TATA STEEL ITS SUMMER INTERNSHIP

TITLE : Collection Insight Project

2019

Author : Soumyabroto Banerjee
B.Tech(3rd Year)
ECE
Roll:T91/ECE/164064

Guide : Mr. PulakKanti Pal and Mr.Saibal Nandi

CERTIFICATE

I, Soumyabroto Banerjee, student of B.Tech(ECE), 3rd Year at the Institute of Radio Physics and Electronics, University of Calcutta, Kolkata, completed my Summer Internship project from TATA STEEL IT DIVISION. The Internship Program was a period of 1 Month, starting on 2nd June, 2019.

During the mentioned duration, I worked on Analytics Projects and had the opportunity to handle Real Life data from Tata Steel. I ended up completing the Collection Insight Project under Mr. PulakKanti Pal and Mr. Saibal Nandi.

Date :30th June, 2019 Name: Soumyabroto Banerjee

ACKNOWLEDGEMENT

A training work owes its success from commencement to completion, to the people in love with researchers at various stages. Let me, in this page express my gratitude to all those who have helped us in various stages of this study. First, I would like to express my sincere gratitude to Mr. AbijitBiswas (Acting Head, IRPEL) for allowing me to undergo the Summer Training of 30 days at TATA STEEL IT Division.

I am grateful to my respected mentors, Mr.PulakKanti Pal and Mr. Saibal Nandi for the help provided in completion of the project which was assigned to me. Without their friendly help and their patience it would have been difficult to complete the assigned task

COMPANY PROFILE

Tata Iron and Steel Company was founded by Jamsetji Tata and established by Dorabji Tata on 26 August 1907, and began producing steel in 1912 as a branch of Jamsetji's Tata Group. By 1939, it operated the largest steel plant in the British Empire. The company launched a major modernization and expansion program in 1951. Later, in 1958, the program was upgraded to 2 million metric tonnes per annum (MTPA) project.By 1970, the company employed around 40,000 people at Jamshedpur, and a further 20,000 in the neighboring coal mines. In 1971 and 1979, there were unsuccessful attempts to nationalize the company.In 1990, the company began to expand, and established its subsidiary, Tata Inc., in New York. The company changed its name from TISCO to Tata Steel Ltd. in 2005.

Tata Steel Limited, with revenues of US$ 20.41 billion in FY18, is a leading global steel company with an annual steel production capacity of 33 MnTPA.

Established in Jamshedpur (India) in 1907, the company took shape from the vision of founder Jamsetji N. Tata and is today one of the world's most geographically-diversified steel producers with operations and commercial presence across the world. Tata Steel Group is spread across five continents with an employee base of over 65,000.

Focusing on Innovation, Technology, Sustainability & People, the Company strives to be the global *steel* industry benchmark for value creation and corporate citizenship and become the most admired brand in metals and minerals space.

In 2018, the Company successfully completed the acquisition of controlling stake of 72.65% in Bhushan Steel Limited now named Tata Steel BSL Limited. Currently, Tata Steel's consolidated India crude steel production capacity stands at 18.6 MnTPA with manufacturing facilities in Jamshedpur in Jharkhand, Kalinganagar and Dhenkanal in Odisha, Sahibabad in Uttar Pradesh and Khopoli in Maharashtra. Recently, Tata Steel has commenced the phase 2 expansion of its Kalinganagar steel plant to 8 MnTPA.

In addition, the Company has several downstream product extensions with manufacturing facilities for Wires, Tubes, Bearings, Agriculture Equipment and Industrial By-products. It also has a Ferro-alloys and Minerals division and a heavy-duty engineering and fabrication unit, Tata Growth Shop.

Tata Steel successfully delivered 11 MnTPA of steel to the Indian market and 1.15 MnTPA to International market in FY18.

In India, Tata Steel operates an end-to-end value chain that extends from mining to finished steel goods, catering to an array of market segments such as automotive, construction, general engineering etc.

The Company possesses and operates captive mines that ensure cost-competitiveness and production efficiencies through an uninterrupted supply of raw material. Tata Steel’s iron ore mines in Jharkhand and Odisha enable 100% captive iron ore usage and the coal mines in West Bokaro&Jharia ensure ~30% captive coal usage in operations. Tata Steel additionally has a dolomite mine, a chromite mine and manganese mines that ensures steady and cost-efficient supply of raw materials to its ferro alloy plants. A robust presence across the value chain helps the Company achieve higher economic efficiencies and customer satisfaction standards which is one of the best in the industry.

In Europe, Tata Steel is amongst the largest steel producers with a crude steel production capacity of over 12.3 MnTPA. It established its presence in the European continent after acquiring Corus in 2007.

With steelmaking facilities in the UK and Netherlands and downstream plants across Europe, it supplies high quality strip steel products to demanding markets such as automotive, construction, packaging and engineering.

In June 2018, Tata Steel signed definitive agreements with thyssenkrupp AG to combine their European steel businesses in a 50/50 joint venture. The proposed joint venture will create a strong pan European steel company that is structurally robust and competitive and will create value for all the stakeholders with a strong focus on performance, quality and technology leadership.

In South-East Asia, Tata Steel operations began in 2004 with the acquisition of NatSteel, Singapore. In 2005, it acquired a majority stake in Thailand-based steelmaker Millennium Steel, which further strengthened its South-East Asian operations.

Business Highlights FY18

- Recorded strong operating performance
- Enhanced the value-added products portfolio

- Commenced Kalinganagar plant expansion
- Pursued inorganic growth opportunities
- Created a sustainable European portfolio
- Successfully resolved the UK Pension scheme
- Made a landmark $1.3 billion bonds offering
- Strengthened the balance sheet through the $2 billion rights issue

Consolidated Financial Results FY18

No.	Label	2017-18 (Rscrore)
1	Revenue	133,016
2	EBITDA	22,045
4	PAT	17,763

BENEFITS OF IT SERVICES

1. FUTURE PROOF SERVICES, USING BEST-OF-BREED TECHNOLOGY

Leading Managed Service Providers (MSPs) must use the best technologies and equipment on the market to deliver services. IT services are constantly upgraded with no additional cost or financial risk to yourself. You never have to worry that your Managed IT Services will become obsolete.

2. LOW CAPITAL OUTLAY AND PREDICTABLE MONTHLY COSTS

The investment in specialist hardware and software will be high. A managed service offers the highest quality enterprise and carrier grade solutions to customers. A fixed monthly payment plan means you know what you're going to get and how much it's going to cost over the contract. No unexpected upgrade charges or changes in charges, guaranteed.

3. FLEXIBLE SERVICE

Managed IT Service Providers can be extremely flexible; a pay-as-you-go payment plan allows for quick growth when necessary, or cost savings when you need to consolidate.

4. CONVERGED SERVICES

Multiple Managed IT Services can be provided over a single "converged" connection, resulting in cost-savings on infrastructure. There are additional productivity and efficiency benefits, in the fact that remote staff working from home have access to all the voice & data applications that your HQ staff use.

5. HIGHLY RESILIENT, SECURE INFRASTRUCTURE

A Managed Service Provider's data centres and managed network infrastructure is much more robust than a standard Enterprise IT service. Infrastructure is run under 24x7x365 management, with government approval security procedures.

6. EXPERTISE

By selecting managed services you gain access to staff with specialist skills. Sometimes you will only need this skill once, so save the expense of training your staff for skills they will never use. Redcentric have invested £100m in its infrastructure and is backed by one of best 24x7x365 technical support and operations team.

7. CENTRALISATION

With a managed network you can benefit from the ability to centralise all your applications and servers within managed data centres, this leads to improved performance of staff, regardless of location. Access to centralised data centres within the network can also provide access to virtual services, as well as storage and backup infrastructure.

8. INCREASED SERVICE LEVELS

A Managed IT Service provides greater control of service levels and performance. With service level agreements in place you can be sure of continuity of service. A managed service company will also offer 24x7x365 support.

9. DISASTER RECOVERY AND BUSINESS CONTINUITY

The delivery of services is the life blood of the Managed Service Provider (MSP). They have designed networks and data centres that will be available, resilient and redundant for maintaining business continuity. You can take advantage of this significant technological investment. Your data will be safe and you voice services will continue to be delivered even if you main office is lost.

10. GREEN AND LEAN

By centralising your critical business systems within data centres and running your applications on a virtual platform, your business can benefit from a huge power saving - lowering your carbon footprint whilst reducing costs.

PROJECT WORK

Problem Statement:

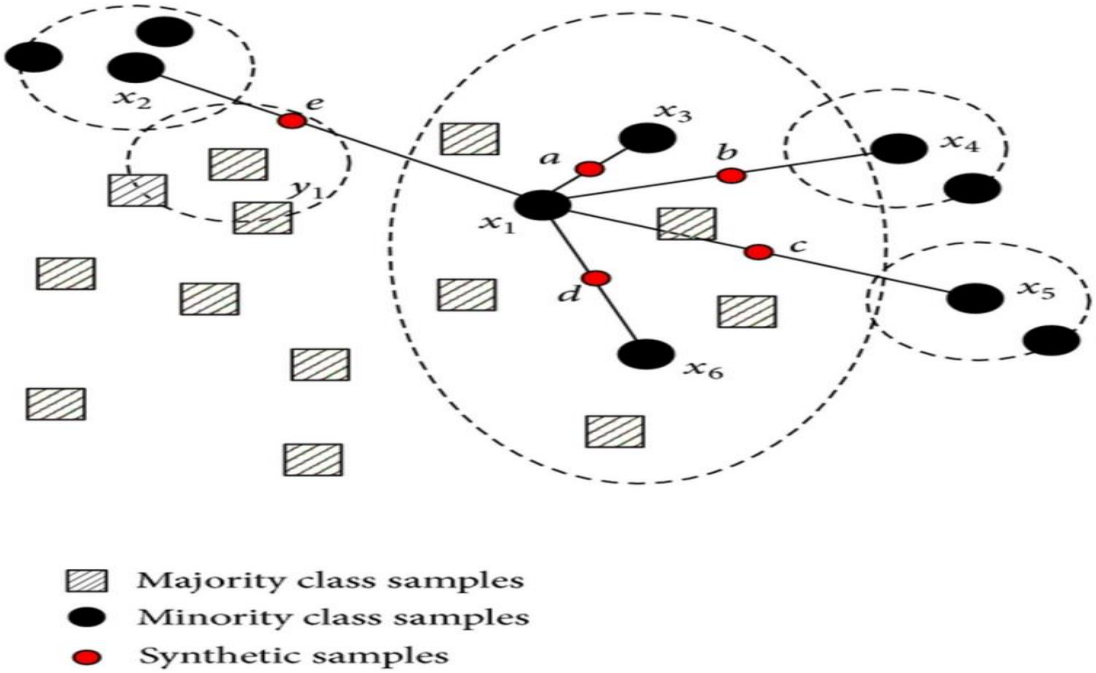
Part 1: Predict and Identify the Customers who have a very high delay in Invoice to Cash Collection.

Part2: Predict and Identify the weekly Collection Insight from a Date, *i.e*The Customers who are likely to pay within the next week of a date given as input.

Part 3: Predict the sales of Cash based Customers for the next 5 days from the current date.

Analytics

- The First Problem:The Dataset provided was the past record of 9 Months of the BPR Only, B2B Customers of Tata Steel
- The Dataset contained Cross Sectional and Time Series Features and was basically a Panel Dataset. So, I could neither use basic approaches of Machine Learning nor pure ARIMA Model.
- The main challenge was to predict the minority class, *i.e,* in the entire dataset contained over 110 thousand data points out of which the Class of No Delay, *i.e.* The Customer transactions which did not incur delay in Invoice to Cash Collection was more than 90 thousand. The Customer Transactions which suffered the significant delay of more than 30 days were very few in number, 224.
    - The technique used to tackle this problem was to generate Synthetic data points using the ADASYN algorithm
    - PAPER :ADASYN: Adaptive Synthetic Sampling Approach for Imbalanced Learning .
    - AUTHORS : Haibo He, Yang Bai, Edwardo A. Garcia, and Shutao Li
    - Source : IEEE JOURNAL 2008
    - The picture of the algorithm is attached.



- The Next task was based on Feature Engineering where two Columns were generatedmanually, one being the number of Pending Invoices prior to a Posting Date of an Invoice for every customer and the last was the Amount of Pending Invoices prior to a Posting Date for every Customer.
- After Feature Engineering choice of Model was very important, being a Supervised Learning Type Classification Problem. After checking accuracy for 3 of the Models, a weighted Voting Classifier was used to predict the Final Output.
- The 3 Models being used are – Random Forest, Adaaboot with base learner as Decision Tree, Bagging Classifier with base estimator as KNN. The Random Forest algorithm performs the best. Though no correct set of reasons are known yet with a few searches I got a few probable reasons:
    - ✓ Random Forests can handle thousands of input variables without variable deletion.
    - ✓ It generates an internal unbiased estimate of the generalization error as the forest building progresses.
    - Prototypes are computed that give information about the relation between the variables and the classification.
- The Final Output contained 3 classes – No Delay (Customers with no Invoice to Cash Delay) ; Low Delay (Customers who pay within 7 days from Due Date) ; High Delay (Customers who don't pay within the 1st week.)
- The Dataset was split into 3 segments : Training – 70%
  Testing and Validation – 15% and 15% each

- Accuracy:
    - Training Accuracy : 99.35
    - Testing Accuracy : 97.28

- The Second problem : This was a very interesting problem where we needed to predict the estimated Clearing Date.
- We used two models in order to predict the Estimated Date of Clearing.
        - First: Model predicting estimated date of clearing from Net Due Date.
        - Second : Model predicting estimated date of clearing from Posting.
    - We use the agreement of both the models in a range of 5 days to predict the final number of Invoices that will be cleared.
- In Second Problem too Random Forest outshone the Other Classifiers.
- The training and Testing data was all data before the Input Posting Date and the Validation Data was all the data points whose invoices were raised before the Posting Date but was not cleared.
    - The Training Accuracy for Model 1 : 99.98
    - The Testing Accuracy for Model 1 : 94.04
    - The Training Accuracy for Model 2 : 99.36
    - The Testing Accuracy for Model 2 : 97.67

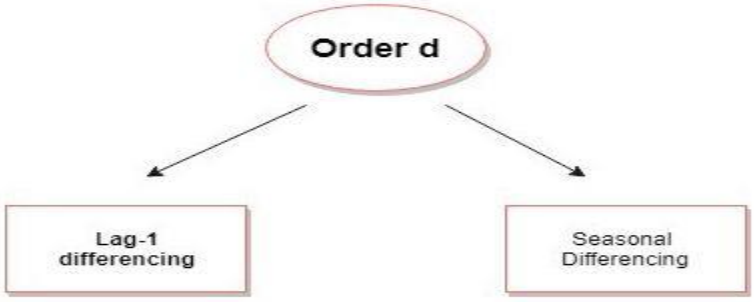
- The Third problem: This problem was not as difficult as the rest of the problems. We used Arima to forecast the sales of the Individual Customers for the next 5 days from a specified date.
    - ARIMA is an Ideology that captures autocorrelation in the series by modelling it directly.
    - Lags of the stationarized series are called “autoregressive” that refers to (**AR**) terms & Lags of the forecast errors are called “moving average” which refers to (**MA**) terms.
    - The autocorrelation function (**ACF**). Intuitively, a stationary **time series** is defined by its mean, variance and **ACF**. A useful result is that any function of a stationary **time series** is also a stationary **time series**.
    - We plotted the PACF and the ACF and used those as q and p, respectively in ARIMA(p,d,q). The differential

- In **time series** analysis, the partial autocorrelation function (**PACF**) gives the partial correlation of a **time series** with its own lagged values, controlling for the values of the **time series** at all shorter lags. It contrasts with the autocorrelation function, which does not control for other lags.
- The Terminologies involved:
  - p = No. of Auto-Regressive Terms
  - d = No. of Non-Seasonal Differences
  - q = No. of Moving Average Terms
    - These 3 are used to model ARIMA(p,d,q)
- Advantages :It is a strong underlined mathematical theory which makes it easy to predict "PREICTIVE INTERVALS" which is it is flexible in capturing a lot of different parameters.
- Disadvantages :No explicit seasonal indices, hard to interpret coefficients or explain "how the model works", there is danger of overfitting or mis-identification if not used with care.

Construction of an ARIMA model

1. *Stationarize* the series, if necessary, by differencing (& perhaps also logging, deflating, etc.)
2. Study the pattern of *autocorrelations* and *partial autocorrelations* to determine if lags of the stationarized series and/or lags of the forecast errors should be included in the forecasting equation
3. Fit the model that is suggested and check its residual diagnostics, particularly the residual ACF and PACF plots, to see if all coefficients are significant and all of the pattern has been explained.
4. Patterns that remain in the ACF and PACF may suggest the need for additional AR or MA terms

First apply differencing



Then fit ARMA (p, q):

$$Y_t = \beta_0 + \beta_1 Y_{t-1} + \ldots + \beta_p Y_{t-p} + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \ldots + \theta_q \varepsilon_{t-q}$$

ARIMA forecasting equation

- Let Y denote the *original* series
- Let y denote the *differenced* (stationarized) series

No difference (d=0): $y_t = Y_t$

First difference (d=1): $y_t = Y_t - Y_{t-1}$

Second difference (d=2): $y_t = (Y_t - Y_{t-1}) - (Y_{t-1} - Y_{t-2})$

$= Y_t - 2Y_{t-1} + Y_{t-2}$

ALGORITHMS USED :

Random Forest Classifier:

The Pseudo-code:

1. Randomly select **"k"** features from total **"m"** features.
    1. Where **k << m**
2. Among the **"k"** features, calculate the node **"d"** using the best split point.
3. Split the node into **daughter nodes** using the **best split**.
4. Repeat **1 to 3** steps until "l" number of nodes has been reached.
5. Build forest by repeating steps **1 to 4** for "n" number times to create **"n" number of trees**.

Predicting Using Random Forest :

To perform prediction using the trained random forest algorithm uses the below pseudocode.

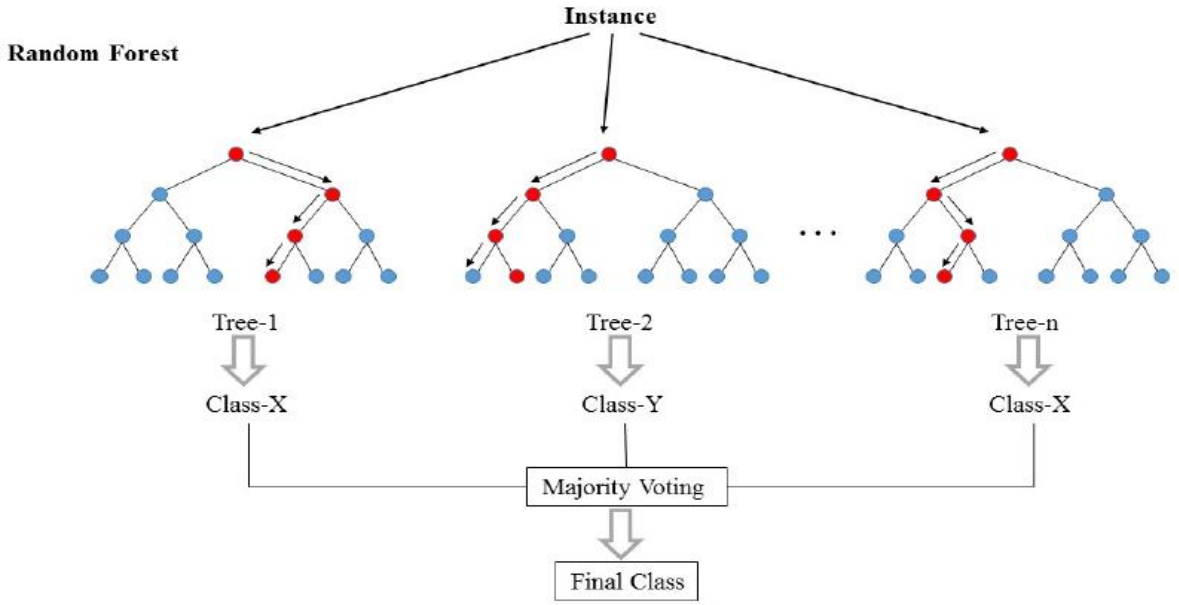
1. Takes the **test features** and use the rules of each randomly created decision tree to predict the outcome and stores the predicted outcome (target)
2. Calculate the **votes** for each predicted target.
3. Consider the **high voted** predicted target as the **final prediction** from the random forest algorithm

Advantages of using Random Forest :

Below are the advantages of Random Forest Algorithm compared with other classification algorithms.

- The over fitting problem will never come when we use the random forest algorithm in any classification problem.
- The same random forest algorithm can be used for both classification and regression task.
- The random forest algorithm can be used for feature engineering.

HOW RANDOM FOREST WORKS:

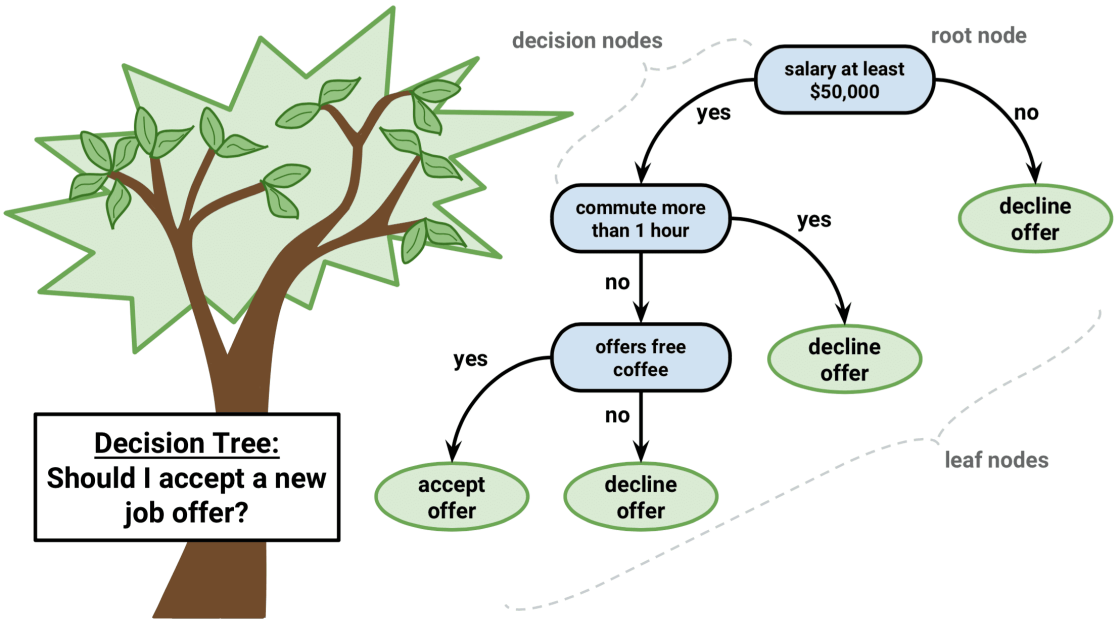


Decision Tree Classifier:

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. Decision trees learn from data to approximate a sine curve with a set of if-then-else decision rules. The deeper the tree, the more complex the decision rules and the fitter the model. Decision tree builds classification or regression models in the form of a tree structure. It breaks down a data set into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node has two or more branches. Leaf node represents a classification or decision. The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data.

Processes Involved:

- Splitting :The process of partitioning the data set into subsets. Splits are formed on a particular variable
- Pruning :The shortening of branches of the tree. Pruning is the process of reducing the size of the tree by turning some branch nodes into leaf nodes, and removing the leaf nodes under the original branch. Pruning is useful because classification trees may fit the training data well, but may do a poor job of classifying new values. A simpler tree often avoids over-fitting.
- Tree Selection :The process of finding the smallest tree that fits the data. Usually this is the tree that yields the lowest cross-validated error.



k-NearestNeighbors (kNN)

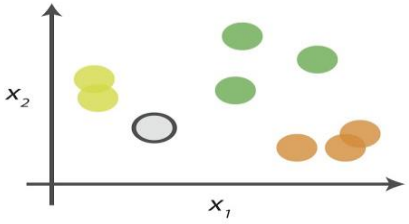
Pseudocode:

We can implement a KNN model by following the below steps:

1. Load the data
2. Initialize the value of k.
3. For getting the predicted class, iterate from 1 to total number of training data points
    1. Calculate the distance between test data and each row of training data. Here we will use Euclidean distance as our distance metric since it's the most popular method. The other metrics that can be used are Chebyshev, cosine, etc.
    2. Sort the calculated distances in ascending order based on distance values
    3. Get top k rows from the sorted array
    4. Get the most frequent class of these rows
    5. Return the predicted class.

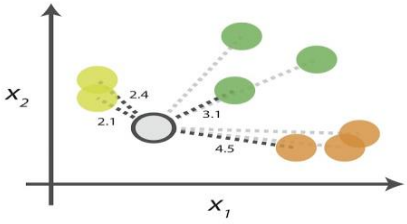
kNN Algorithm

0. Look at the data



Say you want to classify the grey point into a class. Here, there are three potential classes - lime green, green and orange.

1. Calculate distances

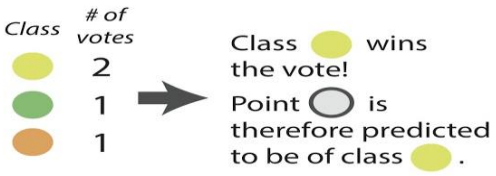


Start by calculating the distances between the grey point and all other points.

2. Find neighbours

Next, find the nearest neighbours by ranking points by increasing distance. The nearest neighbours (NNs) of the grey point are the ones closest in dataspace.

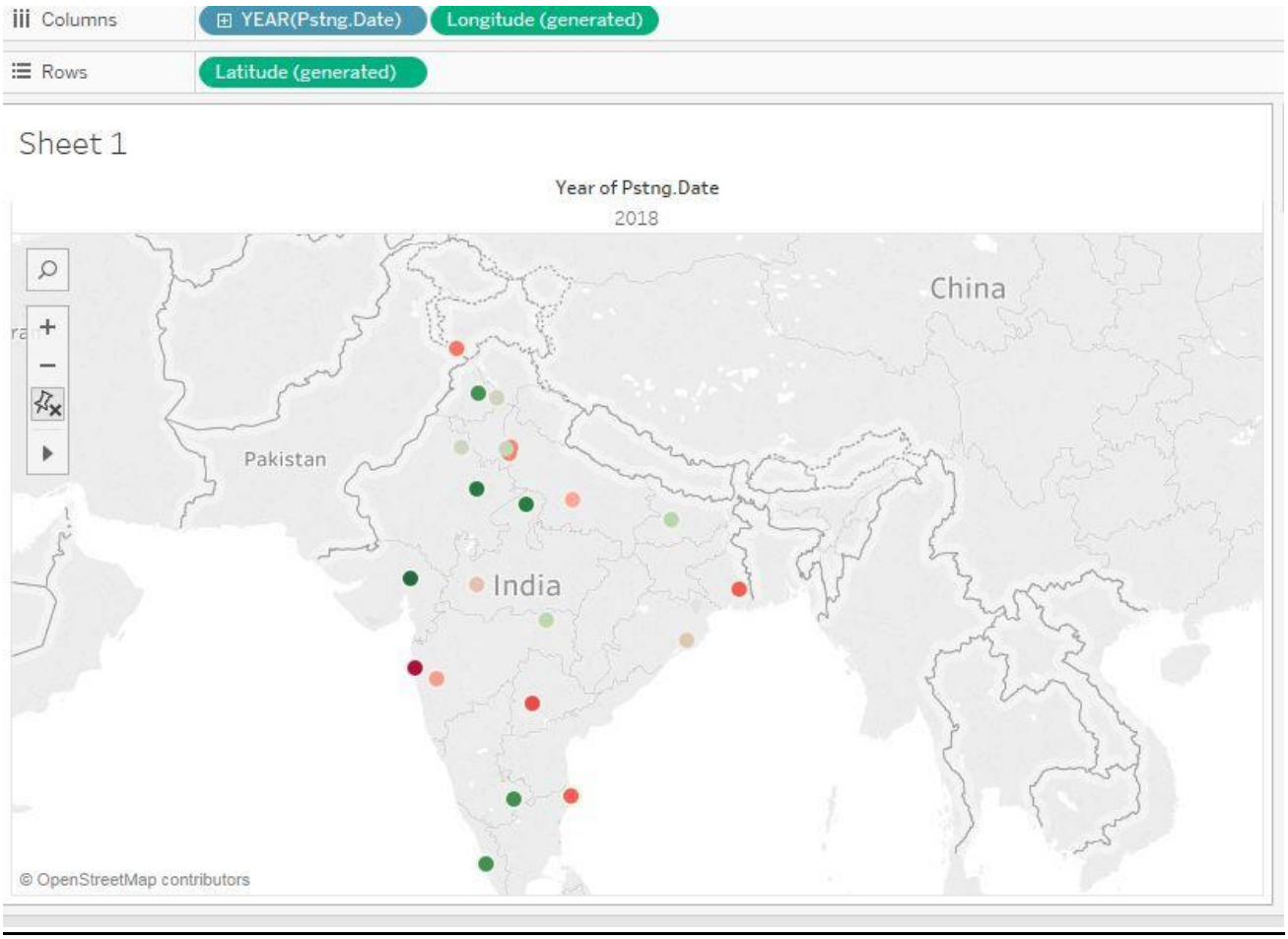
3. Vote on labels



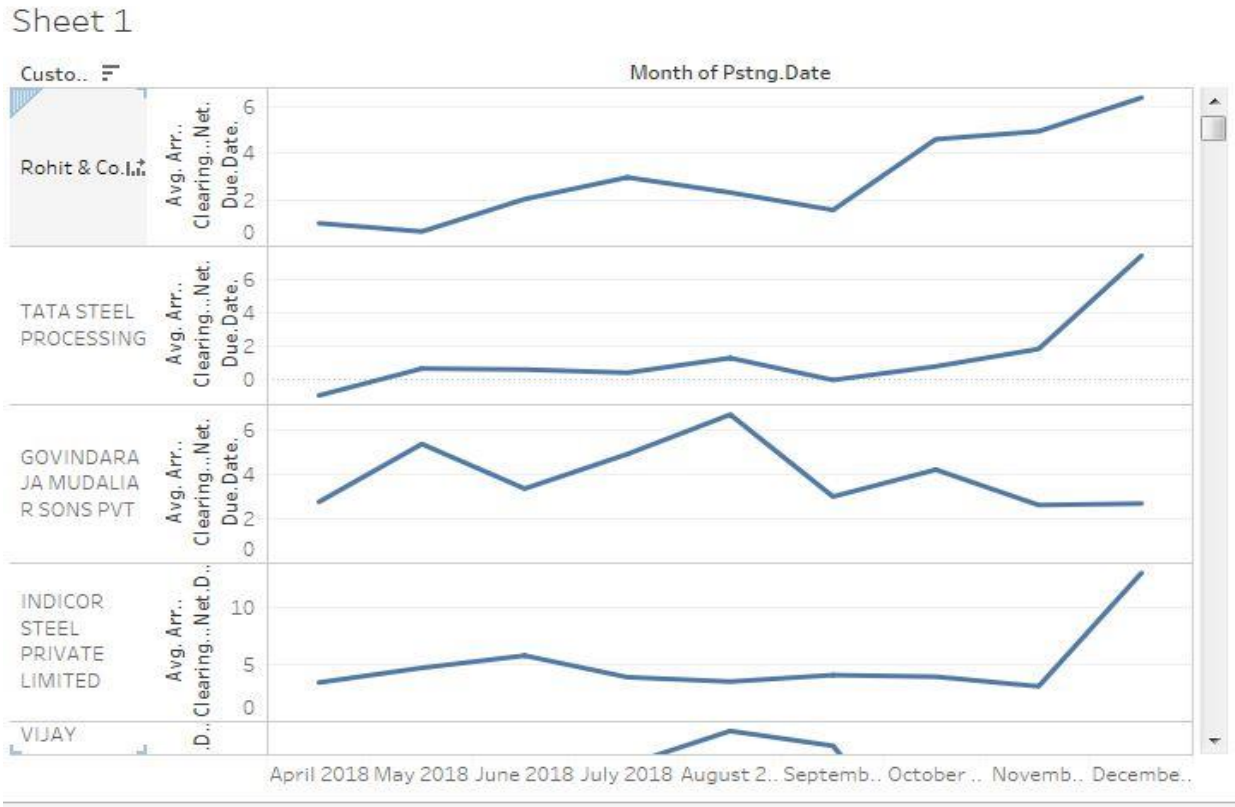
Vote on the predicted class labels based on the classes of the k nearest neighbours. Here, the labels were predicted based on the k=3 nearest neighbours.

DATASET VISUALISATION:

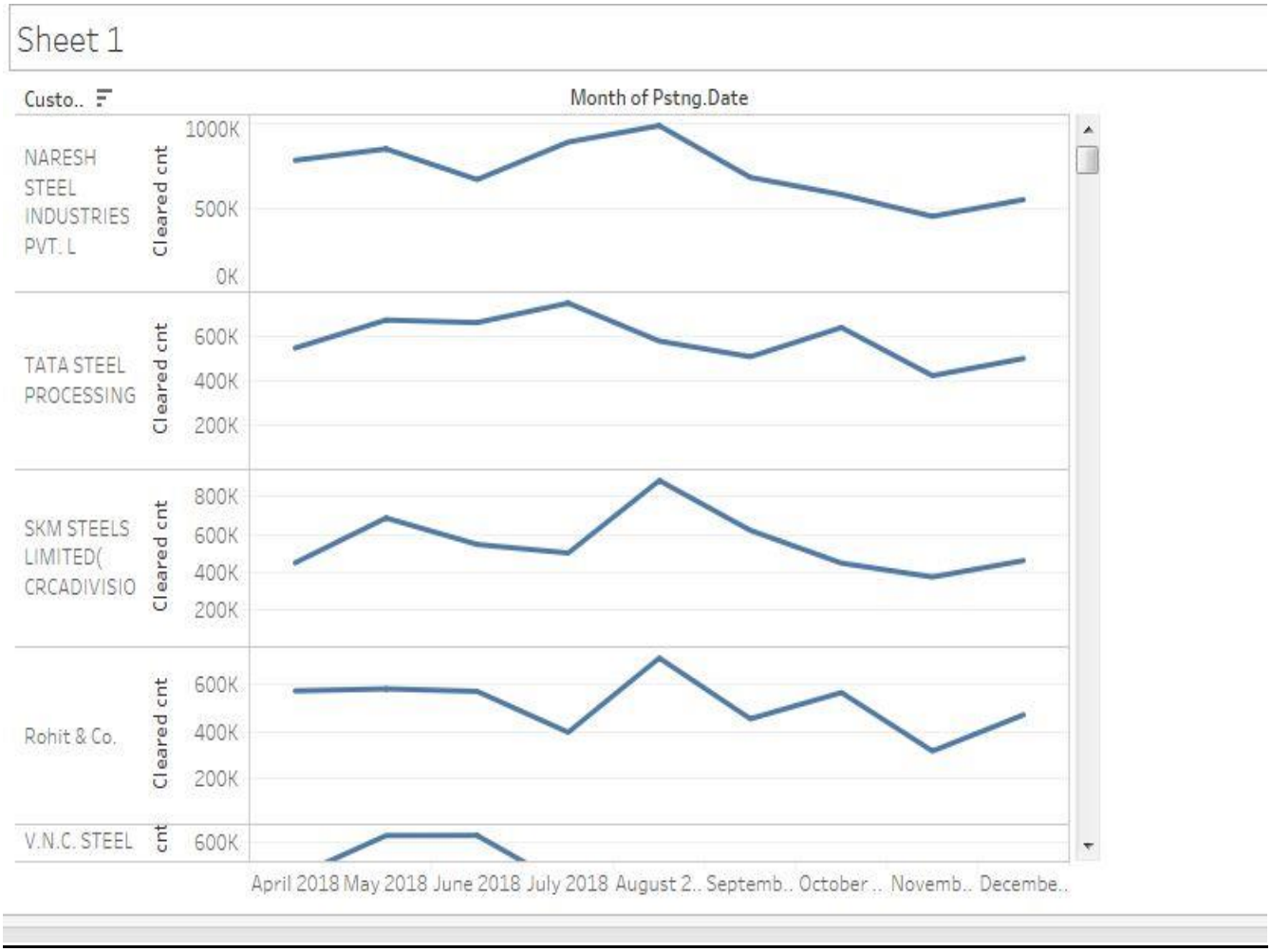
Business Arenas with their Delays: (Tableau visualization)



Average Delay Trends of the Top 4 Customers: (Tableau Visualisation)



Clearing Trend of Top 4 Customers:



CODE FOR PROBLEM 1:

```
#!/usr/bin/env python
# coding: utf-8


def preprocessor(df1,date):
import pandas as pd
importnumpy as np
fromsklearn import preprocessing
   #df1=df1[df1['Status']=='Overdue']
   #print(df1['Status'])
df1.dropna(inplace=True)
   df1 = df1.reset_index()
df1.drop(columns='index',inplace=True)



df1['Pstng.Date']=pd.to_datetime(df1['Pstng.Date'],dayfirst=True).dt.strftime("%Y%m%d").astype(int)
df1['Net.Due.Dt']=pd.to_datetime(df1['Net.Due.Dt'],dayfirst=True).dt.strftime("%Y%m%d").astype(int)
df1['Clearing']=pd.to_datetime(df1['Clearing']).dt.strftime("%Y%m%d").astype(int)
df=df1[df1['Pstng.Date'] < date ]
   df3=df[df['Clearing'] > date]
```

```
df.drop(df[df['Clearing'] > date].index,inplace=True)

   df1=df

   df1 = df1.reset_index()

df1.drop(columns='index',inplace=True)



   df3 = df3.reset_index()

df3.drop(columns='index',inplace=True)



   df2=df1[['Pstng.Date','Net.Due.Dt','Clearing','Status']]

df2.dropna(inplace=True)

df2['Range of Delay']=pd.to_datetime(df2['Clearing'],format='%Y%m%d') -
pd.to_datetime(df2['Net.Due.Dt'],format='%Y%m%d')

df2['PayT']= pd.to_datetime(df2['Net.Due.Dt'],format='%Y%m%d') -
pd.to_datetime(df2['Pstng.Date'],format='%Y%m%d')

df2['Range of Delay']=df2['Range of Delay'].dt.days

df2['Range of Delay']=pd.to_numeric(df2['Range of Delay'])

df2['PayT']=df2['PayT'].dt.days

df2['PayT']=pd.to_numeric(df2['PayT'])


   df2 = df2.reset_index()

df2.drop(columns = 'index',inplace=True)
```

```
   df2['Status'] = np.where(df2['Range of Delay'] > 7, 2, (np.where(df2['Range of Delay'] > 0,
1,0)))#(np.where(df2['Range of Delay'] > 0, 1,0)))))#(np.where(df2['Range of Delay'] > 0,
1,0)))))#(np.where(df2['Range of Delay'] > 10, 2,(np.where(df2['Range of Delay'] > 0,1,0)))))))#
(np.where(df2['Range of Delay'] > 30, 5, (np.where(df2['Range of Delay'] > 15, 4, (np.where(df2['Range of
Delay'] > 7, 3, (np.where(df2['Range of Delay'] > 3, 2, (np.where(df2['Range of Delay'] > 0, 1 ,0))))))))))))))

   #print(df2)

print('Y s in the initial set')

print(np.unique(df2['Status'],return_counts=True))

print('----------------------------------------------------------------------------------------')


df1['Status']=df2['Status']

df1['PayT']=df2['PayT']

df1.dropna(inplace=True)

   y=df1['Status']



busa = preprocessing.LabelEncoder()

ccar= preprocessing.LabelEncoder()

month=preprocessing.LabelEncoder()

zone=preprocessing.LabelEncoder()

bran=preprocessing.LabelEncoder()

payt=preprocessing.LabelEncoder()


df1.drop(columns='Status',inplace=True)

   #df1['Status']=df1['Status'].astype('category').cat.codes
```

```
arr=busa.fit(df1['BusA'])
df1['BusA']=busa.transform(df1['BusA'])
arr=ccar.fit(df1['CCAr'])
df1['CCAr']=ccar.transform(df1['CCAr'])#.astype('category').cat.codes
   #df1['Account']=df1['Account'].astype('category').cat.codes
arr=month.fit(df1['Month'])
df1['Month']=month.transform(df1['Month'])#.astype('category').cat.codes
df1.drop(columns='Reference',inplace=True)
df1.drop(columns='Customer.Name',inplace=True)
df1.drop(columns=['DocumentNo','Year','Clrng.doc.'],inplace=True)
arr=zone.fit(df1['Zone'])
df1['Zone']=zone.transform(df1['Zone'])    #.astype('category').cat.codes
arr=bran.fit(df1['Bran'])
df1['Bran']=bran.transform(df1['Bran'])#.astype('category').cat.codes
arr =payt.fit(df1['PayT'])
   #df1['PayT'] = payt.transform(df1['PayT'])#.astype('category').cat.codes
df1.drop(columns='Doc.Chq.dt',inplace=True)
df1.drop(columns='Ty',inplace=True)
df1.drop(columns='Sale.Type',inplace=True)



   #df1.drop(columns='Clearing',inplace=True)
   df1 = df1.reset_index()
df1.drop(columns='index',inplace=True)
```

```
df1.drop(columns='Arr..Clearing...Net.Due.Date.',inplace=True)

df1.drop(columns='G.L',inplace=True)

df1.drop(columns='Clearing',inplace=True)



   ##################### Oversampling ################################

fromimblearn.over_sampling import ADASYN

sm = ADASYN()

   df4, y = sm.fit_resample(df1, y)

   df1=pd.DataFrame(df4,columns=df1.columns)

   ###################################################################


   #y1=df1.columns

   #x = df1.values #returns a numpy array

   #min_max_scaler = preprocessing.MinMaxScaler()

   #x_scaled = min_max_scaler.fit_transform(x)

   #df1 = pd.DataFrame(x_scaled,columns=df1.columns)


print('Y s in the set')

print(np.unique(y,return_counts=True))

print('-------------------------------------------------------------------------------------')


return df1,df3,y,busa,ccar,month,zone,bran,payt
```

```
def trainer(df1,y):
import pandas as pd
importnumpy as np
fromsklearn import metrics
fromsklearn.ensemble import RandomForestClassifier
fromsklearn import tree
fromsklearn import neighbors
fromsklearn.ensemble import AdaBoostClassifier
fromsklearn.ensemble import VotingClassifier
fromsklearn.ensemble import BaggingClassifier
fromsklearn.model_selection import train_test_split


   #y=np.array(df1['Status'])




   X=df1
   #df1.head(5)
xtrain, xtest, ytrain, ytest = train_test_split(X, y, test_size = 0.3, random_state=30)
   # Random forest Model:
   model= RandomForestClassifier(n_estimators = 300,random_state
=30,n_jobs=3,verbose=2,max_features=None)#,class_weight={0:100,1:10000,2:500})
   # Decion Tree Model:
```

```
decision_tree = tree.DecisionTreeClassifier(random_state=30, max_depth=100)
   # AdaBoost Classifier Model:
clf = AdaBoostClassifier(base_estimator=model,n_estimators=100, random_state=0)
   # KNN :
mo = neighbors.KNeighborsClassifier(n_neighbors =2)
   # Bagging Classifier Model:
bagging = BaggingClassifier(mo,max_samples=0.5, max_features=0.5)
   #Voting Classifier
eclf = VotingClassifier(estimators=[ ('knn_bagging', bagging), ('adaboost', clf), ('RF',model)],voting='soft')
   eclf1 = eclf.fit(xtrain, ytrain)
print('TRAINING DONE .....................')
   eclf1=eclf.predict(xtrain)
   eclf2=eclf.predict(xtest)
print('TRAINING RESULTS----------------------------------------')
print("Accuracy: for Training :",metrics.accuracy_score(ytrain, eclf1)*100)
print("Accuracy: for Testing :",metrics.accuracy_score(ytest, eclf2)*100)
print()
print('Confusion Matrix for Random Forest')
print(metrics.confusion_matrix(eclf2, ytest))
print()
   print('-----------------------------------------------------------------------------------------------------------------------')
returneclf


def predict(df1,clf,busa,ccar,month,zone,bran,payt,date):
```

```
importnumpy as np

import pandas as pd

import copy

fromsklearn import preprocessing



df1['foo']= pd.to_datetime(df1['Net.Due.Dt'],format='%Y%m%d') - pd.to_datetime(date,format='%Y%m%d')


df1['foo']=df1['foo'].dt.days

df1['foo']=pd.to_numeric(df1['foo'])

   df1=df1[df1['foo']<6]

df1.drop(columns='foo',inplace=True)

df1.dropna(inplace=True)

   df1 = df1.reset_index()

df1.drop(columns='index',inplace=True)


df=copy.deepcopy(df1)

print(df.shape)

df['Prediction']=np.zeros(len(df1))

print(df1.shape)

df1.dropna(inplace=True)

   df1 = df1.reset_index()

df1.drop(columns='index',inplace=True)

   #df1['Status']=df1['Status'].astype('category').cat.codes)
```

```
   #df1['Status']=df1['Status'].astype('category').cat.codes


df1['PayT']= pd.to_datetime(df1['Net.Due.Dt'],format='%Y%m%d') -
pd.to_datetime(df1['Pstng.Date'],format='%Y%m%d')

df1['PayT']=df1['PayT'].dt.days

df1['PayT']=pd.to_numeric(df1['PayT'])

df1.drop(columns='Clearing',inplace=True)

   df1 = df1.reset_index()

df1.drop(columns = 'index',inplace=True)

   #df1['Status']=df1['Status'].astype('category').cat.codes

df1['BusA']=busa.transform(df1['BusA'])

df1['CCAr']=ccar.transform(df1['CCAr'])#.astype('category').cat.codes

   #df1['Account']=df1['Account'].astype('category').cat.codes

df1['Month']=month.transform(df1['Month'])#.astype('category').cat.codes

df1.drop(columns='Reference',inplace=True)

df1.drop(columns='Customer.Name',inplace=True)

df1.drop(columns=['DocumentNo','Year'],inplace=True)

df1['Zone']=zone.transform(df1['Zone'])#.astype('category').cat.codes

df1['Bran']=bran.transform(df1['Bran'])#.astype('category').cat.codes

   #df1['PayT'] =payt.transform(df1['PayT'])#.astype('category').cat.codes

df1.drop(columns='Doc.Chq.dt',inplace=True)

df1.drop(columns='Ty',inplace=True)

df1.drop(columns='Sale.Type',inplace=True)

df1.drop(columns='index',inplace=True)
```

```
df1.drop(columns='Arr..Clearing...Net.Due.Date.',inplace=True)

df1.drop(columns='G.L',inplace=True)


print('Starting Prediction ....')

prediction=clf.predict(df1)

print('Generating Probabilities.....')

predictprob=clf.predict_proba(df1)

print('Finished prediction...')

print()

print(df1.head())



arr=['No Delay','LowDelay','High Delay']

   arr1=['Low delay','Low Medium Delay','High Delay']#,'High Delay']#,'30 to 180','>180']#,'8-15 days','16-30 days','31-60 days','61-90 days','90-180 days','>180 days']

for i in range(len(arr)):

df[arr1[i]]=np.empty(len(df))

print('Computing Prediction Table.....')


df['Prediction']=prediction

for i in range(len(arr1)):

df[arr1[i]]=np.around(predictprob[:,i],decimals=2)


print(df.head(5))
```

```
print(df.head())

print('Writing to csv file')

print('----------------------------------------------------------------------------------------')

df.to_csv('Predictedout.csv',index=False)


import warnings

warnings.filterwarnings('ignore')

import pandas as pd

importnumpy as np

import copy

trainpath='MyData.csv'


print('Reading data...')

df1=pd.read_csv(trainpath)

print('Finished Reading...')

print()

print(df1.head(5))

print()

date=20181101

df3=copy.deepcopy(df1)

print('Preprocessing started .......')

df2,df4,y,busa,ccar,month,zone,bran,payt=preprocessor(df1,date)

print(df2.head(5))
```

```
print('Preprocessing Done.....')

print('-----------------------------------------------------------------------------------------------------')


df4.drop(columns=['Status','Clrng.doc.'],inplace=True)


import time

t1=time.time()

print('Training Starting.....')

clf=trainer(df2,y)

print('Training Done.....')

print('Time Taken',time.time()-t1)

print('-----------------------------------------------------------------------------')


print('Feature Importances')

for feature in zip(df2.columns, clf.feature_importances_):

print(feature)

testpath= ''

#test=pd.read_csv(testpath)

predict(df4,clf,busa,ccar,month,zone,bran,payt,date)

print()

print('_____ END _____')

print()

print()
```

CODE FOR PROBLEM 2:

```
#!/usr/bin/env python
# coding: utf-8

# In[88]:



def preprocessor(df1,date):
import pandas as pd
importnumpy as np
fromsklearn import preprocessing
df1.dropna(inplace=True)
   df1 = df1.reset_index()
df1.drop(columns='index',inplace=True)
   df1=df1[df1['Sale.Type']=='Credit']
   df1 = df1.reset_index()
df1.drop(columns='index',inplace=True)
df1['Pstng.Date'] = pd.to_datetime(df1['Pstng.Date'],dayfirst=True).dt.strftime("%Y%m%d").astype(int)
df1['Net.Due.Dt'] = pd.to_datetime(df1['Net.Due.Dt'],dayfirst=True).dt.strftime("%Y%m%d").astype(int)
df1['Clearing'] = pd.to_datetime(df1['Clearing'],dayfirst=True).dt.strftime("%Y%m%d").astype(int)
df=df1[df1['Pstng.Date'] < date ]
   df3=df[df['Clearing'] > date]
```

```
df.drop(df[df['Clearing'] > date].index,inplace=True)

   df1=df

   df1 = df1.reset_index()

df1.drop(columns='index',inplace=True)

   df3 = df3.reset_index()

df3.drop(columns='index',inplace=True)

   df2=df1[['Pstng.Date','Net.Due.Dt','Clearing','Status']]

df2.dropna(inplace=True)

df2['Range of Delay']=pd.to_datetime(df2['Clearing'],format='%Y%m%d') -
pd.to_datetime(df2['Net.Due.Dt'] , format='%Y%m%d')

df2['PayT']= pd.to_datetime(df2['Net.Due.Dt'],format='%Y%m%d') -
pd.to_datetime(df2['Pstng.Date'],format='%Y%m%d')

df2['y2']=pd.to_datetime(df2['Clearing'],format='%Y%m%d') - pd.to_datetime(df2['Pstng.Date'] ,
format='%Y%m%d')

df2['Range of Delay']=df2['Range of Delay'].dt.days

df2['Range of Delay']=pd.to_numeric(df2['Range of Delay'])

df2['PayT']=df2['PayT'].dt.days

df2['PayT']=pd.to_numeric(df2['PayT'])

df2['y2']=df2['y2'].dt.days

df2['y2']=pd.to_numeric(df2['y2'])


   df2 = df2.reset_index()

df2.drop(columns = 'index',inplace=True)
```

```
    df2['Status'] = np.where(df2['Range of Delay'] > 7, 2, (np.where(df2['Range of Delay'] > 0,
1,0)))#,(np.where(df2['Range of Delay'] < 0, 0, 1)))))#(np.where(df2['Range of Delay'] > 0,
1,0)))))#(np.where(df2['Range of Delay'] > 0, 1,0)))))#(np.where(df2['Range of Delay'] > 10,
2,(np.where(df2['Range of Delay'] > 0,1,0)))))))# (np.where(df2['Range of Delay'] > 30, 5,
(np.where(df2['Range of Delay'] > 15, 4, (np.where(df2['Range of Delay'] > 7, 3, (np.where(df2['Range of
Delay'] > 3, 2, (np.where(df2['Range of Delay'] > 0, 1 ,0))))))))))))))

df2['y2'] = np.where(df2['y2'] > 30, 3, (np.where(df2['y2'] > 15, 2,(np.where(df2['y2'] > 0, 1, 0)))))

print('Ranges in the initial set')

print(np.unique(df2['Status'],return_counts=True))

print('-----------------------------------------------------------------------------------------')

print('Y2 s in the initial set')

print(np.unique(df2['y2'],return_counts=True))

print('-----------------------------------------------------------------------------------------')

df1['Status']=df2['Status']

df1['PayT']=df2['PayT']

df1.dropna(inplace=True)

   y1=df1['Status']

   y2=df2['y2']

busa = preprocessing.LabelEncoder()

ccar = preprocessing.LabelEncoder()

month = preprocessing.LabelEncoder()

zone = preprocessing.LabelEncoder()

bran = preprocessing.LabelEncoder()

payt = preprocessing.LabelEncoder()

df1.drop(columns='Status',inplace=True)
```

```
   #df1['Status']=df1['Status'].astype('category').cat.codes
arr=busa.fit(df1['BusA'])
df1['BusA']=busa.transform(df1['BusA'])
arr=ccar.fit(df1['CCAr'])
df1['CCAr']=ccar.transform(df1['CCAr'])#.astype('category').cat.codes
   #df1['Account']=df1['Account'].astype('category').cat.codes
arr=month.fit(df1['Month'])
df1['Month']=month.transform(df1['Month'])#.astype('category').cat.codes
df1.drop(columns='Reference',inplace=True)
df1.drop(columns='Customer.Name',inplace=True)
df1.drop(columns=['DocumentNo','Year','Clrng.doc.'],inplace=True)
arr=zone.fit(df1['Zone'])
df1['Zone']=zone.transform(df1['Zone'])    #.astype('category').cat.codes
arr=bran.fit(df1['Bran'])
df1['Bran']=bran.transform(df1['Bran'])#.astype('category').cat.codes
arr =payt.fit(df1['PayT'])
   #df1['PayT'] = payt.transform(df1['PayT'])#.astype('category').cat.codes
df1.drop(columns='Doc.Chq.dt',inplace=True)
df1.drop(columns='Ty',inplace=True)
df1.drop(columns='Sale.Type',inplace=True)
   #df1.drop(columns='Clearing',inplace=True)
   df1 = df1.reset_index()
df1.drop(columns='index',inplace=True)
df1.drop(columns='Arr..Clearing...Net.Due.Date.',inplace=True)
```

```
df1.drop(columns='G.L',inplace=True)

df1.drop(columns='Clearing',inplace=True)



    ####################Oversampling#################################

fromimblearn.over_sampling import ADASYN

sm = ADASYN()

    df4, y = sm.fit_resample(df1, y)

    df1=pd.DataFrame(df4,columns=df1.columns)

    ################################################################



    return df1,df3,y1,y2,busa,ccar,month,zone,bran,payt



# In[89]:



def trainer(df1,y1,y2):

import pandas as pd

importnumpy as np

fromsklearn import metrics

fromsklearn.ensemble import RandomForestClassifier

fromsklearn import tree
```

```
fromsklearn import neighbors

fromsklearn.ensemble import AdaBoostClassifier

fromsklearn.ensemble import VotingClassifier

fromsklearn.ensemble import BaggingClassifier

fromsklearn.model_selection import train_test_split



   X=df1

   #df1.head(5)

   xtrain1, xtest1, ytrain1, ytest1 = train_test_split(X, y1, test_size = 0.3, random_state=30)

   xtrain2, xtest2, ytrain2, ytest2 = train_test_split(X, y2, test_size = 0.3, random_state=30)

   # Random forest Model:

   model1= RandomForestClassifier(n_estimators = 300,random_state = 100, n_jobs=3, verbose=2,
max_features=None)

   model2= RandomForestClassifier(n_estimators = 300,random_state = 100, n_jobs=3, verbose=2,
max_features=None)#,class_weight={0:100,1:10000,2:500})


eclf = VotingClassifier(estimators=[ ('knn_bagging', bagging), ('adaboost', clf), ('RF',model)],voting='soft')

   eclf1 = eclf.fit(xtrain1, ytrain1)

   eclf1 = model2.fit(xtrain2,ytrain2)

print('TRAINING DONE .....')



   eclf1=eclf.predict(xtrain1)
```

```
    eclf2=eclf.predict(xtest1)
print('TRAINING RESULTS----------------------------------------')
print("Accuracy: for Training for range :",metrics.accuracy_score(ytrain1, eclf1)*100)
print("Accuracy: for Testing for range:",metrics.accuracy_score(ytest1, eclf2)*100)
print()


    eclf3=model2.predict(xtrain2)
    eclf4=model2.predict(xtest2)
print('TRAINING RESULTS----------------------------------------')
print("Accuracy: for Training for y2:",metrics.accuracy_score(ytrain2, eclf3)*100)
print("Accuracy: for Testing for y2 :",metrics.accuracy_score(ytest2, eclf4)*100)
print()

print('Confusion Matrix for Range')
print(metrics.confusion_matrix(eclf2, ytest1))
print()
print('Confusion Matrix for Range')
print(metrics.confusion_matrix(eclf4, ytest2))
print()
    print('-------------------------------------------------------------------------------------------------------------------')
return eclf,model2

def predict(df1,clf1,clf2,busa,ccar,month,zone,bran,payt):
```

```
importnumpy as np

import pandas as pd

import copy

fromsklearn import preprocessing

df=copy.deepcopy(df1)

print(df.shape)

   #df['Prediction']=np.zeros(len(df1))

print(df1.shape)

df1.dropna(inplace=True)

   df1 = df1.reset_index()

df1.drop(columns='index',inplace=True)

   #df1['Status']=df1['Status'].astype('category').cat.codes)

   #df1['Status']=df1['Status'].astype('category').cat.codes


df1['PayT']= pd.to_datetime(df1['Net.Due.Dt'],format='%Y%m%d') -
pd.to_datetime(df1['Pstng.Date'],format='%Y%m%d')

df1['PayT']=df1['PayT'].dt.days

df1['PayT']=pd.to_numeric(df1['PayT'])

df1.drop(columns='Clearing',inplace=True)

   df1 = df1.reset_index()

df1.drop(columns = 'index',inplace=True)

   #df1['Status']=df1['Status'].astype('category').cat.codes

df1['BusA']=busa.transform(df1['BusA'])

df1['CCAr']=ccar.transform(df1['CCAr'])#.astype('category').cat.codes
```

```
   #df1['Account']=df1['Account'].astype('category').cat.codes
df1['Month']=month.transform(df1['Month'])#.astype('category').cat.codes
df1.drop(columns='Reference',inplace=True)
df1.drop(columns='Customer.Name',inplace=True)
df1.drop(columns=['DocumentNo','Year'],inplace=True)
df1['Zone']=zone.transform(df1['Zone'])#.astype('category').cat.codes
df1['Bran']=bran.transform(df1['Bran'])#.astype('category').cat.codes
   #df1['PayT'] =payt.transform(df1['PayT'])#.astype('category').cat.codes
df1.drop(columns='Doc.Chq.dt',inplace=True)
df1.drop(columns='Ty',inplace=True)
df1.drop(columns='Sale.Type',inplace=True)
   df1 = df1.reset_index()
df1.drop(columns='index',inplace=True)
df1.drop(columns='Arr..Clearing...Net.Due.Date.',inplace=True)
df1.drop(columns='G.L',inplace=True)

print('Starting Prediction ....')
prediction=clf1.predict(df1)
print('Generating Probabilities.....')
predictprob=clf1.predict_proba(df1)
print('Finished prediction...')
print()

print('Starting Prediction 2....')
```

```
   prediction1=clf2.predict(df1)
print('Generating Probabilities 2.....')
   predictprob1=clf2.predict_proba(df1)
print('Finished prediction 2...')
print()
print(df1.head())


arr=['No Delay','LowDelay','High Delay']
   arr1=['Low Delay_on_NetDUE','Medium Delay_on_NetDUE','High Delay_on_NetDUE']
   arr2=['Low Delay_on_Pstng','Low Medium Delay_on_Pstng','High Delay_on_Pstng','Very High Delay_on_Pstng']#,'High Delay']#,'30 to 180','>180']#,'8-15 days','16-30 days','31-60 days','61-90 days','90-180 days','>180 days']
for i in range(len(arr)):
df[arr1[i]]=np.empty(len(df))
print('Computing Prediction Table.....')
df['Actual'] = (pd.to_datetime(df['Clearing'],format='%Y%m%d')-pd.to_datetime(date,format='%Y%m%d')).dt.days
df['Actual'] = pd.to_numeric(df['Actual'])
df['Prediction_on_NetDue'] = prediction
df['Prediction_on_Pstng'] = prediction1


for i in range(len(arr1)):
df[arr1[i]]=np.around(predictprob[:,i],decimals=2)
df[arr2[i]]=np.around(predictprob1[:,i],decimals=2)
print(df.head(5))
```

```
days=np.where((df['Prediction_on_Pstng']==0)&(df['Low Delay_on_Pstng']>0.9),1,
        (np.where((df['Prediction_on_Pstng']==0)&(df['Low Delay_on_Pstng']>0.8),2,
        (np.where((df['Prediction_on_Pstng']==0)&(df['Low Delay_on_Pstng']>0.7),3,
        (np.where((df['Prediction_on_Pstng']==0)&(df['Low Delay_on_Pstng']>0.6),4,
        (np.where((df['Prediction_on_Pstng']==0)&(df['Low Delay_on_Pstng']>0.4),5,
        (np.where((df['Prediction_on_Pstng']==0)&(df['Low Medium Delay_on_Pstng']>0.2),6,
        (np.where((df['Prediction_on_Pstng']==1)&(df['Low Delay_on_Pstng']>0.4),8,
        (np.where((df['Prediction_on_Pstng']==1)&(df['Low Delay_on_Pstng']>0.3),9,
        (np.where((df['Prediction_on_Pstng']==1)&(df['Low Medium Delay_on_Pstng']>0.5),10,
        (np.where((df['Prediction_on_Pstng']==1)&(df['Low Medium Delay_on_Pstng']>0.6),11,
        (np.where((df['Prediction_on_Pstng']==1)&(df['Low Medium Delay_on_Pstng']>0.7),13,
        (np.where((df['Prediction_on_Pstng']==1)&(df['Low Medium Delay_on_Pstng']>0.8),15,
        (np.where((df['Prediction_on_Pstng']==2)&(df['Low Medium Delay_on_Pstng']>0.45),16,
        (np.where((df['Prediction_on_Pstng']==2)&(df['Low Medium Delay_on_Pstng']>0.4),18,
        (np.where((df['Prediction_on_Pstng']==2)&(df['Low Medium Delay_on_Pstng']>0.3),20,
        (np.where((df['Prediction_on_Pstng']==2)&(df['High Delay_on_Pstng']>0.7),22,
        (np.where((df['Prediction_on_Pstng']==2)&(df['High Delay_on_Pstng']>0.75),30,
         (np.where((df['Prediction_on_Pstng']==3)&(df['High Delay_on_Pstng']>0.45),35,
         (np.where((df['Prediction_on_Pstng']==3)&(df['High Delay_on_Pstng']>0.35),45,
         (np.where((df['Prediction_on_Pstng']==3)&(df['High Delay_on_Pstng']>0.25),50,60
              ))))))))))))))))))))))))))))))))))))))))))))
```

```
days1=np.where(df['Prediction_on_NetDue']==0,1,np.where(df['Prediction_on_NetDue']==1,5,np.where(df['Prediction_on_NetDue']==2,30,60)))#,60)))

crr=(pd.to_datetime(df1['Pstng.Date'],format='%Y%m%d')+ pd.to_timedelta(days, unit='D')-pd.to_datetime(date,format='%Y%m%d')).dt.days.astype(int)

crr=df[(crr<7) & (crr>-7)]

crr['Payment']='Yes'

arr=(pd.to_datetime(df1['Net.Due.Dt'],format='%Y%m%d')+ pd.to_timedelta(days1, unit='D')-pd.to_datetime(date,format='%Y%m%d')).dt.days.astype(int)

brr=df[(arr<7) & (arr>-7)]

brr['Payment']='Yes'

print('ARR =')

print(brr.head())

print('BRR =')

print(crr.head())


   #df['When']=np.where(brr['Prediction_on_NetDue'] < 2 ,'PAYMENT EXPECTED SOON',np.where(brr['']))

   df6 = pd.merge(crr, brr, how='outer', on=list(crr.columns))

df_out = pd.merge(df,df6,how='outer',on=list(df.columns))


   #df['Difference']=pd.to_datetime(df['Clearing'],format='%Y%m%d') - pd.to_datetime(df['Pstng.Date'],format='%Y%m%d')

print('DF=')
```

```
print(df_out.head())

print('Writing to csv file')

print('-------------------------------------------------------------------------------------------')

df_out.to_csv('Predictedout_1.csv',index=False)

return days


import warnings

warnings.filterwarnings('ignore')

import pandas as pd

importnumpy as np

import copy

trainpath='MyData.csv'



print('Reading data...')

df1=pd.read_csv(trainpath)

print('Finished Reading...')

print()

print(df1.head(5))

print()

date=20181001

df3=copy.deepcopy(df1)

print('Preprocessing started .......')

df2,df4,y1,y2,busa,ccar,month,zone,bran,payt=preprocessor(df1,date)
```

```
print(df2.head(5))

print('Preprocessing Done.....')

print('-----------------------------------------------------------------------------------------------------')


df4.drop(columns=['Status','Clrng.doc.'],inplace=True)


import time

t1=time.time()

print('Training Starting.....')

clf1,clf2=trainer(df2,y1,y2)

print('Training Done.....')

print('Time Taken',time.time()-t1)

print('-----------------------------------------------------------------------------')


print('Feature Importances')

for feature in zip(df2.columns, clf1.feature_importances_,clf2.feature_importances_):

print(feature)

testpath= ''

days=predict(df4,clf1,clf2,busa,ccar,month,zone,bran,payt)

print()

print('_____ END _____')

print()

print()
```

CODE FOR PROBLEM 3:

```
import pandas as pd
importnumpy as np
frompmdarima.arima import auto_arima
importmatplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
import time
df1=pd.read_csv('MyData.csv')
df1=df1[df1['Sale.Type']=='Cash']
df2=df1[['Pstng.Date','Account','Local.Crcy.Amt']]
df2.to_csv('CashData.csv')
df=pd.read_csv('CashData.csv',  parse_dates = ['Pstng.Date'], index_col=1)
df.drop(columns='Unnamed: 0',inplace=True)
print('Total Number of Unique Customers = ',len(np.unique(df['Account'])))
def cash(df,date,offset_days,offset_count):
unique_customer=np.unique(df['Account'])
df=df.sort_index()

forcust in unique_customer[:5]:
df_targets=df[df['Account'] == cust]
    #print(df_targets.head())
```

```
df_targets.drop(columns='Account',inplace=True)

train=df_targets[df_targets.index<= pd.to_datetime(date,format='%Y%m%d') ]

valid=df_targets[df_targets.index>= pd.to_datetime(date,format='%Y%m%d') ]

if(len(train)<2):

print()

print('NO TRANSACTIONS EXPECTED FROM THIS CUSTOMER ',cust)

print()

print('Moving on...')

print('----------------------------------------------------------------------------------------------------')

continue


    d=pd.to_datetime(date,format='%Y%m%d')-pd.to_timedelta(offset_days, unit='D')

if(len(train[train.index>pd.to_datetime(d,format='%Y%m%d')]) <offset_count):

print()

print('NO TRANSACTIONS EXPECTED FROM THIS CUSTOMER ',cust)

print()

print('Moving on...')

print('----------------------------------------------------------------------------------------------------')

continue

    #, error_action='ignore'

model = auto_arima(train, trace=True,error_action='warn',n_iter=10,n_jobs=-1,solver='newton')

model.fit(train)

forecast = model.predict(n_periods=len(valid))

forecast = pd.DataFrame(forecast,index = valid.index)#,columns=['Prediction'])
```

```
plt.title(cust)

plt.plot(train, label='Train')

plt.plot(valid, label='Valid')

plt.plot(forecast, label='Prediction')

plt.show()


print('----------------------------------------------------------------------------------------------------')

date=20181101

offset_days=60

offset_count=5

cash(df,date,offset_days,offset_count)
```

SAMPLE OUTPUT

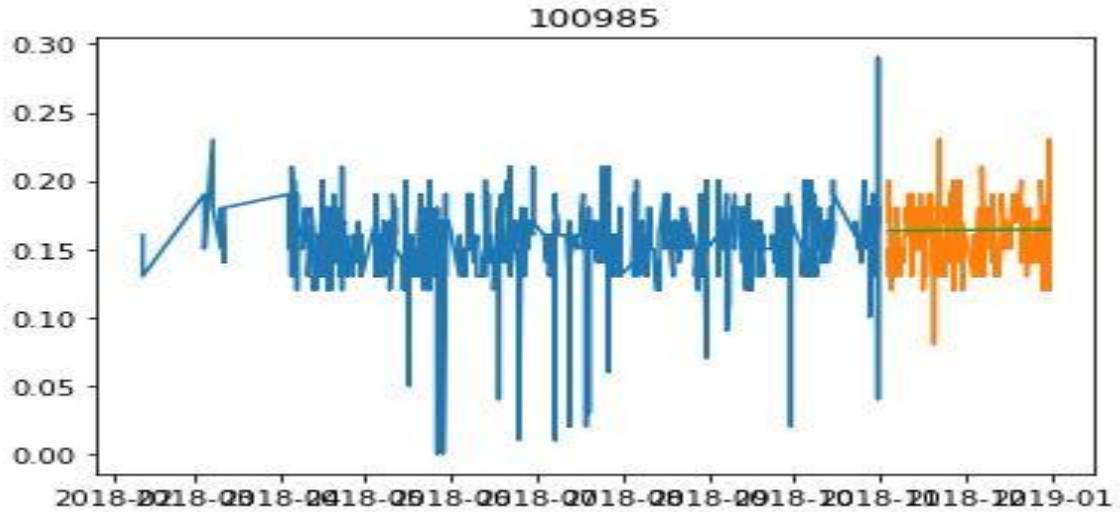
Sample Solution to Problem 1:

BusA	CCAr	Reference	Account	Zone	Bran	Status	Local.Crcy	Arr..Clear	PayT	Pstng.Dat	G.L	Pending_	pending_	Prediction
0	0	4.13E+09	929488	2	21	0	0.1	-5	27	20181231	1221001	0	0	No delay
6	0	1.12E+09	928860	1	9	2	0.04	5	31	20181031	1221001	0.6	4	4-7 days
6	0	1.12E+09	928860	1	9	0	0.12	0	31	20181105	1221001	1.42	10	No delay
6	0	2.15E+09	928860	1	9	3	0.14	8	31	20181027	1221001	0	0	8-15 days
2	0	2.41E+09	928860	1	9	1	0.16	2	31	20181103	1221001	1.26	9	0-3 days

Sample Solution to Problem 2:

	D	E	F	G	I	N	O	P	Q	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF
1	Account	Customer	Zone	Bran	Local.Crcy	Doc.Chq.c	Month	Pstng.Dat	Net.Due.D	Pending_	pending_	Low Delay	Medium D	High Dela	Actual	Predictior	Predictior	Low Delay	Low Medi	High Dela	Paym
2	927878	SAMRAT II	South	SBAD	0.12	29.11.2018	Nov	20181129	20181204	0	0	0.72	0.24	0.05	3	0	1	0.03	0.89	0.08	Yes
3	926346	GRG STEEL	North	FBAD	0.15	24.11.2018	Nov	20181124	20181204	1.6	16	0.95	0.05	0	3	0	1	0	1	0	Yes
4	926346	GRG STEEL	North	FBAD	0.15	24.11.2018	Nov	20181124	20181204	1.6	16	0.95	0.05	0	3	0	1	0	1	0	Yes
5	926346	GRG STEEL	North	FBAD	0.11	21.11.2018	Nov	20181121	20181201	1.5	18	0.62	0.38	0	2	0	1	0.02	0.98	0.01	Yes
6	926346	GRG STEEL	North	FBAD	0.11	21.11.2018	Nov	20181121	20181201	1.5	18	0.62	0.38	0	2	0	1	0.02	0.98	0.01	Yes
7	926346	GRG STEEL	North	FBAD	0.13	21.11.2018	Nov	20181121	20181201	1.5	18	0.63	0.37	0	2	0	1	0.02	0.98	0.01	Yes
8	926346	GRG STEEL	North	FBAD	0.13	21.11.2018	Nov	20181121	20181201	1.5	18	0.63	0.37	0	2	0	1	0.02	0.98	0.01	Yes
9	926346	GRG STEEL	North	FBAD	0.14	21.11.2018	Nov	20181121	20181201	1.5	18	0.67	0.33	0	2	0	1	0.02	0.98	0.01	Yes
10	901635	INTEGRAL	East	BHUB	0.1	30.11.2018	Nov	20181130	20181203	0	0	0.78	0.12	0.09	3	0	0	0.77	0.23	0	Yes
11	901635	INTEGRAL	East	BHUB	0.1	30.11.2018	Nov	20181130	20181203	0	0	0.78	0.12	0.09	3	0	0	0.77	0.23	0	Yes
12	901635	INTEGRAL	East	BHUB	0.1	30.11.2018	Nov	20181130	20181203	0	0	0.78	0.12	0.09	3	0	0	0.77	0.23	0	Yes
13	901635	INTEGRAL	East	BHUB	0.1	30.11.2018	Nov	20181130	20181203	0	0	0.78	0.12	0.09	3	0	0	0.77	0.23	0	Yes
14	856737	BAJRANG	East	GWAH	0.12	29.11.2018	Nov	20181129	20181203	0	0	0.68	0.28	0.04	2	0	1	0.31	0.69	0	Yes
15	856737	BAJRANG	East	GWAH	0.16	29.11.2018	Nov	20181129	20181203	0	0	0.68	0.28	0.04	2	0	1	0.31	0.69	0	Yes
16	856737	BAJRANG	East	GWAH	0.13	29.11.2018	Nov	20181129	20181203	0	0	0.68	0.27	0.04	2	0	1	0.34	0.67	0	Yes
17	854773	ANM ISPA	West	AMED	0.14	30.11.2018	Nov	20181130	20181203	4.38	31	0.99	0.01	0	2	0	1	0	1	0	Yes
18	854773	ANM ISPA	West	AMED	0.16	30.11.2018	Nov	20181130	20181203	4.38	31	0.99	0.01	0	2	0	1	0	1	0	Yes
19	854773	ANM ISPA	West	AMED	0.12	30.11.2018	Nov	20181130	20181203	4.38	31	0.98	0.02	0	2	0	1	0	1	0	Yes
20	854773	ANM ISPA	West	AMED	0.16	30.11.2018	Nov	20181130	20181203	4.38	31	0.99	0.01	0	2	0	1	0	1	0	Yes

Sample Solution to Problem 3:



BIBLIOGRAPHY

- www.tatasteel.com
- www.stackexchange.com
- www.mit.edu.com
- www.stackoverflow.com
- www.arxiv.com
- www.ieee.org
- www.researchgate.net
- www.collab.research.google.com


thank
you