About the data

The data source is CreditCardTransactions by European customers in the year 2023. This is a dataset that is equivalent to 550,000 records, and only the holder's non-identifying key information has been disclosed. The main purpose of this dataset is the building of fraud detection models and algorithms that detect suspicious activity in real time in the financial domain.

Process

We will be analyzing the data by creating a Logistic Regression Model, Support Vector machine,K-nearest Neighbors, Random forest and Decision Tree and test the data for highest accuracy and find out the best method to test the data.

Importing the required liabriries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder

import warnings
warnings.filterwarnings("ignore")
```

Load the Data

```
df=pd.read_csv("creditcard_2023.csv")
```

Explore the Data

```
df.head()
```

	id	V1	V2	V3	V4	V5	V6	V7	V8	V9	..
0	0	-0.260648	-0.469648	2.496266	-0.083724	0.129681	0.732898	0.519014	-0.130006	0.727159	.
1	1	0.985100	-0.356045	0.558056	-0.429654	0.277140	0.428605	0.406466	-0.133118	0.347452	.
2	2	-0.260272	-0.949385	1.728538	-0.457986	0.074062	1.419481	0.743511	-0.095576	-0.261297	.
3	3	-0.152152	-0.508959	1.746840	-1.090178	0.249486	1.143312	0.518269	-0.065130	-0.205698	.
4	4	-0.206820	-0.165280	1.527053	-0.448293	0.106125	0.530549	0.658849	-0.212660	1.049921	.

5 rows × 31 columns

```
df.shape
```

```
(568630, 31)
```

Check for some important statistical insights from the data

```
df.describe().T
```

	count	mean	std	min	25%	50%	7
id	568630.0	2.843145e+05	164149.486121	0.000000	142157.250000	284314.500000	426471.7500
V1	568630.0	-5.638058e-17	1.000001	-3.495584	-0.565286	-0.093638	0.8326
V2	568630.0	-1.319545e-16	1.000001	-49.966572	-0.486678	-0.135894	0.3435
V3	568630.0	-3.518788e-17	1.000001	-3.183760	-0.649299	0.000353	0.6285
V4	568630.0	-2.879008e-17	1.000001	-4.951222	-0.656020	-0.073762	0.7070
V5	568630.0	7.997245e-18	1.000001	-9.952786	-0.293496	0.081088	0.4397
V6	568630.0	-3.958636e-17	1.000001	-21.111108	-0.445871	0.078718	0.4977
V7	568630.0	-3.198898e-17	1.000001	-4.351839	-0.283533	0.233366	0.5259
V8	568630.0	2.109273e-17	1.000001	-10.756342	-0.192257	-0.114524	0.0472
V9	568630.0	3.998623e-17	1.000001	-3.751919	-0.568745	0.092526	0.5592
V10	568630.0	1.991314e-16	1.000001	-3.163276	-0.590101	0.262614	0.5924
V11	568630.0	-1.183592e-16	1.000001	-5.954723	-0.701449	-0.041050	0.7477
V12	568630.0	-5.758017e-17	1.000001	-2.020399	-0.831133	0.162052	0.7446
V13	568630.0	-5.698037e-18	1.000001	-5.955227	-0.696667	0.017608	0.6856
V14	568630.0	-4.078595e-17	1.000001	-2.107417	-0.873206	0.230501	0.7518
V15	568630.0	2.649087e-17	1.000001	-3.861813	-0.621249	-0.039256	0.6654
V16	568630.0	-1.719408e-17	1.000001	-2.214513	-0.716265	0.134026	0.6556
V17	568630.0	-3.398829e-17	1.000001	-2.484938	-0.619491	0.271641	0.5182
V18	568630.0	-5.837989e-17	1.000001	-2.421949	-0.556046	0.087294	0.5443
V19	568630.0	2.479146e-17	1.000001	-7.804988	-0.565308	-0.025979	0.5601
V20	568630.0	-1.579456e-17	1.000001	-78.147839	-0.350240	-0.123378	0.2482
V21	568630.0	4.758361e-17	1.000001	-19.382523	-0.166441	-0.037431	0.1479
V22	568630.0	3.948640e-18	1.000001	-7.734798	-0.490489	0.027329	0.4638

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 568630 entries, 0 to 568629
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   id      568630 non-null  int64  
 1   V1      568630 non-null  float64
 2   V2      568630 non-null  float64
 3   V3      568630 non-null  float64
 4   V4      568630 non-null  float64
 5   V5      568630 non-null  float64
 6   V6      568630 non-null  float64
 7   V7      568630 non-null  float64
 8   V8      568630 non-null  float64
 9   V9      568630 non-null  float64
 10  V10     568630 non-null  float64
 11  V11     568630 non-null  float64
 12  V12     568630 non-null  float64
 13  V13     568630 non-null  float64
 14  V14     568630 non-null  float64
 15  V15     568630 non-null  float64
 16  V16     568630 non-null  float64
 17  V17     568630 non-null  float64
 18  V18     568630 non-null  float64
```

```
 19  V19     568630 non-null  float64
 20  V20     568630 non-null  float64
 21  V21     568630 non-null  float64
 22  V22     568630 non-null  float64
 23  V23     568630 non-null  float64
 24  V24     568630 non-null  float64
 25  V25     568630 non-null  float64
 26  V26     568630 non-null  float64
 27  V27     568630 non-null  float64
 28  V28     568630 non-null  float64
 29  Amount  568630 non-null  float64
 30  Class   568630 non-null  int64
dtypes: float64(29), int64(2)
memory usage: 134.5 MB
```

Check for the count of fraudulent transactions and legit transactions

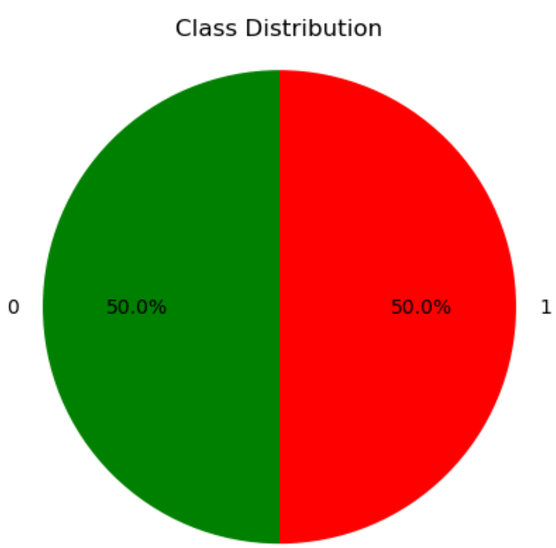
Here '0' represents 'Legit Transaction' while '1' represents 'Fraudulent Transaction'

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

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

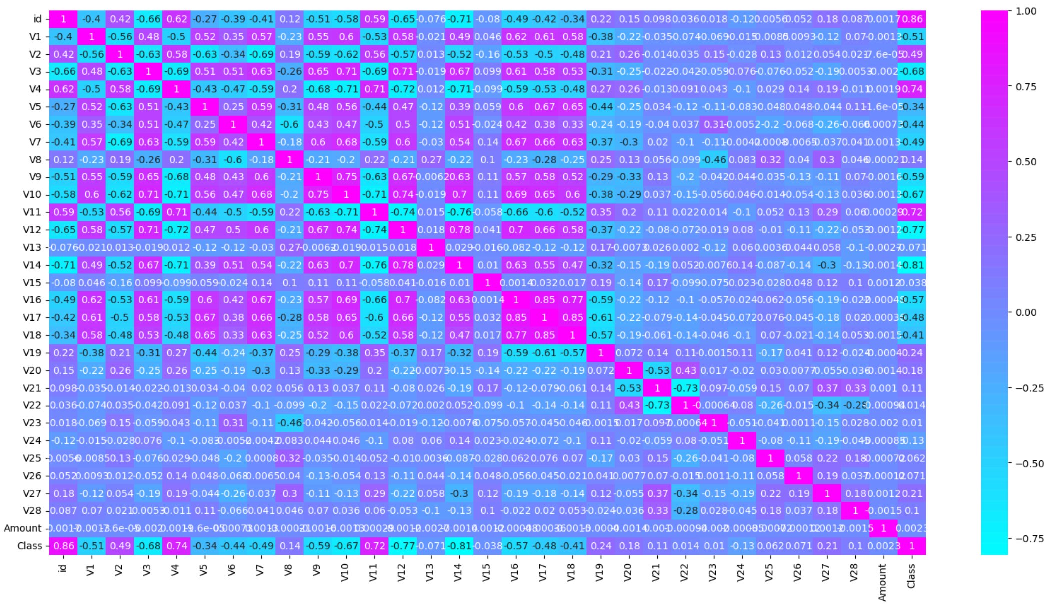
We can visualize this data using pie chart for later use

```
labels =['0','1']
sizes = df['Class'].value_counts()
colors = ['Green', 'Red']
explode = (0, 0)


# Create a pie chart

plt.pie(sizes, labels=labels, colors=colors, explode=explode, autopct='%1.1f%%', startangle=90)


plt.axis('equal')
plt.title('Class Distribution')


# Display the pie chart
plt.show()
```



Heatmap for the Correlations

```
paper = plt.figure(figsize=[20,10])
sns.heatmap(df.corr(),cmap='cool',annot=True)
plt.show()
```



Data Processing

Check for any null values and duplicates in the dataset, delete if any

```
df.isnull().sum()
```

```
id        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
```

```
df.drop_duplicates()
```

	id	V1	V2	V3	V4	V5	V6	V7	V8	
0	0	-0.260648	-0.469648	2.496266	-0.083724	0.129681	0.732898	0.519014	-0.130006	
1	1	0.985100	-0.356045	0.558056	-0.429654	0.277140	0.428605	0.406466	-0.133118	
2	2	-0.260272	-0.949385	1.728538	-0.457986	0.074062	1.419481	0.743511	-0.095576	-
3	3	-0.152152	-0.508959	1.746840	-1.090178	0.249486	1.143312	0.518269	-0.065130	-
4	4	-0.206820	-0.165280	1.527053	-0.448293	0.106125	0.530549	0.658849	-0.212660	
...	...	...	...	...	...	...	...	...	...	
568625	568625	-0.833437	0.061886	-0.899794	0.904227	-1.002401	0.481454	-0.370393	0.189694	-
568626	568626	-0.670459	-0.202896	-0.068129	-0.267328	-0.133660	0.237148	-0.016935	-0.147733	
568627	568627	-0.311997	-0.004095	0.137526	-0.035893	-0.042291	0.121098	-0.070958	-0.019997	-
568628	568628	0.636871	-0.516970	-0.300889	-0.144480	0.131042	-0.294148	0.580568	-0.207723	
568629	568629	-0.795144	0.433236	-0.649140	0.374732	-0.244976	-0.603493	-0.347613	-0.340814	

568630 rows × 31 columns

We won't be using the id column of each transaction as this is a unique value and is not needed for our analysis so we will drop the column id

```
df.drop(['id'], axis = 1, inplace=True)
```

```
#lets check if it is removed or not
df
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9
0	-0.260648	-0.469648	2.496266	-0.083724	0.129681	0.732898	0.519014	-0.130006	0.727159
1	0.985100	-0.356045	0.558056	-0.429654	0.277140	0.428605	0.406466	-0.133118	0.347452
2	-0.260272	-0.949385	1.728538	-0.457986	0.074062	1.419481	0.743511	-0.095576	-0.261297
3	-0.152152	-0.508959	1.746840	-1.090178	0.249486	1.143312	0.518269	-0.065130	-0.205698
4	-0.206820	-0.165280	1.527053	-0.448293	0.106125	0.530549	0.658849	-0.212660	1.049921
...	...	...	...	...	...	...	...	...	...
568625	-0.833437	0.061886	-0.899794	0.904227	-1.002401	0.481454	-0.370393	0.189694	-0.938153
568626	-0.670459	-0.202896	-0.068129	-0.267328	-0.133660	0.237148	-0.016935	-0.147733	0.483894
568627	-0.311997	-0.004095	0.137526	-0.035893	-0.042291	0.121098	-0.070958	-0.019997	-0.122048
568628	0.636871	-0.516970	-0.300889	-0.144480	0.131042	-0.294148	0.580568	-0.207723	0.893527
568629	-0.795144	0.433236	-0.649140	0.374732	-0.244976	-0.603493	-0.347613	-0.340814	0.253971

568630 rows × 30 columns

Define the Target variable y and feature varaible x

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

Spliting the Data

I will split the data into train and test data, with test size of 20% using dataframe x and y.

```
# importing the train test split library from sklean
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=0)
```

```
#Lets see the shapes
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(454904, 29)
(113726, 29)
(454904,)
(113726,)
```

Logistic Regression Model

Lets build a logistic regression model to test the data.

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

```
▾ LogisticRegression
LogisticRegression()
```

```
# Make predictions on the test set

y_pred_test = lr.predict(x_test)
y_pred_train = lr.predict(x_train)
```

Evaluation

```
from sklearn.metrics import classification_report, accuracy_score
```

```
# Evaluate the model

accuracy = accuracy_score(y_test, y_pred_test)
print(f"The Accuracy of the model is : {accuracy * 100:.2f}% for the test data")
```

```
# classification report

print(classification_report(y_test, y_pred_test))
```

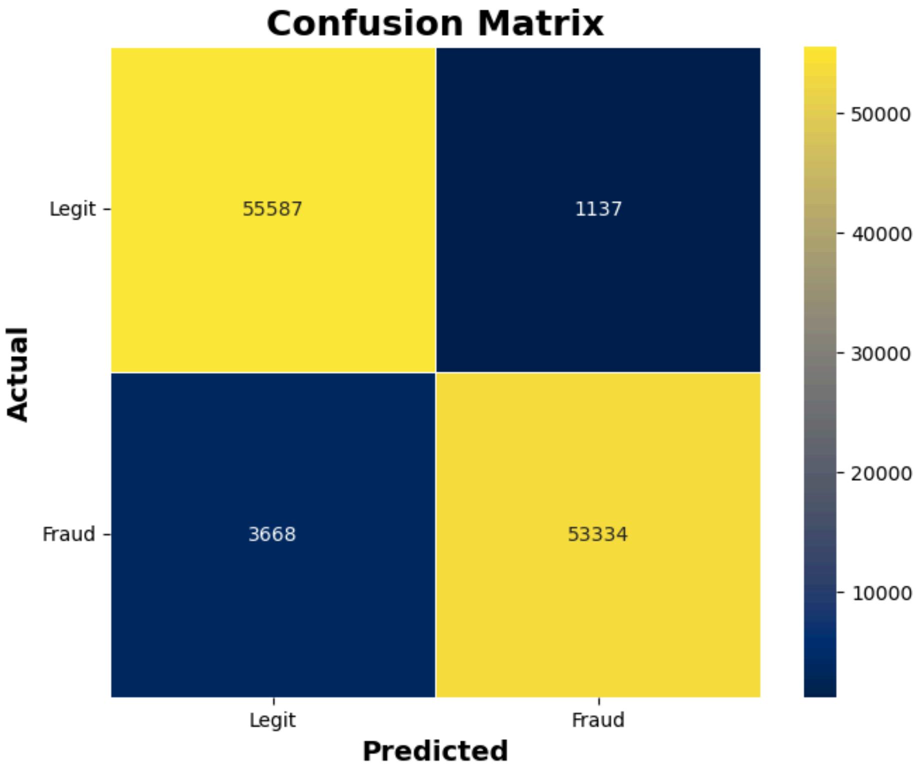
```
The Accuracy of the model is : 95.77% for the test data
              precision    recall  f1-score   support

           0       0.94      0.98      0.96     56724
           1       0.98      0.94      0.96     57002

    accuracy                           0.96    113726
   macro avg       0.96      0.96      0.96    113726
weighted avg       0.96      0.96      0.96    113726
```

Confusion Matrix

```
from sklearn.metrics import confusion_matrix
```

```
cm = confusion_matrix(y_test, y_pred_test)

plt.figure(figsize=(8, 6))

sns.heatmap(
    cm, annot=True, fmt='d', cmap='cividis', linewidths=0.4, square=True, cbar=True,
    xticklabels=["Legit", "Fraud"],
    yticklabels=["Legit", "Fraud"]
)

plt.xlabel('Predicted', fontsize=14, fontweight='bold')
plt.ylabel('Actual', fontsize=14, fontweight='bold')
plt.title('Confusion Matrix', fontsize=18, fontweight='bold')
plt.yticks(rotation=360)
```

```
plt.show()
```



Decision Tree Classifier Method

```
from sklearn.tree import DecisionTreeClassifier
```

```
dt = DecisionTreeClassifier()
dt.fit(x_train, y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
preds_dt_test = dt.predict(x_test)
```

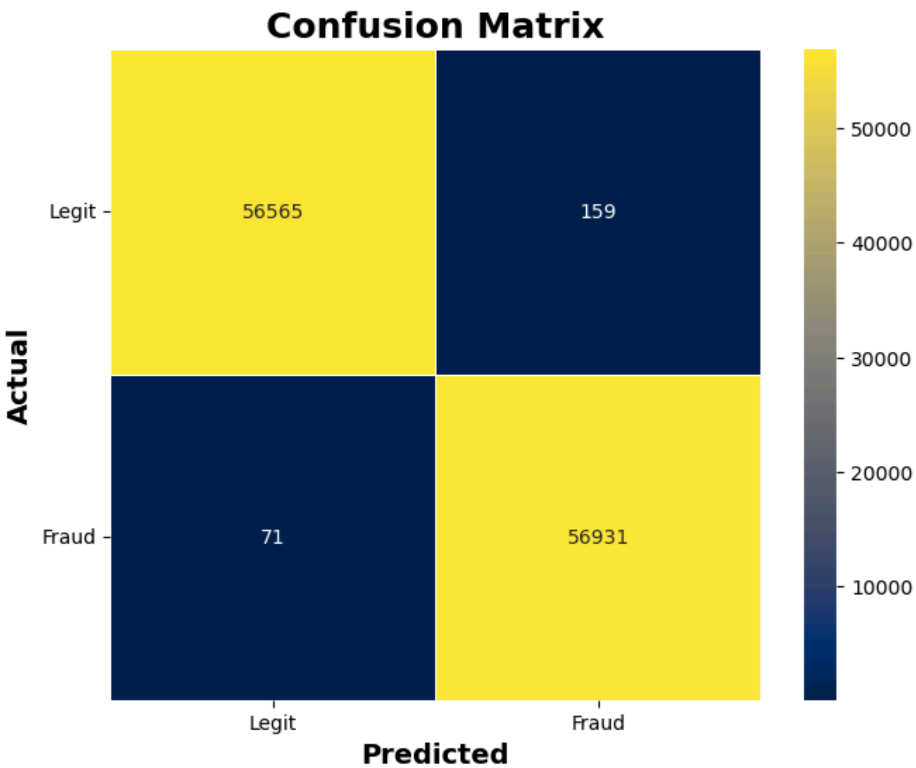
Evaluate

```
# Evaluate the model

accuracy = accuracy_score(y_test, preds_dt_test)
print(f"The Accuracy of the model is : {accuracy * 100:.2f}% for the test data")


# classification report

print(classification_report(y_test, preds_dt_test))
```

```
The Accuracy of the model is : 99.80% for the test data
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56724
           1       1.00      1.00      1.00     57002

    accuracy                           1.00    113726
   macro avg       1.00      1.00      1.00    113726
weighted avg       1.00      1.00      1.00    113726
```

Confusion Matrix

```
cm = confusion_matrix(y_test, preds_dt_test)

plt.figure(figsize=(8, 6))

sns.heatmap(
    cm, annot=True, fmt='d', cmap='cividis', linewidths=0.4, square=True, cbar=True,
    xticklabels=["Legit", "Fraud"],
    yticklabels=["Legit", "Fraud"]
)

plt.xlabel('Predicted', fontsize=14, fontweight='bold')
plt.ylabel('Actual', fontsize=14, fontweight='bold')
plt.title('Confusion Matrix', fontsize=18, fontweight='bold')
plt.yticks(rotation=360)

plt.show()
```




```
# Random Forest


# Importing necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix



# Separate features and target variable
X = df.drop('Class', axis=1)  # Features
y = df['Class']  # Target variable


# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Initialize the Random Forest classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
```

```
# Train the classifier
rf_classifier.fit(X_train, y_train)

# Predict on the test set
y_pred = rf_classifier.predict(X_test)


# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9998241387193781
```

```
# Classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56750
           1       1.00      1.00      1.00     56976

    accuracy                           1.00    113726
   macro avg       1.00      1.00      1.00    113726
weighted avg       1.00      1.00      1.00    113726
```

```
# Confusion matrix
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Confusion Matrix:
[[56744     6]
 [   14 56962]]
```

```
# K-Nearest Neighbour


# Importing necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix



X = df.drop('Class', axis=1)  # Features
y = df['Class']  # Target variable


# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)



# Initialize the KNN classifier
k = 5  # Number of neighbors to consider
knn_classifier = KNeighborsClassifier(n_neighbors=k)


# Train the classifier
knn_classifier.fit(X_train, y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

```
# Predict on the test set
y_pred = knn_classifier.predict(X_test)
```

```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9994108647099168
```

```
# Classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56750
           1       1.00      1.00      1.00     56976

    accuracy                           1.00    113726
   macro avg       1.00      1.00      1.00    113726
weighted avg       1.00      1.00      1.00    113726
```

```
# Confusion matrix
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Confusion Matrix:
[[56686    64]
 [    3 56973]]
```

```
# Support Vector Machine
```

```
# Importing necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
X = df.drop('Class', axis=1)  # Features
y = df['Class']  # Target variable
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
# Initialize the Support Vector Machine classifier
svm_classifier = SVC(kernel='linear', random_state=42)  # Using linear kernel for simplicity
```

```
# Train the classifier
svm_classifier.fit(X_train, y_train)
```

```
SVC
SVC(kernel='linear', random_state=42)
```

```
# Predict on the test set
y_pred = svm_classifier.predict(X_test)
```

```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9986458681392117
```

```
# Classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56750
           1       1.00      1.00      1.00     56976

    accuracy                           1.00    113726
   macro avg       1.00      1.00      1.00    113726
weighted avg       1.00      1.00      1.00    113726
```

```
# Confusion matrix
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Confusion Matrix:
[[56693    57]
 [   97 56879]]
```

Conclusion

- We have done Exploratory Data analysis for different features.
- We prepared our Data and build different ML Models.
- We have seen 5 different models, how they are performing w.r.t Accuracy,Precision,Recall and F1 Scores.
- Decision Tree Method has higher accuracy score on test dataset.
- We have created the confusion matrix in order to see the details of prediction accuracy of each models.