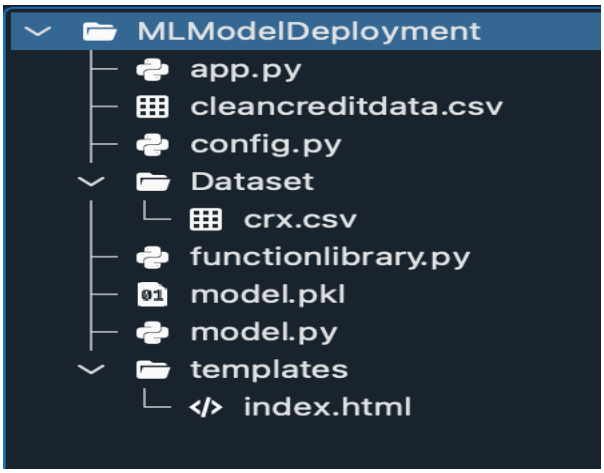
Name: Aditi Dadariya
Batch code: LISUM21
Submission date: May 28, 2023
Submitted to: Data Glacier (Week 4: Deployment on Flask)

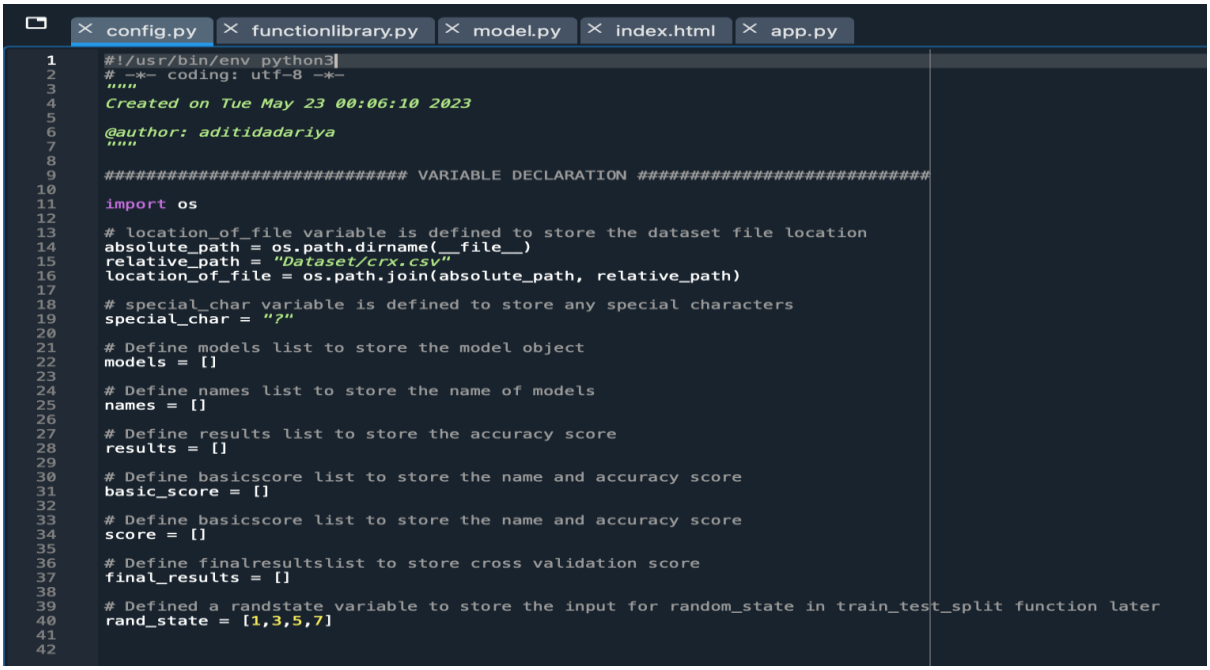
Data Description: The dataset downloaded from Kaggle has application for credit card approval [1]. All the attribute names have been changed to meaningless name to maintain the confidentiality of data. It is mentioned that the dataset contains missing values, however as examined during pre-processing, there are no missing values, instead there are special characters found in the dataset. The datatype is different for attributes in the dataset. These are continuous, nominal with small number of values and nominal with large number of values [1].

Steps followed for model development and deployment on flask:

1. Created a new project i.e. ML Model Deployment in Spyder.



2. Created few python files i.e. model.py, app.py, config.py, functionlibrary.py, index.html
3. Created a folder 'Dataset' where the dataset file 'crx.csv' has been placed.
4. Config.py file: This file has all the global variables that has been utilised in functionlibrary.py file and model.py file.



5. functionlibrary.py file: This file contains all the functions that has been utilised in model.py file to run the data analysis, visualization and modelling.

config.py | functionlibrary.py | model.py | index.html | app.py

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue May 23 00:07:18 2023

@author: aditidadariya
"""

######################### Importing necessary libraries ##########################
import pandas as pd
from matplotlib import pyplot
import seaborn as sns
# Import matplotlib.pyplot to draw and save plots
import matplotlib.pyplot as plt
# Import LabelEncoder to change the data types of attributes
from sklearn.preprocessing import LabelEncoder
# Import parameters.py file to get all the variables here
#import config.py
from config import *
# Import Decision Tree Classifier
from sklearn.tree import DecisionTreeClassifier
# Import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
# Import train_test_split function
from sklearn.model_selection import train_test_split
# Import cross_val_score function
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
# Import StratifiedKFold function
from sklearn.model_selection import StratifiedKFold
# Import GridSearchCV function
from sklearn.model_selection import GridSearchCV
# Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
# Import confusion_matrix function
from sklearn.metrics import confusion_matrix
# Import classification_report function
from sklearn.metrics import classification_report
# Import ConfusionMatrixDisplay function to plot confusion matrix
from sklearn.metrics import ConfusionMatrixDisplay
# Import render from graphviz to convert dot file to png
from graphviz import render
# Import RandomUnderSampler, SMOTE, Pipeline to balance the dataset
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline


######################### Function declarations ##########################

# Newline gives a new line
def Newline():
    print("\r\n")

# ClearLists removes all values from lists
def ClearLists():
    models.clear()
    names.clear()
    results.clear()
    basic_score.clear()
    score.clear()
    final_results.clear()



# ReadDataset function reads the dataset csv file and store it in a dataframe
# location_of_file is variable thats stores the file location and is defined in config.py file
def ReadDataset(location_of_file):
    # Read the data file from specified location
    df = pd.read_csv(location_of_file,header=None)
    # Specifying the column names into the dataset using dataframe, as there are no columns already specified in the dataset
    df.columns = ['A1', 'A2', 'A3', 'A4', 'A5', 'A6', 'A7', 'A8', 'A9', 'A10', 'A11', 'A12', 'A13', 'A14', 'A15', 'A16']
    return df


# for loop will iterate over all the columns in dataframe to find and drop the rows with special characters [4]
def RemoveSpecialChar(special_char,df):
    for eachcol in df:
        # Drop rows where "?" is displayed
        df.drop(df.loc[df[eachcol] == special_char].index, inplace=True)
    return df

```

```python

# Encoder function transforms the character and object value in dataframe [5]
def Encoder(df):
    # columnsToEncode is to get a list of columns having datatype as category and object
    columnsToEncode = list(df.select_dtypes(include=['category','object']))
    # le is an object of LabelEncoder with no parameter
    le = LabelEncoder()
    # for loop will iterate over the columns.
    # it will first try to use LabelEncoder to fit_transform
    # and if there are any error than the except will be executed
    for feature in columnsToEncode:
        try:
            df[feature] = le.fit_transform(df[feature])
        except:
            print('Error encoding '+feature)
    return df


# Define BasicModel function to evaluates the models
# displays the confusion matrix and plot it [13] [14]
# displays the classification_report
def BasicModel(models,X_train,Y_train,X_test,Y_test):
    # Evaluate each model in turns
    for name, model in models:
        # Train the model
        modelfit = model.fit(X_train,Y_train)
        # Predict the response for test dataset
        Y_predict = modelfit.predict(X_test)
        # Store the accuracy in results
        results.append(metrics.accuracy_score(Y_test, Y_predict))
        # Store the model name in names
        names.append(name)
        # Print the prediction of test set
        print('On %s Accuracy is: %f ' % (name, metrics.accuracy_score(Y_test, Y_predict)*100))
        # Store the name and accuracy in basic_score list
        basic_score.append({"Model Name": name, "Accuracy": metrics.accuracy_score(Y_test, Y_predict)*100})
        # Print Confusion Matrix and Classification Report
        print(confusion_matrix(Y_test, Y_predict))
        print(classification_report(Y_test, Y_predict))
        # Plot Confusion Matrix [13] [14]
        cm = confusion_matrix(Y_test, Y_predict, labels=modelfit.classes_)
        cmdisp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=modelfit.classes_)
        cmdisp.plot()
        plt.show()
    return basic_score

# Define BuildModelRS function to evaluate the models based on the random states [8-12]
# rand_state variable has been defined in the config.py file with values 1,3,5,7
# stores the Model Name, Random State and Accuracy of each model in score list
def BuildModelRS(models,rand_state,dfcreditdata,dfcredittarget):
    # Evaluate each model in turn
    for name, model in models:
        # for loop will train and predict the decision tree model on different random states
        for n in rand_state:
            # The training set and test set has been splited using the feature and target dataframes with different random_state
            X_train, X_test, Y_train, Y_test = train_test_split(dfcreditdata, dfcredittarget, test_size=0.3, random_state=n)
            # Train Decision Tree Classifer
            modelfit = model.fit(X_train,Y_train)
            # Predict the response for test dataset
            Y_predict = modelfit.predict(X_test)
            # Store the accuracy in results
            results.append(metrics.accuracy_score(Y_test, Y_predict))
            # Store the model name in names
            names.append(name)
            # Store the Model Name, Random State and Accuracy into score list
            score.append({"Model Name": name, "Random State": int(n), "Accuracy": metrics.accuracy_score(Y_test, Y_predict)*100})
    return score

# Define function BuildModelBalCV to evaluate models on the balanced data and utilize cross validation
def BuildModelBalCV(models,X,Y):
    # Evaluate each model in turn
    for name, model in models:
        # Define StratifiedKFold [17] [18]
        skfold = StratifiedKFold(n_splits=10, random_state= None, shuffle=True)
        # Get the X and Y using StratifiedKFold
        skfold.get_n_splits(X,Y)
        # Evaluate each model with cross validation
        cv_results = cross_val_score(model, X, Y, cv=skfold, scoring='accuracy')
        model = model.fit(X,Y)
        # Store the accuracy in results
        results.append(cv_results)
        # Store the model name in names
        names.append(name)
        # Print the results
        print('On %s: Mean is %f and STD is %f' % (name, cv_results.mean()*100, cv_results.std()))
        # Store the Model Name, Mean and STD into score list
        score.append({"Model Name": name, "Mean": cv_results.mean()*100, "STD": cv_results.std()})
    return score
    return results
    return names

```

```python
# Define BuildFinalModel function to evaluate models with their hyper-parameters
def BuildFinalModel(models,X,Y):
    # Evaluate each model in turn
    for name, model in models:
        # Define StratifiedKFold
        skfold = StratifiedKFold(n_splits=10, random_state= None, shuffle=True)
        # Get the X and Y using StratifiedKFold
        skfold.get_n_splits(X,Y)
        # Evaluate each model with cross validation [21]
        cv_results = cross_val_score(model, X, Y, cv=skfold, scoring='accuracy')
        model = model.fit(X.values,Y)
        # Perform cross-validation and obtain predictions
        #predictions = cross_val_predict(model, X, Y, cv=5)        #predictions = model.fit(X, Y).predict(X)
        # Store the cross validationscore into results
        final_results.append(cv_results)
        # Store model name into names
        names.append(name)
        # Print the results
        #print('On %s: Mean is %f and STD is %f' % (name, cv_results.mean(), cv_results.std()))
        # Store the Model Name, Mean and STD into score list
        score.append({"Model Name": name, "Mean": cv_results.mean(), "STD": cv_results.std()})
        #print('predictions are ')
        #print(predictions)
    return model, score, final_results
    #return model, score, final_results, predictions
```

6. model.py file: This file has the data analysis, visualization and modelling process.
    a. Imported all the libraries.

config.py | functionlibrary.py | model.py | index.html | app.py

```python

################ STEP 1: IMPORTING THE NECESSARY LIBRARIES ###################

# Load all the libraries that will be utilized through the code below
import pandas as pd
#from pandas import read_csv
from matplotlib import pyplot
import seaborn as sns
# Import matplotlib.pyplot to draw and save plots
import matplotlib.pyplot as plt
# Import Decision Tree Classifier
from sklearn.tree import DecisionTreeClassifier
# Import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
# Import train_test_split function
from sklearn.model_selection import train_test_split
# Import cross_val_score function
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
# Import StratifiedKFold function
from sklearn.model_selection import StratifiedKFold
# Import GridSearchCV function
from sklearn.model_selection import GridSearchCV
# Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
# Import confusion_matrix function
from sklearn.metrics import confusion_matrix
# Import classification_report function
from sklearn.metrics import classification_report
# Import ConfusionMatrixDisplay function to plot confusion matrix
from sklearn.metrics import ConfusionMatrixDisplay
# Import render from graphviz to convert dot file to png
from graphviz import render
# Import RandomUnderSampler, SMOTE, Pipeline to balance the dataset
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline
#import config.py
from config import *
#import functionlibrary.py
from functionlibrary import *
import time

```

    b. Read the Credit Card Approval dataset “crx.csv” located in Dataset folder.

```
51
52  ######################## STEP 2: READING THE DATA ############################
53
54
55  # Calling ReadDataset function from functionlibrary.py file.
56  # It reads the dataset file [1] and store it in a dataframe along with its attributes
57  # location_of_file is defined in the config.py file
58  dfcredit = ReadDataset(location_of_file)
59  print(dfcredit.head(5))
60  # Print the number of rows and columns present in dataset
61  print("The dataset has Rows {} and Columns {} ".format(dfcredit.shape[0], dfcredit.shape[1]))
62
63  # Give a new line to clearly format the output
64  Newline()
65
66  # Output:
67   #   A1     A2     A3 A4 A5 A6 A7    A8 A9 A10  A11 A12 A13    A14  A15 A16
68   # 0  b  30.83  0.000  u  g  w  v  1.25  t   t    1   f   g  00202    0   +
69   # 1  a  58.67  4.460  u  g  q  h  3.04  t   t    6   f   g  00043  560   +
70   # 2  a  24.50  0.500  u  g  q  h  1.50  t   f    0   f   g  00280  824   +
71   # 3  b  27.83  1.540  u  g  w  v  3.75  t   t    5   t   g  00100    3   +
72   # 4  b  20.17  5.625  u  g  w  v  1.71  t   f    0   f   s  00120    0   +
73   # The dataset has Rows 690 and Columns 16
74
```

c. Pre-processed the dataset by verifying the missing values, removing the special character “?”, converting the datatypes of variables and encoding the character values to int values.

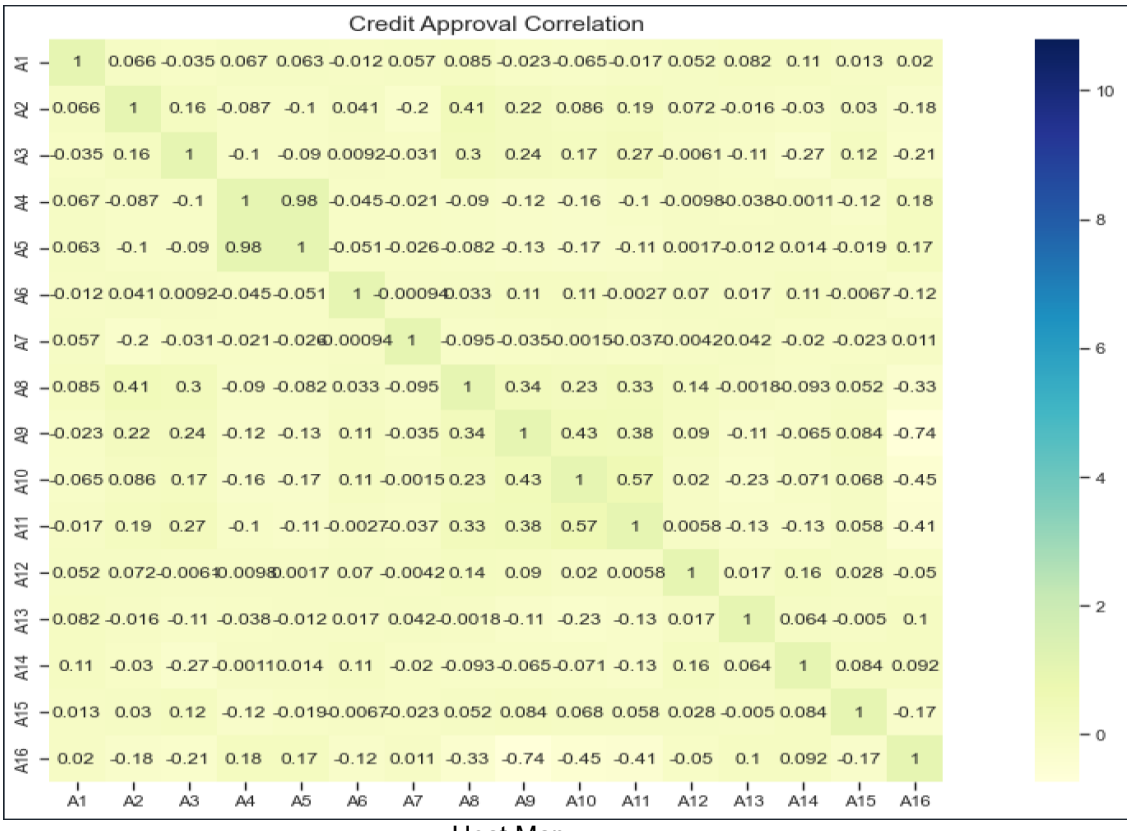
```
75  ###################### STEP 3: PRE-PROCESSING THE DATA ########################
76
77  #======================== 3.1. Understand the data ============================
78
79  # Summarizing the data statisticaly by getting the mean, std and other values for all the columns
80  print(dfcredit.describe())
81
82  # From the output it is clear that the mean, std and other analysis are calculated
83  # only on columns A3, A8, A11, A15, which specifies that other columns are having
84  # some unnecessary data.
85  # To get rid of missing values, Step 3.2 is performed
86
87  # Give a new line to clearly format the output
88  Newline()
89
90  #                A3           A8          A11           A15
91  #count  690.000000  690.000000  690.00000     690.000000
92  #mean     4.758725    2.223406    2.40000    1017.385507
93  #std      4.978163    3.346513    4.86294    5210.102598
94  #min      0.000000    0.000000    0.00000       0.000000
95  #25%      1.000000    0.165000    0.00000       0.000000
96  #50%      2.750000    1.000000    0.00000       5.000000
97  #75%      7.207500    2.625000    3.00000     395.500000
98  #max     28.000000   28.500000   67.00000  100000.000000
99
100 #======================== 3.2. Find the missing values ========================
101
102 # Printing the missing values (NaN)
103 print(dfcredit.isnull().sum())
104
105 # Give a new line to clearly format the output
106 Newline()
107
108 # Output:
109 #A1     0
110 #A2     0
111 #A3     0
112 #A4     0
113 #A5     0
114 #A6     0
115 #A7     0
116 #A8     0
117 #A9     0
118 #A10    0
119 #A11    0
120 #A12    0
121 #A13    0
122 #A14    0
123 #A15    0
124 #A16    0
125 #dtype: int64
126
127 # The output shows that there are no missing values (NaN) avaliable in the data
128
129 # To get a clear view of this kind of data, dataset is verified manually and found that
130 # there are "?" special character specified in many columns
131 # Therefore Step 3.3 will find the rows having "?" and drop the respective row from the dataset
132
133 #================== 3.3. Find and drop unnecessary values ====================
134
135 # Finding the special character "?" in the dataset and droping the rows, to get a clean dataset [4]
136
137 # Calling RemoveSpecialChar function from functionlibrary.py file to remove all the special characters from dataset
138 dfcredit = RemoveSpecialChar(special_char,dfcredit)
139
140 # Summarizing the dataset by printing the number of rows and columns present in dataset
141 print("The updated dataset has Rows {} and Columns {} ".format(dfcredit.shape[0], dfcredit.shape[1]))
142
143 # Give a new line to clearly format the output
144 Newline()
145
146 # Summarizing the data statisticaly again by getting the mean, std and other values for all the columns
147 print(dfcredit.describe())
148
149 # The output again shows that mean, std etc is done on attributes A3, A8, A11 and A15 only, instead of all the attributes
150
151 # Give a new line to clearly format the output
152 Newline()
153
154 # Verify the data types of all attributes in the dataset, to verify the difference between the datatypes
155 print(dfcredit.info())
156
157 # The information shows that the datatype of attributes are of int, float and object types
158 # The data needs to be converted to have consistency among all the attributes,
159 # because the dataset does not have a clear instruction of what all the attributes signifies to.
160 # Therefore, it becomes necessary to have the data of same date types
161
162 # Give a new line to clearly format the output
163 Newline()
164
```

```
#======= 3.4. Analysing and making consistent data type for all attributes ======

# Convert the column values from string/object into integer using labelencoder [5]

# In the dataset, there are mix the data types, such as values as string, float and integer
# To have same datatype, created Encoder function to convert the strings into integer values

# Calling the Encoder function passing the dataframe as parameter that creats a new dataframe to store the new values
dfcredit = Encoder(dfcredit)

# Save the new dataframe into a csv file
df =  pd.DataFrame(dfcredit)
df.to_csv('cleancreditdata.csv')

# Give a new line to clearly format the output
Newline()

# Summarizing the data statisticaly again by getting the mean, std and other values on all the columns
print(dfcredit.describe())
print(dfcredit.info())
# Now the count, mean, std, min, max etc are shown for all attributes, this means the data is clean now

# Give a new line to clearly format the output
Newline()

#======================== 3.5. Class distribution ============================

# As mentioned in the dataset details, "A16" attribute is the class attribute [1]

# Finding the number of rows that belong to each class
print(dfcredit.groupby('A16').size())

# Output is showing that the class "0" is having 296 rows and class "1" is having 357 rows
# By this we understand that there are only 2 classes 0 and 1 available in the data
# which means this is a Binary Classrification type.
# This also show the 0 is assigned to denied requests for applications asking credit card approval
# and 1 is assigned for approved requests for applications asking credit card approval
# This also states that the data is imbalanced.
# To understand the dataset more, Data Visualization by ploting the pair plot and heat graph is done in next step

# Give a new line to clearly format the output
Newline()
```

d. Visualised the data to understand the correlation between variables by creating Pair plot and heat map as displayed below.

```
##################### Step 4: DATA VISUALIZATION ##########################

#========================= 4.1. Plot Pairplot ============================

# To visualize the data, pair plot has been demonstrated below [6]

# Clear the plot dimensions
plt.clf()

# Plot the multivariate plot using pairplot
sns.set(style="ticks", color_codes=True)
grh = sns.pairplot(dfcredit,diag_kind="kde")
plt.show()

# Give a new line to clearly format the output
Newline()

# Wait for the graph to be displayed
time.sleep(30)

# The plots suggests that there is a high correlation and a predictable relationship
# between the diagonal grouping of some pairs of attributes
# As stated earlier that the data is imbalanced, the graph is not clearly linear too.

#========================= 4.2. Plot Heat map ============================

# To visualize the data again, heat map is demonstrated below [7]

# Clear the plot dimensions
plt.clf()

# Plot Heat graph to visualise the correlation
plt.figure(figsize=(20,10))
matrix1=dfcredit.corr()
plt.title('Credit Approval Correlation', size = 15)
sns.heatmap(matrix1, vmax=10.8, square=True, cmap='YlGnBu', annot=True)

# Save the pairplot in png file
#plt.savefig("heatplot.png")

# Give a new line to clearly format the output
Newline()

# Wait for the graph to be displayed
time.sleep(30)

# As observed earlier in the pairplot, the data has high correlation and a predictable relationship
# between the diagonal grouping of some pairs of attributes.
# The same has been observed in Hear map as well.
# To fix the imbalance problem, I have used the balancing technique RandomUnderSampler and SMOTE
# in the further steps after checking the performance of Decision Tree and Linear Discriminent Analysis models
# using the basic HoldOut Validation (i.e. train and test split)
```

Pair Plot



Heat Map

e. Developed 2 machine learning algorithms (Decision Tree Classifier and Linear Discriminant Analysis) to compare between the models based on its accuracy.

```
######################## STEP 5: FEATURE SELECTION ############################

# To implement any model on the data, we need to do feature selection first
# and therefore the dataset is devided into 2 parts.
# Firstly features (all the attributes except the target attribute) and secondly target (class attribute).
# The dfcredit dataframe has been split into dfcredittarget having only column A16, as this is the class attribute,
# and all the other attributes are taken into dfcreditdata dataframe as features

dfcreditdata = dfcredit.drop("A16", axis=1) # Features
dfcredittarget = dfcredit.iloc[:,-1]    # Target

# None of the feature selection methods are used here because the dataset is having very less number of attributes.
# and therefore, the dimentionality reduction has not been performed here.
# Also, the attirbute names have been changed to meaningless by the author [1], hence all the attributes are considered as features

# $$$$$$$$$$$$$$$$$$$$$$ BUILDING DECISION TREE CLASSIFIER AND LINEAR DISCRIMINENET ANALYSIS MODELS $$$$$$$$$$$$$$$$$$$$$$$$$$

############## STEP 6: BUILD BASE MODELS WITH HOLDOUT VALIDATION ##################

# Base Model is built below using the HoldOut Validation (train test split) to evaluate the accuracy

# ============== 6.1. Split the data to train and test set ================

# Spot Check Algorithms with Decision Tree [8] [9] [10] and Linear Discriminant algorithms [11] [12]

# Data is splited into training and test set using the feature and target dataframes
# to understand the model performance with 70% training set and 30% test set
X_train, X_test, Y_train, Y_test = train_test_split(dfcreditdata, dfcredittarget, test_size=0.3, random_state=None)

# ======================== 6.2. Define models ============================

# Clear lists
ClearLists()

# Create model object of Decision Tree Classifier [8] [9] [10]
models.append(('DTCLS', DecisionTreeClassifier()))
# Create model object of Linear Discriminent Analysis [11] [12]
models.append(('LDA', LinearDiscriminantAnalysis()))

# ================== 6.3. Build Model and Evaluate it  ==================

# Calling BasicModel function
BasicModel(models,X_train,Y_train,X_test,Y_test)

# Create a dataframe to store accuracy
dfbasicscore = pd.DataFrame(basic_score)
print(dfbasicscore.head())

# Give a new line to clearly format the output
Newline()

# Output:
    # On DTCLS Accuracy is: 81.632653
    # On LDA Accuracy is: 85.204082
# The output shows that the accuracy for LDA model is better than Decision Tree Classifier.
# To improve the performance of each model, the evaluation is done with respect to different random_state in the next step


################## STEP 7: BUILD MODELS WITH RANDOM STATE ####################

# Model is build with random state to evelaute the accuracy

# ====================== 7.1. Define models ================================

# Clear lists
ClearLists()

# Create model object of Decision Tree Classifier
models.append(('DTCLS', DecisionTreeClassifier()))
# Create model object of Linear Discriminent Analysis
models.append(('LDA', LinearDiscriminantAnalysis()))


# ================== 7.2. Build Model and Evaluate it  ==================

# Calling BuildModelRS to evaluate the models with random states and return the accuracy
BuildModelRS(models,rand_state,dfcreditdata,dfcredittarget)

# Create a dataframe to store accuracy
dfrsscore = pd.DataFrame(score)
print(dfrsscore.head(8))

# Give a new line to clearly format the output
Newline()

# The output shows that the accuracy changes in each randon state. Therefore, data has been balanced in further steps.
```

```
###################### STEP 8: BALANCING THE DATA ######################

# Installed imbalance-learn library using "conda install -c conda-forge imbalanced-learn" [15]

# To balance the data, RandomUnderSampler and SMOTE is utilized to undersample and oversample the data along with pipeline [16]

# Define oversmaple with SMOTE function
oversample = SMOTE()
# Define undersample with RandomUnderSampler function
undersample = RandomUnderSampler()
# Define Steps for oversample and undersample
steps = [('o', oversample), ('u', undersample)]
# Define the pipeline with the steps
pipeline = Pipeline(steps = steps)
# Fit the features and target using pipeline and resample them to get X and Y
X, Y = pipeline.fit_resample(dfcreditdata, dfcredittarget)
# Print the shape of X and Y
#print("The features dataset has Rows {} and Columns {} ".format(X.shape[0], X.shape[1]))
#print("The target dataset has Rows {} and Columns {} ".format(Y.shape[0],0))

# Give a new line to clearly format the output
Newline()

# By balancing the data using oversample and undersample, X and Y now have adequate amount of data avaialble
# Reducing the dimensionality is not needed because the dataset is small with only 15 features

######## STEP 9: OPTIMIZE MODEL ON BALANCED DATA USING CROSS VALIDATION #######

# StratifiedKFold Cross Validation is utilized to improve the performance as it splits the data approximately in the same percentage [17] [18]
# StratifiedKFold Cross Validation is used because there are 2 classes, and the split of train and test set could be properly done

# ========================== 9.1. Define models ==========================

# Clear lists
ClearLists()

# Create model object of Decision Tree Classifier
models.append(('DTCLS', DecisionTreeClassifier()))
# Create model object of Linear Discriminent Analysis
models.append(('LDA', LinearDiscriminantAnalysis()))

# ================== 9.2. Build Model and Evaluate it  ==================

# Calling BuildModelBalCV function to get the accuracy, cross validation results and names of models used
BuildModelBalCV(models,X,Y)
# Create a dataframe to store accuracy
dfscore = pd.DataFrame(score)
print(dfscore.head())

# Give a new line to clearly format the output
Newline()

# Compare Algorithms and plot them in boxplot
pyplot.clf()
pyplot.boxplot(results, labels=names)
pyplot.title('Algorithm Comparison')
pyplot.show()

# Give a new line to clearly format the output
Newline()

# Outout:
    # On DTCLS: Mean is 82.355243 and STD is 0.033680
    # On LDA: Mean is 87.255477 and STD is 0.040058

# LDA is having the highest Mean value of 87.25%.
# Decision Tree has also performed good with Mean as 82.35%
# Looking at the boxplot and the Mean values, it is clear that LDA has performed better even after balancing the data
# along with StratifiedKFold Cross Validation by giving Mean as 87.25%
```

f. Tuned the Decision Tree Classifier and LDA models with the hyperparameters to get the best parameters.

```
################### STEP 10: TUNE DECISION TREE BY GRIDSEARCHCV ###################

# Tuning the Decision Tree Classifier with GridSearchCV to find the best hyperparameter [19] [20]

# Define decision tree classifier model
dtcmodel = DecisionTreeClassifier()
# Define StratifiedKFold
skfold = StratifiedKFold(n_splits=10, random_state= None, shuffle=True)
# Define parameters
param_dict = dict()
param_dict = {"criterion": ["gini", "entropy"], "max_depth": [7,5,3]}
# Build GridSearchCV to get the accuracy
search = GridSearchCV(dtcmodel, param_dict, scoring='accuracy', cv=skfold, n_jobs=-1)
# Fit the GridSearchCV
results = search.fit(X, Y)
# Summarize
dt_accuracy = results.best_score_
dt_para = results.best_params_
print('Decision Tree Mean Accuracy: %f' % results.best_score_)
print('Config: %s' % results.best_params_)

# Give a new line to clearly format the output
Newline()

# Output:
    # Decision Tree Mean Accuracy: 0.865630
    # Config: {'criterion': 'entropy', 'max_depth': 3}
# The best parameter for Decision Tree Classifier are 'criterion' as 'entropy', 'max_depth' as 3,
# using which the performance of decision tree has been increased slightly with 86.56%
```

```python
###################### STEP 11: TUNE LDA BY GRIDSEARCHCV ######################

# Tuning the LDA with GridSearchCV to find the best hyperparameter [19]

# Define LDA model
ldamodel = LinearDiscriminantAnalysis()
# Define StratifiedKFold
skfold = StratifiedKFold(n_splits=10, random_state= None, shuffle=True)
# Define parameters
grid = dict()
grid['solver'] = ['svd', 'lsqr', 'eigen']
# Build GridSearchCV to get the accuracy
search = GridSearchCV(ldamodel, grid, scoring='accuracy', cv=skfold, n_jobs=-1)
# Fit the GridSearchCV
results = search.fit(X, Y)
# Summarize
lda_accuracy = results.best_score_
lda_para = results.best_params_
print('LDA Mean Accuracy: %f' % results.best_score_)
print('Config: %s' % results.best_params_)
#print(lda_accuracy)
#print(lda_para['solver'])

# Give a new line to clearly format the output
Newline()

# Output:
    # LDA Mean Accuracy: 0.871088
    # Config: {'solver': 'svd'}

# The best parameter LDA are 'solver' as 'svd'
# using which the performance of LDA model has been increased slightly to 87.10%

```

g. Created models with best hyperparameters along with cross validation method.

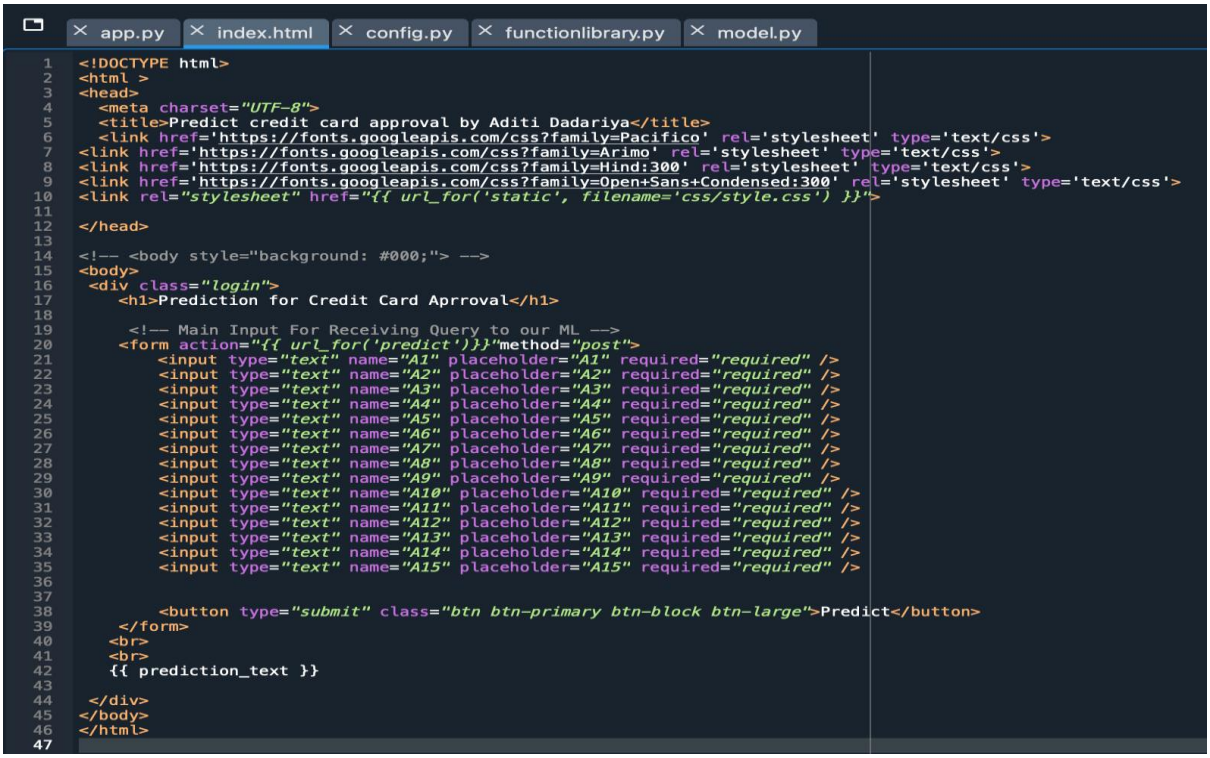
```python

######################### STEP 12: BUILD TUNED MODEL ##########################

# Building the model finally on balanced data with best hyper-parameters along with StratifiedKFold cross validation [21]

# ========================== 12.1. Define models ==============================

# Clear lists
ClearLists()

# Define models with the best hyperparameters found earlier
models.append(('DTCLS', DecisionTreeClassifier(criterion=dt_para['criterion'], max_depth=dt_para['max_depth'])))
models.append(('LDA', LinearDiscriminantAnalysis(solver=lda_para['solver'])))

# ==================== 12.2. Build Model and Evaluate it  =====================

# Calling BuildFinalModel to get the accuracy after tuning
BuildFinalModel(models,X,Y)
# Create a dataframe to store accuracy
dffinalscore = pd.DataFrame(score)
print(dffinalscore.head())

max_mean = dffinalscore['Mean'].max()
max_mean_indx = dffinalscore['Mean'].idxmax()
model_name = dffinalscore.loc[max_mean_indx, 'Model Name']

# Give a new line to clearly format the output
Newline()

print('%s has the max accuracy as %f' % (model_name, max_mean))

# Give a new line to clearly format the output
Newline()

# Compare Algorithms and plot them in boxplot
pyplot.clf()
pyplot.boxplot(final_results, labels=names)
pyplot.title('Algorithm Comparison')
pyplot.show()

# Give a new line to clearly format the output
Newline()

# Output:
    # On DTCLS: Mean is 84.178404 and STD is 0.035138
    # On LDA: Mean is 86.977700 and STD is 0.029924

# Looking at the Mean and Boxplot, it is clear that LDA has performed better than
# Decision Tree after tuning the model with hyperparameters using StratifiedKFold cross validation.
# However, there is not much difference in both the accuracies.

```

h. Trained the best performed model and dumped it in model.pkl file.

```python

######################## STEP 12: BUILD BEST PERFORMED MODEL #########################


# Clear lists
ClearLists()


# Define models with the best hyperparameters found earlier
if model_name == "DTCLS":
    models.append(('DTCLS', DecisionTreeClassifier(criterion=dt_para['criterion'], max_depth=dt_para['max_depth'])))
elif model_name == 'LDA':
    models.append(('LDA', LinearDiscriminantAnalysis(solver=lda_para['solver'])))

# Calling BuildFinalModel to get the prediction after tuning
model, score, final_results = BuildFinalModel(models,X,Y)


# $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$    MODEL.PKL FILE    $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

# Dump the model.pkl file
import pickle
pickle.dump(model, open('model.pkl', 'wb'))

```

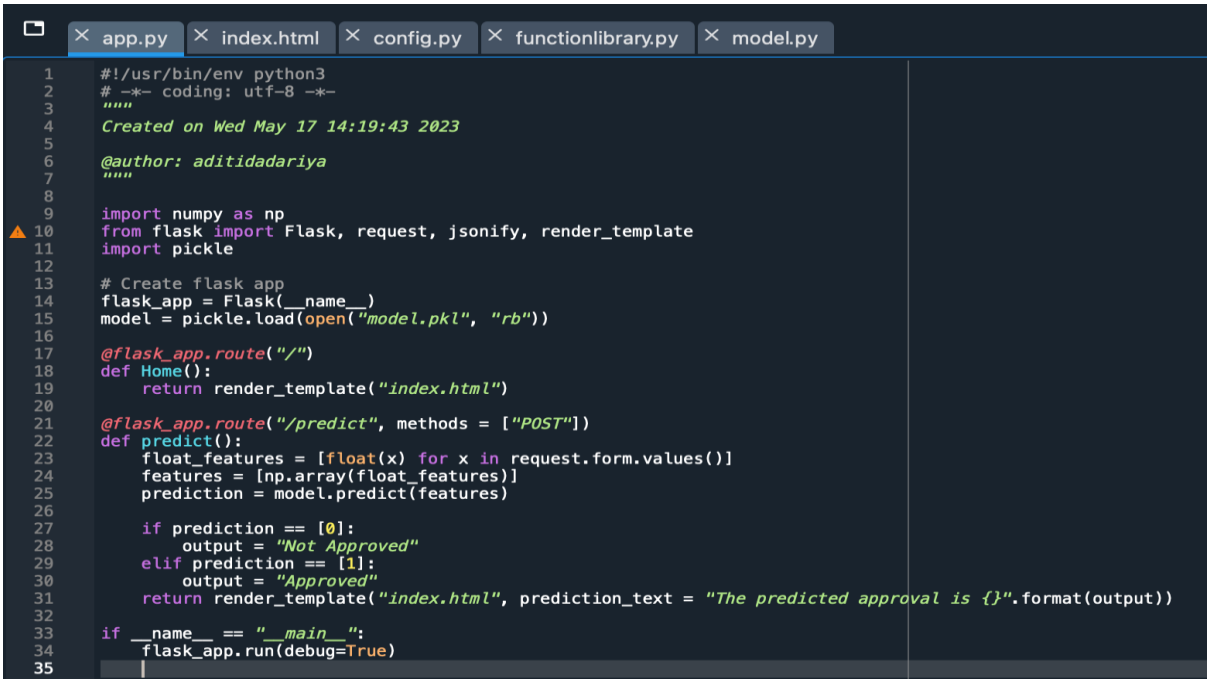
i. When model.py is run, a new model.pkl file is created.

7. index.html file: This file creates a web application with 15 text boxes and a “Predict” button.

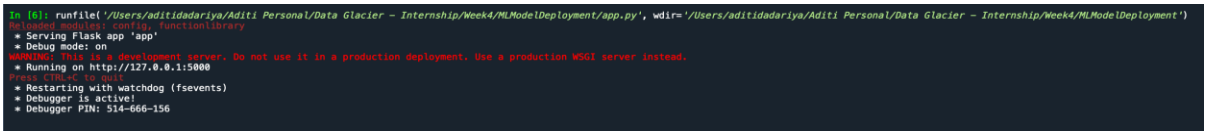


```
<!DOCTYPE html>
<html >
<head>
  <meta charset="UTF-8">
  <title>Predict credit card approval by Aditi Dadariya</title>
  <link href='https://fonts.googleapis.com/css?family=Pacifico' rel='stylesheet' type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Arimo' rel='stylesheet' type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Hind:300' rel='stylesheet' type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Open+Sans+Condensed:300' rel='stylesheet' type='text/css'>
<link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">

</head>

<!-- <body style="background: #000;"> -->
<body>
 <div class="login">
    <h1>Prediction for Credit Card Aprroval</h1>

      <!-- Main Input For Receiving Query to our ML -->
    <form action="{{ url_for('predict')}}"method="post">
        <input type="text" name="A1" placeholder="A1" required="required" />
        <input type="text" name="A2" placeholder="A2" required="required" />
        <input type="text" name="A3" placeholder="A3" required="required" />
        <input type="text" name="A4" placeholder="A4" required="required" />
        <input type="text" name="A5" placeholder="A5" required="required" />
        <input type="text" name="A6" placeholder="A6" required="required" />
        <input type="text" name="A7" placeholder="A7" required="required" />
        <input type="text" name="A8" placeholder="A8" required="required" />
        <input type="text" name="A9" placeholder="A9" required="required" />
        <input type="text" name="A10" placeholder="A10" required="required" />
        <input type="text" name="A11" placeholder="A11" required="required" />
        <input type="text" name="A12" placeholder="A12" required="required" />
        <input type="text" name="A13" placeholder="A13" required="required" />
        <input type="text" name="A14" placeholder="A14" required="required" />
        <input type="text" name="A15" placeholder="A15" required="required" />


        <button type="submit" class="btn btn-primary btn-block btn-large">Predict</button>
    </form>
   <br>
   <br>
   {{ prediction_text }}

 </div>
</body>
</html>
```

8. app.py file: This file runs index.html to open a web application, uses model.pkl file to predict the output and display the output in web application. The web application is at “http://127.0.0.1:5000/”

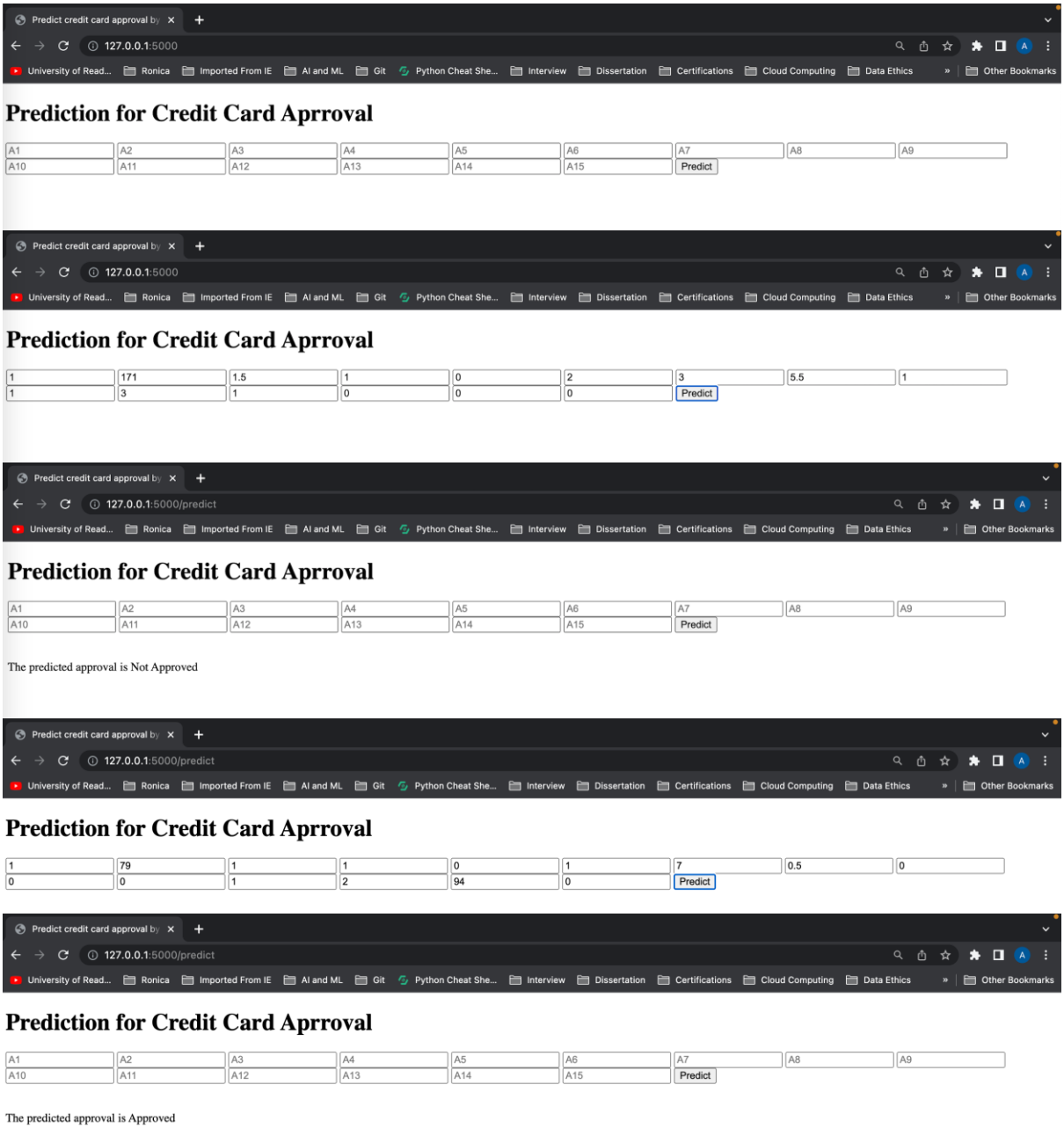


```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed May 17 14:19:43 2023

@author: aditidadariya
"""

import numpy as np
from flask import Flask, request, jsonify, render_template
import pickle

# Create flask app
flask_app = Flask(__name__)
model = pickle.load(open("model.pkl", "rb"))

@flask_app.route("/")
def Home():
    return render_template("index.html")

@flask_app.route("/predict", methods = ["POST"])
def predict():
    float_features = [float(x) for x in request.form.values()]
    features = [np.array(float_features)]
    prediction = model.predict(features)

    if prediction == [0]:
        output = "Not Approved"
    elif prediction == [1]:
        output = "Approved"
    return render_template("index.html", prediction_text = "The predicted approval is {}".format(output))

if __name__ == "__main__":
    flask_app.run(debug=True)
```



```
In [6]: runfile('/Users/aditidadariya/Aditi Personal/Data Glacier - Internship/Week4/MLModelDeployment/app.py', wdir='/Users/aditidadariya/Aditi Personal/Data Glacier - Internship/Week4/MLModelDeployment')
Reloaded modules: config, functionlibrary
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with watchdog (fsevents)
 * Debugger is active!
 * Debugger PIN: 514-666-156
```

9. Web application is below with the inputs for credit cards application provided and the output of prediction displayed.



10. References:
[1] https://archive.ics.uci.edu/ml/datasets/Credit+Approval