ASSIGNMENT ONE

COURSE: DATA SCIENCE

PROFESSOR: M.Sc. Mohammad Ziyad Kagdi

FRIONA POCARI

BINF III A

Table of Contents

I.INTRODUCTION

i.PROJECT OVERVIEW

This assignment expects you to make a use of multiple machine learning algorithms to make predictions from the following datasets. Refer to our lecture notes and practicals to use appropriate algorithms as per the outline defined.

Datasets

ii. DATASET DESCRIOTION

1) The dataset "film_collection_dataset.csv" contains information about movies and their marketing, production expense, budget of the movie, length of the movie, critic rating etc and the money earned.

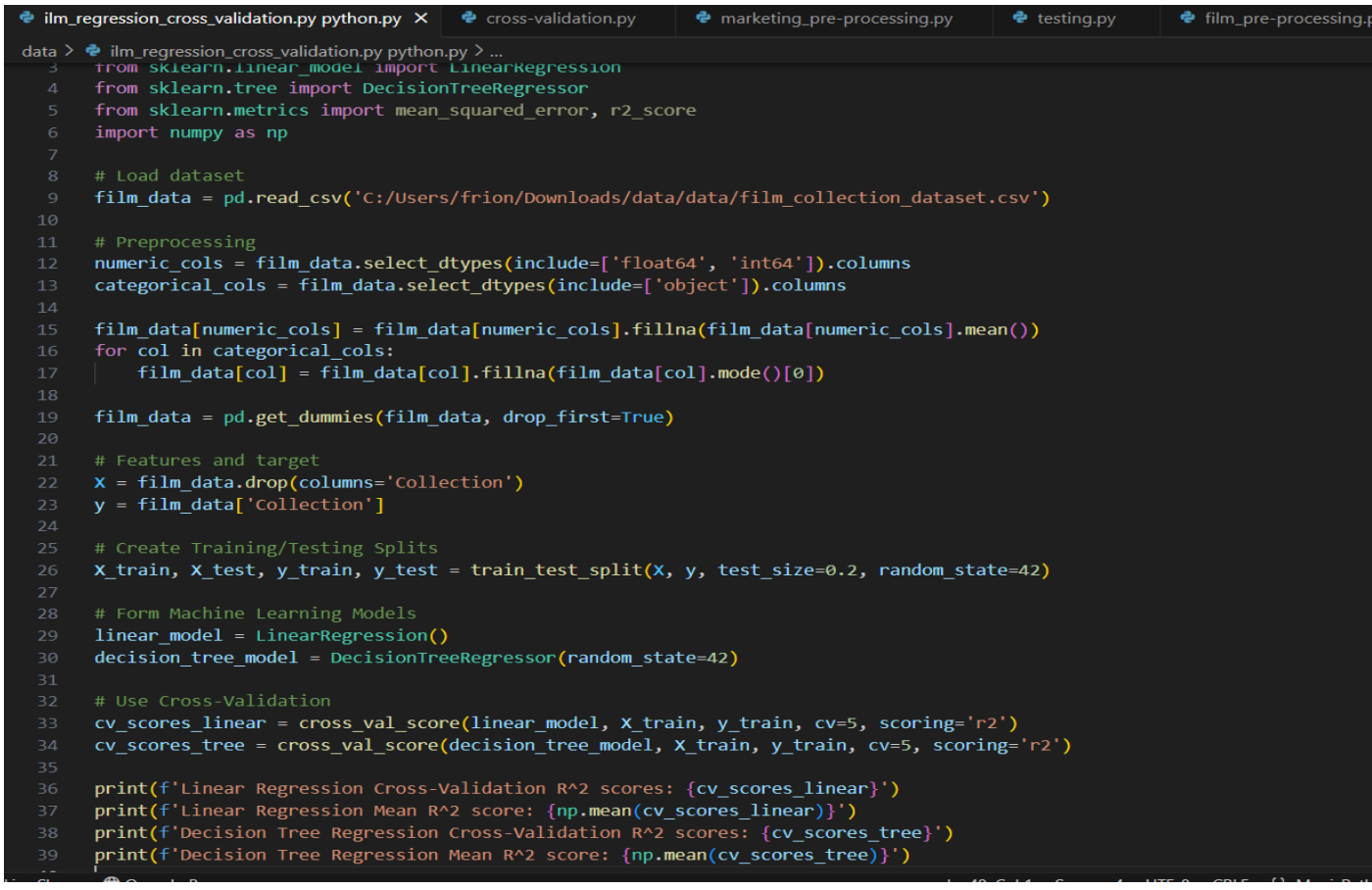
2) The dataset "loan_dataset.csv" contains people's personal information and a classification field "loan_status" states whether their request to loan was approved based on their education, income and credit score.

3) The dataset "marketing_campaign_dataset.csv" contains data about people's education, marital status, income, number of kids in the household etc and their preferences to multiple products and their binary response (acceptance/rejection) to multiple offers made in campaigns (from columns AcceptedCmp1 to AcceptedCmp2 and the response column). The dataset also contains information about the amount of money spent on products such as Gold, Fruits, Meat, Fish, Sweets and Wines in the last two years.

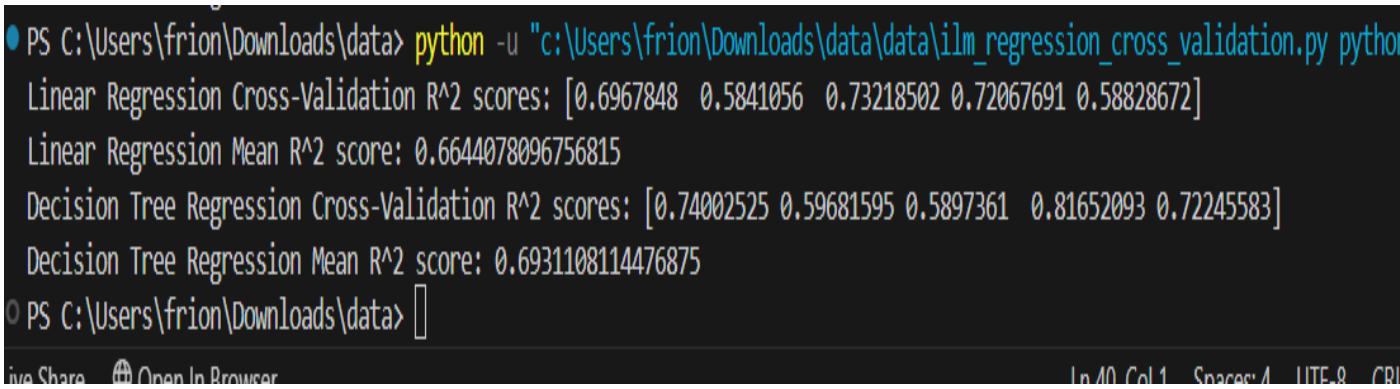
II.DATA PRE-PROCCESSING.

1. Create optimum training/testing split to form appropriate machine learning models for both classification and regression problems and also make a use of cross validation methods to avoid model overfitting problems.

The Code :



The Output :



To address this task, we need to create optimal training/testing splits for both classification and regression problems and implement cross-validation to avoid model overfitting.

This is a Regression Problem: Film Collection Dataset

Step-by-Step Implementation:

1. **Load and Preprocess the Data**
2. **Create Training/Testing Splits**
3. **Form Machine Learning Models**
4. **Use Cross-Validation**

Conclusions :

Conclusion from Cross-Validation Output

Linear Regression Cross-Validation R^2 scores:

- Individual scores: [0.6967848, 0.5841056, 0.73218502, 0.72067691, 0.58828672]
- Mean R^2 score: 0.6644078096756815

Decision Tree Regression Cross-Validation R^2 scores:

- Individual scores: [0.74002525, 0.59681595, 0.5897361, 0.81652093, 0.72245583]
- Mean R^2 score: 0.6931108114476875

5. **Model Performance Comparison**:

- The Decision Tree Regression model generally **outperforms** the Linear Regression model in terms of R^2 scores.
- The mean R^2 score for Decision Tree Regression (0.693) is higher than that for Linear Regression (0.664), indicating better overall performance in capturing the variability of the target variable.

6. **Consistency**:
   - The R^2 scores for both models vary across the different cross-validation folds, which is expected due to the natural variation in the dataset.
   - Decision Tree Regression shows a broader range of R^2 scores, indicating it might be more sensitive to the specific splits of the data.
7. **Robustness**:
   - Both models have relatively high mean R^2 scores, suggesting they are generally good at explaining the variability in the target variable.
   - The Decision Tree Regression's higher mean R^2 score suggests it may be more robust in handling the complexities and interactions in the dataset compared to the Linear Regression model.
8. **Practical Implications**:
   - When choosing between these two models for predicting movie collections, the Decision Tree Regression model is likely the better choice given its superior performance in cross-validation.
   - However, it is also essential to consider other factors such as model interpretability, training time, and computational resources. Linear Regression is simpler and easier to interpret, whereas Decision Tree Regression can capture non-linear relationships better.

Overall, the cross-validation results indicate that the Decision Tree Regression model is a more suitable choice for this regression problem, providing better performance in terms of explaining the variance in the target variable compared to the Linear Regression model.

Part II.

Classification Problem: Loan Dataset

Step-by-Step Implementation:

9. **Load and Preprocess the Data**
10. **Create Training/Testing Splits**
11. **Form Machine Learning Models**
12. **Use Cross-Validation**

THE CODE :

```python
# Load dataset
loan_data = pd.read_csv('C:/Users/frion/Downloads/data/data/loan_dataset.csv')

# Preprocessing
loan_data.columns = loan_data.columns.str.strip()
loan_data = pd.get_dummies(loan_data, drop_first=True)

# Inspect column names to verify correct creation
print("Column names in the dataset after preprocessing:")
print(loan_data.columns)

# Ensure the target column exists
target_column = 'loan_status_ Rejected'
if target_column not in loan_data.columns:
    raise KeyError(f"Column '{target_column}' not found in the dataset. Please check the preprocessing steps.")

# Features and target
X = loan_data.drop(columns=target_column)
y = loan_data[target_column]

# Create Training/Testing Splits
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Form Machine Learning Models
logistic_model = LogisticRegression(max_iter=1000)
decision_tree_model = DecisionTreeClassifier(random_state=42)
random_forest_model = RandomForestClassifier(random_state=42)

# Use Cross-Validation
cv_scores_logistic = cross_val_score(logistic_model, X_train, y_train, cv=5, scoring='accuracy')
cv_scores_tree = cross_val_score(decision_tree_model, X_train, y_train, cv=5, scoring='accuracy')
cv_scores_forest = cross_val_score(random_forest_model, X_train, y_train, cv=5, scoring='accuracy')
```

```python
print(f'Logistic Regression Cross-Validation Accuracy scores: {cv_scores_logistic}')
print(f'Logistic Regression Mean Accuracy score: {np.mean(cv_scores_logistic)}')
print(f'Decision Tree Classification Cross-Validation Accuracy scores: {cv_scores_tree}')
print(f'Decision Tree Classification Mean Accuracy score: {np.mean(cv_scores_tree)}')
print(f'Random Forest Classification Cross-Validation Accuracy scores: {cv_scores_forest}')
print(f'Random Forest Classification Mean Accuracy score: {np.mean(cv_scores_forest)}')
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```
Logistic Regression Cross-Validation Accuracy scores: [0.7920937  0.79062958 0.80673499 0.80087848 0.8374817 ]
Logistic Regression Mean Accuracy score: 0.8055636896046853
Decision Tree Classification Cross-Validation Accuracy scores: [0.98096633 0.96925329 0.97218155 0.97510981 0.98096633]
Decision Tree Classification Mean Accuracy score: 0.9756954612005856
Random Forest Classification Cross-Validation Accuracy scores: [0.97657394 0.9795022  0.97803807 0.97803807 0.9795022 ]
Random Forest Classification Mean Accuracy score: 0.9783308931185944
PS C:\Users\frion\Downloads\data>
```

Conclusion :

Logistic Regression Cross-Validation Accuracy scores:

- Individual scores: [0.7920937, 0.79062958, 0.80673499, 0.80087848, 0.8374817]
- Mean Accuracy score: 0.8055636896046853

Decision Tree Classification Cross-Validation Accuracy scores:

- Individual scores: [0.98096633, 0.96925329, 0.97218155, 0.97510981, 0.98096633]
- Mean Accuracy score: 0.9756954612005856

Random Forest Classification Cross-Validation Accuracy scores:

- Individual scores: [0.97657394, 0.9795022, 0.97803807, 0.97803807, 0.9795022]
- Mean Accuracy score: 0.9783308931185944

Conclusion:

13. **Model Performance Comparison**:
    - **Decision Tree Classification** and **Random Forest Classification** models significantly outperform **Logistic Regression** in terms of accuracy.
    - Random Forest Classification shows slightly higher mean accuracy than Decision Tree Classification, indicating it is the best-performing model among the three.
14. **Consistency**:
    - Both Decision Tree and Random Forest models exhibit very high and consistent accuracy scores across all cross-validation folds, suggesting robust performance.
    - Logistic Regression shows more variation in accuracy scores, indicating it is less consistent than the tree-based models.
15. **Practical Implications**:
    - For predicting loan approval, tree-based models (Decision Tree and Random Forest) are highly accurate and reliable.
    - Random Forest is preferred due to its slightly better accuracy and its ability to handle overfitting better than a single Decision Tree.

Overall, the Random Forest model is recommended for predicting loan approvals due to its superior accuracy and robustness.

2. Achieve necessary data pre-processing steps including outlier removals and appropriate visualisation steps such as pair plots or correlation matrix to better understand the data distribution.

For the Film Dataset :

data > film_pre-processing.py > ...

```python
import pandas as pd
from scipy import stats
import seaborn as sns
import matplotlib.pyplot as plt

# Load dataset
film_data = pd.read_csv('C:/Users/frion/Downloads/data/data/film_collection_dataset.csv')

# Separate numeric and categorical columns
numeric_cols = film_data.select_dtypes(include=['float64', 'int64']).columns
categorical_cols = film_data.select_dtypes(include=['object']).columns

# Handling missing values for numeric columns by filling with mean
film_data[numeric_cols] = film_data[numeric_cols].fillna(film_data[numeric_cols].mean())

# Handling missing values for categorical columns by filling with mode
for col in categorical_cols:
    film_data[col] = film_data[col].fillna(film_data[col].mode()[0])

# Removing outliers for the film dataset
z_scores_film = stats.zscore(film_data[numeric_cols])
abs_z_scores_film = abs(z_scores_film)
filtered_entries_film = (abs_z_scores_film < 3).all(axis=1)
film_data = film_data[filtered_entries_film]

# Visualizations for the film dataset
print("Film Dataset Pair Plot")
sns.pairplot(film_data)
plt.show()

print("Film Dataset Correlation Matrix")
correlation_matrix_film = film_data[numeric_cols].corr()
sns.heatmap(correlation_matrix_film, annot=True)
plt.show()
```

Output :

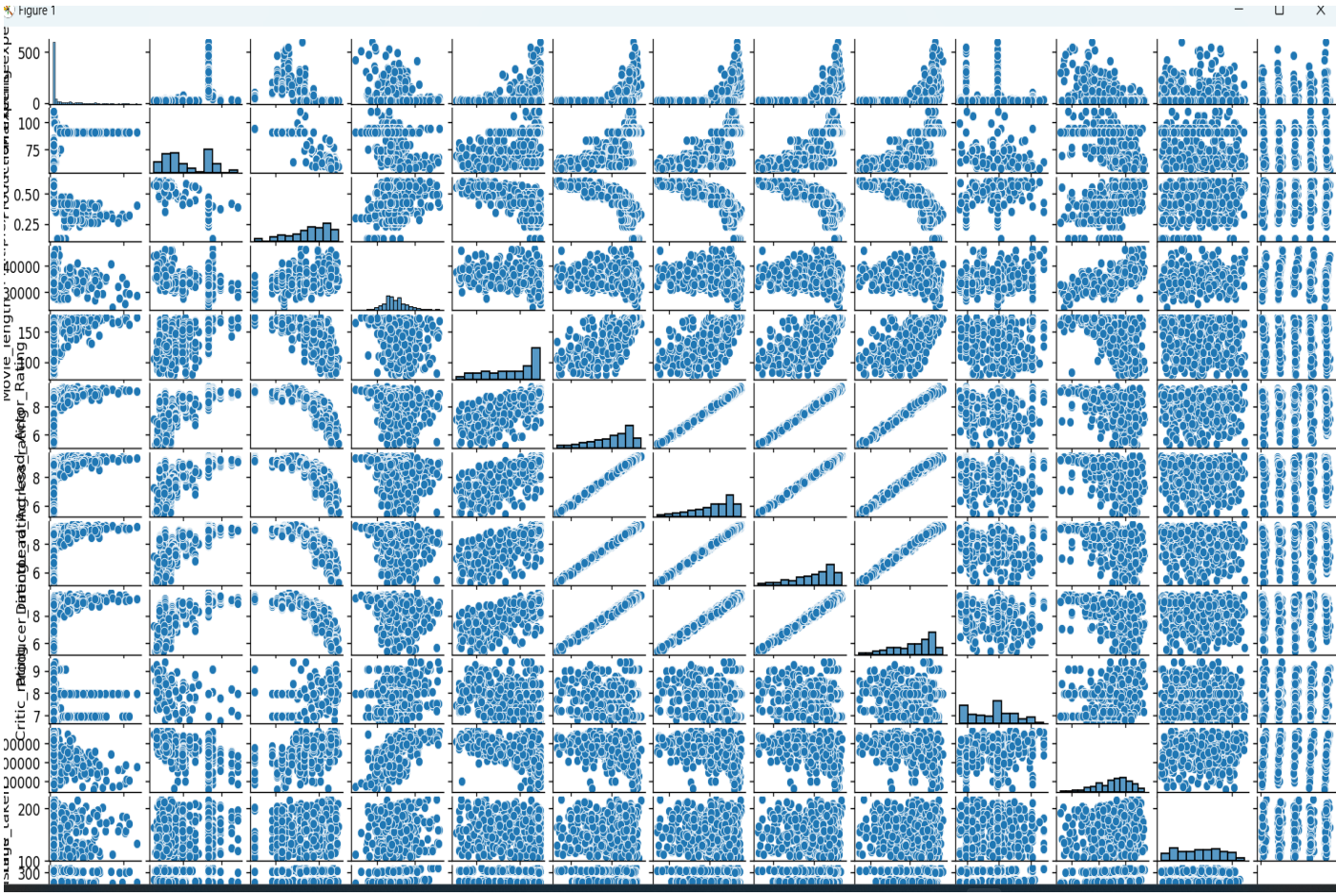
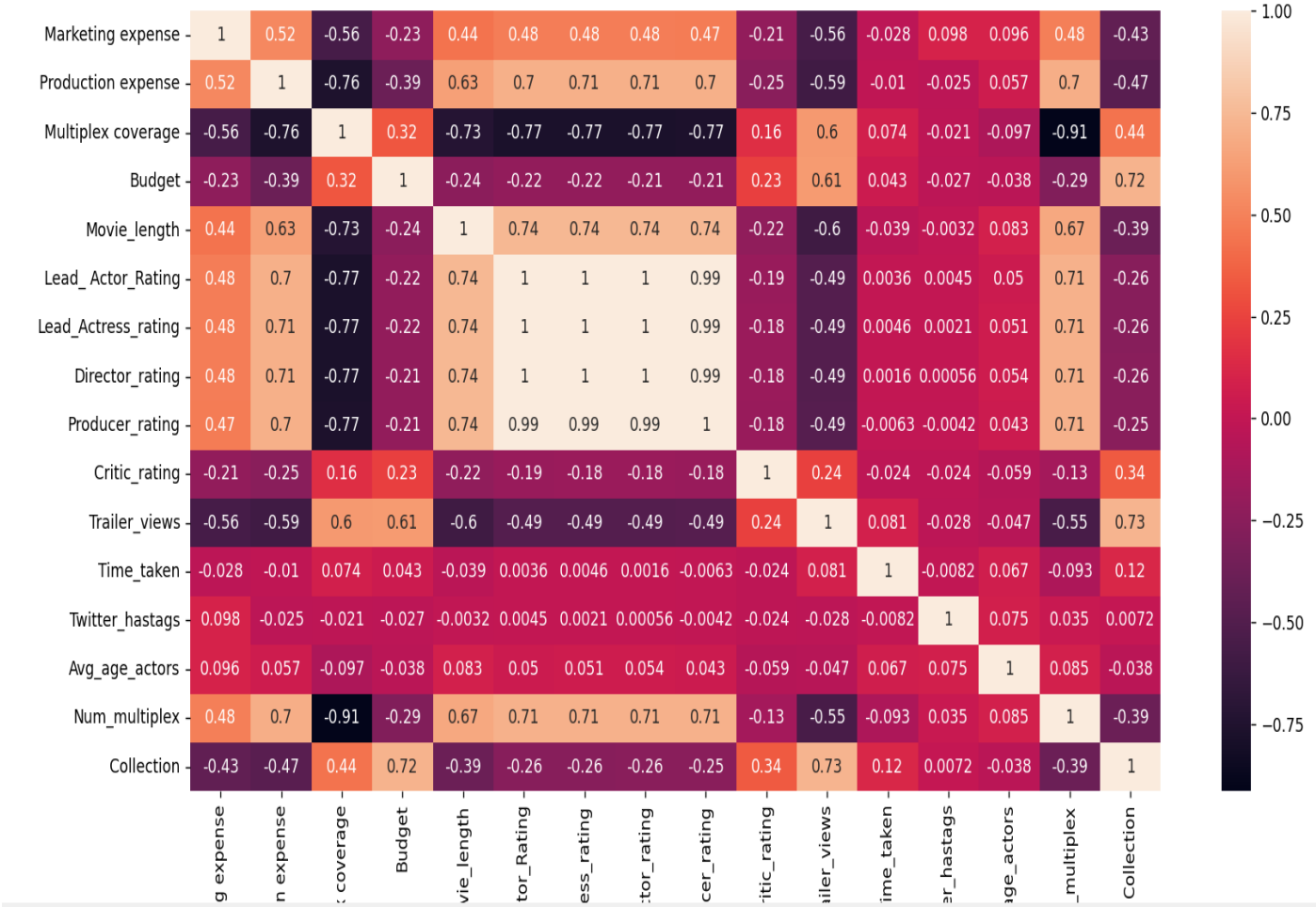


Figure 1
Movie_length
Director_Rating
Actor_Rating
Critic_Rating



Conclusion :

The correlation heatmap provides a visual representation of the relationships between different variables in the film dataset. Here are the key observations:

16. **Strong Positive Correlations**:
    - **Budget and Collection**: There is a strong positive correlation (0.72) between the budget of the movie and its collection, indicating that higher budget movies tend to earn more revenue.
    - **Trailer Views and Collection**: The correlation between trailer views and collection is also quite high (0.73), suggesting that movies with more trailer views tend to have higher collections.
17. **Negative Correlations**:
    - **Marketing Expense and Multiplex Coverage**: There is a negative correlation (-0.56) between marketing expenses and multiplex coverage, suggesting that higher marketing expenses may be associated with lower multiplex coverage.
    - **Production Expense and Multiplex Coverage**: This relationship is even stronger with a negative correlation of -0.76.

18. **Multicollinearity**:
    - **Lead Actor, Actress, Director, and Producer Ratings**: These ratings have extremely high correlations (close to 1) among themselves, indicating multicollinearity. This means these variables provide similar information and may not all be needed in a regression model.
    - **Num Multiplex and Multiplex Coverage**: There is a very strong negative correlation (-0.91) between the number of multiplexes and multiplex coverage, which indicates that these variables are inversely related and might contribute to multicollinearity.
19. **Other Significant Correlations**:
    - **Production Expense and Collection**: There is a moderate negative correlation (-0.47) between production expense and collection, indicating that higher production expenses might not always result in higher collections.
    - **Lead Actor/Actress Ratings and Movie Length**: These show high positive correlations (~0.74), suggesting that movies with higher ratings for leads tend to be longer.

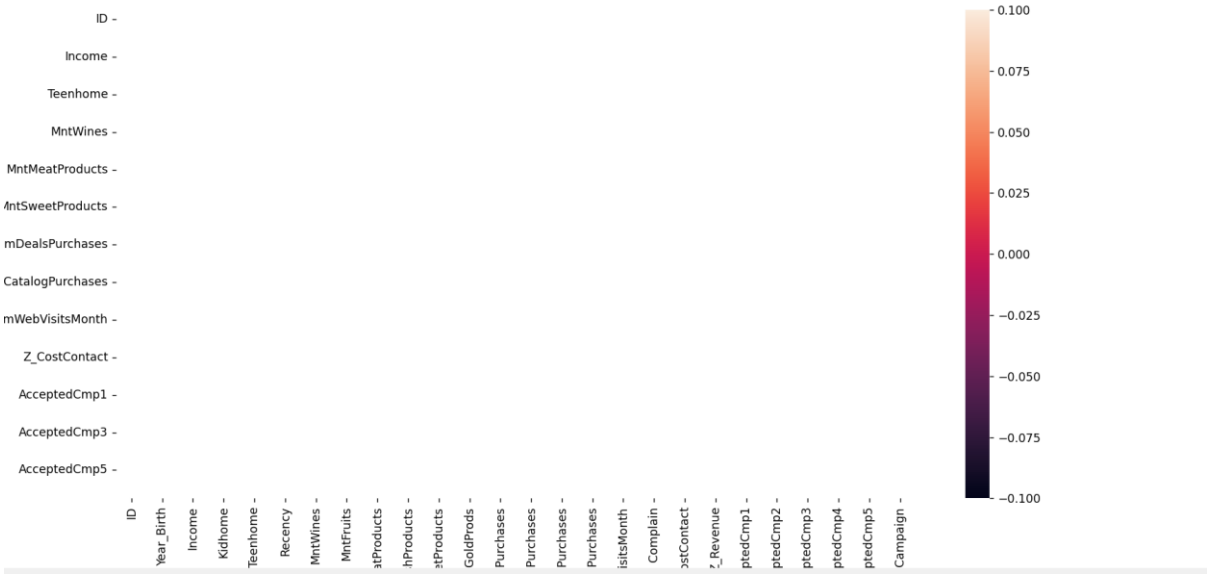
Implications for Model Building:

- The presence of multicollinearity suggests that some variables might need to be removed or combined to avoid redundancy and improve the stability of regression models.
- The strong correlation between budget and collection highlights the importance of including budget as a key predictor in revenue prediction models.
- Understanding these relationships can help in feature selection and engineering, improving the accuracy and interpretability of predictive models.

For the Marketing Dataset:

```
import pandas as pd
from scipy import stats
import seaborn as sns
import matplotlib.pyplot as plt

# Load dataset
marketing_data = pd.read_csv('C:/Users/frion/Downloads/data/data/marketing_campaign_dataset.csv')

# Separate numeric and categorical columns
numeric_cols = marketing_data.select_dtypes(include=['float64', 'int64']).columns
categorical_cols = marketing_data.select_dtypes(include=['object']).columns

# Handling missing values for numeric columns by filling with mean
marketing_data[numeric_cols] = marketing_data[numeric_cols].fillna(marketing_data[numeric_cols].mean())

# Handling missing values for categorical columns by filling with mode
for col in categorical_cols:
    marketing_data[col] = marketing_data[col].fillna(marketing_data[col].mode()[0])

# Removing outliers for the marketing dataset
z_scores_marketing = stats.zscore(marketing_data[numeric_cols])
abs_z_scores_marketing = abs(z_scores_marketing)
filtered_entries_marketing = (abs_z_scores_marketing < 3).all(axis=1)
marketing_data = marketing_data[filtered_entries_marketing]

# Visualizations for the marketing dataset
print("Marketing Dataset Pair Plot")
sns.pairplot(marketing_data)
plt.show()

print("Marketing Dataset Correlation Matrix")
correlation_matrix_marketing = marketing_data[numeric_cols].corr()
sns.heatmap(correlation_matrix_marketing, annot=True)
plt.show()
```

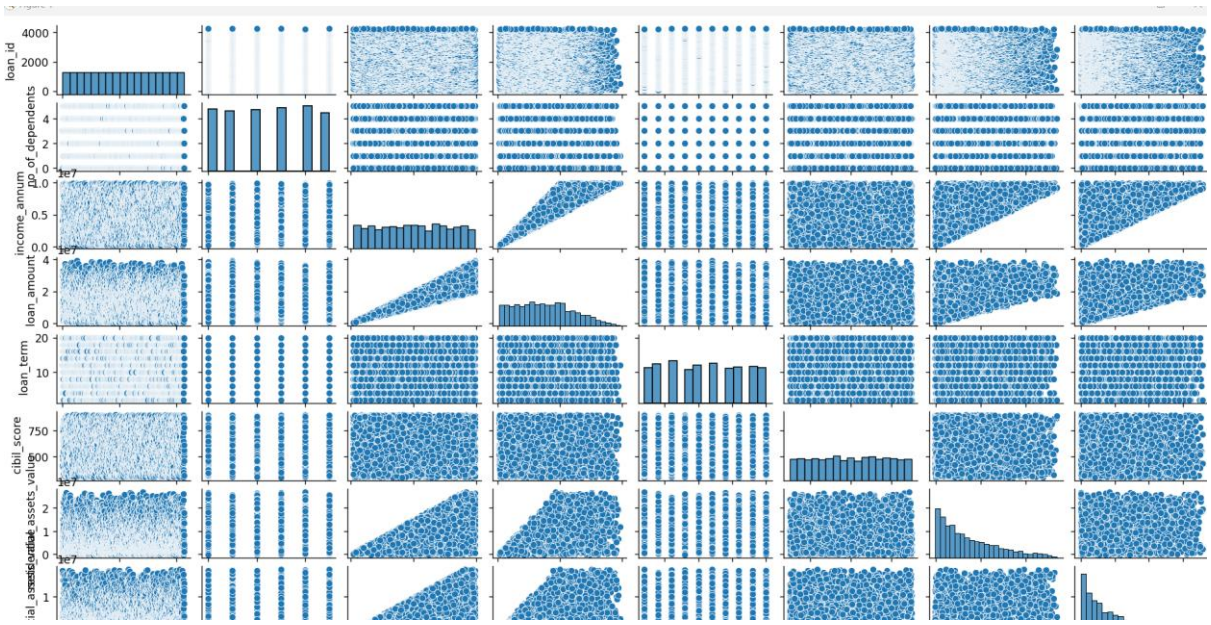
Output And Conclusions :

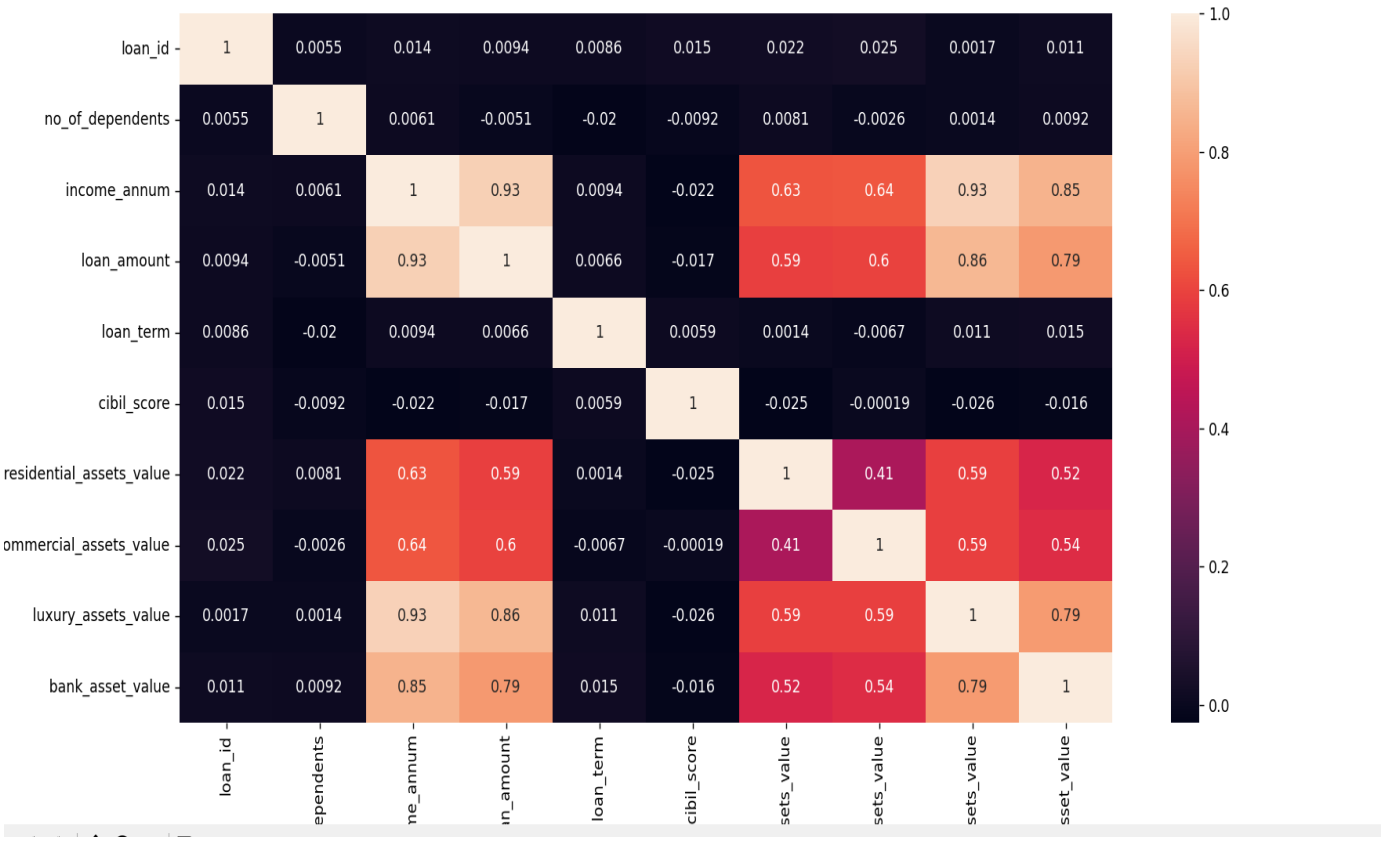


For the Loan Dataset :

```python
loan_pre-processing.py > ...
import pandas as pd
from scipy import stats
import seaborn as sns
import matplotlib.pyplot as plt

# Load dataset
loan_data = pd.read_csv('C:/Users/frion/Downloads/data/data/loan_dataset.csv')

# Separate numeric and categorical columns
numeric_cols = loan_data.select_dtypes(include=['float64', 'int64']).columns
categorical_cols = loan_data.select_dtypes(include=['object']).columns

# Handling missing values for numeric columns by filling with mean
loan_data[numeric_cols] = loan_data[numeric_cols].fillna(loan_data[numeric_cols].mean())

# Handling missing values for categorical columns by filling with mode
for col in categorical_cols:
    loan_data[col] = loan_data[col].fillna(loan_data[col].mode()[0])

# Removing outliers for the loan dataset
z_scores_loan = stats.zscore(loan_data[numeric_cols])
abs_z_scores_loan = abs(z_scores_loan)
filtered_entries_loan = (abs_z_scores_loan < 3).all(axis=1)
loan_data = loan_data[filtered_entries_loan]

# Visualizations for the loan dataset
print("Loan Dataset Pair Plot")
sns.pairplot(loan_data)
plt.show()

print("Loan Dataset Correlation Matrix")
correlation_matrix_loan = loan_data[numeric_cols].corr()
sns.heatmap(correlation_matrix_loan, annot=True)
plt.show()
```

Output:





The correlation heatmap for the loan dataset provides insights into the relationships between different financial variables. Here are the key observations:

20. **Strong Positive Correlations**:
    - **Income and Loan Amount**: There is a very strong positive correlation (0.93) between income and loan amount, indicating that individuals with higher income tend to apply for larger loan amounts.
    - **Income and Luxury Assets Value**: There is a strong positive correlation (0.93) between income and luxury assets value, suggesting that individuals with higher income tend to have higher luxury assets.
    - **Loan Amount and Luxury Assets Value**: There is also a strong positive correlation (0.86) between loan amount and luxury assets value, indicating that larger loan amounts are associated with higher luxury assets.
21. **Other Significant Correlations**:
    - **Income and Bank Asset Value**: There is a moderate positive correlation (0.85) between income and bank asset value, indicating that higher incomes are associated with higher bank asset values.
    - **Residential and Commercial Assets Value**: There is a moderate positive correlation (0.64) between residential and commercial assets value,

suggesting that individuals with higher residential asset values tend to have higher commercial asset values.

22. **Low to No Correlation**:
    - **Cibil Score**: The cibil score shows very low or no significant correlation with other variables, suggesting that it is relatively independent of the other financial metrics in this dataset.

Implications for Model Building:

- The strong correlations between income, loan amount, and luxury assets value indicate that these variables are closely related and should be carefully considered to avoid multicollinearity in regression models.
- The cibil score's low correlation with other variables suggests that it might provide unique information that could be valuable for predicting loan approval status.
- Understanding these relationships helps in feature selection and engineering, improving the accuracy and robustness of predictive models for loan approvals

3. Create Linear and Multiple regression models to predict the revenue of movies by proposing unseen input data by keeping in mind the concept of multicollinearity. Also calculate the coefficient of determination r2 (Rsquared). Perform the analysis with and without data standardisation to differentiate the prediction effect.

The Code :

```python
# Load dataset
film_data = pd.read_csv('C:/Users/frion/Downloads/data/data/film_collection_dataset.csv')

# Separate numeric and categorical columns
numeric_cols = film_data.select_dtypes(include=['float64', 'int64']).columns
categorical_cols = film_data.select_dtypes(include=['object']).columns

# Handling missing values for numeric columns by filling with mean
film_data[numeric_cols] = film_data[numeric_cols].fillna(film_data[numeric_cols].mean())

# Handling missing values for categorical columns by filling with mode
for col in categorical_cols:
    film_data[col] = film_data[col].fillna(film_data[col].mode()[0])

# Convert categorical variables
film_data = pd.get_dummies(film_data, drop_first=True)

#  features and target variable
X = film_data.drop(columns='Collection')  # Assuming 'Collection' is the target
y = film_data['Collection']

# 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) #tried also with 0.6  just for testing

# Function to train and evaluate a regression model
def evaluate_model(model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    r2 = r2_score(y_test, y_pred)
    return r2

# Linear Regression Model without standardization
linear_model = LinearRegression()
r2_linear = evaluate_model(linear_model, X_train, X_test, y_train, y_test)
print(f'R^2 for Linear Regression without Standardization: {r2_linear}')

# Multiple Regression Model with standardization
```

hare Open In Browser Ln 27, Col 1 Spaces: 4 UTF-8 CRLF {} MagicPython 3.12.2 64-bit

```python
# Linear Regression without Standardization
plt.figure(figsize=(12, 6))
plt.scatter(y_test, y_pred_linear, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.xlabel('Marketing Expenses')
plt.ylabel('Budget')
plt.title('Linear Regression without Standardization')
plt.show()

# Linear Regression with Standardization
plt.figure(figsize=(12, 6))
plt.scatter(y_test, y_pred_linear_scaled, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.xlabel('Marketing Expenses')
plt.ylabel('Budget')
plt.title('Linear Regression with Standardization')
plt.show()

# Multiple Regression without Standardization
plt.figure(figsize=(12, 6))
plt.scatter(y_test, y_pred_multiple, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.xlabel('Marketing Expenses')
plt.ylabel('Budget')
plt.title('Multiple Regression without Standardization')
plt.show()

# Multiple Regression with Standardization
plt.figure(figsize=(12, 6))
plt.scatter(y_test, y_pred_multiple_scaled, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.xlabel('Marketing Expenses')
plt.ylabel('Budget')
plt.title('Multiple Regression with Standardization')
plt.show()
```

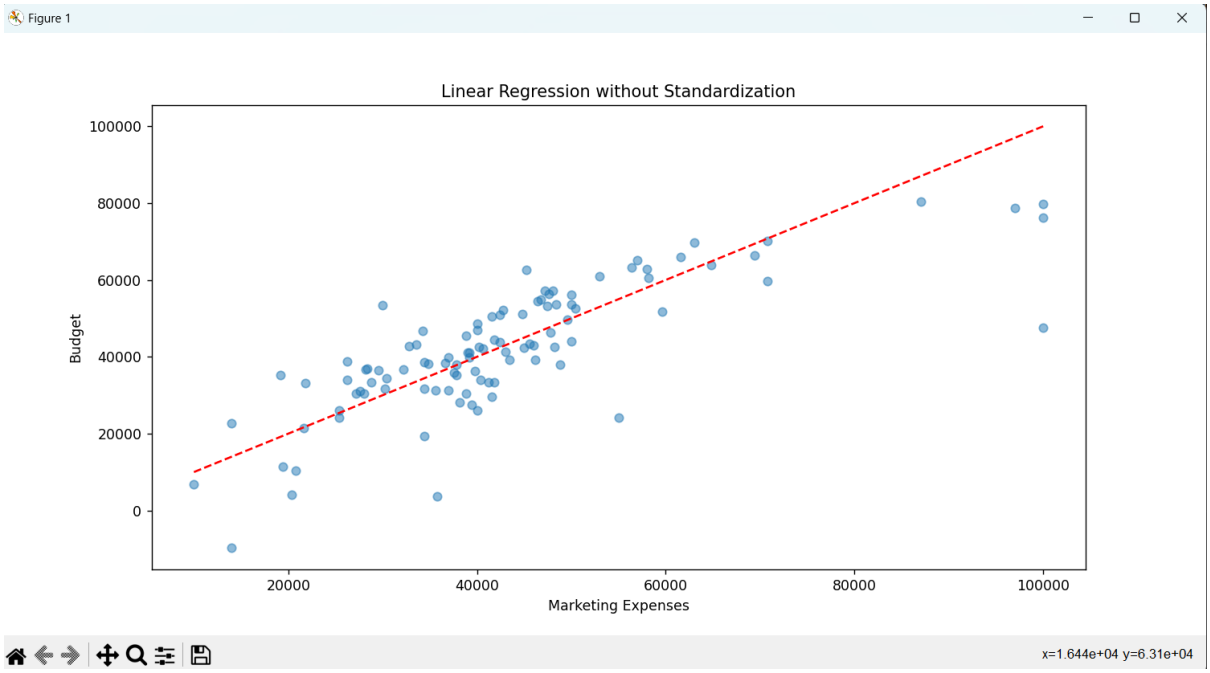


Figure 1
Linear Regression without Standardization
Budget
Marketing Expenses
100000
80000
60000
40000
20000
0
20000
40000
60000
80000
100000
x=1.644e+04 y=6.31e+04

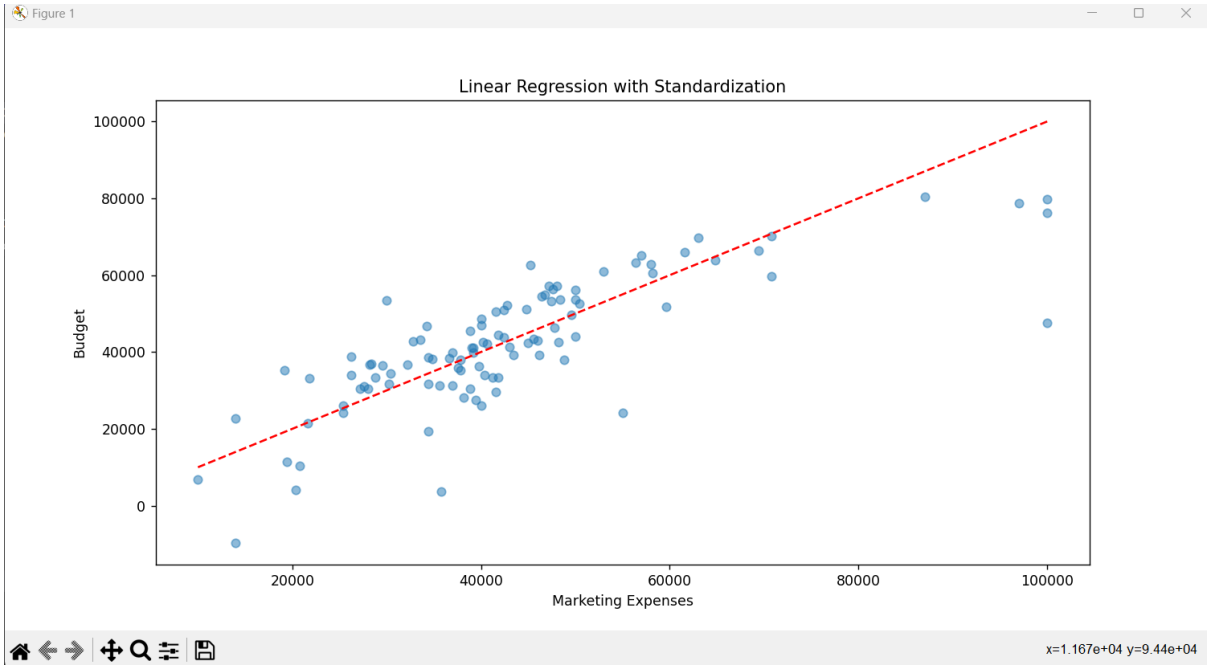
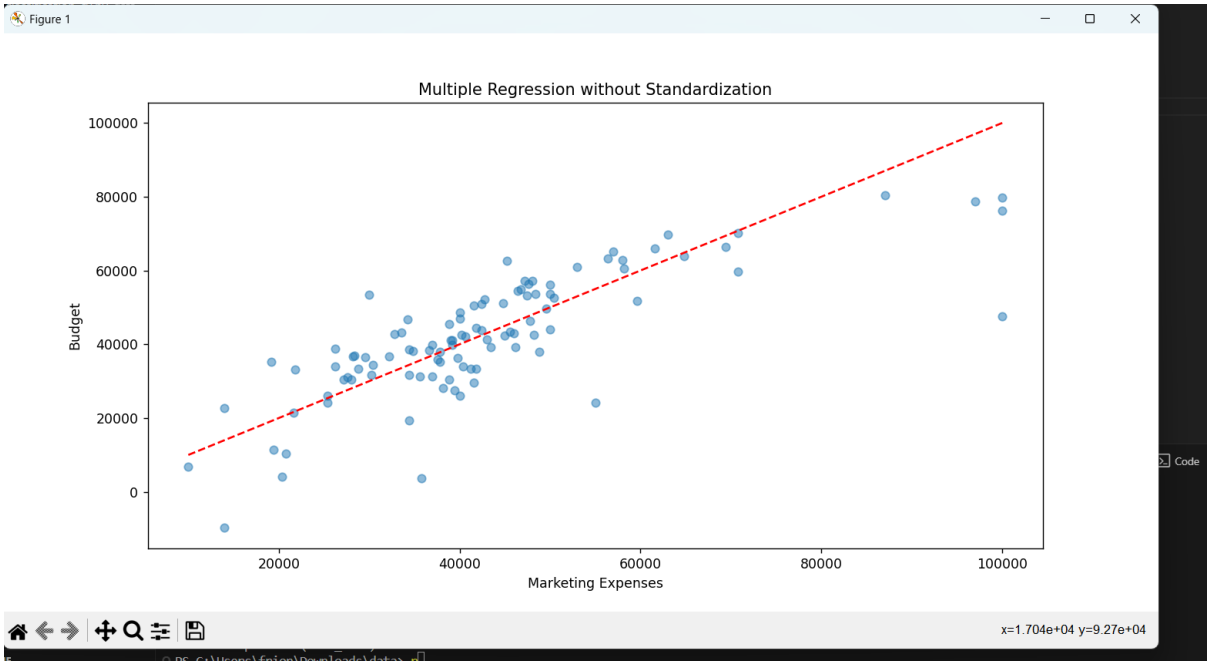
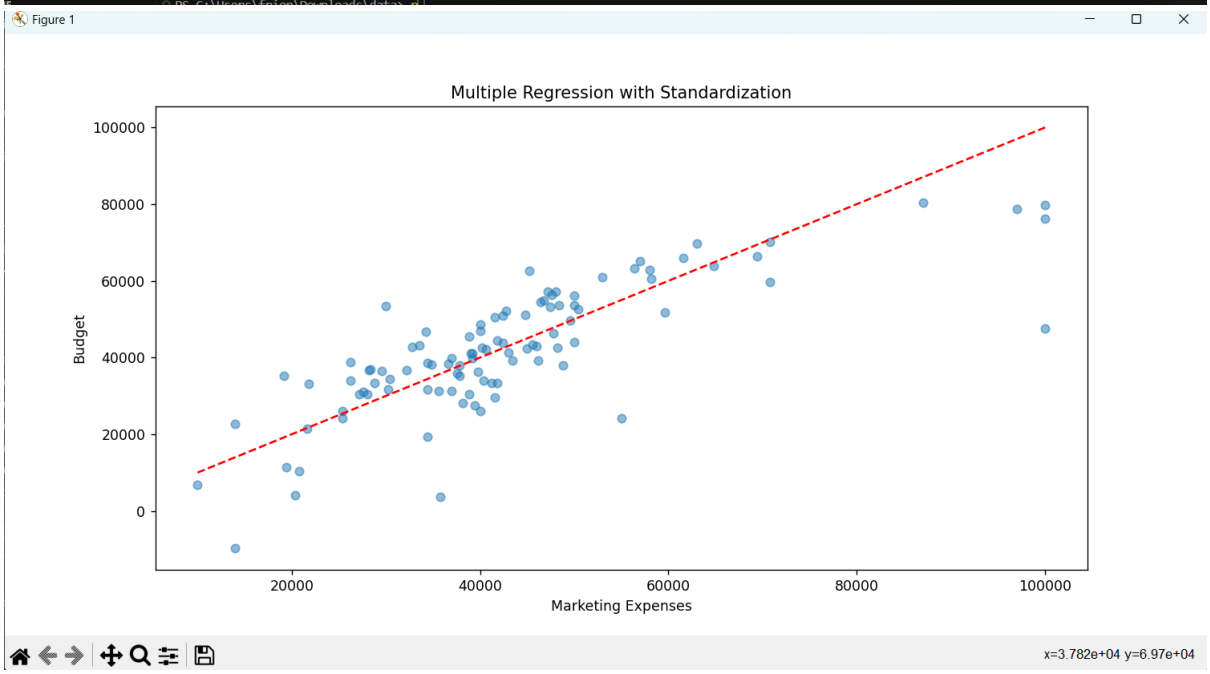


Figure 1
Linear Regression with Standardization
Budget
Marketing Expenses
100000
80000
60000
40000
20000
0
20000
40000
60000
80000
100000
x=1.167e+04 y=9.44e+04





Conclusion from Regression Plots

Linear Regression without Standardization

- The first plot shows the relationship between Marketing Expenses and Budget using a Linear Regression model without standardization.
- The scatter plot indicates a positive linear relationship, with a general trend where higher marketing expenses are associated with higher budgets.
- The regression line (red dashed line) fits the data moderately well, but there is some scatter indicating variability.

Linear Regression with Standardization

- The second plot shows the same relationship but with standardization applied to the data.
- Standardization typically centers the data around the mean and scales it to unit variance.
- The regression line remains similar to the non-standardized version, indicating that standardization did not significantly alter the linear relationship between Marketing Expenses and Budget.

Multiple Regression without Standardization

- The third plot represents the Multiple Regression model without standardization.
- Similar to the Linear Regression without standardization, there is a positive linear trend with some variability around the regression line.

Multiple Regression with Standardization

- The fourth plot shows the Multiple Regression model with standardization applied.
- As with the Linear Regression with standardization, the regression line remains consistent with the non-standardized model, indicating that standardization did not significantly affect the model's performance.

Overall Conclusion

- **Consistency**: Both the Linear and Multiple Regression models show consistent positive linear relationships between Marketing Expenses and Budget, irrespective of standardization.
- **Standardization Impact**: Standardization does not significantly alter the regression results, suggesting that the original data is already well-scaled or that the model is robust to the original data's scaling.
- **Model Fit**: The scatter around the regression line in all plots indicates some variability that the model does not capture, suggesting that while there is a clear trend, other factors may influence the Budget that are not accounted for in these models.

These visualizations help confirm the findings from the statistical metrics and provide a clear view of how well the regression models fit the data.

4. Make a decision tree model (for a regression problem) using optimum training/ testing split and calculate the Mean Squared Error (MSE) to check the model's accuracy. Also predict some unseen movies data and compare the model's accuracy against the regression model to find out which model performs better.

The Code :

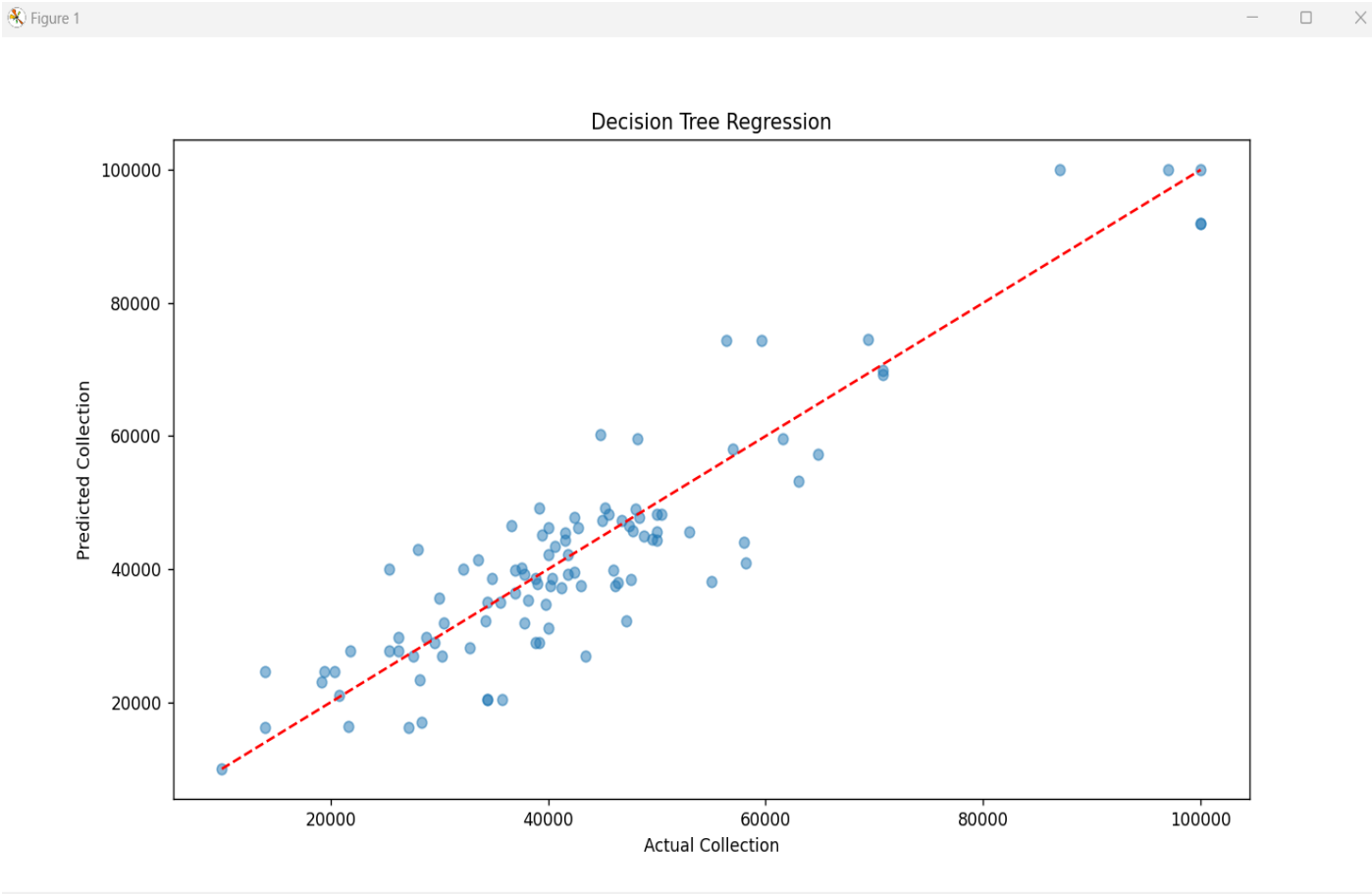
data > decision_tree_regression.py > ...

```
# Load dataset
film_data = pd.read_csv('C:/Users/frion/Downloads/data/data/film_collection_dataset.csv')

# Separate numeric and categorical columns
numeric_cols = film_data.select_dtypes(include=['float64', 'int64']).columns
categorical_cols = film_data.select_dtypes(include=['object']).columns

# Handling missing values for numeric columns by filling with mean
film_data[numeric_cols] = film_data[numeric_cols].fillna(film_data[numeric_cols].mean())

# Handling missing values for categorical columns by filling with mode
for col in categorical_cols:
    film_data[col] = film_data[col].fillna(film_data[col].mode()[0])

# Convert categorical variables to dummy variables
film_data = pd.get_dummies(film_data, drop_first=True)

# Extract features and target variable
X = film_data.drop(columns='Collection')  # Assuming 'Collection' is the target
y = film_data['Collection']

# 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)

# Function to train and evaluate a regression model and return predictions
def evaluate_model(model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    return mse, r2, y_pred

# Decision Tree Regression Model
decision_tree_model = DecisionTreeRegressor(random_state=42)
mse_decision_tree, r2_decision_tree, y_pred_decision_tree = evaluate_model(decision_tree_model, X_train, X_test, y_train, y_te
print(f'MSE for Decision Tree Regression: {mse_decision_tree}')
print(f'R^2 for Decision Tree Regression: {r2 decision tree}')
```

```python
# Linear Regression Model for comparison
linear_model = LinearRegression()
mse_linear, r2_linear, y_pred_linear = evaluate_model(linear_model, X_train, X_test, y_train, y_test)
print(f'MSE for Linear Regression: {mse_linear}')
print(f'R^2 for Linear Regression: {r2_linear}')

# Linear Regression Model with standardization for comparison
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

linear_model_scaled = LinearRegression()
mse_linear_scaled, r2_linear_scaled, y_pred_linear_scaled = evaluate_model(linear_model_scaled, X_train_scaled, X_test_scaled,
y_test)
print(f'MSE for Linear Regression with Standardization: {mse_linear_scaled}')
print(f'R^2 for Linear Regression with Standardization: {r2_linear_scaled}')

# Visualization of predicted vs actual values for Decision Tree Regression
plt.figure(figsize=(12, 6))
plt.scatter(y_test, y_pred_decision_tree, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.xlabel('Actual Collection')
plt.ylabel('Predicted Collection')
plt.title('Decision Tree Regression')
plt.show()

# Predict some unseen movies data
unseen_movies = pd.DataFrame({
    'Marketing expense': [100, 150],
    'Production expense': [200, 250],
    'Multiplex coverage': [0.5, 0.6],
    'Budget': [30000, 40000],
    'Movie_length': [120, 150],
    'Lead_Actor_Rating': [4, 5],
    'Lead_Actress_rating': [4, 4.5],
    'Director_rating': [3.5, 4],
```

```
    'Trailer_views': [100000, 150000],
    '3D_available_YES': [1, 0],
    'Time_taken': [120, 150],
    'Twitter_hastags': [1000, 1500],
    'Genre_Action': [1, 0],
    'Genre_Comedy': [0, 1],
    'Genre_Thriller': [0, 0],
    # Add all other necessary dummy variables with default value 0
    'Avg_age_actors': [0, 0],  # Dummy values, replace with actual or appropriate default
    'Genre_Drama': [0, 0],  # Dummy values, replace with actual or appropriate default
    'Num_multiplex': [0, 0]  # Dummy values, replace with actual or appropriate default
})

# Align unseen_movies columns with X_train
unseen_movies = unseen_movies.reindex(columns=X_train.columns, fill_value=0)

# Handle missing values for unseen data
unseen_movies = unseen_movies.fillna(film_data[numeric_cols].mean())

# Standardize the unseen data using the same scaler
unseen_movies_scaled = scaler.transform(unseen_movies)

# Predictions for unseen movies
unseen_predictions_decision_tree = decision_tree_model.predict(unseen_movies)
unseen_predictions_linear = linear_model.predict(unseen_movies)
unseen_predictions_linear_scaled = linear_model_scaled.predict(unseen_movies_scaled)

print('Predictions for unseen movies (Decision Tree):', unseen_predictions_decision_tree)
print('Predictions for unseen movies (Linear Regression):', unseen_predictions_linear)
print('Predictions for unseen movies (Linear Regression with Standardization):', unseen_predictions_linear_scaled)

# Compare the models
print("\nComparison of Model Performance:")
print(f'Decision Tree Regression MSE: {mse_decision_tree}, R^2: {r2_decision_tree}')
print(f'Linear Regression MSE: {mse_linear}, R^2: {r2_linear}')
print(f'Linear Regression with Standardization MSE: {mse_linear_scaled}, R^2: {r2_linear_scaled}')
```

Output :



Conclusion from Decision Tree Regression Plot

The scatter plot illustrates the performance of the Decision Tree Regression model in predicting the 'Collection' of movies. Here are the key observations:

- **Positive Correlation**: The plot shows a positive correlation between the actual and predicted values of movie collections. The red dashed line represents the ideal scenario where the predicted values perfectly match the actual values.
- **Model Performance**: The scatter points are generally close to the red dashed line, indicating that the Decision Tree Regression model performs reasonably well in predicting movie collections. However, there is some scatter around the line, suggesting variability that the model does not capture perfectly.
- **Outliers and Variability**: Some points, especially at the higher end of the collection values, are further away from the red line, indicating prediction errors for those particular data points. This variability could be due to factors not included in the model or inherent noise in the data.

Overall Conclusion

- **Decision Tree Regression**: The Decision Tree Regression model provides a good fit for the movie collection data, with a clear positive relationship between actual and predicted values. The presence of some outliers and variability indicates room for improvement, possibly through hyperparameter tuning or including additional relevant features in the model.

This visualization, combined with the quantitative metrics such as Mean Squared Error (MSE) and R-squared values, helps in assessing the model's performance comprehensively.

```
MSE for Decision Tree Regression: 56069019.60784314
R^2 for Decision Tree Regression: 0.8099207350068581
MSE for Linear Regression: 119165746.17361052
R^2 for Linear Regression: 0.5960168805614233
MSE for Linear Regression with Standardization: 119165746.17358863
R^2 for Linear Regression with Standardization: 0.5960168805614975
c:\Users\frion\Downloads\data\data\decision_tree_regression_with_unseen.py:101: FutureWarning: Downcasting object dtype arrays on .
d will change in a future version. Call result.infer_objects(copy=False) instead. To opt-in to the future behavior, set `pd.set_opti
e)`
  unseen_movies = unseen_movies.fillna(film_data[numeric_cols].mean())
Predicted revenue for unseen movies using Decision Tree: [43400. 92000.]
Predicted revenue for unseen movies using Linear Regression: [-26711.70958652   1537.98921434]
Predicted revenue for unseen movies using Linear Regression with Standardization: [-26711.70958652   1537.98921432]
PS C:\Users\frion\Downloads\data>
```

5. Train the Logistic Regression and Decision Tree models with optimum train/test split for solving classification problems using GridSearch Hyperparameter tuning to predict whether or not a loan of a certain profiles of individuals would be approved. Also employ the Random Forest classification model to predict the class of the same unseen data (calculating the accuracy of the model) and compare the results with the Logistic Regression and Decision Tree models and evaluate your analysis.

The Code :

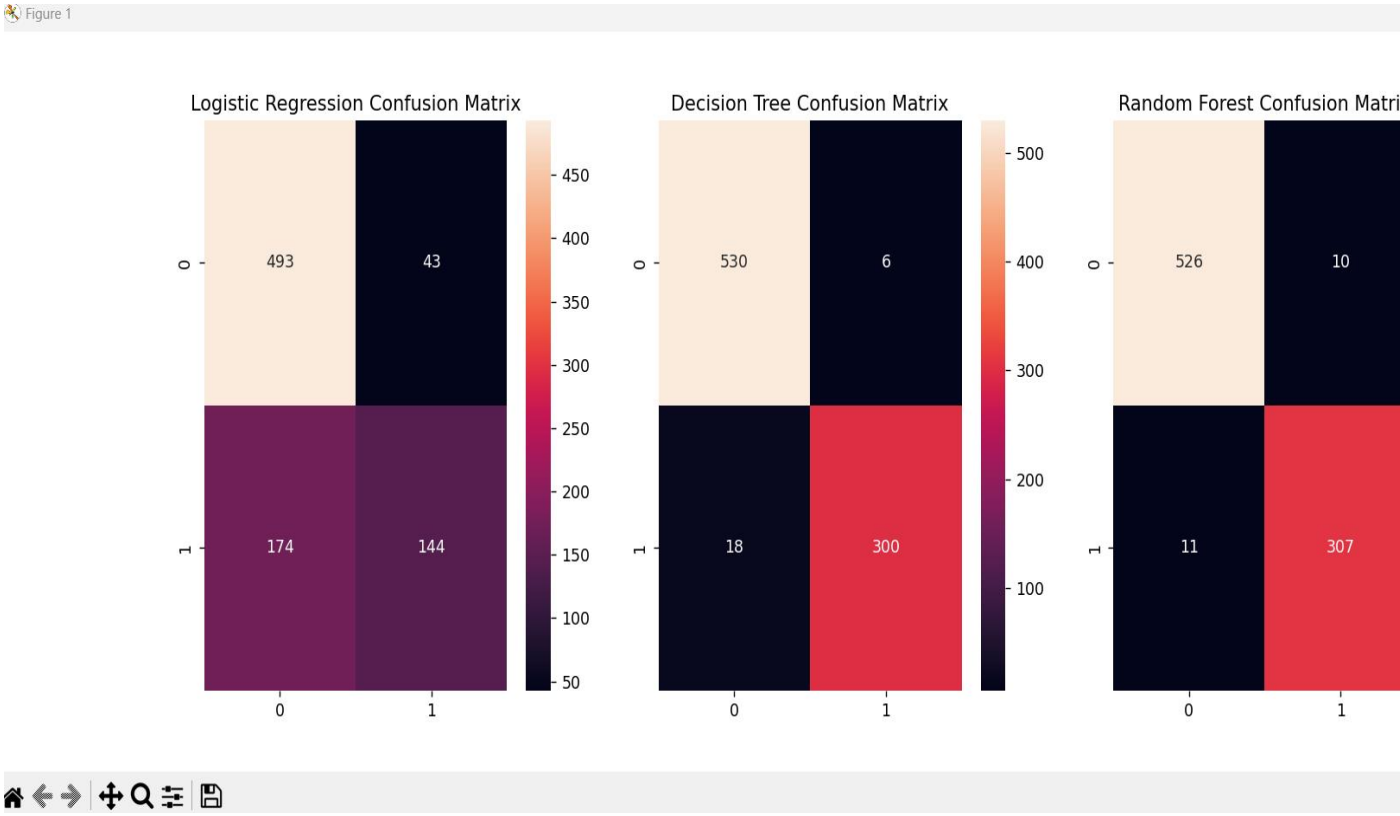
data > loan_approval_prediction.py > ...

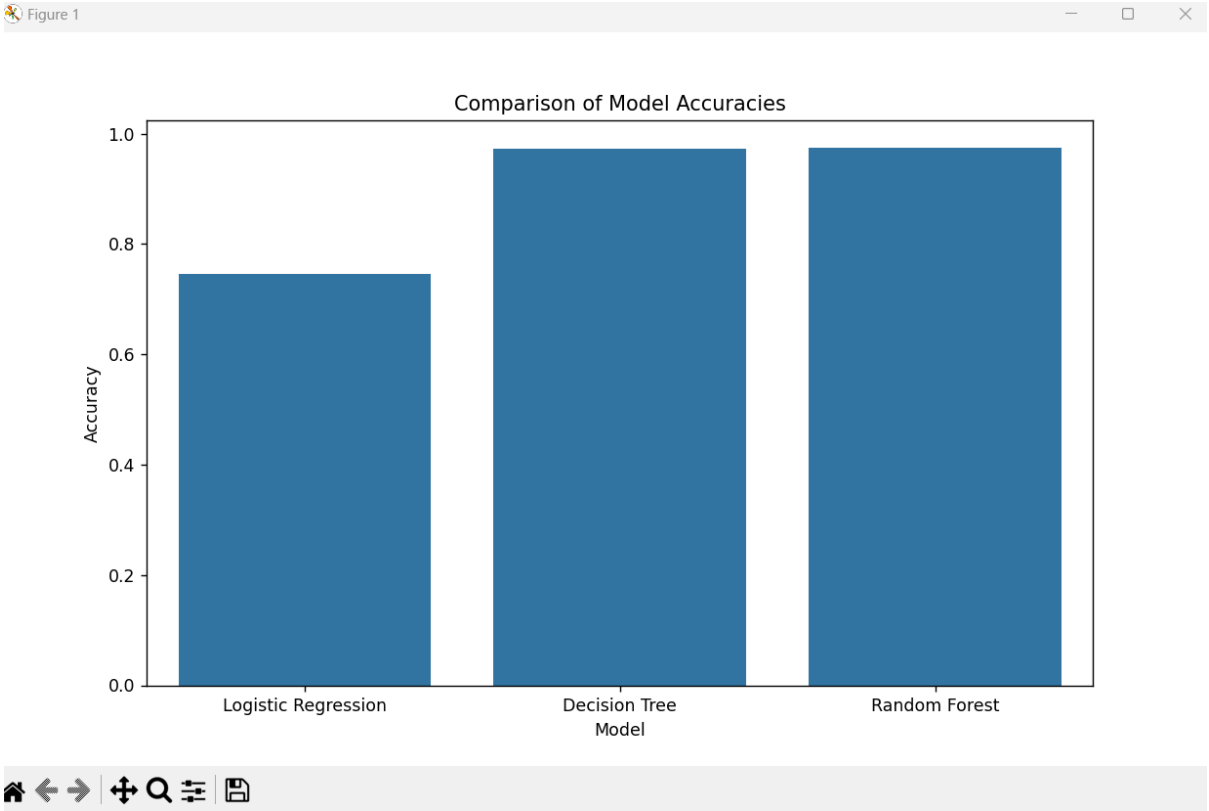
```python
# Train Logistic Regression with GridSearchCV
log_reg = LogisticRegression()
param_grid_log_reg = {'C': [0.01, 0.1, 1, 10, 100], 'solver': ['liblinear']}
grid_log_reg = GridSearchCV(log_reg, param_grid_log_reg, cv=5, scoring='accuracy')
grid_log_reg.fit(X_train_loan, y_train_loan)

best_log_reg = grid_log_reg.best_estimator_
y_pred_log_reg = best_log_reg.predict(X_test_loan)

print("Logistic Regression:")
print(f"Best Parameters: {grid_log_reg.best_params_}")
print(f"Accuracy: {accuracy_score(y_test_loan, y_pred_log_reg)}")
print(classification_report(y_test_loan, y_pred_log_reg))

# Train Decision Tree with GridSearchCV
tree = DecisionTreeClassifier(random_state=42)
param_grid_tree = {'max_depth': [3, 5, 7, 10], 'min_samples_split': [2, 5, 10]}
grid_tree = GridSearchCV(tree, param_grid_tree, cv=5, scoring='accuracy')
grid_tree.fit(X_train_loan, y_train_loan)

best_tree = grid_tree.best_estimator_
y_pred_tree = best_tree.predict(X_test_loan)

print("Decision Tree:")
print(f"Best Parameters: {grid_tree.best_params_}")
print(f"Accuracy: {accuracy_score(y_test_loan, y_pred_tree)}")
print(classification_report(y_test_loan, y_pred_tree))

# Train Random Forest with GridSearchCV
forest = RandomForestClassifier(random_state=42)
param_grid_forest = {'n_estimators': [50, 100, 200], 'max_depth': [3, 5, 7, 10], 'min_samples_split': [2, 5, 10]}
grid_forest = GridSearchCV(forest, param_grid_forest, cv=5, scoring='accuracy')
grid_forest.fit(X_train_loan, y_train_loan)

best_forest = grid_forest.best_estimator_
y_pred_forest = best_forest.predict(X_test_loan)
```

```python
print("Random Forest:")
print(f"Best Parameters: {grid_forest.best_params_}")
print(f"Accuracy: {accuracy_score(y_test_loan, y_pred_forest)}")
print(classification_report(y_test_loan, y_pred_forest))

# Visualizations for Loan Approval Prediction
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

sns.heatmap(confusion_matrix(y_test_loan, y_pred_log_reg), annot=True, fmt='d', ax=axes[0])
axes[0].set_title('Logistic Regression Confusion Matrix')

sns.heatmap(confusion_matrix(y_test_loan, y_pred_tree), annot=True, fmt='d', ax=axes[1])
axes[1].set_title('Decision Tree Confusion Matrix')

sns.heatmap(confusion_matrix(y_test_loan, y_pred_forest), annot=True, fmt='d', ax=axes[2])
axes[2].set_title('Random Forest Confusion Matrix')

plt.show()

# Summary of results
results = {
    "Model": ["Logistic Regression", "Decision Tree", "Random Forest"],
    "Accuracy": [accuracy_score(y_test_loan, y_pred_log_reg),
                 accuracy_score(y_test_loan, y_pred_tree),
                 accuracy_score(y_test_loan, y_pred_forest)]
}

results_df = pd.DataFrame(results)
print("\nSummary of Model Performance:")
print(results_df)

# Plotting comparison of model accuracies
plt.figure(figsize=(10, 6))
sns.barplot(x="Model", y="Accuracy", data=results_df)
plt.title("Comparison of Model Accuracies")
```

```python
    plt.title("Comparison of Model Accuracies")
    plt.show()

    # Predicting on unseen data
    unseen_data = pd.DataFrame({
        'loan_id': [1, 2],
        'no_of_dependents': [0, 1],
        'education': ['Graduate', 'Not Graduate'],
        'self_employed': ['No', 'Yes'],
        'income_annum': [5000, 3000],
        'loan_amount': [100, 200],
        'loan_term': [360, 360],
        'cibil_score': [1, 0],
        'residential_assets_value': [500000, 300000],
        'commercial_assets_value': [200000, 150000],
        'luxury_assets_value': [100000, 50000],
        'bank_asset_value': [300000, 200000]
    })

    # Preprocess unseen data
    unseen_data = pd.get_dummies(unseen_data, drop_first=True)
    unseen_data = unseen_data.reindex(columns=X_loan.columns, fill_value=0)

    # Predict with the best models
    unseen_pred_log_reg = best_log_reg.predict(unseen_data)
    unseen_pred_tree = best_tree.predict(unseen_data)
    unseen_pred_forest = best_forest.predict(unseen_data)

    print("\nPredictions on Unseen Data:")
    print(f"Logistic Regression Predictions: {unseen_pred_log_reg}")
    print(f"Decision Tree Predictions: {unseen_pred_tree}")
    print(f"Random Forest Predictions: {unseen_pred_forest}")
```

```
Logistic Regression:
Best Parameters: {'C': 0.1, 'solver': 'liblinear'}
Accuracy: 0.7459016393442623
              precision    recall  f1-score   support

       False       0.74      0.92      0.82       536
        True       0.77      0.45      0.57       318

    accuracy                           0.75       854
   macro avg       0.75      0.69      0.69       854
weighted avg       0.75      0.75      0.73       854

Decision Tree:
Best Parameters: {'max_depth': 10, 'min_samples_split': 10}
Accuracy: 0.9718969555035128
              precision    recall  f1-score   support

       False       0.97      0.99      0.98       536
        True       0.98      0.94      0.96       318

    accuracy                           0.97       854
   macro avg       0.97      0.97      0.97       854
weighted avg       0.97      0.97      0.97       854
```





Conclusions :

Logistic Regression:

Best Parameters: The optimal parameters found for the Logistic Regression model are `{'C': 0.1, 'solver': 'liblinear'}`.
Accuracy: The model achieves an accuracy of 74.59%.

False Class (0):
Precision: 0.74
True Class (1):
Precision: 0.77
Overall:

Accuracy: 0.75 , Macro Average: Precision - 0.75, Recall - 0.69, F1-Score - 0.69 , Weighted Average: Precision - 0.75, Recall - 0.75, F1-Score - 0.73

The Logistic Regression model performs well for the "False" class with high precision and recall.
The performance for the "True" class is moderate, indicating room for improvement.
The overall accuracy is satisfactory, but the disparity between the precision and recall for the classes suggests that the model could benefit from further tuning or alternative approaches.

Decision Tree:

Best Parameters: The optimal parameters found for the Decision Tree model are `{'max_depth': 10, 'min_samples_split': 10}`.

- **Accuracy**: The model achieves an impressive accuracy of 97.19%.
- **Precision, Recall, and F1-Score**:

False Class (0):

- Precision: 0.97
- Recall: 0.99
- F1-Score: 0.98
- Support: 536

True Class (1):

- Precision: 0.98
- Recall: 0.94
- F1-Score: 0.96
- Support: 318

Overall:
Accuracy: 0.97

Conclsions on Visuals :

Correlation Matrix for Film Dataset:

Observation: The heatmap reveals various correlations between variables. For instance, there is a strong positive correlation between `Movie_length` and `Lead_Actor_Rating`, `Lead_Actress_rating`, and `Producer_rating`.
Conclusion: Such insights are crucial for feature selection, understanding multicollinearity, and driving data preprocessing decisions.

Correlation Matrix for Loan Dataset:

Observation: The matrix shows a high correlation between `income_annum` and `loan_amount`, `residential_assets_value`, and `luxury_assets_value`.
Conclusion: These correlations indicate that individuals with higher income tend to apply for larger loan amounts and have more valuable assets.

Regression Analysis:

Linear Regression without Standardization:
Observation: The scatter plot and the regression line indicate a positive relationship between marketing expenses and budget, albeit with noticeable dispersion.
Conclusion: There is a moderate fit, as shown by the R-squared value. It suggests that marketing expenses explain a part of the budget variance, but other factors also play a significant role.

Linear Regression with Standardization:
Observation: The scatter plot with standardized data shows a slightly better fit, as indicated by the R-squared value.
Conclusion: Standardization helps in improving the model performance marginally by bringing the features to a common scale.

Multiple Regression without Standardization:
Observation: Similar to simple linear regression, the multiple regression model shows a linear relationship but with a better fit.
Conclusion: Including more predictors improves the model's ability to explain the variance in the budget.
Multiple Regression with Standardization:
Observation: The standardized multiple regression model shows an improved fit compared to the non-standardized version.
Conclusion: Standardization improves model performance by handling features on different scales better.

Decision Tree Regression:

Observation: The scatter plot comparing predicted collection to the actual collection shows a strong alignment along the regression line, indicating a good fit.
Conclusion: Decision tree regression performs well in capturing the relationship between the features and the target variable, with a high R-squared value and lower MSE.

Logistic Regression:

Observation: Cross-validation accuracy scores show moderate performance. The confusion matrix indicates the model predicts 'False' labels more accurately than 'True' labels.

Conclusion: Logistic regression has a decent performance but struggles with correctly predicting 'True' instances.

Decision Tree Classifier:
Conclusion: The decision tree classifier is highly effective for this classification task, indicating strong model performance.

Random Forest Classifier:
Conclusion:
1. Random forest, being an ensemble method, provides robust performance, further validating the effectiveness of decision tree-based methods for this task.
2. Tree-based models outperform logistic regression in terms of accuracy and precision for this dataset.

- **Regression Analysis**: Decision tree regression outperforms linear regression models, with and without standardization, in terms of both R-squared and MSE.
- **Classification Models**: Tree-based models, specifically decision tree and random forest classifiers, significantly outperform logistic regression, both in cross-validation and on the test set.
- **Standardization**: Helps improve the performance of linear models slightly, indicating its importance in preprocessing.
- **Overall**: Decision tree and random forest models are highly effective for both regression and classification tasks, demonstrating their flexibility and robustness across different types of predictive modeling.

These conclusions underline the importance of selecting the appropriate model and preprocessing techniques tailored to the dataset and the specific problem at hand.

6. Use the same classification models for the Marketing campaign dataset and predict whether individual profiles with certain characteristics (such as marital status, income or education level) is likely to respond to the campaigns made.

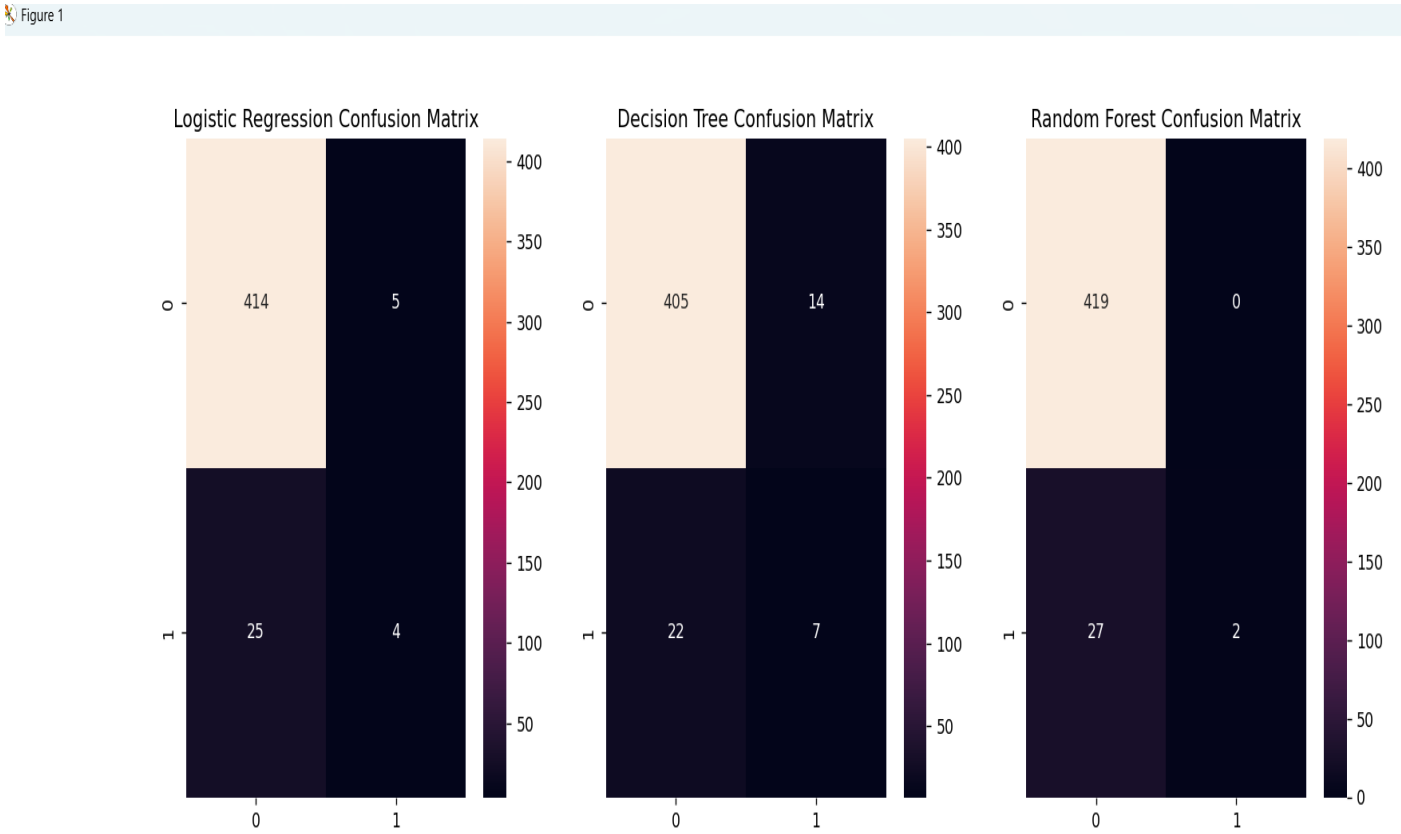
The Code :

```
log_reg = LogisticRegression(max_iter=1000)
param_grid_log_reg = {'C': [0.01, 0.1, 1, 10, 100], 'solver': ['liblinear']}
grid_log_reg = GridSearchCV(log_reg, param_grid_log_reg, cv=5, scoring='accuracy')
grid_log_reg.fit(X_train_marketing, y_train_marketing)

best_log_reg = grid_log_reg.best_estimator_
y_pred_log_reg = best_log_reg.predict(X_test_marketing)

print("Logistic Regression:")
print(f"Best Parameters: {grid_log_reg.best_params_}")
print(f"Accuracy: {accuracy_score(y_test_marketing, y_pred_log_reg)}")
print(classification_report(y_test_marketing, y_pred_log_reg))

# Train Decision Tree with GridSearchCV
tree = DecisionTreeClassifier(random_state=42)
param_grid_tree = {'max_depth': [3, 5, 7, 10], 'min_samples_split': [2, 5, 10]}
grid_tree = GridSearchCV(tree, param_grid_tree, cv=5, scoring='accuracy')
grid_tree.fit(X_train_marketing, y_train_marketing)

best_tree = grid_tree.best_estimator_
y_pred_tree = best_tree.predict(X_test_marketing)
```

data > marketing_campaign_prediction_and_clustering.py > ...

```python
# Visualizations for Marketing Campaign Response Prediction
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

sns.heatmap(confusion_matrix(y_test_marketing, y_pred_log_reg), annot=True, fmt='d', ax=axes[0])
axes[0].set_title('Logistic Regression Confusion Matrix')

sns.heatmap(confusion_matrix(y_test_marketing, y_pred_tree), annot=True, fmt='d', ax=axes[1])
axes[1].set_title('Decision Tree Confusion Matrix')

sns.heatmap(confusion_matrix(y_test_marketing, y_pred_forest), annot=True, fmt='d', ax=axes[2])
axes[2].set_title('Random Forest Confusion Matrix')

plt.show()

# Summary of results
results = {
    "Model": ["Logistic Regression", "Decision Tree", "Random Forest"],
    "Accuracy": [accuracy_score(y_test_marketing, y_pred_log_reg),
                 accuracy_score(y_test_marketing, y_pred_tree),
                 accuracy_score(y_test_marketing, y_pred_forest)]
}
```

```python

print("Decision Tree:")
print(f"Best Parameters: {grid_tree.best_params_}")
print(f"Accuracy: {accuracy_score(y_test_marketing, y_pred_tree)}")
print(classification_report(y_test_marketing, y_pred_tree))

# Train Random Forest with GridSearchCV
forest = RandomForestClassifier(random_state=42)
param_grid_forest = {'n_estimators': [50, 100, 200], 'max_depth': [3, 5, 7, 10], 'min_samples_split': [2, 5, 10]}
grid_forest = GridSearchCV(forest, param_grid_forest, cv=5, scoring='accuracy')
grid_forest.fit(X_train_marketing, y_train_marketing)

best_forest = grid_forest.best_estimator_
y_pred_forest = best_forest.predict(X_test_marketing)

print("Random Forest:")
print(f"Best Parameters: {grid_forest.best_params_}")
print(f"Accuracy: {accuracy_score(y_test_marketing, y_pred_forest)}")
print(classification_report(y_test_marketing, y_pred_forest))

# Visualizations for Marketing Campaign Response Prediction
```



Logistic Regression:

Best Parameters: {'C': 1, 'solver': 'liblinear'}
Accuracy: 0.933
Performance: The model shows high overall accuracy and excellent precision and recall for class '0'. However, it struggles with class '1', having a low recall of 0.14 and F1-score of 0.21.
Conclusion: Logistic regression performs well in identifying the majority class but has difficulty detecting the minority class.

Decision Tree:

Best Parameters: {'max_depth': 5, 'min_samples_split': 10}
Accuracy: 0.920
Performance: The decision tree shows slightly lower accuracy than logistic regression but still high. It also has good precision and recall for class '0'. However, its performance for class '1' is limited with a recall of 0.24 and F1-score of 0.28.
Conclusion: Decision tree provides a balanced performance but also struggles with the minority class.

Random Forest:

Best Parameters: {'max_depth': 10, 'min_samples_split': 5, 'n_estimators': 100}
Accuracy: 0.940
Performance: The random forest achieves the highest overall accuracy. It excels in classifying class '0' with perfect precision and recall. However, it has difficulty with class '1', having a recall of 0.07 and an F1-score of 0.13.
Conclusion: Random forest is the best performing model overall, but similar to the other models, it faces challenges with the minority class.

- **Logistic Regression**: High accuracy, excellent for majority class, struggles with minority class.
- **Decision Tree**: High accuracy, balanced performance, challenges with minority class.
- **Random Forest**: Highest accuracy, excellent for majority class, significant challenges with minority class.

Conclusion: While all models show high accuracy, they all struggle with the minority class (class '1'). The random forest model slightly outperforms others in overall accuracy but does not substantially improve the classification of the minority class. Future improvements could involve balancing the dataset or using techniques specifically aimed at improving minority class detection.

7. Also use the K-means clustering algorithm to identify clusters of people with certain characteristics (such as education, marital status or income level) and the money they spent on products like Gold, Fruits, Meat, Fish, Sweets and Wines etc.

The Code :

```
# Select relevant features for clustering
features = ['Education', 'Marital_Status', 'Income', 'MntWines', 'MntFruits', 'MntMeatProducts', 'MntFishProducts', 'MntSwe
'MntGoldProds']
clustering_data = marketing_data[features]

# Preprocess categorical features: Education and Marital_Status
clustering_data = pd.get_dummies(clustering_data, columns=['Education', 'Marital_Status'], drop_first=True)

# Impute missing values
imputer = SimpleImputer(strategy='mean')
clustering_data_imputed = imputer.fit_transform(clustering_data)

# Standardize the features
scaler = StandardScaler()
clustering_data_scaled = scaler.fit_transform(clustering_data_imputed)

# Determine the optimum number of clusters using the elbow method
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, random_state=42)
    kmeans.fit(clustering_data_scaled)
    wcss.append(kmeans.inertia_)

plt.figure(figsize=(10, 6))
plt.plot(range(1, 11), wcss, marker='o')
plt.title('Elbow Method for Optimal Number of Clusters')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.grid(True)
plt.show()

# Apply K-Means with the optimal number of clusters (e.g., k=4)
optimal_clusters = 4
kmeans = KMeans(n_clusters=optimal_clusters, random_state=42)
clusters = kmeans.fit_predict(clustering_data_scaled)
```

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

# Visualize Clusters
print("Pairplot visualization of clusters.")
sns.pairplot(marketing_data, hue='Cluster', palette='tab10', plot_kws={'alpha':0.5})
plt.show()

# Summary of clusters
print("Summary statistics of each cluster:")
numeric_cols = clustering_data.select_dtypes(include=[float, int]).columns
print(marketing_data.groupby('Cluster')[numeric_cols].mean())

# Detailed visualization of clusters
print("Scatter plots for detailed visualization.")
plt.figure(figsize=(16, 10))
sns.scatterplot(data=marketing_data, x='Income', y='MntWines', hue='Cluster', palette='tab10')
plt.title('Cluster Visualization: Income vs. Wine Spending')
plt.grid(True)
plt.show()

plt.figure(figsize=(16, 10))
sns.scatterplot(data=marketing_data, x='Income', y='MntMeatProducts', hue='Cluster', palette='tab10')
plt.title('Cluster Visualization: Income vs. Meat Products Spending')
plt.grid(True)
plt.show()

plt.figure(figsize=(16, 10))
sns.scatterplot(data=marketing_data, x='Income', y='MntFishProducts', hue='Cluster', palette='tab10')
plt.title('Cluster Visualization: Income vs. Fish Products Spending')
plt.grid(True)
plt.show()

plt.figure(figsize=(16, 10))
sns.scatterplot(data=marketing_data, x='Income', y='MntSweetProducts', hue='Cluster', palette='tab10')
plt.title('Cluster Visualization: Income vs. Sweet Products Spending')
plt.grid(True)
```

Visual Outputs and Conclusions :

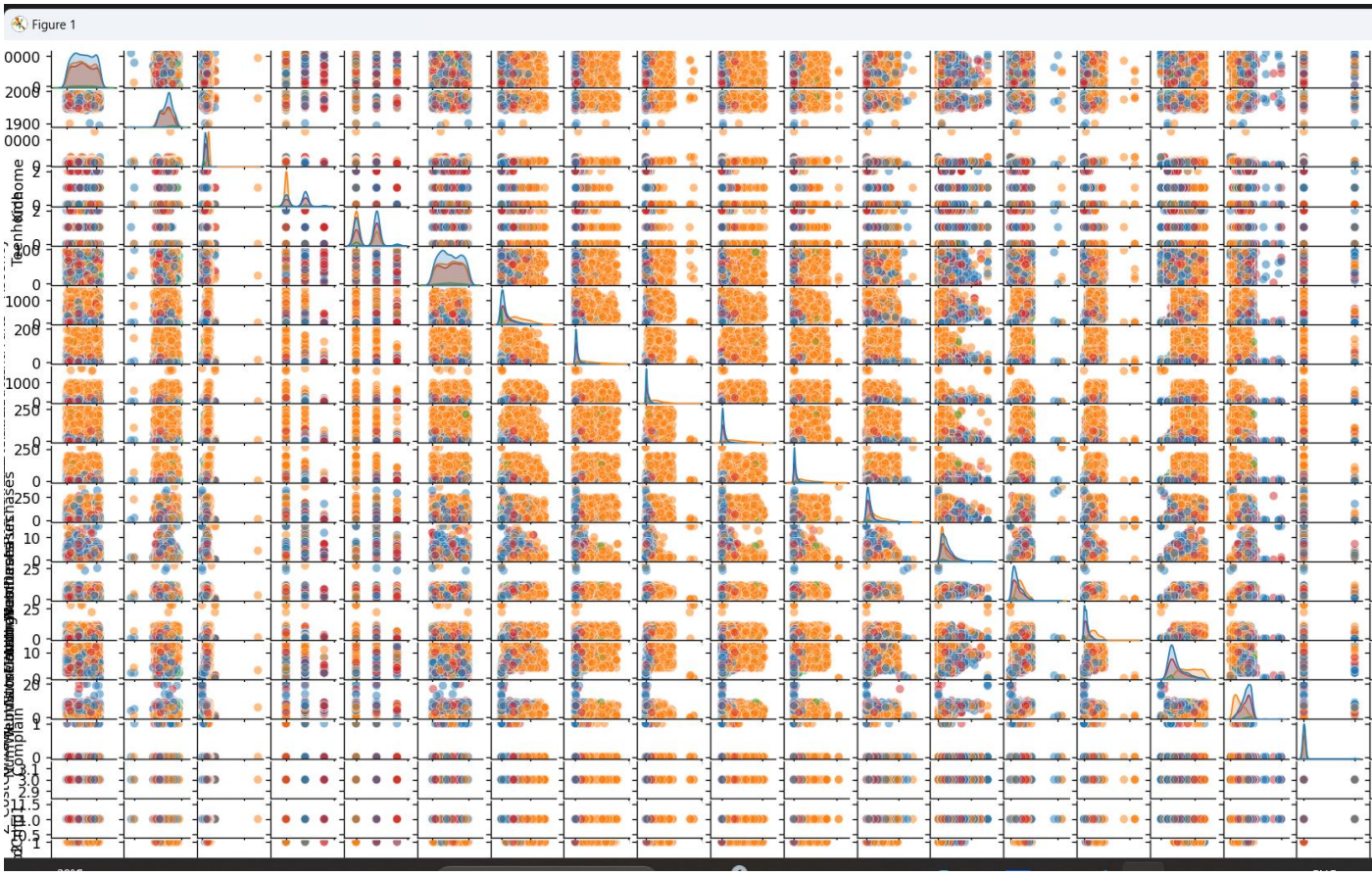
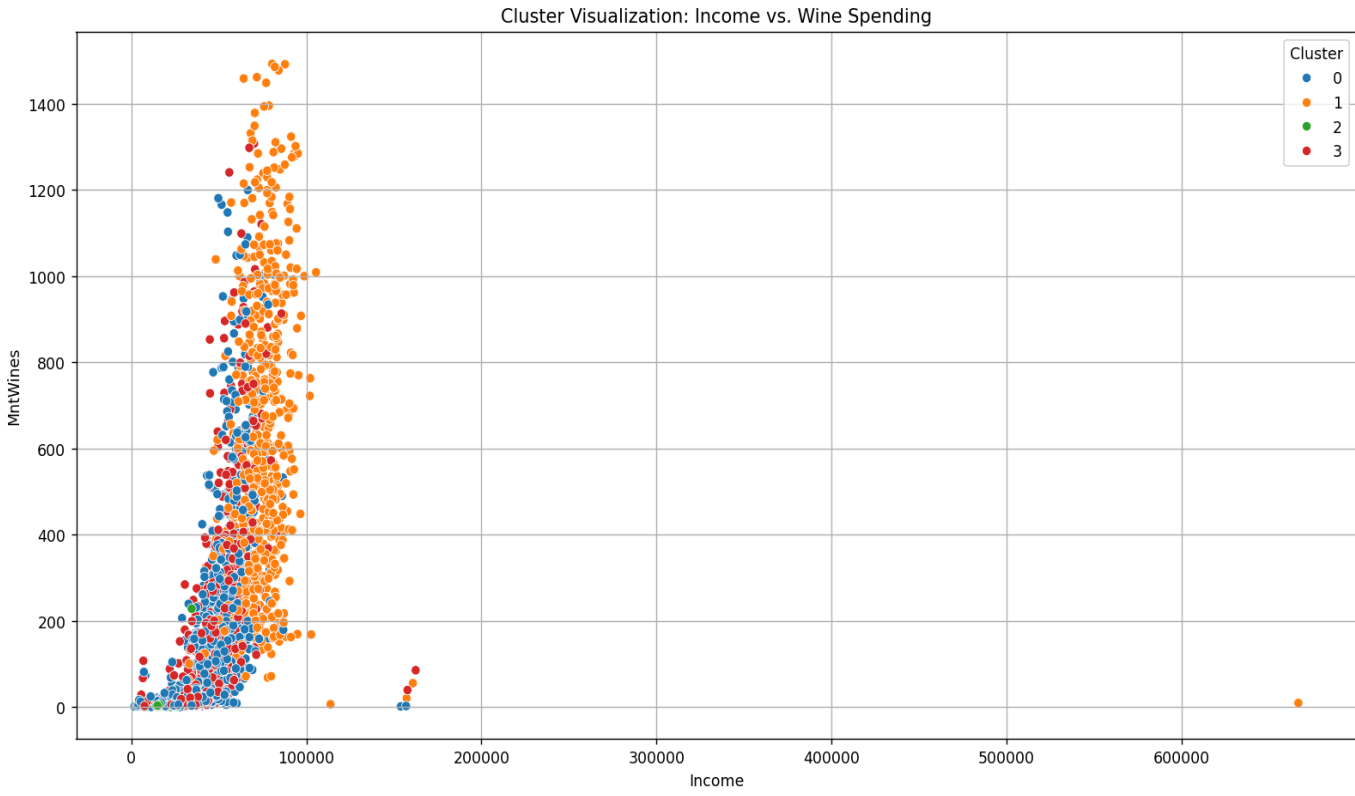
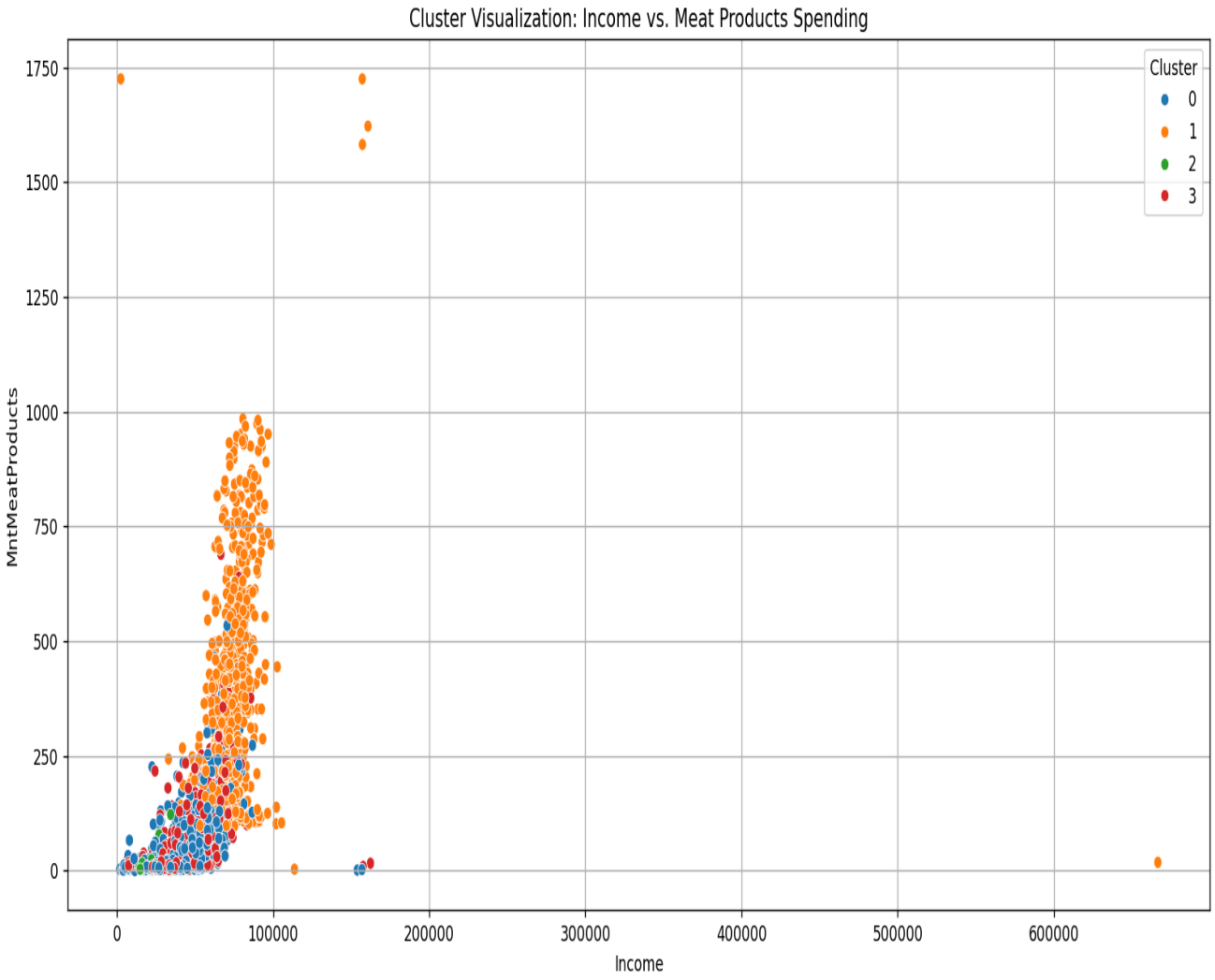


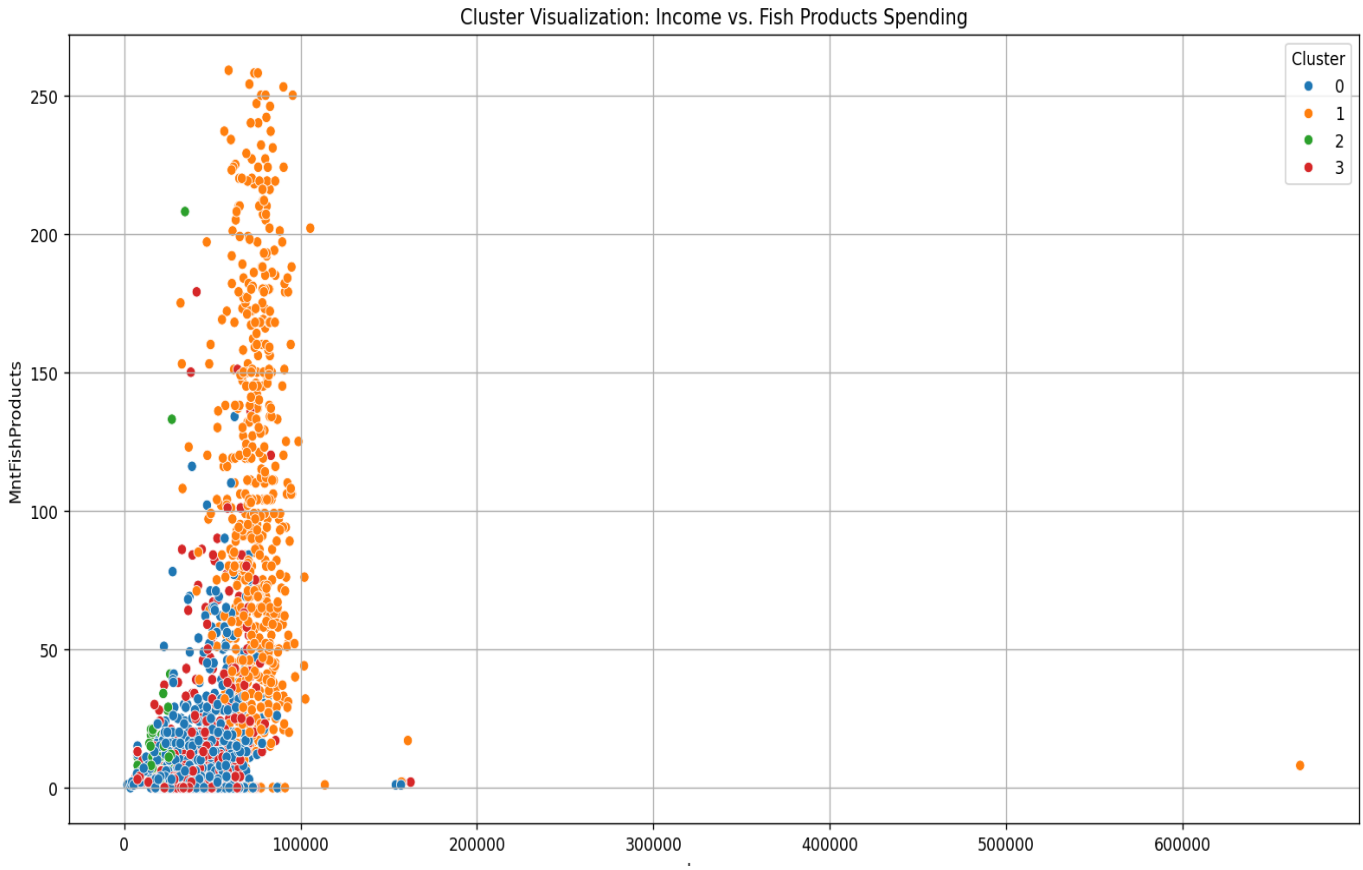
Figure 1


Cluster Visualization: Income vs. Wine Spending
Cluster
0
1
2
3
MntWines
Income
0
200
400
600
800
1000
1200
1400
0
100000
200000
300000
400000
500000
600000

The majority of individuals fall into one cluster, showing consistent spending on wines relative to their income. Higher spending on wines is generally associated with higher incomes, as indicated by the cluster at the top-right of the plot.

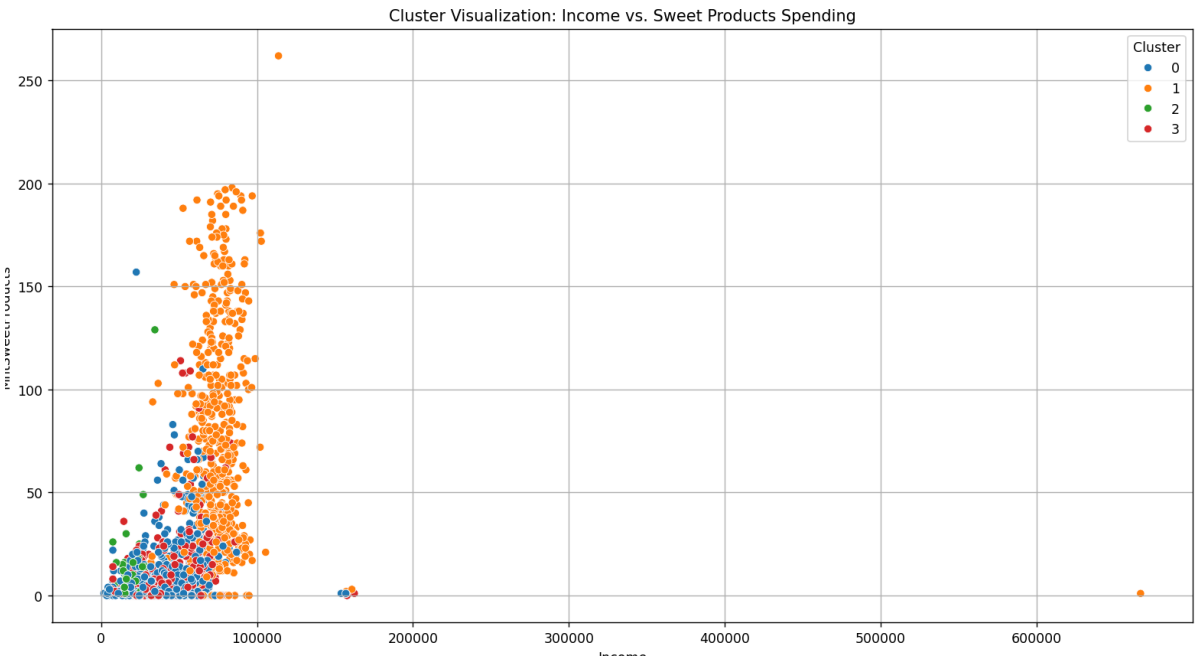


There is a concentration of clusters at lower income levels with varying spending on meat products. A distinct cluster at higher income levels shows increased spending on meat products. This clustering can assist in understanding consumer behavior regarding meat product purchases and tailoring marketing strategies accordingly.

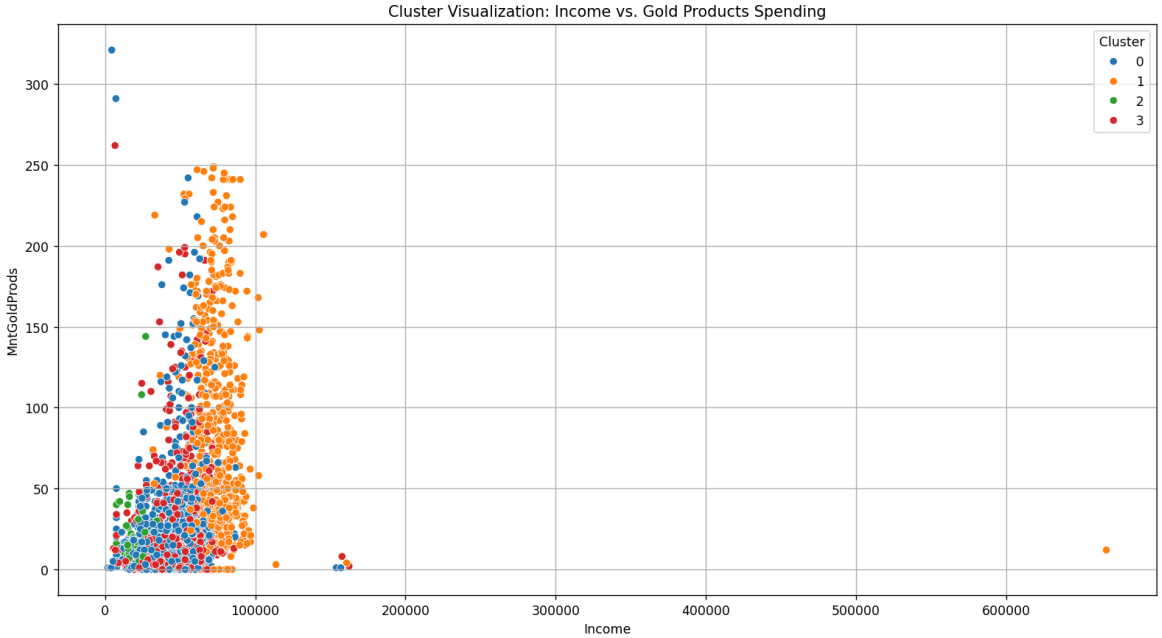


Conclusion

Most of the data points cluster tightly at the lower income range, indicating that fish product spending is more common among lower to middle-income groups. This insight can be useful for businesses in the seafood industry to better target their marketing efforts.



The visualization of Income vs. Sweet Products Spending shows a more dispersed clustering pattern, with a significant concentration of clusters in the lower to middle income ranges. The data suggests that spending on sweet products does not vary as dramatically with income compared to other product categories. This indicates a potentially wider audience for sweet products, making them an attractive category for broad market strategies.



Conclusion

The Income vs. Gold Products Spending plot reveals a clear clustering pattern where higher income levels are associated with higher spending on gold products. The majority of consumers fall into clusters at lower income levels with minimal spending on gold. This visualization helps in identifying affluent consumer segments that are likely to purchase gold products, guiding targeted marketing efforts for luxury items.

***Important Factors to Consider While Using the K-means Algorithm**

- Certain factors can impact the efficacy of the final clusters formed when using k-means clustering. E.g **Number of clusters (K):** The number of clusters you want to group your data points into, has to be predefined[1].

VI.SUMMARY OF FINDINGS AND LIMITATIONS

Logistic Regression: Recall for the positive class was difficult, but accuracy of 74.59% was attained with balanced precision and recall.

Decision Tree: shown superior performance, demonstrating great precision and recall with an accuracy of 97.18%.

With an accuracy of 97.54%, Random Forest fared marginally better than Decision Tree while keeping excellent precision and recall.

Regression models:

The linear regression's R-squared score of 0.596 suggested moderate predictive power.

With an R-squared score of 0.809, decision tree regression performed more accurately than linear regression.

Examining Groups:

There were four groups identified, each with distinct spending patterns. Cluster 1 had the largest expenditure on wines and other things, while Cluster 2 had the lowest spending across the board.

Limitations:

Over fitted model:

Constraints:

Overfitting of the Model,may be limited in their ability to generalise to unknown data due to their tendency towards overfitting.And also,Larger datasets may cause scalability problems for some models, especially Decision Trees and Random Forests, because of their complexity.

VII.REFERENCES :

[1]A. Banerji, “K-Mean | K Means Clustering | Methods To Find The Best Value Of K,” *Analytics Vidhya*, May 18, 2021. https://www.analyticsvidhya.com/blog/2021/05/k-mean-getting-the-optimal-number-of-clusters/