**Project Coversheet**

|  |  |
| --- | --- |
| Full Name | Fatema Tuj Johura |
| Email | fatemasoshi@gmail.com |
| Contact Number | 07440734486 |
| Project Title (Example – Week1, Week2, Week3) | Week 3-Advanced Data Analysis Techniques and Business Insights |

**Project Guidelines and Rules**

**1. Formatting and Submission**

* Format: Use a readable font (e.g., Arial/Times New Roman), size 12, 1.5 line spacing.
* Title: Include Week and Title (Example - Week 1: Travel Ease Case Study.)
* File Format: Submit as PDF or Word file
* Page Limit: 4–5 pages, including the title and references.

**2. Answer Requirements**

* Word Count: Each answer should be within 100–150 words; Maximum 800–1,200 words.
* Clarity: Write concise, structured answers with key points.
* Tone: Use formal, professional language.

**3. Content Rules**

* Answer all questions thoroughly, referencing case study concepts.
* Use examples where possible (e.g., risk assessment techniques).
* Break complex answers into bullet points or lists.

**4. Plagiarism Policy**

* Submit original work; no copy-pasting.
* Cite external material in a consistent format (e.g., APA, MLA).

**5. Evaluation Criteria**

* Understanding: Clear grasp of business analysis principles.
* Application: Effective use of concepts like cost-benefit analysis and Agile/Waterfall.
* Clarity: Logical, well-structured responses.
* Creativity: Innovative problem-solving and examples.
* Completeness: Answer all questions within the word limit.

**6. Deadlines and Late Submissions**

* Deadline: Submit on time; trainees who fail to submit the project will miss the “Certificate of Excellence”

**7. Additional Resources**

* Refer to lecture notes and recommended readings.
* Contact the instructor or peers for clarifications before the deadline.

|  |
| --- |
| **YOU CAN START YOUR PROJECT FROM HERE** |

Week 3-Advanced Data Analysis Techniques and Business Insights

import pandas as pd

# Load the Excel file (reads the first sheet by default)

excel\_file = 'raw\_sales\_data.xlsx' # Replace with your actual file name

df = pd.read\_excel(excel\_file)

# Save the DataFrame to a CSV file

csv\_file = 'raw\_sales\_data.csv'

df.to\_csv(csv\_file, index=False)

print(f"Excel file '{excel\_file}' has been converted to '{csv\_file}'")

Excel file 'raw\_sales\_data.xlsx' has been converted to 'raw\_sales\_data.csv'

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from scipy.stats import zscore

import math

# -------------------------

# Step 0: Load dataset

# -------------------------

df = pd.read\_csv('raw\_sales\_data.csv') # Replace with your actual file name

# Normalize column names to lowercase for consistency

df.columns = df.columns.str.strip().str.lower()

# -------------------------

# Step 1: Handle Missing Values

# -------------------------

# Identify numerical and categorical columns

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

cat\_cols = df.select\_dtypes(include=['object']).columns

# Impute numerical columns with mean

for col in num\_cols:

df[col] = df[col].fillna(df[col].mean())

# Impute categorical columns with mode

for col in cat\_cols:

df[col] = df[col].fillna(df[col].mode()[0])

print("✅ Missing values handled using mean (numerical) and mode (categorical).")

# -------------------------

# Step 2: Detect and Remove Outliers (Z-score)

# -------------------------

# Visualize outliers before removal

n\_cols = len(num\_cols)

n\_rows = math.ceil(n\_cols / 2)

plt.figure(figsize=(14, 4 \* n\_rows))

for i, col in enumerate(num\_cols):

plt.subplot(n\_rows, 2, i + 1)

sns.boxplot(x=df[col])

plt.title(f'Before Outlier Removal - {col}')

plt.tight\_layout()

plt.show()

# Remove outliers using Z-score

z\_scores = np.abs(zscore(df[num\_cols]))

df\_clean = df[(z\_scores < 3).all(axis=1)].copy() # Copy to avoid SettingWithCopyWarning

# Normalize column names again (to be safe)

df\_clean.columns = df\_clean.columns.str.strip().str.lower()

print(f"✅ Outliers removed. Rows reduced from {df.shape[0]} to {df\_clean.shape[0]}.")

# Visualize after outlier removal

plt.figure(figsize=(14, 4 \* n\_rows))

for i, col in enumerate(num\_cols):

plt.subplot(n\_rows, 2, i + 1)

sns.boxplot(x=df\_clean[col])

plt.title(f'After Outlier Removal - {col}')

plt.tight\_layout()

plt.show()

# -------------------------

# Step 3: Standardize Categorical Variables

# -------------------------

# Re-identify categorical columns after cleaning

cat\_cols = df\_clean.select\_dtypes(include=['object']).columns

# Normalize categorical values

for col in cat\_cols:

df\_clean[col] = df\_clean[col].astype(str).str.strip().str.lower()

# Replace inconsistent labels

if 'region' in df\_clean.columns:

df\_clean['region'] = df\_clean['region'].replace({

'n': 'north', 's': 'south', 'e': 'east', 'w': 'west'

})

df\_clean['region'] = df\_clean['region'].str.title()

if 'churned' in df\_clean.columns:

df\_clean['churned'] = df\_clean['churned'].replace({

'yes': 'yes', 'y': 'yes', 'no': 'no', 'n': 'no'

})

df\_clean['churned'] = df\_clean['churned'].str.capitalize()

print("✅ Categorical variables standardized for consistency.")

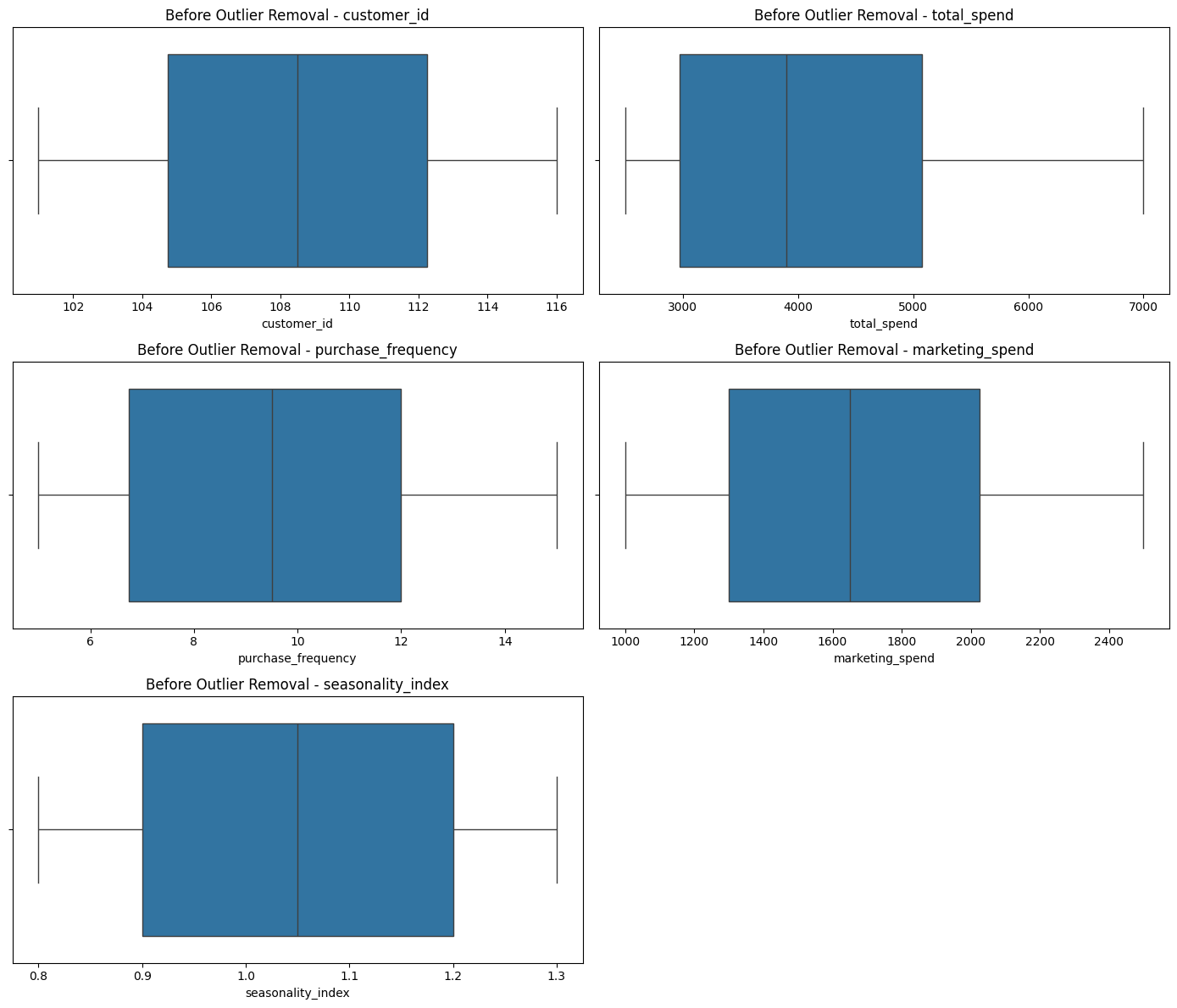
# -------------------------

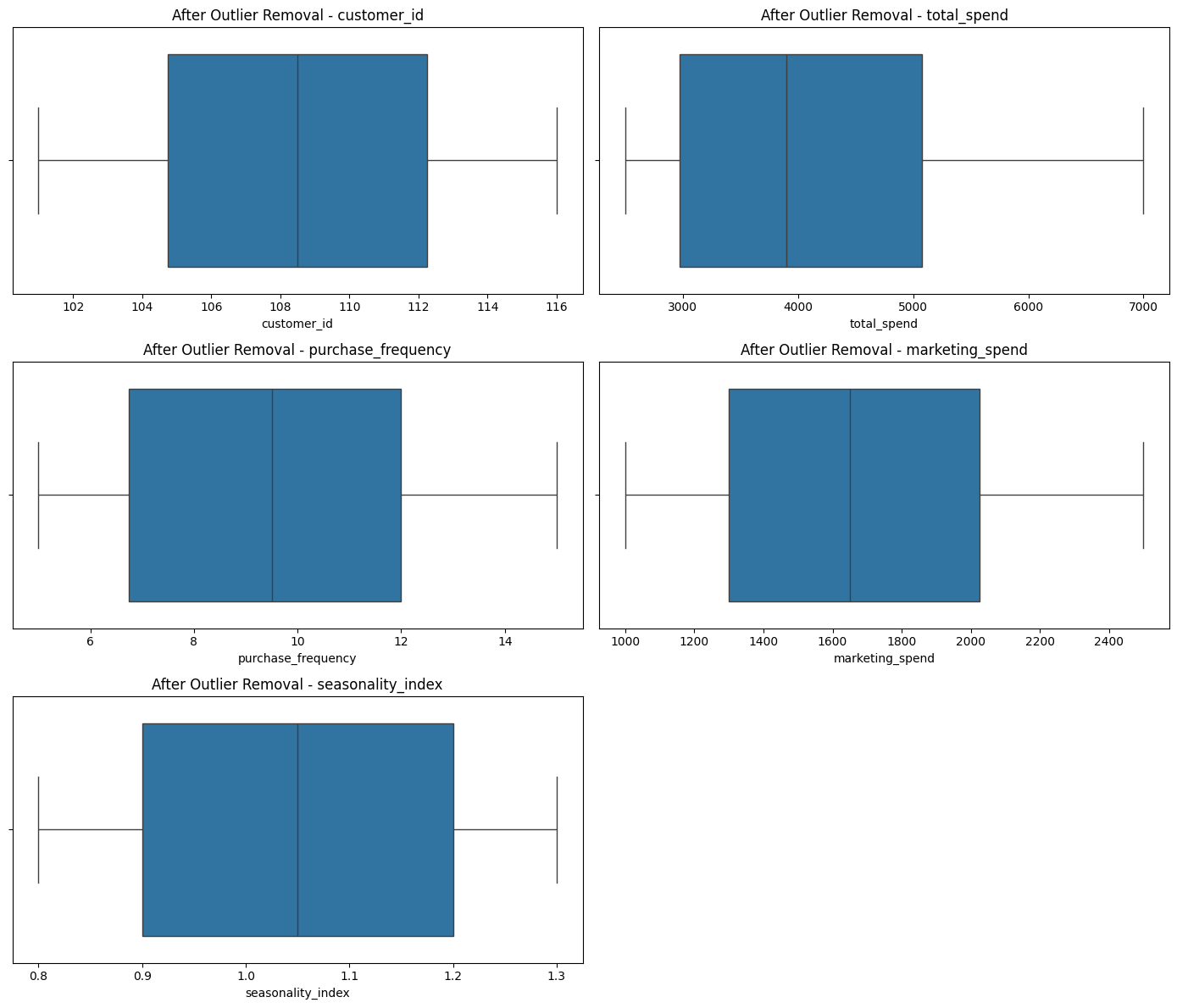
# Step 4: Save Cleaned Data

# -------------------------

df\_clean.to\_csv('final\_cleaned\_data.csv', index=False)

print("✅ Cleaned data saved as 'final\_cleaned\_data.csv'.")





✅ Categorical variables standardized for consistency.

✅ Cleaned data saved as 'final\_cleaned\_data.csv'.

*Step 1: Linear Regression – Predict total\_spend*

df = pd.read\_csv('final\_cleaned\_data.csv')

# Ensure all column names are lowercase for consistency

df.columns = df.columns.str.strip().str.lower()

from sklearn.linear\_model import LinearRegression

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

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

# Features and target

X = df[['marketing\_spend', 'seasonality\_index']]

y = df['total\_spend']

# Drop rows with missing target

mask = y.notna()

X = X[mask]

y = y[mask]

# Train-test split

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

# Model training

lr = LinearRegression()

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

# Prediction & Evaluation

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

print("\nLinear Regression Results")

print("R² Score:", r2\_score(y\_test, y\_pred))

print("MSE:", mean\_squared\_error(y\_test, y\_pred))

Linear Regression Results

R² Score: 0.793258175036579

MSE: 178185.61039034845

Step 2: Logistic Regression – Predict Customer Churn

from sklearn.linear\_model import LogisticRegression

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

# Convert churned to binary if needed

df['churned'] = df['churned'].map({'Yes': 1, 'No': 0, 'yes': 1, 'no': 0})

# Features and target

X = df[['total\_spend', 'marketing\_spend', 'purchase\_frequency', 'seasonality\_index']]

y = df['churned']

# Drop rows with missing target

mask = y.notna()

X = X[mask]

y = y[mask]

# Train-test split

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

# Model training

log\_reg = LogisticRegression(solver='liblinear', max\_iter=1000)

log\_reg.fit(X\_train, y\_train)

# Prediction & Evaluation

y\_pred = log\_reg.predict(X\_test)

print("\n Logistic Regression Results")

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

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

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

Logistic Regression Results

Accuracy: 1.0

Confusion Matrix:

[[2 0]

[0 2]]

Classification Report:

precision recall f1-score support

0 1.00 1.00 1.00 2

1 1.00 1.00 1.00 2

accuracy 1.00 4

macro avg 1.00 1.00 1.00 4

weighted avg 1.00 1.00 1.00 4

from statsmodels.tsa.arima.model import ARIMA

import matplotlib.pyplot as plt

# Step 1: Load cleaned data

df = pd.read\_csv('final\_cleaned\_data.csv')

df.columns = df.columns.str.strip().str.lower()

# Step 2: Simulate monthly sales with synthetic months

monthly\_sales = df[['total\_spend']].copy()

monthly\_sales['month'] = pd.date\_range(start='2023-01-01', periods=len(df), freq='MS')

monthly\_sales.set\_index('month', inplace=True)

monthly\_sales.columns = ['sales']

monthly\_sales.index.freq = 'MS' # Avoid frequency warning

# Step 3: Fit ARIMA model (adjust order if needed)

arima\_model = ARIMA(monthly\_sales['sales'], order=(0, 1, 1))

arima\_fit = arima\_model.fit(method\_kwargs={"maxiter": 500}) # Reduce convergence issues

# Step 4: Forecast next 3 months

forecast = arima\_fit.forecast(steps=3)

forecast.index = pd.date\_range(start=monthly\_sales.index[-1] + pd.DateOffset(months=1), periods=3, freq='MS')

# Step 5: Plot results

monthly\_sales['sales'].plot(label='Actual Sales', figsize=(10, 4))

forecast.plot(label='Forecast', style='--', color='red')

plt.title('ARIMA Monthly Sales Forecast')

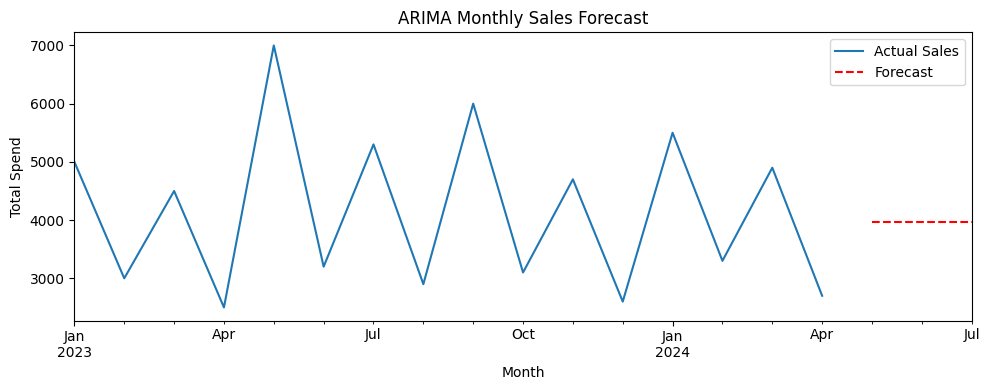
plt.xlabel('Month')

plt.ylabel('Total Spend')

plt.legend()

plt.tight\_layout()

plt.show()



*3. Statistical Analysis for Business Insights*

Step 1: ANOVA – Compare total\_spend across different regions

# Load cleaned data

from scipy.stats import f\_oneway

df = pd.read\_csv('final\_cleaned\_data.csv')

df.columns = df.columns.str.strip().str.lower()\

# Group total\_spend by region

regions = df['region'].unique()

groups = [df[df['region'] == r]['total\_spend'] for r in regions]

# Perform ANOVA test

f\_stat, p\_value = f\_oneway(\*groups)

print("ANOVA Test: Total Spend by Region")

print("F-statistic:", f\_stat)

print("P-value:", p\_value)

# Interpretation

if p\_value < 0.05:

print("✅ Statistically significant difference in sales across regions.")

else:

print("❌ No significant difference in sales across regions.")

ANOVA Test: Total Spend by Region

F-statistic: 39.719626168224295

P-value: 1.6512569414092805e-06

✅ Statistically significant difference in sales across regions.

*Step 2: Hypothesis Testing – Impact of promotions on sales*

from scipy.stats import ttest\_ind

# Simulate a 'promotion\_applied' column (randomly for demo)

np.random.seed(42)

df['promotion\_applied'] = np.random.choice([0, 1], size=len(df)) # 0 = No Promo, 1 = Promo

# Group by promotion status

promo\_group = df[df['promotion\_applied'] == 1]['total\_spend']

no\_promo\_group = df[df['promotion\_applied'] == 0]['total\_spend']

# Perform independent t-test

t\_stat, p\_val = ttest\_ind(promo\_group, no\_promo\_group)

print("\n📊 Hypothesis Test: Promotions vs Total Spend")

print("T-statistic:", t\_stat)

print("P-value:", p\_val)

# Interpretation

if p\_val < 0.05:

print("✅ Promotions significantly impact sales.")

else:

print("❌ No significant impact of promotions on sales.")

📊 Hypothesis Test: Promotions vs Total Spend

T-statistic: -0.9698897952179867

P-value: 0.3485602980432204

❌ No significant impact of promotions on sales.

*Step 3: Factor Analysis – Identify key drivers of purchase behavior*

from sklearn.decomposition import FactorAnalysis

from sklearn.preprocessing import StandardScaler

# Select numeric features for factor analysis

features = df[['total\_spend', 'marketing\_spend', 'purchase\_frequency', 'seasonality\_index']]

features\_scaled = StandardScaler().fit\_transform(features)

# Apply Factor Analysis (e.g., 2 latent factors)

fa = FactorAnalysis(n\_components=2, random\_state=0)

factors = fa.fit\_transform(features\_scaled)

# Loadings (contribution of each feature to each factor)

loadings = pd.DataFrame(fa.components\_.T, columns=['Factor 1', 'Factor 2'], index=features.columns)

print("\n📊 Factor Analysis: Feature Loadings")

print(loadings)

📊 Factor Analysis: Feature Loadings

Factor 1 Factor 2

total\_spend 0.972406 -0.054165

marketing\_spend 0.998248 -0.013650

purchase\_frequency 0.992566 0.065701

seasonality\_index 0.967166 -0.068743

*4. Machine Learning for Customer Segmentation*

# Load cleaned data

df = pd.read\_csv('final\_cleaned\_data.csv')

df.columns = df.columns.str.strip().str.lower()

# Drop non-numeric or identifier columns

features = df[['total\_spend', 'marketing\_spend', 'purchase\_frequency', 'seasonality\_index']]

*Task 1: Decision Tree – Segment Customers Based on Purchasing Behavior*

from sklearn.tree import DecisionTreeClassifier, plot\_tree

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

from sklearn.preprocessing import LabelEncoder

import matplotlib.pyplot as plt

# Encode churn as binary

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

X = features

y = df['churned']

# Train-test split

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

# Train Decision Tree

dt = DecisionTreeClassifier(max\_depth=3, random\_state=42)

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

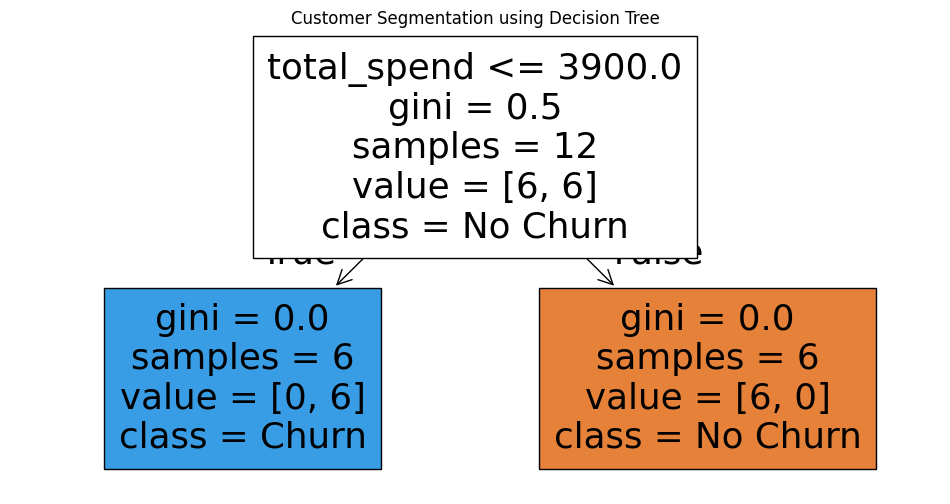
# Plot tree

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

plot\_tree(dt, feature\_names=X.columns, class\_names=['No Churn', 'Churn'], filled=True)

plt.title("Customer Segmentation using Decision Tree")

plt.show()



*Task 2: K-Means Clustering – Group Customers by Spending Category*

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

import seaborn as sns

# Scale features

scaler = StandardScaler()

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

# Apply KMeans

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

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

# Add cluster to DataFrame

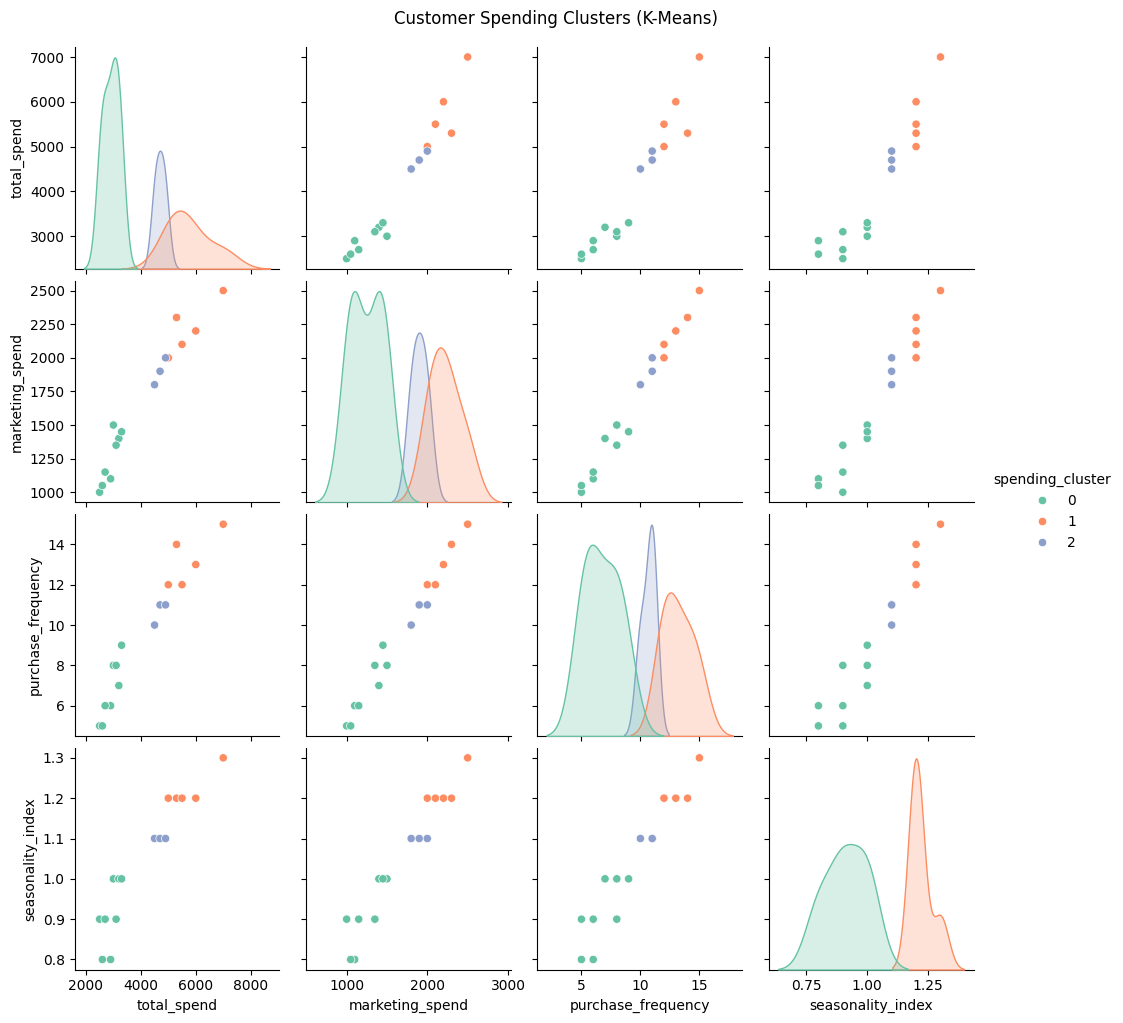
df['spending\_cluster'] = clusters

# Visualize clusters

sns.pairplot(df, vars=features.columns, hue='spending\_cluster', palette='Set2')

plt.suptitle("Customer Spending Clusters (K-Means)", y=1.02)

plt.show()



*Task 3: Ensemble Learning – Predict Churn Using Random Forest and XGBoost*

from sklearn.ensemble import RandomForestClassifier

from xgboost import XGBClassifier

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

# Reuse X and y from earlier

# --------- Random Forest ---------

rf = RandomForestClassifier(n\_estimators=100, random\_state=42)

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

rf\_preds = rf.predict(X\_test)

print("🌲 Random Forest Results:")

print("Accuracy:", accuracy\_score(y\_test, rf\_preds))

print(classification\_report(y\_test, rf\_preds))

🌲 Random Forest Results:

Accuracy: 1.0

precision recall f1-score support

0 1.00 1.00 1.00 2

1 1.00 1.00 1.00 2

accuracy 1.00 4

macro avg 1.00 1.00 1.00 4

weighted avg 1.00 1.00 1.00 4

# --------- XGBoost ---------

xgb = XGBClassifier(use\_label\_encoder=False, eval\_metric='logloss', random\_state=42)

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

xgb\_preds = xgb.predict(X\_test)

print("\n⚡ XGBoost Results:")

print("Accuracy:", accuracy\_score(y\_test, xgb\_preds))

print(classification\_report(y\_test, xgb\_preds))

⚡ XGBoost Results:

Accuracy: 1.0

precision recall f1-score support

0 1.00 1.00 1.00 2

1 1.00 1.00 1.00 2

accuracy 1.00 4

macro avg 1.00 1.00 1.00 4

weighted avg 1.00 1.00 1.00 4