MOHAMMED YOUNUS

Credit Card Fraud Detection using Machine Learning Model.

Problem Statement - For many banks, retaining high profitable customers is the number one business goal. Banking fraud, however, poses a significant threat to this goal for different banks. In terms of substantial financial losses, trust and credibility, this is a concerning issue to both banks and customers alike.

In the banking industry, credit card fraud detection using machine learning is not only a trend but a necessity for them to put proactive monitoring and fraud prevention mechanisms in place. Machine learning is helping these institutions to reduce time consuming manual reviews, costly chargebacks and fees as well as denials of legitimate transactions.

The problem statement chosen for this project is to predict fraudulent credit card transactions with the help of machine learning models.

In this project, we will analyze customer-level data that has been collected and analyzed during a research collaboration of Worldline and the Machine Learning Group.

- Steps to be taken in the Project is sub-divided into the following sections. These are:
  - Load the necessary libraries such as Numpy , Pandas , sklearn.model etc.
  - Loading the dataset as csv file and showing first ten rows.
  - Drop the unnecessary columns from the data.
  - Calculate statistical values and round them up to 3 decimal places.
  - Checking for null values and return their sum of numbers of true values in each column.
  - Handle the null by mean of all values fill into them.
  - Extracting all information about data.
  - Checking shape of data.
  - Visualization on different species of credit cards transaction information using python data visualization.
  - Data preprocessing or (Data cleaning) performed by the one hot encoding in this process we change categorical data into numerical data and the technique is called feature Engineering.
  - Splitting the cleaned data into dependent and independent variables.
  - Splitting the data into train and test sets with train_test_split using sklearn library.

- Import different kind of Classification Models and Train that model with the help of .fit().
- Predicting the trained models and then checking their accuracy score and confusion metrics of the model using confusion metrics & accuracy score.
- Then recall the train_test_split and split the data into training and testing set with different models.
- Then predicting the trained models and checking the accuracy of model and check the accuracy difference.
- And finally predict whether the classification (or detection) of Credit cards is generated or not.

☐ Step-1 – Import necessary libraries.

Loading Necessary Libraries.

```
[2] import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as snr
    from sklearn.preprocessing import OneHotEncoder
```

☐ Step-2 – Loading the dataset as csv file and showing first ten rows.

Load data and show first 10 rows of data.

```
data=pd.read_csv("/content/creditcard.csv")
data.head(10)
```

V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0
109	-0.168252	0.420987	-0.029728	0.476201	0.260314	-0.568671	...	-0.208254	-0.559825	-0.026398	-0.371427	-0.232794	0.105915	0.253844	0.081080	3.67	0
371	1.202613	0.191881	0.272708	-0.005159	0.081213	0.464960	...	-0.167716	-0.270710	-0.154104	-0.780055	0.750137	-0.257237	0.034507	0.005168	4.99	0
380	-0.492199	0.948934	0.428118	1.120631	-3.807864	0.615375	...	1.943465	-1.015455	0.057504	-0.649709	-0.415267	-0.051634	-1.206921	-1.085339	40.80	0
192	-0.271526	2.669599	3.721818	0.370145	0.851084	-0.392048	...	-0.073425	-0.268092	-0.204233	1.011592	0.373205	-0.384157	0.011747	0.142404	93.20	0
367	-0.222187	0.499361	-0.246761	0.651583	0.069539	-0.736727	...	-0.246914	-0.633753	-0.120794	-0.385050	-0.069733	0.094199	0.246219	0.083076	3.68	0

- <u>Step-3</u> – Calculate statistical values and round them up to 3 decimal places.

Calculate statistical values and round them up to 3 decimal places.

```
[13] data.describe().round(3)
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...
count	284807.000	284807.000	284807.000	284807.000	284807.000	284807.000	284807.000	284807.000	284807.000	284807.000	...
mean	94813.860	0.000	0.000	-0.000	0.000	0.000	0.000	-0.000	0.000	-0.000	...
std	47488.146	1.959	1.651	1.516	1.416	1.380	1.332	1.237	1.194	1.099	...
min	0.000	-56.408	-72.716	-48.326	-5.683	-113.743	-26.161	-43.557	-73.217	-13.434	...
25%	54201.500	-0.920	-0.599	-0.890	-0.849	-0.692	-0.768	-0.554	-0.209	-0.643	...
50%	84692.000	0.018	0.065	0.180	-0.020	-0.054	-0.274	0.040	0.022	-0.051	...
75%	139320.500	1.316	0.804	1.027	0.743	0.612	0.399	0.570	0.327	0.597	...
max	172792.000	2.455	22.058	9.383	16.875	34.802	73.302	120.589	20.007	15.595	...

	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
count	284807.000	284807.000	284807.000	284807.000	284807.000	284807.000	284807.000	284807.000	284807.000	284807.000
mean	0.000	-0.000	0.000	0.000	0.000	0.000	-0.000	-0.000	88.350	0.002
std	0.735	0.726	0.624	0.606	0.521	0.482	0.404	0.330	250.120	0.042
min	-34.830	-10.933	-44.808	-2.837	-10.295	-2.605	-22.566	-15.430	0.000	0.000
25%	-0.228	-0.542	-0.162	-0.355	-0.317	-0.327	-0.071	-0.053	5.600	0.000
50%	-0.029	0.007	-0.011	0.041	0.017	-0.052	0.001	0.011	22.000	0.000
75%	0.186	0.529	0.148	0.440	0.351	0.241	0.091	0.078	77.165	0.000
max	27.203	10.503	22.528	4.585	7.520	3.517	31.612	33.848	25691.160	1.000

8 rows × 31 columns

- <u>Step-4</u> – Checking for null values and return their sum of numbers of true values in each column.

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

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

0s completed at 11:38 AM

□

Step-5 – Extracting all information about data.

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

□ Step-6 – Checking shape of data.

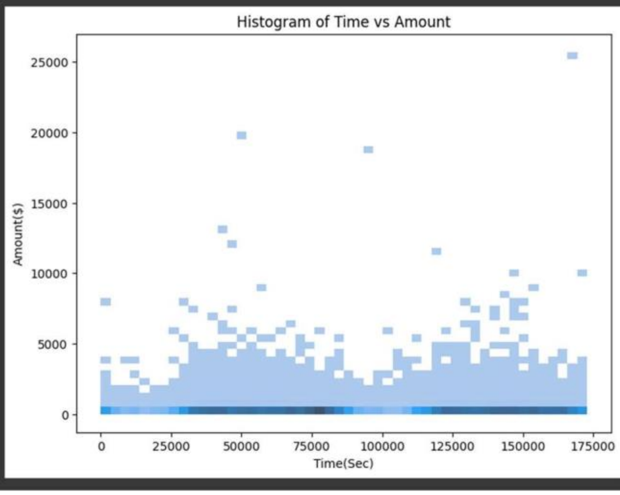
Checking the shape of data.

```
data.shape
```

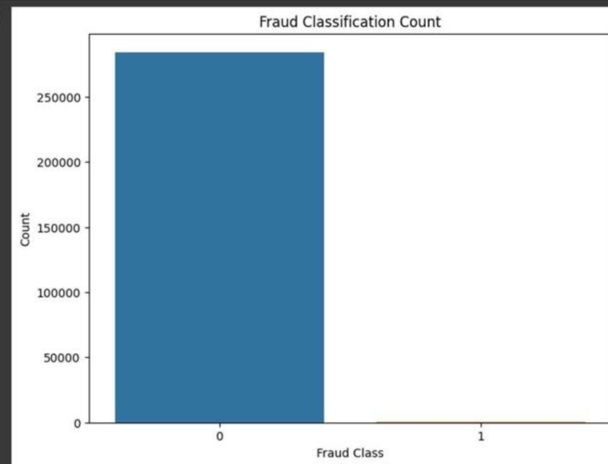
```
(284807, 31)
```

□ Step-7 – Visualization on different species of credit cards transaction information using python data visualization.

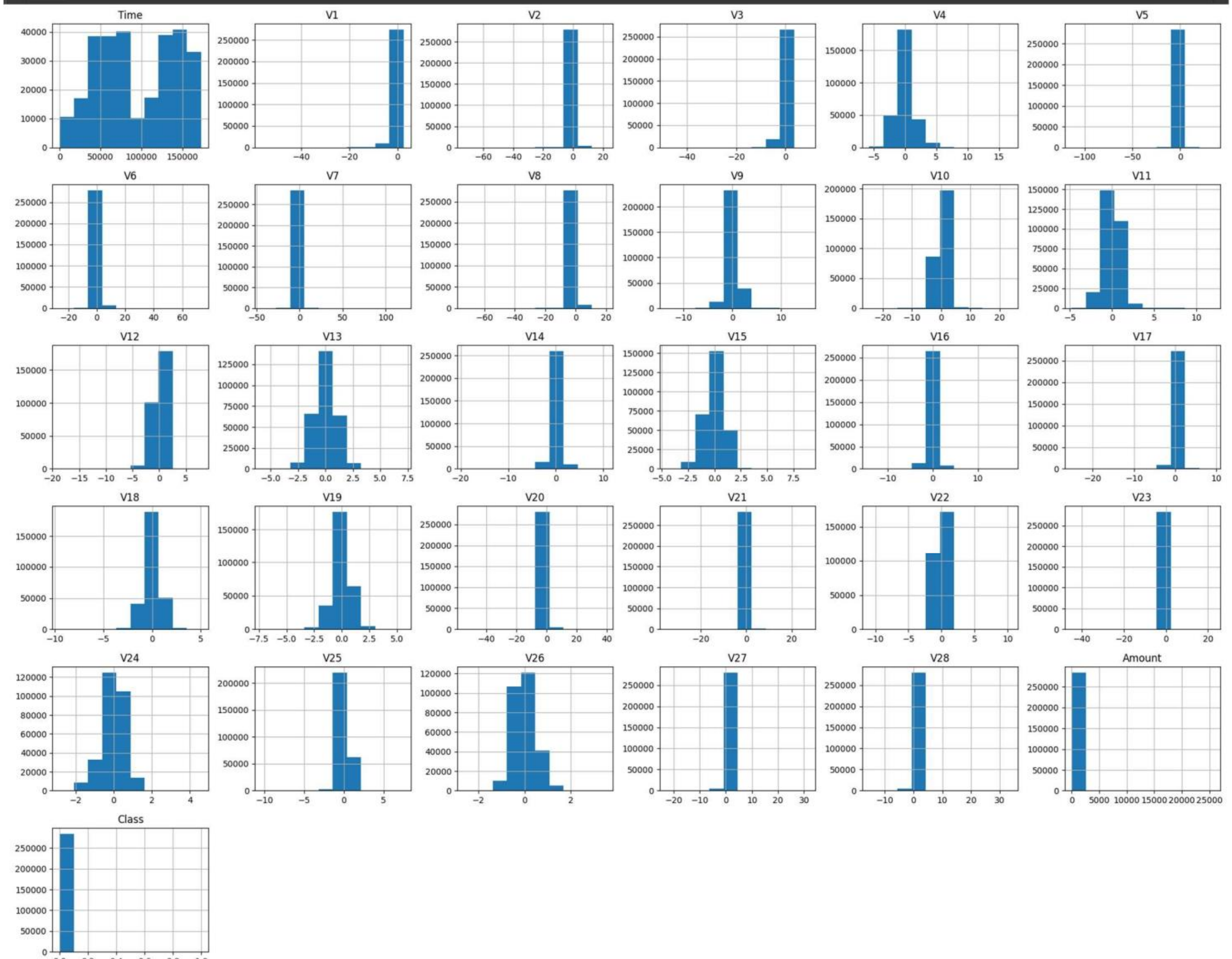
```python
plt.figure(figsize=(8,6))
snr.histplot(data=data, x='Time', y='Amount', bins=50)
plt.title("Histogram of Time vs Amount")
plt.xlabel("Time(Sec)")
plt.ylabel("Amount($)")
plt.show()
```




```python
fig = plt.figure(figsize=(8,6))
snr.countplot(x='Class',data=data)
plt.xlabel('Fraud Class')
plt.ylabel("Count")
plt.title("Fraud Classification Count")
plt.show()
```



```
data.hist(figsize=(25,20))
plt.show()
```



Step-8 – Splitting the data into dependent and independent variables.

Splitting the data into Independent and dependent set.

```
[16] x=data.drop(['Class'],axis=1)
     x.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	V22	V23	V24	V25	V26	V27	V28	Amount
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	0.251412	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.069083	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.524980	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.208038	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	0.408542	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99

5 rows × 30 columns

```
y=data['Class']
y.head()
```

```
0    0
1    0
2    0
3    0
4    0
Name: Class, dtype: int64
```

- Step-9 – Splitting the data into training and testing sets.

Devide the cleaned data into training and testing testing sets.

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.8)
```

- Step-10 – Import first machine learning model K-Nearest neighbor taking n_neighbor=5.

Import first Machine Learning Model 'K-Nearest Neighbor'.

```
[19] from sklearn.neighbors import KNeighborsClassifier
     neighbor=KNeighborsClassifier(n_neighbors=5)
```

- Step-11 – Train the model using .fit() function.

Train the model

```
[20] neighbor.fit(x_train,y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

- Step-12 – Making predictions on model.

-

Make predictions on model.

```
[ ] predictions=neighbor.predict(x_test)
    print(predictions)

    [0 0 0 ... 0 0 0]
```

- Step-13 – Checking confusion metrics and accuracy score of model.

Check confusion metrics and check accuracy score.

```
[22] from sklearn.metrics import confusion_matrix, accuracy_score
     cm=confusion_matrix(y_test, predictions)
     ac=accuracy_score(y_test, predictions)
     print(cm)

     [[56869     1]
      [   85     7]]
```

```
[23] print(ac)

     0.9984902215512096
```

- Step-14 – Import the Second Machine Learning Model ‘Logistic Regression’ and train model and then make prediction.

Creating second machine learning model 'Logistic Regression'.

```
[24] from sklearn.linear_model import LogisticRegression
     log=LogisticRegression()
```

Train the model.

```
[25] log.fit(x_train, y_train)

     /usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
     STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

     Increase the number of iterations (max_iter) or scale the data as shown in:
         https://scikit-learn.org/stable/modules/preprocessing.html
     Please also refer to the documentation for alternative solver options:
         https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
       n_iter_i = _check_optimize_result(
     ▾ LogisticRegression
     LogisticRegression()
```

Make Predictions on model.

```
[26] log_predictions=log.predict(x_test)
     print(log_predictions)

     [0 0 0 ... 0 0 0]
```

Step-15 – Print a confusion metrics and check accuracy score for Logistic Regression Model.

Check confusion metrics and accuracy score.

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm=confusion_matrix(y_test, log_predictions)
ac=accuracy_score(y_test, log_predictions)
print(cm)
```

```
[[56851    19]
 [   37    55]]
```

```
[28] print(ac)
```

```
0.9990168884519505
```

☐ Step-16 – Import the Third Machine Learning Model Decision Tree and train model and then make prediction.

Creating third machine machine learning model 'Decision Tree'.

```
[29] from sklearn.tree import DecisionTreeClassifier
     tree=DecisionTreeClassifier()
```

Train the model.

```
[30] tree.fit(x_train, y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

Make predictions on model.

```
tree_predictions=tree.predict(x_test)
print(tree_predictions)
```

```
[0 0 0 ... 0 0 0]
```

☐ Step-17 – Print a confusion metrics and check accuracy score for Support Vector Machine Model.

Check confusion metrics and accuracy score.

```
[32] from sklearn.metrics import confusion_matrix, accuracy_score
     cm=confusion_matrix(y_test, tree_predictions)
     ac=accuracy_score(y_test, tree_predictions)
     print(cm)
```

```
[[56845    25]
 [   20    72]]
```

```
[33] print(ac)
```

```
0.9992099996488887
```

Conclusion – Here, we have to focus on a high recall in order to detect actual fraudulent transactions in order to save the banks from high-value fraudulent transactions,

After performing several models, we have seen that on performing Machine Learning models I got 99% of accuracy of model by using K-Nearest Neighbor algorithm of Machine Learning, I got 99% of accuracy of model by using Logistic Regression algorithm of Machine Learning and I got 99% of accuracy of model by using Decision Tree algorithm of machine Learning. Overall I got 99% of accuracy for the complete Machine Learning Model of Credit Card Fraud Detection.