CUSTOMER PURCHASE BEHAVIOR

Riyaz Sikora

INSY 5339 - 001

Principles of Business Data Mining

Hasan Zafar

Roshani Atharva Chitre

Banu Priya Pellur Santhosh Kumar

Agenda

1. Introduction
2. Data Description and Source
3. Business Objective
4. Exploratory Data Analysis
5. Methodology - Supervised / Unsupervised
6. Findings and Evaluation
7. Conclusion

Introduction

Customer purchase behavior is the study of how individuals decide to allocate their resources, such as time, money, and effort, toward buying and using goods and services. This behavior is influenced by various demographic, psychological, and situational factors.

Businesses analyze customer purchase behavior to optimize their marketing strategies, enhance customer satisfaction, and drive sales.

Here, we explore the key elements of customer purchase behavior, focusing on specific customer attributes and their impact on purchasing decisions.

Data Description and Source

The source of dataset is from Kaggle.

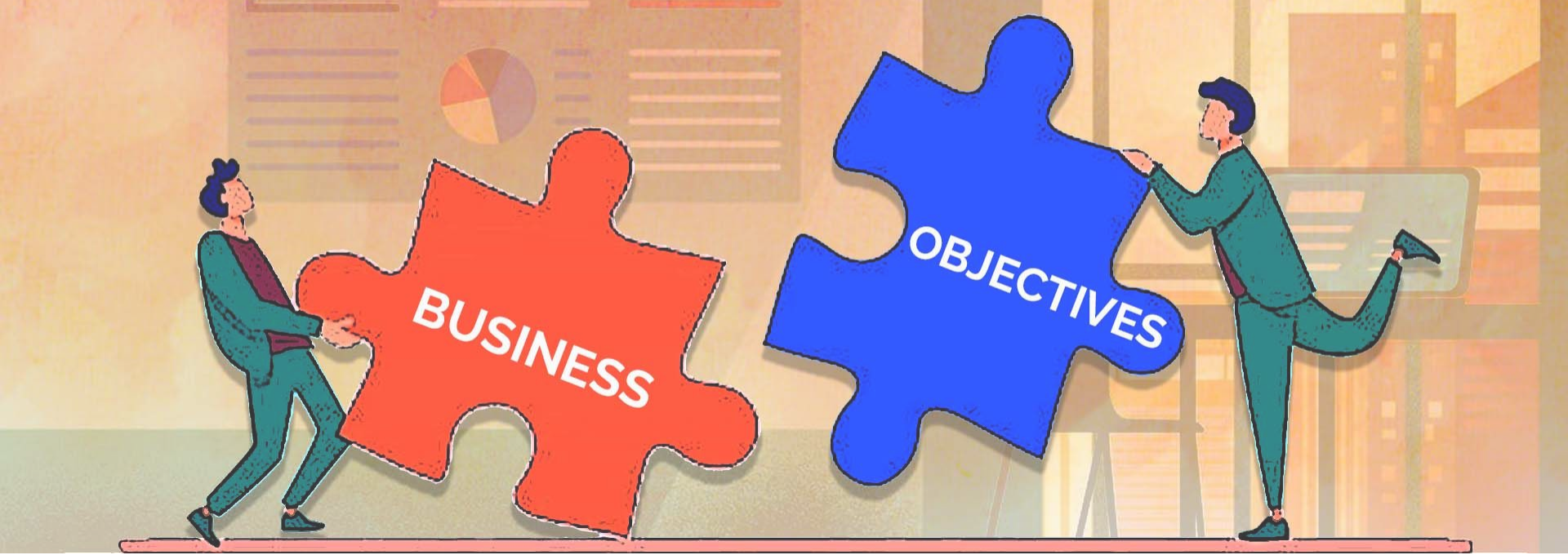
https://www.kaggle.com/datasets/sanyamgoyal401/customer-purchases-behaviour-dataset.

This dataset consists of simulated data representing customer purchase behavior. The data includes various features that provide insight into the demographics and purchase patterns of customers. The data was generated using the simstudy package in R, utilizing various distributions and formulas to create a synthetic dataset that mimics real-world scenarios without representing actual customer data.

File Format: **CSV Number of Rows: 100,000 Number of Columns: 12**

Characteristics of Dataset

S.No	Attributes	Description	Subcategories / range
1	Customer ID	Unique identifier for each user	Distinct
2	Age	Age of the user in years	12 - 49
3	Gender	Gender identity of the user	Female, Male
4	Income	Annual income of the customer.	5000 - 49993
5	Education	Education level of the customer.	High School, College, Bachelors, Masters
6	Region	Region where the customer resides.	East, West, North, South
7	Loyalty status	Loyalty status of the customer.	Regular, Silver, Gold
8	Purchase Frequency	Frequency of purchases made by the customer.	Rare, Occasional, Frequent
9	Purchase Amount	Amount spent by the customer in each purchase.	1118 - 24147
10	Product Category	Category of the purchased product.	Beauty, Books, Clothing, Electronics, Food, Health, Home
11	Promotion Usage	Indicates whether the customer used promotional offers	Promotion Applied - 0, Not Applied - 1
12	Satisfaction Score	Satisfaction score of the customer	0 - 10



How can we identify and analyze the factors affecting customer purchase amounts to enhance our sales strategy?

The main objective of this project is to predict customer purchase amounts using a range of demographic, behavioral, and psychological factors. This will enable us to allocate our marketing budget more effectively by targeting the appropriate market segments.

Objectives

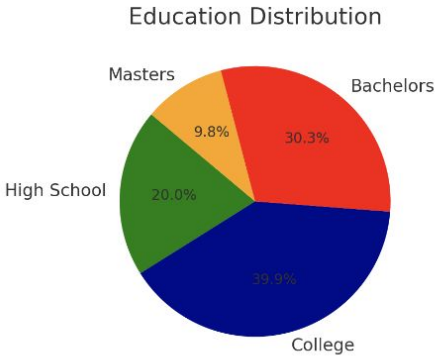
1. **Predict Purchase Amounts**: Develop a model to forecast customer purchase amounts using demographic, behavioral, and psychological factors.
2. **Identify Key Factors**: Determine the most significant variables influencing purchase amounts.
3. **Customer Segmentation**: Group customers based on predicted purchase amounts and other characteristics for targeted marketing.

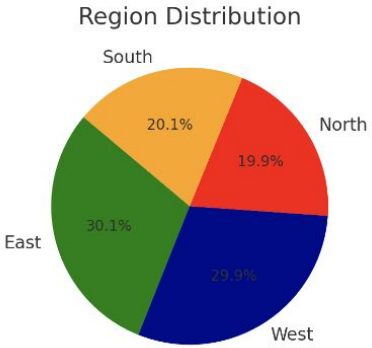
Goal

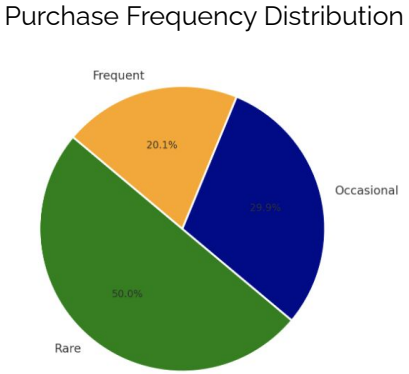
Provide valuable insights into customer behavior enabling us to allocate our marketing budget more effectively by targeting the appropriate market segments.

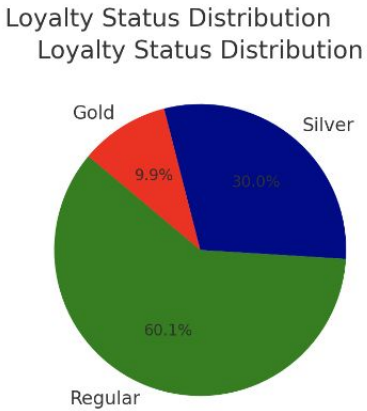
Exploratory Data Analysis

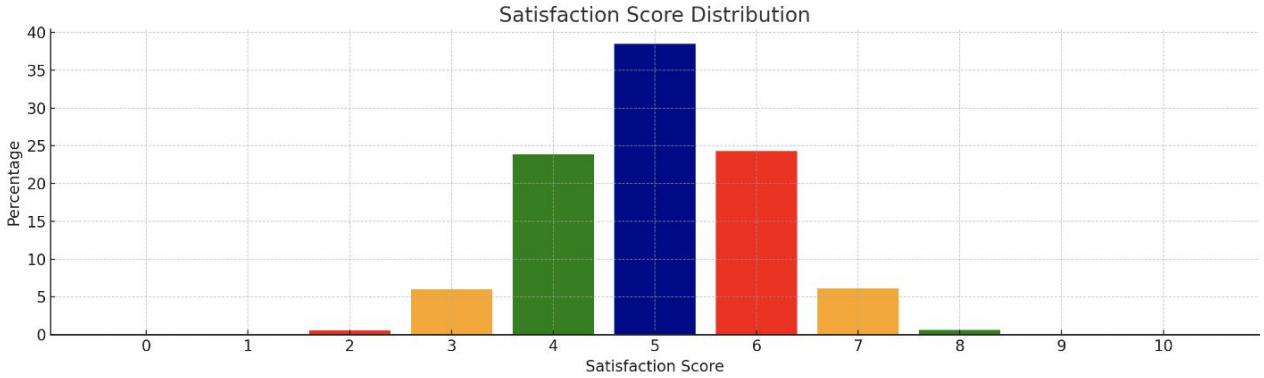
Data Analysis using Visualization



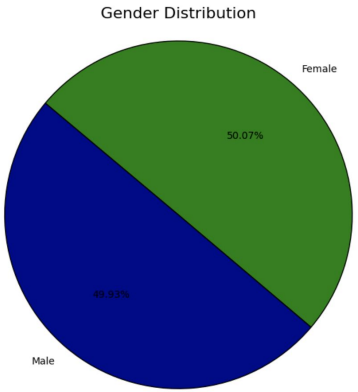


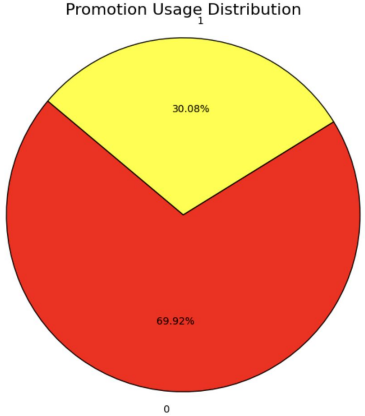


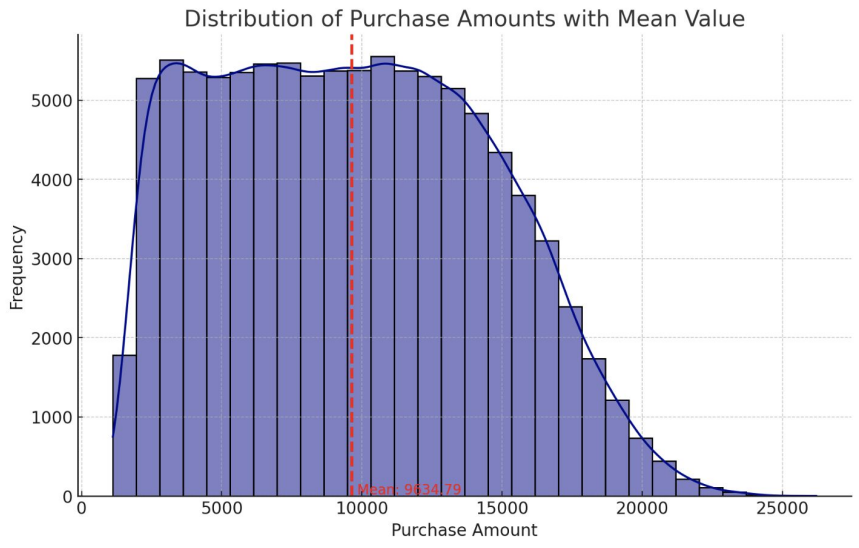




Data Analysis using Visualization



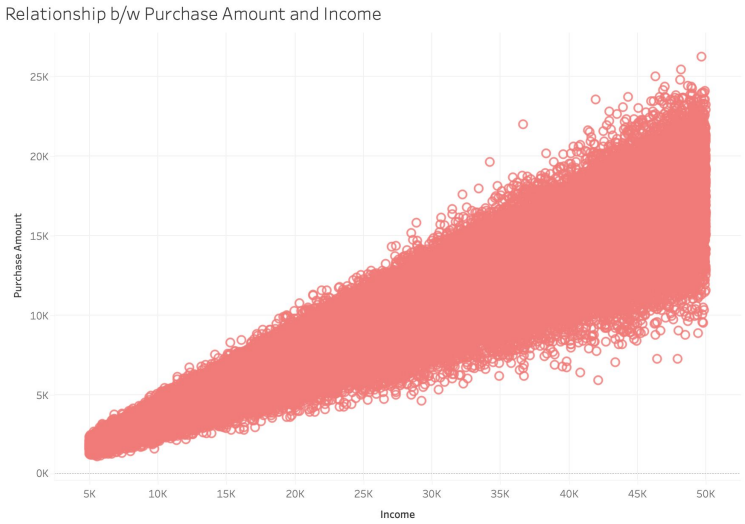




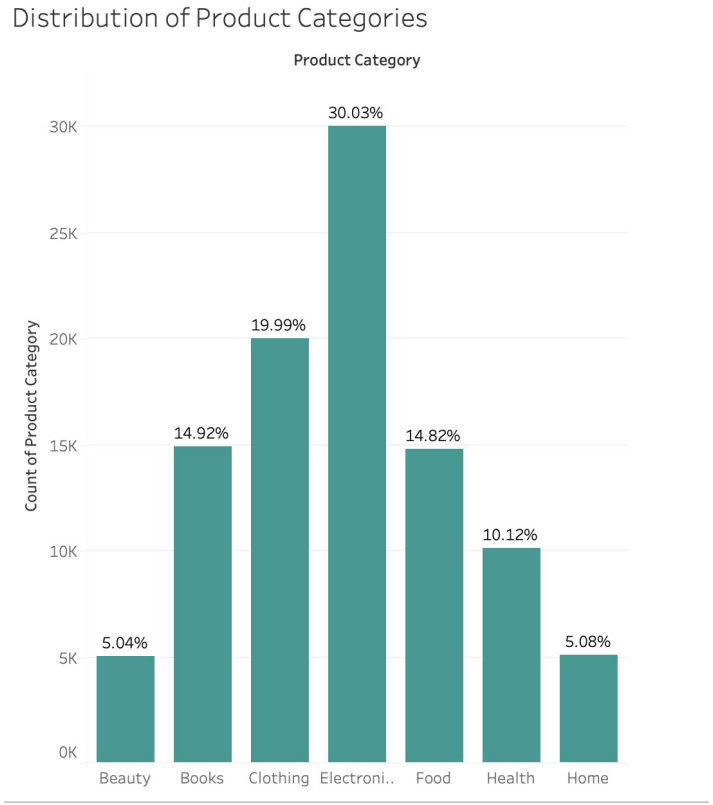
Before doing any further processing, what would your prediction of the target variable be?

A naive prediction for the purchase amount could be the mean value of the target variable which is **9634.79**. This will serve as a baseline for evaluating the performance of predictive models.

Data Analysis using Visualization



Purchase Frequency	Avg. Age
frequent	30.06
occasional	29.97
rare	30.00



Exploratory Data Analysis

```
df = pd.read_csv(io.BytesIO(uploaded['customer_data.csv']))
df
```

	id	age	gender	income	education	region	loyalty_status	purchase_frequency	purchase_amount	product_category	promotion_usage	satisfaction_score
0	1	27	Male	40682	Bachelor	East	Gold	frequent	18249	Books	0	6
1	2	29	Male	15317	Masters	West	Regular	rare	4557	Clothing	1	6
2	3	37	Male	38849	Bachelor	West	Silver	rare	11822	Clothing	0	6
3	4	30	Male	11568	HighSchool	South	Regular	frequent	4098	Food	0	7
4	5	31	Female	46952	College	North	Regular	occasional	19685	Clothing	1	5
...	...	...	...	...	...	...	...	...	...	...	...	...
99995	99996	31	Female	19691	College	West	Regular	occasional	7075	Health	0	7
99996	99997	36	Male	17428	HighSchool	South	Regular	rare	6873	Health	0	5
99997	99998	29	Male	13222	College	West	Regular	frequent	5152	Clothing	0	5
99998	99999	31	Female	40093	Bachelor	West	Regular	frequent	16312	Health	1	5
99999	100000	35	Female	22249	College	West	Silver	rare	9426	Health	0	6

100000 rows × 12 columns

```
[ ] df.shape
```

```
(100000, 12)
```

The dataset comprises of **100,000 rows and 12 columns**, with a mix of categorical and numeric variables.

Exploratory Data Analysis

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 12 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   id                  100000 non-null  int64
 1   age                 100000 non-null  int64
 2   gender              100000 non-null  object
 3   income              100000 non-null  int64
 4   education           100000 non-null  object
 5   region              100000 non-null  object
 6   loyalty_status      100000 non-null  object
 7   purchase_frequency  100000 non-null  object
 8   purchase_amount     100000 non-null  int64
 9   product_category    100000 non-null  object
 10  promotion_usage     100000 non-null  int64
 11  satisfaction_score  100000 non-null  int64
dtypes: int64(6), object(6)
memory usage: 9.2+ MB
```

```
df.isnull().sum()
```

```
id                    0
age                   0
gender                0
income                0
education             0
region                0
loyalty_status        0
purchase_frequency    0
purchase_amount       0
product_category      0
promotion_usage       0
satisfaction_score    0
dtype: int64
```

```
df.duplicated().sum()
```

```
0
```

```
df.describe()
```

	id	age	income	purchase_amount	promotion_usage	satisfaction_score
count	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000
mean	50000.500000	30.003260	27516.269880	9634.790840	0.300800	5.009650
std	28867.657797	4.480535	12996.782587	4799.339449	0.458608	1.038714
min	1.000000	12.000000	5000.000000	1118.000000	0.000000	0.000000
25%	25000.750000	27.000000	16271.750000	5583.000000	0.000000	4.000000
50%	50000.500000	30.000000	27584.500000	9452.000000	0.000000	5.000000
75%	75000.250000	33.000000	38747.250000	13350.000000	1.000000	6.000000
max	100000.000000	49.000000	50000.000000	26204.000000	1.000000	10.000000

- From the EDA, we can see that the data set is clean without any null or duplicated values.
- From the descriptive statistics, we find a large spread of data in the **income and purchase amount** attributes.

Histogram

Age and Satisfaction Score:

Both have distributions that are roughly normal, meaning most users fall around the average values.

Income:

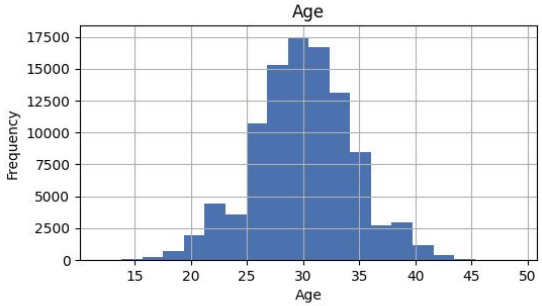
Shows a uniform distribution, indicating that users are evenly spread across the income range.

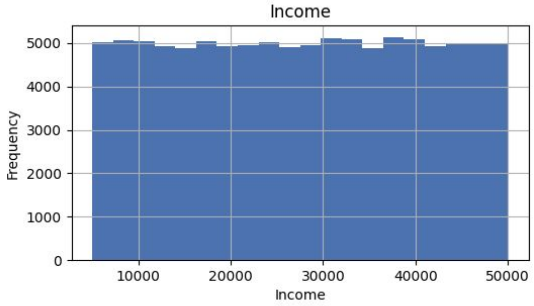
Purchase Amount:

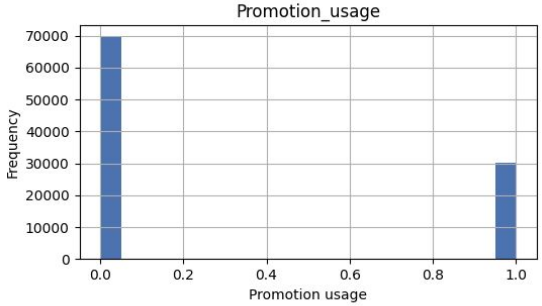
Indicates that most users make smaller purchases, with fewer users making larger ones.

Promotion Usage:

Highlights a clear divide, with the majority of users either not using promotions at all or using them extensively.







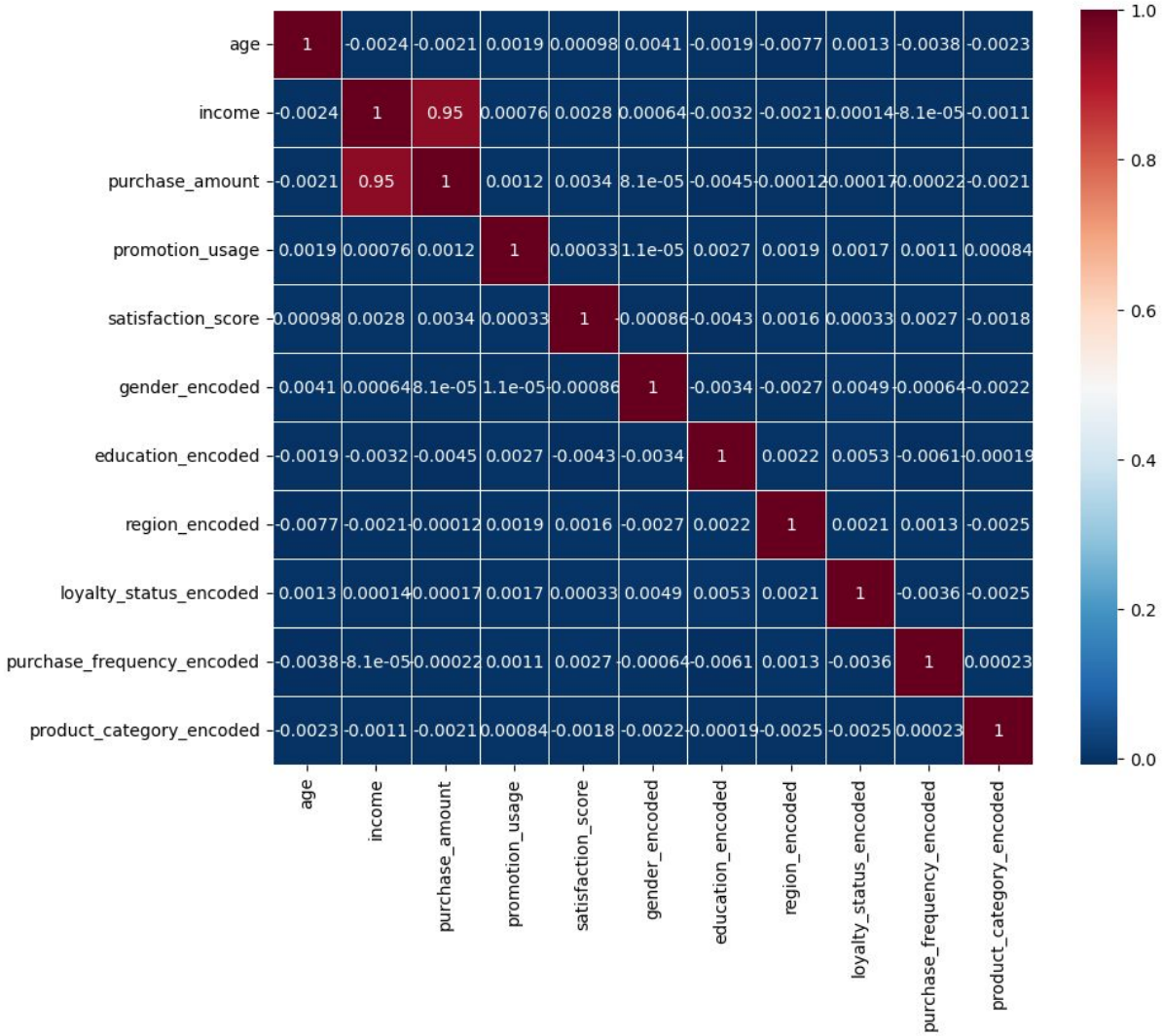
Correlation Matrix

Strong Correlation:

The only notable strong correlation is between income and purchase amount which is **0.95** and indicates that people with higher incomes tend to spend more.

Low/No Correlation:

Most other variables show very low or no correlation with each other, indicating that these features are largely independent of one another in this dataset.



Unsupervised Learning

K - Means Clustering

Using the rest of the variables, is there any other useful information / hidden patterns we can find from the dataset?

Clustering - Preprocessing step

Step 1: Standardize the Data Standardize all features to ensure they contribute equally to the clustering algorithm.

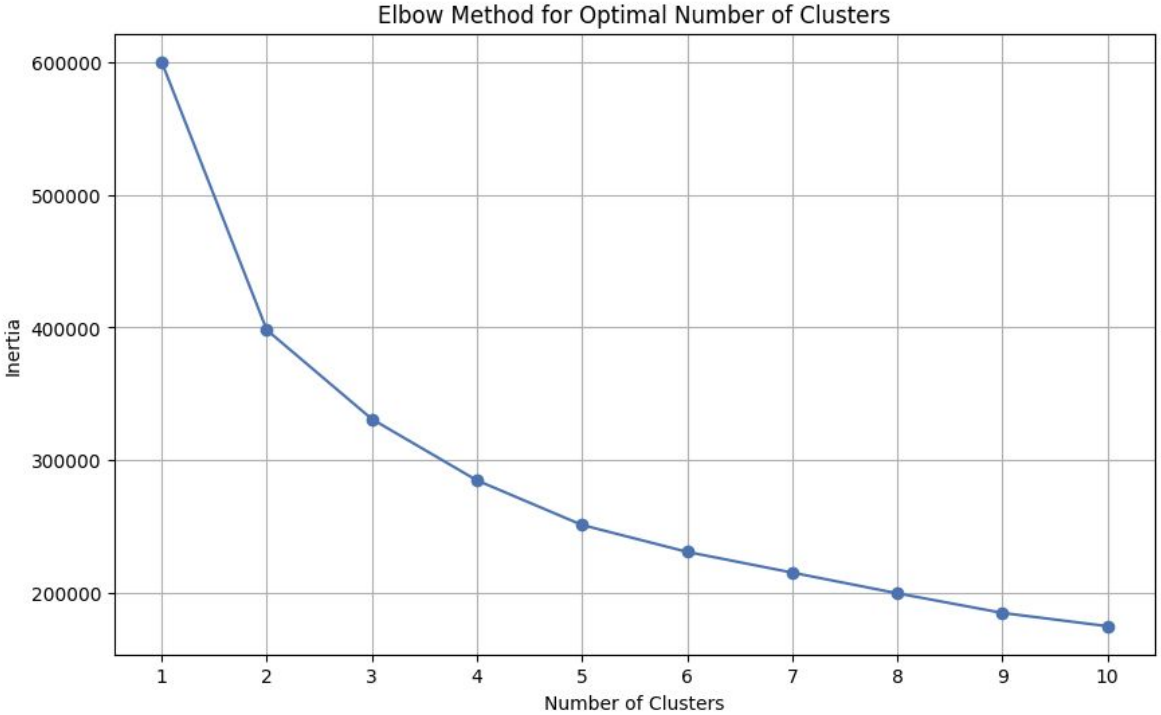
```
from sklearn.preprocessing import StandardScaler

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df_data)
```

+ Code + Text

Step 2: Determine the optimal number of clusters in K-Means clustering using the "Elbow Plot".

```
[ ] import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Range of clusters to try
cluster_range = range(1, 11)
inertia = []

# Calculate inertia for each number of clusters
for k in cluster_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

# Plot the elbow method
plt.figure(figsize=(10, 6))
plt.plot(cluster_range, inertia, marker='o')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal Number of Clusters')
plt.xticks(cluster_range)
plt.grid(True)
plt.show()
```

Clustering - Determining optimal clusters using Elbow Plot



In the provided graph, you would typically see the plot with a sharp bend or elbow at a certain point.

As the plot flattens noticeably after K=3, that means adding more clusters beyond 3 doesn't provide much improvement. Thus, **K=3** would be the optimal number of clusters.

Clustering

Step 3: Perform Clustering

```
from sklearn.cluster import KMeans

# Perform K-Means clustering
kmeans = KMeans(n_clusters=3, random_state=42)  # You can choose the number of clusters based on your data
clusters = kmeans.fit_predict(X_scaled)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4
  warnings.warn(
```

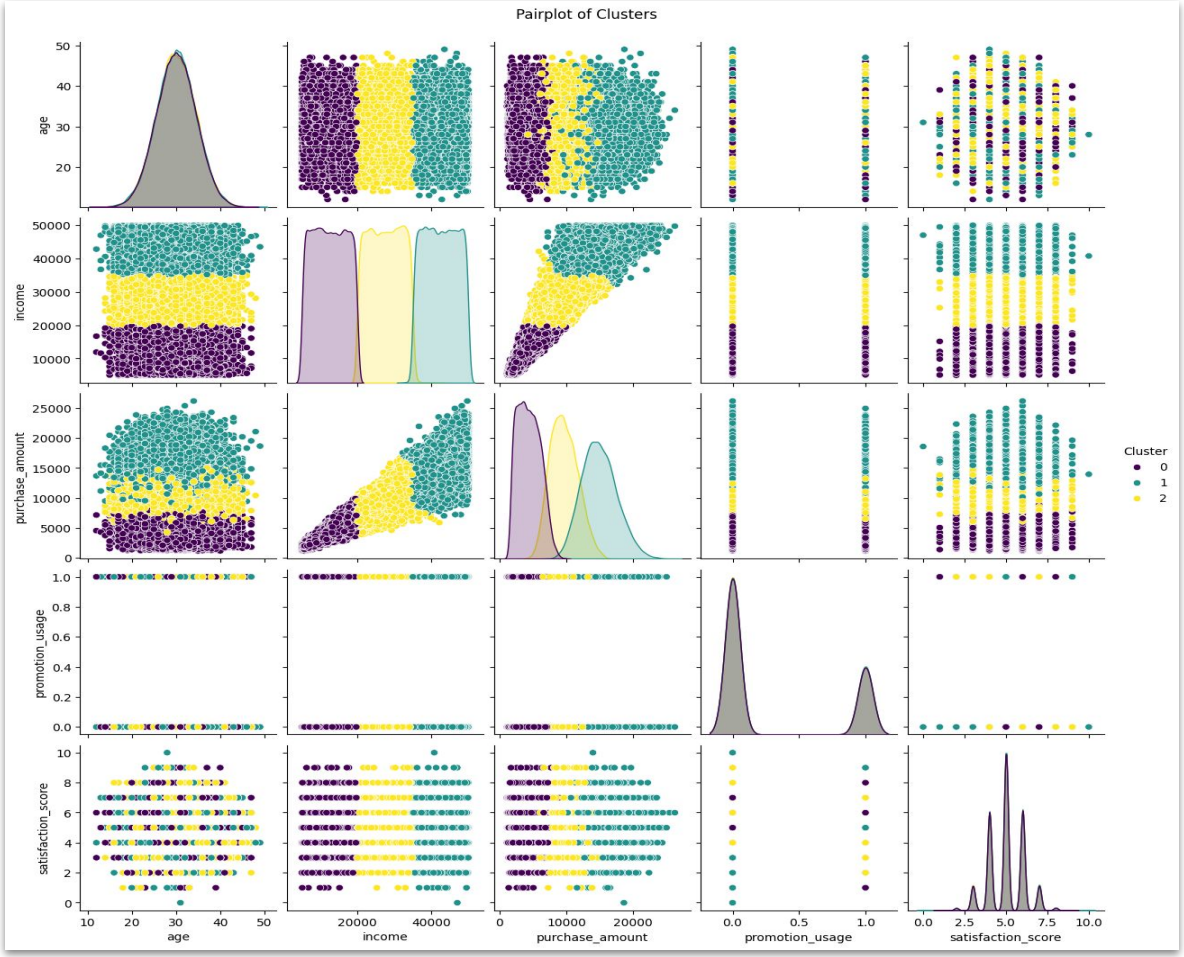
Step 4: Add Cluster Labels to the Data

```
# Add cluster labels to the original data
df_data['Cluster'] = clusters
```

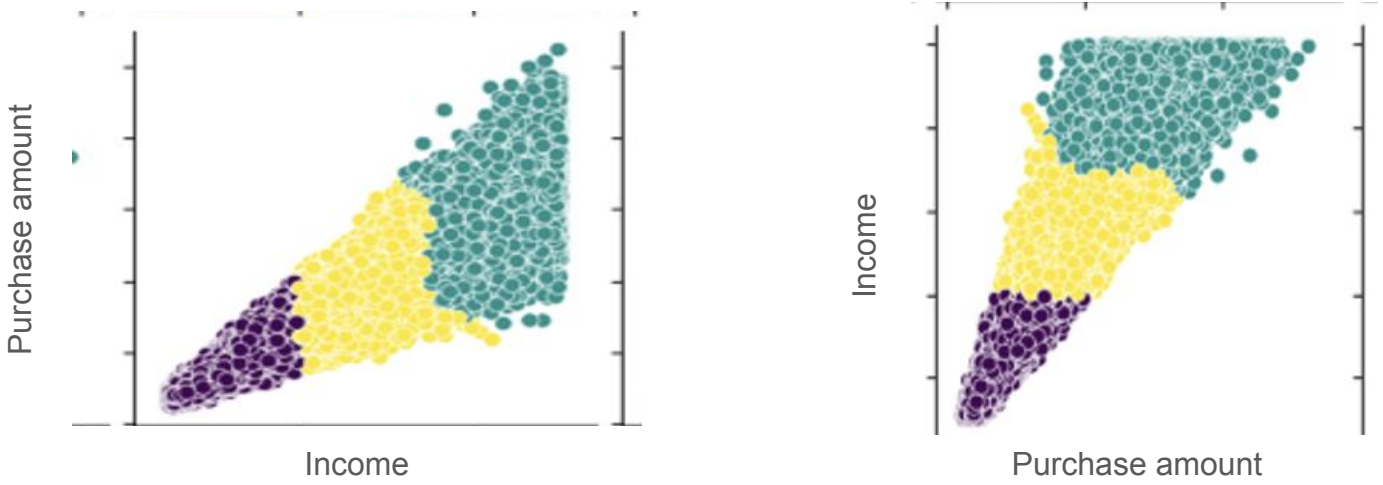
Step 5: Visualize the Clusters Visualize the clusters using a pairplot to see the relationships between different features in the context of clusters.

```
import seaborn as sns
import matplotlib.pyplot as plt

# Visualize clusters using a pairplot
sns.pairplot(df_data, hue='Cluster', palette='viridis')
plt.suptitle('Pairplot of Clusters', y=1.02)
plt.show()
```

Clustering - Pairplot to visualize interesting patterns



Pair plot Interpretation



Cluster 1 - represents **low**-income, low-purchase customers.

Cluster 2 - represents **middle**-income, moderate-purchase customers.

Cluster 3 - represents **high**-income, high-purchase customers.

Techniques Used

1. Regression
    a. Linear Regression
    b. KNN Regression
    c. Random Forest Regression

2. Classification
    a. Logistic Regression
    b. Decision Tree Classifier
    c. Random Forest

Supervised Learning

Regression

Linear Regression with all variables

```python
import statsmodels.formula.api as smf

formula = (
    'purchase_amount ~ age + income + promotion_usage + satisfaction_score + '
    'gender_encoded + education_encoded + region_encoded + '
    'loyalty_status_encoded + purchase_frequency_encoded + product_category_encoded'
)
# Fit the OLS model
m = smf.ols(formula=formula, data=df).fit()

# Print the model summary
print(m.summary())
```

OLS Regression Results

Dep. Variable:	purchase_amount	R-squared:	0.900
Model:	OLS	Adj. R-squared:	0.900
Method:	Least Squares	F-statistic:	8.954e+04
Date:	Wed, 17 Jul 2024	Prob (F-statistic):	0.00
Time:	22:11:04	Log-Likelihood:	-8.7461e+05
No. Observations:	100000	AIC:	1.749e+06
Df Residuals:	99989	BIC:	1.749e+06
Df Model:	10		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-13.5476	45.266	-0.299	0.765	-102.267	75.172
age	0.1757	1.074	0.164	0.870	-1.929	2.280
income	0.3502	0.000	946.239	0.000	0.350	0.351
promotion_usage	4.6160	10.489	0.440	0.660	-15.943	25.175
satisfaction_score	3.5849	4.631	0.774	0.439	-5.492	12.662
gender_encoded	-5.0745	9.621	-0.527	0.598	-23.932	13.783
education_encoded	-7.3358	5.108	-1.436	0.151	-17.348	2.676
region_encoded	7.4387	3.995	1.862	0.063	-0.391	15.268
loyalty_status_encoded	-2.3773	8.036	-0.296	0.767	-18.128	13.373
purchase_frequency_encoded	-0.9892	6.151	-0.161	0.872	-13.046	11.067
product_category_encoded	-3.4269	3.215	-1.066	0.286	-9.728	2.874

Omnibus:	4366.455	Durbin-Watson:	1.995
Prob(Omnibus):	0.000	Jarque-Bera (JB):	15069.964
Skew:	0.006	Prob(JB):	0.00
Kurtosis:	4.902	Cond. No.	2.87e+05

The statistical output shows that only income is significant associated with the target variable. **(P-value < 0.05)**.

Since all other values are not significant we won't use them in our analysis as changes in the predictor do not reliably predict changes in the response.

Linear Regression with Income

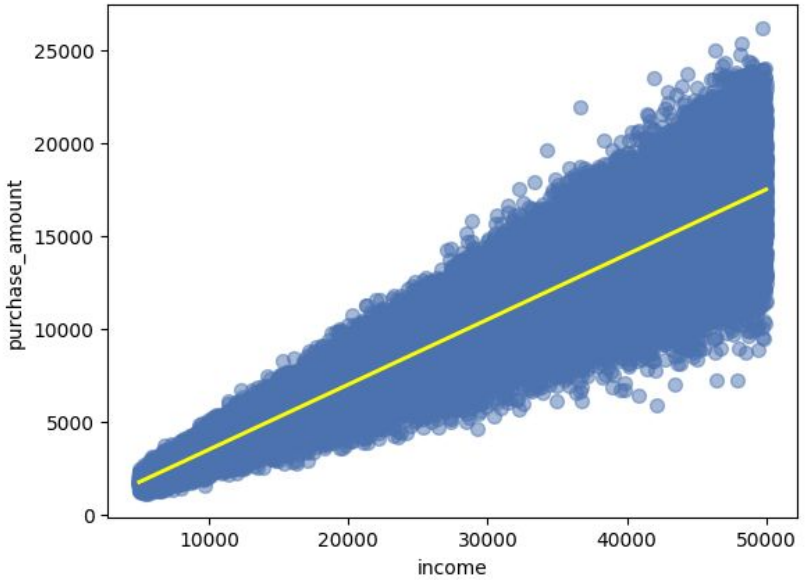
```python
from sklearn.model_selection import train_test_split

X = df[['income']]  # Independent variable
y = df['purchase_amount']  # Dependent variable

# Split the Data into Training and Testing Sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

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

# Train the Linear Regression Model
lr_model1 = LinearRegression()
lr_model1.fit(X_train, y_train)

# Make Predictions
y_pred = lr_model1.predict(X_test)

# Evaluate the Model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Calculate Adjusted R-squared
n = len(y_test)  # Number of observations
p = X.shape[1]  # Number of predictors
adjusted_r2 = 1 - (1 - r2) * (n - 1) / (n - p - 1)

print(f'Mean Squared Error: {mse}')
print(f'R^2 Score: {r2}')
print(f'Adjusted R^2 Score: {adjusted_r2}')
```

```
Mean Squared Error: 2337296.785760103
R^2 Score: 0.8980430941778186
Adjusted R^2 Score: 0.898037995822692
```

We used backward elimination by to eventually be left with **Income**.

This simplifies model with less predictors and giving us a high adj R^2



KNN Regression with Income

```python
from sklearn.neighbors import KNeighborsRegressor
```

```python
#Train the KNN Regression Model
# Instantiate the model with n_neighbors with 5 n neighbors.

knn_model_all = KNeighborsRegressor(n_neighbors=5)
knn_model_all.fit(X_train, y_train)

# Make Predictions
y_pred = knn_model_all.predict(X_test)

# Evaluate the Model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'R^2 Score: {r2}')
```

```
Mean Squared Error: 2819841.7381379995
R^2 Score: 0.8769936534032007
```

```python
# Train the KNN Regression Model with k = 15

knn_model = KNeighborsRegressor(n_neighbors=15)
knn_model.fit(X_train, y_train)

# Make Predictions
y_pred = knn_model.predict(X_test)

# Evaluate the Model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'R^2 Score: {r2}')
```

```
Mean Squared Error: 2492386.604826222
R^2 Score: 0.891277809523837
```

Decent Performance, but lower than Linear Regression.
We notice the highest improvement in k=15.

(still lower than Linear Regression)

Random Forest Regressor with Income

```python
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Create the Random Forest Regressor
rf = RandomForestRegressor(n_estimators=100, random_state=42)

# Train the model
rf.fit(X_train, y_train)

# Make predictions on the test set
y_pred = rf.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

##Adjusted R2
n = len(y_test)
k = X_test.shape[1]

adj_r2 = 1 - (1 - r2) * (n - 1) / (n - k - 1)

print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
print(f"Adjusted r-squared: {adj_r2}")
```

Lowest R^2 and Highest RMSE. Random Forest Regressor shows the most poor results.

```
Mean Squared Error: 3237826.701043468
R-squared: 0.8587604304091504
Adjusted r-squared: 0.8587533677244024
```

Evaluation of Regression Models

Cross Validation in Regression (5-fold vs 10-fold)

```python
from sklearn.model_selection import cross_val_score
import numpy as np
```

CV = 5

```python
# Evaluate Linear Regression with cross-validation
lr_scores = cross_val_score(lr_model1, X, y, cv=5, scoring='neg_mean_squared_error')
lr_rmse_scores = np.sqrt(-lr_scores)

# Evaluate KNN Regression with cross-validation
knn_scores = cross_val_score(knn_model, X, y, cv=5, scoring='neg_mean_squared_error')
knn_rmse_scores = np.sqrt(-knn_scores)

# Evaluate Random Forest Regression with cross-validation
rf_scores = cross_val_score(rf, X, y, cv=5, scoring='neg_mean_squared_error')
rf_rmse_scores = np.sqrt(-rf_scores)

# Display the RMSE scores
print("Linear Regression RMSE: Mean =", lr_rmse_scores.mean(), "Std =", lr_rmse_scores.std())
print("KNN Regression RMSE: Mean =", knn_rmse_scores.mean(), "Std =", knn_rmse_scores.std())
print("Random Forest RMSE: Mean =", rf_rmse_scores.mean(), "Std =", rf_rmse_scores.std())
#print("Random Forest RMSE with multiple independent variables: Mean =", rf_all_rmse_scores.mea
#print("KNN Regression w multiple RMSE: Mean =", knn_all_rmse_scores.mean(), "Std =", knn_all_
```

```
Linear Regression RMSE: Mean = 1521.1960734577538 Std = 10.786188721885734
KNN Regression RMSE: Mean = 1570.303711255338 Std = 9.43634104822446
Random Forest RMSE: Mean = 1786.5361413217452 Std = 10.14451093385385
```

CV = 10

```python
# Evaluate Linear Regression with cross-validation
lr_scores = cross_val_score(lr_model1, X, y, cv=10, scoring='neg_mean_squared_error')
lr_rmse_scores = np.sqrt(-lr_scores)

# Evaluate KNN Regression with cross-validation
knn_scores = cross_val_score(knn_model, X, y, cv=10, scoring='neg_mean_squared_error')
knn_rmse_scores = np.sqrt(-knn_scores)

# Evaluate Random Forest Regression with cross-validation
rf_scores = cross_val_score(rf, X, y, cv=10, scoring='neg_mean_squared_error')
rf_rmse_scores = np.sqrt(-rf_scores)

# Display the RMSE scores
print("Linear Regression RMSE: Mean =", lr_rmse_scores.mean(), "Std =", lr_rmse_scores.std())
print("KNN Regression RMSE: Mean =", knn_rmse_scores.mean(), "Std =", knn_rmse_scores.std())
print("Random Forest RMSE: Mean =", rf_rmse_scores.mean(), "Std =", rf_rmse_scores.std())
#print("Random Forest RMSE with multiple independent variables: Mean =", rf_all_rmse_scores.me
#print("KNN Regression w multiple RMSE: Mean =", knn_all_rmse_scores.mean(), "Std =", knn_all_

print()
```

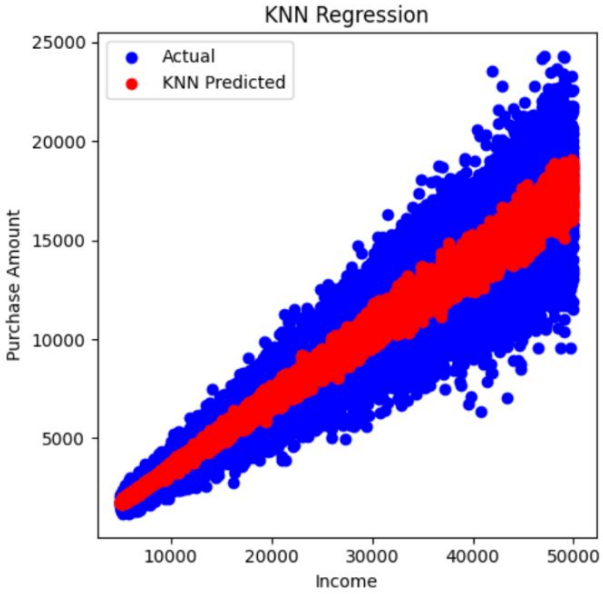
```
Linear Regression RMSE: Mean = 1521.116781442805 Std = 18.676564855851097
KNN Regression RMSE: Mean = 1569.4907294513625 Std = 17.19399046330268
Random Forest RMSE: Mean = 1780.3115648656974 Std = 17.37703826567577
```

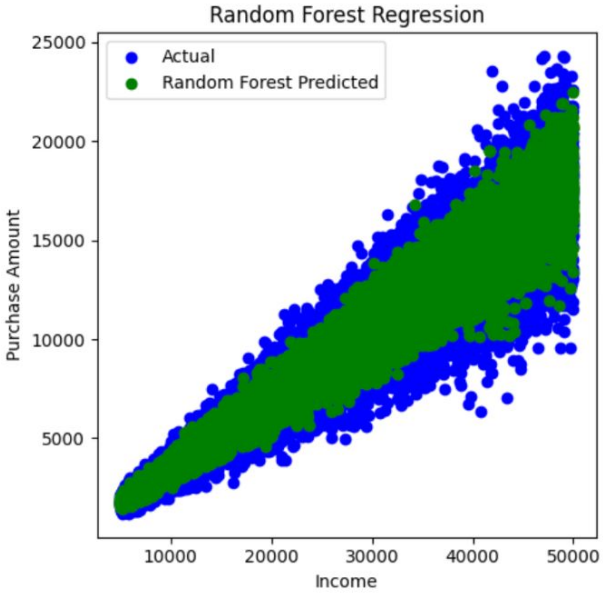
Model Fit Comparison using Scatter Plots

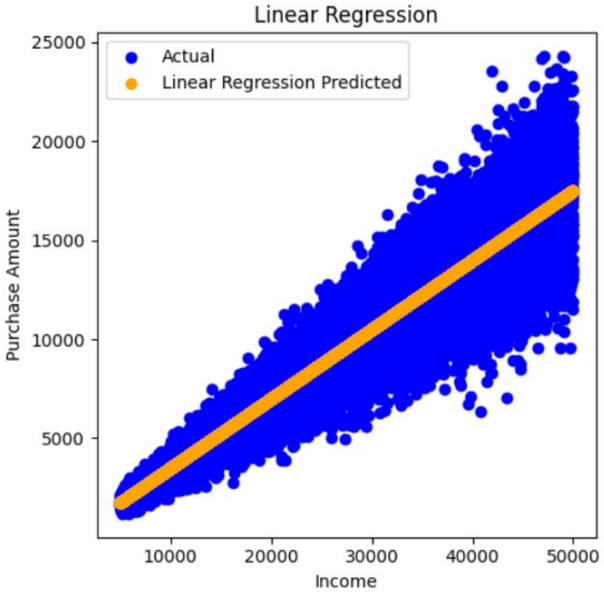
KNN performs reasonably well but shows more deviation in predictions.

Best fits and captures complexity of the model

Struggles to capture complexity of model, especially at higher values

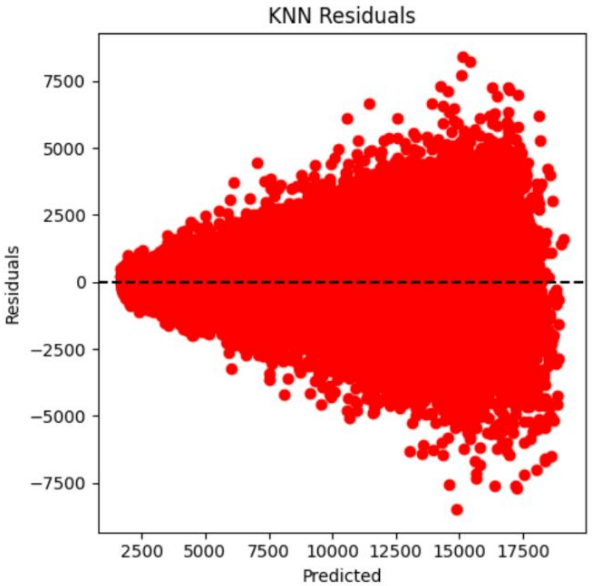




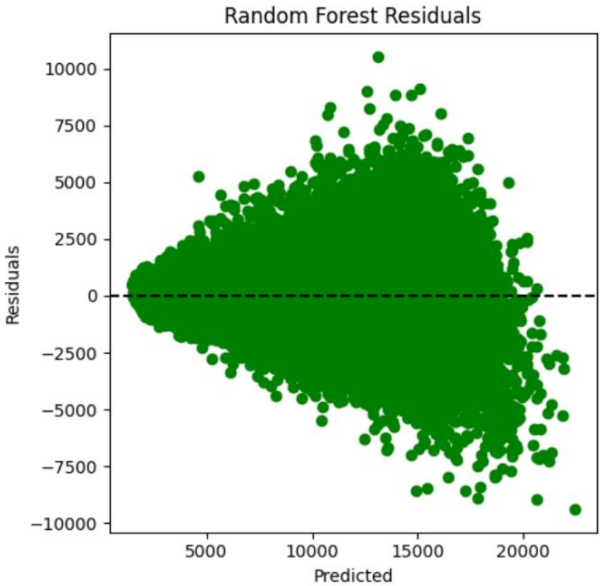


Residual Plot Comparison

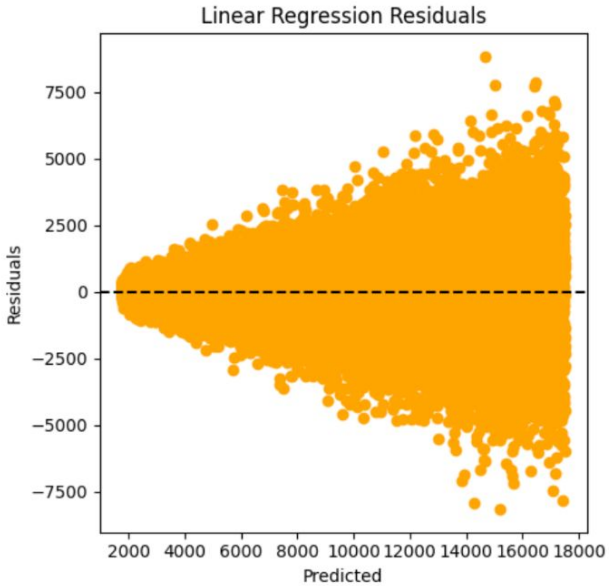
Heteroscedasticity, with increasing error at higher predicted values



Heteroscedasticity, with increasing error at higher predicted values, Handles better than KNN



Clear pattern in residuals, (might be missing some non-linear patterns in the data).

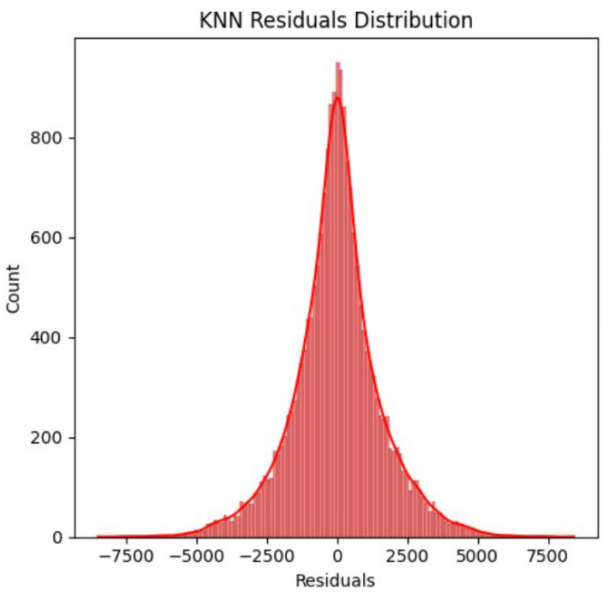


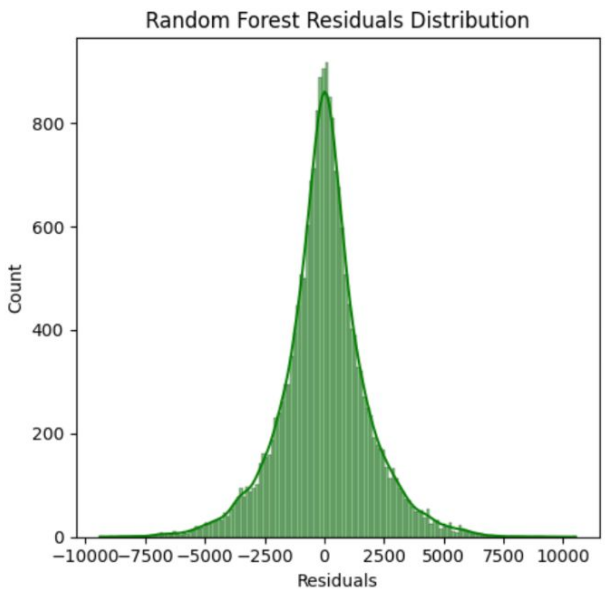
Comparison of Residuals Distribution

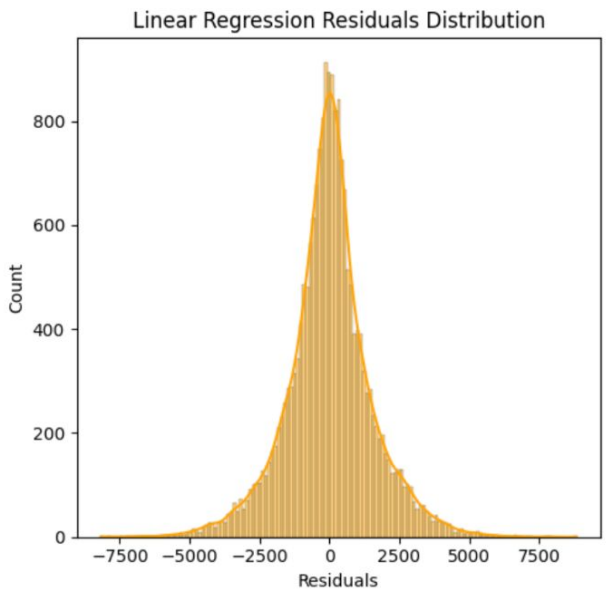
KNN residuals suggest a reasonably well-fitting model, but with some variance in predictions.

Higher variance and potential **overfitting**, despite the overall good performance.

Linear regression residuals suggest a well-fitting model with consistent error distribution.







Highest Spread - -10000 to +10000

Comparison of Learning Curves

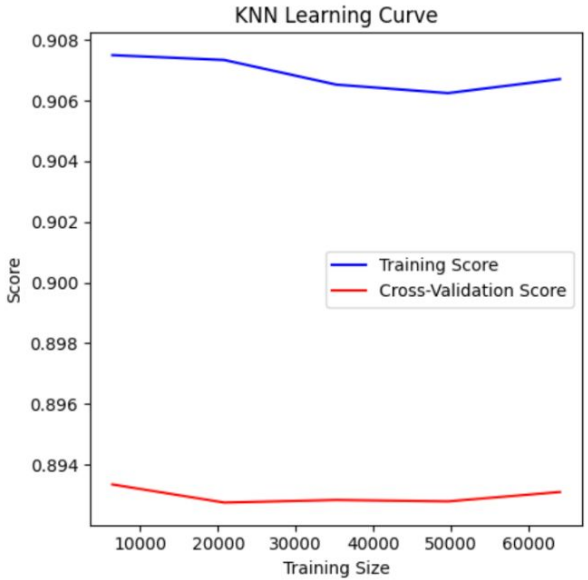
Training Score **0.907**, CV Score **0.894**

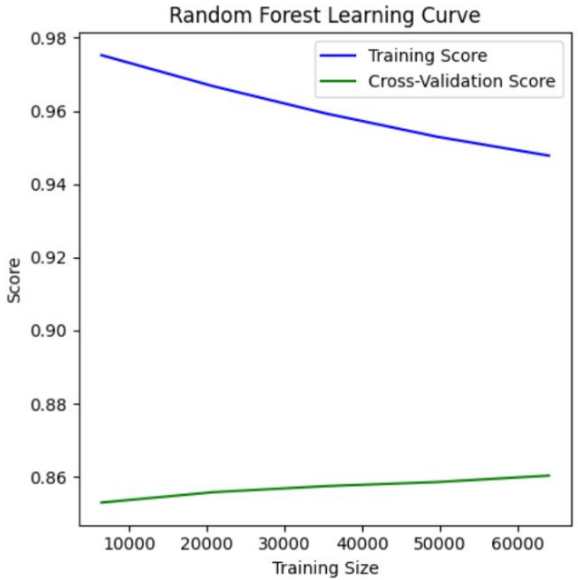
This suggests a slight **overfitting**, but generally **good performance**.

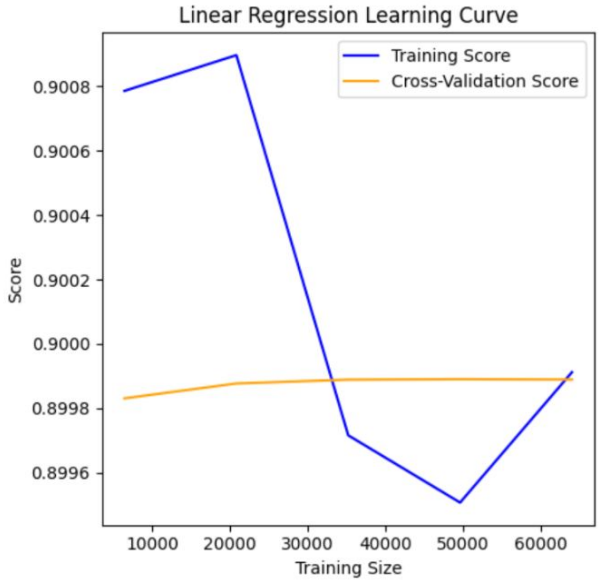
This indicates significant **overfitting**, with the model performing very well on training data **(0.96)** but less so on unseen data **(0.86).**

Increasing data helps reduce overfitting.

Linear regression shows **consistent performance** across training and validation sets, with minimal overfitting. **0.8998,**







Chosen Model and Analysis

KNN: Performs well with slightly higher overfitting, but generally stable and good performance. (Lower than Linear Regression, higher spread)

Random Forest: Shows the best training performance but **significant overfitting**, making it less reliable for unseen data. Requires additional tuning and more data collection to improve performance.

Linear Regression: Has the most **consistent performance** across training and validation sets, indicating reliable generalization. Despite lower complexity, it avoids significant overfitting and provides stable predictions.

*Highest CV score with **0.8998,** lowest RMSE score **1521***

Supervised Learning

Classification

Classification - Logistic regression Vs Decision Trees Vs Random Forest

```python
# Define a binary target variable
threshold = data['purchase_amount'].median()  # Using median to split high vs. low spenders
data['target'] = (data['purchase_amount'] > threshold).astype(int)

# Drop the original purchase_amount column
X = data.drop(['purchase_amount', 'target'], axis=1)
y = data['target']
```

```python
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
```

```python
from sklearn.preprocessing import StandardScaler

# Standardize the data for logistic regression
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```python
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

# Initialize the models
log_reg = LogisticRegression(random_state=42)
dec_tree = DecisionTreeClassifier(random_state=42)
rand_forest = RandomForestClassifier(random_state=42)

# Train the models
log_reg.fit(X_train_scaled, y_train)
dec_tree.fit(X_train, y_train)  # Decision Tree does not need scaling
rand_forest.fit(X_train, y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier(random_state=42)
```

1. **Define a binary target variable**
   - ❏ Purchase amount (0/1)
2. **Data Splitting**
   - ❏ Train - 70% , Test - 30%
   - ❏ Random state - 42
3. **Data Standardization**
4. **Model initialization & Training**
   - ❏ Logistic Regression
   - ❏ Decision Tree
   - ❏ Random Forest

Classification - Logistic regression Vs Decision Trees Vs Random Forest

5. Model Prediction and Evaluation

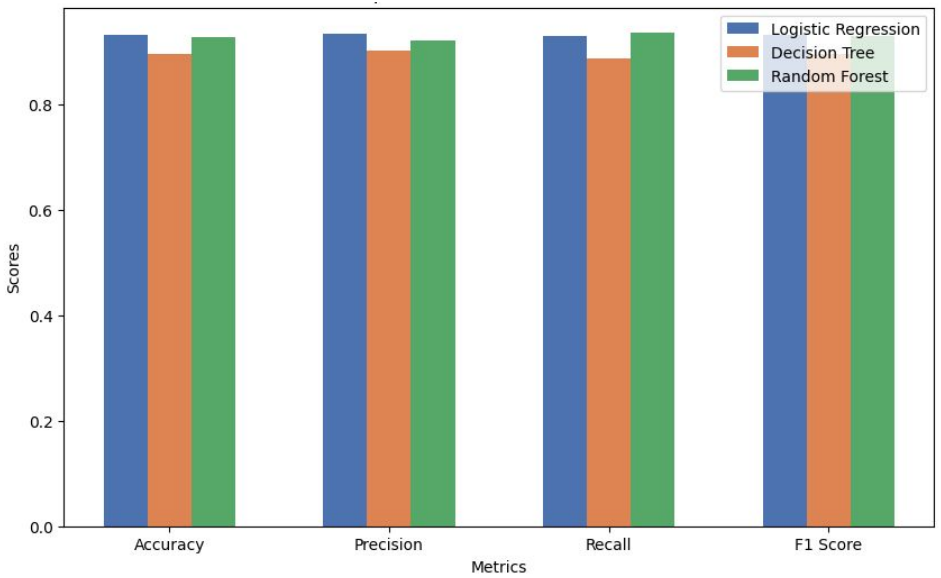
```python
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Make predictions
y_pred_log_reg = log_reg.predict(X_test_scaled)
y_pred_dec_tree = dec_tree.predict(X_test)
y_pred_rand_forest = rand_forest.predict(X_test)

# Evaluate the models
def evaluate_model(y_test, y_pred):
    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)
    return accuracy, precision, recall, f1

log_reg_metrics = evaluate_model(y_test, y_pred_log_reg)
dec_tree_metrics = evaluate_model(y_test, y_pred_dec_tree)
rand_forest_metrics = evaluate_model(y_test, y_pred_rand_forest)

# Print the results
print(f"Logistic Regression: Accuracy={log_reg_metrics[0]:.2f}, Precision={log_reg_metrics[1]:.2f}, Recall={log_reg_metrics[2]:.2f}, F1 Score={log_reg_metrics[3]:.2f}")
print(f"Decision Tree: Accuracy={dec_tree_metrics[0]:.2f}, Precision={dec_tree_metrics[1]:.2f}, Recall={dec_tree_metrics[2]:.2f}, F1 Score={dec_tree_metrics[3]:.2f}")
print(f"Random Forest: Accuracy={rand_forest_metrics[0]:.2f}, Precision={rand_forest_metrics[1]:.2f}, Recall={rand_forest_metrics[2]:.2f}, F1 Score={rand_forest_metrics[3]:.2f}")

```

```
Logistic Regression: Accuracy=0.93, Precision=0.93, Recall=0.93, F1 Score=0.93
Decision Tree: Accuracy=0.90, Precision=0.90, Recall=0.89, F1 Score=0.89
Random Forest: Accuracy=0.93, Precision=0.92, Recall=0.94, F1 Score=0.93
```

Metrics to assess model performance - Accuracy_score, Precision_score, Recall_score, F1_score

Visualizing Model Performance in Classification

	Accuracy	Precision	Recall	F1 Score
Logistic Regression	0.93	**0.93**	0.93	0.93
Decision Tree	0.90	0.90	0.89	0.89
Random Forest	0.93	0.92	**0.94**	0.93

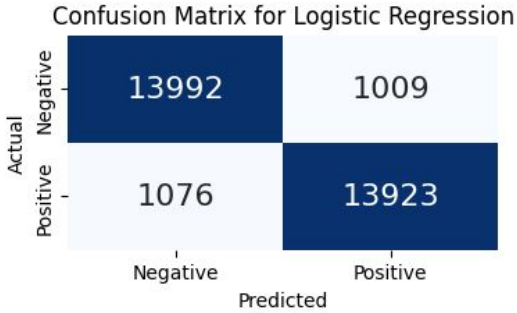


Random Forest appears to be the **best model** in terms of recall and has competitive accuracy and F1 Score, making it the most effective at identifying positive cases and maintaining overall performance.

Logistic Regression performs equally well in accuracy and F1 Score as Random Forest but is slightly lower in recall.

Decision Tree has the lowest metrics across precision, recall, and F1 Score, indicating it performs slightly worse than both Logistic Regression and Random Forest.

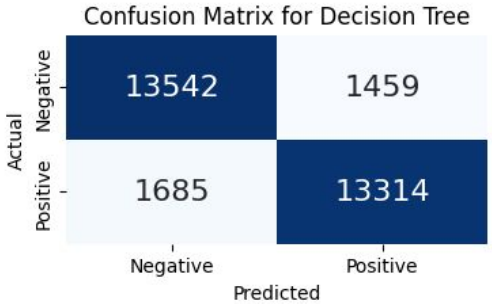
Confusion matrix for the Classification Models



High true positives and true negatives, indicating strong classification ability.

Relatively low false positives and false negatives.

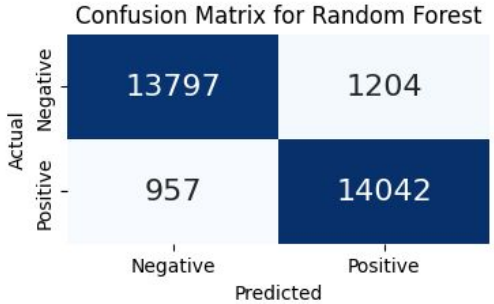
Good performance overall with balanced error rates.



Slightly lower true positives and true negatives compared to Logistic Regression.

Higher false positives and false negatives.

Lower performance with higher error rates.



Highest true positives and true negatives, and the lowest false negatives.

Slightly higher false positives than Logistic Regression but lower than Decision Tree.

Best performance among the three models, most effective at classification.

Evaluation of Classification Models

Cross Validation in Classification (5-fold vs 10-fold)

CV = 5

```python
from sklearn.model_selection import cross_val_score

# Perform cross-validation and print the results
log_reg_cv = cross_val_score(log_reg, X_scaled, y, cv=5, scoring='accuracy')
dec_tree_cv = cross_val_score(dec_tree, X, y, cv=5, scoring='accuracy')
rand_forest_cv = cross_val_score(rand_forest, X, y, cv=5, scoring='accuracy')

log_reg_cv_f1 = cross_val_score(log_reg, X_scaled, y, cv=5, scoring='f1')
dec_tree_cv_f1 = cross_val_score(dec_tree, X, y, cv=5, scoring='f1')
rand_forest_cv_f1 = cross_val_score(rand_forest, X, y, cv=5, scoring='f1')

print(f"Logistic Regression CV Accuracy: {log_reg_cv.mean():.2f} ± {log_reg_cv.std():.2f}")
print(f"Logistic Regression CV F1 Score: {log_reg_cv_f1.mean():.2f} ± {log_reg_cv_f1.std():.2f}\n")

print(f"Decision Tree CV Accuracy: {dec_tree_cv.mean():.2f} ± {dec_tree_cv.std():.2f}")
print(f"Decision Tree CV F1 Score: {dec_tree_cv_f1.mean():.2f} ± {dec_tree_cv_f1.std():.2f}\n")

print(f"Random Forest CV Accuracy: {rand_forest_cv.mean():.2f} ± {rand_forest_cv.std():.2f}")
print(f"Random Forest CV F1 Score: {rand_forest_cv_f1.mean():.2f} ± {rand_forest_cv_f1.std():.2f}")
```

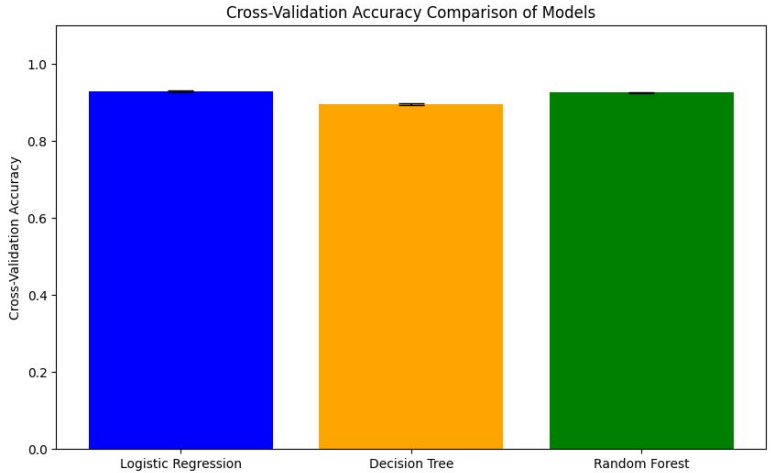
```
Logistic Regression CV Accuracy: 0.93 ± 0.00
Logistic Regression CV F1 Score: 0.93 ± 0.00

Decision Tree CV Accuracy: 0.89 ± 0.00
Decision Tree CV F1 Score: 0.89 ± 0.00

Random Forest CV Accuracy: 0.93 ± 0.00
Random Forest CV F1 Score: 0.93 ± 0.00
```

CV = 10

```python
from sklearn.model_selection import cross_val_score
X_scaled = scaler.transform(X) # Scale the features using the defined scaler
# Perform cross-validation and print the results
log_reg_cv = cross_val_score(log_reg, X_scaled, y, cv=10, scoring='accuracy')
dec_tree_cv = cross_val_score(dec_tree, X, y, cv=10, scoring='accuracy')
rand_forest_cv = cross_val_score(rand_forest, X, y, cv=10, scoring='accuracy')

log_reg_cv_f1 = cross_val_score(log_reg, X_scaled, y, cv=10, scoring='f1')
dec_tree_cv_f1 = cross_val_score(dec_tree, X, y, cv=10, scoring='f1')
rand_forest_cv_f1 = cross_val_score(rand_forest, X, y, cv=10, scoring='f1')

print(f"Logistic Regression CV Accuracy: {log_reg_cv.mean():.2f} ± {log_reg_cv.std():.2f}")
print(f"Logistic Regression CV F1 Score: {log_reg_cv_f1.mean():.2f} ± {log_reg_cv_f1.std():.2f}\n")

print(f"Decision Tree CV Accuracy: {dec_tree_cv.mean():.2f} ± {dec_tree_cv.std():.2f}")
print(f"Decision Tree CV F1 Score: {dec_tree_cv_f1.mean():.2f} ± {dec_tree_cv_f1.std():.2f}\n")

print(f"Random Forest CV Accuracy: {rand_forest_cv.mean():.2f} ± {rand_forest_cv.std():.2f}")
print(f"Random Forest CV F1 Score: {rand_forest_cv_f1.mean():.2f} ± {rand_forest_cv_f1.std():.2f}")
```

```
Logistic Regression CV Accuracy: 0.93 ± 0.00
Logistic Regression CV F1 Score: 0.93 ± 0.00

Decision Tree CV Accuracy: 0.90 ± 0.00
Decision Tree CV F1 Score: 0.89 ± 0.00

Random Forest CV Accuracy: 0.93 ± 0.00
Random Forest CV F1 Score: 0.93 ± 0.00
```

Interpretation of Cross-Validation Results



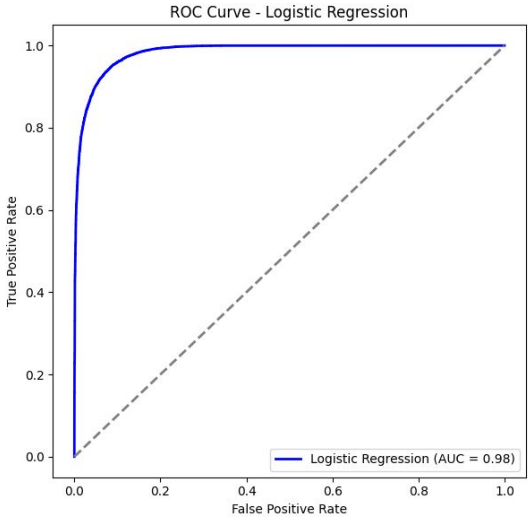
Logistic Regression and Random Forest:

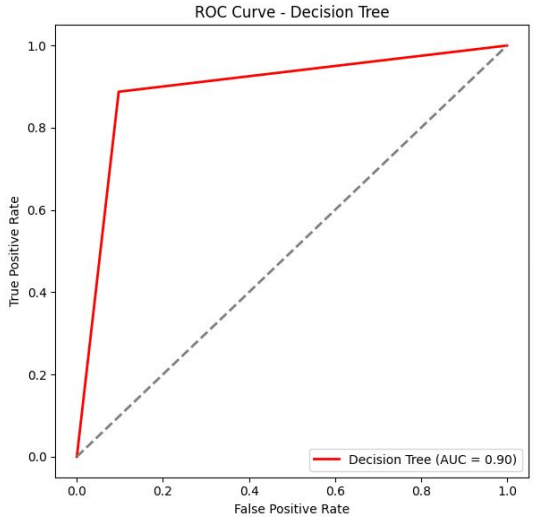
- Both models achieve **high and consistent performance**, with a CV accuracy and F1 score of **0.93.** This indicates that they are reliable and effective at classification tasks.

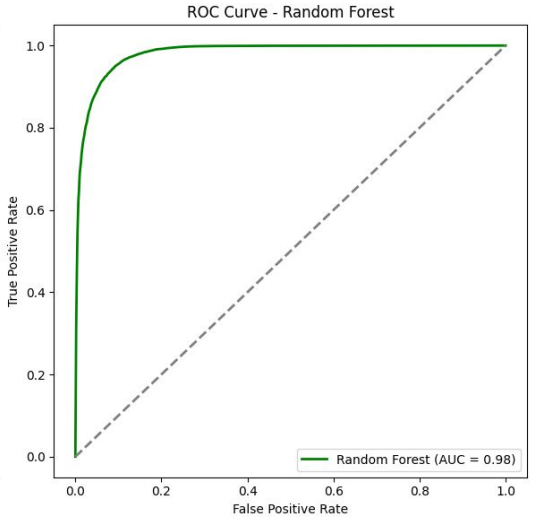
Decision Tree:

- The decision tree model shows **slightly lower performance,** with a CV accuracy of **0.90** and an F1 score of 0.89. While still consistent, it is less effective than logistic regression and random forest.

Computing the ROC - AUC Curve







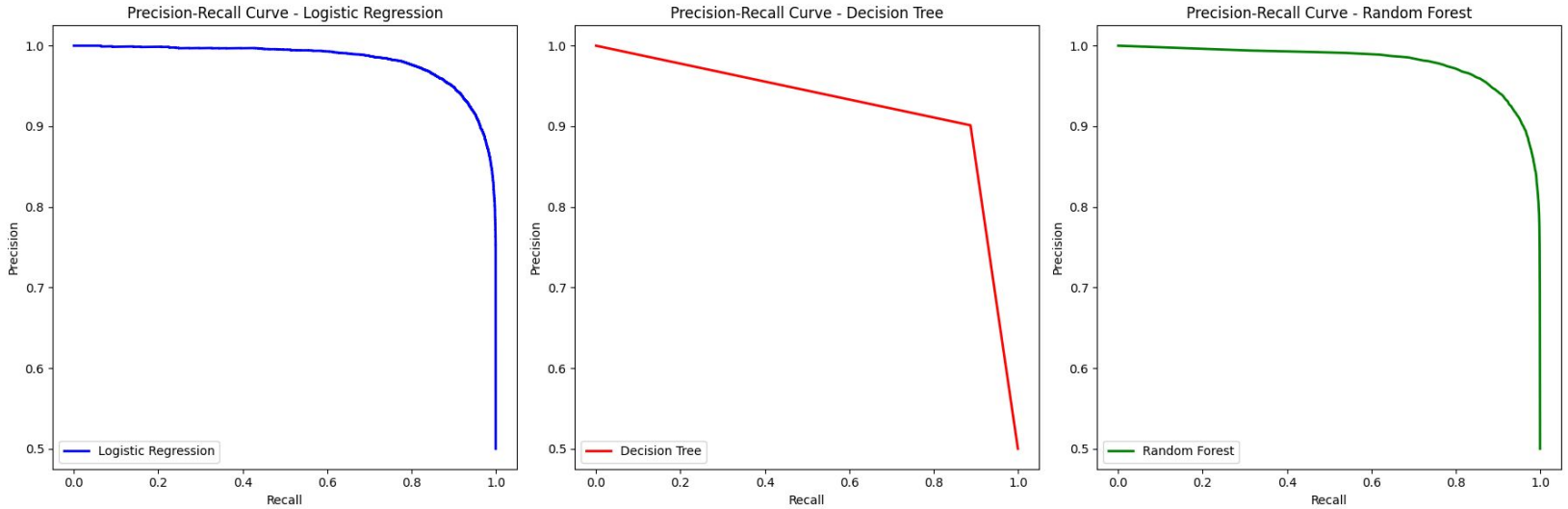
Logistic Regression and Random Forest:

- Both models have an **AUC of 0.98**, indicating exceptional performance and high reliability in distinguishing between positive and negative classes.

Decision Tree:

- The model has an **AUC of 0.90**, showing good performance but slightly less effective compared to Logistic Regression and Random Forest.

Computing the Precision - Recall Curve



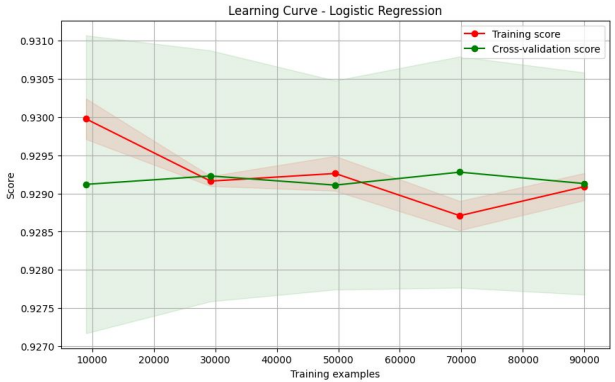
Logistic Regression and Random Forest:

- **High precision** over a wide range of recall values but sharp drop at high recall. Best for minimizing false positives.

Decision Tree:

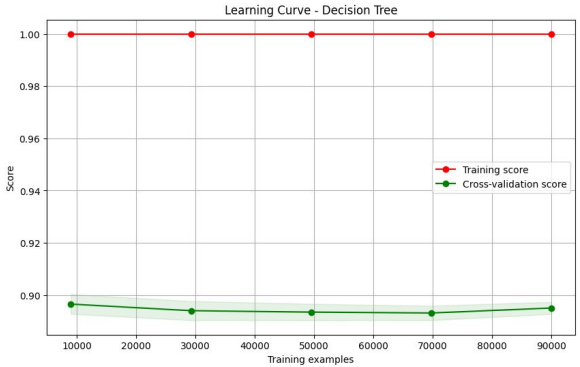
- **Lower precision** and more balanced but less effective than Logistic Regression and Random Forest in handling imbalanced data.

Comparison of Learning Curves



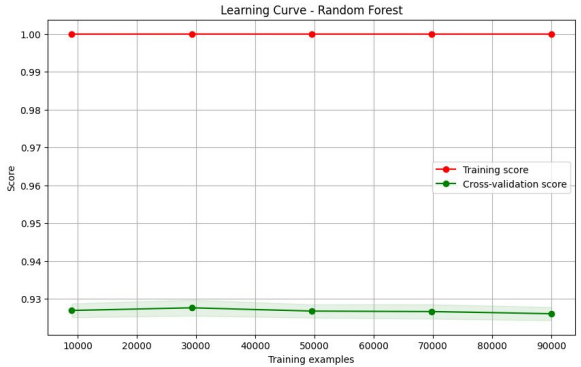
Shows **moderate overfitting** but with close scores, indicating reasonable generalization.

Regularization techniques to reduce overfitting.



Shows **severe overfitting** with a large gap between training and validation scores.

Pruning techniques or limiting the depth of the tree to prevent overfitting.



Shows potentially **less severe overfitting** than the Decision Tree, suggesting a need for further tuning.

Adjusting hyperparameters, such as the number of trees, max depth, or using more data to improve generalization.

Chosen Model and Analysis

Logistic Regression: Offers a **balanced performance** with **minimal overfitting**. It generalizes well to new data and provides stable predictions. It is a good choice if simplicity and interpretability are important.

Decision Trees: Shows **significant overfitting** and **lower performance** metrics, making it less reliable for unseen data. It may require pruning or other techniques to improve generalization.

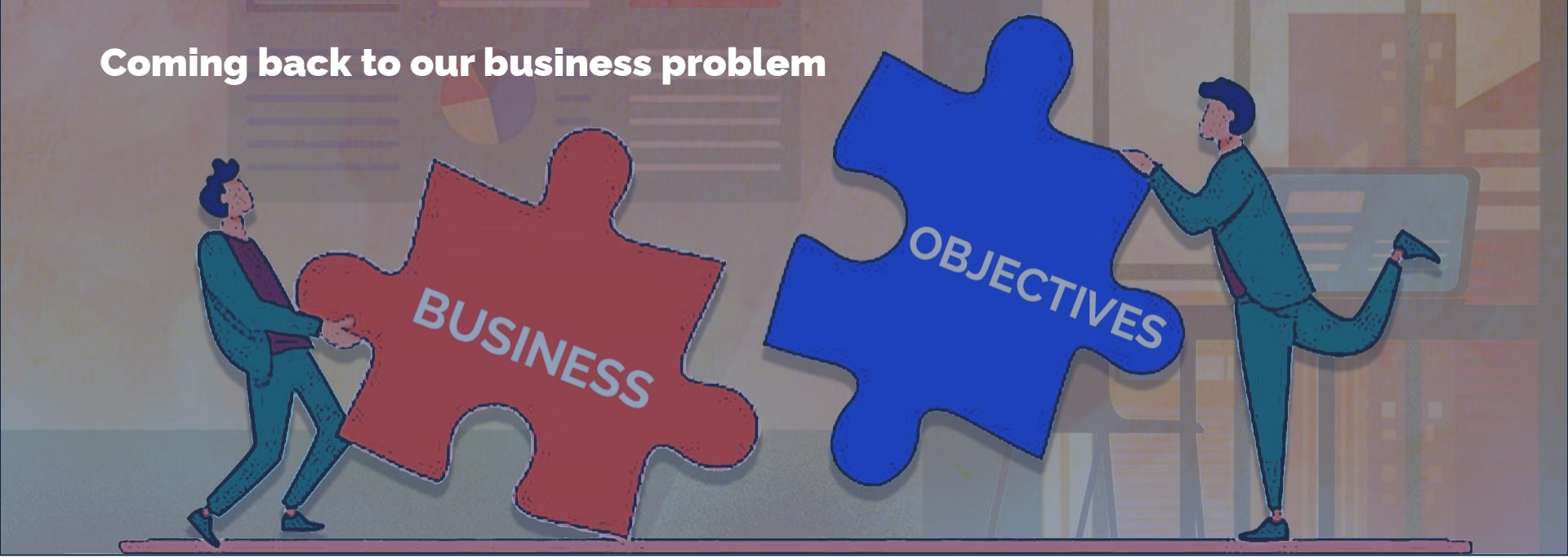
Random Forest: Demonstrates the **best overall performance** but still shows **signs of overfitting**. It achieves high accuracy and F1 scores and is a strong candidate with further tuning.

Conclusion

Comparison of the techniques

- Clustering and Correlation showed patterns of income vs purchase amount.
- The best regression model selected was **Linear Regression.**
- The best Classification model selected was **Random forest.**

Coming back to our business problem



How can we identify and analyze the factors affecting customer purchase amounts to enhance our sales strategy?

The main objective of this project is to identify factors influencing purchase amount, to smartly invest our marketing and advertising spend on the right demographic and to predict a reliable purchase amount given our predictors.

Comparison of the techniques

Which model do we choose?

Linear Regression	Random Forest Classification
Simpler and more interpretable, has coefficients that explain relationship b/w predictors and outcome. Easy for stakeholders to understand.	**Complex Models** that are less interpretable due to ensemble nature and interaction b/w trees.
We require predicting a **continuous purchase amount** given our predictors. Linear Regression predicts a value that can be any real number.	In Random Forest, we're not able to get the exact purchase amount, instead only the **class** where the prediction will fall. (High accuracy does not matter)
Linear Regression is **less computationally intensive**, advantage because dataset is of 100k observations.	RF classification was highly **computationally intensive**, fine tuning and minor changes would cause a lot of wait
Showed consistent performance across training and validation sets, **minimal overfitting.**	**Highly prone to overfitting**, learning curve showed a huge difference in performance between training and validation sets

Thank You

Note: The following slides are for your reference.

Clustering - Model Evaluation

Step 6: Analyze the Cluster Centers Examine the cluster centers to understand the characteristics of each cluster.

```
# Cluster centers in original scale
cluster_centers = scaler.inverse_transform(kmeans.cluster_centers_)

# Create a DataFrame for cluster centers
centers_df = pd.DataFrame(cluster_centers, columns=df_data.columns)  # Exclude the 'Cluster' column
centers_df['Cluster'] = range(1, len(centers_df) + 1)

# Display cluster centers
print(centers_df)
```

```
         age        income  purchase_amount  promotion_usage  \
0  29.998703  12448.407551      4360.368015         0.300280
1  29.987687  42471.278354     14882.932339         0.302621
2  30.023397  27493.073607      9613.416477         0.299491

   satisfaction_score  Cluster
0            5.005790        1
1            5.013329        2
2            5.009796        3
```

```
centers_df
```

	age	income	purchase_amount	promotion_usage	satisfaction_score	Cluster
0	29.998703	12448.407551	4360.368015	0.300280	5.005790	1
1	29.987687	42471.278354	14882.932339	0.302621	5.013329	2
2	30.023397	27493.073607	9613.416477	0.299491	5.009796	3

Evaluating the Model - Regression

```python
# Make predictions on the test set
y_pred = best_model.predict(X_test)

print("Best Model Selected:", best_model_text)
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R-squared:", r2_score(y_test, y_pred))
print()

# Display the predictions
print("Predictions:", y_pred[:10])
```

```
Best Model Selected: Linear Regression
Mean Squared Error: 2337283.0990580847
R-squared: 0.898043691215301

Predictions: 75721     2231.488454
80184     7083.605105
19864     4186.134262
76699     6322.550573
92991     4539.518534
76434    11563.775178
84004    15814.192940
80917    10377.538778
60767     4110.834343
50074    14935.110163
dtype: float64
```

Evaluating the Model - Classification

```python
new_data = pd.DataFrame({
    'age': [30],
    'income': [18000],
    'promotion_usage': [0],
    'satisfaction_score': [5],
    'gender_encoded': [1],
    'education_encoded': [3],
    'region_encoded': [0],
    'loyalty_status_encoded':[2],
    'purchase_frequency_encoded':[2],
    'product_category_encoded':[3],
})

# Standardize the new data
new_data_scaled = scaler.transform(new_data)

# Predictions on the new data
new_pred_log_reg = log_reg.predict(new_data_scaled)
new_pred_dec_tree = dec_tree.predict(new_data)
new_pred_rand_forest = rand_forest.predict(new_data)

print(f"Logistic Regression Prediction: {new_pred_log_reg}")
print(f"Decision Tree Prediction: {new_pred_dec_tree}")
print(f"Random Forest Prediction: {new_pred_rand_forest}")
```

```
Logistic Regression Prediction: [0]
Decision Tree Prediction: [0]
Random Forest Prediction: [0]
```

When predicting the results on the test set, all three models—Logistic Regression, Decision Tree, and Random Forest—consistently predicted the class 0 (Low spend class), which corresponds to a purchase amount below the threshold value of $10,000.