Machine Learning

Milestone 2

Project Name: Hotel Rating

(Team ID: SC_22)

Create a 3 files.py

1. preprocessing_Data.py: - to create all functions used in preprocessing and scaling and outliers.
2. Classification_train.py: - to read and split and preprocess and fit models about data.
3. Classification_test File.py: - to read test data and preprocessing and scaling and read models from file and predict output to all models.

preprocessing_Data.py

1. **Import Libraries**
2. Fill Null Values in train set : -

   Create a function **'fill_Null_train_classi(data)',** which work as follow: -

   1) calculate **missing values in each column'mask_F'** this equal 'data. Isnull (). any(axis=0)'.
   2) calculate **missing values in each row 'mask_R'** this equal 'data. Isnull (). any(axis=1)'.
   3) calculate **percentage of each row with na_values 'p_rows_with_nan'** this equal 'mask_R. sum()/len(data)'.
   4) calculate **percentage of each feature with na_values 'p_features_with_nan'** this equal 'mask_F.sum()/len(data)'.
   5) check **if** percentage of each row with na_values 'p_rows_with_nan' smaller than 0.07 then we deal with data without these rows 'in other hand we delete the rows have na_values.'
   6) **else** we first arrange these features by calculate mean to all of these ('lat', 'lng', 'avarage_score', 'Total_Numbers_of_Reviews') cause the 'Total_Numbers_of_Reviews' depends on Total number of valid reviews the hotel has, and the 'avarage_score' depends on the latest comment in the last year of this hotel and we store in x16 most

frequently value in'Reviewer_Score'(this value is '**HHigh_Reviewer_Score'**).

7) use **fillna function** which take column name and take value to replace Na_values with it, replace Na_values in 'Hotel_Address' column with 'no address' , and 'Additional_Number_of_Scoring' column with '0.0', and 'Review _Date' column with '00/00/0000', and 'Average_Score' column with mean, and 'Hotel_Name' column with 'no name', and 'Reviewer_Nationality' column with 'no info', and 'Negative_Review' column with 'Positive', and 'Review_Total_Negative_Word_Counts' column with '0.0', and 'Total_Number_of_Reviews' column with mean, and 'Positive_Review' column with 'Negative', and 'Review_Total_Positive_Word_Counts' column with '0.0', and 'Total_Number_of_Reviews_Reviewer_Has_Given' column with '0.0', and 'Tags' column with "['no trip', 'no Couple', 'no Room', 'Stayed 0', 'not Submitted']",and 'days_since_review' column with '0 days', and 'lat' column with mean, and 'lng' column with mean, and 'Reviewer_Score' column with 'x16'.
8) Defined data frame 'values' to restore these values in file ('lat'. mean, 'lng'. mean, 'avarage_score'. mean, 'Total_Numbers_of_Reviews'. Mean, 'Reviewer_Score' with x16),

after that return result.

3. Fill Null Values in Test Set

Create a function **'fill_Null_test_classi(data)',** which work as function **'fill_Null_train_classi(data)'** but it has a simple difference: - **open file** valuesOfClassi to read the contain data in it (mean of columns ['lat', 'lng', 'avarage_score', 'Total_Numbers_of_Reviews','Reviewer_Score']) after reading this data continue by this data.

4. Split Date

Create a function **'split_date(data)',** which work as follow: -

1) **split the column of 'Review_Date'** by any sign like "/, \, -", and put result in 'day & month & year' by creating data frame include 3 columns "'Day', 'Month', 'Year' ", and convert data from string to integer then we drop the column of 'Review_Date'.
2) **split the column of 'date_since_review'** by putting the column in a list called 'days' and make an empty list called 'day' then we start to iterate on first list to remove word 'day/s' by replacing it with empty space and then put the data of old list in 'day' list, after that start to convert the data of the 'day' list from string to integer and put it in another list 'res' then we make column 'date_since_review equals to 'res list',

after that return result.

5. Split Tags Columns

Create a function **'split_Tags (SplitDate)',** which work as follow: -

1) Then we **split the 'Tags'** column first create 5 empty lists called ('Trip' ,'Memories', 'Room_Kind', 'Nights', 'the_way_of_submission') and list called 'Tags' and put 'Tags' column in it .
2) after that we **iterate on 'Tags'** list and create another list called 'tag' and we replace each "[,],' " for list tags with empty space then we split each time we find ',' and append in 'tag' list then we loop on 'tag' list and start to create 5 Boolean variables called ('trip', 'mem', 'room', 'night', 'submission') with initial values False, then we create a loop start from k to iterator of outer loop **if** k (each part of list 'tag') contain 'trip' put this in list 'Trip' and make Boolean trip equals true, **else if** k contains 'room' or 'Room' put this in list 'Room_Kind' and make Boolean room equals true,

else if k contains ‘Stayed’ put this in list ‘Nights’ and make Boolean night equals true,

else if k contains ‘Submitted’ put this in list ’the_way_of_submission’ and make Boolean submission equals true,

else if k contains ‘Couple’ or ‘Group’ or ‘children’ or ‘traveler’ put this in list ‘Members and make Boolean mem equals true, and contains the outers loop to check **If** (not trip)

put in the list Trip ‘trip’,

else if (not room) put in the list Room_Kind ‘no Room’, else if (not night) put in the list Nights ‘Stayed 0’,

else if (not submission) put in the list the_way_of_submission ‘not Submitted’,

else if (not mem) put in the list Members ‘no Couple’.

3) Then we **create 5 columns** in the data called

   Trip to put in its ‘Trip’ list, **Member** to put in it ‘Members’ list, **Room** to put in it ‘room_kind’ list, **Submission** to put in it ‘the_way_of_submission’ list.

4) Then we create empty list called ‘numbers’ and **iterate on ‘Nights’** list until its length Split data and take only number from it and put this number on list called ‘converted number.’, then create a list called ‘convert’ to put in it the number from ‘converted number’ after converting it to integer, then append ‘convert’ list in ‘number’ list then **create a column called ‘Nights’** in data and make it equals ‘numbers’ list, after all of that we **drop the ‘Tags’** column,

after that return result.

6. Label Encoder to Convert Data to Numerical

Create a function **‘Feature_Encoder_classi (data)’,** which work as follow: -

1) 1- **Defined object** from label encoder
2) start with ‘Hotel_Address’ column and make **fit transform** it , ‘Hotel_Name’ column and make fit transform it,

‘Reviewer_Nationality’ column and make fit transform it,'Trip' column and make fit transform it, ‘Members’ column and make fit transform it , ‘Room’ column and make fit transform it, ‘Submission’ column and make fit transform it, ‘Negative_Review ‘column and make fit transform it, ‘Positive_Review’ column and make fit transform it,'Reviewer_Score' column and make fit transform it.

3) Start to **arrange our data** by creating data frame called ‘data_num’ and give it the numbers of columns as we need it to be arranged, to make data arrange as past and make the ‘Reviwer_Score’ column at the end, **defined 2 data frames ‘X’,’Y’**, X read all the columns expect the ‘Reviwer_Score’ column, and Y read ‘Reviwer_Score’ column,

after that return result.

7. Feature Scaling

1. Normalizing Data

- Create a function called ‘Normalization’ which take (data)
  1- create object from ‘MinMaxScaler’.
  2- fit the data and transform it and put it in data frame called ‘Normalized_x’.
  3- return’Normalized_X’.

2. Standardize Data.

- Create a function called ‘Standardization.’ Which take (data)
  1- create object scaler from ‘StandardScaler’.
  2- fit the data and transform it and put it in data frame called ‘x_scaled’.
  3- return ‘x_scaled’.

8. Box Plot

Create a function called **‘plot_boxplot’** which take data and column which I want to apply on it, then we use df.boxpolt and give it the column I want draw on it.

9. Outliers

1- Create function **‘delete_outlier (data, name, upper_limit, lower_limit)’** to delete outliers,

- create data frame called ‘update_data’ and put on it data of column if data of column less than or equals upper limit and bigger than or equals lower limit
- after that return update_data.

2- Create function **‘Norm_outlier (data, name, upper_limit, lower_limit)’** to arrange the ranges of outlier,

- create data frame called ‘new_data’ which copy data on it
- check if ‘new_data’ of name bigger than ‘upper_limit’ make it ‘upper_limit.’
- else make it ‘lower_limit’.
- After that, return ‘new_data’.

3- Create function **‘outliers(data)’** to detect outliers.

- calculate the length of data.
- iterate until length.
- create var called name carry columns name.
- calculate Q1 = data[name]. quantile (25%).
- Q3 = data[name]. quantile (75%).
- calculate IQR = Q3-Q1.
- make upper_limit =q3+1.5*IQR.
- lower_limit =q1-1.5*IQR

after all of this

-**If we Want to delete outliers**:

- We call 'delete_outlier' and put the result of it in data frame called data.
- Then return data.

-**If we want to arrange the ranges of outlier**:

- We call 'Norm_outlier' and put the result of it in data frame called data.
- Then return data.

4- Finally, we use z-score technique:

We create function called **'z-score(data)',** which take data to work on as follow: -

- First, we get the length of data columns and store it in variable called n.
- start to iterate on this n and make variable name equals to data. columns[i] and upper_limit=data[name].mean() +3*data[name]. std (), and lower_limit=data[name]. mean()-3*data[name]. std (), then call Norm_outlire ()

  and give it data, name, upper_limit, lower_limit Store the output in data variable.
- return data.

Classification_train.py

1. Import Libraries

2. Read Data

- **Read our dataset** from the csv file, after that we concatenate 'No Negative' and 'No Positive' in values of 'na_values', read all this in data frame **'hotel_data'**.

3. Split Data

1- defined 2 data frames 'X','Y',

- X → read all the columns in 'hotel_data' expect the 'Reviwer_Score.' column.
- Y → read 'Reviwer_Score' column.

2- Split X, Y by 'train_test_split' in **80% to train set** and **20% to test set** return splits in **(X_train, X_test, Y_train, Y_test).**

3- Create a 2 data frame: -

- Data frame called **'data'** to concatenate ('X_train', 'Y_train') in it.
- Data frame called **'data_test'** to concatenate ('X_test', 'Y_test') in it.

4. Fill Null Values in Training Set

-Defined data frame **'data_fill_null'** to return output of function **'fill_Null_train_classi (data)'.**

5. Split Date

- Defined data frame '**SplitDate**' to return output of function '**split_date (data_fill_null')**'.

6. Split Tags Columns

-Defined data frame **'data_processed'** to return output of function **'split_Tags (SplitDate)'**

7. Label Encoder to Convert Data to Numerical

-Defined 2 data frame **'X', 'Y'** to return output of function **'Feature_Encoder_classi (data_processed)**'

8. Feature Scaling

-Defined data frame called **'X'** equals called 'Standardization.' function and pass parameters to it(X).

-Then create 2 data frames called 'dff' and 'dff1.'
dff equals X after are scaling it, dff1 equals Y.

- merge the 2 data frames into **'df_merged'** data frame by concatenate dff and dff1.

9. Outliers

- Defined data frame **'data_cleaned'** to return output of function **'Outliers (df_merged)**'.

- **defined 2 data frames 'X','Y',**

  - X → read all the columns in **'data_cleaned'** expect the 'Reviwer_Score.' column.
  - Y → read 'Reviwer_Score' column.

10. Feature Selection

- **F-classify Method**
  - use built in function called 'SelectKBest (score_func=f_clssif, k=7) 'It returns the best 7 features of data in variable called fs The we are applying fit on fs.
  - Using fit (X, Y. values. ravel ()) function Then we create variable selected_features and store in it the return of X. columns [fs.get_support ()]
  - After that print selected_features.
  - make X=X[selected_features]
  - open selected_features file and store on it the output.

11. Models

1). logistic regression Model

- We start by calculate the time so at the we use ‘start_fT_logistic = time. Time ()’.
- Create object (Logistic) from ’Logistic Regression (random _state=0,
- Fit X, Y=>Logistic. Fit(X, Y).
- We open the ' logistic regression model.pkl' and store on it the object f.
- Then creating a logistic classifier object called ‘logistic’.
- Start to apply gridsaerchview creating a list called ‘grid_log’ which contain ‘C’ and ‘penalty’.
- Then creating a KNN classifier object called ‘knn’.
- Perform the grid search object to search over the hyperparameter grid: create object called ‘logreg_cv’ equals GridSearchCV (logistic, grid_log, CV=5).
- Fit X, Y=> logreg_cv. Fit(X, Y). (Fit the GridSearchCV object to the data).
- We open the’ logistic regression model by grid.pkl’ and store on it the object f.
- End the calculation of time to know the time of fitting: end_fT_logistic = time.time ()
  time_fit_of_logitic = (end_fT_logistic - start_fT_knn) / 60.

2). naive bayes Model

- Create object called ‘naive_bayes’ from GaussianNB()
- Fit X, Y => naive_bayes. Fit(X, Y).
- We open the file lasso regression and store on it the object lasso.
- Create object (Knn) from ‘KNeighborsClassifier (n_neighbors=11, p=2, metric='Manhattan’, weights='distance')’.
- We start by calculate the time so at the we use ‘start_fT_knn = time. Time ()’.

- Create and object called ‘knnn’ equals KNeighborsClassifier ().
- Fit X, Y=>knnn. Fit(X, Y).
- We open the 'naive bayes model.pkl' and store on it the object f.
- Start to apply gridsaerchview :(Define the hyper parameter grid to search over #[3, 5, 7]) :creating a list called ‘param_grid_Knn’ which contain ‘n_neighbors’, ‘weights’ and ‘metric'.
- Then creating a KNN classifier object called ‘knn’.
- Perform the grid search object to search over the hyperparameter grid: create object (grid_search_KNN) = GridSearchCV (knn, param_grid_Knn, CV=5).
- Fit X, Y=> grid_search_KNN. Fit(X, Y).(Fit the GridSearchCV object to the data).
- We open the’ KNN model by grid.pkl’ and store on it the object f.
- End the calculation of time to know the time of fitting:

  end_fT_knn = time.time ()

  time_fit_of_Knn = (end_fT_knn - start_fT_knn) / 60.

4). Decision tree Model

- We start by calculate the time so at the we use ‘start_fT_tree = time.
- Create object (DT) from ‘DecisionTreeClassifier (max_depth= 8, min_samples_leaf = 1, min_samples_split= 2)’.
- Fit X, Y=>DT. Fit(X, Y).
- We open the ‘Decision Tree model.pkl’ and store on it the object f.
- Start to apply gridsaerchview: creating a list called ‘param_grid_DT’ which contain max_depth, min_sampels_split and min_sampels_leaf.
- Then creating a decision tree classifier called ‘DT_grid’.
- Perform the grid search to find the optimal hyper parameters: create object (grid_search_DT) from ‘GridSearchCV (DT_grid, param_grid=param_grid_DT, CV=5)’.
- Fit X, Y=> grid_search_DT. Fit(X, Y).

- We open the' Decision Tree model grid.pk' and store on it the object f.
- End the calculation of time to know the time of fitting: end_fT_tree = time.time ()
  time_fit_of_DT = (end_fT_tree - start_fT_tree) / 60

5).Random Forest Model

- Create object (regressor) from 'RandomForestClassifier (n_estimators=100, max_depth=12)'.
- Make this object equals to RandomForestClassifier ().
- Fit X, Y=>regressor. Fit(X, Y).
- We open the 'Random forest model.pkl' and store on it the object f.

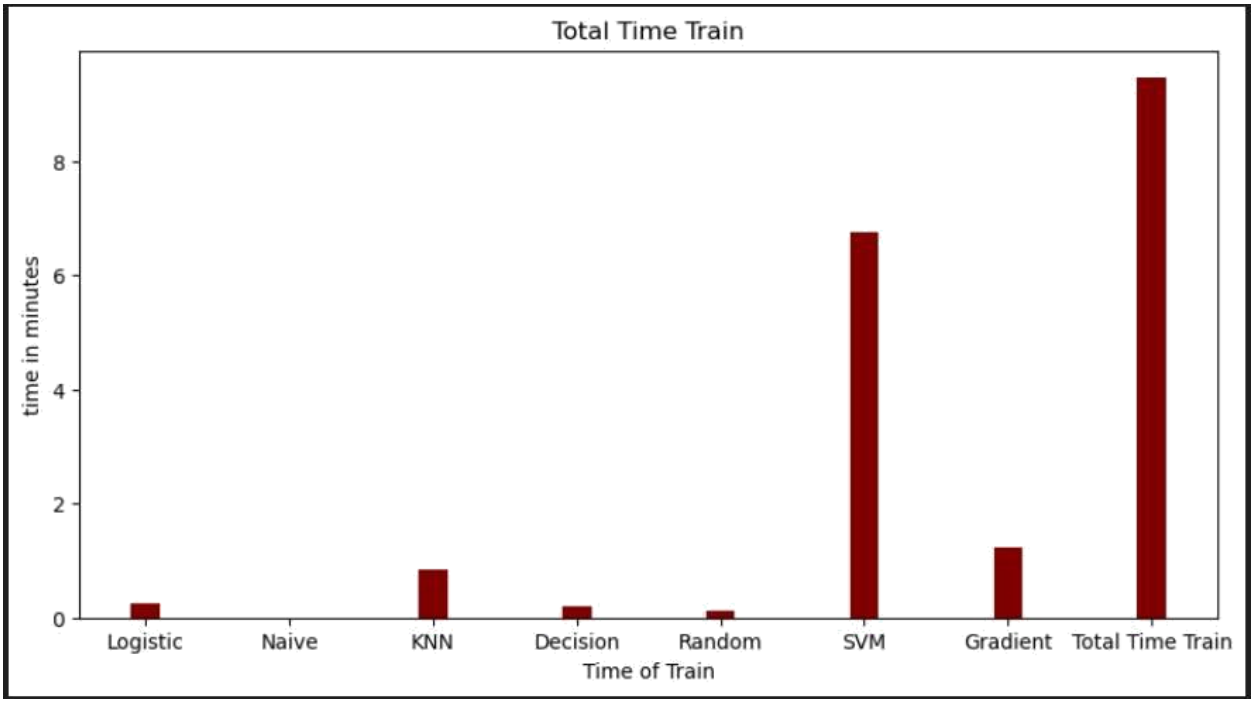
- Create object (svm_model) from 'SVC(C=10, kernel='linear', gamma=1)'.
- Fit X, Y=> svm_model. Fit(X, Y).
- We open the 'SVM model.pkl' and store on it the object f. 7). Gradient Boosting Model
- Create object (gbc) from 'GradientBoostingClassifier (n_estimators=300, learning rate=0.05, random state=100, max_features=5)'.
- Fit X, Y=> gbc. Fit(X, Y).
- We open the 'Gradient Boosting model.pkl' and store on it the object f.

12. Time Of Training

- The end time of training => end_time_train = time.time (). ○ time_train = (end_time_train - start_time_train) / 60.
- Create a dictionary called 'All_time' contain the total time of each model.
- Create a list called 'Time' and put in it the keys from 'All_time'.
- Create a list called 'Values' and put in it the Values from 'All_time'.
- Create object called 'fig' equals to plt.figure(figsize=(10, 5)).

13. creating the bar plot

- Creating bar graph between Time and Values: plt.bar(Time, values, color='maroon', width=0.2).
- Setting the labels of X and Y axis and the plot title:
  - plt.xlabel("Time of Train")
  -plt.ylabel("time in minutes")
  -plt.title("Total Time Train")



At the end we print:

-time of train.

- classification train is done.

Classification Testing File.py

- Import Libraries
- defined 2 data frames 'X','Y'
  - X → read all the columns in 'data-test' expect the 'Reviwer_Score.' column.
  - Y → read 'Reviwer_Score' column.
- Fill Null Values in Training Set

  -Defined data frame **'data_fill_null'** to return output of function **'fill_Null_test_classi (data_test)'.**

- Split Date

  - Defined data frame '**SplitDate**' to return output of function **'split_date (data_fill_null)'**.

- Split Tags Columns

  -Defined data frame **'data_processed'** to return output of function **'split_Tags (splitDate)'.**
- Label Encoder to Convert Data to Numerical

  -Defined 2 data frame **'X', 'Y'** to return output of function **'Feature_Encoder_classi (data_processed)**'.

- Feature Scaling

  -Defined data frame called **'X'** equals called 'Standardization.' function and pass parameters to it(X).

- Feature Selection
  - We open selected_features file to read data from it and load this data in a data frame called '**X'.**
- Models

  1) Logistic regression Model

- We start by calculate the time so at the we use ‘start_T_log = time. Time ()’.
- Open logistic regression model file to read data from it.
- Make predictions on the testing data.
- Calculate accuracy by accuracy_Knn = accuracy_score (y_pred_Kn, Y) * 100 and print it.
- Starting gridview by We open the ‘logistic regression model by grid.pkl’ and load data from it.
- Create object y_pred_log_G = logreg_cv.best_estimator_.predict(X)
- Calculate the accuracy => acc_log_grid = accuracy_score(Y, y_pred_log_G) * 100.
- At the end print the best parameter and the accuracy with the best model.
- End the calculation of time to know the time of fitting:
  end_T_log = time.time ()
  time_predict_of_logistic = (end_T_log - start_T_log) / 60.

2). naive bayes Model

- Open naive bayes model file to read data from it.
- Make predictions on the testing data.
  - y_pred = naïve_bayes. Predict(X).
- Calculate **accuracy** by accuracy = metrics. Accuracy_score (y_pred, Y) and then print it *100.

3)KNN Model

- We start by calculate the time so at the we use ‘start_T_Knn = time.
- Open KNN model file to read data from it.
- Make predictions on the testing data.
  - y_pred_kn = knnn. Predict(X).

- Calculate accuracy by accuracy_Knn = accuracy_score (y_pred_Kn, Y) * 100 and print it.
- Starting gridview by We open the 'KNN model by grid.pkl' and load data from it.
- Create object y_pred_KN_G = grid_search_KNN.best_estimator_.predict(X) To get the best KNN model.
- Calculate the accuracy => acc_Knn_grid = accuracy_score(Y, y_pred_KN_G) * 100
- At the end print the best parameter and the accuracy with the best model.
- End the calculation of time to know the time of fitting: -end_T_Knn = time.time() -Time_predict_of_Knn = (end_T_Knn - start_T_Knn) /60.

4)Decision Tree Model

- We start by calculate the time so at the we use 'start_T_tree = time.
- Open Decision Tree model file to read data from it.
- Make predictions on the testing data.
  - y_pred_T = DT. Predict(X).
- Calculate accuracy by accuracy_score (Y_pred, Y) then print this accuracy *100 and print it.
- Starting gridview by We open the 'Decision Tree model.pkl' and load data from it.
- Create object y_pred_DT_G = grid_search_DT.best_estimator_.predict(X) To get the best DT model.
- Calculate the accuracy => accuracy_DT_grid = accuracy_score(Y, y_pred_DT_G) * 100.

- At the end print the best parameter and the accuracy with the best model.
- End the calculation of time to know the time of fitting: end_T_tree = time.time()

  time_predict_of_DT = (end_T_tree - start_T_tree) / 60

5). Random Forest Model

- Open Random Forest object file to read data from it.
- Make predictions on the testing data.
  - y_pred = regressor. Predict(X).
- Calculate accuracy by accuracy_score (Y_pred, Y) then print this accuracy *100.

6). SVM Model

- Open SVM model file to read data from it.
- Make predictions on the testing data.
  - y_pred = svm_model. Predict(X).
- Calculate accuracy by accuracy_score (Y_pred, Y) then print this accuracy *100.

7). Gradient Boosting Model

- Open Gradient Boosting model file to read data from it.
- Make predictions on the testing data.
  - y_pred = gbc. Predict(X).
- Calculate accuracy by accuracy_score (Y_pred, Y) then print this accuracy *100.

8). Bar graph with grid search of each model

1- accuracy graph:

- set width of bar : barWidth = 0.1.
- Create object ‘fig’ equals plt.subplots(figsize=(8, 9)).
- set height of bars:

  -b1 = [acc_of_logistic, accuracy_Knn, accuracy_DT]

  -b2 = [acc_log_grid, acc_Knn_grid, accuracy_DT_grid]
- Set position of bar on X axis:

-br1 = np.arange(len(b1))

-br2 = [x + barWidth for x in br1]

- Make the plot as follow:

  -plt.bar(br1, b1, width=barWidth, edgecolor='grey', label='accuracy') -plt.bar(br2, b2, color='grey', width=barWidth, edgecolor='grey', label='accuracy after update')

- Putting labels:

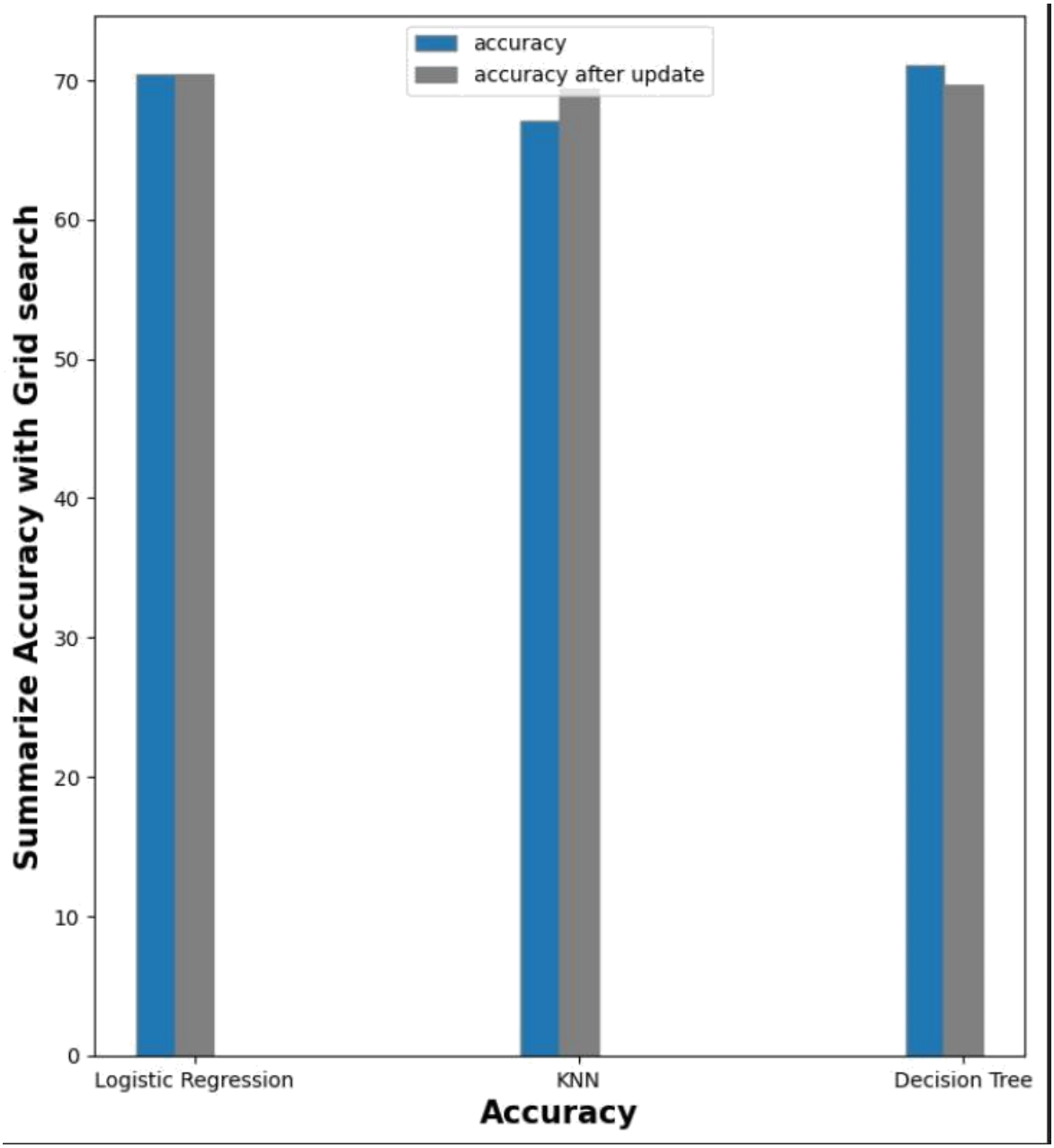
  - plt.xlabel('Accuracy', fontweight='bold', fontsize=15)

  -plt.ylabel('Summarize Accuracy with Grid search', fontweight='bold', fontsize=15)

  -plt.xticks([r + barWidth for r in range(len(b1))],['Logistic Regression', 'KNN', 'Decision Tree'])

- Then use
  - plt.legend() and plt.show() to display the plot.

- The end time of training => end_time_test = time.time()
- time_test = (end_time_test - start_time_test) / 60
- We print the time of test.

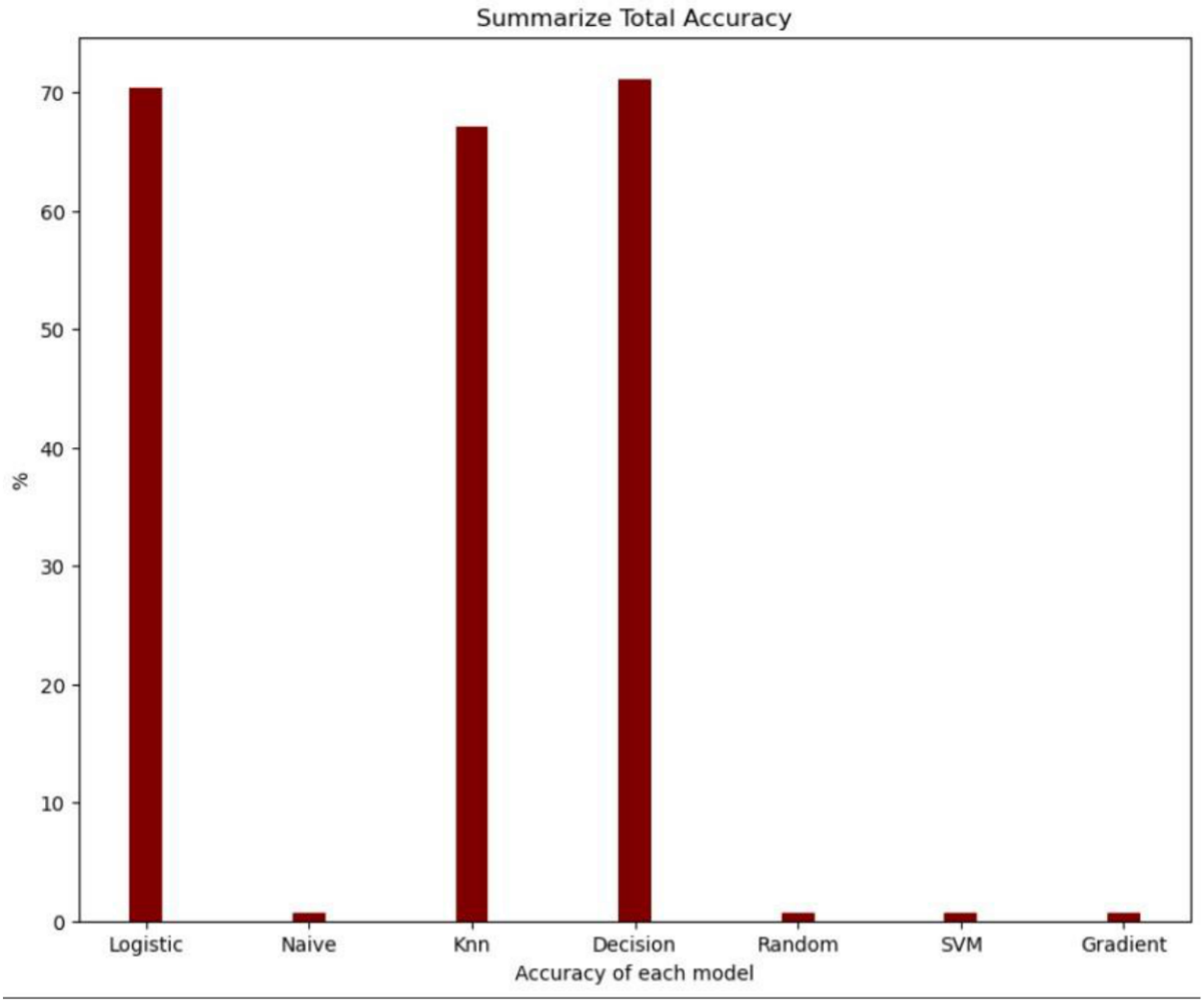


2- summarize total accuracy of all model graph

- Create a dictionary called ‘All_accuracy’ contain Accuracy of each model .
  - Create 2 lists ‘accuracy and values’ Accuracy carry the keys from ‘All_accuracy’ and Values carry the Values of it.
  - Create object fig equals plt.figure(figsize=(10, 8)).
  - Create the bar graph: plt.bar(accuracy, values, color='maroon', width=0.2).
  - Putting titles on graph:

    -plt.xlabel("Accuracy of each model") -plt.ylabel("%")

    -plt.title("Summarize Total Accuracy").



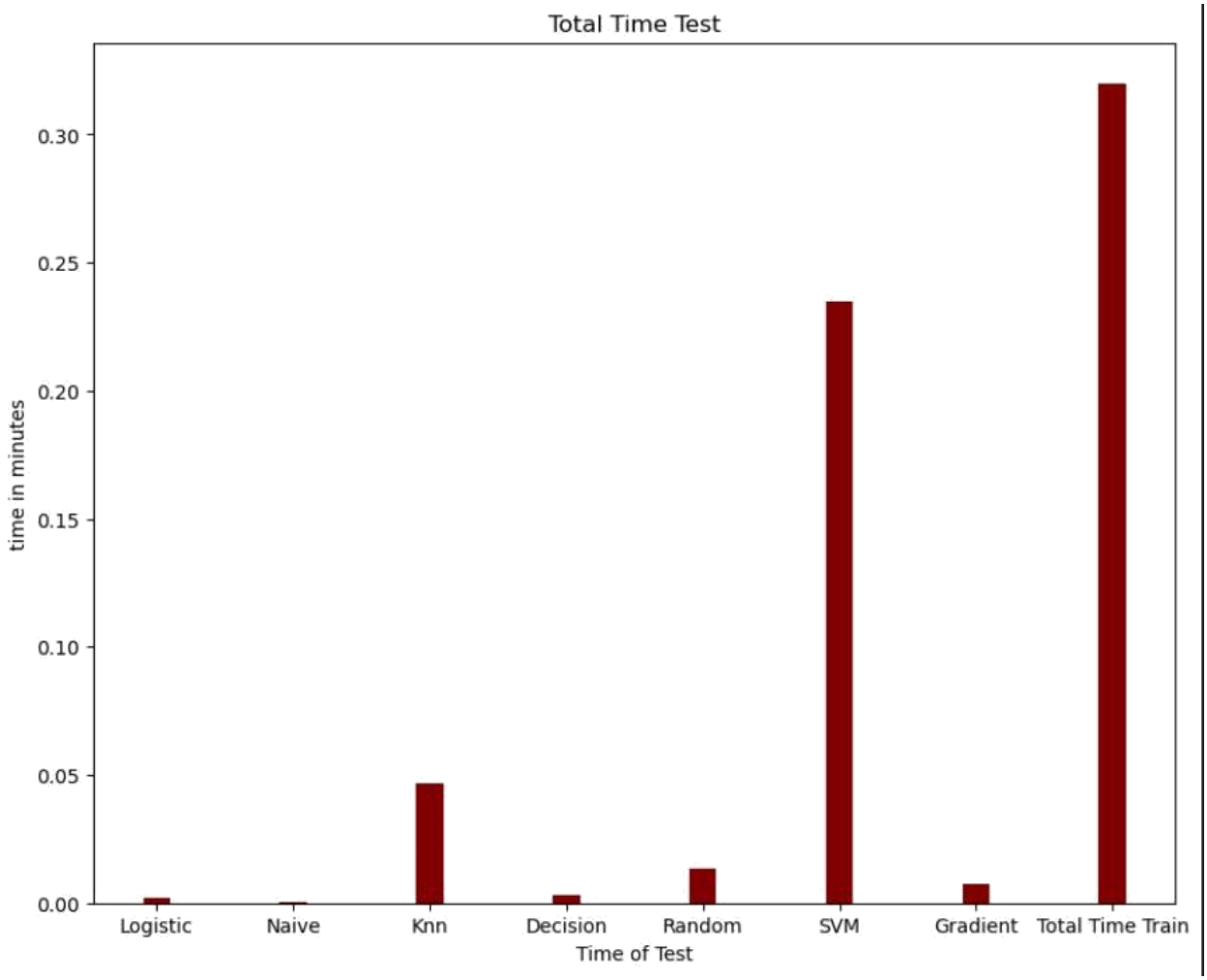
3) time of train graph

- Create a dictionary called ‘All_time contain predict time of each model .
- Create 2 lists ‘time and values’ Accuracy carry the keys from ‘All_accuracy’ and Values carry the Values of it.
- Create object fig equals plt.figure(figsize=(10, 8)).
- Create the bar graph: plt.bar(Time, values, color='maroon', width=0.2).
- Putting titles on graph:

- plt.xlabel("Time of Test")

-plt.ylabel("time in minutes")

-plt.title("Total Time Test")



Observations between all models

1. SVM

- C=.1, kernel='linear', gamma=1

  - acc =70.49087963941123 %

- C=1, kernel='linear', gamma=1

  -acc= 70.51200788787942 %

- C=10, kernel='linear', gamma=1
  - acc= 70.77258961898724 %
- C=10, kernel='sigmoid', gamma=1
  - acc=57.14486935699698 %
- C=10, kernel='rbf', gamma=1
  - acc=68.0400028170998 %

2. Gradient Boosting Model

- n_estimators=300, learning_rate=0.05, random_state=100, max_features=5
  -acc= 72.53327699133742 %
- n_estimators=200, learning_rate=0.05, random_state=100, max_features=5
  -acc=72.46989224593281 %
- n_estimators=100, learning_rate=0.05, random_state=100, max_features=5
  - acc=72.02619902810056 %
- Learning rate
- n_estimators=300, learning_rate=0.1, random_state=100, max_features=5
  -acc=72.25156701176138 %
- n_estimators=300, learning_rate=1, random_state=100, max_features=5
  -acc=69.1879709838721 %
- n_estimators=300, learning_rate=0.5, random_state=100, max_features=5
  -acc=71.32896682864991 %

3. Logistic Regression Model

- random_state=0,'C': 0.1, 'penalty': 'l2'
  - acc=69.30065497570251 %
- random_state=0,C': 1.0, 'penalty': 'l2'

-acc=70.01197267413198 %

- random_state=0, C=10, penalty= 'l2' -acc=70.01197267413198 %
- random_state=0, C=0.1, penalty= 'none' -acc=70.40636664553843 %

4. Random Forest Model

- n_estimators=100,max_depth=9 -acc=71.43460807099092 %
- n_estimators=200,max_depth=9 -acc=70.90640185928586 %
- n_estimators=300,max_depth=9 -acc=71.42756532150152 %
- n_estimators=100,max_depth=5 -acc=70.91344460877527 %
- n_estimators=100,max_depth=12 -acc=71.68814705260934 %
- n_estimators=100,max_depth=15 -acc=71.62476230720473 %

5. KNN Model

- n_neighbors=11, p=2, metric='euclidean' - acc=67.70195084160856 %
- 'metric': 'manhattan', 'n_neighbors': 11, 'weights': 'distance' -acc=70.06831467004719 %
- n_neighbors=7, p=2, metric='euclidean' -acc=66.9765476442003 %
- n_neighbors=5, p=2, metric='euclidean' - acc=66.44834143249525 %
- n_neighbors=11, p=2, metric='manhattan -acc=69.56123670681033 %
- n_neighbors=11, p=2, metric='manhattan',weights='uniform'

-acc=66.86386365236989 %

- n_neighbors=11, p=2, metric='euclidean',weights='uniform' -acc=67.42728361152194 %

6. decision tree Model

- 'max_depth': 5, 'min_samples_leaf': 2, 'min_samples_split': 2 -acc=69.63166420170435
- max_depth': 5, 'min_samples_leaf': 1, 'min_samples_split': 2 -acc=69.86407493485457
- max_depth= 8, min_samples_leaf = 2, min_samples_split= 2 -acc=70.95570110571167 %
- max_depth= 10, min_samples_leaf = 2, min_samples_split= 2 -acc=69.71617719557716 %
- max_depth= 8, min_samples_leaf = 3, min_samples_split= 2 -acc=70.92753010775407 %
- max_depth= 8, min_samples_leaf = 1, min_samples_split= 2 -acc=70.95570110571167 %

This one is the best of all.