Prediction of Fuel Efficiency using Machine Learning

Name: Krishnanunni.R.Nair **Reg no:** 11902442

Roll no: RKM015A20 **Section**: KM015

Declaration

I solemnly declare that the project report is based on my own work carried out during the course of the study under the supervision of Dr Sagar Pandey. I assert the statements made and conclusions drawn are an outcome of my research work. I further certify that

I. The work contained in the report is original and has been done by me under the general supervision of my supervisor.

II. The work has not been submitted to any other Institution for any other degree/diploma/certificate in this university or any other University of India or abroad.

III. I have followed the guidelines provided by the university in writing the report.

IV. Whenever I have used materials (data, theoretical analysis, and text) from other sources, I have given due credit to them in the text of the report and giving their details in the references.

Name: Krishnanunni.R.Nair	**Reg no:** 11902442
Roll no: RKM015A20	**Section**: KM015

Acknowledgement

This research project would not have been possible without the support of many people. The author wishes to express his gratitude to his supervisor, Dr. Sagar Pandey who was abundantly helpful and offered invaluable assistance, support and guidance. The author would also like to convey thanks to the Department of Computer Science and Engineering for the opportunity provide. Endless gratitude to the faculty of Lovely Professional University for taking care of the needs required to complete the research project. The author wishes to express his love and gratitude to his peers for their understanding & help through the duration of his studies.

Index

Abstract

The automotive industry is extremely competitive. With increasing fuel prices and picky consumers, automobile makers are constantly optimizing their processes to increase fuel efficiency. But what if one could have a reliable estimator for a car's mpg given some known specifications about the vehicle? Then, one could beat a competitor to market by both having a more desirable vehicle that is also more efficient, reducing wasted R&D costs and gaining large chunks of the market. Using machine learning on the UCI Machine Learning Repository: Auto MPG dataset, which machine learning methods are most successful, how data collection affects the prediction and which features are most influential for fuel consumption are evaluated. The multivariate dataset consists of 3 multivalued discrete and 5 continuous attributes in addition to 2 attributes which are derived from the dataset. Acceleration on power proved to be the best estimator among the attributes. Regression is applied on the dataset where all the models could predict fuel consumption accurately. Various Machine Learning models have an absolute relative error less than 10%. The random forest model is proved to have the highest accuracy and runs faster, making it suitable for wide application. This method lays a foundation for monitoring database improvement and fine management of urban – rural transportation fuel consumption.

List of Figures

List of Tables

List of Snapshots

1.Introduction

Vehicle energy consumption and pollutant emissions are key problems for the healthy and sustainable development of urban - rural transportation. With the continuous growth of car ownership, the energy consumption of private cars increased 5.1 times, from 130.12 to 680.34 million gallons of fuel, from 2005 to 2015. Based on growth of the population, GDP, and the proportion of secondary and tertiary industries, the trend of future transportation energy consumption can be predicted. The energy consumption of private cars will continue to increase before 2023, when it is expected to reach 1170.38 million gallons of fuel. Therefore, reducing energy consumption has become one of the most important challenges in the transportation field.

This study evaluates methods of machine learning (ML) with help from statistical analysis for predicting fuel consumption in vehicles. The idea is to use historical data describing primary and secondary attributes of a vehicle to predict fuel consumption in gallons per mile. The general problem description is to examine a large number of attributes describing fuel consumption and to employ ML methods to find a regression from such attributes to predict fuel efficiency.

Fuel consumption models for vehicles are of interest to manufacturers, regulators, and consumers. They are needed across all the phases of the vehicle life-cycle. This paper, is focused on modeling average fuel consumption for vehicles. In general, techniques used to develop models for fuel consumption fall under three main categories:

• Physics-based models, which are derived from an in depth understanding of the physical system. These models describe the dynamics of the components of the vehicle at each time stamp using detailed mathematical equations.

• Machine learning models, which are data-driven and represents an abstract mapping from an input space consisting of a selected set of predictors to an output space that represents the target output, in this case average fuel consumption.

• Statistical models, which are also data-driven and establish a mapping between the probability distribution of a selected set of predictors and the target outcome.

Trade-offs among the above techniques are primarily with respect to cost and accuracy as per the requirements of the intended application. The studies of the past are mainly based on 11 features dataset. This dataset is a slightly modified version of the dataset provided in the StatLib library. In line with the use by Ross Quinlan (1993) in predicting the attribute "mpg", 8 of the original instances were removed because they had unknown values for the "mpg" attribute. The data concerns city-cycle fuel consumption in miles per gallon, to be predicted in terms of 3 multivalued discrete and 5 continuous attributes (Quinlan, 1993).

Various Machine Learning models employed have an absolute relative error less than 10%. The random forest model is proved to have the highest accuracy and runs faster, making it suitable for wide application. The work done by L. Breiman, by applying “Random forests,” Machine Learning, plays along the notion established in the paper. Moreover the works done by H. Drucker, J. C. Chris, L. Kaufman, A. Smola, and V. Vapnik, “Support vector regression machines,” in Advances in Neural Information Processing Systems gives an overview of the SVM mechanism but has a relative error more than the Random Forest model.

D. Yang, M. Li, and X. Ban, “Real-time on-board monitoring method of gasoline vehicle fuel consumption based on OBD system reflects on the statistical modelling while works done by X.-h. Zhao, Y. Yao, Y.-p. Wu, C. Chen, and J. Rong, on “Prediction model of driving energy consumption based on PCA and BP network and works of W. J. Zhang, S. X. Yu, Y. F. Peng, Z. J. Cheng, and C. Wang, on “Driving habits analysis on vehicle data using error back-propagation neural network algorithm employs Deep Learning and Neural Network Architecture to produce the results.

Air pollution is one of the world’s single biggest environmental risks to human health, with one in nine deaths linked to poor indoor or outdoor air quality. The World Health Organization (WHO) estimates that 92% of the world’s population lives in locations where local air pollution exceeds WHO limits. Energy efficiency can reduce both indoor and outdoor concentrations of air pollutants. In doing so, energy efficiency drives a range of economic, environmental and health benefits associated with local air quality.

The energy system contributes vitally to economic and social progress around the world, but the associated emissions and negative side effects are costly. Scaling up the Machine Learning models to a global scale helps in proficient saving of fuel and reduces air pollution.

2.Literature Review

The summary of the literature review can be seen in Table 1. Several approaches have been performed on this popular dataset, but the accuracy obtained by all the approaches is less than 10% RMSE

Sr.no	Author	Year	Findings
1.	K. Hu, J. Wu, and M. Liu	2018	Modelling of EVs energy consumption from perspective of field test data using Random Forest and developing driving style questionnaires using CNN.
2.	Z. Xu, T. Wei, S. Easa, X. Zhao, and X. Qu	2018	Modeling relationship between truck fuel consumption and driving behavior using data from internet of vehicles by integrating mobile app to the Deep Learning Module.
3.	H. Wang	2017	Energy consumption in transport: an assessment of changing trend, influencing factors and consumption forecast.

4.	D. Yang, M. Li, and X. Ban	2016	Real-time on-board monitoring method of gasoline vehicle fuel consumption based on OBD system.
5.	S. Wickramanayake and H. M. N. D. Bandara	2016	Fuel consumption prediction of fleet of vehicles using machine learning and deep learning.
6.	X.-h. Zhao, Y. Yao, Y.-p. Wu, C. Chen, and J. Rong	2016	Prediction model of driving energy consumption based on PCA and BP network.
7.	W. J. Zhang, S. X. Yu, Y. F. Peng, Z. J. Cheng, and C. Wang	2015	Driving habits analysis on vehicle data using error back-propagation and neural network algorithm
8.	Z. Ramedani, M. Omid, A. Keyhani, S. Shamshirband, and B. Khoshnevisan	2014	Potential of radial basis function based support vector regression for global fuel consumption.
9.	H.-l. Feng,	2013	Study on prediction model of fuel consumption index in Chongqing city based on SVR model.

10.	G. Guido, A. Vitale, V. Astarita, F. Saccomanno, V. P. Giofré, and V. Gallelli	2012	Estimation of safety performance measures from smartphone sensors and merging with neural network.
11.	D. A. Johnson and M. M. Trivedi	2011	Driving style recognition using a smartphone as a sensor platform and neural network and back propagation network.
12.	J. N. Barkenbus	2010	Eco-driving: an overlooked climate change initiative – fuel consumption analysis using Random Forest Modeling.
13.	T. Hiraoka, Y. Terakado, S. Matsumoto, and S. Yamabe	2009	Quantitative evaluation of eco-driving on fuel consumption based on driving simulator experiments.
14.	K. Ahn and H. Rakha	2008	The effects of route choice decisions on vehicle energy consumption and emissions.
15.	Dan Pelleg	2004	Scalable and Practical Probability Density Estimators for Fuel consumption prediction.
16.	Christopher R. Palmer and Christos Faloutsos	2003	Electricity Based External Similarity of Categorical Attributes.

17.	Thomas Melluish and Craig Saunders and Ilia Nouretdinov and Volodya Vovkand Carol S. Saunders and I. Nouretdinov V.	2001	The typicalness framework: a comparison with the Bayesian approach.
18.	Dan Pelleg and Andrew W. Moore.	2001	Mixtures of Rectangles: Interpretable Soft Clustering.
19.	Zhi-Hua Zhou and Shifu Chen and Zhaoqian Chen.	2000	A Statistics Based Approach for Extracting Priority Rules from Trained Neural Networks.
20.	Mauro Birattari and Gianluca Bontempi and Hugues Bersini	1998	Lazy Learning Meets the Recursive Least Squares Algorithm resulting in maximum mean square error.

Table 2.1

3.Overview

3.1 Dataset

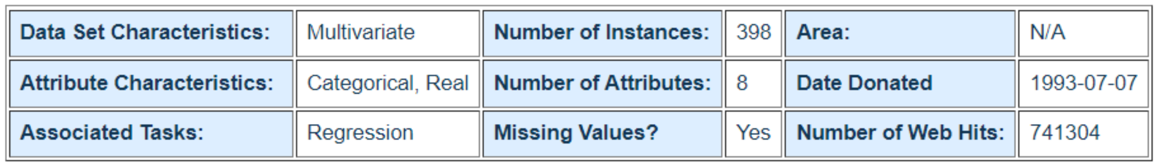
3.1.1 Software Requirement

One requires a python environment, preferablely Anaconda — a virtual environment dedicated to this project.

A Machine Learning directory is created in Jupyter to contain the ipython notebook.

3.1.2 Description of Dataset

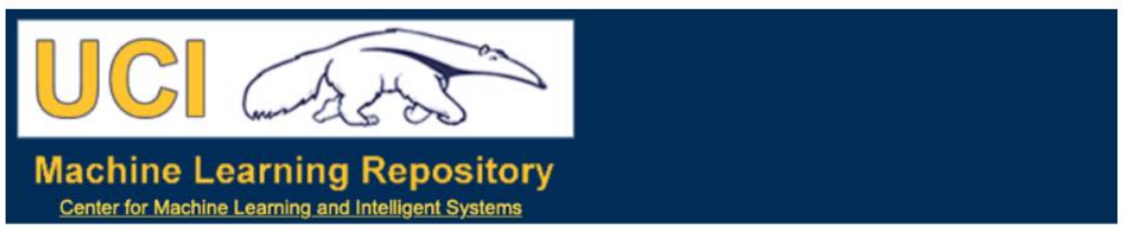
The dataset used for this research purpose was the UCI Machine Learning Repository: Auto MPG Dataset. This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University. The dataset was used in the 1983 American Statistical Association Exposition.



Data Set Characteristics:	Multivariate	**Number of Instances:**	398	**Area:**	N/A
Attribute Characteristics:	Categorical, Real	**Number of Attributes:**	8	**Date Donated**	1993-07-07
Associated Tasks:	Regression	**Missing Values?**	Yes	**Number of Web Hits:**	741304

Table 3.1

This dataset is a slightly modified version of the dataset provided in the StatLib library. In line with the use by Ross Quinlan (1993) in predicting the attribute "mpg", 8 of the original instances were removed because they had unknown values for the "mpg" attribute. The data concerns city-cycle fuel consumption in miles per gallon, to be predicted in terms of 3 multivalued discrete and 5 continuous attributes. (Quinlan, 1993)

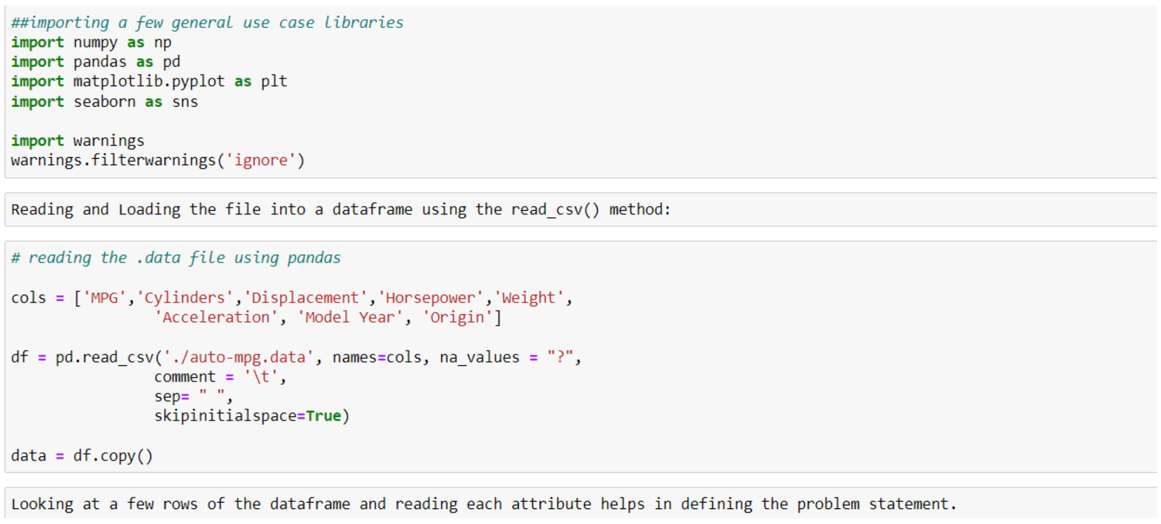

UCI
Machine Learning Repository
Center for Machine Learning and Intelligent Systems

Snapshot 3.1

Attribute Information:

1. mpg: continuous.
2. cylinders: multi-valued discrete.
3. displacement: continuous.
4. horsepower: continuous.
5. weight: continuous.
6. acceleration: continuous.
7. model year: multi-valued discrete.
8. origin: multi-valued discrete.
9. car name: string (unique for each instance).

3.2 Loading of Data



```
##importing a few general use case libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

Reading and Loading the file into a dataframe using the read_csv() method:

```
# reading the .data file using pandas

cols = ['MPG','Cylinders','Displacement','Horsepower','Weight',
                'Acceleration', 'Model Year', 'Origin']

df = pd.read_csv('./auto-mpg.data', names=cols, na_values = "?",
                comment = '\t',
                sep= " ",
                skipinitialspace=True)

data = df.copy()
```

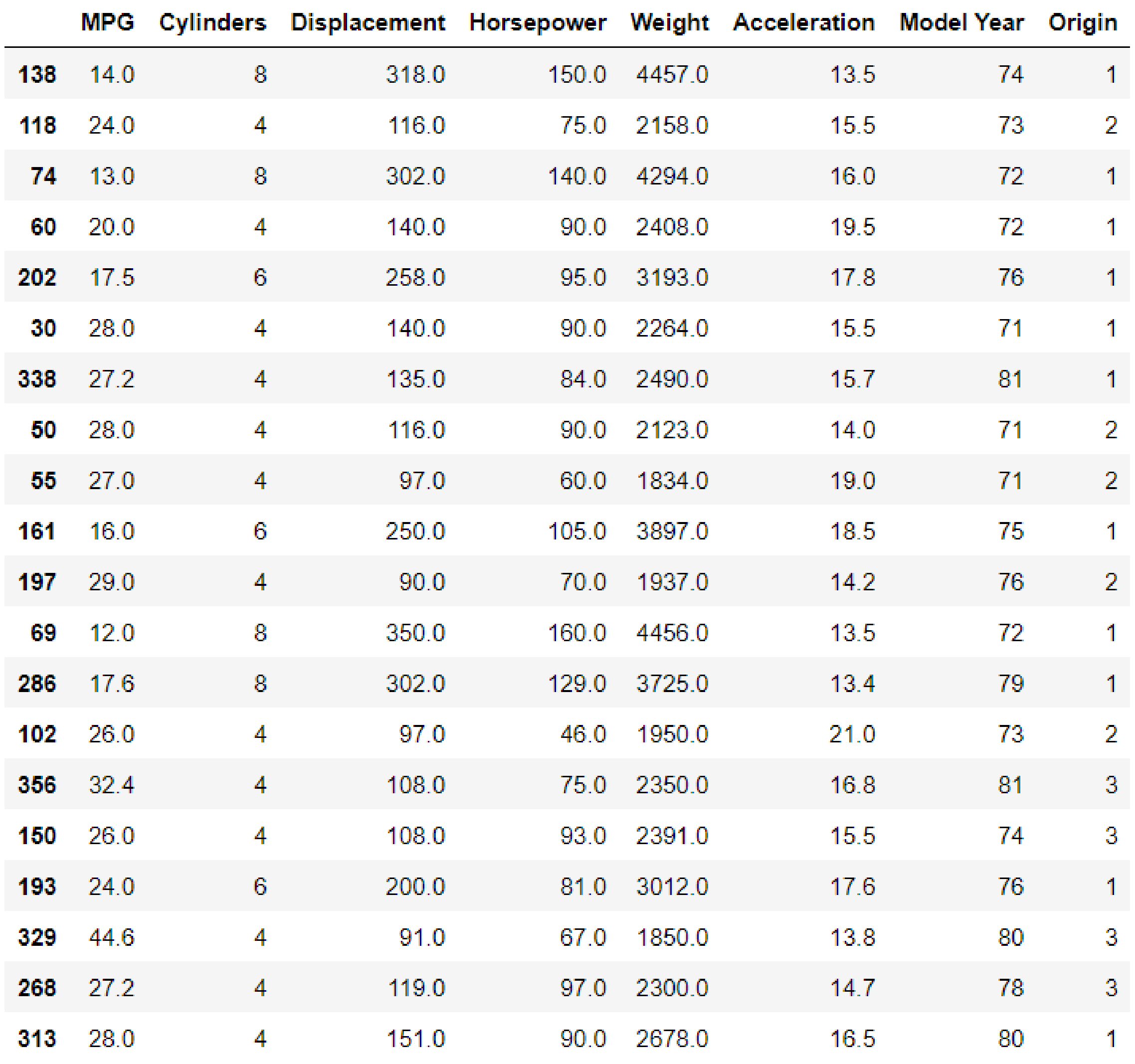
Looking at a few rows of the dataframe and reading each attribute helps in defining the problem statement.

Snapshot 3.2

Next, looking at a few rows of the data frame and reading the description of each attribute helps in defining the problem statement.

3.3 Problem Statement

The data contains the MPG (Mile Per Gallon) variable which is continuous data and tells us about the efficiency of fuel consumption of a vehicle. The aim here is to predict the MPG value for a vehicle, given that we have other attributes of that vehicle.



	MPG	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model Year	Origin
138	14.0	8	318.0	150.0	4457.0	13.5	74	1
118	24.0	4	116.0	75.0	2158.0	15.5	73	2
74	13.0	8	302.0	140.0	4294.0	16.0	72	1
60	20.0	4	140.0	90.0	2408.0	19.5	72	1
202	17.5	6	258.0	95.0	3193.0	17.8	76	1
30	28.0	4	140.0	90.0	2264.0	15.5	71	1
338	27.2	4	135.0	84.0	2490.0	15.7	81	1
50	28.0	4	116.0	90.0	2123.0	14.0	71	2
55	27.0	4	97.0	60.0	1834.0	19.0	71	2
161	16.0	6	250.0	105.0	3897.0	18.5	75	1
197	29.0	4	90.0	70.0	1937.0	14.2	76	2
69	12.0	8	350.0	160.0	4456.0	13.5	72	1
286	17.6	8	302.0	129.0	3725.0	13.4	79	1
102	26.0	4	97.0	46.0	1950.0	21.0	73	2
356	32.4	4	108.0	75.0	2350.0	16.8	81	3
150	26.0	4	108.0	93.0	2391.0	15.5	74	3
193	24.0	6	200.0	81.0	3012.0	17.6	76	1
329	44.6	4	91.0	67.0	1850.0	13.8	80	3
268	27.2	4	119.0	97.0	2300.0	14.7	78	3
313	28.0	4	151.0	90.0	2678.0	16.5	80	1

Snapshot 3.3

3.4 Exploratory Data Analysis

Carrying out exploratory analysis to figure out the important features and to create new combination of features. For this rather simple dataset, the exploration is broken down into a series of steps:

1.Check for Data type of columns.

2.Check for null values.

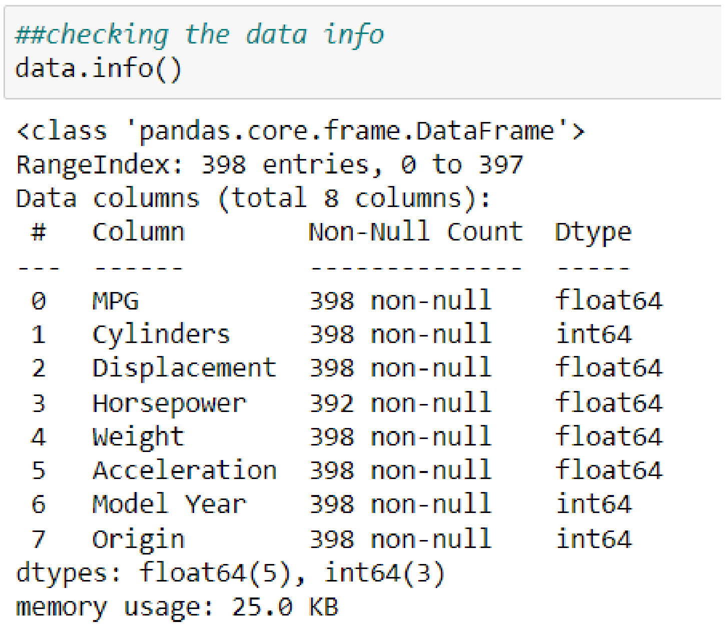
3.Check for outliers.

4.Look for the category distribution in categorical columns.

5.Plot for correlation.

6.Look for new variables.

3.4.1 Checking for Data type of Columns



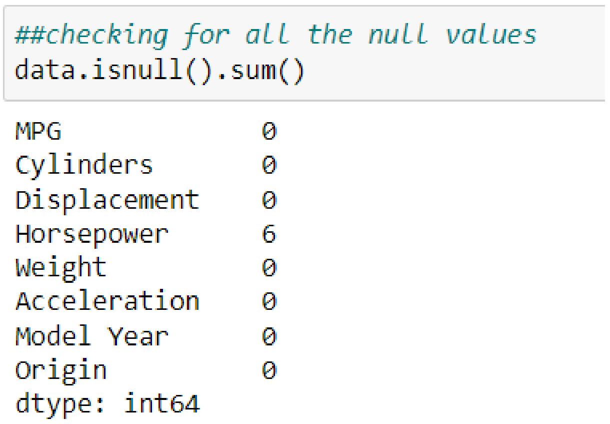
```
##checking the data info
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   MPG           398 non-null    float64
 1   Cylinders     398 non-null    int64
 2   Displacement  398 non-null    float64
 3   Horsepower    392 non-null    float64
 4   Weight        398 non-null    float64
 5   Acceleration  398 non-null    float64
 6   Model Year    398 non-null    int64
 7   Origin        398 non-null    int64
dtypes: float64(5), int64(3)
memory usage: 25.0 KB
```

Snapshot 3.4.1

3.4.2 Checking for null

The horsepower column has 6 missing values. One has to study the column a bit more.

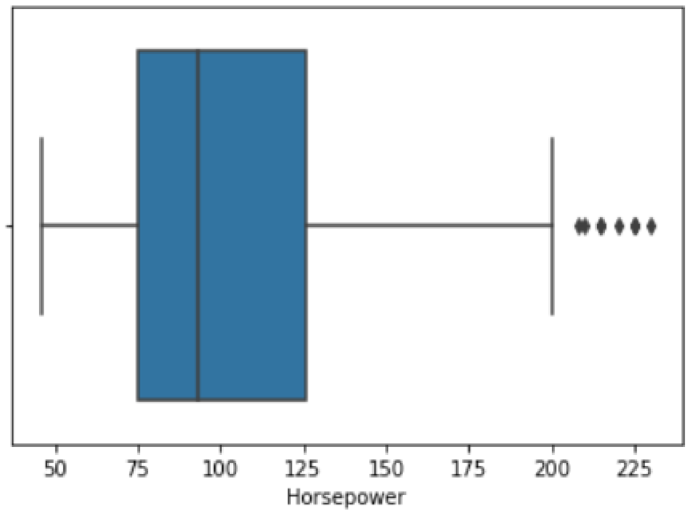


```
##checking for all the null values
data.isnull().sum()
```

```
MPG             0
Cylinders       0
Displacement    0
Horsepower      6
Weight          0
Acceleration    0
Model Year      0
Origin          0
dtype: int64
```

Snapshot 3.4.2

3.4.3 Checking for Outliers



Snapshot 3.4.3

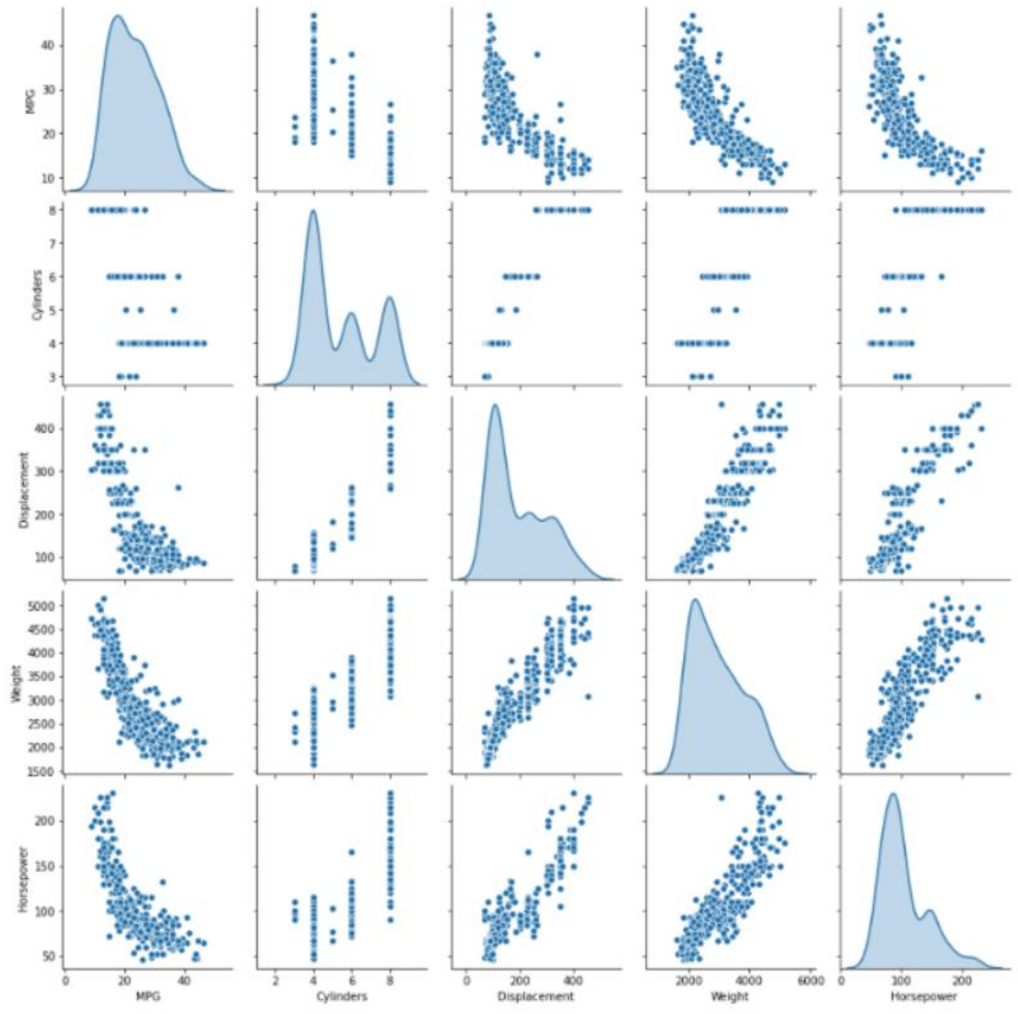
Since there are a few outliers, one can use the median of the column to impute the missing values using the pandas median() method.

3.4.4 Categorical Distribution on Categorical Columns

The two categorical columns are Cylinders and Origin, which only have a few categories of values. Looking at the distribution of the values among these categories tells how the data is distributed.

3.4.5 Plotting for Correlation

The pair plot gives you a brief overview of how each variable behaves with respect to every other variable. For example, the MPG column (our target variable) is negatively correlated with the displacement, weight, and horsepower features



Snapshot 3.4.5

3.4 Data Preparation

One of the most important aspects of Data Preparation is to keep automating the steps in the form of functions and classes. This makes it easier to integrate the methods and pipelines into the main product.

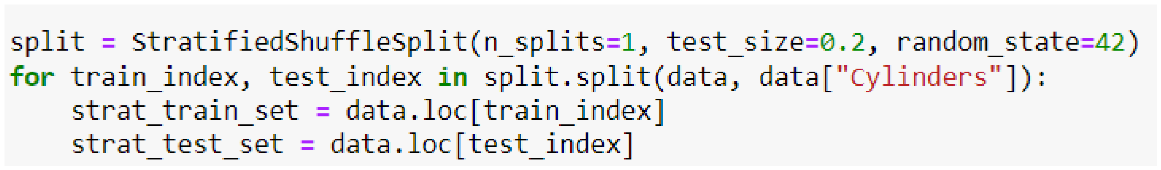
1.Handling Categorical Attribute - OneHotEncoder.

2.Data Cleaning - Imputer.

3.Attribute Addition - Adding Custom Transformation.

4.Setting up Data Transformation Pipeline for Numerical and Categorical Column.

3.5.1 Setting aside Test Data Set



```
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(data, data["Cylinders"]):
    strat_train_set = data.loc[train_index]
    strat_test_set = data.loc[test_index]
```

Snapshot 3.5.1

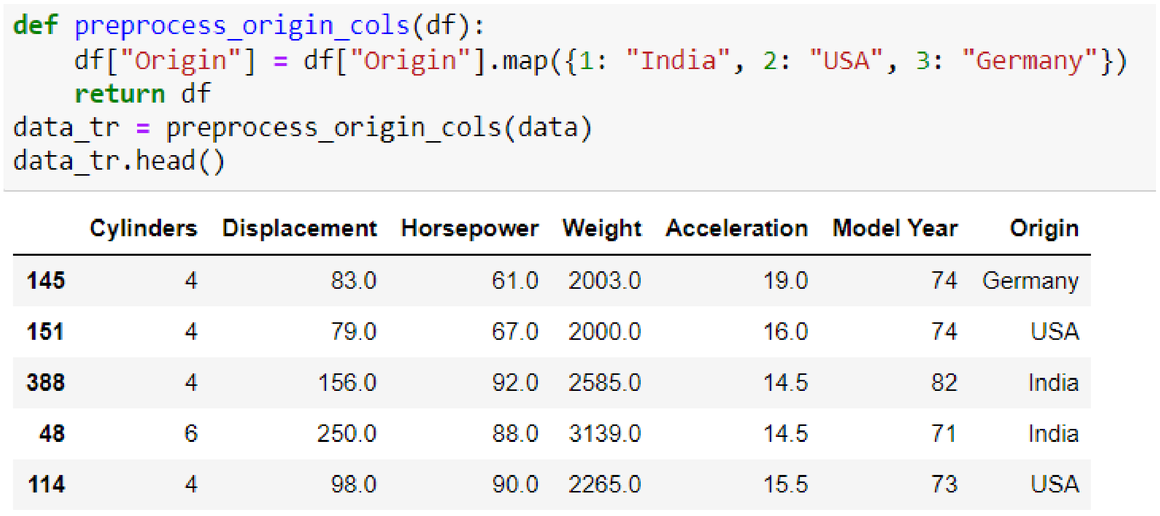
One of the first things one should do, as one need to test the final model on unseen/unbiased data.

There are many ways to split the data into training and testing sets but we want our test set to represent the overall population and not just a few specific categories. Thus, instead of using simple and common train_test_split() method from sklearn, one could use stratified sampling.

Stratified Sampling - Creating homogeneous subgroups called strata from the overall population and sample the right number of instances to each stratum to ensure that the test set is representative of the overall population.

3.5.2 Pre-Processing the Origin Column

The Origin column about the origin of the vehicle has discrete values that look like the code of a country. To add some complication and make it more explicit, these numbers are converted to strings.

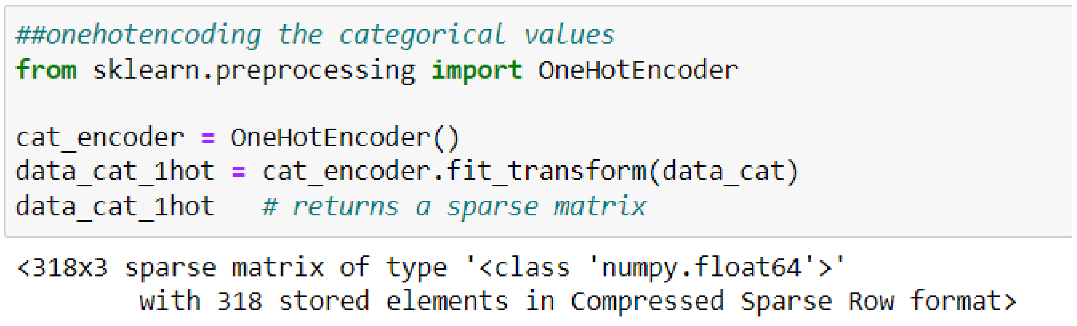


```python
def preprocess_origin_cols(df):
    df["Origin"] = df["Origin"].map({1: "India", 2: "USA", 3: "Germany"})
    return df
data_tr = preprocess_origin_cols(data)
data_tr.head()
```

	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model Year	Origin
145	4	83.0	61.0	2003.0	19.0	74	Germany
151	4	79.0	67.0	2000.0	16.0	74	USA
388	4	156.0	92.0	2585.0	14.5	82	India
48	6	250.0	88.0	3139.0	14.5	71	India
114	4	98.0	90.0	2265.0	15.5	73	USA

Snapshot 3.5.2

3.5.3 One – Hot Encoding the Origin Column

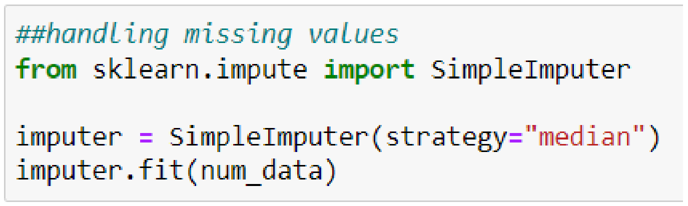


```
##onehotencoding the categorical values
from sklearn.preprocessing import OneHotEncoder

cat_encoder = OneHotEncoder()
data_cat_1hot = cat_encoder.fit_transform(data_cat)
data_cat_1hot   # returns a sparse matrix
```

```
<318x3 sparse matrix of type '<class 'numpy.float64'>'
	with 318 stored elements in Compressed Sparse Row format>
```

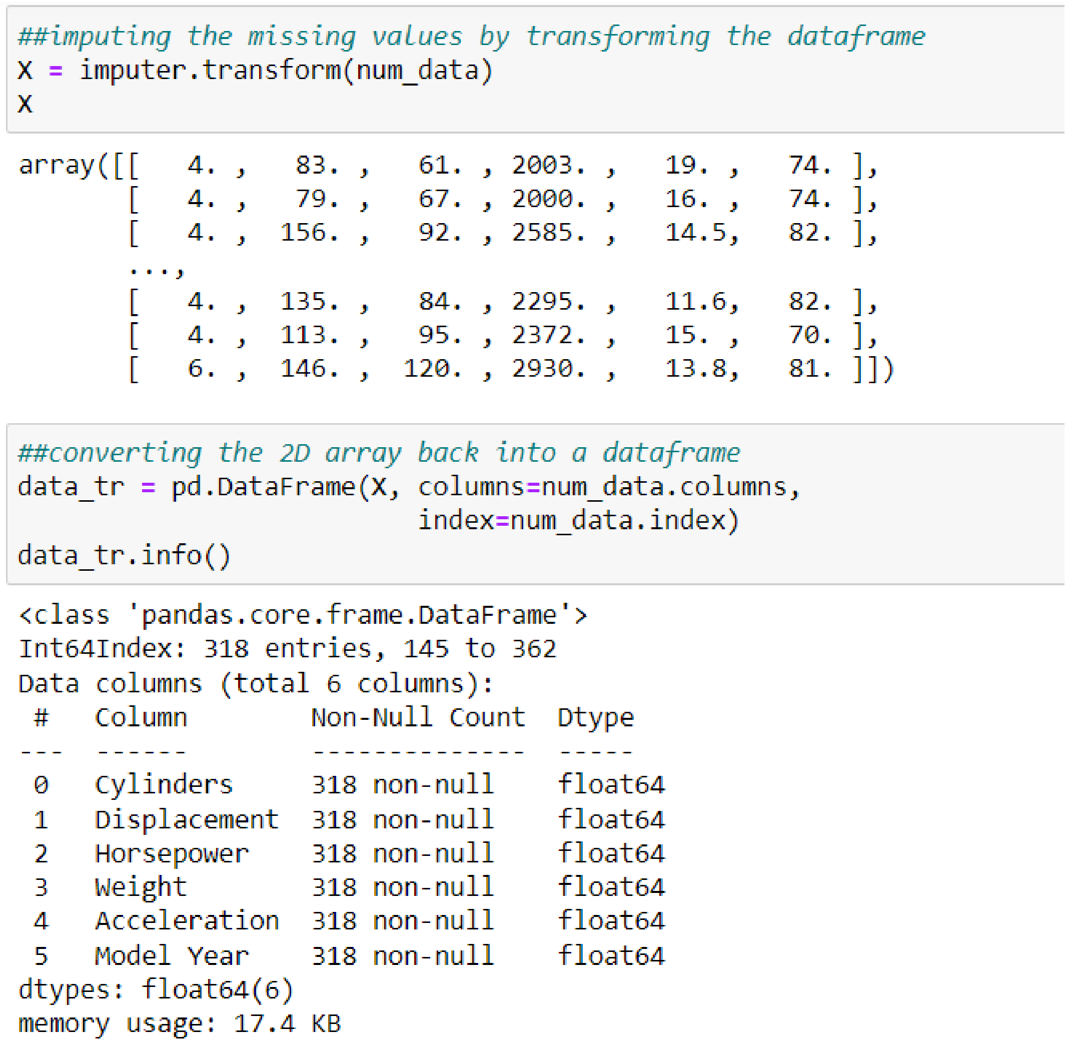
Snapshot 3.5.3

3.5.4 Handling Missing Values using SimpleImputer



```
##handling missing values
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy="median")
imputer.fit(num_data)
```

Snapshot 3.5.4.1



```python
##imputing the missing values by transforming the dataframe
X = imputer.transform(num_data)
X
```

```
array([[   4. ,   83. ,   61. , 2003. ,   19. ,   74. ],
       [   4. ,   79. ,   67. , 2000. ,   16. ,   74. ],
       [   4. ,  156. ,   92. , 2585. ,   14.5,   82. ],
       ...,
       [   4. ,  135. ,   84. , 2295. ,   11.6,   82. ],
       [   4. ,  113. ,   95. , 2372. ,   15. ,   70. ],
       [   6. ,  146. ,  120. , 2930. ,   13.8,   81. ]])
```

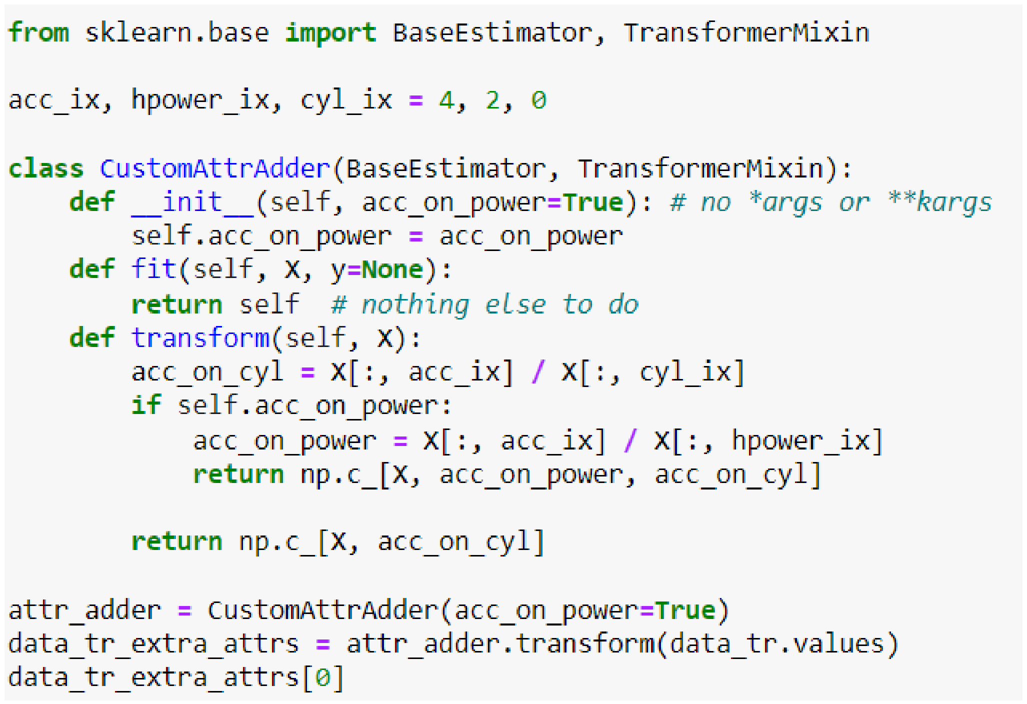
```python
##converting the 2D array back into a dataframe
data_tr = pd.DataFrame(X, columns=num_data.columns,
                          index=num_data.index)
data_tr.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 318 entries, 145 to 362
Data columns (total 6 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Cylinders     318 non-null    float64
 1   Displacement  318 non-null    float64
 2   Horsepower    318 non-null    float64
 3   Weight        318 non-null    float64
 4   Acceleration  318 non-null    float64
 5   Model Year    318 non-null    float64
dtypes: float64(6)
memory usage: 17.4 KB
```

Snapshot 3.5.4.2

3.5.5 Adding Attributes using BaseEstimator and Transformer

Testing for new variables — Analyze the correlation of each variable with the target variable. Found acceleration_on_power and acceleration_on_cylinder as two new variables which turned out to be more positively correlated than the original variables.



```python
from sklearn.base import BaseEstimator, TransformerMixin

acc_ix, hpower_ix, cyl_ix = 4, 2, 0

class CustomAttrAdder(BaseEstimator, TransformerMixin):
    def __init__(self, acc_on_power=True): # no *args or **kargs
        self.acc_on_power = acc_on_power
    def fit(self, X, y=None):
        return self  # nothing else to do
    def transform(self, X):
        acc_on_cyl = X[:, acc_ix] / X[:, cyl_ix]
        if self.acc_on_power:
            acc_on_power = X[:, acc_ix] / X[:, hpower_ix]
            return np.c_[X, acc_on_power, acc_on_cyl]

        return np.c_[X, acc_on_cyl]

attr_adder = CustomAttrAdder(acc_on_power=True)
data_tr_extra_attrs = attr_adder.transform(data_tr.values)
data_tr_extra_attrs[0]
```

Snapshot 3.5.5

In order to make changes to dataset and create new variables, sklearn offers the BaseEstimator class. Using it, new features are created by defining one's own class.

A class is created to add two new features found:

1.acc_on_power — Acceleration divided by Horsepower

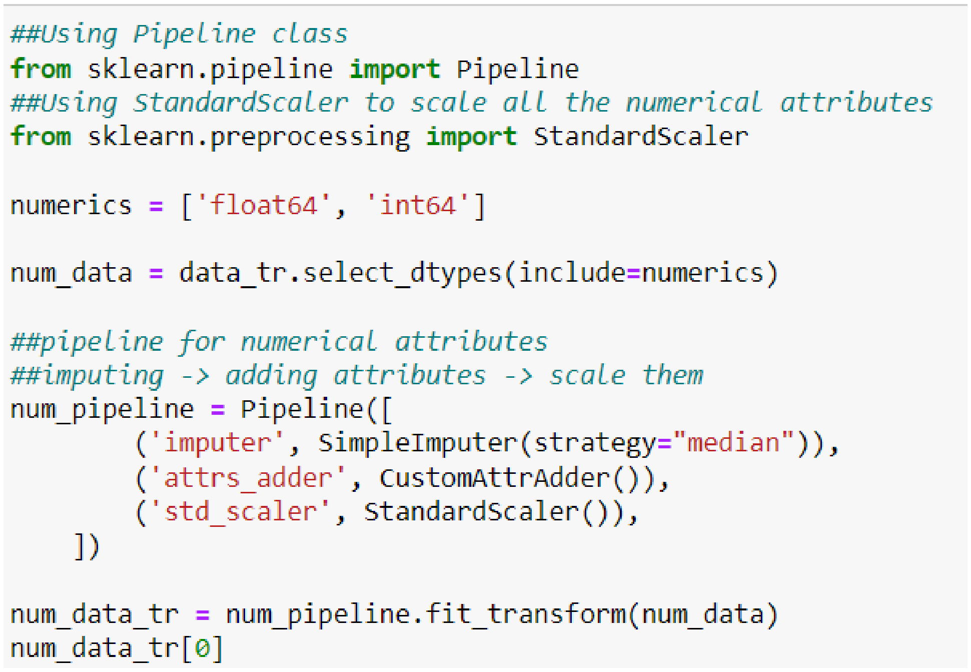
2.acc_on_cyl — Acceleration divided by the number of Cylinders

3.5.6 Creating a Pipeline of Tasks

Setting up Data Transformation Pipeline for numerical and categorical attributes. As one need to automate as much as possible, Sklearn offers a great number of classes and methods to develop such automated pipelines of data transformations.

The major transformations are to be performed on numerical columns. The cascaded of set of transformations are:

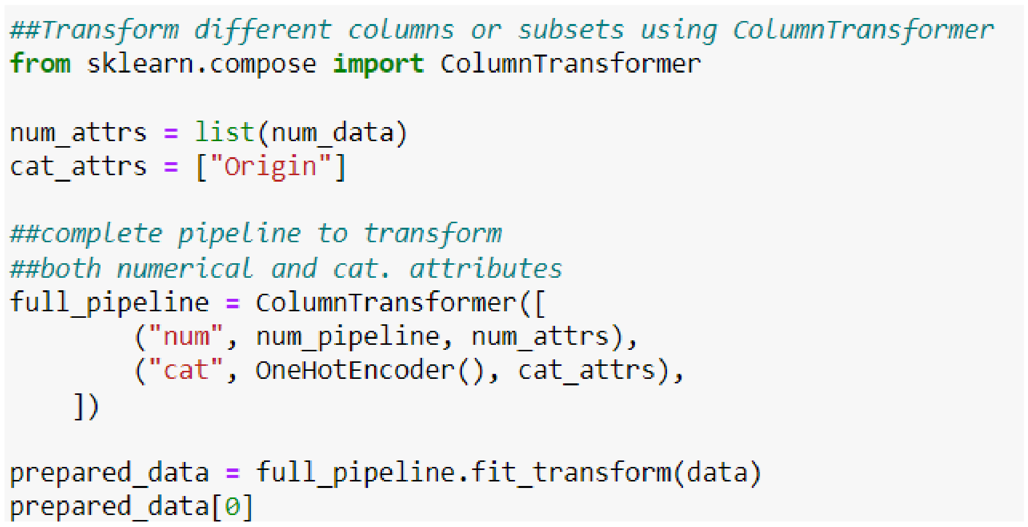
1.Imputing Missing Values — using the SimpleImputer.
2.Custom Attribute Addition— using the custom attribute class.
3.Standard Scaling of each Attribute — scaling the values before feeding them to the ML model, using the standardScaler class.



```
##Using Pipeline class
from sklearn.pipeline import Pipeline
##Using StandardScaler to scale all the numerical attributes
from sklearn.preprocessing import StandardScaler

numerics = ['float64', 'int64']

num_data = data_tr.select_dtypes(include=numerics)

##pipeline for numerical attributes
##imputing -> adding attributes -> scale them
num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy="median")),
        ('attrs_adder', CustomAttrAdder()),
        ('std_scaler', StandardScaler()),
    ])

num_data_tr = num_pipeline.fit_transform(num_data)
num_data_tr[0]
```

Snapshot 3.5.6

3.5.7 Transforming Numerical and Categorical Attributes


```
##Transform different columns or subsets using ColumnTransformer
from sklearn.compose import ColumnTransformer

num_attrs = list(num_data)
cat_attrs = ["Origin"]

##complete pipeline to transform
##both numerical and cat. attributes
full_pipeline = ColumnTransformer([
        ("num", num_pipeline, num_attrs),
        ("cat", OneHotEncoder(), cat_attrs),
    ])

prepared_data = full_pipeline.fit_transform(data)
prepared_data[0]
```

Snapshot 3.5.7

3.6 Selecting and Training Models

1. Selecting and Training a few Algorithms (Linear Regression, Decision Tree, Random Forest, SVM)

2. Evaluation using Mean Squared Error.

3. Model Evaluation using Cross Validation.

4. Hyperparameter Tuning using GridSearchCV.

5. Checking Feature Importance.

6. Evaluating the Final System on Test Data.

7. Saving the Model.

3.6.1 Machine Learning Models

The multivariate dataset consists of 3 multivalued discrete and 5 continuous attributes in addition to 2 attributes which are derived from the dataset. Regression is applied on the dataset where all the models could predict fuel consumption accurately.

Various Machine Learning models have an absolute relative error less than 10%. The random forest model is proved to have the highest accuracy and runs faster, making it suitable for wide application. This method lays a foundation for monitoring database improvement and fine management of urban – rural transportation fuel consumption.

Since Regression is applied, following models are trained:

1. Linear Regression
2. Decision Tree Regressor
3. Random Forest Regressor
4. SVM Regressor

Now, on performing the same for Decision Tree, a 0.0 RMSE value is achieved which is not possible – as there is no "perfect" Machine Learning Model (we've not reached that point yet).

3.6.2 Overfitting

Problem:

Testing the model on the same data one trained on, is a problem. Now, one can't use the test data yet until the best model is finalized and is ready to go into production.

Solution:

Cross-Validation

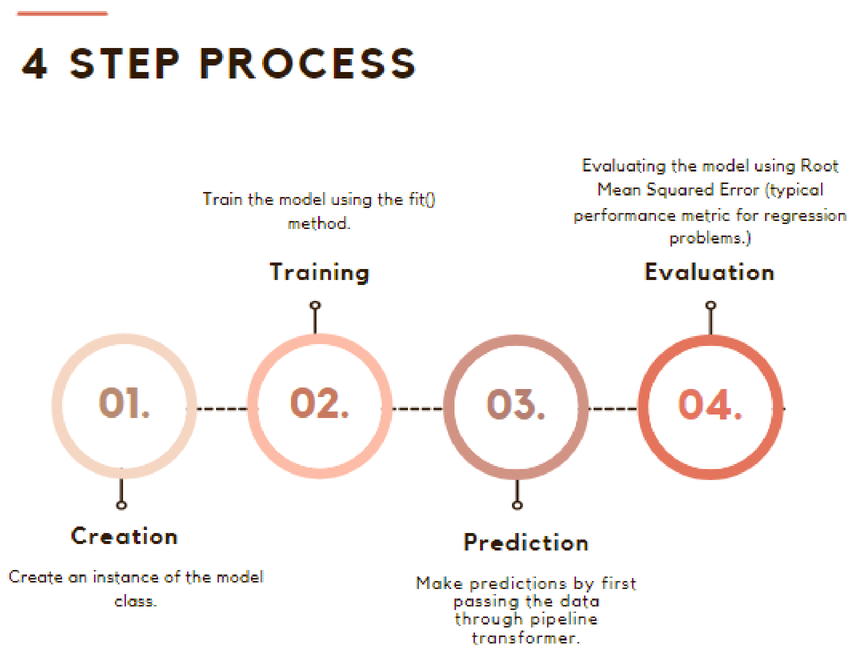
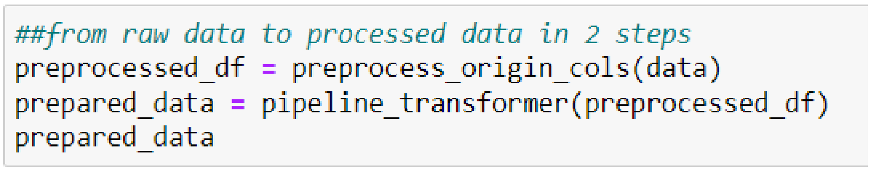

4 STEP PROCESS
01.
Creation
Create an instance of the model class.
02.
Training
Train the model using the fit() method.
03.
Prediction
Make predictions by first passing the data through pipeline transformer.
04.
Evaluation
Evaluating the model using Root Mean Squared Error (typical performance metric for regression problems.)

Figure 3.6.2

3.6.3 Raw Data to Processed Data



```
##from raw data to processed data in 2 steps
preprocessed_df = preprocess_origin_cols(data)
prepared_data = pipeline_transformer(preprocessed_df)
prepared_data
```

Snapshot 3.6.3

3.6.4 Cross Validation

Scikit-Learn's K-fold cross-validation feature randomly splits the training set into K distinct subsets called folds. Then it trains and evaluates the model K times, picking a different fold for evaluation every time and training on the other K-1 folds.

The result is an array containing the K evaluation scores.

All the Machine Learning models have an absolute relative error less than 10%. The random forest model is proved to have the highest accuracy and runs faster, making it suitable for wide application.

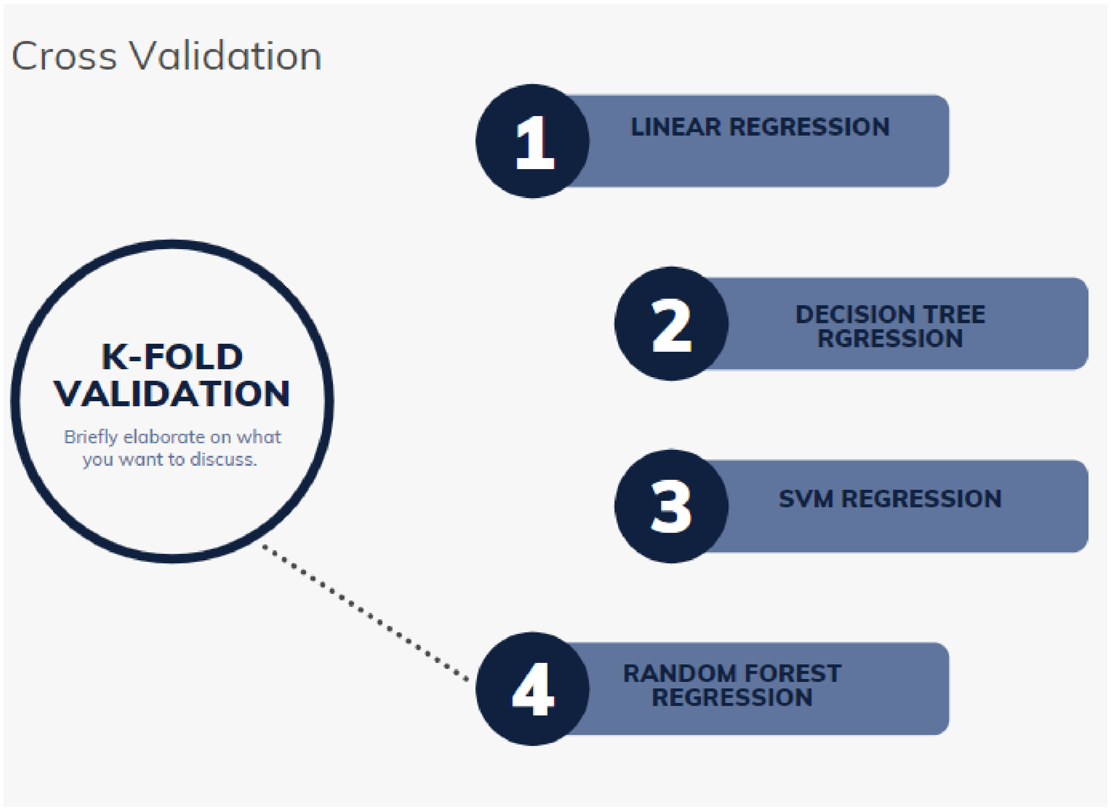


Figure 3.6.4

3.6.5 Linear Regression

Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task**.** Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables. They are considering and the number of independent variables being used.

3.6.6 Decision Tree

Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes.

3.6.7 SVM Regression

Support Vector Regression is a supervised learning algorithm that is used to predict discrete values. Support Vector Regression uses the same principle as the SVMs. The basic idea behind SVR is to find the best fit line. In SVR, the best fit line is the hyperplane that has the maximum number of points.

3.6.8 Random Forest Regression

Every individual decision tree has high variance, but when combining all of them together in parallel then the resultant variance is low, as each individual decision tree gets perfectly trained on the sample data and hence output doesn't depend on one decision tree but multiple decision trees. In the case of a regression problem, the final output is the mean of all the outputs. This is called Aggregation. A Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called Bootstrap and Aggregation, commonly known as bagging. The basic idea is to combine multiple decision trees in determining the final output rather than relying on individual decision trees.

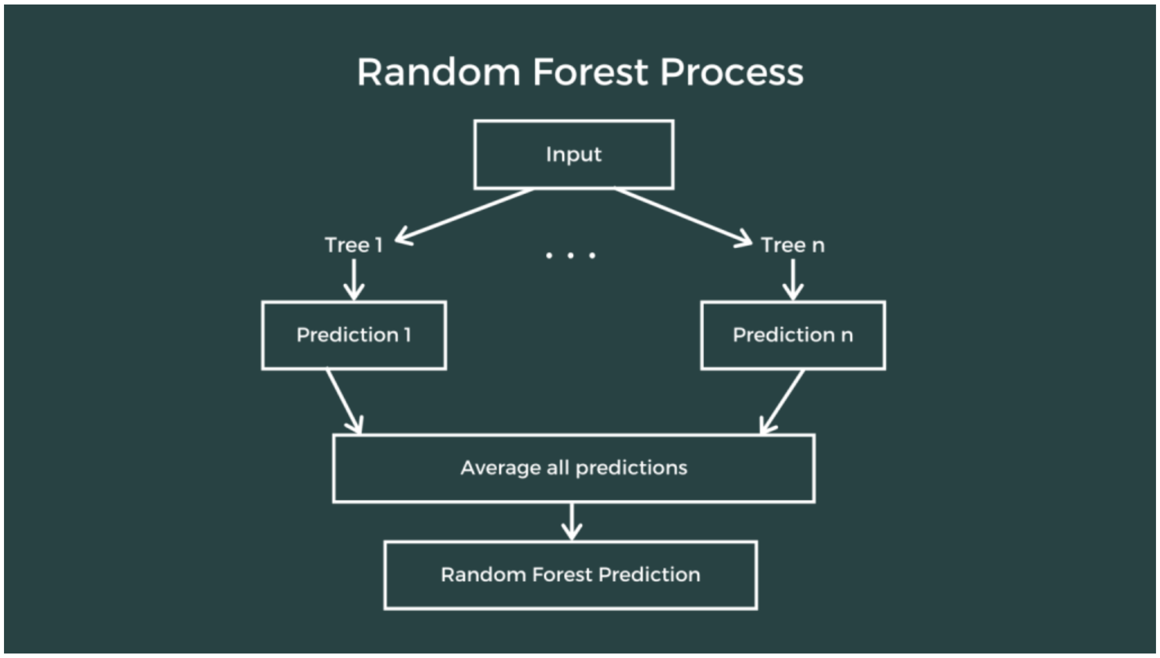
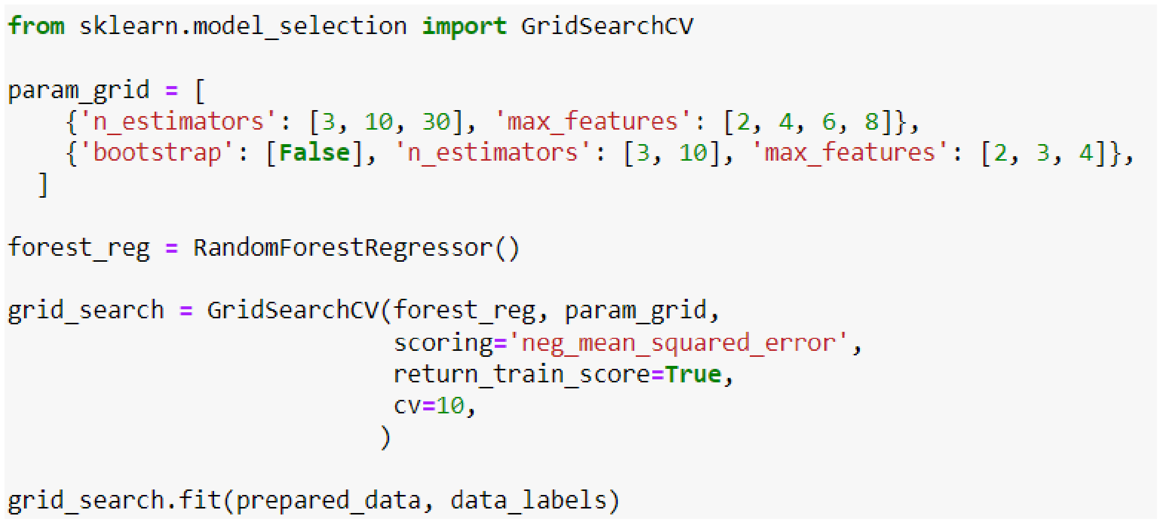


Figure 3.6.8

3.6.9 Hyper Parameter Tuning using GridSearchCV

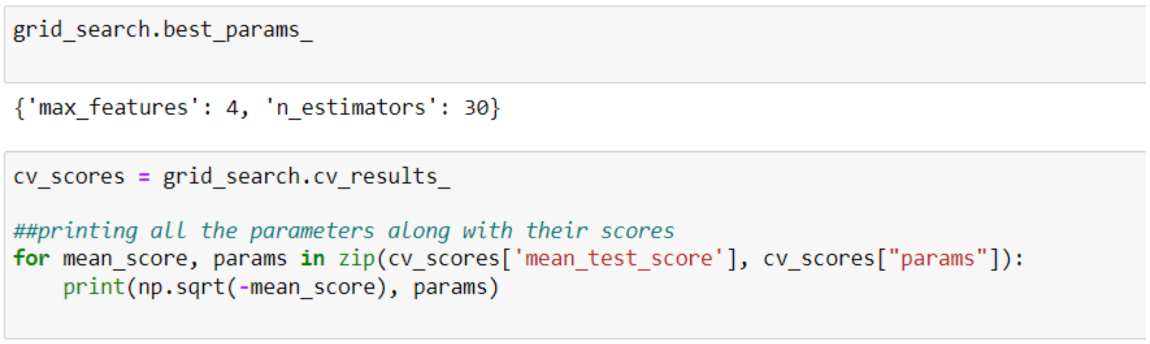
After testing all the models, RandomForestRegressor has performed the best but it still needs to be fine-tuned. A model is like a radio station with a lot of knobs to handle and tune. We can either tune all these knobs manually or provide a range of values/combinations to test.GridSearchCV is used to find out the best combination of hyperparameters for the RandomForest model.



```python
from sklearn.model_selection import GridSearchCV

param_grid = [
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},
  ]

forest_reg = RandomForestRegressor()

grid_search = GridSearchCV(forest_reg, param_grid,
                           scoring='neg_mean_squared_error',
                           return_train_score=True,
                           cv=10,
                          )

grid_search.fit(prepared_data, data_labels)
```

Snapshot 3.6.9.1

GridSearchCV requires one to pass the parameter grid. This is a python dictionary with parameter named as keys mapped with the list of values to test for that parameter. One can pass the model, scoring method, and cross-validation folds to it. Train the model and it returns the best parameters and results for each combination of parameters.



```python
grid_search.best_params_
```

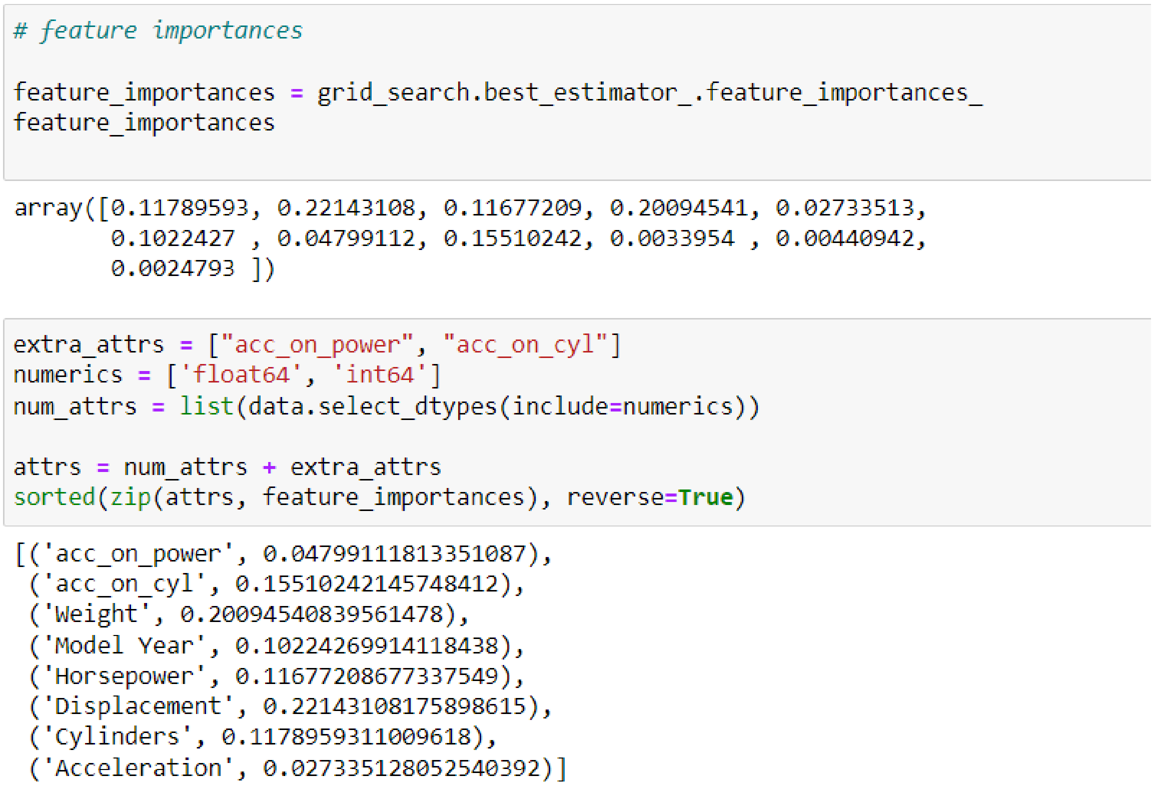
```
{'max_features': 4, 'n_estimators': 30}
```

```python
cv_scores = grid_search.cv_results_

##printing all the parameters along with their scores
for mean_score, params in zip(cv_scores['mean_test_score'], cv_scores["params"]):
    print(np.sqrt(-mean_score), params)
```

Snapshot 3.6.9.2

3.6.10 Checking Feature Importance

Checking the feature importance by enlisting the features and zipping them up with the best_estimator's feature importance attribute.



```
# feature importances

feature_importances = grid_search.best_estimator_.feature_importances_
feature_importances
```

```
array([0.11789593, 0.22143108, 0.11677209, 0.20094541, 0.02733513,
       0.1022427 , 0.04799112, 0.15510242, 0.0033954 , 0.00440942,
       0.0024793 ])
```

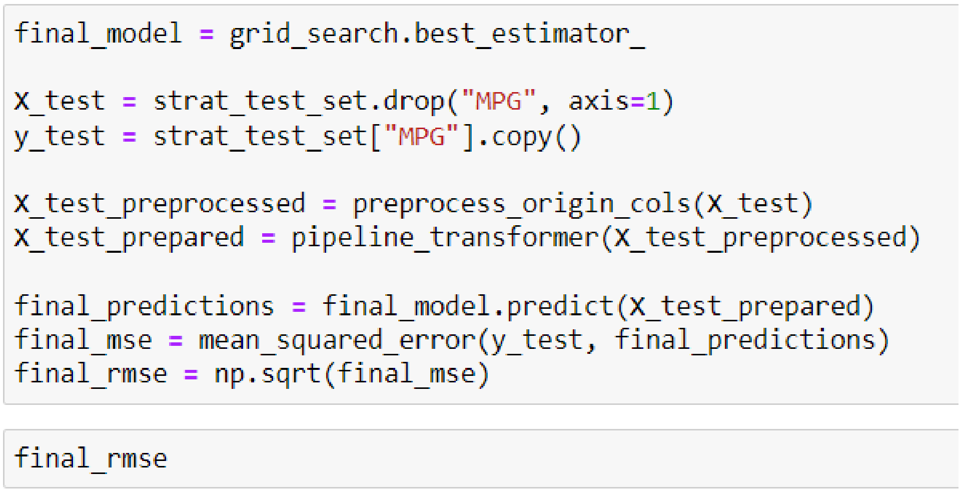
```
extra_attrs = ["acc_on_power", "acc_on_cyl"]
numerics = ['float64', 'int64']
num_attrs = list(data.select_dtypes(include=numerics))

attrs = num_attrs + extra_attrs
sorted(zip(attrs, feature_importances), reverse=True)
```

```
[('acc_on_power', 0.04799111813351087),
 ('acc_on_cyl', 0.15510242145748412),
 ('Weight', 0.20094540839561478),
 ('Model Year', 0.10224269914118438),
 ('Horsepower', 0.11677208677337549),
 ('Displacement', 0.22143108175898615),
 ('Cylinders', 0.1178959311009618),
 ('Acceleration', 0.027335128052540392)]
```

Snapshot 3.6.10

acc_on_power, which is the derived feature, has turned out to be the most important feature. One needs to keep iterating a few times before finalizing the best configuration. The model is ready with the best configuration.

3.6.11 Evaluating on Test Data



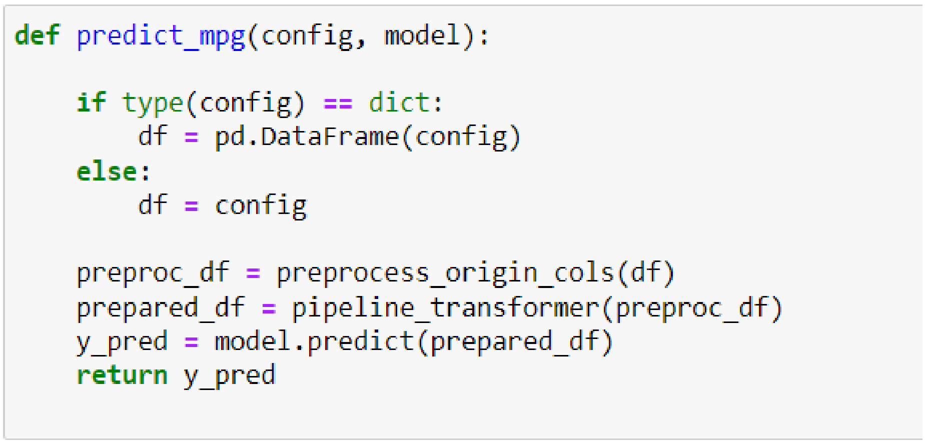
```python
final_model = grid_search.best_estimator_

X_test = strat_test_set.drop("MPG", axis=1)
y_test = strat_test_set["MPG"].copy()

X_test_preprocessed = preprocess_origin_cols(X_test)
X_test_prepared = pipeline_transformer(X_test_preprocessed)

final_predictions = final_model.predict(X_test_prepared)
final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)
```

```python
final_rmse
```

Snapshot 3.6.11

A final RMSE of 3.15 is obtained

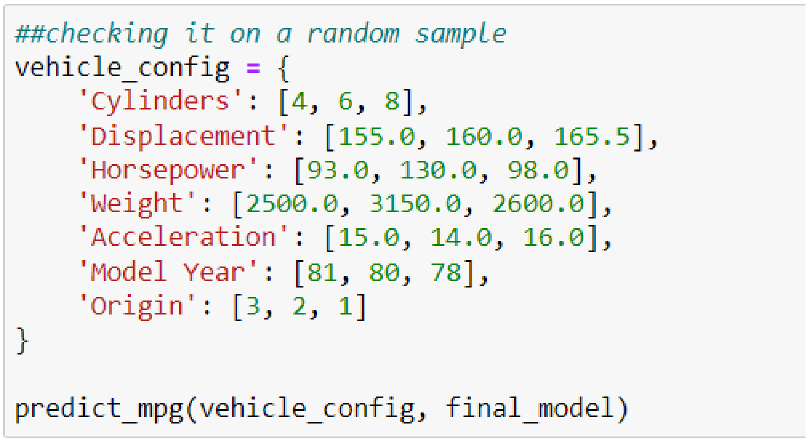
3.6.12 Creating of a function to cover the entire flow



```python
def predict_mpg(config, model):

    if type(config) == dict:
        df = pd.DataFrame(config)
    else:
        df = config

    preproc_df = preprocess_origin_cols(df)
    prepared_df = pipeline_transformer(preproc_df)
    y_pred = model.predict(prepared_df)
    return y_pred
```

Snapshot 3.6.12.1

The model is applied on a random sample data and promising results were obtained.

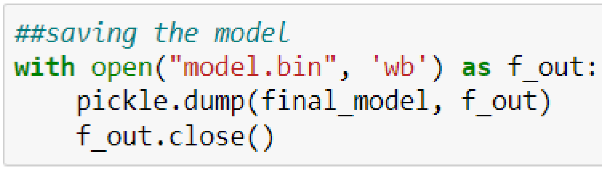


```
##checking it on a random sample
vehicle_config = {
    'Cylinders': [4, 6, 8],
    'Displacement': [155.0, 160.0, 165.5],
    'Horsepower': [93.0, 130.0, 98.0],
    'Weight': [2500.0, 3150.0, 2600.0],
    'Acceleration': [15.0, 14.0, 16.0],
    'Model Year': [81, 80, 78],
    'Origin': [3, 2, 1]
}

predict_mpg(vehicle_config, final_model)
```

Snapshot 3.6.12.2

3.5.13 Saving the Model

.Final model is ready to go into production. For deployment, the model is saved into a file using the pickle model.



```
##saving the model
with open("model.bin", 'wb') as f_out:
    pickle.dump(final_model, f_out)
    f_out.close()
```

Snapshot 3.6.13

4. Circuit Description

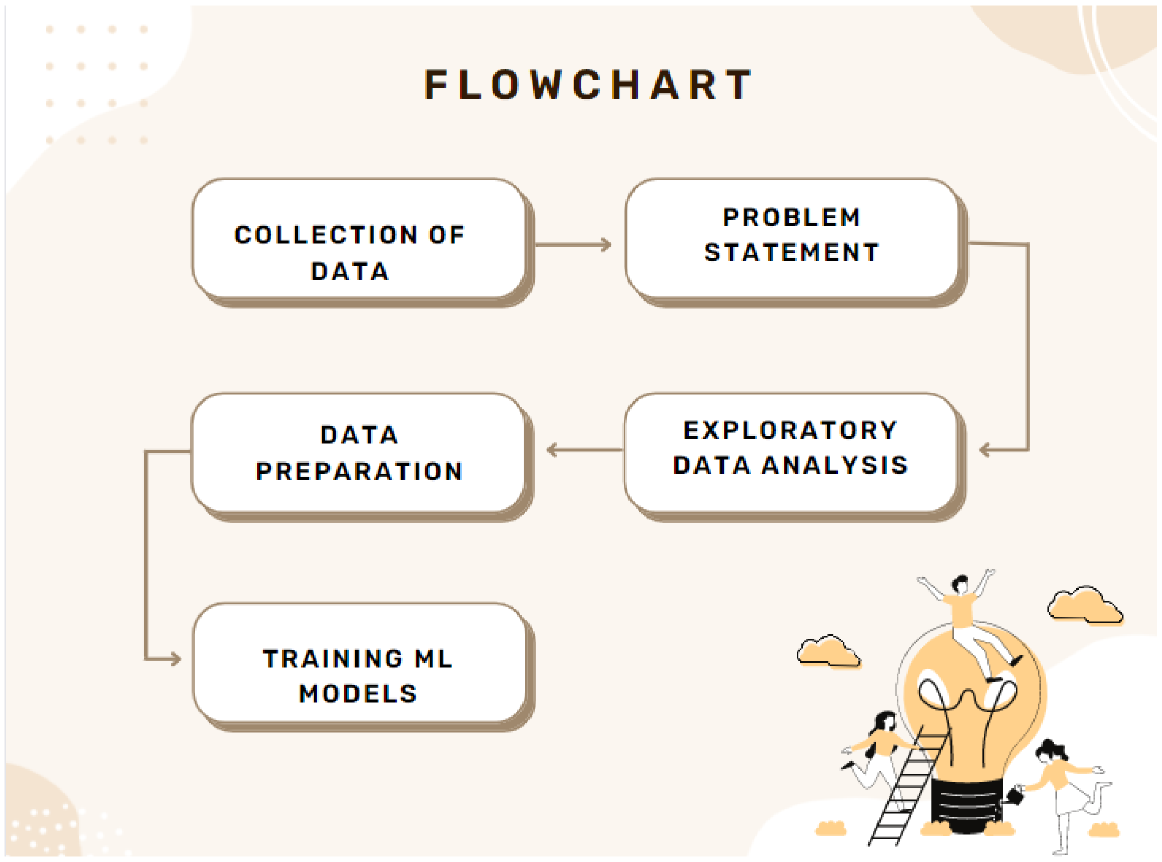


Figure 4

5. Analysis of Results

By applying different machine learning algorithms and then using K-fold validation to see which model comes out to see which model has the least when it is applied to the data. Four models were trained. Linear Regression, Decision Tree Regression and SVM Regression were found to have higher Root Mean Square Error than Random Forest Regression. Random Forest Regression was tested on the test data and showed promising results. The model was trained and saved as a pickle file for future deployment.

6. Conclusion and Future Scope

In this paper, four machine learning methods are proposed in which K-fold analysis was done and promising results were achieved. The conclusion found is that machine learning algorithms performed better in the analysis was Random Forest Regression Many researchers have previously suggested one should use ML and Deep Learning even for small dataset but the use of training and testing the Random Forest Regression alone proved effective results, which is proved in the paper. For the 11 features which were in the dataset, acceleration on power proved to be the estimator among the features and performed better in the Random Forest Regression ML approach when data is applied after preprocessing.

It was also found out that the dataset should be pre-processed otherwise, the feature estimator could have been missed. The Feature attribute was attained by attribute addition by using BaseEstimator and TransformerMixin. The training model gets overfitted sometimes and the accuracy achieved is not sufficient when a model is evaluated for real-world data problems which can vary drastically to the dataset on which the model was trained. Testing the model on the same data one trained on, is a problem. Now, one can't use the test data yet until best model is finalized and is ready to go into production. The K-fold Cross Validation. Random Forest regression came out to be the best and suited model.

For Future use cases and real-world datasets, use of Machine Learning Models and Deep learning i.e creation of a neural network with backtracking might produce promising results. The future of Fuel-Efficiency on real world big data sets looks promising since optimal results were shown by Forest Regression ML model on small datasets. An Upscaling with a similar approach, the future of fuel-efficiency prediction on real-world data looks bright.

7. Reference

1. Dan Pelleg. Scalable and Practical Probability Density Estimators for Scientific Anomaly Detection. School of Computer Science Carnegie Mellon University. 2004.

2. Qingping Tao Ph.D. Making Efficient Learning Algorithms with Exponentially many Features. Qingping Tao A DISSERTATION Faculty of The Graduate College University of Nebraska In Partial Fulfillment of Requirements. 2004.

3. Christopher R. Palmer and Christos Faloutsos. Electricity Based External Similarity of Categorical Attributes. PAKDD. 2003.

4. Dan Pelleg and Andrew W. Moore. Mixtures of Rectangles: Interpretable Soft Clustering. ICML.

5. Jinyan Li and Kotagiri Ramamohanarao and Guozhu Dong. Combining the Strength of Pattern Frequency and Distance for Classification. PAKDD. 2001.

6. Thomas Melluish and Craig Saunders and Ilia Nouretdinov and Volodya Vovk and Carol S. Saunders and I. Nouretdinov V. The typicalness framework: a comparison with the Bayesian approach. Department of Computer Science. 2001.

7. Wai Lam and Kin Keung and Charles X. Ling. PR 1527. Department of Systems Engineering and Engineering Management, The Chinese University of Hong Kong. 2001.

8. Zhi-Hua Zhou and Shifu Chen and Zhaoqian Chen. A Statistics Based Approach for Extracting Priority Rules from Trained Neural Networks. IJCNN 2000.

9. Mauro Birattari and Gianluca Bontempi and Hugues Bersini. Lazy Learning Meets the Recursive Least Squares Algorithm. NIPS. 1998.

10. D. Greig and Hava T. Siegelmann and Michael Zibulevsky. A New Class of Sigmoid Activation Functions That Don't Saturate. 1997.

11. Johannes Furnkranz. Pairwise Classification as an Ensemble Technique. Austrian Research Institute for Artificial Intelligence.