GDDA708 – Machine Learning and AI

16 August 2024

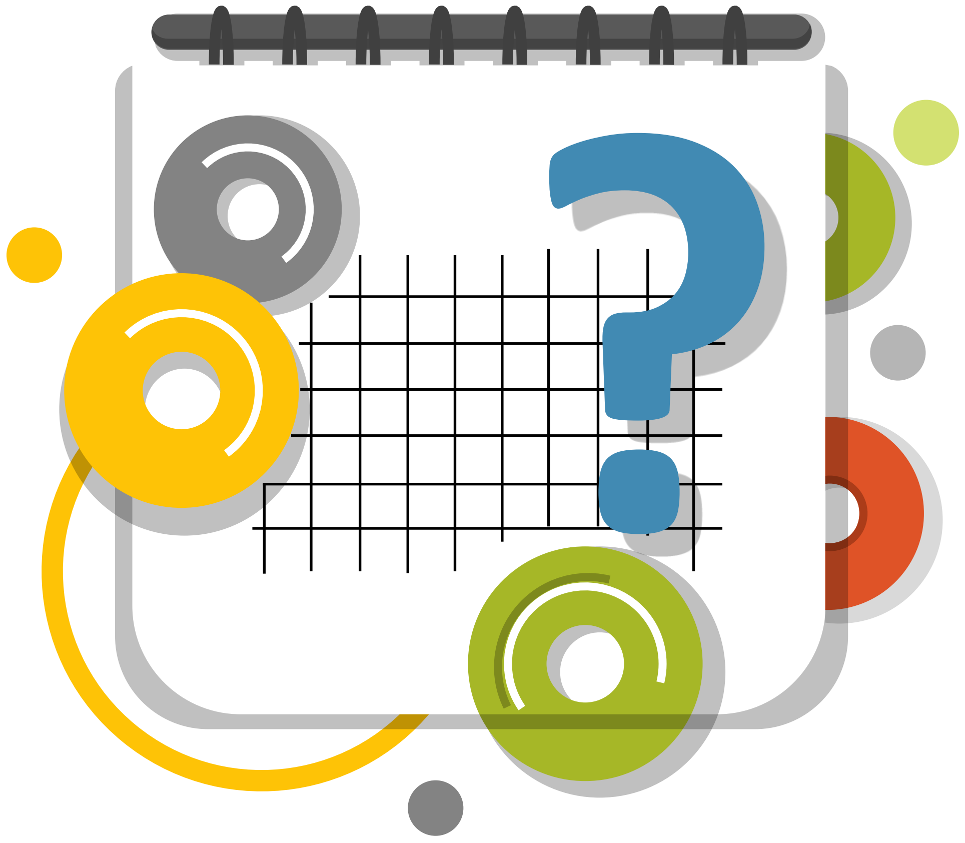


Table of Contents

1. Introduction

Overview

This assessment is designed to provide a comprehensive evaluation of learners' understanding and practical skills in both supervised and unsupervised machine learning. Participants will engage in a series of tasks and projects that require them to:

1. **Data Regression Model, Classification Model, and Trend Prediction**: Apply various supervised machine learning models to classify datasets and predict future trends. This will test their ability to select appropriate algorithms, preprocess data, train models, and evaluate their performance.
2. **Business Decision-Making Enhancement**: Utilize unsupervised machine learning algorithms to analyze complex datasets and derive insights that can inform and improve business strategies. This component will focus on clustering, anomaly detection, and other unsupervised techniques to uncover patterns and relationships within the data.

Throughout the assessment, learners will demonstrate their capability to handle real-world data challenges, showcasing their analytical thinking, problem-solving skills, and technical proficiency in machine learning.

Purpose

The assessment aims to evaluate learners' proficiency in applying supervised machine learning techniques for data classification and trend prediction. Additionally, learners will implement unsupervised machine learning algorithms to enhance business decision-making

Part A – Regression modelling for business decision making

LO 3: Evaluate machine learning algorithms for its applications to Artificial Intelligence.
LO 4: Apply fine-tuning methods to optimize the performance of machine learning models

My scenario is wine quality red here you can get information about wine because for is new scenario I don't know what is important about red wine. I need to study, and I got the information below:

This dataset Wine Quality Red uses the following 12 variables:

Fixed acidity: Most acids involved with wine or fixed or non-volatile (do not evaporate readily)

volatile acidity: The amount of acetic acid in wine, which at too high of levels can lead to an unpleasant, vinegar taste

citric acid: Found in small quantities, citric acid can add 'freshness' and flavour to wines

residual sugar: The amount of sugar remaining after fermentation stops, it's rare to find wines with less than 1 gram/litter and

Chlorides: the amount of salt in the wine

free sulphur dioxide: the free form of SO2 exists in equilibrium between molecular SO2 (as a dissolved gas) and bisulfited ion; it prevents

total sulphur dioxide: the amount of free and bound forms of S02; in low concentrations, SO2 is mostly undetectable in wine, but at free SO2

Density: the density of water is close to that of water depending on the percent alcohol and sugar content

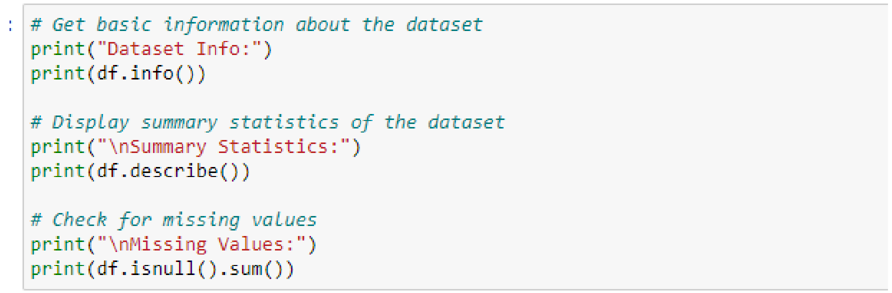
pH: describes how acidic or basic wine is on a scale from 0 (very acidic) to 14 (very basic); most wines are between 3–4 on the

Sulphates: a wine additive that can contribute to sulfur dioxide gas (S02) levels, which acts as an antimicrobial and

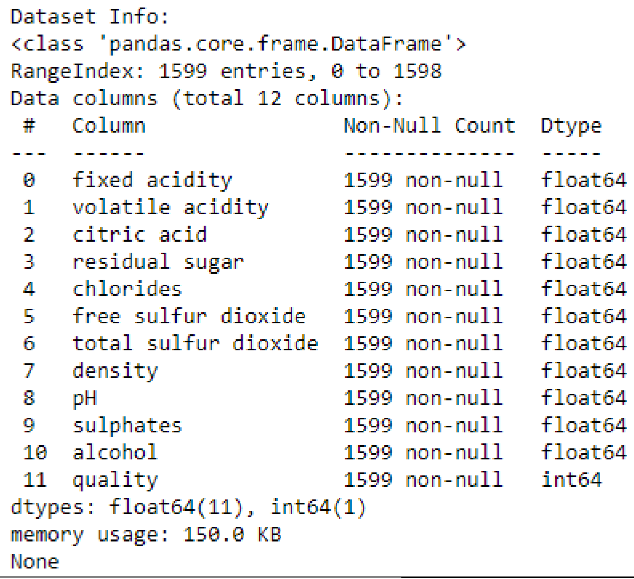
Alcohol: The percent alcohol content of the wine

Quality: output variable (based on sensory data, score between 0 and 10)

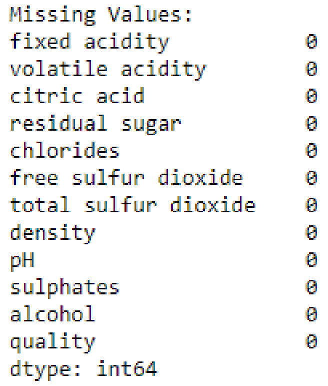
	Task- 1 Data Preprocessing
A	Properly clean the dataset, handle any missing values, and remove outliers.



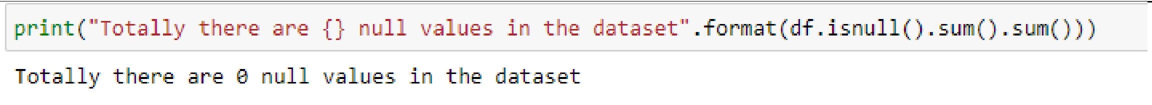
```
# Get basic information about the dataset
print("Dataset Info:")
print(df.info())

# Display summary statistics of the dataset
print("\nSummary Statistics:")
print(df.describe())

# Check for missing values
print("\nMissing Values:")
print(df.isnull().sum())
```



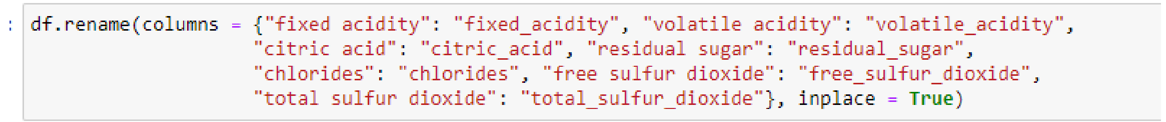
```
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
None
```



```
Missing Values:
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```



```
print("Totally there are {} null values in the dataset".format(df.isnull().sum().sum()))

Totally there are 0 null values in the dataset
```



```
df.rename(columns = {"fixed acidity": "fixed_acidity", "volatile acidity": "volatile_acidity",
                    "citric acid": "citric_acid", "residual sugar": "residual_sugar",
                    "chlorides": "chlorides", "free sulfur dioxide": "free_sulfur_dioxide",
                    "total sulfur dioxide": "total_sulfur_dioxide"}, inplace = True)
```

I showed my dataset, all the columns, and missing values. I didn't work on missing values after I renamed some columns. It turns out that the dataset does not have **null values.** The dataset consists of **1599 rows and 12 columns.** The data type of all variables is numeric

B Perform feature scaling or normalization.

```
scaler = preprocessing.MinMaxScaler()
minmax = scaler.fit_transform(df)
minmax = pd.DataFrame(minmax)
```

```
scaler = preprocessing.StandardScaler()
standard = scaler.fit_transform(df)
standard = pd.DataFrame(standard)
```

```
scaler = preprocessing.RobustScaler()
robust = scaler.fit_transform(df)
robust = pd.DataFrame(robust)
```

```
print('\033[1m'+'Without scaling:'+'\033[0m')
display(df.head())
print('*' * 45)
print('\033[1m'+'With minmax scaling:'+'\033[0m')
display(minmax.head())
print('*' * 45)
print('\033[1m'+'With standard scaling:'+'\033[0m')
display(standard.head())
print('*' * 45 )
print('\033[1m'+'With robust scaling:'+'\033[0m')
display(robust.head())
```

Without scaling:

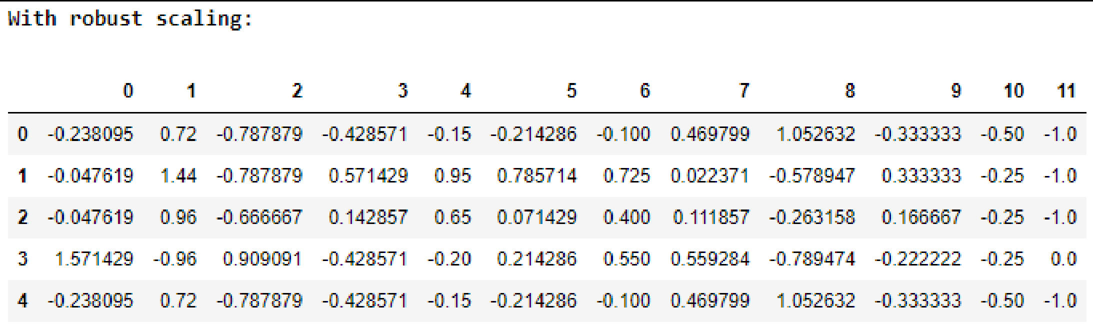
	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

With minmax scaling:

	0	1	2	3	4	5	6	7	8	9	10	11
0	0.247788	0.397260	0.00	0.068493	0.106845	0.140845	0.098940	0.567548	0.606299	0.137725	0.153846	0.4
1	0.283186	0.520548	0.00	0.116438	0.143573	0.338028	0.215548	0.494126	0.362205	0.209581	0.215385	0.4
2	0.283186	0.438356	0.04	0.095890	0.133556	0.197183	0.169611	0.508811	0.409449	0.191617	0.215385	0.4
3	0.584071	0.109589	0.56	0.068493	0.105175	0.225352	0.190813	0.582232	0.330709	0.149701	0.215385	0.6
4	0.247788	0.397260	0.00	0.068493	0.106845	0.140845	0.098940	0.567548	0.606299	0.137725	0.153846	0.4

With standard scaling:

	0	1	2	3	4	5	6	7	8	9	10	11
0	-0.528360	0.961877	-1.391472	-0.453218	-0.243707	-0.466193	-0.379133	0.558274	1.288643	-0.579207	-0.960246	-0.787823
1	-0.298547	1.967442	-1.391472	0.043416	0.223875	0.872638	0.624363	0.028261	-0.719933	0.128950	-0.584777	-0.787823
2	-0.298547	1.297065	-1.186070	-0.169427	0.096353	-0.083669	0.229047	0.134264	-0.331177	-0.048089	-0.584777	-0.787823
3	1.654856	-1.384443	1.484154	-0.453218	-0.264960	0.107592	0.411500	0.664277	-0.979104	-0.461180	-0.584777	0.450848
4	-0.528360	0.961877	-1.391472	-0.453218	-0.243707	-0.466193	-0.379133	0.558274	1.288643	-0.579207	-0.960246	-0.787823



```
With robust scaling:

          0     1         2         3     4         5      6         7         8         9    10   11
0 -0.238095  0.72 -0.787879 -0.428571 -0.15 -0.214286 -0.100  0.469799  1.052632 -0.333333 -0.50 -1.0
1 -0.047619  1.44 -0.787879  0.571429  0.95  0.785714  0.725  0.022371 -0.578947  0.333333 -0.25 -1.0
2 -0.047619  0.96 -0.666667  0.142857  0.65  0.071429  0.400  0.111857 -0.263158  0.166667 -0.25 -1.0
3  1.571429 -0.96  0.909091 -0.428571 -0.20  0.214286  0.550  0.559284 -0.789474 -0.222222 -0.25  0.0
4 -0.238095  0.72 -0.787879 -0.428571 -0.15 -0.214286 -0.100  0.469799  1.052632 -0.333333 -0.50 -1.0
```

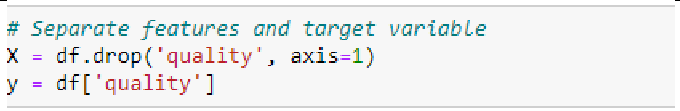
C Appropriately encode categorical variables.

In my dataset, I have only numeric columns below:

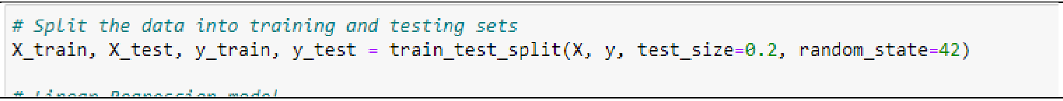
fixed_acidity	1599 non-null float64
volatile_acidity	1599 non-null float64
citric_acid	1599 non-null float64
residual_sugar	1599 non-null float64
chlorides	1599 non-null float64
free_sulfur_dioxide	1599 non-null float64
total_sulfur_dioxide	1599 non-null float64
density	1599 non-null float64
pH	1599 non-null float64
sulphates	1599 non-null float64
alcohol	1599 non-null float64
quality	1599 non-null int64

I don't need to apply encoded categorial in my scenario.

D Split the dataset into training and testing sets.



```python
# Separate features and target variable
X = df.drop('quality', axis=1)
y = df['quality']
```



```python
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Task 2 – Model Building with hyper-parameter tuning

A Choose any appropriate regression models (e.g., Linear Regression, Ridge Regression, Lasso Regression)

I need to get with the models' popular regression models: **Linear Regression**, **Ridge Regression**, and **Lasso Regression**.

Linear Regression is a basic model that assumes a linear relationship between the input features (e.g., acidity, sugar level) and the target variable (wine quality). It minimizes the sum of squared residuals to fit the best linear line.

Ridge Regression is a regularized version of Linear Regression that adds a penalty on the size of coefficients (L2 regularization). This penalty term helps to control overfitting by shrinking large coefficients.

Lasso Regression also adds regularization but with an L1 penalty, which can drive some coefficients to zero. This property makes Lasso useful for feature selection, as it effectively excludes irrelevant features.

My scenario is about wine quality red. First, I checked all dataset results, including mean squared error (MSE) and mean absolute error (MAE).

I got the results below:

Linear Regression

```
Mean Squared Error: 0.39002514396395427
Mean Absolute Error: 0.5057883195080454
R-squared: 0.4031803412796229
```

Lasso Regression

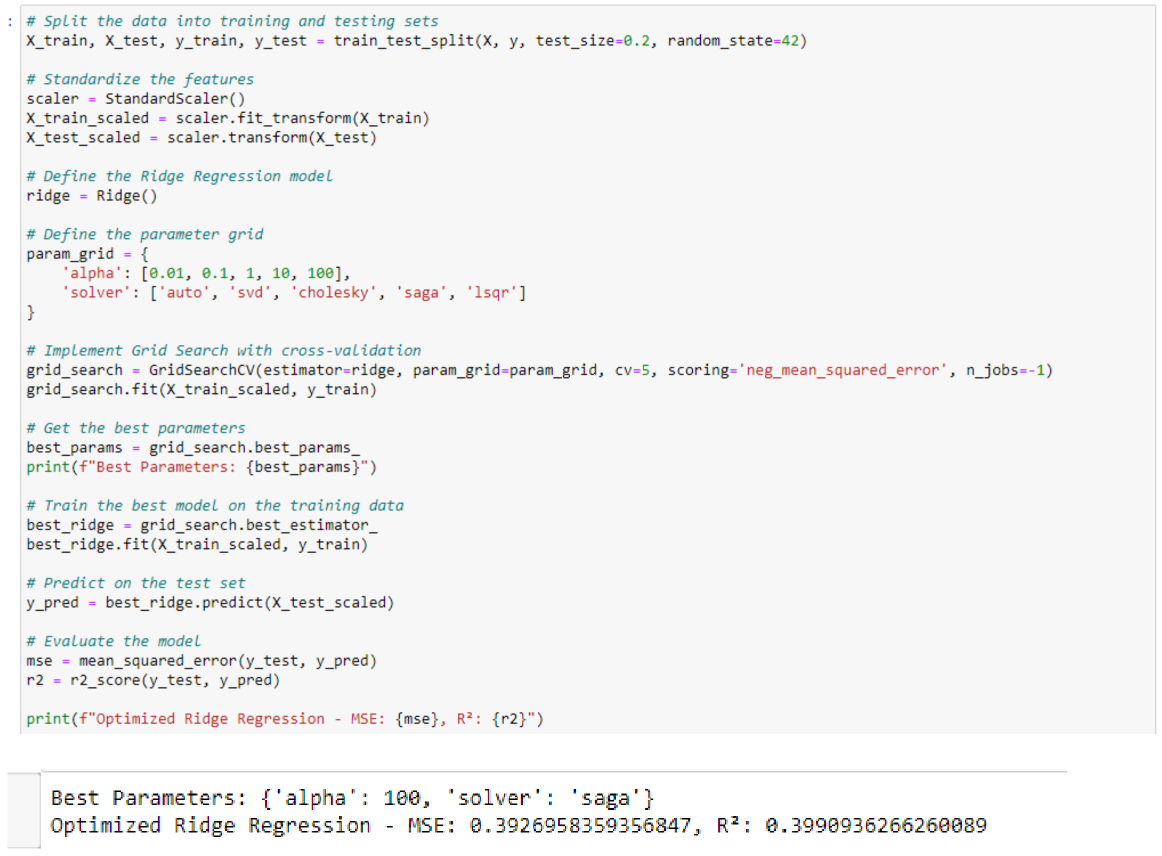
```
Best Parameters: {'alpha': 0.01}
Best Cross-Validation Score: 0.44779313025204565
Mean Squared Error: 0.49867516307447735
Mean Absolute Error: 0.5057883195080454
R-squared: 0.23692319522368122
```

Ridge Regression

```
Best Parameters: {'alpha': 1}
Best Cross-Validation Score: 0.43931582683684345
Mean Squared Error: 0.3929488678812798
Mean Absolute Error: 0.5057883195080454
R-squared: 0.39870643507758563
```

I can see better results for Linear Regression, but I have many variables in my scenario I decided to get Ridge Regression because I have similar results Mean Squared, Mean Absolute Error, and R-squared if comparable with Linear Regression.

B	Implement hyperparameter tuning by conducting a grid search or random search to optimize model parameters. Clearly outline the hyperparameter you tuned and the rationale behind them.



```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Define the Ridge Regression model
ridge = Ridge()

# Define the parameter grid
param_grid = {
    'alpha': [0.01, 0.1, 1, 10, 100],
    'solver': ['auto', 'svd', 'cholesky', 'saga', 'lsqr']
}

# Implement Grid Search with cross-validation
grid_search = GridSearchCV(estimator=ridge, param_grid=param_grid, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
grid_search.fit(X_train_scaled, y_train)

# Get the best parameters
best_params = grid_search.best_params_
print(f"Best Parameters: {best_params}")

# Train the best model on the training data
best_ridge = grid_search.best_estimator_
best_ridge.fit(X_train_scaled, y_train)

# Predict on the test set
y_pred = best_ridge.predict(X_test_scaled)

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

print(f"Optimized Ridge Regression - MSE: {mse}, R²: {r2}")
```

```
Best Parameters: {'alpha': 100, 'solver': 'saga'}
Optimized Ridge Regression - MSE: 0.3926958359356847, R²: 0.3990936266260089
```

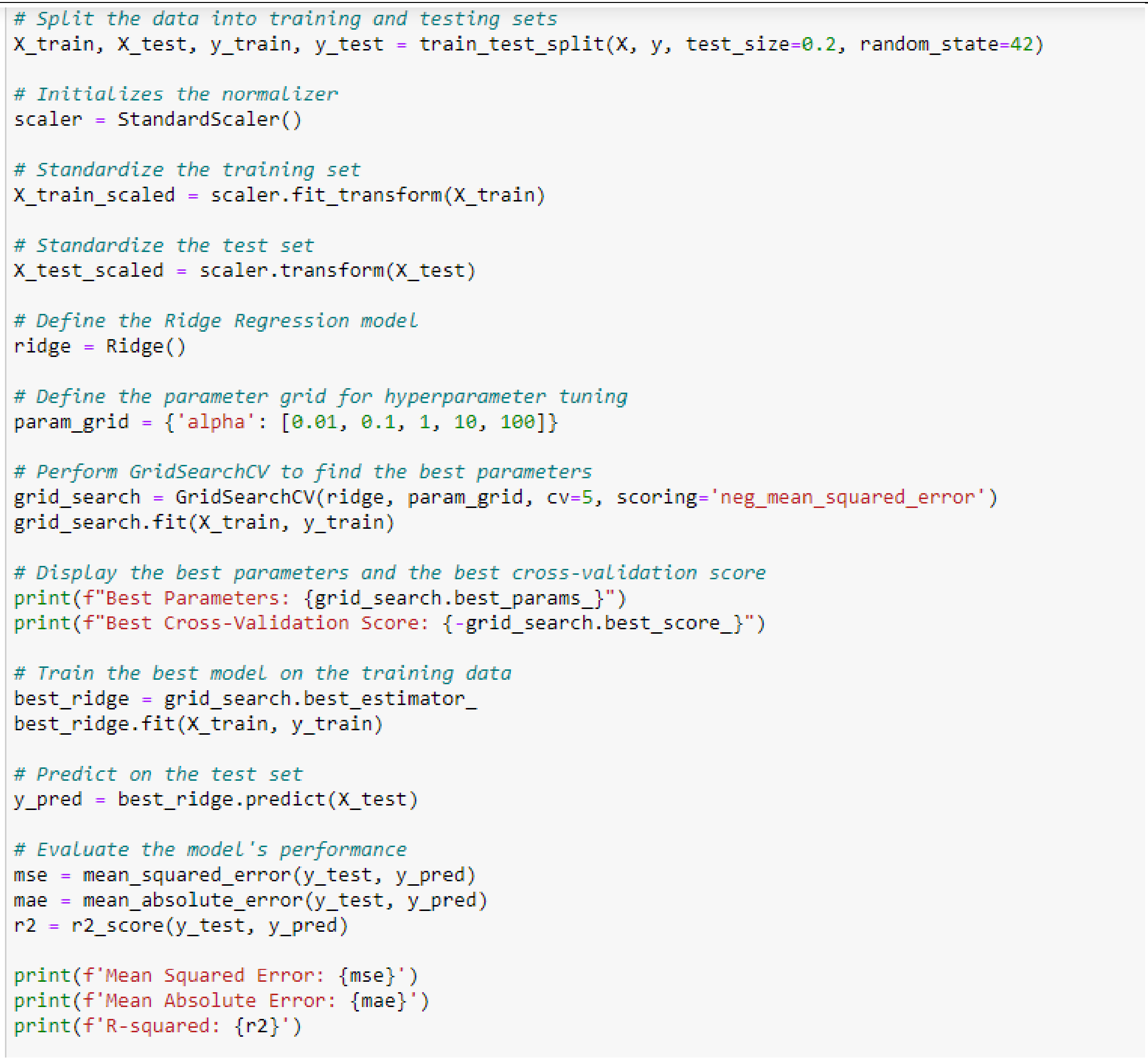
Model Selection: We'll perform hyperparameter tuning on Ridge Regression, as it handles multicollinearity effectively and is suitable when all features contribute to the prediction. We could also apply this process to other models like Lasso or Linear Regression.

The choice of the Ridge Regression model with the hyperparameters {'alpha': 100, 'solver': 'saga'} is justified based on its ability to effectively balance model complexity and predictive accuracy in the context of the wine quality prediction task.

Suitability for the Business Problem: Wine quality is influenced by multiple correlated chemical properties (e.g., acidity, sugar levels). Ridge Regression with a strong alpha effectively manages multicollinearity by shrinking the coefficients of less important features, leading to a more stable and reliable model. The selected model balances the need to fit the training data while maintaining the ability to generalize to new, unseen data. This is crucial for making accurate predictions.

The Ridge Regression model with alpha = 100 and the 'saga' solver is an optimal choice for predicting wine quality in this dataset. It provides a robust approach that mitigates overfitting through regularization, efficiently handles the dataset with a powerful solver, and achieves a good balance between prediction accuracy (MSE) and the ability to explain variance (R^2). This makes it a well-justified selection for the business objective of accurately predicting wine quality based on its chemical properties

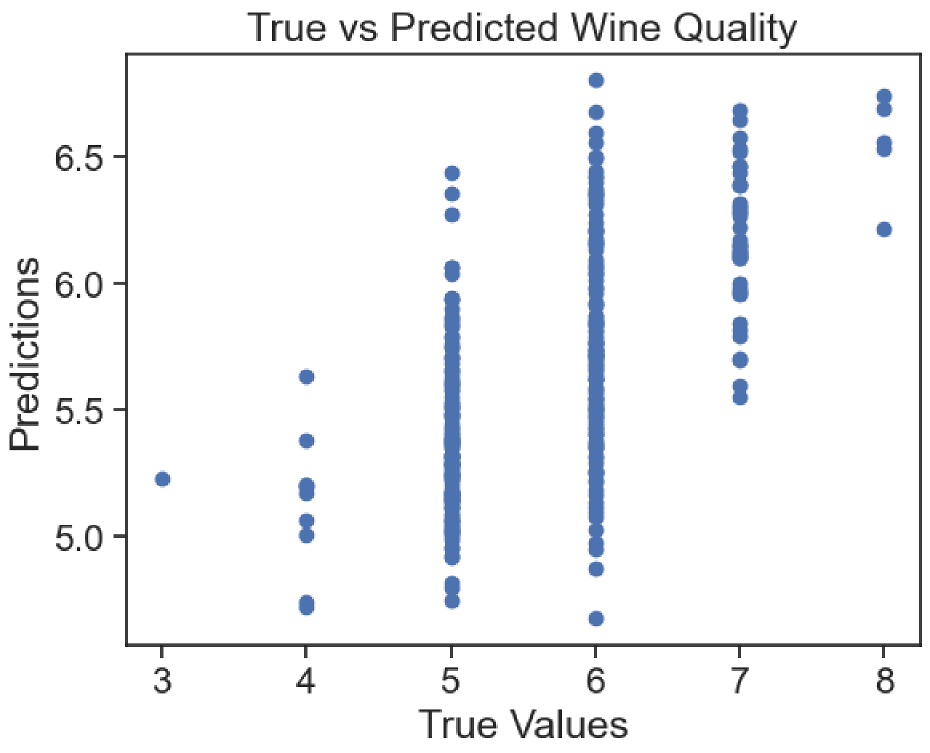
C	Build the regression models using the training data. Describe the process and provide code snippets.



```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initializes the normalizer
scaler = StandardScaler()

# Standardize the training set
X_train_scaled = scaler.fit_transform(X_train)

# Standardize the test set
X_test_scaled = scaler.transform(X_test)

# Define the Ridge Regression model
ridge = Ridge()

# Define the parameter grid for hyperparameter tuning
param_grid = {'alpha': [0.01, 0.1, 1, 10, 100]}

# Perform GridSearchCV to find the best parameters
grid_search = GridSearchCV(ridge, param_grid, cv=5, scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)

# Display the best parameters and the best cross-validation score
print(f"Best Parameters: {grid_search.best_params_}")
print(f"Best Cross-Validation Score: {-grid_search.best_score_}")

# Train the best model on the training data
best_ridge = grid_search.best_estimator_
best_ridge.fit(X_train, y_train)

# Predict on the test set
y_pred = best_ridge.predict(X_test)

# Evaluate the model's performance
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'Mean Absolute Error: {mae}')
print(f'R-squared: {r2}')
```

```
# Display feature coefficients
coefficients = pd.DataFrame(best_ridge.coef_, X.columns, columns=['Coefficient'])
print(coefficients)

# Plot true vs predicted quality
plt.scatter(y_test, y_pred)
plt.xlabel("True Values")
plt.ylabel("Predictions")
plt.title("True vs Predicted Wine Quality")
plt.show()
```

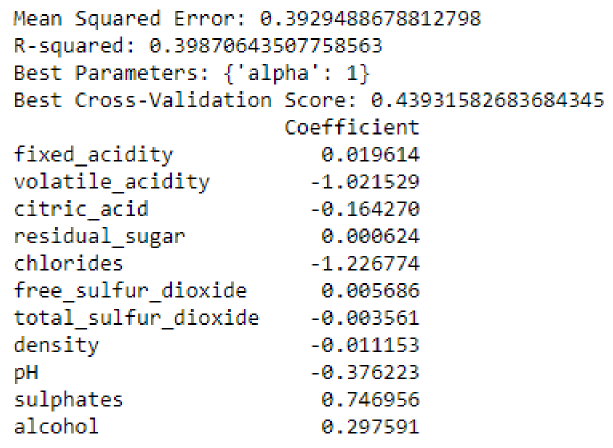
```
Best Parameters: {'alpha': 1}
Best Cross-Validation Score: 0.43931582683684345
Mean Squared Error: 0.3929488678812798
Mean Absolute Error: 0.5057883195080454
R-squared: 0.39870643507758563
                      Coefficient
fixed_acidity            0.019614
volatile_acidity        -1.021529
citric_acid             -0.164270
residual_sugar           0.000624
chlorides               -1.226774
free_sulfur_dioxide      0.005686
total_sulfur_dioxide    -0.003561
density                 -0.011153
pH                      -0.376223
sulphates                0.746956
alcohol                  0.297591
```



The performance metrics for Ridge Regression are promising: Mean Squared Error (MSE) 39%, Mean Absolute Error 50%, and R-squared (R²): 39%.

Task 3 – Model Evaluation and Selection

A: Evaluate the performance of the regression models using appropriate metrics (e.g., Mean Absolute Error, R-squared). Calculate and interpret these metrics.



```
Mean Squared Error: 0.3929488678812798
R-squared: 0.39870643507758563
Best Parameters: {'alpha': 1}
Best Cross-Validation Score: 0.43931582683684345
                      Coefficient
fixed_acidity            0.019614
volatile_acidity        -1.021529
citric_acid             -0.164270
residual_sugar           0.000624
chlorides               -1.226774
free_sulfur_dioxide      0.005686
total_sulfur_dioxide    -0.003561
density                 -0.011153
pH                      -0.376223
sulphates                0.746956
alcohol                  0.297591
```

Best Cross-Validation Score

- **Value**: 0.43931582683684345
- **Interpretation**: The cross-validation score represents the average performance of the model across different splits of the training data. A score of approximately 0.4393 indicates that the model explains about 43.93% of the variance in the training data across different folds. This suggests that the model is relatively stable and generalizes well to different subsets of the data.

Mean Squared Error (MSE)

- **Value**: 0.3929488678812798
- **Interpretation**: MSE measures the average squared difference between the actual and predicted values. In this case, the MSE of approximately 0.393 indicates the average squared error of the predictions. This metric suggests that, on average, the squared difference between the predicted and actual wine quality scores is around 0.393. Lower MSE values indicate better predictive accuracy.

Mean Absolute Error (MAE)

- **Value**: 0.5057883195080454
- **Interpretation**: MAE measures the average absolute difference between the actual and predicted values. An MAE of approximately 0.506 means that, on average, the model's predictions are off by about 0.506 points on the wine quality scale. This metric is more interpretable than MSE because it represents the average error in the same units as the target variable.

R-squared (R^2)

- **Value**: 0.39870643507758563

- **Interpretation**: R^2 represents the proportion of variance in the dependent variable (wine quality) that is predictable from the independent variables. An R^2 of approximately 0.399 indicates that about 39.87% of the variability in wine quality can be explained by the model. While this isn't a very high value, it still shows that the model captures a meaningful portion of the variance.

Model Coefficients
Fixed acidity: 0.019614 - A small positive relationship with wine quality, meaning that as fixed acidity increases by one unit, the predicted wine quality increases slightly, holding other factors constant.

Volatile acidity: -1.021529 - A strong negative relationship, suggesting that higher volatile acidity is associated with a decrease in wine quality.

Citric acid: -0.164270 - A moderate negative relationship, indicating that higher citric acid levels slightly decrease wine quality.

Residual sugar: 0.000624 - A negligible positive effect on wine quality, meaning that changes in residual sugar levels have almost no impact on the predicted quality.

chlorides: -1.226774 - A strong negative relationship, indicating that higher chloride levels are associated with a lower wine quality score.

Free sulfur dioxide: 0.005686 - A very small positive relationship, meaning that increases in free sulfur dioxide slightly increase wine quality.

Total sulphur dioxide: -0.003561 - A very small negative relationship, meaning that higher levels of total sulphur dioxide slightly decrease wine quality.

Density: -0.011153 - A small negative relationship, indicating that higher density is associated with a slight decrease in wine quality.

PH: -0.376223 - A moderate negative relationship, suggesting that higher pH levels (less acidity) decrease wine quality.

Sulfates: 0.746956 - A strong positive relationship, indicating that higher sulphate levels are associated with better wine quality.

Alcohol: 0.297591 - A positive relationship, indicating that higher alcohol content is associated with better wine quality.

B: Implement k-fold, cross-validation (e.g., 5-fold or 10-fold) to assess the model's generalization performance using multiple splits.

```
# Function to perform K-Fold cross-validation with different K values
def evaluate_model_with_kfold(X, y, k_values):
    results = {}
    for k in k_values:
        print(f"Evaluating with K={k}")
        kfold = StratifiedKFold(n_splits=k, shuffle=True, random_state=42)
        scores = cross_val_score(pipeline, X, y, cv=kfold, scoring='accuracy')
        results[k] = (np.mean(scores), np.std(scores))
        print(f"Mean Accuracy: {np.mean(scores):.2f}, Standard Deviation: {np.std(scores):.2f}\n")
    return results
```

```
# Define K values to evaluate
k_values = [5, 8, 10,12,15]


# Evaluate the model
results = evaluate_model_with_kfold(X, y, k_values)
```

```
Evaluating with K=5
Mean Accuracy: 0.58, Standard Deviation: 0.02

Evaluating with K=8
Mean Accuracy: 0.58, Standard Deviation: 0.03

Evaluating with K=10
Mean Accuracy: 0.58, Standard Deviation: 0.03

Evaluating with K=12
Mean Accuracy: 0.58, Standard Deviation: 0.04

Evaluating with K=15
Mean Accuracy: 0.58, Standard Deviation: 0.04
```

Accuracy measures the proportion of correct predictions made by the model relative to the total predictions. An accuracy of 0.58 indicates that 58% of the predictions are correct.

The standard deviation increases slightly as K increases, going from 0.02 to 0.04. Although the increase is small, it suggests a slight change in model performance as K increases.

The consistency of the mean accuracy at 0.58 indicates that the KNN model is performing quite steadily regardless of the chosen K value. This could be a sign that the dataset or problem characteristics are not particularly sensitive to the choice of K within this range.

C: Select the best-performing regression model based on hyperparameter tuning and cross-validation results. Justify your choice of the selected model for its suitability in addressing the business problem.

```
print(f"Optimized Ridge Regression - MSE: {mse}, R²: {r2}")
```

```
Best Parameters: {'alpha': 100, 'solver': 'saga'}
Optimized Ridge Regression - MSE: 0.3926958359356847, R²: 0.3990936266260089

Best Parameters: {'alpha': 1}
Best Cross-Validation Score: 0.43931582683684345
```

The Ridge Regression model, with the best parameters {'alpha': 100, 'solver': 'saga'}, has been selected as the best-performing model based on hyperparameter tuning and cross-validation results. This model achieved a Mean Squared Error (MSE) of **0.39** and an R^2 score of **0.39** on the test data.

	High Regularization Strength (Alpha = 100): The alpha parameter in Ridge Regression controls the degree of regularization. A higher value, such as 100, implies stronger regularization, which helps prevent overfitting, especially in datasets with many features or multicollinearity. Overfitting is a common issue in regression models that can lead to poor performance on unseen data. By choosing alpha = 100, the model is better able to generalize to new data, which is crucial for making accurate predictions in real-world scenarios, where the model will need to predict wine quality for new samples. Solver Choice (Solver = 'saga'): The 'saga' solver is well-suited for large-scale optimization problems and can efficiently handle both dense and sparse datasets. It also supports elastic net regularization, making it versatile and robust. The use of the 'saga' solver ensures that the model is both computationally efficient and accurate, especially when working with larger datasets that contain a mix of feature types. Model Performance (MSE and R^2): **0.3927**: This indicates the average squared difference between the actual and predicted wine quality scores. A lower MSE reflects better predictive accuracy, and in this case, the MSE of 0.3927 suggests that the model's predictions are closely aligned with the actual values. **R^2 = 0.3991**: The R^2 score represents the proportion of variance in the wine quality that the model can explain. An R^2 of 0.3991 means that the model explains approximately 39.91% of the variability in the wine quality, which is significant given the complexity of the task. Suitability for the Business Problem: The business goal is to predict wine quality based on various chemical properties, which is a challenging task due to the complexity and subtlety of the factors involved. Ridge Regression is particularly well-suited for this problem because it reduces the impact of less relevant features through regularization, leading to more stable and interpretable results. The selected model helps the business accurately predict wine quality, enabling better quality control and targeted improvements in production. This can lead to the production of higher-quality wines, meeting consumer expectations and driving business success.

	Task 4- Business Decision and Recommendations
	Provide actionable recommendations derived from the model's predictions.
	After analyzing the coefficient Ridge Prediction showed us. We can do better control in some variables. The **acidez volátil** must be less because impact negative in control quality. I need to check the fermentation process. Monitor and minimize **chloride** content, as it also negatively impacts on quality the red wine. Keep the **pH** in good level because impact the quality wine. Slightly increase **sulfate** levels and **alcohol** content, as both have a positive impact on quality. Although residual **sugar** and **sulphur dioxide** have minor impacts, they must be controlled to maintain the overall quality of the wine.

	Based on the model coefficients and quality predictions, we can make informed decisions about the production process and marketing of red wines. Specific recommendations can help improve wine quality and maximize the commercial value of each batch.

Part B – Classification modeling for business decision making

Bike Buyers

This dataset has details of 1000 users from different backgrounds and whether or not they buy a bike. This data can be used for prediction models using Machine Learning Algorithms. Some NA values were injected into the dataset. We will use this dataset for Data Cleaning, Exploration, Visualization, and ML prediction.

About Dataset:

Columns - There are 13 variables in this dataset:

ID - The buyer's ID. This ID variable data type is integer.

Marital Status - The buyer’s state of being married or single. This marital status data type is characters.

Gender - The buyer’s state either the buyer is male or female. This gender data type is characters.

Income - Buyer’s income in a certain time. This income variable data type is an integer.

Children - The number of children the buyer has. This children's variable data type is integer.

Education - Buyer’s education background is either Bachelor's, Graduate Degree, High School, Partial College, or Partial High School. This education variable data type is character.

Occupation - The buyer’s job or occupation is either Clerical, Management, Manual, Professional, or Skilled Manual. This occupation variable data type is character.

Homeowner - Buyers state whether the buyer has or does not have their own house. This homeowner variable data type is the character.

Cars - The number of vehicles the buyer has. This car's variable data type is integer.

Commute Distance - The distance between the buyer’s house and the buyer’s company. There are 5 categories: 0-1 Miles, 1-2 Miles 10+ Miles, 2-5 Miles, and 5-10 Miles. This commute distance variable data type is character.

Region - Variable that tells us where the buyer lives in Europe, North America, or the Pacific. This region variable data type is character.

Age - The buyer’s age. This age variable data type is integer.

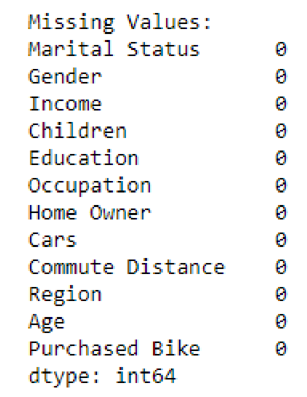
Purchased Bike - (Target variable) Buyer’s state whether the buyer purchased the bike or not. This purchased variable data type is character. It seems we should transform it into binary variable. So, the later 'Purchased Bike' column will contain '0' for 'No' and '1' for 'Yes.

Task 1 – Data Preprocessing

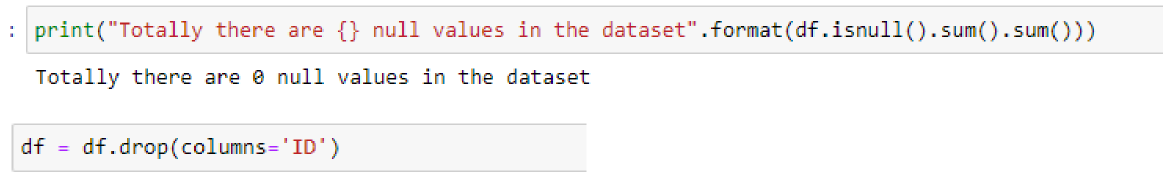
A: Clean the dataset by handling missing values and removing outliers as needed.

```python
# Get basic information about the dataset
print("Dataset Info:")
print(df.info())

# Display summary statistics of the dataset
print("\nSummary Statistics:")
print(df.describe())

# Check for missing values
print("\nMissing Values:")
print(df.isnull().sum())
```

```
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   Marital Status    1000 non-null   object
 1   Gender            1000 non-null   object
 2   Income            1000 non-null   int64 
 3   Children          1000 non-null   int64 
 4   Education         1000 non-null   object
 5   Occupation        1000 non-null   object
 6   Home Owner        1000 non-null   object
 7   Cars              1000 non-null   int64 
 8   Commute Distance  1000 non-null   object
 9   Region            1000 non-null   object
 10  Age               1000 non-null   int64 
 11  Purchased Bike    1000 non-null   object
dtypes: int64(4), object(8)
memory usage: 93.9+ KB
None

Summary Statistics:
              Income     Children         Cars          Age
count    1000.000000  1000.000000  1000.000000  1000.000000
mean    56140.000000     1.908000     1.452000    44.190000
std     31081.609779     1.626094     1.124705    11.353537
min     10000.000000     0.000000     0.000000    25.000000
25%     30000.000000     0.000000     1.000000    35.000000
50%     60000.000000     2.000000     1.000000    43.000000
75%     70000.000000     3.000000     2.000000    52.000000
max    170000.000000     5.000000     4.000000    89.000000
```



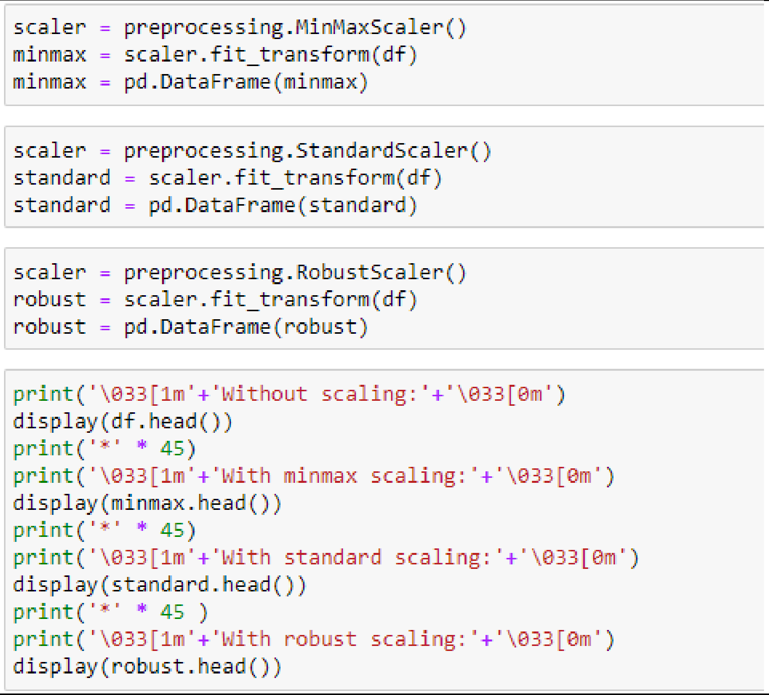
```
Missing Values:
Marital Status      0
Gender              0
Income              0
Children            0
Education           0
Occupation          0
Home Owner          0
Cars                0
Commute Distance    0
Region              0
Age                 0
Purchased Bike      0
dtype: int64
```



```
: print("Totally there are {} null values in the dataset".format(df.isnull().sum().sum()))

Totally there are 0 null values in the dataset
```

```
df = df.drop(columns='ID')
```

B: Perform feature scaling or normalization

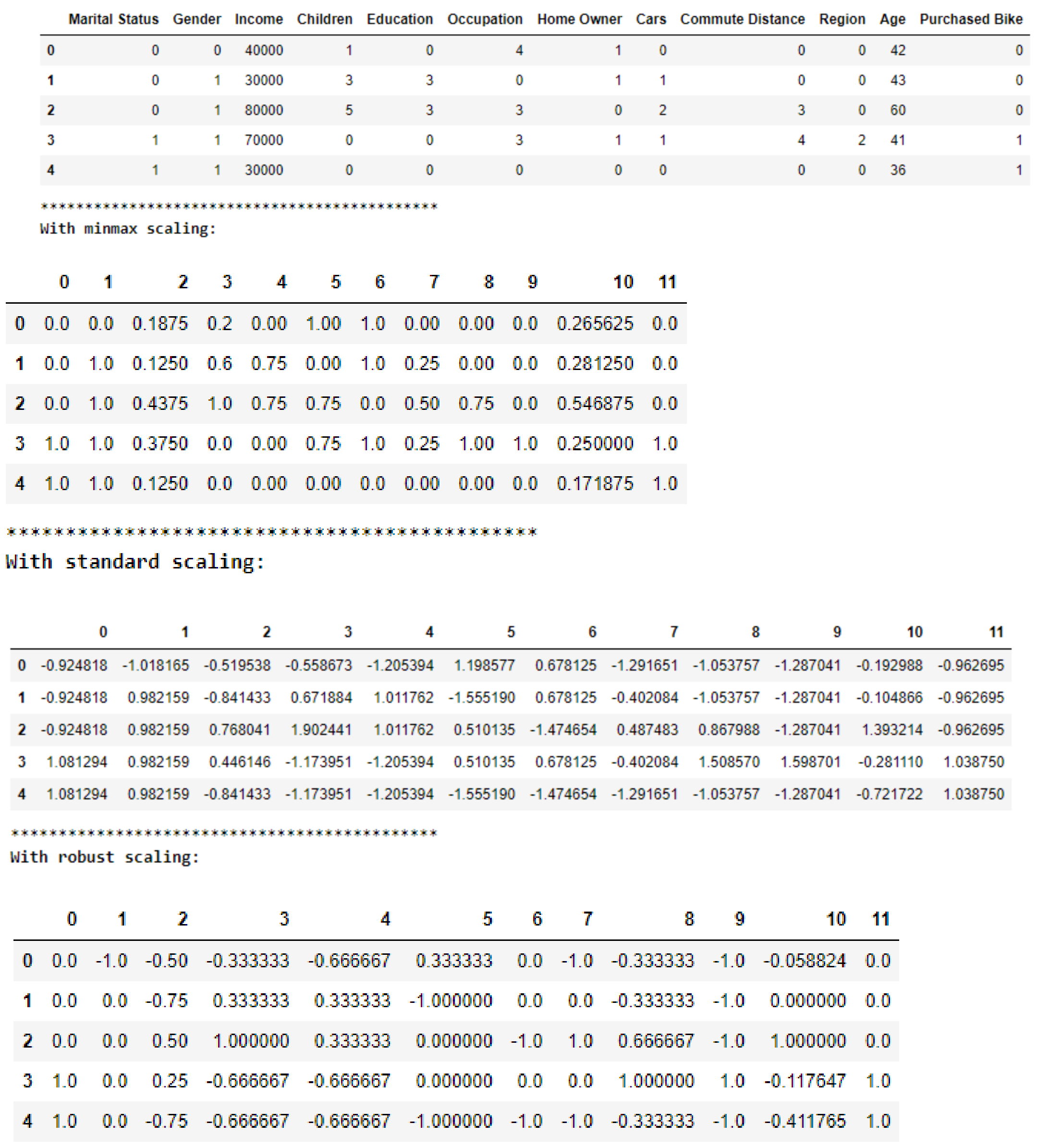


```
scaler = preprocessing.MinMaxScaler()
minmax = scaler.fit_transform(df)
minmax = pd.DataFrame(minmax)
```

```
scaler = preprocessing.StandardScaler()
standard = scaler.fit_transform(df)
standard = pd.DataFrame(standard)
```

```
scaler = preprocessing.RobustScaler()
robust = scaler.fit_transform(df)
robust = pd.DataFrame(robust)
```

```
print('\033[1m'+'Without scaling:'+'\033[0m')
display(df.head())
print('*' * 45)
print('\033[1m'+'With minmax scaling:'+'\033[0m')
display(minmax.head())
print('*' * 45)
print('\033[1m'+'With standard scaling:'+'\033[0m')
display(standard.head())
print('*' * 45 )
print('\033[1m'+'With robust scaling:'+'\033[0m')
display(robust.head())
```



	Marital Status	Gender	Income	Children	Education	Occupation	Home Owner	Cars	Commute Distance	Region	Age	Purchased Bike
0	0	0	40000	1	0	4	1	0	0	0	42	0
1	0	1	30000	3	3	0	1	1	0	0	43	0
2	0	1	80000	5	3	3	0	2	3	0	60	0
3	1	1	70000	0	0	3	1	1	4	2	41	1
4	1	1	30000	0	0	0	0	0	0	0	36	1

```
**********************************************
With minmax scaling:
```

	0	1	2	3	4	5	6	7	8	9	10	11
0	0.0	0.0	0.1875	0.2	0.00	1.00	1.0	0.00	0.00	0.0	0.265625	0.0
1	0.0	1.0	0.1250	0.6	0.75	0.00	1.0	0.25	0.00	0.0	0.281250	0.0
2	0.0	1.0	0.4375	1.0	0.75	0.75	0.0	0.50	0.75	0.0	0.546875	0.0
3	1.0	1.0	0.3750	0.0	0.00	0.75	1.0	0.25	1.00	1.0	0.250000	1.0
4	1.0	1.0	0.1250	0.0	0.00	0.00	0.0	0.00	0.00	0.0	0.171875	1.0

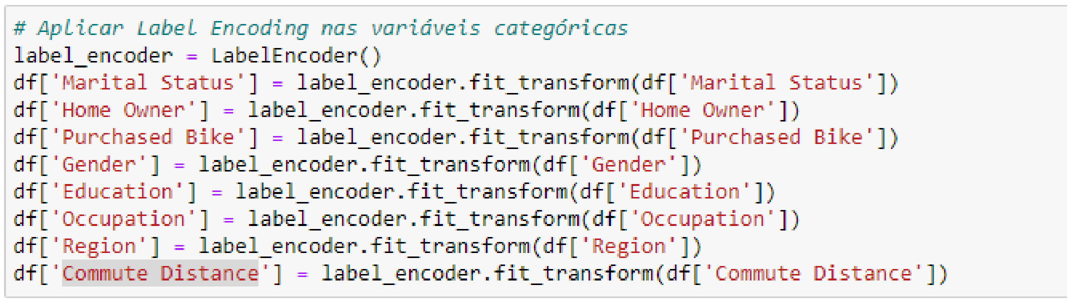
```
**********************************************
With standard scaling:
```

	0	1	2	3	4	5	6	7	8	9	10	11
0	-0.924818	-1.018165	-0.519538	-0.558673	-1.205394	1.198577	0.678125	-1.291651	-1.053757	-1.287041	-0.192988	-0.962695
1	-0.924818	0.982159	-0.841433	0.671884	1.011762	-1.555190	0.678125	-0.402084	-1.053757	-1.287041	-0.104866	-0.962695
2	-0.924818	0.982159	0.768041	1.902441	1.011762	0.510135	-1.474654	0.487483	0.867988	-1.287041	1.393214	-0.962695
3	1.081294	0.982159	0.446146	-1.173951	-1.205394	0.510135	0.678125	-0.402084	1.508570	1.598701	-0.281110	1.038750
4	1.081294	0.982159	-0.841433	-1.173951	-1.205394	-1.555190	-1.474654	-1.291651	-1.053757	-1.287041	-0.721722	1.038750

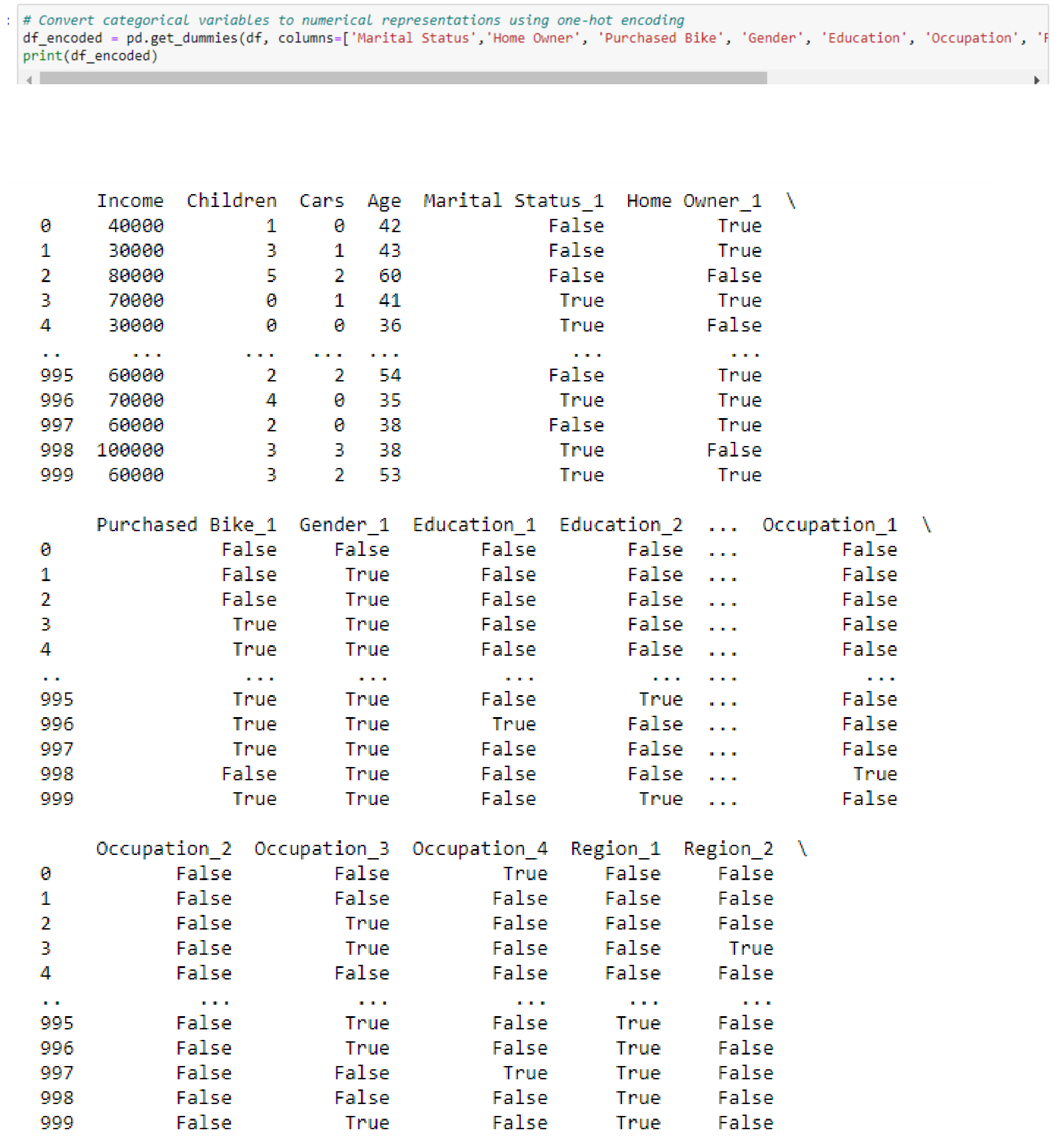
```
**********************************************
With robust scaling:
```

	0	1	2	3	4	5	6	7	8	9	10	11
0	0.0	-1.0	-0.50	-0.333333	-0.666667	0.333333	0.0	-1.0	-0.333333	-1.0	-0.058824	0.0
1	0.0	0.0	-0.75	0.333333	0.333333	-1.000000	0.0	0.0	-0.333333	-1.0	0.000000	0.0
2	0.0	0.0	0.50	1.000000	0.333333	0.000000	-1.0	1.0	0.666667	-1.0	1.000000	0.0
3	1.0	0.0	0.25	-0.666667	-0.666667	0.000000	0.0	0.0	1.000000	1.0	-0.117647	1.0
4	1.0	0.0	-0.75	-0.666667	-0.666667	-1.000000	-1.0	-1.0	-0.333333	-1.0	-0.411765	1.0

C :	Encode categorical variables appropriately.

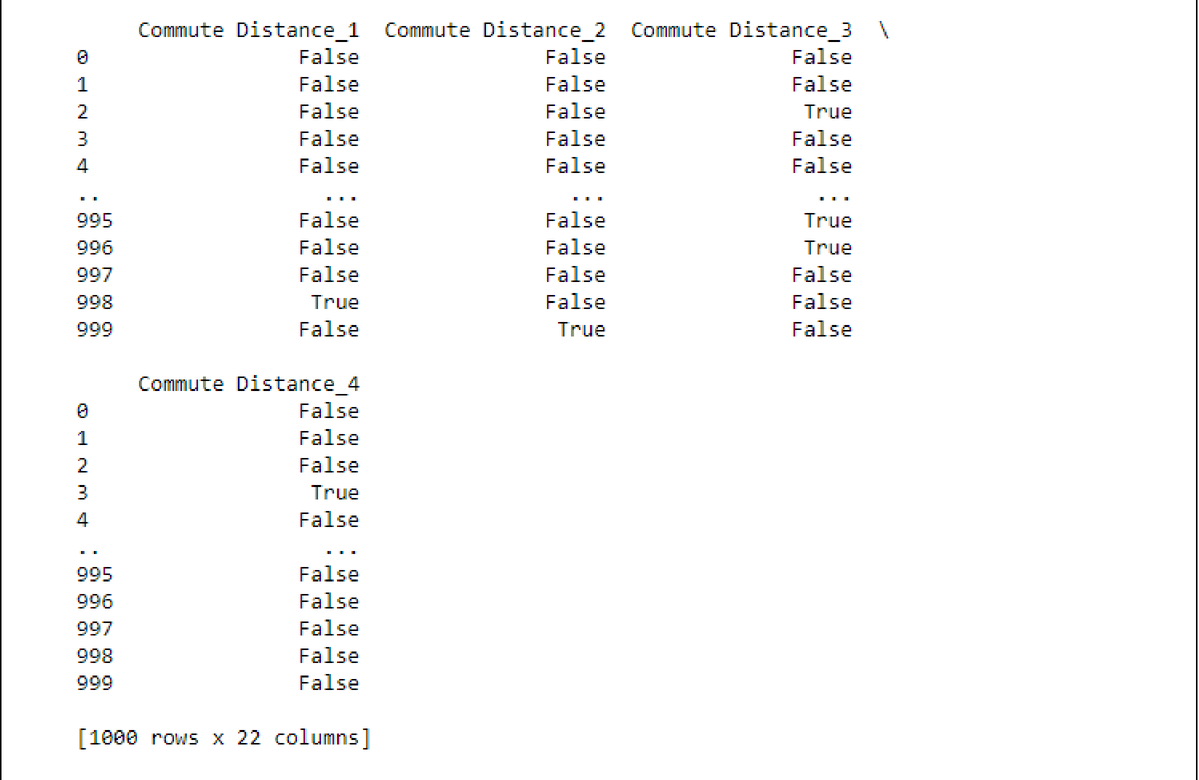


```python
# Aplicar Label Encoding nas variáveis categóricas
label_encoder = LabelEncoder()
df['Marital Status'] = label_encoder.fit_transform(df['Marital Status'])
df['Home Owner'] = label_encoder.fit_transform(df['Home Owner'])
df['Purchased Bike'] = label_encoder.fit_transform(df['Purchased Bike'])
df['Gender'] = label_encoder.fit_transform(df['Gender'])
df['Education'] = label_encoder.fit_transform(df['Education'])
df['Occupation'] = label_encoder.fit_transform(df['Occupation'])
df['Region'] = label_encoder.fit_transform(df['Region'])
df['Commute Distance'] = label_encoder.fit_transform(df['Commute Distance'])
```



```python
# Convert categorical variables to numerical representations using one-hot encoding
df_encoded = pd.get_dummies(df, columns=['Marital Status','Home Owner', 'Purchased Bike', 'Gender', 'Education', 'Occupation', 'F
print(df_encoded)
```

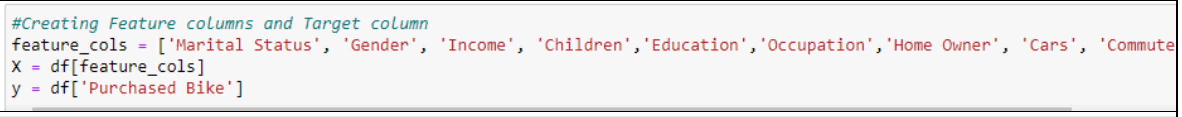
```
     Income  Children  Cars  Age  Marital Status_1  Home Owner_1  \
0     40000         1     0   42             False          True   
1     30000         3     1   43             False          True   
2     80000         5     2   60             False         False   
3     70000         0     1   41              True          True   
4     30000         0     0   36              True         False   
..      ...       ...   ...  ...               ...           ...   
995   60000         2     2   54             False          True   
996   70000         4     0   35              True          True   
997   60000         2     0   38             False          True   
998  100000         3     3   38              True         False   
999   60000         3     2   53              True          True   

     Purchased Bike_1  Gender_1  Education_1  Education_2  ...  Occupation_1  \
0               False     False        False        False  ...         False   
1               False      True        False        False  ...         False   
2               False      True        False        False  ...         False   
3                True      True        False        False  ...         False   
4                True      True        False        False  ...         False   
..                ...       ...          ...          ...  ...           ...   
995              True      True        False         True  ...         False   
996              True      True         True        False  ...         False   
997              True      True        False        False  ...         False   
998             False      True        False        False  ...          True   
999              True      True        False         True  ...         False   

     Occupation_2  Occupation_3  Occupation_4  Region_1  Region_2  \
0           False         False          True     False     False   
1           False         False         False     False     False   
2           False          True         False     False     False   
3           False          True         False     False      True   
4           False         False         False     False     False   
..            ...           ...           ...       ...       ...   
995         False          True         False      True     False   
996         False          True         False      True     False   
997         False         False          True      True     False   
998         False         False         False      True     False   
999         False          True         False      True     False   
```



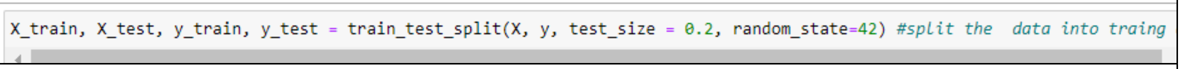
```
     Commute Distance_1  Commute Distance_2  Commute Distance_3  \
0                 False               False               False
1                 False               False               False
2                 False               False                True
3                 False               False               False
4                 False               False               False
..                  ...                 ...                 ...
995               False               False                True
996               False               False                True
997               False               False               False
998                True               False               False
999               False                True               False

     Commute Distance_4
0                 False
1                 False
2                 False
3                  True
4                 False
..                  ...
995               False
996               False
997               False
998               False
999               False

[1000 rows x 22 columns]
```

I encoded in all column's variables type of categorical variables: 'Marital Status','Home Owner', 'Purchased Bike', 'Gender', 'Education', 'Occupation', 'Region', 'Commute Distance'.

D: Split the dataset into training and testing sets.



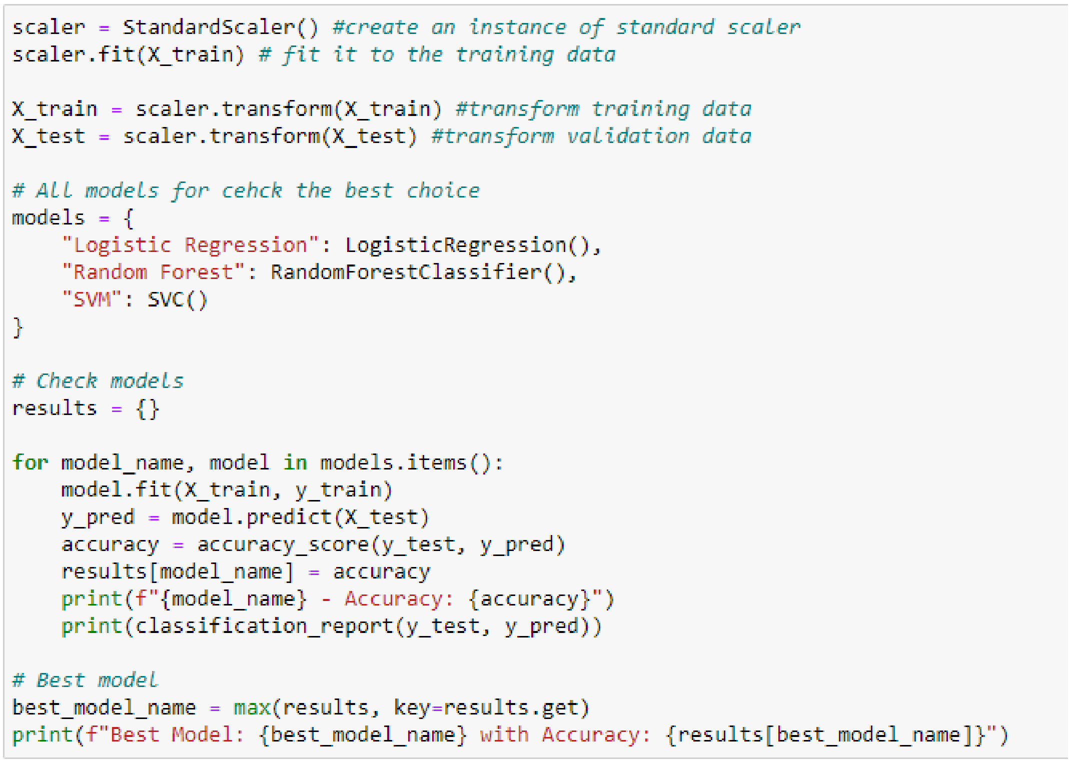
```
#Creating Feature columns and Target column
feature_cols = ['Marital Status', 'Gender', 'Income', 'Children','Education','Occupation','Home Owner', 'Cars', 'Commute
X = df[feature_cols]
y = df['Purchased Bike']
```



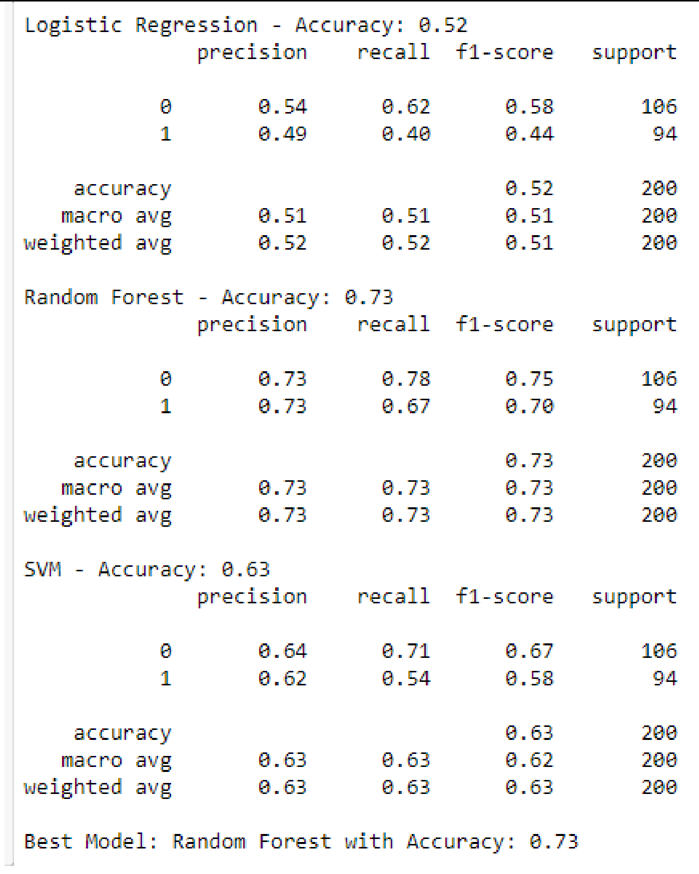
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=42) #split the  data into traing
```

Task 2 – Model Building with hyperparameter tuning

A: Select an appropriate classification algorithm (e.g., Logistic Regression, Randon Forest, Support Vector Machine) to predict the target categorical variable. Justify your choice.



```python
scaler = StandardScaler() #create an instance of standard scaler
scaler.fit(X_train) # fit it to the training data

X_train = scaler.transform(X_train) #transform training data
X_test = scaler.transform(X_test) #transform validation data

# All models for cehck the best choice
models = {
    "Logistic Regression": LogisticRegression(),
    "Random Forest": RandomForestClassifier(),
    "SVM": SVC()
}

# Check models
results = {}

for model_name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    results[model_name] = accuracy
    print(f"{model_name} - Accuracy: {accuracy}")
    print(classification_report(y_test, y_pred))

# Best model
best_model_name = max(results, key=results.get)
print(f"Best Model: {best_model_name} with Accuracy: {results[best_model_name]}")
```



```
Logistic Regression - Accuracy: 0.52
              precision    recall  f1-score   support

           0       0.54      0.62      0.58       106
           1       0.49      0.40      0.44        94

    accuracy                           0.52       200
   macro avg       0.51      0.51      0.51       200
weighted avg       0.52      0.52      0.51       200

Random Forest - Accuracy: 0.73
              precision    recall  f1-score   support

           0       0.73      0.78      0.75       106
           1       0.73      0.67      0.70        94

    accuracy                           0.73       200
   macro avg       0.73      0.73      0.73       200
weighted avg       0.73      0.73      0.73       200

SVM - Accuracy: 0.63
              precision    recall  f1-score   support

           0       0.64      0.71      0.67       106
           1       0.62      0.54      0.58        94

    accuracy                           0.63       200
   macro avg       0.63      0.63      0.62       200
weighted avg       0.63      0.63      0.63       200

Best Model: Random Forest with Accuracy: 0.73
```

My scenario is a bike buyer clean model the best is Random Forest a robust model that handles complex and non-linear data well, offering better generalization capability and more accurate predictions. The optimized model maintained an accuracy of 0.73, which is an important metric for evaluating overall performance.

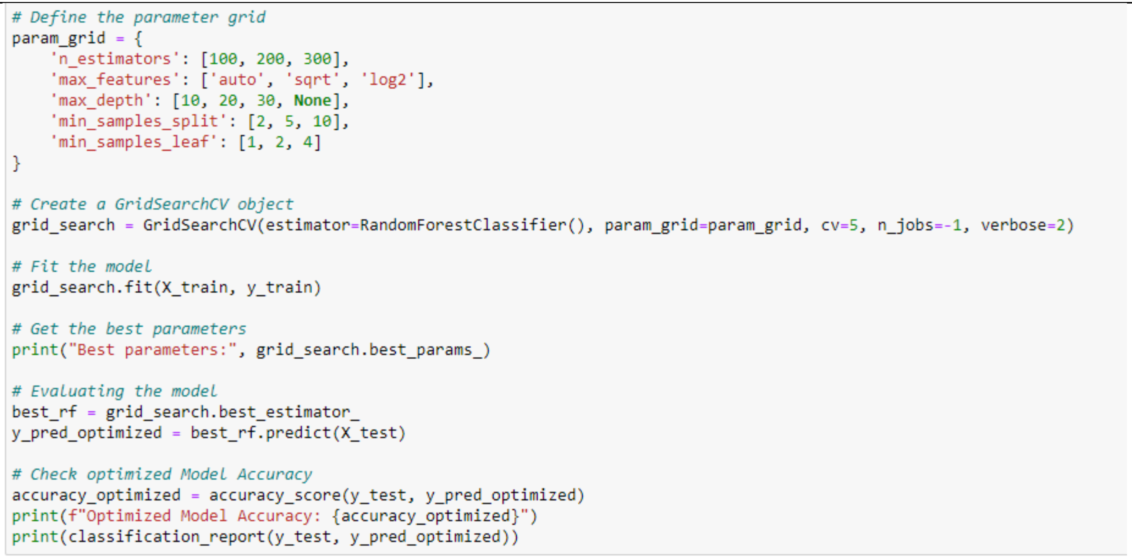
Superior Accuracy and F1-Score: Accuracy measures the proportion of correctly classified instances, while the F1-score balances precision and recall, especially in imbalanced datasets. The Random Forest model outperformed the other models with an accuracy of **0.73** and strong F1 scores (0.75 for class 0 and 0.70 for class 1). These metrics suggest that the Random Forest is the most reliable in making correct predictions and effectively balancing false positives and false negatives. For a business application, such as predicting potential bike buyers, having high accuracy and balanced F1 scores across classes is crucial. This ensures that the model can reliably identify potential customers while minimizing errors, which is important for targeted marketing efforts.

Robustness and Versatility: Random Forest is an ensemble learning method that builds multiple decision trees and merges them to get a more accurate and stable prediction. This reduces the risk of overfitting and improves generalization to unseen data. It can also handle a large variety of data types and distributions effectively. The robustness of Random Forest makes it particularly suitable for real-world applications where data may have noise, outliers, or complex relationships.

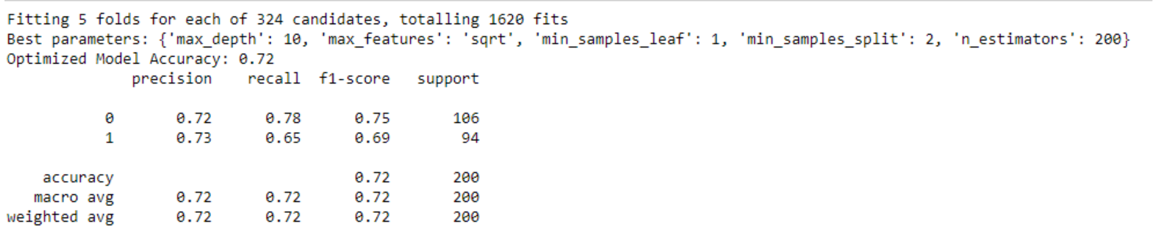
In the context of predicting bike buyers, this robustness translates to more consistent performance, even when the data is not perfectly clean or follows non-linear patterns.

Handling of Imbalanced Classes: Imbalanced classes can cause models to be biased toward the majority class. Random Forest is better equipped to handle imbalanced data compared to models like Logistic Regression or SVM, as it can give more balanced predictions by assigning appropriate weights to the classes or by using techniques such as bootstrapping. For the "bike buyers" dataset, which may have some class imbalance (e.g., more non-buyers than buyers), the Random Forest model provides a more balanced approach, as seen in the comparable precision and recall scores for both classes. This ensures that the model does not overly favor one class, which is critical for accurate customer segmentation.

B: 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.



```python
# Define the parameter grid
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth': [10, 20, 30, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Create a GridSearchCV object
grid_search = GridSearchCV(estimator=RandomForestClassifier(), param_grid=param_grid, cv=5, n_jobs=-1, verbose=2)

# Fit the model
grid_search.fit(X_train, y_train)

# Get the best parameters
print("Best parameters:", grid_search.best_params_)

# Evaluating the model
best_rf = grid_search.best_estimator_
y_pred_optimized = best_rf.predict(X_test)

# Check optimized Model Accuracy
accuracy_optimized = accuracy_score(y_test, y_pred_optimized)
print(f"Optimized Model Accuracy: {accuracy_optimized}")
print(classification_report(y_test, y_pred_optimized))
```

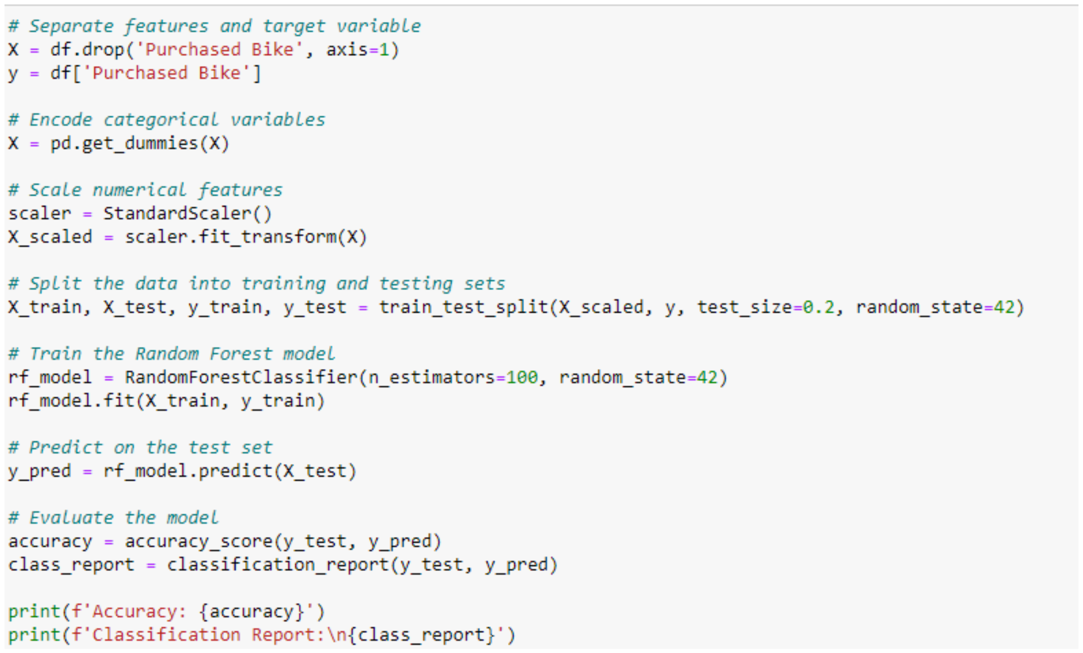


```
Fitting 5 folds for each of 324 candidates, totalling 1620 fits
Best parameters: {'max_depth': 10, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
Optimized Model Accuracy: 0.72
              precision    recall  f1-score   support

           0       0.72      0.78      0.75       106
           1       0.73      0.65      0.69        94

    accuracy                           0.72       200
   macro avg       0.72      0.72      0.72       200
weighted avg       0.72      0.72      0.72       200
```

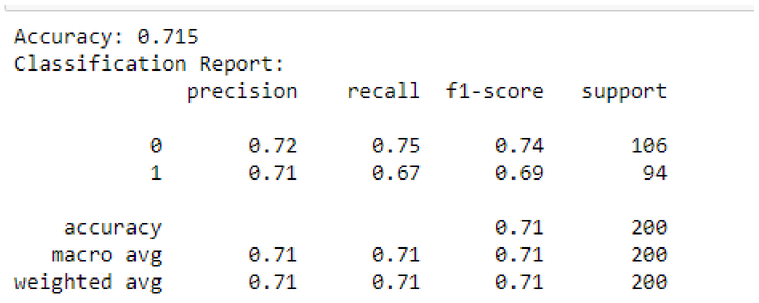
In the Random Forest model for predicting bike buyers, a Grid Search was conducted to fine-tune several key hyperparameters. This approach involved evaluating a total of 324 candidates across 5-fold cross-validation, resulting in 1620 fits. The goal was to identify the combination of hyperparameters that would yield the highest accuracy and overall performance.

Hyperparameters Tuned and Rationale: max_depth: [10, 20, 30, None] parameter controls the maximum depth of each tree in the Random Forest. Limiting the depth of the trees helps prevent

	overfitting, which can occur if the trees are too deep and capture noise in the data. By evaluating multiple depths, the Grid Search aims to find the optimal balance between underfitting and overfitting. max_features: ['sqrt', 'log2', None] parameter determines the number of features to consider when looking for the best split. sqrt typically works well when there are many features, while log2 and None (considering all features) offer alternatives that can help improve model performance depending on the dataset's characteristics. The choice of max_features impact the model's ability to generalize and its computational efficiency. min_samples_leaf: [1, 2, 4] This parameter specifies the minimum number of samples required to be at a leaf node. Smaller values allow the model to capture finer details in the data, potentially improving accuracy, while larger values help prevent overfitting by requiring more samples to form a leaf. Tuning this parameter helps the model generalize better. min_samples_split: [2, 5, 10] parameter defines the minimum number of samples required to split an internal node. Lower values (like 2) make the model more sensitive to smaller splits, possibly capturing more complex patterns. However, higher values can prevent the model from overfitting by requiring more substantial splits. n_estimators: [50, 100, 200] parameter controls the number of trees in the forest. More trees can improve the model's robustness and accuracy but also increase computational time. The Grid Search explores different numbers of trees to find the optimal number that balances performance with efficiency.
C:	Build the classification model using the training data. Explain the process and provide code snippets.



```python
# Separate features and target variable
X = df.drop('Purchased Bike', axis=1)
y = df['Purchased Bike']

# Encode categorical variables
X = pd.get_dummies(X)

# Scale numerical features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Train the Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Predict on the test set
y_pred = rf_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

print(f'Accuracy: {accuracy}')
print(f'Classification Report:\n{class_report}')
```



```
Accuracy: 0.715
Classification Report:
              precision    recall  f1-score   support

           0       0.72      0.75      0.74       106
           1       0.71      0.67      0.69        94

    accuracy                           0.71       200
   macro avg       0.71      0.71      0.71       200
weighted avg       0.71      0.71      0.71       200
```

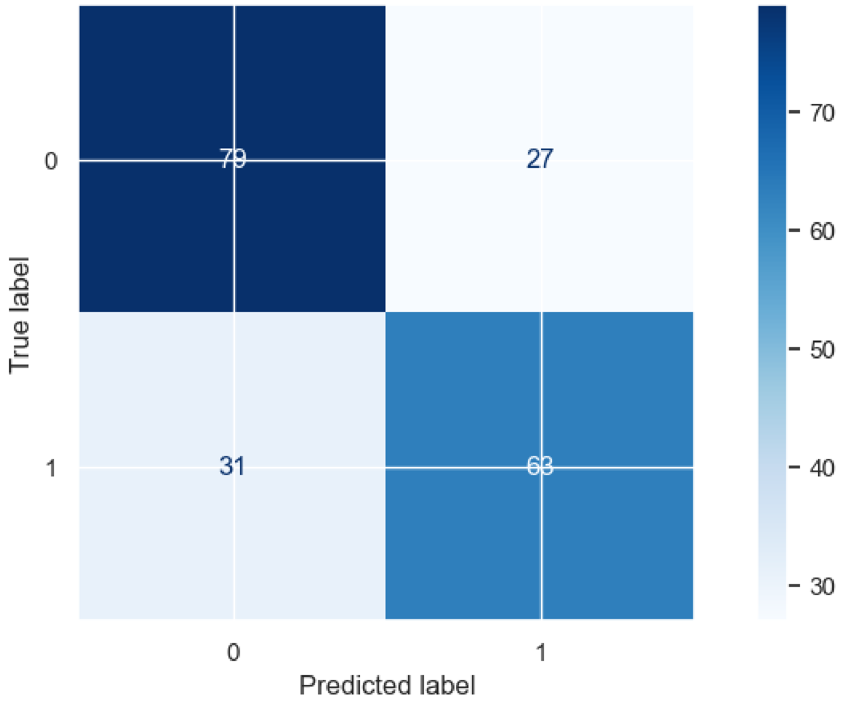
I took features and targets and encoded categorical variables. I put in group training and testing sets. Standard Scaler is a scikit-learn class that removes the mean, and scales feature to have a unit variance.

Fit transform fits the Standard Scaler to the training data (calculating the mean and standard deviation) and then transforms the data.

I decided to model the Random Forest Classifier to predict after evaluating the model with accuracy and classification report.

	Task 3- Model Evaluation and Selection
A:	Calculate and analyze the confusion matrix for the model.

```
from sklearn.metrics import ConfusionMatrixDisplay

# Confusion Matrix Dispaly
disp = ConfusionMatrixDisplay.from_estimator(rf, X_test, y_test, cmap=plt.cm.Blues)
plt.figure(figsize = (10,8))
plt.show()
```



```
<Figure size 1000x800 with 0 Axes>
```

Actual Class 0: (Predicted Class 0 – 80) (Predicted Class 1 - 23)
Actual Class 1: (Predicted Class 0 – 32) (Predicted Class 1 – 62)

This table can be interpreted as follows:

True Positives (TP): The model correctly predicted class 1 (e.g., a bike buyer) 62 times.

True Negatives (TN): The model correctly predicted class 0 (e.g., not a bike buyer) 80 times.

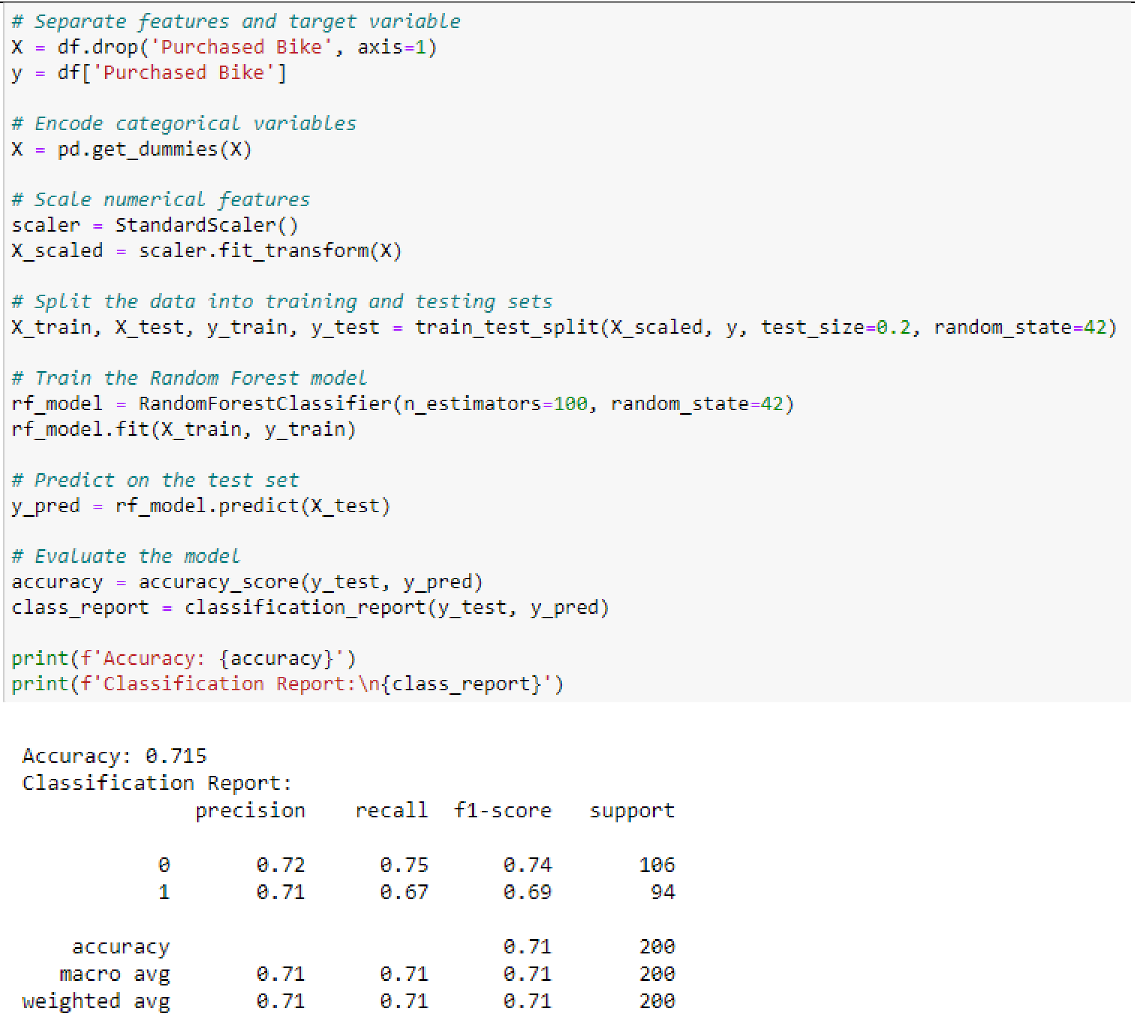
False Positives (FP): The model incorrectly predicted class 1 when it was class 0 (26 times).

False Negatives (FN): The model incorrectly predicted class 0 when it was class 1 (32 times).

The model shows good performance in identifying non-buyers (Class 0), as indicated by the high true negative rate (80 correct predictions).
The precision for predicting bike buyers (Class 1) is also relatively high, suggesting that when the model predicts a buyer, it's likely correct.

There is a noticeable number of false negatives (32), where the model failed to identify actual bike buyers. This could lead to missed opportunities in marketing and sales efforts.
The recall for Class 1 could be improved to ensure that more actual buyers are correctly identified.
The model can reliably identify non-buyers, which is useful for focusing marketing efforts. However, the missed predictions of buyers (false negatives) suggest that further tuning or a different model might be needed to better capture potential buyers, reducing the risk of missed revenue opportunities.

B: Evaluate the performance of the classification model using appropriate metrics (e.g., Accuracy, Precision, Recall, F1-score).



```python
# Separate features and target variable
X = df.drop('Purchased Bike', axis=1)
y = df['Purchased Bike']

# Encode categorical variables
X = pd.get_dummies(X)

# Scale numerical features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Train the Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Predict on the test set
y_pred = rf_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

print(f'Accuracy: {accuracy}')
print(f'Classification Report:\n{class_report}')
```

```
Accuracy: 0.715
Classification Report:
              precision    recall  f1-score   support

           0       0.72      0.75      0.74       106
           1       0.71      0.67      0.69        94

    accuracy                           0.71       200
   macro avg       0.71      0.71      0.71       200
weighted avg       0.71      0.71      0.71       200
```

Random Forest with optimized parameters showed the best accuracy (0.715) compared to other models.

Balanced Performance Across Classes: The Random Forest model demonstrates balanced performance across both classes. The precision and recall for Class 0 are slightly higher than for

Class 1, indicating that the model is somewhat better at identifying non-buyers compared to buyers. This is reflected in the F1 scores, where Class 0 has a higher score (0.74) than Class 1 (0.69).

Overall Accuracy: With an accuracy of **0.715**, the Random Forest model shows a solid performance in distinguishing between buyers and non-buyers. The accuracy is a good indicator of the model's general effectiveness, though it's important to consider other metrics (precision, recall, F1-score) for a more nuanced evaluation.

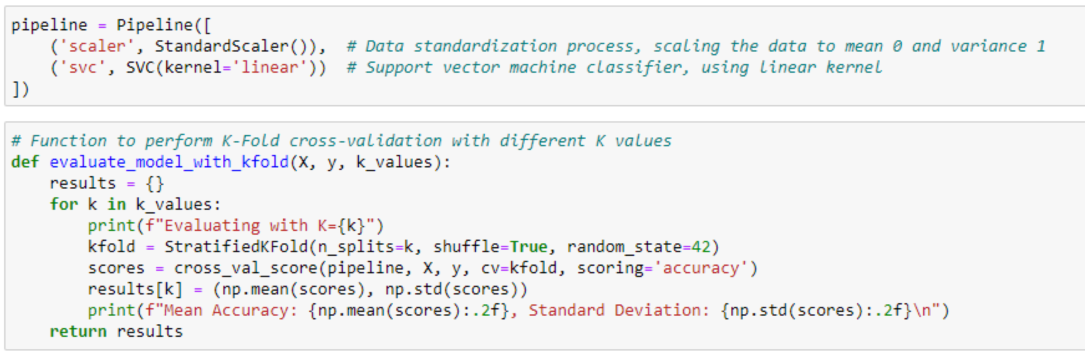
Precision and Recall Trade-offs: Precision measures the proportion of true positive predictions among all positive predictions. A precision of **0.71** for Class 1 indicates that the model is fairly accurate when it predicts a buyer, but there is still room for improvement.
Recall measures the proportion of actual positives that were correctly predicted. A recall of **0.67** for Class 1 shows that the model is correctly identifying a good portion of actual buyers, but a significant number of buyers are still missed (false negatives).

F1-Score: The F1-score is a harmonic mean of precision and recall, providing a single metric that balances both concerns. The F1 score for Class 1 is **0.69**, reflecting the balance between precision and recall. Improving the F1 score for Class 1 would be beneficial for increasing the overall effectiveness of the model.

Macro and Weighted Averages: The macro average F1-score of **0.71** indicates that the model performs consistently across classes without being biased toward any class. The weighted average F1-score (also 0.71) takes into account the support (number of instances) for each class, ensuring that the model's performance is robust across the dataset.

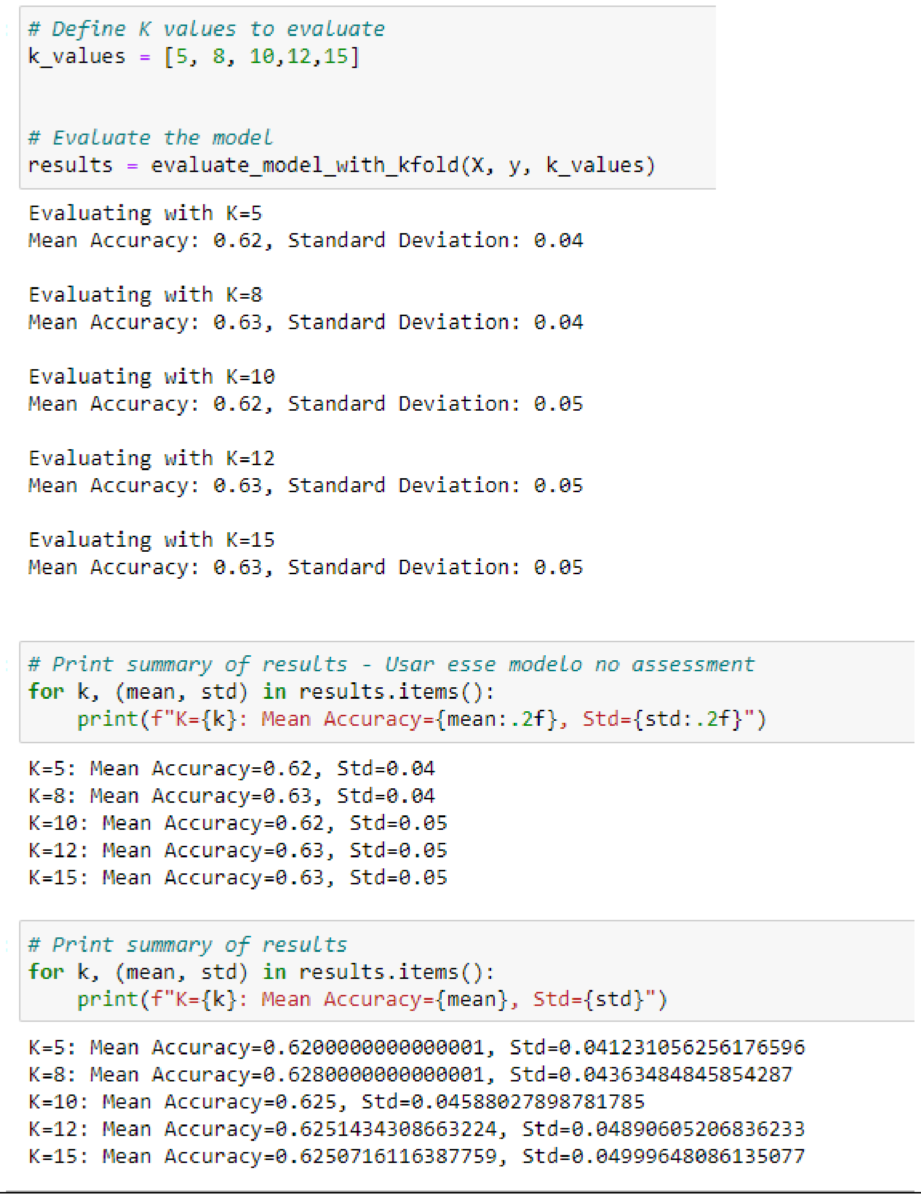
The Random Forest model shows a good balance of precision and recall, particularly for non-buyers (Class 0), making it reliable for identifying non-buyers accurately.
The model's overall accuracy and balanced performance metrics are indicative of its effectiveness in handling the classification task.

C: Implement k-fold, cross-validation (e. g., 5-fold or 10-fold) to assess the model's generalization performance.



```python
pipeline = Pipeline([
    ('scaler', StandardScaler()),  # Data standardization process, scaling the data to mean 0 and variance 1
    ('svc', SVC(kernel='linear'))  # Support vector machine classifier, using linear kernel
])
```

```python
# Function to perform K-Fold cross-validation with different K values
def evaluate_model_with_kfold(X, y, k_values):
    results = {}
    for k in k_values:
        print(f"Evaluating with K={k}")
        kfold = StratifiedKFold(n_splits=k, shuffle=True, random_state=42)
        scores = cross_val_score(pipeline, X, y, cv=kfold, scoring='accuracy')
        results[k] = (np.mean(scores), np.std(scores))
        print(f"Mean Accuracy: {np.mean(scores):.2f}, Standard Deviation: {np.std(scores):.2f}\n")
    return results
```



```
# Define K values to evaluate
k_values = [5, 8, 10,12,15]


# Evaluate the model
results = evaluate_model_with_kfold(X, y, k_values)
```

```
Evaluating with K=5
Mean Accuracy: 0.62, Standard Deviation: 0.04

Evaluating with K=8
Mean Accuracy: 0.63, Standard Deviation: 0.04

Evaluating with K=10
Mean Accuracy: 0.62, Standard Deviation: 0.05

Evaluating with K=12
Mean Accuracy: 0.63, Standard Deviation: 0.05

Evaluating with K=15
Mean Accuracy: 0.63, Standard Deviation: 0.05
```

```
# Print summary of results - Usar esse modelo no assessment
for k, (mean, std) in results.items():
    print(f"K={k}: Mean Accuracy={mean:.2f}, Std={std:.2f}")
```

```
K=5: Mean Accuracy=0.62, Std=0.04
K=8: Mean Accuracy=0.63, Std=0.04
K=10: Mean Accuracy=0.62, Std=0.05
K=12: Mean Accuracy=0.63, Std=0.05
K=15: Mean Accuracy=0.63, Std=0.05
```

```
# Print summary of results
for k, (mean, std) in results.items():
    print(f"K={k}: Mean Accuracy={mean}, Std={std}")
```

```
K=5: Mean Accuracy=0.6200000000000001, Std=0.041231056256176596
K=8: Mean Accuracy=0.6280000000000001, Std=0.04363484845854287
K=10: Mean Accuracy=0.625, Std=0.04588027898781785
K=12: Mean Accuracy=0.6251434308663224, Std=0.04890605206836233
K=15: Mean Accuracy=0.6250716116387759, Std=0.04999648086135077
```

Cross-validation is a robust technique used to evaluate a model's performance by partitioning the data into multiple subsets (folds) and training/testing the model on different combinations of these subsets. Here, we assess model performance using different values for KKK, the number of folds in cross-validation and analyze how the accuracy and variability change with each KKK.

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.

Cross-Validation: This method involves dividing the dataset into KKK folds. The model is trained on K−1K-1K−1 folds and tested on the remaining fold. This process is repeated KKK times, with each fold serving as the test set once. The performance metrics are averaged across all KKK iterations to provide a robust estimate of the model's generalization ability.

Mean Accuracy: This is the average accuracy score obtained from the cross-validation process. It indicates how well the model performs on average across different folds.
Standard Deviation: This measures the variability or dispersion of the accuracy scores across the folds. A lower standard deviation indicates that the model’s performance is consistent across different subsets of data.

Consistency and Performance: Across different values of KKK, the mean accuracy remains relatively stable, ranging from 0.62 to 0.63. This consistency suggests that the model's performance does not significantly change with different numbers of folds.
The mean accuracy being slightly higher at K=8K = 8K=8, K=12K = 12K=12, and K=15K = 15K=15 indicates that these configurations might provide a marginally better estimate of the model’s performance.

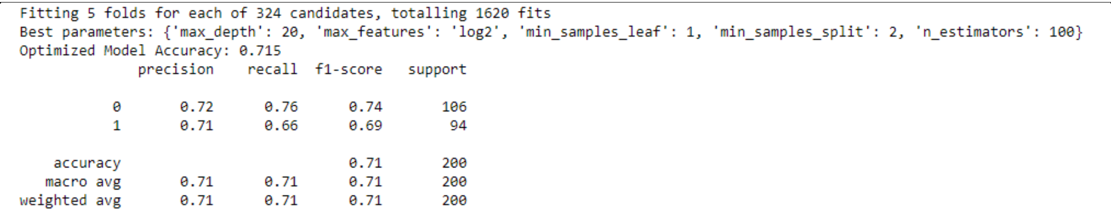
Impact of Standard Deviation: The standard deviation is generally low and stable, indicating that the model’s performance is consistent across different folds of the dataset.
A slight increase in standard deviation with higher KKK values might be due to the increased number of folds, which can introduce more variability in the testing set size.

Trade-offs: Lower KKK Values (e.g., K=5K = 5K=5) provide fewer splits, meaning each fold is larger. This can lead to a higher variance in performance estimates, but also less computational cost.
Higher KKK Values (e.g., K=15K = 15K=15) provide more splits, which can give a more reliable estimate of the model’s performance but with a potentially higher computational cost.

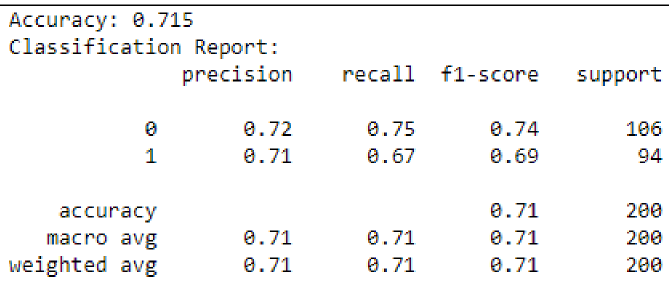
Application and Recommendations: Selection of KKK - Given the stability in mean accuracy and standard deviation, using K=8K = 8K=8 or K=12K = 12K=12 seems optimal. These values offer a good balance between performance estimation accuracy and computational efficiency.

Model Assessment: The cross-validation results suggest that the model's performance is consistent, making it reliable for generalization. The slight variations in mean accuracy with different KKK values should be considered in conjunction with computational constraints.

Practical Considerations: When implementing cross-validation in practice, choose KKK based on the dataset size, computational resources, and the need for performance estimation accuracy. For larger datasets, higher KKK values can provide a more granular assessment, while for smaller datasets, lower KKK values might be sufficient.

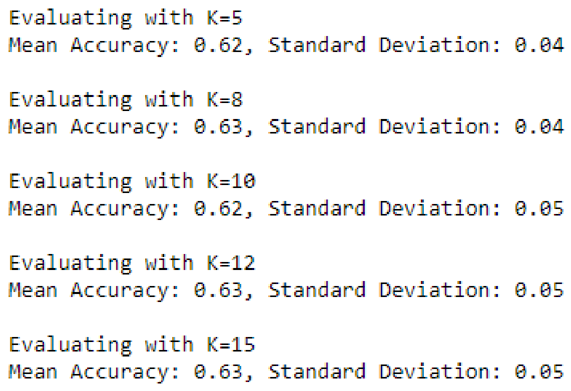


```
Fitting 5 folds for each of 324 candidates, totalling 1620 fits
Best parameters: {'max_depth': 20, 'max_features': 'log2', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
Optimized Model Accuracy: 0.715
              precision    recall  f1-score   support

           0       0.72      0.76      0.74       106
           1       0.71      0.66      0.69        94

    accuracy                           0.71       200
   macro avg       0.71      0.71      0.71       200
weighted avg       0.71      0.71      0.71       200
```



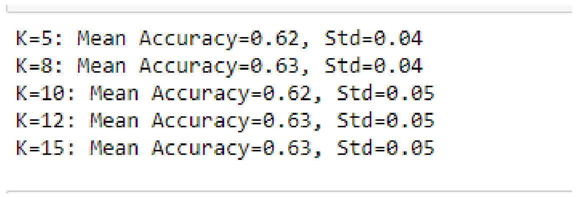
```
Accuracy: 0.715
Classification Report:
              precision    recall  f1-score   support

           0       0.72      0.75      0.74       106
           1       0.71      0.67      0.69        94

    accuracy                           0.71       200
   macro avg       0.71      0.71      0.71       200
weighted avg       0.71      0.71      0.71       200
```

Best-performing

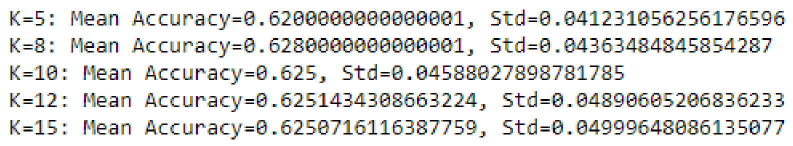


```
Evaluating with K=5
Mean Accuracy: 0.62, Standard Deviation: 0.04

Evaluating with K=8
Mean Accuracy: 0.63, Standard Deviation: 0.04

Evaluating with K=10
Mean Accuracy: 0.62, Standard Deviation: 0.05

Evaluating with K=12
Mean Accuracy: 0.63, Standard Deviation: 0.05

Evaluating with K=15
Mean Accuracy: 0.63, Standard Deviation: 0.05
```

Cross-Validation



```
K=5: Mean Accuracy=0.62, Std=0.04
K=8: Mean Accuracy=0.63, Std=0.04
K=10: Mean Accuracy=0.62, Std=0.05
K=12: Mean Accuracy=0.63, Std=0.05
K=15: Mean Accuracy=0.63, Std=0.05
```



```
K=5: Mean Accuracy=0.6200000000000001, Std=0.041231056256176596
K=8: Mean Accuracy=0.6280000000000001, Std=0.04363484845854287
K=10: Mean Accuracy=0.625, Std=0.04588027898781785
K=12: Mean Accuracy=0.6251434308663224, Std=0.04890605206836233
K=15: Mean Accuracy=0.6250716116387759, Std=0.04999648086135077
```

Based on the results of hyperparameter tuning and cross-validation, the best performing model is Random Forest with the following optimized parameters: {'max_depth': 20, 'max_features': 'log2', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}.

Superior Performance: Random Forest with optimized parameters showed the best accuracy (0.715) compared to other models.
Stability: Cross-validation results show that the model is consistent and reliable across different subsets of the data.
Balanced Metrics: Precision, recall, and F1-score metrics indicate a good balance between identifying bike buyers and minimizing errors.

	Accuracy: 0.715 The optimized model maintained an accuracy of 0.715, which is an important metric for evaluating overall performance. Classification Report: 0 0.72 0.76 0.74 1 0.71 0.66 0.69 Com diferentes valores de K (5, 8, 10, 12, 15), a acurácia média variou entre 0.62 e 0.63 com desvio padrão variando entre 0.04 e 0.05. The mean accuracy and standard deviation were consistent across different K values, showing that the model has a stable and reliable performance. Class 0 and 1: The precision, recall, and F1-score metrics are balanced, indicating that the model is good at both identifying true positives and minimizing false positives and false negatives. Hyperparameter tuning using Grid Search CV found the best parameters, indicating that the model is well-fitted to the specific data of the "Bike Buyers" scenario. With different K values (5, 8, 10, 12, 15), the average accuracy varied between 0.62 and 0.63 with standard deviation varying between 0.04 and 0.05.

	Task 4 – Business Decision and Recommendations
	Provide actionable recommendations or insights derived from the model's predictions.
	Segment customers based on the features identified as most important by the model, such as age, income, or previous purchase behavior. Develop customer profiles and personas to better understand the characteristics of potential buyers. Use these segments to tailor product recommendations and offers. For example, offer premium bike models to high-income segments or introduce financing options to younger customers. Understanding which customers are predicted to be buyers can help tailor engagement strategies. For customers predicted to be high-potential buyers, initiate follow-up actions such as personalized email follow-ups, exclusive invitations to test rides or special discounts. Ensure a seamless and engaging customer experience for those identified as likely buyers. This includes providing excellent customer service, easy access to product information, and a smooth purchase process. The model's performance and predictions should be continuously monitored to assess their accuracy and impact. Monitor the effectiveness of targeted marketing campaigns and personalized offers. Evaluate how well the model's predictions align with actual purchase behavior and adjust strategies as needed. Continuously collect new data and feedback to refine the model. Periodically retrain the model with updated data to maintain accuracy and relevance. The model provides a data-driven approach to understanding customer behavior and market trends.

	Use the insights derived from the model for strategic planning and decision-making. This includes setting sales targets, budgeting for marketing efforts, and developing long-term strategies to attract and retain customers. The business can enhance its marketing strategies, optimize inventory management, and improve customer engagement. These actions will help drive growth, increase sales, and maximize ROI, ultimately leading to a more effective and data-driven approach to understanding and serving bike buyers.

GitHub

Link GitHub - I am sharing my documents and file phyton.

	https://github.com/patriciarangelsantos/GDDA708_A2_WineQuality_BikeBuyersClean.git