Data Mining

Case Study Assignment

Table of Contents

I performed the following steps to implement this case-study assignment

- Problem Statement
- Exploratory Data Analysis
- Preprocess the data
- Select Training data, test data
- Train the model
- Test the model (Predictions and reporting)
- Evaluate the model performance
- Suggest ways of improving the model

Problem Statement:

According to the Data Breach Index, more than 5 million records are being stolen on a daily basis. In today's digital world where trillions of Card transaction happens per day, detection of fraud is challenging. In this case study assignment, we construct a machine learning model from the dataset i.e."card_transdata" given by considering different Supervised Machine Learning Algorithms for calssification purpose i.e. K-Nearest Neighbors, Decision Tree Classifier, Random Forest Classifier. We then analyzed the performance of our model in terms of accuracy, confusion matrix

Exploratory Data Analysis

Import Libraies

- import pandas as pd

  Pandas library is used for data manipulation and analysis

- import numpy as np

  Numpy provides a large set of numeric datatypes that you can use to construct arrays.

- import seaborn as sns

  Seaborn is a library for making statistical graphics in Python

- import matplotlib.pyplot as plt

  Matplotlib. pyplot is a collection of command style functions that make matplotlib work like MATLAB.

- from sklearn.model_selection import train_test_split

train_test_split is a function in Sklearn model selection for splitting data arrays into two subsets: for training data and for testing data.

- from sklearn.tree import DecisionTreeClassifier

  Machine Learning Model we are going to use

- from sklearn.ensemble import RandomForestClassifier

  Machine Learning Model we are going to use

- from sklearn.metrics import precision_recall_curve

  Compute precision-recall pairs for different probability thresholds.

- from sklearn.metrics import plot_precision_recall_curve

  To plot the percision recall curve

- plt.ticklabel_format(useOffset=False)

- import warnings

  Warning messages are typically issued in situations where it is useful to alert the user of some condition in a program, where that condition (normally) doesn't warrant raising an exception and terminating the program.

- warnings.filterwarnings('ignore')

- %matplotlib inline

  %matplotlib inline sets the backend of matplotlib to the 'inline' backend

Load the Dataset:

data = pd.read_csv("card_transdata.csv")

```
data = pd.read_csv("card_transdata.csv")
```

```
data.head(5)
```

Data Analsysis

Shape, Collumn and Dtypes

In data analysis, we check out the shape of the dataset using data.shape and find out the index/name of the columns using data.columnsand to covert this into an array we write data.columns.values. Then we check out the data type of each of the column using data.dtypes and list bascially print out the list of data types we have.

```
In [152]: data.shape
Out[152]: (1000000, 8)

In [153]: data.columns
Out[153]: Index(['distance_from_home', 'distance_from_last_transaction',
                 'ratio_to_median_purchase_price', 'repeat_retailer', 'used_chip',
                 'used_pin_number', 'online_order', 'fraud'],
                dtype='object')

In [154]: data.columns.values
Out[154]: array(['distance_from_home', 'distance_from_last_transaction',
                 'ratio_to_median_purchase_price', 'repeat_retailer', 'used_chip',
                 'used_pin_number', 'online_order', 'fraud'], dtype=object)

In [155]: data.dtypes
Out[155]: distance_from_home                float64
          distance_from_last_transaction    float64
          ratio_to_median_purchase_price    float64
          repeat_retailer                   float64
          used_chip                         float64
          used_pin_number                   float64
          online_order                      float64
          fraud                             float64
          dtype: object

In [156]: list(set(data.dtypes.tolist()))
Out[156]: [dtype('float64')]
```

Now, to find the unique values ineach of the columns we just simple do a for loop which prints out all the unique values in each of the collumn. Now if want to print the unique values in each column along with their column name, i will simply write

for column in data:

    print(f'{column}:{data[column].unique()}')

Mean, Count and Standard Deviation

To check mean, count, standard deviation in any collumn we simply use .describe() function.

```
for column in data:
    print(data[column].unique())
```

```
[57.87785658 10.8299427   5.09107949 ...  2.91485699  4.25872939
 58.10812496]
[0.31114001 0.1755915  0.80515259 ... 1.47268669 0.24202337 0.31811012]
[1.94593998 1.29421881 0.42771456 ... 0.21807549 0.47582206 0.38691985]
[1. 0.]
[1. 0.]
[0. 1.]
[0. 1.]
[0. 1.]
```

```
for column in data:
    print(f'{column}:{data[column].unique()}')
```

```
distance_from_home:[57.87785658 10.8299427   5.09107949 ...  2.91485699  4.25872939
 58.10812496]
distance_from_last_transaction:[0.31114001 0.1755915  0.80515259 ... 1.47268669 0.24202337 0.31811012]
ratio_to_median_purchase_price:[1.94593998 1.29421881 0.42771456 ... 0.21807549 0.47582206 0.38691985]
repeat_retailer:[1. 0.]
used_chip:[1. 0.]
used_pin_number:[0. 1.]
online_order:[0. 1.]
fraud:[0. 1.]
```

```
data['ratio_to_median_purchase_price'].describe()
```

```
count    1000000.000000
mean           1.824182
std            2.799589
min            0.004399
25%            0.475673
50%            0.997717
75%            2.096370
max          267.802942
Name: ratio_to_median_purchase_price, dtype: float64
```

Skewness and Kurtosis of the Target Data

The skewness is a parameter to measure the symmetry of a data set and the kurtosis to measure how heavy its tails are compared to a normal distribution

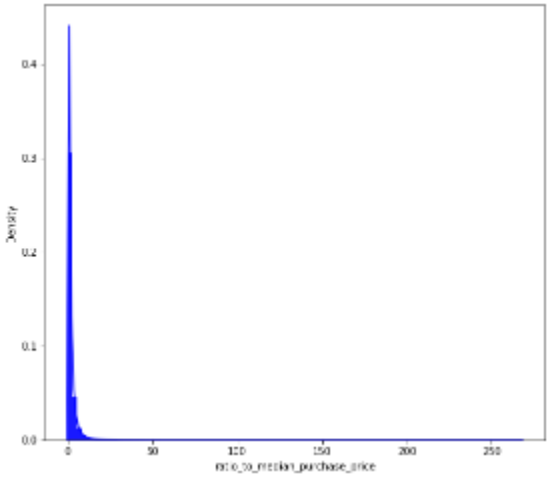
kurtosis(array, axis=0, fisher=True, bias=True) function calculates the kurtosis (Fisher or Pearson) of a data set. It is the the fourth central moment divided by the square of the variance.

```
In [160]: #skewness and kurtosis
          print("Skewness: %f" % data['fraud'].skew())
          print("Kurtosis: %f" % data['fraud'].kurt())
```

```
Skewness: 2.921824
Kurtosis: 6.537067
```

Distribution Plot with respect to different columns

The distplot represents the univariate distribution of data i.e. data distribution of a variable against the density distribution.



Information about Data, Histogram, Corelation and HeatMap

In the next step we write, data. info(), to check if there are any null values in any collumn.

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000000 entries, 0 to 999999
Data columns (total 8 columns):
 #   Column                          Non-Null Count    Dtype
---  ------                          --------------    -----
 0   distance_from_home              1000000 non-null  float64
 1   distance_from_last_transaction  1000000 non-null  float64
 2   ratio_to_median_purchase_price  1000000 non-null  float64
 3   repeat_retailer                 1000000 non-null  float64
 4   used_chip                       1000000 non-null  float64
 5   used_pin_number                 1000000 non-null  float64
 6   online_order                    1000000 non-null  float64
 7   fraud                           1000000 non-null  float64
dtypes: float64(8)
memory usage: 61.0 MB
```

Now, we plot a histogram of all the columns we have using,

```
data.hist(figsize=(16, 20), bins=50, xlabelsize=8, ylabelsize=8);
```

We find the corr() among each collumn using data. corr()

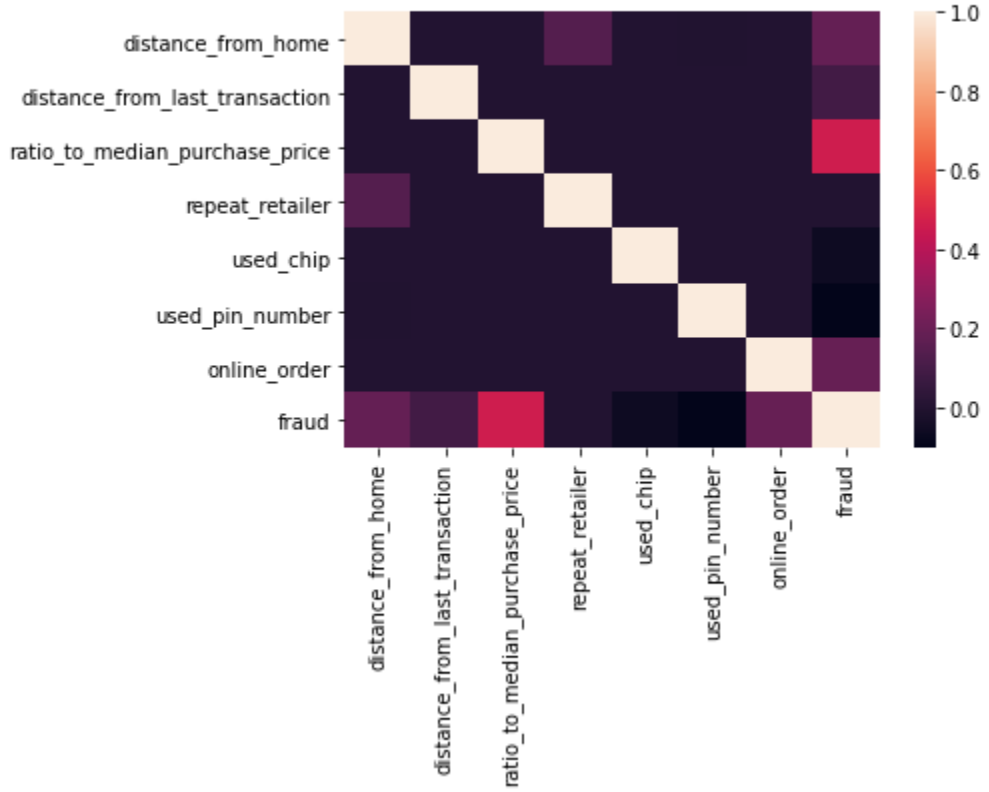
```
In [167]: data.corr()
```

In the next step, we create a heatmap with respect to each collumn using

sns.heatmap(data.corr())

```
sns.heatmap(data.corr())
```

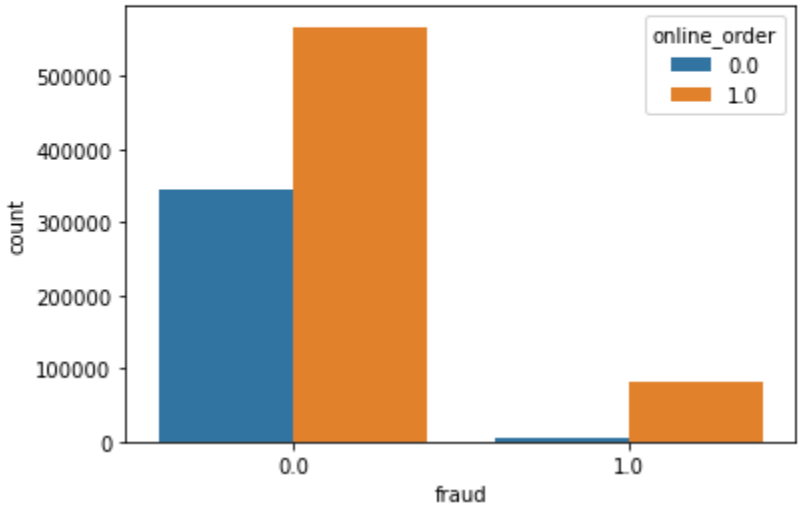
```
<AxesSubplot:>
```



Count Plot

Now, create count plot of each independant variable with respect to a dependant variable an example plot is shown here.

```
In [169]: import seaborn as sns
          import matplotlib.pyplot as plt
          sns.countplot(x ='fraud', hue = "online_order" ,data = data)
          plt.show()
```

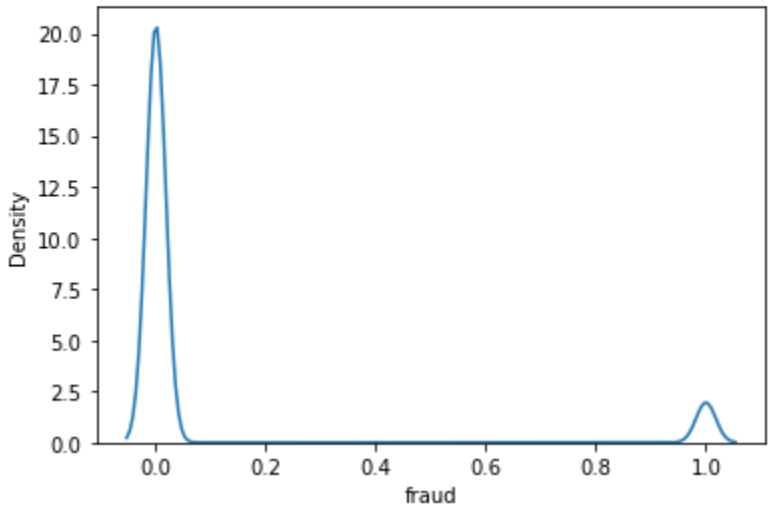


Kde Plot

In the next step, we create kde plot to Plot univariate or bivariate distributions using kernel density estimation.

```
sns.kdeplot(data['fraud'])
```

```
<AxesSubplot:xlabel='fraud', ylabel='Density'>
```

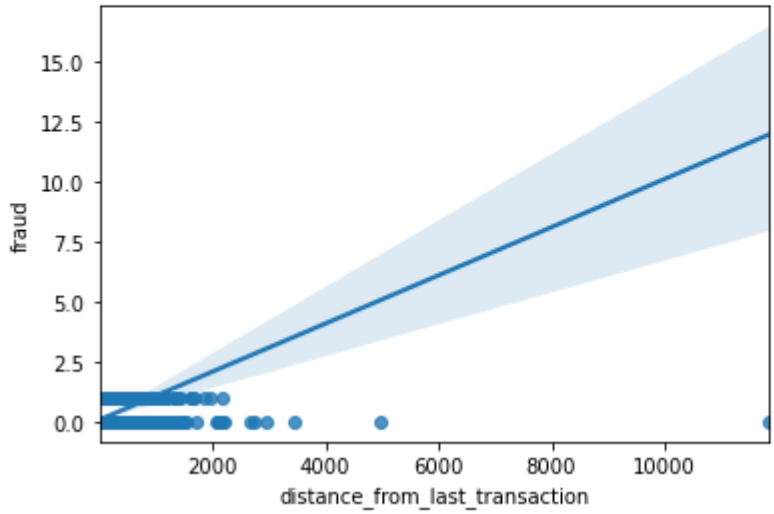


RegPlot

We also create regplot, as shown in the image below

```
sns.regplot(x="distance_from_last_transaction", y="fraud", data=data)
```

```
<AxesSubplot:xlabel='distance_from_last_transaction', ylabel='fraud'>
```



Value Count and Converting it into a Data Frame

In the next step, we do value count with respect to each collumn and create a data frame using to_frame()

```
In [185]: data['distance_from_last_transaction'].value_counts()

Out[185]: 0.022992      1
          122.981697    1
          0.435864      1
          1.662242      1
          0.287522      1
                       ..
          3.569070      1
          15.868800     1
          2.229245      1
          0.816694      1
          0.449071      1
          Name: distance_from_last_transaction, Length: 1000000, dtype: int64
```

```
In [187]: data['distance_from_last_transaction'].value_counts().to_frame()
```

Out[187]:

	distance_from_last_transaction
0.022992	1
122.981697	1
0.435864	1
1.662242	1
0.287522	1
...	...
3.569070	1
15.868800	1
2.229245	1
0.816694	1
0.449071	1

GroupBy Function

In the next step we group by different collumns and check the relation among them

```
data_group_one = data_group_one.groupby(['used_chip'], as_index = False).mean()
```

```
data_group_one
```

	used_chip	used_pin_number	fraud
0	0.0	0.100916	0.100051
1	1.0	0.100037	0.063956

```
data_group_one = data_group_one.groupby(['used_chip'], as_index = True).mean()
```

```
data_group_one
```

used_chip	used_pin_number	fraud
0.0	0.100916	0.100051
1.0	0.100037	0.063956

```
data_group_one = data_group_one.groupby(['used_chip', 'used_pin_number'], as_index = True).mean()
```

```
data_group_one
```

		fraud
used_chip	used_pin_number	
0.0	0.100916	0.100051
1.0	0.100037	0.063956

Preprocess the Data

Now we will preprocess the data, will check if there are any null values/ missing or if there exist any duplicates, we will perform all these steps while doing preprocessing of the data.

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000000 entries, 0 to 999999
Data columns (total 8 columns):
 #   Column                          Non-Null Count    Dtype  
---  ------                          --------------    -----  
 0   distance_from_home              1000000 non-null  float64
 1   distance_from_last_transaction  1000000 non-null  float64
 2   ratio_to_median_purchase_price  1000000 non-null  float64
 3   repeat_retailer                 1000000 non-null  float64
 4   used_chip                       1000000 non-null  float64
 5   used_pin_number                 1000000 non-null  float64
 6   online_order                    1000000 non-null  float64
 7   fraud                           1000000 non-null  float64
dtypes: float64(8)
memory usage: 61.0 MB
```

```
data.isnull()
```

```
data.isnull().sum()
```

```
distance_from_home                0
distance_from_last_transaction    0
ratio_to_median_purchase_price    0
repeat_retailer                   0
used_chip                         0
used_pin_number                   0
online_order                      0
fraud                             0
dtype: int64
```

```
data.drop_duplicates(inplace = True)
```

```
plt.figure(figsize=(10,6))
sns.heatmap(data.isna().transpose(),
            cmap="YlGnBu",
            cbar_kws={'label': 'Missing Data'})
```

Select the Training data & Test data

Now we separate the training and the target data as shown in the figure below

```
In [278]: X = data.drop("fraud", axis =1)
          y = data[["fraud"]]
```

```
In [279]: X.head(5)
```

Now we do the train and test split

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 1)
```

```
X_train.shape
```

```
(800000, 7)
```

```
y_train.shape
```

```
(800000, 1)
```

```
X_test.shape
```

```
(200000, 7)
```

```
y_test.shape
```

```
(200000, 1)
```

Train and Test the Model

Now we train our model using different supervised machine learning algorithms i.e. K- Nearest Neighbors Classifier, Decision Tree Classifier, Random Forest Classifier and then we test our model using the unseen data

K- Nearest Neighbors Classifier

```
In [286]: from sklearn.neighbors import KNeighborsClassifier
          knn = KNeighborsClassifier(n_neighbors = 2, metric = 'euclidean')

In [287]: knn.fit(X_train, y_train)
Out[287]: KNeighborsClassifier(metric='euclidean', n_neighbors=2)

In [288]: y_pred = knn.predict(X_test)
```

Decision Tree Classifier

```
In [233]: data_model = DecisionTreeClassifier(random_state = 10)

In [243]: data_model.fit(X_train, y_train)
Out[243]: DecisionTreeClassifier(random_state=10)

In [244]: data_model.score(X_train, y_train)
Out[244]: 1.0

In [246]: data_model.score(X_test, y_test)
Out[246]: 0.99997
```

Random Forest Classifier

```
In [272]: random_forest = RandomForestClassifier(n_estimators=10)
          random_forest.fit(X_train,y_train)
          Y_pred = random_forest.predict(X_test)
          random_forest.score(X_test, y_test)
```

Model Performance

We evaluate model performance considering different parameters accuracy, specificity, the area under the precision-recall curve, confusion matric

```
In [289]: from sklearn.metrics import classification_report, confusion_matrix, f1_score, accuracy_score
```

```
In [228]: print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.99      1.00      0.99    182383
         1.0       0.95      0.88      0.91     17617

    accuracy                           0.99    200000
   macro avg       0.97      0.94      0.95    200000
weighted avg       0.98      0.99      0.98    200000
```

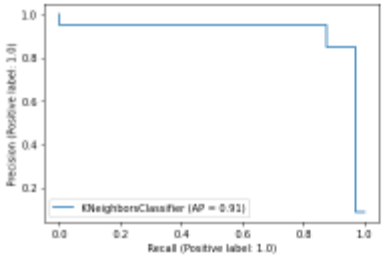
```
In [290]: confusion_matrix(y_test, y_pred)
Out[290]: array([[181573,    810],
                 [  2179,  15438]], dtype=int64)
```

```
In [295]: cm1 = confusion_matrix(y_test, y_pred)
```

```
In [296]: specificity1 = cm1[1,1]/(cm1[1,0]+cm1[1,1])
          print('Specificity : ', specificity1)
          Specificity :  0.8763126525515127
```

```
In [291]: accuracy_score(y_test, y_pred)
Out[291]: 0.985055
```

```
In [292]: disp = plot_precision_recall_curve(knn, X_test, y_test)
```



Ways of improving the model

The following are the two possible ways which can help us to improve the accuracy of the model

1. We can further improve our model by doing feature selection which means considering only those collumns which have an impact on output and removing the other collumns which donot have impact on output orvery least impact. I have shown as an example as well how we can do feature selection by using one of the techniques of feature selection which is ExtraTreeClassifier

To find the Features Critical in the Identification of Card Frauds.

We will use Feature Importance: Feature Importance gives us a score of each feature of our data, the higherthe score, the more important, the more relevant the feature is towards the output variable. We will be using Extra Tree Classifier and extract the top 5 features in the dataset

Import the Required Library

```
from sklearn.ensemble import ExtraTreesClassifier
```

```
model = ExtraTreesClassifier()
```

```
model.fit(X,y)
```

```
ExtraTreesClassifier()
```

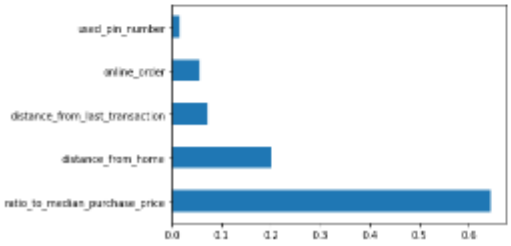
```
print(model.feature_importances_)
```

```
[0.20167485 0.07190987 0.64451701 0.0033286  0.00826106 0.01522033
 0.05508827]
```

```
features_importances = pd.Series(model.feature_importances_, index = X.columns)
```

```
features_importances.nlargest(5).plot(kind = 'barh')
```

```
<AxesSubplot:>
```



2. We can also improve our model by doing hyperparameter tunning means optimizing our model using Grid Search CV or Random Search CV which enables us to select the parameters which gives us the best accuracy.