The University of Faisalabad

Title: Lab Manual

Course Code: AI-414

Course Title: Machine Learning

Instructor Name: Mr. Saeed

Student Name: Mishal Nadeem

Registration Number: 2023-BS-AI-020

Department: Computer Science

Academic Year: 2023-2027

Contents

Machine Learning

Machine Learning is a field of artificial intelligence that enables computers to learn from data and make predictions or decisions without being explicitly programmed. It is widely used in areas like recommendation systems, fraud detection, and image recognition. ML techniques are mainly classified into supervised, unsupervised, and reinforcement learning. By analyzing patterns in data, machine learning helps automate tasks and solve real-world problems efficiently.

Linear Regression

Linear Regression is a supervised learning algorithm used to model the relationship between a dependent variable and one or more independent variables. It assumes a linear relationship, where the output is predicted as a weighted sum of inputs. The goal is to minimize the difference between actual and predicted values using techniques like least squares. Linear regression is commonly used in forecasting, trend analysis, and estimating continuous outcomes such as price or sales.

Classification

Classification is a supervised machine learning technique used to categorize data into predefined classes or labels. Unlike regression, which predicts continuous values, classification deals with discrete outputs (e.g., yes/no, spam/ham, approved/rejected). Popular classification algorithms include Decision Trees, Support Vector Machines, Random Forest, and Neural Networks. It is widely applied in tasks such as email filtering, disease detection, and loan approval systems.

Customer Lifetime Value Prediction using Linear Regression

Summary

This project focuses on predicting Customer Lifetime Value (CLV) through a linear regression model. The process begins with thorough data preprocessing, including handling outliers, standardizing numerical features, and applying one-hot encoding for categorical variables. After selecting relevant features and splitting the data, a linear regression algorithm is implemented. The model's performance is evaluated using various error metrics and visualization techniques to assess accuracy and reliability. The study also explores the influence of individual features on CLV predictions, demonstrating the practical applications and limitations of linear regression in customer analytics.

Abstract

This project aims to develop a predictive model for Customer Lifetime Value (CLV) using linear regression. The dataset undergoes comprehensive preprocessing including cleansing, transformation, and encoding. After feature selection and data splitting, a linear regression model is trained and evaluated. Model performance is assessed using statistical metrics and visual tools to understand the accuracy and reliability of predictions. The project highlights both the strengths and limitations of linear regression in real-world customer data analysis.

Objectives

- To predict Customer Lifetime Value using linear regression.
- To perform essential data preprocessing including outlier detection, standardization, and one-hot encoding.
- To evaluate the model's performance using error metrics and visualizations.
- To interpret feature importance in predicting customer value.

Dataset Description

- **Number of rows:** 1,000
- **Number of columns:** 13
- **Purpose:** This dataset is designed to analyze customer behavior, spending patterns, and factors influencing Customer Lifetime Value (CLV).

Column Description

1. **CustomerID** *(object)*
   Unique identifier for each customer (e.g., CUST100000).
2. **Gender** *(object)*
   Gender of the customer. Values include: Male, Female.
3. **TotalSpend** *(float64)*
   Total amount spent by the customer over a period.
4. **AverageOrderValue** *(float64)*
   Average value of an order placed by the customer.
5. **PurchaseFrequency** *(float64, 1 missing value)*
   Number of purchases made by the customer in the given period. One missing value is present.
6. **IsPremiumMember** *(int64)*
   Indicator of premium membership:
   - 1 = Premium member
   - 0 = Non-premium member
7. **Region** *(object)*
   Geographic region of the customer. Values include: Urban, Suburban, Rural.
8. **CustomerSatisfaction** *(float64)*
   Satisfaction score provided by the customer (typically a scale from 1 to 5).
9. **CLV (Customer Lifetime Value)** *(float64)*
   Predicted total value a customer will bring over their entire relationship with the business.
10. **ValuePerOrder** *(float64)*
    Calculated metric: possibly CLV / PurchaseFrequency or similar.
11. **HighValueCustomer** *(int64)*
    Binary indicator of whether a customer is classified as high-value:
    - 1 = High value
    - 0 = Not high value
12. **SatisfactionSpendScore** *(float64)*
    Composite score derived from customer satisfaction and spending.
13. **OrderValueScore** *(float64)*
    Composite score combining metrics related to order value and possibly order frequency.

Explanation of Steps

Step 1: Importing Libraries

Import essential libraries like pandas, numpy, matplotlib, seaborn, and sklearn for data handling, visualization, and machine learning.

Step 2: Reading Data

Load the dataset (usually in CSV format) using pandas.read_csv().

Step 3: Exploring Data

Use head(), info(), and describe() to understand data types, shape, and basic statistics.

Step 4: Cleansing Data

Handle missing values, fix data types, and remove duplicates to prepare clean data for modeling.

Step 5: Outlier Detection and Removal

Identify outliers using methods like IQR or z-score and remove them to prevent skewing the model.

Step 6: Separate Target Column

Split the dataset into features (X) and target (y, i.e., customer lifetime value).

Step 7: Data Transformation (Standardization)

Use StandardScaler to scale numerical features so all variables contribute equally to the model.

Step 8: Categorical into Numerical (One-Hot Encoding)

Convert categorical features into binary columns using pd.get_dummies().

Step 9: Feature Selection

Choose the most relevant features using correlation or feature importance to improve model accuracy.

Step 10: Data Splitting

Divide data into training and testing sets using train_test_split() to evaluate model performance.

Step 11: Train the Model

Use LinearRegression().fit(X_train, y_train) to train the model on the training data.

Step 12: Make Predictions

Predict on the test set using model.predict(X_test).

Step 13: Evaluate the Model

Use metrics like RMSE, MAE, and R^2 score to assess prediction accuracy.

Step 14: Feature Importance

Check which features influenced the model most by viewing coefficients from the linear regression model.

Step 15: Model Performance Visualization

Plot predicted vs actual values or residuals to visualize how well the model performs.

Step 16: Assessing Error Patterns in Predictions

Analyze residual plots to detect biases or trends in prediction errors.

Step 17: Training vs Cross-Validation Performance

Compare training score and cross-validation score to check for overfitting or underfitting.

Step 1

Importing Libraries

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats
from sklearn.model_selection import learning_curve

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

from sklearn.feature_selection import RFE
```

Step 2

Reading Data

```python
df = pd.read_csv('clv.csv')
```

Step 3

Exploring Data

```python
df.head()
```

	CustomerID	Gender	TotalSpend	AverageOrderValue	PurchaseFrequency	IsPremiumMember	Region	CustomerSatisfaction	CLV	ValuePerOrder
0	CUST100000	Male	10072.94	107.82	9.96	1	Urban	1.54	7596.87	919.063869
1	CUST100001	Male	8003.86	472.38	NaN	1	Suburban	4.29	8185.19	889.317778
2	CUST100002	Male	10840.68	144.50	7.59	1	Urban	4.69	7945.16	1262.011641
3	CUST100003	Female	12596.21	129.08	2.31	0	Rural	4.16	5922.92	3805.501511
4	CUST100004	Male	13518.65	1357.45	3.49	0	Suburban	1.80	11120.52	3010.835189

```python
df.tail()
```

	CustomerID	Gender	TotalSpend	AverageOrderValue	PurchaseFrequency	IsPremiumMember	Region	CustomerSatisfaction	CLV	ValuePerOrder
995	CUST100995	Female	12989.49	776.74	4.88	1	Urban	1.99	10554.57	2209.096939
996	CUST100996	Male	15945.19	247.21	7.97	1	Urban	2.58	9586.66	1777.613155
997	CUST100997	Female	15105.06	480.22	9.82	0	Suburban	3.73	8243.55	1396.031423
998	CUST100998	Male	815.18	9.48	9.34	1	Rural	4.57	5149.51	78.837524
999	CUST100999	Male	7955.21	85.36	9.35	0	Urban	2.26	5690.09	768.619324

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 13 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   CustomerID             1000 non-null   object
 1   Gender                 1000 non-null   object
 2   TotalSpend             1000 non-null   float64
 3   AverageOrderValue      1000 non-null   float64
 4   PurchaseFrequency      999 non-null    float64
 5   IsPremiumMember        1000 non-null   int64
 6   Region                 1000 non-null   object
 7   CustomerSatisfaction   1000 non-null   float64
 8   CLV                    1000 non-null   float64
 9   ValuePerOrder          1000 non-null   float64
 10  HighValueCustomer      1000 non-null   int64
 11  SatisfactionSpendScore 1000 non-null   float64
 12  OrderValueScore        1000 non-null   float64
dtypes: float64(8), int64(2), object(3)
memory usage: 101.7+ KB
```

```
df.describe()
```

	TotalSpend	AverageOrderValue	PurchaseFrequency	IsPremiumMember	CustomerSatisfaction	CLV	ValuePerOrder	HighValueCustomer
count	1000.00000	1000.000000	999.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000
mean	9966.01330	300.982190	4.907588	0.407000	3.002310	6820.164570	2291.902209	0.108000
std	5806.46455	395.747978	2.853593	0.491521	1.144965	3424.895121	2187.735695	0.310536
min	103.75000	1.300000	0.200000	0.000000	1.000000	-19.560000	9.474886	0.000000
25%	4729.18000	89.845000	2.415000	0.000000	1.997500	4417.677500	875.482174	0.000000
50%	10029.15000	185.650000	4.750000	0.000000	3.040000	6501.510000	1689.828250	0.000000
75%	15012.05250	337.320000	7.390000	1.000000	3.962500	8796.480000	2972.268123	0.000000
max	19994.30000	3108.920000	9.980000	1.000000	5.000000	23596.780000	14282.380170	1.000000

Step 4

Cleansing Data

Before Cleansing Data

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

```
CustomerID                0
Gender                    0
TotalSpend                0
AverageOrderValue         0
PurchaseFrequency         1
IsPremiumMember           0
Region                    0
CustomerSatisfaction      0
CLV                       0
ValuePerOrder             0
HighValueCustomer         0
SatisfactionSpendScore    0
OrderValueScore           0
dtype: int64
```

After Cleansing Data

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

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

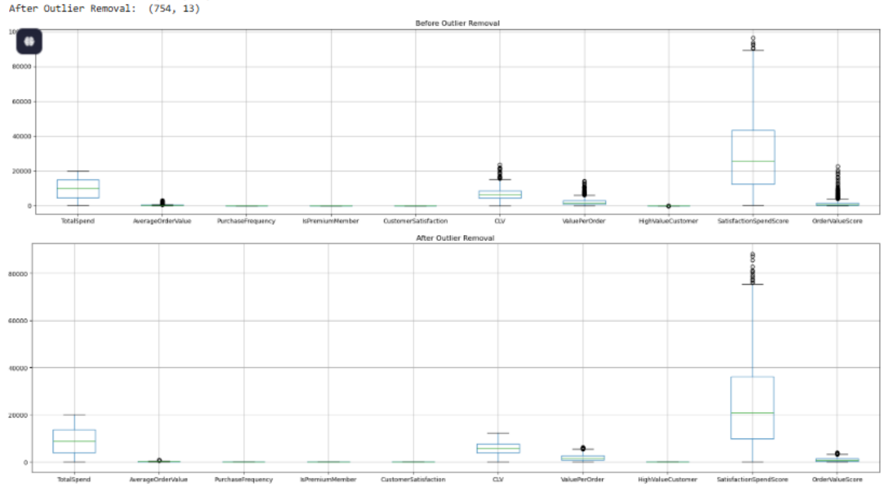
```
CustomerID                 0
Gender                     0
TotalSpend                 0
AverageOrderValue          0
PurchaseFrequency          0
IsPremiumMember            0
Region                     0
CustomerSatisfaction       0
CLV                        0
ValuePerOrder              0
HighValueCustomer          0
SatisfactionSpendScore     0
OrderValueScore            0
dtype: int64
```

Step 5

Outlier Detection and Removal

```
numeric_cols = df.select_dtypes(include=[np.number])
Q1 = numeric_cols.quantile(0.25)
Q3 = numeric_cols.quantile(0.75)
IQR = Q3 - Q1
data_cleaned = df[~((numeric_cols < (Q1 - 1.5 * IQR)) | (numeric_cols > (Q3 + 1.5 * IQR))).any(axis=1)]

print("Before Outlier Removal: ", df.shape)
print("After Outlier Removal: ", data_cleaned.shape)

plt.figure(figsize=(40, 5))
plt.subplot(1, 2, 1)
numeric_cols.boxplot()
plt.title("Before Outlier Removal")
plt.tight_layout()
plt.show()

plt.figure(figsize=(40, 6))
plt.subplot(1, 2, 2)
data_cleaned.select_dtypes(include=[np.number]).boxplot()
plt.title("After Outlier Removal")
plt.tight_layout()
plt.show()
```

```
Before Outlier Removal:  (1000, 13)
After Outlier Removal:  (754, 13)
```



Step 6

Separate Target column

```
target_column = 'CLV'
y = data_cleaned[target_column]
df = data_cleaned.drop(columns=[target_column, 'CustomerID'])
X = df
```

Step 7

Data Transformation (Standardization)

Before Standardization

```
df
```

	Gender	TotalSpend	AverageOrderValue	PurchaseFrequency	IsPremiumMember	Region	CustomerSatisfaction	ValuePerOrder	HighValueCustomer
0	Male	10072.94	107.82	9.96	1	Urban	1.54	919.063869	0
3	Female	12596.21	129.08	2.31	0	Rural	4.16	3805.501511	0
5	Male	10696.38	153.22	4.76	1	Urban	2.65	1857.010417	0
6	Male	14456.30	292.20	7.78	1	Urban	3.40	1646.503417	0
7	Female	2911.43	48.51	9.17	1	Urban	1.64	286.276303	0
...	...	...	...	...	...	...	...	...	...
992	Female	7486.12	109.04	4.14	0	Urban	4.55	1456.443580	0
993	Male	17408.29	351.74	5.86	0	Rural	3.66	2537.651603	0
994	Female	14636.88	186.49	3.22	1	Rural	3.59	3468.454976	0
996	Male	15945.19	247.21	7.97	1	Urban	2.58	1777.613155	0
999	Male	7955.21	85.36	9.35	0	Urban	2.26	768.619324	0

754 rows × 11 columns

After Standardization

```
scaler = StandardScaler()
numerical_cols = df.select_dtypes(include=np.number).columns
df[numerical_cols] = scaler.fit_transform(df[numerical_cols])
```

```
df
```

	Gender	TotalSpend	AverageOrderValue	PurchaseFrequency	IsPremiumMember	Region	CustomerSatisfaction	ValuePerOrder	HighValueCustomer
0	Male	0.182143	-0.535174	1.795233	1.419859	Urban	-1.191734	-0.651965	0.0
3	Female	0.629477	-0.390097	-0.987449	-0.704295	Rural	1.263701	1.392978	0.0
5	Male	0.292669	-0.225367	-0.096263	1.419859	Urban	-0.151454	0.012538	0.0
6	Male	0.959240	0.723026	1.002260	1.419859	Urban	0.551437	-0.136599	0.0
7	Female	-1.087473	-0.939903	1.507871	1.419859	Urban	-1.098015	-1.100273	0.0
...	...	...	...	...	...	...	...	...	...
992	Female	-0.276457	-0.526849	-0.321788	-0.704295	Urban	1.629204	-0.271250	0.0
993	Male	1.482579	1.129324	0.303861	-0.704295	Rural	0.795106	0.494749	0.0
994	Female	0.991254	0.001666	-0.656437	1.419859	Rural	0.729503	1.154192	0.0
996	Male	1.223195	0.416016	1.071372	1.419859	Urban	-0.217057	-0.043712	0.0
999	Male	-0.193295	-0.688440	1.573346	-0.704295	Urban	-0.516958	-0.758550	0.0

754 rows × 11 columns

Step 8

Categorical into Numerical (One-Hot Encoding)

Before One-Hot Encoding

```
df.shape
```

```
(754, 11)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 754 entries, 0 to 999
Data columns (total 11 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Gender                 754 non-null    object
 1   TotalSpend             754 non-null    float64
 2   AverageOrderValue      754 non-null    float64
 3   PurchaseFrequency      754 non-null    float64
 4   IsPremiumMember        754 non-null    float64
 5   Region                 754 non-null    object
 6   CustomerSatisfaction   754 non-null    float64
 7   ValuePerOrder          754 non-null    float64
 8   HighValueCustomer      754 non-null    float64
 9   SatisfactionSpendScore 754 non-null    float64
 10  OrderValueScore        754 non-null    float64
dtypes: float64(9), object(2)
memory usage: 70.7+ KB
```

After One-Hot Encoding

```
cat_cols = df.select_dtypes(include=['object']).columns
df = pd.get_dummies(df, columns=cat_cols, drop_first=False)
```

```
df.shape
```

```
(754, 14)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 754 entries, 0 to 999
Data columns (total 14 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   TotalSpend             754 non-null    float64
 1   AverageOrderValue      754 non-null    float64
 2   PurchaseFrequency      754 non-null    float64
 3   IsPremiumMember        754 non-null    float64
 4   CustomerSatisfaction   754 non-null    float64
 5   ValuePerOrder          754 non-null    float64
 6   HighValueCustomer      754 non-null    float64
 7   SatisfactionSpendScore 754 non-null    float64
 8   OrderValueScore        754 non-null    float64
 9   Gender_Female          754 non-null    bool
 10  Gender_Male            754 non-null    bool
 11  Region_Rural           754 non-null    bool
 12  Region_Suburban        754 non-null    bool
 13  Region_Urban           754 non-null    bool
dtypes: bool(5), float64(9)
memory usage: 62.6 KB
```

Step 9

Feature Selection

```
df_with_target = df.copy()
df_with_target['target'] = y

correlation = df_with_target.corr()['target'].abs()
correlation = correlation.drop('target')

N = 7
top_features = correlation.sort_values(ascending=False).head(N).index
X_selected = df[top_features.tolist()]

print("Selected Features based on correlation:", top_features.tolist())
```

```
Selected Features based on correlation: ['TotalSpend', 'AverageOrderValue', 'SatisfactionSpendScore', 'OrderValueScore', 'ValuePerOrder', 'IsPremiumMem
ber', 'PurchaseFrequency']
```

Step 10

Data Splitting

```
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_size=0.2, random_state=42)
```

```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((603, 7), (151, 7), (603,), (151,))
```

Step 11

Train the model

```
model = LinearRegression()
model.fit(X_train, y_train)
```

```
LinearRegression
LinearRegression()
```

Step 12

Make predictions

```
y_pred_test = model.predict(X_test)
```

Step 13

Evaluate the model

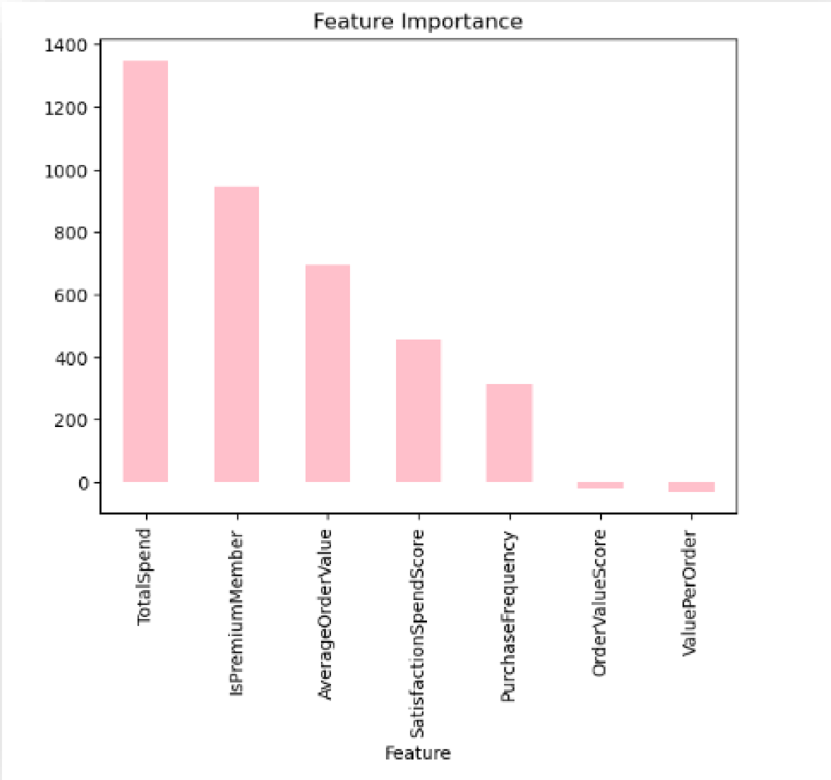
```
mse = mean_squared_error(y_test, y_pred_test)
print("Mean Squared Error:", mse)

rmse = np.sqrt(mse)
print("Root Mean Squared Error:", rmse)

r2 = r2_score(y_test, y_pred_test)
print("R² Score:", r2)
```

```
Mean Squared Error: 318959.4368884507
Root Mean Squared Error: 564.7649395000107
R² Score: 0.9501650750399655
```

Step 14

Feature Importance

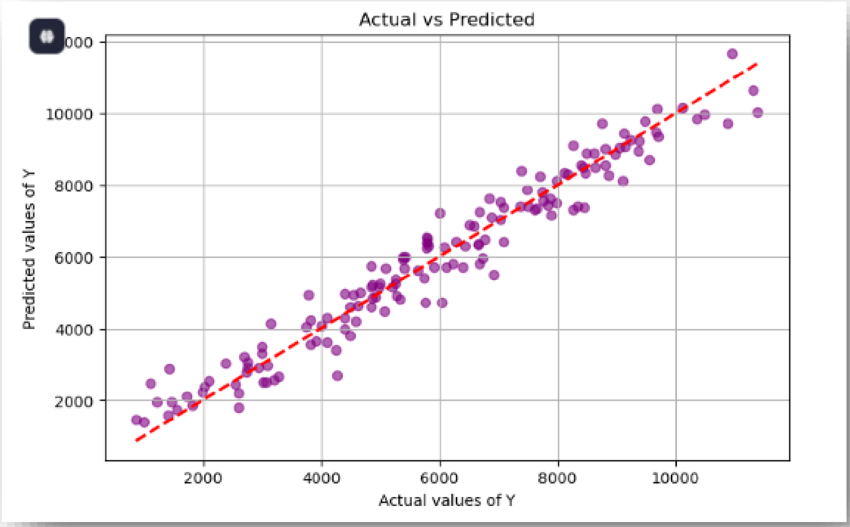
```
features = top_features

importance = model.coef_
importance_df = pd.DataFrame({'Feature': features, 'Importance': importance})

importance_df = importance_df.sort_values(by='Importance', ascending=False)

importance_df.plot(kind='bar', x='Feature', y='Importance', legend=False, title='Feature Importance', color='pink')
plt.show()
```



Step 15

Model Performance Visualization

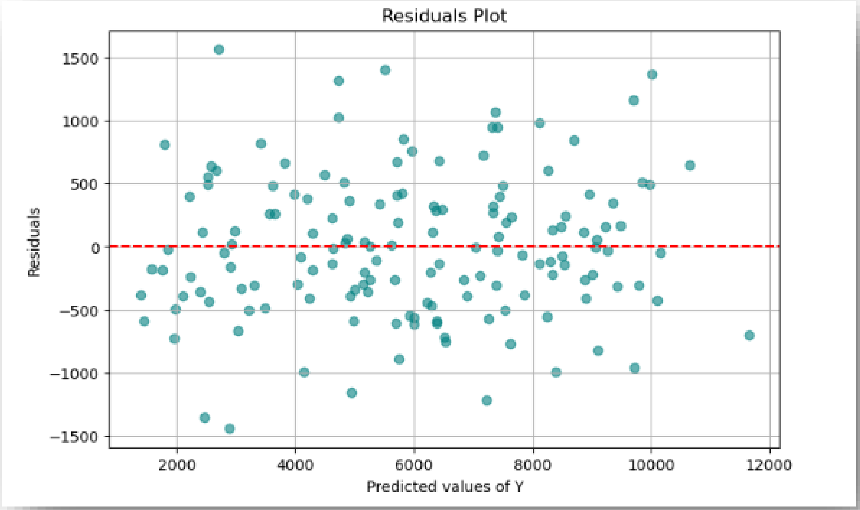
```
plt.figure(figsize=(8, 5))

plt.scatter(y_test, y_pred_test, color='purple', alpha=0.6)

plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)

plt.xlabel('Actual values of Y')
plt.ylabel('Predicted values of Y')
plt.title('Actual vs Predicted')
plt.grid(True)
plt.show()
```



Step 16

Assessing Error Patterns in Predictions

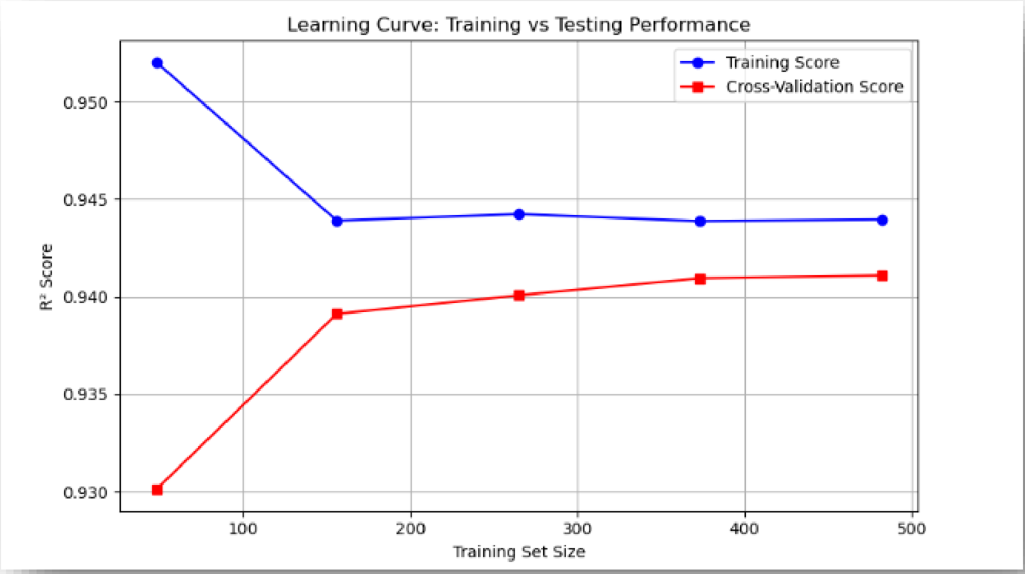
```
residuals = y_test - y_pred_test
plt.figure(figsize=(8, 5))
plt.scatter(y_pred_test, residuals, alpha=0.6, color='teal')
plt.axhline(y=0, color='red', linestyle='--')
plt.xlabel('Predicted values of Y')
plt.ylabel('Residuals')
plt.title('Residuals Plot')
plt.grid(True)
plt.show()
```



Step 17

Training vs Cross-Validation Performance

```
train_sizes, train_scores, test_scores = learning_curve(
    model, X_train, y_train, cv=5, scoring='r2'
)

train_mean = np.mean(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)

plt.figure(figsize=(8, 5))
plt.plot(train_sizes, train_mean, label='Training Score', color='blue', marker='o')
plt.plot(train_sizes, test_mean, label='Cross-Validation Score', color='red', marker='s')
plt.xlabel('Training Set Size')
plt.ylabel('R² Score')
plt.title('Learning Curve: Training vs Testing Performance')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



Conclusion

The linear regression model demonstrated a reasonable capability in predicting Customer Lifetime Value (CLV), supported by consistent performance across multiple error metrics and cross-validation techniques. The success of the model was significantly influenced by thorough preprocessing, including outlier removal, standardization, encoding, and feature selection. These steps helped reduce noise, improve data quality, and enhance the model's ability to learn from patterns in the dataset.

Although the linear regression approach effectively captured general trends in customer behavior, its limitations became evident in handling complex or non-linear relationships within the data. The model tended to underperform in cases involving high variance or intricate feature interactions, which suggests that relying solely on linear assumptions may not be optimal for all datasets.

To further enhance predictive performance, future work could involve experimenting with regularized linear models like Ridge and Lasso Regression, which can handle multicollinearity and overfitting more effectively. Additionally, integrating non-linear models such as Decision Trees, Random Forests, or Gradient Boosting methods could offer improved accuracy, especially for datasets with more complex patterns. Incorporating feature engineering and hyperparameter tuning could also optimize results.

Overall, the project highlights the foundational value of linear regression in predictive modeling while pointing to opportunities for refinement using more advanced techniques. It sets a solid groundwork for building more accurate and robust models in the domain of customer analytics.

Loan Approval Prediction using Classification

Summary

This project addresses the problem of loan approval prediction using multiple machine learning classification techniques. The dataset is carefully preprocessed by handling missing values, encoding categorical variables, and standardizing numerical features. Feature selection methods are applied to reduce dimensionality, and class imbalance is corrected using SMOTE oversampling. The study implements and compares three models: Random Forest, Support Vector Machine (SVM), and a basic Neural Network. These models are evaluated using accuracy, confusion matrix, classification report, and ROC-AUC score to determine the best-performing approach. The results highlight how machine learning can enhance the efficiency and accuracy of loan approval systems, offering data-driven support for financial decision-making.

Abstract

The project explores a loan approval classification problem using various machine learning techniques. The dataset is first cleaned and preprocessed, including handling missing values, encoding categorical variables, and standardizing numerical features. Feature selection is performed to reduce dimensionality. Due to class imbalance, SMOTE is applied to balance the dataset. Three models like Random Forest, Support Vector Machine (SVM), and a simple Neural Network are trained and evaluated. Performance is compared using classification metrics to determine the most effective approach. This study demonstrates how machine learning can assist financial institutions in automating and improving loan approval processes.

Objectives

- To clean and preprocess a loan application dataset for classification tasks.
- To address data imbalance using oversampling techniques like SMOTE.
- To apply and compare the performance of different classification models (Random Forest, SVM, and Neural Networks).
- To evaluate model performance using metrics such as accuracy, confusion matrix, classification report, and ROC-AUC.
- To identify the most relevant features impacting loan approval decisions.

Dataset Description

- **Number of rows:** 1,000
- **Number of columns:** 9
- **Purpose:** This dataset is structured for analyzing factors influencing **loan approvals** and may be used for predictive modeling in **loan eligibility classification** or **credit risk assessment**.

Column Description

1. **ApplicantIncome** *(int64)*
   The monthly income of the loan applicant.
2. **LoanAmount** *(int64)*
   The requested loan amount.
3. **CreditScore** *(int64)*
   A numeric score representing the applicant's creditworthiness. Higher values generally indicate lower risk.
4. **EmploymentStatus** *(object)*
   The applicant’s employment status. Categories include:
   - Employed
   - Unemployed
   - Self-employed
5. **MaritalStatus** *(object)*
   Marital status of the applicant: Married or Single.
6. **LoanTermMonths** *(float64, 1 missing value)*
   Duration of the loan in months. One value is missing.
7. **Dependents** *(float64, 2 missing values)*
   Number of people financially dependent on the applicant. Includes two missing entries.
8. **Education** *(object)*
   Applicant’s education level:
   - Graduate
   - Not Graduate
9. **LoanStatus** *(int64)*
   Target variable:
   - 1 = Loan Approved
   - 0 = Loan Rejected

Explanation of Steps

Step 1: Importing Libraries

Import required libraries like pandas, numpy, matplotlib, seaborn, and classification tools from sklearn.

Step 2: Reading Data

Load the loan dataset using pandas.read_csv().

Step 3: Exploring Data

View data structure, column types, and statistics using head(), info(), and describe().

Step 4: Cleansing Data

Fix missing values, convert data types if needed, and remove duplicates.

Step 5: Outlier Detection and Removal

Detect and remove extreme values using IQR or z-score to improve model stability.

Step 6: Separate Target Column

Divide dataset into features (X) and target (y, e.g., Loan_Status).

Step 7: Data Transformation (Standardization)

Scale numerical features using StandardScaler to normalize the input data.

Step 8: Categorical into Numerical (One-Hot Encoding)

Convert categorical columns into numeric format using get_dummies().

Step 9: Handle Imbalanced Data

Use techniques like SMOTE or class weights to balance the target classes if one is underrepresented.

Step 10: Feature Selection

Pick important features using correlation, feature importance, or tree-based models.

Step 11: Data Splitting

Split the data into training and test sets using train_test_split().

Step 12: Classification Models Training

Train models like Logistic Regression, Decision Tree, or Random Forest on the training data.

Step 13: Performance Analysis and Model Insights (Random Forest)

Evaluate the Random Forest model using accuracy, confusion matrix, precision, recall, F1-score, and view feature importance.

Step 1

Importing Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, learning_curve
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_curve, auc
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.feature_selection import SelectKBest, chi2
from imblearn.over_sampling import SMOTE
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import plot_model
```

Step 2

Reading Data

```
df = pd.read_csv('la.csv')
```

Step 3

Exploring Data

```
df.head()
```

	ApplicantIncome	LoanAmount	CreditScore	EmploymentStatus	MaritalStatus	LoanTermMonths	Dependents	Education	LoanStatus
0	9270	565	664	Unemployed	Married	36.0	2.0	Not Graduate	1
1	2860	703	625	Employed	Married	60.0	0.0	Not Graduate	0
2	7390	781	766	Self-employed	Married	60.0	2.0	Graduate	1
3	7191	928	713	Unemployed	Single	12.0	3.0	Not Graduate	1
4	13964	633	754	Unemployed	Married	48.0	2.0	Graduate	1

```
df.tail()
```

	ApplicantIncome	LoanAmount	CreditScore	EmploymentStatus	MaritalStatus	LoanTermMonths	Dependents	Education	LoanStatus
995	8777	459	526	Employed	Single	24.0	1.0	Not Graduate	1
996	13314	322	775	Employed	Single	48.0	3.0	Not Graduate	1
997	9570	965	776	Unemployed	Single	48.0	0.0	Graduate	1
998	9956	799	564	Employed	Married	48.0	3.0	Graduate	1
999	7124	547	671	Self-employed	Single	48.0	4.0	Not Graduate	1

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   ApplicantIncome  1000 non-null   int64
 1   LoanAmount       1000 non-null   int64
 2   CreditScore      1000 non-null   int64
 3   EmploymentStatus 1000 non-null   object
 4   MaritalStatus    1000 non-null   object
 5   LoanTermMonths   999 non-null    float64
 6   Dependents       998 non-null    float64
 7   Education        1000 non-null   object
 8   LoanStatus       1000 non-null   int64
dtypes: float64(2), int64(4), object(3)
memory usage: 70.4+ KB
```

```
df.describe()
```

	ApplicantIncome	LoanAmount	CreditScore	LoanTermMonths	Dependents	LoanStatus
count	1000.000000	1000.000000	1000.000000	999.000000	998.000000	1000.00000
mean	8614.667000	539.348000	576.627000	36.540541	1.957916	0.87900
std	3967.811171	264.910442	158.216612	16.683188	1.428409	0.32629
min	2004.000000	100.000000	300.000000	12.000000	0.000000	0.00000
25%	5501.000000	315.000000	441.750000	24.000000	1.000000	1.00000
50%	8616.500000	534.000000	577.500000	36.000000	2.000000	1.00000
75%	11730.750000	769.000000	712.250000	48.000000	3.000000	1.00000
max	57578.000000	998.000000	849.000000	60.000000	4.000000	1.00000

Step 4

Cleansing Data

Before Cleansing Data

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

```
ApplicantIncome     0
LoanAmount          0
CreditScore         0
EmploymentStatus    0
MaritalStatus       0
LoanTermMonths      1
Dependents          2
Education           0
LoanStatus          0
dtype: int64
```

After Cleansing Data

Handle missing values in numeric columns (fill with median)

```
numeric_cols = df.select_dtypes(include=['number']).columns
df[numeric_cols] = df[numeric_cols].fillna(df[numeric_cols].median())
```

Handle missing values in categorical columns (fill with mode)

```
categorical_cols = df.select_dtypes(include=['object']).columns
for col in categorical_cols:
    df[col] = df[col].fillna(df[col].mode()[0])
```

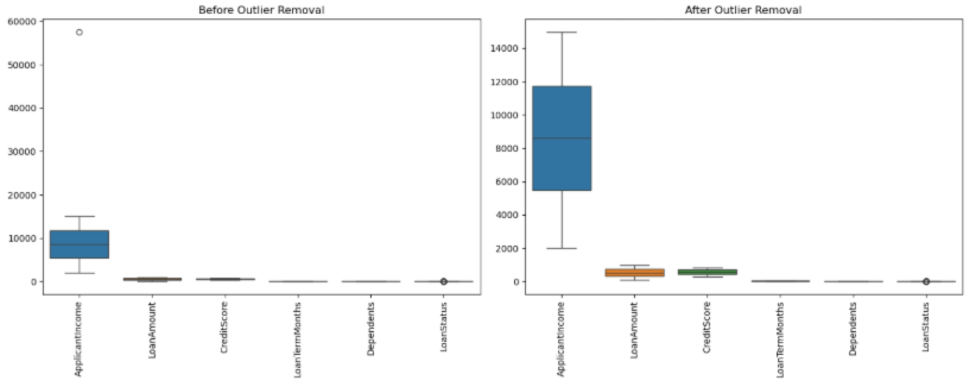
```
df.isnull().sum()

ApplicantIncome     0
LoanAmount          0
CreditScore         0
EmploymentStatus    0
MaritalStatus       0
LoanTermMonths      0
Dependents          0
Education           0
LoanStatus          0
dtype: int64
```

Step 5

Outlier Detection and Removal

```
df_class_1 = df[df['LoanStatus'] == 1]
df_class_0 = df[df['LoanStatus'] == 0]

def remove_outliers_iqr(df_subset):
    numeric_cols = df_subset.select_dtypes(include=[np.number]).columns
    Q1 = df_subset[numeric_cols].quantile(0.25)
    Q3 = df_subset[numeric_cols].quantile(0.75)
    IQR = Q3 - Q1
    mask = ~((df_subset[numeric_cols] < (Q1 - 1.5 * IQR)) |
             (df_subset[numeric_cols] > (Q3 + 1.5 * IQR))).any(axis=1)
    return df_subset[mask]

df_class_1_clean = remove_outliers_iqr(df_class_1.drop(columns=['LoanStatus']))
df_class_0_clean = remove_outliers_iqr(df_class_0.drop(columns=['LoanStatus']))

df_class_1_clean['LoanStatus'] = 1
df_class_0_clean['LoanStatus'] = 0

df_cleaned = pd.concat([df_class_1_clean, df_class_0_clean], axis=0).reset_index(drop=True)

print(f"Original class counts:\n{df['LoanStatus'].value_counts()}")
print(f"\nCleaned class counts:\n{df_cleaned['LoanStatus'].value_counts()}")
print(f"\nRows removed: {len(df) - len(df_cleaned)}")
print(f"Final shape of cleaned dataset: {df_cleaned.shape}")

numeric_cols = df.select_dtypes(include=[np.number]).columns

plt.figure(figsize=(15, 6))

plt.subplot(1, 2, 1)
sns.boxplot(data=df[numeric_cols])
plt.title('Before Outlier Removal')
plt.xticks(rotation=90)

plt.subplot(1, 2, 2)
sns.boxplot(data=df_cleaned[numeric_cols])
plt.title('After Outlier Removal')
plt.xticks(rotation=90)

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

```
Original class counts:
LoanStatus
1    879
0    121
Name: count, dtype: int64

Cleaned class counts:
LoanStatus
1    878
0    121
Name: count, dtype: int64

Rows removed: 1
Final shape of cleaned dataset: (999, 9)
```



Step 6

Separate Target Column

```
X = df_cleaned.drop('LoanStatus', axis=1)
y = df_cleaned['LoanStatus']
```

Step 7

Data Transformation (Standardization)

Before Standardization

```
X.head()
```

	ApplicantIncome	LoanAmount	CreditScore	EmploymentStatus	MaritalStatus	LoanTermMonths	Dependents	Education
0	9270	565	664	Unemployed	Married	36.0	2.0	Not Graduate
1	7390	781	766	Self-employed	Married	60.0	2.0	Graduate
2	7191	928	713	Unemployed	Single	12.0	3.0	Not Graduate
3	13964	633	754	Unemployed	Married	48.0	2.0	Graduate
4	13284	504	532	Self-employed	Married	12.0	3.0	Not Graduate

After Standardization

```
numeric_cols = X.select_dtypes(include=np.number).columns

X[numeric_cols] = X[numeric_cols].astype(float)

scaler = StandardScaler()
X.loc[:, numeric_cols] = scaler.fit_transform(X[numeric_cols])

X.head()
```

	ApplicantIncome	LoanAmount	CreditScore	EmploymentStatus	MaritalStatus	LoanTermMonths	Dependents	Education
0	0.192835	0.097154	0.551353	Unemployed	Married	-0.031007	0.029462	Not Graduate
1	-0.321870	0.912564	1.196788	Self-employed	Married	1.409717	0.029462	Graduate
2	-0.376352	1.467497	0.861415	Unemployed	Single	-1.471730	0.730243	Not Graduate
3	1.477955	0.353857	1.120854	Unemployed	Married	0.689355	0.029462	Graduate
4	1.291785	-0.133124	-0.283914	Self-employed	Married	-1.471730	0.730243	Not Graduate

Step 8

Categorical into Numerical (One-Hot Encoding)

Before One-Hot Encoding

```
X.shape
```

```
(999, 8)
```

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 999 entries, 0 to 998
Data columns (total 8 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   ApplicantIncome   999 non-null    float64
 1   LoanAmount        999 non-null    float64
 2   CreditScore       999 non-null    float64
 3   EmploymentStatus  999 non-null    object
 4   MaritalStatus     999 non-null    object
 5   LoanTermMonths    999 non-null    float64
 6   Dependents        999 non-null    float64
 7   Education         999 non-null    object
dtypes: float64(5), object(3)
memory usage: 62.6+ KB
```

After One-Hot Encoding

```
X = pd.get_dummies(X, drop_first=False)
```

```
X.shape
```

```
(999, 12)
```

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 999 entries, 0 to 998
Data columns (total 12 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   ApplicantIncome              999 non-null    float64
 1   LoanAmount                   999 non-null    float64
 2   CreditScore                  999 non-null    float64
 3   LoanTermMonths               999 non-null    float64
 4   Dependents                   999 non-null    float64
 5   EmploymentStatus_Employed    999 non-null    bool
 6   EmploymentStatus_Self-employed 999 non-null  bool
 7   EmploymentStatus_Unemployed  999 non-null    bool
 8   MaritalStatus_Married        999 non-null    bool
 9   MaritalStatus_Single         999 non-null    bool
 10  Education_Graduate           999 non-null    bool
 11  Education_Not Graduate       999 non-null    bool
dtypes: bool(7), float64(5)
memory usage: 46.0 KB
```

Step 9

Handle Imbalanced Data

Before Handling Imbalanced Data

```
y.value_counts()
```

```
LoanStatus
1    878
0    121
Name: count, dtype: int64
```

After Handling Imbalanced Data

```
smote = SMOTE(random_state=42)
X, y = smote.fit_resample(X, y)
```

```
print(X.shape)
print(y.shape)
y.value_counts()
```

```
(1756, 12)
(1756,)
```

```
LoanStatus
1    878
0    878
Name: count, dtype: int64
```

Step 10

Feature Selection

```
from sklearn.feature_selection import SelectKBest, f_classif

selector = SelectKBest(score_func=f_classif, k=6)
X_selected = selector.fit_transform(X, y)
selected_features = X.columns[selector.get_support()]
print("Selected features:", selected_features)
```

```
Selected features: Index(['ApplicantIncome', 'LoanAmount', 'CreditScore',
       'EmploymentStatus_Employed', 'EmploymentStatus_Unemployed',
       'MaritalStatus_Single'],
      dtype='object')
```

Step 11

Data Splitting

```
X_train, X_test, y_train, y_test = train_test_split(X[selected_features], y, test_size=0.2, random_state=42)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((1404, 6), (352, 6), (1404,), (352,))
```

Step 12

Classification Models Training

a. Random Forest

Train the model

```
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier()
```

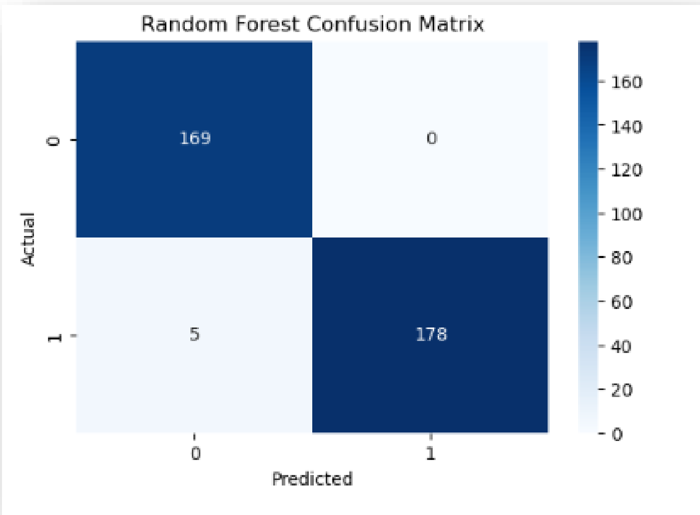
Classification Report

```python
y_pred_rf = rf.predict(X_test)
print("\nRandom Forest")
print("Accuracy:", accuracy_score(y_test, y_pred_rf))
print("Classification Report:\n", classification_report(y_test, y_pred_rf))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_rf))
```

```
Random Forest
Accuracy: 0.9857954545454546
Classification Report:
               precision    recall  f1-score   support

           0       0.97      1.00      0.99       169
           1       1.00      0.97      0.99       183

    accuracy                           0.99       352
   macro avg       0.99      0.99      0.99       352
weighted avg       0.99      0.99      0.99       352

Confusion Matrix:
 [[169   0]
 [  5 178]]
```

Confusion Matrix

```python
cm_rf = confusion_matrix(y_test, y_pred_rf)
plt.figure(figsize=(6,4))
sns.heatmap(cm_rf, annot=True, fmt='d', cmap='Blues')
plt.title('Random Forest Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



b. SVM

Train the model

```
svm = SVC(probability=True)
svm.fit(X_train, y_train)
```

```
SVC
SVC(probability=True)
```

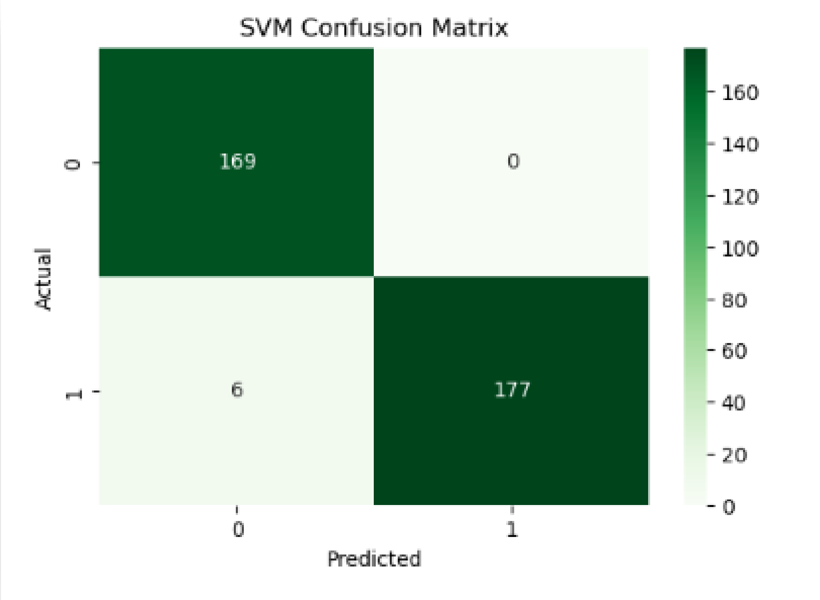
Classification Report

```
y_pred_svm = svm.predict(X_test)
print("\nSVM")
print("Accuracy:", accuracy_score(y_test, y_pred_svm))
print("Classification Report:\n", classification_report(y_test, y_pred_svm))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_svm))
```

```
SVM
Accuracy: 0.9829545454545454
Classification Report:
               precision    recall  f1-score   support

           0       0.97      1.00      0.98       169
           1       1.00      0.97      0.98       183

    accuracy                           0.98       352
   macro avg       0.98      0.98      0.98       352
weighted avg       0.98      0.98      0.98       352

Confusion Matrix:
 [[169   0]
 [  6 177]]
```

Confusion Matrix

```
cm_svm = confusion_matrix(y_test, y_pred_svm)
plt.figure(figsize=(6,4))
sns.heatmap(cm_svm, annot=True, fmt='d', cmap='Greens')
plt.title('SVM Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



c. ANN

Train the model

```
ann = Sequential()
ann.add(Dense(32, input_dim=X_train.shape[1], activation='relu'))
ann.add(Dense(16, activation='relu'))
ann.add(Dense(1, activation='sigmoid'))
ann.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
ann.fit(X_train, y_train, epochs=50, batch_size=32, verbose=1)
ann_preds = (ann.predict(X_test) > 0.5).astype(int)
```


dim` argument to a layer. When using Sequential models, prefer using an `Input(
super().__init__(activity_regularizer=activity_regularizer, **kwargs)

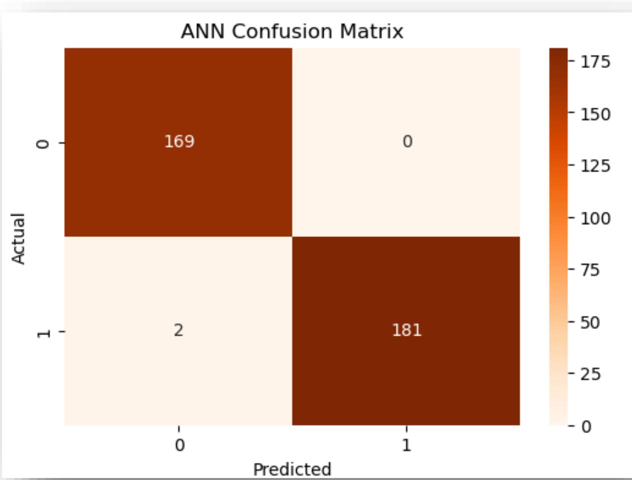
Classification Report

```python
print("\nANN")
print("Accuracy:", accuracy_score(y_test, ann_preds))
print("Classification Report:\n", classification_report(y_test, ann_preds))
print("Confusion Matrix:\n", confusion_matrix(y_test, ann_preds))
```

```
ANN
Accuracy: 0.9943181818181818
Classification Report:
               precision    recall  f1-score   support

           0       0.99      1.00      0.99       169
           1       1.00      0.99      0.99       183

    accuracy                           0.99       352
   macro avg       0.99      0.99      0.99       352
weighted avg       0.99      0.99      0.99       352

Confusion Matrix:
 [[169   0]
 [  2 181]]
```

Confusion Matrix

```python
cm_ann = confusion_matrix(y_test, ann_preds)
plt.figure(figsize=(6,4))
sns.heatmap(cm_ann, annot=True, fmt='d', cmap='Oranges')
plt.title('ANN Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

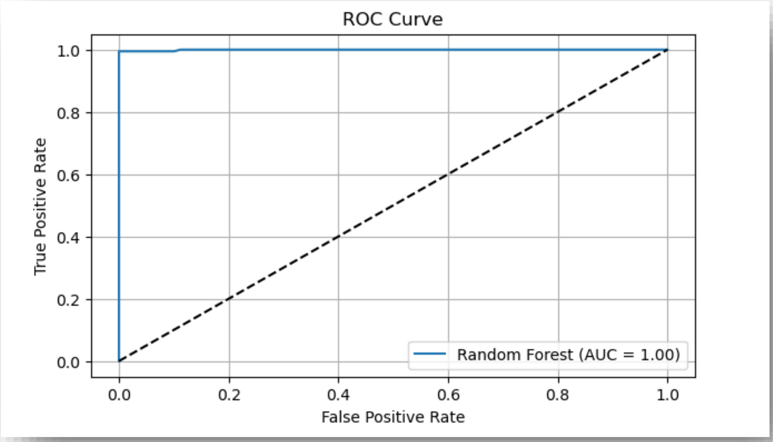


Step 13

Performance Analysis and Model Insights (Random Forest)

ROC Curve

```python
y_prob_rf = rf.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_prob_rf)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(7,4))
plt.plot(fpr, tpr, label=f'Random Forest (AUC = {roc_auc:.2f})')
plt.plot([0,1], [0,1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.grid()
plt.show()
```

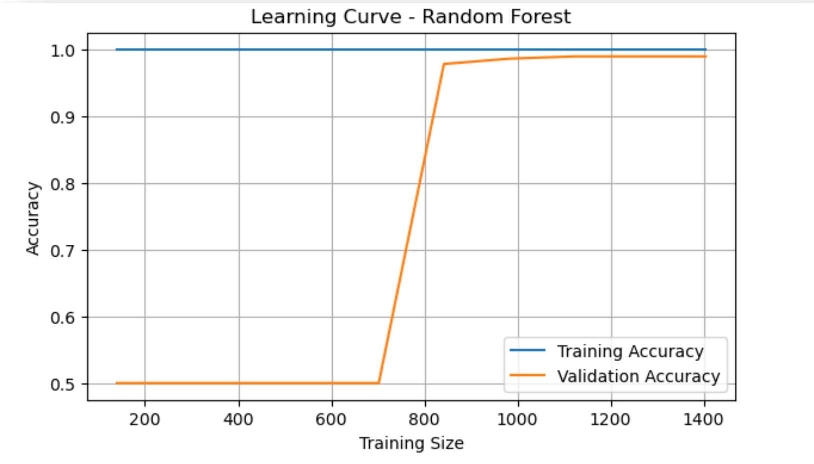



```python
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.ensemble import RandomForestClassifier

skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
model = RandomForestClassifier()

scores = cross_val_score(model, X, y, cv=skf, scoring='roc_auc')
print("AUC scores from cross-validation:", scores)
print("Mean AUC:", scores.mean())
```

```
AUC scores from cross-validation: [0.99927363 0.99917208 0.99709416 0.99970779 0.99980519]
Mean AUC: 0.9990105703955136
```

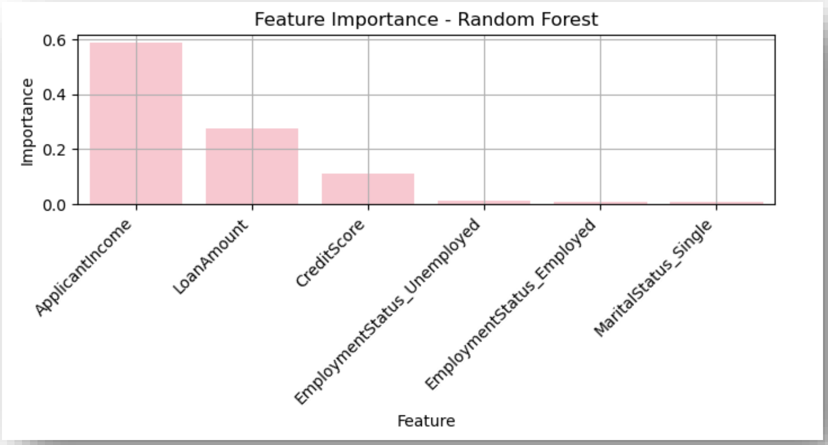
Learning Curve

```
train_sizes, train_scores, test_scores = learning_curve(
    rf, X[selected_features], y, cv=5, scoring='accuracy', train_sizes=np.linspace(0.1, 1.0, 10))

train_mean = np.mean(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)

plt.figure(figsize=(7,4))
plt.plot(train_sizes, train_mean, label='Training Accuracy')
plt.plot(train_sizes, test_mean, label='Validation Accuracy')
plt.xlabel('Training Size')
plt.ylabel('Accuracy')
plt.title('Learning Curve - Random Forest')
plt.legend()
plt.grid()
plt.show()
```




```
importances = rf.feature_importances_
indices = np.argsort(importances)[::-1]
features_sorted = [selected_features[i] for i in indices]
importances_sorted = importances[indices]

plt.figure(figsize=(7, 4))
sns.barplot(x=features_sorted, y=importances_sorted,  color='pink')
plt.title('Feature Importance - Random Forest')
plt.xlabel('Feature')
plt.ylabel('Importance')
plt.xticks(rotation=45, ha='right')
plt.grid()
plt.tight_layout()
plt.show()
```

Feature Importance



Conclusion

The project successfully implements a comprehensive classification pipeline to predict loan approvals using machine learning techniques. Through systematic data preprocessing—including handling missing values, encoding categorical data, and standardizing features—the dataset was prepared for effective model training. The application of SMOTE to address class imbalance proved essential in enhancing the fairness and performance of the classifiers.

Among the three models evaluated—Random Forest, Support Vector Machine (SVM), and a simple Neural Network—each demonstrated unique strengths. The Random Forest and Neural Network models delivered more consistent and robust performance across key evaluation metrics such as accuracy, precision, recall, and ROC-AUC. These results emphasize the importance of model selection based on the specific goals and characteristics of the dataset.

Beyond predictive accuracy, the project also offers valuable insights into the most influential features affecting loan approval decisions, which can be of practical use to financial institutions. The entire workflow reinforces the significance of proper data handling, class balancing, and thorough evaluation when dealing with real-world classification problems.

Overall, this study illustrates the potential of machine learning to transform traditional loan approval processes into automated, data-driven systems that are more efficient, scalable, and objective. With further refinement and deployment, such solutions can enhance decision-making, reduce processing time, and improve the customer experience in financial services.