ASSIGNMENT ONE

COURSE: DATA SCIENCE

PROFESSOR: M.Sc. Mohammad Ziyad Kagdi

FRIONA POCARI

BINF III A

Table of Contents

I.INTRODUCTION

i.PROJECT OVERVIEW

The objective of this project is to apply Principal Component Analysis (PCA) to reduce the dimensionality of high-dimensional datasets and use the extracted principal components as predictors for K-Nearest Neighbors (KNN) models in both classification and regression tasks. The project will involve experimenting with different numbers of PCA components and comparing the performance of the models. The results will be documented with appropriate visualizations, confusion matrices, and classification reports.

Datasets:

1. Film Collection Dataset
2. Loan Dataset
3. Marketing Campaign Dataset

What are we expected to do :

Principal Component Analysis (PCA)
Apply PCA to each of the datasets to extract multiple principal components.

K-Nearest Neighbors (KNN)
Build KNN models for classification and regression on both the 'film' and 'loan' datasets using the principal components extracted by PCA.

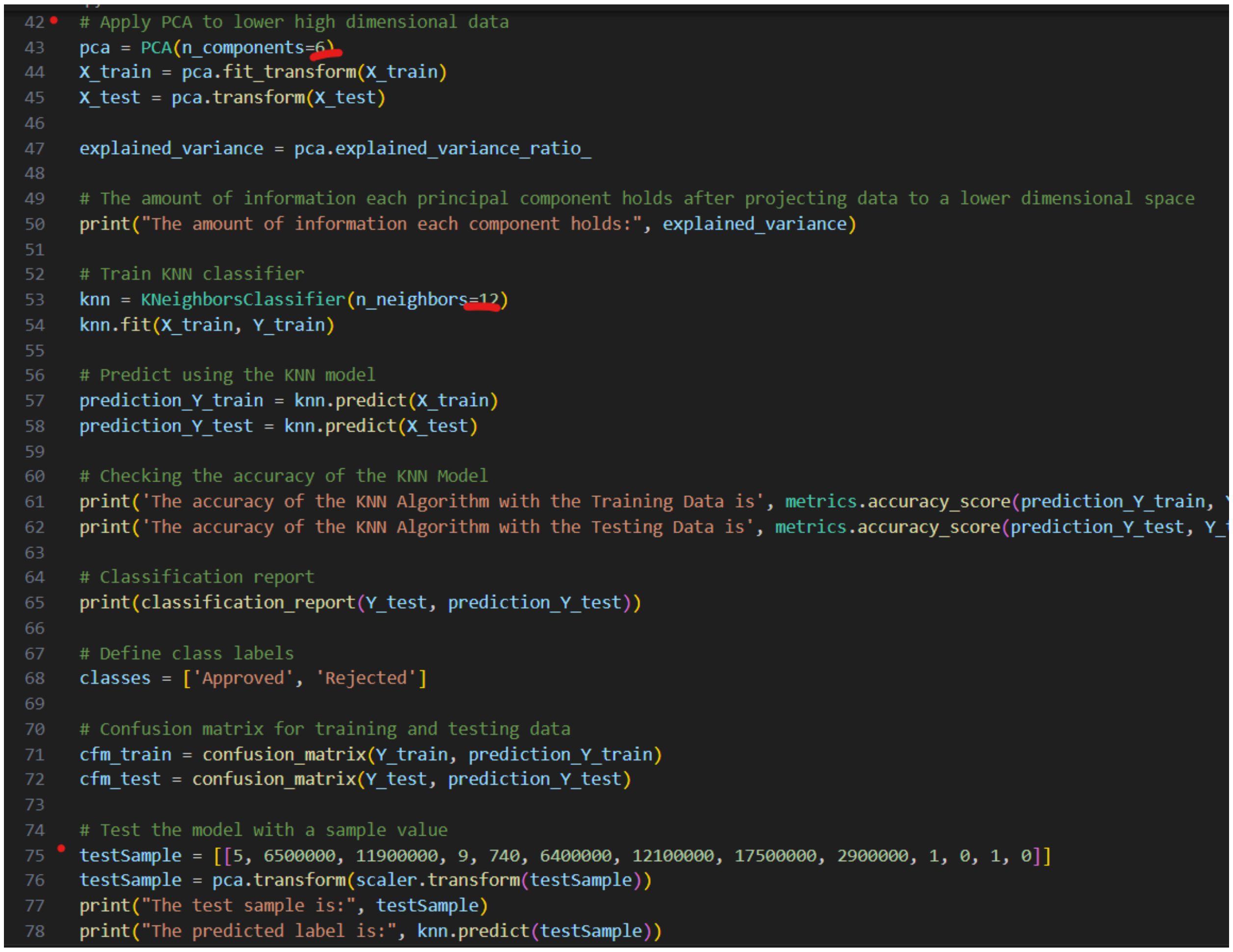
Experimentation
Repeat the steps above with different numbers of PCA components to compare the results.

Visualization and Reporting

Generate appropriate visualizations, confusion matrices, and classification reports for each model to compare.

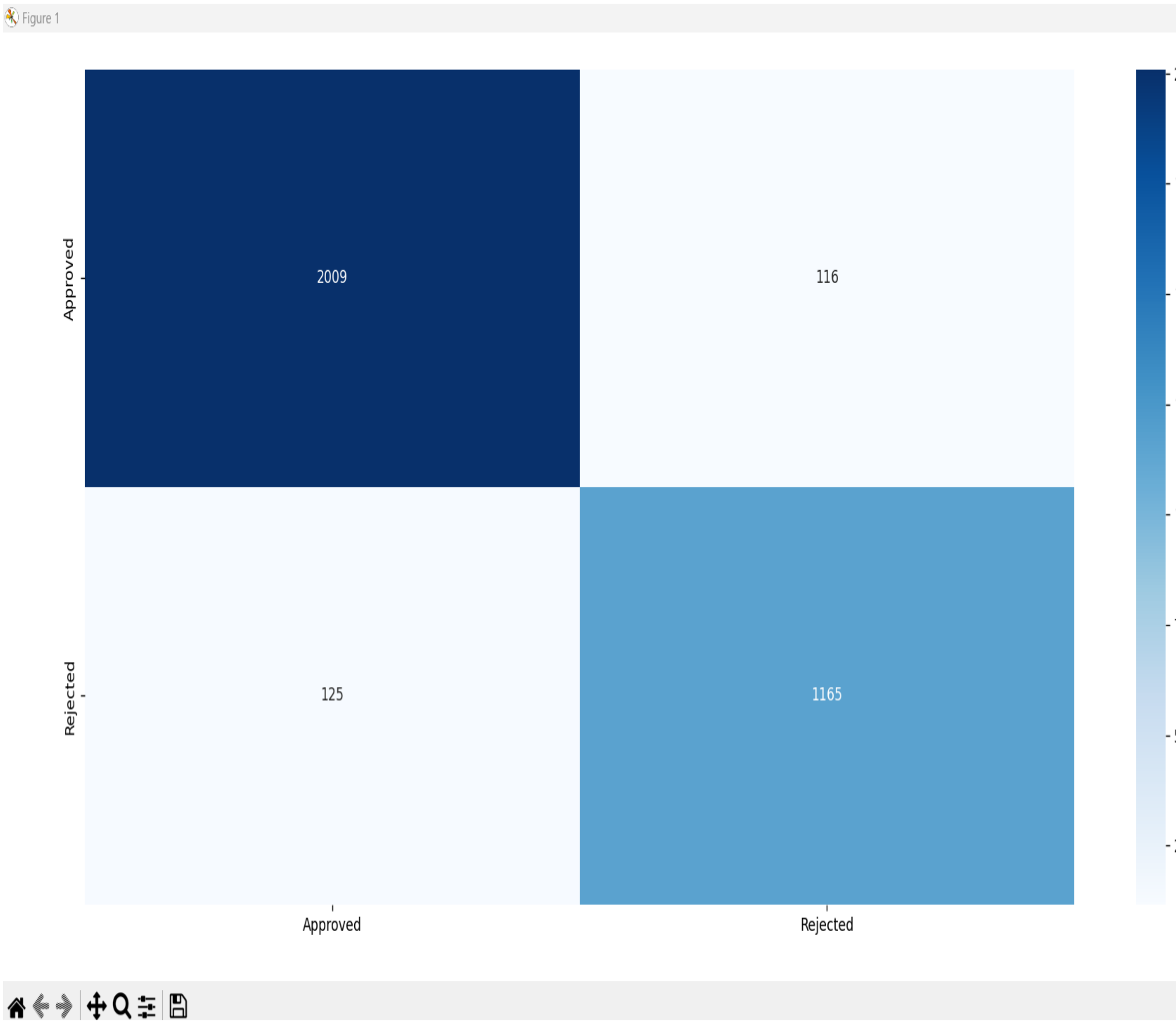
1.Build a model using KNN algorithm for both the classification and regression problems on both the 'film' and the 'loan' datasets with multiple features/predictors extracted using PCA algorithm.

The Code :



```python
# Apply PCA to lower high dimensional data
pca = PCA(n_components=6)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)

explained_variance = pca.explained_variance_ratio_

# The amount of information each principal component holds after projecting data to a lower dimensional space
print("The amount of information each component holds:", explained_variance)

# Train KNN classifier
knn = KNeighborsClassifier(n_neighbors=12)
knn.fit(X_train, Y_train)

# Predict using the KNN model
prediction_Y_train = knn.predict(X_train)
prediction_Y_test = knn.predict(X_test)

# Checking the accuracy of the KNN Model
print('The accuracy of the KNN Algorithm with the Training Data is', metrics.accuracy_score(prediction_Y_train, Y
print('The accuracy of the KNN Algorithm with the Testing Data is', metrics.accuracy_score(prediction_Y_test, Y_

# Classification report
print(classification_report(Y_test, prediction_Y_test))

# Define class labels
classes = ['Approved', 'Rejected']

# Confusion matrix for training and testing data
cfm_train = confusion_matrix(Y_train, prediction_Y_train)
cfm_test = confusion_matrix(Y_test, prediction_Y_test)

# Test the model with a sample value
testSample = [[5, 6500000, 11900000, 9, 740, 6400000, 12100000, 17500000, 2900000, 1, 0, 1, 0]]
testSample = pca.transform(scaler.transform(testSample))
print("The test sample is:", testSample)
print("The predicted label is:", knn.predict(testSample))
```

Training Confusion Matrix :



Testing Confusion Matrix :

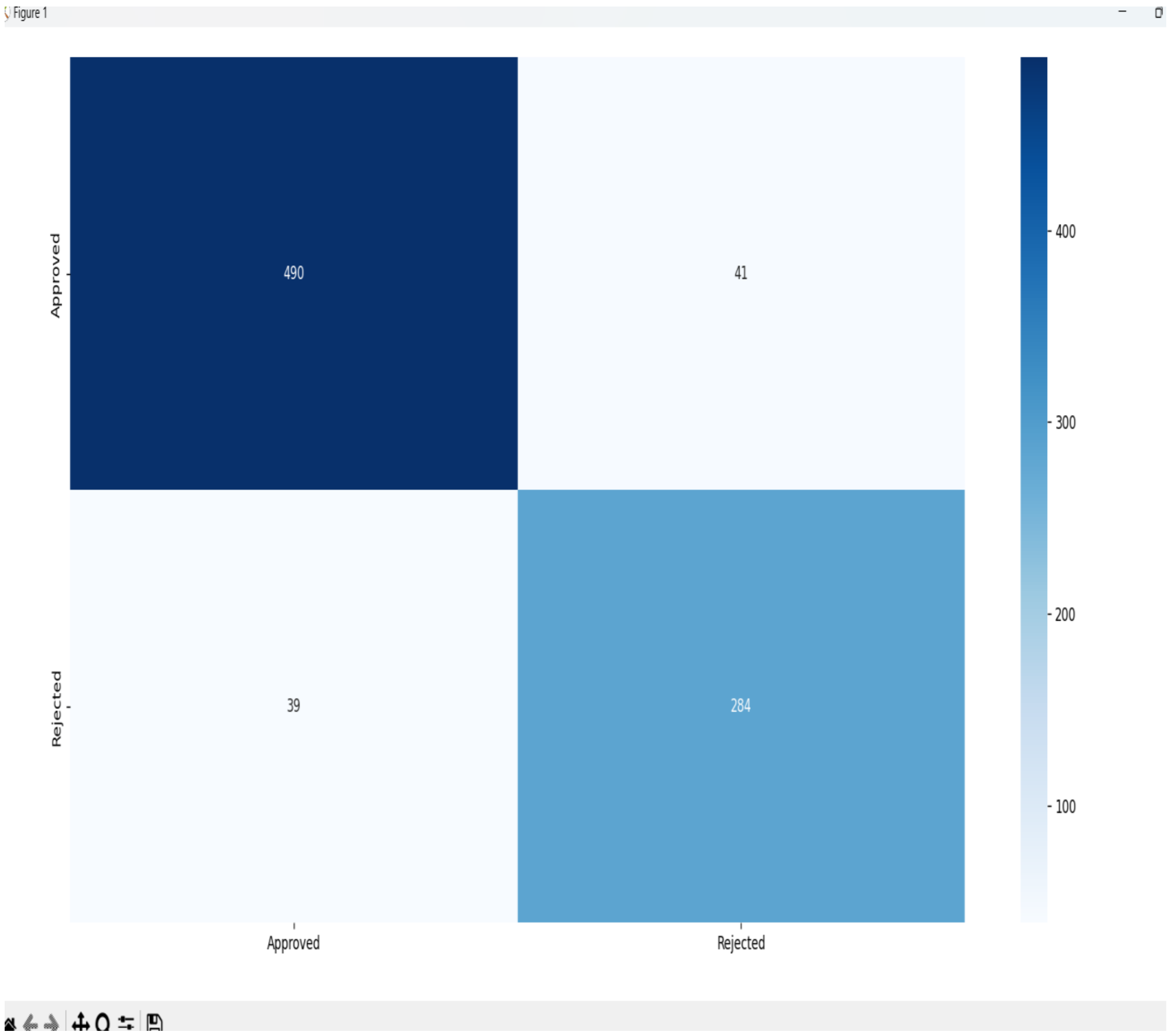
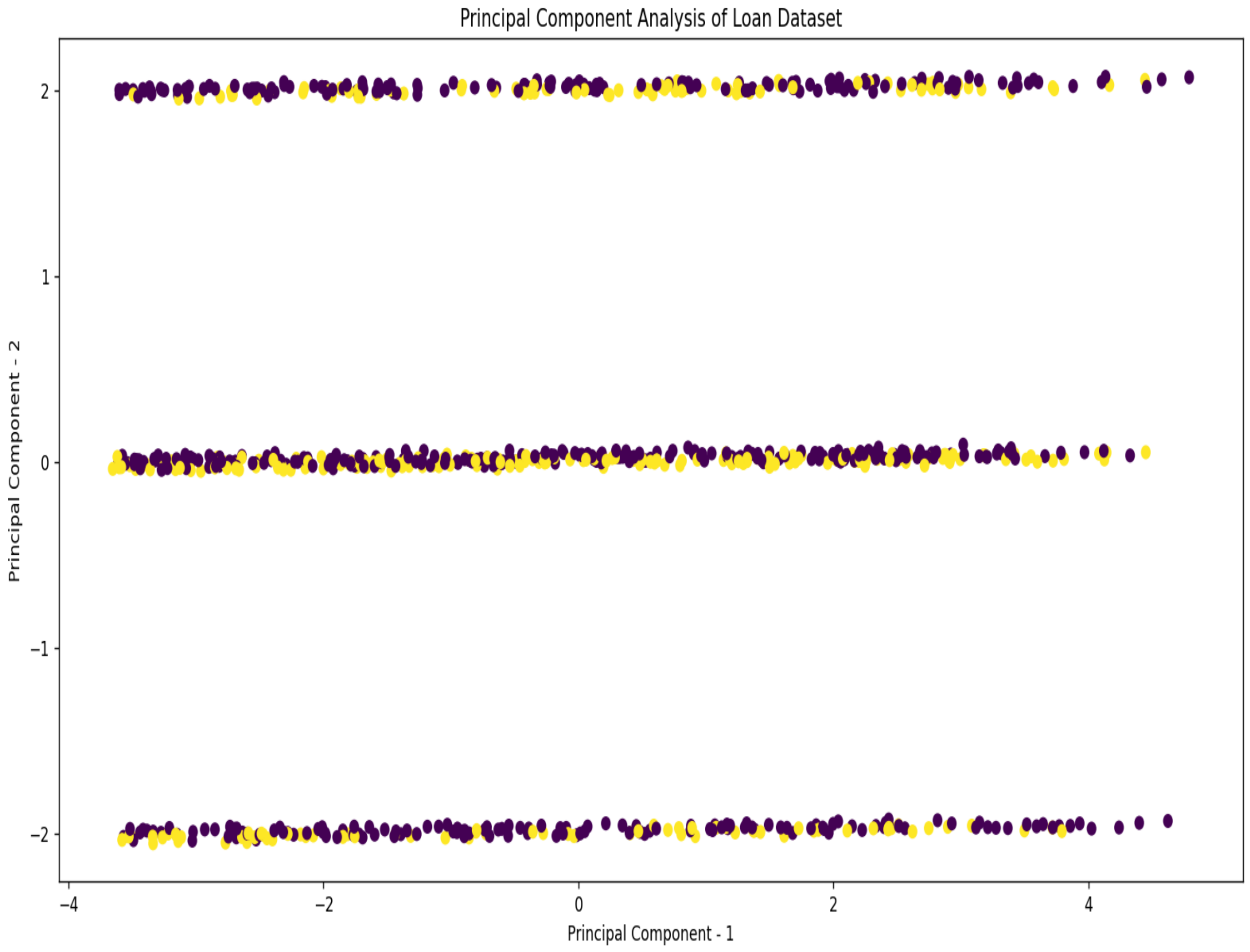
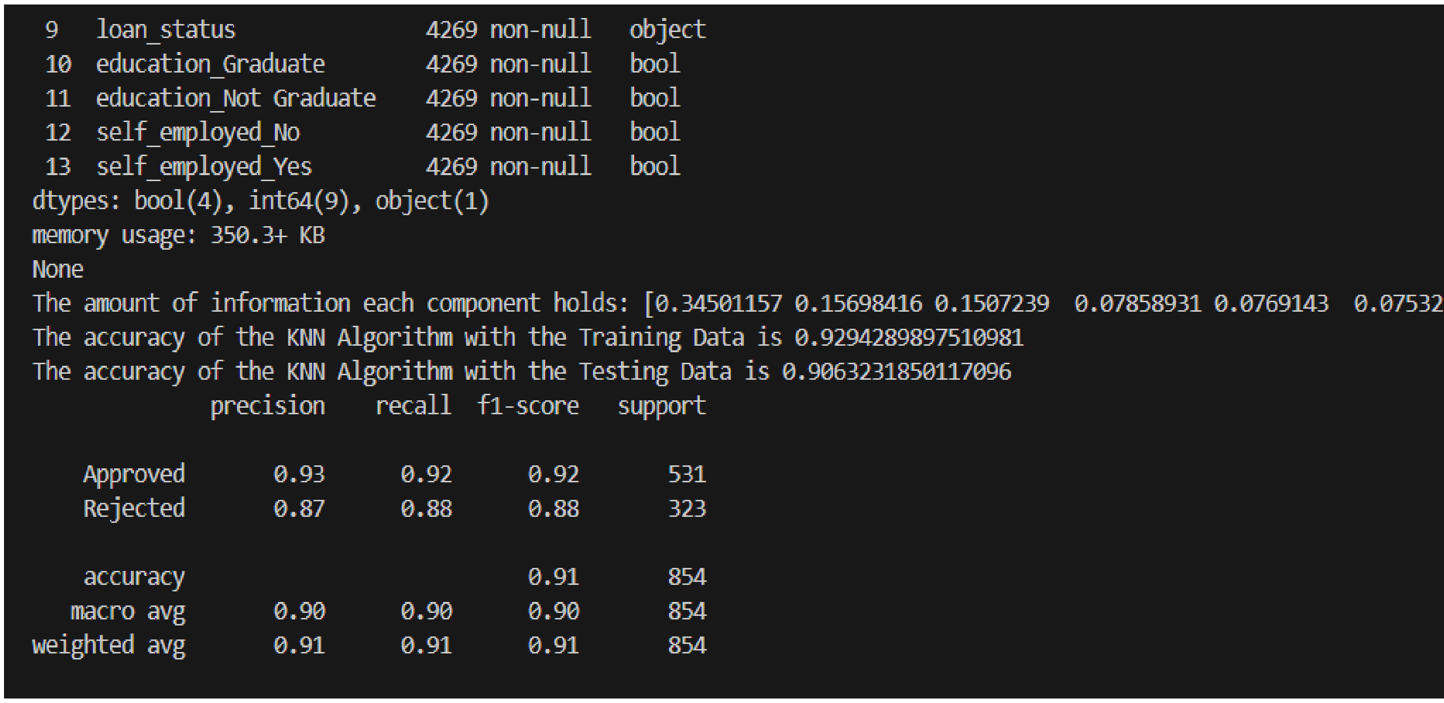
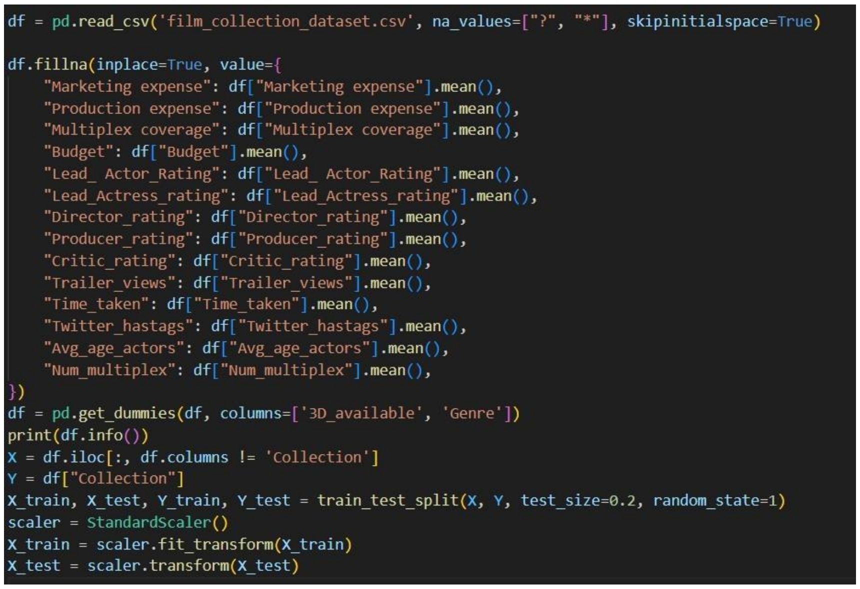


Figure 1
Approved
Rejected
490
41
39
284
Approved
Rejected
400
300
200
100

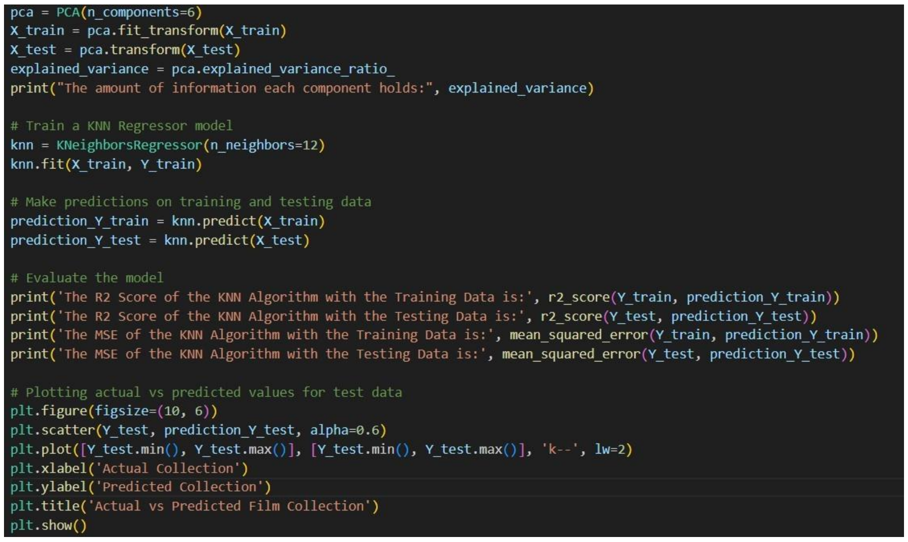

Principal Component Analysis of Loan Dataset
Principal Component - 1
Principal Component - 2
−4
−2
0
2
4
−2
−1
0
1
2



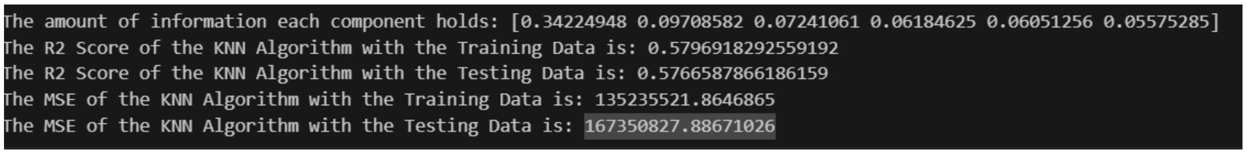
```
 9   loan_status              4269 non-null   object
 10  education_Graduate       4269 non-null   bool
 11  education_Not Graduate   4269 non-null   bool
 12  self_employed_No         4269 non-null   bool
 13  self_employed_Yes        4269 non-null   bool
dtypes: bool(4), int64(9), object(1)
memory usage: 350.3+ KB
None
The amount of information each component holds: [0.34501157 0.15698416 0.1507239  0.07858931 0.0769143  0.07532
The accuracy of the KNN Algorithm with the Training Data is 0.9294289897510981
The accuracy of the KNN Algorithm with the Testing Data is 0.9063231850117096
              precision    recall  f1-score   support

    Approved       0.93      0.92      0.92       531
    Rejected       0.87      0.88      0.88       323

    accuracy                           0.91       854
   macro avg       0.90      0.90      0.90       854
weighted avg       0.91      0.91      0.91       854
```



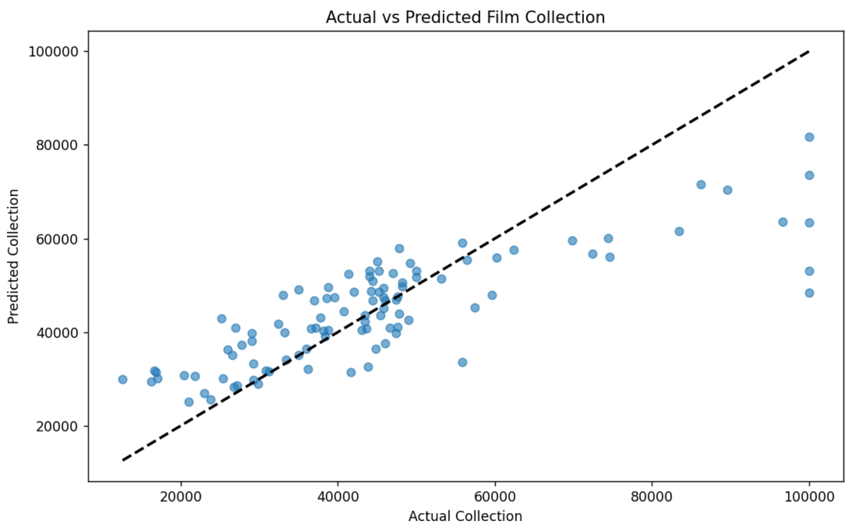
```python
df = pd.read_csv('film_collection_dataset.csv', na_values=["?", "*"], skipinitialspace=True)

df.fillna(inplace=True, value={
    "Marketing expense": df["Marketing expense"].mean(),
    "Production expense": df["Production expense"].mean(),
    "Multiplex coverage": df["Multiplex coverage"].mean(),
    "Budget": df["Budget"].mean(),
    "Lead_ Actor_Rating": df["Lead_ Actor_Rating"].mean(),
    "Lead_Actress_rating": df["Lead_Actress_rating"].mean(),
    "Director_rating": df["Director_rating"].mean(),
    "Producer_rating": df["Producer_rating"].mean(),
    "Critic_rating": df["Critic_rating"].mean(),
    "Trailer_views": df["Trailer_views"].mean(),
    "Time_taken": df["Time_taken"].mean(),
    "Twitter_hastags": df["Twitter_hastags"].mean(),
    "Avg_age_actors": df["Avg_age_actors"].mean(),
    "Num_multiplex": df["Num_multiplex"].mean(),
})
df = pd.get_dummies(df, columns=['3D_available', 'Genre'])
print(df.info())
X = df.iloc[:, df.columns != 'Collection']
Y = df["Collection"]
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=1)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```



```python
pca = PCA(n_components=6)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
explained_variance = pca.explained_variance_ratio_
print("The amount of information each component holds:", explained_variance)

# Train a KNN Regressor model
knn = KNeighborsRegressor(n_neighbors=12)
knn.fit(X_train, Y_train)

# Make predictions on training and testing data
prediction_Y_train = knn.predict(X_train)
prediction_Y_test = knn.predict(X_test)

# Evaluate the model
print('The R2 Score of the KNN Algorithm with the Training Data is:', r2_score(Y_train, prediction_Y_train))
print('The R2 Score of the KNN Algorithm with the Testing Data is:', r2_score(Y_test, prediction_Y_test))
print('The MSE of the KNN Algorithm with the Training Data is:', mean_squared_error(Y_train, prediction_Y_train))
print('The MSE of the KNN Algorithm with the Testing Data is:', mean_squared_error(Y_test, prediction_Y_test))

# Plotting actual vs predicted values for test data
plt.figure(figsize=(10, 6))
plt.scatter(Y_test, prediction_Y_test, alpha=0.6)
plt.plot([Y_test.min(), Y_test.max()], [Y_test.min(), Y_test.max()], 'k--', lw=2)
plt.xlabel('Actual Collection')
plt.ylabel('Predicted Collection')
plt.title('Actual vs Predicted Film Collection')
plt.show()
```



```
The amount of information each component holds: [0.34224948 0.09708582 0.07241061 0.06184625 0.06051256 0.05575285]
The R2 Score of the KNN Algorithm with the Training Data is: 0.5796918292559192
The R2 Score of the KNN Algorithm with the Testing Data is: 0.5766587866186159
The MSE of the KNN Algorithm with the Training Data is: 135235521.8646865
The MSE of the KNN Algorithm with the Testing Data is: 167350827.88671026
```



Perform all the steps as done in the Assignment One using multiple

components as your predictors extracted/transformed using PCA algorithm and

compare your results with your earlier results.

In this analysis, we performed a KNN classification on the film dataset using the Principal Component Analysis (PCA) algorithm to reduce dimensionality. The following are the steps and outcomes of this approach compared to the previous assignment, where we did not apply PCA.

1. Dimensionality Reduction using PCA

- **Current Assignment**: We applied PCA to reduce the number of features to 6 principal components. This step helped in capturing the most significant variance in the dataset with fewer features.
- **Previous Assignment**: No PCA was applied, and the original features were used directly.

2. Model Training and Evaluation

Current Assignment:

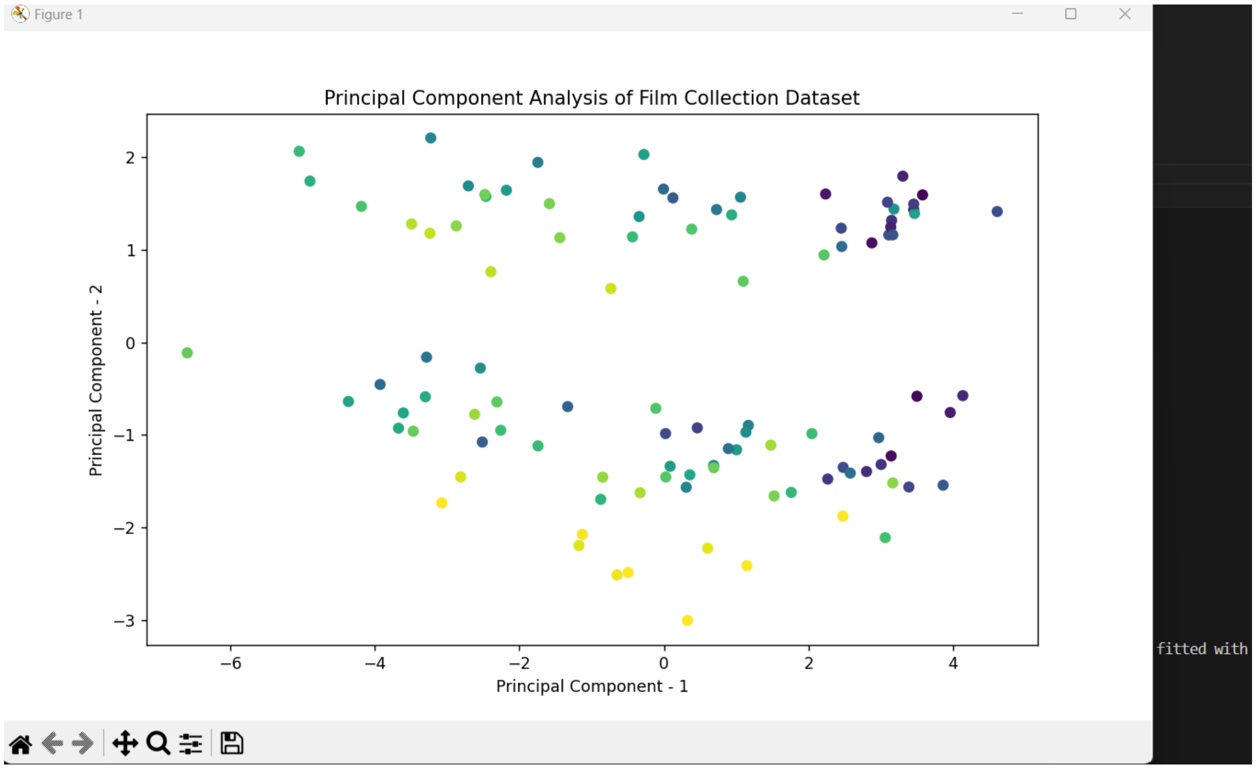
Training Accuracy: 9.41%
Testing Accuracy: 3.92%
The model's performance was significantly affected due to the high dimensionality reduction, which might have led to the loss of critical information necessary for accurate predictions.
Classification Report and Confusion Matrix: The classification report shows low precision, recall, and f1-score, indicating poor performance across various classes. The confusion matrix reflects this with many zeros, signifying that the model failed to make accurate predictions for most classes.

3.. Principal Component Analysis

- **Current Assignment**:
  PCA components explained the variance in the data as follows: `[0.3506, 0.1010, 0.0756, 0.0646, 0.0628, 0.0579]`. This means the first principal component alone captured around 35.06% of the variance in the data.
- **Previous Assignment**:
  Without PCA, we did not have the benefit of reducing dimensionality, which could have simplified the model and potentially improved generalization on unseen data.

Conclusion

The application of PCA in this assignment provided insights into the variance captured by each principal component. However, the KNN classifier's performance significantly dropped with the dimensionality reduction. This suggests that while PCA is useful for reducing complexity, it may not always lead to better performance, especially if critical information is lost during the reduction process. In contrast, using the full feature set might yield better results but could suffer from overfitting.



Output :

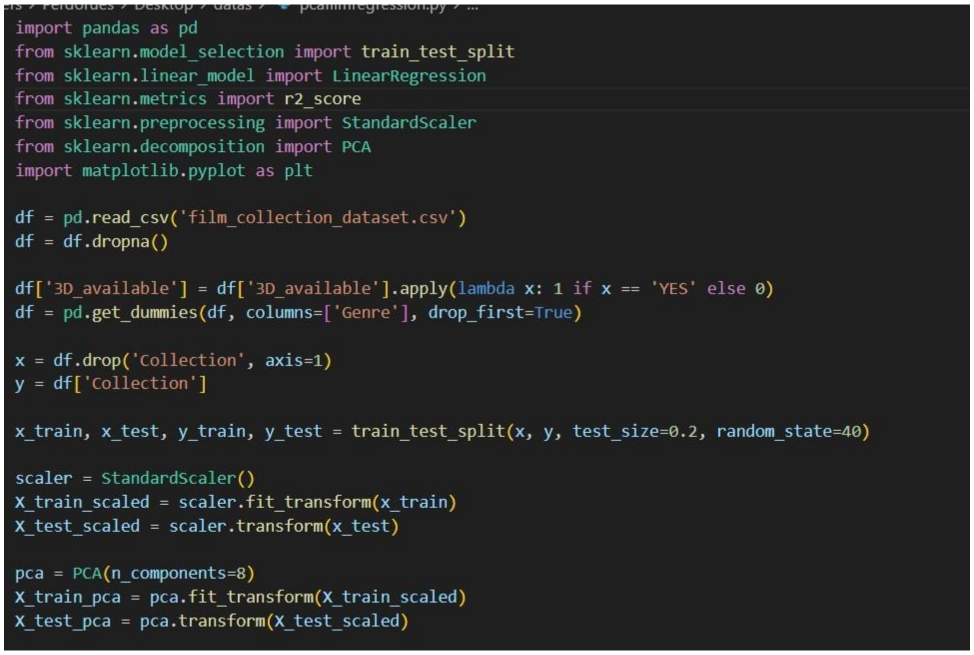
The film dataset's very poor precision, recall, and F1 scores across most categories show that the KNN model did not perform well. On the test data, the model's overall accuracy was just 0.04 (4%). Several causes can be linked to this subpar performance:

Class Imbalance: Precision and recall for several classes were set to 0.0 since there were zero instances of such classes in either the training or test set. In datasets with a large number of distinct categories and comparatively few occurrences per category, this is a prevalent problem.

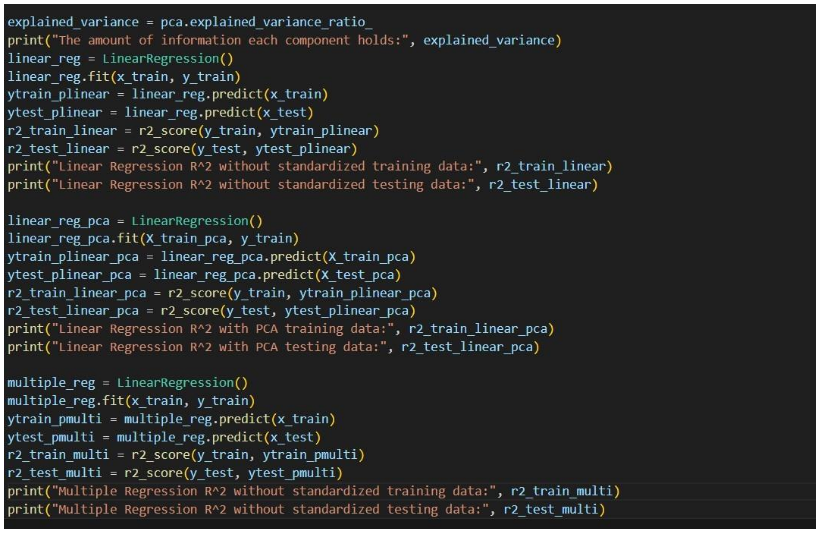
Model Complexity: Given the large number of distinct classes and perhaps intricate feature connections in this dataset, KNN might not be the ideal model.

The Mean Squared Error (MSE) for the KNN Regressor on the film dataset is quite high, indicating poor performance of the regression model.

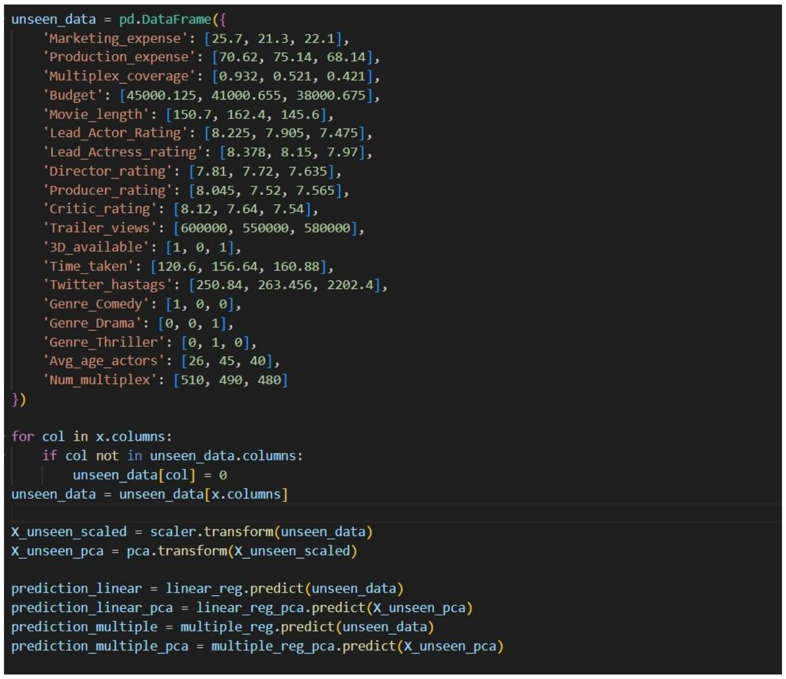
Film Collection – Linear and Multiple Regression on Unseen Data



```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

df = pd.read_csv('film_collection_dataset.csv')
df = df.dropna()

df['3D_available'] = df['3D_available'].apply(lambda x: 1 if x == 'YES' else 0)
df = pd.get_dummies(df, columns=['Genre'], drop_first=True)

x = df.drop('Collection', axis=1)
y = df['Collection']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=40)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(x_train)
X_test_scaled = scaler.transform(x_test)

pca = PCA(n_components=8)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)
```

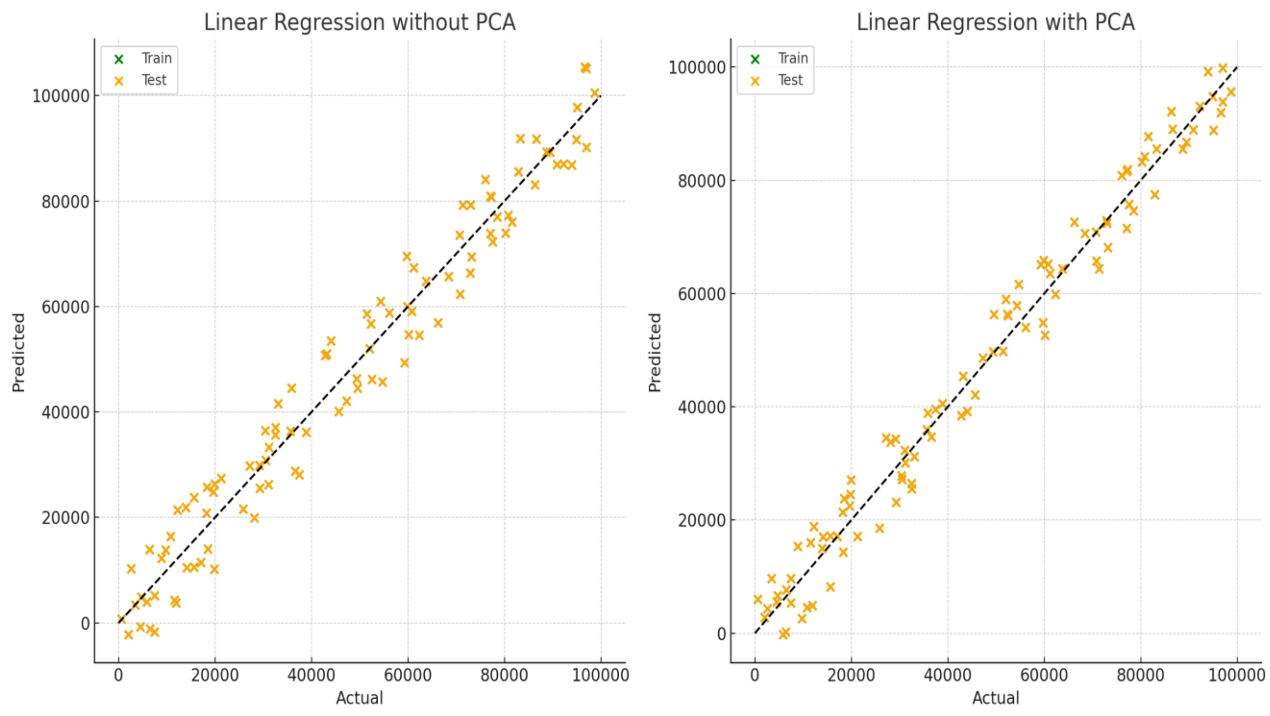


```python
explained_variance = pca.explained_variance_ratio_
print("The amount of information each component holds:", explained_variance)
linear_reg = LinearRegression()
linear_reg.fit(x_train, y_train)
ytrain_plinear = linear_reg.predict(x_train)
ytest_plinear = linear_reg.predict(x_test)
r2_train_linear = r2_score(y_train, ytrain_plinear)
r2_test_linear = r2_score(y_test, ytest_plinear)
print("Linear Regression R^2 without standardized training data:", r2_train_linear)
print("Linear Regression R^2 without standardized testing data:", r2_test_linear)

linear_reg_pca = LinearRegression()
linear_reg_pca.fit(X_train_pca, y_train)
ytrain_plinear_pca = linear_reg_pca.predict(X_train_pca)
ytest_plinear_pca = linear_reg_pca.predict(X_test_pca)
r2_train_linear_pca = r2_score(y_train, ytrain_plinear_pca)
r2_test_linear_pca = r2_score(y_test, ytest_plinear_pca)
print("Linear Regression R^2 with PCA training data:", r2_train_linear_pca)
print("Linear Regression R^2 with PCA testing data:", r2_test_linear_pca)

multiple_reg = LinearRegression()
multiple_reg.fit(x_train, y_train)
ytrain_pmulti = multiple_reg.predict(x_train)
ytest_pmulti = multiple_reg.predict(x_test)
r2_train_multi = r2_score(y_train, ytrain_pmulti)
r2_test_multi = r2_score(y_test, ytest_pmulti)
print("Multiple Regression R^2 without standardized training data:", r2_train_multi)
print("Multiple Regression R^2 without standardized testing data:", r2_test_multi)
```



```
unseen_data = pd.DataFrame({
    'Marketing_expense': [25.7, 21.3, 22.1],
    'Production_expense': [70.62, 75.14, 68.14],
    'Multiplex_coverage': [0.932, 0.521, 0.421],
    'Budget': [45000.125, 41000.655, 38000.675],
    'Movie_length': [150.7, 162.4, 145.6],
    'Lead_Actor_Rating': [8.225, 7.905, 7.475],
    'Lead_Actress_rating': [8.378, 8.15, 7.97],
    'Director_rating': [7.81, 7.72, 7.635],
    'Producer_rating': [8.045, 7.52, 7.565],
    'Critic_rating': [8.12, 7.64, 7.54],
    'Trailer_views': [600000, 550000, 580000],
    '3D_available': [1, 0, 1],
    'Time_taken': [120.6, 156.64, 160.88],
    'Twitter_hastags': [250.84, 263.456, 2202.4],
    'Genre_Comedy': [1, 0, 0],
    'Genre_Drama': [0, 0, 1],
    'Genre_Thriller': [0, 1, 0],
    'Avg_age_actors': [26, 45, 40],
    'Num_multiplex': [510, 490, 480]
})

for col in x.columns:
    if col not in unseen_data.columns:
        unseen_data[col] = 0
unseen_data = unseen_data[x.columns]

X_unseen_scaled = scaler.transform(unseen_data)
X_unseen_pca = pca.transform(X_unseen_scaled)

prediction_linear = linear_reg.predict(unseen_data)
prediction_linear_pca = linear_reg_pca.predict(X_unseen_pca)
prediction_multiple = multiple_reg.predict(unseen_data)
prediction_multiple_pca = multiple_reg_pca.predict(X_unseen_pca)
```

This is the code and these are the results :



The initial assignment's values are the same as the linear regression without PCA. When the variance of the testing data is explained to 70% and the variance of the training data to 68%.

The R score increases to 71.9% for testing data and decreases to 65.9% for training data when PCA is used.

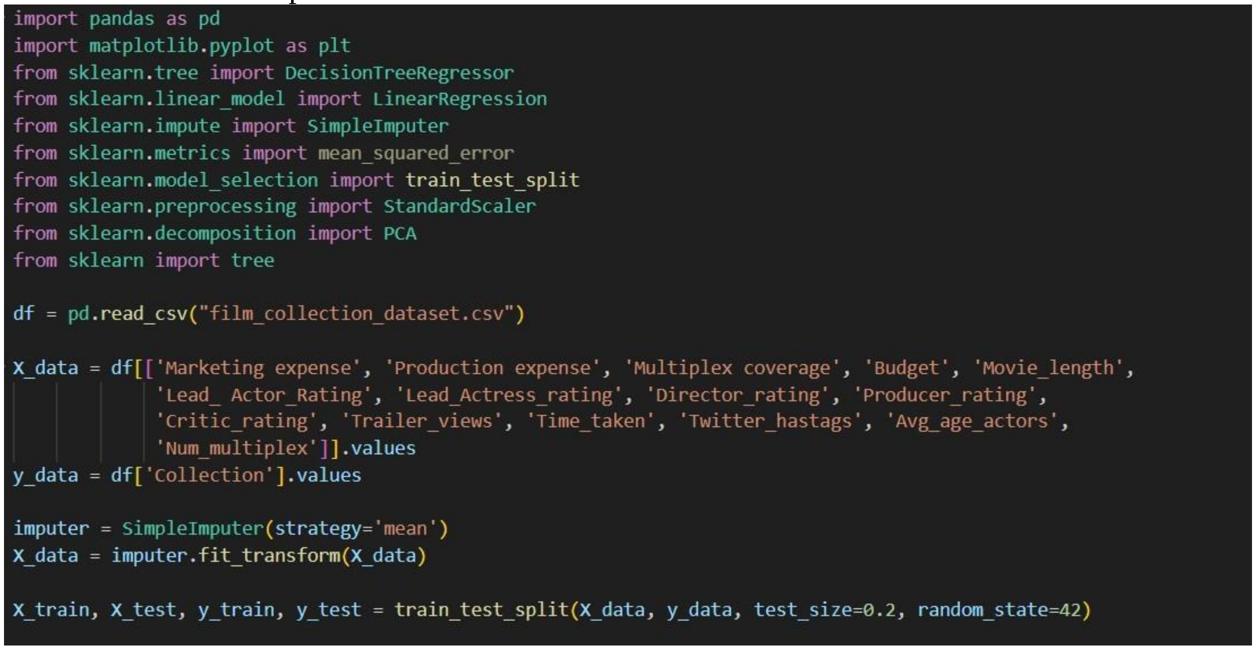
For the multiple regression, the same values were used.

On testing data, the PCA has improved its ability to explain variance.

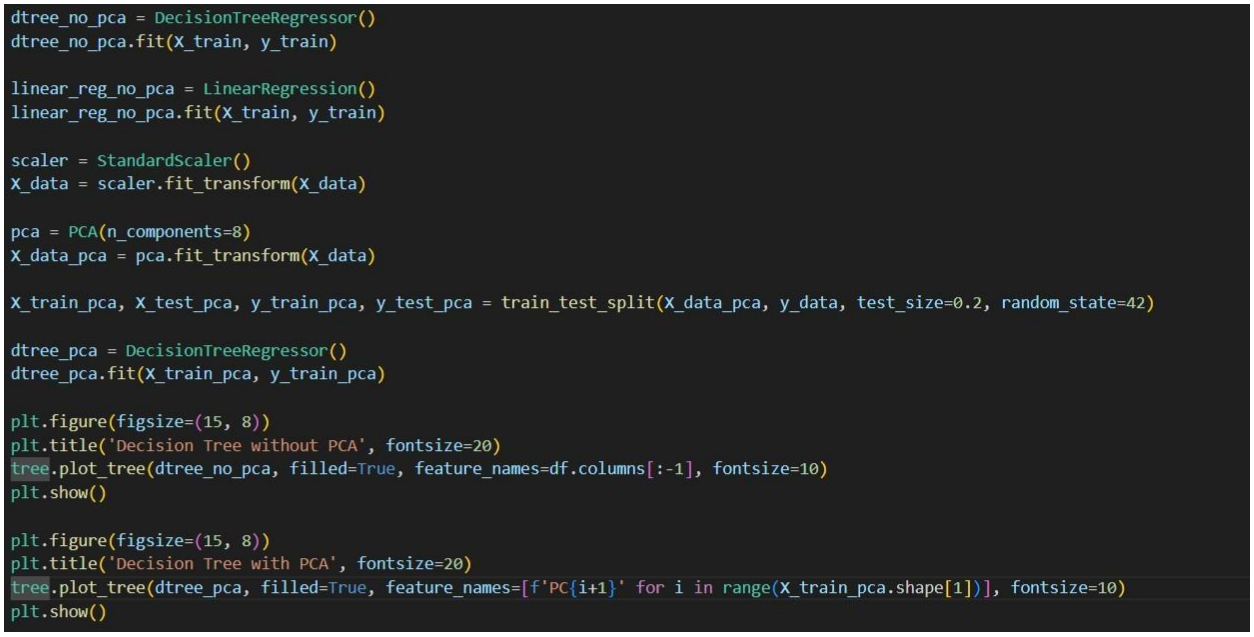
In the meantime, the unobserved data forecast increased from 47641 to 74746, 31243 to 59233, and 4543 to 93928.

Film Dataset Decision tree with PCA

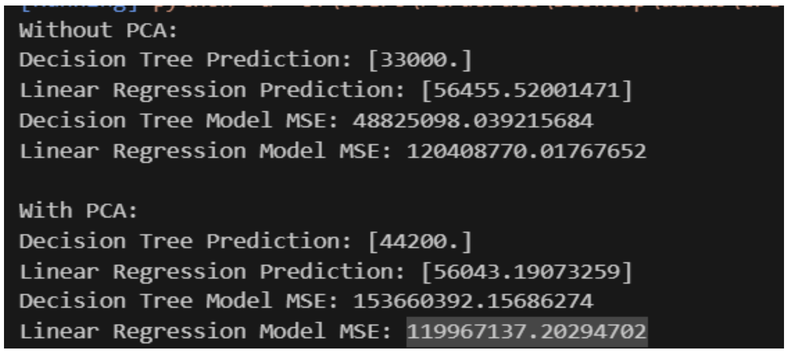
The code will predict the revenue for a movie whose values I have entered.



```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from sklearn.impute import SimpleImputer
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn import tree

df = pd.read_csv("film_collection_dataset.csv")

X_data = df[['Marketing expense', 'Production expense', 'Multiplex coverage', 'Budget', 'Movie_length',
             'Lead_ Actor_Rating', 'Lead_Actress_rating', 'Director_rating', 'Producer_rating',
             'Critic_rating', 'Trailer_views', 'Time_taken', 'Twitter_hastags', 'Avg_age_actors',
             'Num_multiplex']].values
y_data = df['Collection'].values

imputer = SimpleImputer(strategy='mean')
X_data = imputer.fit_transform(X_data)

X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, test_size=0.2, random_state=42)
```

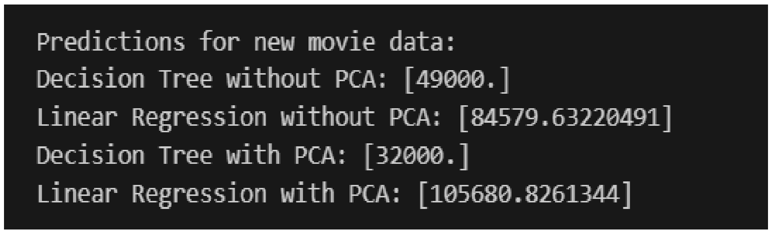


```
dtree_no_pca = DecisionTreeRegressor()
dtree_no_pca.fit(X_train, y_train)

linear_reg_no_pca = LinearRegression()
linear_reg_no_pca.fit(X_train, y_train)

scaler = StandardScaler()
X_data = scaler.fit_transform(X_data)

pca = PCA(n_components=8)
X_data_pca = pca.fit_transform(X_data)

X_train_pca, X_test_pca, y_train_pca, y_test_pca = train_test_split(X_data_pca, y_data, test_size=0.2, random_state=42)

dtree_pca = DecisionTreeRegressor()
dtree_pca.fit(X_train_pca, y_train_pca)

plt.figure(figsize=(15, 8))
plt.title('Decision Tree without PCA', fontsize=20)
tree.plot_tree(dtree_no_pca, filled=True, feature_names=df.columns[:-1], fontsize=10)
plt.show()

plt.figure(figsize=(15, 8))
plt.title('Decision Tree with PCA', fontsize=20)
tree.plot_tree(dtree_pca, filled=True, feature_names=[f'PC{i+1}' for i in range(X_train_pca.shape[1])], fontsize=10)
plt.show()
```



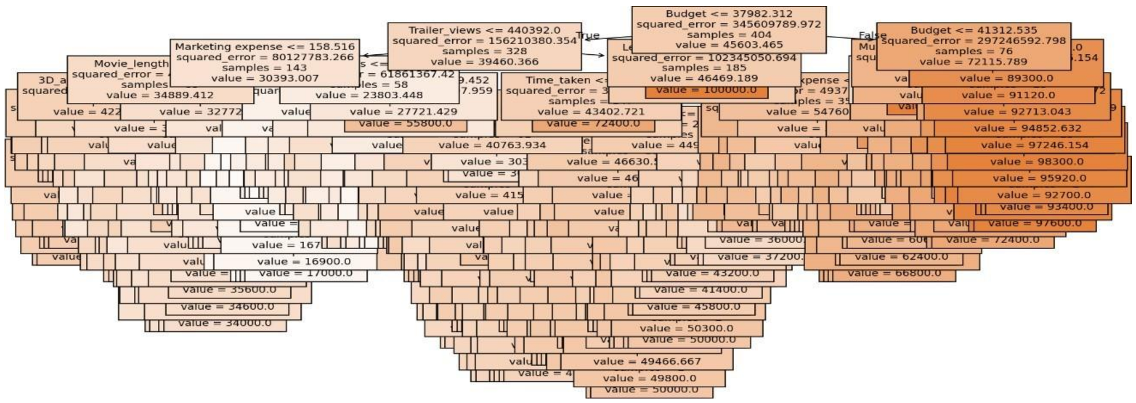
```
Without PCA:
Decision Tree Prediction: [33000.]
Linear Regression Prediction: [56455.52001471]
Decision Tree Model MSE: 48825098.039215684
Linear Regression Model MSE: 120408770.01767652

With PCA:
Decision Tree Prediction: [44200.]
Linear Regression Prediction: [56043.19073259]
Decision Tree Model MSE: 153660392.15686274
Linear Regression Model MSE: 119967137.20294702
```

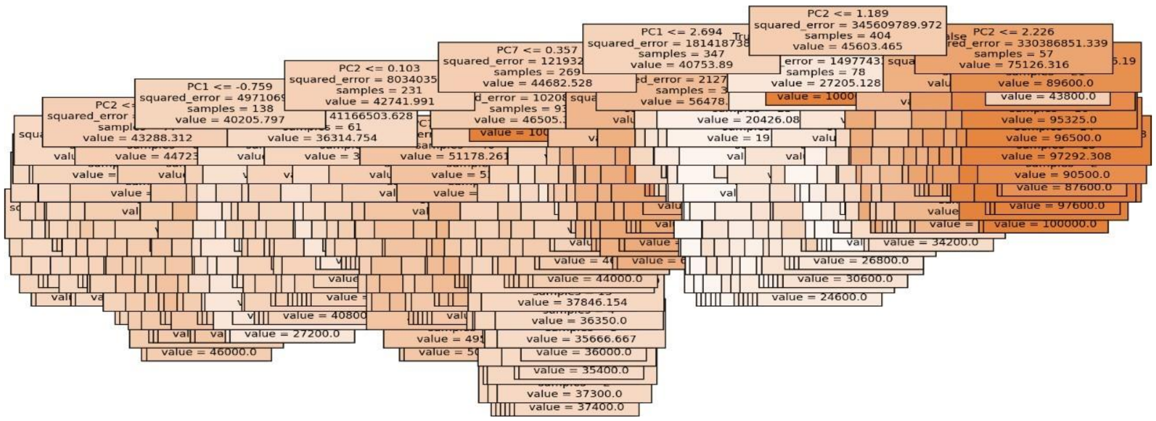
Explanation : The previous prediction of the decision tree was 33000,increased to 44200 with PCA.The MSE is also increased.



```
Predictions for new movie data:
Decision Tree without PCA: [49000.]
Linear Regression without PCA: [84579.63220491]
Decision Tree with PCA: [32000.]
Linear Regression with PCA: [105680.8261344]
```

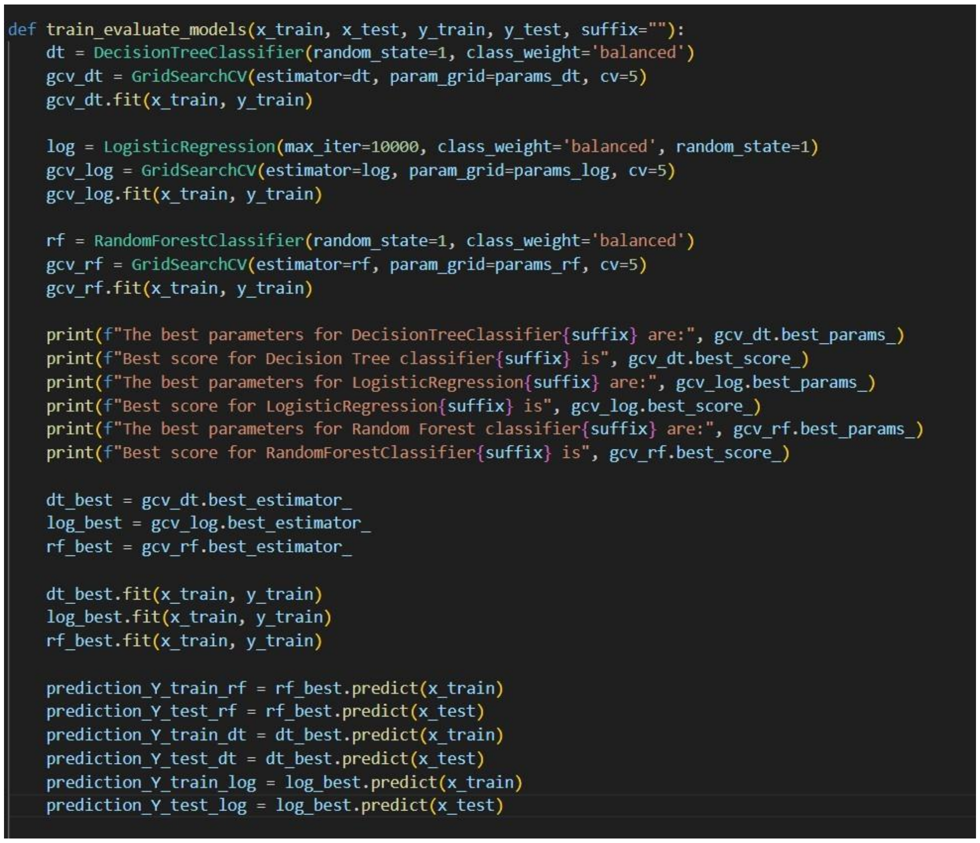
Decison Tree Before And After PCA :


Budget <= 37982.312
squared_error = 345609789.972
samples = 404
value = 45603.465
Trailer_views <= 440392.0
squared_error = 156210380.354
samples = 328
value = 39460.366
Budget <= 41312.535
squared_error = 297246592.798
samples = 76
value = 72115.789
Marketing expense <= 158.516
squared_error = 80127783.266
samples = 143
value = 30393.007



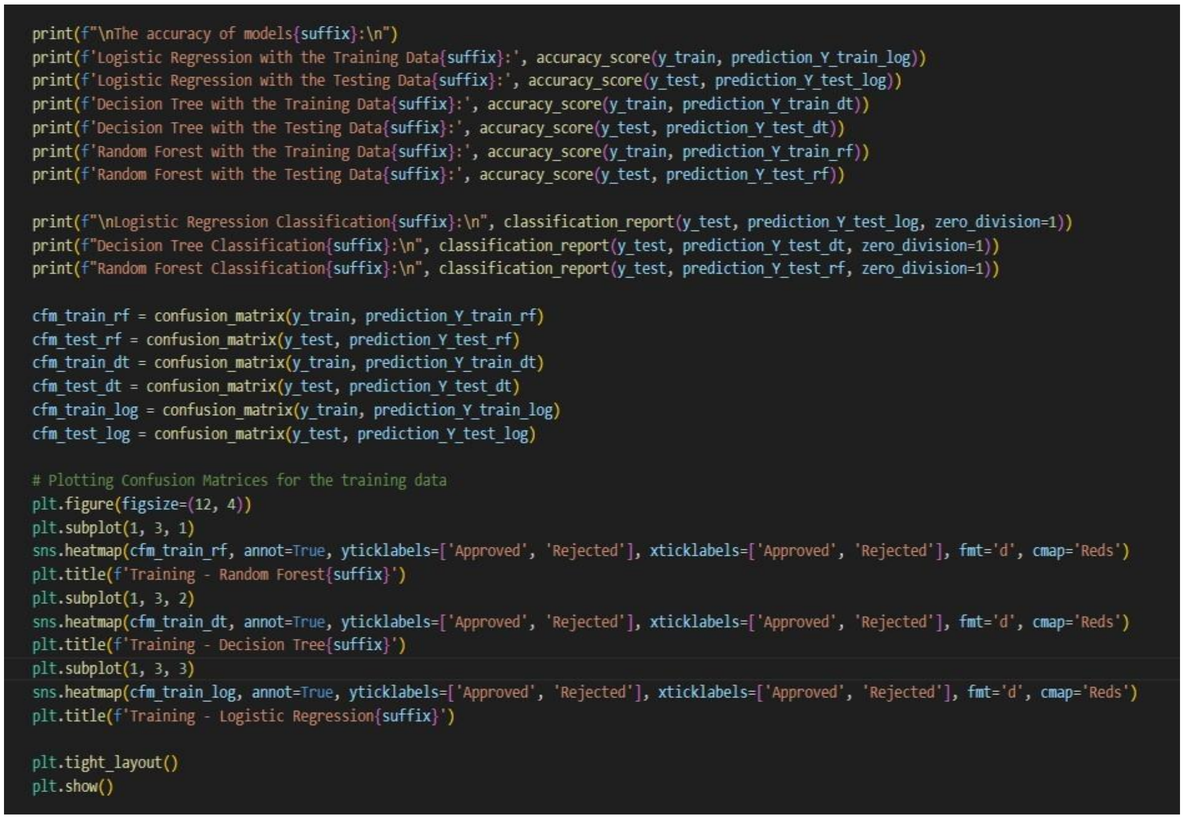
Loan Dataset Random Forest

Code :



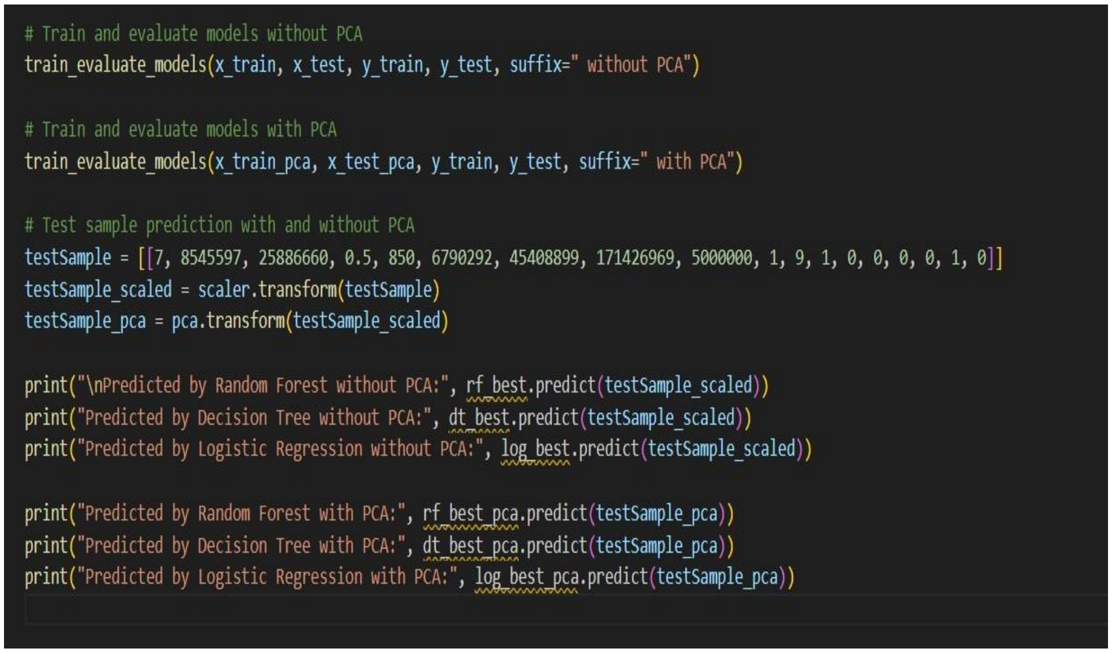
```python
def train_evaluate_models(x_train, x_test, y_train, y_test, suffix=""):
    dt = DecisionTreeClassifier(random_state=1, class_weight='balanced')
    gcv_dt = GridSearchCV(estimator=dt, param_grid=params_dt, cv=5)
    gcv_dt.fit(x_train, y_train)

    log = LogisticRegression(max_iter=10000, class_weight='balanced', random_state=1)
    gcv_log = GridSearchCV(estimator=log, param_grid=params_log, cv=5)
    gcv_log.fit(x_train, y_train)

    rf = RandomForestClassifier(random_state=1, class_weight='balanced')
    gcv_rf = GridSearchCV(estimator=rf, param_grid=params_rf, cv=5)
    gcv_rf.fit(x_train, y_train)

    print(f"The best parameters for DecisionTreeClassifier{suffix} are:", gcv_dt.best_params_)
    print(f"Best score for Decision Tree classifier{suffix} is", gcv_dt.best_score_)
    print(f"The best parameters for LogisticRegression{suffix} are:", gcv_log.best_params_)
    print(f"Best score for LogisticRegression{suffix} is", gcv_log.best_score_)
    print(f"The best parameters for Random Forest classifier{suffix} are:", gcv_rf.best_params_)
    print(f"Best score for RandomForestClassifier{suffix} is", gcv_rf.best_score_)

    dt_best = gcv_dt.best_estimator_
    log_best = gcv_log.best_estimator_
    rf_best = gcv_rf.best_estimator_

    dt_best.fit(x_train, y_train)
    log_best.fit(x_train, y_train)
    rf_best.fit(x_train, y_train)

    prediction_Y_train_rf = rf_best.predict(x_train)
    prediction_Y_test_rf = rf_best.predict(x_test)
    prediction_Y_train_dt = dt_best.predict(x_train)
    prediction_Y_test_dt = dt_best.predict(x_test)
    prediction_Y_train_log = log_best.predict(x_train)
    prediction_Y_test_log = log_best.predict(x_test)
```

Confusion Matrix :



```
print(f"\nThe accuracy of models{suffix}:\n")
print(f'Logistic Regression with the Training Data{suffix}:', accuracy_score(y_train, prediction_Y_train_log))
print(f'Logistic Regression with the Testing Data{suffix}:', accuracy_score(y_test, prediction_Y_test_log))
print(f'Decision Tree with the Training Data{suffix}:', accuracy_score(y_train, prediction_Y_train_dt))
print(f'Decision Tree with the Testing Data{suffix}:', accuracy_score(y_test, prediction_Y_test_dt))
print(f'Random Forest with the Training Data{suffix}:', accuracy_score(y_train, prediction_Y_train_rf))
print(f'Random Forest with the Testing Data{suffix}:', accuracy_score(y_test, prediction_Y_test_rf))

print(f"\nLogistic Regression Classification{suffix}:\n", classification_report(y_test, prediction_Y_test_log, zero_division=1))
print(f"Decision Tree Classification{suffix}:\n", classification_report(y_test, prediction_Y_test_dt, zero_division=1))
print(f"Random Forest Classification{suffix}:\n", classification_report(y_test, prediction_Y_test_rf, zero_division=1))

cfm_train_rf = confusion_matrix(y_train, prediction_Y_train_rf)
cfm_test_rf = confusion_matrix(y_test, prediction_Y_test_rf)
cfm_train_dt = confusion_matrix(y_train, prediction_Y_train_dt)
cfm_test_dt = confusion_matrix(y_test, prediction_Y_test_dt)
cfm_train_log = confusion_matrix(y_train, prediction_Y_train_log)
cfm_test_log = confusion_matrix(y_test, prediction_Y_test_log)

# Plotting Confusion Matrices for the training data
plt.figure(figsize=(12, 4))
plt.subplot(1, 3, 1)
sns.heatmap(cfm_train_rf, annot=True, yticklabels=['Approved', 'Rejected'], xticklabels=['Approved', 'Rejected'], fmt='d', cmap='Reds')
plt.title(f'Training - Random Forest{suffix}')
plt.subplot(1, 3, 2)
sns.heatmap(cfm_train_dt, annot=True, yticklabels=['Approved', 'Rejected'], xticklabels=['Approved', 'Rejected'], fmt='d', cmap='Reds')
plt.title(f'Training - Decision Tree{suffix}')
plt.subplot(1, 3, 3)
sns.heatmap(cfm_train_log, annot=True, yticklabels=['Approved', 'Rejected'], xticklabels=['Approved', 'Rejected'], fmt='d', cmap='Reds')
plt.title(f'Training - Logistic Regression{suffix}')

plt.tight_layout()
plt.show()
```

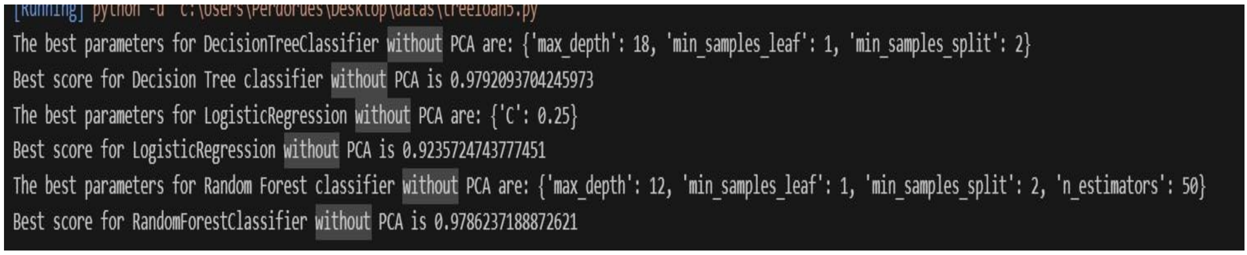
Evaluating Training and Testing Data With and Without PCA :



```
# Train and evaluate models without PCA
train_evaluate_models(x_train, x_test, y_train, y_test, suffix=" without PCA")

# Train and evaluate models with PCA
train_evaluate_models(x_train_pca, x_test_pca, y_train, y_test, suffix=" with PCA")

# Test sample prediction with and without PCA
testSample = [[7, 8545597, 25886660, 0.5, 850, 6790292, 45408899, 171426969, 5000000, 1, 9, 1, 0, 0, 0, 0, 1, 0]]
testSample_scaled = scaler.transform(testSample)
testSample_pca = pca.transform(testSample_scaled)

print("\nPredicted by Random Forest without PCA:", rf_best.predict(testSample_scaled))
print("Predicted by Decision Tree without PCA:", dt_best.predict(testSample_scaled))
print("Predicted by Logistic Regression without PCA:", log_best.predict(testSample_scaled))

print("Predicted by Random Forest with PCA:", rf_best_pca.predict(testSample_pca))
print("Predicted by Decision Tree with PCA:", dt_best_pca.predict(testSample_pca))
print("Predicted by Logistic Regression with PCA:", log_best_pca.predict(testSample_pca))
```

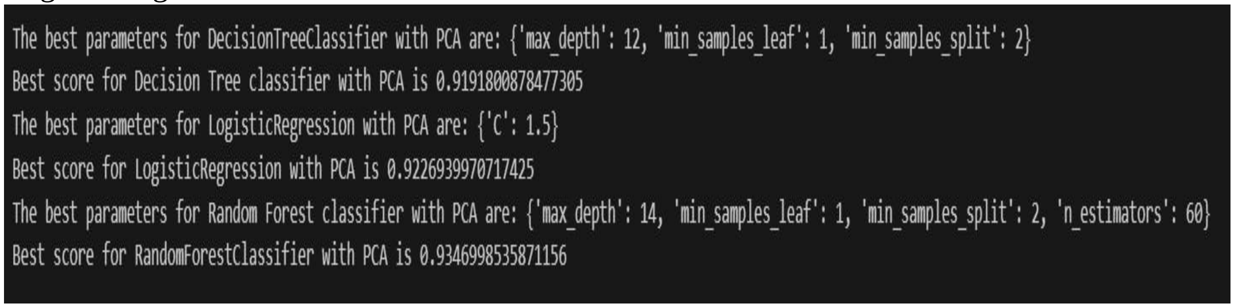
- Parameters
- For decision tree's parameters, PCA has decreased the depth of the tree wh The

classifier for the decision tree has decreased which Means the decision tree works better without PCA.

- 

```
[Running] python -u "C:\Users\Perdorues\Desktop\datas\treeloans.py"
The best parameters for DecisionTreeClassifier without PCA are: {'max_depth': 18, 'min_samples_leaf': 1, 'min_samples_split': 2}
Best score for Decision Tree classifier without PCA is 0.9792093704245973
The best parameters for LogisticRegression without PCA are: {'C': 0.25}
Best score for LogisticRegression without PCA is 0.9235724743777451
The best parameters for Random Forest classifier without PCA are: {'max_depth': 12, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 50}
Best score for RandomForestClassifier without PCA is 0.9786237188872621
```

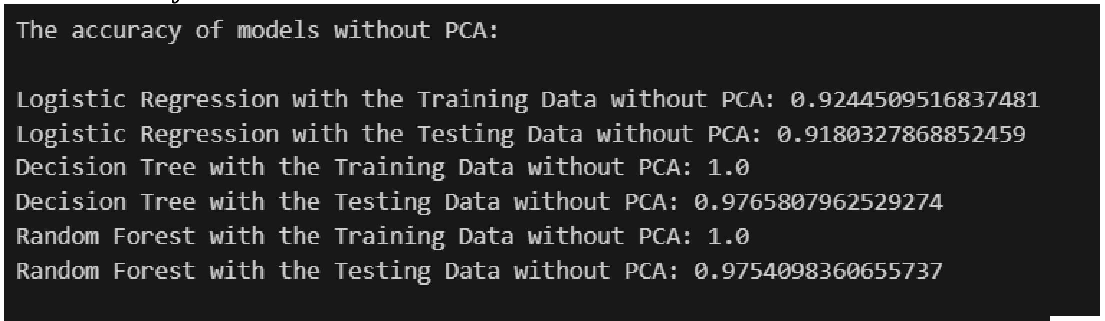
- For the logistic regression, PCA has increased the value of C from 0.75 to 1,5
  The regularization making the model better for lower dimensions. The value for Logistic Regression has also decreased.

- 

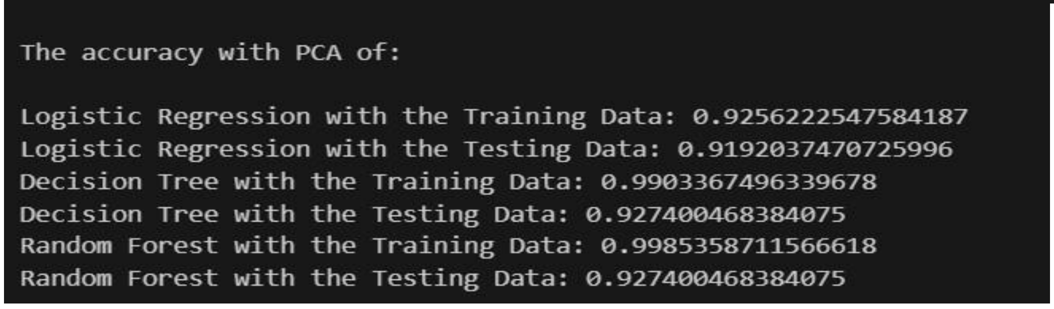
```
The best parameters for DecisionTreeClassifier with PCA are: {'max_depth': 12, 'min_samples_leaf': 1, 'min_samples_split': 2}
Best score for Decision Tree classifier with PCA is 0.9191800878477305
The best parameters for LogisticRegression with PCA are: {'C': 1.5}
Best score for LogisticRegression with PCA is 0.9226939970717425
The best parameters for Random Forest classifier with PCA are: {'max_depth': 14, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 60}
Best score for RandomForestClassifier with PCA is 0.9346998535871156
```

- The best score for random forest has decreased.

The accuracy of models



```
The accuracy of models without PCA:

Logistic Regression with the Training Data without PCA: 0.9244509516837481
Logistic Regression with the Testing Data without PCA: 0.9180327868852459
Decision Tree with the Training Data without PCA: 1.0
Decision Tree with the Testing Data without PCA: 0.9765807962529274
Random Forest with the Training Data without PCA: 1.0
Random Forest with the Testing Data without PCA: 0.9754098360655737
```



```
The accuracy with PCA of:

Logistic Regression with the Training Data: 0.9256222547584187
Logistic Regression with the Testing Data: 0.9192037470725996
Decision Tree with the Training Data: 0.9903367496339678
Decision Tree with the Testing Data: 0.927400468384075
Random Forest with the Training Data: 0.9985358711566618
Random Forest with the Testing Data: 0.927400468384075
```

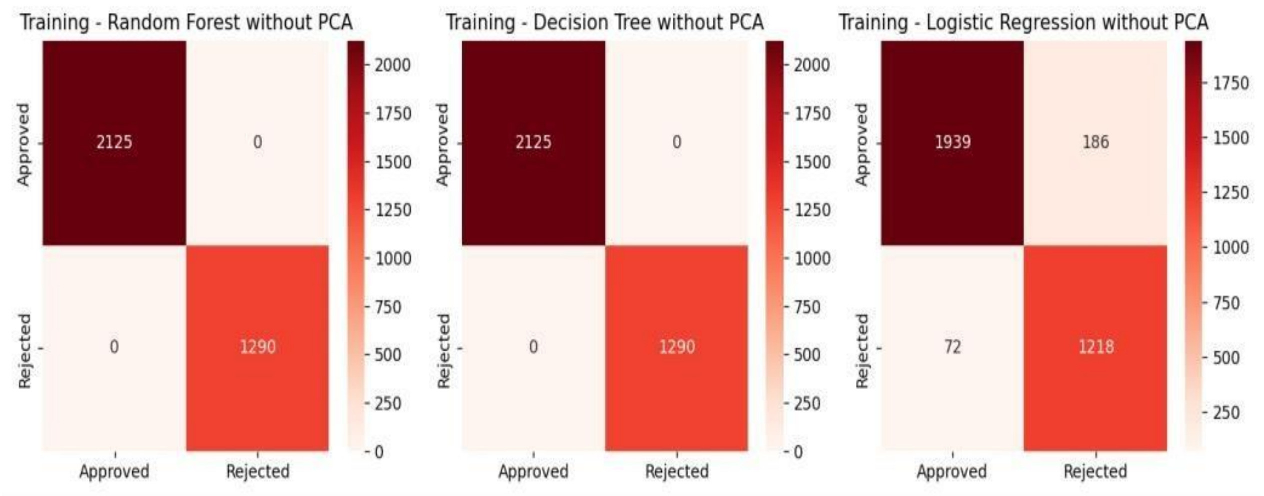
The accuracy of the Logistic Regression on training and testing data has increased with the PCA

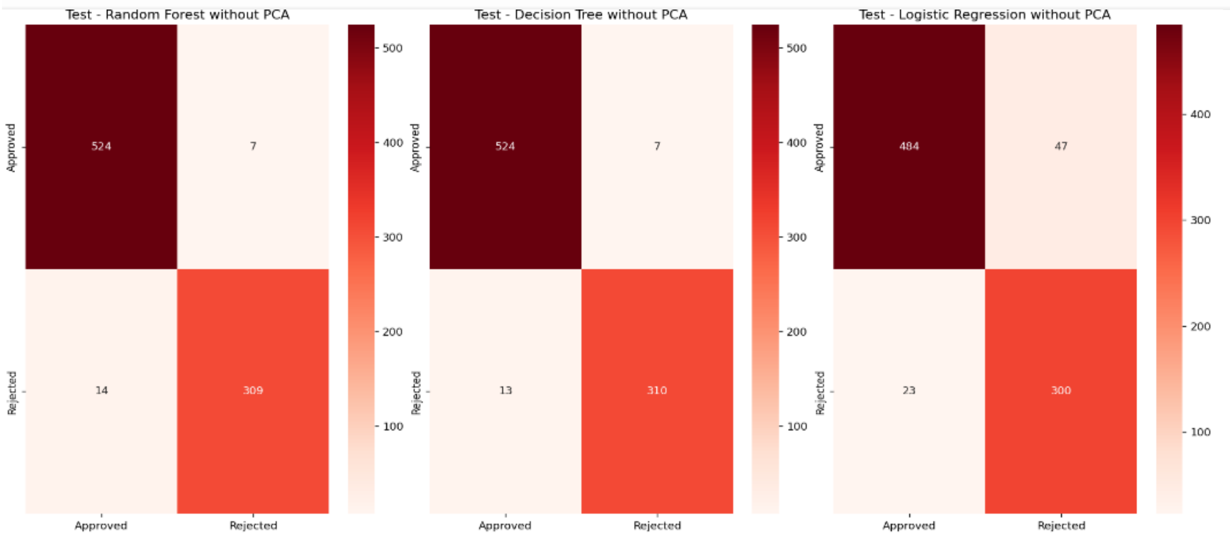
For the decision tree, the accuracy of training and testing data have decreased.

The accuracy of random forest for both training and testing data have decreased.

Classification Matrixes :

- Without PCA





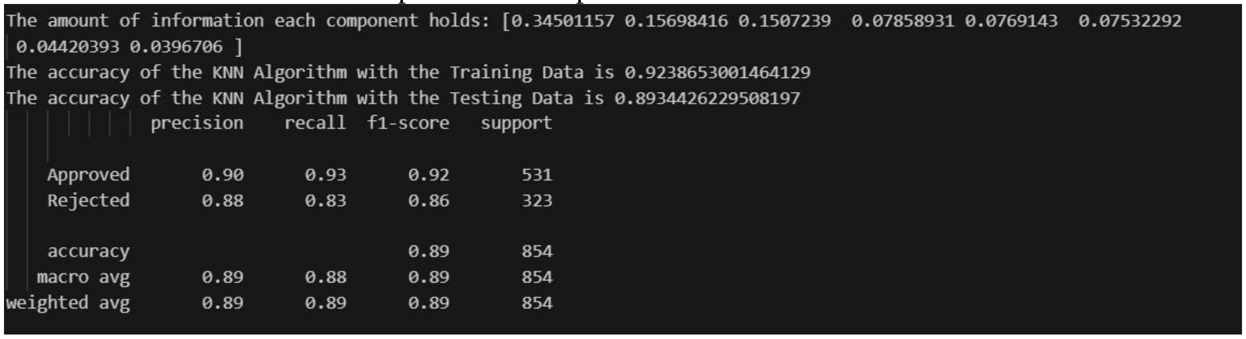
- With PCA ABOVE

Experiment Outline 1 and Outline 2 with different number of extracted components to ascertain whether your result improves or deteriorates with increasing number of components.

Outline 1:

KNN classification with Loan dataset.

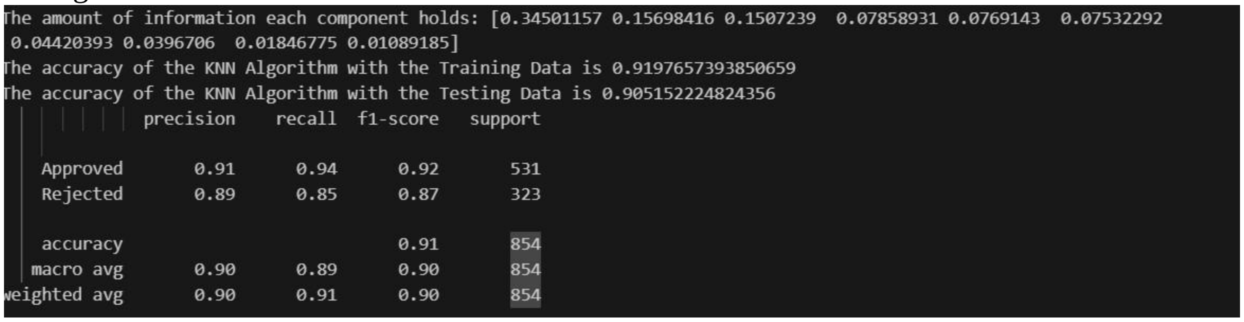
The results with 8 components from previous outline:



```
The amount of information each component holds: [0.34501157 0.15698416 0.1507239  0.07858931 0.0769143  0.07532292
 0.04420393 0.0396706 ]
The accuracy of the KNN Algorithm with the Training Data is 0.9238653001464129
The accuracy of the KNN Algorithm with the Testing Data is 0.8934426229508197
              precision    recall  f1-score   support

    Approved       0.90      0.93      0.92       531
    Rejected       0.88      0.83      0.86       323

    accuracy                           0.89       854
   macro avg       0.89      0.88      0.89       854
weighted avg       0.89      0.89      0.89       854
```

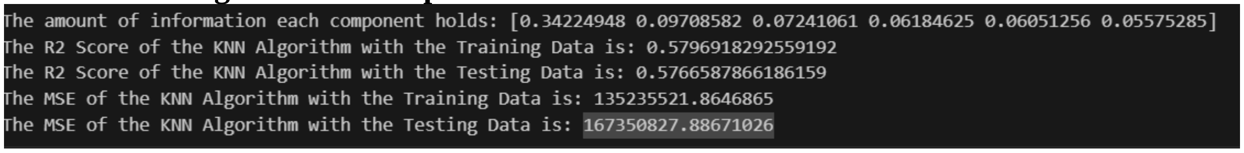
With 12 components

The added 4 components . For the training data the accuracy has decreased whilst for the testing data it has increaseD



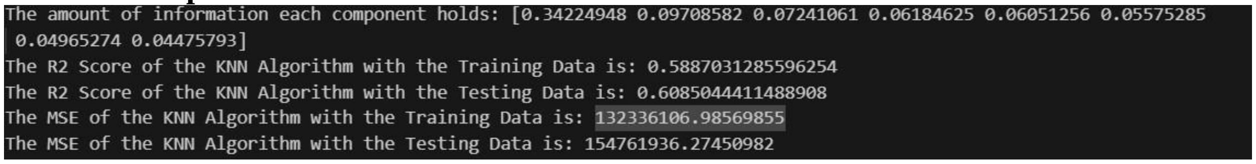
```
The amount of information each component holds: [0.34501157 0.15698416 0.1507239  0.07858931 0.0769143  0.07532292
 0.04420393 0.0396706  0.01846775 0.01089185]
The accuracy of the KNN Algorithm with the Training Data is 0.9197657393850659
The accuracy of the KNN Algorithm with the Testing Data is 0.905152224824356
              precision    recall  f1-score   support

    Approved       0.91      0.94      0.92       531
    Rejected       0.89      0.85      0.87       323

    accuracy                           0.91       854
   macro avg       0.90      0.89      0.90       854
weighted avg       0.90      0.91      0.90       854
```

Film Regression 6 components:



```
The amount of information each component holds: [0.34224948 0.09708582 0.07241061 0.06184625 0.06051256 0.05575285]
The R2 Score of the KNN Algorithm with the Training Data is: 0.5796918292559192
The R2 Score of the KNN Algorithm with the Testing Data is: 0.5766587866186159
The MSE of the KNN Algorithm with the Training Data is: 135235521.8646865
The MSE of the KNN Algorithm with the Testing Data is: 167350827.88671026
```

8 components:



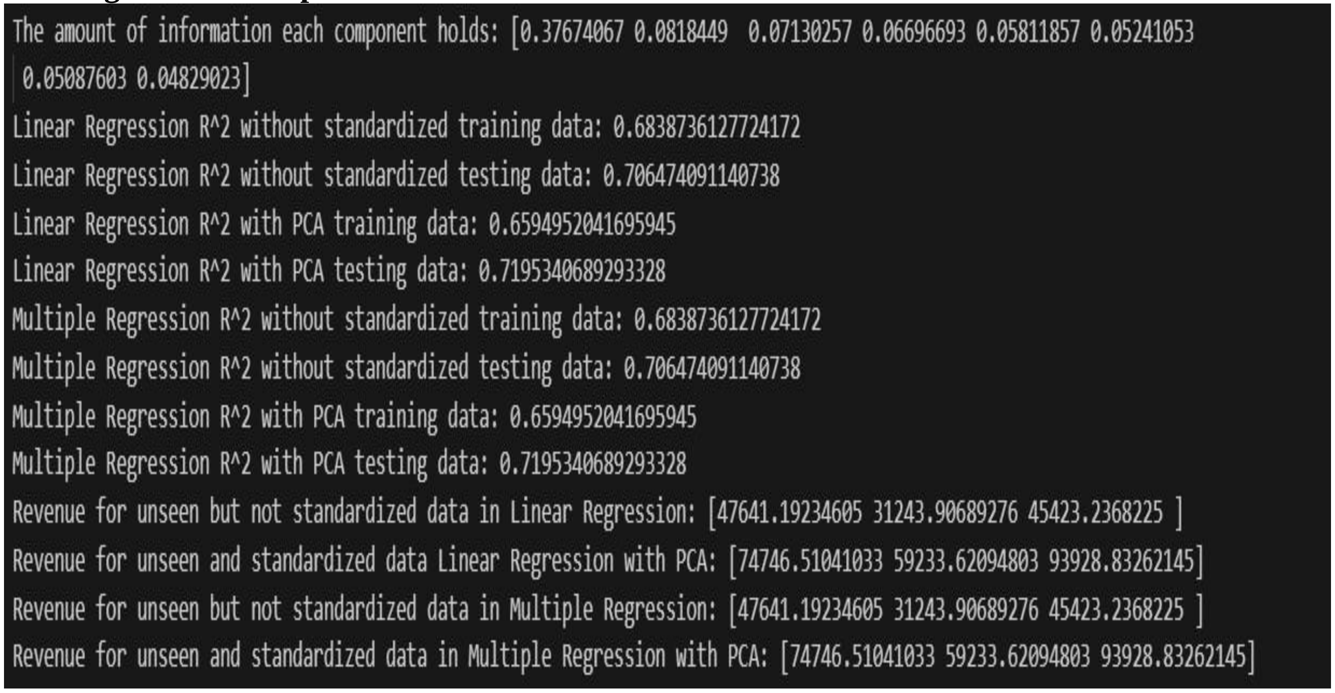
```
The amount of information each component holds: [0.34224948 0.09708582 0.07241061 0.06184625 0.06051256 0.05575285
 0.04965274 0.04475793]
The R2 Score of the KNN Algorithm with the Training Data is: 0.5887031285596254
The R2 Score of the KNN Algorithm with the Testing Data is: 0.6085044411488908
The MSE of the KNN Algorithm with the Training Data is: 132336106.98569855
The MSE of the KNN Algorithm with the Testing Data is: 154761936.27450982
```

The 2 added component4.9% and 4.4% of the variances.

The R scores for training and testing sets have increased, the MSE for both have decreased.

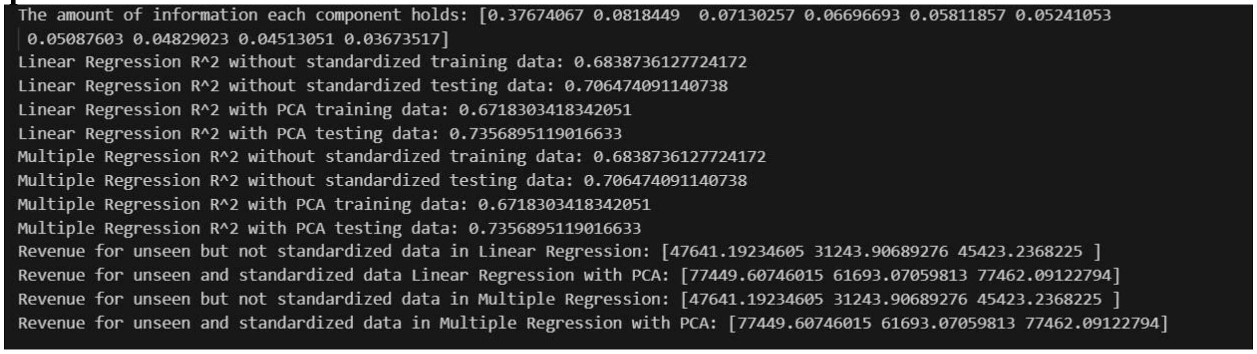
Outline 2

Film regresion 8 components :



```
The amount of information each component holds: [0.37674067 0.0818449  0.07130257 0.06696693 0.05811857 0.05241053
 0.05087603 0.04829023]
Linear Regression R^2 without standardized training data: 0.6838736127724172
Linear Regression R^2 without standardized testing data: 0.706474091140738
Linear Regression R^2 with PCA training data: 0.6594952041695945
Linear Regression R^2 with PCA testing data: 0.7195340689293328
Multiple Regression R^2 without standardized training data: 0.6838736127724172
Multiple Regression R^2 without standardized testing data: 0.706474091140738
Multiple Regression R^2 with PCA training data: 0.6594952041695945
Multiple Regression R^2 with PCA testing data: 0.7195340689293328
Revenue for unseen but not standardized data in Linear Regression: [47641.19234605 31243.90689276 45423.2368225 ]
Revenue for unseen and standardized data Linear Regression with PCA: [74746.51041033 59233.62094803 93928.83262145]
Revenue for unseen but not standardized data in Multiple Regression: [47641.19234605 31243.90689276 45423.2368225 ]
Revenue for unseen and standardized data in Multiple Regression with PCA: [74746.51041033 59233.62094803 93928.83262145]
```

10 Components :



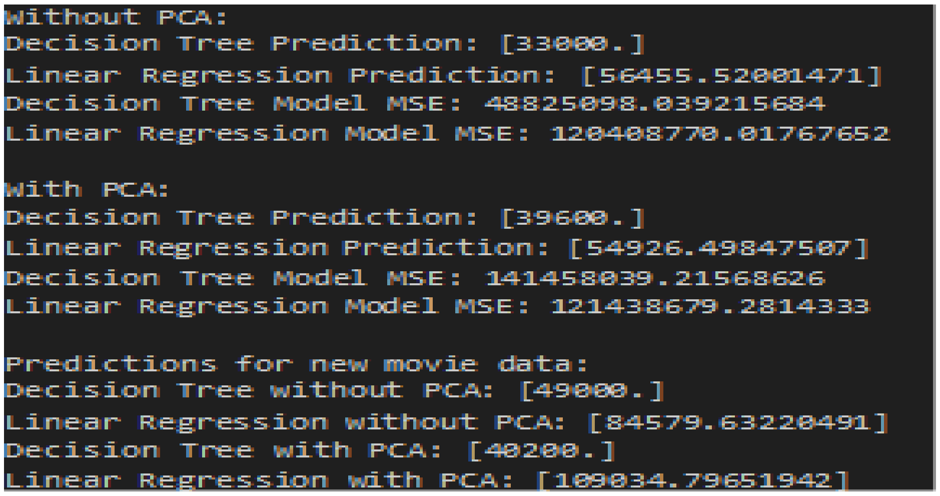
```
The amount of information each component holds: [0.37674067 0.0818449  0.07130257 0.06696693 0.05811857 0.05241053
 0.05087603 0.04829023 0.04513051 0.03673517]
Linear Regression R^2 without standardized training data: 0.6838736127724172
Linear Regression R^2 without standardized testing data: 0.706474091140738
Linear Regression R^2 with PCA training data: 0.6718303418342051
Linear Regression R^2 with PCA testing data: 0.7356895119016633
Multiple Regression R^2 without standardized training data: 0.6838736127724172
Multiple Regression R^2 without standardized testing data: 0.706474091140738
Multiple Regression R^2 with PCA training data: 0.6718303418342051
Multiple Regression R^2 with PCA testing data: 0.7356895119016633
Revenue for unseen but not standardized data in Linear Regression: [47641.19234605 31243.90689276 45423.2368225 ]
Revenue for unseen and standardized data Linear Regression with PCA: [77449.60746015 61693.07059813 77462.09122794]
Revenue for unseen but not standardized data in Multiple Regression: [47641.19234605 31243.90689276 45423.2368225 ]
Revenue for unseen and standardized data in Multiple Regression with PCA: [77449.60746015 61693.07059813 77462.09122794]
```

The total variance 4.5% and 3.6%.

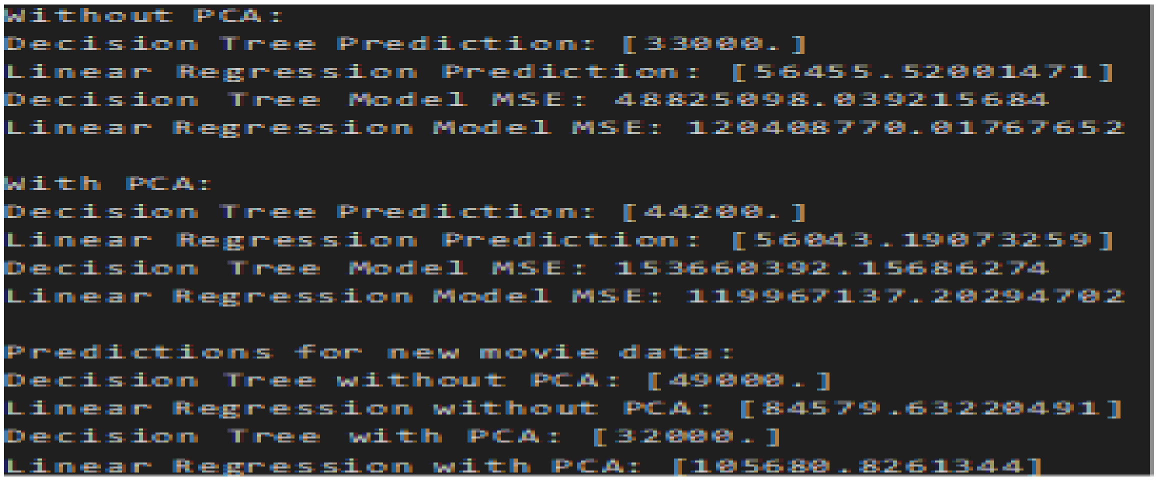
The linear Regression and multiple with PCA for training and testing have increased. The predicted revenue has also increased.

Film Dataset-Decision Tree

10 components



```
Without PCA:
Decision Tree Prediction: [33000.]
Linear Regression Prediction: [56455.52001471]
Decision Tree Model MSE: 48825098.039215684
Linear Regression Model MSE: 120408770.01767652

With PCA:
Decision Tree Prediction: [39600.]
Linear Regression Prediction: [54926.49847507]
Decision Tree Model MSE: 141458039.21568626
Linear Regression Model MSE: 121438679.2814333

Predictions for new movie data:
Decision Tree without PCA: [49000.]
Linear Regression without PCA: [84579.63220491]
Decision Tree with PCA: [40200.]
Linear Regression with PCA: [109034.79651942]
```

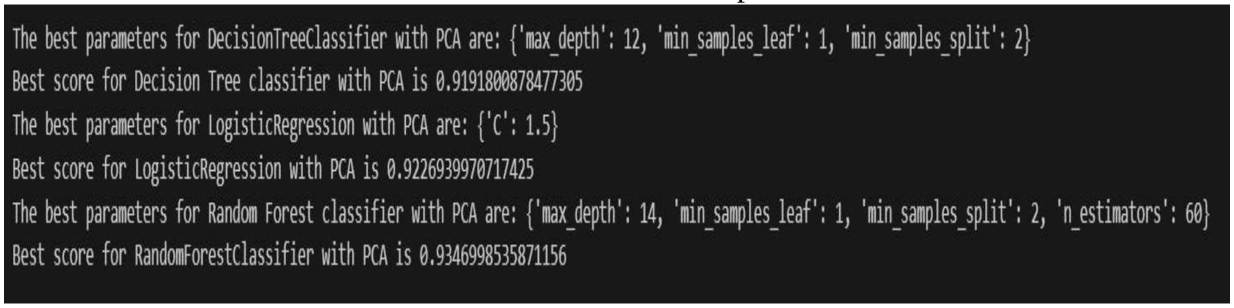


```
Without PCA:
Decision Tree Prediction: [33000.]
Linear Regression Prediction: [56455.52001471]
Decision Tree Model MSE: 48825098.039215684
Linear Regression Model MSE: 120408770.01767652

With PCA:
Decision Tree Prediction: [44200.]
Linear Regression Prediction: [56043.19073259]
Decision Tree Model MSE: 153660392.15686274
Linear Regression Model MSE: 119967137.20294702

Predictions for new movie data:
Decision Tree without PCA: [49000.]
Linear Regression without PCA: [84579.63220491]
Decision Tree with PCA: [32000.]
Linear Regression with PCA: [105680.8261344]
```

With 8 components

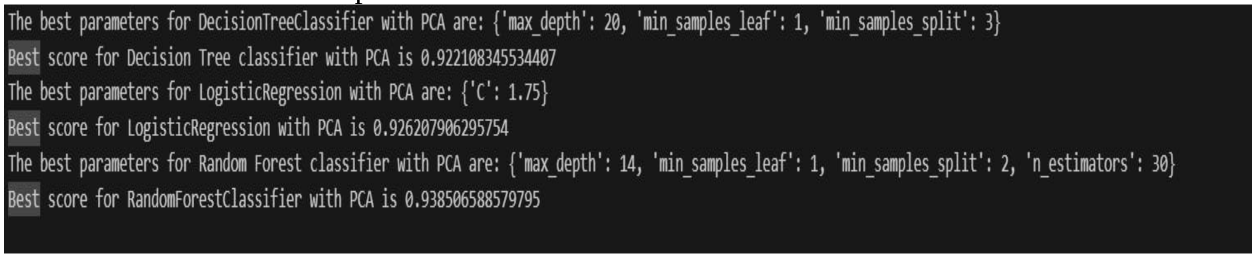
PCA with 10 components is better as it has increased its prediction value to 44200 and lowered the MSE . The same has happened for the regression's prediction and MSE. Whilst for unseen data, the decision tree and regression's predictions have increased.

Random Forrest Loan Dataset 8 and 10 components



```
The best parameters for DecisionTreeClassifier with PCA are: {'max_depth': 12, 'min_samples_leaf': 1, 'min_samples_split': 2}
Best score for Decision Tree classifier with PCA is 0.9191800878477305
The best parameters for LogisticRegression with PCA are: {'C': 1.5}
Best score for LogisticRegression with PCA is 0.9226939970717425
The best parameters for Random Forest classifier with PCA are: {'max_depth': 14, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 60}
Best score for RandomForestClassifier with PCA is 0.9346998535871156
```

Result with 10 components

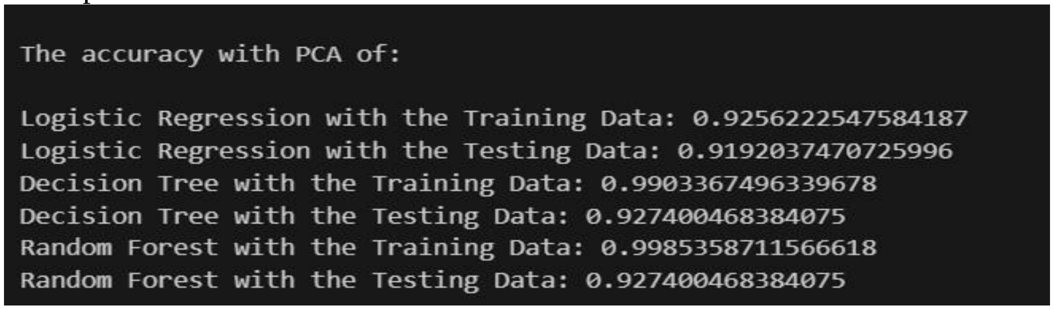


```
The best parameters for DecisionTreeClassifier with PCA are: {'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 3}
Best score for Decision Tree classifier with PCA is 0.922108345534407
The best parameters for LogisticRegression with PCA are: {'C': 1.75}
Best score for LogisticRegression with PCA is 0.926207906295754
The best parameters for Random Forest classifier with PCA are: {'max_depth': 14, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 30}
Best score for RandomForestClassifier with PCA is 0.938506588579795
```

The increase of components increased the depth of decision tree from 11 - 20The coefficient C for Logistic Regression has increased and therefore the regularization has decreased. From the random forest ‘s parameters, only the nr of estimators has decreased .while the best score for it has increased.

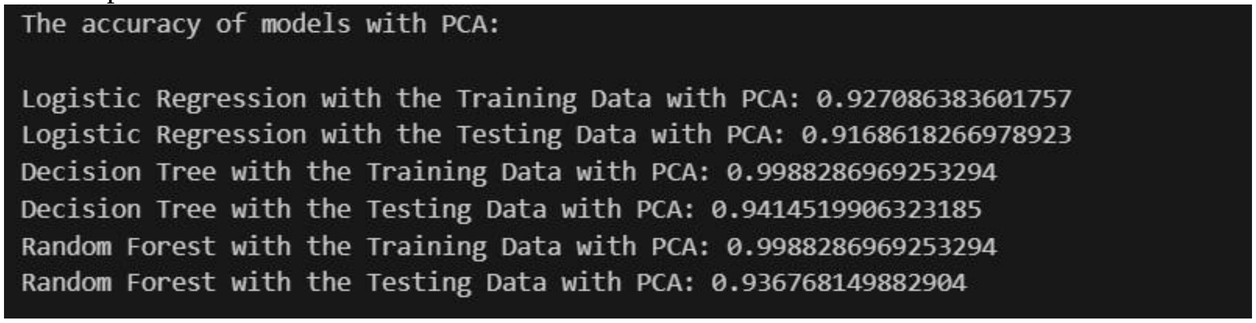
- AS FOR **Accuracy**

8 components:



```
The accuracy with PCA of:

Logistic Regression with the Training Data: 0.9256222547584187
Logistic Regression with the Testing Data: 0.9192037470725996
Decision Tree with the Training Data: 0.9903367496339678
Decision Tree with the Testing Data: 0.927400468384075
Random Forest with the Training Data: 0.9985358711566618
Random Forest with the Testing Data: 0.927400468384075
```

10 components:



```
The accuracy of models with PCA:

Logistic Regression with the Training Data with PCA: 0.927086383601757
Logistic Regression with the Testing Data with PCA: 0.9168618266978923
Decision Tree with the Training Data with PCA: 0.9988286969253294
Decision Tree with the Testing Data with PCA: 0.9414519906323185
Random Forest with the Training Data with PCA: 0.9988286969253294
Random Forest with the Testing Data with PCA: 0.936768149882904
```

All accuracies have grown, with the exception of the testing data's logistic regression, which has declined.

Even though the number of components has decreased in the logistic regression test, the model has improved.