**Code 1:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from scipy.stats import skew, kurtosis, shapiro

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

# Step 1: Read the CSV file

df = pd.read\_csv(r"C:\\Users\\sanke\\OneDrive\\Documents\\6th Sem\\DMPML\\Batting.csv",

encoding='ISO-8859-1')

# Step 2: Identify the variables and check for missing values

print("Data Overview:")

print(df.head())

# Check for missing values

print("\nMissing values:")

print(df.isnull().sum())

# Step 3: Impute missing values with the median (for numerical columns)

# Select only numeric columns

numeric\_cols = df.select\_dtypes(include=[np.number]).columns

# Impute missing values with the median of each numeric column

df[numeric\_cols] = df[numeric\_cols].fillna(df[numeric\_cols].median())

# Verify if there are still missing values

print("\nMissing values after imputation:")

print(df.isnull().sum())

# Calculate skewness for numeric columns

skewness = df[numeric\_cols].skew()

# Print skewness values

Software Engineering Laboratory

print("\nSkewness of numeric columns:")

print(skewness)

# Calculate kurtosis for numeric columns

kurtosis = df[numeric\_cols].kurt()

# Print kurtosis values

print("\nKurtosis of numeric columns:")

print(kurtosis)

# Step 5: Summary statistics for numerical variables

print("\nSummary statistics:")

print(df.describe())

# Step 6: Visualize the distribution of numerical variables

# Histograms of numerical variables

df.hist(bins=10, figsize=(10, 8))

plt.show()

# KDE plots for the first three numeric columns (limit to three)

columns\_to\_plot = df.select\_dtypes(include='number').columns[5:7] # Select first three numeric columns

for col in columns\_to\_plot:

sns.kdeplot(df[col], fill=True) # Use fill=True to avoid FutureWarning

plt.title(f'Distribution of {col}')

plt.show()

# Step 7: Apply natural log transformation to numerical variables

df\_log\_transformed = df.copy()

for col in df.select\_dtypes(include='number').columns:

df\_log\_transformed[col] = np.log1p(df[col]) # log1p to handle log(0) cases

Software Engineering Laboratory

print("\nLog-transformed data:")

print(df\_log\_transformed.head())

# Select only numeric columns

numeric\_cols = df.select\_dtypes(include=[np.number]).columns

# Calculate the correlation matrix for numeric columns

corr\_matrix = df[numeric\_cols].corr()

# Print the correlation matrix

print("\nCorrelation matrix:")

print(corr\_matrix)

# Plot correlation heatmap

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

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

plt.title('Correlation Matrix')

plt.show()

# Step 10: Partition the data into 70% training, 15% validation, and 15% test

train, temp = train\_test\_split(df, test\_size=0.30, random\_state=42)

validation, test = train\_test\_split(temp, test\_size=0.50, random\_state=42)

print(f"\nTraining set size: {len(train)}")

print(f"Validation set size: {len(validation)}")

print(f"Test set size: {len(test)}")

**Code 2:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

# Read the dataset from CSV file

data = pd.read\_csv("C:\\Users\\sanke\\OneDrive\\Documents\\6th Sem\\DMPML\\dirty\_cafe\_sales.csv")

# Rename columns

data.columns = ['Transaction\_ID', 'Item', 'Quantity', 'Price\_Per\_Unit', 'Total\_Spent',

'Payment\_Method', 'Location', 'Transaction\_Date']

# Display initial dataset

print("Initial Dataset:\n", data.head())

# Select specific columns

selected\_columns = data[['Transaction\_ID', 'Item', 'Total\_Spent', 'Payment\_Method']]

print("\nSelected Columns:\n", selected\_columns.head())

# Select specific rows

selected\_rows = data.iloc[:5]

print("\nSelected Rows:\n", selected\_rows)

# Sampling rows

sampled\_data = data.sample(5)

print("\nSampled Rows:\n", sampled\_data)

# Arranging rows by column values

data\_sorted = data.sort\_values(by='Total\_Spent', na\_position='last')

print("\nData Sorted by Total\_Spent:\n", data\_sorted.head())

# Filtering rows where 'Payment\_Method' is 'Cash'

cash\_transactions = data[data['Payment\_Method'] == 'Cash']

print("\nCash Transactions:\n", cash\_transactions.head())

Software Engineering Laboratory

# Transforming values in a field (e.g., uppercase items)

data['Item'] = data['Item'].str.upper()

print("\nTransformed 'Item' column:\n", data[['Transaction\_ID', 'Item']].head())

# Detect and fill missing data

missing\_values = data.isnull().sum()

missing\_percentage = (missing\_values / len(data)) \* 100

print("\nMissing Values:\n", missing\_values)

print("\nPercentage of Missing Values:\n", missing\_percentage)

data\_filled = data.fillna({

'Total\_Spent': 0,

'Payment\_Method': 'UNKNOWN',

'Location': 'UNKNOWN',

'Item': 'UNKNOWN'

})

# Detect rows with complete data

complete\_rows = data\_filled.dropna()

print("\nComplete Data Rows:\n", complete\_rows.head())

# Remove incomplete rows

data\_cleaned = data.dropna()

print("\nData After Removing Incomplete Rows:\n", data\_cleaned.head())

# Summarize dataset

summary = data.describe(include='all')

print("\nSummary Statistics:\n", summary)

# Handling non-numeric values and ensuring 'Total\_Spent' is numeric

data['Total\_Spent'] = pd.to\_numeric(data['Total\_Spent'], errors='coerce') # Convert to numeric, replace

errors with NaN

# Drop NaN values for further calculations

data = data.dropna(subset=['Total\_Spent'])

Software Engineering Laboratory

# Detecting outliers using IQR

Q1 = data['Total\_Spent'].quantile(0.25)

Q3 = data['Total\_Spent'].quantile(0.75)

IQR = Q3 - Q1

lower\_bound = Q1 - 1.5 \* IQR

upper\_bound = Q3 + 1.5 \* IQR

outliers = data[(data['Total\_Spent'] < lower\_bound) | (data['Total\_Spent'] > upper\_bound)]

print("\nOutliers Detected:\n", outliers)

# Draw boxplot to compare 'Total\_Spent'

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

plt.boxplot(data['Total\_Spent'].dropna(), vert=False, patch\_artist=True,

boxprops=dict(facecolor='lightblue'))

plt.title('Boxplot of Total\_Spent')

plt.xlabel('Total Spent')

plt.show()

# Save cleaned data to a new CSV

output\_path = 'cleaned\_transactions.csv'

data\_cleaned.to\_csv(output\_path, index=False)

print(f"\nCleaned data saved to {output\_path}")

**Code 3:**

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.linear\_model import LinearRegression

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

df = pd.read\_csv("C:\\Users\\sanke\\OneDrive\\Documents\\6th

Sem\\DMPML\\Student\_Performance.csv")

df = pd.get\_dummies(df, columns=['Extracurricular Activities'], drop\_first=True)

df.fillna(df.mean(), inplace=True)

X = df.drop("Performance Index", axis=1)

y = df["Performance Index"]

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)

print("Predicted Values: ", y\_pred)

print("Actual Values: ", y\_test.values)

residuals = y\_test.values - y\_pred

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

sns.residplot(x=y\_pred, y=residuals, lowess=True, line\_kws={'color': 'red'})

plt.title("Residuals Plot")

plt.xlabel("Predicted Performance Index")

plt.ylabel("Residuals")

Software Engineering Laboratory

plt.show()

mse = mean\_squared\_error(y\_test, y\_pred)

rmse = np.sqrt(mse)

r2 = r2\_score(y\_test, y\_pred)

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

print(f"Root Mean Squared Error: {rmse}")

print(f"R-Squared: {r2}")

**Code 4:**

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.linear\_model import LogisticRegression

from sklearn.metrics import (accuracy\_score, precision\_score, recall\_score,

f1\_score, roc\_auc\_score, confusion\_matrix,

classification\_report, roc\_curve)

# Load dataset

data = pd.read\_csv("C:\\Users\\sanke\\OneDrive\\Documents\\6th

Sem\\DMPML\\Student\_Performance.csv")

# ==============================================

# DATA EXPLORATION VISUALIZATIONS

# ==============================================

# 1. Distribution of Performance Index (Before Binarization)

plt.figure(figsize=(8, 4))

sns.histplot(data['Performance Index'], kde=True, bins=20)

plt.title("Distribution of Performance Index")

plt.xlabel("Performance Index")

plt.ylabel("Frequency")

plt.show()

# Binarize target

median\_performance = data['Performance Index'].median()

data['High\_Performance'] = (data['Performance Index'] >= median\_performance).astype(int)

# 2. Correlation Heatmap (convert categorical first)

data['Extracurricular Activities'] = data['Extracurricular Activities'].map({'Yes': 1, 'No': 0})

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

corr = data.corr()

Software Engineering Laboratory

sns.heatmap(corr, annot=True, cmap='coolwarm', center=0)

plt.title("Correlation Heatmap")

plt.show()

# ==============================================

# MODEL TRAINING AND EVALUATION

# ==============================================

# Select features and target

X = data[['Hours Studied', 'Previous Scores', 'Extracurricular Activities', 'Sleep Hours', 'Sample Question

Papers Practiced']]

y = data['High\_Performance']

# Split into train and test (80% train, 20% test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train Logistic Regression

model = LogisticRegression(max\_iter=1000)

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

# Predictions

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

y\_pred\_proba = model.predict\_proba(X\_test)[:, 1] # Probabilities for ROC-AUC

# ==============================================

# MODEL EVALUATION VISUALIZATIONS

# ==============================================

# 3. ROC Curve

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred\_proba)

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

plt.plot(fpr, tpr, label=f'Logistic Regression (AUC = {roc\_auc\_score(y\_test, y\_pred\_proba):.2f}')

plt.plot([0, 1], [0, 1], 'k--')

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.title("ROC Curve")

Software Engineering Laboratory

plt.legend()

plt.show()

# 4. Feature Importance (Coefficients)

plt.figure(figsize=(8, 4))

importance = pd.DataFrame({'Feature': X.columns, 'Coefficient': model.coef\_[0]})

importance = importance.sort\_values('Coefficient', ascending=False)

sns.barplot(x='Coefficient', y='Feature', data=importance)

plt.title("Feature Importance (Logistic Regression Coefficients)")

plt.show()

# 5. Prediction Probabilities Distribution

plt.figure(figsize=(8, 4))

sns.histplot(y\_pred\_proba, bins=20, kde=True)

plt.axvline(0.5, color='red', linestyle='--', label='Threshold (0.5)')

plt.title("Distribution of Prediction Probabilities")

plt.xlabel("Probability of High Performance")

plt.legend()

plt.show()

# ==============================================

# MODEL EVALUATION METRICS

# ==============================================

print("=== Model Evaluation ===")

print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))

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

print("\nMetrics:")

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

print(f"Precision: {precision\_score(y\_test, y\_pred):.2f}")

print(f"Recall: {recall\_score(y\_test, y\_pred):.2f}")

print(f"F1-Score: {f1\_score(y\_test, y\_pred):.2f}")

print(f"ROC-AUC Score: {roc\_auc\_score(y\_test, y\_pred\_proba):.2f}")

# ==============================================

# PREDICTIONS ON NEW DATA

Software Engineering Laboratory

# ==============================================

# Simulate new unseen data

new\_data = pd.DataFrame({

'Hours Studied': [6, 3, 8],

'Previous Scores': [85, 60, 90],

'Extracurricular Activities': ['Yes', 'No', 'Yes'],

'Sleep Hours': [7, 5, 8],

'Sample Question Papers Practiced': [2, 4, 5]

})

# Preprocess new data

new\_data['Extracurricular Activities'] = new\_data['Extracurricular Activities'].map({'Yes': 1, 'No': 0})

X\_new = new\_data[['Hours Studied', 'Previous Scores', 'Extracurricular Activities', 'Sleep Hours', 'Sample

Question Papers Practiced']]

# Predict

new\_predictions = model.predict(X\_new)

new\_predictions\_proba = model.predict\_proba(X\_new)[:, 1]

# Add predictions to new\_data

new\_data['Predicted\_High\_Performance'] = new\_predictions

new\_data['Prediction\_Probability'] = new\_predictions\_proba

print("\n=== Predictions on New Data ===")

print(new\_data[['Hours Studied', 'Previous Scores', 'Predicted\_High\_Performance',

'Prediction\_Probability']])

**Code 5:**

# Import necessary libraries

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split, GridSearchCV, cross\_val\_score

from sklearn.tree import DecisionTreeRegressor, plot\_tree

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

import matplotlib.pyplot as plt

import seaborn as sns

# Load dataset

df = pd.read\_csv("C:\\Users\\sanke\\OneDrive\\Documents\\6th

Sem\\DMPML\\Student\_Performance.csv")

# Convert categorical variable to numerical (Yes=1, No=0)

df['Extracurricular Activities'] = df['Extracurricular Activities'].map({'Yes': 1, 'No': 0})

# Define features (X) and target (y)

X = df.drop('Performance Index', axis=1)

y = df['Performance Index']

# Split data into training and testing sets (80% train, 20% test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Initialize and fit Decision Tree Regressor

dtree = DecisionTreeRegressor(random\_state=42)

dtree.fit(X\_train, y\_train)

# Predictions

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

# Model Evaluation Metrics

mse = mean\_squared\_error(y\_test, y\_pred)

rmse = np.sqrt(mse)

mae = mean\_absolute\_error(y\_test, y\_pred)

Software Engineering Laboratory

r2 = r2\_score(y\_test, y\_pred)

print("Model Evaluation Metrics:")

print(f"Mean Squared Error (MSE): {mse:.2f}")

print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")

print(f"Mean Absolute Error (MAE): {mae:.2f}")

print(f"R-squared (R2): {r2:.2f}")

# Cross-Validation (5-fold)

cv\_scores = cross\_val\_score(dtree, X, y, cv=5, scoring='r2')

print("\nCross-Validation Scores (R2):", cv\_scores)

print(f"Mean CV R2: {np.mean(cv\_scores):.2f}")

# Hyperparameter Tuning

param\_grid = {

'max\_depth': [3, 5, 7, 10, None],

'min\_samples\_split': [2, 5, 10],

'min\_samples\_leaf': [1, 2, 4]

}

grid\_search = GridSearchCV(DecisionTreeRegressor(random\_state=42), param\_grid, cv=5, scoring='r2')

grid\_search.fit(X\_train, y\_train)

optimized\_dtree = grid\_search.best\_estimator\_

# --- Visualization 1: Feature Importance (Bar Plot) ---

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

sns.barplot(x='Importance', y='Feature',

data=pd.DataFrame({

'Feature': X.columns,

'Importance': optimized\_dtree.feature\_importances\_

}).sort\_values('Importance', ascending=False))

plt.title("Feature Importance (Optimized Model)")

plt.xlabel("Importance Score")

plt.ylabel("Features")

plt.show()

# --- Visualization 2: Actual vs. Predicted Scatter Plot ---

Software Engineering Laboratory

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

plt.scatter(y\_test, y\_pred, alpha=0.6)

plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--') # Diagonal line

plt.title("Actual vs. Predicted Performance Index")

plt.xlabel("Actual Values")

plt.ylabel("Predicted Values")

plt.show()

# --- Visualization 3: Residual Plot ---

residuals = y\_test - y\_pred

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

sns.histplot(residuals, kde=True, bins=15)

plt.axvline(0, color='r', linestyle='--')

plt.title("Distribution of Residuals")

plt.xlabel("Residuals (Actual - Predicted)")

plt.show()

# --- Original Decision Tree Plot (Optional) ---

# plt.figure(figsize=(20, 10))

# plot\_tree(dtree, feature\_names=X.columns, filled=True, rounded=True)

# plt.title("Decision Tree Visualization")

# plt.show()

**Code 6:**

# Import required libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import silhouette\_score, adjusted\_rand\_score

from sklearn.datasets import load\_iris

# Load and explore dataset

iris = load\_iris()

data = pd.DataFrame(iris.data, columns=iris.feature\_names)

data['target'] = iris.target

print("=== Dataset Overview ===")

print(f"Shape: {data.shape}")

print("\nFirst 5 rows:")

print(data.head())

print("\nTarget distribution:")

print(data['target'].value\_counts())

# Preprocess data

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

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Determine optimal k using Elbow Method

wcss = []

k\_values = range(1, 11)

for k in k\_values:

kmeans = KMeans(n\_clusters=k, init='k-means++', random\_state=42)

kmeans.fit(X\_scaled)

Software Engineering Laboratory

wcss.append(kmeans.inertia\_)

plt.figure(figsize=(15, 5))

plt.subplot(1, 2, 1)

plt.plot(k\_values, wcss, marker='o', linestyle='--')

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

plt.ylabel('WCSS')

plt.title('Elbow Method')

plt.xticks(k\_values)

plt.grid()

# Silhouette Score analysis

silhouette\_scores = []

for k in range(2, 11):

kmeans = KMeans(n\_clusters=k, init='k-means++', random\_state=42)

labels = kmeans.fit\_predict(X\_scaled)

silhouette\_scores.append(silhouette\_score(X\_scaled, labels))

plt.subplot(1, 2, 2)

plt.plot(range(2, 11), silhouette\_scores, marker='o')

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

plt.ylabel('Silhouette Score')

plt.title('Silhouette Analysis')

plt.grid()

plt.tight\_layout()

plt.show()

# Apply K-Means with optimal k=3

optimal\_k = 3

kmeans = KMeans(n\_clusters=optimal\_k, init='k-means++', random\_state=42)

clusters = kmeans.fit\_predict(X\_scaled)

data['cluster'] = clusters

# Visualize clusters (using first two features)

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

sns.scatterplot(x='sepal length (cm)', y='sepal width (cm)',

Software Engineering Laboratory

hue='cluster', palette='viridis',

data=data, s=100)

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1],

s=300, c='red', marker='X', label='Centroids')

plt.title(f'K-Means Clustering (k={optimal\_k}) on Iris Dataset')

plt.legend()

plt.grid()

plt.show()

# Evaluate clustering performance

ari = adjusted\_rand\_score(data['target'], data['cluster'])

print(f"\n=== Clustering Evaluation ===")

print(f"Adjusted Rand Score: {ari:.4f}")

print("\nCluster distribution:")

print(data['cluster'].value\_counts())

# Compare clusters with actual labels

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)

sns.scatterplot(x='sepal length (cm)', y='sepal width (cm)',

hue='target', palette='viridis',

data=data, s=100)

plt.title('Actual Classes')

plt.subplot(1, 2, 2)

sns.scatterplot(x='sepal length (cm)', y='sepal width (cm)',

hue='cluster', palette='viridis',

data=data, s=100)

plt.title('Predicted Clusters')

plt.tight\_layout()

plt.show()

**Code 7:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from mlxtend.frequent\_patterns import apriori, association\_rules

from mlxtend.preprocessing import TransactionEncoder

# Load and preprocess

df = pd.read\_excel('/content/sample\_data/Online Retail.xlsx')

df = df[df['InvoiceNo'].notnull()]

df = df[~df['InvoiceNo'].astype(str).str.contains('C')]

df = df[df['Description'].notnull()]

df = df[df['Quantity'] > 0]

df = df[df['InvoiceNo'].astype(str).str.isnumeric()]

df = df[df['Country'] == 'United Kingdom']

# Keep only top 50 items to simplify

top\_items = df['Description'].value\_counts().head(50).index

df = df[df['Description'].isin(top\_items)]

# Group transactions

basket = df.groupby('InvoiceNo')['Description'].apply(list)

basket = basket[basket.apply(lambda x: len(x) <= 20)]

# Transaction encoding

te = TransactionEncoder()

te\_ary = te.fit(basket).transform(basket)

df\_encoded = pd.DataFrame(te\_ary, columns=te.columns\_)

# Generate rules

def get\_rules(support, confidence):

frequent\_items = apriori(df\_encoded, min\_support=support, use\_colnames=True)

rules = association\_rules(frequent\_items, metric="confidence", min\_threshold=confidence)

return rules

# Prettify rule sets

def prettify\_rules(rules):

rules = rules.copy()

rules['antecedents'] = rules['antecedents'].apply(lambda x: ', '.join(sorted(x)))

rules['consequents'] = rules['consequents'].apply(lambda x: ', '.join(sorted(x)))

return rules

# Generate and prettify rules

rules\_a = prettify\_rules(get\_rules(0.01, 0.3))

rules\_b = prettify\_rules(get\_rules(0.02, 0.4))

rules\_c = prettify\_rules(get\_rules(0.03, 0.5))

# Rule count summary

print("\nRule Count Summary:")

print(f"(a) Support ≥ 1%, Confidence ≥ 30%: {len(rules\_a)} rules")

print(f"(b) Support ≥ 2%, Confidence ≥ 40%: {len(rules\_b)} rules")

print(f"(c) Support ≥ 3%, Confidence ≥ 50%: {len(rules\_c)} rules")

# Display first 5 rules for each threshold

print("\n(a) Support ≥ 1%, Confidence ≥ 30%")

print(rules\_a[['antecedents', 'consequents', 'support', 'confidence', 'lift']].head())

print("\n(b) Support ≥ 2%, Confidence ≥ 40%")

print(rules\_b[['antecedents', 'consequents', 'support', 'confidence', 'lift']].head())

print("\n(c) Support ≥ 3%, Confidence ≥ 50%")

print(rules\_c[['antecedents', 'consequents', 'support', 'confidence', 'lift']].head())

# Combine and sort top rules by lift

all\_rules = pd.concat([rules\_a, rules\_b, rules\_c])

top\_lift = all\_rules.sort\_values(by='lift', ascending=False).drop\_duplicates().head()

print("\nTop 5 Rules Sorted by Lift:")

print(top\_lift[['antecedents', 'consequents', 'support', 'confidence', 'lift']])

# Confidence + lift interpretation

print("\nConfidence & Lift Interpretation:")

for i, row in top\_lift.head(2).iterrows():

print(f"If a customer buys [{row['antecedents']}], there's a {row['confidence']:.2%} chance they'll also buy [{row['consequents']}].")

print(f"Lift of {row['lift']:.2f} means it's {row['lift']:.2f} times more likely than random.")

# Scatter plot

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

sns.scatterplot(data=top\_lift, x='support', y='confidence', size='lift', hue='lift',

palette='viridis', sizes=(100, 600), legend='brief')

plt.title('Top Association Rules (Support vs Confidence)')

plt.xlabel('Support')

plt.ylabel('Confidence')

plt.tight\_layout()

plt.show()

# Grouped lift heatmap

pivot = top\_lift.pivot\_table(index=top\_lift['antecedents'],

columns=top\_lift['consequents'],

values='lift', fill\_value=0)

if not pivot.empty:

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

sns.heatmap(pivot, annot=True, cmap='YlGnBu', fmt='.2f')

plt.title('Grouped Lift Heatmap')

plt.tight\_layout()

plt.show()

else:

print("\nNo valid combinations found for heatmap — pivot table is empty.")

# Display rules with total length ≥ 5

def display\_min\_len\_rules(rules, label):

rules\_len5 = rules[rules['antecedents'].str.count(',') + rules['consequents'].str.count(',') + 2 >= 5]

print(f"\n{label} First 5 Rules with Length ≥ 5:")

print(rules\_len5[['antecedents', 'consequents', 'support', 'confidence', 'lift']].head())

display\_min\_len\_rules(rules\_a, "(a)")

display\_min\_len\_rules(rules\_b, "(b)")

display\_min\_len\_rules(rules\_c, "(c)")

# Bonus: Most frequent consequents

top\_consequents = all\_rules['consequents'].value\_counts().head(10)

plt.figure(figsize=(10,5))

sns.barplot(x=top\_consequents.values, y=top\_consequents.index, palette="mako")

plt.title("Top 10 Frequently Recommended Consequents")

plt.xlabel("Frequency")

plt.ylabel("Consequent Item(s)")

plt.tight\_layout()

plt.show()

# Bonus: Show strongest rules

strong\_rules = all\_rules[(all\_rules['confidence'] > 0.7) & (all\_rules['lift'] > 2)]

print("\nHighly Strong Rules (Confidence > 70% and Lift > 2):")

print(strong\_rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']].head())