BCIS 5140 Artificial Intelligence in Business

H1B VISA PREDICTIONS

Project Final Report

Team 9

Done by:

- ➢ Aparna Rachoori
- ➢ Promodini Venkatakrishnan
- ➢ Sri Shravani Kondeti
- ➢ Ugochi Madumere
- ➢ Tevi Lawson

❖ Table of Contents

❖ Executive summary

With the intent to quench the increasing demand for certain technical professions, the US government has initiated since 1990 a working based visa for foreign graduate students in the US termed as H1B visa, which at its core grants authorization to the latter to work for a given employer within a specific timeframe. Our dataset is a secondhand collection of H1B visa applications by employers comprising 528000 plus instances and 27 attributes dated from October 2016 to June 2017 extracted from Kaggle.com. We outlined the challenge as a classification problem and applied Multinomial Logistic Regression, SGD Classifier, Gradient Boosting, Decision tree and Random forest with the purpose of predicting the case status of the application.

This report is aimed at providing a calibration point for various stakeholders notably for the US government, potential employers and prospective H1B visa holders as such:

- Intent to prescribe case status of H1B petition and guidance to decrease the likelihood of rejection
- In the rise of globalization and the security threat faced by the US, the department of labor could tackle the issue of shortage in qualified cybersecurity workers by prioritizing graduate H1B visa applicants in the field while relying on our extensive study in determining the sampled population growth in the field and associative factors impacting it.
- Our study is also aimed at providing a baseline salary for both prospective employers and employees in the negotiation process and employment clauses.
- This report will assist the IRS in forecasting the revenue from taxes and fees to be generated from the undertaking of this governmental program.

The data cleaning and exploration phase consisted of transforming and normalizing our dataset to create uniformity and ease of modeling. It entails the riddance of null values irrelevant for our models, and a conversion of variables to suit the algorithm of interest. Upon completion of this phase, a simulation process was conducted to explore the possible outcomes, while isolating features that could advance our study.

For our intent and purpose, 6 distinct algorithms were performed to optimize our model design notably Classifier, logistic regression, SDG Classifier, Decision Trees, Random Forest and Gradient Boosting were compared to deduce the best model amongst each other that fit our dataset. The modeling phase is a work of art incorporating experimental design, randomization, accuracy assessment given rise to a result summary and a model selection. The model with the highest accuracy has been identified as random forest with a precision of 83.8% and an accuracy to the grading of 88%.

❖ Introduction

The H-1B program is for employers hiring nonimmigrants as workers in various occupations considering merit and ability. A specialized profession is one that needs the application with highly specialized expertise and the accomplishment of no less than a bachelor's degree. The purpose of in accordance with H-1B stipulations is to provide assistance to employers who cannot contrarily find the required business skills from the U.S. labor force by allowing the provisional employment of skilled individuals who are not otherwise allowed to go to work throughout the united states.

This visa necessitates the aspirant to have an employment offer in the US prior to apply to the US immigration authorities (USCIS). USCIS permits 85,000 H-1B visa applications each year, although the number of the candidates considerably surpass that number. Procedure for selection is alleged to be based on a lottery, therefore the attributes of the applicants alter the result is ambiguous. We assume that this forecast algorithm would be a valuable resource both for the forthcoming H-1B visa applicants and the employers that are sponsoring them.

❖ Background Study

The following studies have been done on the H1B dataset by different scholars:

H1B Visa prediction done by dpandya1 in 2018 considered data from 2011 – 2016 and merged certified withdrawn, withdrawn, and denied as "Denied". We analyzed for all 4 classes to leverage on the granularity of information the user may need to make decisions.

COMPREHENSIVE H1-B VISA ANALYSIS by SAITEJA NAKKA2 conducted only Exploratory Data Analysis to gain insight on the descriptive statistics using Python while we explored, preprocessed, and modeled the data for prediction.

H1B Disclosure Dataset - Predicting the Case Status by Charmi Narendrakumar3 Trivedi and Pearl Firoz Mehta Used WEKA software to run data mining algorithms to understand the relationship between attributes and the target variable and ascertain the best model.

H1B-Case-Status-Prediction by kashyapbhuva3 used only AUC as their evaluation metric and merged the classes to a binary target variable. They did not sample the dataset but rather relied on AUC for accurate predictions as it is suitable for highly imbalanced datasets. They focused more on hyperparameter tuning to optimize the result of their models and its predictability. On the contrary we applied all the classification metrics including AUC and mean square error. We also performed stratified sampling after realizing the results of the imbalanced dataset was unrealistic regardless of the AUC score.

[1] https://www.kaggle.com/dpandya18/h1b-visa-status-prediction

[2] https://www.kaggle.com/tejaeduc/comprehensive-h1-b-visa-analysis

[3] https://github.com/kashyapbhuva/H1B-Case-Status-Prediction_Nov-2019/blob/master/ML%20Project%20v2%5B1144%5D.pptx

VisaMine by the theartist0074 merged Certified and Certified withdrawn as "Certified", leaving them with a Binary target variable, Certified and Denied. "Withdrawn" was excluded from the analysis. We captured all the classes

❖ Metadata

Information contained in the dataset comprises 27 attributes and 528,134 instances. Detailed description is referred to in the following table:

Name of the Attribute	Description
CASE_SUBMITTED_DAY	Date and time the application was filed.
CASE_SUBMITTED_MONTH	Month, the application was filed
CASE_SUBMITTED_YEAR	Year of the application filed
DECISION_DAY	Date on which the decision was documented by the Chicago National Processing Center.
DECISION_MONTH	Month in which the decision was documented by the Chicago National Processing Center.
DECISION_YEAR	Year in which the decision was documented by the Chicago National Processing Center.
VISA_CLASS	Refers to the type of interim application filed to be processed. 1- H1B; 2 - E-3 Australian; 3 – H-1B1 Chile; 4 - H-1B1 Singapore.
EMPLOYER_NAME	Name of the employer filing LCA
EMPLOYER_STATE	State of the Employer
EMPLOYER_COUNTRY	Country of the Employer
SOC_NAME	Name of the Job Profile
NAICS_CODE	Business code linked to the employer asking for perpetual workforce condition, as categorized by the North American Industry Classification System (NAICS)
TOTAL_WORKERS	Count of immigrants requested by the Employers
FULL_TIME_POSITION	Either permanent job or not a permanent job

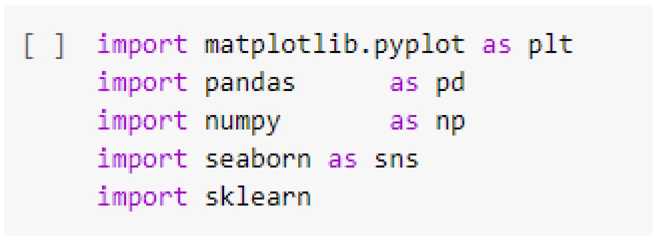
4 https://github.com/theartist007/VisaMine/blob/master/Report.pdf

PREVAILING_WAGE	Wage of an employee requested by employer for temporary labor condition
PW_UNIT_OF_PAY	Wages per “Hourly (HR),” “Bi-weekly (BI),” “Weekly (WK),” “Monthly (MTH),” and “Yearly (YR)”
PW_SOURCE	Wage source and values include "OES", "CBA", "DBA", "SCA" or "Other"
PW_SOURCE_YEAR	Year the Wage source was released
PW_SOURCE_OTHER	Wage source by another consultant
WAGE_RATE_OF_PAY_FROM	Employer’s proposed wage rate
WAGE_RATE_OF_PAY_TO	Maximum proposed wage rate
WAGE_UNIT_OF_PAY	Unit of pay, values include “Hour", "Week", "Bi- Weekly", "Month", "Year"
H-1B_DEPENDENT	Employee might or might not have a dependent
WILLFUL_VIOLATOR	Firm was previously discovered to be Willful Violator or not
WORKSITE_STATE	State data of Employee’s intended work area of employment.
WORKSITE_POSTAL_CODE	Zip code data of Employee’s intended work area of employment.
CASE_STATUS	Status of visa application with 4 values “Certified,” “Certified-Withdrawn,” Denied,” and “Withdrawn

❖ Data import staging

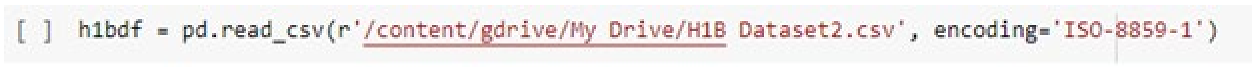
Data Importing:

A Python library is a set of functions and techniques which permits you to perform numerous actions. To execute the task, we will have to install, download, and import the libraries as per the requirement.



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

First step is importing the data into a data frame using the below function. This is a method in pandas to load a csv file.



```
[ ]  h1bdf = pd.read_csv(r'/content/gdrive/My Drive/H1B Dataset2.csv', encoding='ISO-8859-1')
```

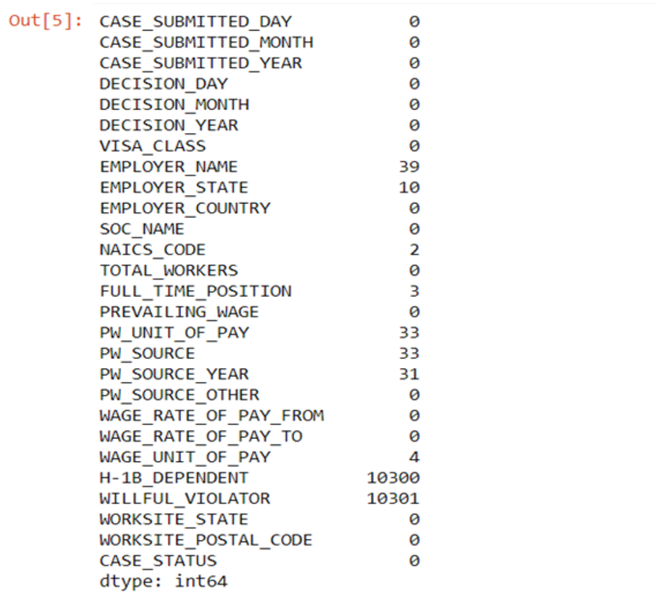
It is extremely evident that a data collection through its authentic shape may include missing values, inconsistent records. Such information will influence the quality of output. Consequently, to enhance the efficient performance of the machine learning models which are essential to train and test the methodology to clean the reliable data. Unwanted fields were also removed.

❖ Exploratory data Analysis and visualization

Missing Values

Certain rows in the data set have few missing values. The missing fields are either filled with false values or the entire row is unused. Below is the list of independent variables with missing values:

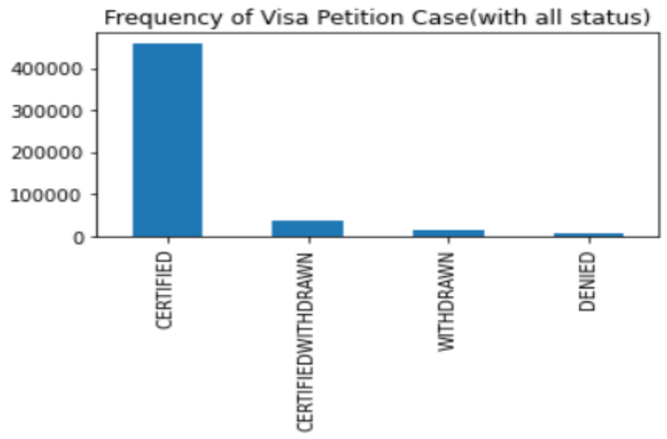
Employer state, Employer Name, Pw_unit_of_pay, pw_source, pw_source_other, H1B_Dependent, Willful_voilator, and Full_time_Position



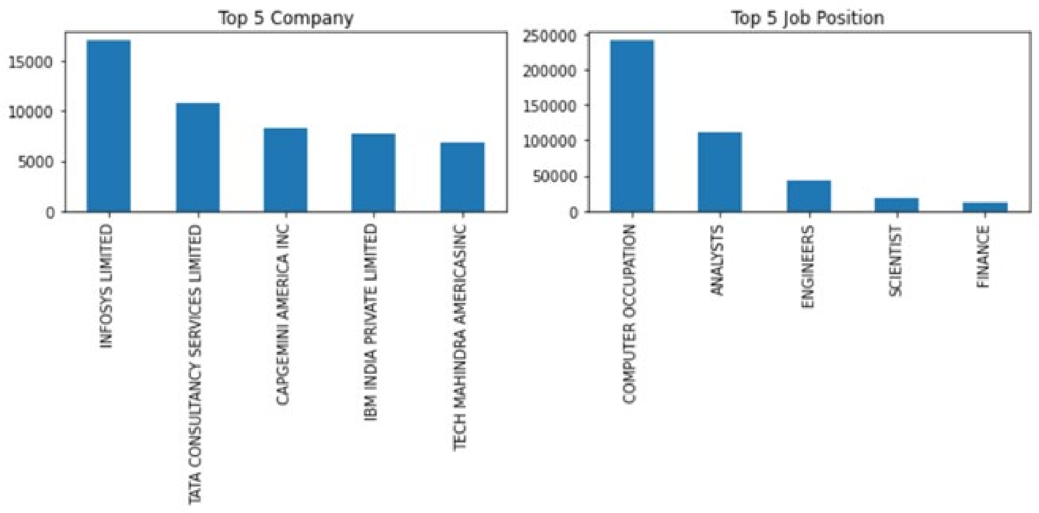
```
Out[5]: CASE_SUBMITTED_DAY          0
        CASE_SUBMITTED_MONTH        0
        CASE_SUBMITTED_YEAR         0
        DECISION_DAY                0
        DECISION_MONTH              0
        DECISION_YEAR               0
        VISA_CLASS                  0
        EMPLOYER_NAME              39
        EMPLOYER_STATE             10
        EMPLOYER_COUNTRY            0
        SOC_NAME                    0
        NAICS_CODE                  2
        TOTAL_WORKERS               0
        FULL_TIME_POSITION          3
        PREVAILING_WAGE             0
        PW_UNIT_OF_PAY             33
        PW_SOURCE                  33
        PW_SOURCE_YEAR             31
        PW_SOURCE_OTHER             0
        WAGE_RATE_OF_PAY_FROM       0
        WAGE_RATE_OF_PAY_TO         0
        WAGE_UNIT_OF_PAY            4
        H-1B_DEPENDENT          10300
        WILLFUL_VIOLATOR        10301
        WORKSITE_STATE              0
        WORKSITE_POSTAL_CODE        0
        CASE_STATUS                 0
        dtype: int64
```

Variable pattern exploration within the data:

A frequency of outcome for the total number of appeals filed over the year is represented in a bar graph. The chart displays the number of certified visa status is higher than the other three case status classes.

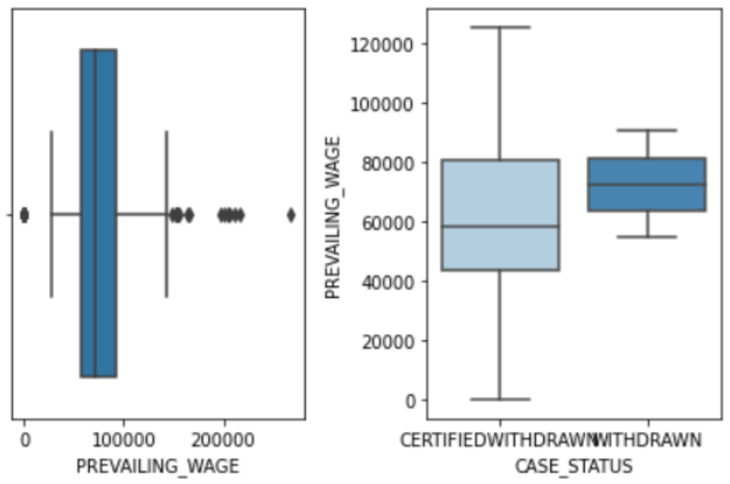


The below bar graph depicts the top five companies that filed the higher number of visa petition cases in different sectors and the top five job profiles for which nonimmigrants are recruited. Hence, the topmost company is **Infosys Limited** and the top occupation is **Computer Analysts**.



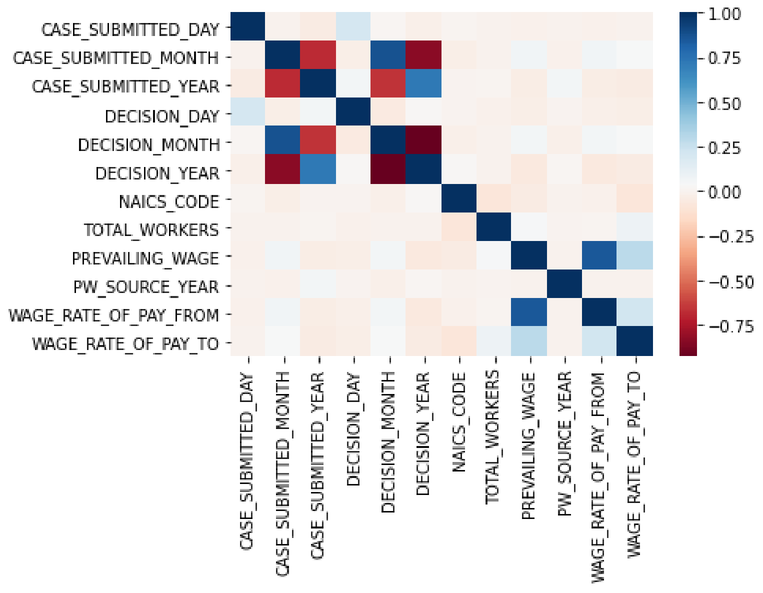
It is observed that there is a difference between the minimum(30K) and maximum(80K) values and also, it is clear that we have multiple outliers in the dataset. From the below box plot of **Prevailing Wage vs Case Status**, we can clearly depict the existence of outliers. And the other box plot depicts the relation between case status and the prevailing wage. Which shows, the certified withdrawn petitions wage range between 45K to 80K, for withdrawn between 65K to 80K.

```
<Figure size 720x576 with 0 Axes>
```



Correlation Heat Map:

The below heat map gives us the graphical display of data in which values are represented in different colors to reflect how the variables are correlated with each other. The dark blue mark indicates high collinearity between variables. Here the following variables are highly correlated: Decision Year and Case Submitted Year, Decision Month and Case Submitted Month, Prevailing Wage and Wage_Rate_of_Pay_From.

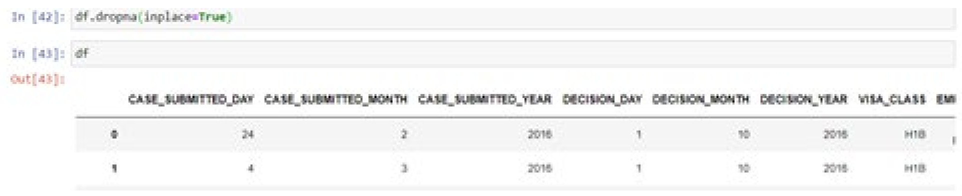


❖ Data cleaning and Preparation

After analyzing the dataset, the following data preparation is conducted.

Removing Null Values
The null values in the dataset are dropped for the following reasons. The independent variables willful_voilator, H1B_Dependent, Full_Time_position, Employer state, Employer Name, Pw_unit_of_pay, pw_source,pw_source_other have the missing values. The independent variable such as Employer state, Employer name cannot have duplicate values and there is only 0.1 percent of the values are missing.

The other independent variables such as willful_voilator, H1B_Dependent, Full_Time_position have yes and no values, so they cannot be biased. In addition to it, only 3 percent are missing. While the other independent variables are negligible.

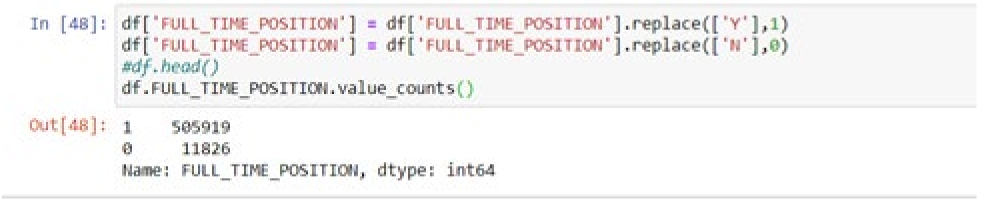


```
In [42]: df.dropna(inplace=True)

In [43]: df
Out[43]:
```

	CASE_SUBMITTED_DAY	CASE_SUBMITTED_MONTH	CASE_SUBMITTED_YEAR	DECISION_DAY	DECISION_MONTH	DECISION_YEAR	VISA_CLASS	EMI
0	24	2	2016	1	10	2016	H1B	
1	4	3	2016	1	10	2016	H1B	

Binary Coding of "Yes" and "No"

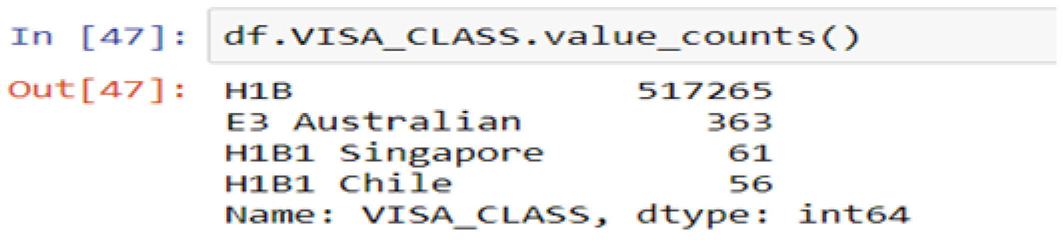
The independent variables willful_voilator, H1B_Dependent, Full_Time_position have 'yes' and 'No' values. To perform analysis, the categorical variable is changed to numerical.



```
In [48]: df['FULL_TIME_POSITION'] = df['FULL_TIME_POSITION'].replace(['Y'],1)
         df['FULL_TIME_POSITION'] = df['FULL_TIME_POSITION'].replace(['N'],0)
         #df.head()
         df.FULL_TIME_POSITION.value_counts()

Out[48]: 1    505919
         0     11826
         Name: FULL_TIME_POSITION, dtype: int64
```

Eliminating other country Visa instances

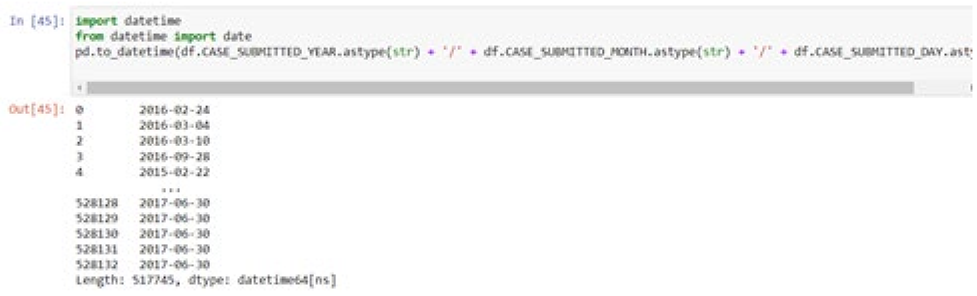
As the analysis is performed only on the United States H1B visa, the other country visa is eliminated from the process.



```
In [47]: df.VISA_CLASS.value_counts()

Out[47]: H1B                 517265
         E3 Australian          363
         H1B1 Singapore          61
         H1B1 Chile              56
         Name: VISA_CLASS, dtype: int64
```

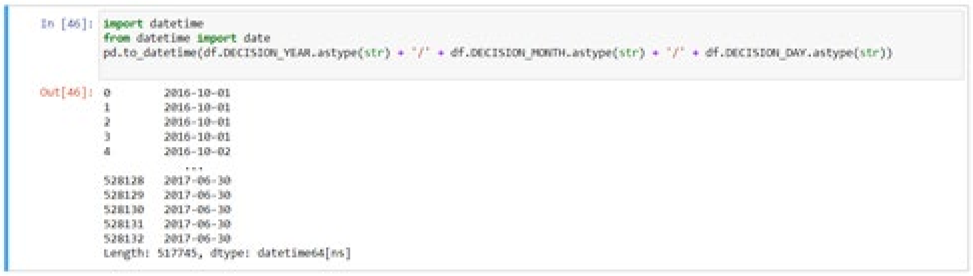
Combining the values to a single column i.e. Timestamp Format

The Independent variable Case_submitted_day, Case_submitted_month, Case_submitted_year indicates on which day, month, the year the case gets submitted to the USCIS. So, it is incorporated into a single column. The Independent variable Decision_Day, Decision_month, Decison_Year indicates on which day, month, the year the case gets decided by the USCIS. So, it is incorporated into a single column.



```
In [45]: import datetime
         from datetime import date
         pd.to_datetime(df.CASE_SUBMITTED_YEAR.astype(str) + '/' + df.CASE_SUBMITTED_MONTH.astype(str) + '/' + df.CASE_SUBMITTED_DAY.ast

Out[45]: 0         2016-02-24
         1         2016-03-04
         2         2016-03-10
         3         2016-09-28
         4         2015-02-22
                      ...
         528128    2017-06-30
         528129    2017-06-30
         528130    2017-06-30
         528131    2017-06-30
         528132    2017-06-30
         Length: 517745, dtype: datetime64[ns]
```

Fig: Case Submission Time Stamp

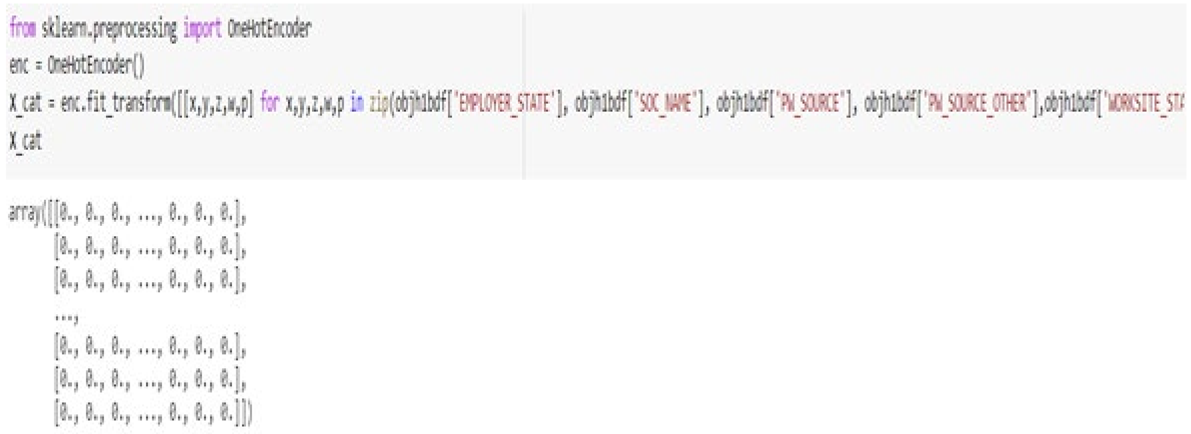


```
In [46]: import datetime
         from datetime import date
         pd.to_datetime(df.DECISION_YEAR.astype(str) + '/' + df.DECISION_MONTH.astype(str) + '/' + df.DECISION_DAY.astype(str))

Out[46]: 0         2016-10-01
         1         2016-10-01
         2         2016-10-01
         3         2016-10-01
         4         2016-10-02
                      ...
         528128    2017-06-30
         528129    2017-06-30
         528130    2017-06-30
         528131    2017-06-30
         528132    2017-06-30
         Length: 517745, dtype: datetime64[ns]
```

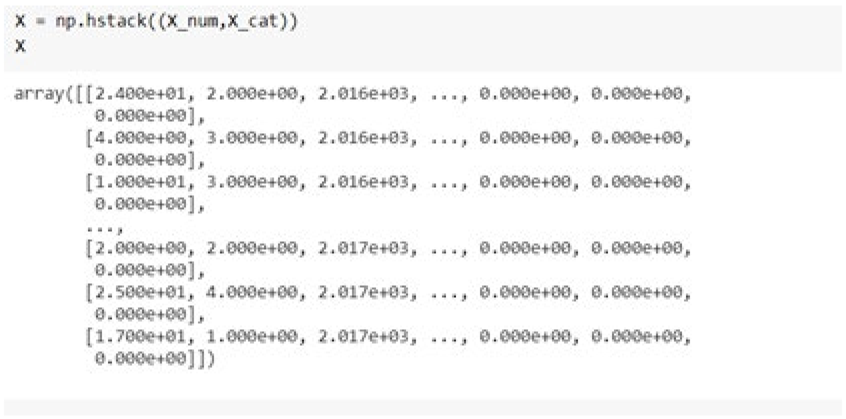
Fig: Case Decision Time Stamp

Converting Categorical variable into numerical

The following independent variables have categorical values and hence they are changed into numerical variables. To perform this the independent variables are separated, and one-hot encoding is performed. Later they are combined with numerical data.



```
from sklearn.preprocessing import OneHotEncoder
enc = OneHotEncoder()
X_cat = enc.fit_transform([[x,y,z,w,p] for x,y,z,w,p in zip(objh1bdf['EMPLOYER_STATE'], objh1bdf['SOC_NAME'], objh1bdf['PW_SOURCE'], objh1bdf['PW_SOURCE_OTHER'],objh1bdf['WORKSITE_STA
X_cat

array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

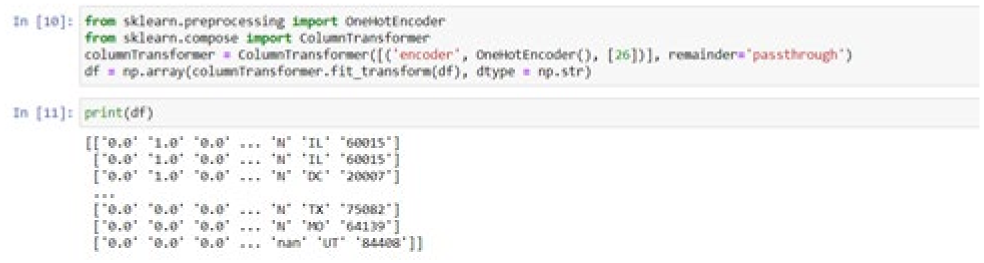


```
X = np.hstack((X_num,X_cat))
X
```

```
array([[2.400e+01, 2.000e+00, 2.016e+03, ..., 0.000e+00, 0.000e+00,
        0.000e+00],
       [4.000e+00, 3.000e+00, 2.016e+03, ..., 0.000e+00, 0.000e+00,
        0.000e+00],
       [1.000e+01, 3.000e+00, 2.016e+03, ..., 0.000e+00, 0.000e+00,
        0.000e+00],
       ...,
       [2.000e+00, 2.000e+00, 2.017e+03, ..., 0.000e+00, 0.000e+00,
        0.000e+00],
       [2.500e+01, 4.000e+00, 2.017e+03, ..., 0.000e+00, 0.000e+00,
        0.000e+00],
       [1.700e+01, 1.000e+00, 2.017e+03, ..., 0.000e+00, 0.000e+00,
        0.000e+00]])
```

One Hot Encoding to the Label

In our dataset, Case_Status is considered as the Dependent variable. There are four different values in the target variable, and they are Certified, Certified Withdrawn, Withdrawn, Denied. To run a different model for analysis, one hot encoding is performed.



```
In [10]: from sklearn.preprocessing import OneHotEncoder
         from sklearn.compose import ColumnTransformer
         columnTransformer = ColumnTransformer([('encoder', OneHotEncoder(), [26])], remainder='passthrough')
         df = np.array(columnTransformer.fit_transform(df), dtype = np.str)

In [11]: print(df)

         [['0.0' '1.0' '0.0' ... 'N' 'IL' '60015']
          ['0.0' '1.0' '0.0' ... 'N' 'IL' '60015']
          ['0.0' '1.0' '0.0' ... 'N' 'DC' '20007']
          ...
          ['0.0' '0.0' '0.0' ... 'N' 'TX' '75082']
          ['0.0' '0.0' '0.0' ... 'N' 'MO' '64139']
          ['0.0' '0.0' '0.0' ... 'nan' 'UT' '84408']]
```

❖ Model sampling, analysis, and comparison

Initial Modelling (Pre-Re Sampling)

The dataset is passed into separate arrays i.e. feature array X and label array y. The feature array is scaled before the data i.e. both feature array X and label y are split into train (80%) and test datasets for the ease of the modelling (15%). A part of the training data is held back i.e validation data (20%) making training data (65%) that can be used to estimate the model while doing the hypermeter tuning.

Training Dataset: The model is initially trained on a training dataset using a supervising learning method and output vector that can be compared with the original label. The model parameters are adjusted based on the results. The model fitting with training data consists of variable selection and parameter estimation.

Validation Dataset: The fitted model is now used to predict the output for the held back samples from the training dataset. It gives the unbiased evaluation of the model while tuning the model hyperparameters. If the model performs well for training samples but not with validation samples, it is called overfitting.

Test Dataset : It can also be called a hold-out dataset as the samples in this dataset are separated right before the model training and any of these samples are never used in training i.e. totally new set samples to test our model.

Scaling: Min-Max scaling is preferred in our context because standard scaler does not seem to be apt for the dataset that has outliers. Our dataset has outliers in terms of Prevailing Wage of the Employee. These outliers denote the high wages and are more likely to get an H1B application approved. These high wage employees (instances) are important for our analysis.

Logistic Regression (Pre- Re Sampling)

Logistic regression is a statistical model that basically uses a logistic (sigmoid) function to model a binary label and classify into two classes based on the probability of certain classes. Logistic Regression is used when the label is categorical and can be of three types: Binary Logistic Regression, Multinomial Logistic Regression, and Ordinary Logistic Regression.

Binary Logistic Regression can be used when the label has two classes. If the label has multiple classes one of the other two can be chosen. If the label has ordered classes, then Multinomial Logistic Regression is performed, and Ordinary Logistic Regression can be preferred when the label has multiple ordered classes.

The general equation is

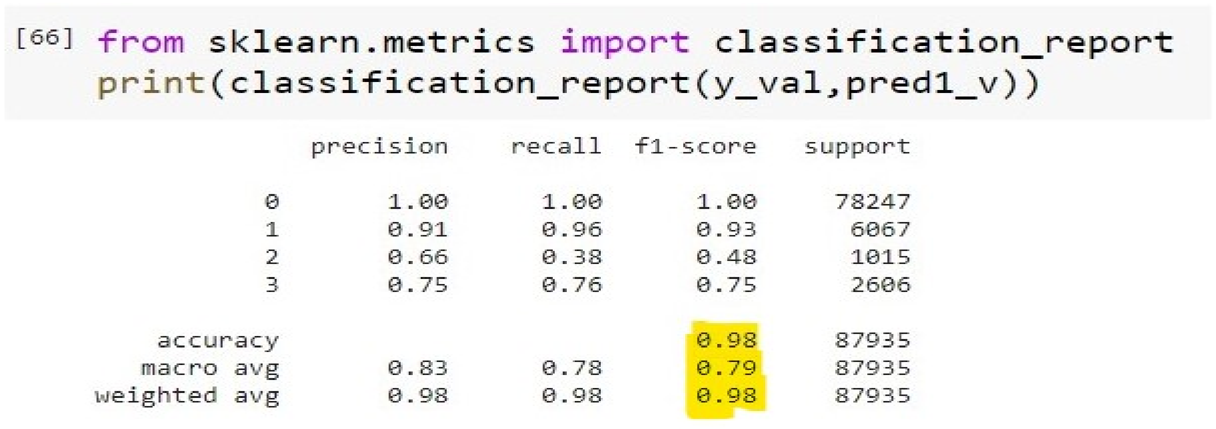
$$P = \frac{1}{1+e^{-(\beta_0+\beta_1 X_1+\beta_2 X_2+\ldots\beta_n X_n)}} = \frac{1}{1+e^{-(\beta_0+\sum \beta_i X_i)}}.$$

$$\ell = \log_b \frac{p}{1-p} = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

Multinomial logistic regression (MLR) predicts the probabilities of Labels which are the combinations of feature variables and uses maximum likelihood method instead of least squares method in model fitting. However, there are certain assumptions to proceed with MLR.

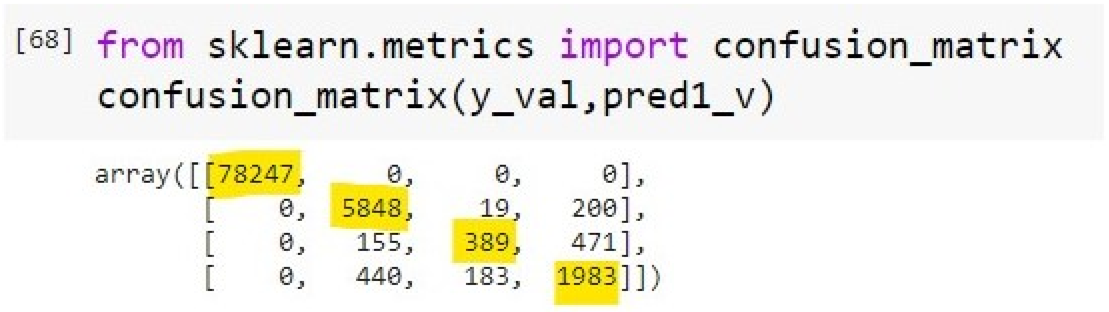
- Each feature variable should have a single value for each instance.
- Feature variables are not necessarily completely independent; low collinearity can be assumed.
- The odds of preferring one class over other should be there regardless of irrelevant alternatives.

MLR fits best for our problem statement of classifying the case status of an H1B application into 4 different classes viz. CERTIFIED, CERTIFIED WITHDRAWN, WITHDRAWN and DENIED. MLR is performed on the training dataset and the classification report and the confusion matrix of the model are shown below.



```
[66] from sklearn.metrics import classification_report
     print(classification_report(y_val,pred1_v))

              precision    recall  f1-score   support

           0       1.00      1.00      1.00     78247
           1       0.91      0.96      0.93      6067
           2       0.66      0.38      0.48      1015
           3       0.75      0.76      0.75      2606

    accuracy                           0.98     87935
   macro avg       0.83      0.78      0.79     87935
weighted avg       0.98      0.98      0.98     87935
```

Fig: Classification Report (Pre-Re Sampling)



```
[68] from sklearn.metrics import confusion_matrix
     confusion_matrix(y_val,pred1_v)

array([[78247,     0,     0,     0],
       [    0,  5848,    19,   200],
       [    0,   155,   389,   471],
       [    0,   440,   183,  1983]])
```

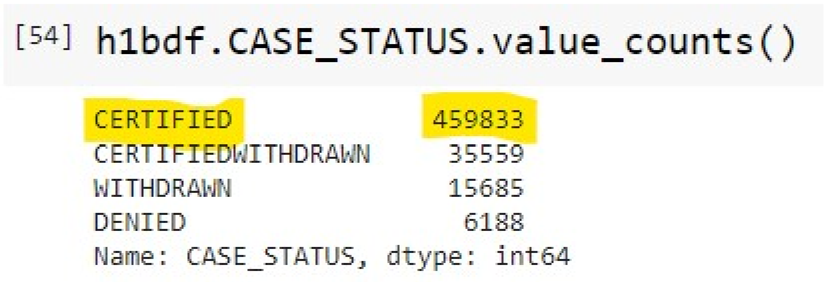
Fig: Confusion Matrix (Pre-Re Sampling)

If we observe the accuracy, it is 98% for weighted average accuracy. As our label has 4 different classes, weighted average accuracy is considered. Weighted Average aggregates the contributions of all the classes to compute the average accuracy. The Confusion Matrix gives the values of True Positive, True Negative, False Positive and False Negative values for all the 4 classes. The diagonal elements in the confusion matrix depict the correctly classified H1B application into 4 different classes. The accuracy of the model is pretty good but if we observe number of applications to be in certified class i.e. if we observe the first row and first column of the confusion matrix, the first element that depicts the certified H1B applications is the only non-null value indicating that none of the applications wrongly classified. This fact sounds too ideal to be true. This means that our model is basically error-free for the CERTIFIED class of H1B applications that indicates the biasedness of our model for CERTIFIED class of H1B applications. Hence, we went back to analyzing the data again.

Our second phase of data analysis was solely focused on finding the factor for biasedness of our model. We re-analyzed the dataset and observed that the label is imbalanced which implies the dataset is imbalanced. Our model was working fine and fit well to the dataset, but the bias is caused by the imbalance label of the dataset.

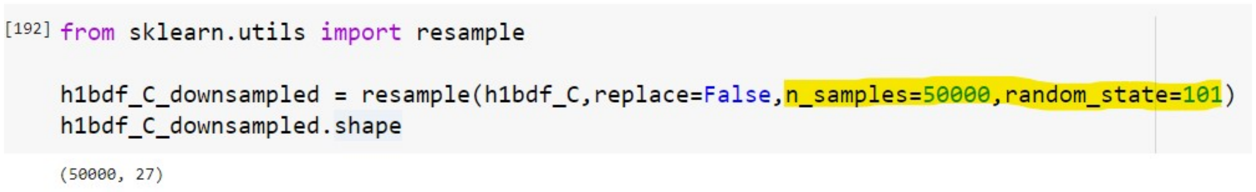
Data Re-Analysis

The number of instances of certified class are 459833 i.e. almost 89% of the total instances. The dataset has a major imbalance among the classes and majority class being the dominant. This is the reason for biases in our model. We have to do the down-sampling of the majority class in order to reduce the balance of the number of instances in all the classes in turn reducing the bias in the model. We resampled the majority class "CERTIFIED" to balance the 4 classes and extracted 50,000 i.e around 11% of samples for further processing. This might cause the loss of information but this is required to achieve a balanced dataset and unbiased model for H1B Visa Prediction. We considered the Min-Max scaler to scale the feature vector as the range of values vary. Machine learning algorithms tend to ignore the values of the lower range which might cause the loss of information. The data is split as in the initial modelling. Random State 101 is mentioned throughout to maintain the uniformity in the state.



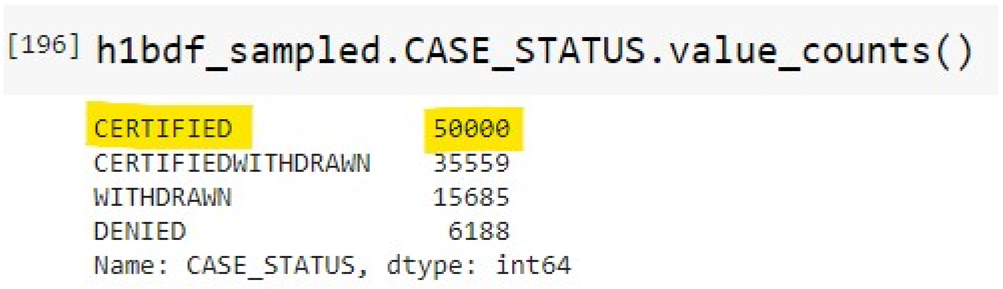
```
[54] h1bdf.CASE_STATUS.value_counts()

CERTIFIED              459833
CERTIFIEDWITHDRAWN      35559
WITHDRAWN               15685
DENIED                   6188
Name: CASE_STATUS, dtype: int64
```

Fig: Label Instances per class (Pre-Down Sampling)



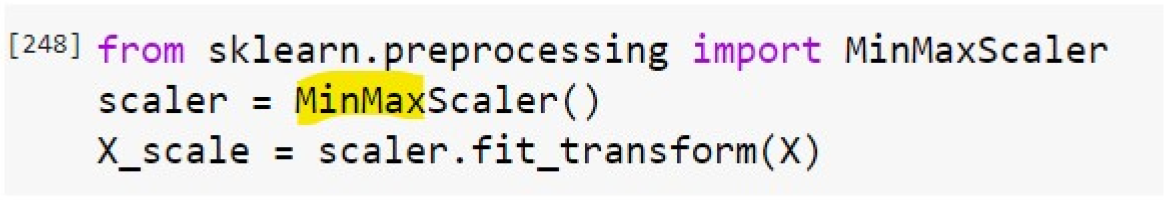
```
[192] from sklearn.utils import resample

      h1bdf_C_downsampled = resample(h1bdf_C,replace=False,n_samples=50000,random_state=101)
      h1bdf_C_downsampled.shape

      (50000, 27)
```

Fig: Re-Sampling the majority class



```
[196] h1bdf_sampled.CASE_STATUS.value_counts()

CERTIFIED              50000
CERTIFIEDWITHDRAWN     35559
WITHDRAWN              15685
DENIED                  6188
Name: CASE_STATUS, dtype: int64
```

Fig: Label instances per class (Post-Re Sampling)

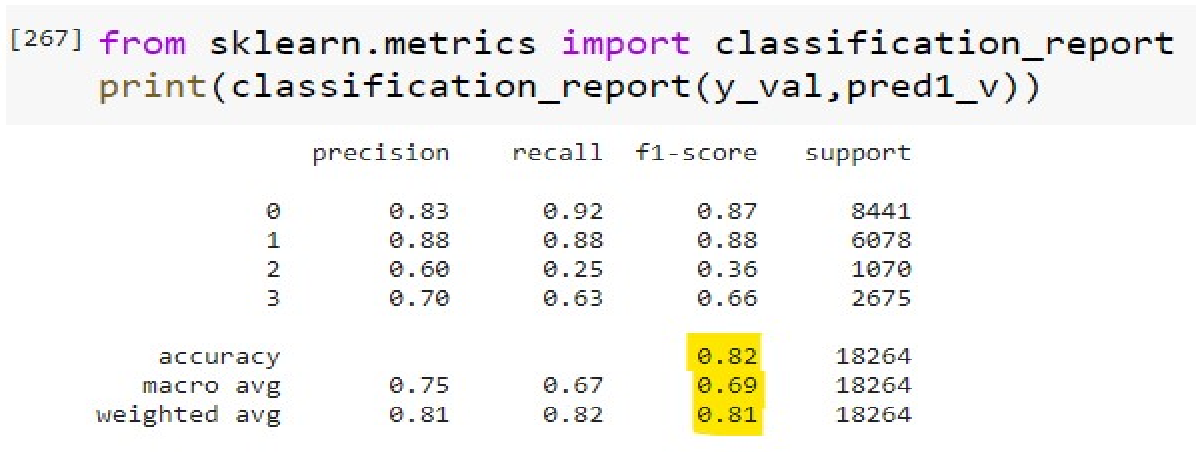


```
[248] from sklearn.preprocessing import MinMaxScaler
      scaler = MinMaxScaler()
      X_scale = scaler.fit_transform(X)
```

Fig: Min-Max Scaling

Multinomial Logistic Regression (Post-Re Sampling)

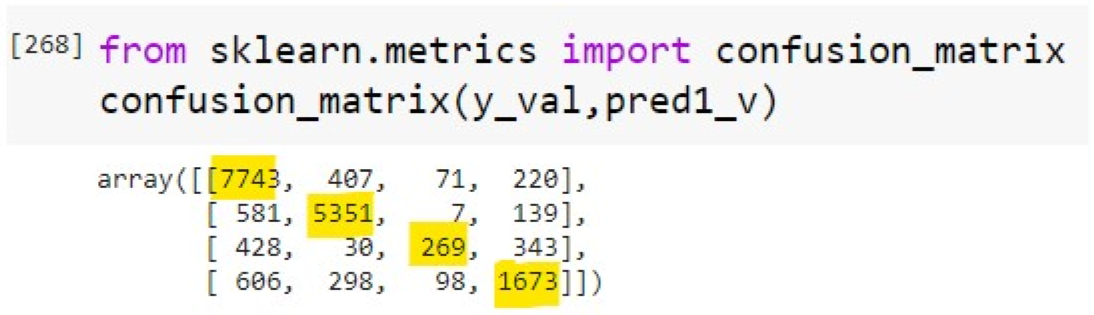
Multinomial Logistic Regression is again performed on the sampled training dataset and classification report, confusion matrix and the predicted probabilities are shown below. If we observe, the accuracy post re-sampling is 81% and went down by 17% compared to the accuracy pre re-sampling. The confusion matrix has all non-null elements in proportion indicating that there exist TF, TN, FP, FN for all the classes. The model is no more biased towards the CERTIFIED class of the label. The number of correctly classified CERTIFIED H1B applications are 51% of whole. The probability in the predicted probabilities array explains the probability with which an H1B application gets classified into one of the 4 classes of the Label. Performance measures viz. accuracy, F1 score, Area under the curve and MSE are calculated and shown below.



```
[267] from sklearn.metrics import classification_report
      print(classification_report(y_val,pred1_v))
```

```
              precision    recall  f1-score   support

           0       0.83      0.92      0.87      8441
           1       0.88      0.88      0.88      6078
           2       0.60      0.25      0.36      1070
           3       0.70      0.63      0.66      2675

    accuracy                           0.82     18264
   macro avg       0.75      0.67      0.69     18264
weighted avg       0.81      0.82      0.81     18264
```

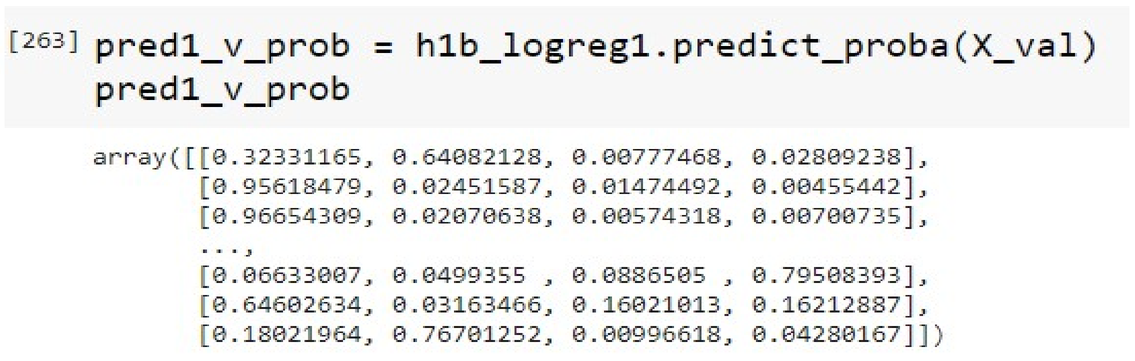
Fig: Classification Report (Post-Re Sampling)



```
[268] from sklearn.metrics import confusion_matrix
      confusion_matrix(y_val,pred1_v)
```

```
array([[7743,  407,   71,  220],
       [ 581, 5351,    7,  139],
       [ 428,   30,  269,  343],
       [ 606,  298,   98, 1673]])
```

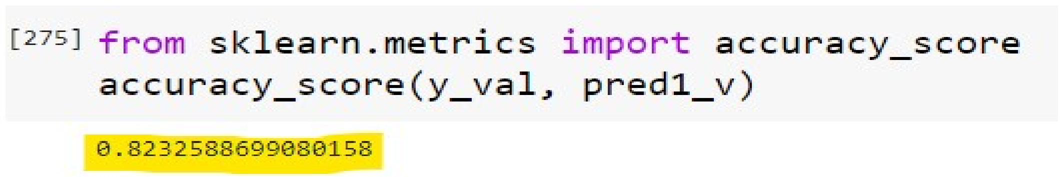
Fig: Confusion Matrix (Post-Re Sampling)



```
[263] pred1_v_prob = h1b_logreg1.predict_proba(X_val)
      pred1_v_prob
```

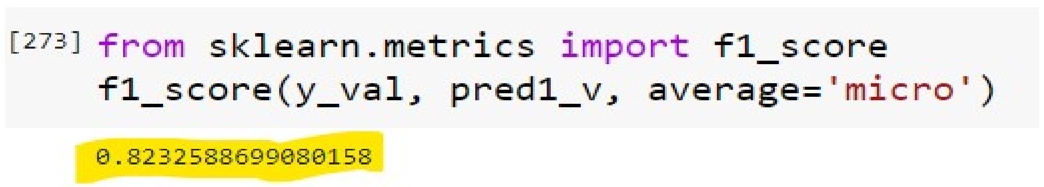
```
array([[0.32331165, 0.64082128, 0.00777468, 0.02809238],
       [0.95618479, 0.02451587, 0.01474492, 0.00455442],
       [0.96654309, 0.02070638, 0.00574318, 0.00700735],
       ...,
       [0.06633007, 0.0499355 , 0.0886505 , 0.79508393],
       [0.64602634, 0.03163466, 0.16021013, 0.16212887],
       [0.18021964, 0.76701252, 0.00996618, 0.04280167]])
```

Fig: Predicted Probabilities (Post-Re Sampling)



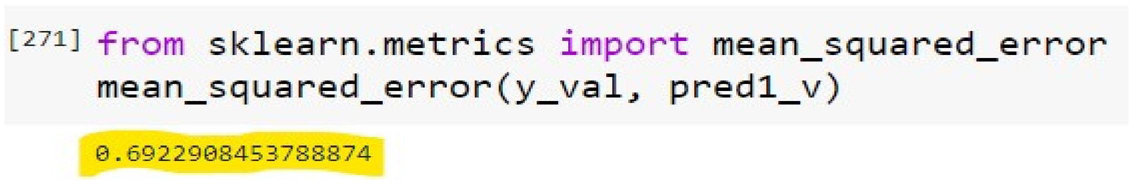
```
[275] from sklearn.metrics import accuracy_score
      accuracy_score(y_val, pred1_v)

0.8232588699080158
```

Fig: Accuracy (MLR)



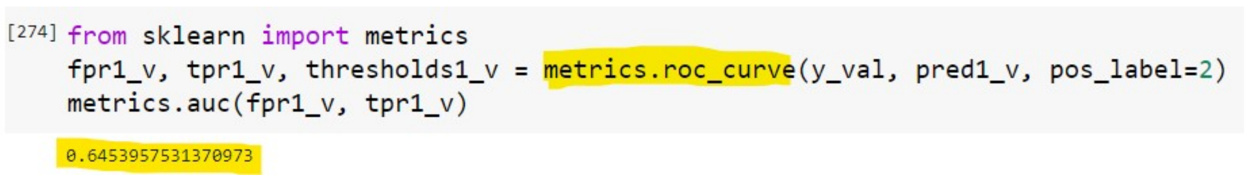
```
[273] from sklearn.metrics import f1_score
      f1_score(y_val, pred1_v, average='micro')

0.8232588699080158
```

Fig: F1 score (MLR)



```
[271] from sklearn.metrics import mean_squared_error
      mean_squared_error(y_val, pred1_v)

0.6922908453788874
```

Fig: MSE (MLR)

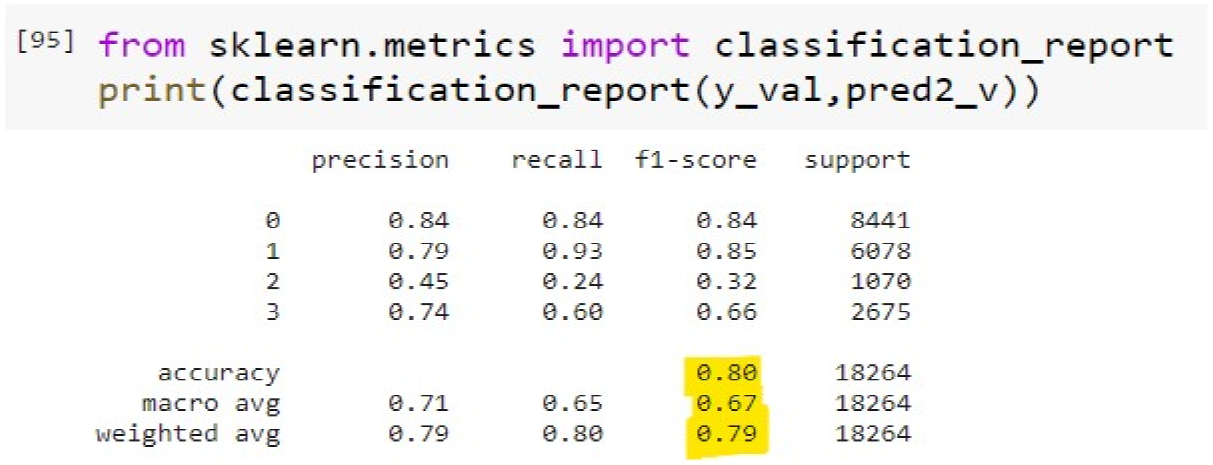


```
[274] from sklearn import metrics
      fpr1_v, tpr1_v, thresholds1_v = metrics.roc_curve(y_val, pred1_v, pos_label=2)
      metrics.auc(fpr1_v, tpr1_v)

0.6453957531370973
```

Fig: AUC (MLR)

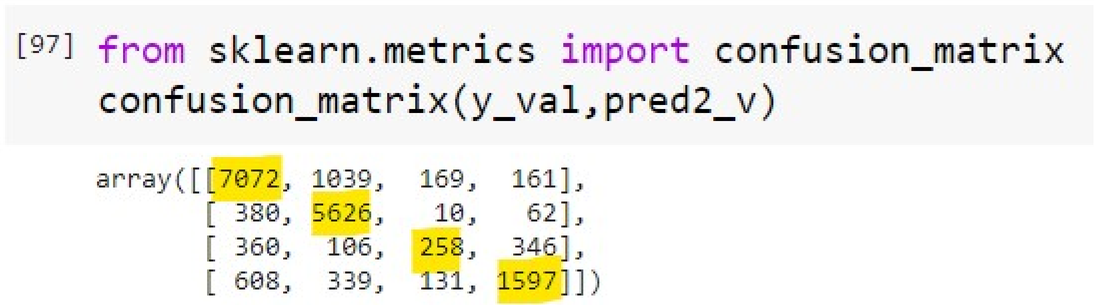
Stochastic Gradient Descent Classifier (SGDClassifier)

Stochastic Gradient Descent Classifier as the second model applied to the training dataset. SGD classifier is a linear classifier that uses stochastic gradient descent for learning the algorithm and the default loss function used is hinge. This gives a linear Support Vector Machine Classifier. The advantages for considering SGD Classifier over SVM are the learning algorithm and the response time for training the model. The classification report, the confusion matrix and the performance measures are shown below. If we observe, the weighted average accuracy 70% and the correctly classified CERTIFIED H1B Applications are around 70% and 49% respectively which are lesser than that of Multinomial Logistic Regression.



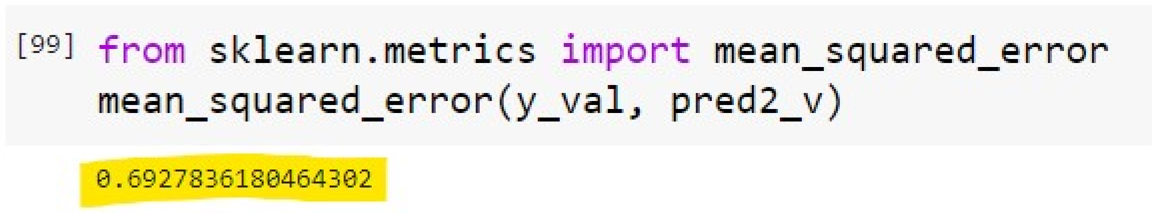
```
[95] from sklearn.metrics import classification_report
     print(classification_report(y_val,pred2_v))

              precision    recall  f1-score   support

           0       0.84      0.84      0.84      8441
           1       0.79      0.93      0.85      6078
           2       0.45      0.24      0.32      1070
           3       0.74      0.60      0.66      2675

    accuracy                           0.80     18264
   macro avg       0.71      0.65      0.67     18264
weighted avg       0.79      0.80      0.79     18264
```

Fig: Classification Report (SGD)



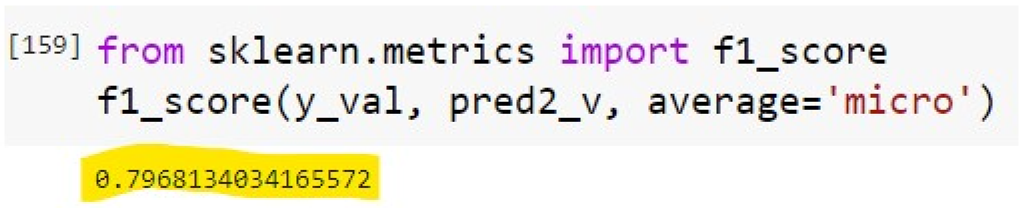
```
[97] from sklearn.metrics import confusion_matrix
     confusion_matrix(y_val,pred2_v)

array([[7072, 1039,  169,  161],
       [ 380, 5626,   10,   62],
       [ 360,  106,  258,  346],
       [ 608,  339,  131, 1597]])
```

Fig: Confusion Matrix (SGD)



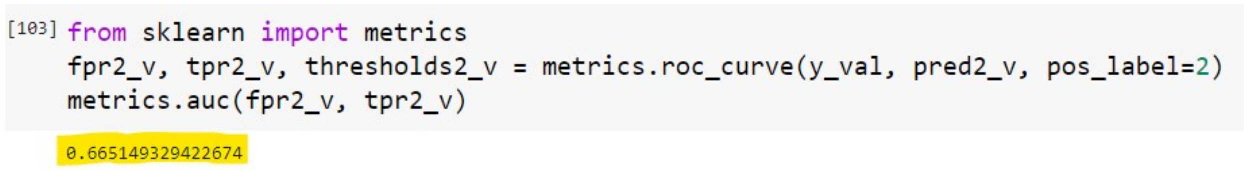
```
[99] from sklearn.metrics import mean_squared_error
     mean_squared_error(y_val, pred2_v)

0.6927836180464302
```

Fig: MSE (SGD)



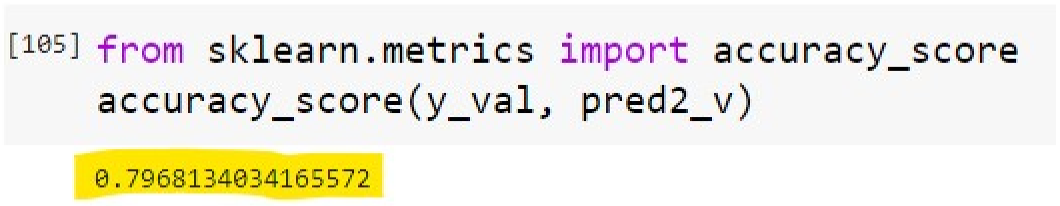
```
[159] from sklearn.metrics import f1_score
      f1_score(y_val, pred2_v, average='micro')

      0.7968134034165572
```

Fig: F1 score (SGD)



```
[103] from sklearn import metrics
      fpr2_v, tpr2_v, thresholds2_v = metrics.roc_curve(y_val, pred2_v, pos_label=2)
      metrics.auc(fpr2_v, tpr2_v)

      0.665149329422674
```

Fig: AUC (SGD)



```
[105] from sklearn.metrics import accuracy_score
      accuracy_score(y_val, pred2_v)

      0.7968134034165572
```

Fig: Accuracy (SGD)

Decision Tree:

Decision Tree is built by iteratively calculating different features of the training sample and at each node, using the feature that best splits the data. Decision tree can work with both numeric and categorical data. In this project we used the gini index (a measure of impurity for classification) metrics to evaluate the performance of the categorical split. The tree starts with a variable that has the highest gini value and splits down. The lower the gini index, the purer the variable at the leaf node. We used the maximum depth of the tree as a metric for our growth -stop condition which was set at "4" after carefully iterating through different figures.

Below is an image of a decision tree:

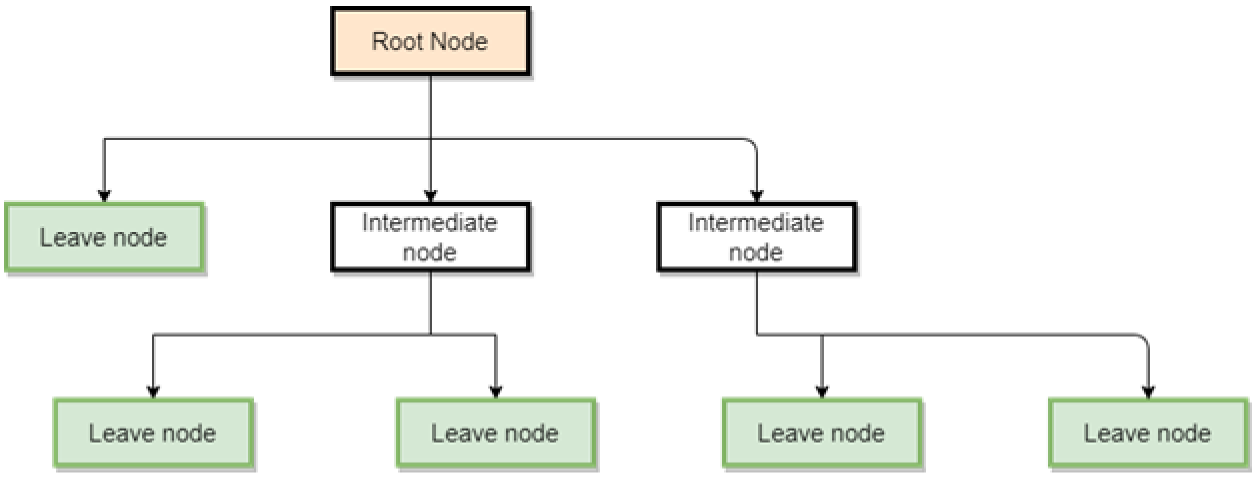
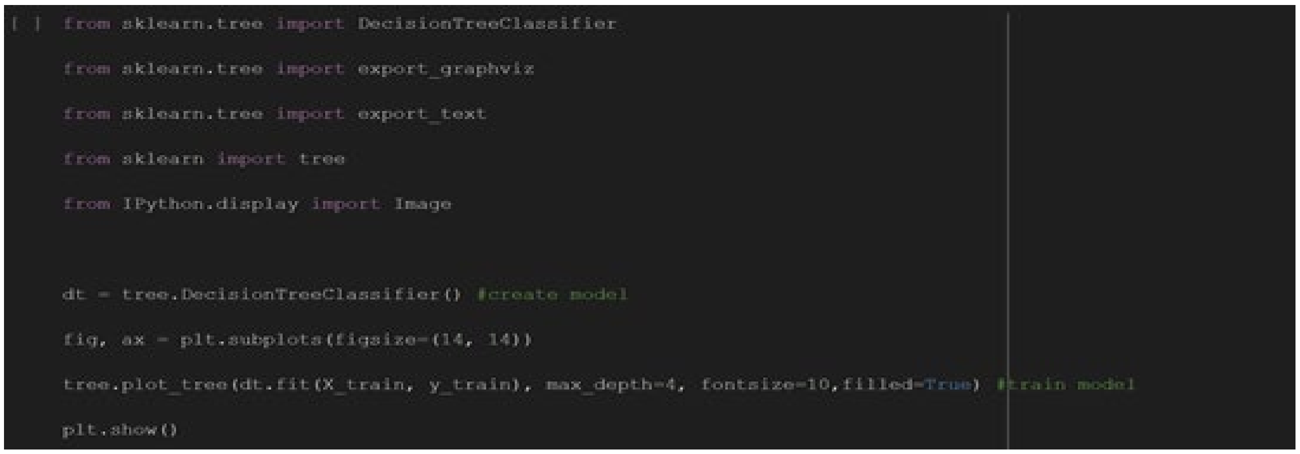


Figure of a decision tree. https://towardsdatascience.com/decision-trees-explained

The Root Node starts the graph and assesses the feature that best splits the data. Intermediate nodes further assess the features. Leaf nodes are the final nodes of the tree, predictions of a categorical or numerical variable are done at this node.

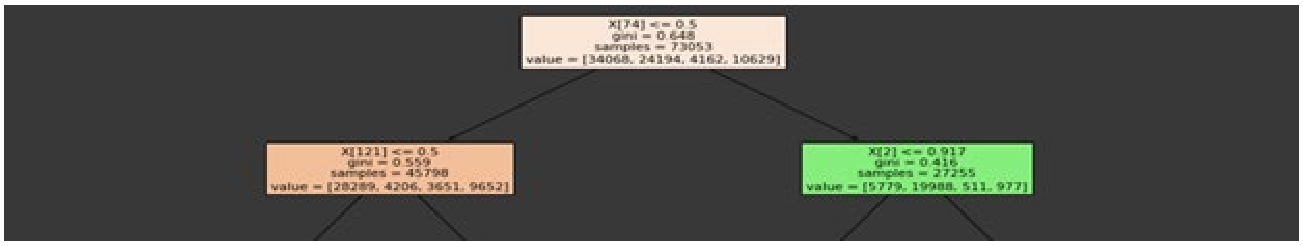
Steps:

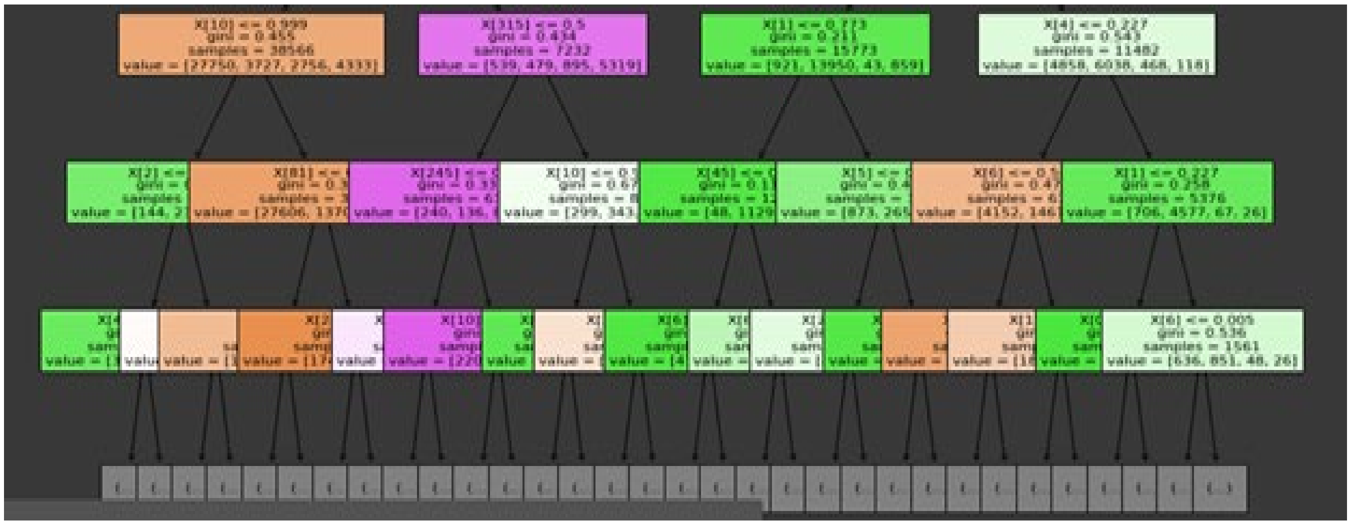
1.We imported the decision tree packages from sklearn and the image/visualization from ipython and sklearn.



```
[ ]  from sklearn.tree import DecisionTreeClassifier

     from sklearn.tree import export_graphviz

     from sklearn.tree import export_text

     from sklearn import tree

     from IPython.display import Image


     dt = tree.DecisionTreeClassifier() #create model

     fig, ax = plt.subplots(figsize=(14, 14))

     tree.plot_tree(dt.fit(X_train, y_train), max_depth=4, fontsize=10,filled=True) #train model

     plt.show()
```

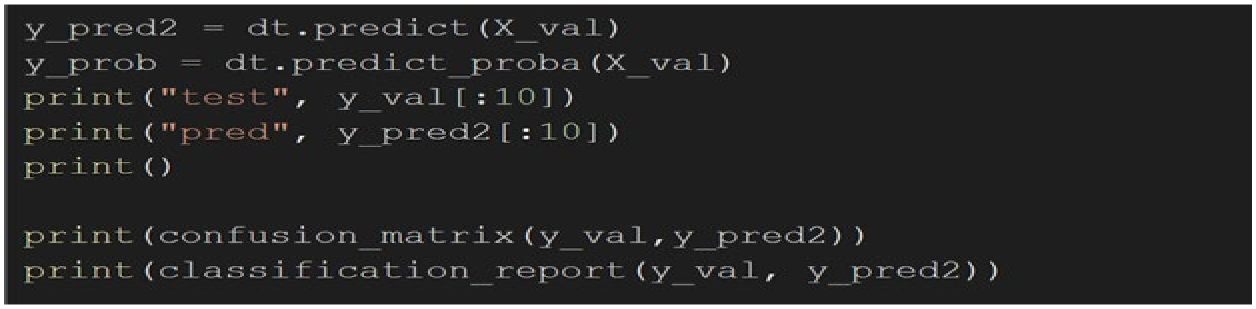
2.We then fit the trained the data into the decision model and generated the plot:



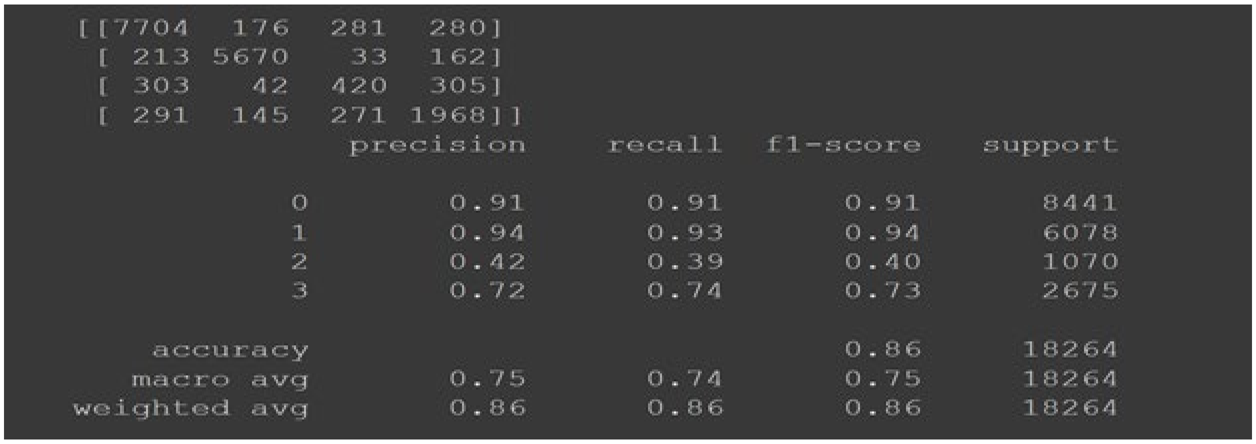


Although the plot does not show the features due the fact that we are dealing with a multiclass classification problem and the dataset is encoded and broken down into additional features representing each class thereby, it hurts its interpretability. This can be resolved by merging the multiclass into a binary classification, hence reducing the number of features. However, for the sake of this project, we decided to consider all classes in the H1B dataset so as to be more detailed in our prediction.

3.We generated the classification report and performance metrics for each class: 15,762 accurate predictions out of 18,264

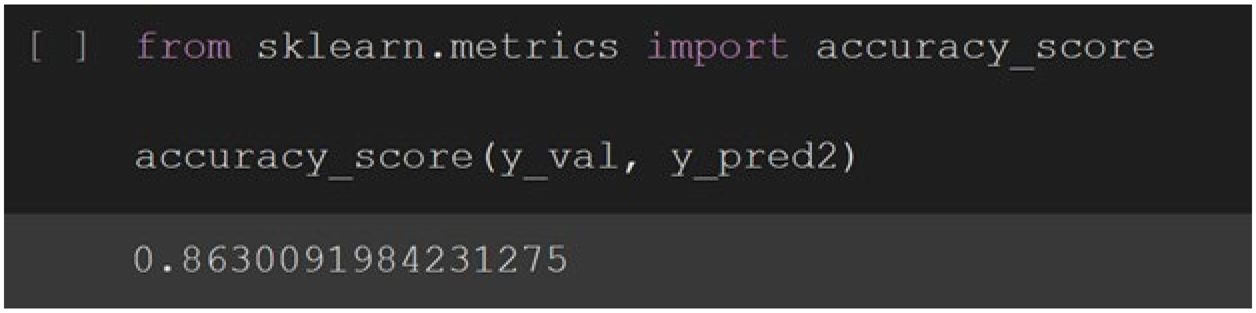


```
y_pred2 = dt.predict(X_val)
y_prob = dt.predict_proba(X_val)
print("test", y_val[:10])
print("pred", y_pred2[:10])
print()

print(confusion_matrix(y_val,y_pred2))
print(classification_report(y_val, y_pred2))
```



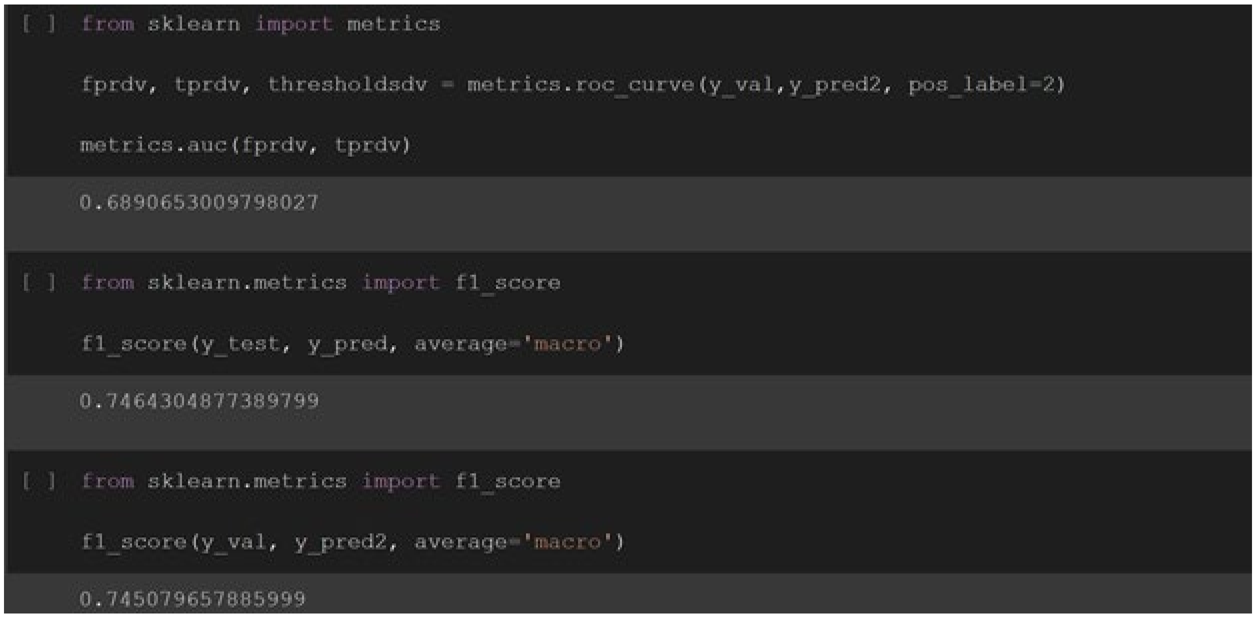
```
[[7704  176  281  280]
 [ 213 5670   33  162]
 [ 303   42  420  305]
 [ 291  145  271 1968]]
              precision    recall  f1-score   support

           0       0.91      0.91      0.91      8441
           1       0.94      0.93      0.94      6078
           2       0.42      0.39      0.40      1070
           3       0.72      0.74      0.73      2675

    accuracy                           0.86     18264
   macro avg       0.75      0.74      0.75     18264
weighted avg       0.86      0.86      0.86     18264
```

4.We generated the performance metrics, aggregated for all the classes in the test and validation set:



```
[ ] from sklearn.metrics import accuracy_score

    accuracy_score(y_val, y_pred2)
```

```
0.8630091984231275
```



```
[ ] from sklearn import metrics

    fprdv, tprdv, thresholdsdv = metrics.roc_curve(y_val,y_pred2, pos_label=2)

    metrics.auc(fprdv, tprdv)
```

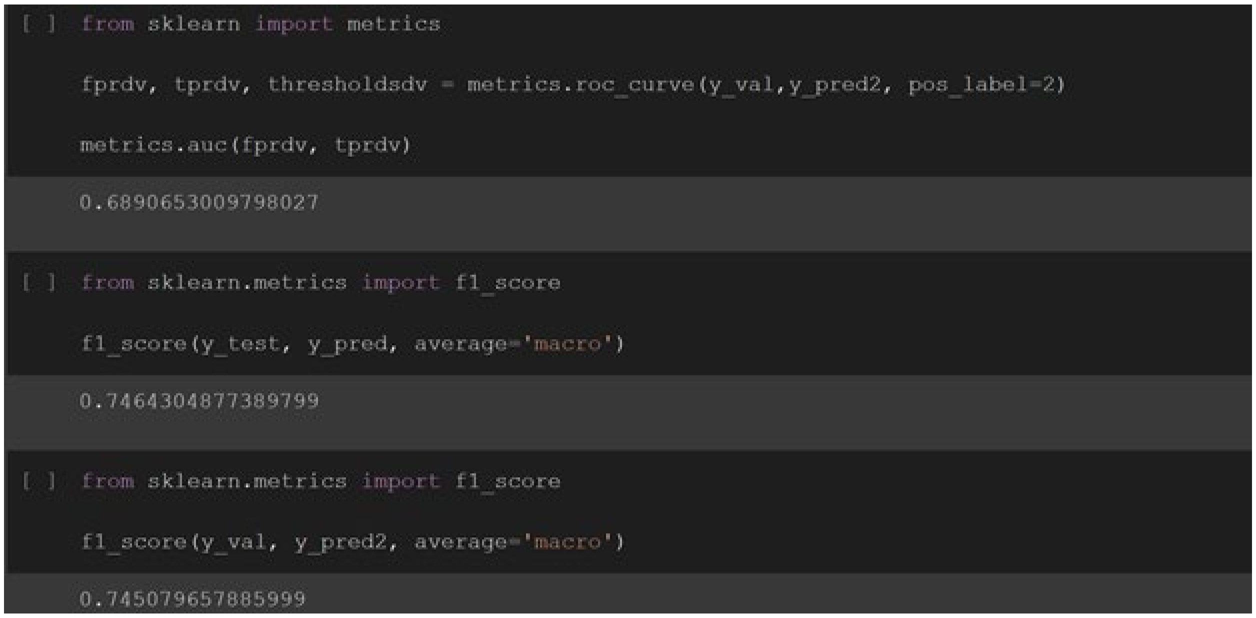
```
0.6890653009798027
```

```
[ ] from sklearn.metrics import f1_score

    f1_score(y_test, y_pred, average='macro')
```

```
0.7464304877389799
```

```
[ ] from sklearn.metrics import f1_score

    f1_score(y_val, y_pred2, average='macro')
```

```
0.745079657885999
```



```
[ ] from sklearn import metrics

    fprdv, tprdv, thresholdsdv = metrics.roc_curve(y_val,y_pred2, pos_label=2)

    metrics.auc(fprdv, tprdv)
```

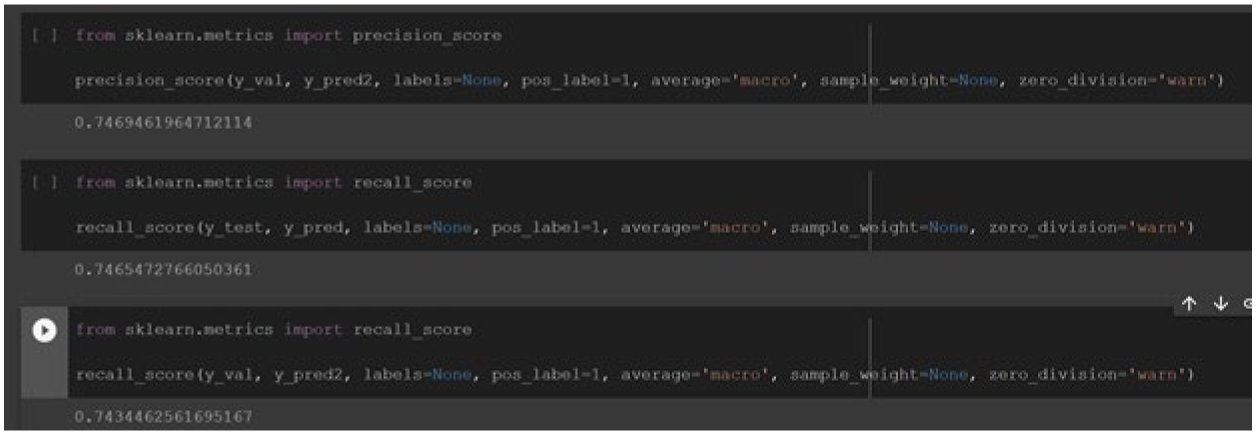
```
0.6890653009798027
```

```
[ ] from sklearn.metrics import f1_score

    f1_score(y_test, y_pred, average='macro')
```

```
0.7464304877389799
```

```
[ ] from sklearn.metrics import f1_score

    f1_score(y_val, y_pred2, average='macro')
```

```
0.745079657885999
```



```
[ ] from sklearn.metrics import precision_score

    precision_score(y_val, y_pred2, labels=None, pos_label=1, average='macro', sample_weight=None, zero_division='warn')

    0.7469461964712114

[ ] from sklearn.metrics import recall_score

    recall_score(y_test, y_pred, labels=None, pos_label=1, average='macro', sample_weight=None, zero_division='warn')

    0.7465472766050361

    from sklearn.metrics import recall_score

    recall_score(y_val, y_pred2, labels=None, pos_label=1, average='macro', sample_weight=None, zero_division='warn')

    0.7434462561695167
```

This entire process was applied to the dataset before and after applying stratified sampling and the results are summarized in the "model selection" section.

Advantages of Decision Trees

- Easy to understand and can train smaller amounts of data.
- Manages missing values instinctively and handles both categorical and numerical variables.
- Can be used for both Regression and Classification

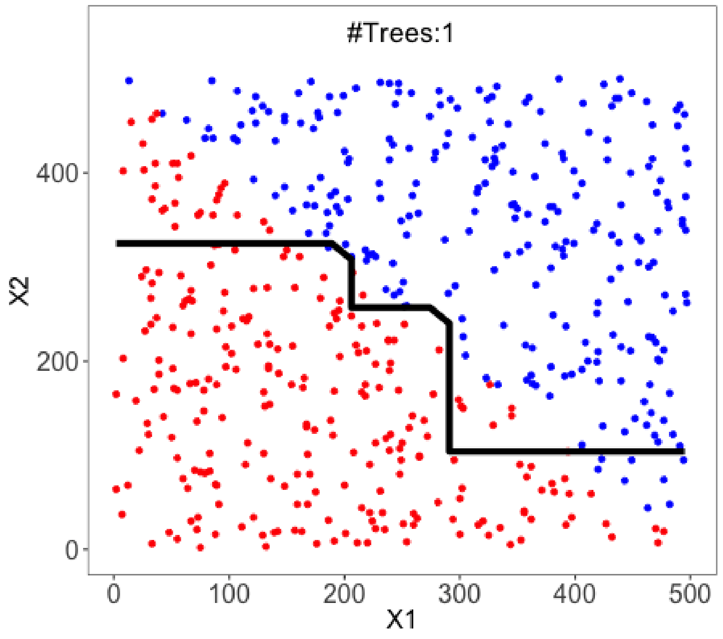
Disadvantages

- Prone to overfitting to the training data
- Sensible to outliers.
- Weak learners: a single decision tree typically does not make good predictions; therefore multiple trees are sometimes merged to make 'forests' to deliver stronger ensemble models. These trees are called "Random Forest", so we went further to apply this model to our dataset as discussed in the next section.

Random Forest

Random forest is also an ensemble tree-based learning algorithm. It comprises multiple decision trees each centered on a random sample of the training data. They are typically more accurate than single decision trees as Each tree individually predicts for the new data and random forest splits out the mean prediction from those trees. In other words, votes from each decision tree are aggregated to come up with the final class of the test object.

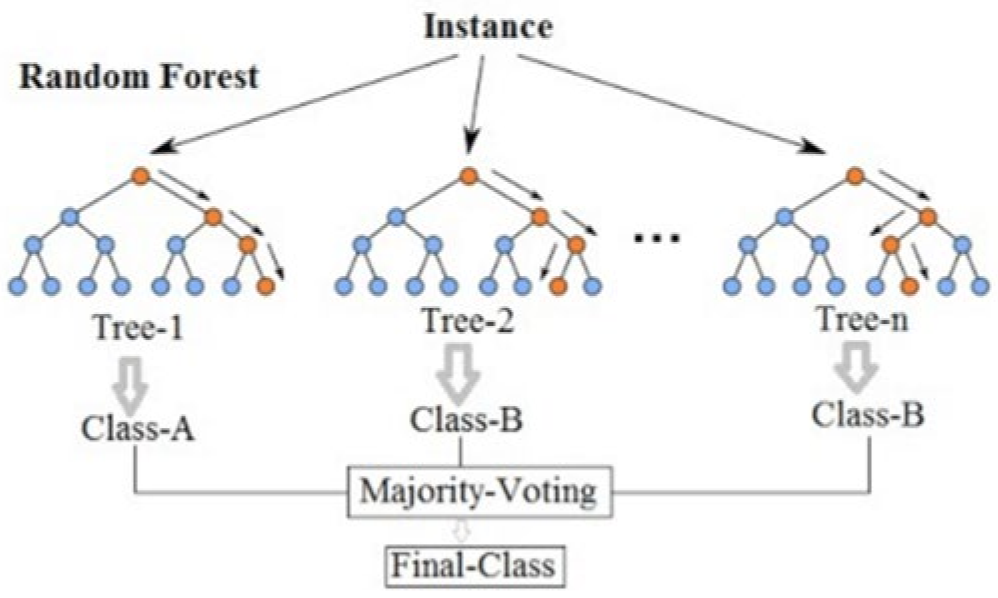
The figure below shows that the decision boundary becomes more accurate and stable as more trees are included.



Source:https://towardsdatascience.com/why-random-forests-outperform-decision-trees-1b0f175a0b5

Random forests outperform single decision trees because the trees are unpruned (bagged) and diverse, which leads to a higher resolution in the feature space and as features are added, the decision boundary becomes smoother , compared to a single decision tree that requires pruning to avoid overfitting. Random Forest trains data using bootstrap aggregation (or sampling without replacement, alongside randomly selecting features for a split. Therefore, the randomness and voting mechanism in Random Forest smartly solves overfitting that could occur in single decision trees.

Although random forests are accurate, they are considered as black-box models (hard to explain).



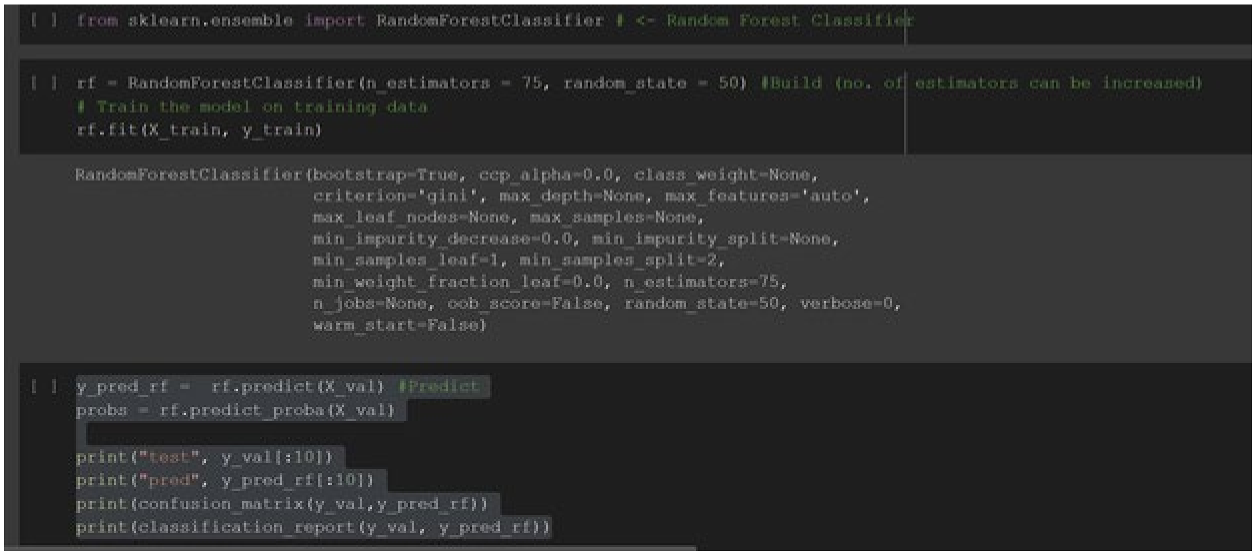
Structure of Random Forest Classification

Source:https://medium.com/swlh/random-forest-classification-and-its-implementation-d5d840dbead0

<u>Implementation</u>

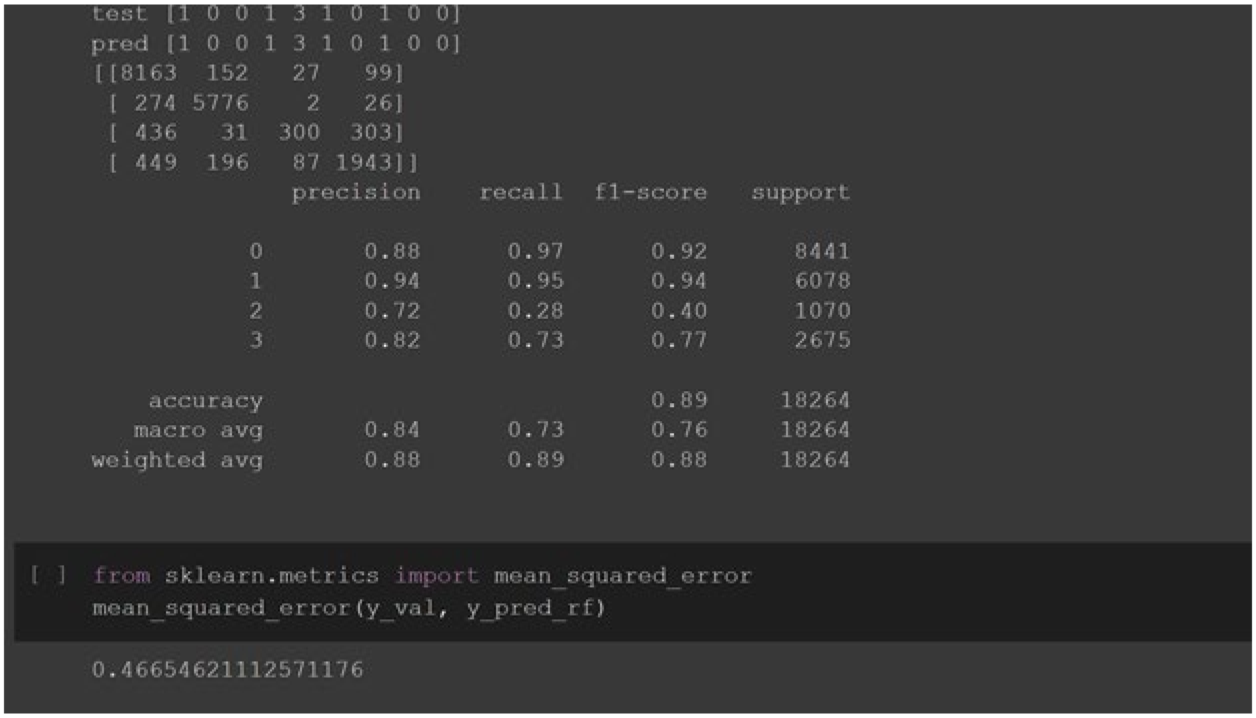
The same steps were applied as with previous models, the following codes were used and results were generated:

1. Imported the random forest package from scikit learn, trained and fit the validation set into the trained model for prediction



```
[ ] from sklearn.ensemble import RandomForestClassifier # <- Random Forest Classifier

[ ] rf = RandomForestClassifier(n_estimators = 75, random_state = 50) #Build (no. of estimators can be increased)
    # Train the model on training data
    rf.fit(X_train, y_train)

    RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                           criterion='gini', max_depth=None, max_features='auto',
                           max_leaf_nodes=None, max_samples=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=75,
                           n_jobs=None, oob_score=False, random_state=50, verbose=0,
                           warm_start=False)

[ ] y_pred_rf =  rf.predict(X_val) #Predict
    probs = rf.predict_proba(X_val)

    print("test", y_val[:10])
    print("pred", y_pred_rf[:10])
    print(confusion_matrix(y_val,y_pred_rf))
    print(classification_report(y_val, y_pred_rf))
```

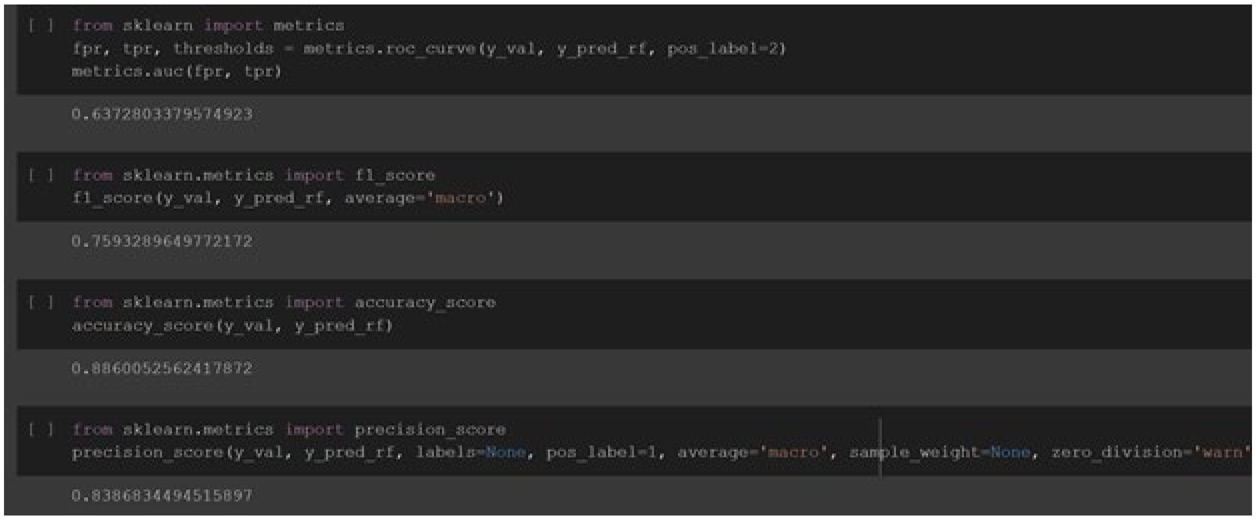
2. Generated classification report and confusion matrix for each class: 16,182 accurate predictions out of 18,264



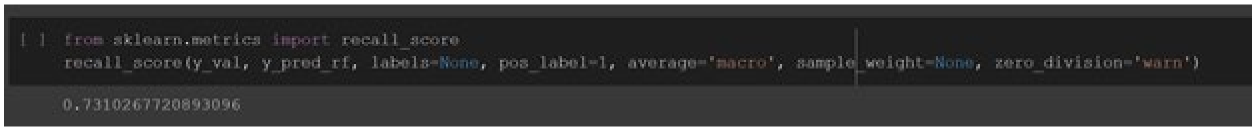
```
test [1 0 0 1 3 1 0 1 0 0]
pred [1 0 0 1 3 1 0 1 0 0]
[[8163  152   27   99]
 [ 274 5776    2   26]
 [ 436   31  300  303]
 [ 449  196   87 1943]]
              precision    recall  f1-score   support

           0       0.88      0.97      0.92      8441
           1       0.94      0.95      0.94      6078
           2       0.72      0.28      0.40      1070
           3       0.82      0.73      0.77      2675

    accuracy                           0.89     18264
   macro avg       0.84      0.73      0.76     18264
weighted avg       0.88      0.89      0.88     18264
```

```
[ ] from sklearn.metrics import mean_squared_error
    mean_squared_error(y_val, y_pred_rf)

    0.46654621112571176
```

3. Generated performance matrix for the entire validation set



```
[ ] from sklearn import metrics
    fpr, tpr, thresholds = metrics.roc_curve(y_val, y_pred_rf, pos_label=2)
    metrics.auc(fpr, tpr)

    0.6372803379574923

[ ] from sklearn.metrics import f1_score
    f1_score(y_val, y_pred_rf, average='macro')

    0.7593289649772172

[ ] from sklearn.metrics import accuracy_score
    accuracy_score(y_val, y_pred_rf)

    0.8860052562417872

[ ] from sklearn.metrics import precision_score
    precision_score(y_val, y_pred_rf, labels=None, pos_label=1, average='macro', sample_weight=None, zero_division='warn'

    0.8386834494515897
```



```
[ ] from sklearn.metrics import recall_score
    recall_score(y_val, y_pred_rf, labels=None, pos_label=1, average='macro', sample_weight=None, zero_division='warn')

    0.7310267720893096
```

In fitting the validation set to the trained model for prediction, the number of trees as given by n_estimators are set to 75. We iterated through 55, 65, 75, 85 and 100 and noticed no difference in the output after 75 (which produced the best result).

Advantages

- One of the most accurate learning algorithms (Produces a highly accurate classifier) which operates effectively on large databases.
- Manages thousands of input variables and can determine the most substantial variables and helps in avoiding the overfitting problem
- Effective in estimating missing data and maintains accuracy with a large proportion of missing data

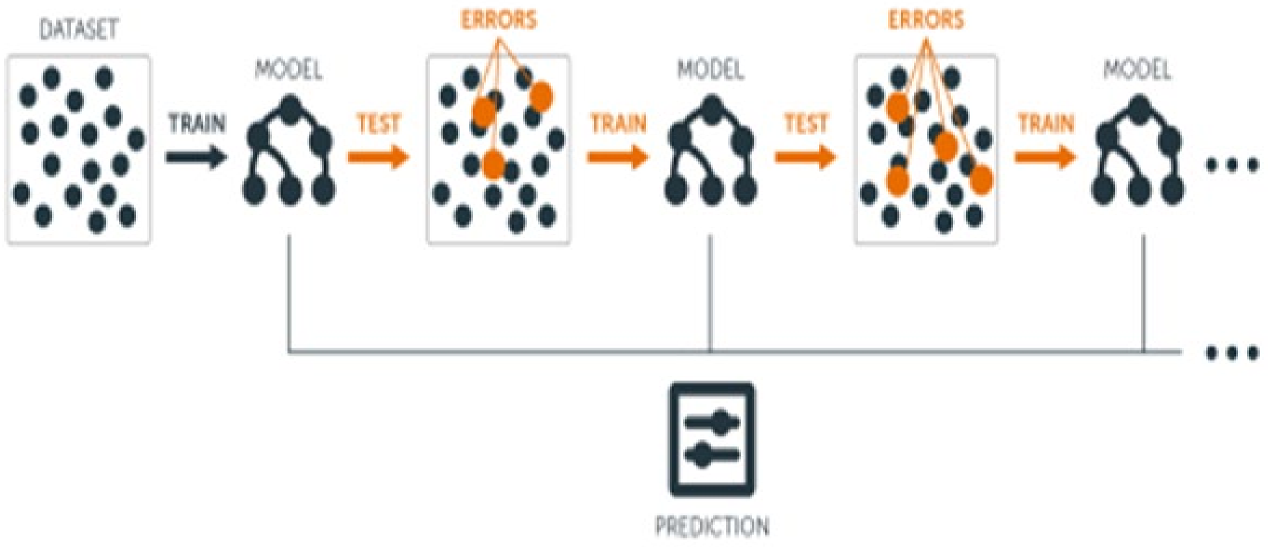
Disadvantages

- Arguably difficult to interpret due to its complexity (although this has been addressed with tools such as variable importance, partial dependence plots, LIME etc)
- Longer training period compared to decision trees

Gradient Boosting

Gradient boosting classifiers are a group of machine learning algorithms that merge many weak learning models together to create a strong predictive model. Decision trees are usually used when doing gradient boosting. Gradient boosting models are increasingly popular because of their effectiveness at classifying complex datasets. While random forests build an ensemble of deep individual trees, GBMs builds an ensemble of shallow trees. The trees come in sequence with each tree learning and improving on the previous one. Gradient Boosting when properly tuned works effectively compared to other algorithms. It is similar to gradient descent in a neural network. The idea of the “Boost” is as a result of weakness in shallow trees' predictability. It works well with models that have high bias and low variance. Boosting attacks bias-variance-tradeoff by starting with a weak model like a decision tree with only a few splits, and sequentially boosts its performance by building new trees continually, each new tree in the sequence acts on the training rows where the previous tree has the largest prediction errors.

Sequential ensemble approach.



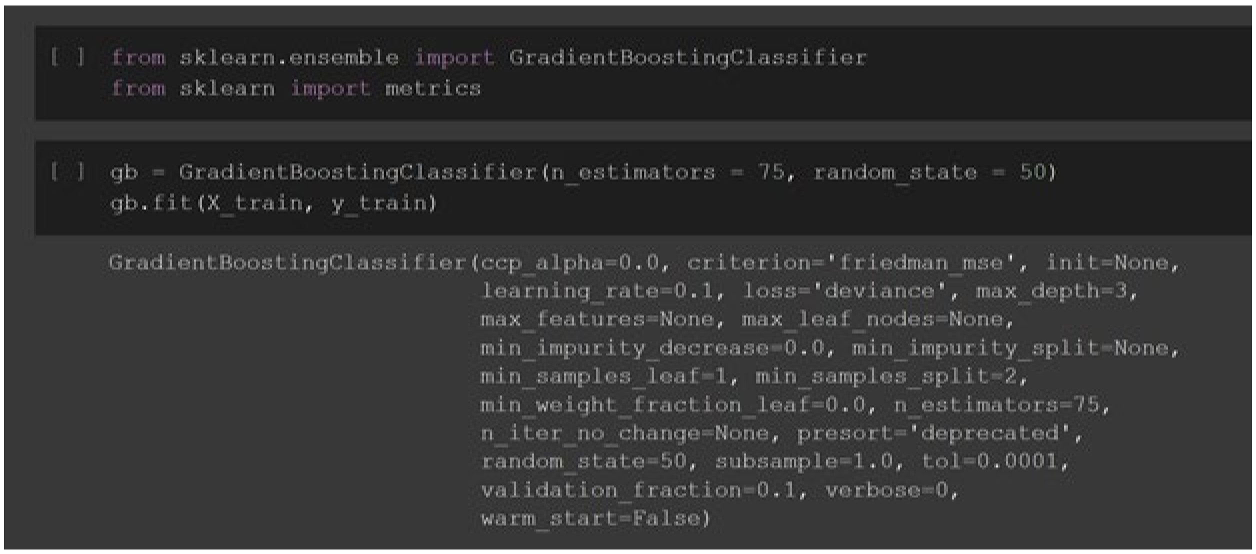
Source: *https://bradleyboehmke.github.io/HOML/gbm.html*

From the diagram above we can see that Gradient boosting has two necessary parts: a weak learner and an additive component. Weak learners are sequentially corrected by their predecessors, thereby converting them to strong learners to arrive at an accurate predictor at the end of the process. GBM classifiers use decision trees as their weak learners which output real values and as new learners are added into the model the output of the decision trees can be added together to correct for errors in the predictions, thereby significantly reducing errors over time . GBMs typically work effectively with multiclass classification problems as evident in our dataset.

Implementation

The same steps were applied as with previous models, the following codes were used, and results were generated:

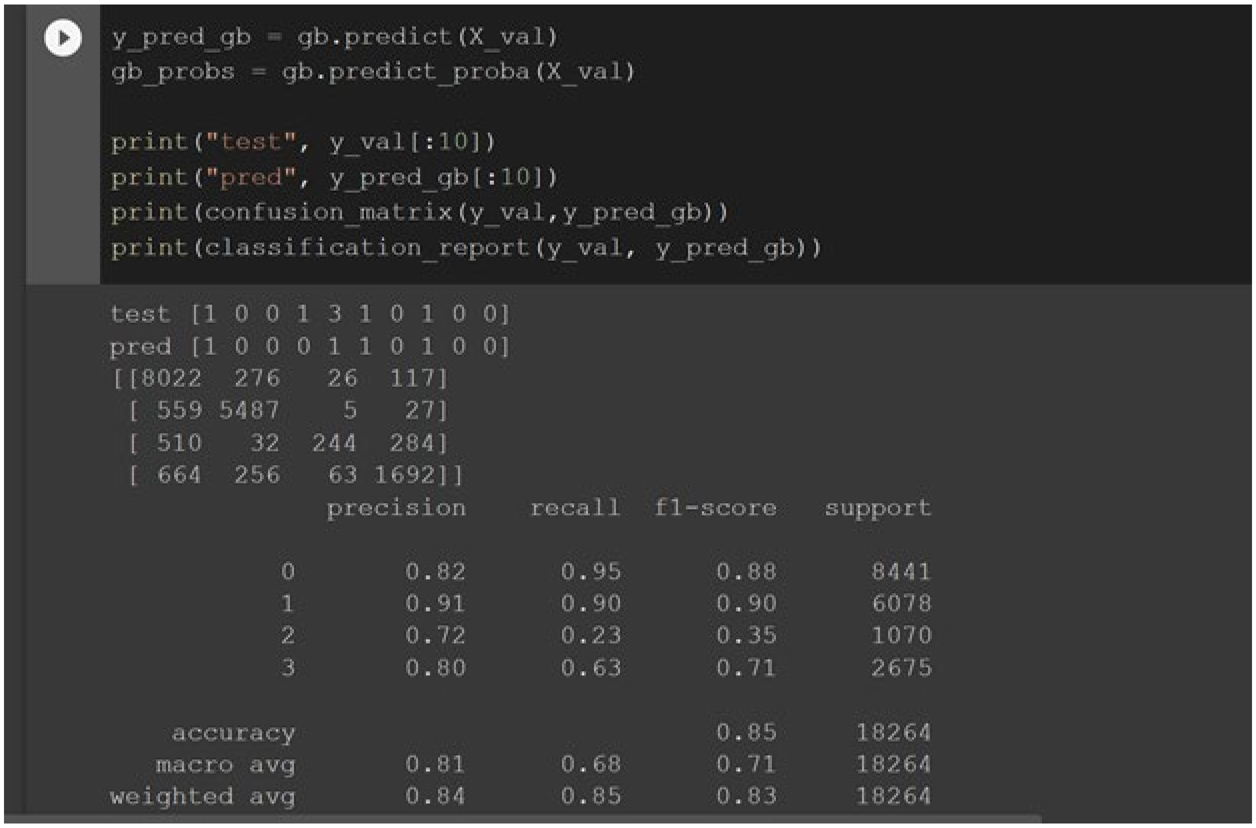
Note: In training the model, the number of trees (n_estimators) was also stable at 75 after iterating between different figures.



```
[ ] from sklearn.ensemble import GradientBoostingClassifier
    from sklearn import metrics
```

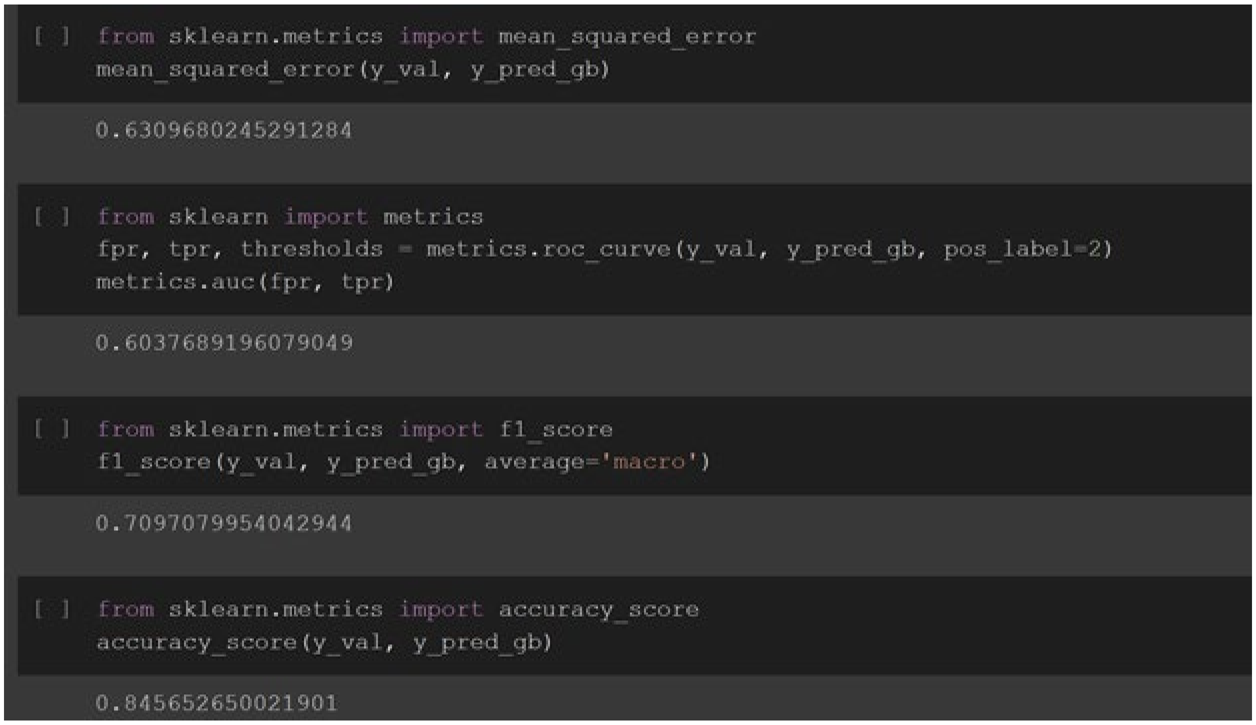
```
[ ] gb = GradientBoostingClassifier(n_estimators = 75, random_state = 50)
    gb.fit(X_train, y_train)
```

```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=75,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=50, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

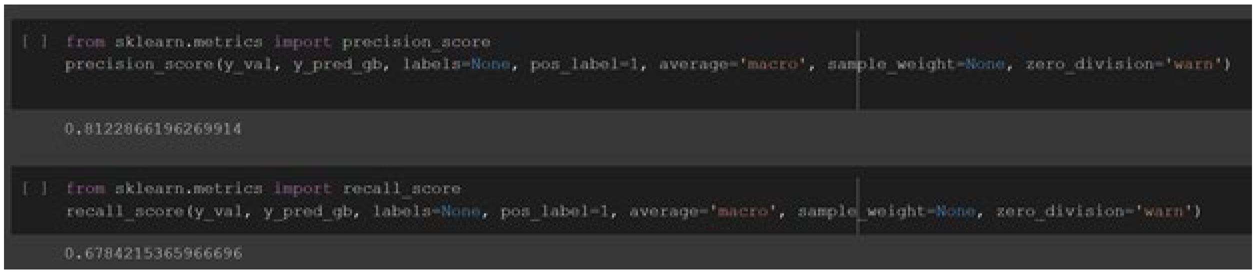


```
y_pred_gb = gb.predict(X_val)
gb_probs = gb.predict_proba(X_val)

print("test", y_val[:10])
print("pred", y_pred_gb[:10])
print(confusion_matrix(y_val,y_pred_gb))
print(classification_report(y_val, y_pred_gb))
```

```
test [1 0 0 1 3 1 0 1 0 0]
pred [1 0 0 0 1 1 0 1 0 0]
[[8022  276   26  117]
 [ 559 5487    5   27]
 [ 510   32  244  284]
 [ 664  256   63 1692]]
              precision    recall  f1-score   support

           0       0.82      0.95      0.88      8441
           1       0.91      0.90      0.90      6078
           2       0.72      0.23      0.35      1070
           3       0.80      0.63      0.71      2675

    accuracy                           0.85     18264
   macro avg       0.81      0.68      0.71     18264
weighted avg       0.84      0.85      0.83     18264
```



```
[ ] from sklearn.metrics import mean_squared_error
    mean_squared_error(y_val, y_pred_gb)

    0.6309680245291284

[ ] from sklearn import metrics
    fpr, tpr, thresholds = metrics.roc_curve(y_val, y_pred_gb, pos_label=2)
    metrics.auc(fpr, tpr)

    0.6037689196079049

[ ] from sklearn.metrics import f1_score
    f1_score(y_val, y_pred_gb, average='macro')

    0.7097079954042944

[ ] from sklearn.metrics import accuracy_score
    accuracy_score(y_val, y_pred_gb)

    0.845652650021901
```



```
[ ] from sklearn.metrics import precision_score
    precision_score(y_val, y_pred_gb, labels=None, pos_label=1, average='macro', sample_weight=None, zero_division='warn')

    0.8122866196269914

[ ] from sklearn.metrics import recall_score
    recall_score(y_val, y_pred_gb, labels=None, pos_label=1, average='macro', sample_weight=None, zero_division='warn')

    0.6784215365966696
```

<u>Advantages</u>:

- More popular with predictive accuracy and does not require data preprocessing.
- Manages missing values
- Flexible (can optimize different loss functions and provide several options for hyperparameter tuning) and works with categorical and numerical variables

<u>Disadvantages</u>

- Sensitive to outliers (since every classifier is obliged to fix errors in the predecessors). Hence, the model may tend to overfit
- Requires high computing power when compared to decision trees and random forest and is impossible to scale up.
- Interpretability (although this has been addressed with tools such as variable importance, partial dependence plots, LIME etc)

Model Comparison

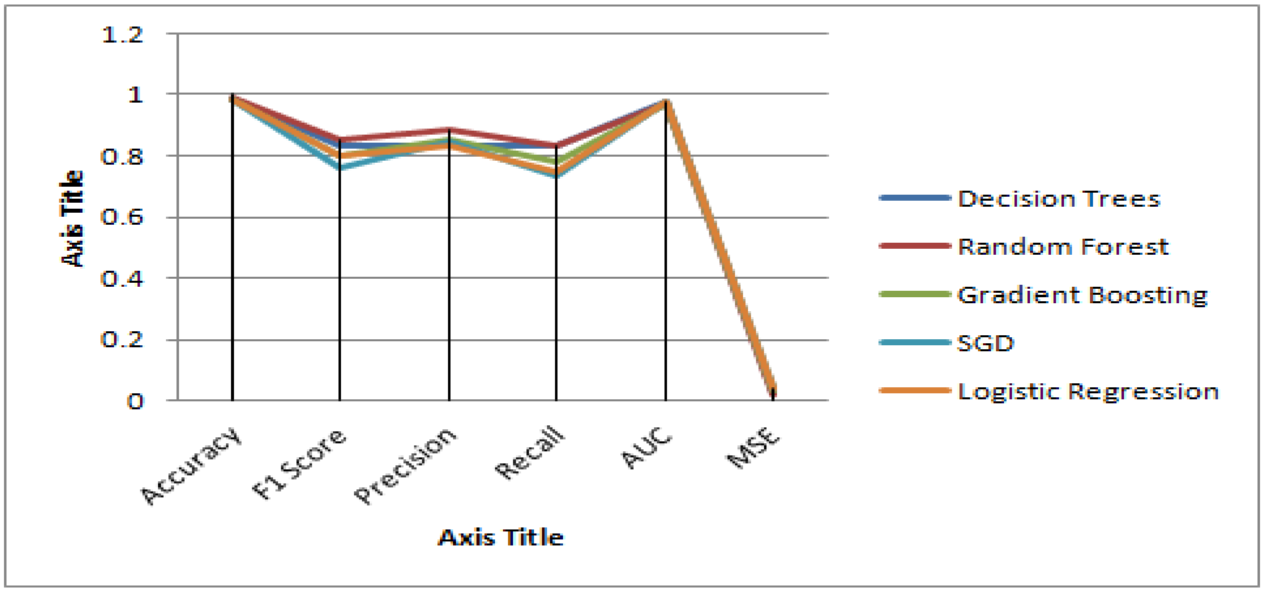
We tried out 5 different algorithms, to improve the validation results and considered six main performance metrics to evaluate the effectiveness of the classification models:

1. **Accuracy**: This tests a model's ability to correctly predict classes for a given threshold
2. **Precision (Specifitivity)**: Tests a model's ability to correctly detect positive classes from all predicted positive classes
3. **Recall (Sensitivity):** Tests the ability to correctly detect positive classes from all actual positive classes
4. **F1 Score**: This is the harmonic mean of precision and recall and it measures a model's accuracy on a dataset
5. **Area Under the ROC Curve (AUC)**: Measures how much a model is capable of distinguishing between classes considering all possible thresholds (suitable for high class imbalance)
6. **Mean Square Error (MSE)**: Estimates the average of the squares of the error, that is the average squared difference between the estimated values and the actual value.

The use of several metrics rather than a single help to understand tradeoffs between different kinds of errors and experiences. Validation dataset was used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters. The higher the values of the metrics (close to one or 100%), the better the model. This is apart from the Mean Square Error, the tendency for the model to minimize errors makes it a better model. Below is a summary of the model results before and after sampling:

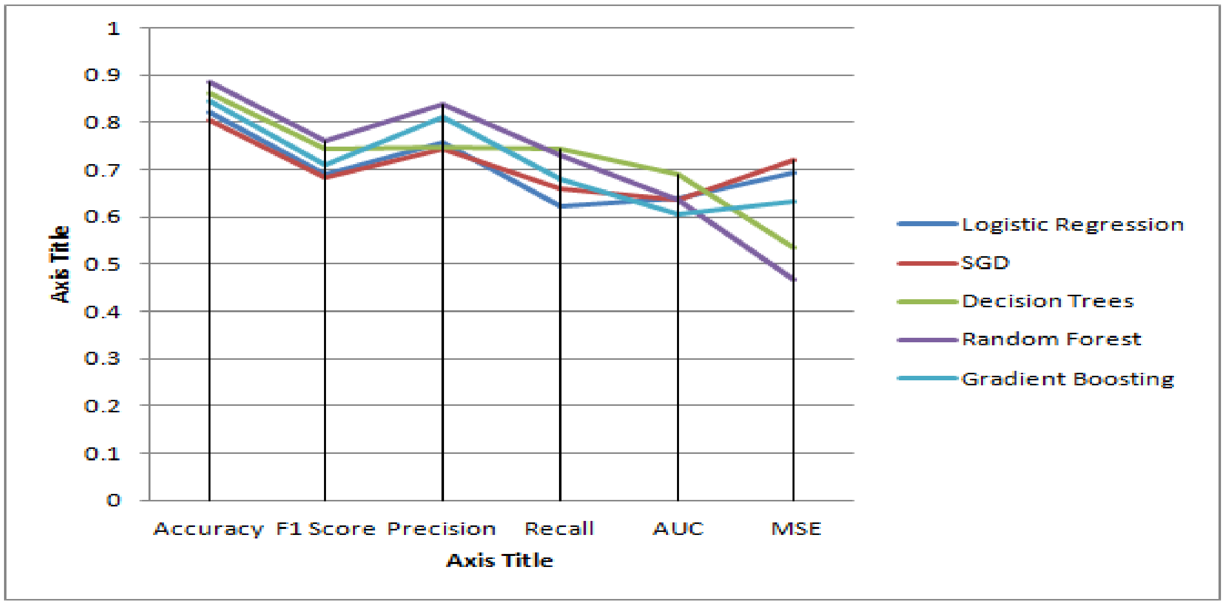
Before Sampling the dataset:

Classifier	Accuracy	F1 Score	Precision	Recall	AUC	MSE
Decision Trees	0.98549	0.83009	0.83001	0.83018	0.97419	0.02829
Random Forest	0.98891	0.85165	0.88479	0.83412	0.97315	0.02163
Gradient Boosting	0.98495	0.79929	0.85328	0.78384	0.97363	0.03233
SGD	0.98202	0.76342	0.84579	0.7358	0.97406	0.04277
Logistic Regression	0.9832	0.79769	0.83199	0.74821	0.97441	0.0391



After Sampling the Dataset.

Classifier	Accuracy	F1 Score	Precision	Recall	AUC	MSE
Logistic Regression	0.82189	0.69122	0.7569	0.62351	0.64044	0.69185
SGD	0.80551	0.68146	0.74335	0.65796	0.63499	0.71933
Decision Trees	0.863	0.74507	0.74694	0.74344	0.68906	0.53345
Random Forest	0.886	0.75932	0.83868	0.73102	0.63728	0.46654
Gradient Boosting	0.84565	0.7097	0.81228	0.67842	0.60376	0.63096

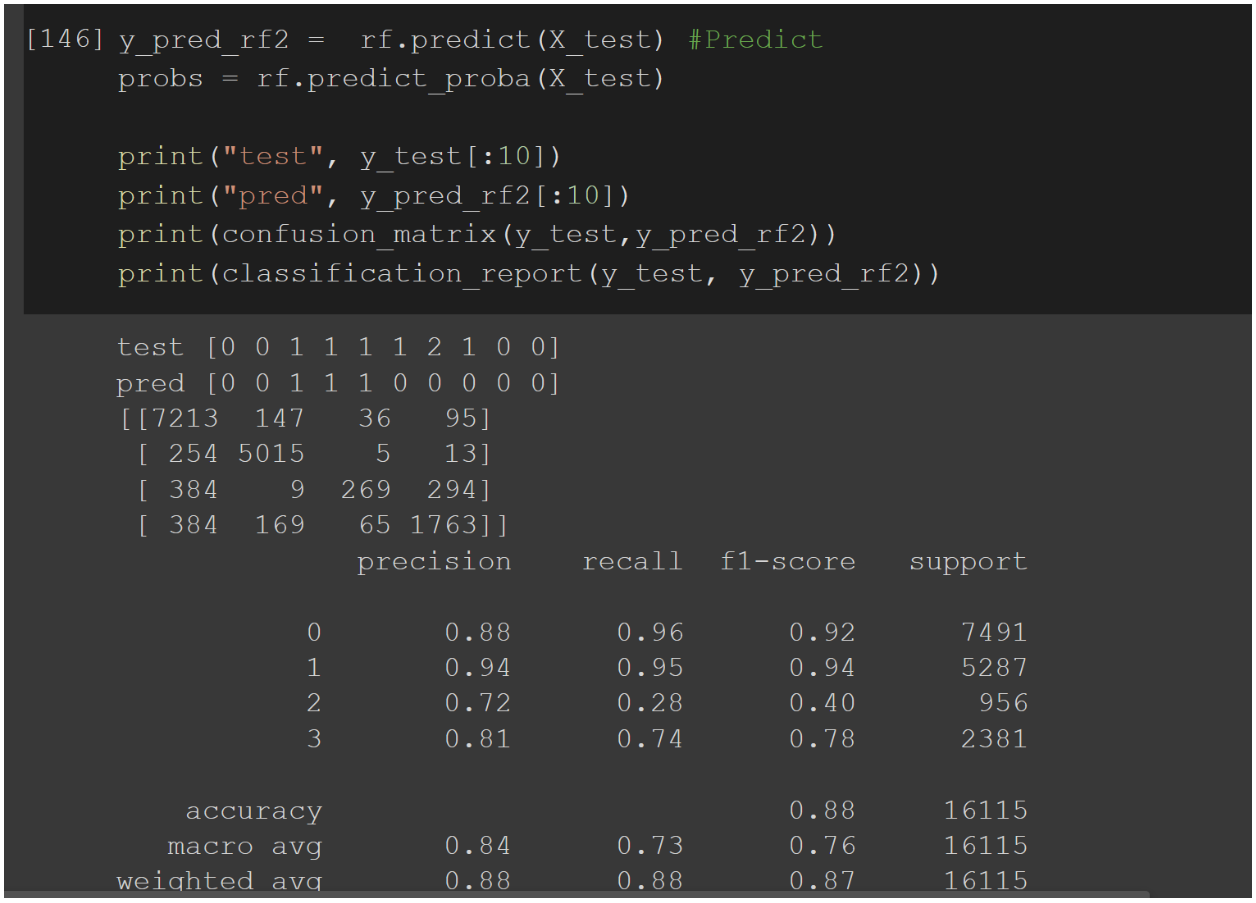


Interpretation

In the first instance (before sampling) we were dealing with a highly imbalanced dataset therefore we can see that the accuracy of each model is between 98% and 99% which means that the model could be biased towards the higher class. After undersampling the dataset, the figures dropped to a more realistic margin as seen on the second table/plot. Regardless of the results before and after sampling, **Random Forest proved to be the best model** with the highest accuracy and lowest MSE, and also performed better in the majority of the metrics compared to other algorithms. Random Forests are highly parsimonious and effectively handle any issues with bias, overfitting, normalization of the data, as our dataset is very complex and prone to these anomalies.

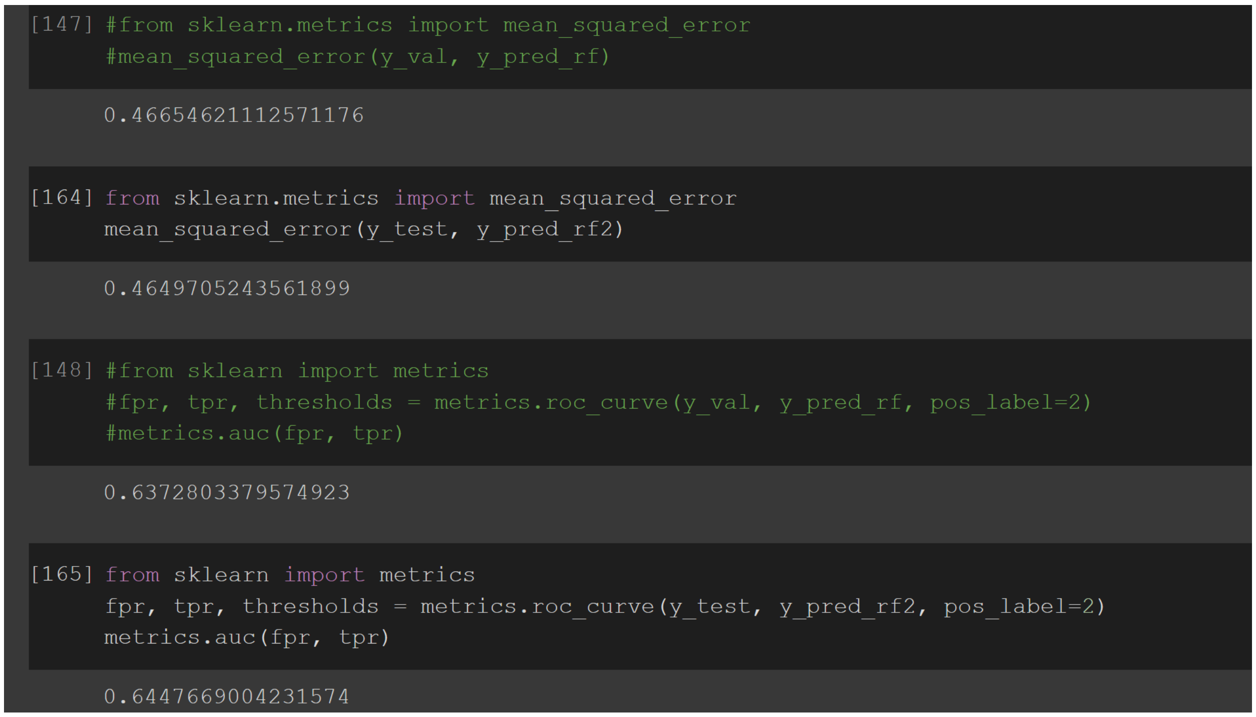
Model Testing

After selecting the best model (Random Forest), we fit the Test set into the model which is necessary to evaluate how well the model will do with data outside the training set and for further deployment in real time use. The following codes were used, and the results were generated:

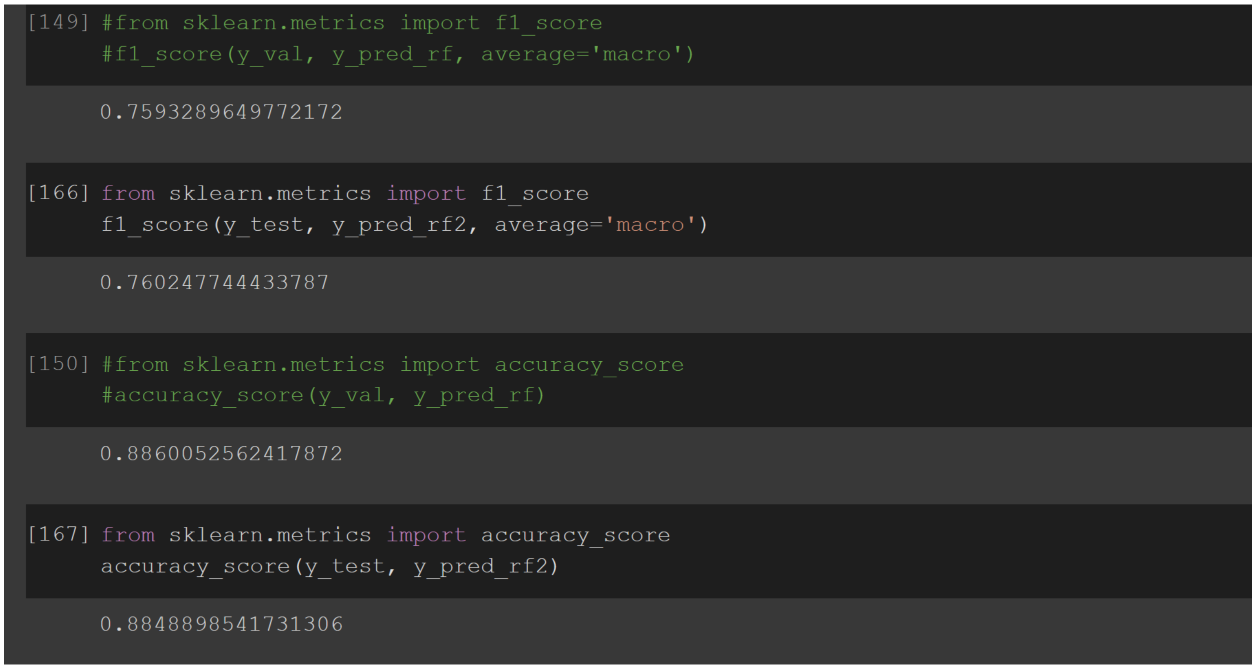


```
[146] y_pred_rf2 =  rf.predict(X_test) #Predict
      probs = rf.predict_proba(X_test)

      print("test", y_test[:10])
      print("pred", y_pred_rf2[:10])
      print(confusion_matrix(y_test,y_pred_rf2))
      print(classification_report(y_test, y_pred_rf2))
```

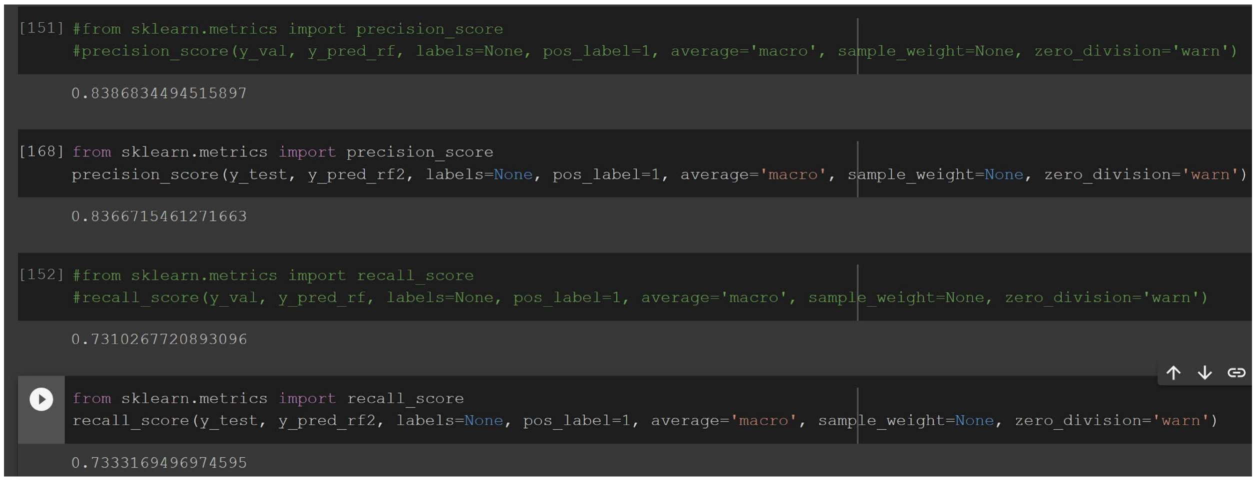
```
test [0 0 1 1 1 1 2 1 0 0]
pred [0 0 1 1 1 0 0 0 0 0]
[[7213  147   36   95]
 [ 254 5015    5   13]
 [ 384    9  269  294]
 [ 384  169   65 1763]]
              precision    recall  f1-score   support

           0       0.88      0.96      0.92      7491
           1       0.94      0.95      0.94      5287
           2       0.72      0.28      0.40       956
           3       0.81      0.74      0.78      2381

    accuracy                           0.88     16115
   macro avg       0.84      0.73      0.76     16115
weighted avg       0.88      0.88      0.87     16115
```



```
[147] #from sklearn.metrics import mean_squared_error
      #mean_squared_error(y_val, y_pred_rf)

      0.46654621112571176

[164] from sklearn.metrics import mean_squared_error
      mean_squared_error(y_test, y_pred_rf2)

      0.4649705243561899

[148] #from sklearn import metrics
      #fpr, tpr, thresholds = metrics.roc_curve(y_val, y_pred_rf, pos_label=2)
      #metrics.auc(fpr, tpr)

      0.6372803379574923

[165] from sklearn import metrics
      fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_rf2, pos_label=2)
      metrics.auc(fpr, tpr)

      0.6447669004231574
```



```
[149] #from sklearn.metrics import f1_score
      #f1_score(y_val, y_pred_rf, average='macro')

      0.7593289649772172

[166] from sklearn.metrics import f1_score
      f1_score(y_test, y_pred_rf2, average='macro')

      0.760247744433787

[150] #from sklearn.metrics import accuracy_score
      #accuracy_score(y_val, y_pred_rf)

      0.8860052562417872

[167] from sklearn.metrics import accuracy_score
      accuracy_score(y_test, y_pred_rf2)

      0.8848898541731306
```



```
[151] #from sklearn.metrics import precision_score
      #precision_score(y_val, y_pred_rf, labels=None, pos_label=1, average='macro', sample_weight=None, zero_division='warn')

      0.8386834494515897

[168] from sklearn.metrics import precision_score
      precision_score(y_test, y_pred_rf2, labels=None, pos_label=1, average='macro', sample_weight=None, zero_division='warn')

      0.8366715461271663

[152] #from sklearn.metrics import recall_score
      #recall_score(y_val, y_pred_rf, labels=None, pos_label=1, average='macro', sample_weight=None, zero_division='warn')

      0.7310267720893096

      from sklearn.metrics import recall_score
      recall_score(y_test, y_pred_rf2, labels=None, pos_label=1, average='macro', sample_weight=None, zero_division='warn')

      0.7333169496974595
```

Summary of Random Forest results (Undersampled data):

Set	**Accuracy**	**F1 Score**	**Precision**	**Recall**	**AUC**	**MSE**
Test	**0.88488**	**0.76024**	**0.83667**	**0.73331**	**0.64476**	**0.46497**
Validation	0.886	0.75932	0.83868	0.73102	0.63728	0.46654

❖ Business implications and recommendations

The Business recommendation to obtain the H1B visa are stated below. This insight is recommended based on the obtained dataset by running the multinomial logistic regression.

- The employee with a high-paying job is most likely to get an H1B visa.

  With a median salary of 72000 USD a year for all H1B visa holders, computer related jobs with almost a 98% percent approval rate is nearly averaging 78000 per year, a net differential of 6000. Given a marginal tax rate of 15%, and an aggregate count of over 242000 H1B holders in that field, the tax increment estimation from the median salary tax rate equates 217,800,000 USD.
  In the last few years with the increase of cybersecurity threat, the US employers have struggled in filling those positions. Given the much needed qualified labor in cybersecurity, and the average cybersecurity salary of 90,000 USD as of 2020, the USCIS should prioritize cybersecurity occupations which have the potential to increase its tax revenue by at least 25% for all new H1B holders in the field of cybersecurity.

- The Dependents who come along with an H1B employee are most likely to get an H1B visa.
- The employee who had some violations under his employer is less likely to get an H1B visa.
- The summary result of our model suggests that H1B full time employees with high technical skill are more likely to see their H1B visa being extended. In the process of negotiating salaries, our data model is a vital baseline in deriving salaries per occupation and state to estimate the salary of new hires.

Improvement:

- The target variable classified being two classes, whether certified or denied will help to perform a better analysis. As well as it will help us to avoid over or under-sampling.
- As the entire dataset is biased towards the certified it was not easy to perform the analysis, so bringing up the balanced dataset can avoid such problems.
- To improve the results hyperparameter tuning is the best advice.
- Instead of down sampling to the CERTIFIED class, the model can be implemented by considering the up sampling of the OTHER CLASSES.

❖ Conclusion

In the light of the accuracy of our model and variance analysis, our dataset exhibits characteristics that could be well beneficial to employers as well as the department of labor. Numerous challenges were faced given the size of the dataset hence, high latency in computational time and the singularity of features and attributes. After normalization, our deep analysis suggests that while in its entirety, 89% of the cases submitted are granted with the H1B visa, a great disparity lies in the distribution with the top 3 occupations representing 78% of the overall number of cases approved. These occupations are IT jobs, Analysts and Engineering signaling a shortage of qualified candidates. Other factors such as employer name, location and period of submission have been impactful in deciding any given submitted case in a fashion that is deemed coherent and representational of the status quo. Our decision model rendered on the current average marginal tax rates and an average inflation index of 0.3% per year will generate an additional tax revenue of 1.1 billion to the federal government.

❖ References

http://uc-r.github.io/gbm_regression

https://www.sciencedirect.com/topics/medicine-and-dentistry/logistic-regression-analysis

https://corporatefinanceinstitute.com/resources/knowledge/other/boosting/

https://bradleyboehmke.github.io/HOML/gbm.html
https://towardsdatascience.com/why-random-forests-outperform-decision-trees-1b0f175a0b5

https://towardsdatascience.com/decision-trees-explained-3ec41632ceb6

https://medium.com/datadriveninvestor/visualizing-scikit-model-performance-fb26ff16f7c6https://medium.com/datadriveninvestor/visualizing-scikit-model-performance-fb26ff16f7c6

https://medium.com/swlh/random-forest-classification-and-its-implementation-d5d840dbead0

https://bradleyboehmke.github.io/HOML/gbm.html

https://medium.com/swlh/random-forest-classification-and-its-implementation-d5d840dbead0

https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

http://theprofessionalspoint.blogspot.com/2019/02/advantages-and-disadvantages-of.html

https://www.quora.com/What-are-the-advantages-and-disadvantages-for-a-random-forest-algorithm