Importing Libraries

```
In [ ]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
```

```
In [ ]: df = pd.read_csv('/content/creditcard.csv', error_bad_lines=False)
```

```
<ipython-input-2-0451fd357e02>:1: FutureWarning: The error_bad_lines argument h
as been deprecated and will be removed in a future version. Use on_bad_lines in
the future.


  df = pd.read_csv('/content/creditcard.csv', error_bad_lines=False)
<ipython-input-2-0451fd357e02>:1: DtypeWarning: Columns (5) have mixed types. S
pecify dtype option on import or set low_memory=False.
  df = pd.read_csv('/content/creditcard.csv', error_bad_lines=False)
```

```
In [ ]: # Dimensions of the Data

        df.shape
```

```
Out[3]: (284807, 31)
```

```
In [ ]: df.head()
```

Out[4]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	
0	0.0	-1.359807	-0.978206	2.536347	1.378155	-0.33832077	0.462388	0.239599	0.098698	0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060017649	-0.082361	-0.078803	0.085102	-0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198133	1.800499	0.791461	0.247676	-1
3	1.0	-0.966272	3.712444	1.792993	-0.863291	-0.01030888	1.247203	0.237609	0.377436	-1
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193377	0.095921	0.592941	-0.270533	0

5 rows × 31 columns

In [ ]:

```
# Information about the whole Dataset

df.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  object 
 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(29), int64(1), object(1)
memory usage: 67.4+ MB
```

In [ ]:

```
# Converting all columns into same datatype 'float64'

for col in df.columns[:-1]:
  if df[col].dtypes != 'float64':
    df[col] = df[col].astype('float64')
```

In [ ]:

```
# Checking the DataTypes of whole Dataset

df.dtypes
```

Out[9]:

```
Time      float64
V1        float64
V2        float64
V3        float64
V4        float64
V5        float64
V6        float64
V7        float64
V8        float64
V9        float64
V10       float64
V11       float64
V12       float64
V13       float64
V14       float64
V15       float64
V16       float64
V17       float64
V18       float64
V19       float64
V20       float64
V21       float64
V22       float64
V23       float64
V24       float64
V25       float64
V26       float64
V27       float64
V28       float64
Amount    float64
Class       int64
dtype: object
```

Statistical Analysis

In [ ]:
```
df.describe()
```

Out[10]:

	Time	V1	V2	V3	V4	V5	
count	284807.000000	2.848070e+05	284807.000000	2.848070e+05	2.848070e+05	284807.000000	
mean	94813.859575	1.759061e-12	0.000011	-9.654937e-13	8.321385e-13	-0.000007	
std	47488.145955	1.958696e+00	1.651324	1.516255e+00	1.415869e+00	1.380246	
min	0.000000	-5.640751e+01	-72.715728	-4.832559e+01	-5.683171e+00	-113.743307	-
25%	54201.500000	-9.203734e-01	-0.598559	-8.903648e-01	-8.486401e-01	-0.691603	
50%	84692.000000	1.810880e-02	0.065507	1.798463e-01	-1.984653e-02	-0.054337	
75%	139320.500000	1.315642e+00	0.803734	1.027196e+00	7.433413e-01	0.611914	
max	172792.000000	2.454930e+00	22.057729	9.382558e+00	1.687534e+01	34.801666	

8 rows × 31 columns

Checking Missing Values

In [ ]:
```
missing_values = []
for col in df.columns:
  missing_values.append(df[col].isna().sum())
```

In [ ]:
```
Col = df.columns
```

In [ ]:
```
Col = pd.DataFrame(Col)
missing_values= pd.DataFrame(missing_values)
```

In [ ]:
```
result_missing = pd.concat([Col, missing_values], axis = 1)
result_missing.columns = ['Columns', 'Missing_values']
```

In [ ]:

```
result_missing
```

Out[15]:

	Columns	Missing_values
0	Time	0
1	V1	0
2	V2	0
3	V3	0
4	V4	0
5	V5	0
6	V6	0
7	V7	0
8	V8	0
9	V9	0
10	V10	0
11	V11	0
12	V12	0
13	V13	0
14	V14	0
15	V15	0
16	V16	0
17	V17	0
18	V18	0
19	V19	0
20	V20	0
21	V21	0
22	V22	0
23	V23	0
24	V24	0
25	V25	0
26	V26	0
27	V27	0
28	V28	0
29	Amount	0
30	Class	0

Data Cleaning

In [ ]:

```
for i, value in enumerate(df['V2']):
    if isinstance(value, str):
        # Find the second decimal point in the value
        second_dot_index = value.find('.', value.find('.') + 1)

        if second_dot_index != -1:
            # Remove the second decimal point
            df.at[i, 'V2'] = value[:second_dot_index] + value[second_dot_index +
```

In [ ]:

```
character_to_replace = "'"

# Iterate through all columns except the last column
for column in df.columns[:-1]:
    df[column] = df[column].apply(lambda x: x.replace(character_to_replace, '') i
```

In [ ]:

```
character_to_replace = "."

# Iterate through all columns except the last column
for column in df.columns[:-1]:
    df[column] = df[column].apply(lambda x: x.replace(character_to_replace, '') i
```

In [ ]:

```
# Compute Missing Values

from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values = np.nan, strategy = 'mean')
df.loc[:,df.columns[:-1]] = imputer.fit_transform(df.loc[:,df.columns[:-1]])
```

In [ ]:

```
df['Class'].value_counts()
```

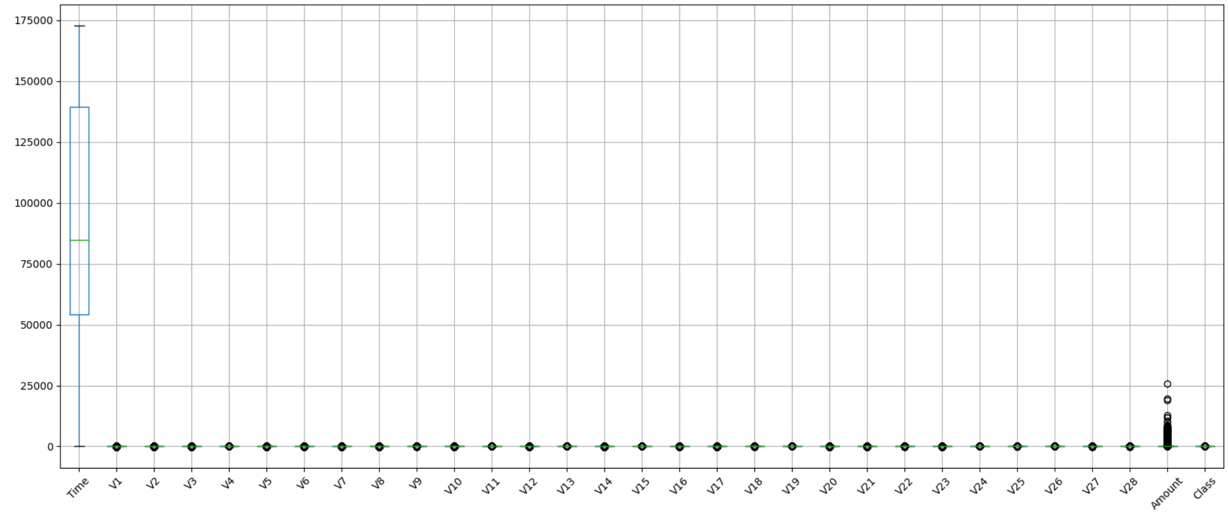
Out[17]:

```
0    284315
1       492
Name: Class, dtype: int64
```

BoxPlot

```
In [ ]: plt.figure(figsize = (20, 8))
        df.boxplot(rot = 45)
```

```
Out[18]: <Axes: >
```



Checking Outliers of each column

```
In [ ]: lst = []
        for col in df.columns[:-1]:
          percentile25 = df[col].quantile(0.25)
          percentile75 = df[col].quantile(0.75)
          iqr = percentile75 - percentile25

          upper_bound = percentile75 + 1.5 * iqr
          lower_bound = percentile25 - 1.5 * iqr

          lst.append(sum((df[col] > upper_bound) | (df[col] < lower_bound)))
```

```
In [ ]: col = df.columns[:-1]
        col = pd.DataFrame(col)
```

```
In [ ]: lst = pd.DataFrame(lst)
In [ ]: pd.concat([col, lst], axis = 1)
```

```
Out[22]:
              0      0
     0     Time      0
     1       V1   7062
     2       V2  13526
     3       V3   3363
     4       V4  11148
     5       V5  12295
     6       V6  22965
     7       V7   8948
     8       V8  24134
     9       V9   8283
    10      V10   9496
    11      V11    780
    12      V12  15348
    13      V13   3368
    14      V14  14149
    15      V15   2894
    16      V16   8184
    17      V17   7420
    18      V18   7533
    19      V19  10205
    20      V20  27770
    21      V21  14497
    22      V22   1317
    23      V23  18541
    24      V24   4774
    25      V25   5367
    26      V26   5596
    27      V27  39163
    28      V28  30342
    29   Amount  31904
```

Anomaly Detection Algorithm

In [ ]:

```
# Data Splitting

x = df.iloc[:, 1:-1].values
y = df.iloc[:, -1].values
```

In [ ]:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, rando
```

Model 1

Random Forest

In [ ]:

```
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier(random_state = 42)
rf_model.fit(x_train, y_train)
```

Out[25]:

```
RandomForestClassifier(random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [ ]:

```
rf_prediction = rf_model.predict(x_test)
```

In [ ]:

```
rf_prediction[:10]
```

Out[27]:

```
array([1, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

In [ ]:

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, confus
```

Confusion Matrix

In [ ]:

```
confusion_matrix(y_test, rf_prediction)
```

Out[29]:

```
array([[56862,     2],
       [   23,    75]])
```

Accuracy Score

In [ ]:

```
accuracy_score(y_test, rf_prediction)
```

Out[30]:

```
0.9995611109160493
```

Precision Score

In [ ]:

```
precision_score(y_test, rf_prediction)
```

Out[31]:

```
0.974025974025974
```

Recall Score

In [ ]:

```
recall_score(y_test, rf_prediction)
```

Out[32]:

```
0.7653061224489796
```

Model 2

Isolation Forest

In [ ]:

```
# Model Implementation

from sklearn.ensemble import IsolationForest
model_test = IsolationForest(contamination = 'auto')
model_train = IsolationForest(contamination = 'auto')
model_train.fit(x_train)
model_test.fit(x_test)
```

Out[35]:

```
IsolationForest()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [ ]:

```
x_train_prediction = model_train.predict(x_train)
x_test_prediction = model_test.predict(x_test)
```

In [ ]:

```
pd.DataFrame(x_test_prediction).value_counts()
```

Out[37]:

```
 1    82420
-1     3023
dtype: int64
```

In [ ]:

```
pd.DataFrame(x_train_prediction).value_counts()
```

Out[38]:

```
 1    193086
-1      6278
dtype: int64
```

TruePositive, FalsePositive, TrueNegative, FalseNegative

In [ ]:

```
# For Training Data

tp_train = sum((x_train_prediction == -1) & (y_train == 1))
fp_train = sum((x_train_prediction == -1) & (y_train == 0))
fn_train = sum((x_train_prediction == 1) & (y_train == 1))
tn_train = sum((x_train_prediction == 1) & (y_train == 0))
```

In [ ]:

```
# For 2nd Iteration

tp_test = sum((x_test_prediction == -1) & (y_test == 1))
fp_test = sum((x_test_prediction == -1) & (y_test == 0))
fn_test = sum((x_test_prediction == 1) & (y_test == 1))
tn_test = sum((x_test_prediction == 1) & (y_test == 0))
```

Precision Score

In [ ]:

```
precision_train = tp_train / (tp_train + fp_train)
precision_test = tp_test / (tp_test + fp_test)
print('Precision on Training Data is {}'.format(precision_train))
print('Precision on Testing Data is {}'.format(precision_test))
```

```
Precision on Training Data is 0.04635234151003504
Precision on Testing Data is 0.03572609990076083
```

Recall Score (Sensitivity)

In [ ]:

```
recall_train = tp_train / (tp_train + fn_train)
recall_test = tp_test / (tp_test + fn_test)
print('Sensitivity on Training Data is {}'.format(recall_train))
print('Sensitivity on Testing Data is {}'.format(recall_test))
```

```
Sensitivity on Training Data is 0.8174157303370787
Sensitivity on Testing Data is 0.7941176470588235
```

Accuracy Score

In [ ]:

```
accuracy_score_train = (tp_train+ tn_train)/ (tp_train+tn_train+fp_train+fn_trai
accuracy_score_test = (tp_test+ tn_test)/ (tp_test+tn_test+fp_test+fn_test)
```

In [ ]:

```
print('Accuracy Score on Training Data is {}'.format(accuracy_score_train))
print('Accuracy on Testing Data is {}'.format(accuracy_score_test))
```

```
Accuracy Score on Training Data is 0.9696434662225878
Sensitivity on Testing Data is 0.9655559846915488
```

Model 3

OneClassSVM

In [41]:

```
xx_train, xx_test, yy_train, yy_test = train_test_split(x_test, y_test, test_siz
```

In [42]:

```
from sklearn.svm import OneClassSVM
svm_model = OneClassSVM(gamma = 'auto')
svm_prediction = svm_model.fit_predict(xx_train)
```

TruePositive, FalsePositive, TrueNegative, FalseNegative

In [44]:

```
# For Training Data

tp_train = sum((svm_prediction == -1) & (yy_train == 1))
fp_train = sum((svm_prediction == -1) & (yy_train == 0))
fn_train = sum((svm_prediction == 1) & (yy_train == 1))
tn_train = sum((svm_prediction == 1) & (yy_train == 0))
```

Precision Score

In [45]:

```
precision_train = tp_train / (tp_train + fp_train)
print('Precision on Training Data is {}'.format(precision_train))
```

```
Precision on Training Data is 0.0037232174218475586
```

Recall Score

In [46]:

```
recall_train = tp_train / (tp_train + fn_train)
print('Sensitivity on Training Data is {}'.format(recall_train))
```

```
Sensitivity on Training Data is 0.9636363636363636
```

Accuracy Score

In [47]:

```
accuracy_score_train = (tp_train+ tn_train)/ (tp_train+tn_train+fp_train+fn_trai
print('Accuracy Score on Training Data is {}'.format(accuracy_score_train))
```

```
Accuracy Score on Training Data is 0.5019837786594572
```

Accuracy Score of OneClassSVM is slightly lower than other algorithms. **Reason:** Training data is too small, I have split it into shorter size(1/10) of the original data because SVM is taking too much time to train.