Assignment-1

Credit Card Fraud Detection using KNN Algorithm

Introduction:

In recent years, as there is advancement of technology, most of them are using credit card for buying their needs so the frauds associated with it is also rising gradually. In the present world, almost all the enterprises from small to big industries are using the credit card as mode of payment. Credit card fraud is happening in all organization such as appliances industry, automobile industry, banks and so on.

Technologies & Tools

- Python
- NumPy
- Pandas
- Scikit-learn (sklearn)
- Matplotlib / Seaborn (for visualization)
- Jupyter Notebook / Google Colab

Dataset

- The dataset used is the Credit Card Fraud Detection dataset from Kaggle.
- It contains transactions made by European cardholders in September 2013.
- Total: **284,807 transactions**
- Fraudulent transactions: **492** (approx. 0.17%)

Project Workflow

1. **Data Preprocessing**
   - Handling missing values (if any)
   - Feature scaling (StandardScaler)
   - Splitting the dataset into training and testing sets
2. **Dimensionality Reduction with PCA**
   - Reducing the feature space to **2 principal components** for visualization and efficiency
3. **Model Building**
   - Using **K-Nearest Neighbors (KNN)** for classification
   - Hyperparameter tuning for the best value of **K**
4. **Model Evaluation**
   - Confusion Matrix
   - Accuracy, Precision, Recall, F1-Score
5. **Visualization**
   - Plotting decision boundaries (using 2D PCA components)

- Visualizing fraud vs. non-fraud in PCA space

Code with output:

In [1]:

```
import pandas as pd
import numpy as np
import time
from matplotlib import pyplot as plt
import seaborn as sns
from mpl_toolkits.mplot3d import Axes3D
plt.style.use('ggplot')
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import normalize
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score,
f1_score, matthews_corrcoef, classification_report, roc_curve
import joblib
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

In [2]:

```
from google.colab import drive

drive.mount('/content/drive')

df=pd.read_csv('/content/drive/MyDrive/creditcard.csv')
df.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
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	0.128539	-0.189115	0.133558	-0.021053	149.62	0
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	0.167170	0.125895	-0.008983	0.014724	2.69	0
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	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
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	0.647376	-0.221929	0.062723	0.061458	123.50	0
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	-0.206010	0.502292	0.219422	0.215153	69.99	0

5 rows × 31 columns

Fig 1. Read dataset

In [3]:

```
df.shape
```

```
(284807, 31)
```

Fig 2: number of rows and columns

In [4]:

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

```
np.int64(0)
```

Fig 3: Checking if any value is null or not

In [5]:

```
All = df.shape[0]
fraud = df[df['Class'] == 1]
nonFraud = df[df['Class'] == 0]
x = len(fraud)/All
y = len(nonFraud)/All
print('frauds :',x*100,'%')
print('non frauds :',y*100,'%')
print(len(fraud))
print(len(nonFraud))
```

```
frauds : 0.1727485630620034 %
non frauds : 99.82725143693798 %
492
284315
```

Fig 4: Fraud vs non-fraud cases

In [6]:

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

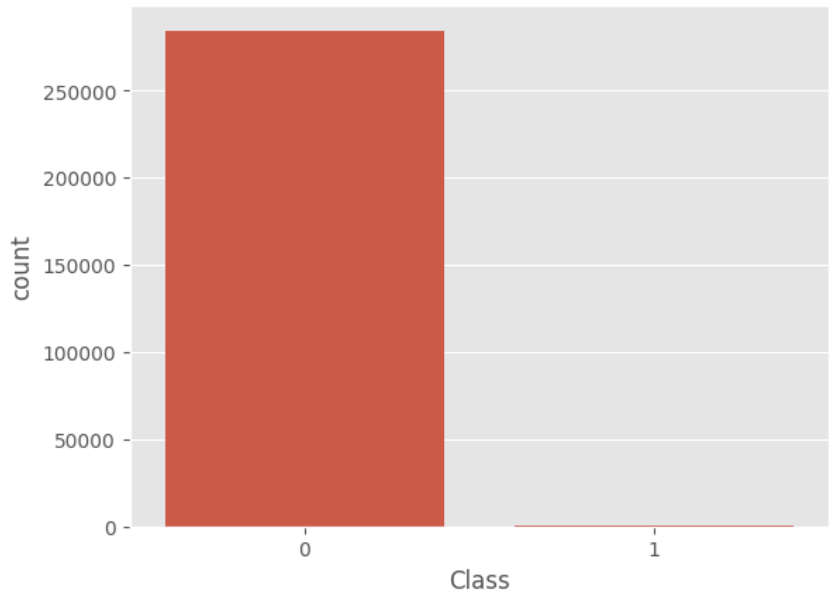

Fig 5: Graph of fraud vs non-fraud cases

In [7]:

```
amount= df['Amount'].values
sns.distplot(amount)
```

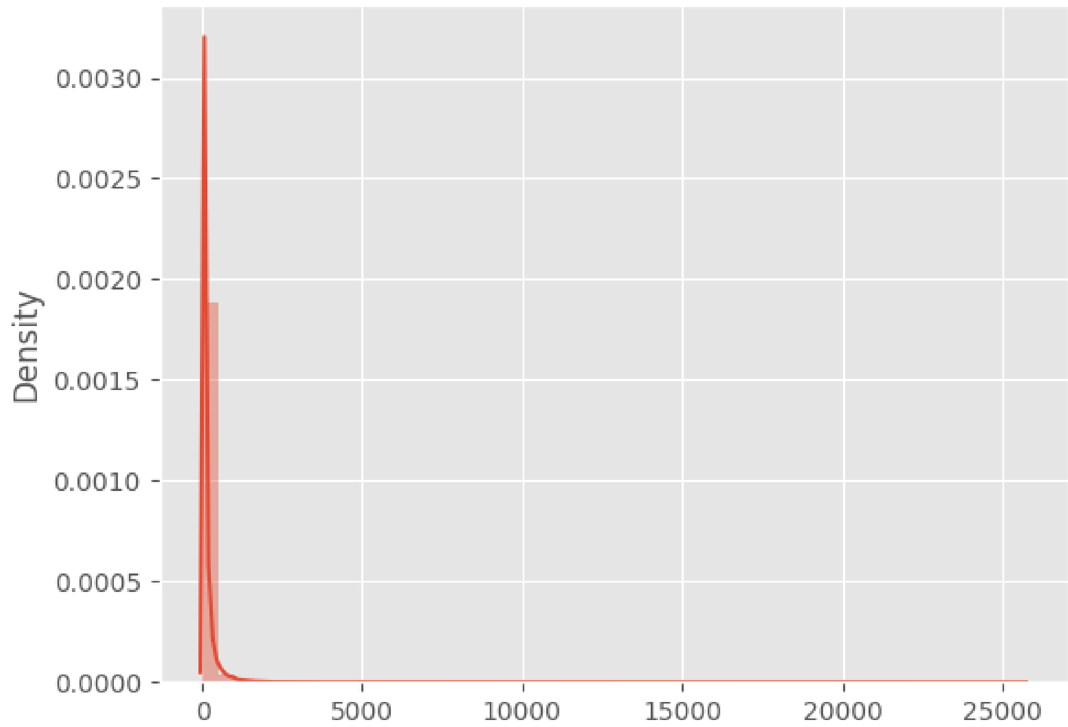

Fig 6: Amount of fraud committed

In [8]:

```
time= [df['Time'].values]
sns.distplot(time)
```

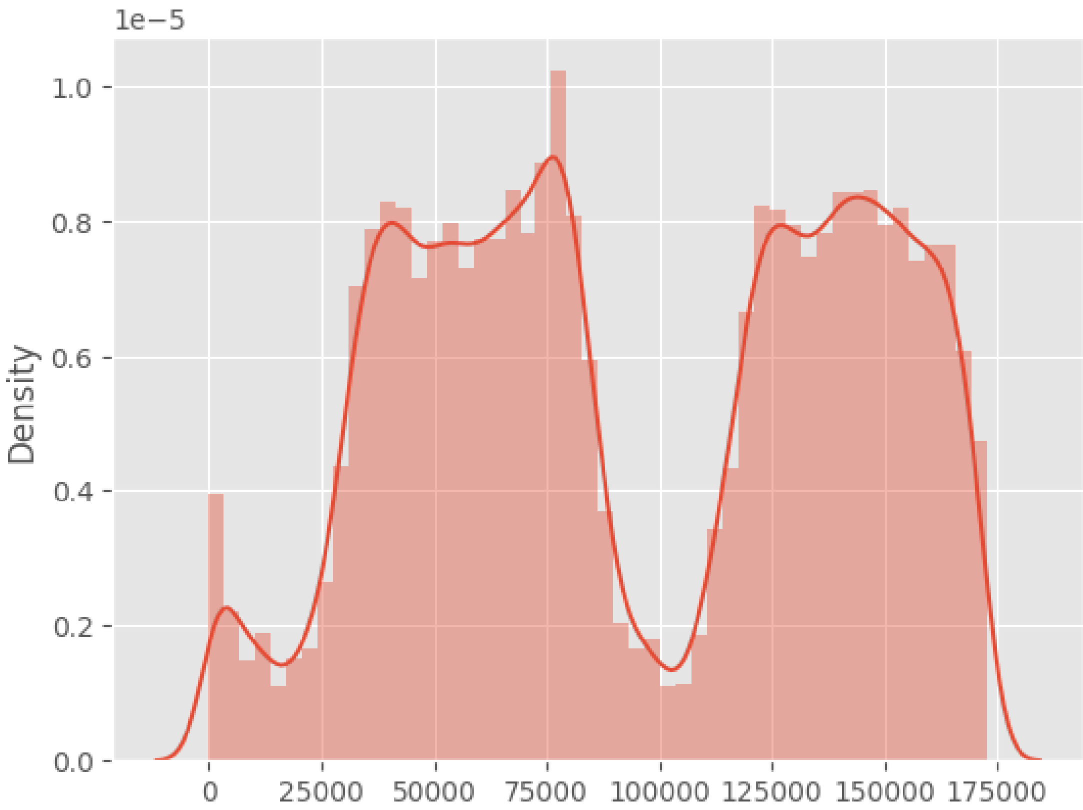

Fig 7: Number of times a fraud is committed

In [9]:

```
import matplotlib.gridspec as gridspec
anomalous_features = df.iloc[:,1:29].columns
plt.figure(figsize=(12,28*4))
gs = gridspec.GridSpec(28, 1)
for i, cn in enumerate(df[anomalous_features]):
 ax = plt.subplot(gs[i])
 sns.distplot(df[cn][df.Class == 1], bins=50)
 sns.distplot(df[cn][df.Class == 0], bins=50)
 ax.set_xlabel(")
 ax.set_title('histogram of feature: ' + str(cn))
plt.show()
```

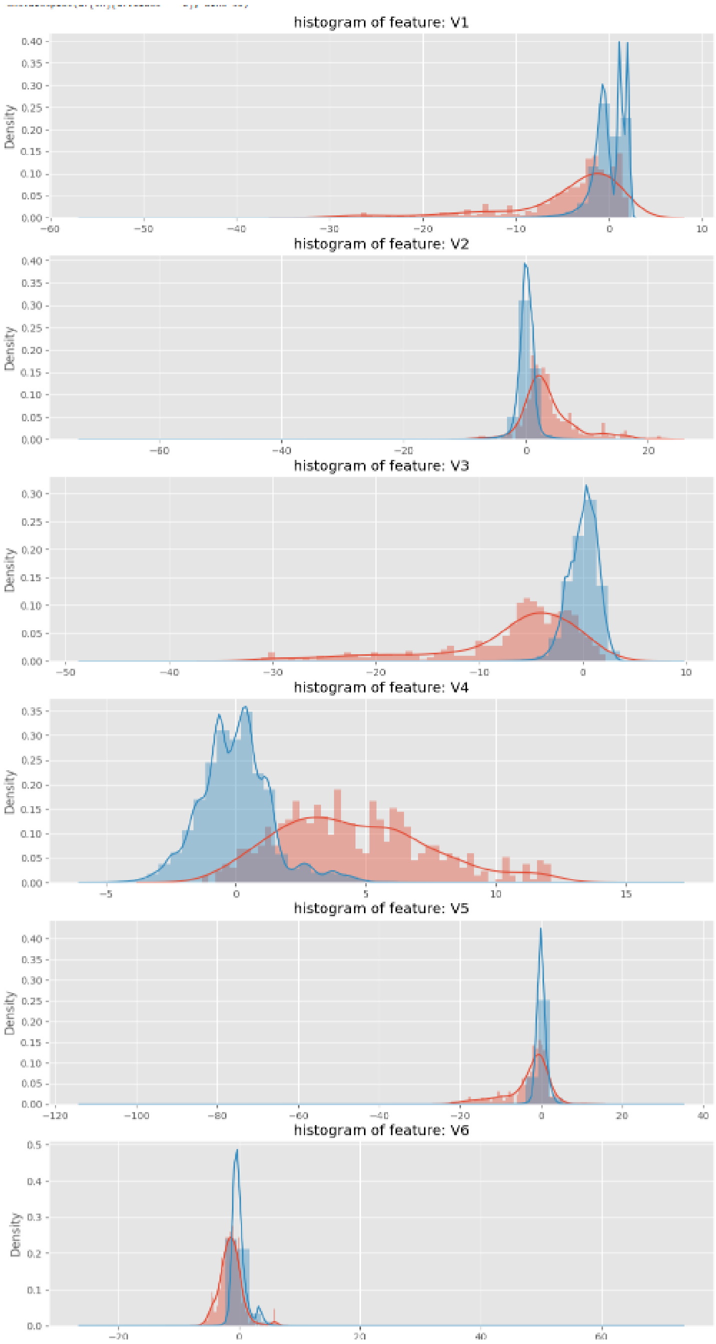

Fig 8: Histograms showing different features

In [10]:

```
df['Vamount'] = StandardScaler().fit_transform(df['Amount'].values.reshape(-1,1))
df['Vtime'] = StandardScaler().fit_transform(df['Time'].values.reshape(-1,1))
df = df.drop(['Time','Amount'], axis = 1)
df.head()
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V22	V23	V24	V25	V26	V27	V28	Class	Vamount	Vtime
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	...	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	0	0.244964	-1.996583
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	0	-0.342475	-1.996583
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	...	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	0	1.160686	-1.996562
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	...	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	0	0.140534	-1.996562
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	0	-0.073403	-1.996541

5 rows × 31 columns

Fig 9: Scaling the time and amount of data between (-1,1)

In [11]:

```
X = df.drop(['Class'], axis = 1)
y = df['Class']
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(X.values)
principalDf = pd.DataFrame(data = principalComponents
 , columns = ['principal component 1', 'principal component 2'])
finalDf = pd.concat([principalDf, y], axis = 1)
finalDf.head()
```

	principal component 1	principal component 2	Class
0	-1.571678	0.675572	0
1	1.086213	0.282673	0
2	-2.053411	-1.077634	0
3	-1.150107	0.427442	0
4	-1.143820	1.341999	0

Fig 10: Using Principal component analysis for dimension reduction

In [12]:

```
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)
targets = [0, 1]
colors = ['r', 'g']
for target, color in zip(targets,colors):
 indicesToKeep = finalDf['Class'] == target
 ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
 , finalDf.loc[indicesToKeep, 'principal component 2']
 , c = color
 , s = 50)
```

```
ax.legend(targets)
ax.grid()
```

Fig 11: Visualization to see how distinct or overlapping the data points are

In [13]:

```
df = df.sample(frac=1)
frauds = df[df['Class'] == 1]
non_frauds = df[df['Class'] == 0][:492]
new_df = pd.concat([non_frauds, frauds])
new_df = new_df.sample(frac=1, random_state=42)
labels = ['non frauds','fraud']
classes = pd.value_counts(new_df['Class'], sort = True)
classes.plot(kind = 'bar', rot=0)
plt.title("Transaction class distribution")
plt.xticks(range(2), labels)
plt.xlabel("Class")
plt.ylabel("Frequency")
```

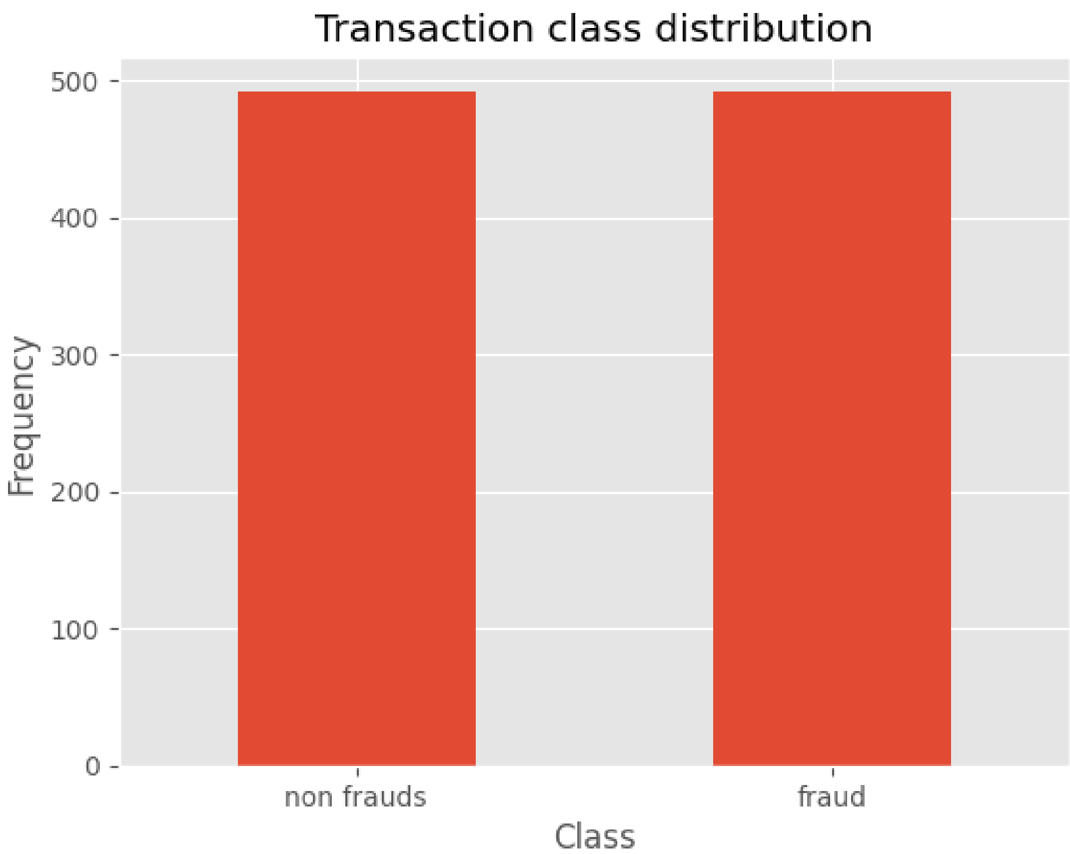

Fig 12: preparing and visualizing the distribution of a binary-class dataset

In [14]:

```
features = new_df.drop(['Class'], axis = 1)
labels = pd.DataFrame(new_df['Class'])
feature_array = features.values
label_array = labels.values
```

In [15]:

```
X_train,X_test,y_train,y_test = train_test_split(feature_array,label_array,test_size=0.20)
X_train = normalize(X_train)
X_test=normalize(X_test)
```

In [16]:

```
neighbours = np.arange(1,25)
train_accuracy =np.empty(len(neighbours))
test_accuracy = np.empty(len(neighbours))
for i,k in enumerate(neighbours):
 knn=KNeighborsClassifier(n_neighbors=k,algorithm="kd_tree",n_jobs=-1)
 knn.fit(X_train,y_train.ravel())
 train_accuracy[i] = knn.score(X_train, y_train.ravel())
 test_accuracy[i] = knn.score(X_test, y_test.ravel())
```

In [17]:

```
plt.title('k-NN Varying number of neighbors')
plt.plot(neighbours, test_accuracy, label='Testing Accuracy')
```

```
plt.plot(neighbours, train_accuracy, label='Training accuracy')
plt.legend()
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
plt.show()
```

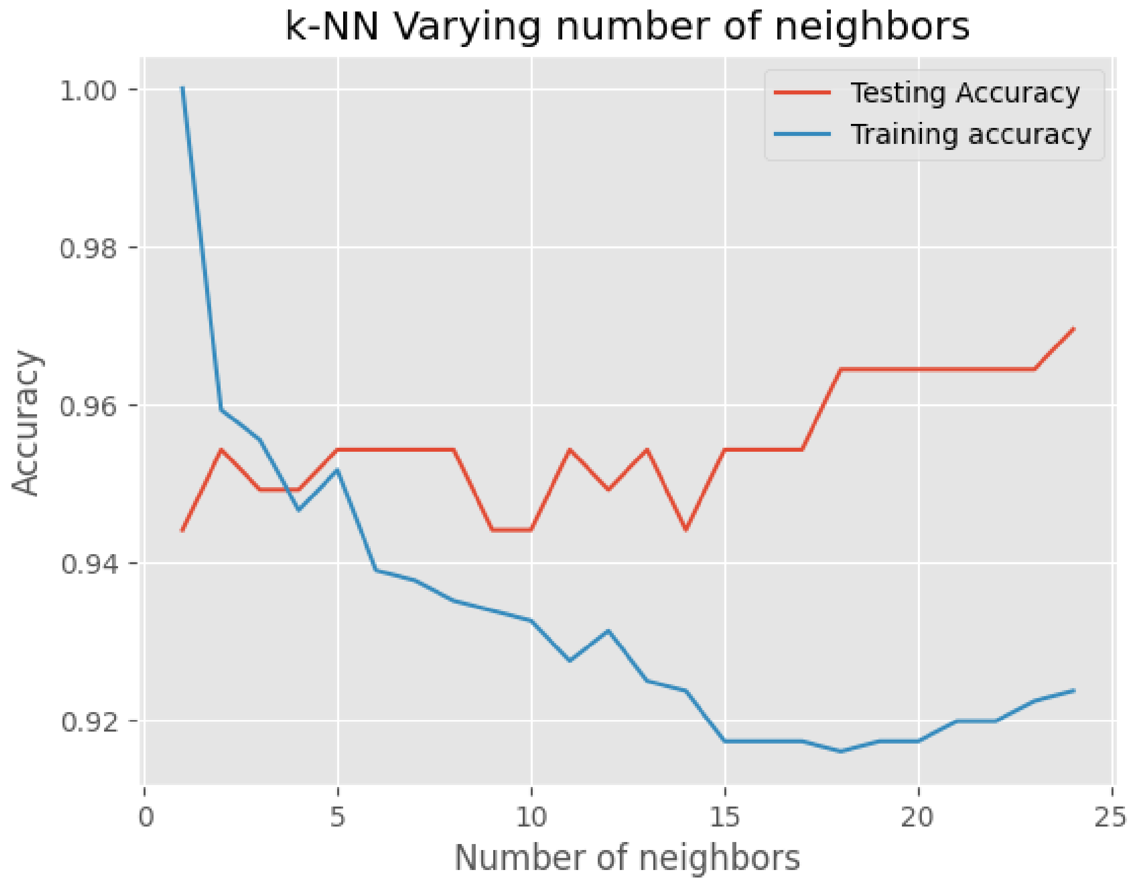

Fig 13: plot to visualize the performance of a k-Nearest Neighbors (k-NN) classifier

In [18]:

```
idx = np.where(test_accuracy == max(test_accuracy))
x = neighbours[idx]
knn=KNeighborsClassifier(n_neighbors=x[0],algorithm="kd_tree",n_jobs=-1)
knn.fit(X_train,y_train.ravel())
```

```
KNeighborsClassifier
KNeighborsClassifier(algorithm='kd_tree', n_jobs=-1, n_neighbors=np.int64(24))
```

Fig 14: Documentation for KNN Algorithm

In [19]:

```
knn_predicted_test_labels=knn.predict(X_test)
from pylab import rcParams
rcParams['figure.figsize'] = 14, 8
plt.subplot(222)
plt.scatter(X_test[:, 0], X_test[:, 1], c=knn_predicted_test_labels)
```

plt.title(" Number of Blobs")

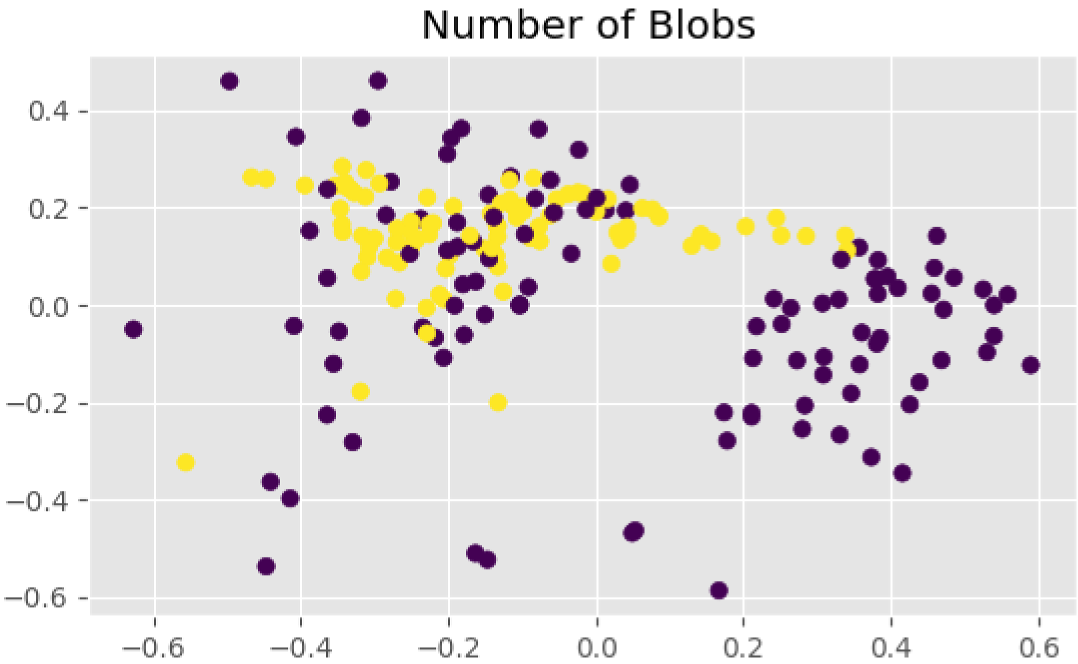

Fig 15: visualizing the predictions made by k-NN classifier

In [20]:

```
knn_accuracy_score = accuracy_score(y_test,knn_predicted_test_labels)
knn_precison_score = precision_score(y_test,knn_predicted_test_labels)
knn_recall_score = recall_score(y_test,knn_predicted_test_labels)
knn_f1_score = f1_score(y_test,knn_predicted_test_labels)
knn_MCC = matthews_corrcoef(y_test,knn_predicted_test_labels)
```

In [21]:

```
print(knn_accuracy_score)
```

```
0.9695431472081218
```

Fig 16: Accuracy for KNN

In [22]:

```
print(knn_f1_score)
```

```
0.967741935483871
```

Fig 17: F1-score for KNN

In [23]:

```
print(knn_precison_score)
```

```
0.9782608695652174
```

Fig 18: Precision score for KNN