```
pip install palmerpenguins
```

```
Collecting palmerpenguins
  Downloading palmerpenguins-0.1.4-py3-none-any.whl (17 kB)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from palmerpenguin
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from palmerpenguins
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas-
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-date
Installing collected packages: palmerpenguins
Successfully installed palmerpenguins-0.1.4
```

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from palmerpenguins import load_penguins # For penguins dataset
import pandas as pd # For dataframes
import matplotlib.pyplot as plt # For plotting functions
import seaborn as sns # For additional plotting functions
from sklearn.cluster import KMeans # For k-Means
from sklearn.model_selection import GridSearchCV # For grid search
from sklearn.metrics import silhouette_score # For metrics and scores
from sklearn.preprocessing import StandardScaler # For standardizing data
```

```
# 1. Import the Vehicle dataset, summarize it and explain the output
data = pd.read_csv("Vehicle.csv");
print(data.describe())
```

```
              Year  Selling_Price  Present_Price      Kms_Driven       Owner
count   301.000000     301.000000     301.000000      301.000000  301.000000
mean   2013.627907       4.661296       7.628472    36947.205980    0.043189
std       2.891554       5.082812       8.644115    38886.883882    0.247915
min    2003.000000       0.100000       0.320000      500.000000    0.000000
25%    2012.000000       0.900000       1.200000    15000.000000    0.000000
50%    2014.000000       3.600000       6.400000    32000.000000    0.000000
75%    2016.000000       6.000000       9.900000    48767.000000    0.000000
max    2018.000000      35.000000      92.600000   500000.000000    3.000000
```

```
# 2. Show the structure and dimension of the dataset and explain it
print(data.info()) # tells how many null values, datatypes, column names
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 301 entries, 0 to 300
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Car_Name       301 non-null    object
 1   Year           301 non-null    int64
 2   Selling_Price  301 non-null    float64
 3   Present_Price  301 non-null    float64
 4   Kms_Driven     301 non-null    int64
 5   Fuel_Type      301 non-null    object
 6   Seller_Type    301 non-null    object
 7   Transmission   301 non-null    object
 8   Owner          301 non-null    int64
dtypes: float64(2), int64(3), object(4)
memory usage: 21.3+ KB
None
```

```
# 3. Show the column names of the Vehicle dataset and the first 3 rows and the last 6 rows of i
print("\n3. Column names:")
print(data.columns)
print("\nFirst 3 rows:")
```

```
print("\nFirst 3 rows:")
print(data.head(3))
print("\nLast 6 rows:")
print(data.tail(6))
```

```
3. Column names:
Index(['Car_Name', 'Year', 'Selling_Price', 'Present_Price', 'Kms_Driven',
       'Fuel_Type', 'Seller_Type', 'Transmission', 'Owner'],
      dtype='object')

First 3 rows:
  Car_Name  Year  Selling_Price  Present_Price  Kms_Driven Fuel_Type  \
0     ritz  2014           3.35           5.59       27000    Petrol
1      sx4  2013           4.75           9.54       43000    Diesel
2     ciaz  2017           7.25           9.85        6900    Petrol

  Seller_Type Transmission  Owner
0      Dealer       Manual      0
1      Dealer       Manual      0
2      Dealer       Manual      0

Last 6 rows:
    Car_Name  Year  Selling_Price  Present_Price  Kms_Driven Fuel_Type  \
295     city  2015           8.55          13.09       60076    Diesel
296     city  2016           9.50          11.60       33988    Diesel
297     brio  2015           4.00           5.90       60000    Petrol
298     city  2009           3.35          11.00       87934    Petrol
299     city  2017          11.50          12.50        9000    Diesel
300     brio  2016           5.30           5.90        5464    Petrol

    Seller_Type Transmission  Owner
295      Dealer       Manual      0
296      Dealer       Manual      0
297      Dealer       Manual      0
298      Dealer       Manual      0
299      Dealer       Manual      0
300      Dealer       Manual      0
```

```
# 4. Show the average Kms_Driven for each type of car (Car_Name)
average = data.groupby("Car_Name")["Kms_Driven"].mean()   #first grouping the Cars and how many kms the
print("\n4. Average Kms_Driven for each type of car:")
print(average)
```

```
4. Average Kms_Driven for each type of car:
Car_Name
800                  127000.000000
Activa 3g            250250.000000
Activa 4g              1300.000000
Bajaj  ct 100         35000.000000
Bajaj Avenger 150      7000.000000
                          ...
sx4                   50740.000000
verna                 42747.285714
vitara brezza          2071.000000
wagon r               40644.750000
xcent                 27448.333333
Name: Kms_Driven, Length: 98, dtype: float64
```

```
average_selling_price_by_year = data.groupby("Year")["Selling_Price"].mean()
print("\n5. Average Selling_Price of the cars in each year:")
print(average_selling_price_by_year)
```

```
5. Average Selling_Price of the cars in each year:
Year
2003    1.300000
```

```
2004    1.500000
2005    2.487500
2006    1.437500
2007    0.160000
2008    1.002857
2009    2.816667
2010    5.262667
2011    2.375263
2012    3.841304
2013    3.540909
2014    4.762105
2015    5.927049
2016    5.213200
2017    6.209143
2018    9.250000
Name: Selling_Price, dtype: float64
```

```
n"]].drop_duplicates()  #drop duplicates removes all the duplcates and keeps only the unique values
ssion:")

```

```
6. Unique combinations of Car_Name, Fuel_Type, Seller_Type, and Transmission:
    Car_Name Fuel_Type Seller_Type Transmission
0       ritz    Petrol      Dealer       Manual
1        sx4    Diesel      Dealer       Manual
2       ciaz    Petrol      Dealer       Manual
3    wagon r    Petrol      Dealer       Manual
4      swift    Diesel      Dealer       Manual
..       ...       ...         ...          ...
259    amaze    Petrol      Dealer       Manual
263     jazz    Petrol      Dealer       Manual
275     city    Petrol      Dealer    Automatic
285     jazz    Petrol      Dealer    Automatic
287    amaze    Petrol      Dealer    Automatic

[135 rows x 4 columns]
```

```
# Count the occurrences of unique combinations
combination_counts = data.groupby(['Car_Name', 'Fuel_Type', 'Seller_Type', 'Transmission']).size().rese

# Display combinations in ascending order of count
ascending_order = combination_counts.sort_values(by='Count', ascending=True)
print("Combinations in Ascending Order:")
print(ascending_order)

# Display combinations in descending order of count
descending_order = combination_counts.sort_values(by='Count', ascending=False)
print("\nCombinations in Descending Order:")
print(descending_order)
```

```
Combinations in Ascending Order:
                     Car_Name Fuel_Type Seller_Type Transmission  Count
0                         800    Petrol  Individual       Manual      1
86                    elantra    Petrol      Dealer    Automatic      1
85                    elantra    Diesel      Dealer       Manual      1
82                      creta    Petrol      Dealer       Manual      1
77                    corolla    Petrol      Dealer    Automatic      1
..                        ...       ...         ...          ...    ...
46  Royal Enfield Classic 350    Petrol  Individual       Manual      7
97                   fortuner    Diesel      Dealer    Automatic      8
68                       brio    Petrol      Dealer       Manual      9
80              corolla altis    Petrol      Dealer       Manual     11
76                       city    Petrol      Dealer       Manual     19

[135 rows x 5 columns]
```

```
Combinations in Descending Order:
                     Car_Name Fuel_Type Seller_Type Transmission  Count
76                       city    Petrol      Dealer       Manual     19
80              corolla altis    Petrol      Dealer       Manual     11
68                       brio    Petrol      Dealer       Manual      9
97                   fortuner    Diesel      Dealer    Automatic      8
130                     verna    Petrol      Dealer       Manual      7
..                        ...       ...         ...          ...    ...
45   Royal Enfield Bullet 350    Petrol  Individual       Manual      1
44        Mahindra Mojo XT300    Petrol  Individual       Manual      1
43                  KTM RC390    Petrol  Individual       Manual      1
41              KTM 390 Duke     Petrol  Individual       Manual      1
67                       brio    Petrol      Dealer    Automatic      1

[135 rows x 5 columns]
```

```
# 9. Check for missing values
missing_values = data.isnull().sum().sort_values(ascending=False)
print("9. Missing values in the dataset:")
print(missing_values)
```

```
9. Missing values in the dataset:
Car_Name         0
Year             0
Selling_Price    0
Present_Price    0
Kms_Driven       0
Fuel_Type        0
Seller_Type      0
Transmission     0
Owner            0
dtype: int64
```

```
# 10. Replace missing values with the most repeated value in each column

# Iterate through each column with missing values
for column in missing_values.index:
    # Find the most common (mode) value in the column
    most_common_value = data[column].mode()[0]

    # Fill missing values in the column with the most common value
    data[column].fillna(most_common_value, inplace=True)

# Check if missing values were replaced successfully
missing_values_after_replace = data.isnull().sum().sum()
print("\n10. Missing values after replacement:", missing_values_after_replace)
```

```
10. Missing values after replacement: 0
```

```
data.drop_duplicates(inplace=True)
print("\n11. Dataset after removing duplicates:")
```

```
11. Dataset after removing duplicates:
```

```python
# 12. Replace values in attributes (Fuel_Type, Seller_Type, Transmission)
data['Fuel_Type'].replace({"Petrol": 0, "Diesel": 1, "CNG": 2}, inplace=True)
data['Seller_Type'].replace({"Dealer": 0, "Individual": 1}, inplace=True)
data['Transmission'].replace({"Manual": 0, "Automatic": 1}, inplace=True)

# Show the conversion output
print("\n12. Dataset after attribute value conversion:")
print(data[['Fuel_Type', 'Seller_Type', 'Transmission']])
```

```
12. Dataset after attribute value conversion:
     Fuel_Type  Seller_Type  Transmission
0            0            0             0
1            1            0             0
2            0            0             0
3            0            0             0
4            1            0             0
..         ...          ...           ...
296          1            0             0
297          0            0             0
298          0            0             0
299          1            0             0
300          0            0             0

[299 rows x 3 columns]
```

```python
# 13. Add a new 'Age' field based on the 'Year' column
current_year = 2023  # Assuming the current year is 2023
data['Age'] = current_year - data['Year']

# Show the output with the 'Age' field
print("\n13. Dataset with the 'Age' field:")
print(data[['Year', 'Age']])
```

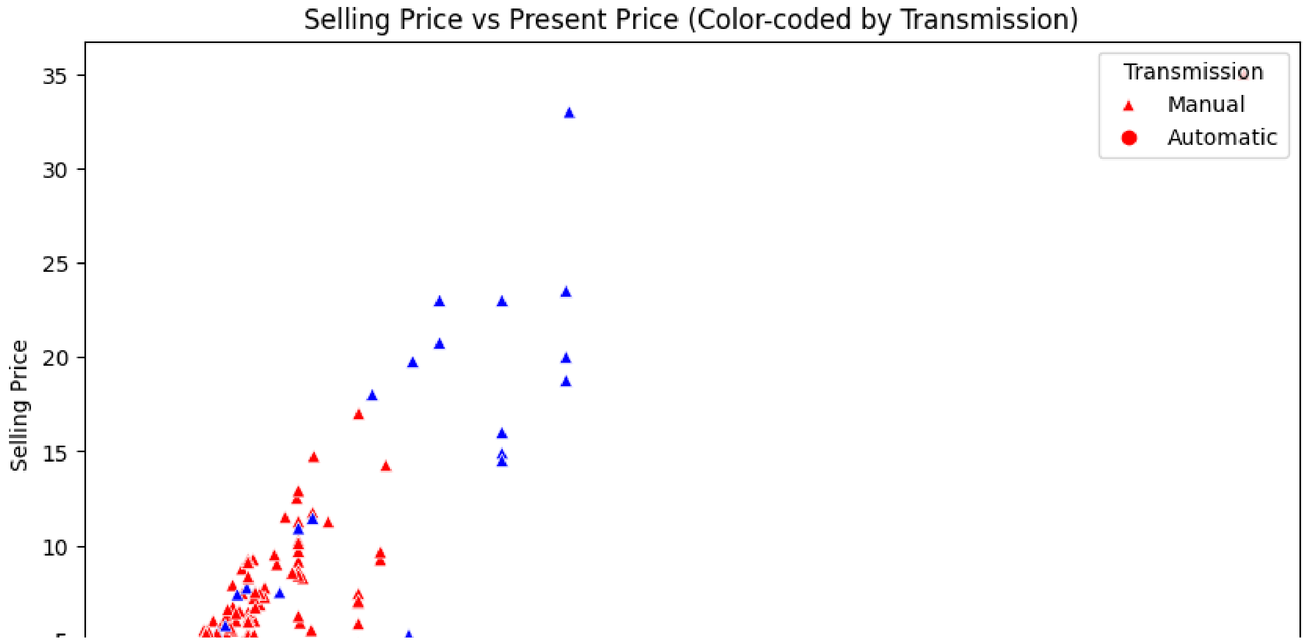
```
13. Dataset with the 'Age' field:
     Year  Age
0    2014    9
1    2013   10
2    2017    6
3    2011   12
4    2014    9
..    ...  ...
296  2016    7
297  2015    8
298  2009   14
299  2017    6
300  2016    7

[299 rows x 2 columns]
```

```python
new_dataset = data[['Car_Name', 'Selling_Price', 'Present_Price', 'Kms_Driven']]

# Show the output of the new dataset
print("\n14. New dataset:")
print(new_dataset)

# 15. Shuffle the rows of the dataset randomly
shuffled_data = data.sample(frac=1, random_state=42)  # random_state for reproducibility

# Show the output of the shuffled dataset
print("\n15. Shuffled dataset:")
print(shuffled_data)
```

```
14. New dataset:
```

```
      Car_Name  Selling_Price  Present_Price  Kms_Driven
0         ritz           3.35           5.59       27000
1          sx4           4.75           9.54       43000
2         ciaz           7.25           9.85        6900
3      wagon r           2.85           4.15        5200
4        swift           4.60           6.87       42450
..         ...            ...            ...         ...
296       city           9.50          11.60       33988
297       brio           4.00           5.90       60000
298       city           3.35          11.00       87934
299       city          11.50          12.50        9000
300       brio           5.30           5.90        5464

[299 rows x 4 columns]

15. Shuffled dataset:
                      Car_Name  Year  Selling_Price  Present_Price  \
283                       city  2016           8.99          11.80
267                       city  2016           8.35           9.40
166            Hero Passion Pro  2016           0.45           0.55
9                         ciaz  2015           7.45           8.92
78               corolla altis  2010           5.25          22.83
..                         ...   ...            ...            ...
190            Bajaj Pulsar 150  2008           0.20           0.75
72               corolla altis  2013           7.45          18.61
108  Royal Enfield Thunder 350  2016           1.20           1.50
272                       city  2015           7.50          10.00
104  Royal Enfield Classic 350  2017           1.35           1.47

     Kms_Driven  Fuel_Type  Seller_Type  Transmission  Owner  Age
283        9010          0            0             0      0    7
267       19434          1            0             0      0    7
166        1000          0            1             0      0    7
9         42367          1            0             0      0    8
78        80000          0            0             1      0   13
..          ...        ...          ...           ...    ...  ...
190       60000          0            1             0      0   15
72        56001          0            0             0      0   10
108       18000          0            1             0      0    7
272       27600          0            0             0      0    8
104        4100          0            1             0      0    6

[299 rows x 10 columns]
```

```
# a. Create the scatter plot with color-coding
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Present_Price', y='Selling_Price', data=data, hue='Transmission', palette={0: 'red',

# Add labels, title, and legend
plt.xlabel('Present Price')
plt.ylabel('Selling Price')
plt.title('Selling Price vs Present Price (Color-coded by Transmission)')
plt.legend(title='Transmission', loc='upper right', labels=['Manual', 'Automatic'])

plt.show()


```



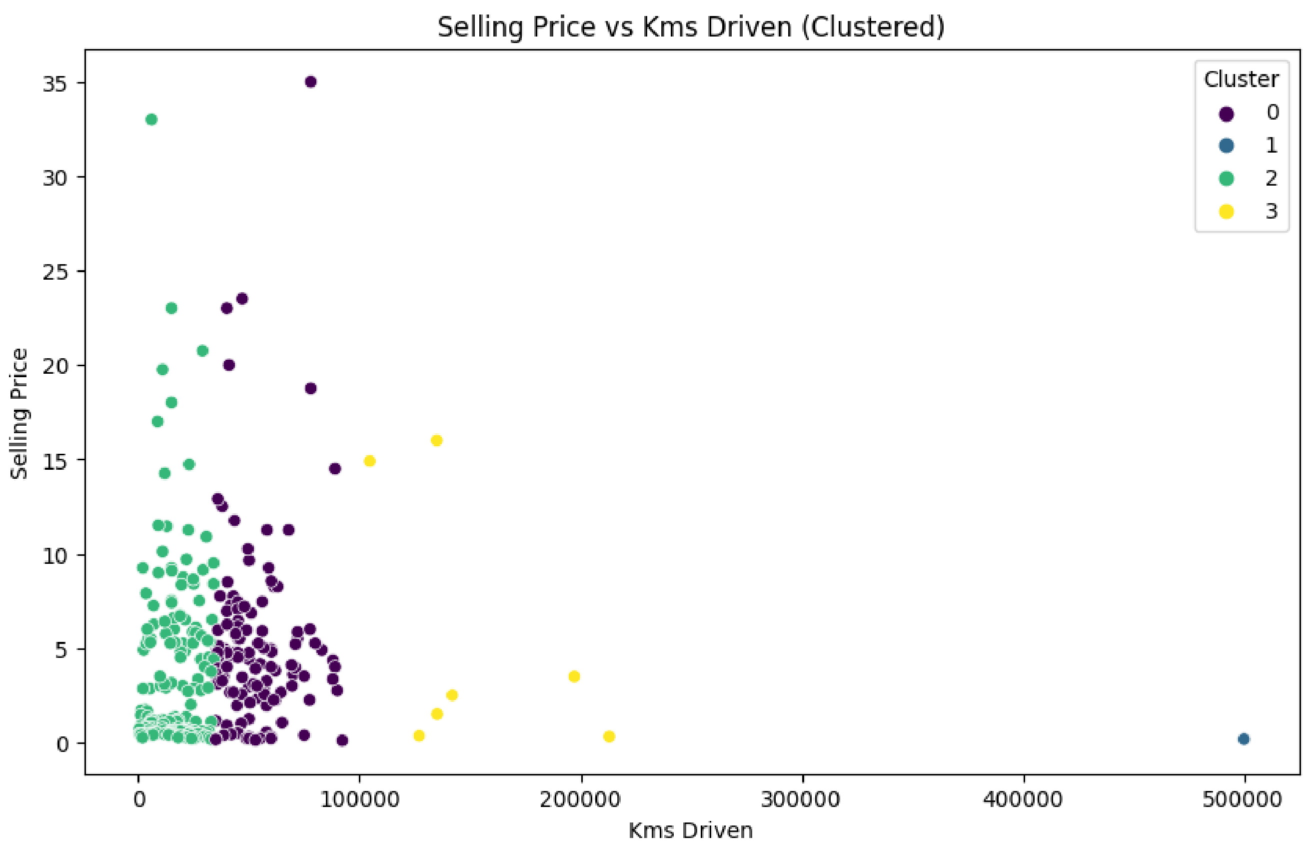
c. What do you understand from the output:

This scatter plot visualizes the relationship between Selling Price and Present Price of vehicles. The points are color-coded based on the transmission type, with red representing manual transmission (0) and blue representing automatic transmission (1). We can observe that vehicles with manual transmission tend to have a wider range of selling prices, while those with automatic transmission are more clustered in the lower price range.

```
# 17. Box plot of Selling_Price Vs Transmission and Fuel_Type
plt.figure(figsize=(12, 6))
sns.boxplot(x='Transmission', y='Selling_Price', hue='Fuel_Type', data=data)
plt.xlabel('Transmission')
plt.ylabel('Selling Price')
plt.title('Box Plot of Selling Price vs Transmission and Fuel Type')
plt.legend(title='Fuel Type', loc='upper right')
plt.show()
```

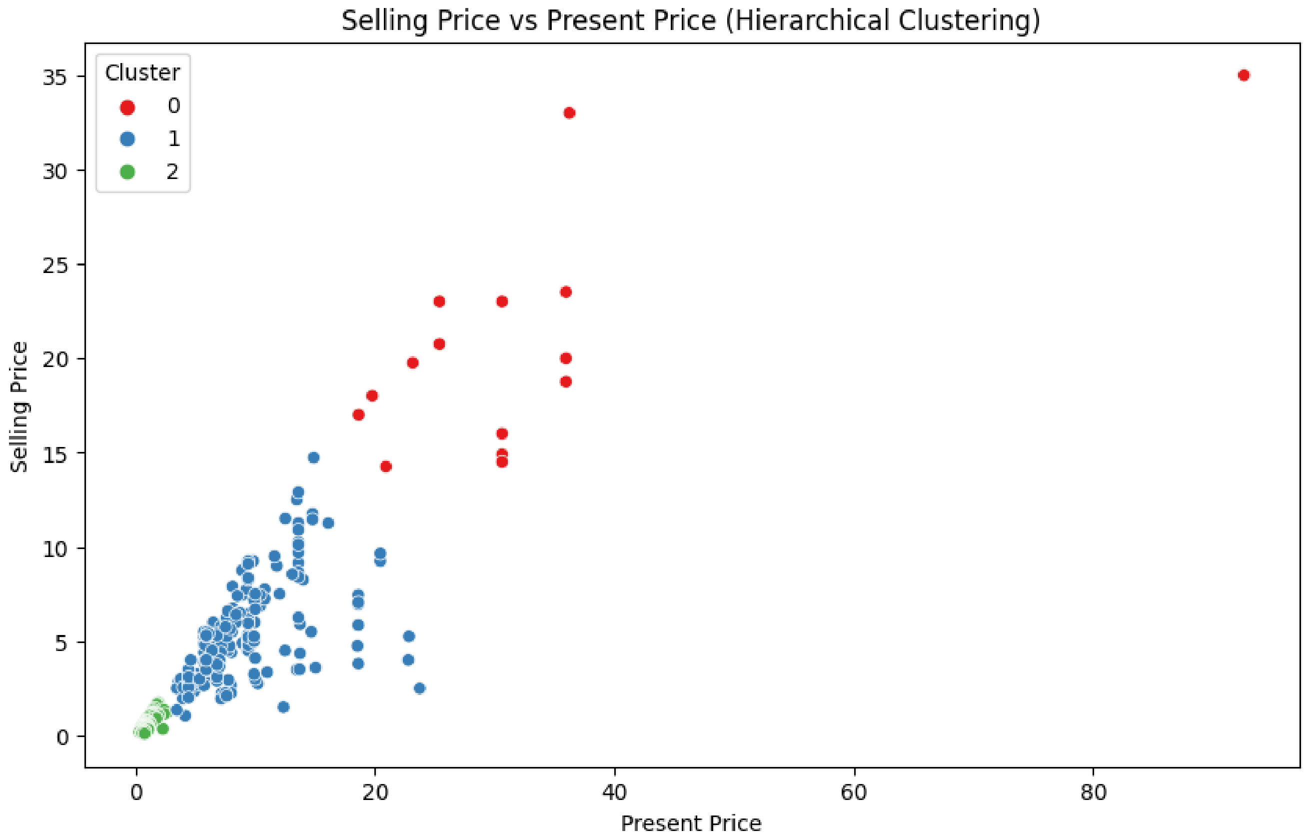
```
# 18. Scatter plot of Selling_Price Vs Kms_Driven with k-means clustering (4 clusters)
from sklearn.cluster import KMeans

# Select the features for clustering
features = data[['Selling_Price', 'Kms_Driven']]

# Apply k-means clustering
kmeans = KMeans(n_clusters=4, random_state=0)
data['Cluster'] = kmeans.fit_predict(features)

# Create a scatter plot with color-coded clusters
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Kms_Driven', y='Selling_Price', data=data, hue='Cluster', palette='viridis')
plt.xlabel('Kms Driven')
plt.ylabel('Selling Price')
plt.title('Selling Price vs Kms Driven (Clustered)')
plt.legend(title='Cluster')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default va
  warnings.warn(
```

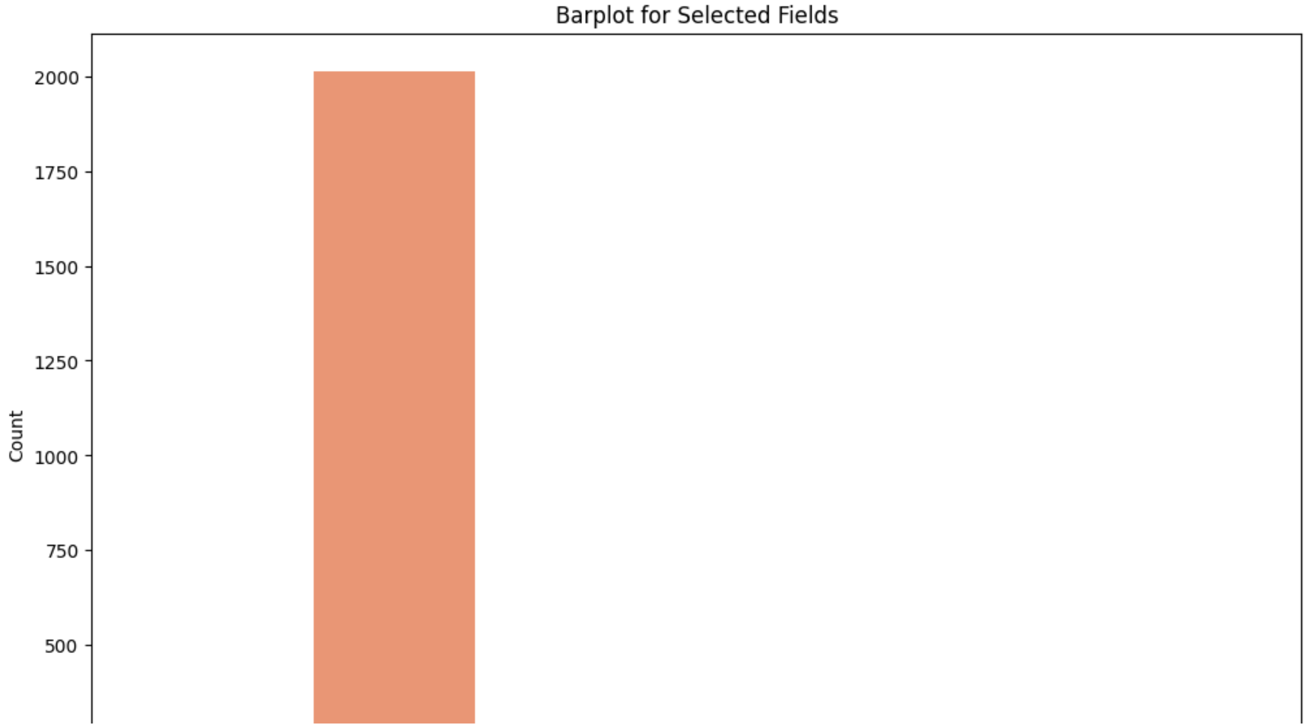



```
# 19. Scatter plot of Selling_Price Vs Present_Price with hierarchical clustering (3 clusters)
from sklearn.cluster import AgglomerativeClustering

# Select the features for clustering
features = data[['Selling_Price', 'Present_Price']]

# Apply hierarchical clustering
hierarchical_clustering = AgglomerativeClustering(n_clusters=3)
data['Hierarchical_Cluster'] = hierarchical_clustering.fit_predict(features)

# Create a scatter plot with color-coded clusters
```

```
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Present_Price', y='Selling_Price', data=data, hue='Hierarchical_Cluster', palette='S
plt.xlabel('Present Price')
plt.ylabel('Selling Price')
plt.title('Selling Price vs Present Price (Hierarchical Clustering)')
plt.legend(title='Cluster')
plt.show()
```



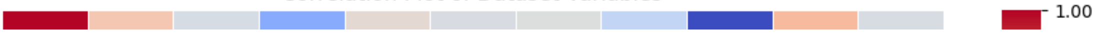

```
# 20. Create a barplot for selected fields with labels, titles, and colors
selected_fields = ['Age', 'Year', 'Transmission', 'Seller_Type', 'Fuel_Type', 'Owner']

# Create a new DataFrame with selected fields
selected_data = data[selected_fields]

# Plot the barplot
plt.figure(figsize=(12, 8))
sns.barplot(data=selected_data, palette='Set2')
plt.xlabel('Attributes')
plt.ylabel('Count')
plt.title('Barplot for Selected Fields')
plt.xticks(rotation=45)
plt.show()
```



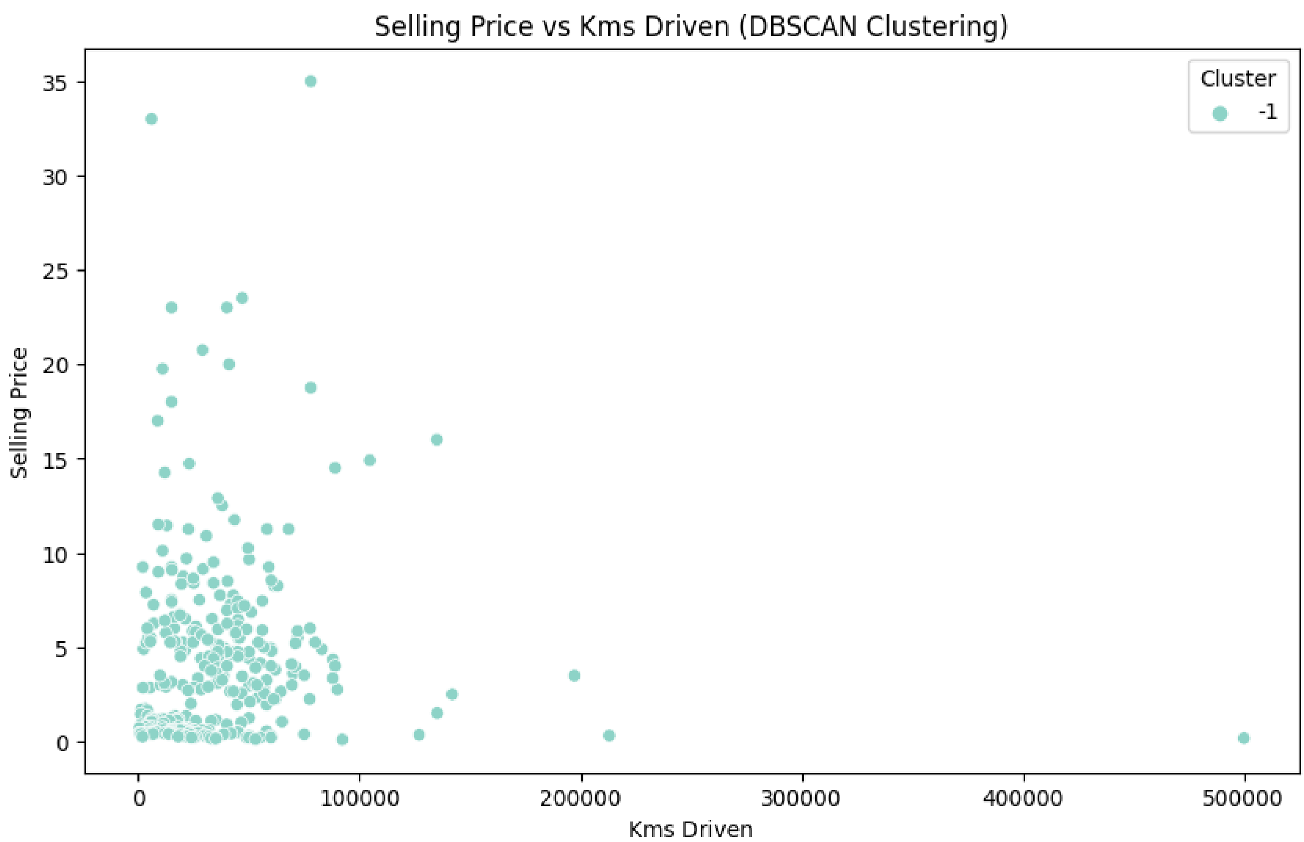

```
# 21. Correlation plot of the whole dataset
corr_matrix = data.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Plot of Dataset Variables')
plt.show()
```

```
<ipython-input-21-a1fb9262dd1e>:2: FutureWarning: The default value of numeric_only in DataFrame.cor
  corr_matrix = data.corr()
```

Correlation Plot of Dataset Variables



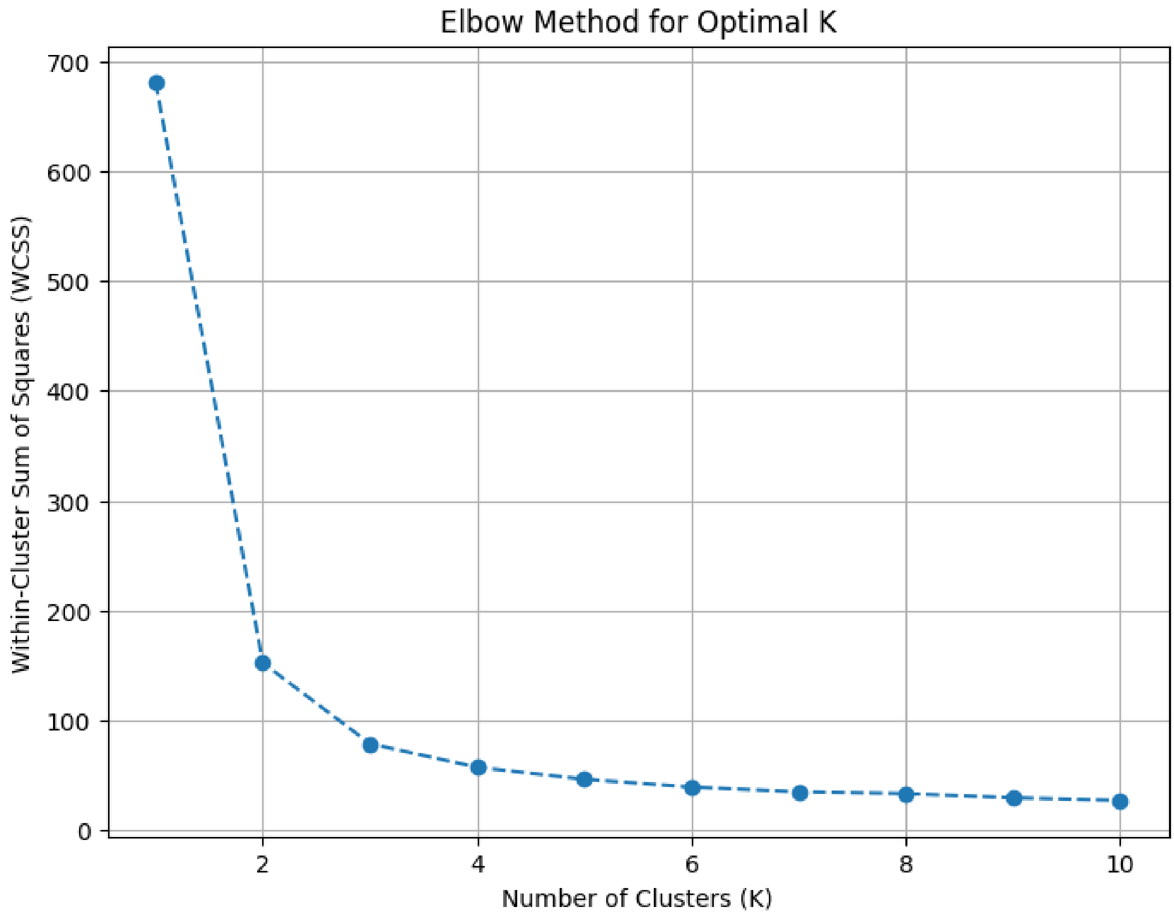

```
# 22. Scatter plot of Selling_Price Vs Kms_Driven with DBSCAN clustering (3 clusters)
from sklearn.cluster import DBSCAN

# Select the features for clustering
features = data[['Selling_Price', 'Kms_Driven']]

# Apply DBSCAN clustering
dbscan = DBSCAN(eps=0.5, min_samples=5)
data['DBSCAN_Cluster'] = dbscan.fit_predict(features)

# Create a scatter plot with color-coded clusters and add a legend
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Kms_Driven', y='Selling_Price', data=data, hue='DBSCAN_Cluster', palette='Set3')
plt.xlabel('Kms Driven')
plt.ylabel('Selling Price')
plt.title('Selling Price vs Kms Driven (DBSCAN Clustering)')
plt.legend(title='Cluster')
plt.show()
```

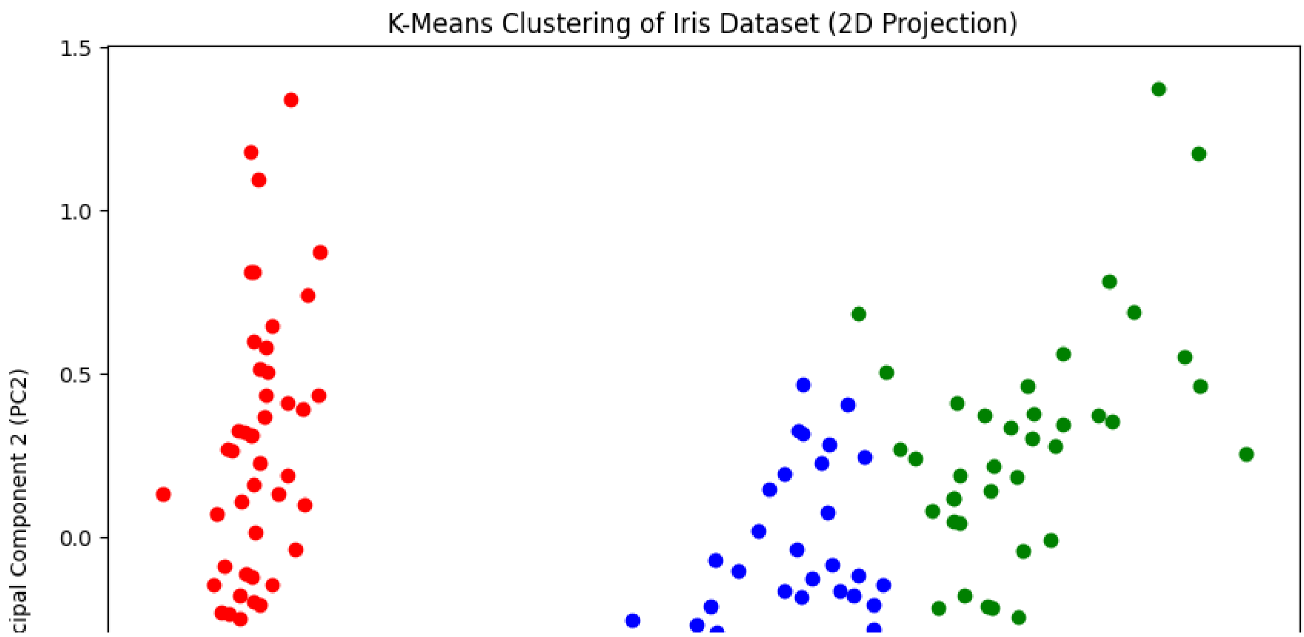


Part B

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
import numpy as np

```

```
# Load the Iris dataset
iris = load_iris()
X = iris.data  # Features (sepal length, sepal width, petal length, petal width)

# Find the optimal number of clusters (K) using the elbow method
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init='auto', random_state=0)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

# Plot the elbow graph to find the optimal K value
plt.figure(figsize=(8, 6))
plt.plot(range(1, 11), wcss, marker='o', linestyle='--')
plt.title('Elbow Method for Optimal K')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Within-Cluster Sum of Squares (WCSS)')
plt.grid()
plt.show()

# Based on the elbow method, choose the optimal K value (e.g., K=3)
optimal_k = 3

# Perform K-Means clustering with the optimal K value
kmeans = KMeans(n_clusters=optimal_k, init='k-means++', max_iter=300, n_init='auto', random_state=0)
kmeans.fit(X)

# Reduce dimensionality to 2D for visualization
pca = PCA(n_components=2)
X_2d = pca.fit_transform(X)

# Add cluster labels to the data
iris_df = pd.DataFrame(X_2d, columns=['PC1', 'PC2'])
iris_df['Cluster'] = kmeans.labels_

# Plot the clusters in 2D
plt.figure(figsize=(10, 8))
colors = ['r', 'g', 'b']
for cluster in range(optimal_k):
    cluster_data = iris_df[iris_df['Cluster'] == cluster]
    plt.scatter(cluster_data['PC1'], cluster_data['PC2'], c=colors[cluster], label=f'Cluster {cluster}'

plt.title('K-Means Clustering of Iris Dataset (2D Projection)')
plt.xlabel('Principal Component 1 (PC1)')
plt.ylabel('Principal Component 2 (PC2)')
plt.legend()
plt.show()
```

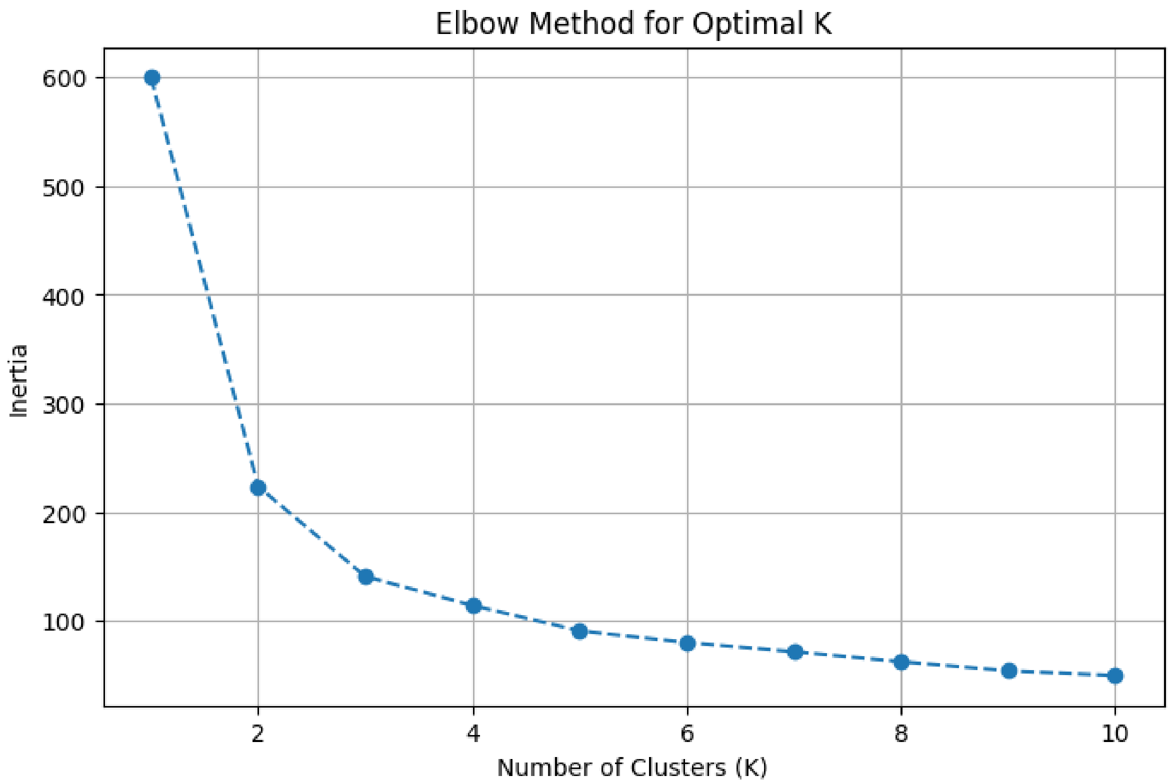




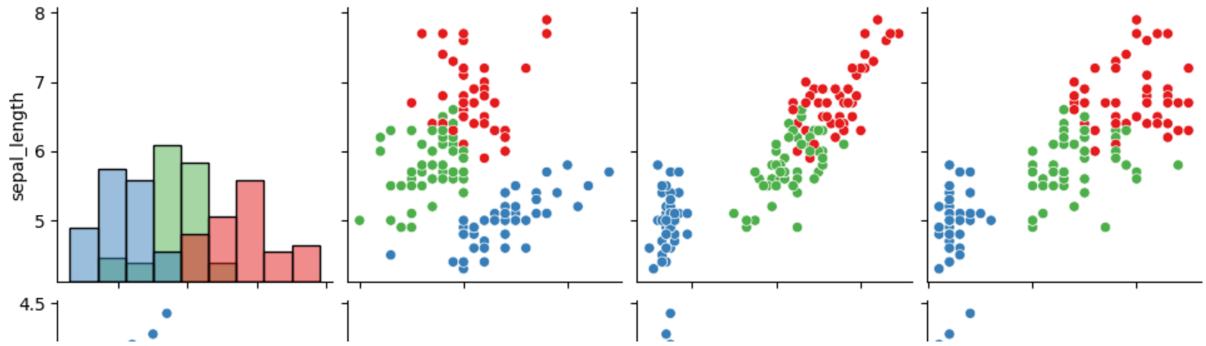

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Load the Iris flower dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
column_names = ["sepal_length", "sepal_width", "petal_length", "petal_width", "species"]
data = pd.read_csv(url, header=None, names=column_names)

# Explore the data
print(data.head())
```

```

# Remove the 'species' column
X = data.drop('species', axis=1)

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Choose the number of clusters (K) using the Elbow method
inertia = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

# Plot the Elbow method results
plt.figure(figsize=(8, 5))
plt.plot(range(1, 11), inertia, marker='o', linestyle='--')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal K')
plt.grid(True)
plt.show()

# From the Elbow method, it seems K=3 is a good choice
kmeans = KMeans(n_clusters=3, random_state=42)
data['cluster'] = kmeans.fit_predict(X_scaled)

# Visualize the clustered data
sns.pairplot(data=data, hue='cluster', palette='Set1', diag_kind='hist')
plt.show()

```

```
   sepal_length  sepal_width  petal_length  petal_width      species
0           5.1          3.5           1.4          0.2  Iris-setosa
1           4.9          3.0           1.4          0.2  Iris-setosa
2           4.7          3.2           1.3          0.2  Iris-setosa
3           4.6          3.1           1.5          0.2  Iris-setosa
4           5.0          3.6           1.4          0.2  Iris-setosa
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default va
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default va
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default va
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default va
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default va
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default va
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default va
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default va
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default va
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default va
  warnings.warn(
```




```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default va
  warnings.warn(
```




```
1 # Import necessary libraries
2 import pandas as pd
```

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Load the breast cancer dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/breast-cancer-
column_names = ["id", "clump_thickness", "uniformity_cell_size", "uniformity_cell_shape", "marginal_adh
data = pd.read_csv(url, header=None, names=column_names)

# Data preprocessing
# Replace missing values represented as '?' with NaN and convert 'bare_nuclei' to numeric
data['bare_nuclei'] = pd.to_numeric(data['bare_nuclei'], errors='coerce')
data = data.dropna()  # Drop rows with missing values

# Map class labels to 0 (benign) and 1 (malignant)
data['class'] = data['class'].map({2: 0, 4: 1})

# Split the data into features (X) and target (y)
X = data.drop(['id', 'class'], axis=1)
y = data['class']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train the KNN classifier
knn = KNeighborsClassifier(n_neighbors=5)  # You can choose the number of neighbors (K) as needed
knn.fit(X_train, y_train)

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

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
classification_report_str = classification_report(y_test, y_pred)

# Print the evaluation results
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", confusion)
print("Classification Report:\n", classification_report_str)

```

```
Accuracy: 0.9560975609756097
Confusion Matrix:
 [[125   2]
 [  7  71]]
Classification Report:
               precision    recall  f1-score   support

           0       0.95      0.98      0.97       127
           1       0.97      0.91      0.94        78

    accuracy                           0.96       205
   macro avg       0.96      0.95      0.95       205
weighted avg       0.96      0.96      0.96       205
```