CREDIT CARD FRAUD DETECTION

Vandana Yalla

Uploading & Exploring the Data

```
import pandas as pd
```

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

```
data.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267

5 rows × 31 columns

```
pd.options.display.max_columns = None
```

```
data.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	-0.551600	-0.617801	-0.991390	-0.
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.612727	1.065235	0.489095	-0.
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.624501	0.066084	0.717293	-0.
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	-0.226487	0.178228	0.507757	-0.
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	-0.822843	0.538196	1.345852	-1.

```
data.tail()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	4.356170	-1.593105	2.711941	-0.6
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	-0.975926	-0.150189	0.915802	1.2
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	-0.484782	0.411614	0.063119	-0.1
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	-0.399126	-1.933849	-0.962886	-1.0
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	-0.915427	-1.040458	-0.031513	-0.1

```
data.shape
```

(284807, 31)

```
print("Number of columns: {}".format(data.shape[1]))
print("Number of rows: {}".format(data.shape[0]))
```

```
Number of columns: 31
Number of rows: 284807
```

```
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
```

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

	0
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

```
data = data.dropna()
data.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	-0.551600	-0.617801	-0.991390	-0.
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.612727	1.065235	0.489095	-0.
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.624501	0.066084	0.717293	-0.
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	-0.226487	0.178228	0.507757	-0.
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	-0.822843	0.538196	1.345852	-1.

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

	0
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

```
data.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	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16	1.487313e-15	-5.556467e-16	1.213481e-16	-2.406331
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	-6.430976
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	-5.142873
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	5.971390
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

Double-click (or enter) to edit

```
from sklearn.preprocessing import StandardScaler
```

```
#Normalising the Amount Column
sc = StandardScaler()
data['Amount'] = sc.fit_transform(pd.DataFrame(data['Amount']))
```

```
data.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	-0.551600	-0.617801	-0.991390	-0.
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.612727	1.065235	0.489095	-0.
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.624501	0.066084	0.717293	-0.
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	-0.226487	0.178228	0.507757	-0.
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	-0.822843	0.538196	1.345852	-1.

```
# Removintg the Time column as it is not relevant in this context
data = data.drop( ['Time'], axis =1 )
```

```
data.head()
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	-0.551600	-0.617801	-0.991390	-0.311169
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.612727	1.065235	0.489095	-0.143772
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.624501	0.066084	0.717293	-0.165946
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	-0.226487	0.178228	0.507757	-0.287924
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	-0.822843	0.538196	1.345852	-1.119670

```
data.duplicated().any()
```

True

```
data = data.drop_duplicates()
```

```
data.shape
```

(275663, 30)

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

	count
Class	
0	275190
1	473

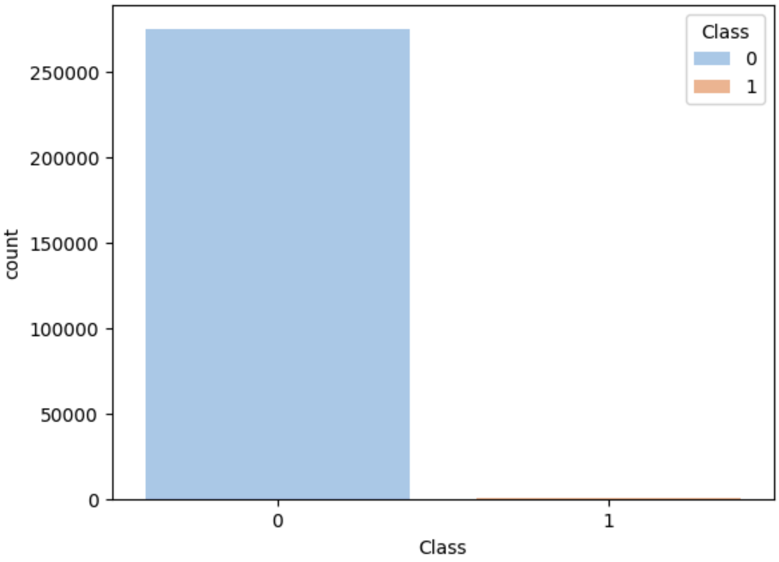
dtype: int64

We can see above that we have a highly imbalanced data set.

```
import seaborn as sns
import matplotlib.pyplot as plt
```

```
#sns.countplot(data['Class'])
#plt.show()

#counts = data["Class"].value_counts()
#plt.bar(counts.index, counts.values)
sns.countplot(x= 'Class',hue= 'Class', data =  data, palette = "pastel")
plt.show()
```



Proceeding with ML Models without rectificing the Imbalance in the Dataset

```
X = data.drop( 'Class' , axis = 1 )
y = data['Class']
```

```
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, random_state = 42)
```

```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
```

```
classifier =  {
   "Logistic Regression": LogisticRegression(),
   "Decision Tree Classifier": DecisionTreeClassifier(),
   "Random Forest Classifier": RandomForestClassifier()
}
for name,clf in classifier.items():
  print(f"\n----------{name}----------")
  clf.fit(X_train, y_train)
  y_pred = clf.predict(X_test)
  print(f"\n Accuracy Score: {accuracy_score(y_test,y_pred)}")
  print(f"\n F1 Score: {f1_score(y_test,y_pred)}")
  print(f"\n Precision Score: {precision_score(y_test,y_pred)}")
  print(f"\n Recall Score: {recall_score(y_test,y_pred)}")
```

```
----------Logistic Regression----------

 Accuracy Score: 0.9992563437505668

 F1 Score: 0.7354838709677419

 Precision Score: 0.890625

 Recall Score: 0.6263736263736264

----------Decision Tree Classifier----------

 Accuracy Score: 0.998911722561805

 F1 Score: 0.6875

 Precision Score: 0.6534653465346535

 Recall Score: 0.7252747252747253

----------Random Forest Classifier----------

 Accuracy Score: 0.9994558612809026

 F1 Score: 0.8148148148148148
```

Precision Score: 0.9295774647887324

Recall Score: 0.7252747252747253

UNDER SAMPLING

```
normal = data[data['Class'] == 0]
fraud = data[data['Class'] == 1]
```

```
normal.shape
```

(275190, 30)

```
fraud.shape
```

(473, 30)

```
normal_sample = normal.sample(n= 473)
```

```
normal_sample.shape
```

(473, 30)

```
new_data = pd.concat([normal_sample, fraud], ignore_index=True)
```

```
new_data.head()
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14
0	-0.494230	0.733320	0.160413	-0.512351	0.665302	-0.783576	1.171696	-0.344670	1.184962	-1.014506	-1.140251	0.120300	-0.676340	-1.869127
1	1.694153	-1.387191	-0.800455	-0.579831	-0.697550	0.309532	-0.595485	0.068244	2.090922	-0.699917	-1.735768	0.659313	0.405766	-0.606603
2	2.278688	-0.605677	-1.770042	-1.093033	0.160237	-0.527311	-0.190896	-0.340034	-0.601186	0.754961	-1.206650	-0.149267	1.294559	-0.298805
3	1.133394	-0.155550	0.732484	0.587702	-0.685832	-0.362420	-0.310261	-0.049633	0.417638	-0.131202	-0.725793	0.262874	0.657969	-0.088045
4	-0.503285	1.281522	2.288594	3.578588	-1.280811	1.514414	-1.546395	-2.937389	0.179539	0.585006	-0.911733	1.461982	1.141132	-0.967224

```
new_data.shape
```

(946, 30)

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

Class	count
0	473
1	473

dtype: int64

```
X0 = new_data.drop( 'Class' , axis = 1 )
y0 = new_data['Class']
```

```
X0_train, X0_test, y0_train, y0_test = train_test_split(X0, y0, test_size = 0.2, random_state = 42)
```

```
classifier =  {
    "Logistic Regression": LogisticRegression(),
    "Decision Tree Classifier": DecisionTreeClassifier(),
    "Random Forest Classifier": RandomForestClassifier()
}
for name,clf in classifier.items():
  print(f"\n----------{name}----------")
  clf.fit(X0_train, y0_train)
  y0_pred = clf.predict(X0_test)
  print(f"\n Accuracy Score: {accuracy_score(y0_test,y0_pred)}")
  print(f"\n F1 Score: {f1_score(y0_test,y0_pred)}")
  print(f"\n Precision Score: {precision_score(y0_test,y0_pred)}")
  print(f"\n Recall Score: {recall_score(y0_test,y0_pred)}")
```

----------Logistic Regression----------

Accuracy Score: 0.9421052631578948

F1 Score: 0.9441624365482234

Precision Score: 0.9789473684210527

Recall Score: 0.9117647058823529

----------Decision Tree Classifier----------

Accuracy Score: 0.9210526315789473

F1 Score: 0.9238578680203046

Precision Score: 0.9578947368421052

Recall Score: 0.8921568627450981

----------Random Forest Classifier----------

Accuracy Score: 0.9315789473684211

F1 Score: 0.9333333333333333

Precision Score: 0.978494623655914

Recall Score: 0.8921568627450981

OVER SAMPLING

```
X1 = data.drop('Class', axis = 1)
y1 = data['Class']
```

```
X1.shape
```

(275663, 29)

```
y1.shape
```

(275663,)

```
from imblearn.over_sampling import SMOTE
```

```
X_res, y_res = SMOTE().fit_resample(X1,y1)
```

```
y_res.value_counts()
```

Class	count
0	275190
1	275190

dtype: int64

```
X1_train, X1_test, y1_train, y1_test = train_test_split(X_res, y_res, test_size = 0.2, random_state = 42)
```

```
classifier = {
    "Logistic Regression": LogisticRegression(),
    "Decision Tree Classifier": DecisionTreeClassifier(),
    "Random Forest Classifier": RandomForestClassifier()
}

for name, clf in classifier.items():
    print(f"\n----------{name}----------")
    clf.fit(X1_train, y1_train)
    y1_pred = clf.predict(X1_test)
    print(f"\n Accuaracy: {accuracy_score(y1_test, y1_pred)}")
    print(f"\n Precision: {precision_score(y1_test, y1_pred)}")
    print(f"\n Recall: {recall_score(y1_test, y1_pred)}")
    print(f"\n F1 Score: {f1_score(y1_test, y1_pred)}")
```

----------Logistic Regression----------

Accuaracy: 0.9451106508230677

Precision: 0.9733366847773546

Recall: 0.915222806028762

F1 Score: 0.943385618710294

----------Decision Tree Classifier----------

Accuaracy: 0.9982194120425888

Precision: 0.9973682777646696