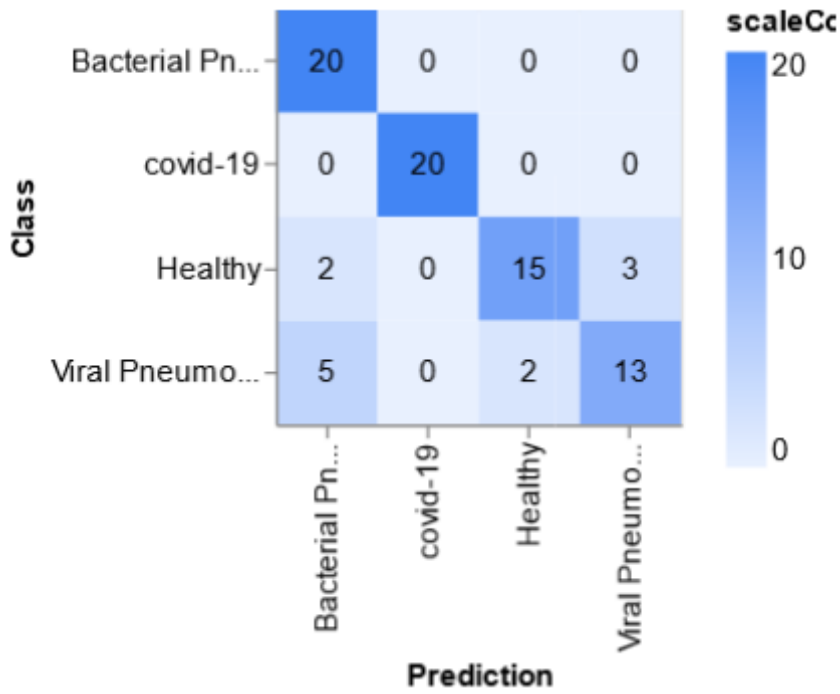
Problem 1

Model Performance Evaluation

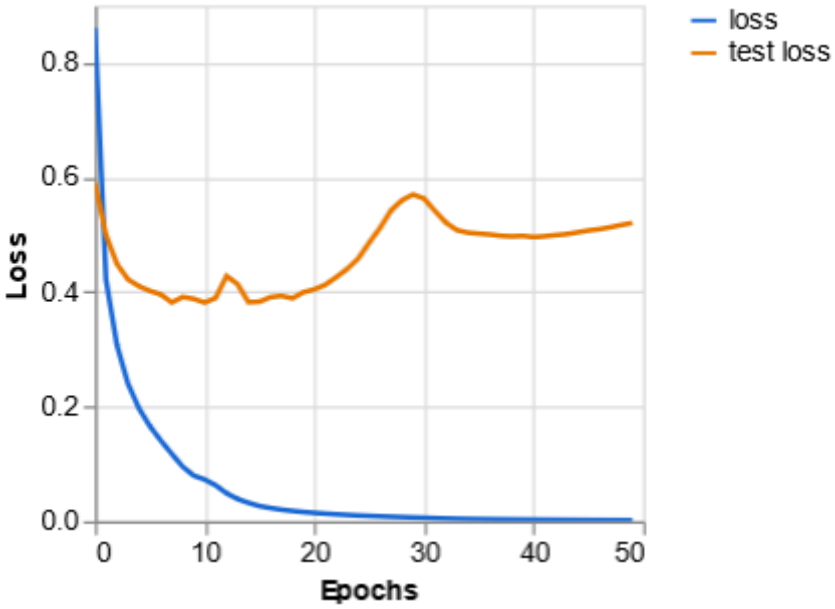
The confusion matrix reveals the model's performance across different classes:

- Bacterial Pneumonia and COVID-19: The model achieved perfect classification for both classes, correctly identifying all 20 examples in each.
- Healthy: Five out of 20 healthy cases were misclassified, with two predicted as Bacterial Pneumonia and three as Viral Pneumonia.
- Viral Pneumonia: This class posed the most challenge for the model, with seven out of 20 examples misclassified. Five were incorrectly identified as Bacterial Pneumonia, and two as Healthy.



Overall, the model accurately detects Bacterial Pneumonia and COVID-19 but shows less precision with Viral Pneumonia, frequently mistaking it for Bacterial Pneumonia. While it performs reasonably well in identifying healthy cases, there are occasional confusions with pneumonia classes.

Loss Curve Analysis



The loss curves show that the training loss consistently decreases, indicating the model's ability to learn. The test loss initially decreases rapidly, signifying effective learning. However, after 20 epochs, the validation loss begins to increase while the training loss continues to decrease, indicating overfitting.

Convergence, where the training loss stabilizes around a minimum value, occurs around epoch 30. This indicates that the model has learned the optimal parameters for the given data and architecture. Furthermore, the relatively stable loss suggests the learning rate is appropriate. Based on the loss curves, an optimal checkpoint for the model could be selected between epochs 14 and 20, balancing high test accuracy with low test loss.

Hyperparameter Tuning

Three hyperparameters were tuned to optimize the model's performance: number of epochs, batch size, and learning rate.

Epochs: reducing the number of epochs from 50 to 30 did not significantly impact performance and reduced training time. This supports the idea of using early stopping to prevent overfitting and save computational resources. Interestingly, in one instance, reducing epochs to 10 significantly improved the model (substantially higher test accuracy and substantially lower test loss), potentially due to beneficial random weight initialization. This highlights the impact of randomness in training. However, this improvement was not consistently reproducible.

Batch size: A batch size of 512 resulted in nearly identical training and validation loss curves because it approached the total dataset size of 532. This is because the model was effectively updating its weights based on the entire dataset in each iteration, minimizing the difference between training and validation performance. Smaller batch sizes, while reducing training time, can introduce more variability in gradient estimates, potentially leading to less stable training. Both test accuracy and test loss improve when using mini-batch gradient descent (batch size > 1) compared to stochastic gradient descent (batch size = 1). However, excessively large batch sizes may also hinder performance.

Learning rate: a learning rate of 0.01 caused fluctuations, likely due to overshooting optimal solutions. At 0.1, the model failed to learn, resulting in an accuracy of 0.25, similar to random guessing. Conversely, a very small learning rate of 0.00001 hindered generalization, as the model struggled to escape local minima. Generally, lower learning rates require longer training times but can lead to improved test accuracy and lower test loss. As observed in our experiments, learning rates that are an order of magnitude too high or too lower result in suboptimal model performance.

Problem 2

For this problem that was realized with the help of a Jupyter Notebook, we have included [x] to reference the cell (labeled [x]) where the code supporting this statement is executed.

Handling Missing Values

We begin by addressing missing values in the dataset. Any rows containing `NaN`, `None`, or null values are removed. [1]

Encoding Categorical Variables

Next, we identify the categorical features: `Make`, `Model`, `Type`, `Origin`, and `DriveTrain`. While features like `Make` and `Model` are nominal (with no inherent order), other variables such as `DriveTrain`, `Type`, and `Origin` may exhibit an ordinal relationship. [2]

For example, in the case of `DriveTrain` (values: 'Rear', 'All', 'Front'), we hypothesize a potential ordinal relationship based on vehicle pricing.

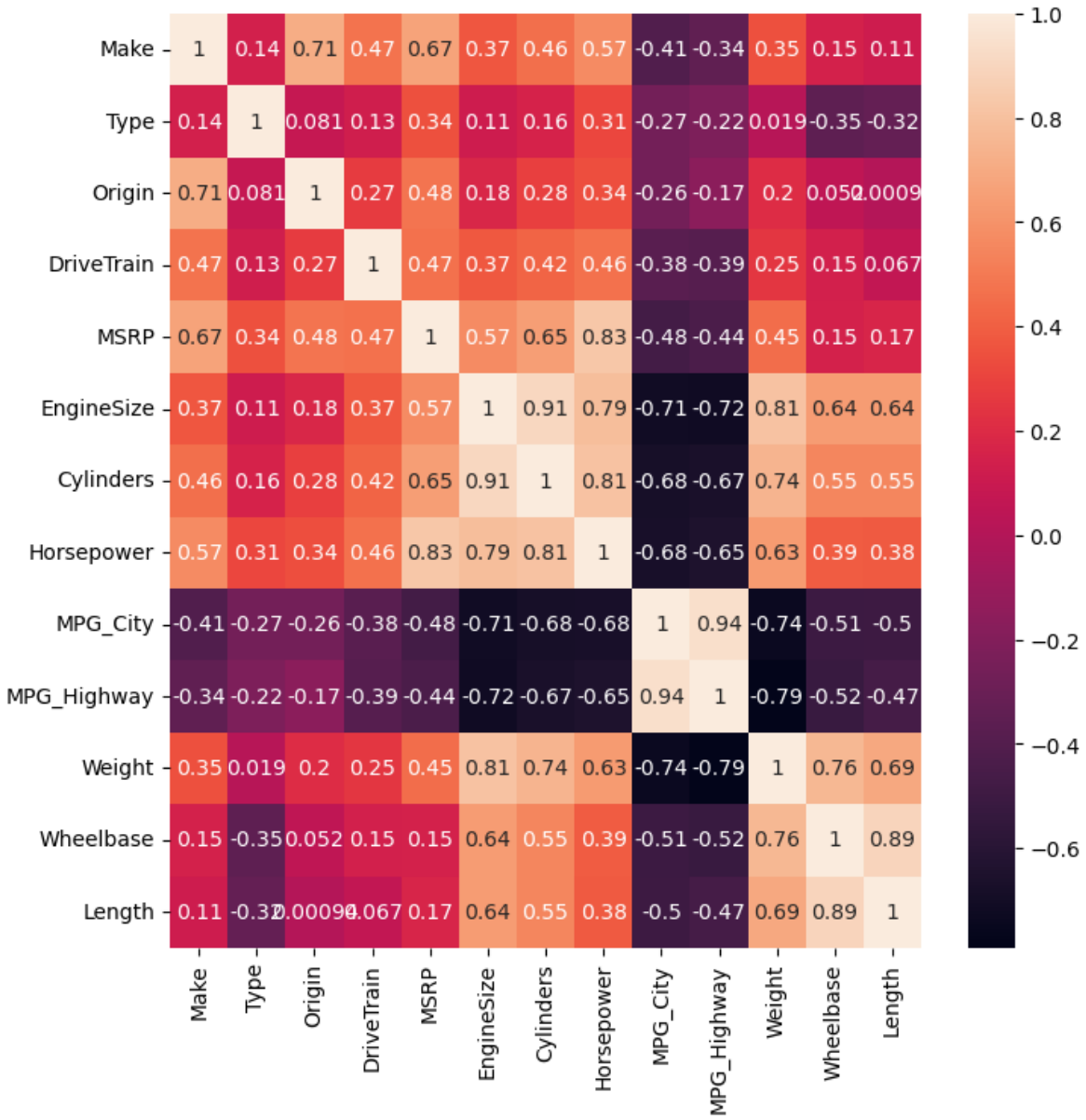
By displaying the average MSRP for each DriveTrain category ('All', 'Rear', 'Front') in descending order [3], we can observe that vehicles with 'Rear' drive are, on average, more expensive than those with 'All' or 'Front' drive, indicating a clear ordering. Based on this, we assign ordinal ranks to the `DriveTrain` variable: 'Rear' = 3, 'All' = 2, and 'Front' = 1. This ordinal encoding allows the model to better capture the relationship between `DriveTrain` and price, as opposed to one-hot encoding, which would create more features without capturing this inherent order.

We applied a similar approach to the `Type` [4] and `Origin` [5] variables, where a natural ranking can help the model generalize better.

Feature Exploration

By exploring the frequency of values across categorical features, we made some key observations:

- The `Model` feature is not useful, as most car models appear only once, offering little value to the model. [6]
- The Type category has a 'Hybrid' class with only three samples, suggesting it may not contribute much to the model's learning. [7]
- Some outliers can be directly identified within the Make and Type features. Some Makes (5 out of 38) appeared only a few times, and the "Hybrid" category within Type contained only three samples. Removing these outliers could potentially improve future model performance. [6] [7]



By visualizing the correlation matrix above [8], we observed:

- `Wheelbase` and `Length` are highly correlated with each other but have low correlation with `MSRP` (0.15 and 0.17, respectively).
- `Make`, `Type`, `Origin` and `DriveTrain` have some sort of correlation with `MSRP` (0.67, 0.34, 0.48 and 0.47, respectively).
- `MPG_Highway` and `MPG_City` are highly correlated (0.94).
- `Cylinders` is highly correlated with `EngineSize` (0.91).

Based on these correlations, we decided to remove [10]:

- `Length` and `Wheelbase` due to their low correlation with `MSRP`.
- `MPG_Highway` and `EngineSize` because they are highly correlated with `MPG_City` and `Cylinders`, respectively, and the latter features are more predictive of `MSRP`.

In addition, we conducted a feature importance analysis using a RandomForestRegressor, which confirmed that certain features like `Type`, `Origin`, `DriveTrain`, `EngineSize`, `MPG_City`, and `Wheelbase` are not heavily relied upon for predicting `MSRP`. [10] [11] [12]

Make	Type	Origin	DriveTrain	EngineSize	Cylinders	Horsepower	MPG_City	MPG_Highway	Weight	Wheelbase	Length
0.15	0.01	0.02	0.0	0.01	0.01	0.69	0.01	0.01	0.04	0.03	0.01

- A RandomForestRegressor model using all features achieved an average R^2 score of 0.91.
- When only `Make` and `Horsepower` were kept, the model's performance dropped slightly to 0.85.
- If we retain only the most important features (`Make`, `Origin`, `Cylinders`, `Horsepower`, `MPG_City`, and `Weight`), the model still achieved an R^2 of 0.91, removing nearly half of the original features without performance loss.

This analysis allowed us to confidently reduce the feature set.

After these experiments, we decided to keep the following six features: `Make`, `Origin`, `Cylinders`, `Horsepower`, `MPG_City`, and `Weight`. Removing `DriveTrain` had no significant impact on performance, and dropping any of `Cylinders`, `MPG_City`, or `Origin` reduced the model's performance by only 0.01 on average.

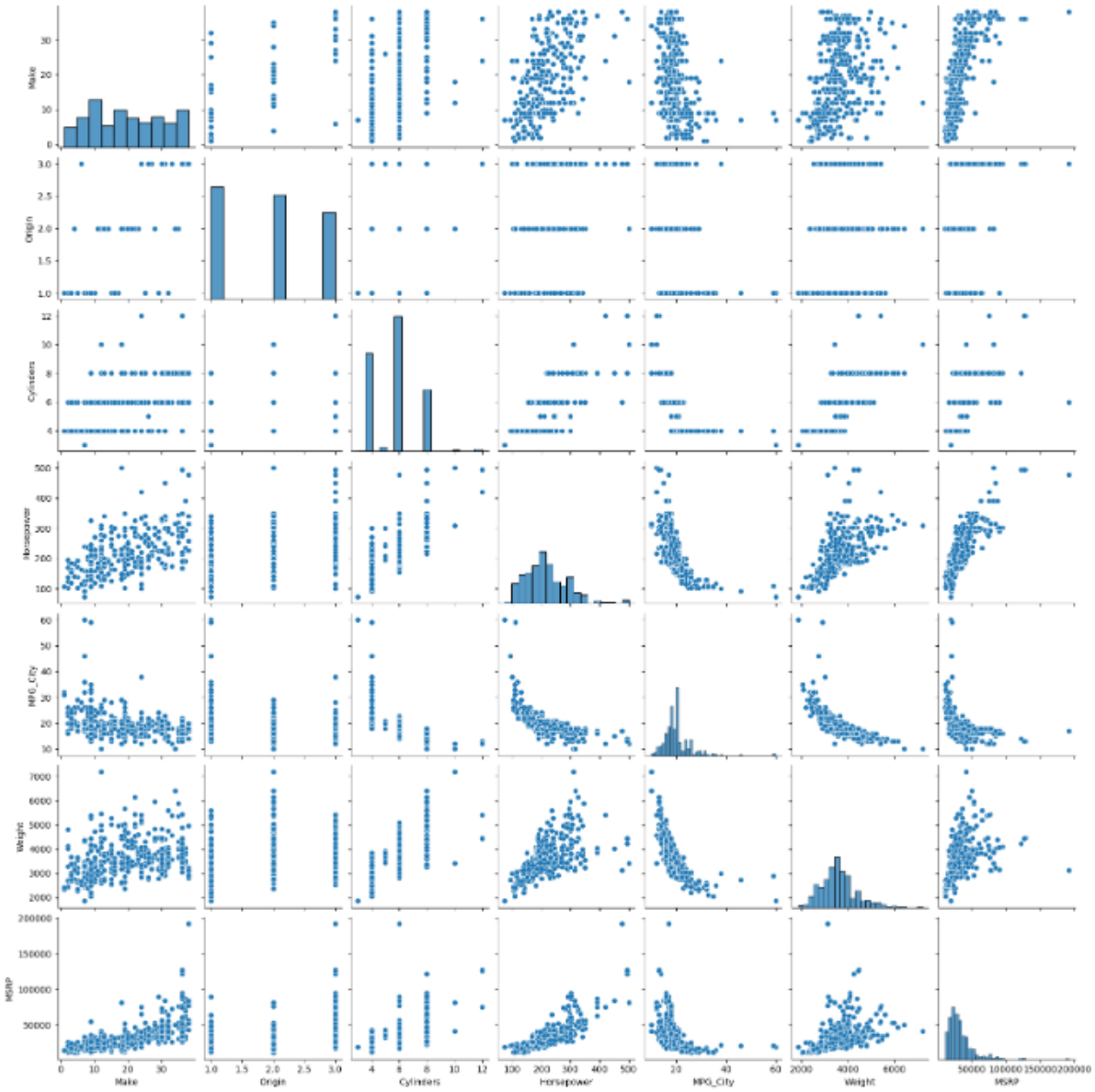
These results were averaged over 100 different random states, with the data shuffled each time for validation. [10]

Dataset Statistics

Before scaling, we inspected the summary statistics of the selected features (after encoding ordinal variables). Scaling makes sense only after this step, as otherwise, the statistics would simply show means of 0 and standard deviations of 1 (standardization). [13]

	Make	Type	Origin	DriveTrain	MSRP	EngineSize	Cylinders	Horsepower	MPG_City	MPG_Highway	Weight	Wheelbase	Length
count	428.000	428.000	428.000	428.000	428.000	428.000	428.000	428.000	428.000	428.000	428.000	428.000	428.000
mean	19.708	4.166	1.918	1.729	32774.855	3.197	5.799	215.886	20.061	26.843	3577.953	108.154	186.362
std	10.510	0.965	0.807	0.845	19431.717	1.109	1.560	71.836	5.238	5.741	758.983	8.312	14.358
min	1.000	1.000	1.000	1.000	10280.000	1.300	3.000	73.000	10.000	12.000	1850.000	89.000	143.000
25%	10.000	4.000	1.000	1.000	20334.250	2.375	4.000	165.000	17.000	24.000	3104.000	103.000	178.000
50%	19.000	4.000	2.000	1.000	27635.000	3.000	6.000	210.000	19.000	26.000	3474.500	107.000	187.000
75%	29.250	5.000	3.000	3.000	39205.000	3.900	6.000	255.000	21.250	29.000	3977.750	112.000	194.000
max	38.000	6.000	3.000	3.000	192465.000	8.300	12.000	500.000	60.000	66.000	7190.000	144.000	238.000

Data Scaling



We observed the following patterns in the numeric features before scaling [14]:

- Features such as `Cylinders`, `Horsepower`, `MPG_City`, and `Weight` follow distributions that are close to Gaussian, suggesting that standardization (scaling to zero mean and unit variance) could be appropriate for these variables.
- The target variable MSRP, however, is left-skewed, implying that MinMax normalization (scaling to the 0-1 range) might be more effective for this variable. While standardization or normalization may yield similar results, normalization could slightly improve the model's performance for highly skewed data.

Outliers and Patterns

Using the pair plots from the previous section, we identified some outliers in the numeric data. For instance, extreme values can be observed at the upper end of Cylinders, Horsepower, and Weight. These outliers could potentially affect model performance, and handling them (e.g., by removing or capping) might be a future improvement step.

Furthermore, these pair plots revealed some relationships between features and `MSRP` appear nonlinear (e.g., `Horsepower` vs. `MSRP`), indicating that models like Decision Trees or Random Forests, which can capture nonlinearity, may perform better than linear models.

Selection of the regression model

I chose to train a random forest regression model because it can capture complex, non-linear relationships between features more effectively than a linear regression model. It is also more robust against overfitting compared to a decision tree model. While random forests generally perform better than polynomial regression, one downside is their lack of interpretability compared to linear regression. Additionally, random forests tend to perform poorly when predicting values outside the range of the training data due to the way decision trees are constructed. Despite these considerations, random forests offer a robust and stable approach compared to a single decision tree.

Model Performance

Even with this initial intuition, I compared the performance of the three models (linear regression, decision tree, and random forest) to ensure the best model choice. Splitting the dataset into 80% for training and 20% for testing, we obtained the following results for an average R^2 score across multiple random states with shuffled datasets [10]:

- **Random Forest:** 0.907
- **Linear Regression:** 0.842
- **Decision Tree:** 0.872

As expected, the random forest model outperformed the other models, demonstrating its superior ability to capture complex patterns. However, both linear regression and decision tree showed decent predictive power, indicating the presence of strong relationships in the data.

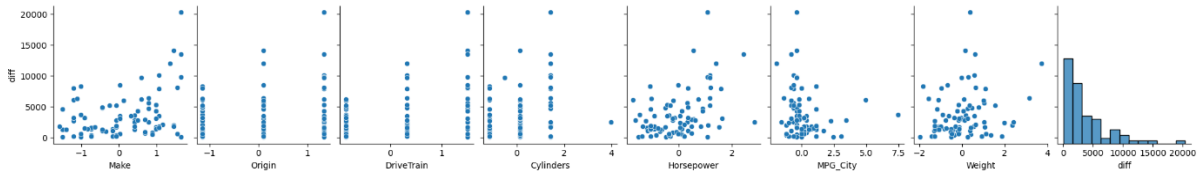
Predicted vs. Actual Prices

Since we initially normalized the target output, we used the `inverse_transform` method of `MinMaxScaler` to revert this normalization for interpretability. Here is the table of predicted vs. actual car prices on the 15 first samples of the test dataset (to reduce the size to the image on the page) [15]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Predictions	14102.36	15597.18	25143.92	50406.55	27237.82	43355.6	41310.27	69881.9	20950.11	15088.74	30080.46	15059.29	37073.2	41968.47	14458.59
Actual Price	15460.00	14300.00	33260.00	47955.00	24130.00	43755.0	44240.00	69995.0	21410.00	13839.00	32235.00	17750.00	37630.0	39640.00	14610.00

Model Prediction Analysis

The model's performance is overall satisfactory but shows some variability. While some predictions are accurate (e.g., around $1,000 off the actual value, such as prediction n°14), others are significantly off (e.g., $8,000 for prediction n°2).



The pairplots [17] reveal that no single feature consistently leads to high errors. The features in the plots are standardized, so points further from zero represent outliers. For ordinal features, the points form lines as many data points share the same category.

The pairplot also suggests that the Random Forest model is not entirely robust to outliers. For features like Make and Horsepower, outliers tend to have higher prediction errors.

The analysis shows that the Random Forest model achieves an error of less than $2500 for a significant portion of the data. The following table summarizes the error distribution [16]:

Error Intervals	Percentage	Cumulative Sum Percentage
(28.489, 1610.418]	36.05	36.05
(1610.418, 3172.045]	25.58	61.63
(3172.045, 4733.673]	11.63	73.26
(4733.673, 6295.301]	10.47	83.73
(6295.301, 7856.928]	2.33	86.06
(7856.928, 9418.556]	5.81	91.87
(9418.556, 10980.184]	3.49	95.36
(10980.184, 12541.812]	1.16	96.52
(12541.812, 14103.439]	1.16	97.68
(14103.439, 15665.067]	1.16	98.84
(15665.067, 17226.695]	0.00	98.84
(17226.695, 18788.322]	0.00	98.84
(18788.322, 20349.95]	1.16	100.00

The cumulative sum in the table indicates that 75-80% of the test predictions are within $5000 of the actual value. This is a reasonable performance considering the average car price is around $32,000, according to the statistical summary. While good, this level of accuracy might not be sufficient for certain applications requiring higher precision.

Potential Improvements

To enhance the model, I recommend the following 3 axes:

1. **Increasing Data Volume**: More data can help stabilize the model's R^2 score, which fluctuated by a substantial amount (±0.1), indicating variability in generalization.
2. **Hyperparameter Tuning:** Using techniques like grid search can be employed to find optimal hyperparameters for the Random Forest model, potentially enhancing its performance.
3. **Outlier Removal**: As previously discussed, we noticed some outliers, in Make or Type for example. We could either perform an analysis on the dataset similar to what we have done or use techniques such as IQR or Z-score to identify and remove outliers, improving the model's generalization. For example, rare makes like Hummer may confuse the model due to their limited representation in the dataset.