Big_Data_Analytics_Project_Data_Transformation_&_Modeling_Mar16

March 16, 2025

[74]:
```python
import pandas as pd

url = "https://archive.ics.uci.edu/static/public/350/data.csv"
data = pd.read_csv(url, sep= ',')
print(data.head())
```

```
   ID      X1  X2  X3  X4  X5  X6  X7  X8  X9  …    X15    X16    X17   X18  \
0   1   20000   2   2   1  24   2   2  -1  -1  …      0      0      0     0
1   2  120000   2   2   2  26  -1   2   0   0  …   3272   3455   3261     0
2   3   90000   2   2   2  34   0   0   0   0  …  14331  14948  15549  1518
3   4   50000   2   2   1  37   0   0   0   0  …  28314  28959  29547  2000
4   5   50000   1   2   1  57  -1   0  -1   0  …  20940  19146  19131  2000

     X19    X20   X21   X22   X23  Y
0    689      0     0     0     0  1
1   1000   1000  1000     0  2000  1
2   1500   1000  1000  1000  5000  0
3   2019   1200  1100  1069  1000  0
4  36681  10000  9000   689   679  0

[5 rows x 25 columns]
```

[75]:
```python
#Renaming columns
data.rename(columns={'X1': 'LIMIT_BAL', 'X2': 'SEX', 'X3': 'EDUCATION', 'X4': 'MARRIAGE', 'X5': 'AGE', 'X6': 'PAY_0', 'X7': 'PAY_2','X8': 'PAY_3', 'X9': 'PAY_4', 'X10': 'PAY_5', 'X11': 'PAY_6', 'X12': 'BILL_AMT1', 'X13': 'BILL_AMT2', 'X14': 'BILL_AMT3', 'X15': 'BILL_AMT4', 'X16': 'BILL_AMT5', 'X17': 'BILL_AMT6', 'X18': 'PAY_AMT1', 'X19': 'PAY_AMT2', 'X20': 'PAY_AMT3', 'X21': 'PAY_AMT4', 'X22': 'PAY_AMT5', 'X23': 'PAY_AMT6'}, inplace=True)
data.head()
```

[75]:
```
   ID  LIMIT_BAL  SEX  EDUCATION  MARRIAGE  AGE  PAY_0  PAY_2  PAY_3  PAY_4  \
0   1      20000    2          2         1   24      2      2     -1     -1
1   2     120000    2          2         2   26     -1      2      0      0
2   3      90000    2          2         2   34      0      0      0      0
3   4      50000    2          2         1   37      0      0      0      0
```

```
4   5      50000    1          2         1   57      -1       0      -1       0

   …  BILL_AMT4  BILL_AMT5  BILL_AMT6  PAY_AMT1  PAY_AMT2  PAY_AMT3  \
0  …          0          0          0         0       689         0
1  …       3272       3455       3261         0      1000      1000
2  …      14331      14948      15549      1518      1500      1000
3  …      28314      28959      29547      2000      2019      1200
4  …      20940      19146      19131      2000     36681     10000

   PAY_AMT4  PAY_AMT5  PAY_AMT6  Y
0         0         0         0  1
1      1000         0      2000  1
2      1000      1000      5000  0
3      1100      1069      1000  0
4      9000       689       679  0

[5 rows x 25 columns]
```

[76]:
```python
# Replacing education values = 0, 5 and 6 with 4, since 0, 5 and 6 are not defined

fill = (data.EDUCATION == 0) | (data.EDUCATION == 5) | (data.EDUCATION == 6)
data.loc[fill, 'EDUCATION'] = 4

print('EDUCATION ' + str(sorted(data['EDUCATION'].unique())))
```

```
EDUCATION [1, 2, 3, 4]
```

[77]:
```python
# Replacing marital status value = 0 to 3, since 0 is not defined

fill = (data.MARRIAGE == 0)
data.loc[fill, 'MARRIAGE'] = 3

print('MARRIAGE ' + str(sorted(data['MARRIAGE'].unique())))
```

```
MARRIAGE [1, 2, 3]
```

[78]:
```python
# Applying One-Hot Encoding technique to categorical variables

from sklearn.preprocessing import OneHotEncoder

categorical_variables = ['SEX', 'EDUCATION', 'MARRIAGE', 'PAY_0', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6']

encoder = OneHotEncoder(sparse_output=False)
one_hot_encoded = encoder.fit_transform(data[categorical_variables])
```

```
one_hot_df = pd.DataFrame(one_hot_encoded, columns=encoder.
 ↪get_feature_names_out(categorical_variables))

df_encoded = pd.concat([data, one_hot_df], axis=1)

df_encoded = df_encoded.drop(categorical_variables, axis=1)
print(f"Encoded Employee data : \n{df_encoded}")
```

```
Encoded Employee data :
          ID  LIMIT_BAL  AGE  BILL_AMT1  BILL_AMT2  BILL_AMT3  BILL_AMT4  \
0          1      20000   24       3913       3102        689          0
1          2     120000   26       2682       1725       2682       3272
2          3      90000   34      29239      14027      13559      14331
3          4      50000   37      46990      48233      49291      28314
4          5      50000   57       8617       5670      35835      20940
…        …          …   …        …          …          …          …
29995  29996     220000   39     188948     192815     208365      88004
29996  29997     150000   43       1683       1828       3502       8979
29997  29998      30000   37       3565       3356       2758      20878
29998  29999      80000   41      -1645      78379      76304      52774
29999  30000      50000   46      47929      48905      49764      36535

       BILL_AMT5  BILL_AMT6  PAY_AMT1  …  PAY_6_-2  PAY_6_-1  PAY_6_0  \
0              0          0         0  …       1.0       0.0      0.0
1           3455       3261         0  …       0.0       0.0      0.0
2          14948      15549      1518  …       0.0       0.0      1.0
3          28959      29547      2000  …       0.0       0.0      1.0
4          19146      19131      2000  …       0.0       0.0      1.0
…            …          …        …  …       …        …
29995      31237      15980      8500  …       0.0       0.0      1.0
29996       5190          0      1837  …       0.0       0.0      1.0
29997      20582      19357         0  …       0.0       0.0      1.0
29998      11855      48944     85900  …       0.0       1.0      0.0
29999      32428      15313      2078  …       0.0       0.0      1.0

       PAY_6_2  PAY_6_3  PAY_6_4  PAY_6_5  PAY_6_6  PAY_6_7  PAY_6_8
0          0.0      0.0      0.0      0.0      0.0      0.0      0.0
1          1.0      0.0      0.0      0.0      0.0      0.0      0.0
2          0.0      0.0      0.0      0.0      0.0      0.0      0.0
3          0.0      0.0      0.0      0.0      0.0      0.0      0.0
4          0.0      0.0      0.0      0.0      0.0      0.0      0.0
…          …      …      …      …      …      …      …
29995      0.0      0.0      0.0      0.0      0.0      0.0      0.0
29996      0.0      0.0      0.0      0.0      0.0      0.0      0.0
29997      0.0      0.0      0.0      0.0      0.0      0.0      0.0
29998      0.0      0.0      0.0      0.0      0.0      0.0      0.0
29999      0.0      0.0      0.0      0.0      0.0      0.0      0.0
```

[30000 rows x 89 columns]

[79]:
```python
numeric_variables = ['LIMIT_BAL', 'AGE', 'BILL_AMT1','BILL_AMT2','BILL_AMT3','BILL_AMT4','BILL_AMT5','BILL_AMT6', 'PAY_AMT1','PAY_AMT2','PAY_AMT3','PAY_AMT4','PAY_AMT5','PAY_AMT6']

#Calculating and printing the Pearson correlation matrix
print("Pearson Correlation Matrix of numeric variables:")
pearson_correlation_matrix = data[numeric_variables].corr().round(2)
pearson_correlation_matrix
```

Pearson Correlation Matrix of numeric variables:

[79]:

	LIMIT_BAL	AGE	BILL_AMT1	BILL_AMT2	BILL_AMT3	BILL_AMT4	\
LIMIT_BAL	1.00	0.14	0.29	0.28	0.28	0.29	
AGE	0.14	1.00	0.06	0.05	0.05	0.05	
BILL_AMT1	0.29	0.06	1.00	0.95	0.89	0.86	
BILL_AMT2	0.28	0.05	0.95	1.00	0.93	0.89	
BILL_AMT3	0.28	0.05	0.89	0.93	1.00	0.92	
BILL_AMT4	0.29	0.05	0.86	0.89	0.92	1.00	
BILL_AMT5	0.30	0.05	0.83	0.86	0.88	0.94	
BILL_AMT6	0.29	0.05	0.80	0.83	0.85	0.90	
PAY_AMT1	0.20	0.03	0.14	0.28	0.24	0.23	
PAY_AMT2	0.18	0.02	0.10	0.10	0.32	0.21	
PAY_AMT3	0.21	0.03	0.16	0.15	0.13	0.30	
PAY_AMT4	0.20	0.02	0.16	0.15	0.14	0.13	
PAY_AMT5	0.22	0.02	0.17	0.16	0.18	0.16	
PAY_AMT6	0.22	0.02	0.18	0.17	0.18	0.18	

	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2	PAY_AMT3	PAY_AMT4	\
LIMIT_BAL	0.30	0.29	0.20	0.18	0.21	0.20	
AGE	0.05	0.05	0.03	0.02	0.03	0.02	
BILL_AMT1	0.83	0.80	0.14	0.10	0.16	0.16	
BILL_AMT2	0.86	0.83	0.28	0.10	0.15	0.15	
BILL_AMT3	0.88	0.85	0.24	0.32	0.13	0.14	
BILL_AMT4	0.94	0.90	0.23	0.21	0.30	0.13	
BILL_AMT5	1.00	0.95	0.22	0.18	0.25	0.29	
BILL_AMT6	0.95	1.00	0.20	0.17	0.23	0.25	
PAY_AMT1	0.22	0.20	1.00	0.29	0.25	0.20	
PAY_AMT2	0.18	0.17	0.29	1.00	0.24	0.18	
PAY_AMT3	0.25	0.23	0.25	0.24	1.00	0.22	
PAY_AMT4	0.29	0.25	0.20	0.18	0.22	1.00	
PAY_AMT5	0.14	0.31	0.15	0.18	0.16	0.15	
PAY_AMT6	0.16	0.12	0.19	0.16	0.16	0.16	

	PAY_AMT5	PAY_AMT6

LIMIT_BAL	0.22	0.22
AGE	0.02	0.02
BILL_AMT1	0.17	0.18
BILL_AMT2	0.16	0.17
BILL_AMT3	0.18	0.18
BILL_AMT4	0.16	0.18
BILL_AMT5	0.14	0.16
BILL_AMT6	0.31	0.12
PAY_AMT1	0.15	0.19
PAY_AMT2	0.18	0.16
PAY_AMT3	0.16	0.16
PAY_AMT4	0.15	0.16
PAY_AMT5	1.00	0.15
PAY_AMT6	0.15	1.00

[80]:

```
# Printing summary statistics of numeric variables before replacing outliers
  ↪with median:
print("\nSummary Statistics of numeric variables:")
data[numeric_variables].describe().transpose()
```

Summary Statistics of numeric variables:

[80]:

	count	mean	std	min	25%
LIMIT_BAL	30000.0	167484.322667	129747.661567	10000.0	50000.00
AGE	30000.0	35.485500	9.217904	21.0	28.00
BILL_AMT1	30000.0	51223.330900	73635.860576	-165580.0	3558.75
BILL_AMT2	30000.0	49179.075167	71173.768783	-69777.0	2984.75
BILL_AMT3	30000.0	47013.154800	69349.387427	-157264.0	2666.25
BILL_AMT4	30000.0	43262.948967	64332.856134	-170000.0	2326.75
BILL_AMT5	30000.0	40311.400967	60797.155770	-81334.0	1763.00
BILL_AMT6	30000.0	38871.760400	59554.107537	-339603.0	1256.00
PAY_AMT1	30000.0	5663.580500	16563.280354	0.0	1000.00
PAY_AMT2	30000.0	5921.163500	23040.870402	0.0	833.00
PAY_AMT3	30000.0	5225.681500	17606.961470	0.0	390.00
PAY_AMT4	30000.0	4826.076867	15666.159744	0.0	296.00
PAY_AMT5	30000.0	4799.387633	15278.305679	0.0	252.50
PAY_AMT6	30000.0	5215.502567	17777.465775	0.0	117.75

	50%	75%	max
LIMIT_BAL	140000.0	240000.00	1000000.0
AGE	34.0	41.00	79.0
BILL_AMT1	22381.5	67091.00	964511.0
BILL_AMT2	21200.0	64006.25	983931.0
BILL_AMT3	20088.5	60164.75	1664089.0
BILL_AMT4	19052.0	54506.00	891586.0
BILL_AMT5	18104.5	50190.50	927171.0

```
BILL_AMT6   17071.0   49198.25    961664.0
PAY_AMT1     2100.0    5006.00    873552.0
PAY_AMT2     2009.0    5000.00   1684259.0
PAY_AMT3     1800.0    4505.00    896040.0
PAY_AMT4     1500.0    4013.25    621000.0
PAY_AMT5     1500.0    4031.50    426529.0
PAY_AMT6     1500.0    4000.00    528666.0
```

[81]:
```
# Defining function to replace outliers with the median
def replace_outliers_with_median(data, column):
    median = data[column].median()
    q1 = data[column].quantile(0.25)
    q3 = data[column].quantile(0.75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr

    data[column] = data[column].apply(lambda x: median if x < lower_bound or x␣
 ↪> upper_bound else x)
```

[82]:
```
# Applying the function to the columns with outliers
replace_outliers_with_median(data, 'LIMIT_BAL')
replace_outliers_with_median(data, 'BILL_AMT1')
replace_outliers_with_median(data, 'BILL_AMT2')
replace_outliers_with_median(data, 'BILL_AMT3')
replace_outliers_with_median(data, 'BILL_AMT4')
replace_outliers_with_median(data, 'BILL_AMT5')
replace_outliers_with_median(data, 'BILL_AMT6')
replace_outliers_with_median(data, 'PAY_AMT1')
replace_outliers_with_median(data, 'PAY_AMT2')
replace_outliers_with_median(data, 'PAY_AMT3')
replace_outliers_with_median(data, 'PAY_AMT4')
replace_outliers_with_median(data, 'PAY_AMT5')
replace_outliers_with_median(data, 'PAY_AMT6')
```

[83]:
```
print("\nSummary Statistics of numeric variables after replacing outliers with␣
 ↪median:")
data[numeric_variables].describe().transpose()
```

```
Summary Statistics of numeric variables after replacing outliers with median:
```

[83]:
```
             count          mean            std      min       25%       50%  \
LIMIT_BAL  30000.0  164824.322667  125192.989579  10000.0  50000.00  140000.0
AGE        30000.0      35.485500       9.217904     21.0     28.00      34.0
BILL_AMT1  30000.0   33109.792100   37794.502441 -15308.0   3563.00   22381.5
BILL_AMT2  30000.0   31669.887567   36414.965831 -69777.0   2984.75   21198.5
```

```
BILL_AMT3  30000.0   29736.798283   34293.746628 -61506.0   2667.75   20088.5
BILL_AMT4  30000.0   26625.608833   30764.323883 -65167.0   2329.00   19052.0
BILL_AMT5  30000.0   24247.883050   28331.916539 -61372.0   1763.75   18104.5
BILL_AMT6  30000.0   23287.670000   27946.193005 -57060.0   1259.75   17071.0
PAY_AMT1   30000.0    2681.008300    2557.378286       0.0   1000.00    2100.0
PAY_AMT2   30000.0    2586.259267    2533.473459       0.0    833.00    2009.0
PAY_AMT3   30000.0    2267.026400    2396.721279       0.0    390.00    1800.0
PAY_AMT4   30000.0    1911.001400    2056.702179       0.0    296.00    1500.0
PAY_AMT5   30000.0    1926.580500    2075.388113       0.0    252.50    1500.0
PAY_AMT6   30000.0    1893.753100    2071.970037       0.0    117.75    1500.0

                 75%       max
LIMIT_BAL  240000.00  520000.0
AGE            41.00      79.0
BILL_AMT1   48707.50  162296.0
BILL_AMT2   47812.25  155508.0
BILL_AMT3   44887.75  146410.0
BILL_AMT4   37803.00  132754.0
BILL_AMT5   32030.50  122830.0
BILL_AMT6   30563.00  121062.0
PAY_AMT1     3706.00   11013.0
PAY_AMT2     3500.00   11249.0
PAY_AMT3     3005.00   10673.0
PAY_AMT4     2816.25    9584.0
PAY_AMT5     2913.50    9700.0
PAY_AMT6     2853.50    9817.0
```

[84]:
```python
#Normalizing the numeric attributes
from sklearn.preprocessing import MinMaxScaler

# Initializing the scaler
scaler = MinMaxScaler()
# Fitting and transforming the data
X = scaler.fit_transform(data[numeric_variables])

y = data['Y']
```

[85]:
```python
# Validating minimum and maximum values are set as 0.00 and 1.00 respectively
normalized_df_summary = pd.DataFrame(X).describe()
normalized_df_summary
```

[85]:
```
                  0             1             2             3             4  \
count  30000.000000  30000.000000  30000.000000  30000.000000  30000.000000
mean       0.303577      0.249750      0.272617      0.450305      0.438845
std        0.245476      0.158929      0.212802      0.161640      0.164940
min        0.000000      0.000000      0.000000      0.000000      0.000000
25%        0.078431      0.120690      0.106253      0.322976      0.308652
```

```
50%       0.254902      0.224138      0.212211      0.403824      0.392440
75%       0.450980      0.344828      0.360440      0.521958      0.511715
max       1.000000      1.000000      1.000000      1.000000      1.000000

                    5             6             7             8             9  \
count  30000.000000  30000.000000  30000.000000  30000.000000  30000.000000
mean       0.463784      0.464815      0.451082      0.243440      0.229910
std        0.155437      0.153809      0.156894      0.232214      0.225218
min        0.000000      0.000000      0.000000      0.000000      0.000000
25%        0.341025      0.342753      0.327415      0.090802      0.074051
50%        0.425518      0.431464      0.416181      0.190684      0.178594
75%        0.520258      0.507066      0.491927      0.336511      0.311139
max        1.000000      1.000000      1.000000      1.000000      1.000000

                   10            11            12            13
count  30000.000000  30000.000000  30000.000000  30000.000000
mean       0.212408      0.199395      0.198617      0.192905
std        0.224559      0.214597      0.213958      0.211059
min        0.000000      0.000000      0.000000      0.000000
25%        0.036541      0.030885      0.026031      0.011994
50%        0.168650      0.156511      0.154639      0.152796
75%        0.281552      0.293849      0.300361      0.290669
max        1.000000      1.000000      1.000000      1.000000
```

[86]:
```python
from sklearn.model_selection import train_test_split

# Splitting data into training and testing sets: training set 70%, test set 30%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)
```

[87]:
```python
X_train[0:5,]
```

[87]:
```
array([[0.45098039, 0.32758621, 0.08619175, 0.30972768, 0.29582139,
        0.32925763, 0.33317771, 0.32034224, 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        ],
       [0.07843137, 0.03448276, 0.10550438, 0.32072708, 0.30687874,
        0.35350973, 0.38643446, 0.32404756, 0.23136293, 0.20632945,
        0.44973297, 0.15651085, 0.06804124, 0.30355506],
       [0.07843137, 0.25862069, 0.34633792, 0.51018044, 0.53209469,
        0.57155633, 0.56975494, 0.51754416, 0.        , 0.41781492,
        0.        , 0.2090985 , 0.36082474, 0.        ],
       [0.37254902, 0.56896552, 0.70658882, 0.78763344, 0.7993228 ,
        0.8473886 , 0.88366033, 0.32034224, 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        ],
       [0.45098039, 0.24137931, 0.09758789, 0.31863639, 0.40062333,
        0.41566585, 0.40574478, 0.4475528 , 0.18314719, 0.17859365,
        0.10493769, 0.15651085, 0.15463918, 0.15279617]])
```

[88]:
```
from collections import Counter
# summarizing class distribution before applying SMOTE
print(Counter(y_train))
```

```
Counter({0: 16324, 1: 4676})
```

[89]:
```
from imblearn.over_sampling import SMOTE
# Applying SMOTE to transform the dataset
oversample = SMOTE(sampling_strategy=0.5)

#Fitting and applying the transform
X_train, y_train = oversample.fit_resample(X_train, y_train)
```

[90]:
```
# summarizing the new class distribution after applying SMOTE
print(Counter(y_train))
```

```
Counter({0: 16324, 1: 8162})
```

[91]:
```
from imblearn.under_sampling import RandomUnderSampler
# Applying undersampling to transform the dataset
undersample = RandomUnderSampler(sampling_strategy=0.5)

#Fitting and applying the transform
X_train, y_train = undersample.fit_resample(X_train, y_train)
```

[92]:
```
# summarizing the new class distribution after applying undersampling
print(Counter(y_train))
```

```
Counter({0: 16324, 1: 8162})
```

[93]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
# Initializing the logistic regression model as baseline model

log_reg = LogisticRegression()
```

[94]:
```
# Performing cross-validation on the training set
cv_scores = cross_val_score(log_reg, X_train, y_train, cv=5, scoring='accuracy')
print("Cross-Validation Accuracy Scores on Training Set:", cv_scores)
print("Average Cross-Validation Accuracy on Training Set:", cv_scores.mean())
```

```
Cross-Validation Accuracy Scores on Training Set: [0.67109024 0.67245252
0.66836839 0.67204411 0.66714315]
Average Cross-Validation Accuracy on Training Set: 0.670219681836189
```

[95]:
```
# Training the logistic regression model on the full training set
log_reg.fit(X_train, y_train)
```

[95]: LogisticRegression()

[96]:
```
# Making predictions with Logistic Regression on the test set
y_pred_test = log_reg.predict(X_test)
```

[97]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, recall_score,␣
↪precision_score, f1_score

# Calculating evaluation metrics for Logistic Regression
accuracy = accuracy_score(y_test, y_pred_test)
precision = precision_score(y_test, y_pred_test)
recall = recall_score(y_test, y_pred_test)
f1 = f1_score(y_test, y_pred_test)

conf_matrix = confusion_matrix(y_test, y_pred_test)
print("Confusion Matrix Logistic Regression:\n", conf_matrix)
print("Accuracy Logistic Regression:", accuracy)
print("Precision Logistic Regression:", precision)
print("Recall Logistic Regression:", recall)
print("F1 score Logistic Regression:", f1)
```

```
Confusion Matrix Logistic Regression:
 [[6945   95]
 [1886   74]]
Accuracy Logistic Regression: 0.7798888888888889
Precision Logistic Regression: 0.4378698224852071
Recall Logistic Regression: 0.03775510204081633
F1 score Logistic Regression: 0.06951620479098168
```

[98]:
```
tn, fp, fn, tp = confusion_matrix(y_test, y_pred_test).ravel()
print("True Negatives_LG (TN):", tn)
print("False Positives_LG (FP):", fp)
print("False Negatives_LG (FN):", fn)
print("True Positives_LG (TP):", tp)
```

```
True Negatives_LG (TN): 6945
False Positives_LG (FP): 95
False Negatives_LG (FN): 1886
True Positives_LG (TP): 74
```

[99]:
```
from sklearn.ensemble import RandomForestClassifier
# Initializing the Random Forest classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
```

[100]:
```
# Training the Random Forest classifier
rf_classifier.fit(X_train, y_train)
```

[100]: RandomForestClassifier(random_state=42)

[101]:
```python
# Making predictions with Random Forest classifier on the test set
y_pred_test = rf_classifier.predict(X_test)
```

[102]:
```python
# Calculating evaluation metrics for Random Forest
accuracy = accuracy_score(y_test, y_pred_test)
precision = precision_score(y_test, y_pred_test)
recall = recall_score(y_test, y_pred_test)
f1 = f1_score(y_test, y_pred_test)

conf_matrix = confusion_matrix(y_test, y_pred_test)
print("Confusion Matrix Random Forest:\n", conf_matrix)
print("Accuracy Random Forest:", accuracy)
print("Precision Random Forest:", precision)
print("Recall Random Forest:", recall)
print("F1 score Random Forest:", f1)
```

```
Confusion Matrix Random Forest:
 [[6473  567]
 [1379  581]]
Accuracy Random Forest: 0.7837777777777778
Precision Random Forest: 0.5060975609756098
Recall Random Forest: 0.29642857142857143
F1 score Random Forest: 0.3738738738738739
```

[103]:
```python
from sklearn.ensemble import GradientBoostingClassifier
# Initializing the Gradient Boosting Classifier
gb_clf = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1,␣
↪max_depth=3, random_state=42)
```

[104]:
```python
# Training the Gradient Boosting model
gb_clf.fit(X_train, y_train)
```

[104]: GradientBoostingClassifier(random_state=42)

[105]:
```python
# Making predictions with Gradient Boosting classifier
y_pred_test = gb_clf.predict(X_test)
```

[106]:
```python
# Calculating evaluation metrics for Gradient Boosting
accuracy = accuracy_score(y_test, y_pred_test)
precision = precision_score(y_test, y_pred_test)
recall = recall_score(y_test, y_pred_test)
f1 = f1_score(y_test, y_pred_test)

conf_matrix = confusion_matrix(y_test, y_pred_test)
print("Confusion Matrix Gradient Boosting:\n", conf_matrix)
```

```
print("Accuracy Gradient Boosting:", accuracy)
print("Precision Gradient Boosting:", precision)
print("Recall Gradient Boosting:", recall)
print("F1 score Gradient Boosting:", f1)
```

```
Confusion Matrix Gradient Boosting:
 [[6527  513]
 [1439  521]]
Accuracy Gradient Boosting: 0.7831111111111111
Precision Gradient Boosting: 0.5038684719535783
Recall Gradient Boosting: 0.26581632653061227
F1 score Gradient Boosting: 0.34802939211756845
```

[ ]:
```
!apt-get install -y texlive-xetex texlive-fonts-recommended texlive-latex-extra
!apt-get install texlive texlive-latex-extra pandoc
!jupyter nbconvert --to pdf "/content/Big_Data_Analytics_Project_Data␣
↪Transformation & Modeling_Mar16.ipynb"
```

```
Reading package lists… Done
Building dependency tree… Done
Reading state information… Done
The following additional packages will be installed:
  dvisvgm fonts-droid-fallback fonts-lato fonts-lmodern fonts-noto-mono fonts-
texgyre
  fonts-urw-base35 libapache-pom-java libcommons-logging-java libcommons-parent-
java
  libfontbox-java libfontenc1 libgs9 libgs9-common libidn12 libijs-0.35
libjbig2dec0 libkpathsea6
  libpdfbox-java libptexenc1 libruby3.0 libsynctex2 libteckit0 libtexlua53
libtexluajit2 libwoff1
  libzzip-0-13 lmodern poppler-data preview-latex-style rake ruby ruby-net-
telnet ruby-rubygems
  ruby-webrick ruby-xmlrpc ruby3.0 rubygems-integration t1utils teckit tex-
common tex-gyre
  texlive-base texlive-binaries texlive-latex-base texlive-latex-recommended
texlive-pictures
  texlive-plain-generic tipa xfonts-encodings xfonts-utils
Suggested packages:
  fonts-noto fonts-freefont-otf | fonts-freefont-ttf libavalon-framework-java
  libcommons-logging-java-doc libexcalibur-logkit-java liblog4j1.2-java poppler-
utils ghostscript
  fonts-japanese-mincho | fonts-ipafont-mincho fonts-japanese-gothic | fonts-
ipafont-gothic
  fonts-arphic-ukai fonts-arphic-uming fonts-nanum ri ruby-dev bundler debhelper
gv
  | postscript-viewer perl-tk xpdf | pdf-viewer xzdec texlive-fonts-recommended-
doc
  texlive-latex-base-doc python3-pygments icc-profiles libfile-which-perl
```