School of IT & Business Technologies
Graduate Diploma in Data Analytics (Level 7)
Cover Sheet and Student Declaration

This sheet must be signed by the student and attached to the submitted assessment.

Course Title:	Machine Learning and AI	**Course code:**	GDDA708
Student Name:	Mira Torririt	**Student ID:**	764707793
Assessment No & Type:	Assessment 2[Portfolio]	**Cohort:**	GDDA7123C
Due Date:	07/03/24	**Date Submitted:**	07/03/24
Tutor's Name:	Harsh Tiwari		
Assessment Weighting	60%		
Total Marks	100		

Student Declaration:

I declare that:

- I have read the New Zealand School of Education Ltd policies and regulations on assessments and understand what plagiarism is.
- I am aware of the penalties for cheating and plagiarism as laid down by the New Zealand School of Education Ltd.
- This is an original assessment and is entirely my own work.
- Where I have quoted or made use of the ideas of other writers, I have acknowledged the source.
- This assessment has been prepared exclusively for this course and has not been or will not be submitted as assessed work in any other course.
- It has been explained to me that this assessment may be used by NZSE Ltd, for internal and/or external moderation.
- If I am late in handing in this assessment without prior approval (see student regulations in handbook), marks will be deducted, to a maximum of 50%.

Student signature:

Date: 07/03/24

Tutor only to complete		
Assessment result:	**Mark /100**	**Grade**

Assessment 2: GDDA708 – Machine Learning and AI

Portfolio: Regression and Classification

Mira Torririt

GDD708

Lecturer: Dr. Harshvardhan Tiwari

School of Technology
Graduate Diploma in Data Analytics (Level 7)

March 07, 2024

Table of Contents

Introduction

Regression and classification are supervised learning algorithms that can be used in forecasting. Both are considered effective instruments in solving market problems. Regression helps us determine patterns in continuous data using predictions. For example, it uses records to determine the likelihood of rainfall in a specific region. It also shows trends based on past data. On the other hand, classification works on the identification of a design or a role that separates the items into categories or classes. It creates a rule likened to the "If-Then" rule (Sarangam, A. 2021).

Linear regression is a statistical method for showing relationships between independent and dependent variables. But why is it not suitable for classification problems? The answer is that linear regression predicts continuous values, whereas the classification's target variable is discrete (Kumar, A., 2021).

In this assessment, I will demonstrate how to use regression and classification and some techniques that make them effective.

Part A - Regression

The dataset used is from Kaggle. It concerns insurance charges and the factors affecting their cost: age, sex, BMI, number of children, whether the client is a smoker and region. In this part, a linear regression has been built to predict the insurance cost.

Note: The dataset is also available in my GitHub account via this link: https://github.com/Myres16/Data-Analytics-Assessments/tree/main/708

Task 1 – Data Preprocessing

The dataset was loaded in the Python notebook by importing pandas and two additional libraries for future use in data preparation.

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

```
C:\Users\torri\anaconda3\Lib\site-packages\pandas\core\arrays\masked.py:60: UserWarning: Pandas requires version '1.3.6' or new
er of 'bottleneck' (version '1.3.5' currently installed).
  from pandas.core import (
```

```
insurance_df = pd.read_csv('insurance.csv')
insurance_df
```

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520
...	...	...	...	...	...	...	...
1333	50	male	30.970	3	no	northwest	10600.54830
1334	18	female	31.920	0	no	northeast	2205.98080
1335	18	female	36.850	0	no	southeast	1629.83350
1336	21	female	25.800	0	no	southwest	2007.94500
1337	61	female	29.070	0	yes	northwest	29141.36030

1338 rows × 7 columns

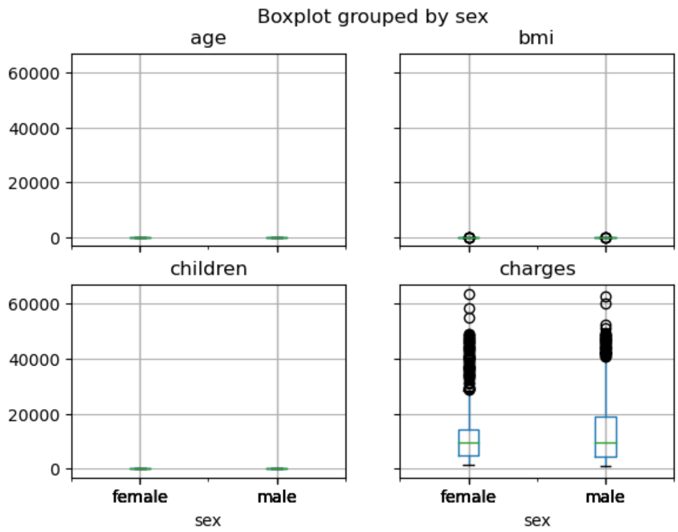
a) Data Cleaning – the dataset has no missing value but has outliers. In order to check the missing value, I used the function df.isnull().sum().

```
# Checking for missing values and outliers.
insurance_df.isnull().sum()
```

```
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64
```

I found the outliers using the function df.boxplot and, for visualization using the function plt.show().

```
# Outliers and its visualization.
# Found 2 columns with outliers (bmi and charges)
insurance_df.boxplot(column=['age', 'bmi', 'children', 'charges'], by='sex')
plt.show()
```



To manage the outliers in the charge's column, I used the robust z-score to remove all the outliers. Initially, I used the z-score, but outliers were remaining.

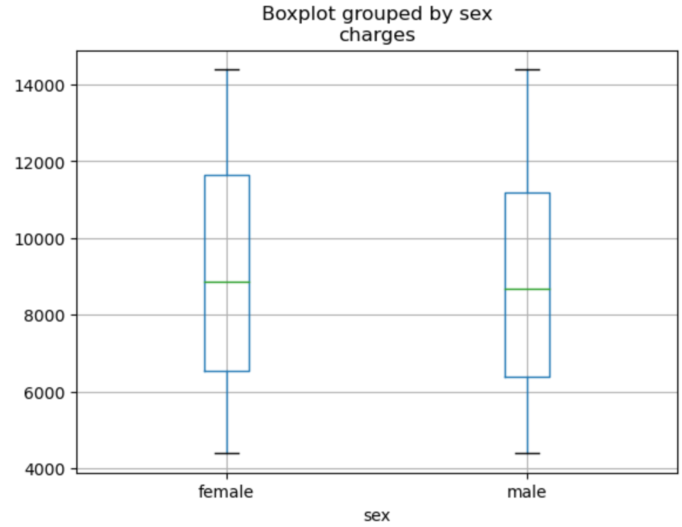
```
# Removing the outliers

def get_outliers(df, column):
    median_values = df[column].median()
    mad_charges = np.median(np.abs(df[column] - median_values))
    df['z_score'] = (df[column] - median_values) / mad_charges
    threshold = 1
    return (df['z_score'].abs() < threshold)

outliers = get_outliers(insurance_df, 'charges')
# Remove outliers from the dataset
insurance_df = insurance_df[outliers]
insurance_df
```

	age	sex	bmi	children	smoker	region	charges	z_score
2	28	male	33.00	3	no	southeast	4449.46200	-0.982827
6	46	female	33.44	1	no	southeast	8240.58960	-0.227435
7	37	female	27.74	3	no	northwest	7281.50560	-0.418535
8	37	male	29.83	2	no	northeast	6406.41070	-0.592900
13	56	female	39.82	0	no	southeast	11090.71780	0.340460
...	...	...	...	...	...	...	...	...
1329	52	male	38.60	2	no	southwest	10325.20600	0.187930
1330	57	female	25.74	2	no	southeast	12629.16560	0.646999
1331	23	female	33.40	0	no	southwest	10795.93733	0.281724
1332	52	female	44.70	3	no	southwest	11411.68500	0.404413
1333	50	male	30.97	3	no	northwest	10600.54830	0.242792

669 rows × 8 columns

Displaying if the outliers were removed using df.boxplot function and plt.show for visualization.

```
# Checking if the outliers were removed.
insurance_df.boxplot(column=['charges'], by='sex')
plt.show()
```



Following the same process to check and manage the outliers in BMI column.

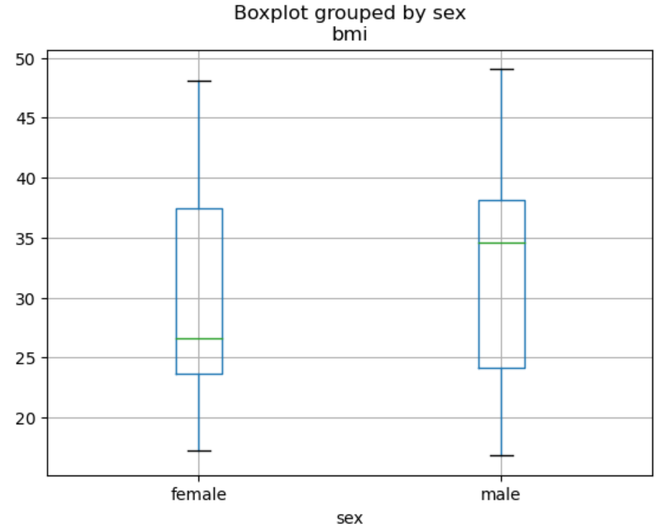
```
# Identify outliers
outliers = get_outliers(insurance_df, 'bmi')

# Remove outliers from the dataset
insurance_df = insurance_df[~outliers]
insurance_df
```

```
C:\Users\torri\AppData\Local\Temp\ipykernel_28272\3987391177.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy
  df['z_score'] = (df[column] - median_values) / mad_charges
```

	age	sex	bmi	children	smoker	region	charges	z_score
0	19	female	27.90	0	yes	southwest	16884.92400	-0.513382
1	18	male	33.77	1	no	southeast	1725.55230	0.914842
2	28	male	33.00	3	no	southeast	4449.46200	0.727494
4	32	male	28.88	0	no	northwest	3866.85520	-0.274939
5	31	female	25.74	0	no	southeast	3756.62160	-1.038929
...	...	...	...	...	...	...	...	...
1331	23	female	33.40	0	no	southwest	10795.93733	0.824818
1333	50	male	30.97	3	no	northwest	10600.54830	0.233577
1334	18	female	31.92	0	no	northeast	2205.98080	0.464720
1335	18	female	36.85	0	no	southeast	1629.83350	1.664234
1336	21	female	25.80	0	no	southwest	2007.94500	-1.024331

856 rows × 8 columns

```
# Running boxplot to check if outliers are removed
insurance_df.boxplot(column=['bmi'], by='sex')
plt.show()
```



b. Feature scaling has been done to all the variables except the charges, which are considered an outcome variable.

```
# b) Feature scaling

from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Min-Max Scaling
min_max_scaler = MinMaxScaler()
insurance_df[['age','bmi','children']] = min_max_scaler.fit_transform(insurance_df[['age','bmi','children']])
insurance_df
```

```
C:\Users\torri\AppData\Local\Temp\ipykernel_28272\1931882604.py:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy
  insurance_df[['age','bmi','children']] = min_max_scaler.fit_transform(insurance_df[['age','bmi','children']])
```

	age	sex	bmi	children	smoker	region	charges	z_score
0	0.021739	female	0.370257	0.0	yes	southwest	16884.92400	-0.513382
1	0.000000	male	0.729498	0.2	no	southeast	1725.55230	0.914842
2	0.217391	male	0.682375	0.6	no	southeast	4449.46200	0.727494
4	0.304348	male	0.430233	0.0	no	northwest	3866.85520	-0.274939
5	0.282609	female	0.238066	0.0	no	southeast	3756.62160	-1.038929
...	...	...	...	...	...	...	...	...
1331	0.108696	female	0.706854	0.0	no	southwest	10795.93733	0.824818
1333	0.695652	male	0.558140	0.6	no	northwest	10600.54830	0.233577
1334	0.000000	female	0.616279	0.0	no	northeast	2205.98080	0.464720
1335	0.000000	female	0.917993	0.0	no	southeast	1629.83350	1.664234
1336	0.065217	female	0.241738	0.0	no	southwest	2007.94500	-1.024331

856 rows × 8 columns

c) Encoding the categorical values using a label encoder; pd.get_dummies

```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
insurance_df['smoker'] = label_encoder.fit_transform(insurance_df['smoker'])
insurance_df
```

```
C:\Users\torri\AppData\Local\Temp\ipykernel_28272\373363711.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy
  insurance_df['smoker'] = label_encoder.fit_transform(insurance_df['smoker'])
```

	age	sex	bmi	children	smoker	region	charges	z_score
0	0.021739	female	0.370257	0.0	1	southwest	16884.92400	-0.513382
1	0.000000	male	0.729498	0.2	0	southeast	1725.55230	0.914842
2	0.217391	male	0.682375	0.6	0	southeast	4449.46200	0.727494
4	0.304348	male	0.430233	0.0	0	northwest	3866.85520	-0.274939
5	0.282609	female	0.238066	0.0	0	southeast	3756.62160	-1.038929
...	...	...	...	...	...	...	...	...
1331	0.108696	female	0.706854	0.0	0	southwest	10795.93733	0.824818
1333	0.695652	male	0.558140	0.6	0	northwest	10600.54830	0.233577
1334	0.000000	female	0.616279	0.0	0	northeast	2205.98080	0.464720
1335	0.000000	female	0.917993	0.0	0	southeast	1629.83350	1.664234
1336	0.065217	female	0.241738	0.0	0	southwest	2007.94500	-1.024331

856 rows × 8 columns

```
# c) Encoding the categorical values

insurance_df_encoded = pd.get_dummies(insurance_df, columns=['sex', 'region'], drop_first=True)
insurance_df_encoded
```

	age	bmi	children	smoker	charges	z_score	sex_male	region_northwest	region_southeast	region_southwest
0	0.021739	0.370257	0.0	1	16884.92400	-0.513382	False	False	False	True
1	0.000000	0.729498	0.2	0	1725.55230	0.914842	True	False	True	False
2	0.217391	0.682375	0.6	0	4449.46200	0.727494	True	False	True	False
4	0.304348	0.430233	0.0	0	3866.85520	-0.274939	True	True	False	False
5	0.282609	0.238066	0.0	0	3756.62160	-1.038929	False	False	True	False
...	...	...	...	...	...	...	...	...	...	...
1331	0.108696	0.706854	0.0	0	10795.93733	0.824818	False	False	False	True
1333	0.695652	0.558140	0.6	0	10600.54830	0.233577	True	True	False	False
1334	0.000000	0.616279	0.0	0	2205.98080	0.464720	False	False	False	False
1335	0.000000	0.917993	0.0	0	1629.83350	1.664234	False	False	True	False
1336	0.065217	0.241738	0.0	0	2007.94500	-1.024331	False	False	False	True

856 rows × 10 columns

Converting ‘True and False’ to binary for consistency and interpretability and to improve the model performance:

insurance_df_encoded[['sex_male', 'region_northwest', 'region_southeast', 'region_southwest']] = insurance_df_encoded[['sex_male', 'region_northwest', 'region_southeast', 'region_southwest']].replace({True: 1, False: 0})
insurance_df_encoded

```
']] = insurance_df_encoded[['sex_male', 'region_northwest', 'region_southeast', 'region_southwest']].replace({True: 1, False: 0})
```

```
C:\Users\torri\AppData\Local\Temp\ipykernel_28272\182238360.py:1: FutureWarning: Downcasting behavior in `replace` is deprecate
d and will be removed in a future version. To retain the old behavior, explicitly call `result.infer_objects(copy=False)`. To o
pt-in to the future behavior, set `pd.set_option('future.no_silent_downcasting', True)`
  insurance_df_encoded[['sex_male', 'region_northwest', 'region_southeast', 'region_southwest']] = insurance_df_encoded[['sex_m
ale', 'region_northwest', 'region_southeast', 'region_southwest']].replace({True: 1, False: 0})
```

	age	bmi	children	smoker	charges	z_score	sex_male	region_northwest	region_southeast	region_southwest
0	0.021739	0.370257	0.0	1	16884.92400	-0.513382	0	0	0	1
1	0.000000	0.729498	0.2	0	1725.55230	0.914842	1	0	1	0
2	0.217391	0.682375	0.6	0	4449.46200	0.727494	1	0	1	0
4	0.304348	0.430233	0.0	0	3866.85520	-0.274939	1	1	0	0
5	0.282609	0.238066	0.0	0	3756.62160	-1.038929	0	0	1	0
...	...	...	...	...	...	...	...	...	...	...
1331	0.108696	0.706854	0.0	0	10795.93733	0.824818	0	0	0	1
1333	0.695652	0.558140	0.6	0	10600.54830	0.233577	1	1	0	0
1334	0.000000	0.616279	0.0	0	2205.98080	0.464720	0	0	0	0
1335	0.000000	0.917993	0.0	0	1629.83350	1.664234	0	0	1	0
1336	0.065217	0.241738	0.0	0	2007.94500	-1.024331	0	0	0	1

856 rows × 10 columns

d) Splitting the data to testing and training. This process is considered a fundamental step in machine learning for model evaluation, hyperparameter tuning, model selection, and cross-validation.

```python
# d) Splitting the data to testing and training

from sklearn.model_selection import train_test_split

X = insurance_df_encoded.drop(columns='charges')
y = insurance_df_encoded['charges']

# Splitting the dataset into training (80%) and testing(20%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("Training set - Features:", X_train.shape, "Target:", y_train.shape)
print("Testing set - Features:", X_test.shape, "Target:", y_test.shape)
```

```
Training set - Features: (684, 9) Target: (684,)
Testing set - Features: (172, 9) Target: (172,)
```

Task 2 – Model Building with hyper-parameter tuning

a) Two additional libraries were imported to perform the linear regression model: Linear Regression and mean_squared_error, r2_score. The rationality for choosing this model assumes a linear relationship between the dependent and the independent variables.

```python
# a) Linear Regression Model

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

model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
print("R^2 Score:", r2)
```

```
Mean Squared Error: 4748550.245079115
Root Mean Squared Error: 2179.116849799275
R^2 Score: 0.774186649869275
```

b. For hyperparameter tuning, the randomized search was performed as it accommodates the hyperparameter's continuous nature. This means that it is capable of taking any real value within a certain range.

```
# b) Hyper-parameter tuning using Random Search

from sklearn.model_selection import RandomizedSearchCV

# Define the hyperparameters and their possible values
param_dist = {
    'fit_intercept': [True, False]
}

# Initialize the linear regression model
model = LinearRegression()

# Initialize the random search with cross-validation
random_search = RandomizedSearchCV(
    model, param_distributions=param_dist, n_iter=4, cv=5, scoring='neg_mean_squared_error', random_state=42
)

# Perform the random search on the training data
random_search.fit(X_train, y_train)

# Get the best parameters
best_params = random_search.best_params_

print("Best Hyperparameters:", best_params)
```

```
Best Hyperparameters: {'fit_intercept': False}
```

Performing a new linear regression model with the best hyperparameter fit intercept: False for comparison with the previous one.

```
# Performing a new linear regression model with the best hyperparameter fit intercept: False

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

# Create a new linear regression model with the best hyperparameters
best_model = LinearRegression()

# Train the model on the training set
best_model.fit(X_train, y_train)

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

# Evaluate the model performance
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error on Test Set:", mse)
```

```
Mean Squared Error on Test Set: 4748550.245079115
```

c) I created a linear regression model by importing additional libraries: train_test_split, mean_absolute_error, mean_squared_error, and r2_score. I first created the data using the ones with numerical values in my X-axis and then called charges value as my y-axis. Then, set the seed to 42 (example) as a string point/value. Lastly, the X and y axes are split to train and test.

```
# c) Building a linear regression model

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

#np.random.seed(42)
# Data
X= insurance_df_encoded.drop(columns='charges')
y= insurance_df_encoded['charges']

#split into training and test
X_train, X_test, y_train,y_test = train_test_split(X,y,test_size=0.20, random_state=42)


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

```
▾ LinearRegression
LinearRegression()
```

Task 3 – Model Evaluation and Selection

a) Performance evaluation of the regression model.

```
# a) Evaluation of regression model

#np.random.seed(42)
# Data
X= insurance_df_encoded.drop(columns='charges')
y= insurance_df_encoded['charges']

#split into training and test
X_train, X_test, y_train,y_test = train_test_split(X,y, test_size=0.2, random_state=42)


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

y_pred = model.predict(X_test)

# Calculate evaluation metrics
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print(f"MAE: {mae:.2f}")
print(f"MSE: {mse:.2f}")
print(f"RMSE: {rmse:.2f}")
print(f"R-squared: {r2:.2f}")
```

```
MAE: 944.09
MSE: 4748550.25
RMSE: 2179.12
R-squared: 0.77
```

There is no specific benchmark to measure the MAE, MSE, RMSE, and R-squared, as they will vary depending on the nature of the target value. In this scenario, the 'charges' column is the outcome value. Generally, a lower MAE, MSE, and RMSE have better performance. The R-squared result is considered high, suggesting a good fit model.

b) Implementing k-fold validation to assess the model – This process is done to enhance the model selection. It also helps address issues associated with variability. Two cross-validations were performed for comparison, and I have **chosen the 10-Folds** due to the size of my data.

b.1) 5-Folds

```
# c) Implementing k-fold cross-validation (# Using 5-fold)
# generalization performance.

from sklearn.model_selection import KFold
from sklearn.linear_model import LinearRegression

X= insurance_df_encoded.drop(columns='charges')
y= insurance_df_encoded['charges']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Assuming 'X' contains features and 'y' contains labels
kf = KFold(n_splits=5, shuffle=True, random_state=42)
model = LinearRegression()  # Increase max_iter if needed
mse_scores = []
for train_index, test_index in kf.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

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

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

    # Calculate Mean Squared Error (you can use other metrics)
    mse = mean_squared_error(y_test, y_pred)
    print(f'MSE: {mse}')
    # Append the result to the list
    mse_scores.append(mse)
    # ...

# Calculate the average performance metric across all folds
average_mse = np.mean(mse_scores)
# Calculate average performance metrics across all folds
# ...
print(f'Average Mean Squared Error across {num_folds}-fold cross-validation: {average_mse}')
```

```
MSE: 4748550.245079114
MSE: 2901006.2025891417
MSE: 3746578.7695778464
MSE: 5160353.178539603
MSE: 4811499.655379784
Average Mean Squared Error across 5-fold cross-validation: 4273597.610233097
```

b.2) 10- Folds

```
# Implementing k-fold cross-validation (Using 10-fold)

X= insurance_df_encoded.drop(columns='charges')
y= insurance_df_encoded['charges']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Assuming 'X' contains features and 'y' contains labels
kf = KFold(n_splits=10, shuffle=True, random_state=42)
model = LinearRegression()  # Increase max_iter if needed
mse_scores = []
for train_index, test_index in kf.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

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

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

    # Calculate Mean Squared Error (you can use other metrics)
    mse = mean_squared_error(y_test, y_pred)
    print(f'MSE: {mse}')
    # Append the result to the list
    mse_scores.append(mse)
    # ...

# Calculate the average performance metric across all folds
average_mse = np.mean(mse_scores)
# Calculate average performance metrics across all folds
# ...
print(f'Average Mean Squared Error across {num_folds}-fold cross-validation: {average_mse}')
```

```
MSE: 2993554.1246525864
MSE: 6541735.933885871
MSE: 2122792.9177777735
MSE: 3635388.4900862984
MSE: 5030623.194743144
MSE: 2502522.4683902874
MSE: 1560105.8331802967
MSE: 8871583.793375876
MSE: 5564163.3256392125
MSE: 4103540.685955899
Average Mean Squared Error across 5-fold cross-validation: 4292601.076768724
```

c) After fitting the data in the RandomSearchCV, I came up with the best hyperparameters ({'positive': False, 'n_jobs': -1, 'fit_intercept': False, 'copy_X': False}) which are helpful in building a robust data model in predicting outcomes.

```
In [29]: # c) Best performing Linear regression model based on the hyperparameter tuning and cross validation result.
         from sklearn.model_selection import RandomizedSearchCV

         X= insurance_df_encoded.drop(columns='charges')
         y= insurance_df_encoded['charges']

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

         # Define the hyperparameters and their possible values
         param_dist = {
             'fit_intercept': [False, True]
         }

         # Initialize the linear regression model
         model = LinearRegression()

         # Initialize the random search with cross-validation
         random_search = RandomizedSearchCV(
             model, param_distributions=param_dist, n_iter=4, cv=5, scoring='neg_mean_squared_error', random_state=42
         )

         # Perform the random search on the training data
         random_search.fit(X_train, y_train)

         # Get the best parameters
         best_params = random_search.best_params_

         print("Best Hyperparameters:", best_params)

         # Initialize the linear regression model
         final_model = LinearRegression(**best_params)
         # Train the model
         final_model.fit(X_train, y_train)


         # Make predictions on the test set
         final_y_pred = final_model.predict(X_test)

         # Calculate evaluation metrics
         mae = mean_absolute_error(y_test, final_y_pred)
         mse = mean_squared_error(y_test, final_y_pred)
         rmse = np.sqrt(mse)
         r2 = r2_score(y_test, final_y_pred)

         print(f"MAE: {mae:.2f}")
         print(f"MSE: {mse:.2f}")
         print(f"RMSE: {rmse:.2f}")
         print(f"R-squared: {r2:.2f}")

         Best Hyperparameters: {'fit_intercept': False}
         MAE: 944.09
         MSE: 4748550.25
         RMSE: 2179.12
         R-squared: 0.77

         C:\Users\torri\anaconda3\Lib\site-packages\sklearn\model_selection\_search.py:307: UserWarning: The total space of parameters 2
         is smaller than n_iter=4. Running 2 iterations. For exhaustive searches, use GridSearchCV.
           warnings.warn(

In [ ]:
```

Task 4 – Business Decision and Recommendations

The R-squared score indicates that 77% of the variance in the target variables is explained in the linear model. Generally, a higher R-squared suggests a better fit, depending on the context of the interpretation. I recommend continuously monitoring the model's performance and making improvements when necessary.

Part B – Classification

The dataset used is from Kaggle. It contains previous loan applicants who applied for a loan based on a land property. The goal is to create a model to predict whether an application will be approved or rejected. The variables include loan ID, gender, married (or not), number of dependents, education, self-employed, applicant income, co-applicant income, loan history, and loan status.

Note: The dataset is also available in my GitHub account via this link: https://github.com/Myres16/Data-Analytics-Assessments/tree/main/708

Task 1 – Data Preprocessing

The dataset was loaded in the Python notebook by importing pandas and two additional libraries for future use in data preparation.

```
import pandas as pd
```

```
C:\Users\torri\anaconda3\Lib\site-packages\pandas\core\arrays\masked.py:60: UserWarning: Pandas requires version '1.3.6' or new
er of 'bottleneck' (version '1.3.5' currently installed).
  from pandas.core import (
```

```
loan_df = pd.read_csv('loan_data.csv')
loan_df
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0
1	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0
2	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0
3	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0
4	LP001013	Male	Yes	0	Not Graduate	No	2333	1516.0	95.0	360.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...
376	LP002953	Male	Yes	3+	Graduate	No	5703	0.0	128.0	360.0	1.0
377	LP002974	Male	Yes	0	Graduate	No	3232	1950.0	108.0	360.0	1.0
378	LP002978	Female	No	0	Graduate	No	2900	0.0	71.0	360.0	1.0
379	LP002979	Male	Yes	3+	Graduate	No	4106	0.0	40.0	180.0	1.0
380	LP002990	Female	No	0	Graduate	Yes	4583	0.0	133.0	360.0	0.0

381 rows × 13 columns

a) Look for missing values.

```
loan_df.isnull().sum()
```

```
Loan_ID               0
Gender                5
Married               0
Dependents            8
Education             0
Self_Employed        21
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount            0
Loan_Amount_Term     11
Credit_History       30
Property_Area         0
Loan_Status           0
dtype: int64
```

Columns with missing values: Gender, Dependents, Self_Employed, Loan_Amount_Term, Credit_History

```
# Remove records which doesn't tell if they are self-employed or not
loan_df = loan_df.dropna(subset='Self_Employed', axis=0)
loan_df
```

Encode gender to integer so we can perform Imputation.

```
# Encode gender to int so we can perform Imputation on it
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
le.fit(loan_df['Gender'])

loan_df['Gender'] = le.transform(loan_df['Gender'])
loan_df['Gender'].fillna(loan_df['Gender'].mean(), inplace=True)
loan_df['Dependents'].fillna(0, inplace=True)
loan_df['Loan_Amount_Term'].fillna(loan_df['Loan_Amount_Term'].mean(), inplace=True)
loan_df['Credit_History'].fillna(loan_df['Credit_History'].mean(), inplace=True)
loan_df
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	LP001003	1	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0
1	LP001005	1	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0
2	LP001006	1	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0
3	LP001008	1	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0
4	LP001013	1	Yes	0	Not Graduate	No	2333	1516.0	95.0	360.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...
376	LP002953	1	Yes	3+	Graduate	No	5703	0.0	128.0	360.0	1.0
377	LP002974	1	Yes	0	Graduate	No	3232	1950.0	108.0	360.0	1.0
378	LP002978	0	No	0	Graduate	No	2900	0.0	71.0	360.0	1.0
379	LP002979	1	Yes	3+	Graduate	No	4106	0.0	40.0	180.0	1.0
380	LP002990	0	No	0	Graduate	Yes	4583	0.0	133.0	360.0	0.0

360 rows × 13 columns

Now, there is no more null data in the dataset.

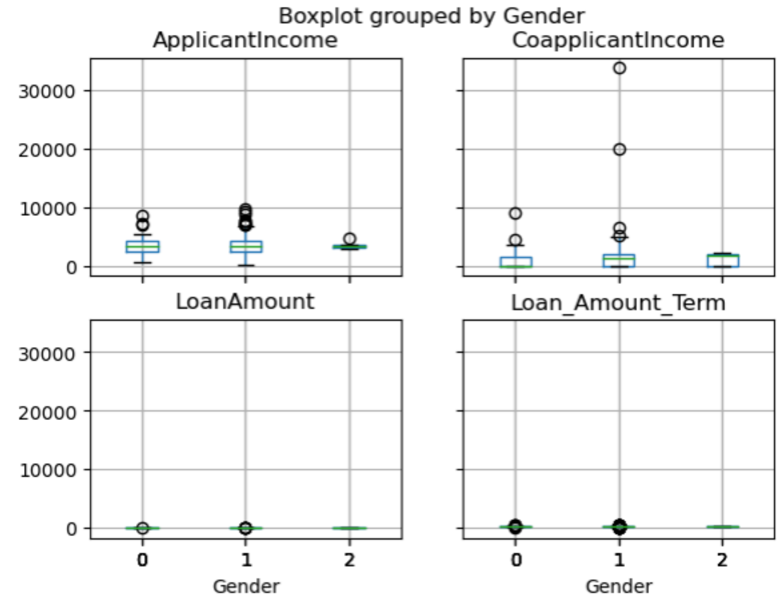
```
loan_df.isnull().sum()
```

```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

Removing outliers.

I used a boxplot to visualize outliers. I selected 4 numerical variables that can be a factor in predicting loan approval.

```
# Outliers and its visualization.
# Found 4 columns with outliers (ApplicantIncome, CoapplicantIncome, LoanAmount, and Loan_Amo
import matplotlib.pyplot as plt

loan_df.boxplot(column=['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Te
plt.show()
```



To manage the outliers for the 4 variables, I used the robust z-score to remove all the outliers. Initially, I used the z-score, but outliers were remaining.

I remove outliers by variables so as not to complicate and miss other outliers during the process.

Removing outliers for Applicant Income:

```
# Removing outliers
import pandas as pd
import numpy as np
from scipy import stats

def get_outliers(df, column):
    median_values = df[column].median()
    mad_charges = np.median(np.abs(df[column] - median_values))
    df['z_score'] = (df[column] - median_values) / mad_charges
    threshold = 2
    return (df['z_score'].abs() > threshold)

outliers = get_outliers(loan_df, 'ApplicantIncome')
# Remove outliers from the dataset
loan_df = loan_df[~outliers]
loan_df
```

Removing outliers for Coapplicant Income:

```
outliers = get_outliers(loan_df, 'CoapplicantIncome')
# Remove outliers from the dataset
loan_df = loan_df[~outliers]
loan_df
```

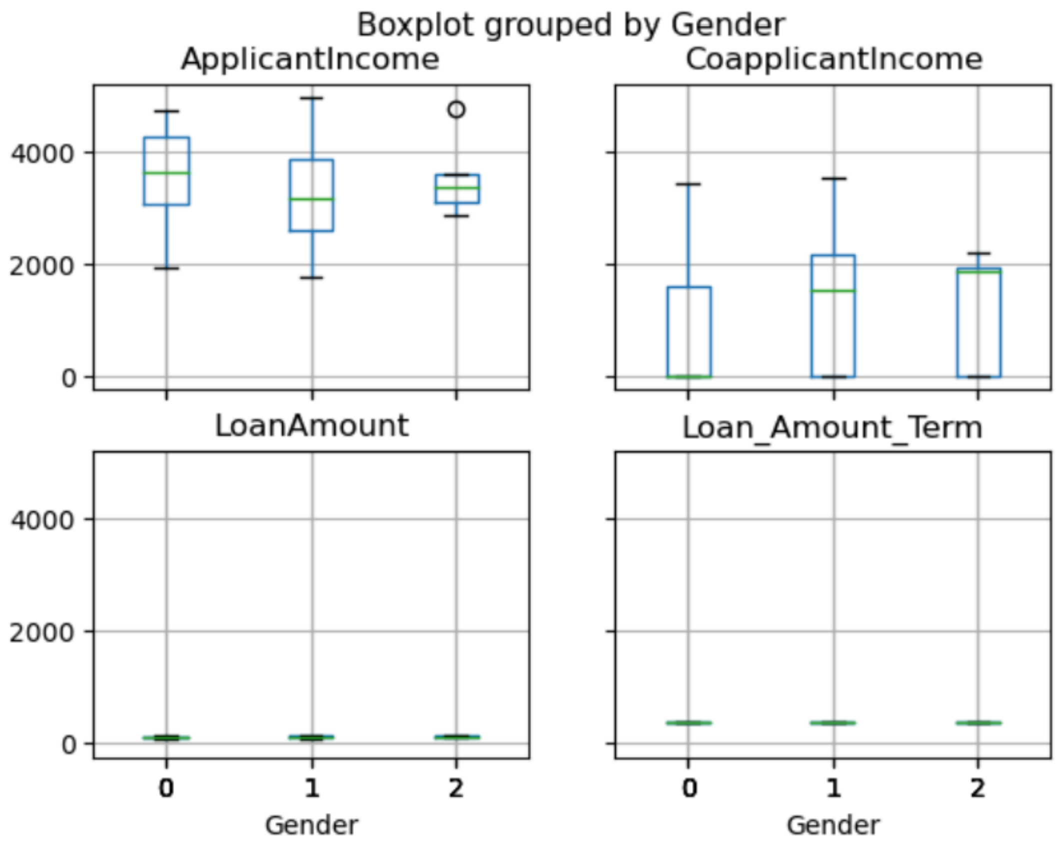
Removing outliers for Loan Amount

```
outliers = get_outliers(loan_df, 'LoanAmount')
# Remove outliers from the dataset
loan_df = loan_df[~outliers]
loan_df
```

Removing outliers for Loan Amount Term

```
outliers = get_outliers(loan_df, 'Loan_Amount_Term')
# Remove outliers from the dataset
loan_df = loan_df[~outliers]
loan_df
```

After removing outliers. Only a few outliers remain, but this is negligible.



Perform feature scaling or normalization – Using **Standard Scalar.**

Scaling and normalizing three variables that can be used for Model regression.

```
# Perform feature scaling or normalization. - Using StandardScaler
from sklearn.preprocessing import StandardScaler

scale = StandardScaler()
X = loan_df[['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount']]
loan_df[['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount']] = scale.fit_transform(X)
loan_df
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome
0	LP001003	1	Yes	1	Graduate	No	1.496168	0.315996
2	LP001006	1	Yes	0	Not Graduate	No	-0.899403	1.098083
4	LP001013	1	Yes	0	Not Graduate	No	-1.198850	0.323357
7	LP001029	1	No	0	Graduate	No	-1.773787	1.541572
9	LP001032	1	No	0	Graduate	No	1.935755	-1.071517
...	...	...	...	...	...	...	...	...
371	LP002926	1	Yes	2	Graduate	Yes	-0.728120	-1.071517
374	LP002940	1	No	0	Not Graduate	No	0.597829	-1.071517
375	LP002943	1	No	0	Graduate	No	-0.415498	-1.071517
377	LP002974	1	Yes	0	Graduate	No	-0.122040	0.722681
380	LP002990	0	No	0	Graduate	Yes	1.496168	-1.071517

197 rows × 14 columns

Encoding categorical variables

Using One-Hot to format categorical variables and create a new column for them.

```
# Encode categorical variables appropriately.
# using one-hot for categorical variables
categorical_variables = ['Married', 'Education', 'Self_Employed', 'Property_Area', 'Loan_Status']
loan_df = pd.get_dummies(loan_df, columns=categorical_variables, drop_first=True)
loan_df[
    ['Married_Yes',
     'Education_Not Graduate',
     'Self_Employed_Yes',
     'Property_Area_Semiurban',
     'Property_Area_Urban',
     'Loan_Status_Y']] = loan_df[
                            ['Married_Yes',
                             'Education_Not Graduate',
                             'Self_Employed_Yes',
                             'Property_Area_Semiurban',
                             'Property_Area_Urban',
                             'Loan_Status_Y']].replace({True: 1, False: 0})
loan_df
```

Credit_History	Married_Yes	Education_Not Graduate	Self_Employed_Yes	Property_Area_Semiurban	Property_Area_Urban	Loan_Status_Y
1.0	1	0	1	0	1	1
1.0	0	0	0	0	1	1
1.0	1	0	0	0	1	1
1.0	1	0	0	0	1	1
0.0	0	0	0	0	1	0
...	...	...	...	...	...	...
1.0	0	1	0	1	0	1
1.0	1	0	0	1	0	1
1.0	1	0	0	0	0	1
1.0	0	0	0	0	0	1
1.0	1	0	0	0	0	1

To prepare the dataset for modeling, we use train_test_split to group the data into one group of 20% of the total.

```
# Split the dataset into training and testing sets.
from sklearn.model_selection import train_test_split

X = loan_df[['ApplicantIncome', 'LoanAmount', 'CoapplicantIncome', 'Loan_Amount_Term', 'Credit_History']]
y = loan_df['Loan_Status_Y']

# Assuming 'X' contains features and 'y' contains labels
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print('Training')
print(X_train)
print('Testing')
print(X_test)
```

Task 2 – Model Building with Hyperparameter Tuning

a) I selected Logistic Regression as it is commonly used for Binary classification. The bank loan prediction predicts an approval or rejection status, and logistic regression is well-suited for binary classification tasks.

```
# a) Select an appropriate classification algorithm (e.g., Logistic Regression, Random Forest,
# Support Vector Machine) to predict the target categorical variable. Justify your choice.
# Using Logistic Regression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

X = loan_df[['ApplicantIncome', 'LoanAmount', 'CoapplicantIncome', 'Loan_Amount_Term', 'Credit_History']]
y = loan_df['Loan_Status_Y']

# Assuming 'X' contains features and 'y' contains labels
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

prediction = model.predict(X_test)
X_test['Prediction'] = prediction
X_test
#accuracy = accuracy_score(y_test, prediction)
#print(f"Prediction: {prediction}")
```

	ApplicantIncome	LoanAmount	CoapplicantIncome	Loan_Amount_Term	Credit_History	Prediction
182	-0.277708	1.017427	0.883055	360.0	1.000000	1
281	0.839336	1.225260	0.604594	360.0	1.000000	1
11	0.020033	-0.437399	-0.707394	360.0	0.000000	0
138	-0.348892	1.132890	1.129916	360.0	1.000000	1
82	-1.003924	-0.483584	0.562463	360.0	1.000000	1
378	-0.397489	-0.552861	-0.707394	360.0	1.000000	1
216	-0.210631	-0.483584	-0.707394	360.0	1.000000	1
30	0.315037	0.901965	0.830391	360.0	1.000000	1
285	-0.096325	1.271445	0.418296	360.0	0.000000	0
244	-0.785580	0.948150	0.883713	360.0	1.000000	1
58	0.347207	-1.037803	-0.707394	240.0	1.000000	1
366	-1.095642	-0.783786	-0.707394	360.0	0.830303	1
33	1.484100	-1.176358	-0.707394	360.0	1.000000	1
309	-0.715765	-0.460491	-0.707394	360.0	1.000000	1
130	-0.210631	0.971243	1.281325	360.0	1.000000	1
99	3.998819	-0.460491	-0.707394	180.0	1.000000	1
280	2.506004	0.994335	-0.707394	360.0	1.000000	1
1	-0.329043	-0.668324	-0.707394	360.0	1.000000	1
319	0.389644	0.994335	2.782903	360.0	0.830303	1
361	-0.269494	0.948150	0.747445	360.0	0.000000	0
209	-0.671274	-0.645231	-0.707394	360.0	1.000000	1
125	0.338309	-0.922341	0.958759	360.0	1.000000	1
298	-0.365319	-0.575954	0.388014	180.0	0.000000	0
168	-0.166825	-1.499653	-0.707394	360.0	1.000000	1

b) Performing GridSearchCV with the Logistic Regression - GridSearchCV will look for common and effective parameters to improve the performance of a Logistic Regression model.

```
# Implement hyperparameter tuning by conducting a grid search or random search to
# optimize model parameters. Clearly outline the hyperparameters you tuned and the
# rationale behind them.
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression


X = loan_df[['ApplicantIncome', 'LoanAmount', 'CoapplicantIncome', 'Loan_Amount_Term', 'Credit_History']]
y = loan_df['Loan_Status_Y']

# Define the hyperparameter grid
param_grid = {
    'C': [0.1, 1, 10],  # Regularization parameter
    'solver': ['lbfgs', 'liblinear', 'saga'],  # Solver options
    'max_iter': [100, 500, 1000]  # Maximum number of iterations
}

# Perform grid search with cross-validation
grid_search = GridSearchCV(LogisticRegression(), param_grid, cv=5, scoring='accuracy')
grid_search.fit(X, y)

# Get the best hyperparameters
best_params = grid_search.best_params_
print("Best hyperparameters:", best_params)
```

```
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
Best hyperparameters: {'C': 1, 'max_iter': 100, 'solver': 'lbfgs'}
```

c) Building the classification model for the training dataset

```
# Build the classification model using the training data. Explain the process and provide code snippets.
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X = loan_df[['ApplicantIncome', 'LoanAmount', 'CoapplicantIncome', 'Loan_Amount_Term', 'Credit_History']]
y = loan_df['Loan_Status_Y']

# Assuming 'X' contains features and 'y' contains labels
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create the logistic regression model
model = LogisticRegression(max_iter=1000)  # Increase max_iter if needed
model.fit(X_train, y_train)

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

# Evaluate model performance using accuracy score
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

✓ 0.0s

```
Accuracy: 0.88
```

First, we need to identify the feature variables that will be a factor in determining the outcome variable, the Loan_Status_Y.

We use the train_test_split to have an 80%/20% dataset. The 20% is the test data, and the rest are used for training.

Based on the result, it yielded 0.88 or 88% accuracy. This means that by entering new applications, my Model can predict and be accurate for 88%.

Task 3 – Model Evaluation and Selection

a) Using Random Forest Classifier to test for confusion matrix

```
# a) Calculate and analyse the confusion matrix for the model.
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
from sklearn.ensemble import RandomForestClassifier

X = loan_df[['ApplicantIncome', 'LoanAmount', 'CoapplicantIncome', 'Loan_Amount_Term', 'Credit_History']]
y = loan_df['Loan_Status_Y']

# Assuming 'X' contains features and 'y' contains labels
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create the logistic regression model
model = RandomForestClassifier()  # Increase max_iter if needed
model.fit(X_train, y_train)

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

# Assuming 'y_test' contains true labels and 'y_pred' contains predicted labels
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
```

✓ 0.0s

```
Confusion Matrix:
[[ 4  3]
 [ 1 16]]
```

b) Using the confusion matrix to get the ravel and compute the accuracy of the training

```
# b) Evaluate the performance of the classification model using appropriate metrics (e.g.,
# Accuracy, Precision, Recall, F1-score).
TP, TN, FP, FN = cm.ravel()
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1_score = f1_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-Score: {f1_score:.2f}")
```

✓ 0.0s

```
Accuracy: 0.83
Precision: 0.84
Recall: 0.94
F1-Score: 0.89
```

c) I used 5-fold to split the dataset into five and test its accuracy. This helps me fine-grain testing in a smaller group of datasets.

```
# c) Implement k-fold cross-validation (e.g., 5-fold or 10-fold) to assess the model's
# generalization performance.
# Using 5-fold
from sklearn.model_selection import KFold
from sklearn.linear_model import LogisticRegression

X = loan_df[['ApplicantIncome', 'LoanAmount', 'CoapplicantIncome', 'Loan_Amount_Term', 'Credit_History']]
y = loan_df['Loan_Status_Y']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Assuming 'X' contains features and 'y' contains labels
kf = KFold(n_splits=5, shuffle=True, random_state=42)
model = LogisticRegression(max_iter=1000)  # Increase max_iter if needed

for train_index, test_index in kf.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    
    # Evaluate model performance
    accuracy = accuracy_score(y_test, y_pred)

    print("Accuracy", accuracy)
    # Evaluate model performance (e.g., accuracy, precision, recall, F1-score)
    # ...

# Calculate average performance metrics across all folds
# ...
```

✓ 0.0s

```
Accuracy 0.875
Accuracy 0.6666666666666666
Accuracy 0.875
Accuracy 0.7916666666666666
Accuracy 0.8695652173913043
```

d) Although Logistic Regression is common in Classification, Random Forest Classifier performed better with less processing time. Like Logistic Regression, Random Forest Classifier also gave a high accuracy score and can provide a close-to-accurate prediction.

```
# d) Select the best-performing classification model based on hyperparameter tuning and
# cross-validation results and justify the choice of the selected model for its suitability in
# addressing the business problem.

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

X = loan_df[['ApplicantIncome', 'LoanAmount', 'CoapplicantIncome', 'Loan_Amount_Term', 'Credit_History']]
y = loan_df['Loan_Status_Y']

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

# Define hyperparameter grid for Random Forest
param_grid = {
    'n_estimators': [100, 200, 500],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5],
    # Add other hyperparameters as needed
}

# Create Random Forest model
rf_model = RandomForestClassifier()

# Perform grid search with cross-validation
grid_search = GridSearchCV(rf_model, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X, y)

# Get best hyperparameters
best_params = grid_search.best_params_

# Train final model with best hyperparameters
final_rf_model = RandomForestClassifier(**best_params)
final_rf_model.fit(X, y)

# Evaluate performance on test data
y_pred = final_rf_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-Score: {f1:.2f}")
```

✓ 13.9s

```
Accuracy: 0.96
Precision: 0.94
Recall: 1.00
F1-Score: 0.97
```

Task 4 – Business Decision and Recommendations

Banking analysts leverage the Random Forest Classifier to gain valuable insights into loan approval opportunities. The model provides high-confidence predictions by considering critical features such as applicant income, co-applicant income, credit history, loan term, and loan amount. These insights empower analysts to streamline the loan approval process, identify potential risks, and make informed decisions.

References

Kumar, A. N. (2021, January 27). Why Linear Regression is not suitable for classification?Analytics Vidhaya. Medium. https://medium.com/analytics-vidhya/why-linear-regression-is-not-suitable-for-classification-cd724dd61cb8#:~:text=There%20are%20two%20things%20that,new%20data%20points%20are%20added.

Sarangam, Ajay. (2021, March 04). Classification vs Regression: An Easy Guide in 6 Points. Unext. https://u-next.com/blogs/artificial-intelligence/classification-vs-regression/#:~:text=The%20key%20distinction%20between%20Classification,Spam%20or%20Not%20Spam%2C%20etc.

Miri, Choi. (n.d). Medical Cost Personal Datasets. Kaggle. https://www.kaggle.com/datasets/mirichoi0218/insurance

Jikadara, Bhavik. (n.d.) Loan Status Prediction. Kaggle. https://www.kaggle.com/datasets/bhavikjikadara/loan-status-prediction/data

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
C:\Users\torri\anaconda3\Lib\site-packages\pandas\core\arrays\masked.py:60: UserWarning: Pand
as requires version '1.3.6' or newer of 'bottleneck' (version '1.3.5' currently installed).
  from pandas.core import (
```

In [2]:
```
insurance_df = pd.read_csv('insurance.csv')
insurance_df
```

Out[2]:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520
...	...	...	...	...	...	...	...
1333	50	male	30.970	3	no	northwest	10600.54830
1334	18	female	31.920	0	no	northeast	2205.98080
1335	18	female	36.850	0	no	southeast	1629.83350
1336	21	female	25.800	0	no	southwest	2007.94500
1337	61	female	29.070	0	yes	northwest	29141.36030

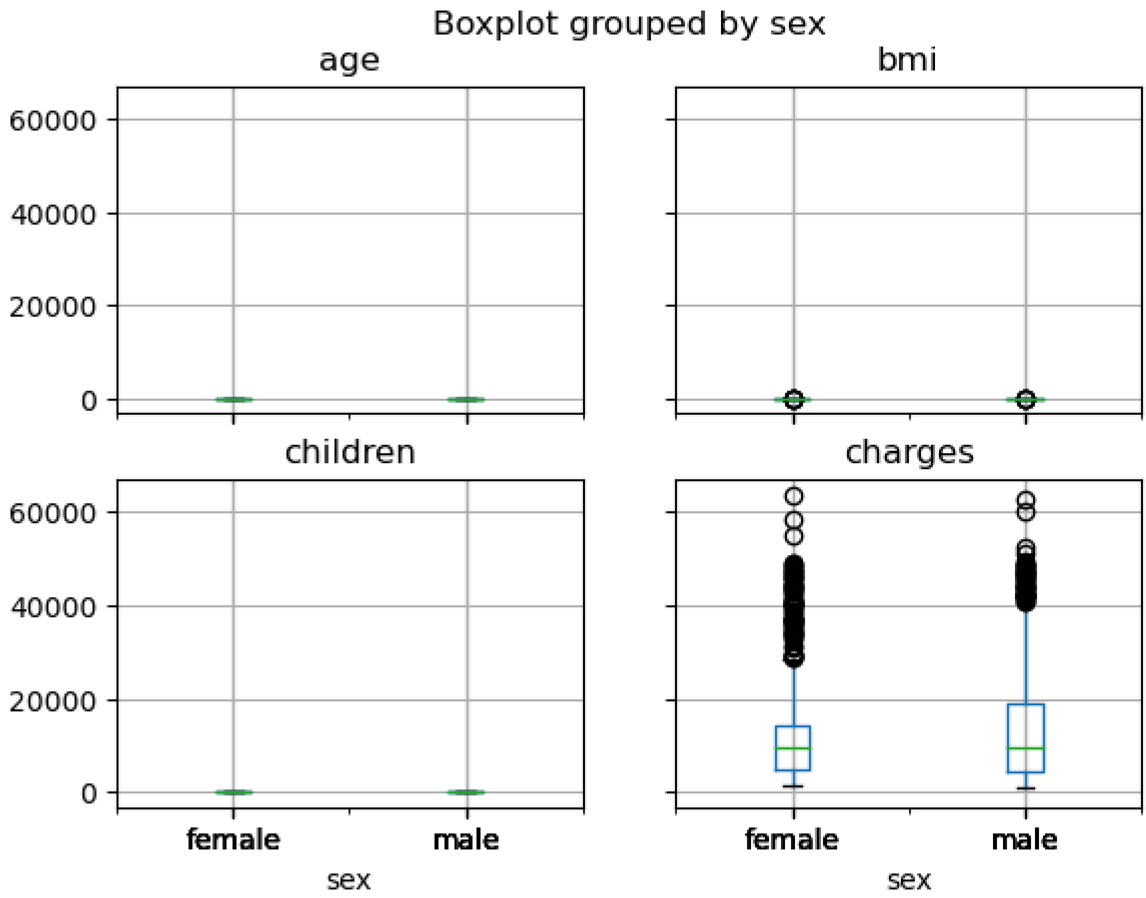
1338 rows × 7 columns

Task 1

In [3]:
```
# a)  Checking for missing values and outliers.
insurance_df.isnull().sum()
```

Out[3]:
```
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64
```

In [4]:
```
# Outliers and its visualization.
# Found 2 columns with outliers (bmi and charges)
insurance_df.boxplot(column=['age', 'bmi', 'children', 'charges'], by='sex')
plt.show()
```



In [5]:

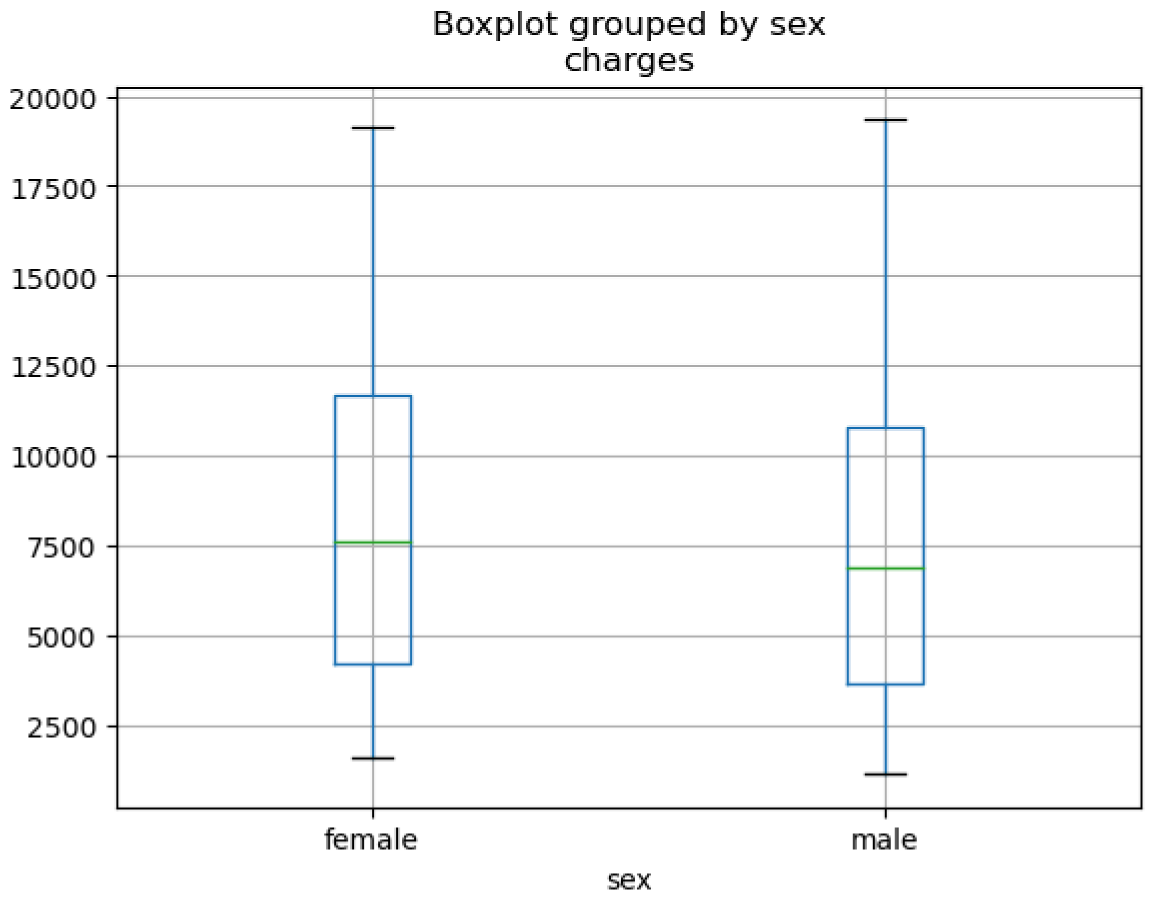
```
# Removing the outliers

def get_outliers(df, column):
    median_values = df[column].median()
    mad_charges = np.median(np.abs(df[column] - median_values))
    df['z_score'] = (df[column] - median_values) / mad_charges
    threshold = 2
    return (df['z_score'].abs() > threshold)

outliers = get_outliers(insurance_df, 'charges')
# Remove outliers from the dataset
insurance_df = insurance_df[~outliers]
insurance_df
```

Out[5]:

	age	sex	bmi	children	smoker	region	charges	z_score
0	19	female	27.90	0	yes	southwest	16884.9240	1.494970
1	18	male	33.77	1	no	southeast	1725.5523	-1.525573
2	28	male	33.00	3	no	southeast	4449.4620	-0.982827
4	32	male	28.88	0	no	northwest	3866.8552	-1.098913
5	31	female	25.74	0	no	southeast	3756.6216	-1.120877
...	...	...	...	...	...	...	...	...
1332	52	female	44.70	3	no	southwest	11411.6850	0.404413
1333	50	male	30.97	3	no	northwest	10600.5483	0.242792
1334	18	female	31.92	0	no	northeast	2205.9808	-1.429846
1335	18	female	36.85	0	no	southeast	1629.8335	-1.544645
1336	21	female	25.80	0	no	southwest	2007.9450	-1.469306

1052 rows × 8 columns

In [6]:
```python
# Checking if the outliers were removed.
insurance_df.boxplot(column=['charges'], by='sex')
plt.show()
```



In [7]:
```python
# Identify outliers
outliers = get_outliers(insurance_df, 'bmi')

# Remove outliers from the dataset
insurance_df = insurance_df[~outliers]
insurance_df
```

```
C:\Users\torri\AppData\Local\Temp\ipykernel_28188\3987391177.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guid
e/indexing.html#returning-a-view-versus-a-copy
  df['z_score'] = (df[column] - median_values) / mad_charges
```

Out[7]:

	age	sex	bmi	children	smoker	region	charges	z_score
0	19	female	27.90	0	yes	southwest	16884.92400	-0.513382
1	18	male	33.77	1	no	southeast	1725.55230	0.914842
2	28	male	33.00	3	no	southeast	4449.46200	0.727494
4	32	male	28.88	0	no	northwest	3866.85520	-0.274939
5	31	female	25.74	0	no	southeast	3756.62160	-1.038929
...	...	...	...	...	...	...	...	...
1331	23	female	33.40	0	no	southwest	10795.93733	0.824818
1333	50	male	30.97	3	no	northwest	10600.54830	0.233577
1334	18	female	31.92	0	no	northeast	2205.98080	0.464720
1335	18	female	36.85	0	no	southeast	1629.83350	1.664234
1336	21	female	25.80	0	no	southwest	2007.94500	-1.024331

856 rows × 8 columns

In [8]:

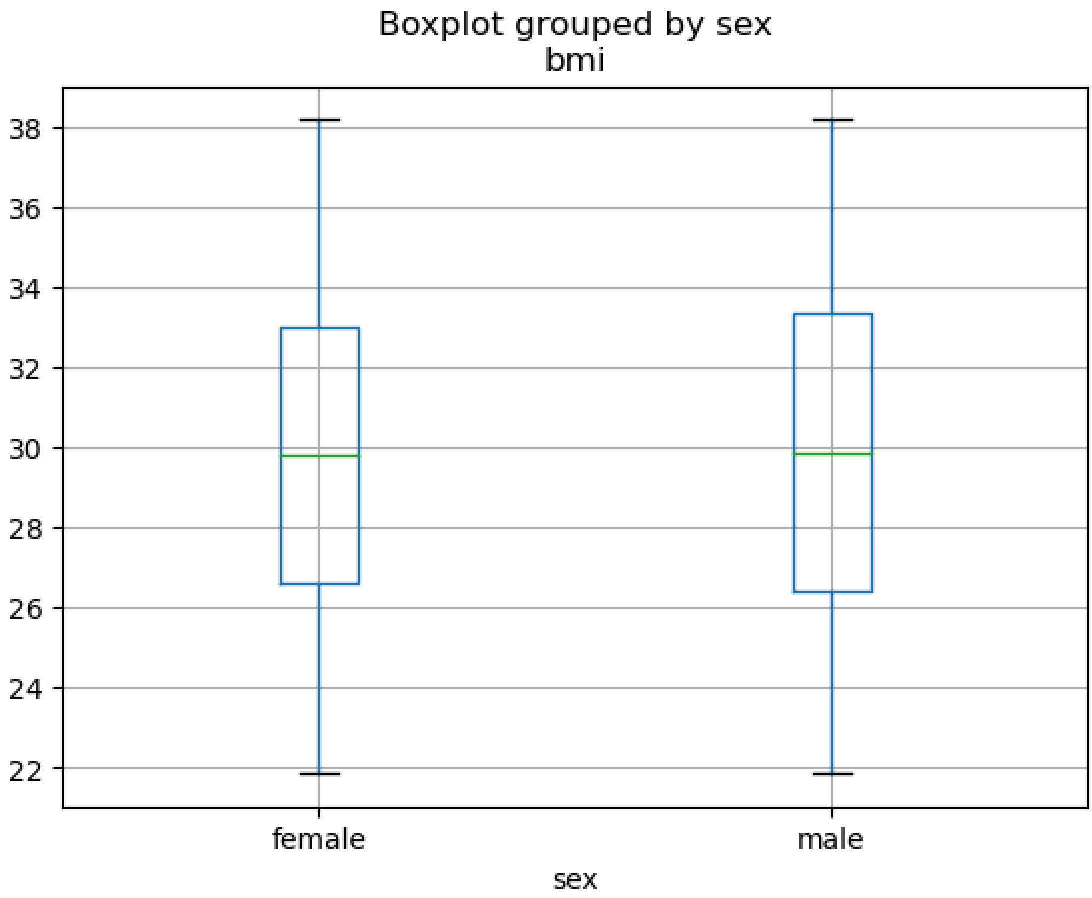
```
insurance_df.describe()
```

Out[8]:

	age	bmi	children	charges	z_score
count	856.000000	856.000000	856.000000	856.000000	856.000000
mean	38.471963	29.858884	1.098131	7798.805491	-0.036768
std	13.849033	4.191816	1.233906	4530.721669	1.019907
min	18.000000	21.850000	0.000000	1121.873900	-1.985401
25%	26.000000	26.577500	0.000000	4021.162025	-0.835158
50%	38.000000	29.830000	1.000000	7263.721750	-0.043796
75%	50.000000	33.155000	2.000000	11170.977413	0.765207
max	64.000000	38.190000	5.000000	19361.998800	1.990268

In [9]:

```
# Running boxplot to check if outliers are removed
insurance_df.boxplot(column=['bmi'], by='sex')
plt.show()
```



In [10]:
```
# b) Feature scaling

from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Min-Max Scaling
min_max_scaler = MinMaxScaler()
insurance_df[['age','bmi','children']] = min_max_scaler.fit_transform(insurance_df[['age','bm
insurance_df
```

```
C:\Users\torri\AppData\Local\Temp\ipykernel_28188\1931882604.py:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guid
e/indexing.html#returning-a-view-versus-a-copy
  insurance_df[['age','bmi','children']] = min_max_scaler.fit_transform(insurance_df[['ag
e','bmi','children']])
```

Out[10]:

	age	sex	bmi	children	smoker	region	charges	z_score
0	0.021739	female	0.370257	0.0	yes	southwest	16884.92400	-0.513382
1	0.000000	male	0.729498	0.2	no	southeast	1725.55230	0.914842
2	0.217391	male	0.682375	0.6	no	southeast	4449.46200	0.727494
4	0.304348	male	0.430233	0.0	no	northwest	3866.85520	-0.274939
5	0.282609	female	0.238066	0.0	no	southeast	3756.62160	-1.038929
...	...	...	...	...	...	...	...	...
1331	0.108696	female	0.706854	0.0	no	southwest	10795.93733	0.824818
1333	0.695652	male	0.558140	0.6	no	northwest	10600.54830	0.233577
1334	0.000000	female	0.616279	0.0	no	northeast	2205.98080	0.464720
1335	0.000000	female	0.917993	0.0	no	southeast	1629.83350	1.664234
1336	0.065217	female	0.241738	0.0	no	southwest	2007.94500	-1.024331

856 rows × 8 columns

In [11]:

```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
insurance_df['smoker'] = label_encoder.fit_transform(insurance_df['smoker'])
insurance_df
```

```
C:\Users\torri\AppData\Local\Temp\ipykernel_28188\373363711.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guid
e/indexing.html#returning-a-view-versus-a-copy
  insurance_df['smoker'] = label_encoder.fit_transform(insurance_df['smoker'])
```

Out[11]:

	age	sex	bmi	children	smoker	region	charges	z_score
0	0.021739	female	0.370257	0.0	1	southwest	16884.92400	-0.513382
1	0.000000	male	0.729498	0.2	0	southeast	1725.55230	0.914842
2	0.217391	male	0.682375	0.6	0	southeast	4449.46200	0.727494
4	0.304348	male	0.430233	0.0	0	northwest	3866.85520	-0.274939
5	0.282609	female	0.238066	0.0	0	southeast	3756.62160	-1.038929
...	...	...	...	...	...	...	...	...
1331	0.108696	female	0.706854	0.0	0	southwest	10795.93733	0.824818
1333	0.695652	male	0.558140	0.6	0	northwest	10600.54830	0.233577
1334	0.000000	female	0.616279	0.0	0	northeast	2205.98080	0.464720
1335	0.000000	female	0.917993	0.0	0	southeast	1629.83350	1.664234
1336	0.065217	female	0.241738	0.0	0	southwest	2007.94500	-1.024331

856 rows × 8 columns

In [12]:

```
# c) Encoding the categorical values

insurance_df_encoded = pd.get_dummies(insurance_df, columns=['sex', 'region'], drop_first=Tru
insurance_df_encoded
```

Out[12]:

	age	bmi	children	smoker	charges	z_score	sex_male	region_northwest	region_southe
0	0.021739	0.370257	0.0	1	16884.92400	-0.513382	False	False	Fa
1	0.000000	0.729498	0.2	0	1725.55230	0.914842	True	False	T
2	0.217391	0.682375	0.6	0	4449.46200	0.727494	True	False	T
4	0.304348	0.430233	0.0	0	3866.85520	-0.274939	True	True	Fa
5	0.282609	0.238066	0.0	0	3756.62160	-1.038929	False	False	T
...	...	...	...	...	...	...	...	...	
1331	0.108696	0.706854	0.0	0	10795.93733	0.824818	False	False	Fa
1333	0.695652	0.558140	0.6	0	10600.54830	0.233577	True	True	Fa
1334	0.000000	0.616279	0.0	0	2205.98080	0.464720	False	False	Fa
1335	0.000000	0.917993	0.0	0	1629.83350	1.664234	False	False	T
1336	0.065217	0.241738	0.0	0	2007.94500	-1.024331	False	False	Fa

856 rows × 10 columns

In [13]:

```
insurance_df_encoded[['sex_male', 'region_northwest', 'region_southeast', 'region_southwest']
insurance_df_encoded
```

```
C:\Users\torri\AppData\Local\Temp\ipykernel_28188\182238360.py:1: FutureWarning: Downcasting
behavior in `replace` is deprecated and will be removed in a future version. To retain the ol
d behavior, explicitly call `result.infer_objects(copy=False)`. To opt-in to the future behav
ior, set `pd.set_option('future.no_silent_downcasting', True)`
  insurance_df_encoded[['sex_male', 'region_northwest', 'region_southeast', 'region_southwes
t']] = insurance_df_encoded[['sex_male', 'region_northwest', 'region_southeast', 'region_sout
hwest']].replace({True: 1, False: 0})
```

Out[13]:

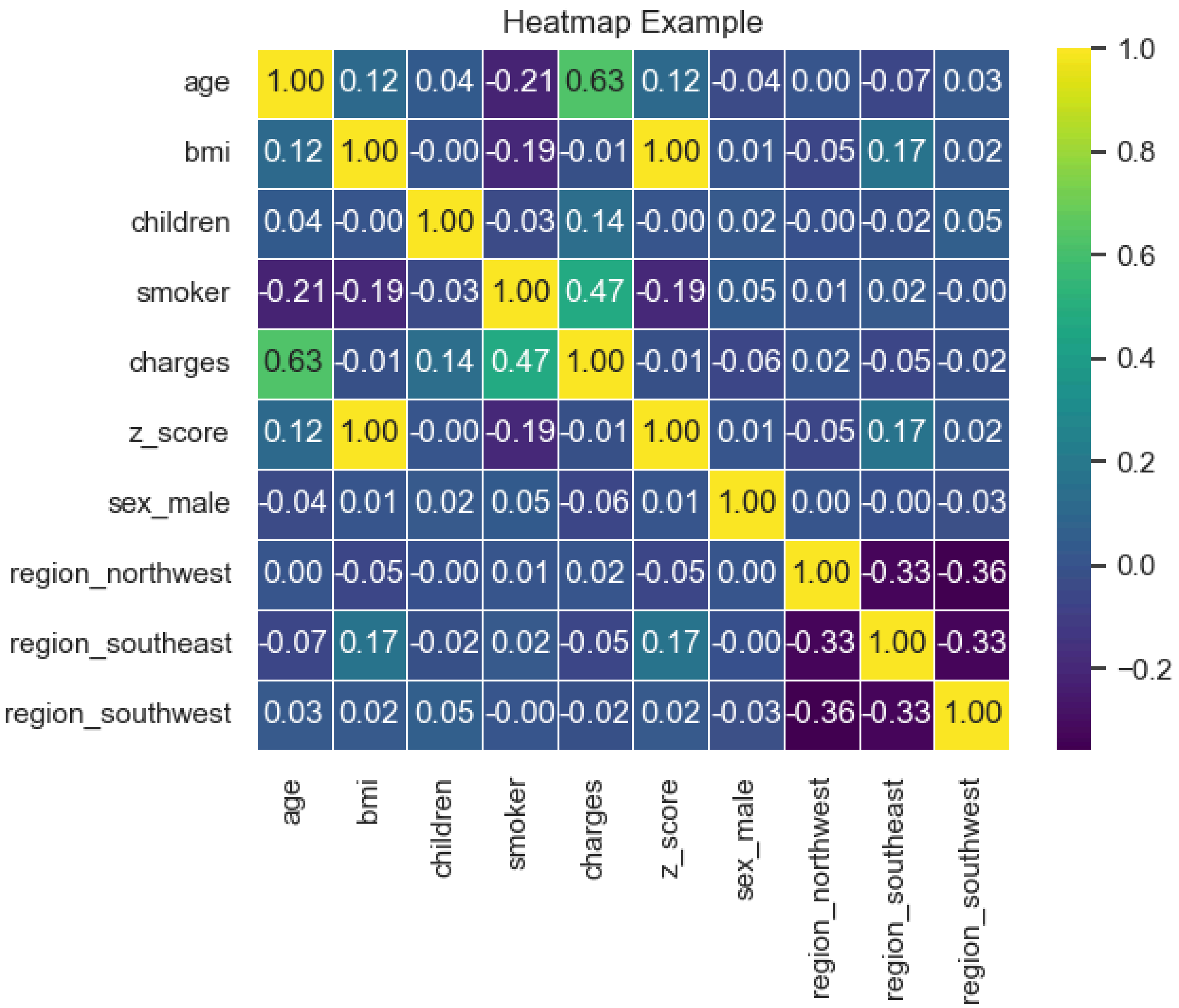
	age	bmi	children	smoker	charges	z_score	sex_male	region_northwest	region_southe
0	0.021739	0.370257	0.0	1	16884.92400	-0.513382	0	0	
1	0.000000	0.729498	0.2	0	1725.55230	0.914842	1	0	
2	0.217391	0.682375	0.6	0	4449.46200	0.727494	1	0	
4	0.304348	0.430233	0.0	0	3866.85520	-0.274939	1	1	
5	0.282609	0.238066	0.0	0	3756.62160	-1.038929	0	0	
...	...	...	...	...	...	...	...	...	
1331	0.108696	0.706854	0.0	0	10795.93733	0.824818	0	0	
1333	0.695652	0.558140	0.6	0	10600.54830	0.233577	1	1	
1334	0.000000	0.616279	0.0	0	2205.98080	0.464720	0	0	
1335	0.000000	0.917993	0.0	0	1629.83350	1.664234	0	0	
1336	0.065217	0.241738	0.0	0	2007.94500	-1.024331	0	0	

856 rows × 10 columns

In [14]:

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

# Create a heatmap using Seaborn
sns.set()
sns.heatmap(insurance_df_encoded.corr(), annot=True, cmap='viridis', fmt=".2f", linewidths=.5
```

```python
plt.title('Heatmap Example')
plt.show()
```



In [15]:

```python
# d) Splitting the data to testing and training

from sklearn.model_selection import train_test_split

X = insurance_df_encoded.drop(columns='charges')
y = insurance_df_encoded['charges']

# Splitting the dataset into training (80%) and testing(20%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("Training set - Features:", X_train.shape, "Target:", y_train.shape)
print("Testing set - Features:", X_test.shape, "Target:", y_test.shape)
```

```
Training set - Features: (684, 9) Target: (684,)
Testing set - Features: (172, 9) Target: (172,)
```

Task 2

In [16]:

```python
# a) Linear Regression Model

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

model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)
```

```python
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
print("R^2 Score:", r2)
```

```
Mean Squared Error: 4748550.245079115
Root Mean Squared Error: 2179.116849799275
R^2 Score: 0.774186649869275
```

In [17]:

```python
# b) Hyper-parameter tuning using Random Search

from sklearn.model_selection import RandomizedSearchCV

# Define the hyperparameters and their possible values
param_dist = {
    'fit_intercept': [True, False]
}

# Initialize the linear regression model
model = LinearRegression()

# Initialize the random search with cross-validation
random_search = RandomizedSearchCV(
    model, param_distributions=param_dist, n_iter=4, cv=5, scoring='neg_mean_squared_error',
)

# Perform the random search on the training data
random_search.fit(X_train, y_train)

# Get the best parameters
best_params = random_search.best_params_

print("Best Hyperparameters:", best_params)
```

```
Best Hyperparameters: {'fit_intercept': False}
C:\Users\torri\anaconda3\Lib\site-packages\sklearn\model_selection\_search.py:307: UserWarnin
g: The total space of parameters 2 is smaller than n_iter=4. Running 2 iterations. For exhaus
tive searches, use GridSearchCV.
  warnings.warn(
```

In [18]:

```python
# Performing a new linear regression model with the best hyperparameter fit intercept: False

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

# Create a new linear regression model with the best hyperparameters
best_model = LinearRegression()

# Train the model on the training set
best_model.fit(X_train, y_train)

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

# Evaluate the model performance
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error on Test Set:", mse)
```

```
Mean Squared Error on Test Set: 4748550.245079115
```

In [19]:

```python
# c) Building a linear regression model

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

#np.random.seed(42)
# Data
X= insurance_df_encoded.drop(columns='charges')
y= insurance_df_encoded['charges']
```

```
#split into training and test
X_train, X_test, y_train,y_test = train_test_split(X,y,test_size=0.20, random_state=42)


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

Out[19]:

```
▾ LinearRegression
LinearRegression()
```

Task 3

In [20]:

```
# a) Evaluation of regression model

#np.random.seed(42)
# Data
X= insurance_df_encoded.drop(columns='charges')
y= insurance_df_encoded['charges']

#split into training and test
X_train, X_test, y_train,y_test = train_test_split(X,y, test_size=0.2, random_state=42)


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

y_pred = model.predict(X_test)

# Calculate evaluation metrics
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print(f"MAE: {mae:.2f}")
print(f"MSE: {mse:.2f}")
print(f"RMSE: {rmse:.2f}")
print(f"R-squared: {r2:.2f}")
```

```
MAE: 944.09
MSE: 4748550.25
RMSE: 2179.12
R-squared: 0.77
```

In [22]:

```
# b) Implementing k-fold cross-validation (# Using 5-fold)
# generalization performance.

from sklearn.model_selection import KFold
from sklearn.linear_model import LinearRegression

X= insurance_df_encoded.drop(columns='charges')
y= insurance_df_encoded['charges']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Assuming 'X' contains features and 'y' contains labels
kf = KFold(n_splits=5, shuffle=True, random_state=42)
model = LinearRegression()  # Increase max_iter if needed
mse_scores = []
for train_index, test_index in kf.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
```

```python
    # Train the model
    model.fit(X_train, y_train)

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

    # Calculate Mean Squared Error (you can use other metrics)
    mse = mean_squared_error(y_test, y_pred)
    print(f'MSE: {mse}')
    # Append the result to the list
    mse_scores.append(mse)
    # ...

# Calculate the average performance metric across all folds
average_mse = np.mean(mse_scores)
# Calculate average performance metrics across all folds
# ...
print(f'Average Mean Squared Error across {num_folds}-fold cross-validation: {average_mse}')
```

```
MSE: 4748550.245079114
MSE: 2901006.2025891417
MSE: 3746578.7695778464
MSE: 5160353.178539603
MSE: 4811499.655379784
Average Mean Squared Error across 5-fold cross-validation: 4273597.610233097
```

In [23]:

```python
# Implementing k-fold cross-validation (Using 10-fold)

X= insurance_df_encoded.drop(columns='charges')
y= insurance_df_encoded['charges']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Assuming 'X' contains features and 'y' contains labels
kf = KFold(n_splits=10, shuffle=True, random_state=42)
model = LinearRegression()  # Increase max_iter if needed
mse_scores = []
for train_index, test_index in kf.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

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

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

    # Calculate Mean Squared Error (you can use other metrics)
    mse = mean_squared_error(y_test, y_pred)
    print(f'MSE: {mse}')
    # Append the result to the list
    mse_scores.append(mse)
    # ...

# Calculate the average performance metric across all folds
average_mse = np.mean(mse_scores)
# Calculate average performance metrics across all folds
# ...
print(f'Average Mean Squared Error across {num_folds}-fold cross-validation: {average_mse}')
```

```
MSE: 2993554.1246525864
MSE: 6541735.933885871
MSE: 2122792.9177777735
MSE: 3635388.4900862984
MSE: 5030623.194743144
MSE: 2502522.4683902874
MSE: 1560105.8331802967
MSE: 8871583.793375876
MSE: 5564163.3256392125
MSE: 4103540.685955899
Average Mean Squared Error across 5-fold cross-validation: 4292601.076768724
```

In [31]:

```
# c) Best performing Linear regression model based on the hyperparameter tuning and cross val
from sklearn.model_selection import RandomizedSearchCV

X= insurance_df_encoded.drop(columns='charges')
y= insurance_df_encoded['charges']

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

# Define the hyperparameters and their possible values
param_dist = {
    'fit_intercept': [False, True],
    'copy_X' : [True, False],
    'n_jobs': [-1, None, 5, 10],
    'positive': [True, False]
}

# Initialize the linear regression model
model = LinearRegression()

# Initialize the random search with cross-validation
random_search = RandomizedSearchCV(
    model, param_distributions=param_dist, n_iter=4, cv=5, scoring='neg_mean_squared_error',
)

# Perform the random search on the training data
random_search.fit(X_train, y_train)

# Get the best parameters
best_params = random_search.best_params_

print("Best Hyperparameters:", best_params)

# Initialize the linear regression model
final_model = LinearRegression(**best_params)
# Train the model
final_model.fit(X_train, y_train)


# Make predictions on the test set
final_y_pred = final_model.predict(X_test)

# Calculate evaluation metrics
mae = mean_absolute_error(y_test, final_y_pred)
mse = mean_squared_error(y_test, final_y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, final_y_pred)

print(f"MAE: {mae:.2f}")
print(f"MSE: {mse:.2f}")
print(f"RMSE: {rmse:.2f}")
print(f"R-squared: {r2:.2f}")
```

```
Best Hyperparameters: {'positive': False, 'n_jobs': -1, 'fit_intercept': False, 'copy_X': Fal
se}
```

In [ ]:

In [1]:
```
import pandas as pd
```

```
C:\Users\torri\anaconda3\Lib\site-packages\pandas\core\arrays\masked.py:60: UserWarn
ing: Pandas requires version '1.3.6' or newer of 'bottleneck' (version '1.3.5' curre
ntly installed).
  from pandas.core import (
```

In [2]:
```
loan_df = pd.read_csv('loan_data.csv')
loan_df
```

Out[2]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantInc
0	LP001003	Male	Yes	1	Graduate	No	4583	15
1	LP001005	Male	Yes	0	Graduate	Yes	3000	
2	LP001006	Male	Yes	0	Not Graduate	No	2583	23
3	LP001008	Male	No	0	Graduate	No	6000	
4	LP001013	Male	Yes	0	Not Graduate	No	2333	15
...	...	...	...	...	...	...	...	
376	LP002953	Male	Yes	3+	Graduate	No	5703	
377	LP002974	Male	Yes	0	Graduate	No	3232	19
378	LP002978	Female	No	0	Graduate	No	2900	
379	LP002979	Male	Yes	3+	Graduate	No	4106	
380	LP002990	Female	No	0	Graduate	Yes	4583	

381 rows × 13 columns

In [3]:
```
# Clean the dataset by handling missing values and removing outliers as needed.
# Look for missing values
loan_df.isnull().sum()
```

Out[3]:
```
Loan_ID               0
Gender                5
Married               0
Dependents            8
Education             0
Self_Employed        21
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount            0
Loan_Amount_Term     11
Credit_History       30
Property_Area         0
Loan_Status           0
dtype: int64
```

In [4]:
```python
# Remove records which doesn't tell if they are self-employed or not
loan_df = loan_df.dropna(subset='Self_Employed', axis=0)
loan_df
```

Out[4]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantInc
0	LP001003	Male	Yes	1	Graduate	No	4583	15
1	LP001005	Male	Yes	0	Graduate	Yes	3000	
2	LP001006	Male	Yes	0	Not Graduate	No	2583	23
3	LP001008	Male	No	0	Graduate	No	6000	
4	LP001013	Male	Yes	0	Not Graduate	No	2333	15
...	...	...	...	...	...	...	...	
376	LP002953	Male	Yes	3+	Graduate	No	5703	
377	LP002974	Male	Yes	0	Graduate	No	3232	19
378	LP002978	Female	No	0	Graduate	No	2900	
379	LP002979	Male	Yes	3+	Graduate	No	4106	
380	LP002990	Female	No	0	Graduate	Yes	4583	

360 rows × 13 columns

In [5]:
```
# Encode gender to int so we can perform Imputation on it
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
le.fit(loan_df['Gender'])

loan_df['Gender'] = le.transform(loan_df['Gender'])
loan_df['Gender'].fillna(loan_df['Gender'].mean(), inplace=True)
loan_df['Dependents'].fillna(0, inplace=True)
loan_df['Loan_Amount_Term'].fillna(loan_df['Loan_Amount_Term'].mean(), inplace=True)
loan_df['Credit_History'].fillna(loan_df['Credit_History'].mean(), inplace=True)
loan_df
```

```
C:\Users\torri\AppData\Local\Temp\ipykernel_23244\2496849096.py:6: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pa
ndas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  loan_df['Gender'] = le.transform(loan_df['Gender'])
C:\Users\torri\AppData\Local\Temp\ipykernel_23244\2496849096.py:7: FutureWarning: A
value is trying to be set on a copy of a DataFrame or Series through chained assignm
ent using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  loan_df['Gender'].fillna(loan_df['Gender'].mean(), inplace=True)
C:\Users\torri\AppData\Local\Temp\ipykernel_23244\2496849096.py:7: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pa
ndas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  loan_df['Gender'].fillna(loan_df['Gender'].mean(), inplace=True)
C:\Users\torri\AppData\Local\Temp\ipykernel_23244\2496849096.py:8: FutureWarning: A
value is trying to be set on a copy of a DataFrame or Series through chained assignm
ent using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  loan_df['Dependents'].fillna(0, inplace=True)
C:\Users\torri\AppData\Local\Temp\ipykernel_23244\2496849096.py:8: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pa
ndas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  loan_df['Dependents'].fillna(0, inplace=True)
C:\Users\torri\AppData\Local\Temp\ipykernel_23244\2496849096.py:9: FutureWarning: A
value is trying to be set on a copy of a DataFrame or Series through chained assignm
ent using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  loan_df['Loan_Amount_Term'].fillna(loan_df['Loan_Amount_Term'].mean(), inplace=Tru
e)
C:\Users\torri\AppData\Local\Temp\ipykernel_23244\2496849096.py:9: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pa
ndas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  loan_df['Loan_Amount_Term'].fillna(loan_df['Loan_Amount_Term'].mean(), inplace=Tru
e)
C:\Users\torri\AppData\Local\Temp\ipykernel_23244\2496849096.py:10: FutureWarning: A
value is trying to be set on a copy of a DataFrame or Series through chained assignm
ent using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  loan_df['Credit_History'].fillna(loan_df['Credit_History'].mean(), inplace=True)
C:\Users\torri\AppData\Local\Temp\ipykernel_23244\2496849096.py:10: SettingWithCopyW
arning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pa
ndas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  loan_df['Credit_History'].fillna(loan_df['Credit_History'].mean(), inplace=True)
```

Out[5]:

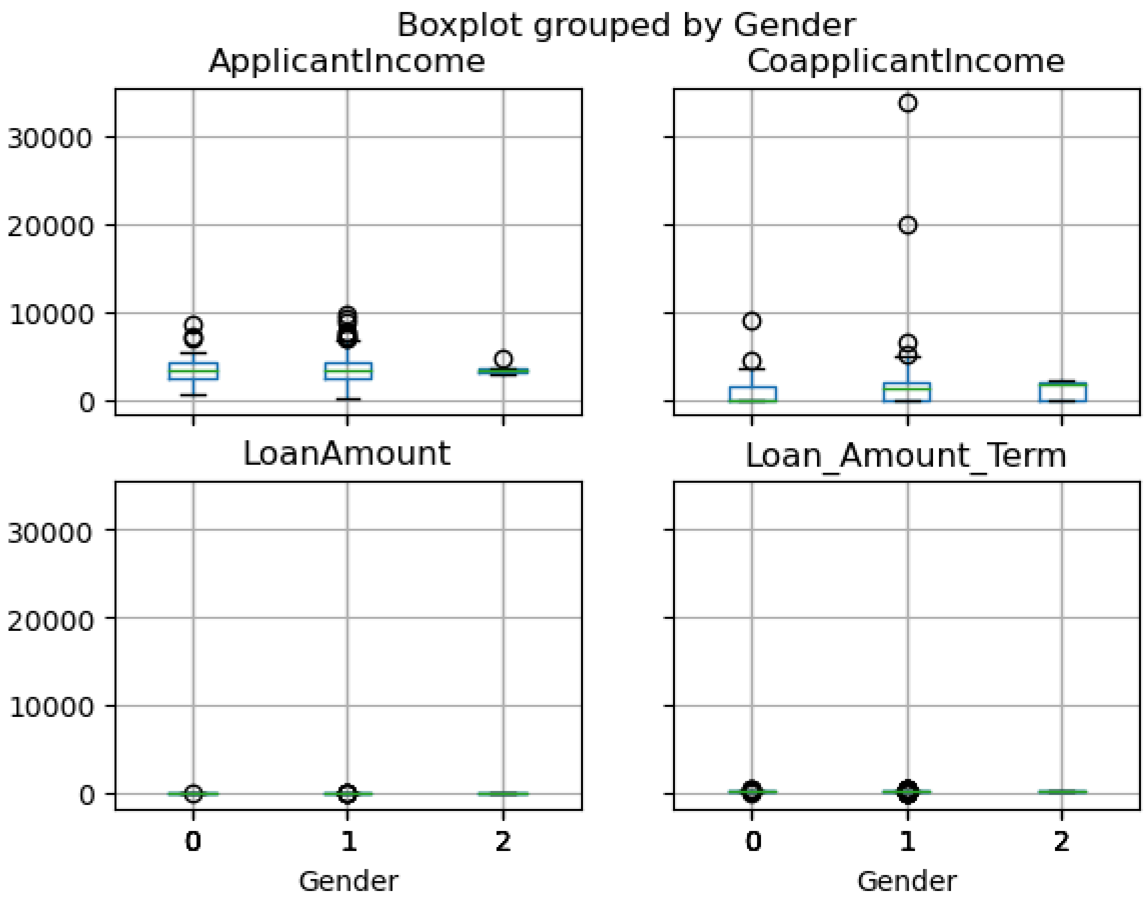
	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantInc
0	LP001003	1	Yes	1	Graduate	No	4583	15
1	LP001005	1	Yes	0	Graduate	Yes	3000	
2	LP001006	1	Yes	0	Not Graduate	No	2583	23
3	LP001008	1	No	0	Graduate	No	6000	
4	LP001013	1	Yes	0	Not Graduate	No	2333	15
...	...	...	...	...	...	...	...	
376	LP002953	1	Yes	3+	Graduate	No	5703	
377	LP002974	1	Yes	0	Graduate	No	3232	19
378	LP002978	0	No	0	Graduate	No	2900	
379	LP002979	1	Yes	3+	Graduate	No	4106	
380	LP002990	0	No	0	Graduate	Yes	4583	

360 rows × 13 columns

In [6]:
```
loan_df.isnull().sum()
```

Out[6]:
```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

In [7]:
```
# Outliers and its visualization.
# Found 4 columns with outliers (ApplicantIncome, CoapplicantIncome, LoanAmount, and L
import matplotlib.pyplot as plt

loan_df.boxplot(column=['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', 'Loan_Am
plt.show()
```



In [8]:

```python
# Removing outliers
import pandas as pd
import numpy as np
from scipy import stats

def get_outliers(df, column):
    median_values = df[column].median()
    mad_charges = np.median(np.abs(df[column] - median_values))
    df['z_score'] = (df[column] - median_values) / mad_charges
    threshold = 2
    return (df['z_score'].abs() > threshold)

outliers = get_outliers(loan_df, 'ApplicantIncome')
# Remove outliers from the dataset
loan_df = loan_df[~outliers]
loan_df
```

```
C:\Users\torri\AppData\Local\Temp\ipykernel_23244\1167691856.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df['z_score'] = (df[column] - median_values) / mad_charges
```

Out[8]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantInc
0	LP001003	1	Yes	1	Graduate	No	4583	15
1	LP001005	1	Yes	0	Graduate	Yes	3000	
2	LP001006	1	Yes	0	Not Graduate	No	2583	23
4	LP001013	1	Yes	0	Not Graduate	No	2333	15
5	LP001024	1	Yes	2	Graduate	No	3200	7
...	...	...	...	...	...	...	...	
375	LP002943	1	No	0	Graduate	No	2987	
377	LP002974	1	Yes	0	Graduate	No	3232	19
378	LP002978	0	No	0	Graduate	No	2900	
379	LP002979	1	Yes	3+	Graduate	No	4106	
380	LP002990	0	No	0	Graduate	Yes	4583	

304 rows × 14 columns

In [9]:
```python
outliers = get_outliers(loan_df, 'CoapplicantIncome')
# Remove outliers from the dataset
loan_df = loan_df[~outliers]
loan_df
```

```
C:\Users\torri\AppData\Local\Temp\ipykernel_23244\1167691856.py:9: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pa
ndas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df['z_score'] = (df[column] - median_values) / mad_charges
```

Out[9]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantInc
0	LP001003	1	Yes	1	Graduate	No	4583	15
1	LP001005	1	Yes	0	Graduate	Yes	3000	
2	LP001006	1	Yes	0	Not Graduate	No	2583	23
4	LP001013	1	Yes	0	Not Graduate	No	2333	15
5	LP001024	1	Yes	2	Graduate	No	3200	7
...	...	...	...	...	...	...	...	
375	LP002943	1	No	0	Graduate	No	2987	
377	LP002974	1	Yes	0	Graduate	No	3232	19
378	LP002978	0	No	0	Graduate	No	2900	
379	LP002979	1	Yes	3+	Graduate	No	4106	
380	LP002990	0	No	0	Graduate	Yes	4583	

293 rows × 14 columns

In [10]:
```python
outliers = get_outliers(loan_df, 'LoanAmount')
# Remove outliers from the dataset
loan_df = loan_df[~outliers]
loan_df
```

```
C:\Users\torri\AppData\Local\Temp\ipykernel_23244\1167691856.py:9: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pa
ndas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df['z_score'] = (df[column] - median_values) / mad_charges
```

Out[10]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantInc
0	LP001003	1	Yes	1	Graduate	No	4583	15
2	LP001006	1	Yes	0	Not Graduate	No	2583	23
4	LP001013	1	Yes	0	Not Graduate	No	2333	15
7	LP001029	1	No	0	Graduate	No	1853	28
9	LP001032	1	No	0	Graduate	No	4950	
...	...	...	...	...	...	...	...	
373	LP002936	1	Yes	0	Graduate	No	3859	33
374	LP002940	1	No	0	Not Graduate	No	3833	
375	LP002943	1	No	0	Graduate	No	2987	
377	LP002974	1	Yes	0	Graduate	No	3232	19
380	LP002990	0	No	0	Graduate	Yes	4583	

236 rows × 14 columns

In [11]:
```python
outliers = get_outliers(loan_df, 'Loan_Amount_Term')
# Remove outliers from the dataset
loan_df = loan_df[~outliers]
loan_df
```

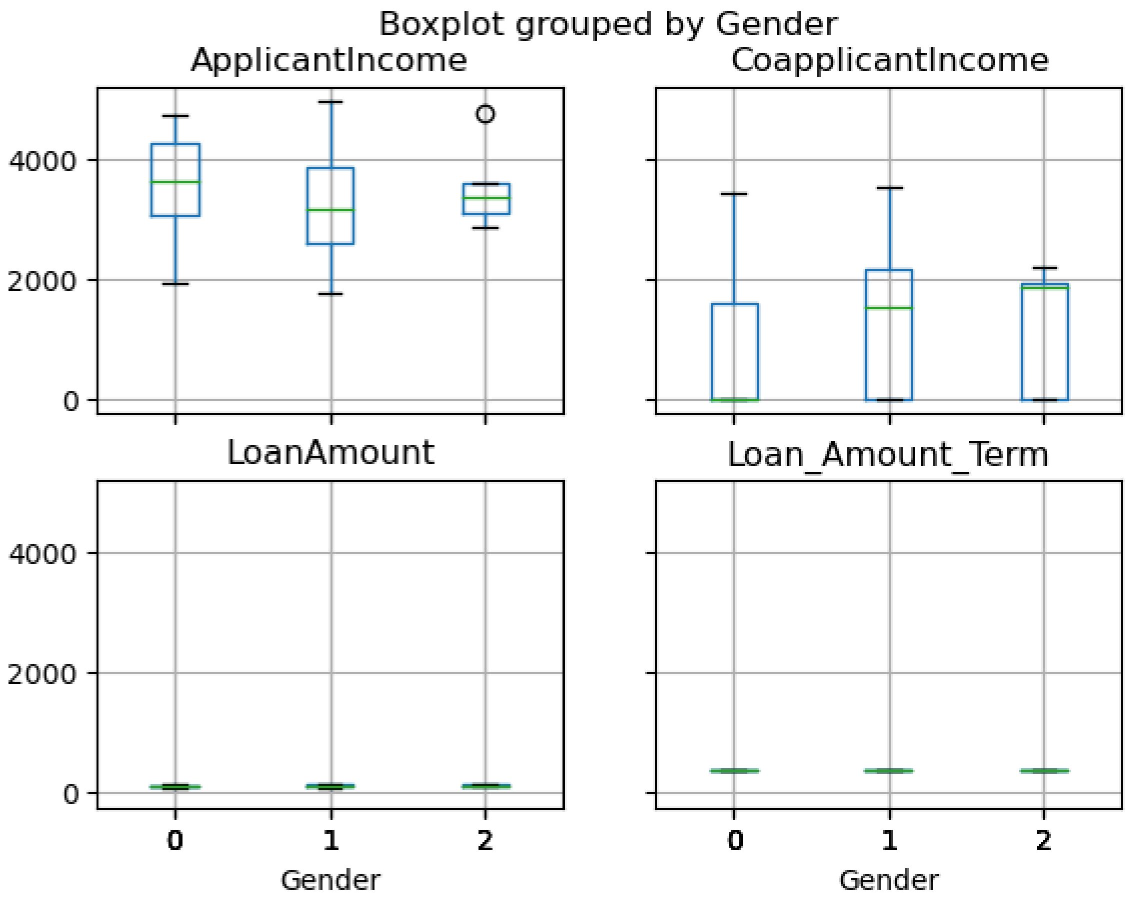
```
C:\Users\torri\AppData\Local\Temp\ipykernel_23244\1167691856.py:9: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pa
ndas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df['z_score'] = (df[column] - median_values) / mad_charges
```

Out[11]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantInc
0	LP001003	1	Yes	1	Graduate	No	4583	15
2	LP001006	1	Yes	0	Not Graduate	No	2583	23
4	LP001013	1	Yes	0	Not Graduate	No	2333	15
7	LP001029	1	No	0	Graduate	No	1853	28
9	LP001032	1	No	0	Graduate	No	4950	
...	...	...	...	...	...	...	...	
371	LP002926	1	Yes	2	Graduate	Yes	2726	
374	LP002940	1	No	0	Not Graduate	No	3833	
375	LP002943	1	No	0	Graduate	No	2987	
377	LP002974	1	Yes	0	Graduate	No	3232	19
380	LP002990	0	No	0	Graduate	Yes	4583	

197 rows × 14 columns

In [12]:
```
# Outliers and its visualization.
# Found 4 columns with outliers (ApplicantIncome, CoapplicantIncome, LoanAmount, and L
import matplotlib.pyplot as plt

loan_df.boxplot(column=['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', 'Loan_Am
plt.show()
```



In [13]:
```python
# Perform feature scaling or normalization. - Using StandardScaler
from sklearn.preprocessing import StandardScaler

scale = StandardScaler()
X = loan_df[['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount']]
loan_df[['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount']] = scale.fit_transform
loan_df
```

```
C:\Users\torri\AppData\Local\Temp\ipykernel_23244\3066053457.py:6: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pa
ndas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  loan_df[['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount']] = scale.fit_transf
orm(X)
```

Out[13]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantInc
0	LP001003	1	Yes	1	Graduate	No	1.496168	0.31
2	LP001006	1	Yes	0	Not Graduate	No	-0.899403	1.09
4	LP001013	1	Yes	0	Not Graduate	No	-1.198850	0.32
7	LP001029	1	No	0	Graduate	No	-1.773787	1.54
9	LP001032	1	No	0	Graduate	No	1.935755	-1.07
...	...	...	...	...	...	...	...	
371	LP002926	1	Yes	2	Graduate	Yes	-0.728120	-1.07
374	LP002940	1	No	0	Not Graduate	No	0.597829	-1.07
375	LP002943	1	No	0	Graduate	No	-0.415498	-1.07
377	LP002974	1	Yes	0	Graduate	No	-0.122040	0.72
380	LP002990	0	No	0	Graduate	Yes	1.496168	-1.07

197 rows × 14 columns

In [14]:
```
loan_df
```

Out[14]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantInc
0	LP001003	1	Yes	1	Graduate	No	1.496168	0.31
2	LP001006	1	Yes	0	Not Graduate	No	-0.899403	1.09
4	LP001013	1	Yes	0	Not Graduate	No	-1.198850	0.32
7	LP001029	1	No	0	Graduate	No	-1.773787	1.54
9	LP001032	1	No	0	Graduate	No	1.935755	-1.07
...	...	...	...	...	...	...	...	
371	LP002926	1	Yes	2	Graduate	Yes	-0.728120	-1.07
374	LP002940	1	No	0	Not Graduate	No	0.597829	-1.07
375	LP002943	1	No	0	Graduate	No	-0.415498	-1.07
377	LP002974	1	Yes	0	Graduate	No	-0.122040	0.72
380	LP002990	0	No	0	Graduate	Yes	1.496168	-1.07

197 rows × 14 columns

In [15]:
```python
# Encode categorical variables appropriately.
# using one-hot for categorical variables
categorical_variables = ['Married', 'Education', 'Self_Employed', 'Property_Area', 'L
loan_df = pd.get_dummies(loan_df, columns=categorical_variables, drop_first=True)
loan_df[
    ['Married_Yes',
     'Education_Not Graduate',
     'Self_Employed_Yes',
     'Property_Area_Semiurban',
     'Property_Area_Urban',
     'Loan_Status_Y']] = loan_df[
                            ['Married_Yes',
                             'Education_Not Graduate',
                             'Self_Employed_Yes',
                             'Property_Area_Semiurban',
                             'Property_Area_Urban',
                             'Loan_Status_Y']].replace({True: 1, False: 0})
loan_df
```

```
C:\Users\torri\AppData\Local\Temp\ipykernel_23244\2297040438.py:5: FutureWarning: Do
wncasting behavior in `replace` is deprecated and will be removed in a future versio
n. To retain the old behavior, explicitly call `result.infer_objects(copy=False)`. T
o opt-in to the future behavior, set `pd.set_option('future.no_silent_downcasting',
True)`
  loan_df[['Married_Yes', 'Education_Not Graduate', 'Self_Employed_Yes', 'Property_A
rea_Semiurban', 'Property_Area_Urban', 'Loan_Status_Y']] = loan_df[['Married_Yes',
'Education_Not Graduate', 'Self_Employed_Yes', 'Property_Area_Semiurban', 'Property_
Area_Urban', 'Loan_Status_Y']].replace({True: 1, False: 0})
```

Out[15]:

	Loan_ID	Gender	Dependents	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Te
0	LP001003	1	1	1.496168	0.315996	0.905919	36
2	LP001006	1	0	-0.899403	1.098083	0.460450	36
4	LP001013	1	0	-1.198850	0.323357	-0.931641	36
7	LP001029	1	0	-1.773787	1.541572	0.126348	36
9	LP001032	1	0	1.935755	-1.071517	0.738868	36
...	...	...	...	...	...	...	
371	LP002926	1	2	-0.728120	-1.071517	-0.319121	36
374	LP002940	1	0	0.597829	-1.071517	-0.096386	36
375	LP002943	1	0	-0.415498	-1.071517	-1.321426	36
377	LP002974	1	0	-0.122040	0.722681	-0.207754	36
380	LP002990	0	0	1.496168	-1.071517	1.184337	36

197 rows × 15 columns

In [16]:
```
# Split the dataset into training and testing sets.
from sklearn.model_selection import train_test_split

X = loan_df[['ApplicantIncome', 'LoanAmount', 'CoapplicantIncome', 'Loan_Amount_Term'
y = loan_df['Loan_Status_Y']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state

print('Training')
print(X_train)
print('Testing')
print(X_test)
```

```
26          0.496017  -0.096386           0.463212          360.0
309        -1.076675  -2.045313          -1.071517          360.0
106         0.195373   0.237715           0.735562          360.0
203         0.248075   0.014981          -1.071517          360.0
202        -1.528241  -1.321426           0.891980          360.0
278         0.498413  -0.653223          -1.071517          360.0
17         -0.640682   0.571817           1.001472          360.0
128         0.115121   0.905919           0.078610          360.0
368        -1.241970  -0.430488           0.328877          360.0
197        -0.487365  -0.040703           1.141327          360.0
171         0.678080  -1.210059          -1.071517          360.0
259        -1.397682   0.182032           1.136727          360.0
80         -0.451431  -1.711211          -1.071517          360.0
147        -0.022624  -0.875957          -1.071517          360.0
120        -0.879041  -1.210059           1.228737          360.0
282         0.158241   1.017286           0.041806          360.0
333        -0.784416  -0.987324           0.126455          360.0
47          0.996691   0.237715          -1.071517          360.0
121         0.198966   1.295704          -0.075047          360.0
```

Task 2 Model Building with hyperparameter tuning

In [17]:

```
# a) Select an appropriate classification algorithm (e.g., Logistic Regression, Randor
# Support Vector Machine) to predict the target categorical variable. Justify your cho
# Using Logistic Regression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

X = loan_df[['ApplicantIncome', 'LoanAmount', 'CoapplicantIncome', 'Loan_Amount_Term'
y = loan_df['Loan_Status_Y']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state

model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

prediction = model.predict(X_test)
X_test['Prediction'] = prediction
X_test
```

Out[17]:

	ApplicantIncome	LoanAmount	CoapplicantIncome	Loan_Amount_Term	Credit_History	Prediction
262	-1.294672	-0.096386	0.799049	360.0	1.000000	1
201	0.458886	0.460450	-1.071517	360.0	1.000000	1
27	0.318745	-1.766895	-1.071517	360.0	1.000000	1
131	0.618191	-0.653223	-0.167057	360.0	1.000000	1
302	1.625529	-0.653223	-1.071517	360.0	1.000000	1
358	-0.201094	-0.430488	0.827573	360.0	0.000000	0
122	0.698443	0.683184	-1.071517	360.0	1.000000	1
97	-1.761809	-0.764590	-0.113692	360.0	1.000000	1
30	0.727190	1.240021	1.077840	360.0	1.000000	1
322	0.316349	-0.653223	0.112654	360.0	1.000000	1
335	1.134437	-0.653223	-1.071517	360.0	1.000000	1
26	0.496017	-0.096386	0.463212	360.0	1.000000	1
309	-1.076675	-2.045313	-1.071517	360.0	1.000000	1
106	0.195373	0.237715	0.735562	360.0	1.000000	1
203	0.248075	0.014981	-1.071517	360.0	0.830303	1
202	-1.528241	-1.321426	0.891980	360.0	0.830303	1
278	0.498413	-0.653223	-1.071517	360.0	1.000000	1
17	-0.640682	0.571817	1.001472	360.0	1.000000	1
128	0.115121	0.905919	0.078610	360.0	0.000000	0
368	-1.241970	-0.430488	0.328877	360.0	1.000000	1
197	-0.487365	-0.040703	1.141327	360.0	1.000000	1
171	0.678080	-1.210059	-1.071517	360.0	1.000000	1
259	-1.397682	0.182032	1.136727	360.0	1.000000	1
80	-0.451431	-1.711211	-1.071517	360.0	1.000000	1
147	-0.022624	-0.875957	-1.071517	360.0	1.000000	1
120	-0.879041	-1.210059	1.228737	360.0	1.000000	1
282	0.158241	1.017286	0.041806	360.0	1.000000	1
333	-0.784416	-0.987324	0.126455	360.0	1.000000	1
47	0.996691	0.237715	-1.071517	360.0	0.000000	0
121	0.198966	1.295704	-0.075047	360.0	1.000000	1
144	-0.214270	1.574122	0.565344	360.0	1.000000	1
207	1.896228	1.017286	-1.071517	360.0	0.000000	0
354	1.357225	0.460450	-1.071517	360.0	0.830303	1
40	1.535695	1.240021	-1.071517	360.0	1.000000	1
244	-1.198850	1.351388	1.152369	360.0	1.000000	1
173	-0.276555	-1.766895	0.124615	360.0	1.000000	1
98	-0.293324	0.516133	0.106213	360.0	0.000000	0
35	-0.825140	0.460450	2.093633	360.0	0.000000	0
211	-0.477783	-0.764590	-1.071517	360.0	1.000000	1
133	0.470863	1.072970	0.561663	360.0	1.000000	1

In [18]:

```python
# Implement hyperparameter tuning by conducting a grid search or random search to
# optimize model parameters. Clearly outline the hyperparameters you tuned and the
# rationale behind them.
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression


X = loan_df[['ApplicantIncome', 'LoanAmount', 'CoapplicantIncome', 'Loan_Amount_Term'
y = loan_df['Loan_Status_Y']

# Define the hyperparameter grid
param_grid = {
    'C': [0.1, 1, 10],  # Regularization parameter
    'solver': ['lbfgs', 'liblinear', 'saga'],  # Solver options
    'max_iter': [100, 500, 1000]  # Maximum number of iterations
}

# Perform grid search with cross-validation
grid_search = GridSearchCV(LogisticRegression(), param_grid, cv=5, scoring='accuracy'
grid_search.fit(X, y)

# Get the best hyperparameters
best_params = grid_search.best_params_
print("Best hyperparameters:", best_params)
```

```
C:\Users\torri\anaconda3\Lib\site-packages\sklearn\linear_model\_sag.py:350: Conve
rgenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
C:\Users\torri\anaconda3\Lib\site-packages\sklearn\linear_model\_sag.py:350: Conve
rgenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
C:\Users\torri\anaconda3\Lib\site-packages\sklearn\linear_model\_sag.py:350: Conve
rgenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
C:\Users\torri\anaconda3\Lib\site-packages\sklearn\linear_model\_sag.py:350: Conve
rgenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
C:\Users\torri\anaconda3\Lib\site-packages\sklearn\linear_model\_sag.py:350: Conve
rgenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
C:\Users\torri\anaconda3\Lib\site-packages\sklearn\linear_model\_sag.py:350: Conve
rgenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
C:\Users\torri\anaconda3\Lib\site-packages\sklearn\linear_model\_sag.py:350: Conve
```

In [19]:
```
# Build the classification model using the training data. Explain the process and prov
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X = loan_df[['ApplicantIncome', 'LoanAmount', 'CoapplicantIncome', 'Loan_Amount_Term'
y = loan_df['Loan_Status_Y']

# Assuming 'X' contains features and 'y' contains labels
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state

# Create the logistic regression model
model = LogisticRegression(max_iter=1000)  # Increase max_iter if needed
model.fit(X_train, y_train)

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

# Evaluate model performance using accuracy score
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 0.88
```

Task 3 Model Evaluation and Selection

In [20]:
```
# a) Calculate and analyse the confusion matrix for the model.
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_
from sklearn.ensemble import RandomForestClassifier

X = loan_df[['ApplicantIncome', 'LoanAmount', 'CoapplicantIncome', 'Loan_Amount_Term'
y = loan_df['Loan_Status_Y']

# Assuming 'X' contains features and 'y' contains labels
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state

# Create the logistic regression model
model = RandomForestClassifier()  # Increase max_iter if needed
model.fit(X_train, y_train)

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

# Assuming 'y_test' contains true labels and 'y_pred' contains predicted labels
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
```

```
Confusion Matrix:
[[ 6  3]
 [ 3 28]]
```

In [21]:
```
# b) Evaluate the performance of the classification model using appropriate metrics (e
# Accuracy, Precision, Recall, F1-score).
TP, TN, FP, FN = cm.ravel()
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1_score = f1_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-Score: {f1_score:.2f}")
```

```
Accuracy: 0.85
Precision: 0.90
Recall: 0.90
F1-Score: 0.90
```

In [22]:
```
# c) Implement k-fold cross-validation (e.g., 5-fold or 10-fold) to assess the model's
# generalization performance.
# Using 5-fold
from sklearn.model_selection import KFold
from sklearn.linear_model import LogisticRegression

X = loan_df[['ApplicantIncome', 'LoanAmount', 'CoapplicantIncome', 'Loan_Amount_Term'
y = loan_df['Loan_Status_Y']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state
# Assuming 'X' contains features and 'y' contains labels
kf = KFold(n_splits=5, shuffle=True, random_state=42)
model = LogisticRegression(max_iter=1000)  # Increase max_iter if needed

for train_index, test_index in kf.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)


    # Evaluate model performance
    accuracy = accuracy_score(y_test, y_pred)

    print("Accuracy", accuracy)
    # Evaluate model performance (e.g., accuracy, precision, recall, F1-score)
    # ...

# Calculate average performance metrics across all folds
# ...
```

```
Accuracy 0.875
Accuracy 0.925
Accuracy 0.8205128205128205
Accuracy 0.7435897435897436
Accuracy 0.8974358974358975
```

In [23]:

```
# d) Select the best-performing classification model based on hyperparameter tuning ar
# cross-validation results and justify the choice of the selected model for its suital
# addressing the business problem.

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

X = loan_df[['ApplicantIncome', 'LoanAmount', 'CoapplicantIncome', 'Loan_Amount_Term'
y = loan_df['Loan_Status_Y']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state

# Define hyperparameter grid for Random Forest
param_grid = {
    'n_estimators': [100, 200, 500],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5],
    # Add other hyperparameters as needed
}

# Create Random Forest model
rf_model = RandomForestClassifier()

# Perform grid search with cross-validation
grid_search = GridSearchCV(rf_model, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X, y)

# Get best hyperparameters
best_params = grid_search.best_params_

# Train final model with best hyperparameters
final_rf_model = RandomForestClassifier(**best_params)
final_rf_model.fit(X, y)

# Evaluate performance on test data
y_pred = final_rf_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-Score: {f1:.2f}")
```

```
Accuracy: 1.00
Precision: 1.00
Recall: 1.00
F1-Score: 1.00
```

In [ ]: