Network Operating System

Paweł Pozorski, Zuzanna Sieńko

December 23, 2024

Abstract

In the following document, we share our route for the development of the project infrastructure.

Contents

1 Deploying model using Snowpark, Python, and VS Code 2
1.1 Creating trial Snowflake Account 2

2 Loading Data 3

3 EDA 4

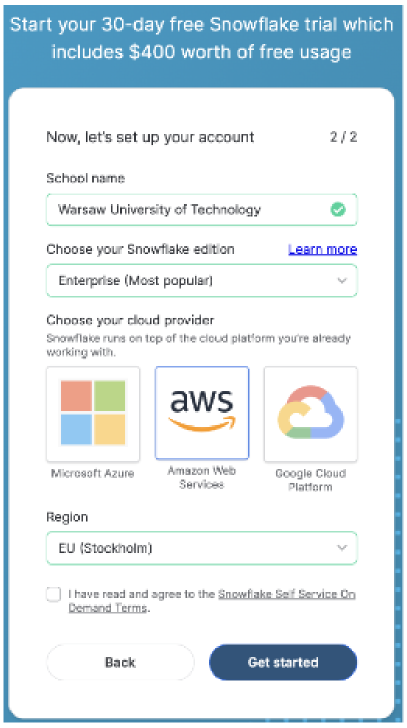
4 Feature Engineering 7
4.1 Summary of Training Data After Feature Engineering 7

5 Model 8
5.1 Model selection and optimization 8
5.2 Performance 8
5.3 Model Registry and deployment 10

1 Deploying model using Snowpark, Python, and VS Code

1.1 Creating trial Snowflake Account

For the purpose of the project, we'll use a free, trial Snowflake account that is available for public use for up to 30 days or 400 USD of credits. As our cloud provider, we'll choose AWS as it was introduced on the labs.


Start your 30-day free Snowflake trial which includes $400 worth of free usage

Now, let's set up your account 2 / 2

School name: Warsaw University of Technology

Choose your Snowflake edition: Enterprise (Most popular)

Choose your cloud provider
Snowflake runs on top of the cloud platform you're already working with.

Region: EU (Stockholm)

I have read and agree to the Snowflake Self Service On Demand Terms.

Figure 1: Registration form.

Using Snowsight, we create a basic setup for Snowflake:

```
CREATE WAREHOUSE IF NOT EXISTS ml_warehouse
  WITH WAREHOUSE_SIZE = 'XSMALL'
  AUTO_SUSPEND = 60  -- Automatically suspends after 60 seconds of inactivity
  AUTO_RESUME = TRUE;

CREATE ROLE IF NOT EXISTS DATA_SCIENTIST;
GRANT ROLE DATA_SCIENTIST TO ROLE SYSADMIN;
GRANT ALL PRIVILEGES ON WAREHOUSE ml_warehouse TO ROLE DATA_SCIENTIST;

CREATE DATABASE ml_database;
CREATE SCHEMA datasets;

GRANT OWNERSHIP ON DATABASE ml_database TO ROLE DATA_SCIENTIST;
GRANT OWNERSHIP ON SCHEMA datasets TO ROLE DATA_SCIENTIST;
```

Now let's try to access it from Snowpark. For installation, we've followed these instructions. After providing required connection details described here, we create simple code to validate if it works correctly:

```
from snowflake.snowpark import Session
from snowflake.ml.utils.connection_params import SnowflakeLoginOptions

options = SnowflakeLoginOptions(login_file=".snowsql/config", connection_name="ml")

sp_session = Session.builder.configs(options).create()

sp_session.sql("DESCRIBE DATABASE ml_database;").collect()
```

2 Loading Data

Here we've downloaded our dataset of choice (source), unzipped it, and made some adjustments to its structure. The next steps include creating a Snowflake stage, uploading the file, and verifying its presence.

Name	**Size (bytes)**	**MD5**	**Last Modified**
auto_mpg_stage/auto-mpg.data	30288	b26b22a6...	Sat, 4 Jan 2025 15:01:04 GMT

Table 1: Uploaded File Details in Snowflake Stage

The final table structure used for training is as follows:

Column Name	**Data Type**	**Nullable**
MPG	DoubleType	True
CYLINDERS	LongType	True
DISPLACEMENT	DoubleType	True
HORSEPOWER	DoubleType	True
WEIGHT	DoubleType	True
ACCELERATION	DoubleType	True
MODEL_YEAR	LongType	True
ORIGIN	LongType	True

Table 2: Final Table Structure Used for Training

3 EDA

Before we begin, this part aims to show power of cloud resources not to be a compleate data science project. Therefore, below we show some basic calculations performed in snowpark. Each havy-calculation part is performed natively on cloud, so this code can be used on very large datasets as well.

Let's take peak at our data:

COLUMN	**ROW 1**	**ROW 2**	**ROW 3**
ID	1	2	3
MPG	18.0	15.0	18.0
CYLINDERS	8	8	8
DISPLACEMENT	307.0	350.0	318.0
HORSEPOWER	130.0	165.0	150.0
WEIGHT	3504.0	3693.0	3436.0
ACCELERATION	12.0	11.5	11.0
MODEL_YEAR	70	70	70
ORIGIN	1	1	1
CAR_NAME	chevrolet,chevelle,malibu	buick,skylark,320	plymouth,satellite

Table 3: Summary of Car Data

We've got 398 rows. Car name will be irrelevant for our target - MPG, so we drop it. Let's see their statistics:

COLUMN	**count**	**mean**	**stddev**	**min**	**max**
MPG	398.0	23.514572864321607	7.81598431256579	9.0	46.6
CYLINDERS	398.0	5.454774	1.7010041152213595	3.0	8.0
DISPLACEMENT	398.0	193.42587939698493	104.26983817119591	68.0	455.0
HORSEPOWER	392.0	104.46938775510205	38.49115993282849	46.0	230.0
WEIGHT	398.0	2970.424623115578	846.8417741973268	1613.0	5140.0
ACCELERATION	398.0	15.568090452261307	2.7576889298126743	8.0	24.8
MODEL_YEAR	398.0	76.01005	3.6976266712582	70.0	82.0
ORIGIN	398.0	1.572864	0.8020548609665052	1.0	3.0

Table 4: Summary Statistics of Selected Features

Which columns are categorical?

COL_NAME	**NO_DISTINCT**	**VAL_DISTINCT**
MPG	129	18, 15, 16, 17, 14
CYLINDERS	5	8, 4, 6, 3, 5
DISPLACEMENT	82	307, 350, 304, 302, 429
HORSEPOWER	93	130, 165, 150, 198, 220
WEIGHT	351	3504, 3693, 3436, 3433, 3449
ACCELERATION	95	12, 11.5, 11, 10.5, 10
MODEL_YEAR	13	70, 71, 72, 73, 74
ORIGIN	3	1, 3, 2
CAR_NAME	305	chevrolet,chevelle,malibu, buick,skylark

Table 5: Distinct Values and Categories of Features

Let's visualize our dataset:
Below is the correlation matrix visualizing relationships between features:

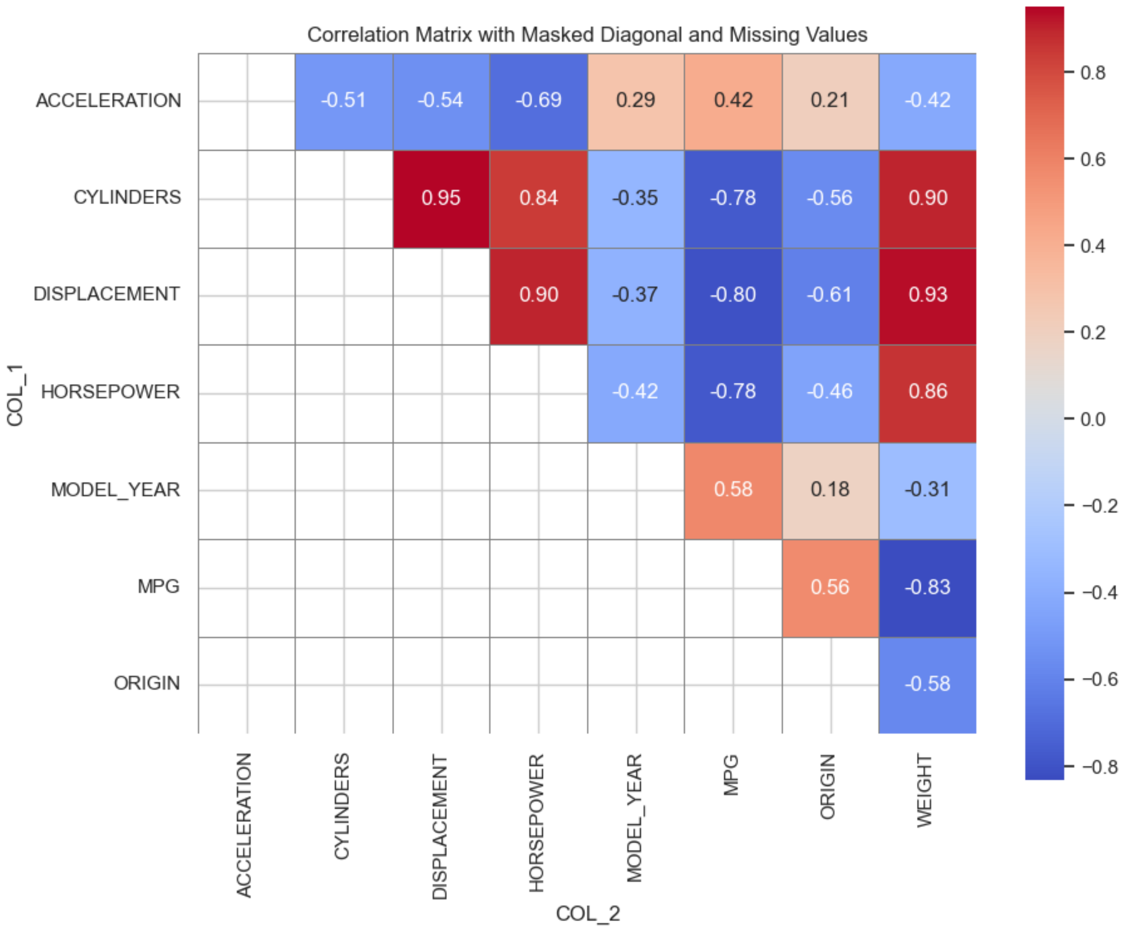


Figure 2: Correlation Matrix

Below are the histograms showing the distribution of features:

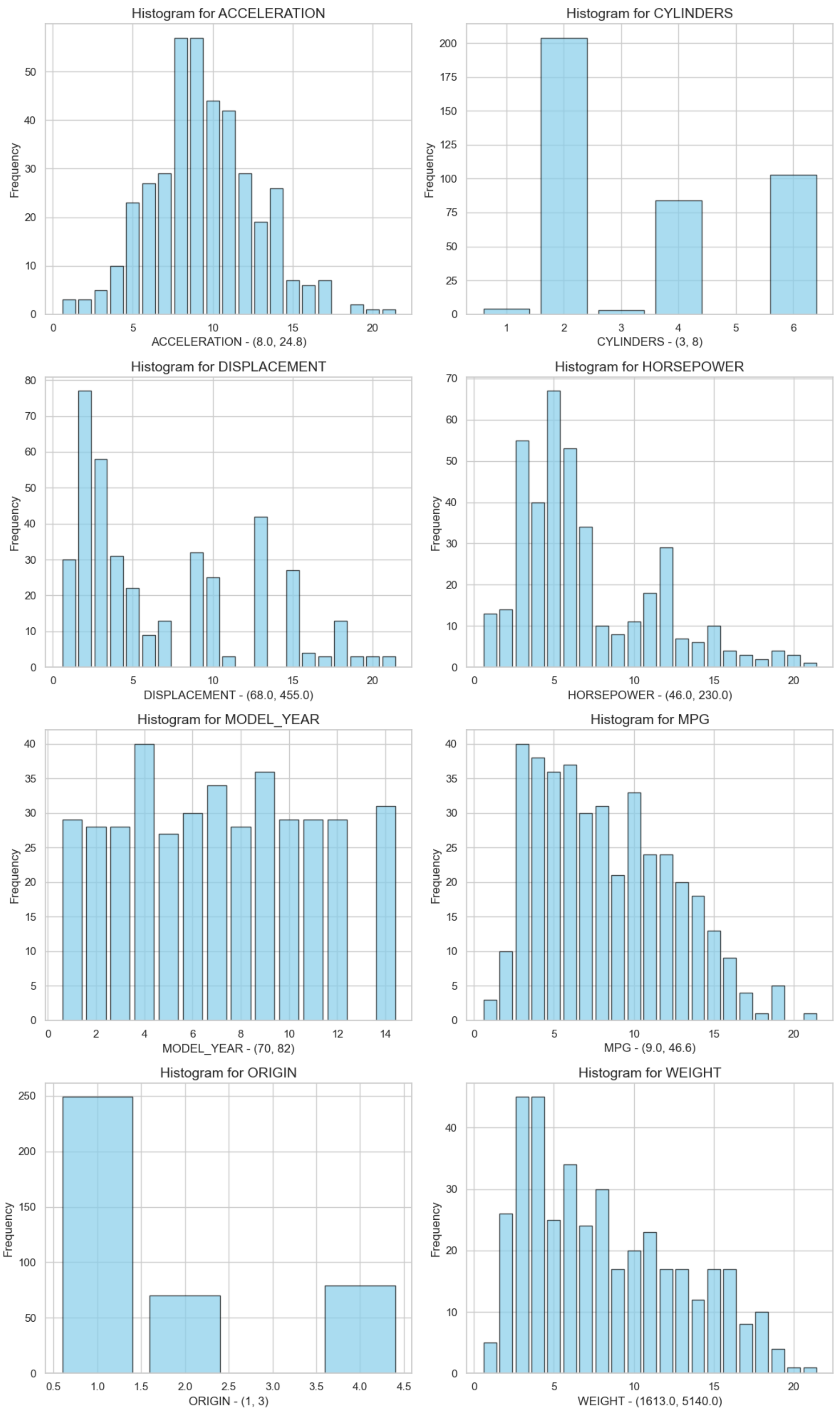


Figure 3: Histograms of Features

4 Feature Engineering

Feature engineering played a pivotal role in transforming raw data into a format suitable for effective model training and deployment. The process was initiated by splitting the dataset into two subsets: an 80% training set and a 20% testing set. The splits were stored as separate Snowflake tables, named `_TRAIN` and `_TEST`, enabling easy and organized access for subsequent processing steps.

One of the key challenges addressed during feature engineering was the handling of missing values in numeric columns, such as `DISPLACEMENT`, `HORSEPOWER`, `WEIGHT`, and `ACCELERATION`. Missing values were imputed using the mean of each column calculated from the training data.

Following the imputation, the numeric columns underwent standardization which involved calculating the mean and standard deviation for each numeric column in the training set. This normalization step ensured that all numeric features were on a comparable scale, reducing the risk of any single feature disproportionately influencing the model. The same standardization parameters were applied to the testing set to maintain consistency.

In addition to numeric transformations, the process also accounted for non-numeric or categorical features. After standardizing the numeric data, these columns were merged back with the rest of the dataset using a unique identifier (`ID`) to ensure no data was lost or misaligned.

To optimize the computational efficiency of these operations, the Snowflake warehouse size was temporarily increased. By scaling the warehouse to a larger configuration, resource-intensive tasks such as imputations, standardizations, and dataset merging were executed swiftly, minimizing processing time without compromising accuracy. The final processed datasets were stored in Snowflake as `_TRAIN_PROCESSED` and `_TEST_PROCESSED` tables, marking the completion of the feature engineering pipeline.

4.1 Summary of Training Data After Feature Engineering

COLUMN	**COUNT**	**MEAN**	**STDDEV**	**MIN**	**MAX**
ID	325.0	192.670769	114.77145530139451	1.0	398.0
DISPLACEMENT	325.0	-3.416070845000482e-17	1.0015420209622192	-1.2093191989845817	2.468712540370696
HORSEPOWER	325.0	-6.593016730850929e-17	1.0015420209622192	-1.5402008231827011	3.161288340018991
WEIGHT	325.0	4.116365368225581e-17	1.0015420209622192	-1.65722792748579	2.5678160937830627
ACCELERATION	325.0	3.4502315534504865e-16	1.001542020962218	-2.652613525208721	3.2712565094101262
MPG	325.0	23.124615384615385	7.743667878513744	9.0	46.6
CYLINDERS	325.0	5.516923	1.7259429306903518	3.0	8.0
MODEL_YEAR	325.0	75.790769	3.687555152129931	70.0	82.0
ORIGIN	325.0	1.550769	0.7825496789341876	1.0	3.0

Table 6: Summary of Training Data After Feature Engineering

5 Model

5.1 Model selection and optimization

In this study, we employed **XGBRegressor**, a robust regression model known for its ability to handle diverse data distributions effectively. To optimize the model, we utilized **GridSearchCV**, which explored various combinations of hyperparameters to determine the most effective configuration. After evaluation, the best model was selected based on the following hyperparameters:

- Learning Rate: 0.1
- Number of Estimators: 500

The primary selection criterion for the optimal model was the minimization of the **negative mean absolute percentage error (MAPE)**, a metric that quantifies the accuracy of the model's predictions. To enhance computational efficiency and accelerate the optimization process, the computational resources in Snowflake's warehouse were dynamically adjusted during the grid search.

5.2 Performance

The optimized model achieved a MAPE of 0.0848, demonstrating strong predictive accuracy in estimating the target variable, `MPG`. This indicates that the model is well-calibrated and provides reliable predictions for the given dataset.

The following plot visualizes the effect of varying the learning rate and number of estimators on the MAPE. This graphical representation illustrates how the combination of these hyperparameters influences the model's performance, aiding in the identification of the optimal configuration.

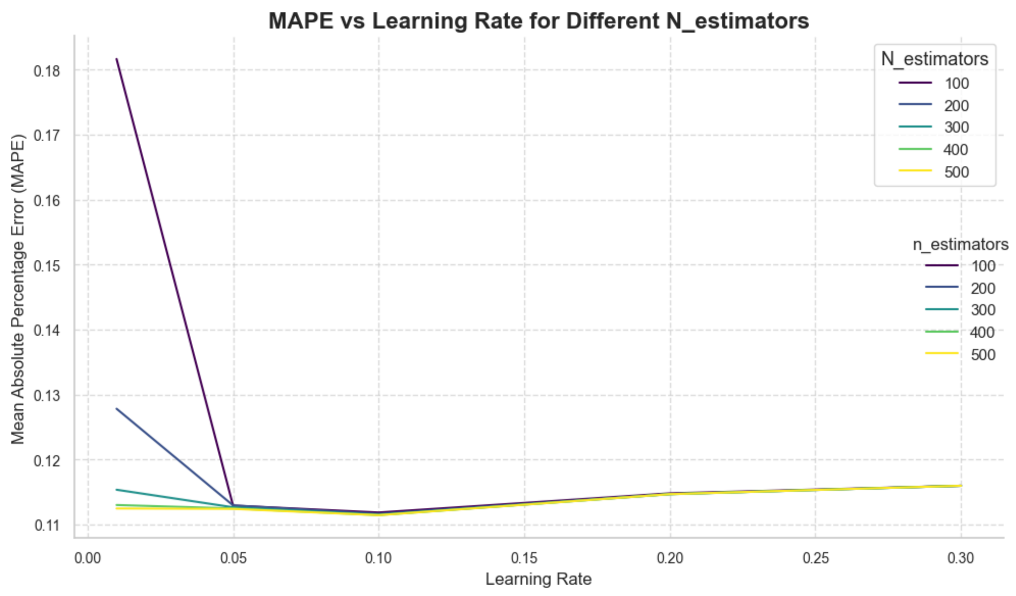


Figure 4: MAPE vs Learning Rate for Varying Numbers of Estimators

For a more in-depth comparison, the table below presents a sample of the actual MPG values alongside the corresponding predicted MPG values from the model:

MPG	PREDICTED_MPG
22.0	21.01
14.0	13.05
15.0	14.78
13.0	13.28
18.0	17.49

Table 7: Actual vs Predicted MPG

The following scatter plot presents a comparison of the actual vs predicted MPG values. A red identity line is included to represent perfect predictions, where the predicted values exactly match the actual ones. The closer the points are to this line, the more accurate the model's predictions. As shown, there is a satisfactory alignment between the actual and predicted values, confirming the model's reliability.

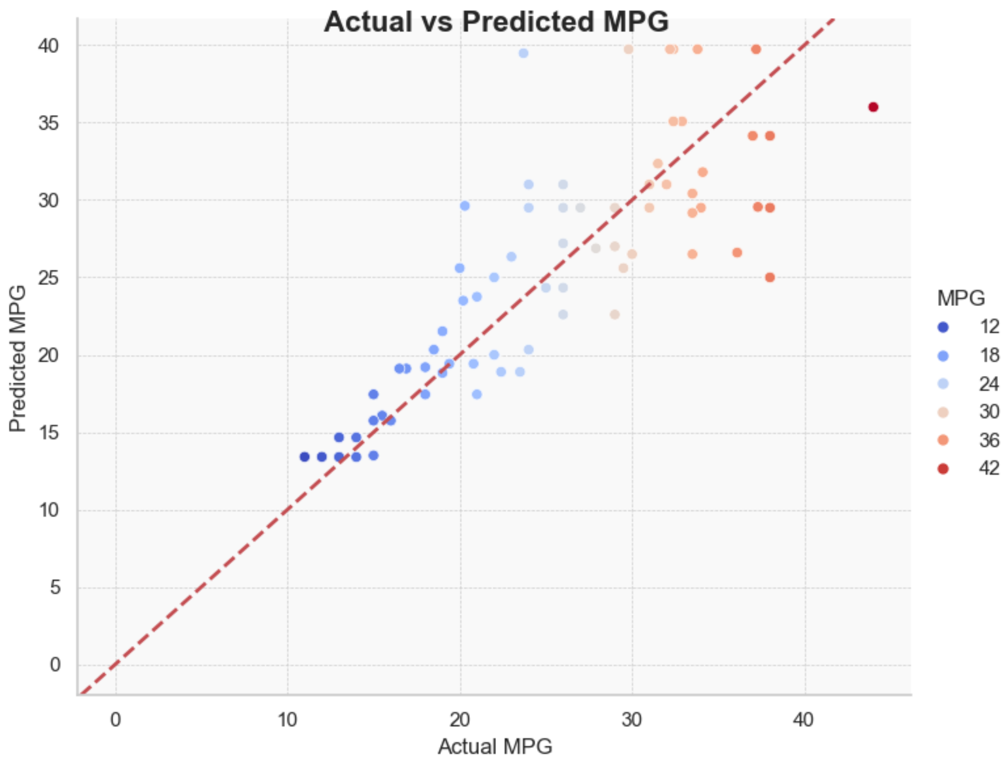


Figure 5: Model Predictions

To further enhance the interpretability of the model, we utilized SHAP (Shapley Additive Explanations) value plots, which highlight the importance of each feature in contributing to individual predictions. The analysis revealed that the most impactful features in the model's decision-making process were MODEL_YEAR and MPG. This insight provides valuable information about the underlying factors that influence the model's predictions, contributing to a better understanding of its behavior.

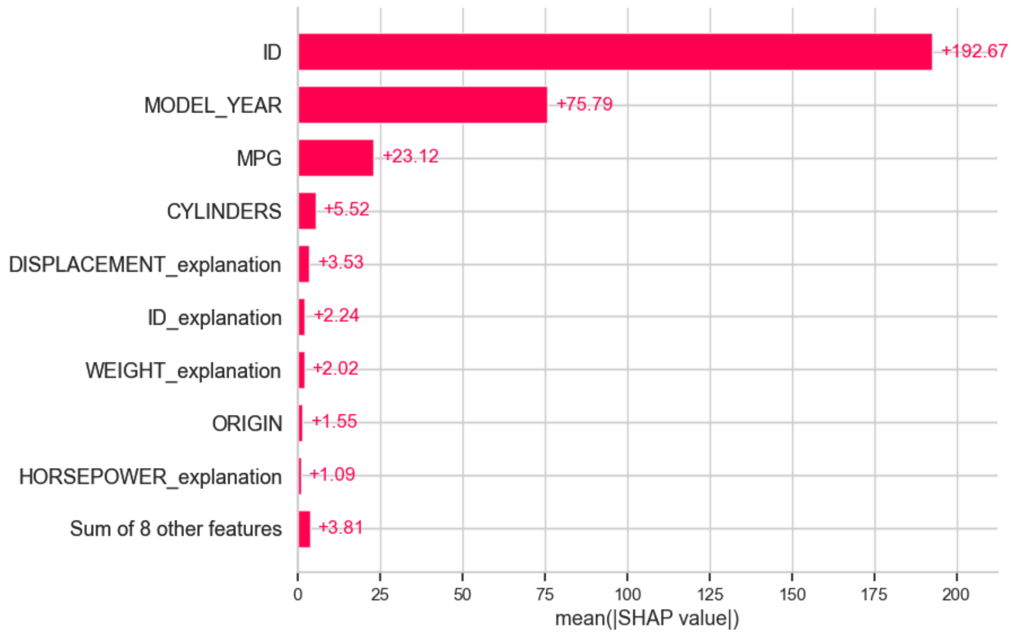


Figure 6: SHAP Value Plot Highlighting Feature Importance

5.3 Model Registry and deployment

To manage and deploy the trained XGBRegressor model, we utilized the Snowflake model registry. After optimizing the model's parameters, it was logged into the registry with version V0. The first 100 rows of the training data (excluding the target variable `MPG`) were used to define the feature schema for the model. The model was registered along with the negative mean absolute percentage error as the evaluation metric.

To avoid conflicts, any existing model with the same name (`MPG_PREDICTION`) was deleted before logging the new version.

Model explainability was enabled through SHAP values, which were generated using the model's `explain` function. This provided insights into feature importance, particularly highlighting `MODEL_YEAR` and `MPG`.