A PROJECT REPORT ON

"CREDIT CARD FRAUD DETECTION"

By **Eshan Naik**

TLS21A2393

INDEX

ABSTRACT

In today's world with the increasing number of online transactions due to e commerce websites, online payment of bills, etc. making the threat of a Credit Card Fraud is ever increasing. In the year 2019 to 2020 the number of identity thefts went up by 113% and the number of identity thefts due to credit cards increased by 44.6%. Out of a total of 1.4 million identity thefts in the USA almost 400 thousand of them were due to Credit Card Frauds. This resulted in a total loss of over 24 billion US dollars globally of which USA contributed over 16 billion US dollars only due to Credit Card Frauds.

Due to this increasing rate in Credit Card Frauds many researchers have started to develop multiple Machine Learning and Deep Learning Models to counter this issue. The main aim of this project is to analysis a credit card dataset and make Machine Learning and Deep Learning models to predict if a transaction or account is due to a Credit Card Fraud given some data. In this project I work on a general Credit Card Detection Dataset from the month September 2013.

INTRODUCTION

The main aim of this project is to predict if a Credit Card Fraud has occurred given a Dataset.

The Dataset chosen is Credit Card Fraud Detection Dataset from the month September 2013. The total number of transactions that are present in the Dataset is 284,807 of which 492 are frauds which is 0.173% percent of the Dataset. The Dataset has been got from Kaggle and consists of a total of 31 columns.

It consists of only numerical input variables which are the result of a PCA transformation except the columns ‘Time’ and ‘Amount’.

The Main columns present in this Dataset are ‘Time’, ’Amount’ and ‘Class’ while the rest 28 columns labelled V1, V2, V3, etc. are factors. The Class column has values 1 and 0 which determine if a fraud has occurred or not, 0 representing not a fraud and 1 representing a fraud.

The Models that we will be using consist of an Auto Encoder, Logistic Regression, Random Forest, K Nearest Neighbours. In the end we will compare the performance of each model.

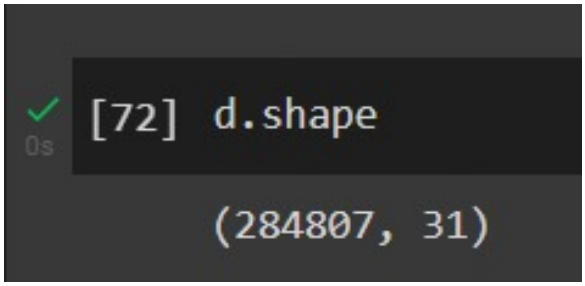
TASKS

1. DATA ACQUISITION AND CLEANING
2. DATA VISUALIZATION
3. DATA MODELLING
4. TESTING
5. COMPARISON AND MEASUREMENT

1. DATA ACQUISITION AND CLEANING

A single dataset has been used which has a total of 31 columns and 284807 rows.

This dataset has been acquired from Kaggle.com.



The main columns are –

- Time – Contains the seconds elapsed between each transaction and the first transaction in the dataset.
- Amount – Contains the transaction amount.
- Class – Has two values 0 and 1, where 0 represents a fraud and 1 represents a fraud.

  The other 28 columns consist of only numeric values which have been transformed using PCA Transformation and consist of various factors for our Models to learn.

There are no missing values in the dataset and hence the only pre-processing that is required is normalization of the dataset columns excluding the 3 major columns using Min - Max Normalization from the sklearn package.

This dataset is highly imbalanced as the total number of fraud cases represented are only 0.173% of the total dataset making it harder for the Models to learn. To combat this, we will be using only 1000 Not Fraud Cases.

Input variables based on physicochemical tests:

- Time
- Amount
- 28 Columns consisting of Factors.

Output variable based on sensory data:

- Fraud or Not a Fraud (0 or 1)

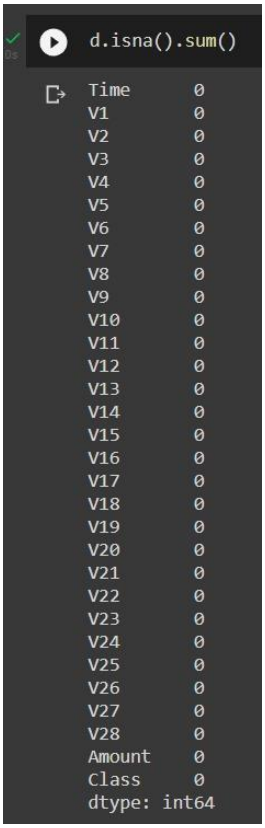
The packages imported are –

- Pandas
- Numpy
- Matplotlib.pyplot
- Keras
    - Layers
    - Models
- Sklearn
    - Manifold
    - Preprocessing
    - Model_slection
    - Linear_model
    - Metrics
    - Ensemble
    - Neighbors
- Seaborn

```
[ ]  import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns
     import keras
     from keras.layers import Dense,Input
     from sklearn.manifold import TSNE
     from sklearn.preprocessing import MinMaxScaler
     from tensorflow.keras import regularizers
     from keras.models import Model, Sequential
     from sklearn.model_selection import train_test_split
     from sklearn.linear_model import LogisticRegression
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.metrics import classification_report, accuracy_score
```

2. DATA VISUALIZATION

First we see if there are any missing values



As we can see there exists no missing values.

We will now see the statistics summary of the dataset in the below image.

```
d.describe()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e
mean	94813.859575	3.919560e-15	5.688174e-16	-8.769071e-15	2.782312e-15	-1.552563e-15	2.010663e-15	-1.694249e-15	-1.927028e-16	-3.137024e
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00	1.098632e
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01	-1.343407e
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01	-6.430976e
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02	-5.142873e
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01	5.971390e
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01	1.559499e

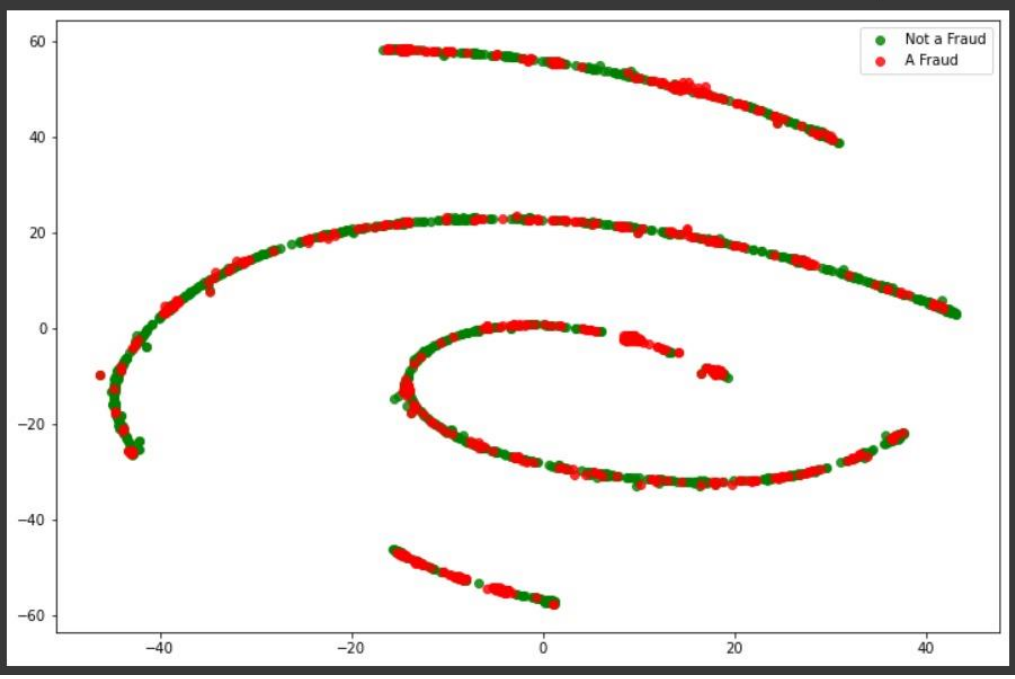
Below is a Pie Chart made using matplotlib.pyplot to visualise the total number of fraud and non-fraud cases.

```
[77] l=["Not Fraud","Fraud"]
     a=[zero,one]
     plt.pie(x=a,labels=l,autopct="%1.1f%%")
     plt.legend()
     plt.show()
```

As we can see Non Fraud cases make up 99.9% of the dataset and Fraud cases make up only 0.1% of the dataset making the dataset very imbalanced.
To combat this, we will consider only 1000 non fraud cases taken at random from the dataset.

Now we visualise the new data using TSNE

```
tsne = TSNE(n_components=2, random_state=0)
x1 = tsne.fit_transform(x)

plt.figure(figsize=(12,8))
plt.scatter(x1[np.where(y==0),0],x1[np.where(y==0),1],color='green',alpha=0.8, label="Not a Fraud")
plt.scatter(x1[np.where(y==1),0],x1[np.where(y==1),1],color='red',alpha=0.8, label="A Fraud")
plt.legend()
plt.show()
```



The red dots represent fraud cases and the green dots represent not fraud cases.

3. DATA MODELLING

During the data visualization part, we learnt that the number of fraud cases to the number of non-fraud cases was very low and hence had to sample out only 1000 non fraud cases. We also saw the statistical summary of the dataset and also concluded that the dataset doesn't consist of any missing values.

In this section we will look at the different models the we have run on the dataset.

Namely-

- Auto Encoders using Dense Layers
- Logistic Regression
- Random Forest
- K Nearest Neighbours

Auto Encoders:

An Auto Encoder is a special type of feed forward neural network which does the following:

- Encodes its input xi into a hidden representation h
- Decodes the input again from this hidden representation
- The model is trained to minimize a certain loss function which will ensure that xˆi is close to xi (we will see some such loss functions soon)

In this project we have used 5 Dense Layers and 1 Input Layer –

- The Dense layers 1,2 and 3 have a tanh activation function whereas Dense layer 2 has a sigmoid function.
- The Final output layer consists of the 5th Dense layer with a softmax activation function as this is the best activation function when the output is either 1 or 0.
- The first Dense Layer also has regularization.

The Optimizer used is the adam's optimizer and the loss function is mse.

The metric to measure loss is accuracy.

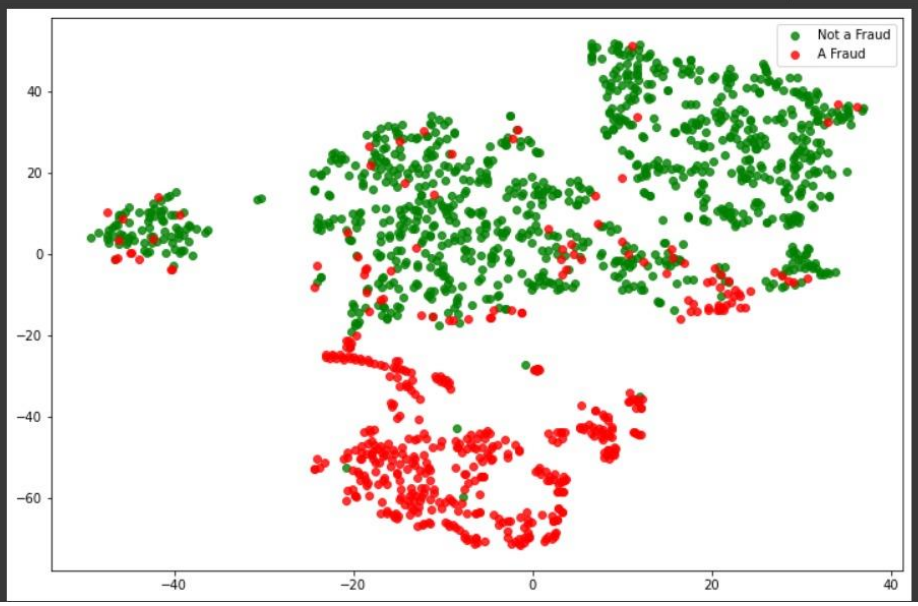
```
i = Input(shape=(x.shape[1],))

h1 = Dense(units=128,activation='tanh',activity_regularizer=regularizers.l1(10e-5))(i)
code = Dense(units=64,activation='sigmoid')(h1)

h2 = Dense(units=64,activation='tanh')(code)
h3 = Dense(units=128,activation='tanh')(h2)

o = Dense(x.shape[1],activation="softmax")(h3)

autoencoder = Model(i,o)

autoencoder.compile(optimizer="adam",loss="mse",metrics=['accuracy'])
```

We run this Model for 200 epochs with a validation split of 20% and a batch size of 64.

```
[122] autoencoder.fit(x_nf[0:2000], x_nf[0:2000], batch_size = 64, epochs = 200, shuffle = True, validation_split = 0.20)
    13/13 [==============================] - 0s 4ms/step - loss: 0.3454 - accuracy: 0.8363 - val_loss: 0.3445 - val_accuracy: 0.8800
    Epoch 172/200
    13/13 [==============================] - 0s 6ms/step - loss: 0.3454 - accuracy: 0.8363 - val_loss: 0.3445 - val_accuracy: 0.8600
    Epoch 173/200
    13/13 [==============================] - 0s 4ms/step - loss: 0.3454 - accuracy: 0.8350 - val_loss: 0.3445 - val_accuracy: 0.8100
    Epoch 174/200
    13/13 [==============================] - 0s 4ms/step - loss: 0.3454 - accuracy: 0.8400 - val_loss: 0.3445 - val_accuracy: 0.8000
    Epoch 175/200
    13/13 [==============================] - 0s 6ms/step - loss: 0.3454 - accuracy: 0.8375 - val_loss: 0.3444 - val_accuracy: 0.8400
    Epoch 176/200
    13/13 [==============================] - 0s 4ms/step - loss: 0.3454 - accuracy: 0.8425 - val_loss: 0.3444 - val_accuracy: 0.8800
    Epoch 177/200
    13/13 [==============================] - 0s 5ms/step - loss: 0.3454 - accuracy: 0.8350 - val_loss: 0.3445 - val_accuracy: 0.8750
    Epoch 178/200
    13/13 [==============================] - 0s 5ms/step - loss: 0.3454 - accuracy: 0.8275 - val_loss: 0.3445 - val_accuracy: 0.8050
    Epoch 179/200
    13/13 [==============================] - 0s 5ms/step - loss: 0.3454 - accuracy: 0.8388 - val_loss: 0.3444 - val_accuracy: 0.8750
    Epoch 180/200
    13/13 [==============================] - 0s 4ms/step - loss: 0.3454 - accuracy: 0.8438 - val_loss: 0.3444 - val_accuracy: 0.8750
    Epoch 181/200
    13/13 [==============================] - 0s 6ms/step - loss: 0.3454 - accuracy: 0.8462 - val_loss: 0.3444 - val_accuracy: 0.8700
    Epoch 182/200
    13/13 [==============================] - 0s 7ms/step - loss: 0.3454 - accuracy: 0.8363 - val_loss: 0.3444 - val_accuracy: 0.8750
    Epoch 183/200
    13/13 [==============================] - 0s 5ms/step - loss: 0.3454 - accuracy: 0.8587 - val_loss: 0.3445 - val_accuracy: 0.8750
    Epoch 184/200
    13/13 [==============================] - 0s 5ms/step - loss: 0.3454 - accuracy: 0.8475 - val_loss: 0.3445 - val_accuracy: 0.8750
    Epoch 185/200
    13/13 [==============================] - 0s 6ms/step - loss: 0.3454 - accuracy: 0.8138 - val_loss: 0.3445 - val_accuracy: 0.8800
    Epoch 186/200
    13/13 [==============================] - 0s 6ms/step - loss: 0.3454 - accuracy: 0.7900 - val_loss: 0.3445 - val_accuracy: 0.6950
    Epoch 187/200
    13/13 [==============================] - 0s 4ms/step - loss: 0.3454 - accuracy: 0.7950 - val_loss: 0.3445 - val_accuracy: 0.7950
```

Below is a visualization of the predictions made by the Auto Encoder model

```
tsne = TSNE(n_components=2, random_state=0)
normal1 = tsne.fit_transform(rep_x)

plt.figure(figsize=(12,8))
plt.scatter(normal1[np.where(rep_y==0),0],normal1[np.where(rep_y==0),1],color='green',alpha=0.8, label="Not a Fraud")
plt.scatter(normal1[np.where(rep_y==1),0],normal1[np.where(rep_y==1),1],color='red',alpha=0.8, label="A Fraud")
plt.legend()
plt.show()
```



Logistic Regression:

Logistic regression is a statistical model that uses a logistic function to model a binary dependent variable. Our binary dependent model being if a fraud (1) has occurred or not (0). The function that is used is sigmoid function. This is a supervised learning model.

There exist many different types of Logistic Regression Models like –

- Binary – The output will have only 2 values that is either 0 or 1. We use this model as our dataset best suits this model.
- Multinomial – There are 3 or more outputs possible and there exists no ordering
- Ordinal – Here there are 3 or more outputs possible and they are ordered.

As stated above we will be using a Binary Logistic Regression Model as our output can only be 0 or 1.

We train the Logistic Regression Model on 80% of the dataset which is known as x_train and y_train leaving the rest of the dataset for testing the model.

```
[90] x_train, x_test, y_train, y_test = train_test_split(rep_x, rep_y, test_size=0.20)

[91] lr = LogisticRegression()
     lr.fit(x_train,y_train)

     LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                        intercept_scaling=1, l1_ratio=None, max_iter=100,
                        multi_class='auto', n_jobs=None, penalty='l2',
                        random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                        warm_start=False)
```

Random Forest:

Random forests are one of the most commonly used ensemble methods which consists of multiple decision trees. They improve prediction performance over classification trees by averaging the multiple decision trees. The algorithm creates several random subsets of the original data, in this case the training set, and calculates the classification trees, then the final result is the average of all trees.

The name random forest derives from the random process of splitting the data and creating many trees, or a forest.

Below we train the Random Forest Classifier model on x_train and y_train.

```
[99] rf=RandomForestClassifier(n_estimators=100,random_state=0)
     rf.fit(x_train,y_train)

     RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                            criterion='gini', max_depth=None, max_features='auto',
                            max_leaf_nodes=None, max_samples=None,
                            min_impurity_decrease=0.0, min_impurity_split=None,
                            min_samples_leaf=1, min_samples_split=2,
                            min_weight_fraction_leaf=0.0, n_estimators=100,
                            n_jobs=None, oob_score=False, random_state=0, verbose=0,
                            warm_start=False)
```

K Nearest Neighbours:

The KNN model is based on the concept of Clustering. In clustering we group results that are similar to each other as close as possible while results less related are far from each other, which results in the formation of clusters. Hence we can say that this model is based on the concept of similarity. In our problem this model makes two clusters one for fraud cases and the other for non-fraud cases. If a non-fraud case is placed in the cluster with fraud cases, then that is an error and vice versa.

We train the K Neighbours Classifier on x_train and y_train.

```
[111] knn = KNeighborsClassifier()
      knn.fit(x_train,y_train)

      KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                           metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                           weights='uniform')
```

4. TESTING

We are keeping 20% of our dataset to treat it as unseen data and be able and test the performance of our models. This 20 percent is stored as x_test and y_test.

We use Classification Report to get the F1 score and Accuracy Score to get to accuracy of the model.

F1 score combines the two concepts called recall and precision to give a type of accuracy score. Recall is the ratio of the correctly predicted positive observations to all the observations in the class. Precision is the ratio between the number of correctly predicted positive observations against the total number of positive cases predicted by it.

Accuracy is the ratio between the correctly predicted observations to the total number of observations.

Auto Encoder

First we will test the Auto Encoder Model using the validation split.

```
13/13 [==============================] - 0s 7ms/step - loss: 0.3399 - accuracy: 0.8338 - val_loss: 0.3393 - val_accuracy: 0.8700
Epoch 183/200
13/13 [==============================] - 0s 4ms/step - loss: 0.3399 - accuracy: 0.8675 - val_loss: 0.3393 - val_accuracy: 0.8950
Epoch 184/200
13/13 [==============================] - 0s 6ms/step - loss: 0.3399 - accuracy: 0.8413 - val_loss: 0.3393 - val_accuracy: 0.8600
Epoch 185/200
13/13 [==============================] - 0s 4ms/step - loss: 0.3399 - accuracy: 0.8675 - val_loss: 0.3393 - val_accuracy: 0.8800
Epoch 186/200
13/13 [==============================] - 0s 4ms/step - loss: 0.3398 - accuracy: 0.8600 - val_loss: 0.3393 - val_accuracy: 0.8600
Epoch 187/200
13/13 [==============================] - 0s 5ms/step - loss: 0.3398 - accuracy: 0.8450 - val_loss: 0.3393 - val_accuracy: 0.9050
Epoch 188/200
13/13 [==============================] - 0s 6ms/step - loss: 0.3399 - accuracy: 0.8438 - val_loss: 0.3393 - val_accuracy: 0.8350
Epoch 189/200
13/13 [==============================] - 0s 5ms/step - loss: 0.3399 - accuracy: 0.8562 - val_loss: 0.3393 - val_accuracy: 0.8850
Epoch 190/200
13/13 [==============================] - 0s 4ms/step - loss: 0.3399 - accuracy: 0.8525 - val_loss: 0.3393 - val_accuracy: 0.9100
Epoch 191/200
13/13 [==============================] - 0s 5ms/step - loss: 0.3399 - accuracy: 0.8662 - val_loss: 0.3393 - val_accuracy: 0.8750
Epoch 192/200
13/13 [==============================] - 0s 5ms/step - loss: 0.3398 - accuracy: 0.8675 - val_loss: 0.3393 - val_accuracy: 0.8800
Epoch 193/200
13/13 [==============================] - 0s 6ms/step - loss: 0.3399 - accuracy: 0.8587 - val_loss: 0.3393 - val_accuracy: 0.8300
Epoch 194/200
13/13 [==============================] - 0s 6ms/step - loss: 0.3399 - accuracy: 0.8662 - val_loss: 0.3393 - val_accuracy: 0.8600
Epoch 195/200
13/13 [==============================] - 0s 6ms/step - loss: 0.3399 - accuracy: 0.8687 - val_loss: 0.3393 - val_accuracy: 0.8950
Epoch 196/200
13/13 [==============================] - 0s 6ms/step - loss: 0.3399 - accuracy: 0.8550 - val_loss: 0.3393 - val_accuracy: 0.9000
Epoch 197/200
13/13 [==============================] - 0s 4ms/step - loss: 0.3399 - accuracy: 0.8700 - val_loss: 0.3393 - val_accuracy: 0.8900
Epoch 198/200
13/13 [==============================] - 0s 6ms/step - loss: 0.3399 - accuracy: 0.8587 - val_loss: 0.3393 - val_accuracy: 0.8550
Epoch 199/200
13/13 [==============================] - 0s 6ms/step - loss: 0.3399 - accuracy: 0.8500 - val_loss: 0.3393 - val_accuracy: 0.8550
Epoch 200/200
13/13 [==============================] - 0s 5ms/step - loss: 0.3398 - accuracy: 0.8675 - val_loss: 0.3393 - val_accuracy: 0.8650
<keras.callbacks.History at 0x7f0402327ed0>
```

We can see that the accuracy achieved by the auto encoder model is 86.50%.

Logistic Regression

Below we test the Logistic Regression Model and we see that the model gets and F1 score of 0.89 and an accuracy score of 89.29%, which is higher than the Auto Encoder model.

```
[ ] y_pred_lr = lr.predict(x_test)

print ("Classification Report: ")
print (classification_report(y_test, y_pred_lr))

Classification Report:
              precision    recall  f1-score   support

         0.0       0.87      0.98      0.92       192
         1.0       0.96      0.73      0.83       107

    accuracy                           0.89       299
   macro avg       0.91      0.86      0.88       299
weighted avg       0.90      0.89      0.89       299

[ ] AC_LR = accuracy_score(y_test, y_pred_lr)
print ("Accuracy Score: ", AC_LR)

Accuracy Score:  0.8929765886287625
```

Random Forest Classifier

Now we will test our Random Forest Classifier Model. We can see that the model achieves an F1 score of 0.95 and an accuracy of 94.98%, which is better than our Logistic Regression Model and our Auto Encoder.

```
[ ]  y_pred_rf = rf.predict(x_test)

[ ]  print ("Classification Report: ")
     print (classification_report(y_test, y_pred_rf))

     Classification Report:
                   precision    recall  f1-score   support

              0.0       0.95      0.97      0.96       192
              1.0       0.95      0.91      0.93       107

         accuracy                           0.95       299
        macro avg       0.95      0.94      0.94       299
     weighted avg       0.95      0.95      0.95       299


[ ]  AC_RFC = accuracy_score(y_test, y_pred_rf)
     print ("Accuracy Score: ", AC_RFC)

     Accuracy Score:  0.9498327759197325
```

K Nearest Neighbours

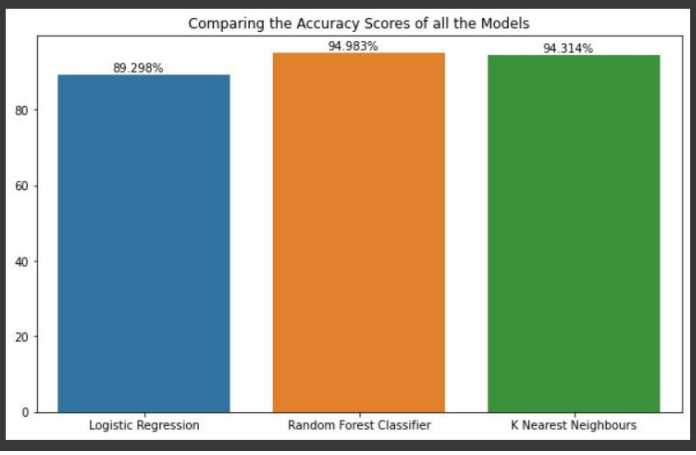
The final model we will test is the K Nearest Neighbour Model. As we can see we get an F1 score of 0.94 and an accuracy of 94.31%, which is lower than the Random Forest Model.

```
[ ]  y_pred_knn=knn.predict(x_test)

[ ]  print ("Classification Report: ")
     print (classification_report(y_test, y_pred_knn))

     Classification Report:
                   precision    recall  f1-score   support

              0.0       0.93      0.98      0.96       192
              1.0       0.97      0.87      0.92       107

         accuracy                           0.94       299
        macro avg       0.95      0.93      0.94       299
     weighted avg       0.94      0.94      0.94       299


     AC_KNN = accuracy_score(y_test,y_pred_knn)
     print ("Accuracy Score: ", AC_KNN)

     Accuracy Score:  0.9431438127090301
```

5. COMPARING AND MEASURING

In this final section we will compare the accuracies of all the models using seaborn to make a bar chart with the accuracies of all the models.

```
plt.figure(figsize=(10,6))
scores=[AC_LR,AC_RFC,AC_KNN]
label=['Logistic Regression','Random Forest Classifier','K Nearest Neighbours']
for i in range(0,len(scores)):
    scores[i]=np.round(scores[i]*100,decimals=3)
ax=sns.barplot(x=label,y=scores)
plt.title('Comparing the Accuracy Scores of all the Models')
for p in ax.patches:
    width, height = p.get_width(), p.get_height()
    x, y = p.get_xy()
    ax.annotate('{:.3f}%'.format(height), (x +0.25, y + height + 0.8))
plt.show()
```



From this chart we can conclude that the Random Forest Model did the best with an accuracy of 94.983%.

PYTHON CODE

Original file is located at

https://colab.research.google.com/drive/1W1ESrNB3gcI_phv0oPYuKesuvaJST6UR

Credit Card Fraud

Importing and Understanding the dataset

"""

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

```
import seaborn as sns
import keras
from keras.layers import Dense,Input
from sklearn.manifold import TSNE
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras import regularizers
from keras.models import Model, Sequential
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, accuracy_score

d = pd.read_csv('creditcard.csv')

d.head()

zero = d['Class'].isin(['0']).sum()
zero

one = d['Class'].isin(['1']).sum()
one

d.info()

d.shape

d.describe()

d.isna().sum()
```

```
d.dropna(inplace=True)

d.isna().sum()

"""### Visualization"""

l=["Not Fraud","Fraud"]
a=[zero,one]
plt.pie(x=a,labels=l,autopct="%1.1f%%")
plt.legend()
plt.show()

"""##### Due to imbalance in data we will be taking only 1000 non fraud cases"""

fraud = d[d['Class']==1]
non_fraud = d[d['Class']==0].sample(1000)

df = non_fraud.append(fraud).sample(frac=1).reset_index(drop=True)

x = df.drop(['Class'], axis = 1).values
y = df['Class'].values

x.shape

print(y)

tsne = TSNE(n_components=2, random_state=0)
x1 = tsne.fit_transform(x)

plt.figure(figsize=(12,8))
plt.scatter(x1[np.where(y==0),0],x1[np.where(y==0),1],color='green',alpha=0.8, label="Not a Fraud")
```

```
plt.scatter(x1[np.where(y==1),0],x1[np.where(y==1),1],color='red',alpha=0.8, label="A Fraud")
plt.legend()
plt.show()

"""### Auto Encoder"""

mms=MinMaxScaler()
x_mms = mms.fit_transform(x)
x_nf = x_mms[y==0]
x_f = x_mms[y==1]

i = Input(shape=(x.shape[1],))

h1 = Dense(units=128,activation='tanh',activity_regularizer=regularizers.l1(10e-5))(i)
code = Dense(units=64,activation='sigmoid')(h1)

h2 = Dense(units=64,activation='tanh')(code)
h3 = Dense(units=128,activation='tanh')(h2)

o = Dense(x.shape[1],activation="softmax")(h3)

autoencoder = Model(i,o)

autoencoder.compile(optimizer="adam",loss="mse",metrics=['accuracy'])

autoencoder.fit(x_nf[0:2000], x_nf[0:2000], batch_size = 64, epochs = 200, shuffle = True, validation_split =
0.20)

hr = Sequential()

hr.add(autoencoder.layers[0])
```

```
hr.add(autoencoder.layers[1])

hr.add(autoencoder.layers[2])


normal = hr.predict(x_nf[:6000])

fraud = hr.predict(x_f)


rep_x = np.append(normal, fraud, axis = 0)

y_n = np.zeros(normal.shape[0])

y_f = np.ones(fraud.shape[0])

rep_y = np.append(y_n, y_f)


tsne = TSNE(n_components=2, random_state=0)

normal1 = tsne.fit_transform(rep_x)


plt.figure(figsize=(12,8))

plt.scatter(normal1[np.where(rep_y==0),0],normal1[np.where(rep_y==0),1],color='green',alpha=0.8, label="Not a Fraud")

plt.scatter(normal1[np.where(rep_y==1),0],normal1[np.where(rep_y==1),1],color='red',alpha=0.8, label="A Fraud")

plt.legend()

plt.show()


temp_x = d.drop(['Class'], axis = 1).sample(10000)

temp_y = d['Class'].sample(10000)


x_test_mms = mms.fit_transform(temp_x)


# temp_x = x_test_mms.drop(['Class'], axis = 1)

# temp_y = x_test_mms['Class']


x_train, x_test, y_train, y_test = train_test_split(temp_x, temp_y, test_size=0.20)

AC_AE = autoencoder.evaluate(x_test,y_test)
```

```
# y_pred_AE = autoencoder.predict(x_test)
# AC_AE = accuracy_score(rep_x,rep_y)
# print("test Loss", AC_AE[0])
print ("Accuracy Score: ", AC_AE)

"""### Logistic Regression"""

x_train, x_test, y_train, y_test = train_test_split(rep_x, rep_y, test_size=0.20)

lr = LogisticRegression()
lr.fit(x_train,y_train)

y_pred_lr = lr.predict(x_test)

print ("Classification Report: ")
print (classification_report(y_test, y_pred_lr))

AC_LR = accuracy_score(y_test, y_pred_lr)
print ("Accuracy Score: ", AC_LR)

"""### Random Forest"""

rf=RandomForestClassifier(n_estimators=100,random_state=0)
rf.fit(x_train,y_train)

y_pred_rf = rf.predict(x_test)

print ("Classification Report: ")
print (classification_report(y_test, y_pred_rf))

AC_RFC = accuracy_score(y_test, y_pred_rf)
```

```
print ("Accuracy Score: ", AC_RFC)

"""### K Nearest Neighbours"""

knn = KNeighborsClassifier()
knn.fit(x_train,y_train)

y_pred_knn=knn.predict(x_test)

print ("Classification Report: ")
print (classification_report(y_test, y_pred_knn))

AC_KNN = accuracy_score(y_test,y_pred_knn)
print ("Accuracy Score: ", AC_KNN)

"""### Comparing Scores"""

plt.figure(figsize=(10,6))
scores=[AC_LR,AC_RFC,AC_KNN]
label=['Logistic Regression','Random Forest Classifier','K Nearest Neighbours']
for i in range(0,len(scores)):
    scores[i]=np.round(scores[i]*100,decimals=3)
ax=sns.barplot(x=label,y=scores)
plt.title('Comparing the Accuracy Scores of all the Models')
for p in ax.patches:
    width, height = p.get_width(), p.get_height()
    x, y = p.get_xy()
    ax.annotate('{:.3f}%'.format(height), (x +0.25, y + height + 0.8))
plt.show()
```

CONCLUSION

This report uses one datasets consisting of Credit Card Frauds from the month September, 2013. The goal of my project is to determine if a fraud has occurred or not based on 30 parameters.

Before starting the predictions, we talk about the different packages we will need and pre-processing the data to make sure no missing values are present as a preparation phase.

Then we visualise the data to get a better understanding of the data using pie charts and TSNE. From this phase we get to know that the dataset is highly imbalanced so we sample out 1000 random non fraud cases to make the dataset more balanced.

Now we train our 4 different models and test them on a validation dataset which is 20% percent of the dataset. A brief description on all the models is provided before using them.

Now we test the models and compare them using a bar plot and the accuracies they have achieved.

The best model among the models used was Random Forest Classifier with and accuracy of 94.983%.

We then end the report with the python code.

As we can see this is how researchers are trying to use Machine Learning Models and Deep Learning Models to combat the exponentially growing issue of Credit Card Frauds around the world.