American University of Science and Technology

Faculty of Engineering

Department of Computer Science

Final Project

Fall 2021 – 2022

Monday Feburary 1st, 2022

Louay Khaddaj (12190509)

Fady Saoudy (20210158)

Submitted to:

Dr. Charbel Boustan

TABLE OF CONTENTS

Chapter 1 - Introduction

This Chapter introduces the course details, course code and its name, role of this course, what topics are covered throughout the whole semester and information that give brief explanation about the course. Moreover, the type of the project was stated and explained in details. Learning outcomes from this course and this project were explored through bullet points with a view to determining the goal of this project.

1.1 – Introduction

Course Code: CSI 419

Course Name: Introduction to Data Science

Topics Covered:

- Python Language Basics and Jupyter Notebook.
- Built-In functions, files and data structures used in data science.
- Libraries Numpy, Scipy, Sci-kit learn, Pandas, Matplotlib.
- Exploring and loading any dataset.
- Preprocessing of data.
- Data Wrangling and Cleaning.
- Data Aggregation and grouping.
- Plotting and visualization.

Other Information: Loading any dataset using the panda library is the key point behind each and every problem set. Exploration of data is then conducted to discover the data and perform important operations such as feature engineering, visualization of the data and plotting the data in form of graphs. Also, building predictive models using the machine learning algorithms was the key concepts behind this course. It’s good to mention that any dataset used was real data.

Abstract: As the world is moving nowadays towards modern technology, it’s good to mention that security is threatened by the fact of this digital world. This fast transition in technology will cause a boom in cyberattacks. Every human being in this generation deals with banks and finance either physically or online. Most of the online transactions are done through credit cards but online transactions go hand in hand with abnormal activities. Cyber attackers take advantage of the fact that people uses their credit card and commit illegal actions that may cost huge loss for banks and people simultaneously. So to confront this issue and to find these fishy transactions it’s appropriate to build a system that can detect fraud transactions and abnormal activities.

The main idea behind this project is to build a python program using Jupyter notebook that can detect fraud transactions by applying machine learning algorithms. So, a dataset that contains real data is being used that contains both normal and abnormal transactions. This makes the

problem a classification model not a regression. After the processing of the data, six models will be built and then evaluated to check which model is the best one that can classify the transactions into fraud and not fraud.

1.2 – Type of this Project

Considering the implementation in python language in Jupyter notebook and applying six different machine learning algorithms where multiple steps and bunch of methods were used, this makes the project a practical not a theoretical one. Since Python facilitates the efforts of implementing machine learning algorithms where it has extensive amount of packages, its being used as the core programming language in this project. Moreover, the use of python nowadays in the market is the most rational thing to use when implementing such kind of systems.

1.3 – Learning Outcomes

- Downloading and importing the necessary libraries.
- Loading the data.
- Data Preprocessing or Explanatory Data Analysis.
- Feature Engineering and splitting the data into training set and testing set.
- Applying 6 machine learning algorithms specifically classification models such as Logistic regression, Decision Trees, Random Forests, KNN and Support Vector machines.
- Evaluation of models in terms of accuracy recall, precision and F1 Score.

This chapter introduced the key points of this report, it learning outcomes, type of this project and an overview of the report. In Chapter 2, we'll be covering the problem and the challenges that we faced.

Chapter 2 – Problem Statement

This chapter tackles the problems that complicated the work and some challenges.

2.1 – Problem Statement

With the substantial growth in online transaction especially credit card transaction where people rely heavily on credit cards to fulfil their needs, this have lead to exponential increase in fraud transactions. These fraudulent transactions incurred huge losses for insurance companies and other financial institutions. According to a survey done by the FTC where they received nearly three million complaints from consumers in 2018. Consumer reports that they have lost about $1.48 billion to fraud in 2018—38 percent more than the year before. Debt collection complaints dropped to the number two spot after topping the FTC's list of consumer complaints for the previous three years. [1]

In this project, we propose to examine six different classification models. We will evaluate each of these models from accuracy to precision, recall and F1Score to check which algorithm best classify the fraudulent transactions. Specifically, we shall employ the KNN, random forest, decision trees, Logistic regression, support vector machines and XGBoost models.

2.2 – Challenges Faced

Some of the challenges faced are:

- Due to the huge amount of computational resources needed to train and test the dataset, this renders the work to be slower and time consuming.
- Installing the appropriate libraries and packages.
- Choosing the test size of the dataset.
- Build a heat map for confusion matrix.

In this Chapter, the challenges faced were presented. In the following chapter, the proposed methodology will be explained.

Chapter 3 – Proposed Solution

This Chapter aims to present the solution and the implementation of the code including all the necessary documentation.

3.1 – Importing Necessary Libraries

Credit Card Fraud Detection

Importing Libraries

In [1]:

```
import numpy as np
import pandas as pd
import plotly.graph_objects as go
import matplotlib.pyplot as plt
import plotly.express as px
import seaborn as sns
from pylab import rcParams
import warnings
from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler, MinMaxScaler, MaxAbsScaler
from sklearn.metrics import accuracy_score, precision_score, confusion_matrix, recall_score, f1_score

from sklearn.tree import DecisionTreeClassifier # Decision Tree algorithm
from sklearn.neighbors import KNeighborsClassifier # KNN algorithm
from sklearn.linear_model import LogisticRegression # Logistic regression algorithm
from sklearn.svm import SVC # SVM algorithm
from sklearn.ensemble import RandomForestClassifier # Random forest tree algorithm
from xgboost import XGBClassifier # XGBoost algorithm

import itertools
warnings.filterwarnings('ignore')
```

- NumPy is an open-source numerical Python library.
- Pandas are a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language. [2]
- Plotly's Python graphing library makes interactive, publication-quality graphs. Examples of how to make line plots, scatter plots, area charts, bar charts, error bars, box plots, histograms, heatmaps, subplots, multiple-axes, polar charts, and bubble charts. [3]
- Seaborn and Matplotlib are 2 open source libraries in python for creating data visualization graphs.
- It is a collection of algorithms and tools for machine learning. “It is very useful to create machine learning and statistical models such as classification, regression and clustering.
- Standard Scalar will be used to normalized the values that follows the process of calculating the Z-Score.
- Train test split and the evaluation measures such accuracy score, confusion matrix, recall score and F1_score is used.
- All the six different classifiers are being imported from the scikit learn library.

3.2 – Explanatory data Analysis (EDA)

Read the Dataset from the csv file using pandas

```
In [2]: 1 data=pd.read_csv('creditcard.csv')
```

Exploring the dataset

```
In [3]: 1 data.head(5).T.style.set_properties(**{'background-color': 'grey',
        2                                        'color': 'white',
        3                                        'border-color': 'white'})
```

Out[3]:

	0	1	2	3	4
Time	0.000000	0.000000	1.000000	1.000000	2.000000
V1	-1.359807	1.191857	-1.358354	-0.966272	-1.158233
V2	-0.072781	0.266151	-1.340163	-0.185226	0.877737
V3	2.536347	0.166480	1.773209	1.792993	1.548718
V4	1.378155	0.448154	0.379780	-0.863291	0.403034
V5	-0.338321	0.060018	-0.503198	-0.010309	-0.407193
V6	0.462388	-0.082361	1.800499	1.247203	0.095921
V7	0.239599	-0.078803	0.791461	0.237609	0.592941
V8	0.098698	0.085102	0.247676	0.377436	-0.270533
V9	0.363787	-0.255425	-1.514654	-1.387024	0.817739
V10	0.090794	-0.166974	0.207643	-0.054952	0.753074
V11	-0.551600	1.612727	0.624501	-0.226487	-0.822843
V12	-0.617801	1.065235	0.066084	0.178228	0.538196
V13	-0.991390	0.489095	0.717293	0.507757	1.345852
V15	1.468177	0.635558	2.345865	-0.631418	0.175121
V16	-0.470401	0.463917	-2.890083	-1.059647	-0.451449
V17	0.207971	-0.114805	1.109969	-0.684093	-0.237033
V18	0.025791	-0.183361	-0.121359	1.965775	-0.038195
V19	0.403993	-0.145783	-2.261857	-1.232622	0.803487
V20	0.251412	-0.069083	0.524980	-0.208038	0.408542
V21	-0.018307	-0.225775	0.247998	-0.108300	-0.009431
V22	0.277838	-0.638672	0.771679	0.005274	0.798278
V23	-0.110474	0.101288	0.909412	-0.190321	-0.137458
V24	0.066928	-0.339846	-0.689281	-1.175575	0.141267
V25	0.128539	0.167170	-0.327642	0.647376	-0.206010
V26	-0.189115	0.125895	-0.139097	-0.221929	0.502292
V27	0.133558	-0.008983	-0.055353	0.062723	0.219422
V28	-0.021053	0.014724	-0.059752	0.061458	0.215153
Amount	149.620000	2.690000	378.660000	123.500000	69.990000
Class	0.000000	0.000000	0.000000	0.000000	0.000000

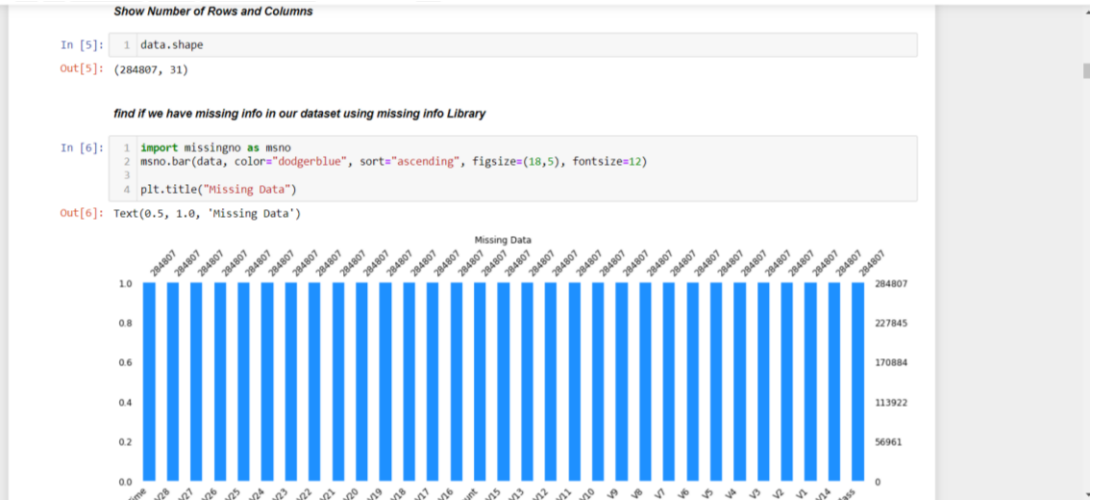
Show all the columns

```
In [4]: 1 data.columns
```

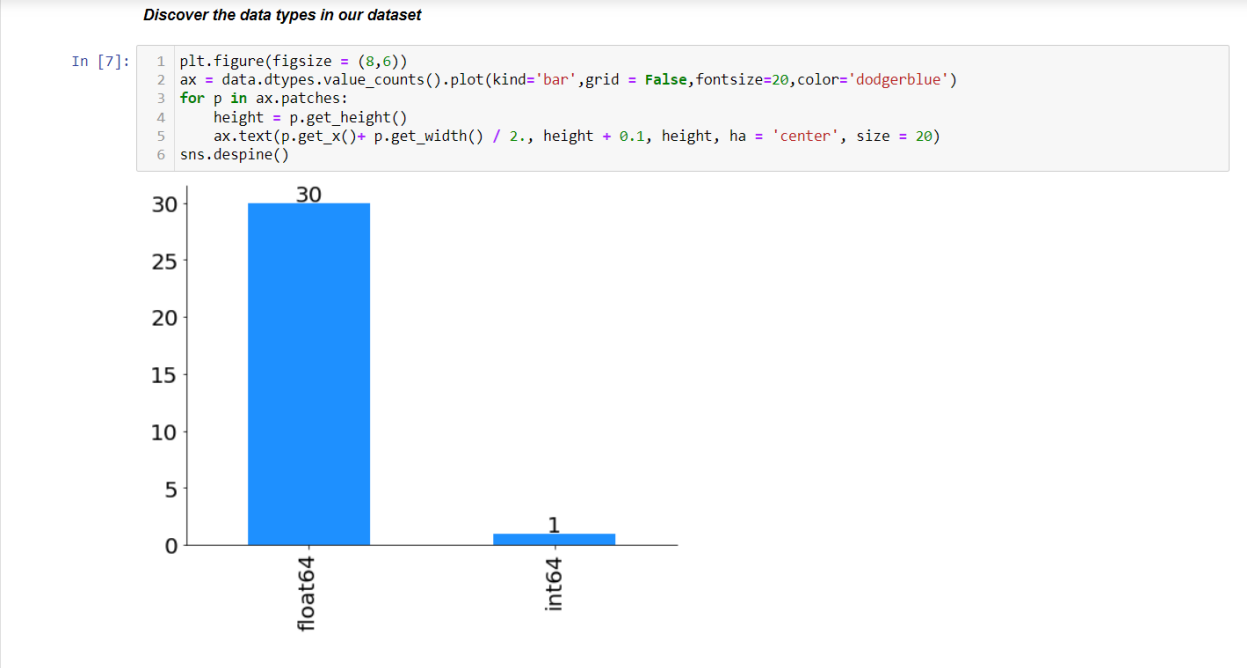
```
Out[4]: Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
               'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
               'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
               'Class'],
              dtype='object')
```

The data set is being loaded successfully and then the head of the first 5 elements were displayed on the screen. Moreover, it appears that we have 28 variables from V1 to V28 plus the time in a

separation from previous transactions and amounts. It's good to mention that this dataset contains real bank transactions but they are being hidden and transformed and the result is due to PCA dimensionality reduction. This was used in order to protect sensitive information in this dataset. For example, hiding the identity of the individual who made the credit-card transaction, in addition to location. The class here is going to be a 0 if it's a valid normal credit card transaction and then 1 is going to be a fraudulent transaction. This dataset is being downloaded as a csv file from kaggle. [5]



The number of rows here is 284807 and the number of columns is 31. Then, the missing values are checked and no missing values obtained.



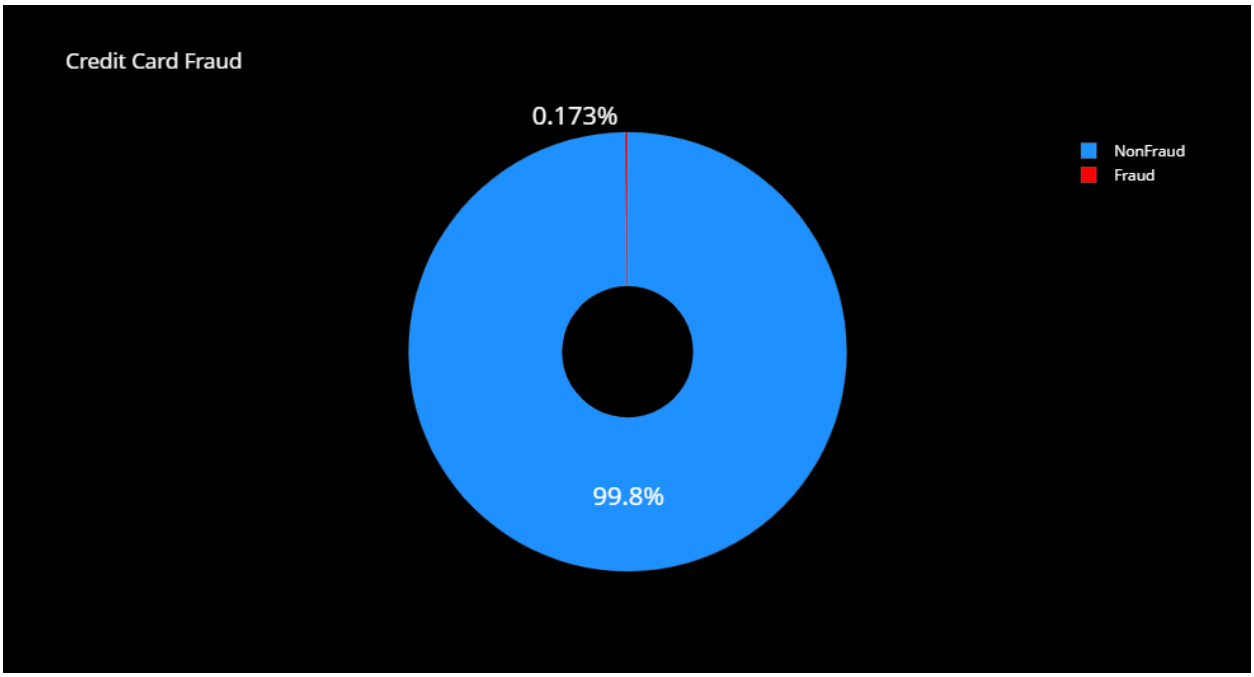
In the above figure we checked the datatypes of our variables where 30 of our variables are float64 and 1 variable of type integer int64.

Number of Fraud and Nonfraud Cases

```
In [9]:
1 Fraud = data[data['Class']==1]
2 print('Number of Fraud Cases:',len(Fraud))
3 NonFraud = data[data['Class']==0]
4 print('Number of Non Fraud Cases:',len(NonFraud))
```

```
Number of Fraud Cases: 492
Number of Non Fraud Cases: 284315
```

After assigning 1 to the class of fraud cases and 0 for valid cases, the result was 492 cases considered as fraud cases and 284315 cases as valid cases.



This is a Pie chart is plotted using the plotly library where values are being calculated by dividing the amount of fraud transactions over the total transaction and similarly the amount of valid transactions are calculated in the same way.

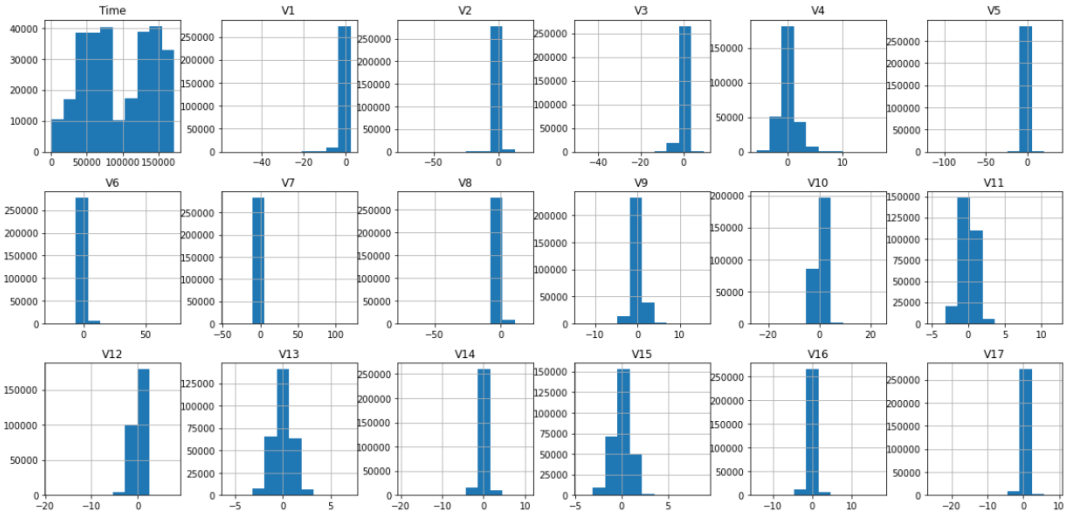
```
In [11]: 1 data.describe()
```

Out[11]:

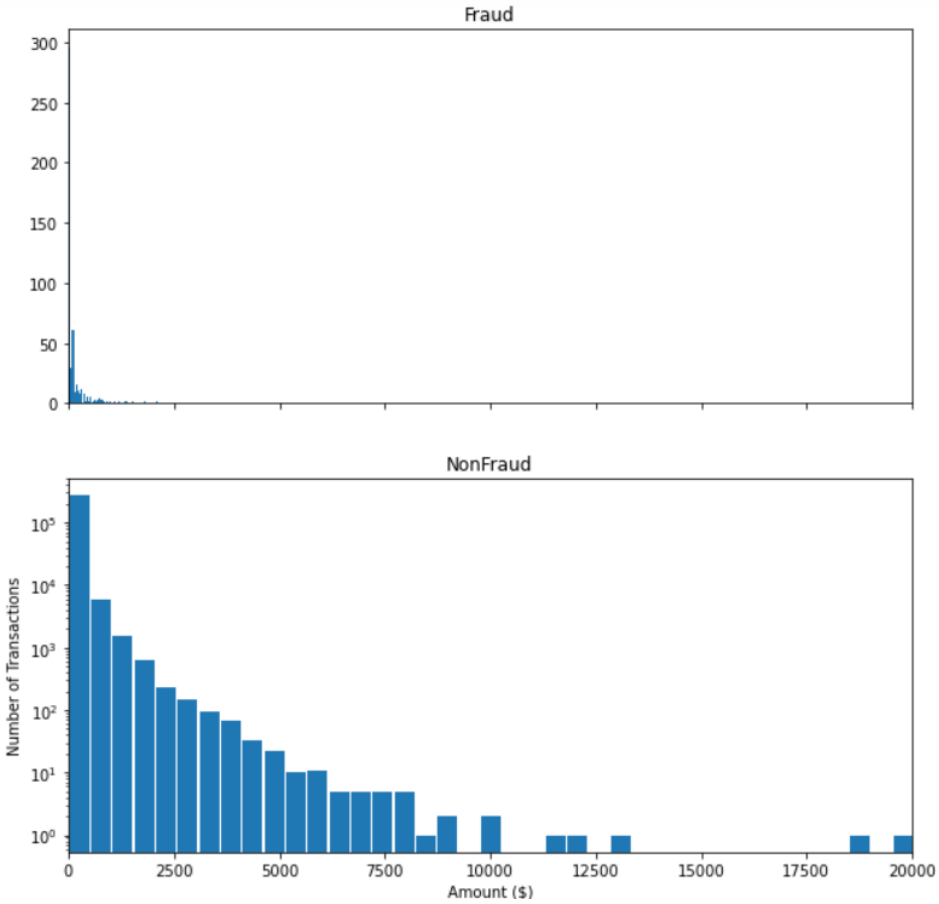
V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0e+05	...	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	284807.000000	284807.000000
0e-15	...	1.473120e-16	8.042109e-16	5.282512e-16	4.456271e-15	1.426896e-15	1.701640e-15	-3.662252e-16	-1.217809e-16	88.349619	0.001727
2e+00	...	7.345240e-01	7.257016e-01	6.244603e-01	6.056471e-01	5.212781e-01	4.822270e-01	4.036325e-01	3.300833e-01	250.120109	0.041527
7e+01	...	-3.483038e+01	-1.093314e+01	-4.480774e+01	-2.836627e+00	-1.029540e+01	-2.604551e+00	-2.256568e+01	-1.543008e+01	0.000000	0.000000
6e-01	...	-2.283949e-01	-5.423504e-01	-1.618463e-01	-3.545861e-01	-3.171451e-01	-3.269839e-01	-7.083953e-02	-5.295979e-02	5.600000	0.000000
3e-02	...	-2.945017e-02	6.781943e-03	-1.119293e-02	4.097606e-02	1.659350e-02	-5.213911e-02	1.342146e-03	1.124383e-02	22.000000	0.000000
0e-01	...	1.863772e-01	5.285536e-01	1.476421e-01	4.395266e-01	3.507156e-01	2.409522e-01	9.104512e-02	7.827995e-02	77.165000	0.000000
9e+01	...	2.720284e+01	1.050309e+01	2.252841e+01	4.584549e+00	7.519589e+00	3.517346e+00	3.161220e+01	3.384781e+01	25691.160000	1.000000

After checking all the statistical indicators, the class is the important part here so we have the max as 1 in fraud and as for the non-fraud we have a min of 0 that is non fraud.

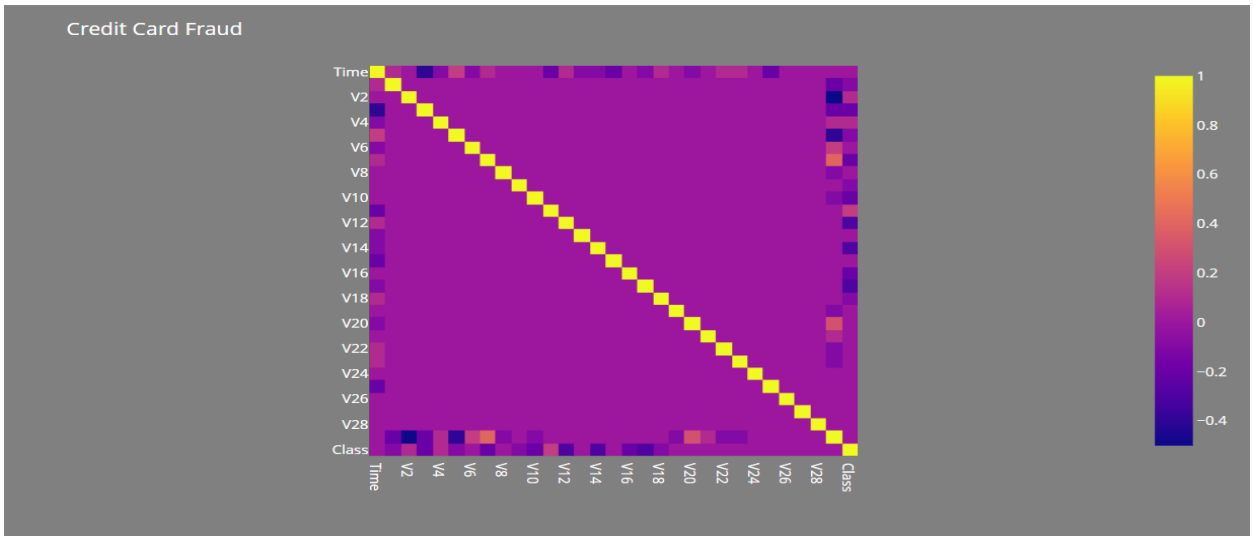
```
In [14]: 1 data.hist(figsize = (20,20))
         2 plt.show()
```



Plotting a histogram of each of parameter from V1 to V28. As we can see most of our V's are clustered right around 0 with some fairly large outliers or no outliers.

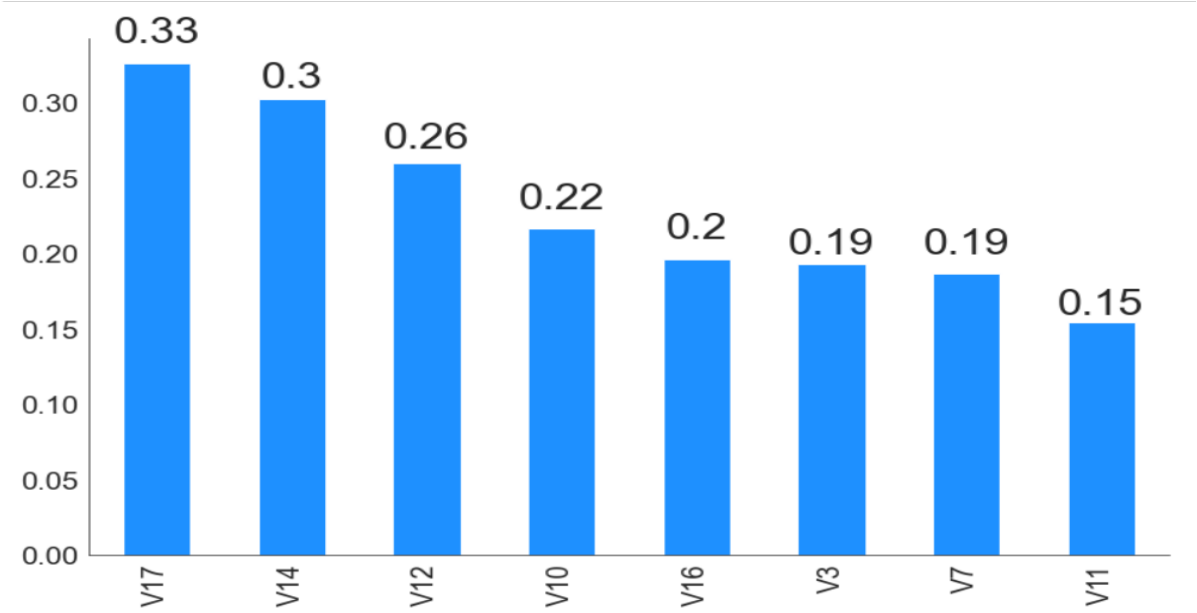


These 2 graphs show the amount of transaction per class where we can see very low amount of transactions for the fraud cases unlike the valid cases where we can see that when the amount of transactions is between 0 and 2500$ the number of transactions is absolutely high where there is more than 100 thousand transaction and low transactions for higher amounts more than 10k $ scoring only 1 transaction.



The above figure is a correlation matrix with the heat map where there's a lot of values close to zero so there are not strong relationships between the different V parameters the v1 though v28 most of them are fairly unrelated to the others. However, the class is the interesting part here where there are some variations in the relationships between the different parameters and the class here, so the lighter ones are going to be a positive correlation while the darker would be strong negative correlation. Moreover, there is not a strong correlation between amount and whether its fraudulent or not or even time whether its fraudulent or not.

V3, V7, V10, V11, V12, V14, V16, and V17 have a strong correlation with target(class) compared to other features. And down below is the figure that illustrates the correlation between these parameters and the class.



Now, It's good to check for duplicate entries and drop these entries where 1081 rows are duplicated and then dropped.

checking duplicate entries

In [19]:
```
duplicate_data = data[data.duplicated()]
print("Number of Duplicate rows : ",duplicate_data.shape)
```

```
Number of Duplicate rows :  (1081, 31)
```

Dropping Duplicate Entries

In [20]:
```
#dropping duplicates
data.drop_duplicates(inplace=True)
```

checking duplicate entries again

In [21]:
```
#checking duplicate entries again
duplicate_data = data[data.duplicated()]
print("Number of Duplicate rows : ",duplicate_data.shape)
```

```
Number of Duplicate rows :  (0, 31)
```

After checking the minimum and maximum amount of data it appears that the difference is huge and its good practice to normalize the data using the standard scalar that follows the Z-Score approach. So comparing the amounts with normalized amount would look like this:

In [23]:
```
scaler = StandardScaler()
data["NormalizedAmount"] = scaler.fit_transform(data["Amount"].values.reshape(-1, 1))
print(data['Amount'].head(10))
print('----------------------------------------------')
print(data['NormalizedAmount'].head(10))
print('----------------------------------------------')
```

```
0    149.62
1      2.69
2    378.66
3    123.50
4     69.99
5      3.67
6      4.99
7     40.80
8     93.20
9      3.68
Name: Amount, dtype: float64
----------------------------------------------
0    0.244200
1   -0.342584
2    1.158900
3    0.139886
4   -0.073813
5   -0.338670
6   -0.333399
7   -0.190387
8    0.018879
9   -0.338630
Name: NormalizedAmount, dtype: float64
----------------------------------------------
```

3.3 – Training and Testing

Before splitting our dataset into training set and testing set its good to declare 2 variables that refers to independent and dependent variables. The test size of the testing set is 30% as the best practice while 70% would be for the testing set. Moreover, the training and testing set shape is presented below:

Training and Testing

In [24]:
```
dependent_var = data.drop('Class', axis = 1).values # axis = 1 column wise operation
independent_var = data['Class'].values
```

In [25]:
```
# size of the test set is 25%
dependent_var_train, dependent_var_test, independent_var_train, independent_var_test = train_test_split(dependent_var, indep
#random_state = 1 maintains the initial and last value and randomize the rest
print("Shape of dependent variable train: ", dependent_var_train.shape)
print("Shape of dependent variable test: ", dependent_var_test.shape)
```

```
Shape of dependent variable train:  (198608, 31)
Shape of dependent variable test:  (85118, 31)
```

3.4 – Model Building

Machine Learning and data science are interrelated fields it's one of the many tools used in data science. Although there are many machine different algorithm, six different machine learning algorithms are used in this project namely Decision Tree, K-Nearest Neighbors (KNN), Logistic Regression, Support Vector Machine (SVM), Random Forest, and XGBoost. Due to the fact that the problem is considered as a classification problem, these models are a good fit for this problem. Before we start showing the models, it's good to introduce some key concepts:

the fit() function is used to train the models on the dataset and the predictions made by the models are recorded using the predict() function.

The max_depth parameter specifies the maximum depth of each tree. The default value for max_depth is None, which means that each tree will expand until every leaf is pure. A pure leaf is one where all of the data on the leaf comes from the same class. [4]

Decision Tree

```
In [27]:
1 decision_tree = DecisionTreeClassifier(criterion = 'entropy')

In [28]:
1 decision_tree.fit(dependent_var_train, independent_var_train)
2 predictions_dt = decision_tree.predict(dependent_var_test)

Decision Tree Score:  99.90953734815197

In [64]:
1 decision_tree_score = decision_tree.score(dependent_var_test, independent_var_test) * 100
2 print("Decision Tree Score: ", decision_tree_score)

Decision Tree Score:  99.90953734815197
```

The first model would be the decision tree where the main criteria used is the entropy that allows to build the tree where the lowest entropy would encounter for the highest information gain and would be considered as the root of the tree. The decision tree classifier is used but it was imported before alongside with other classifiers such as Logistic Regression, KNeighborsClassifier, RandomForestClassifier, XGBClassifier, SVC. Then the model is being fit by this classifier and the predicted values are stored in a variable called prediction_dt where it follows the correct the name conventions. The accuracy of this classifier is 99.909%.

Random Forest

```
In [29]:
1 random_forest = RandomForestClassifier(n_estimators= 100)

In [30]:
1 random_forest.fit(dependent_var_train, independent_var_train)
2 predictions_rf = random_forest.predict(dependent_var_test)

In [31]:
1 random_forest_score = random_forest.score(dependent_var_test, independent_var_test) * 100
2 print("Random Forest Score: ", random_forest_score)

Random Forest Score:  99.95418125425878
```

The second model is Random Forest. The RandomForestClassifier is used to build this model where the n_estimators accounts for the number of trees. Higher number of trees give you better performance but makes the code slower. Then the model is being fit by this classifier and the predicted values are stored in a variable called prediction_rf where it follows the correct the name conventions. The accuracy of this classifier is 99.954%.

The other models are implemented in the same way but each giving different accuracy.

KNN

In [32]:
```
n = 5
knn = KNeighborsClassifier(n_neighbors = n)
```

In [33]:
```
knn.fit(dependent_var_train, independent_var_train)
predictions_knn = knn.predict(dependent_var_test)
```

In [34]:
```
KNN_score = knn.score(dependent_var_test, independent_var_test) * 100
print("KNN Score: ", KNN_score)
```

```
KNN Score:  99.85784440423883
```

Logistic Regression

In [35]:
```
lr = LogisticRegression()
```

In [36]:
```
lr.fit(dependent_var_train, independent_var_train)
predictions_lr = lr.predict(dependent_var_test)
```

In [37]:
```
lr_score = lr.score(dependent_var_test, independent_var_test) * 100
print("Logistic Regression Score: ", lr_score)
```

```
Logistic Regression Score:  99.91541154632392
```

SVM

In [38]:
```
svm = SVC()
```

In [39]:
```
svm.fit(dependent_var_train, independent_var_train)
predictions_svm = svm.predict(dependent_var_test)
```

In [40]:
```
svm_score = svm.score(dependent_var_test, independent_var_test) * 100
print("SVM Score: ", svm_score)
```

```
SVM Score:  99.85314504570127
```

XGBoost

In [41]:
```
xgb = XGBClassifier(max_depth = 4)
```

In [42]:
```
xgb.fit(dependent_var_train, independent_var_train)
predictions_xgb = xgb.predict(dependent_var_test)
```

```
[18:25:01] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'b
inary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

In [43]:
```
xgb_score = xgb.score(dependent_var_test, independent_var_test) * 100
print("XGB Score: ", xgb_score)
```

```
XGB Score:  99.95535609389319
```

Confusion Matrix:

This is a performance measurement technique for ML classifiers. It is represented by tables to help the ML engineer correctly identify the performance of his or her classification model. So, a confusion matrix will be showing the degree of success of each model when comparing the predicted values to the real values. Then confusion matrix is plotted as a correlation table at first but then in a form of a heat map. This visualization of all the classification models help to identify which model performed better than the other in prediction. Now the function to draw a confusion matrix is shown below and all the results are displayed.

Implementation:

```
def plot_confusion_matrix(cm, classes, title, normalize = False, cmap = plt.cm.Blues):
    title = 'Confusion Matrix of {}'.format(title)
    if normalize:
        cm = cm.astype(float) / cm.sum(axis=1)[:, np.newaxis]

    plt.imshow(cm, interpolation = 'nearest', cmap = cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation = 45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment = 'center',
                 color = 'white' if cm[i, j] > thresh else 'black')

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

Here we started by defining a function called plot_confusion_matrix that takes 5 parameters

cm: confusion matrix,
classes: [0,1]
title: the title of our figure.
Normalize=false: because in our case we need to show how many cases are classified in real numbers. We well show an example later on with and without normalize,
Cmap: to give a specific color to our cm.

Confusion Matrix - Decision Tree:

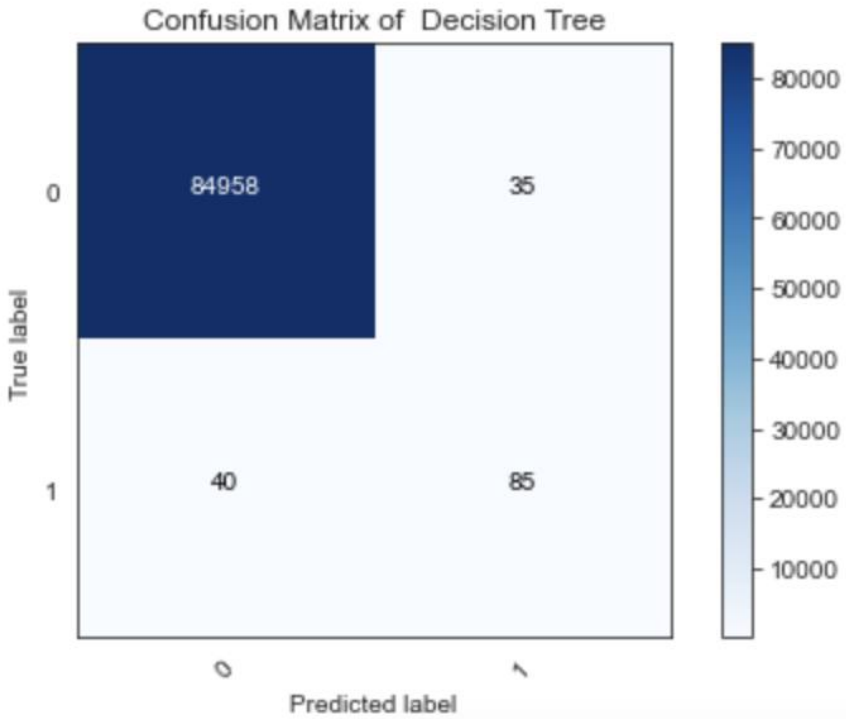
```
confusion_matrix_dt = confusion_matrix(independent_var_test, predictions_dt.round())
print("Confusion Matrix - Decision Tree")
print(confusion_matrix_dt)
```

The confusion_matrix for the decision tree is shown below:

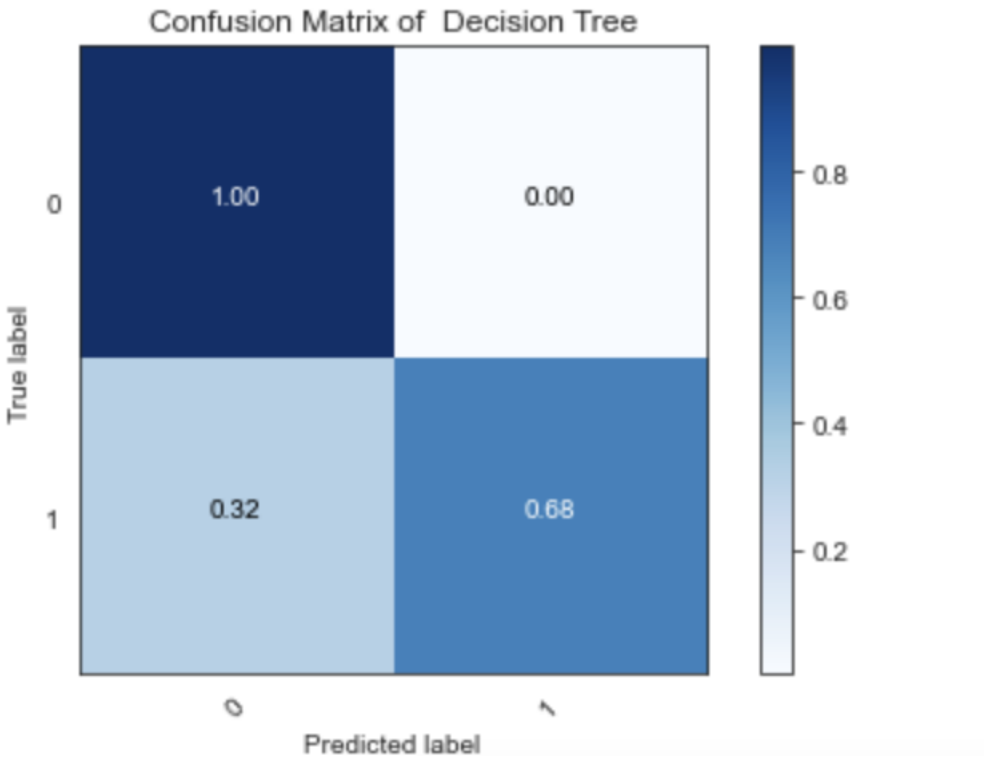
```
Confusion Matrix - Decision Tree
[[84958    35]
 [   40    85]]
```

Then we plot our Confusion Matrix - Decision Tree:

Ps: normalize = False

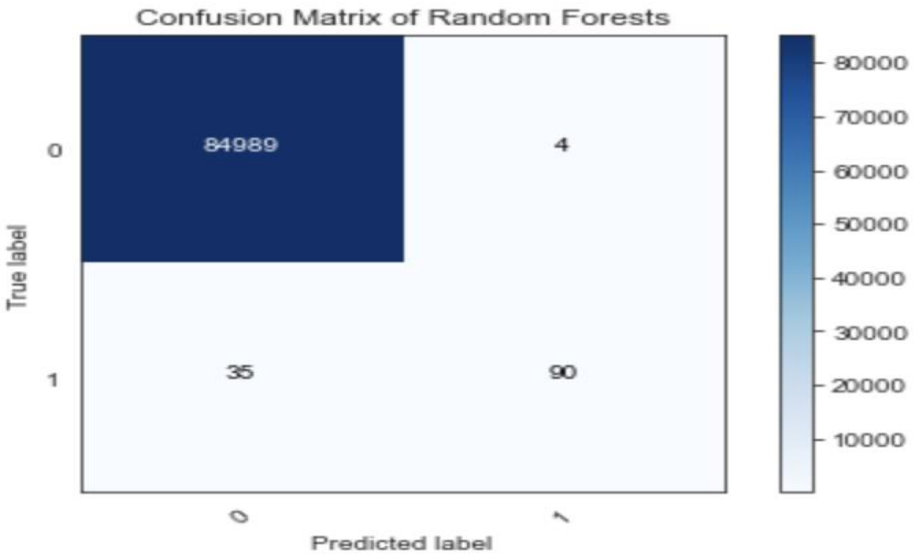


normalize = True



Confusion Matrix - Random Forests:

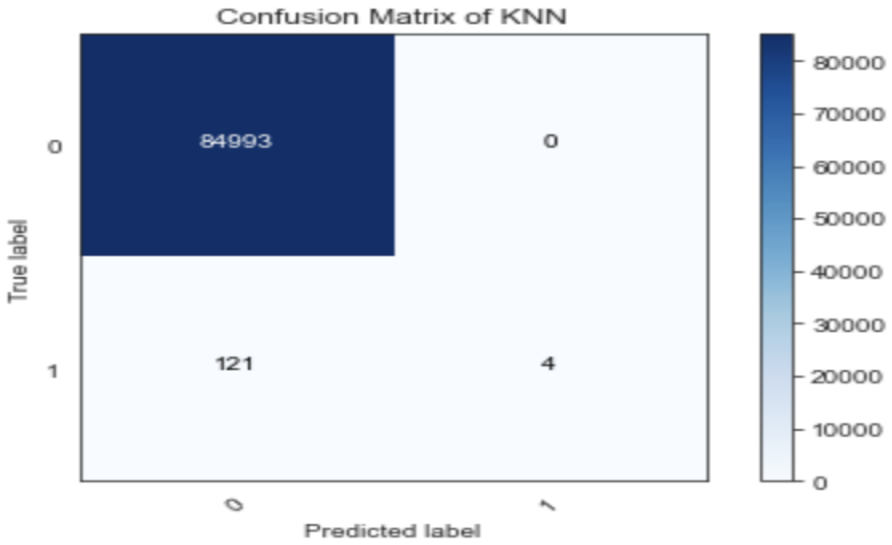
```
]: confusion_matrix_rf = confusion_matrix(independent_var_test, predictions_rf.round())
   print("Confusion Matrix - Random Forests")
   print(confusion_matrix_rf)

   Confusion Matrix - Random Forests
   [[84989     4]
    [   35    90]]
```



Confusion Matrix – KNN:

```
confusion_matrix_knn = confusion_matrix(independent_var_test, predictions_knn.round())
print("Confusion Matrix - KNN")
print(confusion_matrix_knn)
```

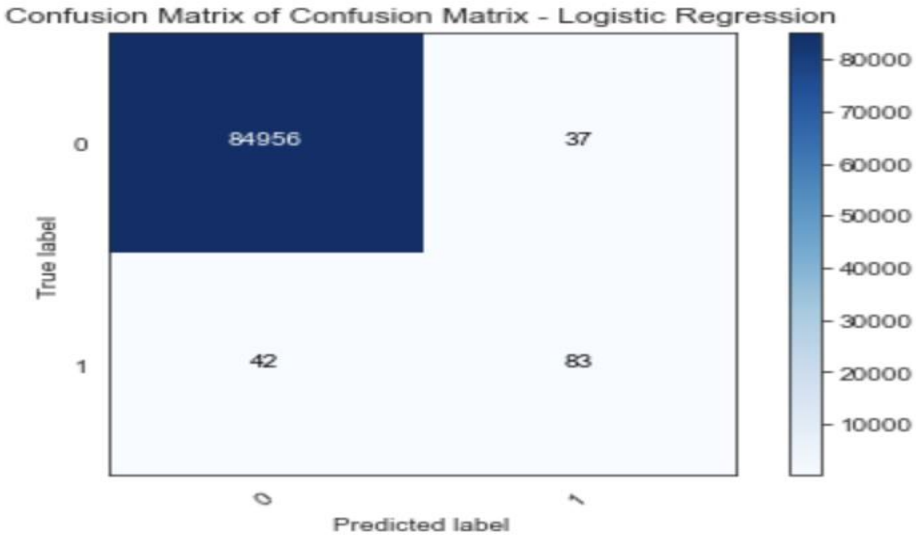
```
Confusion Matrix - KNN
[[84993     0]
 [  121     4]]
```



Confusion Matrix - Logistic Regression:

```
confusion_matrix_lr = confusion_matrix(independent_var_test, predictions_lr.round())
print("Confusion Matrix - Logistic Regression")
print(confusion_matrix_lr)
```

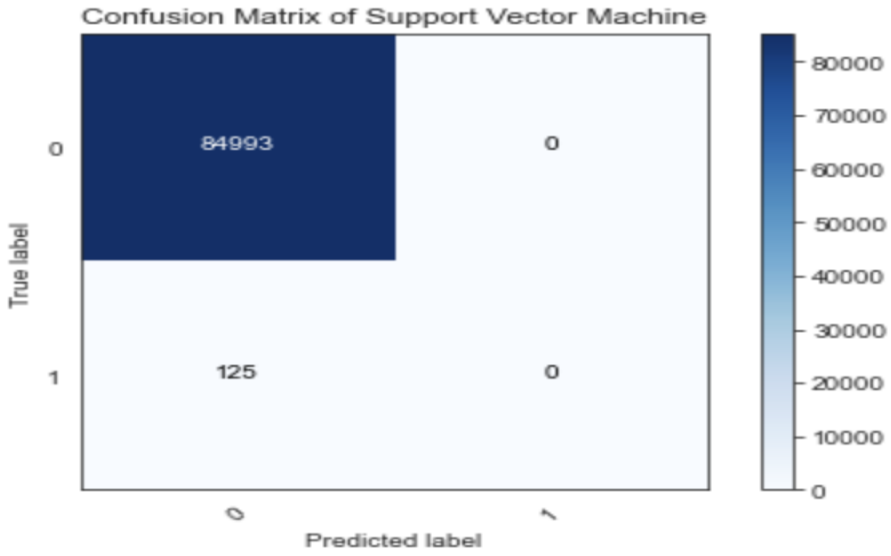
```
Confusion Matrix - Logistic Regression
[[84956    37]
 [   42    83]]
```



Confusion Matrix - Support Vector Machine:

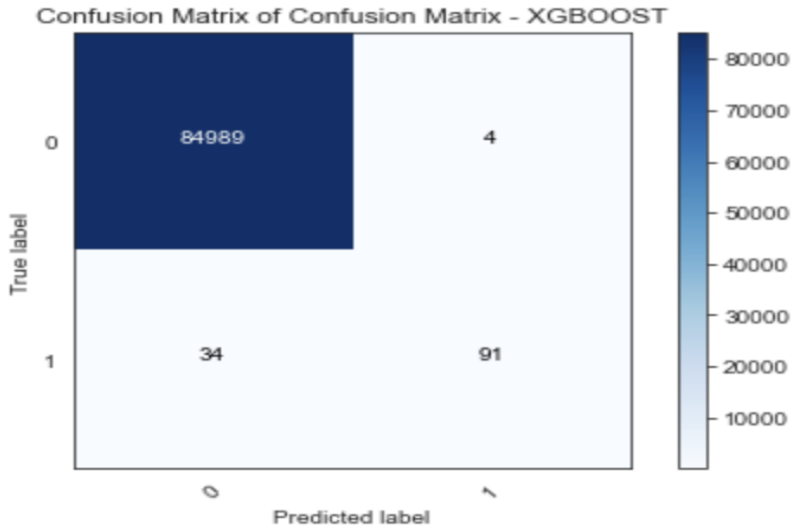
```
confusion_matrix_svm = confusion_matrix(independent_var_test, predictions_svm.round())
print("Confusion Matrix - Support Vector Machine")
print(confusion_matrix_svm)
```

```
Confusion Matrix - Support Vector Machine
[[84993     0]
 [  125     0]]
```



Confusion Matrix – XGBOOST:

```
confusion_matrix_xgb = confusion_matrix(independent_var_test, predictions_xgb.round())
print("Confusion Matrix - XGBOOST")
print(confusion_matrix_xgb)
```



Now comparing the results obtained in all of the above confusion matrix it appears that XGBOOST was the best classifier having accuracy of 99.955. So, we have 84989 as true positive and 4 are false negative. That says, out of 84989+4=84993, we have 849893 that are successfully classified as a valid transaction and 4 were falsely classified as normal but in fact they are fraudulent. Note that the first row contains the TP and the FN

TP: the number of positive cases correctly classified by the classifier.

TN: the number of negative cases correctly classified by the classifier.

FP: the number of positive cases that are incorrectly classified by the classifier.

FN: the number of negative cases that are incorrectly classified by the classifier.

As for the second row. It looks like there were 125 transactions whose fraud value was 1. The classifier correctly predicted 91(TN) of them as 1, and 34(FP) of them wrongly as 0. The wrongly predicted values can be considered as the error of the model. Comparing to other models the XGBOOST classifier performed better than other classifiers where it predicted the fraud transaction and the valid ones' better than other classifiers. Now to be more sure the evaluation of these classifiers is done below using the accuracy and F1-Score.

Accuracy can be calculated by dividing the number of correct prediction by the total predictions. As for the F1-Score which is the harmonic mean of the model's precision and recall and be calculated as follows:

$$\frac{2 \times precision \times recall}{precision + recall}$$

It's good to note that precision and recall and F1-Score are easily calculated in python by importing the necessary packages in the sci-kit learn library.

```
print("Evaluation of Decision Tree Model")
print()
metrics(independent_var_test, predictions_dt.round())
```

```
Evaluation of Decision Tree Model

Accuracy: 0.99910
Precision: 0.70000
Recall: 0.67200
F1-score: 0.68571
```

```
print("Evaluation of Random Forest Model")
print()
metrics(independent_var_test, predictions_rf.round())
```

```
Evaluation of Random Forest Model

Accuracy: 0.99954
Precision: 0.95745
Recall: 0.72000
F1-score: 0.82192
```

```
print("Evaluation of KNN Model")
print()
metrics(independent_var_test, predictions_knn.round())
```

```
Evaluation of KNN Model

Accuracy: 0.99858
Precision: 1.00000
Recall: 0.03200
F1-score: 0.06202
```

```
print("Evaluation of logistic Regression Model")
print()
metrics(independent_var_test, predictions_lr.round())
```

```
Evaluation of logistic Regression Model

Accuracy: 0.99915
Precision: 0.73874
Recall: 0.65600
F1-score: 0.69492
```

```
print("Evaluation of Supportt Vector Machine Model")
print()
metrics(independent_var_test, predictions_svm.round())
```

```
Evaluation of Supportt Vector Machine Model

Accuracy: 0.99853
Precision: 0.00000
Recall: 0.00000
F1-score: 0.00000
```

```
print("Evaluation of XGBOOST Model")
print()
metrics(independent_var_test, predictions_xgb.round())
```

```
Evaluation of XGBOOST Model

Accuracy: 0.99955
Precision: 0.95789
Recall: 0.72800
F1-score: 0.82727
```

XGBOOST and random forests ranks the 2 best classifiers in this report.

Chapter 4 – Conclusion

4.1 – Conclusion

This project consolidated the explanatory data analysis with the implementation of machine learning algorithm. The dataset was fully explored and visualized using different techniques of visualization either bar graph, correlation matrix, histograms and pie charts. This helped to make the data clearer and more understandable. Then, six classification models were implemented starting from decision trees to XGBOOST. All things considered, these models were evaluated using different evaluation metrics from F1-Score to accuracy, recall and precision. The results were satisfying where XGBOOST had the best accuracy of 99.995% followed by random forests. Finally, due to the fact that the data is real data, it can be noticed that more than 99% of transaction are valid and not fraud while the fraud cases accounts for 0.17%. This creates an imbalance between the classes and this imbalance issue will cause bias towards the non-fraud transaction accounting for higher accuracy. Thus, there are various techniques to deal with this imbalance such as oversampling or under sampling that can be discussed in the future.

4.2 – Percentage Completion

100% completion of this project. Fully functional code at the end. No errors occurred

Source Code:

```
import numpy as np
import pandas as pd
import plotly.graph_objects as go
import matplotlib.pyplot as plt
import plotly.express as px
import seaborn as sns

import warnings
from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler, MinMaxScaler, MaxAbsScaler
from sklearn.metrics import accuracy_score, precision_score, confusion_matrix, recall_score, f1_score
#from imblearn.over_sampling import SMOTE
#from collections import Counter
from sklearn.tree import DecisionTreeClassifier # Decision Tree algorithm
from sklearn.neighbors import KNeighborsClassifier # KNN algorithm
from sklearn.linear_model import LogisticRegression # Logistic regression algorithm
from sklearn.svm import SVC # SVM algorithm
from sklearn.ensemble import RandomForestClassifier # Random forest tree algorithm
from xgboost import XGBClassifier # XGBoost algorithm

import itertools
warnings.filterwarnings('ignore')
```

```
data=pd.read_csv('creditcard.csv')
```

```
data.head(5).T.style.set_properties(**{'background-color': 'grey',
                            'color': 'white',
                            'border-color': 'white'})
```

```
data.columns
```

```
data.shape
```

```
import missingno as msno
msno.bar(data, color="dodgerblue", sort="ascending", figsize=(18,5), fontsize=12)

plt.title("Missing Data")
```

```
plt.figure(figsize = (8,6))
ax = data.dtypes.value_counts().plot(kind='bar',grid = False,fontsize=20,color='dodgerblue')
for p in ax.patches:
    height = p.get_height()
    ax.text(p.get_x()+ p.get_width() / 2., height + 0.1, height, ha = 'center', size = 20)
sns.despine()
```

```
data.memory_usage()
```

```
Fraud = data[data['Class']==1]
print('Number of Fraud Cases:',len(Fraud))
NonFraud = data[data['Class']==0]
print('Number of Non Fraud Cases:',len(NonFraud))
```

```
colors = ['dodgerblue', 'red']
labels = ['NonFraud','Fraud']
values = data['Class'].value_counts()/data['Class'].shape[0] # shape[0] gives number of row count
fig = go.Figure(data=[go.Pie(labels = labels,
                             values=values,hole=.3)])
fig.update_traces(hoverinfo='label+percent', textinfo='percent', textfont_size=20,
                  marker=dict(colors=colors, line=dict(color='white', width=0.1)))
fig.update_layout(
    title_text="Credit Card Fraud",
    title_font_color="white",
    legend_title_font_color="white",
    paper_bgcolor="black",
    plot_bgcolor='black',
    font_color="white",
)
fig.show()
```

```
data.describe()
```

```
Fraud.Amount.describe()
```

```
NonFraud.Amount.describe()
```

```
data.hist(figsize = (20,20))
plt.show()
```

```
#The towers or bars of a histogram are called bins.
f, (ax1, ax2) = plt.subplots(2, sharex=True, figsize=(10,10))
f.suptitle('Amount per transaction by class')
bins = 50
ax1.hist(Fraud.Amount, bins = bins, rwidth=0.9)
ax1.set_title('Fraud')
ax2.hist(NonFraud.Amount, bins = bins, rwidth=0.9)
ax2.set_title('NonFraud')
plt.xlabel('Amount ($)')
plt.ylabel('Number of Transactions')
plt.xlim((0, 20000))#used to get or set the x-limits of the current axes
plt.yscale('log')# converts the y-axis to a logarthmic scale

plt.show();
```

```
# correlation gives us relation between each varibale. how much each variable is contributing.
plt.figure(figsize=(30,30))
corr = data.corr().round(1) #  Rounds mean to nearest integer, e.g. 1.95 = 2 and 1.05 = 1
fig = px.imshow(corr)# px.imshow displays multichannel (RGB) or single-channel ("grayscale") image data.
fig.update_layout(
    title_text="Credit Card Fraud",
    title_font_color="white",
    legend_title_font_color="yellow",
    paper_bgcolor="grey",
    plot_bgcolor='black',
    font_color="white",
)
fig.show()
```

```
# find relation betweeb the variables
data.corr()
```

```
features = [
    "V3",
    "V7",
    "V10",
    "V11",
    "V12",
    "V14",
    "V16",
    "V17",
]

plt.figure(figsize=(13,8))
sns.set_style("white")
# we used absolute values because we have negative values
ax = abs(data[features].corrwith(data.Class)).sort_values(ascending=False).plot(kind='bar',color='dodgerblue',fontsize=
# A patch is a 2D artist with a face color and an edge color.
for p in ax.patches:
    height = p.get_height().round(2)
    ax.text(p.get_x() + p.get_width() / 2., height+0.01, height, ha = 'center', size = 30)
#Remove the top and right spines from plot(s).
sns.despine()
```

```
duplicate_data = data[data.duplicated()]
print("Number of Duplicate rows : ",duplicate_data.shape)
```

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

```
duplicate_data = data[data.duplicated()]
print("Number of Duplicate rows : ",duplicate_data.shape)
```

```
min(data.Amount),max(data.Amount)
```

```
scaler = StandardScaler()
data["NormalizedAmount"] = scaler.fit_transform(data["Amount"].values.reshape(-1, 1))
print(data['Amount'].head(5))
print('-----------------------------------------')
print(data['NormalizedAmount'].head(5))
print('-----------------------------------------')
```

```
dependent_var = data.drop('Class', axis = 1).values # axis = 1 column wise operation
independent_var = data['Class'].values
```

```
# size of the test set is 30% => test_size
dependent_var_train, dependent_var_test, independent_var_train, independent_var_test = train_test_split(dependent_var,
#random_state = 1 maintains the initial and last value and randomize the rest
print("Shape of dependent variable train: ", dependent_var_train.shape)
print("Shape of dependent variable test: ", dependent_var_test.shape)
```

```
decision_tree = DecisionTreeClassifier(criterion = 'entropy')
```

```
random_forest = RandomForestClassifier(n_estimators= 100)
```

```
decision_tree.fit(dependent_var_train, independent_var_train)
predictions_dt = decision_tree.predict(dependent_var_test)
#Checking the accuracy by percentage
decision_tree_score = decision_tree.score(dependent_var_test, independent_var_test) * 100
print("Decision Tree Score: ", decision_tree_score)
```

```
decision_tree = DecisionTreeClassifier(criterion = 'entropy')
```

```
n = 5
knn = KNeighborsClassifier(n_neighbors = n)
```

```
random_forest = RandomForestClassifier(n_estimators= 100)
```

```
random_forest.fit(dependent_var_train, independent_var_train)
predictions_rf = random_forest.predict(dependent_var_test)

random_forest_score = random_forest.score(dependent_var_test, independent_var_test) * 100
print("Random Forest Score: ", random_forest_score)
```

```
n = 5
knn = KNeighborsClassifier(n_neighbors = n)
```

```
knn.fit(dependent_var_train, independent_var_train)
predictions_knn = knn.predict(dependent_var_test)
```

```
KNN_score = knn.score(dependent_var_test, independent_var_test) * 100
print("KNN Score: ", KNN_score)
```

```
lr = LogisticRegression()
```

```
lr.fit(dependent_var_train, independent_var_train)
predictions_lr = lr.predict(dependent_var_test)
```

```
lr_score = lr.score(dependent_var_test, independent_var_test) * 100
print("Logistic Regression Score: ", lr_score)
```

```
svm = SVC()
```

```
svm.fit(dependent_var_train, independent_var_train)
predictions_svm = svm.predict(dependent_var_test)
```

```
svm_score = svm.score(dependent_var_test, independent_var_test) * 100
print("SVM Score: ", svm_score)
```

```
xgb = XGBClassifier(max_depth = 4)
```

```
xgb.fit(dependent_var_train, independent_var_train)
predictions_xgb = xgb.predict(dependent_var_test)
```

```
xgb_score = xgb.score(dependent_var_test, independent_var_test) * 100
print("XGB Score: ", xgb_score)
```

```
def plot_confusion_matrix(cm, classes, title, normalize = False, cmap = plt.cm.Blues):
    title = 'Confusion Matrix of {}'.format(title)
    if normalize:
        cm = cm.astype(float) / cm.sum(axis=1)[:, np.newaxis]

    plt.imshow(cm, interpolation = 'nearest', cmap = cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation = 45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment = 'center',
                 color = 'white' if cm[i, j] > thresh else 'black')

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

```
confusion_matrix_dt = confusion_matrix(independent_var_test, predictions_dt.round())
print("Confusion Matrix - Decision Tree")
print(confusion_matrix_dt)
```

```
plot_confusion_matrix(confusion_matrix_dt, classes=[0, 1], title= "Decision Tree",normalize = False)
```

```
confusion_matrix_rf = confusion_matrix(independent_var_test, predictions_rf.round())
print("Confusion Matrix - Random Forests")
print(confusion_matrix_rf)
```

```
plot_confusion_matrix(confusion_matrix_rf, classes=[0, 1], title= "Random Forests")
```

```
plot_confusion_matrix(confusion_matrix_knn, classes=[0, 1], title= "KNN")
```

```
confusion_matrix_knn = confusion_matrix(independent_var_test, predictions_knn.round())
print("Confusion Matrix - KNN")
print(confusion_matrix_knn)
```

```
plot_confusion_matrix(confusion_matrix_knn, classes=[0, 1], title= "KNN")
```

```
confusion_matrix_lr = confusion_matrix(independent_var_test, predictions_lr.round())
print("Confusion Matrix - Logistic Regression")
print(confusion_matrix_lr)
```

```
plot_confusion_matrix(confusion_matrix_lr, classes=[0, 1], title= "Logistic Regression")
```

```
confusion_matrix_svm = confusion_matrix(independent_var_test, predictions_svm.round())
print("Confusion Matrix - Support Vector Machine")
print(confusion_matrix_svm)
```

```
plot_confusion_matrix(confusion_matrix_svm, classes=[0, 1], title= "Support Vector Machine")
```

```
confusion_matrix_xgb = confusion_matrix(independent_var_test, predictions_xgb.round())
print("Confusion Matrix - XGBOOST")
print(confusion_matrix_xgb)
```

```
plot_confusion_matrix(confusion_matrix_xgb, classes=[0, 1], title= "Confusion Matrix - XGBOOST")
```

```
def metrics(actuals, predictions):
    print("Accuracy: {:.5f}".format(accuracy_score(actuals, predictions)))
    print("Precision: {:.5f}".format(precision_score(actuals, predictions)))
    print("Recall: {:.5f}".format(recall_score(actuals, predictions)))
    print("F1-score: {:.5f}".format(f1_score(actuals, predictions)))
```

```
print("Evaluation of Decision Tree Model")
print()
metrics(independent_var_test, predictions_dt.round())
```

```
print("Evaluation of Random Forest Model")
print()
metrics(independent_var_test, predictions_rf.round())
```

```
print("Evaluation of logistic Regression Model")
print()
metrics(independent_var_test, predictions_lr.round())
```

```
print("Evaluation of KNN Model")
print()
metrics(independent_var_test, predictions_knn.round())
```

```
print("Evaluation of logistic Regression Model")
print()
metrics(independent_var_test, predictions_lr.round())
```

```
print("Evaluation of Support Vector Machine Model")
print()
metrics(independent_var_test, predictions_svm.round())
```

```
print("Evaluation of XGBOOST Model")
print()
metrics(independent_var_test, predictions_xgb.round())
```

References

[1]"Imposter Scams Top Complaints Made to FTC in 2018", Federal Trade Commission, 2022. [Online]. Available: https://www.ftc.gov/news-events/press-releases/2019/02/imposter-scams-top-complaints-made-ftc-2018.

[2]"pandas - Python Data Analysis Library", Pandas.pydata.org, 2022. [Online]. Available: https://pandas.pydata.org/.

[3]"Plotly Python Graphing Library", Plotly.com, 2022. [Online]. Available: https://plotly.com/python/.

[4]"Optimizing Hyperparameters in Random Forest Classification", Medium, 2022. [Online]. Available: https://towardsdatascience.com/optimizing-hyperparameters-in-random-forest-classification-ec7741f9d3f6.

[5]"Credit Card Fraud Detection", Kaggle.com, 2022. [Online]. Available: https://www.kaggle.com/mlg-ulb/creditcardfraud.