**Q1: Data Frame**

1. **Read the dataset**

```
python

Copy code

import pandas as pd


# Load the Iris dataset

url = "https://github.com/neurospin/pystatsml/raw/master/datasets/iris.csv"

df = pd.read_csv(url)
```

2. **Print column names**

```
python

Copy code

print(df.columns.tolist())
```

3. **Get numerical columns**

```
python

Copy code

numerical_columns = df.select_dtypes(include=['float64', 'int64']).columns.tolist()

print(numerical_columns)
```

4. **Compute mean of numerical columns grouped by species**

```
python

Copy code

stats_table = df.groupby("species")[numerical_columns].mean().reset_index()

print(stats_table)
```

---

**Q2: McCulloch-Pitts (M-P) Neuron**

Write a Python program to define the weights and bias for the described M-P neuron:

```
python

Copy code

def mcculloch_pitts(x, y, z):
```

```
weights = [1, 1]  # Equal weights for x and y
bias = -1        # Threshold value

# Weighted sum
weighted_sum = weights[0] * x + weights[1] * y + bias

# Activation logic
if x == -1 and y == 1:
    return z  # z is the output if condition matches
return -1  # Otherwise
```

Example usage:

```
python
Copy code
print(mcculloch_pitts(-1, 1, 1))  # Outputs 1
print(mcculloch_pitts(0, 1, -1))  # Outputs -1
```

---

**Q3: Data Preparation**

1. **Load the heart dataset**

```
python
Copy code
heart_url = "https://www.kaggle.com/zhaoyingzhu/heartcsv/download"
heart_data = pd.read_csv(heart_url)
```

2. **Perform operations**
   a) **Find Shape of Data**

```
python
Copy code
print(heart_data.shape)
```

b) **Find Missing Values**

```
python
Copy code
```

```
print(heart_data.isnull().sum())
```

c) **Find Data Types**

```
python

Copy code

print(heart_data.dtypes)
```

d) **Find Zeros**

```
python

Copy code

zeros_per_column = (heart_data == 0).sum()

print(zeros_per_column)
```

e) **Find Mean Age**

```
python

Copy code

mean_age = heart_data['Age'].mean()

print(mean_age)
```

f) **Extract Specific Columns and Split Data**

```
python

Copy code

selected_columns = ['Age', 'Sex', 'ChestPain', 'RestBP', 'Chol']

subset = heart_data[selected_columns]


# Split into training and testing datasets

train_data = subset.sample(frac=0.75, random_state=42)

test_data = subset.drop(train_data.index)


print(train_data.shape, test_data.shape)
```

**Q4: Matrix and Vector Computations**

The matrix MMM and vector vvv are given as:

M=[123230133],v=[302]M = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 0 \\ 1 & 3 & 3 \end{bmatrix}, \quad v = \begin{bmatrix} 3 \\ 0 \\ 2 \end{bmatrix}M=121233303,v=302

We need to compute:

1. M·vM \cdot vM·v
2. vT·Mv^T \cdot MvT·M
3. vT·vv^T \cdot vvT·v

Using Python and NumPy:

python

Copy code

```
import numpy as np

# Define M and v
M = np.array([[1, 2, 3], [2, 3, 0], [1, 3, 3]])
v = np.array([3, 0, 2])

# Compute M · v
Mv = np.dot(M, v)

# Compute v^T · M
vT_M = np.dot(v.T, M)

# Compute v^T · v
vT_v = np.dot(v.T, v)

# Display results
print("M · v =", Mv)
print("v^T · M =", vT_M)
print("v^T · v =", vT_v)
```

**Results**:

- M·v=[969]M \cdot v = \begin{bmatrix} 9 \\ 6 \\ 9 \end{bmatrix}M·v=969
- vT·M=[91515]v^T \cdot M = \begin{bmatrix} 9 & 15 & 15 \end{bmatrix}vT·M=[91515]
- vT·v=13v^T \cdot v = 13vT·v=13

---

**Q5: Regression Technique**

To analyze the temperature dataset, follow these steps:

1. **Load the dataset**:

```
python

Copy code

import pandas as pd


# Load the temperature data

url = "https://www.kaggle.com/venky73/temperatures-of-india/download"

data = pd.read_csv(url)

print(data.head())
```

2. **Preprocess the data**:
    - Handle missing values (if any).
    - Normalize or standardize the data if necessary.

```
python

Copy code

# Check for missing values

print(data.isnull().sum())


# Fill missing values (if any)

data.fillna(data.mean(), inplace=True)
```

3. **Fit a regression model**:

```
python

Copy code

from sklearn.linear_model import LinearRegression

from sklearn.model_selection import train_test_split


# Define features (X) and target (y)

X = data[['Month']]  # Example feature
```

```
y = data['Temperature']  # Target variable


# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)


# Fit the regression model
model = LinearRegression()
model.fit(X_train, y_train)


# Predict and evaluate
predictions = model.predict(X_test)
print("Model Coefficients:", model.coef_)
print("Model Intercept:", model.intercept_)
```

4. **Plot results (optional)**:

```
python
Copy code
import matplotlib.pyplot as plt


plt.scatter(X_test, y_test, color='blue', label='Actual')
plt.plot(X_test, predictions, color='red', label='Predicted')
plt.xlabel('Month')
plt.ylabel('Temperature')
plt.legend()
plt.show()
```

**Q6. Classification Technique (Decision Tree for Graduate Admissions)**

**A. Data Preprocessing**

```
python
Copy code
import pandas as pd
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import LabelEncoder

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score, classification_report


# Load Dataset

url = 'https://www.kaggle.com/mohansacharya/graduate-admissions'

data = pd.read_csv(url)


# Data Cleaning & Transformation

data['Admitted'] = data['Admitted'].apply(lambda x: 1 if x == 'yes' else 0)  # Adjust target column

X = data.drop('Admitted', axis=1)

y = data['Admitted']
```

**B. Data Preparation (Train-Test Split)**

```
python

Copy code

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

**C. Apply Machine Learning Algorithm**

```
python

Copy code

# Train Decision Tree Classifier

clf = DecisionTreeClassifier(random_state=42)

clf.fit(X_train, y_train)


# Make Predictions

y_pred = clf.predict(X_test)
```

**D. Model Evaluation**

```
python

Copy code

# Evaluate Model

accuracy = accuracy_score(y_test, y_pred)
```

```
print("Accuracy:", accuracy)
print(classification_report(y_test, y_pred))
```

---

**Q7. Clustering Techniques (Mall Customers Dataset)**

**Load Dataset and Preprocess**

```
python
Copy code
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Load Dataset
url = 'https://www.kaggle.com/shwetabh123/mall-customers'
data = pd.read_csv(url)

# Select Relevant Features
features = data[['Annual Income', 'Spending Score']]
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)
```

**Apply K-Means Clustering**

```
python
Copy code
# Apply K-Means
kmeans = KMeans(n_clusters=3, random_state=42)
data['Cluster'] = kmeans.fit_predict(scaled_features)

# Visualize Clusters
plt.figure(figsize=(10, 6))
for cluster in range(3):
    plt.scatter(features.iloc[data['Cluster'] == cluster, 0],
```

```
        features.iloc[data['Cluster'] == cluster, 1], label=f'Cluster {cluster}')

plt.title('K-Means Clustering of Mall Customers')
plt.xlabel('Annual Income')
plt.ylabel('Spending Score')
plt.legend()
plt.show()
```

---

**Q5. Linear Regression for Month-Wise Temperature**

**A. Apply Linear Regression**

```
python
Copy code
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Assuming `temperature_data` is loaded
X = temperature_data[['Month']]  # Features
y = temperature_data['Temperature']  # Target

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train Model
lr = LinearRegression()
lr.fit(X_train, y_train)

# Predictions
y_pred = lr.predict(X_test)
```

**B. Performance Assessment**

```
python

Copy code

# Evaluate Model

mse = mean_squared_error(y_test, y_pred)

mae = mean_absolute_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)


print(f"MSE: {mse}, MAE: {mae}, R2: {r2}")
```

**C. Visualization**

```
python

Copy code

# Plot Regression Line

plt.scatter(X_test, y_test, color='blue', label='Actual')

plt.plot(X_test, y_pred, color='red', label='Predicted')

plt.title('Month-Wise Temperature Prediction')

plt.xlabel('Month')

plt.ylabel('Temperature')

plt.legend()

plt.show()
```

**Q8. Association Rule Learning (Market Basket Optimization)**

**A. Data Preprocessing**

```
python

Copy code

import pandas as pd

from mlxtend.frequent_patterns import apriori, association_rules


# Load Dataset

dataset = pd.read_csv('https://www.kaggle.com/hemantkumar05/market-basket-optimization',
header=None)
```

```
# Convert Dataset into a Transaction List
transactions = []
for i in range(0, len(dataset)):
    transactions.append([str(dataset.values[i, j]) for j in range(0, dataset.shape[1]) if
str(dataset.values[i, j]) != 'nan'])
```

**B. Generate List of Transactions**

```
python
Copy code
from mlxtend.preprocessing import TransactionEncoder


# Transform Transaction Data
te = TransactionEncoder()
te_array = te.fit(transactions).transform(transactions)
df = pd.DataFrame(te_array, columns=te.columns_)
```

**C. Train Apriori Algorithm**

```
python
Copy code
# Apply Apriori
frequent_itemsets = apriori(df, min_support=0.01, use_colnames=True)


# Generate Association Rules
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)
```

**D. Visualize the Rules**

```
python
Copy code
# Print and Visualize Rules
print(rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']])
```

---

**Q9. Clustering Techniques (Mall Customers Dataset)**

**A. Apply Preprocessing**

```
python

Copy code

from sklearn.preprocessing import StandardScaler


# Load Dataset

data = pd.read_csv('https://www.kaggle.com/shwetabh123/mall-customers')


# Select Spending Score for Clustering

X = data[['Spending Score']]

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)
```

**B. Perform Train-Test Split**

(Not required for unsupervised learning; we use the entire dataset.)

**C. Apply Machine Learning Algorithms**

**1. K-Means Clustering**

```
python

Copy code

from sklearn.cluster import KMeans


# Apply K-Means

kmeans = KMeans(n_clusters=3, random_state=42)

data['KMeans_Cluster'] = kmeans.fit_predict(X_scaled)
```

**2. Hierarchical Clustering**

```
python

Copy code

from scipy.cluster.hierarchy import linkage, dendrogram, fcluster

import matplotlib.pyplot as plt
```

```
# Apply Hierarchical Clustering

Z = linkage(X_scaled, method='ward')

dendrogram(Z)

plt.title('Dendrogram for Hierarchical Clustering')

plt.show()


# Assign Clusters

data['Hierarchical_Cluster'] = fcluster(Z, t=3, criterion='maxclust')
```

**D. Evaluate Model**

```
python

Copy code

# Visualize Clusters (K-Means Example)

plt.scatter(X_scaled[data['KMeans_Cluster'] == 0], [0] * len(X_scaled[data['KMeans_Cluster'] == 0]), label='Cluster 0')

plt.scatter(X_scaled[data['KMeans_Cluster'] == 1], [0] * len(X_scaled[data['KMeans_Cluster'] == 1]), label='Cluster 1')

plt.scatter(X_scaled[data['KMeans_Cluster'] == 2], [0] * len(X_scaled[data['KMeans_Cluster'] == 2]), label='Cluster 2')

plt.legend()

plt.show()
```

**E. Apply Cross-Validation**

Cross-validation is typically not applicable for clustering. Instead, clustering performance can be evaluated using metrics like the silhouette score:

```
python

Copy code

from sklearn.metrics import silhouette_score


silhouette_avg = silhouette_score(X_scaled, data['KMeans_Cluster'])

print("Silhouette Score for K-Means:", silhouette_avg)
```

**Q9. Clustering Techniques (Mall Customers Dataset)**

**A. Preprocessing**

```
python

Copy code

from sklearn.preprocessing import StandardScaler


# Load the dataset

data = pd.read_csv('https://www.kaggle.com/shwetabh123/mall-customers')


# Select Spending Score for Clustering

X = data[['Spending Score']]


# Standardize the data

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)
```

**B. Apply Machine Learning Algorithms**

**1. K-Means Clustering**

```
python

Copy code

from sklearn.cluster import KMeans


# Apply K-Means

kmeans = KMeans(n_clusters=3, random_state=42)

data['KMeans_Cluster'] = kmeans.fit_predict(X_scaled)
```

**2. DBSCAN Clustering**

```
python

Copy code

from sklearn.cluster import DBSCAN


# Apply DBSCAN
```

```
dbscan = DBSCAN(eps=0.5, min_samples=5)
data['DBSCAN_Cluster'] = dbscan.fit_predict(X_scaled)
```

**C. Evaluate Models**

```
python
Copy code
from sklearn.metrics import silhouette_score

# Evaluate K-Means
kmeans_silhouette = silhouette_score(X_scaled, data['KMeans_Cluster'])
print("Silhouette Score for K-Means:", kmeans_silhouette)

# Evaluate DBSCAN
dbscan_silhouette = silhouette_score(X_scaled, data['DBSCAN_Cluster'], metric='euclidean')
print("Silhouette Score for DBSCAN:", dbscan_silhouette)
```

**D. Cross-Validation**

Cross-validation for clustering is non-trivial; use silhouette scores as a proxy for evaluating different numbers of clusters in K-Means.

```
python
Copy code
# Example: Cross-validation for K-Means
for n_clusters in range(2, 6):
  kmeans = KMeans(n_clusters=n_clusters, random_state=42)
  clusters = kmeans.fit_predict(X_scaled)
  silhouette_avg = silhouette_score(X_scaled, clusters)
  print(f"Silhouette Score for {n_clusters} clusters: {silhouette_avg}")
```

---

**Q10. PCA on the Iris Dataset**

**A. Describe the Dataset**

The Iris dataset contains 150 samples with four features: sepal length, sepal width, petal length, and petal width, classified into three species. Standardization is necessary because PCA is sensitive to the scale of the data.

**B. Standardization**

```
python
Copy code
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler

# Load dataset
iris = load_iris()
X = iris.data
y = iris.target

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

**C. Compute PCA**

```
python
Copy code
from sklearn.decomposition import PCA

# Apply PCA
pca = PCA(n_components=4)
X_pca = pca.fit_transform(X_standardized)
```

**D. Principal Component Directions and Correlations**

```
python
Copy code
# Explained variance
print("Explained Variance Ratio:", pca.explained_variance_ratio_)

# Principal components
print("Principal Components:\n", pca.components_)
```

**E. Plot Samples Projected into First K PCs**

```
python
Copy code
import matplotlib.pyplot as plt


# Plot the first two principal components
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis', edgecolor='k')
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.title('PCA on Iris Dataset')
plt.colorbar()
plt.show()
```

**F. Color Samples by Their Species**

```
python
Copy code
# Visualize with species
species = iris.target_names
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis', edgecolor='k')
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.title('PCA - Iris Species')
plt.colorbar(ticks=[0, 1, 2], label='Species')
plt.clim(-0.5, 2.5)
plt.show()
```