# K-Means Clustering

**Aim**: To group passengers into clusters based on similarity between a combination of features such as Pclass, Sex, Age, Fare and Embarked using k-means clustering and perform outlier detection

**K-Means Clustering**:
K-means clustering is a method used to group data based on their similarities. The algorithm begins by randomly selecting central points, known as centroids, and assigns each data point to the closest centroid, forming clusters. Once the points are grouped, the centroids are recalculated by averaging the positions of the points in each cluster. This process is repeated until the centroids no longer change. The objective of K-means clustering is to organize data into clusters where similar points are grouped together.

**Algorithm**:
1. Load the Titanic dataset.
2. Select relevant features for clustering.
3. Convert categorical variables into numerical values using one-hot encoding.
4. Handle missing values by imputing numerical columns.
5. Scale the data using StandardScaler.
6. Apply K-Means clustering.
7. Use PCA for 2D visualization of clusters.
8. Visualize the clusters and mark the centroids.

**Code**:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
import seaborn as sns
from sklearn.metrics import silhouette_score

# Load Titanic dataset
df = pd.read_csv('Titanic-Dataset.csv')
df.head()

# Select relevant features
```

```
data = df[['Pclass', 'Sex', 'Age', 'Fare', 'Embarked']]

# One-hot encode categorical features
data = pd.get_dummies(data, drop_first=True)

# Impute missing values in 'Age' using mean strategy
imputer = SimpleImputer(strategy='mean')
data['Age'] = imputer.fit_transform(data[['Age']])

# Handle missing values in 'Embarked_Q'
data['Embarked_Q'] = data['Embarked_Q'].fillna(0)

# Scale the features
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)

# Apply KMeans clustering with 3 clusters
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(data_scaled)

# Get the cluster labels and centers
labels = kmeans.labels_
cluster_centers = kmeans.cluster_centers_

# Calculate distances from each point to cluster centroids
distances = kmeans.transform(data_scaled)
min_distances = np.min(distances, axis=1)

# Define outlier threshold based on the 95th percentile of
minimum distances
threshold = np.percentile(min_distances, 95)
outliers = min_distances > threshold

# Apply PCA for 2D visualization
pca = PCA(n_components=2)
data_pca = pca.fit_transform(data_scaled)

# Plot the clusters and outliers in PCA space
plt.figure(figsize=(8, 6))

# Plot normal points (non-outliers)
sns.scatterplot(x=data_pca[~outliers, 0],
y=data_pca[~outliers, 1], palette='Set1', s=100, marker='o',
edgecolor='black', label='Normal Points')

# Plot outliers
```

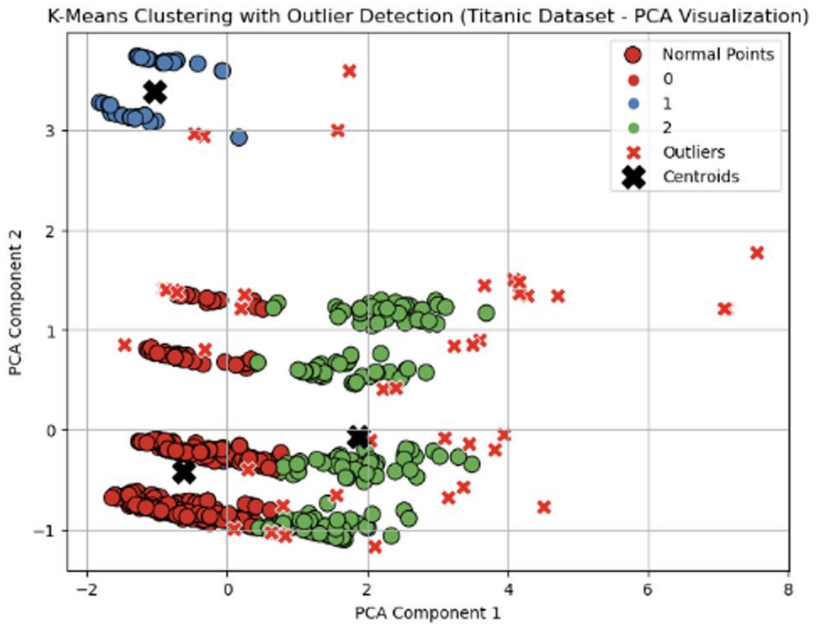
```
sns.scatterplot(x=data_pca[outliers, 0], y=data_pca[outliers,
1], color='red', s=100, marker='X', label='Outliers')

# Mark kasa musa cluster centers in PCA space
pca_centroids = pca.transform(cluster_centers)
plt.scatter(pca_centroids[:, 0], pca_centroids[:, 1], s=200,
c='black', marker='X', label='Centroids')

# Add labels and title
plt.title('K-Means Clustering with Outlier Detection (Titanic
Dataset - PCA Visualization)')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.legend()
plt.grid(True)
plt.show()

# Calculate Silhouette Score (higher is better, ranges from -1
to 1)
sil_score = silhouette_score(data_scaled, labels)
print(f"Silhouette Score: {sil_score}")
```

**Output**:




```
Silhouette Score: 0.3743369955029974
```