COMP309 Assignment 4

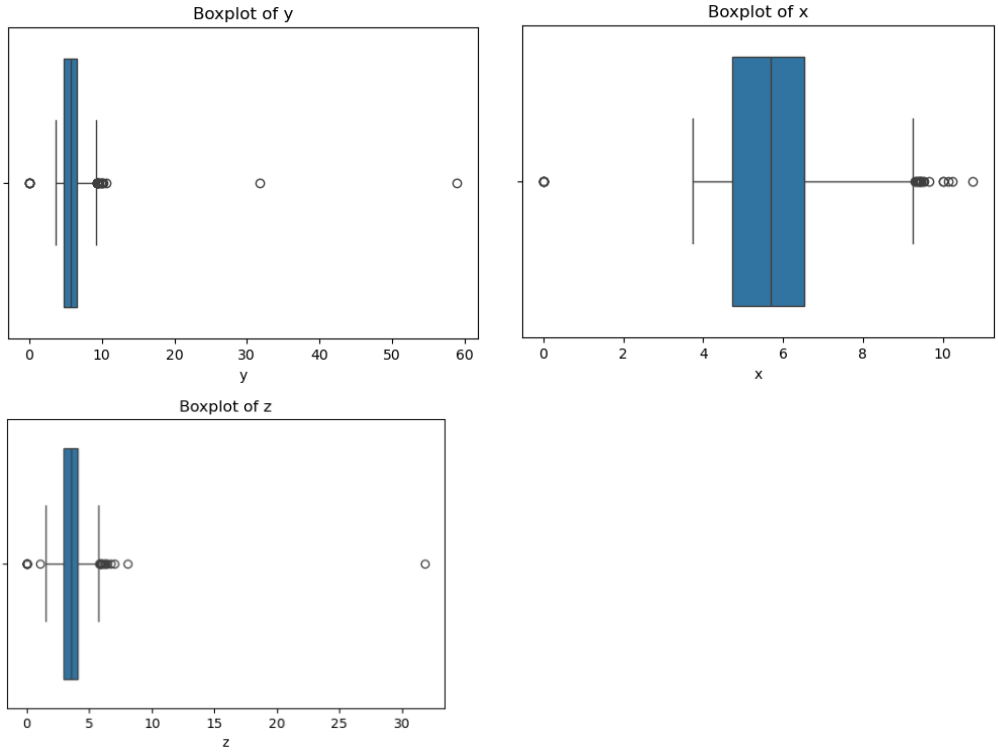
Viy Moodley (300565283)

Part 1: Performance Metrics in Regression

Preprocessing Before Regression:

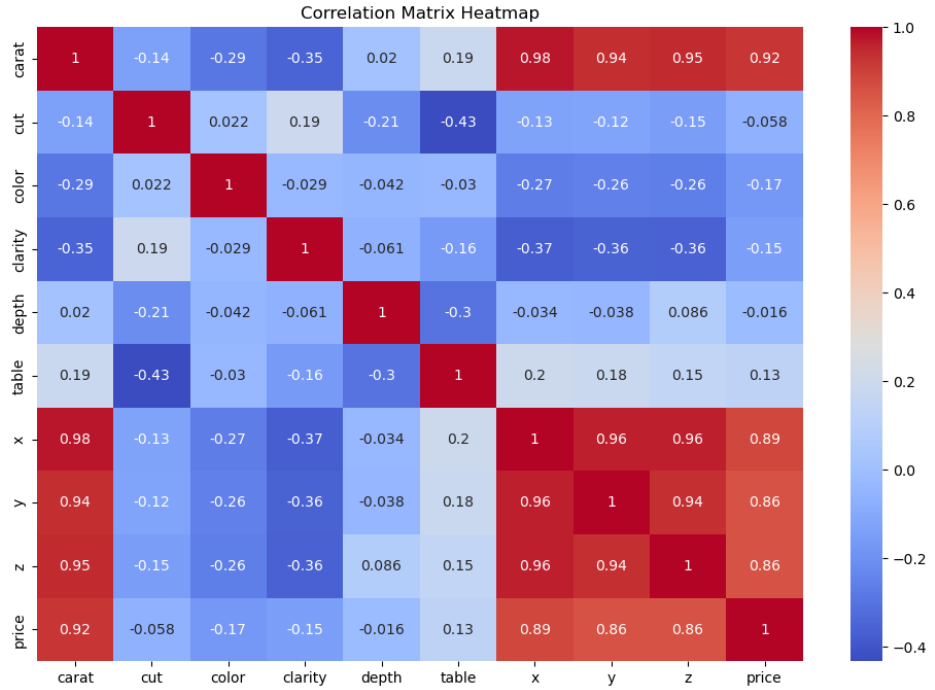
Based on the exploratory data analysis (EDA), the following preprocessing steps were applied:

- **Dropped data:** The column ‘Unnamed: 0’ was dropped from the dataset, given it appeared to be a secondary index column. There were no null values, so no imputation was needed.
- **Train split:** The data was split into a 70:30 train test split and EDA was then performed on the training data to prevent bias and/or leakage
- **Handling Outliers:** Outliers were identified in all the numerical variables. More specifically "x", "y", and "z", which represent the physical dimensions of diamonds. These columns had values of zero, which are physically impossible for a diamond. Additionally, extreme outliers in "carat", "depth", and "table" were addressed, as they could distort the model's predictions. Outliers were removed by using the 1st and 99th percentiles for these variables to minimise their influence.



- **Standardisation:** Variables such as "carat", "depth", "table", "x", "y", and "z" had different ranges, so they were standardised using StandardScaler to ensure that all numeric features were on the same scale. This step was critical for models that are

sensitive to the magnitude of the features, such as K-Nearest Neighbors (KNN) and Support Vector Regression (SVR).



- **Multicollinearity:** The dimensions "x", "y", and "z" showed a high correlation with each other and with "carat", which represents the overall size of a diamond. While multicollinearity can reduce the statistical power of regression models, all these variables were retained because they are likely to capture different aspects of a diamond's size and shape, which are crucial in predicting price

- **Handling Categorical Variables:** Interestingly, all the categorical diamond features "*cut*", "*color*", and "*clarity*" were ordinal, meaning they had a ranking that reflects their quality. These were mapped to numerical values using a ranking system (e.g., "Fair" = 1 to "Ideal" = 5). This encoding helped the regression models to incorporate the hierarchy of these categories into the predictions. I opted out of using an encoder because I wanted to have more control over the hierarchy of the categorical variables.

For more details, see my Jupyter Notebook.

- Compare the performance of different regression algorithms in terms of MSE, RMSE, RSE, and MAE, then analyse and discuss their differences and provide conclusions.

Model	**MSE**	**RMSE**	**RSE**	**MAE**	**Execution Time**
Linear Regression	1182287.64	1087.33	0.09	728.04	0.07
K-Neighbors Regression	382026.63	618.08	0.03	342.34	0.36
Ridge Regression	1182227.68	1087.30	0.09	728.15	0.01

Decision Tree Regression	395294.84	628.72	0.03	319.54	0.20
Random Forest Regression	222476.13	471.67	0.02	241.32	11.38
Gradient Boosting Regression	273555.45	523.03	0.02	296.85	2.81
SGD Regression	1201374.90	1096.07	0.09	723.13	0.07
Support Vector Regression(SVR)	5615857.36	2369.78	0.44	1178.62	99.62
Linear SVR Regression	1996745.48	1413.06	0.16	789.66	0.03
Multi-Layer Perceptron Regression	378721.15	615.40	0.03	365.96	66.85

Comparison of Regression Algorithms:

Simple models like Linear Regression and Ridge Regression assume a straightforward, linear relationship between features like carat, cut, and clarity, and the target variable (price). These models were quick to execute, taking less than 0.1 seconds, but their simplicity limited their ability to capture the more complex relationships within the data. Their MSE values (around 1,182,000) indicated that while they provided a reasonable baseline, they were not effective at capturing non-linear interactions between features, which is crucial for accurate price prediction. Ridge Regression, which includes regularisation to control overfitting, performed slightly better in terms of speed and accuracy, but both models fell short in performance compared to more complex methods.

More advanced models like Decision Tree Regression performed better, with an MSE of 395,294, because they are designed to handle non-linear relationships by splitting the data into subsets based on feature values. However, decision trees are prone to overfitting, which can reduce their ability to generalise well to new data. This model captured some of the complexity in diamond pricing but was outperformed by even more sophisticated models. K-Nearest Neighbors (KNN) also showed strong performance with an MSE of 382,026. KNN, being a non-parametric model, does not assume any specific relationship between the variables, making it adaptable to the structure of the data. Its success suggests that local clusters of similar diamonds in terms of features like carat and clarity were important for price prediction.

The most effective models were ensemble methods, particularly Random Forest Regression and Gradient Boosting Regression. Random Forest had the lowest MSE (222,476) and MAE (241.32), followed closely by Gradient Boosting with an MSE of 273,555. These models work by combining predictions from multiple decision trees, allowing them to capture both linear and non-linear relationships in the data. The strength of these models lies in their ability to reduce overfitting: Random Forest achieves this by averaging the predictions of multiple trees, while Gradient Boosting improves each tree's accuracy by sequentially

correcting the errors of the previous trees. Although these models were slower to execute—Random Forest took over 11 seconds—they were significantly more accurate than the simpler models, making them ideal for handling the complexity of diamond pricing.

On the other end of the spectrum, Support Vector Regression (SVR) and Linear SVR performed poorly, with SVR recording the highest MSE of 5,615,857. These models likely struggled because they are sensitive to high-dimensional, complex data like this. SVR generally requires careful tuning to perform well, and in this case, it likely failed to find an optimal solution due to inadequate hyperparameter tuning or the inherent complexity of the dataset. Similarly, Stochastic Gradient Descent (SGD) Regression, another linear model, produced high error values, confirming that it could not capture the non-linear relationships present in the data.

The models that performed best—Random Forest, Gradient Boosting, and KNN—were able to effectively capture the complexity and non-linear patterns in the data. Diamond pricing depends on the intricate interactions between features like carat size, cut quality, and clarity, and these models excelled at capturing these relationships. Random Forest and Gradient Boosting, as ensemble models, were particularly successful because they aggregate the predictions of multiple decision trees, reducing overfitting and improving prediction accuracy. KNN, which bases its predictions on local data patterns, was also effective, likely benefiting from the distribution of "carat" values, where spikes at popular sizes (such as 1 carat) played a significant role in determining price.

In conclusion, Random Forest Regression emerged as the most effective model for predicting diamond prices, offering the lowest error metrics and the most accurate results. Gradient Boosting Regression was also highly effective, offering a balance between accuracy and computational efficiency. While KNN performed well, its slower execution time makes it less practical for larger datasets. On the other hand, simpler models like Linear Regression, Ridge Regression, and SGD Regression, along with more complex but poorly tuned models like SVR, were not able to capture the complex relationships in the data. Overall, ensemble models like Random Forest and Gradient Boosting are well-suited for datasets where feature interactions are intricate and non-linear, as they deliver robust and accurate predictions.

Part 2: Performance Metrics in Classification

Classifier	Accuracy	Precision	Recall	F1-Score	AUC
KNN	0.83	0.66	0.60	0.63	0.86
Naive Bayes	0.53	0.33	0.94	0.49	0.74
SVM	0.86	0.76	0.58	0.66	0.90
Decision Tree	0.81	0.60	0.62	0.61	0.75
Random Forest	0.85	0.72	0.60	0.65	0.90

AdaBoost	0.86	0.76	0.61	0.68	0.91
Gradient Boosting	0.87	0.79	0.61	0.69	0.92
LDA	0.84	0.72	0.56	0.63	0.89
MLP	0.83	0.65	0.63	0.64	0.89
Logistic Regression	0.85	0.73	0.59	0.65	0.90

Preprocessing Before Classification:

Before applying classification algorithms to the dataset, several preprocessing steps were necessary. First, handling missing values was crucial as columns like workclass, occupation, and native_country had missing data. To address this, a SimpleImputer was used with the most frequent value strategy, ensuring that the dataset remained intact while preventing data loss. Categorical variables, such as workclass, education, and sex, were encoded using one-hot encoding to convert these categories into binary indicator variables, enabling classification models to process them correctly. Furthermore, numeric variables, including age, fnlwgt, education_num, capital_gain, capital_loss, and hours_per_week, were standardised using a standard scaler. This step ensured that variables with different magnitudes were placed on a uniform scale, which is particularly important for distance-based models like K-Nearest Neighbors and models sensitive to feature magnitudes like Support Vector Machines. Additionally, duplicate records found in the dataset were removed to avoid redundant information, and the target variable income was encoded into a binary format using LabelEncoder to facilitate classification tasks where binary outcomes are required.
For more details, see my Jupyter Notebook.

Best Performing Algorithms

According to the metrics, the two best-performing algorithms were:

- **Accuracy**:
  - Gradient Boosting (0.87)
  - AdaBoost (0.86), SVM (0.86)
- **Precision**:
  - Gradient Boosting (0.79)
  - AdaBoost (0.76), SVM (0.76)
- **Recall**:
  - Naive Bayes (0.94)
  - Gradient Boosting (0.61), AdaBoost (0.61)
- **F1-Score**:
  - Gradient Boosting (0.69)
  - AdaBoost (0.68)
- **AUC**:
  - Gradient Boosting (0.92)
  - AdaBoost (0.91)

Are the Best Algorithms the Same Across Metrics?

No, the best algorithms differ across the performance metrics. For example, **Naive Bayes** excelled in recall due to its nature of assuming conditional independence, which made it favour the detection of positive cases. However, it performed poorly in precision and accuracy due to its simplistic assumptions that don't align well with complex data like this. On the other hand, **Gradient Boosting** consistently ranked highly across accuracy, precision, and AUC, indicating it's the most balanced performer in terms of overall model performance. **AdaBoost** also performed similarly, though slightly lower in recall compared to Naive Bayes.

- The results varied across models, with Gradient Boosting emerging as the best performer with an accuracy of 0.87, precision of 0.79, recall of 0.61, F1-score of 0.69, and AUC of 0.92. AdaBoost and SVM also performed well, achieving accuracies of 0.86, with AdaBoost demonstrating strong performance in terms of recall (0.61) and AUC (0.91), and SVM excelling in precision (0.76). Naive Bayes showed the highest recall (0.94) but struggled with precision and accuracy due to its simplistic assumptions about feature independence.
- When comparing the algorithms across performance metrics, it was evident that the best-performing algorithms varied depending on the metric in focus. For instance, Naive Bayes excelled in recall due to its probabilistic approach that tends to favour positive classifications. However, its accuracy and precision were lower, reflecting its limitations in handling complex data distributions. In contrast, Gradient Boosting and AdaBoost consistently ranked among the top in accuracy, precision, and AUC, demonstrating a balanced performance across different evaluation metrics. This consistency highlights the strength of ensemble learning methods in iteratively improving prediction accuracy. Although Naive Bayes was optimal for recall, Gradient Boosting and AdaBoost were the most balanced classifiers overall, making them more suitable for datasets where precision and overall model performance are critical.

Conclusion

- **Gradient Boosting** and **AdaBoost** emerged as the most balanced classifiers across various metrics, offering high performance across accuracy, precision, recall, F1-score, and AUC. However, they were slightly outperformed by Naive Bayes in terms of recall, which could be beneficial in highly imbalanced datasets. The differences arise due to how these algorithms optimise decision boundaries, with ensemble methods like Gradient Boosting and AdaBoost using iterative learning to refine predictions, while Naive Bayes relies on probabilistic assumptions.