Steps Involved:

1. Importing the required packages into our python environment.
2. Importing the data
3. Processing the data to our needs and Exploratory Data Analysis
4. Feature Selection and Data Split
5. Building two types of classification models
6. Evaluating the created classification models using the evaluation metrics

Importing the Packages:

For this project, our primary packages are going to be Pandas to work with data, NumPy to work with arrays, Matplotlib & seaborne for visualization, scikit-learn for data split, label encoding, building, and evaluating the classification models. Let's import all of our primary packages into our python environment.

```
#Importing Libraries

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
import warnings
%matplotlib inline
warnings.filterwarnings('ignore')
```

Importing the Data:

The given data-set contains 24 attributes i.e, ['TRAN_AMT', 'ACCT_PRE_TRAN_AVAIL_BAL', 'CUST_AGE', 'OPEN_ACCT_CT', 'WF_dvc_age', 'PWD_UPDT_TS', 'CARR_NAME', 'RGN_NAME', 'STATE_PRVNC_TXT', 'ALERT_TRGR_CD', 'DVC_TYPE_TXT', 'AUTHC_PRIM_TYPE_CD', 'AUTHC_SCNDRY_STAT_TXT', 'CUST_ZIP', 'CUST_STATE', 'PH_NUM_UPDT_TS', 'CUST_SINCE_DT', 'TRAN_TS', 'TRAN_DT', 'ACTN_CD', 'ACTN_INTNL_TXT', 'TRAN_TYPE_CD', 'ACTVY_DT', 'FRAUD_NONFRAUD']

Data Processing and EDA

The first step is to convert the object datatypes to int, float, and timestamp as required.

```
df_train['TRAN_AMT'] = df_train['TRAN_AMT'].apply(lambda x: float(x))
df_train['CUST_ZIP'] = df_train['CUST_ZIP'].apply(lambda x: int(x))
df_train['ACCT_PRE_TRAN_AVAIL_BAL'] = df_train['ACCT_PRE_TRAN_AVAIL_BAL'].apply(lambda x: float(x))
df_train['OPEN_ACCT_CT'] = df_train['OPEN_ACCT_CT'].apply(lambda x: int(x))
df_train['CUST_AGE'] = df_train['CUST_AGE'].apply(lambda x: int(x))
df_train['WF_dvc_age'] = df_train['WF_dvc_age'].apply(lambda x: int(x))
df_train['TRAN_TS'] = df_train['TRAN_TS'].apply(lambda x: pd.Timestamp(x))
```

```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14000 entries, 1 to 14000
Data columns (total 24 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   TRAN_AMT                14000 non-null  object
 1   ACCT_PRE_TRAN_AVAIL_BAL 14000 non-null  object
 2   CUST_AGE                14000 non-null  object
 3   OPEN_ACCT_CT            14000 non-null  object
 4   WF_dvc_age              14000 non-null  object
 5   PWD_UPDT_TS             10875 non-null  object
 6   CARR_NAME               11291 non-null  object
 7   RGN_NAME                11291 non-null  object
 8   STATE_PRVNC_TXT         11291 non-null  object
 9   ALERT_TRGR_CD           14000 non-null  object
 10  DVC_TYPE_TXT            12239 non-null  object
 11  AUTHC_PRIM_TYPE_CD      14000 non-null  object
 12  AUTHC_SCNDRY_STAT_TXT   13926 non-null  object
 13  CUST_ZIP                14000 non-null  object
 14  CUST_STATE              13964 non-null  object
 15  PH_NUM_UPDT_TS          6939 non-null   object
 16  CUST_SINCE_DT           14000 non-null  object
 17  TRAN_TS                 14000 non-null  object
 18  TRAN_DT                 14000 non-null  object
 19  ACTN_CD                 14000 non-null  object
 20  ACTN_INTNL_TXT          14000 non-null  object
 21  TRAN_TYPE_CD            14000 non-null  object
 22  ACTVY_DT                14000 non-null  object
 23  FRAUD_NONFRAUD          14000 non-null  object
dtypes: object(24)
memory usage: 2.6+ MB
```

From the first code *df_trian.info()* we identified that the dataset has a lot of missing values.

Interestingly the number of missing values in CARR_NAME, RGN_NAME, and STATE_PRVNC_TXT are the same, and also they are missing in the same rows. Instead of removing these rows, we need to understand the significance of this on the FRAUD_NONFRAUD variable.

Also, the variables TRANS_DT and ACTVY_DT can be derived from TRANS_TS. ACTN_CD, 'ACTN_INTNL_TXT', 'TRAN_TYPE_CD' have unique values throughout the data. Hence there is no way these variables can contribute significantly to the prediction model. Hence TRANS_DT, ACTVY_DT, ACTN_CD, 'ACTN_INTNL_TXT', 'TRAN_TYPE_CD' can be dropped.

```
df_train[(df_train.CARR_NAME.isnull() & df_train.RGN_NAME.isnull())]['FRAUD_NONFRAUD'].value_counts()
# this means that wherever the CARR_NAME is null, the RGN_NAME and STATE_PRVNC_TXT is null
# almost 47% of the fraud values are present when these values are null
```

```
Fraud        1969
Non-Fraud     740
Name: FRAUD_NONFRAUD, dtype: int64
```

Almost 47% of the total fraud values are present when the CARR_NAME, RGN_NAME, and STATE_PRVNC_TXT are null. This means these values have a high significance in the target variable being FRAUD. Hence, instead of deleting these columns, the null values are replaced with No-Value to capture the significance of empty data points in the model.

Similarly to understand the significance of empty values in other variables like 'PWD_UPDT_TS','DVC_TYPE_TXT', 'AUTHC_SCNDRY_STAT_TXT', 'CUST_STATE', 'PH_NUM_UPDT_TS' the values are replaced with No-Value.

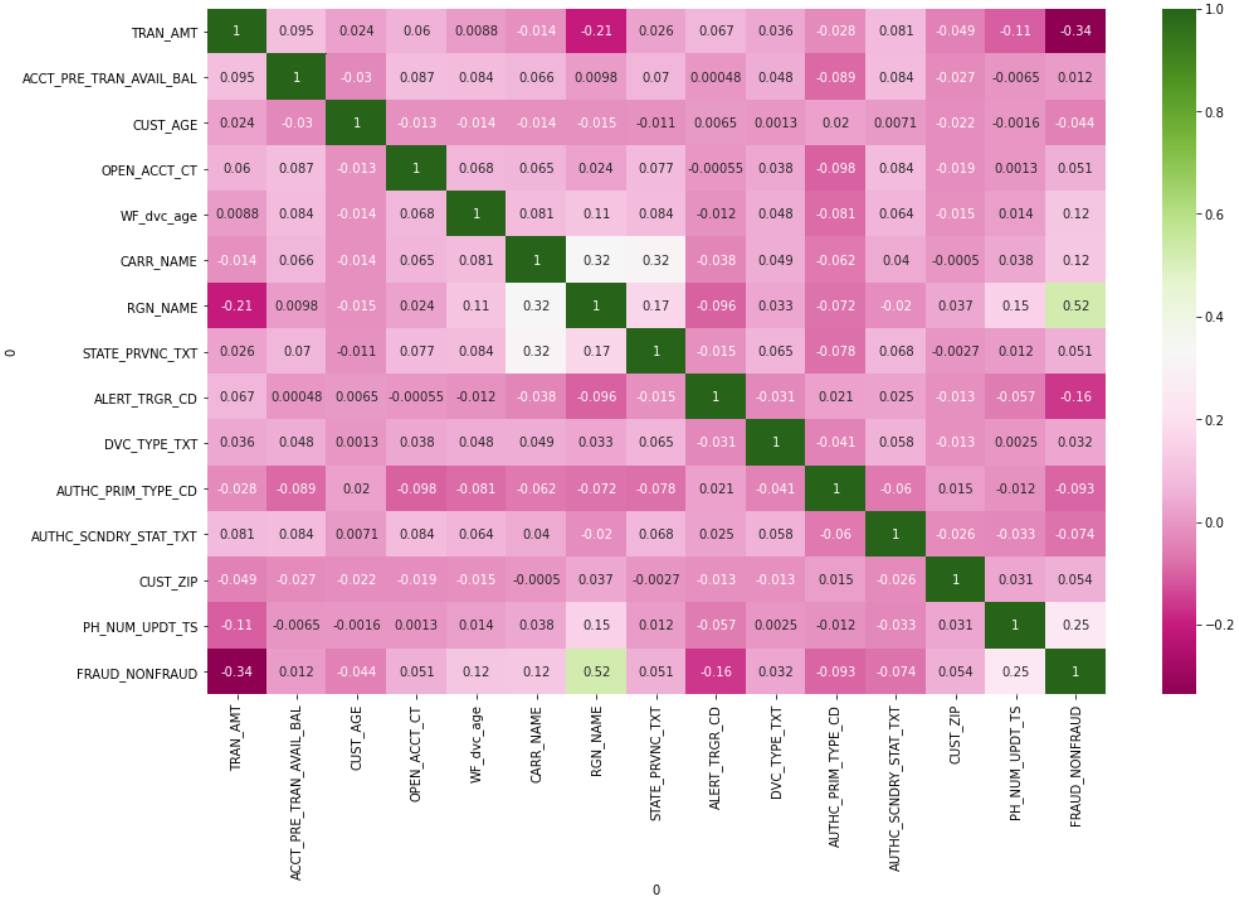
```
df_train["PH_NUM_UPDT_TS"].fillna('Not Given',inplace=True)
df_train["CARR_NAME"].fillna("No Value",inplace = True)
df_train["RGN_NAME"].fillna("No Value",inplace = True)
df_train["STATE_PRVNC_TXT"].fillna("No Value",inplace = True)
df_train["DVC_TYPE_TXT"].fillna("No Value",inplace = True)
df_train["AUTHC_SCNDRY_STAT_TXT"].fillna('ALLOW',inplace=True)
```

Label Encoding:

To ensure there is uniformity in labels between both the train and train datasets, the datasets are combined for label-encoding.

```
from sklearn.preprocessing import LabelEncoder
for col in combined_clean.columns:
    if combined_clean[col].dtype == "object":
        le = LabelEncoder()
        le.fit(list(combined_clean[col].astype(str).values))
        combined_clean[col] = le.transform(list(combined_clean[col].astype(str).values))
```

Understanding the correlation between the FRAUD_NONFRAUD and other variables using a correlation matrix heatmap.



But in the correlation matrix it was found that only the TRAN_AMT, WF_dvc_age, CARR_NAME, RGN_NAME, ALERT_TRGR_CD, PH_NUM_UPDT_TS have a high correlation with the FRAUD_NONFRAUD variable. The model was built using only these variables and all other variables are dropped.

Also, please note that to understand the significance of the variables PWD_UPDT_TS, CUST_SINC_DT, a set of new variables which are different variations of these variables have been created but no significance was found. Hence they are not included in the report/model. Multiple iterations have been made by removing the null values and replacing the values to understand the significance of the variables. The above-shown method was the best-case scenario among all.

Feature Selection & Data Split

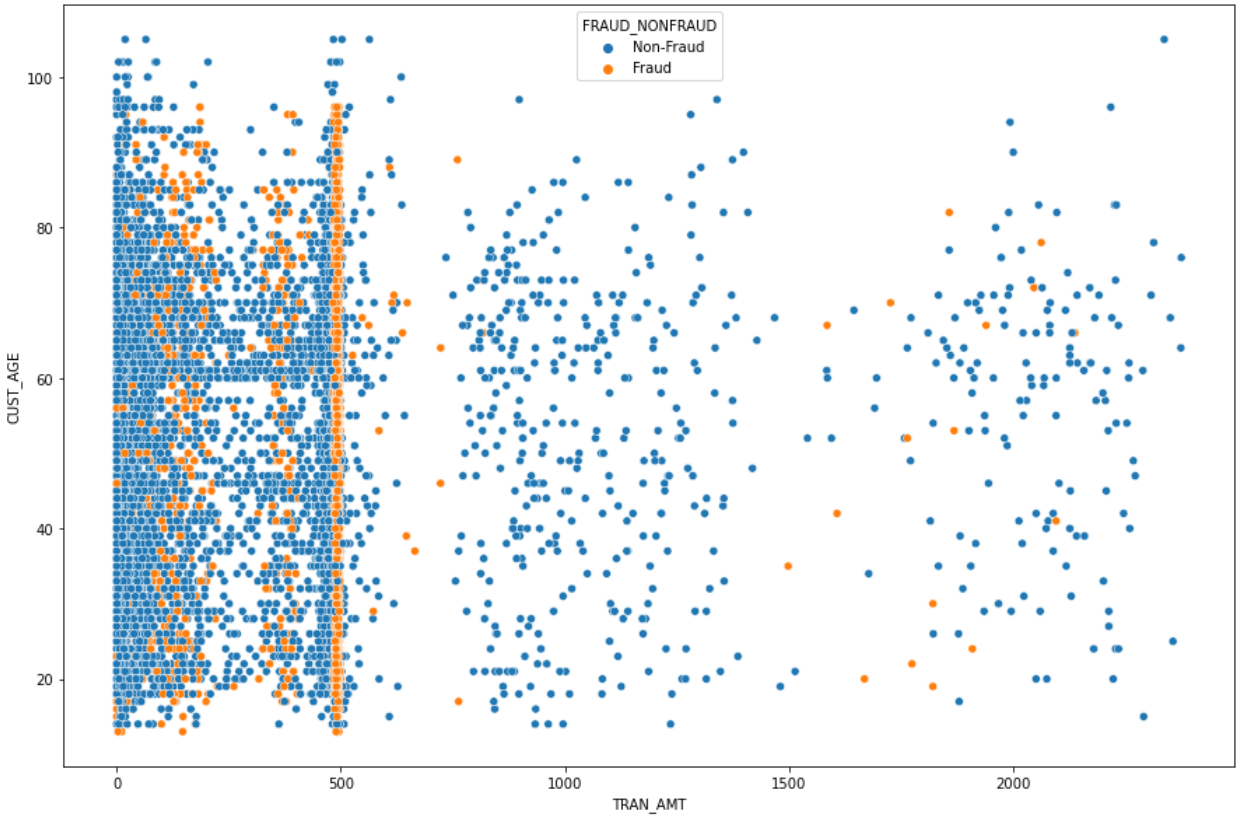
In this process, we are going to define the independent (X) and the dependent variables (Y). Using the defined variables, we will split the data into a training set and testing set which is further used for modeling and evaluating. We can split the data easily using the 'train_test_split' algorithm in python. As discussed above these are variables accounted for in the model

target = 'FRAUD_NONFRAUD'

col= ['TRAN_AMT', 'WF_dvc_age', 'CARR_NAME', 'RGN_NAME','PH_NUM_UPDT_TS', 'ALERT_TRGR_CD']

Modeling:

In this step, we will be building two different types of classification models namely Decision Tree, Logistic Regression. As visualized in the below scatter plot, the target variable classes are not very well separated which means that the decision tree model works better than logistic regression as it bisects the space into smaller and smaller regions.



Even though there are many more models which we can use, these are the most popular models used for solving classification problems. All these models can be built feasibly using the algorithms provided by the scikit-learn package.

Model #1 (Logistic Regression)	Model #2 (Decision Tree Model)

`from sklearn.linear_model import LogisticRegression`
 	
`# instantiate the model (using the default parameters)`
`logreg = LogisticRegression()`
 	
`# fit the model with data`
`logreg.fit(X_train,y_train)`
`y_pred=logreg.predict(X_test)`
	`from sklearn.tree import DecisionTreeClassifier`
	`clf_gini = DecisionTreeClassifier(criterion = "gini",`
	`random_state = 100,`
	`max_depth=5,`
	`min_samples_leaf=10)`
	
	`# Performing training`
	`clf_gini.fit(X_train, y_train)`
	`y_pred = clf_gini.predict(X_test)`

Evaluation:

As mentioned earlier it is very clear that the performance of the decision tree model is better than the logistic regression model. Hence, the test data set is also predicted using the decision tree model fit.

Logistic Regression Model Results	Decision Tree Model Results

`print("Accuracy:",metrics.accuracy_score(y_test, y_pred))`
`print("Precision:",metrics.precision_score(y_test, y_pred))`
`print("Recall:",metrics.recall_score(y_test, y_pred,pos_label=0))`
`print("F1 Score:",metrics.f1_score(y_test, y_pred))`
 	
`Accuracy: 0.8046428571428571`
`Precision: 0.8269767441860465`
`Recall: 0.5608028335301063`
`F1 Score: 0.8666829149402876`
	`print("Accuracy:",metrics.accuracy_score(y_test, y_pred))`
	`print("Precision:",metrics.precision_score(y_test, y_pred))`
	`print("Recall:",metrics.recall_score(y_test, y_pred))`
	`print("F1 Score:",metrics.f1_score(y_test, y_pred))`
	
	`#The decision tree model performed better than logistic regression model as`
	`#there was high variance in the FRAUD_NONFRAUD variable`
	
	`Accuracy: 0.9331428571428572`
	`Precision: 0.9429477020602218`
	`Recall: 0.9635627530364372`
	`F1 Score: 0.9531437725270324`