Credit Card

The goal of this CREDIT CARD PROJECT is to find the transaction is fraud or not. We have a clientdata in which we have total number of 31 columns and 5 rows. So, first we will visualize the
dataset so that we can check which type of opeartion is performed . Firstly we are going to applycluster and then we will check how well it is performing and if it is not performing then we will
try some other algorithm.

Project planning –

Read client data and check records.
Check null values if exist and remove/replace null values
if required.Rename data frame column if required.
Scale Raw data as per model requirement.
Perform descriptive statistics and calculate mean,
median etc.Create box plot for numerical column.
Group data and create box plot for grouped data if required.
Check correlation between variable and draw
correlation matrix.Draw histogram of data and check
density (KDE) is required.
Check type of data for regression or classification.
Perform train and test split for client data and fit into required model.
Create model as per requirement and perform
classification/regression/clustering.Try to apply some other model and
check the best model.
Create confusion matrix and classification report for these model.

Read client data and check records.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt  // for ploting charts
import seaborn as sns       // for data visualization
```

In [4]:

```
data=pd.read_csv("creditcard.csv")
data.head(5)
```

```
In [11]:
```

Out[11]:	Time	V1	V2	V3	V4	V5	V6	V7	V8	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817

5 rows × 31 columns

```
In [ ]:
```

```
x=data.iloc[:6,1:6]x.head()
```

```
In [6]:
```

Out[6]:		V1	V2	V3	V4	V5
	0	-1.359807	-0.072781	2.536347	1.378155	-0.338321
	1	1.191857	0.266151	0.166480	0.448154	0.060018
	2	-1.358354	-1.340163	1.773209	0.379780	-0.503198
	3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309
	4	-1.158233	0.877737	1.548718	0.403034	-0.407193

```
data["Class"].unique()
```

```
In [7]:
```

```
Out[7]:
```

```
data.shape
```

In [8]:

```
array([0, 1], dtype=int64)
```

Out[8]:

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

In [9]:

```
(284807, 31)
```

Check null values if exist and remove / replace null values if required.~

Out[9]:	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

```
In [10]:
```

```
V27       0
V28       0
Amount    0
Class     0

data.head(2)
dtype: int64
```

Out[10]:	Time	V1	V2	V3	V4	V5	V6	V7	V8	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.3637
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.2554

2 rows × 31 columns

```
Rename dataframe is required.
 In [ ]:
```

```
data.rename(columns={'Amount':'Amt'},inplace = True)// Rename colunmn
data.head(2)
In [11]:
```

Out[11]:	Time	V1	V2	V3	V4	V5	V6	V7	V8	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.3637
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.2554

2 rows × 31 columns

```
data.describe()
```

In [12]:

Out[12]:

	Time	V1	V2	V3	V4	V5
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05
mean	94813.859575	3.918649e-15	5.682686e-16	-8.761736e-15	2.811118e-15	-1.552103e-15
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01

8 rows × 31 columns

```
Create boxplot for numerical column.
```

In [ ]:

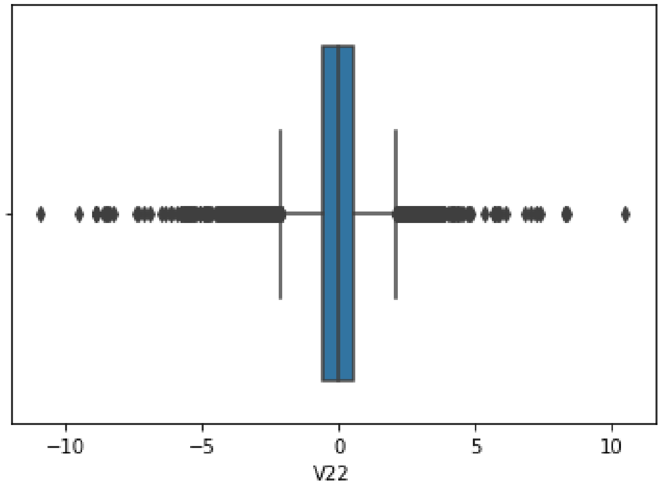
In [13]:

Out[13]:

Create Histogram

In [ ]:

```
<AxesSubplot:xlabel='V22'>
```



```
import seaborn as sns
import pandas as pd
from scipy.stats import norm
data_df=pd.read_csv("creditcard.csv")
data_df
sns.distplot(data_df["V5"],
             hist_kws = {'color':'#DC143C', 'edgecolor':'#aaff00',
                         'linewidth':5, 'linestyle':'--', 'alpha':0.4}) //Histogram
```

In [7]:

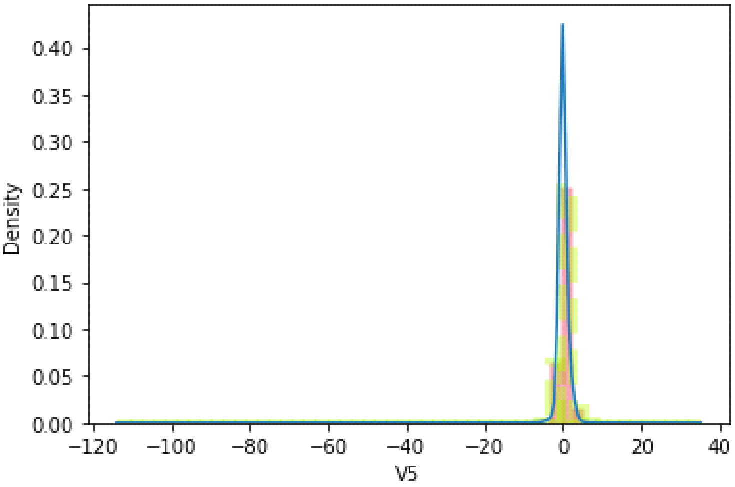
Out[7]:

```
sns.countplot(x= 'Class',data= data)
```

In [21]:

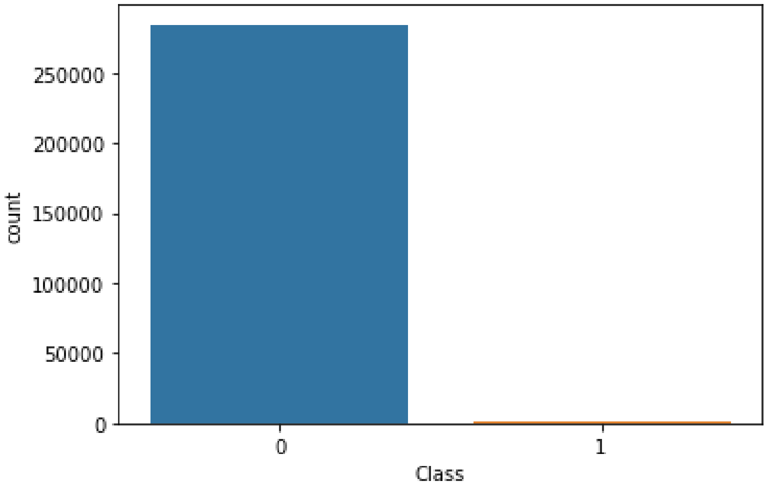
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557:   FutureWarn
ing: `distplot` is a deprecated function and will be removed in a future version. Pl ease
adapt  your  code  to  use  either  `displot`  (a  figure-level  function  with  similar  f
lexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg,  FutureWarning)
```

```
<AxesSubplot:xlabel='V5',  ylabel='Density'>
```



Out[21]:

```
<AxesSubplot:xlabel='Class',  ylabel='count'>
```



In [ ]:

Check correlation between variable and drawcorrelation matrix.

```
x.corr()// Correlation
```

In [16]:

Out[16]:

	V1	V2	V3	V4	V5
V1	1.000000	0.322754	-0.922589	-0.094628	0.576076
V2	0.322754	1.000000	-0.343689	-0.086421	0.561510
V3	-0.922589	-0.343689	1.000000	0.261916	-0.538175
V4	-0.094628	-0.086421	0.261916	1.000000	-0.500494
V5	0.576076	0.561510	-0.538175	-0.500494	1.000000

```
sns.heatmap(x.corr(),annot=True)
```

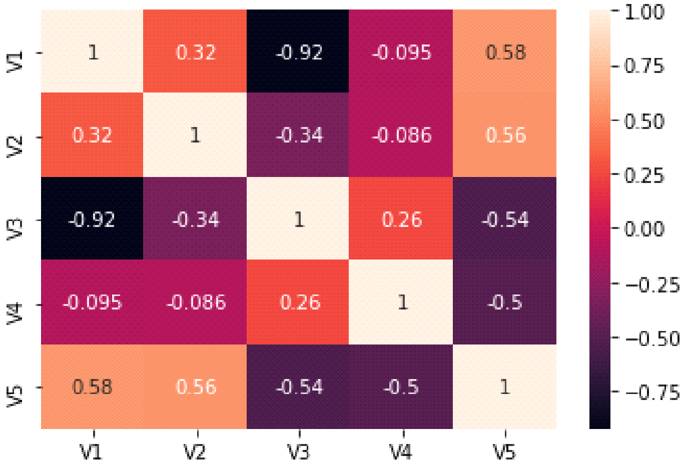
In [17]:

Out[17]:

```
plt.scatter(x.V4,x.V5,color="blue",marker="*")
```

In [18]:

```
<AxesSubplot:>
```



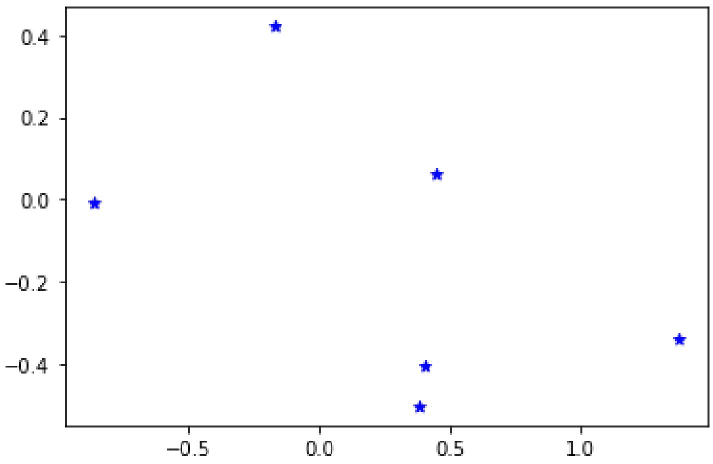
Out[18]:

```
from sklearn.preprocessing import StandardScalerscaler=StandardScaler()
scaled=scaler.fit_transform(x)
df=pd.DataFrame(scaled)
```

```
df.head(5)
```

```
In [19]:
```

```
<matplotlib.collections.PathCollection at 0x1de1240a580>
```



Out[19]:		0	1	2	3	4
	0	-0.760542	-0.204114	1.441207	1.642641	-0.652498
	1	2.091907	0.236095	-1.832810	0.272821	0.593193
	2	-0.758918	-1.850205	0.386917	0.172111	-1.168105
	3	-0.320618	-0.350159	0.414249	-1.658837	0.373266
	4	-0.535207	1.030430	0.076778	0.206362	-0.867878

Create model as per requirement andperform clustering .

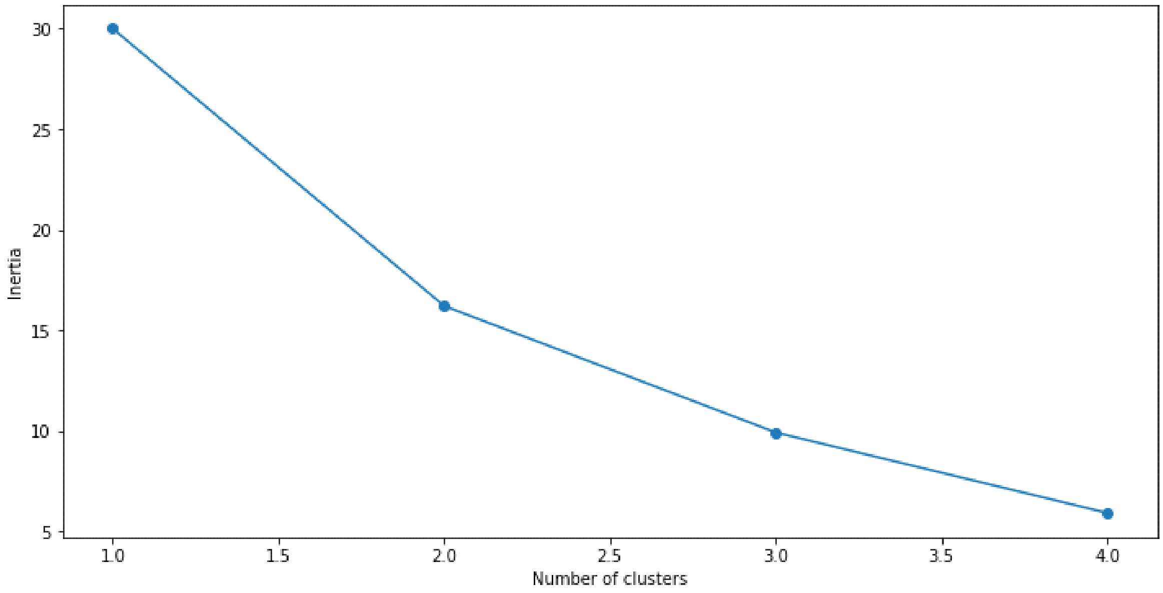
```
from sklearn.cluster import KMeans //ClusteringSSE
= []
for cluster in range(1,5):
    kmeans = KMeans(n_clusters = cluster, init='k-means++')
    kmeans.fit(scaled)
    SSE.append(kmeans.inertia_)

frame  =  pd.DataFrame({'Cluster':range(1,5),  'SSE':SSE})
plt.figure(figsize=(12,6))
plt.plot(frame['Cluster'],  frame['SSE'],  marker='o')plt.xlabel('Number
of clusters')
plt.ylabel('Inertia')
```

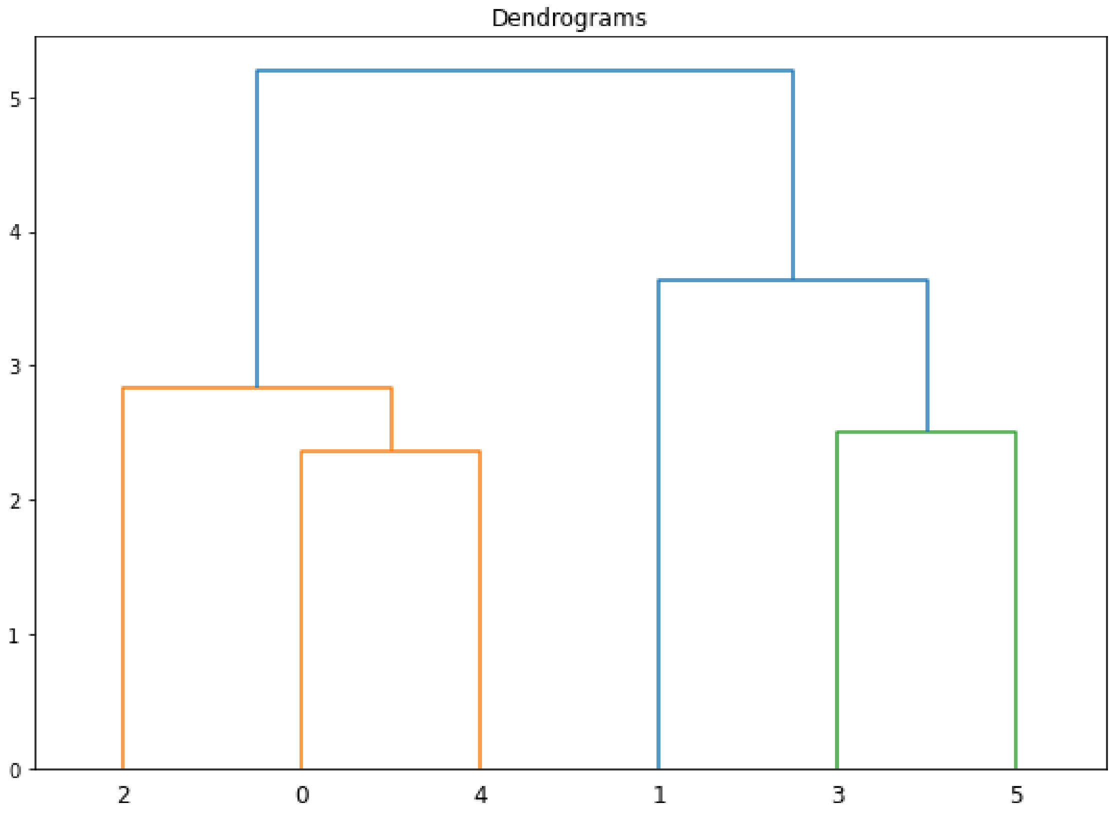
In [20]:

Out[20]:

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:881:   UserWarni
ng: KMeans is known to have a memory leak on Windows with MKL, when there are less c
hunks than available threads. You can avoid it by setting the environment variable O
MP_NUM_THREADS=1.
  warnings.warn(
Text(0, 0.5, 'Inertia')
```



```
import scipy.cluster.hierarchy as shc
plt.figure(figsize=(10,7))
plt.title("Dendrograms")
dend=shc.dendrogram(shc.linkage(scaled[0:1000,:],method='ward'))
In [21]:
```



```
kmeans  =  KMeans(n_clusters=2,  init='k-means++')
kmeans.fit(scaled)
```

In [22]:

Out[22]:

In [23]:

```
KMeans(n_clusters=2)
```

```
kmeans.inertia_
          16.200064510984635
```

Out[23]:

```
pred=kmeans.predict(scaled[0:50,:])pred
```

In [26]:

```
Out[26]:
```

```
In [27]:
```

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

```
Out[27]:
```

```
from sklearn.model_selection import train_test_split
In [29]:
```

```
np.unique(pred)
array([0, 1])
```

Perform train and test split for client data and fit into required model .

```
X_train, X_test, y_train, y_test = train_test_split(data.iloc[:,1:27],data.Class,tra
In [30]:
```

```
from sklearn.linear_model import LogisticRegressionmodel =
LogisticRegression()
In [31]:
```

```
model.fit(X_train,  y_train)
In [32]:
```

```
Out[32]:
```

```
y_predicted  =  model.predict(X_test)
y_predicted
In [33]:
```

```
LogisticRegression()
```

```
Out[33]:
```

```
from  sklearn.metrics  import  confusion_matrix  //Confusion  matrix
from  sklearn.metrics  import  classification_report
# confusion matrix
matrix  =  confusion_matrix(y_test,y_predicted,  labels=[1,0])
print('Confusion matrix : \n',matrix)
In [35]:
```

```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

Create confusion matrix and classification report for these model.

```
In [36]:
```

```
Confusion matrix :
 [[  101    49]

model.score(X_test,y_test)
 [    13 85280]]
```

```
Out[36]:
```

```
0.9992743700478681
```

```
In [12]:
```

```
Out[12]:
```

```
In [13]:
```

```
LinearRegression()
```

```
Out[13]:
```

```
reg.intercept_
In [14]:
```

```
reg.coef_
array([ 2.90009847e-16, -1.69936442e-15,  1.44977739e-16])
```

```
Out[14]:
```

```
9.604066317127357e-16
```