Payment Date Prediction

Importing related Libraries

In [ ]:

```
import numpy as np
import pandas as pd
import datetime as dt
from scipy import stats

import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.ensemble import RandomForestRegressor
pd.options.mode.chained_assignment = None # default='warn'
```

Store the dataset into the Dataframe

In [ ]:

```
df=pd.read_csv('D:\HIGH RADIUS\dataset.csv')
```

Check the shape of the dataframe

In [ ]:

```
df.shape
```

Check the Detail information of the dataframe

In [ ]:

```
df.info()
```

Display All the column names

In [ ]:

```
df.columns
```

Describe the entire dataset

In [ ]:

```
df.describe()
```

Data Cleaning

- Show top 5 records from the dataset

In [ ]:

```
df.head()
```

Display the Null values percentage against every columns (compare to the total number of records)

- Output expected : area_business - 100% null, clear_data = 20% null, invoice_id = 0.12% null

In [ ]:

```
df.isna().sum()*100/len(df)
```

Display Invoice_id and Doc_Id

- Note - Many of the would have same invoice_id and doc_id

In [ ]:

```
df[['invoice_id','doc_id']]
```

Write a code to check - 'baseline_create_date',"document_create_date",'document_create_date.1' - these columns are almost same.

- Please note, if they are same, we need to drop them later

In [ ]:

```
df[['baseline_create_date','document_create_date','document_create_date.1']]
```

Please check, Column 'posting_id' is constant columns or not

In [ ]:

```
df['posting_id'].nunique()
```

Please check 'isOpen' is a constant column and relevant column for this project or not

In [ ]:

```
df['isOpen'].nunique()
```

Write the code to drop all the following columns from the dataframe

- 'area_business'
- "posting_id"
- "invoice_id"
- "document_create_date"
- "isOpen"
- 'document type'
- 'document_create_date.1

In [ ]:

```
df.drop(columns= ['area_business'], axis= 1, inplace= True)
df.drop(columns= ['posting_id'], axis= 1, inplace= True)
df.drop(columns= ['invoice_id'], axis= 1, inplace= True)
df.drop(columns= ['document_create_date'], axis= 1, inplace= True)
df.drop(columns= ['isOpen'], axis= 1, inplace= True)
df.drop(columns= ['document type'], axis= 1, inplace= True)
df.drop(columns= ['document_create_date.1'], axis= 1, inplace= True)
```

Please check from the dataframe whether all the columns are removed or not

In [ ]:

```
df.columns
```

Show all the Dublicate rows from the dataframe

In [ ]:

```
df[df.duplicated(keep='first')]
```

Display the Number of Dublicate Rows

In [ ]:

```
df[df.duplicated()].shape[0]
```

Drop all the Dublicate Rows

In [ ]:

```
df.drop_duplicates(keep='first',inplace=True)
```

Now check for all dublicate rows now

- Note - It must be 0 by now

In [ ]:

```
df[df.duplicated()].shape[0]
```

Check for the number of Rows and Columns in your dataset

In [ ]:

```
df.shape
```

Find out the total count of null values in each columns

In [ ]:

```
df.isna().sum()
```

#Data type Conversion

Please check the data type of each column of the dataframe

In [ ]:

```
df.dtypes
```

Check the datatype format of below columns

- clear_date
- posting_date
- due_in_date
- baseline_create_date

In [ ]:

```
df[['clear_date','posting_date','due_in_date','baseline_create_date']].dtypes
```

converting date columns into date time formats

- clear_date
- posting_date
- due_in_date
- baseline_create_date

- **Note - You have to convert all these above columns into "%Y%m%d" format**

In [ ]:

```
df['clear_date']=pd.to_datetime(df['clear_date'])
df['posting_date']=pd.to_datetime(df['posting_date'])
df['due_in_date']=pd.to_datetime(df['due_in_date'],format='%Y%m%d')
df['baseline_create_date']=pd.to_datetime(df['baseline_create_date'],format='%Y%m%d')
```

Please check the datatype of all the columns after conversion of the above 4 columns

In [ ]:

```
df.dtypes
```

the invoice_currency column contains two different categories, USD and CAD

- Please do a count of each currency

In [ ]:

```
df['invoice_currency'].value_counts()
```

display the "total_open_amount" column value

In [ ]:

```
print(df['total_open_amount'])
```

Convert all CAD into USD currency of "total_open_amount" column

- 1 CAD = 0.7 USD
- Create a new column i.e "converted_usd" and store USD and convered CAD to USD

In [ ]:

```
df['converted_usd']=np.where(df['invoice_currency']=='CAD',df['total_open_amount']*0.7,df['
```

Display the new "converted_usd" column values

In [ ]:

```
df['converted_usd']
```

Display year wise total number of record

- Note - use "buisness_year" column for this

In [ ]:

```
df['buisness_year'].value_counts()
```

Write the code to delete the following columns

- 'invoice_currency'
- 'total_open_amount',

In [ ]:

```
df.drop(['invoice_currency','total_open_amount'],axis=1,inplace=True)
```

Write a code to check the number of columns in dataframe

In [ ]:

```
df.shape[1]
```

Splitting the Dataset

Look for all columns containing null value

- Note - Output expected is only one column

In [ ]:

```
df.columns[df.isna().any()]
```

Find out the number of null values from the column that you got from the above code

In [ ]:

```
df['clear_date'].isna().sum()
```

On basis of the above column we are spliting data into dataset

- First dataframe (refer that as maindata) only containing the rows, that have NULL data in that column (This is going to be our train dataset)
- Second dataframe (refer that as nulldata) that contains the columns, that have Not Null data in that column (This is going to be our test dataset)

In [ ]:

```
maindata = df[~pd.isnull(df['clear_date'])]
nulldata = df[pd.isnull(df['clear_date'])]
```

Check the number of Rows and Columns for both the dataframes

In [ ]:

```
maindata.shape
```

In [ ]:

```
nulldata.shape
```

Display the 5 records from maindata and nulldata dataframes

In [ ]:

```
maindata.sample(5)
```

In [ ]:

```
nulldata.sample(5)
```

Considering the maindata

Generate a new column "Delay" from the existing columns

- Note - You are expected to create a new column 'Delay' from two existing columns, "clear_date" and "due_in_date"
- Formula - Delay = clear_date - due_in_date

In [ ]:

```
maindata['Delay']=(maindata['clear_date']-maindata['due_in_date'])
```

Generate a new column "avgdelay" from the existing columns

- Note - You are expected to make a new column "avgdelay" by grouping "name_customer" column with reapect to mean of the "Delay" column.
- This new column "avg_delay" is meant to store "customer_name" wise delay
- groupby('name_customer')['Delay'].mean(numeric_only=False)
- Display the new "avg_delay" column

In [ ]:

```
avgdelay=maindata.groupby('name_customer')['Delay'].mean(numeric_only=False)
```

In [ ]:

```
avgdelay
```

You need to add the "avg_delay" column with the maindata, mapped with "name_customer" column

- Note - You need to use map function to map the avgdelay with respect to "name_customer" column

In [ ]:

```
maindata["Avg_Delay"] = maindata["name_customer"].map(avgdelay)
```

Observe that the "avg_delay" column is in days format. You need to change the format into seconds

- Days_format : 17 days 00:00:00
- Format in seconds : 1641600.0

In [ ]:

```
maindata['Avg_Delay']=maindata['Avg_Delay'].dt.total_seconds()
```

Display the maindata dataframe

In [ ]:

```
maindata
```

Since you have created the "avg_delay" column from "Delay" and "clear_date" column, there is no need of these two columns anymore

- You are expected to drop "Delay" and "clear_date" columns from maindata dataframe

In [ ]:

```
maindata.drop(['Delay','clear_date'],axis=1,inplace=True)
```

Splitting of Train and the Test Data

You need to split the "maindata" columns into X and y dataframe

- Note - y should have the target column i.e. "avg_delay" and the other column should be in X
- X is going to hold the source fields and y will be going to hold the target fields

In [ ]:

```
X=maindata.drop('Avg_Delay',axis=1)
```

In [ ]:

```
y=maindata['Avg_Delay']
```

You are expected to split both the dataframes into train and test format in 60:40 ratio

- Note - The expected output should be in "X_train", "X_loc_test", "y_train", "y_loc_test" format

In [ ]:

```
from sklearn.model_selection import train_test_split
X_train,X_loc_test,y_train,y_loc_test=train_test_split(X,y,train_size=0.6)
```

Please check for the number of rows and columns of all the new dataframes (all 4)

In [ ]:

```
X_train.shape
```

In [ ]:

```
X_loc_test.shape
```

In [ ]:

```
y_train.shape
```

In [ ]:

```
y_loc_test.shape
```

Now you are expected to split the "X_loc_test" and "y_loc_test" dataset into "Test" and "Validation" (as the names given below) dataframe with 50:50 format

- Note - The expected output should be in "X_val", "X_test", "y_val", "y_test" format

In [ ]:

```
X_val,X_test,y_val,y_test=train_test_split(X_loc_test,y_loc_test,train_size=0.5)
```

Please check for the number of rows and columns of all the 4 dataframes

In [ ]:

```
X_val.shape
```

In [ ]:

```
X_test.shape
```

In [ ]:

```
y_val.shape
```

In [ ]:

```
y_test.shape
```

Exploratory Data Analysis (EDA)

Distribution Plot of the target variable (use the dataframe which contains the target field)

- Note - You are expected to make a distribution plot for the target variable

In [ ]:

```
sns.kdeplot(y_train, color='g')
```

You are expected to group the X_train dataset on 'name_customer' column with 'doc_id' in the x_train set

Need to store the outcome into a new dataframe

- Note code given for groupby statement- X_train.groupby(by=['name_customer'], as_index=False)['doc_id'].count()

In [ ]:

```
df_docID=X_train.groupby(by=['name_customer'], as_index=False)['doc_id'].count()
```

You can make another distribution plot of the "doc_id" column from x_train

In [ ]:

```
sns.kdeplot(X_train['doc_id'],color='r')
```

Create a Distribution plot only for business_year and a seperate distribution plot of "business_year" column along with the doc_id" column

In [ ]:

```
sns.kdeplot(X_train['buisness_year'], color='y')
```

In [ ]:

```
sns.catplot(data=X_train,x='buisness_year',y='doc_id')
```

Feature Engineering

Display and describe the X_train dataframe

In [ ]:

```
X_train
```

In [ ]:

```
X_train.describe()
```

The "business_code" column inside X_train, is a categorical column, so you need to perform Labelencoder on that particular column

- Note - call the Label Encoder from sklearn library and use the fit() function on "business_code" column
- Note - Please fill in the blanks (two) to complete this code

In [ ]:

```
from sklearn.preprocessing import LabelEncoder
business_coder = LabelEncoder()
business_coder.fit(X_train['business_code'])
```

You are expected to store the value into a new column i.e. "business_code_enc"

- Note - For Training set you are expected to use fit_trainsform()
- Note - For Test set you are expected to use the trainsform()
- Note - For Validation set you are expected to use the trainsform()

- Partial code is provided, please fill in the blanks

In [ ]:

```
X_train['business_code_enc'] = business_coder.fit_transform(X_train['business_code'])
```

In [ ]:

```
X_val['business_code_enc'] = business_coder.transform(X_val['business_code'])
X_test['business_code_enc'] = business_coder.transform(X_test['business_code'])
```

Display "business_code" and "business_code_enc" together from X_train dataframe

In [ ]:

```
X_train[['business_code','business_code_enc']]
```

Create a function called "custom" for dropping the columns 'business_code' from train, test and validation dataframe

- Note - Fill in the blank to complete the code

In [ ]:

```
def custom(col ,traindf = X_train,valdf = X_val,testdf = X_test):
    traindf.drop(col, axis =1,inplace=True)
    valdf.drop(col,axis=1 , inplace=True)
    testdf.drop(col,axis=1 , inplace=True)

    return traindf,valdf ,testdf
```

Call the function by passing the column name which needed to be dropped from train, test and validation dataframes. Return updated dataframes to be stored in X_train ,X_val, X_test

- Note = Fill in the blank to complete the code

In [ ]:

```
X_train, X_val , X_test = custom(['business_code'])
```

Manually replacing str values with numbers, Here we are trying manually replace

the customer numbers with some specific values like, 'CCCA' as 1, 'CCU' as 2 and so on. Also we are converting the datatype "cust_number" field to int type.

- We are doing it for all the three dataframes as shown below. This is fully completed code. No need to modify anything here

In [ ]:

```
X_train['cust_number'] = X_train['cust_number'].str.replace('CCCA',"1").str.replace('CCU',"
X_test['cust_number'] = X_test['cust_number'].str.replace('CCCA',"1").str.replace('CCU',"2"
X_val['cust_number'] = X_val['cust_number'].str.replace('CCCA',"1").str.replace('CCU',"2").
```

It differs from LabelEncoder by handling new classes and providing a value for it [Unknown]. Unknown will be added in fit and transform will take care of new item. It gives unknown class id.

This will fit the encoder for all the unique values and introduce unknown value

- Note - Keep this code as it is, we will be using this later on.

In [ ]:

```
#For encoding unseen labels
class EncoderExt(object):
    def __init__(self):
        self.label_encoder = LabelEncoder()
    def fit(self, data_list):
        self.label_encoder = self.label_encoder.fit(list(data_list) + ['Unknown'])
        self.classes_ = self.label_encoder.classes_
        return self
    def transform(self, data_list):
        new_data_list = list(data_list)
        for unique_item in np.unique(data_list):
            if unique_item not in self.label_encoder.classes_:
                new_data_list = ['Unknown' if x==unique_item else x for x in new_data_list]
        return self.label_encoder.transform(new_data_list)
```

Use the user define Label Encoder function called "EncoderExt" for the "name_customer" column

- Note - Keep the code as it is, no need to change

In [ ]:

```
label_encoder = EncoderExt()
label_encoder.fit(X_train['name_customer'])
X_train['name_customer_enc']=label_encoder.transform(X_train['name_customer'])
X_val['name_customer_enc']=label_encoder.transform(X_val['name_customer'])
X_test['name_customer_enc']=label_encoder.transform(X_test['name_customer'])
```

As we have created the a new column "name_customer_enc", so now drop

"name_customer" column from all three dataframes

- Note - Keep the code as it is, no need to change

In [ ]:

```
X_train ,X_val, X_test = custom(['name_customer'])
```

Using Label Encoder for the "cust_payment_terms" column

- Note - Keep the code as it is, no need to change

In [ ]:

```
label_encoder1 = EncoderExt()
label_encoder1.fit(X_train['cust_payment_terms'])
X_train['cust_payment_terms_enc']=label_encoder1.transform(X_train['cust_payment_terms'])
X_val['cust_payment_terms_enc']=label_encoder1.transform(X_val['cust_payment_terms'])
X_test['cust_payment_terms_enc']=label_encoder1.transform(X_test['cust_payment_terms'])
```

In [ ]:

```
X_train ,X_val, X_test = custom(['cust_payment_terms'])
```

Check the datatype of all the columns of Train, Test and Validation dataframes realted to X

- Note - You are expected yo use dtype

In [ ]:

```
X_train.dtypes
```

In [ ]:

```
X_test.dtypes
```

In [ ]:

```
X_val.dtypes
```

From the above output you can notice their are multiple date columns with datetime format

In order to pass it into our model, we need to convert it into float format

You need to extract day, month and year from the "posting_date" column

1. Extract days from "posting_date" column and store it into a new column "day_of_postingdate" for train, test and validation dataset
2. Extract months from "posting_date" column and store it into a new column "month_of_postingdate" for train, test and validation dataset
3. Extract year from "posting_date" column and store it into a new column "year_of_postingdate" for train, test and validation dataset

- Note - You are supposed yo use
- dt.day
- dt.month
- dt.year

In [ ]:

```
X_train['day_of_postingdate'] = X_train['posting_date'].dt.day
X_train['month_of_postingdate'] = X_train['posting_date'].dt.month
X_train['year_of_postingdate'] = X_train['posting_date'].dt.year

X_val['day_of_postingdate'] = X_val['posting_date'].dt.day
X_val['month_of_postingdate'] = X_val['posting_date'].dt.month
X_val['year_of_postingdate'] = X_val['posting_date'].dt.year


X_test['day_of_postingdate'] = X_test['posting_date'].dt.day
X_test['month_of_postingdate'] = X_test['posting_date'].dt.month
X_test['year_of_postingdate'] = X_test['posting_date'].dt.year
```

pass the "posting_date" column into the Custom function for train, test and validation dataset

In [ ]:

```
X_train ,X_val, X_test = custom(['posting_date'])
```

You need to extract day, month and year from the "baseline_create_date" column

1. Extract days from "baseline_create_date" column and store it into a new column "day_of_createdate" for train, test and validation dataset
2. Extract months from "baseline_create_date" column and store it into a new column "month_of_createdate" for train, test and validation dataset
3. Extract year from "baseline_create_date" column and store it into a new column "year_of_createdate" for train, test and validation dataset

- Note - You are supposed yo use
- dt.day
- dt.month
- dt.year

- Note - Do as it is been shown in the previous two code boxes

Extracting Day, Month, Year for 'baseline_create_date' column

In [ ]:

```
X_train['day_of_createdate'] = X_train['baseline_create_date'].dt.day
X_train['month_of_createdate'] = X_train['baseline_create_date'].dt.month
X_train['year_of_createdate'] = X_train['baseline_create_date'].dt.year
X_test['day_of_createdate'] = X_test['baseline_create_date'].dt.day
X_test['month_of_createdate'] = X_test['baseline_create_date'].dt.month
X_test['year_of_createdate'] = X_test['baseline_create_date'].dt.year
X_val['day_of_createdate'] = X_val['baseline_create_date'].dt.day
X_val['month_of_createdate'] = X_val['baseline_create_date'].dt.month
X_val['year_of_createdate'] = X_val['baseline_create_date'].dt.year
```

pass the "baseline_create_date" column into the Custom function for train, test and validation dataset

In [ ]:

```
X_train,X_val,X_test = custom(['baseline_create_date'])
```

You need to extract day, month and year from the "due_in_date" column

1. Extract days from "due_in_date" column and store it into a new column "day_of_due" for train, test and validation dataset
2. Extract months from "due_in_date" column and store it into a new column "month_of_due" for train, test and validation dataset
3. Extract year from "due_in_date" column and store it into a new column "year_of_due" for train, test and validation dataset

- Note - You are supposed yo use
- dt.day
- dt.month
- dt.year
- Note - Do as it is been shown in the previous code

In [ ]:

```
X_train['day_of_due'] = X_train['due_in_date'].dt.day
X_train['month_of_due'] = X_train['due_in_date'].dt.month
X_train['year_of_due'] = X_train['due_in_date'].dt.year
X_test['day_of_due'] = X_test['due_in_date'].dt.day
X_test['month_of_due'] = X_test['due_in_date'].dt.month
X_test['year_of_due'] = X_test['due_in_date'].dt.year
X_val['day_of_due'] = X_val['due_in_date'].dt.day
X_val['month_of_due'] = X_val['due_in_date'].dt.month
X_val['year_of_due'] = X_val['due_in_date'].dt.year
```

pass the "due_in_date" column into the Custom function for train, test and validation dataset

In [ ]:

```
X_train,X_val,X_test = custom(['due_in_date'])
```

Check for the datatypes for train, test and validation set again

- Note - all the data type should be in either int64 or float64 format

In [ ]:

```
X_train.dtypes
```

In [ ]:

```
X_test.dtypes
```

In [ ]:

```
X_val.dtypes
```

Feature Selection

Filter Method

- Calling the VarianceThreshold Function
- Note - Keep the code as it is, no need to change

In [ ]:

```
from sklearn.feature_selection import VarianceThreshold
constant_filter = VarianceThreshold(threshold=0)
constant_filter.fit(X_train)
len(X_train.columns[constant_filter.get_support()])
```

- Note - Keep the code as it is, no need to change

In [ ]:

```
constant_columns = [column for column in X_train.columns
                    if column not in X_train.columns[constant_filter.get_support()]]
print(len(constant_columns))
```

- transpose the feature matrice
- print the number of duplicated features
- select the duplicated features columns names
- Note - Keep the code as it is, no need to change

In [ ]:

```
x_train_T = X_train.T
print(x_train_T.duplicated().sum())
duplicated_columns = x_train_T[x_train_T.duplicated()].index.values
```

Filtering depending upon correlation matrix value

- We have created a function called handling correlation which is going to return fields based on the correlation matrix value with a threshold of 0.8
- Note - Keep the code as it is, no need to change

In [ ]:

```
def handling_correlation(X_train,threshold=0.8):
    corr_features = set()
    corr_matrix = X_train.corr()
    for i in range(len(corr_matrix .columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i, j]) >threshold:
                colname = corr_matrix.columns[i]
                corr_features.add(colname)
    return list(corr_features)
```

- Note : Here we are trying to find out the relevant fields, from X_train
- Please fill in the blanks to call handling_correlation() function with a threshold value of 0.85

In [ ]:

```
train=X_train.copy()
handling_correlation(train.copy(),threshold=0.85)
```

Heatmap for X_train

- Note - Keep the code as it is, no need to change

In [ ]:

```
colormap = plt.cm.RdBu
plt.figure(figsize=(14,12))
plt.title('Pearson Correlation of Features', y=1.05, size=20)
sns.heatmap(X_train.merge(y_train , on = X_train.index ).corr(),linewidths=0.1,vmax=1.0,
            square=True, cmap='gist_rainbow_r', linecolor='white', annot=True)
```

Calling variance threshold for threshold value = 0.8

- Note - Fill in the blanks to call the appropriate method

In [ ]:

```
from sklearn.feature_selection import VarianceThreshold
sel = VarianceThreshold(0.8)
sel.fit(X_train)
```

In [ ]:

```
sel.variances_
```

Important features columns are

- 'year_of_createdate'
- 'year_of_due'
- 'day_of_createdate'
- 'year_of_postingdate'
- 'month_of_due'
- 'month_of_createdate'

Modelling

Now you need to compare with different machine learning models, and needs to find out the best predicted model

- Linear Regression
- Decision Tree Regression
- Random Forest Regression
- Support Vector Regression
- Extreme Gradient Boost Regression

You need to make different blank list for different evaluation matrix

- MSE
- R2
- Algorithm

In [ ]:

```
MSE_Score = []
R2_Score = []
Algorithm = []
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

You need to start with the baseline model Linear Regression

- Step 1 : Call the Linear Regression from sklearn library
- Step 2 : make an object of Linear Regression

- Step 3 : fit the X_train and y_train dataframe into the object
- Step 4 : Predict the output by passing the X_test Dataset into predict function

- Note - Append the Algorithm name into the algorithm list for tracking purpose

In [ ]:

```
from sklearn.linear_model import LinearRegression
Algorithm.append('LinearRegression')
regressor = LinearRegression()
regressor.fit(X_train, y_train)
predicted= regressor.predict(X_test)
```

Check for the

- Mean Square Error
- R Square Error

for y_test and predicted dataset and store those data inside respective list for comparison

In [ ]:

```
MSE_Score.append(mean_squared_error(y_test, predicted))
R2_Score.append(r2_score(y_test, predicted))
```

Check the same for the Validation set also

In [ ]:

```
predict_test= regressor.predict(X_val)
print(mean_squared_error(y_val, predict_test, squared=False))
print(r2_score(y_val,predict_test))
```

Display The Comparison Lists

In [ ]:

```
for i in Algorithm, MSE_Score, R2_Score:
    print(i,end=',')
```

You need to start with the baseline model Support Vector Regression

- Step 1 : Call the Support Vector Regressor from sklearn library
- Step 2 : make an object of SVR
- Step 3 : fit the X_train and y_train dataframe into the object
- Step 4 : Predict the output by passing the X_test Dataset into predict function

- Note - Append the Algorithm name into the algorithm list for tracking purpose

In [ ]:

```
from sklearn.svm import SVR
Algorithm.append('Support_Vector')
svr = SVR()
svr.fit(X_train, y_train)
predicted= svr.predict(X_test)
```

Check for the

- Mean Square Error
- R Square Error

for "y_test" and "predicted" dataset and store those data inside respective list for comparison

In [ ]:

```
MSE_Score.append(mean_squared_error(y_test, predicted))
R2_Score.append(r2_score(y_test, predicted))
```

Check the same for the Validation set also

In [ ]:

```
predict_test= svr.predict(X_val)
print(mean_squared_error(y_val, predict_test, squared=False))
print(r2_score(y_val,predict_test))
```

Display The Comparison Lists

In [ ]:

```
for i in Algorithm, MSE_Score, R2_Score:
 print(i,end=',')
```

Your next model would be Decision Tree Regression

- Step 1 : Call the Decision Tree Regressor from sklearn library
- Step 2 : make an object of Decision Tree
- Step 3 : fit the X_train and y_train dataframe into the object
- Step 4 : Predict the output by passing the X_test Dataset into predict function

- Note - Append the Algorithm name into the algorithm list for tracking purpose

In [ ]:

```
from sklearn import tree
Algorithm.append('Decision Tree')
DTC=tree.DecisionTreeRegressor()
DTC.fit(X_train,y_train)
predicted=DTC.predict(X_test)
```

Check for the

- Mean Square Error
- R Square Error

for y_test and predicted dataset and store those data inside respective list for comparison

In [ ]:

```
MSE_Score.append(mean_squared_error(y_test, predicted))
R2_Score.append(r2_score(y_test, predicted))
```

Check the same for the Validation set also

In [ ]:

```
predict_test= DTC.predict(X_val)
print(mean_squared_error(y_val, predict_test, squared=False))
print(r2_score(y_val,predict_test))
```

Display The Comparison Lists

In [ ]:

```
for i in Algorithm, MSE_Score, R2_Score:
 print(i,end=',')
```

Your next model would be Random Forest Regression

- Step 1 : Call the Random Forest Regressor from sklearn library
- Step 2 : make an object of Random Forest
- Step 3 : fit the X_train and y_train dataframe into the object
- Step 4 : Predict the output by passing the X_test Dataset into predict function

- Note - Append the Algorithm name into the algorithm list for tracking purpose

In [ ]:

```
from sklearn.ensemble import RandomForestRegressor
RF=RandomForestRegressor()
Algorithm.append('Random Forest')
RF.fit(X_train,y_train)
predicted=RF.predict(X_test)
```

Check for the

- Mean Square Error
- R Square Error

for y_test and predicted dataset and store those data inside respective list for comparison

In [ ]:

```
MSE_Score.append(mean_squared_error(y_test, predicted))
R2_Score.append(r2_score(y_test, predicted))
```

Check the same for the Validation set also

In [ ]:

```
predict_test= RF.predict(X_val)
print(mean_squared_error(y_val, predict_test, squared=False))
print(r2_score(y_val,predict_test))
```

Display The Comparison Lists

In [ ]:

```
for i in Algorithm, MSE_Score, R2_Score:
 print(i,end=',')
```

The last but not the least model would be XGBoost or Extreme Gradient Boost Regression

- Step 1 : Call the XGBoost Regressor from xgb library
- Step 2 : make an object of Xgboost
- Step 3 : fit the X_train and y_train dataframe into the object
- Step 4 : Predict the output by passing the X_test Dataset into predict function

- Note - Append the Algorithm name into the algorithm list for tracking purpose### Extreme Gradient Boost Regression
- Note - No need to change the code

In [ ]:

```
import xgboost as xgb
Algorithm.append('XGB Regressor')
regressor = xgb.XGBRegressor()
regressor.fit(X_train, y_train)
predicted = regressor.predict(X_test)
```

Check for the

- Mean Square Error
- R Square Error

for y_test and predicted dataset and store those data inside respective list for comparison

In [ ]:

```
MSE_Score.append(mean_squared_error(y_test, predicted))
R2_Score.append(r2_score(y_test, predicted))
```

Check the same for the Validation set also

In [ ]:

```
predict_test= regressor.predict(X_val)
print(mean_squared_error(y_val, predict_test, squared=False))
print(r2_score(y_val,predict_test))
```

Display The Comparison Lists

In [ ]:

```
for i in Algorithm, MSE_Score, R2_Score:
 print(i,end=',')
```

You need to make the comparison list into a comparison dataframe

In [ ]:

```
comparison=pd.DataFrame(list(zip(Algorithm,MSE_Score,R2_Score)),columns=['Algorithm','MSE',
```

In [ ]:

```
comparison
```

Now from the Comparison table, you need to choose the best fit

model

- Step 1 - Fit X_train and y_train inside the model
- Step 2 - Predict the X_test dataset
- Step 3 - Predict the X_val dataset

- Note - No need to change the code

In [ ]:

```
regressorfinal = xgb.XGBRegressor()
regressorfinal.fit(X_train, y_train)
predictedtrain=regressorfinal.predict(X_train)
predictedfinal = regressorfinal.predict(X_test)
predict_testfinal = regressorfinal.predict(X_val)
```

Calculate the Mean Square Error for test dataset

- Note - No need to change the code

In [ ]:

```
mean_squared_error(y_test,predictedfinal,squared=False)
```

Calculate the mean Square Error for validation dataset

In [ ]:

```
mean_squared_error(y_val,predict_testfinal)
```

Calculate the R2 score for test

In [ ]:

```
r2_score(y_test,predictedfinal)
```

Calculate the R2 score for Validation

In [ ]:

```
r2_score(y_val,predict_testfinal)
```

Calculate the Accuracy for train Dataset

In [ ]:

```
r2_score(y_train,predictedtrain)*100
```

Calculate the accuracy for validation

In [ ]:

```
r2_score(y_val,predict_testfinal)*100
```

Calculate the accuracy for test

In [ ]:

```
r2_score(y_test,predictedfinal)*100
```

Specify the reason behind choosing your machine learning model

- Note : Provide your answer as a text here

Now you need to pass the Nulldata dataframe into this machine learning model

In order to pass this Nulldata dataframe into the ML model, we need to perform the following

- Step 1 : Label Encoding
- Step 2 : Day, Month and Year extraction
- Step 3 : Change all the column data type into int64 or float64
- Step 4 : Need to drop the useless columns

Display the Nulldata

In [ ]:

```
nulldata
```

Check for the number of rows and columns in the nulldata

In [ ]:

```
nulldata.shape
```

Check the Description and Information of the nulldata

In [ ]:

```
nulldata.describe()
```

In [ ]:

```
nulldata.info()
```

Storing the Nulldata into a different dataset

for BACKUP

In [ ]:

```
nullcopy=nulldata.copy()
```

Call the Label Encoder for Nulldata

- Note - you are expected to fit "business_code" as it is a categorical variable
- Note - No need to change the code

In [ ]:

```
from sklearn.preprocessing import LabelEncoder
business_codern = LabelEncoder()
business_codern.fit(nulldata['business_code'])
nulldata['business_code_enc'] = business_codern.transform(nulldata['business_code'])
```

Now you need to manually replacing str values with numbers

- Note - No need to change the code

In [ ]:

```
nulldata['cust_number'] = nulldata['cust_number'].str.replace('CCCA',"1").str.replace('CCU'
```

You need to extract day, month and year from the "clear_date", "posting_date", "due_in_date", "baseline_create_date" columns

1. Extract day from "clear_date" column and store it into 'day_of_cleardate'

2. Extract month from "clear_date" column and store it into 'month_of_cleardate'

3. Extract year from "clear_date" column and store it into 'year_of_cleardate'

4. Extract day from "posting_date" column and store it into 'day_of_postingdate'

5. Extract month from "posting_date" column and store it into 'month_of_postingdate'

6. Extract year from "posting_date" column and store it into 'year_of_postingdate'

7. Extract day from "due_in_date" column and store it into 'day_of_due'

8. Extract month from "due_in_date" column and store it into 'month_of_due'

9. Extract year from "due_in_date" column and store it into 'year_of_due'

10. Extract day from "baseline_create_date" column and store it into 'day_of_createdate'

11. Extract month from "baseline_create_date" column and store it into 'month_of_createdate'

12. Extract year from "baseline_create_date" column and store it into 'year_of_createdate'

- Note - You are supposed To use -
- dt.day
- dt.month
- dt.year

In [ ]:

```
nulldata['day_of_due'] = nulldata['due_in_date'].dt.day
nulldata['month_of_due'] = nulldata['due_in_date'].dt.month
nulldata['year_of_due'] = nulldata['due_in_date'].dt.year
nulldata['day_of_posting'] = nulldata['posting_date'].dt.day
nulldata['month_of_posting'] = nulldata['posting_date'].dt.month
nulldata['year_of_posting'] = nulldata['posting_date'].dt.year
nulldata['day_of_base'] = nulldata['baseline_create_date'].dt.day
nulldata['month_of_base'] = nulldata['baseline_create_date'].dt.month
nulldata['year_of_base'] = nulldata['baseline_create_date'].dt.year
```

Use Label Encoder1 of all the following columns -

- 'cust_payment_terms' and store into 'cust_payment_terms_enc'
- 'business_code' and store into 'business_code_enc'
- 'name_customer' and store into 'name_customer_enc'

Note - No need to change the code

In [ ]:

```
nulldata['cust_payment_terms_enc']=label_encoder1.transform(nulldata['cust_payment_terms'])
nulldata['business_code_enc']=label_encoder1.transform(nulldata['business_code'])
nulldata['name_customer_enc']=label_encoder.transform(nulldata['name_customer'])
```

Check for the datatypes of all the columns of Nulldata

In [ ]:

```
nulldata.dtypes
```

Now you need to drop all the unnecessary columns -

- 'business_code'
- "baseline_create_date"
- "due_in_date"
- "posting_date"
- "name_customer"
- "clear_date"
- "cust_payment_terms"
- 'day_of_cleardate'
- "month_of_cleardate"
- "year_of_cleardate"

In [ ]:

```
nulldata.drop(["business_code",
"baseline_create_date",
"due_in_date",
"posting_date",
"name_customer",
"clear_date",
"cust_payment_terms",
 "clear_date"],axis=1,inplace=True)
```

Check the information of the "nulldata" dataframe

In [ ]:

```
nulldata.info()
```

Compare "nulldata" with the "X_test" dataframe

- use info() method

In [ ]:

```
X_test.info()
```

You must have noticed that there is a mismatch in the column sequence while compairing the dataframes

- Note - In order to fed into the machine learning model, you need to edit the sequence of "nulldata", similar to the "X_test" dataframe

- Display all the columns of the X_test dataframe
- Display all the columns of the Nulldata dataframe
- Store the Nulldata with new sequence into a new dataframe

- Note - The code is given below, no need to change

In [ ]:

```
X_test.columns
```

In [ ]:

```
nulldata.columns
```

In [ ]:

```
nulldata2=nulldata[['cust_number', 'buisness_year', 'doc_id', 'converted_usd',
 'business_code_enc', 'name_customer_enc', 'cust_payment_terms_enc',
 'day_of_posting', 'month_of_posting', 'year_of_posting',
 'day_of_base', 'month_of_base', 'year_of_base',
 'day_of_due', 'month_of_due', 'year_of_due']]
```

Display the Final Dataset

In [ ]:

```
nulldata2
```

Now you can pass this dataset into you final model and store it into "final_result"

In [ ]:

```
final_result=regressorfinal.predict(nulldata2)
```

you need to make the final_result as dataframe, with a column name "avg_delay"

- Note - No need to change the code

In [ ]:

```
final_result = pd.Series(final_result,name='avg_delay')
```

Display the "avg_delay" column

In [ ]:

```
final_result
```

Now you need to merge this final_result dataframe with the BACKUP of "nulldata" Dataframe which we have created in earlier steps

In [ ]:

```
nullcopy.reset_index(drop=True,inplace=True)
Final = nullcopy.merge(final_result , on = nulldata.index )
```

Display the "Final" dataframe

In [ ]:

```
Final
```

Check for the Number of Rows and Columns in your "Final" dataframe

In [ ]:

```
Final.shape
```

Now, you need to do convert the below fields back into date and time format

- Convert "due_in_date" into datetime format
- Convert "avg_delay" into datetime format
- Create a new column "clear_date" and store the sum of "due_in_date" and "avg_delay"
- display the new "clear_date" column
- Note - Code is given below, no need to change

In [ ]:

```
Final['clear_date'] = pd.to_datetime(Final['due_in_date']) + pd.to_timedelta(Final['avg_del
```

Display the "clear_date" column

In [ ]:

```
Final['clear_date']
```

Convert the average delay into number of days format

- Note - Formula = avg_delay//(24 * 3600)
- Note - full code is given for this, no need to change

In [ ]:

```
Final['avg_delay'] = Final.apply(lambda row: row.avg_delay//(24 * 3600), axis = 1)
```

Display the "avg_delay" column

In [ ]:

```
Final['avg_delay']
```

Now you need to convert average delay column into bucket

- Need to perform binning
- create a list of bins i.e. bins= [0,15,30,45,60,100]
- create a list of labels i.e. labels = ['0-15','16-30','31-45','46-60','Greatar than 60']
- perform binning by using cut() function from "Final" dataframe

- Please fill up the first two rows of the code

In [ ]:

```
bins=[0,15,30,45,60,100]
labels =['0-15','16-30','31-45','46-60','Greatar than 60']
Final['Aging Bucket'] = pd.cut(Final['avg_delay'], bins=bins, labels=labels, right=False)
```

Now you need to drop "key_0" and "avg_delay" columns from the "Final" Dataframe

In [ ]:

```
Final.drop(['key_0','avg_delay'],axis=1,inplace=True)
```

Display the count of each categoty of new "Aging Bucket" column

In [ ]:

```
Final['Aging Bucket'].value_counts()
```

Display your final dataset with aging buckets

In [ ]:

```
Final.sample(5)
```

Store this dataframe into the .csv format

In [ ]:

```
FinalCSV=Final.to_csv(index=False)
```

END OF THE PROJECT