**1. Calculate the mean and standard deviation**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# Hardcoded values

values = [15, 20, 35, 40, 50, 60, 75, 80, 90, 100]

df = pd.DataFrame({"Values": values})

# Calculate mean & standard deviation

print("Mean:", df["Values"].mean())

print("Standard Deviation:", df["Values"].std())

# Bar Chart

df["Values"].plot(kind='bar', title="Bar Chart of Values", color="skyblue")

plt.show()

# Line Chart

df["Values"].plot(kind='line', title="Line Chart of Values", marker='o', color="red")

plt.show()

# Scatter Plot

df.plot(x=df.index, y="Values", kind="scatter", title="Scatter Plot", color="green")

plt.show()

===================================================================

**2. Read the CSV file**

import seaborn as sns

import pandas as pd

# Load Titanic dataset

titanic = sns.load\_dataset('titanic')

# Display first few rows

print("First 5 rows of the dataset:")

print(titanic.head())

===================================================================  
**3. Perform data filtering, and calculate aggregate statistics**

import pandas as pd

import seaborn as sns

# Load Titanic dataset

df = sns.load\_dataset('titanic').dropna(subset=['fare']) # Remove missing fares

# Display first few rows

print(df.head())

# Filter passengers where Fare > 50

filtered = df[df['fare'] > 50]

print(filtered[['survived', 'pclass', 'sex', 'age', 'fare', 'embark\_town']])

# Total & average fare by class

print(df.groupby('pclass')['fare'].agg(['sum', 'mean']))

# Total fare & count by embark town

print(df.groupby('embark\_town')['fare'].agg(['sum', 'count']))

==================================================================================**4. Calculate total sales by month.**

import pandas as pd

import seaborn as sns

# Load Titanic dataset

df = sns.load\_dataset('titanic')

# Convert 'age' to a date-like format for demonstration (assuming age as years since birth)

df['birth\_year'] = 1912 - df['age'].fillna(df['age'].median()) # Titanic sank in 1912

df['birth\_year'] = df['birth\_year'].astype(int)

df['Year'] = pd.to\_datetime(df['birth\_year'], format='%Y').dt.to\_period('Y')

# Group by Year and count passengers

yearly\_counts = df.groupby('Year')['survived'].count()

# Print result

print(yearly\_counts)

=========================================================================

**5. Implement the Clustering using K-means.**

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.datasets import make\_blobs

# Generate sample data

X, y = make\_blobs(n\_samples=300, centers=4, cluster\_std=0.6, random\_state=42)

# Plot raw data

plt.scatter(X[:, 0], X[:, 1], s=50, c='gray', marker='o')

plt.title("Raw Data")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.show()

# Apply K-means clustering

kmeans = KMeans(n\_clusters=4, random\_state=42, n\_init=10)

kmeans.fit(X)

# Get cluster labels and centroids

labels = kmeans.labels\_

centroids = kmeans.cluster\_centers\_

# Plot clustered data

plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=50)

plt.scatter(centroids[:, 0], centroids[:, 1], s=200, c='red', marker='X', label='Centroids')

plt.title("K-means Clustering")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.legend()

plt.show()

==================================================================================

**6. Classification using Random Forest.**

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.datasets import load\_iris

from sklearn.metrics import accuracy\_score, classification\_report

# Load dataset

X, y = load\_iris(return\_X\_y=True)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Train RandomForest model

rf = RandomForestClassifier(n\_estimators=100, random\_state=42).fit(X\_train, y\_train)

# Predictions

y\_pred = rf.predict(X\_test)

# Metrics

print(f"Accuracy: {accuracy\_score(y\_test, y\_pred) \* 100:.2f}%")

print("Classification Report:\n", classification\_report(y\_test, y\_pred))

print("Feature Importances:", dict(zip(load\_iris().feature\_names, rf.feature\_importances\_)))

============================================================================

**7. Regression Analysis using Linear Regression**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

# Data

df = pd.DataFrame({'YearsExperience': range(1, 11), 'Salary': np.arange(40000, 60000, 2000)})

# Split data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df[['YearsExperience']], df['Salary'], test\_size=0.2, random\_state=42)

# Train model

model = LinearRegression().fit(X\_train, y\_train)

# Predictions

y\_pred = model.predict(X\_test)

# Metrics

print(f"MSE: {mean\_squared\_error(y\_test, y\_pred):.2f}")

print(f"R² Score: {r2\_score(y\_test, y\_pred):.2f}")

# Plot results

plt.scatter(X\_test, y\_test, color='blue', label='Actual')

plt.plot(X\_test, y\_pred, color='red', label='Predicted')

plt.xlabel('Years of Experience')

plt.ylabel('Salary')

plt.title('Linear Regression: Salary vs Experience')

plt.legend()

plt.show()

# Prediction for new data

print(f"Predicted Salary (11 years): ${model.predict([[11]])[0]:,.2f}")

========================================================================

**8. Association Rule Mining using Apriori.**

import pandas as pd

from itertools import combinations

# Transaction data

df = pd.DataFrame({

'Milk': [1, 1, 0, 1, 1],

'Bread': [1, 1, 1, 1, 0],

'Butter': [0, 1, 1, 1, 1],

'Cheese': [1, 0, 1, 1, 1]

})

# Compute item frequencies (support)

support\_threshold = 0.6

total\_transactions = len(df)

# Find frequent individual items

frequent\_items = {}

for item in df.columns:

support = df[item].sum() / total\_transactions

if support >= support\_threshold:

frequent\_items[frozenset([item])] = support

# Find frequent item pairs

frequent\_pairs = {}

for item1, item2 in combinations(df.columns, 2):

support = (df[item1] & df[item2]).sum() / total\_transactions

if support >= support\_threshold:

frequent\_pairs[frozenset([item1, item2])] = support

# Generate association rules (confidence metric)

confidence\_threshold = 0.7

rules = []

for itemset, support in frequent\_pairs.items():

item1, item2 = list(itemset)

confidence\_1 = support / frequent\_items[frozenset([item1])]

confidence\_2 = support / frequent\_items[frozenset([item2])]

if confidence\_1 >= confidence\_threshold:

rules.append((item1, item2, confidence\_1))

if confidence\_2 >= confidence\_threshold:

rules.append((item2, item1, confidence\_2))

# Display results

print("Frequent Itemsets:")

for itemset, support in {\*\*frequent\_items, \*\*frequent\_pairs}.items():

print(f"{set(itemset)}: Support = {support:.2f}")

print("\nAssociation Rules:")

for antecedent, consequent, confidence in rules:

print(f"{antecedent} → {consequent} (Confidence: {confidence:.2f})")

=======================================================================

**9. Visualize the result of the clustering and compare**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make\_blobs

from sklearn.cluster import KMeans, DBSCAN

# Generate synthetic dataset

X, \_ = make\_blobs(n\_samples=300, centers=3, cluster\_std=1.0, random\_state=42)

# K-Means Clustering

kmeans = KMeans(n\_clusters=3, random\_state=42, n\_init=10)

kmeans\_labels = kmeans.fit\_predict(X)

# DBSCAN Clustering

dbscan = DBSCAN(eps=0.8, min\_samples=5)

dbscan\_labels = dbscan.fit\_predict(X)

# Plot results

fig, axes = plt.subplots(1, 2, figsize=(12, 5))

for ax, labels, title, centers in zip(

axes,

[kmeans\_labels, dbscan\_labels],

["K-Means Clustering", "DBSCAN Clustering"],

[kmeans.cluster\_centers\_, None],

):

ax.scatter(X[:, 0], X[:, 1], c=labels, cmap="viridis", edgecolor="k")

if centers is not None:

ax.scatter(centers[:, 0], centers[:, 1], c="red", marker="X", s=200, label="Centroids")

ax.legend()

ax.set\_title(title)

plt.show()

=========================================================================

**10. Visualize the correlation matrix using a pseudocolor plot.**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Generate a random dataset

np.random.seed(42)

data = pd.DataFrame(np.random.rand(10, 5), columns=['A', 'B', 'C', 'D', 'E'])

# Compute the correlation matrix

corr\_matrix = data.corr()

# Create a heatmap

plt.figure(figsize=(8, 6))

sns.heatmap(corr\_matrix, annot=True, cmap='coolwarm', linewidths=0.5, fmt=".2f")

plt.title('Correlation Matrix Heatmap')

plt.show()

=======================================================================

**11. Use of degrees distribution of a network.**

import networkx as nx

import matplotlib.pyplot as plt

from collections import Counter

# Generate a random Erdős-Rényi graph

G = nx.erdos\_renyi\_graph(n=100, p=0.05)

# Compute degree distribution

degree\_count = Counter(dict(G.degree()).values())

degrees, counts = zip(\*sorted(degree\_count.items()))

# Plot degree distribution

plt.figure(figsize=(8, 6))

plt.bar(degrees, counts, color='b', alpha=0.7, edgecolor='k')

plt.xlabel('Degree (k)')

plt.ylabel('Number of Nodes')

plt.title('Degree Distribution of the Network')

plt.grid(axis='y', linestyle='--', alpha=0.6)

plt.show()

# Plot log-log degree distribution

plt.figure(figsize=(8, 6))

plt.scatter(degrees, counts, color='r', alpha=0.7)

plt.xscale('log')

plt.yscale('log')

plt.xlabel('Log(Degree)')

plt.ylabel('Log(Count)')

plt.title('Log-Log Degree Distribution')

plt.grid(which='both', linestyle='--', alpha=0.6)

plt.show()

===============================================================

**12. Graph visualization of a network using maximum, minimum, median, first quartile and third quartile.**

import networkx as nx

import matplotlib.pyplot as plt

import numpy as np

import matplotlib.lines as mlines

# Generate a random Erdős-Rényi graph

G = nx.erdos\_renyi\_graph(n=100, p=0.1)

# Compute degree statistics

degree\_sequence = np.array([G.degree(node) for node in G.nodes()])

max\_degree, min\_degree = degree\_sequence.max(), degree\_sequence.min()

median\_degree = np.median(degree\_sequence)

q1, q3 = np.percentile(degree\_sequence, [25, 75])

print(f"Max Degree: {max\_degree}, Min Degree: {min\_degree}, Median: {median\_degree}, Q1: {q1}, Q3: {q3}")

# Define color categories

bins = [min\_degree, q1, median\_degree, q3, max\_degree]

colors = ['blue', 'green', 'yellow', 'orange', 'purple', 'red']

node\_colors = [colors[np.digitize(deg, bins)] for deg in degree\_sequence]

# Draw graph

plt.figure(figsize=(10, 8))

pos = nx.spring\_layout(G, seed=42) # Fixed layout for consistency

nx.draw(G, pos, with\_labels=True, node\_size=400, node\_color=node\_colors, font\_size=8, edge\_color='gray')

# Create legend

legend\_labels = [

("Max Degree", 'red', max\_degree),

("Min Degree", 'blue', min\_degree),

("Q1 (≤)", 'green', round(q1)),

("Median (≤)", 'yellow', round(median\_degree)),

("Q3 (≤)", 'orange', round(q3)),

("Above Q3", 'purple', "")

]

handles = [mlines.Line2D([], [], color=c, marker='o', markersize=10, label=f"{label} {value}")

for label, c, value in legend\_labels]

plt.legend(handles=handles, loc="upper left", fontsize=9)

plt.title("Network Visualization with Degree-Based Coloring")

plt.show()