Faculty of Engineering & Technology

Electrical & Computer Engineering Department

MACHINE LEARNING AND DATA SCIENCE

ENCS5341

Assignment #2 Report

Prepared by:

Student Name: Razi Atyani **Student ID:** 1200028

Student Name: Abdalkarim Eiss **Student ID:** 1200015

Instructor: Dr. Yazan Abu Farha && Dr. Ismail Khater

Section: 1 & 2

Date: 11/26/2024

Abstract

This report focuses on the development and evaluation of regression models to predict car prices using a dataset from Yalla Motors. The project emphasizes the application of both linear and nonlinear regression techniques, incorporating data preprocessing, feature selection, and regularization methods to enhance model accuracy and generalization. Key steps include data cleaning, exploratory data analysis, implementation of linear models (e.g., Linear Regression, LASSO, Ridge), and nonlinear approaches like Polynomial Regression and Radial Basis Function (RBF). Models are evaluated using performance metrics such as Mean Squared Error (MSE) and R-squared, followed by hyperparameter tuning with grid search to identify optimal configurations. Finally, the best-performing model is validated on unseen test data. The report details the methodology, results, and insights gained, with visualizations supporting the analysis.

Table of Contents

List of Figures

1 About Dataset

The dataset consists of 6,308 entries and contains information about cars. It includes the following 9 columns:

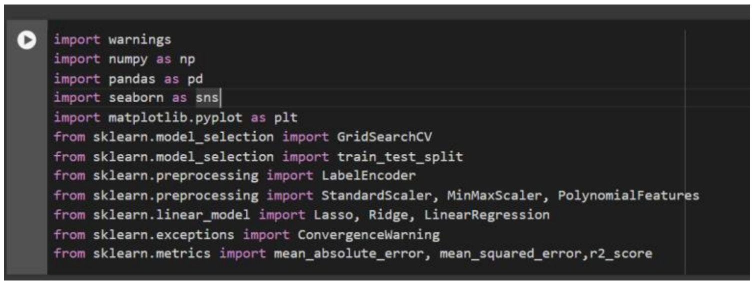
- **Car Name:** The car’s name.
- **Price:** The price of the car, expressed as a string with currency or other notations.
- **Engine Capacity:** The engine size, measured in liters (L).
- **Cylinder:** The number of cylinders or type of engine (e.g., electric or N/A for electric cars). This column has missing values.
- **Horse Power:** The power output of the car's engine, measured in horsepower.
- **Top Speed:** The maximum speed the car can achieve.
- **Seats:** The seating capacity of the car.
- **Brand:** The manufacturer or brand of the car.
- **Country:** The origin or market location of the car.

Observations:

- ➢ Some columns, like cylinder, have missing or inconsistent data, such as "N/A" or descriptions mixed with numbers.
- ➢ Several numeric fields, such as price and horse_power, are stored as strings.
- ➢ The dataset spans multiple brands and countries, offering a diverse range of cars, including electric and traditional fuel vehicles.

2 Libraries

Figure 1 shows the libraries that are used in the assignment code.



```python
import warnings
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler, MinMaxScaler, PolynomialFeatures
from sklearn.linear_model import Lasso, Ridge, LinearRegression
from sklearn.exceptions import ConvergenceWarning
from sklearn.metrics import mean_absolute_error, mean_squared_error,r2_score
```

Figure 1: All Libraries

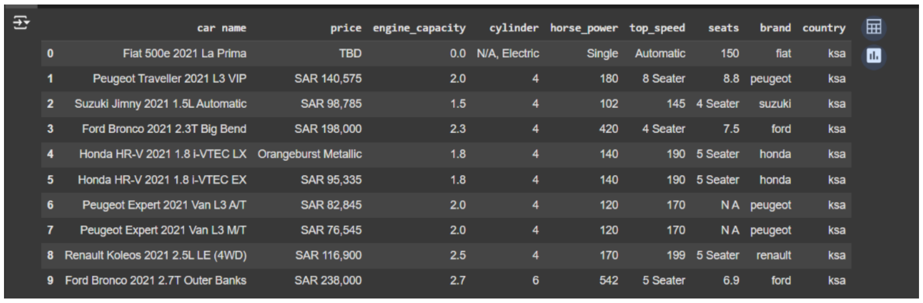
- **warnings:** Provides a way to issue and control warning messages in Python. It is useful for suppressing or handling warnings that occur during execution.
- **numpy (imported as np):** A powerful library for numerical computations in Python. It supports multi-dimensional arrays and provides mathematical functions for operations like linear algebra, statistics, and Fourier transforms.
- **pandas (imported as pd):** A data manipulation and analysis library that provides data structures like DataFrames and Series. It is widely used for cleaning, transforming, and analyzing structured data.
- **seaborn (imported as sns):** A visualization library built on top of Matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.
- **matplotlib.pyplot (imported as plt):** A core plotting library in Python used to create static, animated, and interactive visualizations. It works well for creating charts like line plots, bar plots, and scatter plots.
- **sklearn.model_selection (functions GridSearchCV and train_test_split)**:
    - **GridSearchCV:** Used for hyperparameter tuning of machine learning models.
    - **train_test_split:** Splits data into training and testing subsets for model evaluation.
- **sklearn.preprocessing (functions LabelEncoder, StandardScaler, MinMaxScaler, PolynomialFeatures):**
    - **LabelEncoder:** Converts categorical labels into numerical labels.
    - **StandardScaler:** Scales features by removing the mean and scaling to unit variance.
    - **MinMaxScaler:** Scales features to a specified range (e.g., 0 to 1).
    - **PolynomialFeatures:** Generates polynomial and interaction features for modeling.
- **sklearn.linear_model (classes Lasso, Ridge, LinearRegression):**
    - **Lasso:** Linear regression model using L1 regularization to perform feature selection.
    - **Ridge:** Linear regression model using L2 regularization to avoid overfitting.
    - **LinearRegression:** A basic linear regression model.
- **sklearn.exceptions (class ConvergenceWarning):** Handles warnings related to optimization algorithms that may not converge during model training.
- **sklearn.metrics (functions mean_absolute_error, mean_squared_error, r2_score):**

- **mean_absolute_error:** Measures average magnitude of errors in predictions without considering their direction.
- **mean_squared_error:** Measures average squared difference between actual and predicted values.
- **r2_score:** Indicates the proportion of variance in the dependent variable explained by the model.

3 Data Preprocessing

3.1 Dataset Reading

Figure 2 shows the dataset reading results.



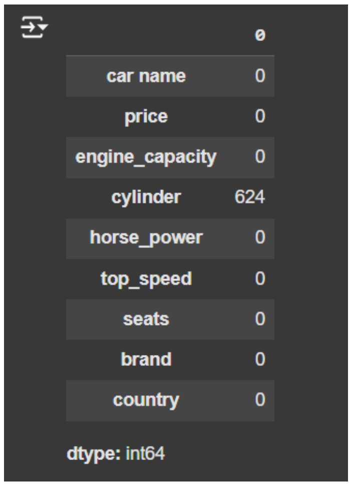
	car name	price	engine_capacity	cylinder	horse_power	top_speed	seats	brand	country
0	Fiat 500e 2021 La Prima	TBD	0.0	N/A, Electric	Single	Automatic	150	fiat	ksa
1	Peugeot Traveller 2021 L3 VIP	SAR 140,575	2.0	4	180	8 Seater	8.8	peugeot	ksa
2	Suzuki Jimny 2021 1.5L Automatic	SAR 98,785	1.5	4	102	145	4 Seater	suzuki	ksa
3	Ford Bronco 2021 2.3T Big Bend	SAR 198,000	2.3	4	420	4 Seater	7.5	ford	ksa
4	Honda HR-V 2021 1.8 i-VTEC LX	Orangeburst Metallic	1.8	4	140	190	5 Seater	honda	ksa
5	Honda HR-V 2021 1.8 i-VTEC EX	SAR 95,335	1.8	4	140	190	5 Seater	honda	ksa
6	Peugeot Expert 2021 Van L3 A/T	SAR 82,845	2.0	4	120	170	N A	peugeot	ksa
7	Peugeot Expert 2021 Van L3 M/T	SAR 76,545	2.0	4	120	170	N A	peugeot	ksa
8	Renault Koleos 2021 2.5L LE (4WD)	SAR 116,900	2.5	4	170	199	5 Seater	renault	ksa
9	Ford Bronco 2021 2.7T Outer Banks	SAR 238,000	2.7	6	542	5 Seater	6.9	ford	ksa

Figure 2: The Dataset Reading Results

As noted, the dataset provided contains several attributes related to various car models, such as price, engine capacity, cylinder count, horsepower, top speed, seats, brand, and country of origin. However, the dataset is not ready for modeling due to inconsistencies and missing values in several columns. For example, the "price" column has a value marked as "TBD" for one car, while the "engine capacity" column contains non-numeric entries like "Electric" and "N/A". Similarly, the "horsepower" column includes an invalid entry of "Single," and the "top speed" column has some "N/A" values. These non-numeric entries and missing data need to be cleaned, either by imputing, removing, or replacing them with appropriate placeholders. Additionally, the categorical columns, such as "brand" and "country," should be standardized to ensure consistency. Preprocessing these columns is essential for preparing the data for successful machine learning modeling.

3.2 Document Missing Values

Figure 3 shows all columns (features) with its number of null values.



```
                 0
      car name   0
         price   0
engine_capacity  0
      cylinder 624
   horse_power   0
     top_speed   0
         seats   0
         brand   0
       country   0

dtype: int64
```

Figure 3: All Columns with Its Number of Null Values

3.3 Missing Values Handling

The cylinder column contained missing values, which were 9.892200380469246% of all data in the column. They were filled with the mode of the column (most frequently occurring value which was 4). Mode imputation is suitable for discrete numerical data. The mode imputation was used to keep the samples and to avoid data dropping.

After the dataset was checked. It didn't have duplicated rows or empty records.

3.4 Defining Upper Limits

To remove unrealistic data points, upper limits for key features were defined based on domain knowledge or internet research:

- **Horsepower:** 1,500.0
- **Top speed:** 530.0 km/h
- **Engine capacity:** 8.4
- **Cylinder count:** 16

This step helps in capping the values and identifying outliers beyond practical ranges.

3.5 Feature Adjustments

The price column (target) was adjusted to extract and standardize currency and numeric values.

- **The Process:**
    - Currency code and price value were extracted using string operations.
    - Prices were converted to dollar equivalents using predefined exchange rates.
    - Invalid Values were converted to -1 and then were removed.
    - Rows with invalid or missing price data were removed.

Additional columns were adjusted to ensure data consistency and integrity:

- **Engine Capacity:**
    - Converted to a numeric value and capped at the defined limit of 8.4.
    - Values exceeding the limit were set to -1.
- **Cylinders, Horsepower, and Top Speed:**
    - Similar numeric conversions and capping were applied to these columns, using respective upper limits.
    - Values exceeding the limit were set to -1.

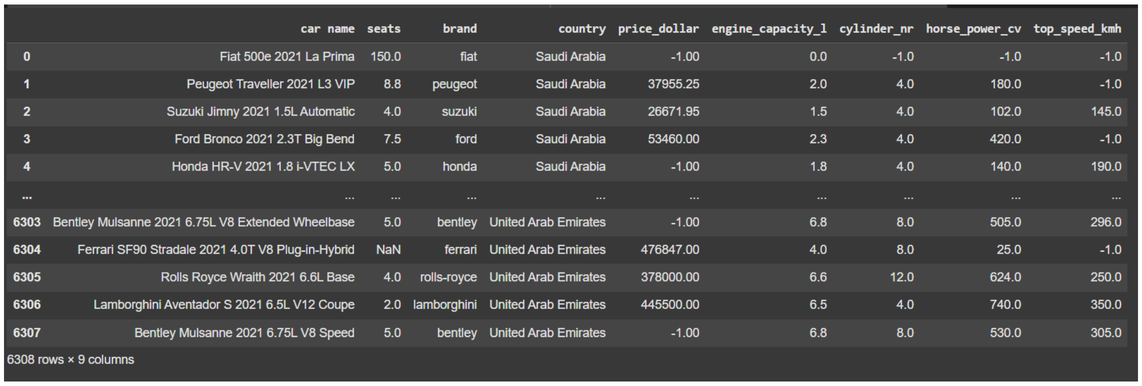
Seats Column Cleanup:

- Non-numeric characters (e.g., "Seater") were removed.
- Invalid entries (e.g., "NA", "N/A") were replaced with NaN, and the column was converted to numeric format.
- Also, it was cleaned and imputed by filling missing values with the mode of the column. The mode imputation was used to keep the samples and to avoid data dropping.

The country column was cleaned by mapping abbreviations to full names for better readability:

- **Example:** 'uae' → 'United Arab Emirates'.
- This ensures uniformity in the data for encoding.

Figure 4 shows the dataset through the data cleaning steps. It is after the previous steps.



	car name	seats	brand	country	price_dollar	engine_capacity_l	cylinder_nr	horse_power_cv	top_speed_kmh
0	Fiat 500e 2021 La Prima	150.0	fiat	Saudi Arabia	-1.00	0.0	-1.0	-1.0	-1.0
1	Peugeot Traveller 2021 L3 VIP	8.8	peugeot	Saudi Arabia	37955.25	2.0	4.0	180.0	-1.0
2	Suzuki Jimny 2021 1.5L Automatic	4.0	suzuki	Saudi Arabia	26671.95	1.5	4.0	102.0	145.0
3	Ford Bronco 2021 2.3T Big Bend	7.5	ford	Saudi Arabia	53460.00	2.3	4.0	420.0	-1.0
4	Honda HR-V 2021 1.8 i-VTEC LX	5.0	honda	Saudi Arabia	-1.00	1.8	4.0	140.0	190.0
...	...	...	...	...	...	...	...	...	...
6303	Bentley Mulsanne 2021 6.75L V8 Extended Wheelbase	5.0	bentley	United Arab Emirates	-1.00	6.8	8.0	505.0	296.0
6304	Ferrari SF90 Stradale 2021 4.0T V8 Plug-in-Hybrid	NaN	ferrari	United Arab Emirates	476847.00	4.0	8.0	25.0	-1.0
6305	Rolls Royce Wraith 2021 6.6L Base	4.0	rolls-royce	United Arab Emirates	378000.00	6.6	12.0	624.0	250.0
6306	Lamborghini Aventador S 2021 6.5L V12 Coupe	2.0	lamborghini	United Arab Emirates	445500.00	6.5	4.0	740.0	350.0
6307	Bentley Mulsanne 2021 6.75L V8 Speed	5.0	bentley	United Arab Emirates	-1.00	6.8	8.0	530.0	305.0

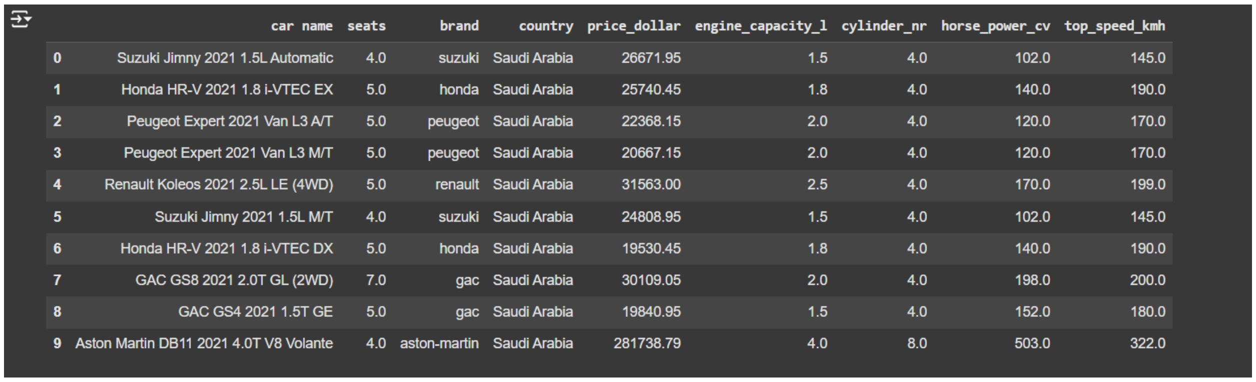
6308 rows × 9 columns

Figure 4: The Dataset Through Data Cleaning Steps

After that, the rows with invalid data were removed to ensure data quality.

The rows with values marked as -1 in key columns (price_dollar, engine_capacity_l, cylinder_nr, horse_power_cv, and top_speed_kmh) were removed. After that, the dataset was 4397 rows.

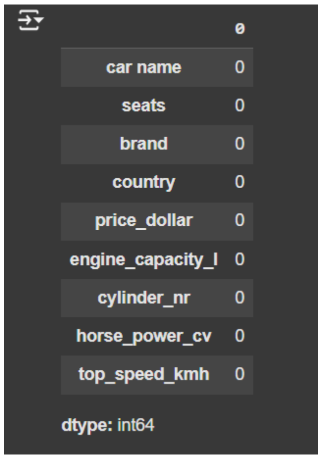
Figure 5 shows the first ten rows of the dataset after the rows with invalid data were removed.



	car name	seats	brand	country	price_dollar	engine_capacity_l	cylinder_nr	horse_power_cv	top_speed_kmh
0	Suzuki Jimny 2021 1.5L Automatic	4.0	suzuki	Saudi Arabia	26671.95	1.5	4.0	102.0	145.0
1	Honda HR-V 2021 1.8 i-VTEC EX	5.0	honda	Saudi Arabia	25740.45	1.8	4.0	140.0	190.0
2	Peugeot Expert 2021 Van L3 A/T	5.0	peugeot	Saudi Arabia	22368.15	2.0	4.0	120.0	170.0
3	Peugeot Expert 2021 Van L3 M/T	5.0	peugeot	Saudi Arabia	20667.15	2.0	4.0	120.0	170.0
4	Renault Koleos 2021 2.5L LE (4WD)	5.0	renault	Saudi Arabia	31563.00	2.5	4.0	170.0	199.0
5	Suzuki Jimny 2021 1.5L M/T	4.0	suzuki	Saudi Arabia	24808.95	1.5	4.0	102.0	145.0
6	Honda HR-V 2021 1.8 i-VTEC DX	5.0	honda	Saudi Arabia	19530.45	1.8	4.0	140.0	190.0
7	GAC GS8 2021 2.0T GL (2WD)	7.0	gac	Saudi Arabia	30109.05	2.0	4.0	198.0	200.0
8	GAC GS4 2021 1.5T GE	5.0	gac	Saudi Arabia	19840.95	1.5	4.0	152.0	180.0
9	Aston Martin DB11 2021 4.0T V8 Volante	4.0	aston-martin	Saudi Arabia	281738.79	4.0	8.0	503.0	322.0

Figure 5: The First Ten Rows of the Dataset

Figure 6 shows all columns (features) with its number of null values after the missing values were handled.

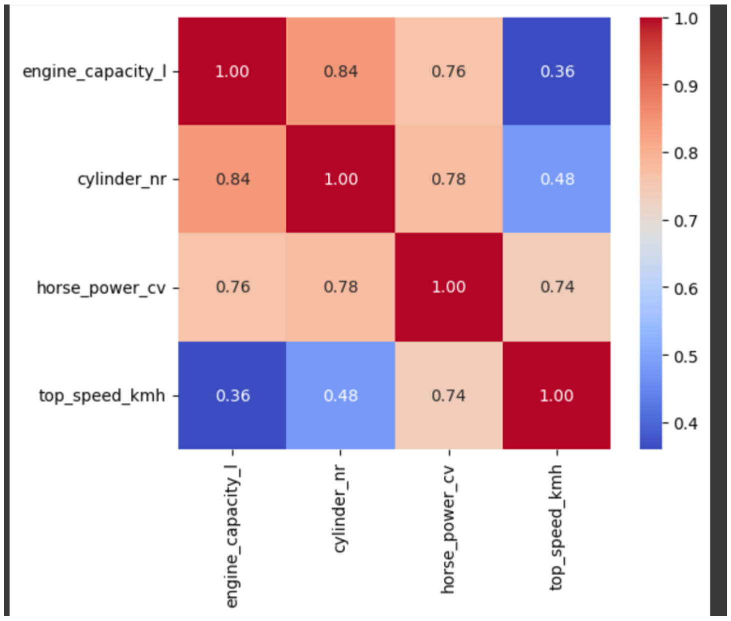


```
                    0
car name            0
seats               0
brand               0
country             0
price_dollar        0
engine_capacity_l   0
cylinder_nr         0
horse_power_cv      0
top_speed_kmh       0

dtype: int64
```

Figure 6: The Number of Null Values After the Missing Values were Handled

3.6 Correlation Between Features

Figure 7 shows the correlation results between features using heatmap.



	engine_capacity_l	cylinder_nr	horse_power_cv	top_speed_kmh
engine_capacity_l	1.00	0.84	0.76	0.36
cylinder_nr	0.84	1.00	0.78	0.48
horse_power_cv	0.76	0.78	1.00	0.74
top_speed_kmh	0.36	0.48	0.74	1.00

Figure 7: Correlation Results Between Features

As noted, the correlations between features are somehow high, for example the engine capacity and cylinder and horsepower while the top speed is a bit low in most of them and that's give us indication about most important features.

3.7 Outlier Detection and Removal

Outliers were removed from numeric columns using the Interquartile Range (IQR) method:

For each numeric column except the price (target) and the seats:

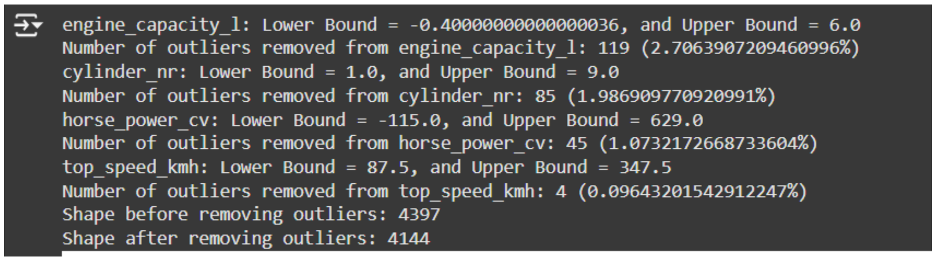
The lower and upper bounds were calculated as:

lower_bound = q1 - 1.5 * iqr

upper_bound = q3 + 1.5 * iqr

- Rows outside these bounds were dropped.

Figure 8 shows the results before and after removing the outliers.



```
engine_capacity_l: Lower Bound = -0.40000000000000036, and Upper Bound = 6.0
Number of outliers removed from engine_capacity_l: 119 (2.7063907209460996%)
cylinder_nr: Lower Bound = 1.0, and Upper Bound = 9.0
Number of outliers removed from cylinder_nr: 85 (1.986909770920991%)
horse_power_cv: Lower Bound = -115.0, and Upper Bound = 629.0
Number of outliers removed from horse_power_cv: 45 (1.0732172668733604%)
top_speed_kmh: Lower Bound = 87.5, and Upper Bound = 347.5
Number of outliers removed from top_speed_kmh: 4 (0.09643201542912247%)
Shape before removing outliers: 4397
Shape after removing outliers: 4144
```

Figure 8: The Results Before and After Removing Outliers

Figure 9 shows the distribution of key vehicle attributes after outlier removal. For **engine capacity (L)**, most values lie between 1 and 4 liters, with a single minor outlier at 6 liters. The **number of cylinders** is distributed between 4 and 6, with no significant outliers remaining. For **horsepower (CV)**, most values range between 150 and 400, with the median around 250 CV, though a few high outliers persist near 600 CV. Lastly, **top speed (km/h)** shows a clean distribution between 150 and 300 km/h, with a median close to 250 km/h and no outliers. These cleaned distributions ensure reliable and representative analysis of the dataset.

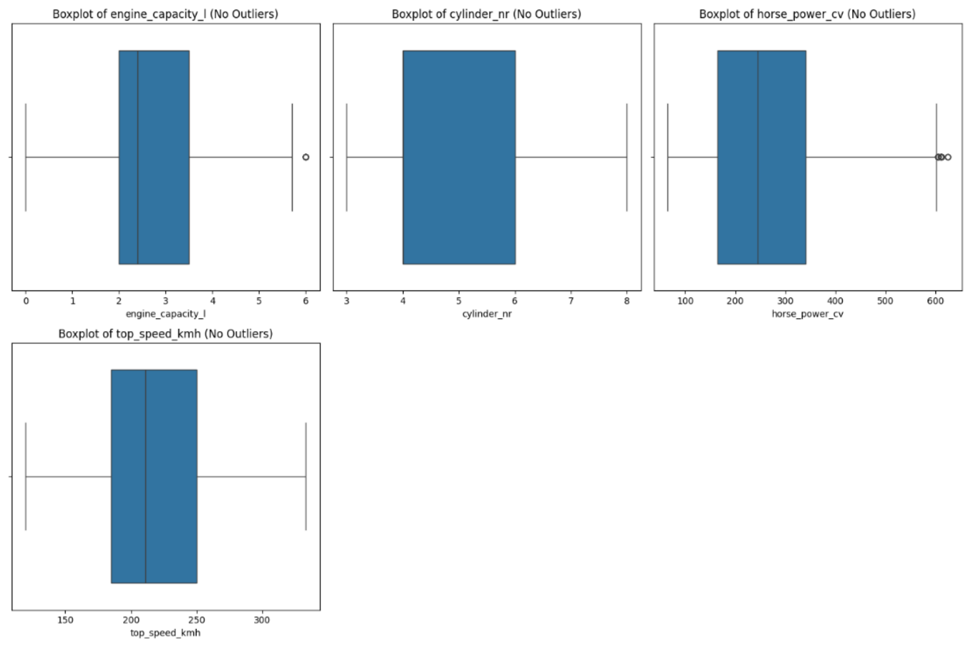
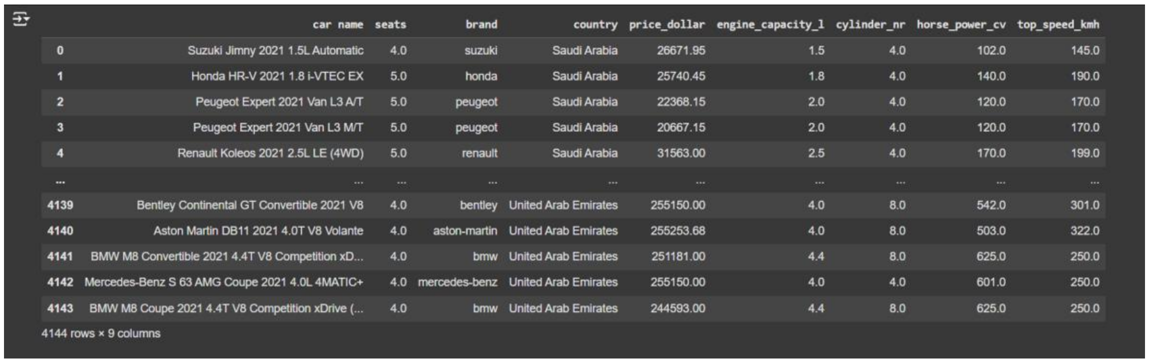


Figure 9: The Key Vehicle Attributes After the Outliers Removal

As for seats, since this attribute does not have a wide range of unique values, no outlier removal was performed. The dataset shows that most vehicles have 5 seats, followed by 4, 7, and 2 seats, while configurations with more than 8 seats are relatively rare.

Also, as for price, since this attribute is the target feature, the outliers removal wasn't applied to avoid any biasing in prediction.

The final cleaned dataset was reset for indexing to maintain consistency. Figure 10 shows the final cleaned dataset.



	car name	seats	brand	country	price_dollar	engine_capacity_l	cylinder_nr	horse_power_cv	top_speed_kmh
0	Suzuki Jimny 2021 1.5L Automatic	4.0	suzuki	Saudi Arabia	26671.95	1.5	4.0	102.0	145.0
1	Honda HR-V 2021 1.8 i-VTEC EX	5.0	honda	Saudi Arabia	25740.45	1.8	4.0	140.0	190.0
2	Peugeot Expert 2021 Van L3 A/T	5.0	peugeot	Saudi Arabia	22368.15	2.0	4.0	120.0	170.0
3	Peugeot Expert 2021 Van L3 M/T	5.0	peugeot	Saudi Arabia	20667.15	2.0	4.0	120.0	170.0
4	Renault Koleos 2021 2.5L LE (4WD)	5.0	renault	Saudi Arabia	31563.00	2.5	4.0	170.0	199.0
...	...	...	...	...	...	...	...	...	...
4139	Bentley Continental GT Convertible 2021 V8	4.0	bentley	United Arab Emirates	255150.00	4.0	8.0	542.0	301.0
4140	Aston Martin DB11 2021 4.0T V8 Volante	4.0	aston-martin	United Arab Emirates	255253.68	4.0	8.0	503.0	322.0
4141	BMW M8 Convertible 2021 4.4T V8 Competition xD...	4.0	bmw	United Arab Emirates	251181.00	4.4	8.0	625.0	250.0
4142	Mercedes-Benz S 63 AMG Coupe 2021 4.0L 4MATIC+	4.0	mercedes-benz	United Arab Emirates	255150.00	4.0	4.0	601.0	250.0
4143	BMW M8 Coupe 2021 4.4T V8 Competition xDrive (...	4.0	bmw	United Arab Emirates	244593.00	4.4	8.0	625.0	250.0

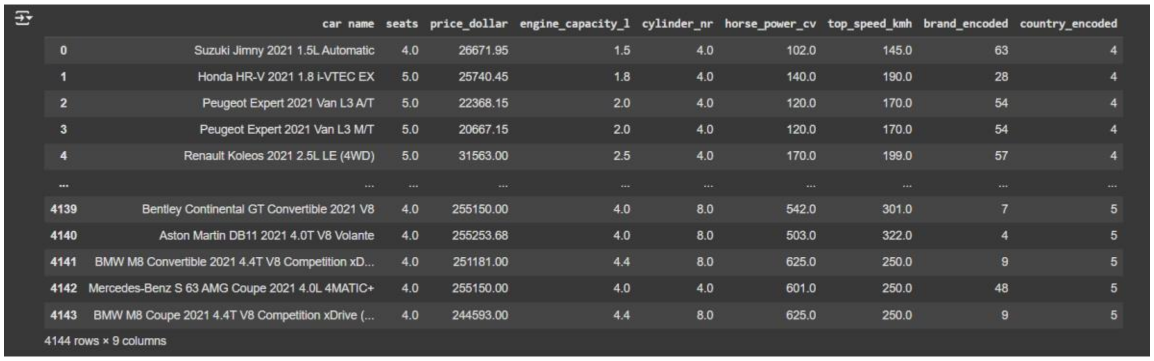
4144 rows × 9 columns

Figure 10: The Final Cleaned Dataset

4 Feature Engineering

4.1 Categorical To Numerical Conversion

Figure 11 shows the conversion of categorical columns (brand and country) to numerical columns. As noted, the label encoder was used. It assigns a unique integer to each category within a feature (categorical column), enabling the categorical data to be used effectively in machine learning models. On the other hand, the one-Hot-Encoding wasn't used because the number of resulted columns (features) will be very large, approximately 85 columns, and this isn't suitable for our dataset and our case. Also, the non-linear (polynomial with higher degrees) models weren't trained on this large number of features (columns). So, the label encoder was used.

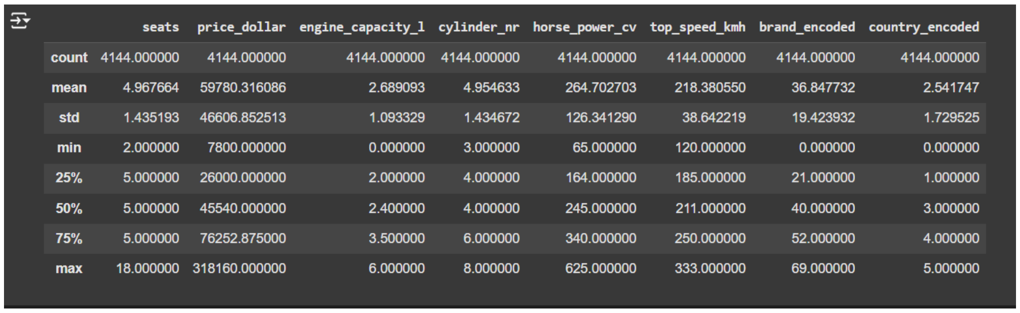


	car name	seats	price_dollar	engine_capacity_l	cylinder_nr	horse_power_cv	top_speed_kmh	brand_encoded	country_encoded
0	Suzuki Jimny 2021 1.5L Automatic	4.0	26671.95	1.5	4.0	102.0	145.0	63	4
1	Honda HR-V 2021 1.8 i-VTEC EX	5.0	25740.45	1.8	4.0	140.0	190.0	28	4
2	Peugeot Expert 2021 Van L3 A/T	5.0	22368.15	2.0	4.0	120.0	170.0	54	4
3	Peugeot Expert 2021 Van L3 M/T	5.0	20667.15	2.0	4.0	120.0	170.0	54	4
4	Renault Koleos 2021 2.5L LE (4WD)	5.0	31563.00	2.5	4.0	170.0	199.0	57	4
...	...	...	...	...	...	...	...	...	...
4139	Bentley Continental GT Convertible 2021 V8	4.0	255150.00	4.0	8.0	542.0	301.0	7	5
4140	Aston Martin DB11 2021 4.0T V8 Volante	4.0	255253.68	4.0	8.0	503.0	322.0	4	5
4141	BMW M8 Convertible 2021 4.4T V8 Competition xD...	4.0	251181.00	4.4	8.0	625.0	250.0	9	5
4142	Mercedes-Benz S 63 AMG Coupe 2021 4.0L 4MATIC+	4.0	255150.00	4.0	4.0	601.0	250.0	48	5
4143	BMW M8 Coupe 2021 4.4T V8 Competition xDrive (...	4.0	244593.00	4.4	8.0	625.0	250.0	9	5

4144 rows × 9 columns

Figure 11: The Label Encoder Results

Figure 12 shows a description about the dataset. This dataset summary provides key statistics for a car dataset with 4144 entries and no missing values. It includes features like the number of seats, price in dollars, engine capacity (liters), cylinder count, horsepower, top speed (km/h), and encoded brand and country values. The average car has approximately 5 seats, costs $59,780, and has an engine capacity of 2.69 liters with 4.95 cylinders, producing 264 horsepower and reaching a top speed of 218 km/h. Prices vary significantly, ranging from $7,800 to $318,160, with a high standard deviation of $46,606, indicating substantial diversity in the dataset. The horsepower ranges from 65 to 625, reflecting a mix of economy and performance cars.

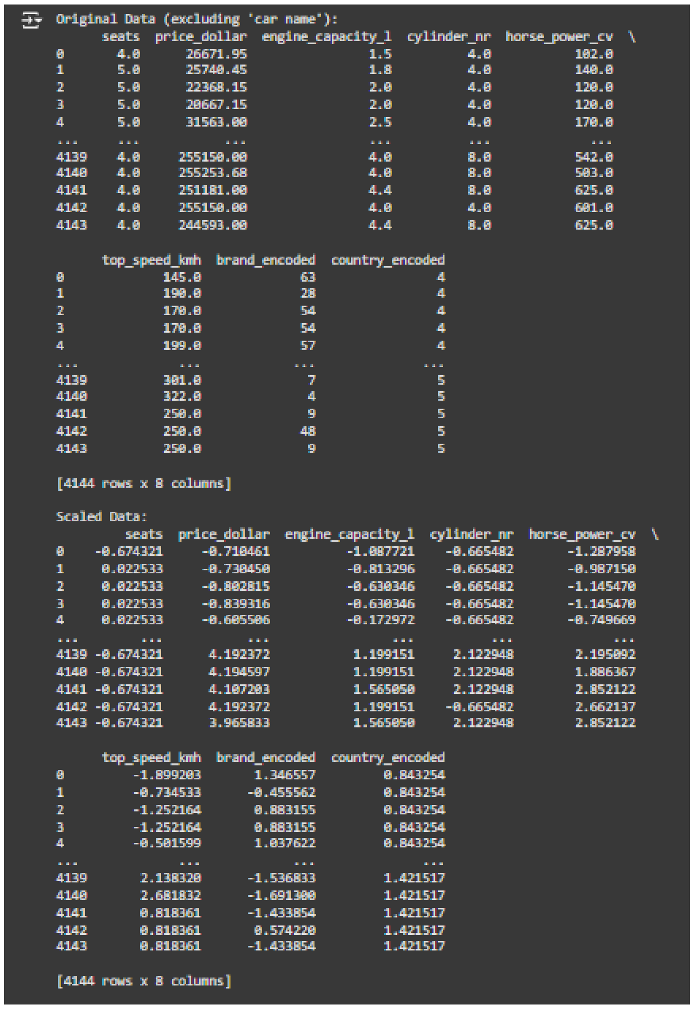


	seats	price_dollar	engine_capacity_l	cylinder_nr	horse_power_cv	top_speed_kmh	brand_encoded	country_encoded
count	4144.000000	4144.000000	4144.000000	4144.000000	4144.000000	4144.000000	4144.000000	4144.000000
mean	4.967664	59780.316086	2.689093	4.954633	264.702703	218.380550	36.847732	2.541747
std	1.435193	46606.852513	1.093329	1.434672	126.341290	38.642219	19.423932	1.729525
min	2.000000	7800.000000	0.000000	3.000000	65.000000	120.000000	0.000000	0.000000
25%	5.000000	26000.000000	2.000000	4.000000	164.000000	185.000000	21.000000	1.000000
50%	5.000000	45540.000000	2.400000	4.000000	245.000000	211.000000	40.000000	3.000000
75%	5.000000	76252.875000	3.500000	6.000000	340.000000	250.000000	52.000000	4.000000
max	18.000000	318160.000000	6.000000	8.000000	625.000000	333.000000	69.000000	5.000000

Figure 12: A Description About the Dataset

4.2 The Normalization

Figure 13 shows the standard scalar normalization method results.



```
Original Data (excluding 'car name'):
      seats  price_dollar  engine_capacity_l  cylinder_nr  horse_power_cv  \
0       4.0      26671.95                1.5          4.0           102.0
1       5.0      25740.45                1.8          4.0           140.0
2       5.0      22368.15                2.0          4.0           120.0
3       5.0      20667.15                2.0          4.0           120.0
4       5.0      31563.00                2.5          4.0           170.0
...     ...           ...                ...          ...             ...
4139    4.0     255150.00                4.0          8.0           542.0
4140    4.0     255253.68                4.0          8.0           503.0
4141    4.0     251181.00                4.4          8.0           625.0
4142    4.0     255150.00                4.0          4.0           601.0
4143    4.0     244593.00                4.4          8.0           625.0

      top_speed_kmh  brand_encoded  country_encoded
0             145.0             63                4
1             190.0             28                4
2             170.0             54                4
3             170.0             54                4
4             199.0             57                4
...             ...            ...              ...
4139          301.0              7                5
4140          322.0              4                5
4141          250.0              9                5
4142          250.0             48                5
4143          250.0              9                5

[4144 rows x 8 columns]

Scaled Data:
         seats  price_dollar  engine_capacity_l  cylinder_nr  horse_power_cv  \
0    -0.674321     -0.710461          -1.087721    -0.665482       -1.287958
1     0.022533     -0.730450          -0.813296    -0.665482       -0.987150
2     0.022533     -0.802815          -0.630346    -0.665482       -1.145470
3     0.022533     -0.839316          -0.630346    -0.665482       -1.145470
4     0.022533     -0.605506          -0.172972    -0.665482       -0.749669
...        ...           ...                ...          ...             ...
4139 -0.674321      4.192372           1.199151     2.122948        2.195092
4140 -0.674321      4.194597           1.199151     2.122948        1.886367
4141 -0.674321      4.107203           1.565050     2.122948        2.852122
4142 -0.674321      4.192372           1.199151    -0.665482        2.662137
4143 -0.674321      3.965833           1.565050     2.122948        2.852122

      top_speed_kmh  brand_encoded  country_encoded
0         -1.899203       1.346557         0.843254
1         -0.734533      -0.455562         0.843254
2         -1.252164       0.883155         0.843254
3         -1.252164       0.883155         0.843254
4         -0.501599       1.037622         0.843254
...             ...            ...              ...
4139       2.138320      -1.536833         1.421517
4140       2.681832      -1.691300         1.421517
4141       0.818361      -1.433854         1.421517
4142       0.818361       0.574220         1.421517
4143       0.818361      -1.433854         1.421517

[4144 rows x 8 columns]
```

Figure 13: The Standard Scalar Normalization Results

The Standard Scaler is a data preprocessing technique used in machine learning to standardize or normalize features by removing the mean and scaling them to unit variance. This type of normalization was chosen because it has a lot of benefits like:

- **Improved Model Performance:** Helps algorithms converge faster by normalizing feature magnitudes.
- **Better Comparability:** Features with different units or scales are brought to a similar range.
- **Enhanced Numerical Stability:** Reduces computational errors in optimization algorithms.
- It ensures that all features contribute equally to the learning process, especially for algorithms sensitive to the scale of input data.

The car name feature (column) was dropped due to its data type (object) and it isn’t useful for learning models especially regression models.

5 Splitting the Dataset

The dataset was spilt as 60% training dataset, 20% testing dataset and 20% validation dataset.

The dataset for each one as following:

- Train shape: (2486, 8)
- Test shape: (829, 8)
- Validation shape: (829, 8)

6 Building Regression Models

6.1 Evaluation Metrices

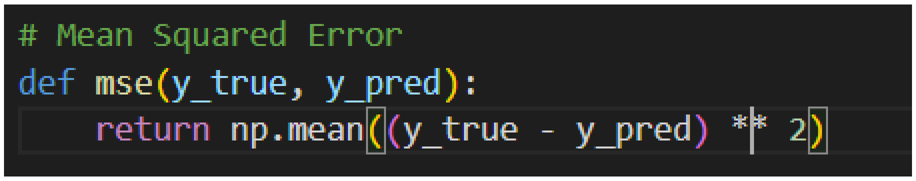
MSE:

Mean Squared Error (MSE) is a commonly used metric in regression analysis to evaluate the accuracy of a model's predictions. It measures the average of the squared differences between the predicted values and the actual values. The formula for MSE is given as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Figure 14: MSE Equation

Where n is the number of data points, y_i is the actual value, and $y^{\wedge}{}_i$ is the predicted value. Squaring the errors ensures that larger deviations from the actual values are penalized more heavily than smaller ones. This makes MSE sensitive to outliers, as even a single large error can significantly increase the overall score. MSE is often used when large errors are particularly undesirable. However, its sensitivity to outliers can sometimes be a drawback in datasets with extreme values. Figure 15 shows the MSE implementation.



```
# Mean Squared Error
def mse(y_true, y_pred):
    return np.mean((y_true - y_pred) ** 2)
```

Figure 15: MSE Implementation

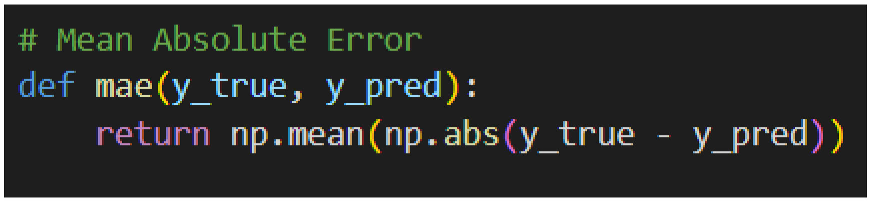
Mean Absolute Error (MAE):

Mean Absolute Error (MAE) is another widely used metric for regression that measures the average magnitude of the errors in a set of predictions, without considering their direction. The formula for MAE is:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Figure 16: MAE Equation

Unlike MSE, MAE computes the absolute differences between predicted and actual values, avoiding the squaring operation. This makes MAE less sensitive to outliers compared to MSE. It provides a more interpretable metric, as it represents the average absolute deviation of predictions from the true values in the same units as the target variable. However, because it does not square the errors, MAE does not penalize larger deviations as strongly as MSE, which might make it less suitable when extreme errors are particularly critical. Figure 17 shows the MAE implementation.



```
# Mean Absolute Error
def mae(y_true, y_pred):
    return np.mean(np.abs(y_true - y_pred))
```

Figure 17: MAE Implementation

R^2 Score (Coefficient of Determination):

The R^2 score, also known as the coefficient of determination, measures the proportion of variance in the dependent variable that is explained by the independent variables in the model. It is defined as:

$$R^2 = 1 - \frac{\mathrm{SS}_{\mathrm{res}}}{\mathrm{SS}_{\mathrm{tot}}}$$

Figure 18: R Squared Score Equation

Where SS_{res} is the sum of squared residuals and SS_{tot} is the total sum of squares.

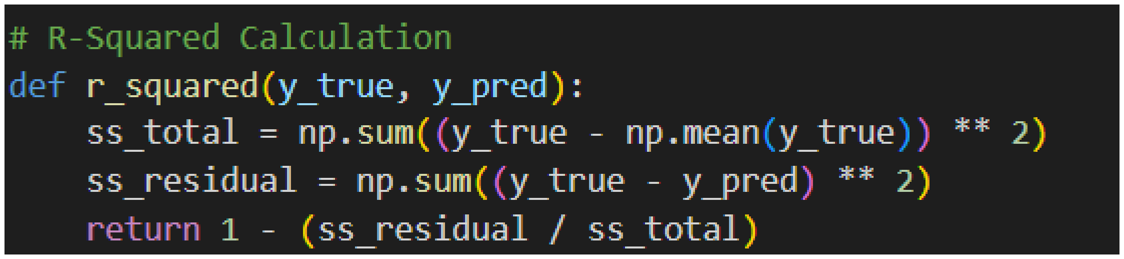
$$\mathrm{SS}_{\mathrm{res}} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Figure 19: Sum of Squared Redial Equation

$$\mathrm{SS}_{\mathrm{tot}} = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

Figure 20: The Total Sum of Squares Equation

With y^ being the mean of the actual values. R^2 ranges from 0 to 1, where 1 indicates perfect predictions, and 0 means the model explains no variance, performing as poorly as simply predicting the mean of the target variable. Negative values can occur if the model performs worse than a baseline model. While R^2 is useful for understanding the explanatory power of a model, it does not directly reflect prediction accuracy and can be misleading in some cases, such as with non-linear relationships. Figure 21 shows the R^2 implementation.



```python
# R-Squared Calculation
def r_squared(y_true, y_pred):
    ss_total = np.sum((y_true - np.mean(y_true)) ** 2)
    ss_residual = np.sum((y_true - y_pred) ** 2)
    return 1 - (ss_residual / ss_total)
```

Figure 21: R Squared Implementation

6.2 Linear Regression

6.2.1 Closed Form

Linear regression in its closed form provides a direct mathematical solution for determining the optimal coefficients of a linear model, bypassing iterative approaches like gradient descent. The linear regression model assumes that the relationship between the independent variables (features) and the dependent variable (target) is linear, expressed as $y=X\beta+\epsilon$, where y is the vector of observed outputs, X is the matrix of input features, β is the vector of coefficients to be estimated, and ϵ represents the residual errors. The objective of linear regression is to minimize the sum of squared errors (SSE), defined as $\|y-X\beta\|^2$.

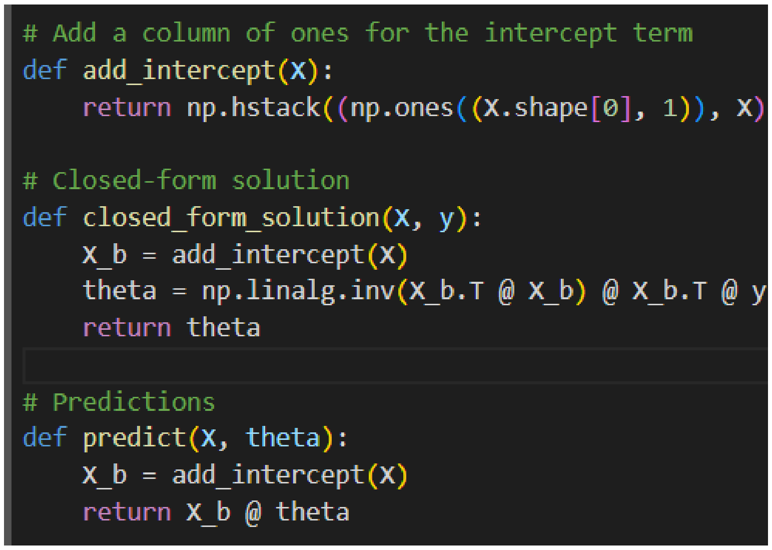
The closed-form solution is derived by setting the gradient of the SSE with respect to β to zero, leading to the formula $\beta=(X^TX)^{-1} X^Ty$.

This equation provides the coefficients directly by leveraging linear algebra. Here,

X^TX is the Gram matrix, a square, symmetric matrix obtained from the dot product of the feature matrix with itself, and $(X^TX)^{-1}$ is its inverse. The term X^Ty captures the correlation between the features and the target. This solution assumes that X^TX is invertible; if not, alternative approaches like regularization (e.g., Ridge Regression) may be necessary.

The closed-form solution is advantageous for small datasets because it provides an exact and efficient computation of the coefficients. However, it has limitations, especially for large datasets or high-dimensional data, where the computational cost of matrix inversion $O(n^3)$ becomes prohibitive. Additionally, the method is sensitive to multicollinearity (high correlation among features) or numerical instability if the Gram matrix is poorly conditioned. In practice, while the closed-form solution is useful for theoretical understanding and small-scale problems, iterative methods like gradient descent are often preferred for large-scale application.

Figure 22 shows the closed form implementation and the prediction implementation.



```
# Add a column of ones for the intercept term
def add_intercept(X):
    return np.hstack((np.ones((X.shape[0], 1)), X)

# Closed-form solution
def closed_form_solution(X, y):
    X_b = add_intercept(X)
    theta = np.linalg.inv(X_b.T @ X_b) @ X_b.T @ y
    return theta


# Predictions
def predict(X, theta):
    X_b = add_intercept(X)
    return X_b @ theta
```

Figure 22: The Closed Form Implementation

6.2.2 Gradient Descent

Gradient descent is an iterative optimization algorithm used to minimize a cost function and is commonly employed in linear regression when the dataset is large or when computational efficiency is a concern. Instead of solving the coefficients directly as in the closed-form solution, gradient descent updates the coefficients iteratively by moving in the direction of the steepest descent of the cost function. For linear regression, the cost function is typically the mean squared error (MSE), defined as $J(\beta)= (1/2m)\ \|y-X\beta\|^2$, where m is the number of training examples.

The algorithm starts with an initial guess for the coefficients, $\beta^{(0)}$, and updates them iteratively using the rule: $\beta^{(k+1)} = \beta^{(k)} - \alpha \nabla J(\beta^{(k)})$

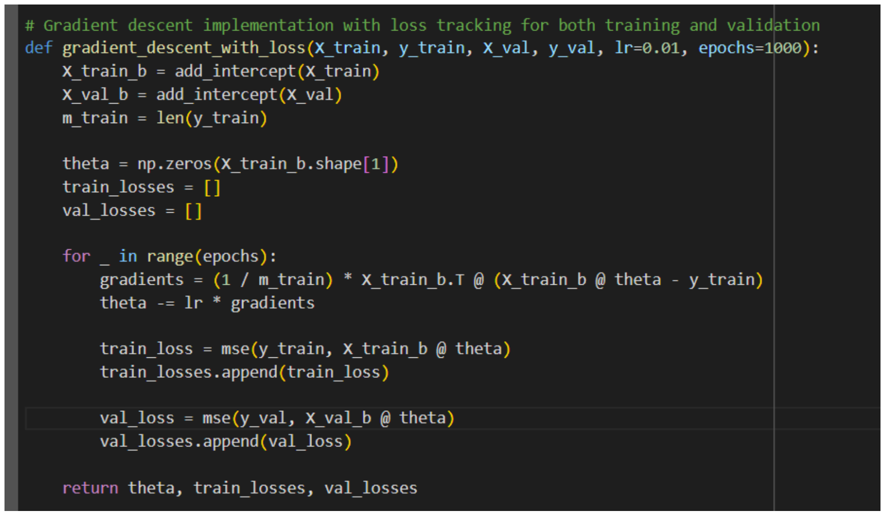
Here, α is the learning rate, a hyperparameter that determines the step size of each iteration, and $\nabla J(\beta)$ is the gradient of the cost function with respect to the coefficients. For linear regression, the gradient is given by: $\nabla J(\beta)=(-1/m)\ X^T (y-X\beta)$

At each iteration, the algorithm computes the gradient and adjusts the coefficients by moving in the direction opposite to the gradient, which reduces the cost function. This process continues until the algorithm converges to a minimum, typically when the change in the cost function or the coefficients is smaller than a predefined threshold.

Gradient descent has several advantages, particularly for large datasets or high-dimensional data, as it avoids the computationally expensive matrix inversion required in the closed-form solution. However, its performance depends heavily on the choice of the learning rate. A learning rate that is too small can make convergence slow, while a learning rate that is too large can cause the algorithm to overshoot the minimum or fail to converge. There are also variants of gradient descent, such as stochastic gradient descent (SGD) and mini-batch gradient descent, which can further improve efficiency and scalability.

While gradient descent may require more iterations to reach a solution compared to the closed-form method, it is often preferred for large-scale problems and when handling datasets with complex structures or high dimensions.

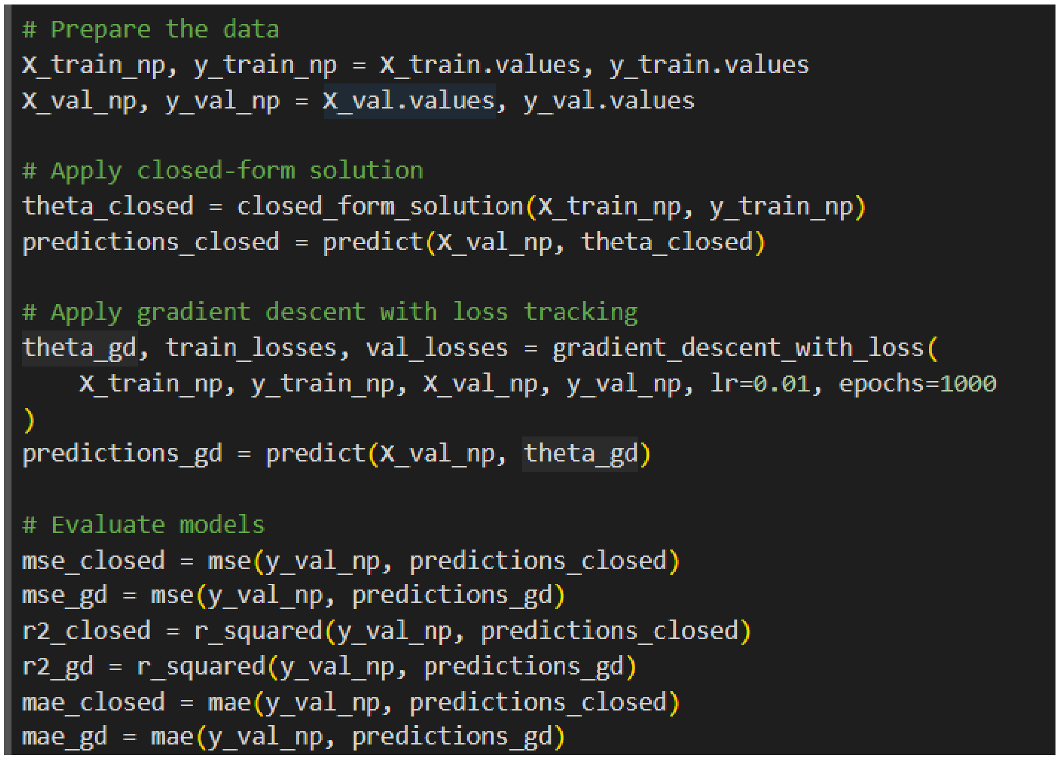
Figure 23 shows the Gradient descent implementation.



```python
# Gradient descent implementation with loss tracking for both training and validation
def gradient_descent_with_loss(X_train, y_train, X_val, y_val, lr=0.01, epochs=1000):
    X_train_b = add_intercept(X_train)
    X_val_b = add_intercept(X_val)
    m_train = len(y_train)

    theta = np.zeros(X_train_b.shape[1])
    train_losses = []
    val_losses = []

    for _ in range(epochs):
        gradients = (1 / m_train) * X_train_b.T @ (X_train_b @ theta - y_train)
        theta -= lr * gradients

        train_loss = mse(y_train, X_train_b @ theta)
        train_losses.append(train_loss)

        val_loss = mse(y_val, X_val_b @ theta)
        val_losses.append(val_loss)

    return theta, train_losses, val_losses
```

Figure 23: The Gradient Descent Implementation

Figure 24 shows the applying of closed-form and Gradient descent. Also, the evaluation on the validation data.



```python
# Prepare the data
X_train_np, y_train_np = X_train.values, y_train.values
X_val_np, y_val_np = X_val.values, y_val.values

# Apply closed-form solution
theta_closed = closed_form_solution(X_train_np, y_train_np)
predictions_closed = predict(X_val_np, theta_closed)

# Apply gradient descent with loss tracking
theta_gd, train_losses, val_losses = gradient_descent_with_loss(
    X_train_np, y_train_np, X_val_np, y_val_np, lr=0.01, epochs=1000
)
predictions_gd = predict(X_val_np, theta_gd)

# Evaluate models
mse_closed = mse(y_val_np, predictions_closed)
mse_gd = mse(y_val_np, predictions_gd)
r2_closed = r_squared(y_val_np, predictions_closed)
r2_gd = r_squared(y_val_np, predictions_gd)
mae_closed = mae(y_val_np, predictions_closed)
mae_gd = mae(y_val_np, predictions_gd)
```

Figure 24: The Applying of Closed-Form and Gradient Descent

6.2.3 Results

- **Closed-form solution parameters (weights):** [0.00425179 -0.04504214 -0.14967301 0.16514485 0.7370466 0.18881489 0.0106752 0.04549837]
- **Gradient descent parameters (weights):** [0.00520215 -0.03815865 -0.05491023 0.15217944 0.60055529 0.27226325 0.00544603 0.04955858]
- **MSE (Closed-form):** 0.17692151166874895
- **MSE (Gradient Descent):** 0.17979041362444645
- **MAE (Closed-form):** 0.3055672219663439
- **MAE (Gradient Descent):** 0.30853151175824184
- **R-squared (Closed-form):** 0.8018672178798371
- **R-squared (Gradient Descent):** 0.7986543608295503
    - ➢ As noted, the closed form better a little than the Gradient descent.

Figure 25 shows the true versus predicted values for closed form and gradient descent.

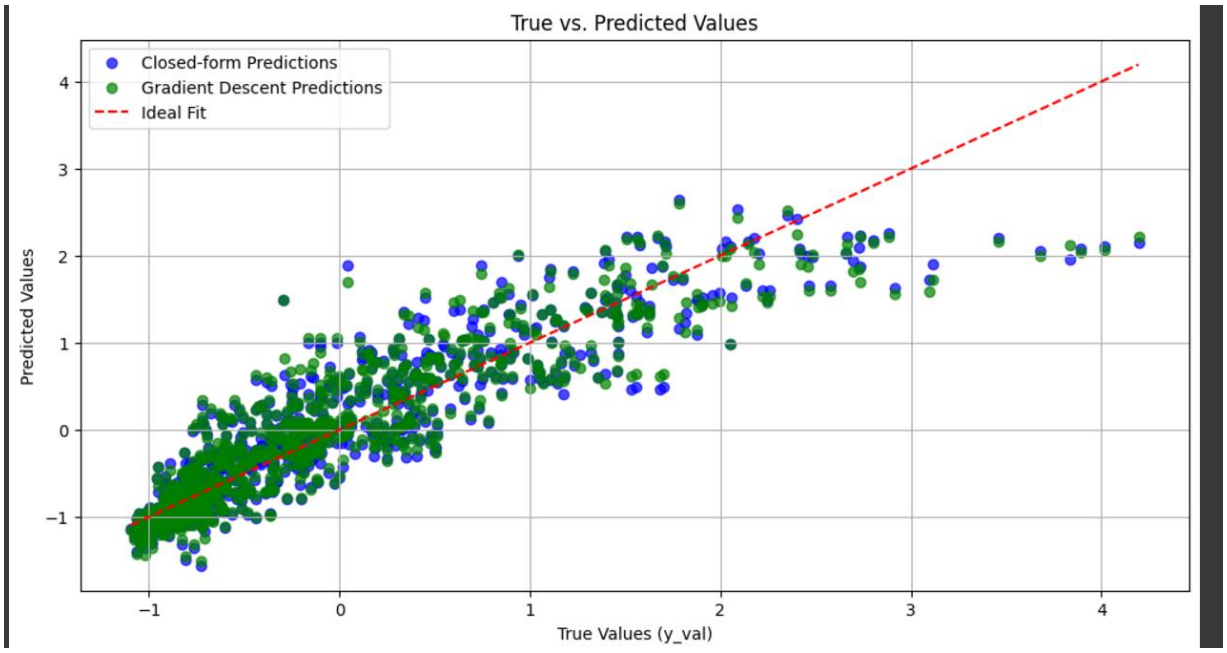


Figure 25: True Vs. Predicted Values for Closed Form and Gradient Descent

As noted, the predicted values of the Gradient descent are approximately closer to the closed-form predicted values.

Figure 26 shows the learning curve for the gradient descent.

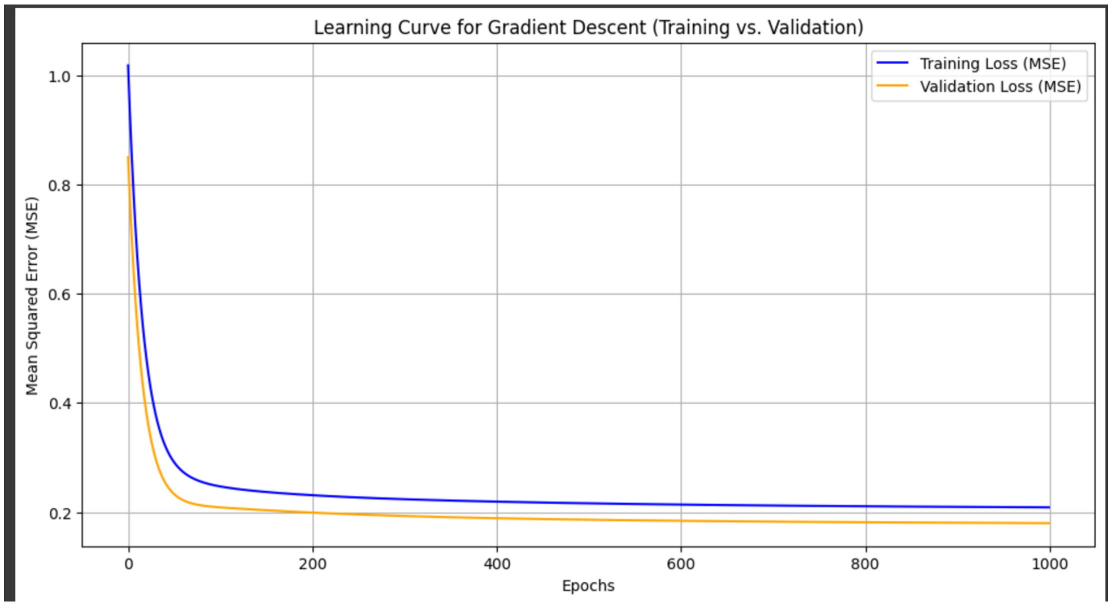


Figure 26: The Learning Curve for the Gradient Descent

As noted, the validation error and the training error are small through increase epochs so our model performs well.

Figure 27 shows the residual plot (closed-form vs. Gradient descent).

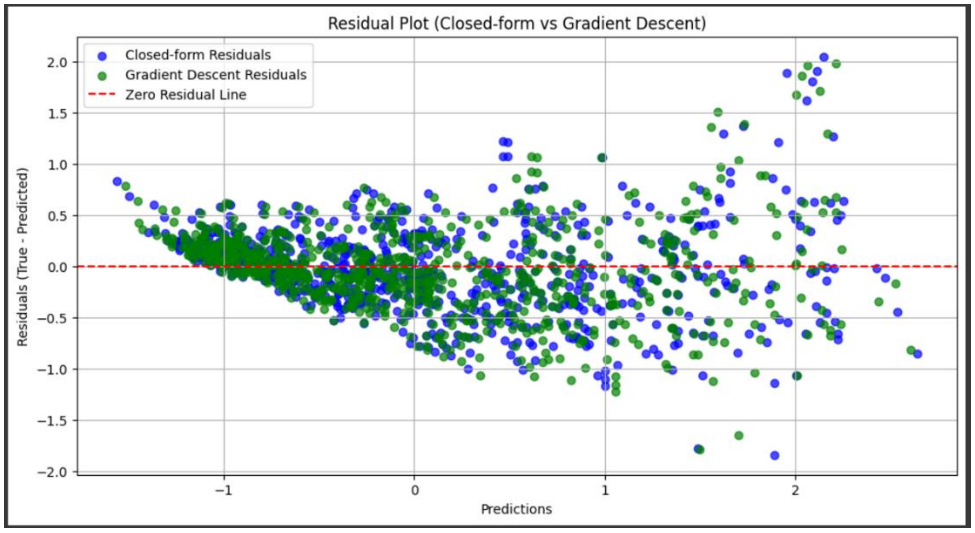


Figure 27: Residual Plot (Closed-form Vs. Gradient Descent)

A residuals plot is a graphical representation used to evaluate the fit of a regression model by displaying the residuals, which are the differences between the observed values and the predicted values of the dependent variable. It typically plots the residuals on the y-axis against either the predicted values or an independent variable on the x-axis.

Residual=Observed Value−Predicted Value.

As noted, for the linear regression the datapoints is close to the fit line while there are some that outlies far but that indicate that the model is perform well.

Figure 28 shows the error distribution (closed-form vs. Gradient descent).

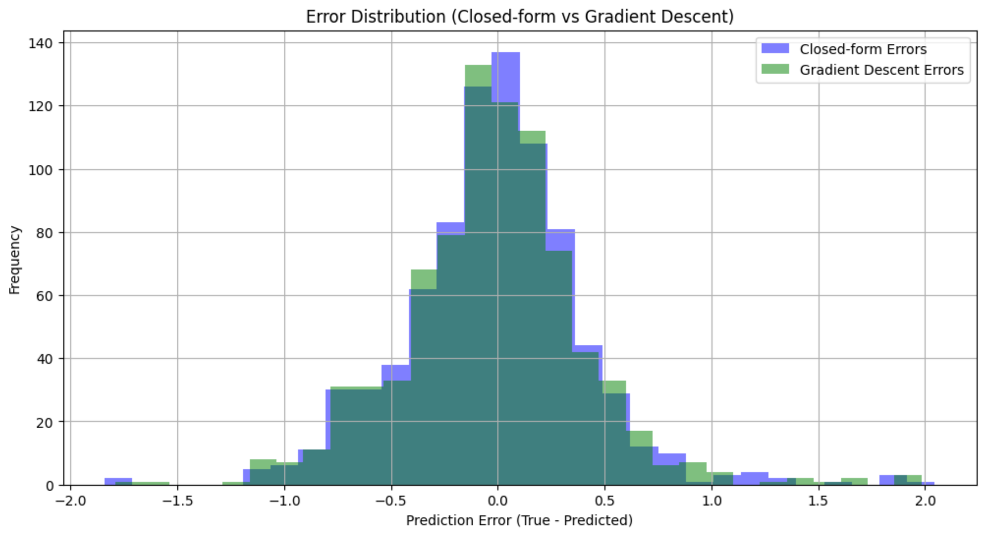


Figure 28: Error Distribution (Closed-form Vs. Gradient Descent)

As noted, the error distribution for the two models are very close and very small.

6.3 Grid Search, Lasso and Ridge Regularization

6.3.1 Grid Search

Grid search is a systematic approach to hyperparameter tuning that exhaustively tests all possible combinations of a predefined set of hyperparameters to identify the optimal configuration for a machine learning model. The goal is to find the hyperparameter values that yield the best performance based on a chosen evaluation metric, such as accuracy, precision, or mean squared error. Grid search operates by creating a grid of hyperparameter values, where each dimension of the grid corresponds to a specific hyperparameter and its range of possible values. For each combination in the grid, the model is trained and validated using cross-validation to ensure robust performance estimation. Cross-validation splits the data into training and validation subsets multiple times, and the average performance across these splits is calculated for each hyperparameter combination. This prevents overfitting to a single validation set and provides a more reliable measure of model performance.

While grid search is exhaustive and thorough, it can be computationally expensive, especially when the hyperparameter grid is large or when the model training process is time-consuming. The computational cost grows exponentially with the number of hyperparameters and their respective value ranges. To mitigate this, practitioners often limit the grid to a smaller set of carefully chosen hyperparameter values or use alternative methods like random search, which samples combinations randomly and is often more efficient. Despite its cost, grid search remains a popular and effective method for optimizing hyperparameters when computational resources are sufficient.

6.3.2 Lasso Regularization

Lasso regularization, or Least Absolute Shrinkage and Selection Operator, is a type of linear regression technique that adds an L1 penalty to the loss function. The objective function for lasso regression is given by:

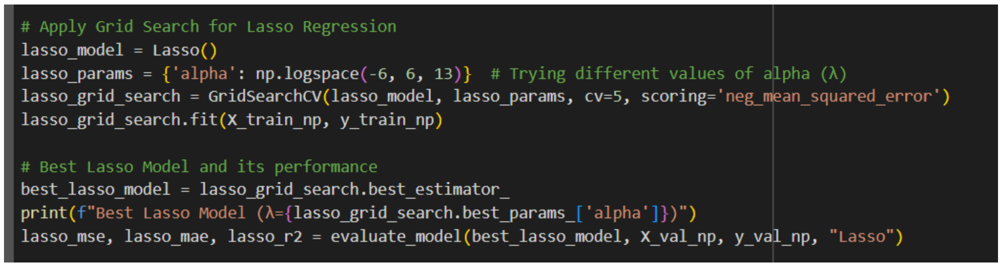
$$\textbf{Minimize: } \|y - X\beta\|^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

Figure 29: LASSO Regularization Equation

Here, λ is the regularization parameter that controls the strength of the penalty, p is the number of features, and β_j are the coefficients of the model. The L1 penalty encourages sparsity by shrinking some coefficients exactly to zero, effectively performing feature selection. This makes lasso particularly useful when there are many features, but only a subset is truly relevant for the model.

By eliminating irrelevant features, lasso simplifies the model and reduces the risk of overfitting. However, it can struggle in situations where features are highly correlated, as it tends to select one feature from a group of correlated variables and ignores the rest. Additionally, lasso may not perform as well as other methods when the number of features exceeds the number of observations.

Figure 30 shows the lasso with grid search implementation.



```
# Apply Grid Search for Lasso Regression
lasso_model = Lasso()
lasso_params = {'alpha': np.logspace(-6, 6, 13)}  # Trying different values of alpha (λ)
lasso_grid_search = GridSearchCV(lasso_model, lasso_params, cv=5, scoring='neg_mean_squared_error')
lasso_grid_search.fit(X_train_np, y_train_np)

# Best Lasso Model and its performance
best_lasso_model = lasso_grid_search.best_estimator_
print(f"Best Lasso Model (λ={lasso_grid_search.best_params_['alpha']})")
lasso_mse, lasso_mae, lasso_r2 = evaluate_model(best_lasso_model, X_val_np, y_val_np, "Lasso")
```

Figure 30: LASSO with Grid Search Implementation

6.3.3 Ridge Regularization

Ridge regularization, also known as Tikhonov regularization, adds an L2 penalty to the loss function. The objective function for ridge regression is:

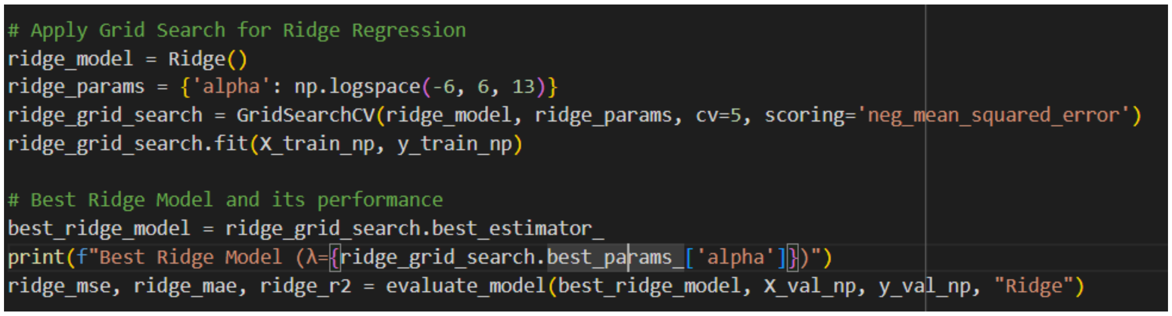
$$\text{Minimize: } \|y - X\beta\|^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

Figure 31: Ridge Regularization Equation

In this formulation, the L2 penalty discourages large coefficient values by adding the squared magnitude of the coefficients to the loss. Unlike lasso, ridge does not shrink coefficients to zero; instead, it reduces them proportionally. This makes ridge regression particularly effective in handling multicollinearity, where features are highly correlated, as it distributes the coefficient values among correlated features.

Ridge regression is useful when all features are believed to have some relevance to the model, even if their individual contributions vary in magnitude. However, it does not perform feature selection, so it retains all features in the final model, which may not be ideal when interpretability is a concern. Ridge is less suited for sparse data where many features are irrelevant.

Figure 32 shows the ridge with grid search implementation.



```python
# Apply Grid Search for Ridge Regression
ridge_model = Ridge()
ridge_params = {'alpha': np.logspace(-6, 6, 13)}
ridge_grid_search = GridSearchCV(ridge_model, ridge_params, cv=5, scoring='neg_mean_squared_error')
ridge_grid_search.fit(X_train_np, y_train_np)

# Best Ridge Model and its performance
best_ridge_model = ridge_grid_search.best_estimator_
print(f"Best Ridge Model (λ={ridge_grid_search.best_params_['alpha']})")
ridge_mse, ridge_mae, ridge_r2 = evaluate_model(best_ridge_model, X_val_np, y_val_np, "Ridge")
```

Figure 32: Ridge with Grid Search Implementation

6.3.4 Results

Best Ridge Model (λ=1.0)

Ridge Performance on Validation Set:

- MSE: 0.1769
- MAE: 0.3055
- R-squared: 0.8019

Best Lasso Model (λ=1e-05)

Lasso Performance on Validation Set:

- MSE: 0.1769
- MAE: 0.3056
- R-squared: 0.8019

Comparison of Models:

Ridge MSE: 0.1769, MAE: 0.3055, R-squared: 0.8019

Lasso MSE: 0.1769, MAE: 0.3056, R-squared: 0.8019

As noted, the two Models results are the same and this will be shown on the visualization. But they're better than closed-form and Gradient descent.

Figure 33 shows the MSE vs. lambda for Ridge and Lasso with grid search. For Ridge, the test score improves significantly at higher values of λ (e.g., beyond 10^3), suggesting strong regularization benefits. For Lasso, the test score increases sharply at λ values near 10^-1 and stabilizes thereafter, indicating that it achieves its best performance with moderate regularization. Both plots emphasize the importance of tuning λ for optimal model performance.

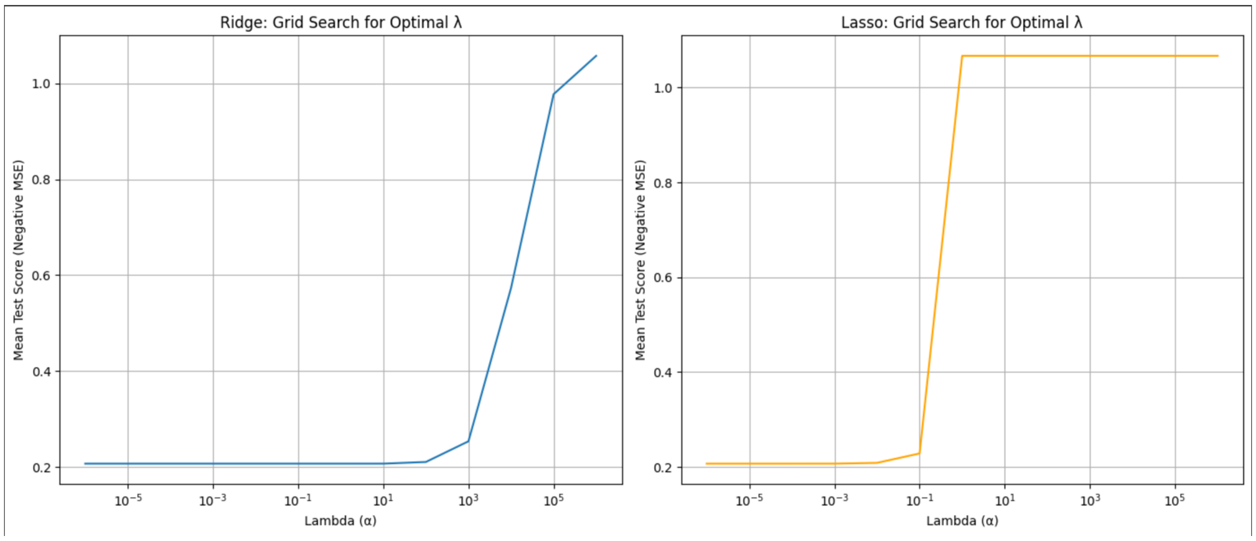


Figure 33: MSE Vs. Lambda for Ridge and Lasso with Grid Search

Figure 34 shows the predicted vs. actual values for Ridge and Lasso.

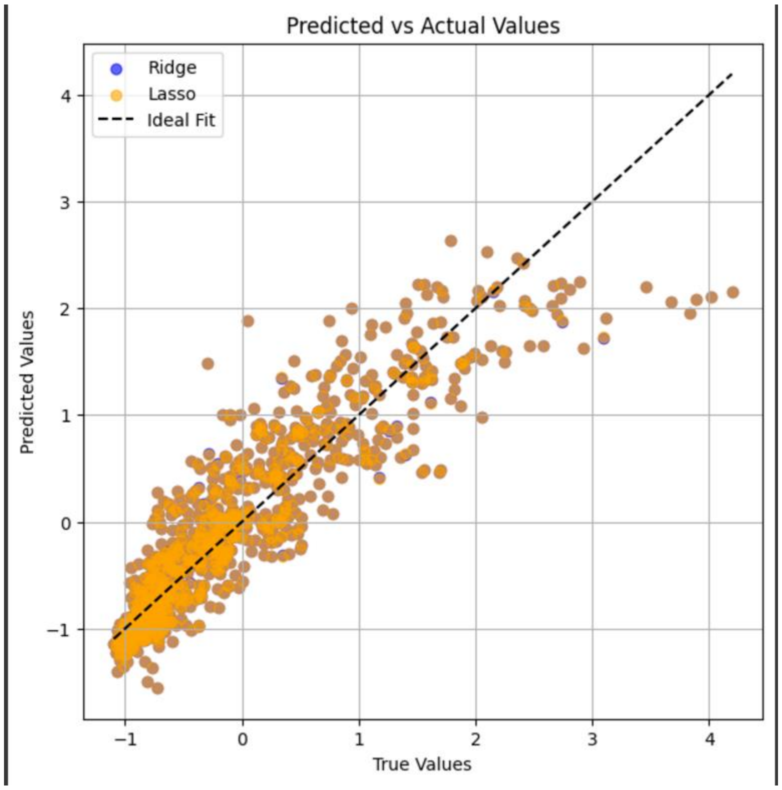


Figure 34: Predicted Vs. Actual Values for Ridge and Lasso

Figure 35 shows the residual plot for Lasso and Ridge.

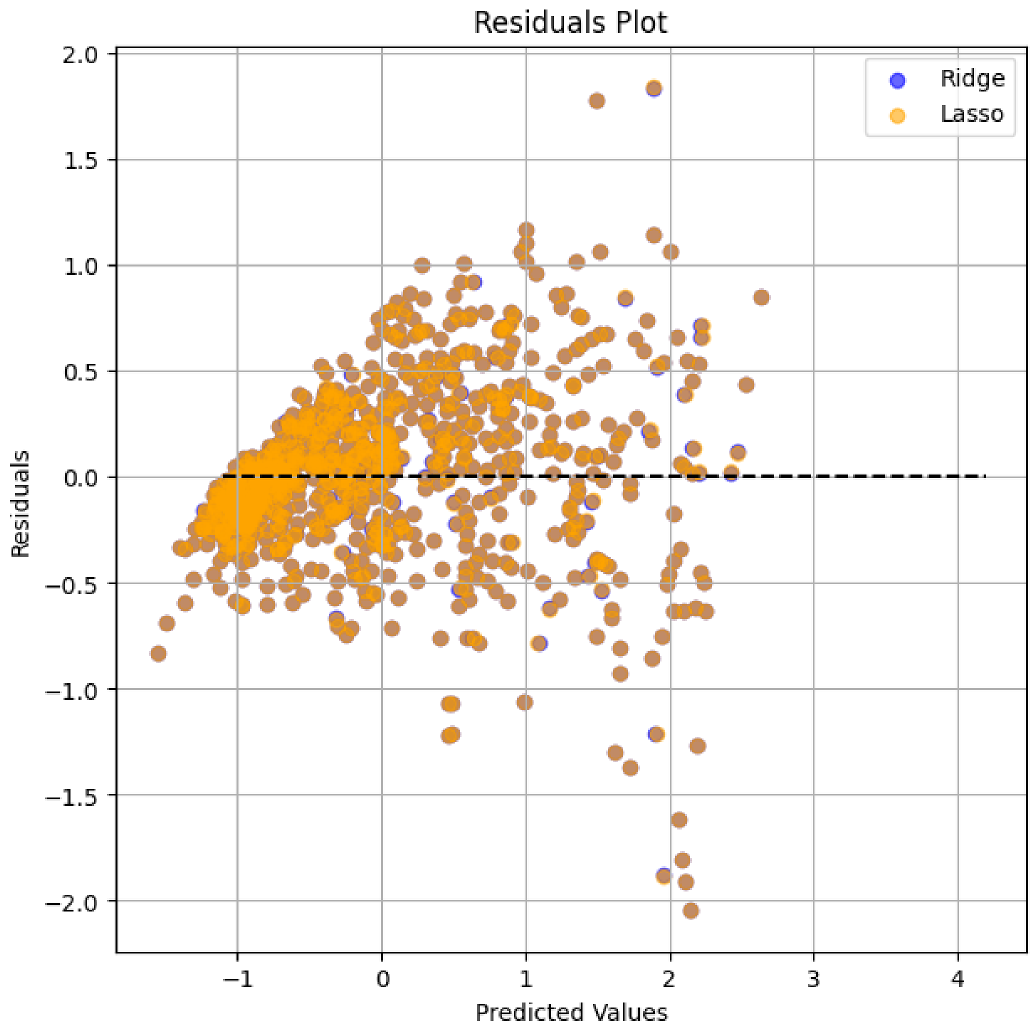


Figure 35: The Residual Plot for Lasso and Ridge

As noted, for the lasso and ridge the datapoints is close to the fit line while there are some that outlies far but that indicate that the models perform well.

Figure 36 shows the Ridge and Lasso important features.

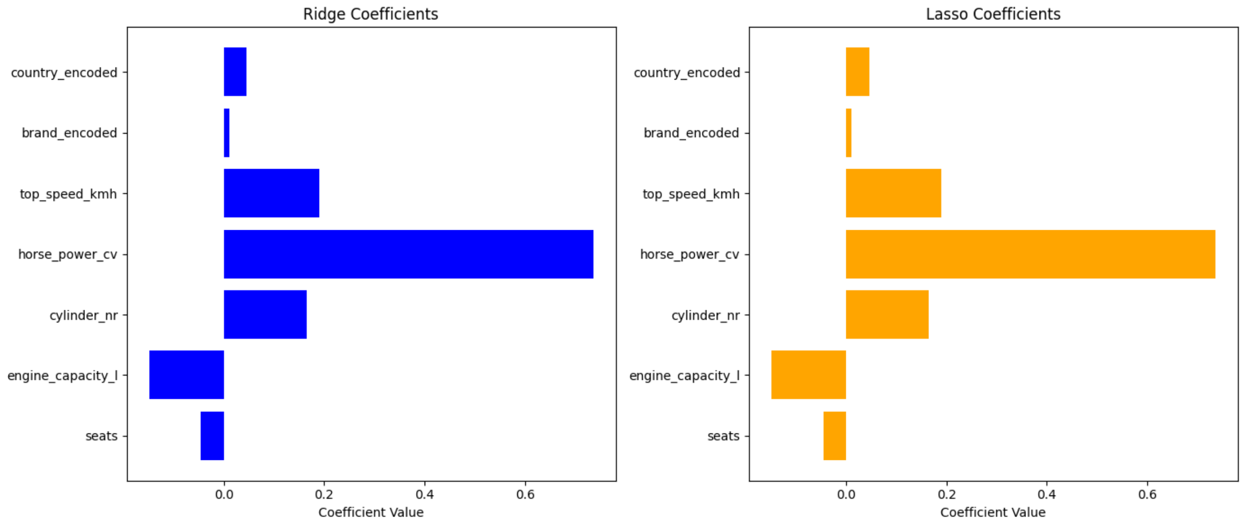


Figure 36: The Ridge and Lasso Important Features

As noted, the hourse_power_cv is more important feature. Also, the brand_encoded and engine_capacity_l are the less important features.

Figure 37 shows the Ridge and Lasso validation curve.

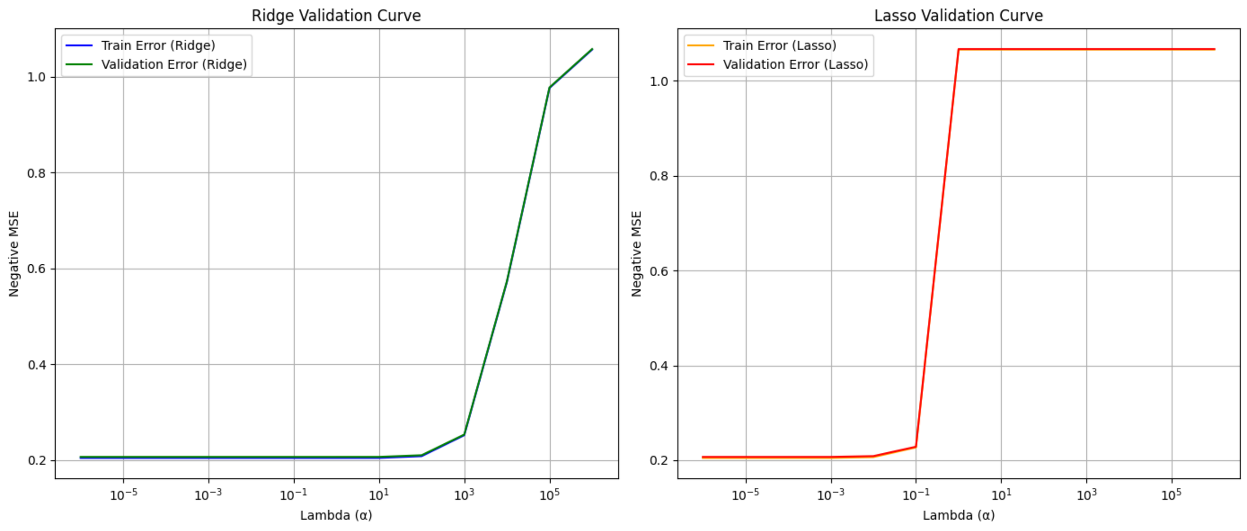


Figure 37: The Ridge and Lasso Validation Curve

As noted, the figure illustrates the impact of regularization on Ridge and Lasso regression models through validation curves. For Ridge (left), as the regularization parameter λ increases, the training error gradually decreases, while the validation error improves significantly for higher λ values, reflecting the model's ability to reduce overfitting. In contrast, for Lasso (right), both training and validation errors remain constant for small λ values but drop sharply at a critical point as λ increases, stabilizing afterward, due to Lasso's aggressive feature selection by shrinking some coefficients to zero. The curves highlight that Ridge is effective for handling multicollinearity by balancing bias and variance, while Lasso simplifies models by performing feature selection.

6.4 Nonlinear Models

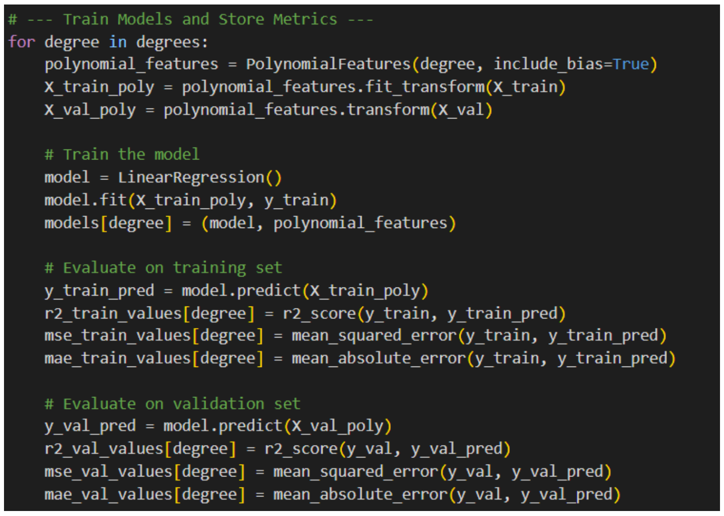
6.4.1 Polynomial (From Degree 2 To 10)

Polynomial Regression extends linear regression by modeling the relationship between the independent variable(s) and the dependent variable as a polynomial of degree n, allowing it to capture non-linear patterns in the data. By introducing polynomial terms (e.g., x^2, x^3 x^2, x^3), it can model curved relationships that linear regression cannot. While it is flexible and effective for fitting complex trends, higher-degree polynomials can lead to overfitting, making the model sensitive to noise and less generalizable. To address this, regularization techniques like Ridge or Lasso are often used. Polynomial regression is useful in scenarios such as modeling growth curves, seasonal trends, and other non-linear relationships in data.

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \epsilon$$

Figure 38: A Polynomial with Degree 2 Equation

Figure 39 shows the polynomial regression implementation.

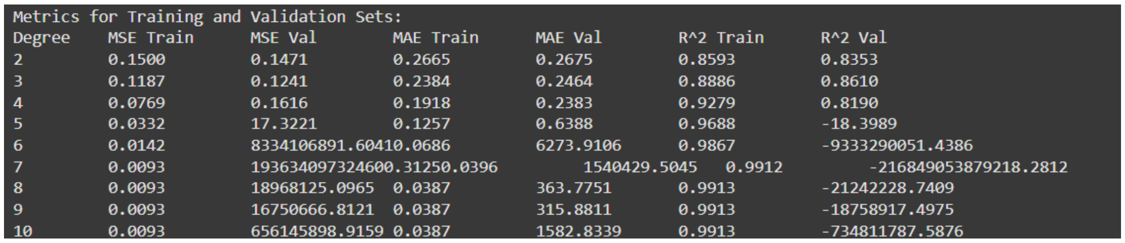


```
# --- Train Models and Store Metrics ---
for degree in degrees:
    polynomial_features = PolynomialFeatures(degree, include_bias=True)
    X_train_poly = polynomial_features.fit_transform(X_train)
    X_val_poly = polynomial_features.transform(X_val)

    # Train the model
    model = LinearRegression()
    model.fit(X_train_poly, y_train)
    models[degree] = (model, polynomial_features)

    # Evaluate on training set
    y_train_pred = model.predict(X_train_poly)
    r2_train_values[degree] = r2_score(y_train, y_train_pred)
    mse_train_values[degree] = mean_squared_error(y_train, y_train_pred)
    mae_train_values[degree] = mean_absolute_error(y_train, y_train_pred)

    # Evaluate on validation set
    y_val_pred = model.predict(X_val_poly)
    r2_val_values[degree] = r2_score(y_val, y_val_pred)
    mse_val_values[degree] = mean_squared_error(y_val, y_val_pred)
    mae_val_values[degree] = mean_absolute_error(y_val, y_val_pred)
```

Figure 39: Polynomial Regression Implementation

6.4.2 Polynomial Results

Figure 40 shows the results for (2 to 10) degrees.



```
Metrics for Training and Validation Sets:
Degree    MSE Train      MSE Val        MAE Train      MAE Val        R^2 Train      R^2 Val
2         0.1500         0.1471         0.2665         0.2675         0.8593         0.8353
3         0.1187         0.1241         0.2384         0.2464         0.8886         0.8610
4         0.0769         0.1616         0.1918         0.2383         0.9279         0.8190
5         0.0332         17.3221        0.1257         0.6388         0.9688         -18.3989
6         0.0142         8334106891.60410.0686         6273.9106      0.9867         -9333290051.4386
7         0.0093         193634097324600.31250.0396          1540429.5045   0.9912          -216849053879218.2812
8         0.0093         18968125.0965  0.0387         363.7751       0.9913         -21242228.7409
9         0.0093         16750666.8121  0.0387         315.8811       0.9913         -18758917.4975
10        0.0093         656145898.9159 0.0387         1582.8339      0.9913         -734811787.5876
```

Figure 40: The Results for (2 To 10) Degrees

As noted, the degree 3 is better than other degrees and from 4 to 10 degree the model starts with overfitting. The best model till now is the polynomial with degree 3.

Figure 41 shows the true versus predicted values for polynomial models.

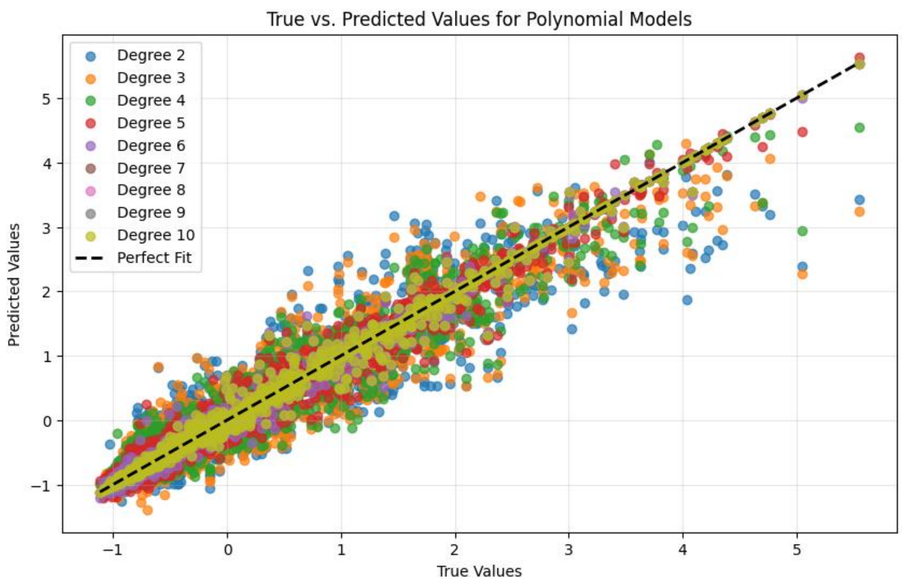


Figure 41: True Vs. Predicted Values for Polynomial Models

As noted, the lower degrees predicted values is far and understandable from the higher degrees which are memories the data than learning from it.

Figure 42 shows the residuals versus predicted values for polynomial models.

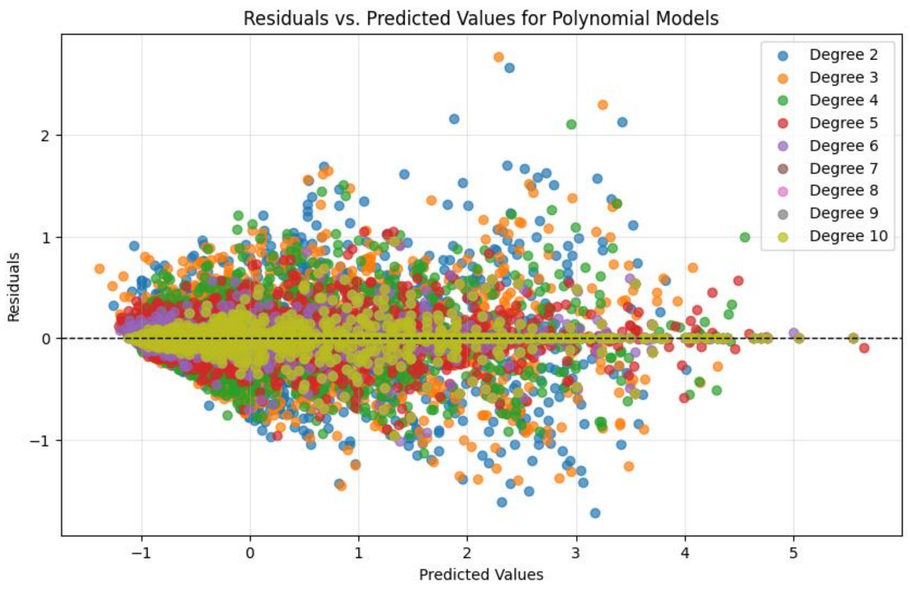


Figure 42: Residuals Vs. Predicted Values for Polynomial Models

As noted, the lower degrees residuals values are far and understandable from the higher degrees which are memories the data than learning from it.

6.4.3 Radial Basis Function (RBF)

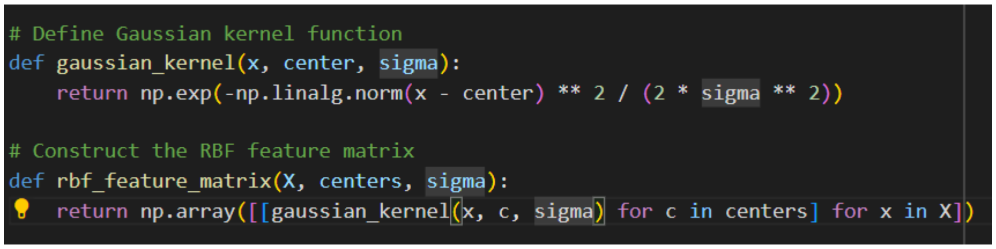
The Radial Basis Function (RBF) is a type of kernel function widely used in machine learning, particularly in Support Vector Machines (SVMs) and Radial Basis Function Networks (RBFNs). It transforms the input data into a higher-dimensional space, enabling the model to capture complex, non-linear relationships. The most commonly used RBF is the Gaussian kernel, defined as following:

$$b(x) = e^{\frac{-(x-c_k)^2}{2\sigma^2}}$$

Figure 43: RBF Equation

where γ controls the width of the kernel. The RBF measures similarity between two points based on their distance, with closer points having higher similarity. Its flexibility and effectiveness in handling non-linear data make it a popular choice, but careful tuning of parameters (e.g., γ γ) is crucial to avoid overfitting or underfitting.

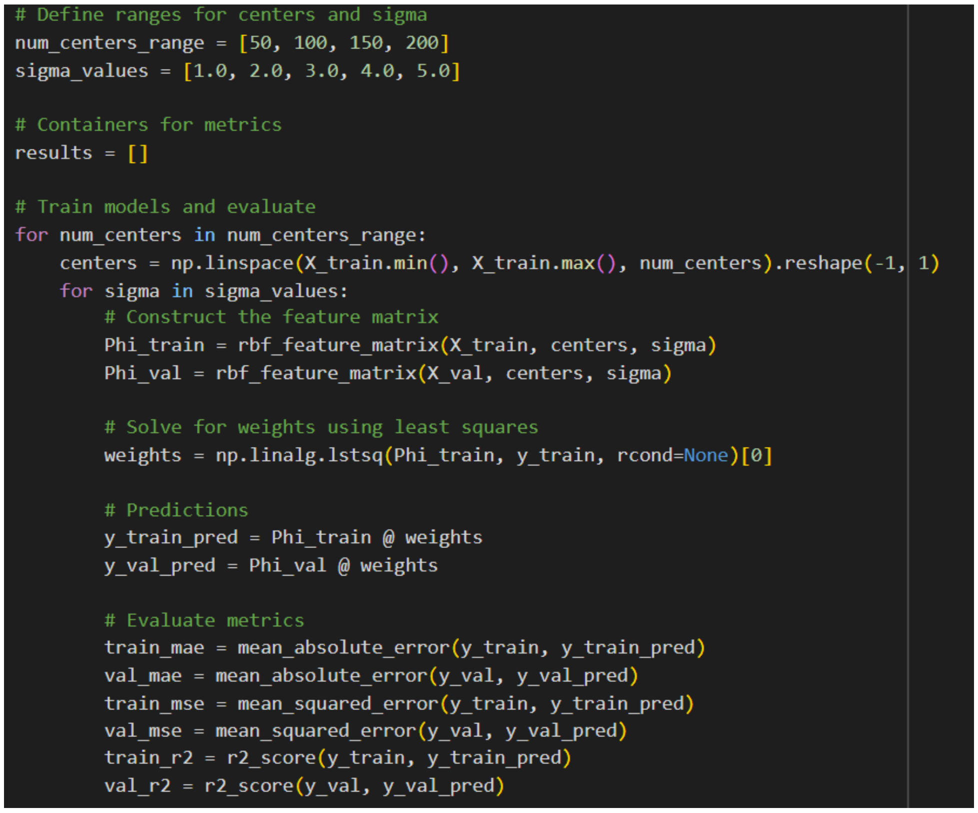
Figure 44 shows the RBF implementation.



```python
# Define Gaussian kernel function
def gaussian_kernel(x, center, sigma):
    return np.exp(-np.linalg.norm(x - center) ** 2 / (2 * sigma ** 2))

# Construct the RBF feature matrix
def rbf_feature_matrix(X, centers, sigma):
    return np.array([[gaussian_kernel(x, c, sigma) for c in centers] for x in X])
```

Figure 44: RBF Implementation

Figure 45 shows the implementation of finding the sigma and center values.



```python
# Define ranges for centers and sigma
num_centers_range = [50, 100, 150, 200]
sigma_values = [1.0, 2.0, 3.0, 4.0, 5.0]

# Containers for metrics
results = []

# Train models and evaluate
for num_centers in num_centers_range:
    centers = np.linspace(X_train.min(), X_train.max(), num_centers).reshape(-1, 1)
    for sigma in sigma_values:
        # Construct the feature matrix
        Phi_train = rbf_feature_matrix(X_train, centers, sigma)
        Phi_val = rbf_feature_matrix(X_val, centers, sigma)

        # Solve for weights using least squares
        weights = np.linalg.lstsq(Phi_train, y_train, rcond=None)[0]

        # Predictions
        y_train_pred = Phi_train @ weights
        y_val_pred = Phi_val @ weights

        # Evaluate metrics
        train_mae = mean_absolute_error(y_train, y_train_pred)
        val_mae = mean_absolute_error(y_val, y_val_pred)
        train_mse = mean_squared_error(y_train, y_train_pred)
        val_mse = mean_squared_error(y_val, y_val_pred)
        train_r2 = r2_score(y_train, y_train_pred)
        val_r2 = r2_score(y_val, y_val_pred)
```

Figure 45: Finding the Sigma and Center Values

As noted, multi-values of sigma and center are tried to find the best values.

6.4.4 RBF Results

After multi-values of sigma and center are tried: Best Model: Centers = 200, Sigma = 5.0

Validation MAE = 0.4935, Validation MSE = 0.5071, Validation R² = 0.5442

As noted, this is the worst model in our case.

Figure 46 shows the true versus predicted values for RBF model.

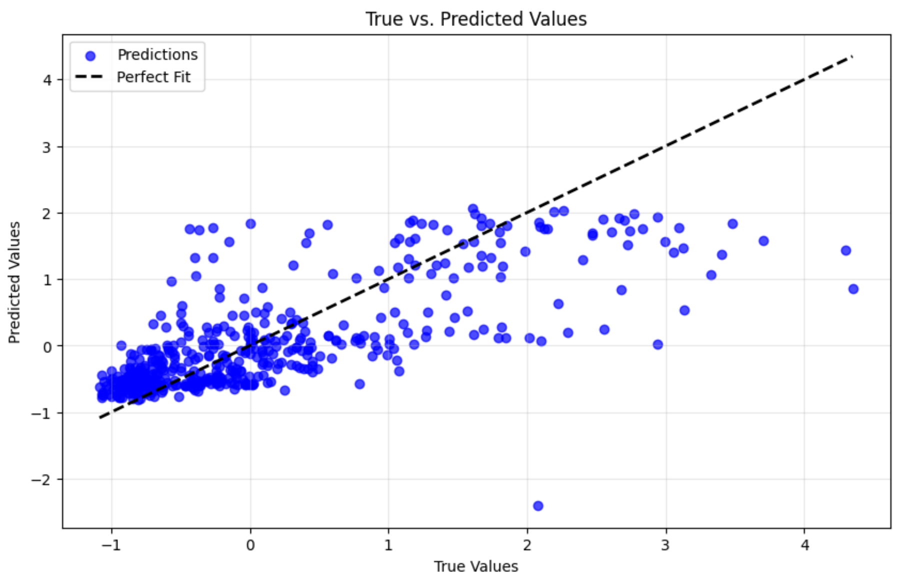


Figure 46: True Vs. Predicted Values for RBF Model

As noted, the predicted data points are far from the fit line and this indicates that the model doesn't perform well.

Figure 47 shows the residuals versus predicted values for RBF model.

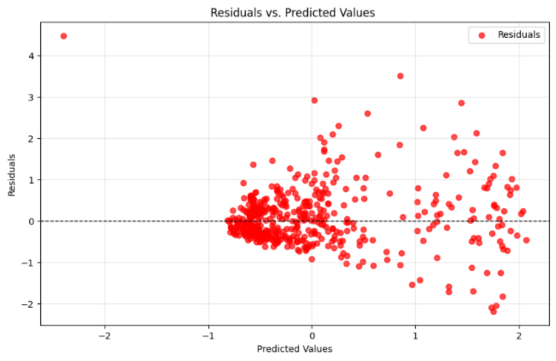
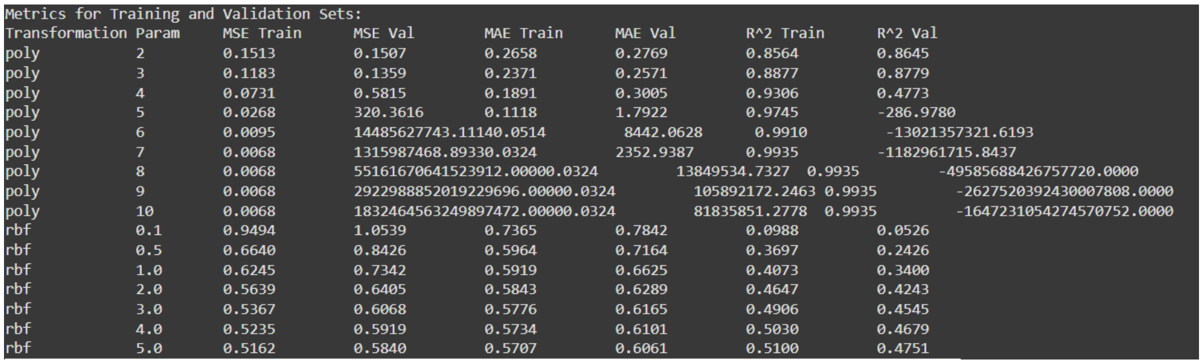


Figure 47: Residuals Vs. Predicted Values for RBF Model

As noted, the predicted data points are far from the zero line and this indicates that the model doesn’t perform well.

Figure 48 shows comparison between RBF and polynomial.



```
Metrics for Training and Validation Sets:
Transformation Param     MSE Train      MSE Val        MAE Train      MAE Val        R^2 Train      R^2 Val
poly           2         0.1513         0.1507         0.2658         0.2769         0.8564         0.8645
poly           3         0.1183         0.1359         0.2371         0.2571         0.8877         0.8779
poly           4         0.0731         0.5815         0.1891         0.3005         0.9306         0.4773
poly           5         0.0268         320.3616       0.1118         1.7922         0.9745         -286.9780
poly           6         0.0095         14485627743.11140.0514         8442.0628      0.9910         -13021357321.6193
poly           7         0.0068         1315987468.89330.0324         2352.9387      0.9935         -1182961715.8437
poly           8         0.0068         55161670641523912.00000.0324         13849534.7327  0.9935         -49585688426757720.0000
poly           9         0.0068         2922988852019229696.00000.0324         105892172.2463 0.9935         -2627520392430007808.0000
poly           10        0.0068         1832464563249897472.00000.0324         81835851.2778  0.9935         -1647231054274570752.0000
rbf            0.1       0.9494         1.0539         0.7365         0.7842         0.0988         0.0526
rbf            0.5       0.6640         0.8426         0.5964         0.7164         0.3697         0.2426
rbf            1.0       0.6245         0.7342         0.5919         0.6625         0.4073         0.3400
rbf            2.0       0.5639         0.6405         0.5843         0.6289         0.4647         0.4243
rbf            3.0       0.5367         0.6068         0.5776         0.6165         0.4906         0.4545
rbf            4.0       0.5235         0.5919         0.5734         0.6101         0.5030         0.4679
rbf            5.0       0.5162         0.5840         0.5707         0.6061         0.5100         0.4751
```

Figure 48: Comparison Between RBF and Polynomial

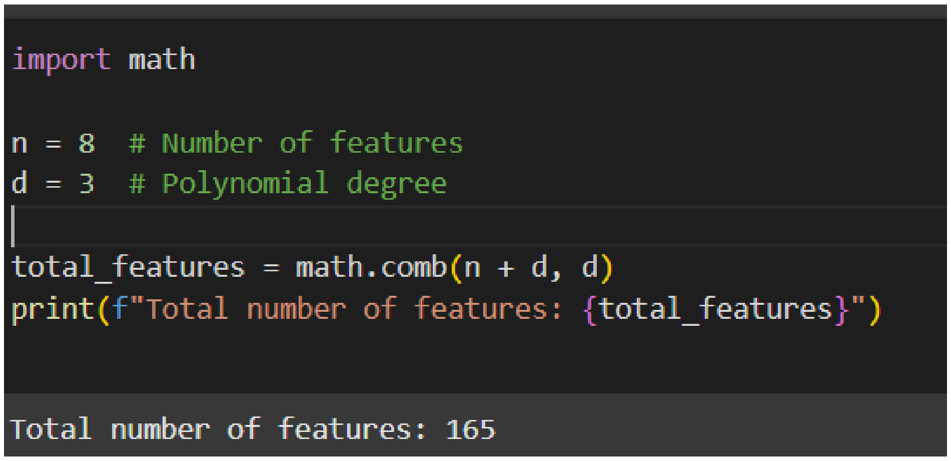
As noted, the best model is a **polynomial with degree 3. Also,** the Ridge will be used with the polynomial with degree 3 to avoid the overfitting.

7 The Best Model Tuning and Evaluation

7.1 Feature Selection and Forward Selection

Feature selection is a crucial step in the data preprocessing pipeline that aims to identify the most relevant features for model building, improving model performance and interpretability while reducing computational cost. One popular method for feature selection is Forward Selection, a stepwise selection process that begins with an empty model and sequentially adds features. At each step, the feature that provides the most significant improvement in a chosen evaluation metric (e.g., accuracy, R², or AIC) is added to the model. This iterative process continues until adding more features no longer enhances the model's performance. Forward selection helps in identifying a subset of features that contribute the most to the predictive power of the model, ensuring that irrelevant or redundant features are excluded.

For the best model which is polynomial with degree 3, the total number of features were calculated. Figure 49 shows the implementation of this calculation.

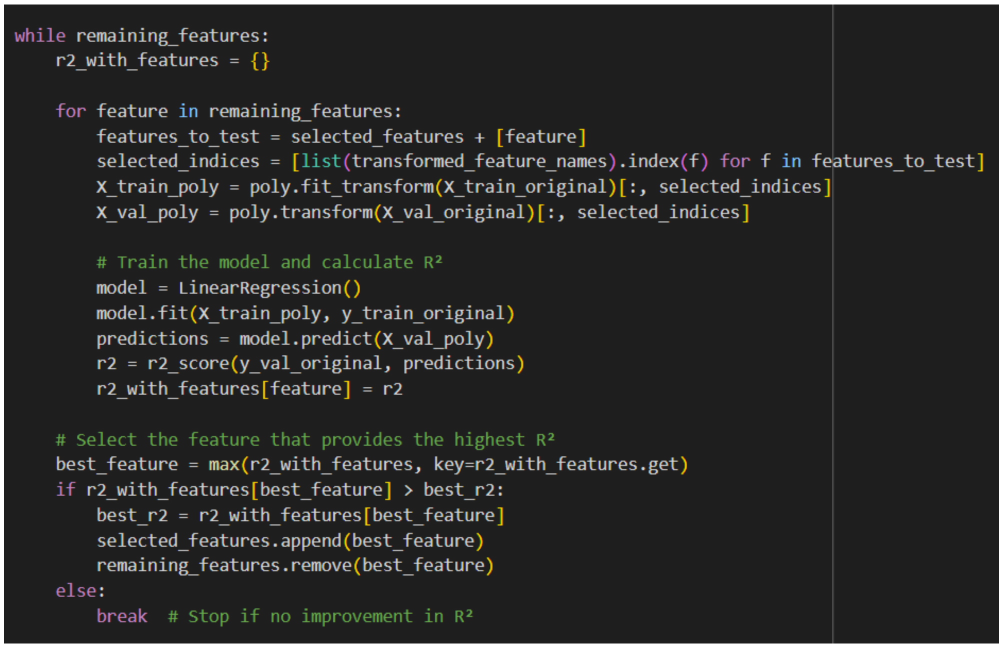


```
import math

n = 8  # Number of features
d = 3  # Polynomial degree

total_features = math.comb(n + d, d)
print(f"Total number of features: {total_features}")
```

```
Total number of features: 165
```

Figure 49: Total Number of Features for the Best Model Calculation

Figure 50 shows the feature selection and forward selection implementation before tuning.



```
while remaining_features:
    r2_with_features = {}

    for feature in remaining_features:
        features_to_test = selected_features + [feature]
        selected_indices = [list(transformed_feature_names).index(f) for f in features_to_test]
        X_train_poly = poly.fit_transform(X_train_original)[:, selected_indices]
        X_val_poly = poly.transform(X_val_original)[:, selected_indices]

        # Train the model and calculate R²
        model = LinearRegression()
        model.fit(X_train_poly, y_train_original)
        predictions = model.predict(X_val_poly)
        r2 = r2_score(y_val_original, predictions)
        r2_with_features[feature] = r2

    # Select the feature that provides the highest R²
    best_feature = max(r2_with_features, key=r2_with_features.get)
    if r2_with_features[best_feature] > best_r2:
        best_r2 = r2_with_features[best_feature]
        selected_features.append(best_feature)
        remaining_features.remove(best_feature)
    else:
        break  # Stop if no improvement in R²
```

Figure 50: Feature Selection and Forward Selection Implementation Before Tuning

As a result, the number of selected features was 63 features before tuning. These features are the features that makes improvements on R^2 value.

Best R^2 Score on Validation Set (Before Tuning): 0.8777891154371279

7.2 Applying Regularization Techniques with Tuning

Performing Ridge hyperparameter tuning:

Best Ridge Hyperparameters: {'alpha': 10}

Validation Metrics After Ridge Tuning:

MSE: 0.1096, **MAE:** 0.2416, R^2: 0.8773

Performing LASSO Regression with:

LASSO Optimal Alpha: 0.002873262666349012

LASSO Validation R^2: 0.8675, **MSE:** 0.1183, **MAE:** 0.2481

As a result of the comparison between Lasso and Ridge in tuning, the Ridge was better than Lasso. Its R^2 was greater than that from Lasso. So, the tuning using Ridge with polynomial with degree 3 performs well and better than using Lasso with polynomial with degree 3.

7.3 Evaluation with Test Set

Figure 51 shows the true versus predicted values for best model which is the polynomial with degree 3 with tuning using Ridge regularization.

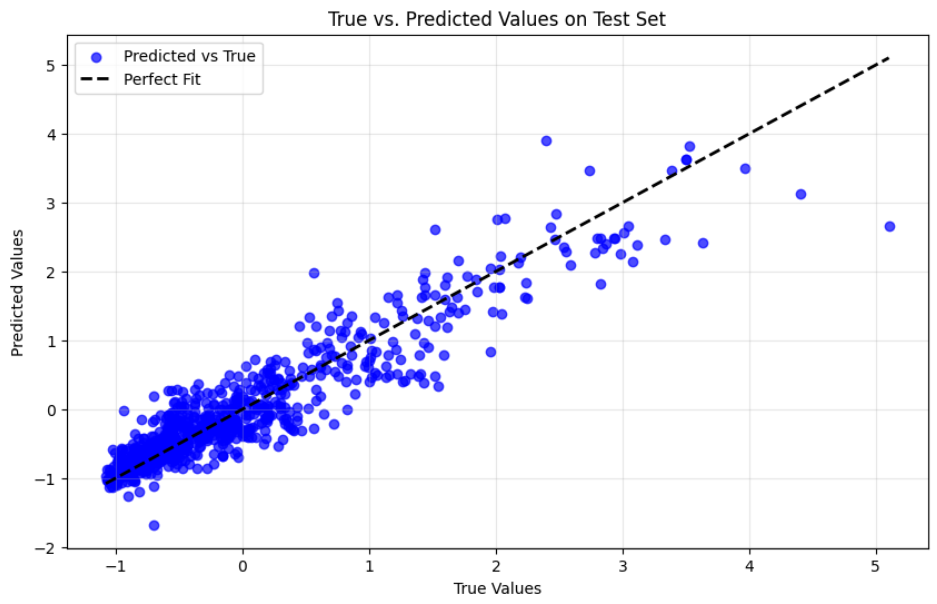


Figure 51: True Vs. Predicted Values for the Best Model

As a result, the evaluation of the model on the test set demonstrates strong performance, as reflected by the metrics. The Mean Squared Error (MSE) of 0.1171 indicates a low average squared difference between predicted and actual values, while the Mean Absolute Error (MAE) of 0.2434 shows the average absolute deviation of predictions from the true values is small. Additionally, the R² score of 0.8704 suggests that approximately 87% of the variance in the target variable is explained by the model, indicating a high level of accuracy and a strong fit to the data.

8 Optional

The hourse_power feature (column) was chosen to be the target value because it's the most feature correlated with other features.

Figure 52 shows true versus predicted values for best model which is the polynomial with degree 3 with tuning using Ridge regularization using new target (hourse_power).

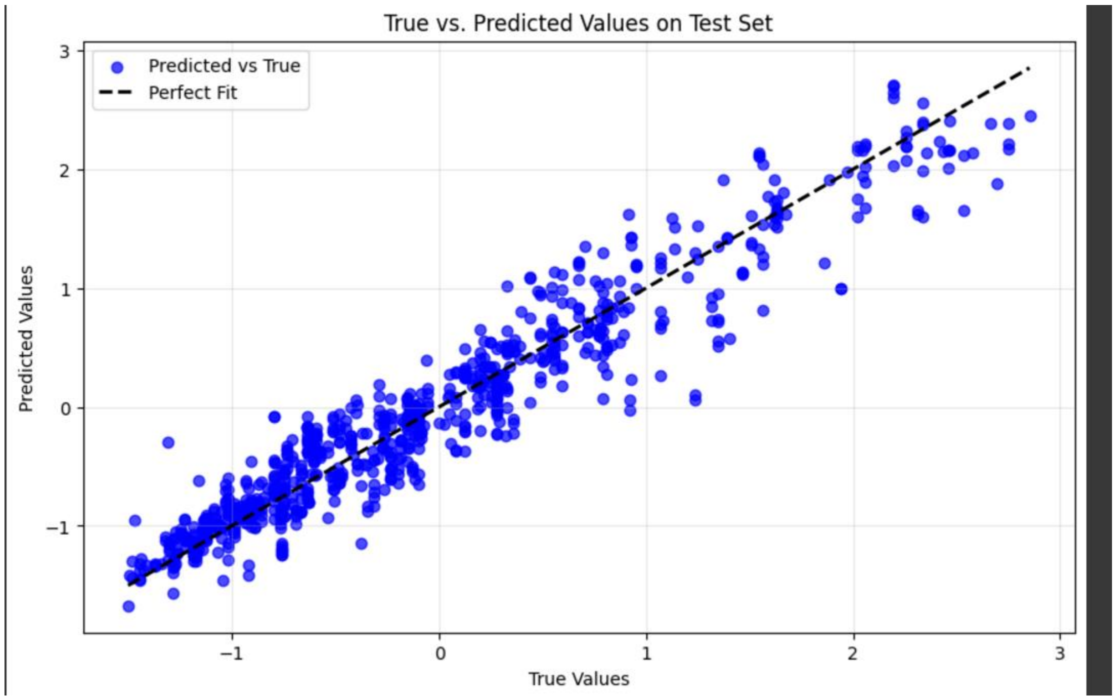


Figure 52: True Vs. Predicted Values for the Best Model with New Target

As a result:

Best R² Score on Validation Set (Before Tuning): 0.9301701764931369

The number of selected features: 64

Performing Ridge hyperparameter tuning:

Best Ridge Hyperparameters: {'alpha': 1}

Validation Metrics After Ridge Tuning:

MSE: 0.0701, **MAE:** 0.1927, **R^2:** 0.9301

Evaluating on test set:

Test Set Metrics:

MSE: 0.0757, **MAE:** 0.2061, **R^2:** 0.9180

9 Limitations on the Best Model

Using a polynomial of degree 3 introduces flexibility to model non-linear relationships but comes with significant limitations. It can lead to overfitting, as higher-degree polynomials capture noise in the training data, reducing generalization to unseen data. The number of features grows exponentially, increasing computational complexity and memory usage, while the resulting features often exhibit multicollinearity, making model coefficients unstable. Additionally, such models are harder to interpret, which can be problematic in scenarios requiring explainability. Polynomial features are also sensitive to scaling, magnifying input values and causing numerical instability if not standardized. These limitations make it crucial to balance complexity, interpretability, and computational efficiency when using polynomial transformations.

10 Conclusion

In conclusion, this report presents the development and evaluation of regression models for predicting car prices using a dataset from Yalla Motors. The project successfully applied both linear and nonlinear regression techniques, leveraging data preprocessing, feature selection, and regularization to improve model accuracy and generalization. Key methodologies included data cleaning, exploratory data analysis, implementation of linear models like Linear Regression, LASSO, and Ridge, as well as nonlinear approaches such as Polynomial Regression and Radial Basis Function (RBF). Among these, the best-performing model was identified as the Polynomial Regression model with a degree of 3, tuned using Ridge regularization. This model demonstrated strong performance on the test set, achieving a Mean Squared Error (MSE) of 0.1171, a Mean Absolute Error (MAE) of 0.2434, and an R^2 value of 0.8704, indicating low error rates and high predictive accuracy. The report provides detailed insights, supported by comprehensive visualizations, highlighting the effectiveness of the chosen approaches and the superior performance of the selected model.