Questions Pdf

Q1. Decisionn Tree – iris data set

```
from sklearn.datasets import load_iris

from sklearn.tree import DecisionTreeClassifier, export_text, plot_tree

import matplotlib.pyplot as plt

# Load Iris dataset

iris = load_iris()

X, y = iris.data, iris.target

# Fit Decision Tree model

clf = DecisionTreeClassifier(criterion='gini', random_state=0)

clf.fit(X, y)

# Plot the tree

plt.figure(figsize=(12, 8))

plot_tree(clf, filled=True, feature_names=iris.feature_names, class_names=iris.target_names)

plt.title("Decision Tree on Iris Dataset")

plt.show()

# Text-based rules

rules = export_text(clf, feature_names=iris.feature_names)

print("Decision Rules:\n")

print(rules)
```

Q2. K-means clustering – elbow method iris data

```
import pandas as pd

from sklearn.datasets import load_iris

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

import seaborn as sns

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

scaler = StandardScaler()
```

```
data_scaled = scaler.fit_transform(df)
inertias = []
k_range = range(1, 11)
for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(data_scaled)
    inertias.append(kmeans.inertia_)
plt.figure(figsize=(8, 5))
plt.plot(k_range, inertias, 'bo-')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Method to Find Optimal k')
plt.grid(True)
plt.show()
kmeans = KMeans(n_clusters=3, random_state=42)
clusters = kmeans.fit_predict(data_scaled)
df['Cluster'] = clusters
plt.figure(figsize=(8, 5))
sns.scatterplot(x=df.iloc[:, 0], y=df.iloc[:, 1], hue=df['Cluster'], palette='viridis')
plt.title('K-Means Clustering on Iris Dataset')
plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[1])
plt.legend(title='Cluster')
plt.grid(True)
plt.show()
print("\nCluster Centers (in standardized values):")
print(kmeans.cluster_centers_)
```

Q.3 K-means clustering – silhoutee method iris data

```
import pandas as pd

from sklearn.datasets import load_iris

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette_score

import matplotlib.pyplot as plt

import seaborn as sns

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

scaler = StandardScaler()

data_scaled = scaler.fit_transform(df)

silhouette_scores = []

k_values = range(2, 11)

for k in k_values:

    kmeans = KMeans(n_clusters=k, random_state=42)

    labels = kmeans.fit_predict(data_scaled)

    score = silhouette_score(data_scaled, labels)

    silhouette_scores.append(score)

plt.figure(figsize=(8, 5))

plt.plot(k_values, silhouette_scores, 'bo-')

plt.xlabel('Number of Clusters (k)')

plt.ylabel('Silhouette Score')

plt.title('Silhouette Analysis for Optimal k')

plt.grid(True)

plt.show()

best_k = k_values[silhouette_scores.index(max(silhouette_scores))]

print(f"\nBest number of clusters (k) based on silhouette score: {best_k}")

kmeans = KMeans(n_clusters=best_k, random_state=42)

df['Cluster'] = kmeans.fit_predict(data_scaled)

plt.figure(figsize=(8, 5))
```

```
sns.scatterplot(x=df.iloc[:, 0], y=df.iloc[:, 1], hue=df['Cluster'], palette='Set2')
plt.title('K-Means Clustering on Iris Dataset')
plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[1])
plt.legend(title='Cluster')
plt.grid(True)
plt.show()
print("\nCluster centers (in scaled space):")
print(kmeans.cluster_centers_)
```

Q.4 A)One sampled T test- student score, mean=70

```
import scipy.stats as stats
import numpy as np
scores = np.array([72, 88, 64, 74, 67, 79, 85, 75, 89, 77])
mu = 70
t_stat, p_value = stats.ttest_1samp(scores, mu)
print("Sample Mean:", np.mean(scores))
print("T-Statistic:", t_stat)
print("P-Value:", p_value)
alpha = 0.05
if p_value < alpha:
   print("\nConclusion: Reject the Null Hypothesis (Significant difference from 70)")
else:
   print("\nConclusion: Fail to Reject the Null Hypothesis (No significant difference from 70)")
```

B) Feature Scaling – BostonHousing dataset

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler, StandardScaler
df = pd.read_csv('BostonHousing.csv')
df = df[['rm', 'lstat']]
```

```
print("Original DataFrame:")
print(df.head())
minmax_scaler = MinMaxScaler()
df_minmax = df.copy()
df_minmax[['rm', 'lstat']] = minmax_scaler.fit_transform(df_minmax[['rm', 'lstat']])
print("\nDataFrame after Min-Max Scaling:")
print(df_minmax.head()
standard_scaler = StandardScaler()
df_standard = df.copy()
df_standard[['rm', 'lstat']] = standard_scaler.fit_transform(df_standard[['rm', 'lstat']])
print("\nDataFrame after Standard Scaling:")
print(df_standard.head())
```

Q5.A) Chi – squared test for employe aptitude and job proficiency

```
import pandas as pd
import scipy.stats as stats
# Given scores
aptitude = [85, 65, 50, 68, 87, 74, 65, 96, 68, 94, 73, 84, 85, 87, 91]
jobprof  = [70, 90, 80, 89, 88, 86, 78, 67, 86, 90, 92, 94, 99, 93, 87]
# Discretize into categories (optional but typical for Chi-Square test)
df = pd.DataFrame({'aptitude': aptitude, 'jobprof': jobprof})
# Convert to categories (low/medium/high)
df['aptitude_cat'] = pd.cut(df['aptitude'], bins=[0, 70, 85, 100], labels=['Low', 'Medium', 'High'])
df['jobprof_cat']  = pd.cut(df['jobprof'], bins=[0, 70, 85, 100], labels=['Low', 'Medium', 'High'])
# Create contingency table
contingency_table = pd.crosstab(df['aptitude_cat'], df['jobprof_cat'])
print("Contingency Table:\n", contingency_table)
# Apply Chi-Square Test
chi2, p, dof, expected = stats.chi2_contingency(contingency_table)
print("\nChi2 Statistic:", chi2)
print("Degrees of Freedom:", dof)
```

```
print("P-value:", p)
print("Expected Frequencies:\n", expected)
# Conclusion
alpha = 0.05
if p < alpha:
    print("\nConclusion: Reject Null Hypothesis. Significant association between aptitude and job proficiency.")
else:
    print("\nConclusion: Fail to Reject Null Hypothesis. No significant association found.")
```

B) Logistic regression on iris data

```
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
# Load Iris dataset
iris = load_iris()
X = iris.data
y = (iris.target == 0).astype(int)  # 1 if Setosa, 0 otherwise
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
# Train Logistic Regression
model = LogisticRegression()
model.fit(X_train, y_train)
# Predictions
y_pred = model.predict(X_test)
# Evaluation
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

Q6.Mutiple Linear Regression – housing dataset

```
import pandas as pd

from sklearn.linear_model import LinearRegression

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error, r2_score

data = {

    'Bedrooms': [3, 3, 2, 3, 3, 3, 4, 3, 3, 3],

    'Bathrooms': [1, 2.25, 1, 3, 2, 4.5, 2.25, 1.5, 1, 2.5],

    'Sqft_living': [1180, 2570, 770, 1960, 1680, 5420, 1715, 1060, 1780, 1890],

    'Floors': [1, 2, 1, 1, 1, 1, 2, 1, 1, 2],

    'Grade': [7, 7, 6, 7, 8, 11, 7, 7, 7, 7],

    'Sqft_above': [1180, 2170, 770, 1050, 1680, 3890, 1715, 1060, 1050, 1890],

    'Sqft_basement': [0, 400, 0, 910, 0, 1530, 0, 0, 730, 0],

    'Price': [221900, 538000, 180000, 604000, 510000, 267800, 257500, 291850, 229500, 323000]

}

df = pd.DataFrame(data)

X = df.drop('Price', axis=1)

y = df['Price']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Model Training

model = LinearRegression()

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)

print("Mean Squared Error (MSE):", mse)

print("R-squared (R²) Score:", r2)

print("\nPredicted vs Actual Prices:\n")

for actual, predicted in zip(y_test, y_pred):

    print(f"Actual: {actual}, Predicted: {int(predicted)}")
```

Q7. A) Feature Scaling For CarDetails or CarData

```
import pandas as pd

from sklearn.preprocessing import MinMaxScaler, StandardScaler

df = pd.read_csv('CarDetails.csv')

print("Original DataFrame:")

print(df.head())

num_cols = ['Sell Price', 'Buy Price', 'Profit']

minmax_scaler = MinMaxScaler()

df_minmax = df.copy()

df_minmax[num_cols] = minmax_scaler.fit_transform(df_minmax[num_cols])

print("\nDataFrame after Min-Max Scaling:")

print(df_minmax.head())

standard_scaler = StandardScaler()

df_standard = df.copy()

df_standard[num_cols] = standard_scaler.fit_transform(df_standard[num_cols])

print("\nDataFrame after Standard Scaling:")

print(df_standard.head())
```

B) Multiple Linear regression for Pima Indian diabetes dataset

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, r2_score

df = pd.read_csv("pima_diabetes.csv")  # Replace with your actual filename

print(df.head())

X = df.drop('Outcome', axis=1)  # 'Outcome' is the target (binary, but we'll use it in regression)

y = df['Outcome']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LinearRegression()

model.fit(X_train, y_train)

y_pred = model.predict(X_test)
```

```
mse = mean_squared_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)

print("\nModel Evaluation:")

print(f"Mean Squared Error (MSE): {mse}")

print(f"R-squared (R²): {r2}")

print("\nFirst 10 predictions vs actual values:")

for actual, predicted in zip(y_test[:10], y_pred[:10]):

    print(f"Actual: {actual}, Predicted: {round(predicted, 3)}")
```

Q8. A. Creating Pivot Tables in Excel for Car Dataset Analysis

To create pivot tables for the requested analyses:

1. **Cars by make, model, and color:**
    - Create a pivot table with "Make" and "Model" in rows, "Color" in columns, and count of entries as values.
    - This will show a breakdown of how many cars you have in each category.
2. **Profit margin by make:**
    - Calculate profit margin as (Sell Price - Buy Price)/Sell Price
    - Create a pivot table with "Make" in rows and average of profit margin in values.
3. **Average cost of vehicles:**
    - Create a pivot table with average of "Buy Price" as values.
    - You can also break this down by make or model if needed.
4. **Percentage of cars by color:**
    - Create a pivot table with "Color" in rows and count of entries as values.
    - Calculate the percentage by dividing each color count by the total count.
    -

B) Logistics Regression on iris data

```
import pandas as pd

import numpy as np

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
confusion_matrix
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
iris = load_iris()
X = iris.data
y = iris.target
y_binary = (y > 0).astype(int)
X_train, X_test, y_train, y_test = train_test_split(X, y_binary, test_size=0.3, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
model = LogisticRegression(random_state=42)
model.fit(X_train_scaled, y_train)
y_pred = model.predict(X_test_scaled)
y_pred_prob = model.predict_proba(X_test_scaled)[:, 1]
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
        xticklabels=['Not Setosa', 'Setosa'],
        yticklabels=['Not Setosa', 'Setosa'])
```

```
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
feature_importance = pd.DataFrame({
    'Feature': iris.feature_names,
    'Coefficient': model.coef_[0]
})
feature_importance['Abs_Coefficient'] = abs(feature_importance['Coefficient'])
feature_importance = feature_importance.sort_values('Abs_Coefficient', ascending=False)
print("\nFeature Importance:")
print(feature_importance)
```

Q9. A) Feature Scaling for Country_Data

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler, StandardScaler
df = pd.read_csv('Country_Data.csv')
num_cols = ['Age', 'Salary']
print("Original DataFrame:")
print(df.head(10))
minmax_scaler = MinMaxScaler()
df_minmax = df.copy()
df_minmax[num_cols] = minmax_scaler.fit_transform(df_minmax[num_cols])
print("\nMin-Max Scaled Data (Age & Salary only):")
print(df_minmax[num_cols].head(10))
standard_scaler = StandardScaler()
df_standard = df.copy()
df_standard[num_cols] = standard_scaler.fit_transform(df_standard[num_cols])
print("\nStandardized Data (Age & Salary only):")
print(df_standard[num_cols].head(10))
```

B) PCA for iris data

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
iris = load_iris()
X = iris.data
y = iris.target
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
pca = PCA()
X_pca = pca.fit_transform(X_scaled)
explained_variance_ratio = pca.explained_variance_ratio_
plt.figure(figsize=(8, 6))
plt.plot(np.cumsum(explained_variance_ratio), marker='o', linestyle='--')
plt.title('Explained Variance Ratio')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance Ratio')
plt.grid(True)
plt.show()
n_components = np.argmax(np.cumsum(explained_variance_ratio) >= 0.95) + 1
print(f"Number of principal components to explain 95% variance: {n_components}")
pca = PCA(n_components=n_components)
X_reduced = pca.fit_transform(X_scaled)
plt.figure(figsize=(8, 6))
plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=y, cmap='viridis', s=50, alpha=0.5)
plt.title('Data in Reduced-dimensional Space')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
```

```
plt.colorbar(label='Target')
plt.show()
```

Q10. A) Feature Dummification for Country data

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
df = pd.read_csv("Country_Data.csv")
print("Columns in dataset:", df.columns)
le = LabelEncoder()
df['Country_Code'] = le.fit_transform(df['Country'])
df['Purchased_Code'] = le.fit_transform(df['Purchased'])
print(df.head())
```

B) Logistic Regression on iris same as above code

Q11. A) Transformation Functions on Car data

```
import pandas as pd
import numpy as np
# Read data from the existing CSV file
# Assuming your file is named 'car_data.csv' - adjust the name if needed
car_df = pd.read_csv('car_data.csv')
# Display the loaded data
print("Original Car Data:")
print(car_df)
# 1. Display records of cars having Sell Price greater than 4000
print("\nCars with Sell Price greater than 4000:")
expensive_cars = car_df[car_df['Sell_Price'] > 4000]
print(expensive_cars)
# 2. Sort the car data in ascending order (by Make)
print("\nCars sorted by Make in ascending order:")
sorted_cars = car_df.sort_values(by='Make')
```

```
print(sorted_cars)

# 3. Group the data according to "Make" of car

print("\nCars grouped by Make:")

grouped_cars = car_df.groupby('Make')

# Display summary statistics for each group

group_summary = grouped_cars.agg({

    'Mileage': 'mean',

    'Sell_Price': 'mean',

    'Buy_Price': 'mean',

    'Model': 'count'  # Count of cars for each make

}).rename(columns={'Model': 'Count'})

print(group_summary)

# Additional analysis: Calculate profit for each car

car_df['Profit'] = car_df['Sell_Price'] - car_df['Buy_Price']

print("\nCar data with calculated profit:")

print(car_df[['Make', 'Model', 'Sell_Price', 'Buy_Price', 'Profit']])

# Summary of profit by Make

print("\nAverage profit by Make:")

print(car_df.groupby('Make')['Profit'].mean().sort_values(ascending=False))
```

B) PCA for iris data same as before

Q12.A) One way anova test for exam score of students

```
import scipy.stats as stats

class_A = [85, 90, 88, 82, 87]

class_B = [76, 78, 80, 81, 75]

class_C = [92, 88, 94, 89, 90]

f_statistic, p_value = stats.f_oneway(class_A, class_B, class_C)

print("F-statistic:", f_statistic)

print("P-value:", p_value)

alpha = 0.05
```

```
if p_value < alpha:
    print("Reject the null hypothesis: Significant difference between class means.")
else:
    print("Fail to reject the null hypothesis: No significant difference between class means.")
```

B) PCA for Wine Quality Dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
# Step 1: Load the Wine Quality dataset
data = pd.read_csv('winequality-red.csv', sep=';')
# Step 2: Separate features and target
X = data.drop('quality', axis=1)
y = data['quality']
# Step 3: Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Step 4: Apply PCA
pca = PCA()
X_pca = pca.fit_transform(X_scaled)
# Step 5: Plot explained variance to choose number of components
plt.figure(figsize=(8, 5))
plt.plot(np.cumsum(pca.explained_variance_ratio_), marker='o', color='blue')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('Explained Variance vs. Number of Components')
plt.grid(True)
plt.tight_layout()
```

```
plt.show()
# Optional: Based on the plot, choose 2 components
pca_2d = PCA(n_components=2)
X_2d = pca_2d.fit_transform(X_scaled)
# Step 6: Visualize the data in 2D PCA space
pca_df = pd.DataFrame(data=X_2d, columns=['PC1', 'PC2'])
pca_df['Quality'] = y
plt.figure(figsize=(10, 6))
sns.scatterplot(data=pca_df, x='PC1', y='PC2', hue='Quality', palette='viridis', s=70)
plt.title('Wine Quality Data in 2D PCA Space')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title='Quality')
plt.tight_layout()
plt.show()
```

Q13. A) Two Sampled T test for the difference in time to complete the th task between two groups of employees.

```
from scipy import stats
group1 = [85, 95, 100, 80, 90, 97, 104, 95, 88, 92, 94, 99]
group2 = [83, 85, 96, 92, 100, 104, 94, 95, 88, 90, 93, 94]
t_stat, p_value = stats.ttest_ind(group1, group2)
print("Two-Sample T-Test Results")
print("--------------------------")
print(f"T-Statistic: {t_stat:.3f}")
print(f"P-Value: {p_value:.4f}")
alpha = 0.05
if p_value < alpha:
  print("Conclusion: Reject the Null Hypothesis")
  print("=> There is a significant difference between the two groups.")
else:
```

```
    print("Conclusion: Fail to Reject the Null Hypothesis")
    print("=> There is no significant difference between the two groups.")
```

B) Multiple Linear Regression on the "Pima Indian Diabetes dataset same as before

Q14. Decision Tree for tenis player dataset

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from sklearn import tree
import matplotlib.pyplot as plt
# Data from the image
data = {
    'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain', 'Rain', 'Overcast', 'Sunny', 'Sunny', 'Rain',
                'Sunny', 'Overcast', 'Overcast', 'Rain'],
    'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Strong', 'Strong', 'Strong', 'Weak', 'Strong', 'Weak',
             'Strong', 'Weak', 'Strong', 'Strong'],
    'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'No', 'Yes', 'Yes',
                   'Yes', 'Yes', 'Yes', 'No']
}
# Convert to DataFrame
df = pd.DataFrame(data)
# Label Encoding for categorical features
le_outlook = LabelEncoder()
le_wind = LabelEncoder()
le_play = LabelEncoder()
df['Outlook'] = le_outlook.fit_transform(df['Outlook'])
df['Wind'] = le_wind.fit_transform(df['Wind'])
df['PlayTennis'] = le_play.fit_transform(df['PlayTennis'])
```

```
# Features and Labels
X = df[['Outlook', 'Wind']]
y = df['PlayTennis']
# Train the Decision Tree
model = DecisionTreeClassifier(criterion='entropy', random_state=0)
model.fit(X, y)
# Predict on training data (since no test set provided)
predictions = model.predict(X)
# Accuracy
accuracy = accuracy_score(y, predictions)
print("Model Accuracy: {:.2f}%".format(accuracy * 100))
# Visualize the decision tree
plt.figure(figsize=(10,6))
tree.plot_tree(model, feature_names=['Outlook', 'Wind'], class_names=['No', 'Yes'], filled=True)
plt.show()
```

Q15 A) Implement Decision Tree Model on Titanic dataset using Python/R and interpret decision rules of classification. Perform Linear Regression on the following dataset in Python/R for predicting the weight of the person depending on height

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor, plot_tree
height = [151, 174, 138, 186, 128, 136, 179, 163, 152, 131]
weight = [63, 81, 56, 91, 47, 57, 76, 72, 62, 60]
df = pd.DataFrame({'Height': height, 'Weight': weight})
lin_reg = LinearRegression()
lin_reg.fit(df[['Height']], df['Weight'])
# Predict weight using linear regression
height_input = pd.DataFrame({'Height': [170]})
lin_pred = lin_reg.predict(height_input)
print(f"Linear Regression: Predicted weight for height 170 cm: {lin_pred[0]:.2f} kg")
```

```
tree_reg = DecisionTreeRegressor()

tree_reg.fit(df[['Height']], df['Weight'])

# Predict weight using decision tree

tree_pred = tree_reg.predict(height_input)

print(f"Decision Tree: Predicted weight for height 170 cm: {tree_pred[0]:.2f} kg")

plt.figure(figsize=(10, 6))

# Scatter actual data

plt.scatter(df['Height'], df['Weight'], color='blue', label='Actual Data')

# Plot Linear Regression line

x_range = pd.DataFrame({'Height': range(min(height), max(height)+1)})

plt.plot(x_range, lin_reg.predict(x_range), color='red', label='Linear Regression')

# Plot Decision Tree steps

plt.plot(x_range, tree_reg.predict(x_range), color='orange', linestyle='--', label='Decision Tree
Regression')

# Highlight prediction points

plt.scatter(170, lin_pred[0], color='red', marker='x', s=100, label='LR Prediction (170 cm)')

plt.scatter(170, tree_pred[0], color='orange', marker='o', s=100, label='DT Prediction (170 cm)')

plt.xlabel('Height (cm)')

plt.ylabel('Weight (kg)')

plt.title('Linear vs Decision Tree Regression: Height vs Weight')

plt.legend()

plt.grid(True)

plt.tight_layout()

plt.show()

plt.figure(figsize=(12, 6))

plot_tree(tree_reg, feature_names=['Height'], filled=True, rounded=True)

plt.title("Decision Tree Structure")

plt.show()
```

B) Perform Transformation Functions on CarDetails

```
import pandas as pd

df = pd.read_csv("CarDetails.csv")

if df['Mileage'].dtype == object:

    df['Mileage'] = df['Mileage'].str.replace(',', '').astype(int)

# A. Filter records where Buy Price >= 3000

filtered_df = df[df['Buy Price'] >= 3000]

print("\nA. Cars with Buy Price >= 3000:\n", filtered_df)

# B. Sort by Buy Price (ascending)

sorted_df = df.sort_values(by='Buy Price')

print("\nB. Sorted by Buy Price (ascending):\n", sorted_df)

# C. Group by 'Model' column

grouped = df.groupby('Model').size().reset_index(name='Count')

print("\nC. Grouped by Model:\n", grouped)
```

Q16. A) linear regression for predicting the salary of the person depending on his/her years of experience.

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear_model import LinearRegression

# Data

X = np.array([2, 10, 4, 20, 8, 12, 22]).reshape(-1, 1)  # Years of Experience

y = np.array([30000, 95000, 45000, 178000, 84000, 120000, 200000])  # Salary

# Model

model = LinearRegression()

model.fit(X, y)

# Prediction (Optional: predict for, say, 15 years of experience)

experience = 15

predicted_salary = model.predict(np.array([[experience]]))

print(f"Predicted salary for {experience} years of experience: ${predicted_salary[0]:,.2f}")

# Plot
```

```
plt.scatter(X, y, color='blue', label='Actual Data')
plt.plot(X, model.predict(X), color='red', label='Linear Regression Line')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.title('Salary vs Experience')
plt.legend()
plt.grid(True)
plt.show()
```

B) Vlookup for customerdata like supply id , status

#1. Find the **Part Name** for Part Number **A002**:

Formula:

```excel

=VLOOKUP("A002", B2:D21, 2, FALSE)

# 2. Find the **Supplier ID** for the Part Name **“Ball Joint”**:

**Formula**:

```excel

=VLOOKUP("Ball Joint", C2:A21, 2, FALSE)

#3. Find the **Part Price** for Part Name **“muffler”**:

**Formula**:

```excel

=VLOOKUP("muffler", C2:D21, 2, FALSE)

#4. Find the **Status** of Part Number **A008**:

**Formula**:

```excel

=VLOOKUP("A008", B2:E21, 4, FALSE)

#Steps to Apply VLOOKUP in Excel:

1. Open Excel and select the cell where you want the result.

2. Type `=VLOOKUP(...)` with the appropriate parameters.

3. Press **Enter** to get the result.

4. Make sure to use **FALSE** as the fourth argument to ensure exact match.

5. Adjust the table range to match your actual data location if it differs