

GDDA708 – Machine Learning and AI

16 August 2024

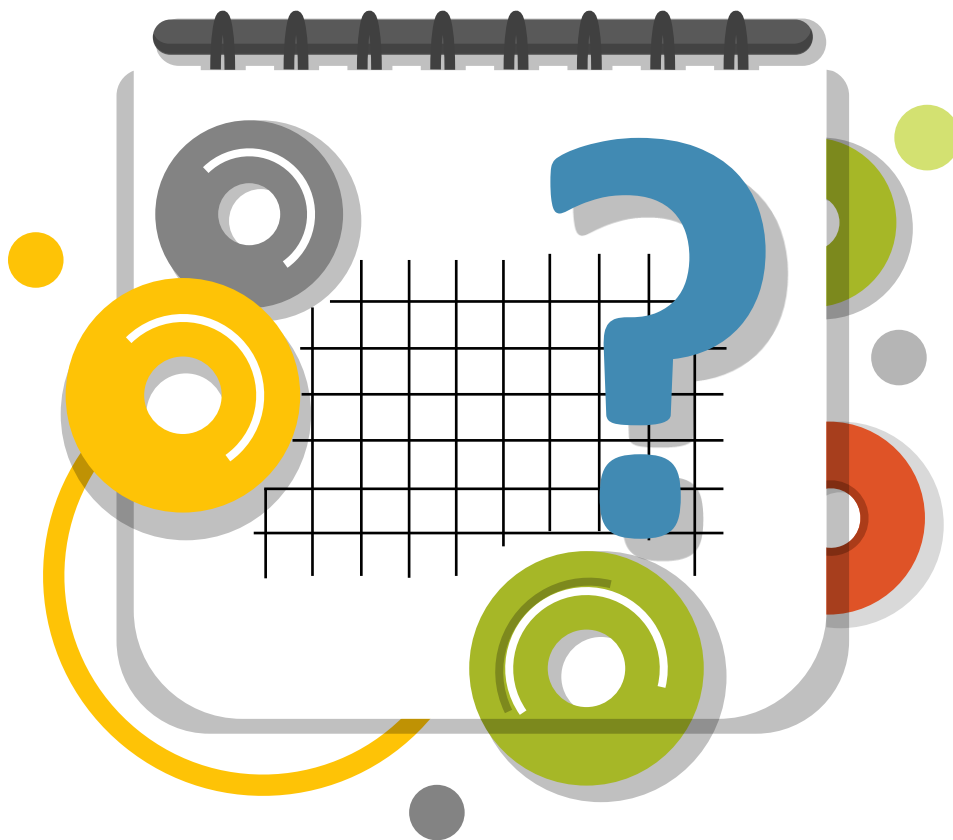


Table of Contents

GDDA708 – Machine Learning and AI	0
1. Introduction	2
Overview	2
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.....	2
Purpose	2
Part A – Regression modeling for business decision making	3
Task - 1 Data Preprocessing	3
Task 2 – Model Building with hyper-parameter tuning.....	6
Task 3 – Model Evaluation and Selection	10
Task 4 - Business Decision and Recommendations.....	13
Part B – Classification modeling for business decision-making.....	15
Task 1 – Data Preprocessing	15
Task 2 – Model Building with hyperparameter tuning.....	19
Task 3 - Model Evaluation and Selection	24
Task 4 – Business Decision and Recommendations.....	31

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

Task- 1 Data Preprocessing	
A	Properly clean the dataset, handle any missing values, and remove outliers.
	<pre> : # 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()) </pre>
	<p>Dataset Info:</p> <pre> <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 </pre>
	<p>Missing Values:</p> <pre> 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 </pre>
	<pre> print("Totally there are {} null values in the dataset".format(df.isnull().sum().sum())) </pre> <p>Totally there are 0 null values in the dataset</p>

```
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)
```

My data does not have null values in the first image above. I showed my dataset, all columns, and missing values. I didn't work on missing values after I renamed some columns.

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 dados, I have only numeric columns below:											
	fixed_acidity non-null float64 volatile_acidity non-null float64 citric_acid non-null float64 residual_sugar non-null float64 chlorides non-null float64 free_sulfur_dioxide non-null float64 total_sulfur_dioxide non-null float64 density non-null float64 pH non-null float64 sulphates non-null float64 alcohol non-null float64 quality non-null int64 1599 range I don't need to apply encoded categorial in my scenario.											

D	Split the dataset into training and testing sets.											
	<pre># Separate features and target variable X = df.drop('quality', axis=1) y = df['quality']</pre>											
	<pre># 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)</pre>											
	<pre># Linear Regression model</pre>											

Task 2 – Model Building with hyper-parameter tuning	
A	<p>Choose any appropriate regression models (e.g., Linear Regression, Ridge Regression, Lasso Regression)</p> <p>I need to get with the popular models: Linear Regression, Ridge Regression, and Lasso Regression. I will show you a little about the models in question: Linear Regression, Ridge Regression and Lasso Regression.</p> <p>Linear Regression is a model that shows us the linear relationship between characteristics such as acidity, sugar level and the main variable (wine quality).</p> <p>Ridge Regression is a regularized model of Linear Regression that includes a penalty on the size of the coefficients (L2 regularization). This penalty behaviour helps to contain overfitting by decreasing the coefficients.</p> <p>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.</p> <p>Lasso Regression also uses regularization but with the L1 penalty that contains some zero coefficients. This behaviour makes Lasso useful in feature selection because it does not consider irrelevant behaviours.</p> <p>My scenario is about wine quality red. First, I checked all dataset results about mean squared error - MSE and mean absolute error - MAE.</p> <p>I got the results below:</p> <p>Linear Regression</p> <pre>Mean Squared Error: 0.39002514396395427 Mean Absolute Error: 0.5057883195080454 R-squared: 0.4031803412796229</pre> <p>Lasso Regression</p> <pre>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</pre> <p>Ridge Regression</p> <pre>Best Parameters: {'alpha': 1} Best Cross-Validation Score: 0.43931582683684345 Mean Squared Error: 0.3929488678812798 Mean Absolute Error: 0.5057883195080454 R-squared: 0.39870643507758563</pre> <p>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.</p>

B	<p>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.</p> <pre> : # 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}") </pre> <p>Best Parameters: {'alpha': 100, 'solver': 'saga'}</p> <p>Optimized Ridge Regression - MSE: 0.3926958359356847, R²: 0.3990936266260089</p>
	<p>Model: The performance of the hyperparameter adjustment in Ridge Regression deals with the expected multicollinearity when the scenario contributes to the prediction. We can apply this model to others such as Lasso or Linear Regression.</p> <p>This option of the Ridge Regression model with the hyperparameters {'alpha': 100, 'solver': 'saga'} is possible due to its ability to effectively balance the complexity of the model and the accuracy in the context of wine quality.</p> <p>Business scenario analysis: Wine quality has multiple chemical properties that can influence, for example, acidity and sugar levels. Regression Ridge with strong alpha manages multicollinearity by decreasing the coefficients of the features with less impact, making the model more balanced and with effective responses.</p> <p>A regression Ridge with alpha=100 is an optimal choice for having precision in wine quality. With a robust approach that reduces overfitting due to regularization, it explores the data set in the best way with an optimal solver, thus reaching a balance between precision and prediction and achieving the ability to explain the truthfulness (R²). Making the selection justified with the precision of the wine quality by exploring the information of the chemical property.</p>

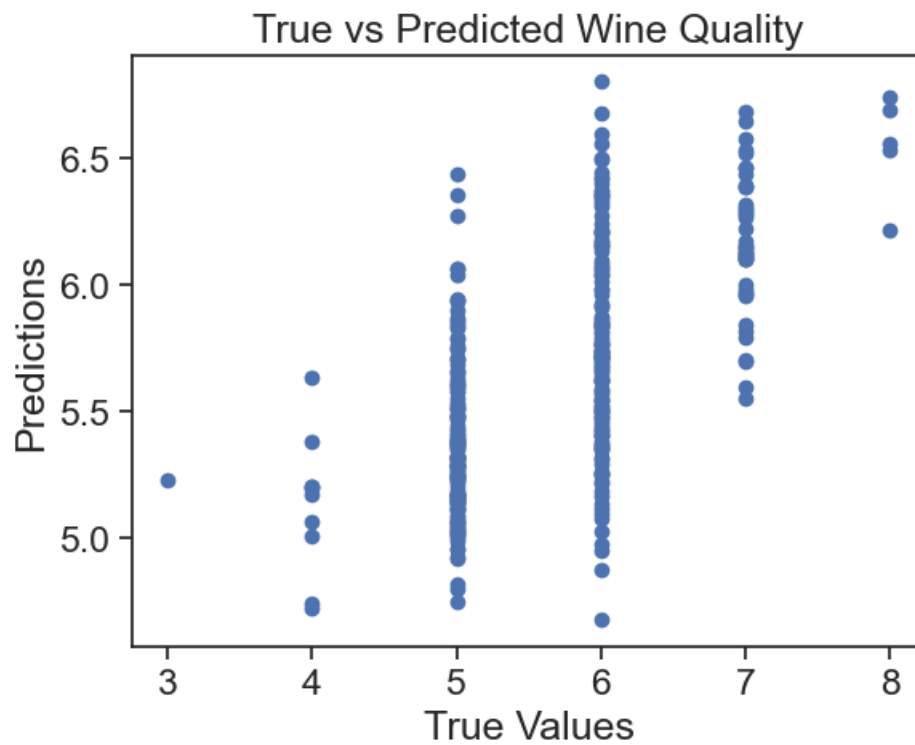
C	Build the regression models using the training data. Describe the process and provide code snippets.
	<pre> # Split the data into training and testing sets X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42) # Initialize the normalizer scaler = StandardScaler() # 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}') </pre>

```
# 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



Ridge Regression and its performance metrics: Mean Squared Error 39%, Mean Absolute Error 50%, and R-squared: 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.																								
	<p>Mean Squared Error: 0.3929488678812798 R-squared: 0.39870643507758563 Best Parameters: {'alpha': 1} Best Cross-Validation Score: 0.43931582683684345</p> <table> <thead> <tr> <th></th><th>Coefficient</th></tr> </thead> <tbody> <tr> <td>fixed_acidity</td><td>0.019614</td></tr> <tr> <td>volatile_acidity</td><td>-1.021529</td></tr> <tr> <td>citric_acid</td><td>-0.164270</td></tr> <tr> <td>residual_sugar</td><td>0.000624</td></tr> <tr> <td>chlorides</td><td>-1.226774</td></tr> <tr> <td>free_sulfur_dioxide</td><td>0.005686</td></tr> <tr> <td>total_sulfur_dioxide</td><td>-0.003561</td></tr> <tr> <td>density</td><td>-0.011153</td></tr> <tr> <td>pH</td><td>-0.376223</td></tr> <tr> <td>sulphates</td><td>0.746956</td></tr> <tr> <td>alcohol</td><td>0.297591</td></tr> </tbody> </table>		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
	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																								
	<p>Best Cross-Validation Score</p> <ul style="list-style-type: none"> • Value: 0.43931582683684345 • Interpretation: The cross-validation has the average performance at different points with the division of the data into observations. With the score between 0.4393, it shows me that the model has about 43.93 of variance in the data at different points. This indicates that the model is safe to use and generalizes to different subsets of information. <p>Mean Squared Error (MSE)</p> <ul style="list-style-type: none"> • Value: 0.3929488678812798 • Interpretation: The MSE checks the mean squared difference between the actual and predicted values. In my scenario, the MSE is 0.393, showing a mean squared error in the predictions. This suggests that the mean and the squared difference between the wine quality scores are 0.393. Lower MSEs show us more assertiveness in the prediction. • <p>Mean Absolute Error (MAE)</p> <ul style="list-style-type: none"> • Value: 0.5057883195080454 • Interpretation: The MAE shows the average absolute difference between the actual and predicted values. An MAE close to 0.506 shows us that the mean and the model predictions are in error at the wine quality scale points. The metric is understood as MSE because it shows us the average errors at the same points of the key variable. <p>R-squared (R^2)</p> <ul style="list-style-type: none"> • Value: 0.39870643507758563 • Interpretation: R^2 shows us the proportion of variance in the main variable (wine quality) that can be seen in the independent variables. An R^2 with a value of 0.399 shows us that 39.87% of the variation in wine quality can be analysed in the model. This value is not that 																								

	<p>high, but it shows us that we can still analyse it because there is still a relevant portion to be verified.</p> <p>Model Coefficients</p> <p>Fixed acidity: 0.019614 - There is a positive relationship with the quality of the wine, and we can understand that as fixed acidity increases, the quality of the wine also improves.</p> <p>Volatile acidity: -1.021529 - The volatile acidity variable is connected to the quality of the wine; if the acidity value increases, the quality of the wine decreases.</p> <p>Citric acid: -0.164270 - The negative connection shows us that higher citric acid levels decrease the quality of the wine.</p> <p>Residual sugar: 0.000624 - Any change in residual sugar levels has no relevance to the expected quality.</p> <p>chlorides: -1.226774 - A negative connection between higher chloride levels impacting wine quality scores</p> <p>Free sulfur dioxide: 0.005686 - A positive but low connection because if there is an increase in free sulfur dioxide it slightly improves the quality of the wine.</p> <p>Total sulphur dioxide: -0.003561 - A negative but low connection, if the total sulfur dioxide levels decrease the quality of the wine slightly.</p> <p>Density: -0.011153 - A negative connection with the density index is related to a small decrease in wine quality.</p> <p>PH: -0.376223 - A slight negative connection, pH levels (decreases acidity) impacts the quality of the wine.</p> <p>Sulfates: 0.746956 - A strong positive connection shows us that high sulfate levels are related to better wine quality.</p> <p>Alcohol: 0.297591 - A positive connection shows us that the higher the alcohol content, the better the quality of the wine.</p>
B :	Implement k-fold, cross-validation (e.g., 5-fold or 10-fold) to assess the model's generalization performance using multiple splits.

	<pre> # 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 </pre>
	<p>Accuracy shows us correct predictions according to the model. An accuracy of 0.58 shows us that 58% of the predictions are correct.</p> <p>The standard deviation increases slightly as K increases, going from 0.02 to 0.04. Even with a small increase, it impacts a change in the model's performance.</p> <p>The consistency of the average precision, which is 0.58, shows us that the KNN model is working regularly, regardless of the selected K value. This makes us understand that the data set or the problem point is not linked to the choice of K in the interval.</p>
C:	<p>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.</p>
	<pre> 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 </pre>
	<p>The Regression Ridge model with the best parameters {'alpha': 100, 'solver': 'saga'}, was chosen as the best option based on the results of hyperparameter adjustments and cross-validation. The model showed us a mean squared error (MSE) of 0.39 and an R² score of 0.39.</p>

	<p>High regularization strength (alpha=100): The alpha parameter of Ridge Regression tells us the degree of regularization. A high value indicates strong regularization, preventing overfitting, especially in multicollinearity scenarios. Overfitting is a common scenario in regression models that leads to poor performance. By selecting alpha=100, the model behaves more generically with new data, which is essential for making more assertive predictions.</p> <p>Solver Selection (Solver = 'saga'): The 'saga' solver performs very well in optimizing problems in a large scenario and can work with dense and sparse data sets. It can also be used with elastic network regularization, responding dynamically and robustly. This ensures that the model works efficiently and accurately, especially with large sets.</p> <p>Model Performance (MSE and R²): 0.3927 shows us the mean squared difference in actual and predicted wine quality scores. Low MSE gives us the understanding that impacts predictive accuracy and, in my scenario, an MSE of 0.3927 has the model predictions in alignment with the actual values.</p> <p>R² = 0.3991: shows us the variance in wine quality that the model can explain. An R² of 0.3991 explains that the model is accounting for approximately 39.91% of the variability in wine quality.</p> <p>Suitability for the business problem: The business objective is to manage wine quality based on chemical performances, encountering many challenges to maintain the same level of quality for all harvests. With the analysis together with the Ridge Regression model it fits this challenging scenario showing us more stable and interpretable results.</p> <p>The model used in the scenario helps us in the business by maintaining the accuracy of the results, thus being able to more easily manage the quality of the wine and being able to have quality control and adjust improvements in production in real-time. Generating value to customers and to the business with high quality in our processes.</p>
--	---

	Task 4- Business Decision and Recommendations
	Provide actionable recommendations derived from the model's predictions.
	<p>After analyzing the coefficient Ridge Prediction showed us. We can do better control in some variables.</p> <p>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.</p> <p>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.</p> <p>Although residual sugar and sulphur dioxide have minor impacts, they must be controlled to maintain the overall quality of the wine.</p> <p>Based on the model coefficients and quality predictions, we can make 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.</p> <p>By analyzing the Ridge Prediction coefficients, we can improve the control of some variables.</p>

	<p>Starting with volatile acidity, it needs to be managed and kept at low levels because it has a negative relationship with the quality of the control. Check the fermentation process. Manage and control the chloride content, leaving it at a lower level, because it has a negative impact on the quality of the wine.</p> <p>Manage the pH at good levels because it has a positive relationship with the quality of the wine. Slightly improving the sulfate levels and the alcohol content also has a positive relationship with the quality of the wine. The other variables, such as residual sugar and sulfur dioxide, have a lesser relationship, but they must be managed to continue to have an overall quality of the wine.</p> <p>After checking the model coefficients and the quality predictions, we will make decisions on the production and marketing of red wines.</p>
--	---

Part B – Classification modeling for business decision making

Task 1 – Data Preprocessing																																														
A:	Clean the dataset by handling missing values and removing outliers as needed.																																													
	<pre># 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())</pre> <p>Dataset Info:</p> <pre><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</pre> <p>Summary Statistics:</p> <table><tr><th></th><th>Income</th><th>Children</th><th>Cars</th><th>Age</th></tr><tr><td>count</td><td>1000.000000</td><td>1000.000000</td><td>1000.000000</td><td>1000.000000</td></tr><tr><td>mean</td><td>56140.000000</td><td>1.908000</td><td>1.452000</td><td>44.190000</td></tr><tr><td>std</td><td>31081.609779</td><td>1.626094</td><td>1.124705</td><td>11.353537</td></tr><tr><td>min</td><td>10000.000000</td><td>0.000000</td><td>0.000000</td><td>25.000000</td></tr><tr><td>25%</td><td>30000.000000</td><td>0.000000</td><td>1.000000</td><td>35.000000</td></tr><tr><td>50%</td><td>60000.000000</td><td>2.000000</td><td>1.000000</td><td>43.000000</td></tr><tr><td>75%</td><td>70000.000000</td><td>3.000000</td><td>2.000000</td><td>52.000000</td></tr><tr><td>max</td><td>170000.000000</td><td>5.000000</td><td>4.000000</td><td>89.000000</td></tr></table>		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
	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																																										

	<p>Missing Values:</p> <pre> 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 </pre>
	<pre> : 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') </pre>
	<p>The image shows us information about the datasets such as columns, types, range, and non-null. After we can see the summary statistics all columns with mean, 25%, 50%, min, max, and last print show us missing values in my dataset.</p>
B:	Perform feature scaling or normalization
	<pre> 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()) </pre>

	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.

```
# 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'])
```

```
: # 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', 'Region'])
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	

	<pre> 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] </pre>
	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.
:	
	<pre> #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'] </pre>
	<pre> X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=42) #split the data into training </pre>
	Here I can show the target column in my scenario I got to do X nine variables (marital status, gender, income, children, education, occupation, homeowner, cars commute), and Y I got only purchased bike.

	Task 2 – Model Building with hyperparameter tuning
A:	Select an appropriate classification algorithm (e.g., Logistic Regression, Random Forest, Support Vector Machine) to predict the target categorical variable. Justify your choice.

```
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 cekck 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]}")
```

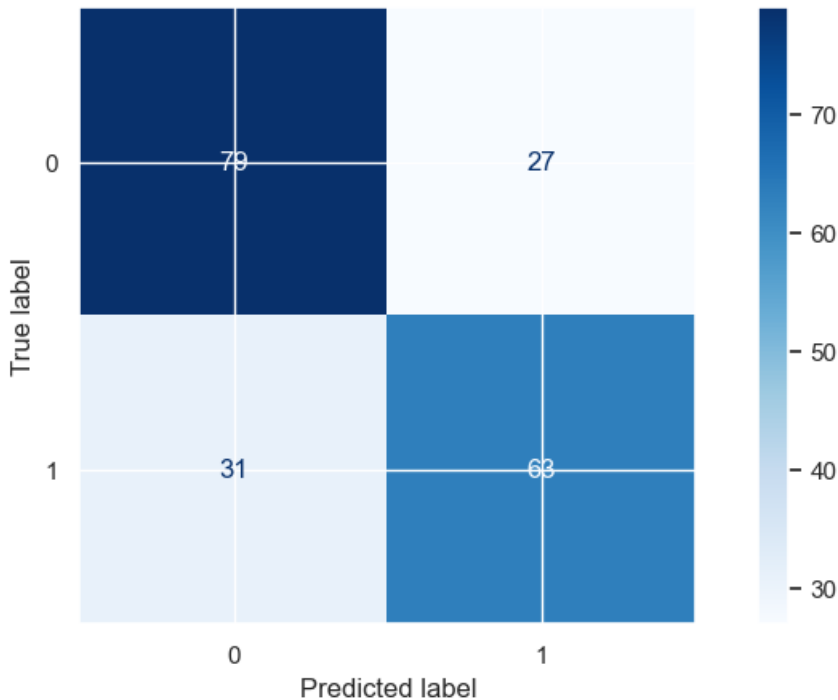
	Logistic Regression - Accuracy: 0.52				
		precision	recall	f1-score	support
	0	0.54	0.62	0.58	106
	1	0.49	0.40	0.44	94
	accuracy			0.52	200
	macro avg	0.51	0.51	0.51	200
	weighted avg	0.52	0.52	0.51	200
	Random Forest - Accuracy: 0.73				
		precision	recall	f1-score	support
	0	0.73	0.78	0.75	106
	1	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				
	<p>My dataset is a clean bicycle buyer model, fitting the Random Forest model, being robust and able to handle complex and non-linear data, being able to be more generalist and make more accurate predictions. The best model performed with an accuracy of 0.73, which is a good metric for analyzing overall performance.</p> <p>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.</p> <p>Superior Accuracy and F1-Score: Accuracy measures the number of correctly classified scenarios, now in the F1 score it maintains the division between precision and recall, in imbalanced data sets. The Random Forest model performed best with an accuracy of 0.73 and strong F1 scores (0.75 for group 0 and 0.70 for class 1). These values show us that Random Forest is the most assertive for correct predictions and balancing false positives and false negatives. For a commercial environment, we can predict potential bicycle buyers, having high accuracy and</p>				

	<p>balanced F1 scores in the groups is essential. Ensuring that the model can show us potential customers assertively, reducing errors, which is essential for marketing-focused work.</p> <p>Handling of Imbalanced Classes: Unbalanced groups can show us that the models are biased towards the majority groups. Random Forest is better prepared to deal with unbalanced data compared to other models such as Logistic Regression or SVM. For the “bike buyers” scenario, where there may be an imbalance in the groups, the Random Forest model has a more balanced behaviour, as already mentioned in the precision and recall scores. This shows us that the model will not favor a certain group in the data.</p>																														
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.																														
	<pre># 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))</pre> <p>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</p> <table><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr><tr><td>0</td><td>0.72</td><td>0.78</td><td>0.75</td><td>106</td></tr><tr><td>1</td><td>0.73</td><td>0.65</td><td>0.69</td><td>94</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.72</td><td>200</td></tr><tr><td>macro avg</td><td>0.72</td><td>0.72</td><td>0.72</td><td>200</td></tr><tr><td>weighted avg</td><td>0.72</td><td>0.72</td><td>0.72</td><td>200</td></tr></table>		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
	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																											
	<p>For the Random Forest model to generate the bicycle buyer prediction, a grid search was included to adjust several key hyperparameters. This context generated an evaluation of 324 candidates in a 5-fold cross-validation, resulting in 1620 adjustments. To select the combination of hyperparameters that would generate the highest performance accuracy.</p> <p>Hyperparameters Tuned and Rationale: max_depth: [10, 20, 30, None] parameter manages the maximum size of each Random Forest tree. Setting the depth of the trees helps us combat overfitting. By evaluating the depths, Grid Search aims to have a perfect balance between underfitting and overfitting.</p>																														

	<p>max_features: ['sqrt', 'log2', None] The parameter defines the number of features that will be analyzed when searching for the best split. sqrt works well when there are many features, while log2 and None (evaluating all features) can offer us possibilities that help us have a favorable performance in the model depending on the data set. The selection of max_features impacts the performance of the model to generalize and its efficiency.</p> <p>min_samples_leaf: [1, 2, 4] The parameter has the minimum number of samples to be in a leaf. Lower values allow us to capture finer details in the data, improving accuracy, while higher values can help us avoid overfitting by requiring more samples to generate a leaf. Balancing this parameter helps us to effectively manage the model.</p> <p>min_samples_split: [2, 5, 10] The parameter shows us the minimum number of samples needed to divide an internal node. Minimum values show that the model is more sensitive to smaller divisions, thus accessing more complex patterns. However, higher values can prevent the model from overfitting, requiring more divisions.</p> <p>n_estimators: [50, 100, 200] The parameter manages the number of trees in the forest. More trees improve robustness and model accuracy but may take longer to process. Grid Search works with different numbers of trees to find the number that balances performance with efficiency.</p>
C:	Build the classification model using the training data. Explain the process and provide code snippets.
	<pre> # 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}') </pre>

	<pre> 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 0.71 0.71 200 macro avg 0.71 0.71 0.71 200 weighted avg 0.71 0.71 0.71 200 </pre>				
	<p>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.</p> <p>Fit transform fits the Standard Scaler to the training data (calculating the mean and standard deviation) and then transforms the data.</p> <p>I decided to model the Random Forest Classifier to predict after evaluating the model with accuracy and classification report.</p>				

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

	<pre>from sklearn.metrics import ConfusionMatrixDisplay # Confusion Matrix Display disp = ConfusionMatrixDisplay.from_estimator(rf, X_test, y_test, cmap=plt.cm.Blues) plt.figure(figsize = (10,8)) plt.show()</pre>  <p>The figure is a confusion matrix for a binary classification task. The y-axis is labeled 'True label' with categories 0 and 1. The x-axis is labeled 'Predicted label' with categories 0 and 1. The matrix cells contain the following counts: True Positives (TP) = 62, True Negatives (TN) = 80, False Positives (FP) = 26, and False Negatives (FN) = 32. A color bar on the right indicates the count scale from 30 to 70.</p> <table><tr><th></th><th>Predicted 0</th><th>Predicted 1</th></tr><tr><th>True 0</th><td>80</td><td>26</td></tr><tr><th>True 1</th><td>32</td><td>62</td></tr></table> <p><Figure size 1000x800 with 0 Axes></p>		Predicted 0	Predicted 1	True 0	80	26	True 1	32	62
	Predicted 0	Predicted 1								
True 0	80	26								
True 1	32	62								
	<p>Actual Group 0: (Predicted Class 0 – 80) (Predicted Group 1 - 26)</p> <p>Actual Group 1: (Predicted Class 0 – 32) (Predicted Group 1 – 62)</p> <p>This table can be interpreted as follows:</p> <p>True Positives (TP): The model correctly collected group 1 (e.g. a bicycle buyer) 62 times.</p> <p>True Negatives (TN): The model correctly collected group 0 (e.g. not a bicycle buyer) 80 times.</p> <p>False Positives (FP): The model incorrectly collected group 1 when it was group 0 (26 times).</p> <p>False Negatives (FN): The model incorrectly collected group 0 when it was group 1 (32 times).</p> <p>The model can show us that it performed well in collecting customers who did not buy (Group 0), just as it shows in the indication of the high rate of true negatives (80 correct predictions). The accuracy in predicting bicycle buyers (Group 1) is high, showing us that when the model predicts a buyer, it is likely to be correct.</p> <p>There are a reasonable number of false negatives (32), in this scenario, the model failed to collect real bike buyers. We understand that these were missed opportunities for marketing and sales efforts.</p>									

	<p>The recall for Group 1 should be changed to ensure that we are collecting real buyers correctly. The model effectively shows non-buyers, which is very useful for marketing efforts. With the missed buyer predictions (false negatives) we are shown that further adjustments or that a different model may be necessary to show us potential buyers and reduce the risk of missed opportunities.</p>
B:	<p>Evaluate the performance of the classification model using appropriate metrics (e.g., Accuracy, Precision, Recall, F1-score).</p>
	<pre> # 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}') </pre> <pre> 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 macro avg 0.71 weighted avg 0.71 </pre> <p>Random Forest with optimized parameters showed the best accuracy (0.715) compared to other models.</p>
	<p>Balanced Performance Across Classes: Random Forest shows us balanced performance across groups. The precision and recall for Group 0 are slightly higher than Group 1, showing us that the model is slightly better at picking up non-buyers compared to buyers. This can be seen in the F1 scores, where Group 0 has a higher score (0.74) than Group 1 (0.69).</p>

	<p>Overall Accuracy: With an accuracy of 0.715, Random Forest performs effectively in collecting data from buyers and non-buyers. Accuracy is a good indicator of model performance, although it is interesting to analyze other metrics (precision, recall, F1 score) for a more effective analysis.</p> <p>Precision and Recall Trade-offs: Check the proportion of true positive predictions out of all positive predictions. An accuracy of 0.71 for Group 1 shows that the model is good at predicting a buyer, but still has room for improvement.</p> <p>Recall checks the proportion of true positives that were correctly identified. A recall of 0.67 for Group 1 shows that the model is correctly capturing a good portion of the true buyers, but we still have a number of buyers that are still missing (false negatives).</p> <p>F1-Score: The F1-score is a harmonic average of precision and recall, giving us a single metric that keeps all concerns at bay. The F1-score for Group 1 is 0.69, showing a balance between precision and recall. Changing the F1-score for Group 1 would benefit the model's effectiveness.</p> <p>Macro and Weighted Averages: The average macro F1 score of 0.71 shows us that the model performed robustly across all groups without being influenced by any one group. The average weighted F1 score (also 0.71) can be considered the support (number of instances) for each group, thus performing robustly across the entire dataset.</p> <p>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.</p> <p>The model's overall accuracy and balanced performance metrics are indicative of its effectiveness in handling the classification task.</p> <p>Random Forest has a good and balanced performance in precision and recall, especially for non-buyers (Group 0), proving to be reliable in accurately collecting non-buyers. The overall accuracy of the model and the balanced metrics found in the indices show effective performance in classification tasks.</p>
C:	Implement k-fold, cross-validation (e. g., 5-fold or 10-fold) to assess the model's generalization performance.
	<pre> 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]) # 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 </pre>

	<pre># 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</pre> <p>Cross-validation is a robust technique used to verify the performance of a model by separating the data into multiple subsets (folds) and training/testing the model on different possibilities of these subsets. We will evaluate the performance of the model with different values for KKK, the number of folds in the cross-validation, and analyze how the accuracy and variability change with each KKK.</p>
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 focuses on splitting the dataset into KKK folds. The model is trained on K-1K-1K-1 folds and tested on the remaining fold. This flow is repeated KKK many times, with

<p>each fold serving as the test set once. The performance metrics are averaged over KKK iterations to show us a robust estimate that can generalize the model.</p> <p>Mean Accuracy: This is the average accuracy score that can be obtained by the cross-validation process. It shows us how good the performance is across different folds</p> <p>Standard Deviation: Mede a variabilidade ou dispersao das pontuacoes de precisao nas dobras. Um desvio padrao mais baixa nos mostra que o desempenho do modelo e firme em diferentes subconjuntos de dados.</p> <p>Consistency and Performance: With different KKK values, the average accuracy remains with constant behavior, with a variation from 0.62 to 0.63. This movement suggests that the model performance does not change with the different number of folds. The average accuracy being slightly higher at K=8K = 8K=8, K=12K = 12K=12, and K=15K = 15K=15 shows us that these parameters can provide a better estimate than the model performance.</p> <p>Impact of Standard Deviation: The standard deviation is slightly lower and stable, showing us that the model performance is robust across different folds of the dataset. A slight increase in standard deviation with higher KKK values may be due to the larger number of folds, which can bring us more variety in the test set size.</p> <p>Trade-offs: Lower KKK values (e.g. K=5K = 5K=5) give us smaller divisions, so we can understand that each fold is larger. This can lead to a variance in performance estimates but with less computational cost. High values (for example, K=15K = 15K=15) give us more divisions, which can provide us with a more effective estimate of the model's performance, but with a higher computational cost.</p> <p>Application and Recommendations: KKK Selection – With the stability in mean accuracy and standard deviation, we can use K=8K = 8K=8 or K=12K = 12K=12 which is optimal. These values show us the balance between the accuracy of the performance estimation and the computational efficiency.</p> <p>Model Assessment: The cross-validation results show that the model performs robustly and is reliable for generalization. The few variations in mean accuracy with different KKK values should be considered in conjunction with computational constraints.</p> <p>Practical Considerations: By implementing cross-validation with practice, we select the KKK based on the size of the dataset, computational resources, and the accuracy of the performance estimation. With large datasets, larger KKK values give us a more granular evaluation, but for smaller datasets, low KKK values are sufficient.</p>
--

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: With Random Forest with optimized parameters, we can understand the best precision (0.715) when checking with the other models.

Stability: From the cross-validation results we can verify that the model is effective and reliable in different data subsets.

Balanced Metrics: Precision, recall, and F1 score metrics show us a balance between collecting bike buyers and reducing errors.

Accuracy: 0.715 0.715 The optimized model can maintain an accuracy of 0.715 an important metric for performance evaluation.

Classification Report: 0 0.72 0.76 0.74
1 0.71 0.66 0.69

With all these values of 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.

The average accuracy and standard deviation were robust across all K value scenarios, we can see that the model performs effectively and reliably.

Group 0 and 1: Group 0 and 1: The precision, recall, and F1 score metrics are balanced, we can see that the model is good both in collecting true positives and in reducing false positives and false negatives.

Hyperparameter tuning using Grid Search CV searched for the best parameters, showing us that the model is well adjusted to the data within the “Bicycle Buyers” scenario.

With all these values of 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.
	<p>Customer segments are based on features collected as important by the model, such as age, income, or previous purchasing behavior. Develop analyses on customer profiles to check the behavior of potential buyers. Use these segments to recommend and offer products. For example, you can offer the best bicycle models to high-income customers or make it easier for younger customers to buy a bicycle.</p> <p>Study and understand which customers will be potential buyers and apply engagement strategies. For potential buyers, we can start a follow-up with an e-mail, exclusive test ride invitations, or special offers.</p>

	<p>Offering a seamless customer experience focused on the profiles collected as potential buyers. Including excellent customer service, easy access to product information, and a transparent and efficient purchasing process.</p> <p>Model analyses must be monitored, the necessary changes applied and their impact accurately assessed.</p> <p>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.</p> <p>The model provides a data-driven approach to understanding customer behavior and market trends.</p> <p>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.</p> <p>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.</p>

GitHub

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

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

Reference

Link more information below about wine quality and fields another link about bike buyers clean.

	https://www.kaggle.com/datasets/ruthgn/wine-quality-data-set-red-white-wine
	https://medium.com/batech/bike-buyers-analysis-fa09a9e5d730