Part A

The classification goal of our models was to train the models to classify cars by price given all other features about the car, customer, and date of purchase. With this goal in mind, we started preprocessing by dropping columns in the dataset that would not be relevant when predicting the price paid for a vehicle, so, we dropped 'Car_id', 'Dealer_No ', 'Year_x', 'Customer Name', 'Phone', 'Dealer_Name'. We also dropped the ‘Make’ column because this is a repeat column of the ‘Company’ column and we do not need both. From joining two datasets in a year, we had two columns named ‘Year_x’ and ‘Year_y’ which we retitled appropriately. We then explored the characteristics of our data using histograms and a heatmap to gauge the characteristics of the data and the relevance of columns to one another. We noticed from the heat map that, in relation to price, all other numerical columns seem reasonably correlated so we decided not to drop any columns based on the heatmap. From the histogram, we noticed that the distribution of data varied. We decided before we applied any processing techniques like oversampling or dropping outliers in order to achieve more uniform distribution, we would first finish implementing the models and if the models were struggling with classification, we would come back and apply more preprocessing. We ended up with very good results from our models so we decided processing of the data for the purpose of creating more uniform distribution was unnecessary.

The models we are using require categorical data to be encoded so we used one-hot encoding and label encoding to address this. One hot encoding implies there is no hierarchical relationship between categories so we applied this encoding technique to columns without any hierarchical relationship. Label encoding assumes there is a hierarchical relationship between the categories so we applied this encoding to columns with hierarchy, like vehicle size, for example. For the numerical data, we used a min-max scaler to scale all numerical data to the same scale so that values of extremely large magnitude did not have a disproportionately large impact on the models.

We did not have to worry about other data quality issues like null values because we handled this sort of processing early on in the project.

Part B3

The three models we implemented are the decision tree classifier, the random forest classifier, and the gradient boosting classifier. Screenshots detailing the precision, recall, f1, accuracy, and runtime measures from the three models are below:

Figure 1: Screenshot of the decision tree performance report.

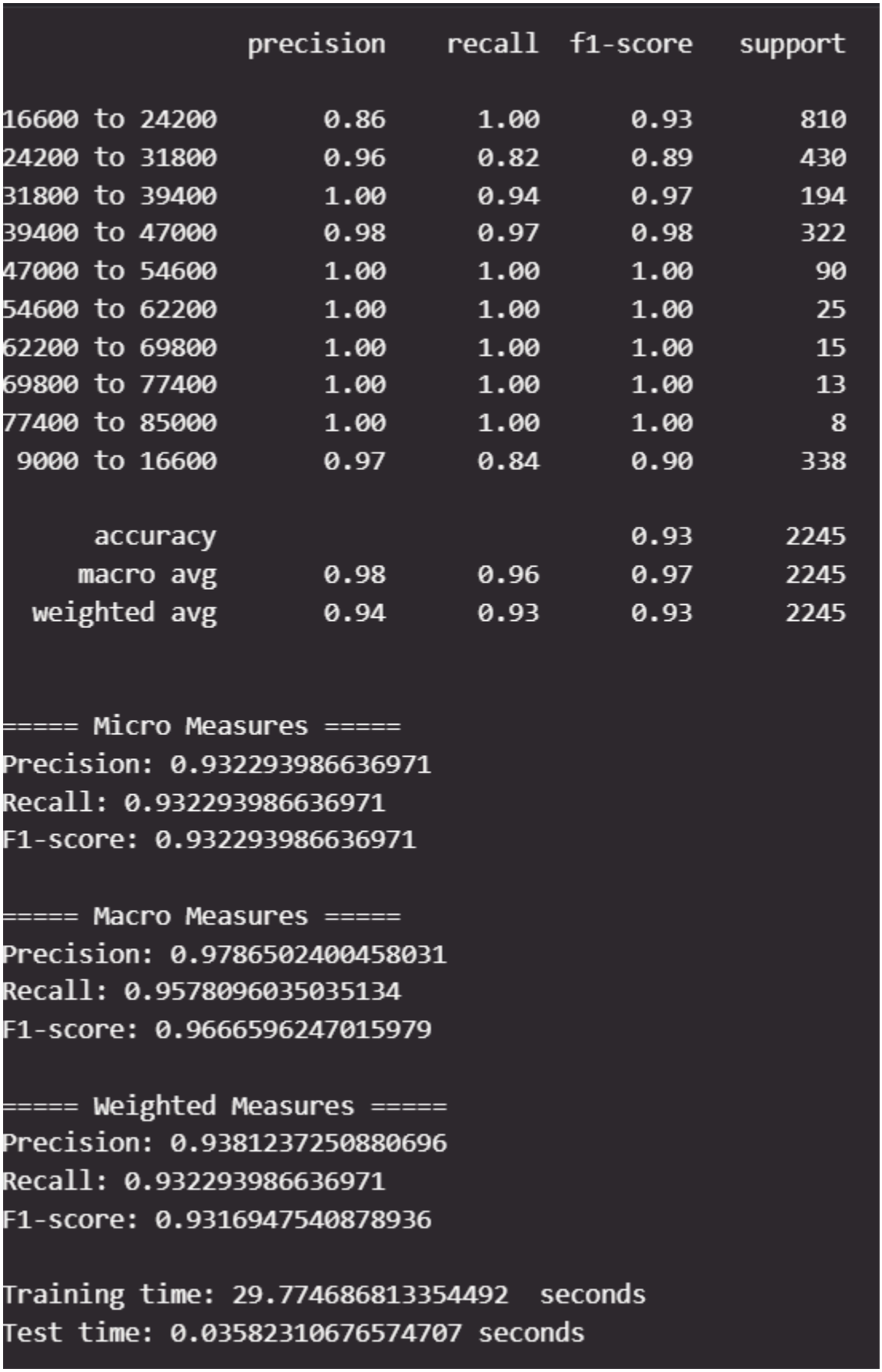
	precision	recall	f1-score	support
16600 to 24200	0.98	1.00	0.99	810
24200 to 31800	0.99	0.97	0.98	430
31800 to 39400	1.00	0.98	0.99	194
39400 to 47000	1.00	1.00	1.00	322
47000 to 54600	0.99	1.00	0.99	90
54600 to 62200	1.00	1.00	1.00	25
62200 to 69800	1.00	1.00	1.00	15
69800 to 77400	1.00	1.00	1.00	13
77400 to 85000	1.00	1.00	1.00	8
9000 to 16600	1.00	0.99	0.99	338
accuracy			0.99	2245
macro avg	1.00	0.99	0.99	2245
weighted avg	0.99	0.99	0.99	2245

```
===== Micro Measures =====
Precision: 0.9906458797327394
Recall: 0.9906458797327394
F1-score: 0.9906458797327394

===== Macro Measures =====
Precision: 0.9958334420554014
Recall: 0.9940601638420521
F1-score: 0.9949183094513137

===== Weighted Measures =====
Precision: 0.9907251839412609
Recall: 0.9906458797327394
F1-score: 0.9906310214063048

Training time: 0.16675591468811035  seconds
Test time: 0.003578662872314453 seconds
```

Figure 2: Screenshot of the random forest performance report.

```
                precision    recall  f1-score   support

16600 to 24200       1.00      1.00      1.00       810
24200 to 31800       1.00      1.00      1.00       430
31800 to 39400       1.00      0.99      1.00       194
39400 to 47000       1.00      1.00      1.00       322
47000 to 54600       1.00      1.00      1.00        90
54600 to 62200       1.00      1.00      1.00        25
62200 to 69800       1.00      1.00      1.00        15
69800 to 77400       1.00      0.85      0.92        13
77400 to 85000       1.00      1.00      1.00         8
 9000 to 16600       1.00      1.00      1.00       338

      accuracy                           1.00      2245
     macro avg       1.00      0.98      0.99      2245
  weighted avg       1.00      1.00      1.00      2245


===== Micro Measures =====
Precision: 0.9986636971046771
Recall: 0.9986636971046771
F1-score: 0.9986636971046771

===== Macro Measures =====
Precision: 0.999630996309963
Recall: 0.9840999206978589
F1-score: 0.9912234258503011

===== Weighted Measures =====
Precision: 0.9986686281116708
Recall: 0.9986636971046771
F1-score: 0.9986272371136998

Training time: 1.0616328716278076  seconds
Test time: 0.03758525848388672 seconds
```

Figure 3: Screenshot of the gradient boosting performance report.



	precision	recall	f1-score	support
16600 to 24200	0.86	1.00	0.93	810
24200 to 31800	0.96	0.82	0.89	430
31800 to 39400	1.00	0.94	0.97	194
39400 to 47000	0.98	0.97	0.98	322
47000 to 54600	1.00	1.00	1.00	90
54600 to 62200	1.00	1.00	1.00	25
62200 to 69800	1.00	1.00	1.00	15
69800 to 77400	1.00	1.00	1.00	13
77400 to 85000	1.00	1.00	1.00	8
9000 to 16600	0.97	0.84	0.90	338
accuracy			0.93	2245
macro avg	0.98	0.96	0.97	2245
weighted avg	0.94	0.93	0.93	2245

```
===== Micro Measures =====
Precision: 0.932293986636971
Recall: 0.932293986636971
F1-score: 0.932293986636971

===== Macro Measures =====
Precision: 0.9786502400458031
Recall: 0.9578096035035134
F1-score: 0.9666596247015979

===== Weighted Measures =====
Precision: 0.9381237250880696
Recall: 0.932293986636971
F1-score: 0.9316947540878936

Training time: 29.774686813354492  seconds
Test time: 0.03582310676574707 seconds
```

From the results above, we observe that all three models had good performance for precision, recall, f1-score, and accuracy. Random forest produced the best results overall except for in macro precision and recall where the decision tree performed marginally better. In runtime, the decision tree was the fastest during training and testing. Random forest was still very fast during both testing and training while gradient boosting was fast during testing but very slow during training, taking almost 30 seconds longer than the other two models. From this we conclude that the gradient boosting was the least suitable model for this particular classification application while both decision tree or random forest were better options. Based on the good performance of all three models, we conclude that the car features, customer details, CPI, and date of purchase information were very good indicators for the selling price of a car. This information would be useful for potential buyers who could use this data to estimate the price of the car they would pay while car dealerships could use this information to estimate their profit over a certain period of time by looking at the selling price of cars similar to the cars they sell during the time period in question.

Based on the performance of the models, the models are also excellent candidates for helping fill null values in preprocessing in an appropriate scenario rather than filling in null values with other traditional preprocessing techniques to clean data such as dropping null rows.

Part C2

To find outliers in our dataset, we implemented the one-class SVM model. So that it could identify outliers in all features of our dataset, we one-hot encoded and label encoded categorical data so that the whole dataset appropriately used numerical data. We also used a min-max scaler on numerical data to prevent different scales from affecting the performance of the model. To identify specific outliers, we had the rows identified as outliers added to a pandas dataframe which we printed to tables. We also provided counts of the number of outliers, and calculated the percentage of the data that was determined to be outliers. We found that there was a low number of outliers, with approximately only 1-2% of the data being outliers. We implemented the SVM model after having already implemented our decision tree, random forest, and gradient boosting classifiers, which produced great results in measures of precision, recall, f1 scores, and accuracy. Given the good performance of our classification models, and the low number of outliers, we decided it was best we leave the outliers untouched in the dataset. Since our models already produced extremely accurate classification, we were concerned that removing the outliers may result in the models overfitting our data, and losing general applicability. From the SVM we learned that a relatively low amount of our data contained outliers and based on the good performance of our classifiers, we can assume these outliers are not especially extreme as they did not prevent the classifiers from learning the dataset well.