model was applied which gave an accuracy of 92%. Figure 25 shows the code for XG Boost model.

```
In [54]: # evaluate xgboost algorithm for classification
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from xgboost import XGBClassifier
# define dataset
X, y = make_classification(n_samples=1000, n_features=20, n_informative=15, n_redundant=5, random_state=7)
# define the model
model = XGBClassifier()
# evaluate the model
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
n_scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))

Accuracy: 0.925 (0.028)
```

Figure 25: XG Boost Model

- Figure 26 shows the code to calculate MAE value for the XG Boost model as part of evaluation.

```
In [56]:
# evaluate xgboost ensemble for regression
from numpy import mean
from numpy import std
from sklearn.datasets import make_regression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedKFold
from xgboost import XGBRegressor
# define dataset
X, y = make_regression(n_samples=1000, n_features=20, n_informative=15, noise=0.1, random_state=7)
# define the model
model = XGBRegressor()
# evaluate the model
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
n_scores = cross_val_score(model, X, y, scoring='neg_mean_absolute_error', cv=cv, n_jobs=-1, error_score='raise')
# report performance
print('MAE: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))

MAE: -76.447 (3.859)
```

Figure 26: Calculating MAE for XG Boost Model

References

Anaconda (2021). Installing anaconda.
URL: *https://docs.anaconda.com/anaconda/install/windows/*

learning Lounge, C. (2021). Download, setup install python on windows[2021].
URL: *https://medium.com/co-learning-lounge/how-to-download-install-python-on-windows-2021-44a707994013*