Importing the Libraies

```
import string
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.svm import LinearSVC
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
```

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

Reading the Data

```
data = pd.read_csv('data.csv',error_bad_lines=False)
```

```
b'Skipping line 2810: expected 2 fields, saw 5\nSkipping line 4641: expected 2 field
```

Knowing more about the nature of the data

```
data.shape
```

```
(161178, 2)
```

```
data.head
```

```
<bound method NDFrame.head of                  password  strength
0               kzde5577       1.0
1               kino3434       1.0
2              visi7k1yr       1.0
3               megzy123       1.0
4            lamborghin1       1.0
...                  ...       ...
161173           bruno13       0.0
```

```
161174         kundan165      1.0
161175         ghost2003      1.0
161176  y8Sg0HTc5Ng3RFJX      2.0
161177       liaoruyin11      NaN

[161178 rows x 2 columns]>
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 161178 entries, 0 to 161177
Data columns (total 2 columns):
 #   Column    Non-Null Count   Dtype  
---  ------    --------------   -----  
 0   password  161178 non-null  object 
 1   strength  161177 non-null  float64
dtypes: float64(1), object(1)
memory usage: 2.5+ MB
```

```
data.describe()
```

	strength
count	161177.000000
mean	0.989918
std	0.508018
min	0.000000
25%	1.000000
50%	1.000000
75%	1.000000
max	2.000000

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

```
1.0    119564
0.0     21619
2.0     19994
Name: strength, dtype: int64
```

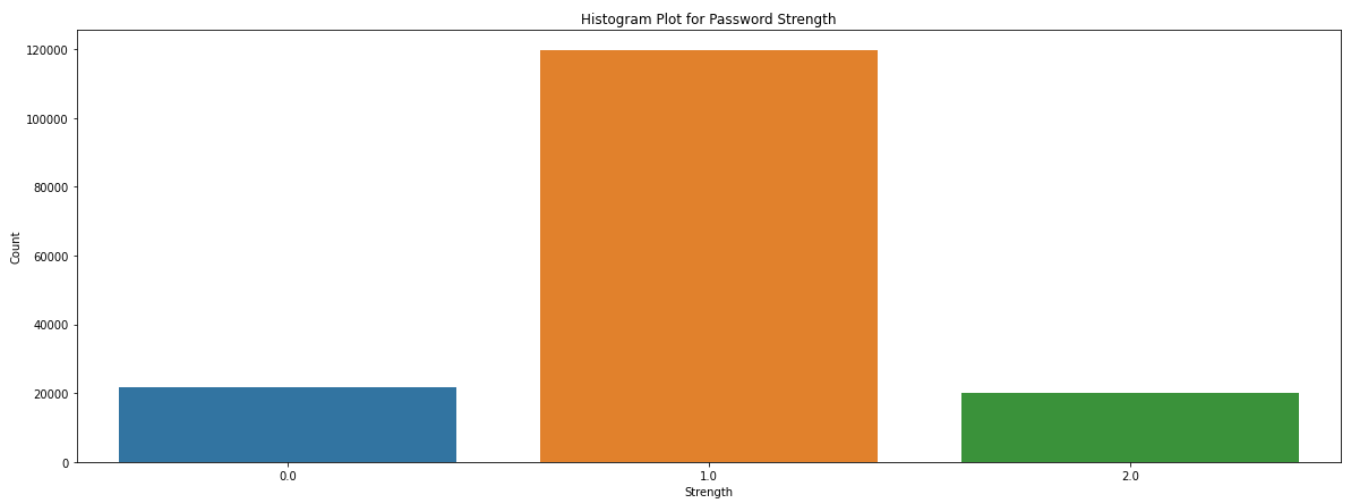
```
data.dropna(inplace=True)
data.shape
```

```
(161177, 2)
```

Illustrate the distribution of the data

Histogram Plot for Password Strength

```
plt.figure(figsize=(20,7))
sns.countplot(x=data['strength'])
plt.title('Histogram Plot for Password Strength')
plt.xlabel('Strength')
plt.ylabel('Count')
plt.show()
```



Helper Functions

- **length**: is used to compute the length of string
- **count_capital**: is used to count the capital letters of string
- **count_small**: is used to count the small letters of string
- **count_special**: is used to count the special characters of string
- **count_numbers**: is used to count the numbers exists of string

```
length        = lambda str_val: len(str_val)
count_capital = lambda str_val: sum(1 for i in str_val if i.isupper())
```

```
count_capital = lambda str_val: sum(1 for i in str_val if i.isupper())
count_small   = lambda str_val: sum(1 for i in str_val if i.islower())
count_special = lambda str_val: sum(1 for i in str_val if i not in string.ascii_letters+st
count_number  = lambda str_val: sum(1 for i in str_val if i in string.digits)
```

Applying different functions on the data

```
data['length']  = pd.DataFrame(data.password.apply(length))
data['small']   = pd.DataFrame(data.password.apply(count_small))
data['capital'] = pd.DataFrame(data.password.apply(count_capital))
data['special'] = pd.DataFrame(data.password.apply(count_special))
data['numeric'] = pd.DataFrame(data.password.apply(count_number))
```

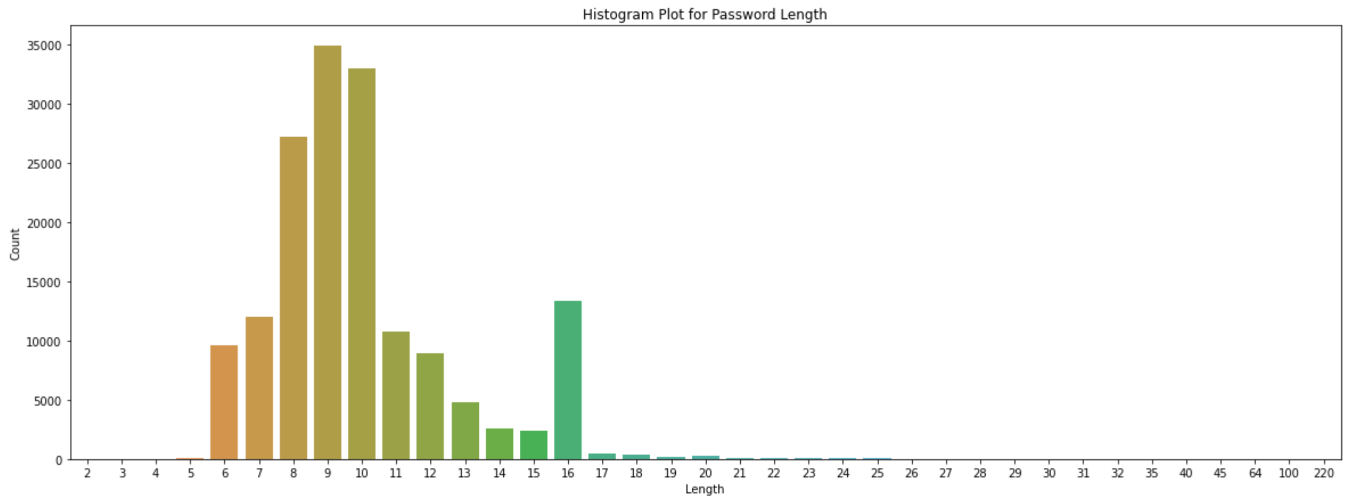
Data shape after adding new columns

```
data.head()
```

	password	strength	length	capital	small	special	number
0	kzde5577	1.0	8	0	4	0	4
1	kino3434	1.0	8	0	4	0	4
2	visi7k1yr	1.0	9	0	7	0	2
3	megzy123	1.0	8	0	5	0	3
4	lamborghin1	1.0	11	0	10	0	1

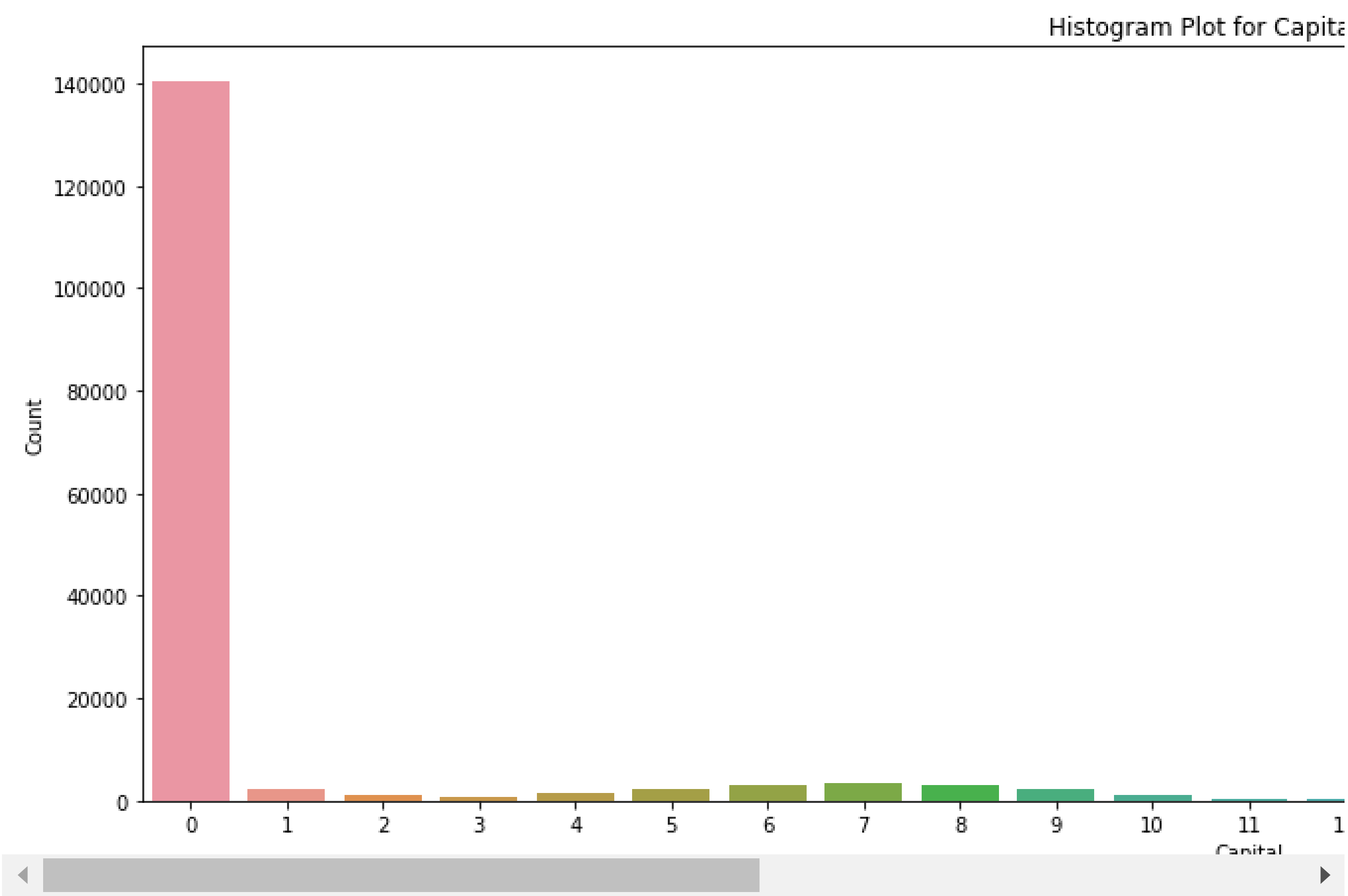
Histogram for Password Length

```
plt.figure(figsize=(20,7))
sns.countplot(x=data['length'])
plt.title('Histogram Plot for Password Length')
plt.xlabel('Length')
plt.ylabel('Count')
plt.show()
```



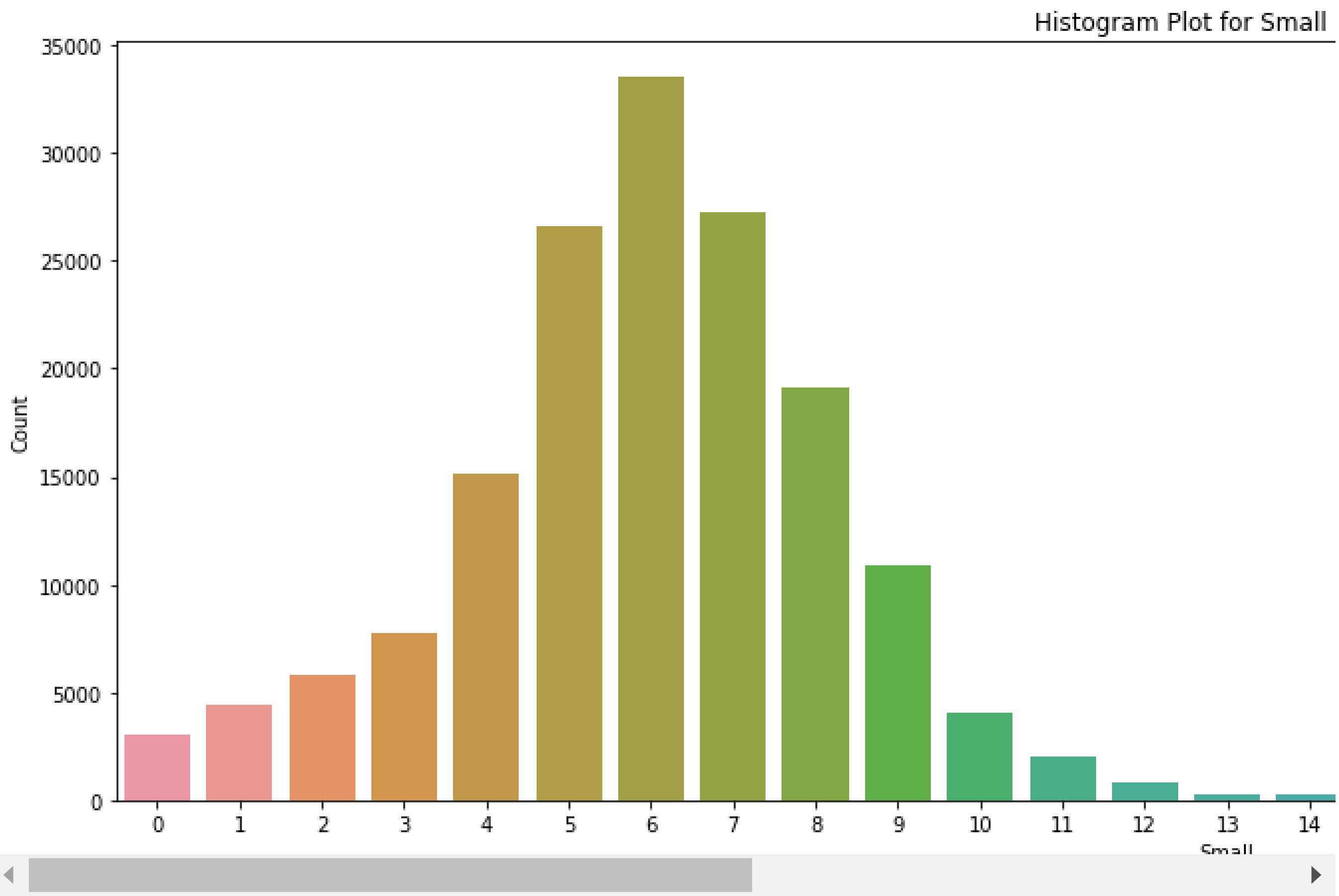
Histogram for Capital Letters

```
plt.figure(figsize=(20,7))
sns.countplot(x=data['capital'])
plt.title('Histogram Plot for Capital Letters')
plt.xlabel('Capital')
plt.ylabel('Count')
plt.show()
```



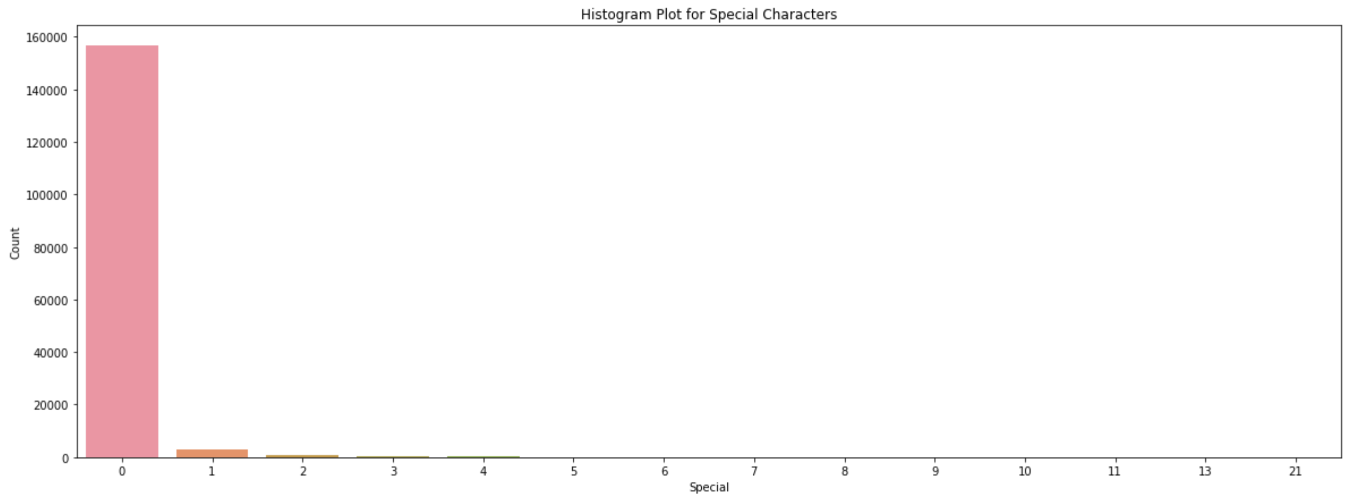
Histogram for Small Letters

```
plt.figure(figsize=(20,7))
sns.countplot(x=data['small'])
plt.title('Histogram Plot for Small Letters')
plt.xlabel('Small')
plt.ylabel('Count')
plt.show()
```



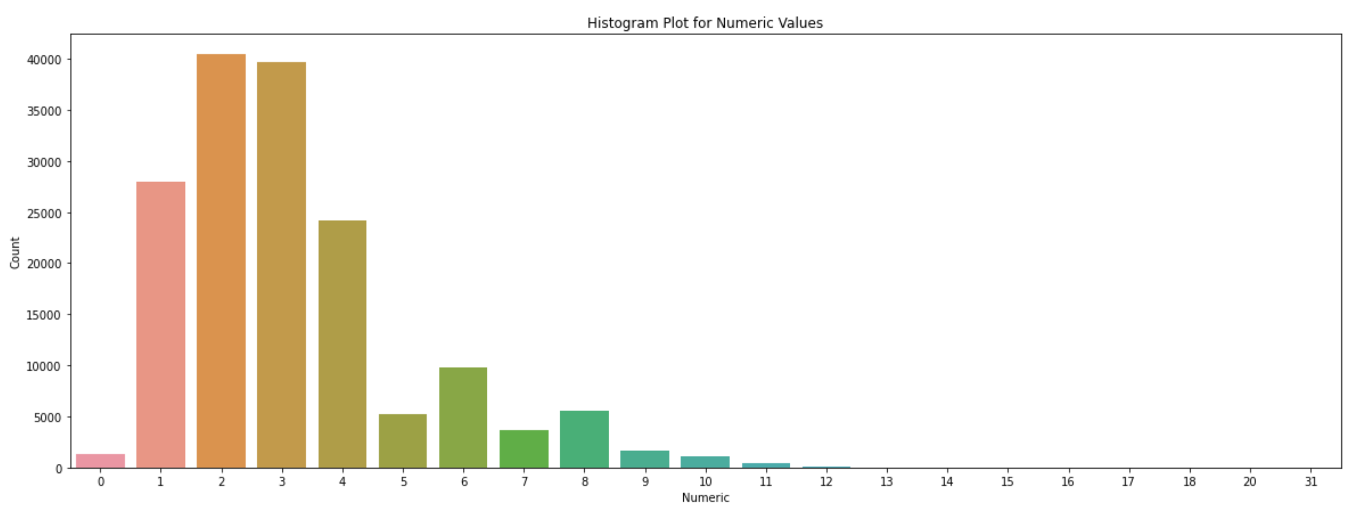
Histogram for Special Characters

```
plt.figure(figsize=(20,7))
sns.countplot(x=data['special'])
plt.title('Histogram Plot for Special Characters')
plt.xlabel('Special')
plt.ylabel('Count')
plt.show()
```



Histogram for Numeric Values

```
plt.figure(figsize=(20,7))
sns.countplot(x=data['numeric'])
plt.title('Histogram Plot for Numeric Values')
plt.xlabel('Numeric')
plt.ylabel('Count')
plt.show()
```



Preparing the data for training models

After adding new columns, we need to study the relation between the newly added ones "length, small, capital, special, numeric " and the old one "strength".
Which one has the most bowerfull affect on the length, in cour case affect means wieght

```
y_values = data['strength'].values
x_values = data[['length','capital','small','special','numeric']].values
```

```
x_values.shape, y_values.shape
```

```
((161177, 5), (161177,))
```

Split data into train & test data

```
x_train, x_test, y_train, y_test = train_test_split(x_values, y_values, test_size=0.3, rar
```

```
x_train.shape, y_train.shape
```

```
((112823, 5), (112823,))
```

```
x_test.shape, y_test.shape
```

```
((48354, 5), (48354,))
```

Applying standard scaler on data

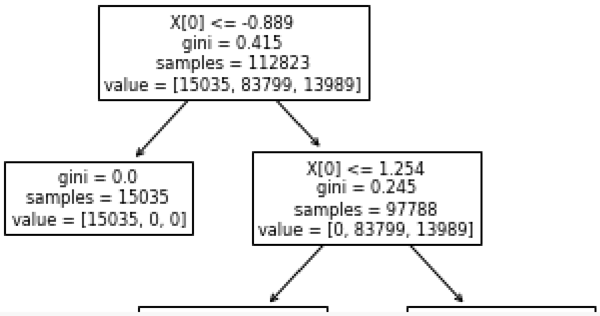
```
StanScaler = StandardScaler()
x_train_scaled = StanScaler.fit_transform(x_train)
x_test_scaled  = StanScaler.transform(x_test)
```

Applying Decision Tree Classifier algorithm

```
DTC_Model = DecisionTreeClassifier()
DTC_Model = DTC_Model.fit(x_train_scaled, y_train)
```

```
plot_tree(DTC_Model)
```

```
[Text(0.4, 0.8333333333333334, 'X[0] <= -0.889\ngini = 0.415\nsamples =
112823\nvalue = [15035, 83799, 13989]'),
 Text(0.2, 0.5, 'gini = 0.0\nsamples = 15035\nvalue = [15035, 0, 0]'),
 Text(0.6, 0.5, 'X[0] <= 1.254\ngini = 0.245\nsamples = 97788\nvalue = [0, 83799,
13989]'),
 Text(0.4, 0.16666666666666666, 'gini = 0.0\nsamples = 83799\nvalue = [0, 83799,
0]'),
 Text(0.8, 0.16666666666666666, 'gini = 0.0\nsamples = 13989\nvalue = [0, 0,
13989]')]
```




```
dtc_y_pred = DTC_Model.predict(x_test_scaled)
print("The accuracy of the model is: ",accuracy_score(y_test, dtc_y_pred)*100," % !!!")
```

```
The accuracy of the model is:  100.0  % !!!
```

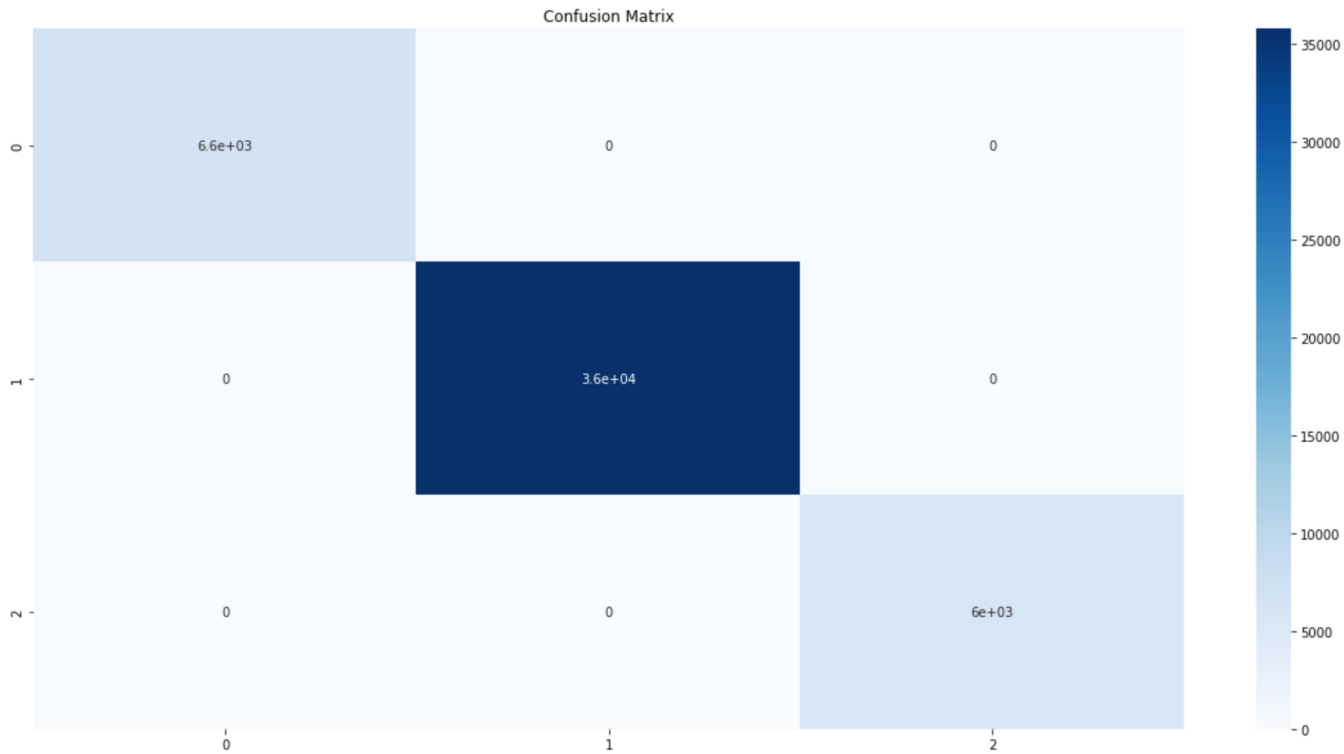
```
print(classification_report(y_test, dtc_y_pred))
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      6584
         1.0       1.00      1.00      1.00     35765
         2.0       1.00      1.00      1.00      6005

    accuracy                           1.00     48354
   macro avg       1.00      1.00      1.00     48354
weighted avg       1.00      1.00      1.00     48354
```

```
dtc_cm = confusion_matrix(y_test, dtc_y_pred)
print(dtc_cm)
```

```
[[ 6584     0     0]
 [    0 35765     0]
 [    0     0  6005]]
```

```
plt.figure(figsize=(20,10))
sns.heatmap(dtc_cm, annot=True, cmap='Blues')
plt.title('Confusion Matrix')
plt.show()
```



Applying Logistic Regression algorithm

```
LR_Model = LogisticRegression()
LR_Model = LR_Model.fit(x_train_scaled, y_train)
```

```
lr_y_pred = LR_Model.predict(x_test_scaled)
print("The accuracy of the model is: ",accuracy_score(y_test, lr_y_pred)*100," % !!!")
```

```
The accuracy of the model is:  99.99793191876577  % !!!
```

```
print(classification_report(y_test, lr_y_pred))
```

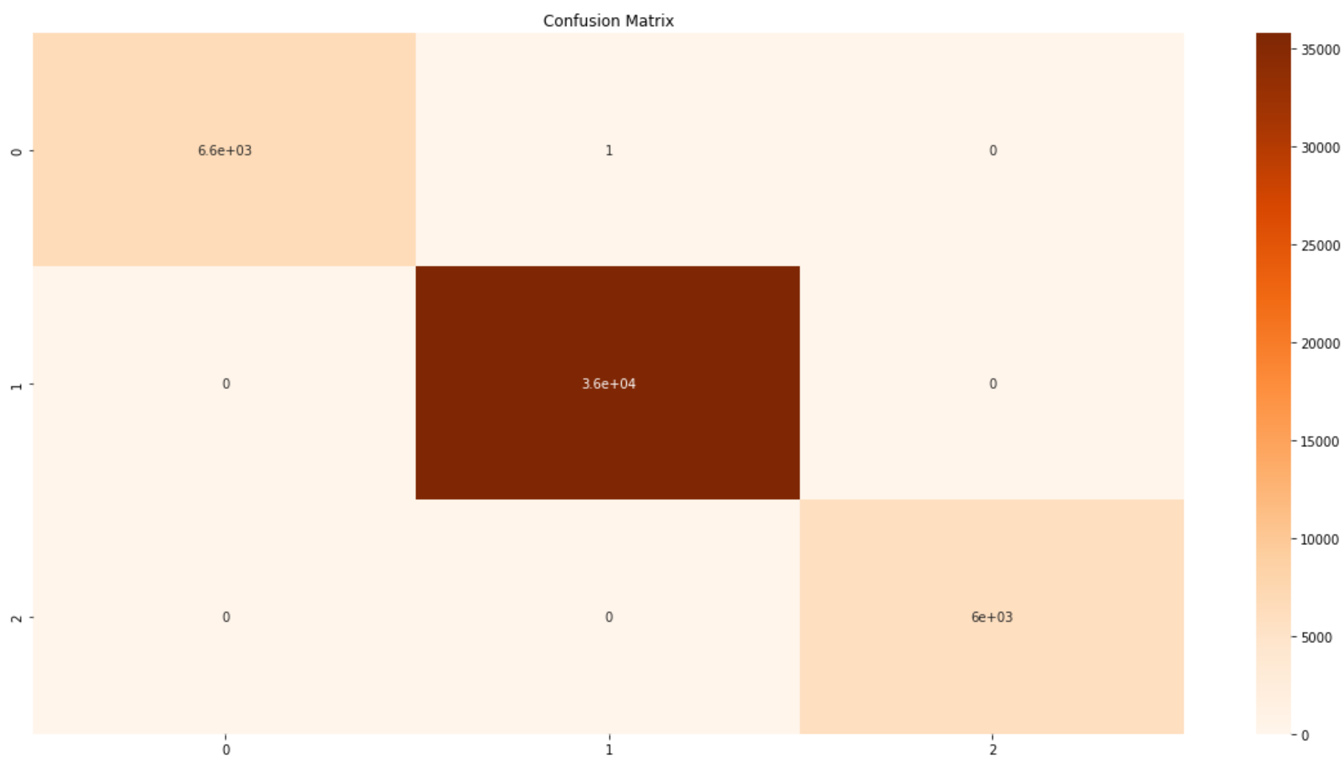
```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      6584
         1.0       1.00      1.00      1.00     35765
         2.0       1.00      1.00      1.00      6005
```

```
    accuracy                           1.00     48354
   macro avg       1.00      1.00      1.00     48354
weighted avg       1.00      1.00      1.00     48354
```

```
lr_cm = confusion_matrix(y_test, lr_y_pred)
print(lr_cm)
```

```
[[ 6583     1     0]
 [    0 35765     0]
 [    0     0  6005]]
```

```
plt.figure(figsize=(20,10))
sns.heatmap(lr_cm, annot=True, cmap='Oranges')
plt.title('Confusion Matrix')
plt.show()
```



Applying Linear Support vector Machine algorithm

```
LSVC_Model = LinearSVC()
LSVC_Model = LSVC_Model.fit(x_train_scaled, y_train)
```

```
lsvc_y_pred = LSVC_Model.predict(x_test_scaled)
print("The accuracy of the model is: ", accuracy_score(y_test, lsvc_y_pred)*100," % !!!")
```

```
The accuracy of the model is:  99.68978781486537  % !!!
```

```
print(classification_report(y_test, lsvc_y_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.99      1.00      1.00      6584
         1.0       1.00      1.00      1.00     35765
         2.0       1.00      0.99      0.99      6005

    accuracy                           1.00     48354
   macro avg       0.99      1.00      1.00     48354
weighted avg       1.00      1.00      1.00     48354
```

```
lsvc_cm = confusion_matrix(y_test, lsvc_y_pred)
print(lsvc_cm)
```

```
[[ 6583     1     0]
 [   63 35679    23]
 [    0    63  5942]]
```

```
plt.figure(figsize=(20,10))
sns.heatmap(lsvc_cm, annot=True, cmap='YlOrRd')
plt.title('Confusion Matrix')
plt.show()
```

Confusion Matrix