Python Project: Predicting MPG (Miles Per Gallon) on cars

Abraham Deniz

April 27, 2020

Contents

Introduction

In this report, I will present my statistical analysis of a cars data set. The data set, named *Auto-mpg*, is a data set taken from the UCI machine learning repository data sets (link: https://archive.ics.uci.edu/ml/datasets/Auto+MPG). The data set contains information about cars.

Here is an overview of the variables in the data set and their type:

Variable	Type
MPG (Miles Per Gallon)	continuous
Cylinders	discrete
Displacement	continuous
Horsepower	continuous
Weight	continuous
Acceleration	continuous
Model year	discrete
Origin	discrete
Car name	string (unique for each car)

Table 1: Overview of the variables and their types in the data set.

The data set consists of $n = 398$ observations of the $p = 9$ variables given in table 1. The problem posed on the website is to predict city-cycle fuel consumption in miles per gallon (MPG variable). It doesn't appear that any earlier work has been made. Also, it's unclear whether all input variables are relevant or not to predict MPG. Thus, it would be interesting to test feature selection methods.

In this report, $D = \{X, y\}$ will denote the data set, where y is the response vector (containing the n observed values of MPG) and X is the $n \times p$ design matrix (containing the n observed values of the variables in table 1 except for MPG). The goal is to find a suitable function $F : \mathbb{R}^{n \times p} \to \mathbb{R}^n$ that minimizes

$$(y - F(X, \theta))'(y - F(X, \theta)),$$

the sum of squares of the elements in the vector $\epsilon = y - F(X, \theta)$, where θ is the parameter vector that F depends on. If $\tilde{F}$ is such a function and if we would observe some new data X^{new} then we would predict MPG by $\tilde{F}(X^{\text{new}}, \hat{\theta})$, where $\hat{\theta}$ is an estimate of the parameter vector θ. An example of a function that could, potentially, solve the problem is $F(X, \theta = \beta) = X \cdot \beta$, where β is a parameter vector of unknown coefficients β_i of length $p = 8$. That would correspond to a linear model and the method to approximate β would be Ordinary Least Squares (OLS). The estimate of β, denoted by $\hat{\beta}$, is then

$$\hat{\beta} = (X'X)^{-1}X'y$$

and if we were to observe new data X^{new} then we would predict MPG by

$$\hat{\text{MPG}} = X^{\text{new}} \cdot \hat{\beta}$$

Since we want to use the model to predict, we want to be able to know if our predictions of MPG will be poor or not. Thus, we split the data set D into a training data set, denoted by D^{train}, and a test data set, denoted by D^{test}. In mathematical terms,

$$D = D^{\text{train}} \cup D^{\text{test}}.$$

The test set contains 98 randomly chosen observations and the training set contains the remaining 300 observations. The reason for why I chose this split is that I need a fairly large test data set in order to make the prediction tests less sensitive to random variations in the data. The training set is used to train the model for MPG and the test set is used as new data to predict MPG, using the trained model, and evaluate the predictions. Predictions are evaluated by the prediction R-squared, mean squared error (MSE) and the mean absolute error (MAE).

Methods

Since MPG is a continuous instance, I will consider different regression methods in this project. The methods include linear regression, regularized regression (Ridge, lasso, elastic net), regression trees and random forest. However, before I split the data into train and test data, I will explore the variables by making histograms, and the variables' relationships to the outcome variable MPG by looking at the correlation matrix and making scatter plots.

I will also make variable importance plots from the random forest method in order to see which of the variables described in table 1 is the most important variable(s) to predict MPG. This could be used as a feature selection method.

Here is a short description of the methods I consider.

2.1 Linear Regression

Linear regression is the most common method to use first when one wants to model a continuous instance with the help of continuous explanatory variables. In linear regression, one assumes that the outcome variable (in our case MPG) can be modelled as a linear combination of the explanatory variables and one tries to estimate the linear coefficients. Though it might not be optimal to use linear regression in many applications, especially if some of the explanatory variables are highly correlated. In our case, the sample correlations are given in the following table:

	MPG	Cylinders	Displacement	Horsepower	Weight	Acceleration	Modelyear	Origin
MPG	1	-0.775	-0.804	-0.771	-0.832	0.42	0.579	0.563
Cylinders	-0.775	1	0.951	0.839	0.896	-0.505	-0.349	-0.563
Displacement	-0.804	0.951	1	0.894	0.933	-0.544	-0.37	-0.609
Horsepower	-0.771	0.839	0.894	1	0.861	-0.684	-0.412	-0.454
Weight	-0.832	0.896	0.933	0.861	1	-0.417	-0.306	-0.581
Acceleration	0.42	-0.505	-0.544	-0.684	-0.417	1	0.288	0.206
Modelyear	0.579	-0.349	-0.37	-0.412	-0.306	0.288	1	0.181
Origin	0.563	-0.563	-0.609	-0.454	-0.581	0.206	0.181	1

Table 2: Sample correlations between the variables in the data set.

As can be seen, many of the explanatory variables are highly correlated, especially Cylinders and Displacement with a sample correlation (approximately) equal to 0.951. Thus, Linear regression will not be used as a method to find a suitable model in this project. Though, some diagnostic plots will be presented in the results section, just to make sure whether some of the basic assumptions are fulfilled or not.

2.2 Regularized/Penalized regression

The source of the problem in Linear Regression is the fact that the linear coefficients vector, β, is estimated by

$$\hat{\beta} = (X'X)^{-1}X'y.$$

If many of the explanatory variables are highly correlated then the inverse, $(X'X)^{-1}$, will be numerically unstable and the estimated coefficients will have high estimation variance. We know that $\hat{\beta}$ is an unbiased estimator of the true coefficients β. The question we now ask is if we can reduce the variance by allowing for some bias. The source of the high variance is the numerical instability of the inverse operation on $X'X$. We could stabilize this by adding something to the diagonal of $X'X$ before taking the inverse. This would result in Ridge regression and the resulting estimator would look like this:

$$\hat{\beta}_R = (X'X + r \cdot I)^{-1}X'y.$$

The value of the constant r is usually chosen through Cross Validation. When the value of r increases then the variance of $\hat{\beta}_R$ decreases and the bias of $\hat{\beta}_R$ increases. When $r = 0$ then Ridge regression is the same as regular linear regression.

The least squares criterion is written as

$$\min_{\beta}(y - X\beta)'(y - X\beta) = \min_{\beta} ||y - X\beta||^2 = \min_{\beta} \sum_{i=1}^{n}(y_i - x_i \cdot \beta)^2.$$

The penalized least squares problem related to ridge regression can be written as

$$\min_{\beta}(y - X\beta)^{'}(y - X\beta) \text{ subject to } ||\beta||^2 \leq \tau.$$

What the problem states is that we want to minimize $(y - X\beta)^{'}(y - X\beta)$ but we don't want the sum of the estimates coefficients (squared) to be too large, i.e. we penalize the magnitudes of the $\hat{\beta}$'s. This constraint can be achieved if all β's are small or perhaps if one is big but the rest are very small, and of course anything in between these extremes. How do we solve such constrained optimization problems? We use Lagrangian methods, replacing the constraint with an added penalty:

$$\min_{\beta}(y - X\beta)^{'}(y - X\beta) + r\beta^{'}\beta.$$

We solve this penalized problem for different values of r until the constraint $\beta^{'}\beta \leq \tau$ is fulfilled. If $r = 0$ then we solve the problem. If the constraint is fulfilled for this solution, we are done. If not, we increase r and solve again. If the constraint is fulfilled then we're done, otherwise we increase r again. The optimal solution to the constrained optimization problem is the one with the smallest r that leads to a solution that satisfies the constraint. How do we solve the penalized problem? Let's take the derivative with respect to each β_j and set it to 0:

$$-2X^{'}(y - X\beta) + 2r\beta = 0$$

which we can solve for β:

$$\hat{\beta} = (X^{'}X + r \cdot I)^{-1}X^{'}y = \hat{\beta}_R.$$

So, the ridge estimator is actually identical to the penalized regression estimate with a constraint on the sum of squares of the regression coefficients. If one were to penalize with the L1-norm instead of the L2-norm then the penalized regression would be called Lasso instead of Ridge. But one could also penalize using both the L1-norm and the L2-norm. That would correspond to elastic net.

2.3 Regression Trees

Regression trees is like a parlour game of "Twenty Questions". The questions you can ask are related to thresholds on the variables: e.g. "Is the observed value of variable x_j less than or equal to 2, or greater than 2?". This kind of question is called a "split". Depending on which side of the split an observation i falls, we will attribute a constant fitted value. There will be one fitted value for all the observations for which the answer to the above question is "yes", and similarly for all the observations for which the answer is "no". The gain of the split is measured in terms of RSS (Residual Sum of Squares). before the split, the RSS is

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

After the split, let's say n_1 observations answered "yes", and n_2 observations answered "no". The RSS has now been reduced to

$$\sum_{i:x_{ij}\leq 2}(y_i - \hat{\mu}_1)^2 + \sum_{i:x_{ij}>2}(y_i - \hat{\mu}_2)^2$$

where

$$\hat{\mu}_1 = \sum_{i:x_{ij}\leq 2} y_i/n_1, \quad \hat{\mu}_2 = \sum_{i:x_{ij}>2} y_i/n_2.$$

For each branch we can now ask a new question to refine the predictions. Again, this leads to a reduction in RSS. The combination of splits, e.g. $\{X_1 \leq 2\} \cap \{X_3 > 5\} \cap \{X_8 \leq 2.3\}$, constitute a rectangle in, here, 3 dimensional space. This example rectangle corresponds to asking 3 questions. For each rectangle, we calculate the mean value of the observations for which the answer to each question is "yes". That is the corresponding prediction.

The results of the splits are illustrated using a "Tree", where each split is a node and the branches correspond to the different answers. The length of the branches are drawn proportional to the reduction in RSS. At the bottom of the tree we have the so-called "leaves" (a high-dimensional rectangle) for which we calculate the predicted value from the mean of the observations corresponding to "yes" answers.

2.4 Random Forest

Random forest is a generalization of the regression trees. The idea with random forest is to grow many (B) trees and then use the average of all trees to predict. The essential idea is to average many noisy but approximately unbiased models, and hence reduce the variance. Trees are ideal candidates for bagging, since they can capture complex interaction structures in the data, and if grown sufficiently deep, have relatively low bias. Since trees are notoriously noisy, they benefit greatly from the averaging. Moreover, since each tree generated in bagging is identically distributed (i.d.), the expectation of an average of B such trees is the same as the expectation of any one of them. This means the bias of bagged trees is the same as that of the individual (bootstrap) trees, and the only hope of improvement is through variance reduction.

2.4.1 Random Forest Algorithm

1. For $b = 1, \ldots, B$:
   (a) Draw a bootstrap sample of size N from the training data. The non-sampled data is called the Out-of-bag sample, OOB_b.
   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size $n_{\min}$ is reached:
      i. Select m variables at random from the p variables.
      ii. Pick the best variable/split-point among the m.
      iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_{b=1}^{B}$.

To make a prediction at a new point x:

$$\hat{f}_{\text{rf}}^{B}(x) = \frac{1}{B} \cdot \sum_{b=1}^{B} T_b(x).$$

2.5 Handling the missing data

Unfortunately, the data set has some missing values. More specifically, the variable Horsepower has six missing values. Since I can't find the reason to why the missing data is missing, I have to either remove the data of the other variables at those missing points or impute values at those missing points. I chose to impute values. If one chooses to impute the missing values, the ideal value to impute is a value drawn from the underlying distribution of that variable. Let's take a look at some summary statistics of the variable Horsepower using the non-missing data:

Mean	Standard Deviation	Min	25th percentile	Median	75th percentile	Max
104.47	38.49	46	75	93.5	126	230

Table 3: Summary Statistics for the variable Horsepower.

As we can see, the distribution of Horsepower ranges from 46 to 230. One other thing to note (that isn't shown here) is that Horsepower only takes on integer values. Since there is no theoretical distribution that can be described with these summary statistics, I chose to impute the mean of the distribution, rounded to the nearest integer. The downside of imputing the mean of the non-missing data is that the spread (measured by the variance and/or standard deviation) is not accounted for.

Results

3.1 Histograms of the continuous variables

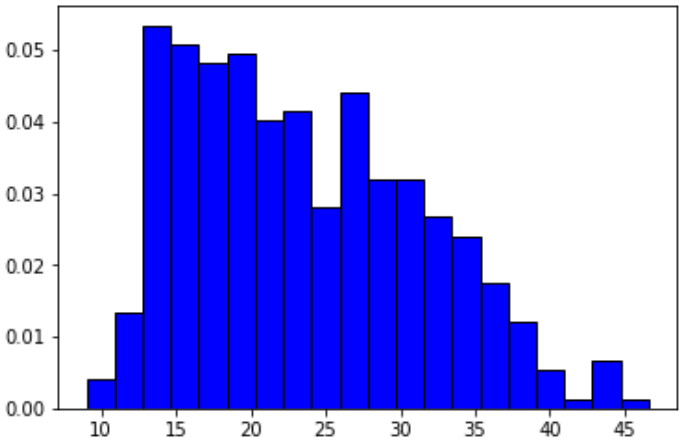

Figure 1: Histogram of MPG. This looks like a triangular-distribution with $a = 9$, $c = 13$ and $b = 46$.

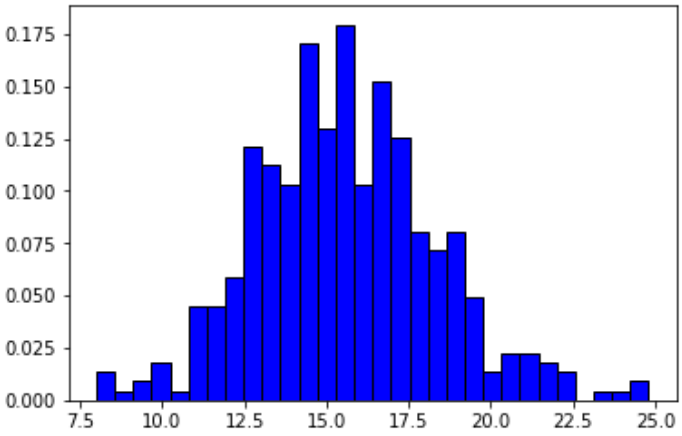

Figure 2: Histogram of acceleration. This looks like a truncated normal distribution with mean 15.5 and standard deviation 2.75, truncated to the interval [7.5 , 25].

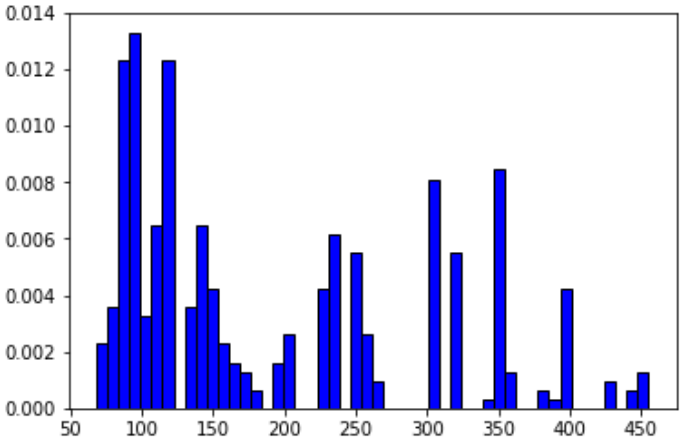

Figure 3: Histogram of displacement. This doesn't look like any of the regular probability distributions.

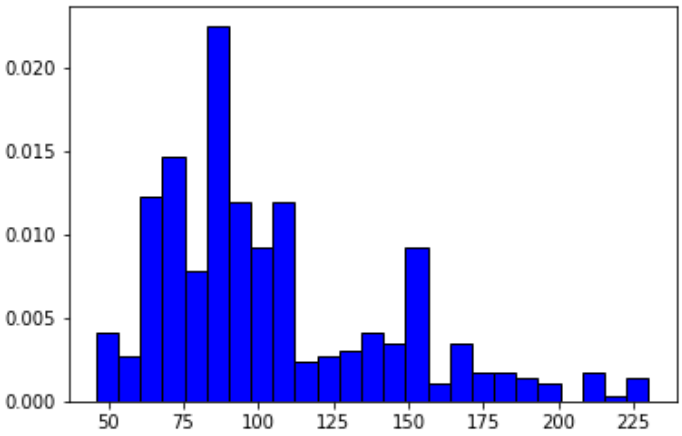

Figure 4: Histogram of horsepower. This doesn't look like any of the regular probability distributions.

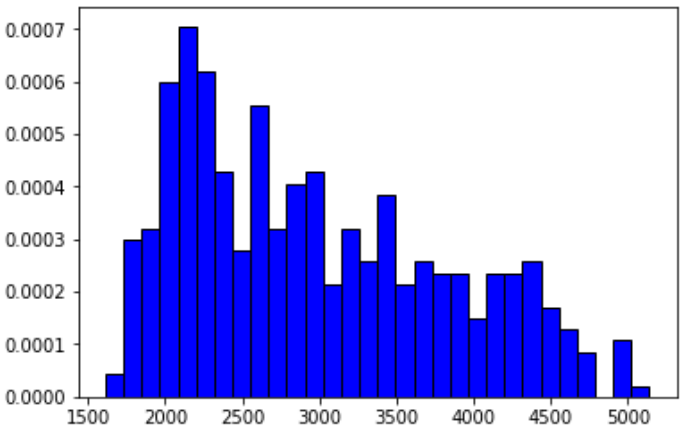

Figure 5: Histogram of weight. This doesn't look like any of the regular probability distributions.

3.2 Scatter plots between MPG and continuous variables

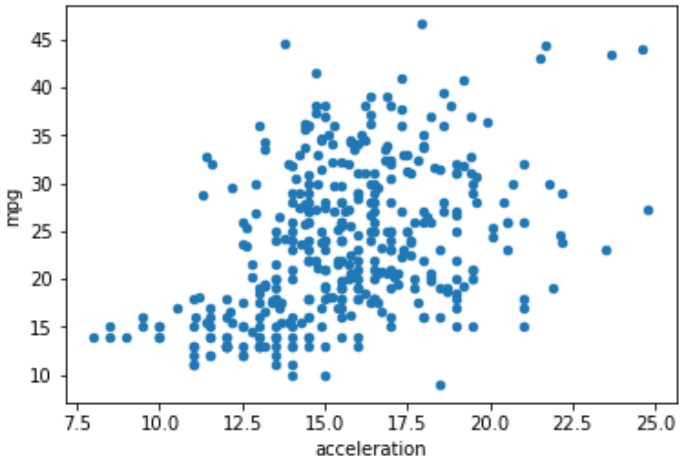

Figure 6: Scatter plot between MPG and acceleration.

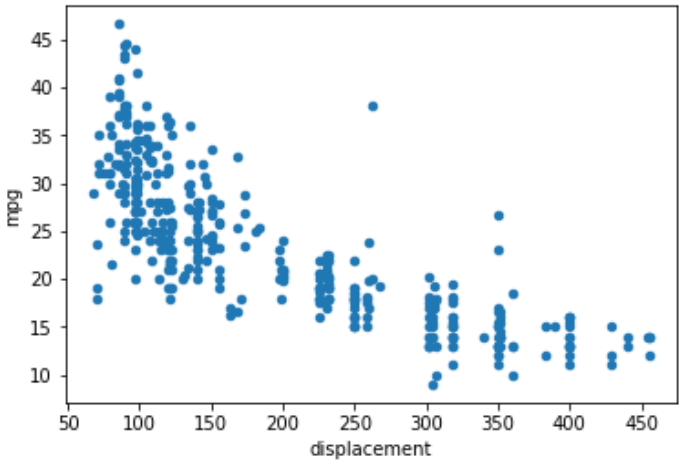

Figure 7: Scatter plot between MPG and displacement.

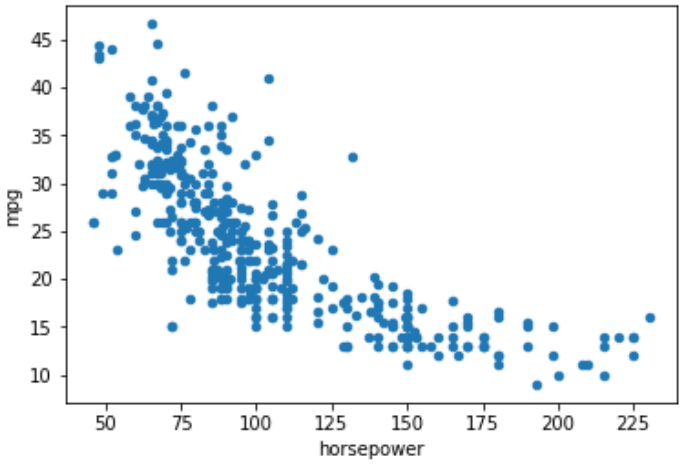

Figure 8: Scatter plot between MPG and horsepower.

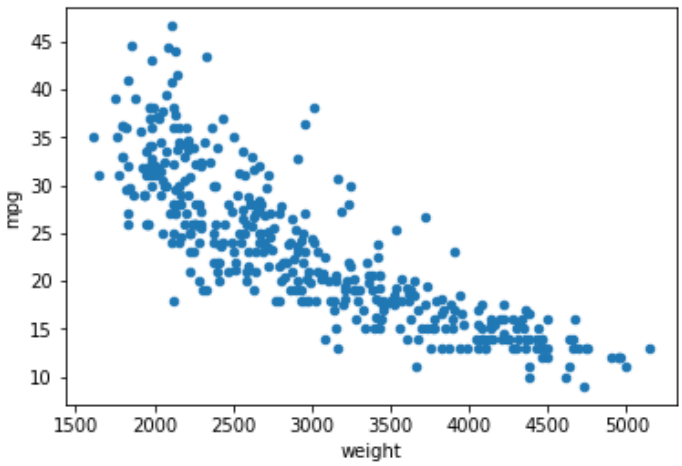

Figure 9: Scatter plot between MPG and weight.

3.3 Linear Regression

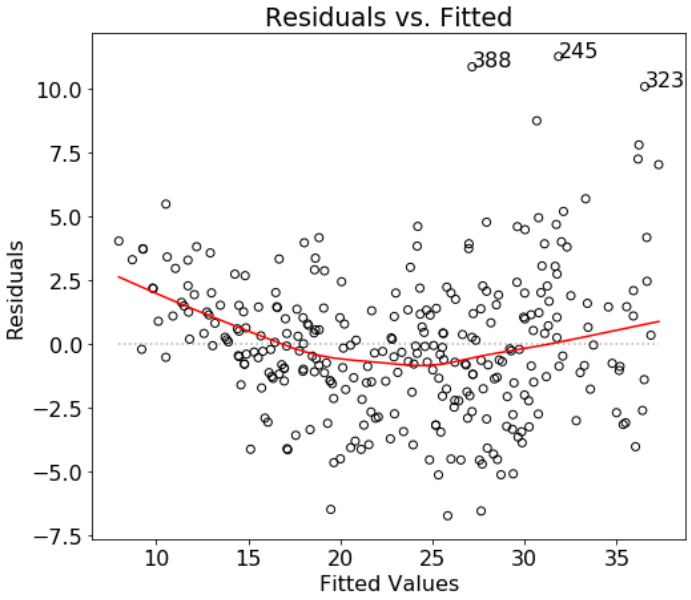

Figure 10: Residuals vs fitted values diagnostic plot.

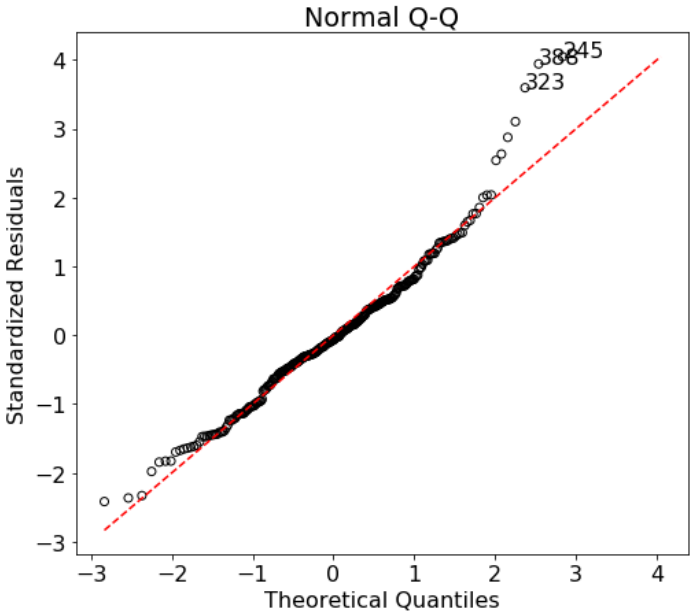

Figure 11: Normal QQ-plot.

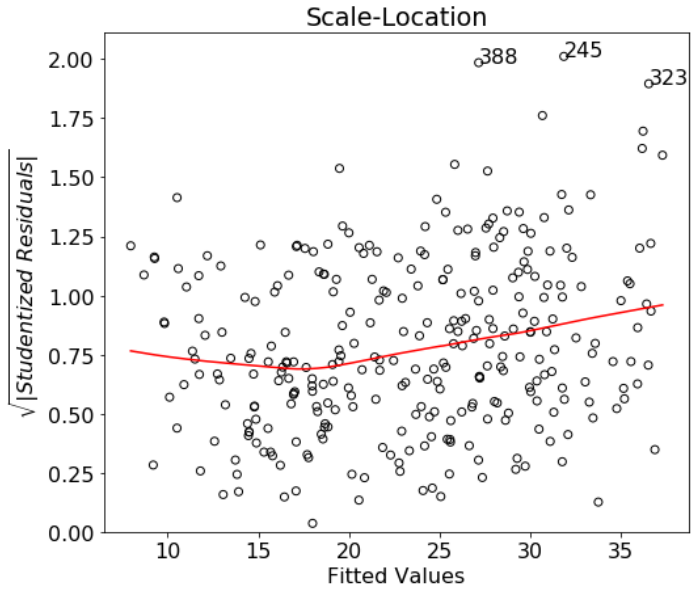

Figure 12: Scale-Location diagnostic plot.

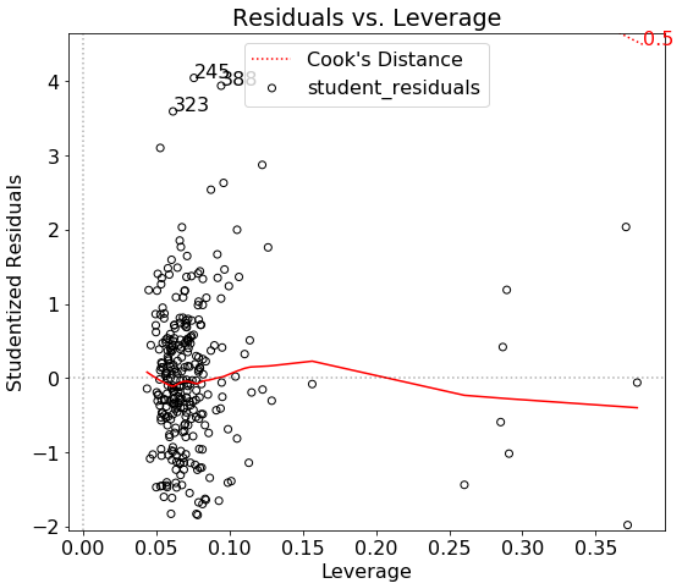

Figure 13: Residuals vs Leverage diagnostic plot.

From the residuals vs fitted values plot, I conclude that there is no trend in the residuals and that the errors are symmetrical around 0. However, we can clearly see three observations that might be potential outliers, namely observation 388, 245 and 323. The QQ-plot indicates that the errors are normally distributed, since most of the observations lie in a straight line. The Scale-Location plot indicates that the errors have constant variance since there is no trend shown in the plot. In the Residuals vs Leverage plot, I can see that the three potential outliers (observations 388, 245 and 323) have high leverage (approximately between 3.5 and 4), and thus conclude that these observations are outliers. Though, since I can't find out the reason that these three observations are outliers, I will not drop them.

The adjusted R-squared (also known as the coefficient of determination) is equal to 0.856, which means that 85.6% of the variance in MPG can be explained by the regression on the other variables. This indicates that the assumed linear model is reasonably sufficient. The mean absolute error is ≈ 2.37 and the mean-squared error is ≈ 8.93. The prediction R-squared is ≈ 0.8503, which is pretty high and indicates that the linear model is good to predict new observations.

3.4 Regression Trees

By using regression trees, the mean absolute error is ≈ 2.93 and the mean squared error is ≈ 17.44 and the prediction R-squared is ≈ 0.7075. These fit measurements indicates that a tree-type model is worse at predicting compared to a linear model.

3.5 Random Forest

By using random forest, the mean absolute error is ≈ 2.13 and the mean squared error is ≈ 8.57 and the prediction R-squared is ≈ 0.8562. This means that the random forest model works better at predicting than the linear model.

3.5.1 Random Forest importance

The importance of the variables are given in this table:

Variable	Importance
Displacement	0.39
Weight	0.22
Horsepower	0.14
Cylinders_4	0.08
Modelyear_{80}	0.04
Acceleration	0.03
Modelyear_{82}	0.02
Origin_2	0.01
Origin_3	0.01
Modelyear_{73}	0.01
Modelyear_{79}	0.01
Modelyear_{81}	0.01

Table 4: The remaining variables not listed has no importance. The discrete variables with an index indicates that the variable is equal to the value of the index, e.g. Modelyear_{81} means that the model year of the car is 1981.

3.6 Ridge Regression

Now we wonder if we can obtain a better linear model by using ridge regression instead of linear regression. The optimal value for r is chosen through 5-fold cross-validation and was equal to 1. Using ridge regression with $r = 1$, the mean absolute error is ≈ 2.31, the mean squared error is ≈ 8.596 and the prediction R-squared is ≈ 0.8558. This means that the linear model with ridge regression is slightly better at predicting new observations than the linear model with linear regression.

3.7 Lasso Regression

By penalizing with the L1-norm instead of the L2-norm, can we obtain an even better linear model? The optimal value for r is chosen through 5-fold cross-validation and was equal to 0.01. Using Lasso regression with $r = 0.01$, the mean absolute error is ≈ 2.33, the mean squared error is 8.73 and the prediction R-squared is ≈ 0.8536. This means that the linear model with ridge regression is slightly better at predicting new observations than the linear model with Lasso regression.

Discussion

The goal of this project was to see if we can fit a model to the auto-mpg data set and use this model to predict MPG on unobserved cars (by unobserved, I refer to cars that are not included in the data set). By taking a first look at the scatter plots between MPG and the explanatory variables, I can see an almost-linear relationship between MPG and displacement/horsepower/weight. Thus, it makes sense to fit a linear model to this data set. Maybe some transformation could help make the relationship between MPG and acceleration more linear, but then the interpretation of the model would be different.

By looking at the fit measurements (the mean absolute error, the mean squared error and the predicted R-squared) that is calculated with the test data, I conclude that the linear model works pretty well at predicting MPG for unobserved cars. The tree model performed the worst at predicting MPG according to the fit measurements, but by using a random forest model (which predicts with an average of many tree models), the prediction was better than the predictions from the linear model. This was expected, since the correlations shown in table 2 indicates that the estimated parameters for the linear model have high estimation variance and the estimation variance is decreased when predicting with an average of many models, which is exactly what random forest does.

Considering that we have a multi collinearity problem according to table 2 (since the explanatory variables are highly correlated), the estimates $\hat{\beta}_j$ of β_j have high estimation variance. Because of this, I choosed to perform penalized regression (Ridge and Lasso) to obtain a linear model that is better at predicting MPG. According to the result, the linear model obtained through Ridge regression was best at predicting compared to the regular linear model and the Lasso linear model, though the random forest model is still the best model to predict MPG. The final linear model, that was obtained through Ridge regression, is given here:

$$\begin{aligned}\hat{\text{MPG}} &= 0.0177361556 \cdot \text{displacement} - 0.0407755874 \cdot \text{horsepower} - 0.00531136888 \cdot \text{weight} - 0.0131940632 \cdot \text{acceleration} \\ &-3.80503446 \cdot \mathbb{1}_{\text{cylinders}=3} + 2.44404451 \cdot \mathbb{1}_{\text{cylinders}=4} + 1.33760377 \cdot \mathbb{1}_{\text{cylinders}=5} - 0.512445802 \cdot \mathbb{1}_{\text{cylinders}=6} + 0.535831976 \cdot \mathbb{1}_{\text{cylinders}=8} \\ &-1.33253081 \cdot \mathbb{1}_{\text{origin}=1} + 0.240197978 \cdot \mathbb{1}_{\text{origin}=2} + 1.09233283 \cdot \mathbb{1}_{\text{origin}=3} - 2.72807005 \cdot \mathbb{1}_{\text{modelyear}=1970} - 2.30074137 \cdot \mathbb{1}_{\text{modelyear}=1971} \\ &-3.51766134 \cdot \mathbb{1}_{\text{modelyear}=1972} - 3.60758313 \cdot \mathbb{1}_{\text{modelyear}=1973} - 1.29429264 \cdot \mathbb{1}_{\text{modelyear}=1974} - 1.85348637 \cdot \mathbb{1}_{\text{modelyear}=1975} \\ &-1.39489380 \cdot \mathbb{1}_{\text{modelyear}=1976} - 0.140250017 \cdot \mathbb{1}_{\text{modelyear}=1977} - 0.288866217 \cdot \mathbb{1}_{\text{modelyear}=1978} + 2.53594846 \cdot \mathbb{1}_{\text{modelyear}=1979} \\ &+6.72863537 \cdot \mathbb{1}_{\text{modelyear}=1980} + 3.43032674 \cdot \mathbb{1}_{\text{modelyear}=1981} + 4.43093439 \cdot \mathbb{1}_{\text{modelyear}=1982},\end{aligned}$$

where $\mathbb{1}_A$ is the indicator function for an event A (oddly enough, the output didn't include an intercept).

According to the variable importance that are given in table 4, the most important variables to predict MPG is Displacement, Weight, Horsepower and Cylinders (at least when Cylinders=4). Interestingly, these variables had the highest correlation with MPG according to table 2.

Something that I regret that I didn't check for was if there were any interactions. However, the fact that the regression tree model was worse at predicting compared to the linear model indicates that there aren't any interactions, but to be completely sure, one would have to produce Coplots. Another thing I regret that I didn't try was if a generalized linear model would fit the data. However, since the histogram of MPG showed that MPG seems to follow a triangular distribution, it's highly doubtful that a generalized linear model would fit for this data set, since the triangular distribution is not included in the exponential family distributions.

To conclude the discussion, the random forest model is the best model for this data set in terms of the fit measurements. However, the linear model produced from the ridge regression is also a good model.

Appendix for Python Programming Code

Note: In order to use this Python code, one first needs to import the data set as an numpy array named autompgdata. Otherwise, this code won't work.

```
# Import all the packages that is needed for the project
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from collections import Counter
#import seaborn as sns
from scipy import stats
#from random import sample


#from sklearn import datasets, linear_model
#from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
#import sklearn as sk
from statsmodels.nonparametric.smoothers_lowess import lowess
#from matplotlib import rcParams
import statsmodels.formula.api as sm
from sklearn import metrics
from sklearn import tree
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
```

```
# First, import the dataset as an numpy array named autompgdata
df=autompgdata[:,0:8]
data=pd.DataFrame(data=df)
n,p=data.shape
data.columns=["mpg","cylinders","displacement","horsepower","weight","acceleration","modelyear","origin"]
data1 = data.apply(pd.to_numeric, errors='coerce')
ind=np.arange(1, n+1)

def which(self):
    try:
        self = list(iter(self))
    except TypeError as e:
        raise Exception("""'which' method can only be applied to iterables.
        {}""".format(str(e)))
    indices = [i for i, x in enumerate(self) if bool(x) == True]
    return(indices)

data1.horsepower[which(data.horsepower!="?")].describe() # Summary Statistics of Horsepower
data1.horsepower[which(data.horsepower=="?")]=round(data1.horsepower[which(data.horsepower!="?")].mean())
data2=data1.set_index(ind)
```

```
# Histograms
plt.hist(data2['horsepower'], density=True, bins=25, color='blue', edgecolor='black')
#plt.hist(data2['cylinders'], density=True, bins=25, color='blue', edgecolor='black')
plt.hist(data2['mpg'], density=True, bins=20, color='blue', edgecolor='black')
plt.hist(data2['displacement'], density=True, bins=50, color='blue', edgecolor='black')
plt.hist(data2['weight'], density=True, bins=30, color='blue', edgecolor='black')
plt.hist(data2['acceleration'], density=True, bins=30, color='blue', edgecolor='black')
```

```
# Scatter plots
data2.plot(kind='scatter', y='mpg', x='displacement', color='yellow')
data2.plot(kind='scatter', y='mpg', x='weight')
data2.plot(kind='scatter', y='mpg', x='horsepower')
data2.plot(kind='scatter', y='mpg', x='acceleration')
```

```
# Counter is the same as table in R
Counter(data1['cylinders'])
Counter(data1['origin'])
Counter(data1['modelyear'])
```

```
# Dealing with the the variables cylinders, origin and modelyear
data2=pd.get_dummies(data=data2, columns=['cylinders', 'origin', 'modelyear'])
```

```
# Splitting dataset into training data and testing data
y=data2.mpg
data2=data2.drop('mpg', axis=1)
X_train, X_test, y_train, y_test = train_test_split(data2, y, test_size=98/398)
correlationmatrix=data1.corr()
```

```
# Linear regression
model=sm.OLS(y_train, X_train)
results=model.fit()
print(results.summary())
```

```
# Diagnostic plots: First residuals vs fitted values plot
residuals_OLS = results.resid
fitted_OLS = results.fittedvalues
smoothed = lowess(residuals_OLS,fitted_OLS)
top3 = abs(residuals_OLS).sort_values(ascending = False)[:3]

plt.rcParams.update({'font.size': 16})
plt.rcParams["figure.figsize"] = (8,7)
fig, ax = plt.subplots()
ax.scatter(fitted_OLS, residuals_OLS, edgecolors = 'k', facecolors = 'none')
ax.plot(smoothed[:,0],smoothed[:,1],color = 'r')
ax.set_ylabel('Residuals')
ax.set_xlabel('Fitted Values')
ax.set_title('Residuals vs. Fitted')
ax.plot([min(fitted_OLS),max(fitted_OLS)],[0,0],color = 'k',linestyle = ':', alpha = .3)
for i in top3.index:
    ax.annotate(i,xy=(fitted_OLS[i],residuals_OLS[i]))
plt.show()
```

```
# 2nd diagnostic plot: QQ-plot
sorted_student_residuals_OLS = pd.Series(results.get_influence().resid_studentized_internal)
sorted_student_residuals_OLS.index = results.resid.index
sorted_student_residuals_OLS = sorted_student_residuals_OLS.sort_values(ascending = True)
df_OLS = pd.DataFrame(sorted_student_residuals_OLS)
df_OLS.columns = ['sorted_student_residuals']
df_OLS['theoretical_quantiles'] = stats.probplot(df_OLS['sorted_student_residuals'], dist = 'norm',
fit = False)[0]
rankings = abs(df_OLS['sorted_student_residuals']).sort_values(ascending = False)
top3 = rankings[:3]

fig, ax = plt.subplots()
```

```
x = df_OLS['theoretical_quantiles']
y = df_OLS['sorted_student_residuals']
ax.scatter(x,y, edgecolor = 'k',facecolor = 'none')
ax.set_title('Normal Q-Q')
ax.set_ylabel('Standardized Residuals')
ax.set_xlabel('Theoretical Quantiles')
ax.plot([np.min([x,y]),np.max([x,y])],[np.min([x,y]),np.max([x,y])], color = 'r', ls = '--')
for val in top3.index:
    ax.annotate(val,xy=(df_OLS['theoretical_quantiles'].loc[val],
    df_OLS['sorted_student_residuals'].loc[val]))
plt.show()
```

```
# 3rd diagnostic plot: Scale-location plot
student_residuals_OLS = results.get_influence().resid_studentized_internal
sqrt_student_residuals_OLS = pd.Series(np.sqrt(np.abs(student_residuals_OLS)))
sqrt_student_residuals_OLS.index = results.resid.index
smoothed = lowess(sqrt_student_residuals_OLS,fitted_OLS)
top3 = abs(sqrt_student_residuals_OLS).sort_values(ascending = False)[:3]

fig, ax = plt.subplots()
ax.scatter(fitted_OLS, sqrt_student_residuals_OLS, edgecolors = 'k', facecolors = 'none')
ax.plot(smoothed[:,0],smoothed[:,1],color = 'r')
ax.set_ylabel('$\sqrt{|Studentized \ Residuals|}$')
ax.set_xlabel('Fitted Values')
ax.set_title('Scale-Location')
ax.set_ylim(0,max(sqrt_student_residuals_OLS)+0.1)
for i in top3.index:
    ax.annotate(i,xy=(fitted_OLS[i],sqrt_student_residuals_OLS[i]))
plt.show()
```

```
# 4th diagnostic plot: residuals vs leverage plot
student_residuals_OLS = pd.Series(results.get_influence().resid_studentized_internal)
student_residuals_OLS.index = results.resid.index
df_OLS = pd.DataFrame(student_residuals_OLS)
df_OLS.columns = ['student_residuals']
df_OLS['leverage'] = results.get_influence().hat_matrix_diag
smoothed = lowess(df_OLS['student_residuals'],df_OLS['leverage'])
sorted_student_residuals = abs(df_OLS['student_residuals']).sort_values(ascending = False)
top3 = sorted_student_residuals[:3]

fig, ax = plt.subplots()
x = df_OLS['leverage']
y = df_OLS['student_residuals']
xpos = max(x)+max(x)*0.01
```

```
ax.scatter(x, y, edgecolors = 'k', facecolors = 'none')
ax.plot(smoothed[:,0],smoothed[:,1],color = 'r')
ax.set_ylabel('Studentized Residuals')
ax.set_xlabel('Leverage')
ax.set_title('Residuals vs. Leverage')
ax.set_ylim(min(y)-min(y)*0.15,max(y)+max(y)*0.15)
ax.set_xlim(-0.01,max(x)+max(x)*0.05)
plt.tight_layout()
for val in top3.index:
    ax.annotate(val,xy=(x.loc[val],y.loc[val]))

cooksx = np.linspace(min(x), xpos, 50)
p = len(results.params)
poscooks1y = np.sqrt((p*(1-cooksx))/cooksx)
poscooks05y = np.sqrt(0.5*(p*(1-cooksx))/cooksx)
negcooks1y = -np.sqrt((p*(1-cooksx))/cooksx)
negcooks05y = -np.sqrt(0.5*(p*(1-cooksx))/cooksx)

ax.plot(cooksx,poscooks1y,label = "Cook's Distance", ls = ':', color = 'r')
ax.plot(cooksx,poscooks05y, ls = ':', color = 'r')
ax.plot(cooksx,negcooks1y, ls = ':', color = 'r')
ax.plot(cooksx,negcooks05y, ls = ':', color = 'r')
ax.plot([0,0],ax.get_ylim(), ls=":", alpha = .3, color = 'k')
ax.plot(ax.get_xlim(), [0,0], ls=":", alpha = .3, color = 'k')
ax.annotate('1.0', xy = (xpos, poscooks1y[-1]), color = 'r')
ax.annotate('0.5', xy = (xpos, poscooks05y[-1]), color = 'r')
ax.annotate('1.0', xy = (xpos, negcooks1y[-1]), color = 'r')
ax.annotate('0.5', xy = (xpos, negcooks05y[-1]), color = 'r')
ax.legend()
plt.show()
```

```
# Fit measuremenmts for the linear regression
y_predicted_OLS=results.predict(X_test)
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_predicted_OLS))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_predicted_OLS))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_predicted_OLS)))
predR2_OLS=metrics.r2_score(y_test, y_predicted_OLS)
```

```
# CART (Regression trees)
classifier=tree.DecisionTreeRegressor()
clf=classifier.fit(X_train,y_train)
predicted_values_CART=clf.predict(X=X_test)
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, predicted_values_CART))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, predicted_values_CART))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, predicted_values_CART)))
predR2_CART=metrics.r2_score(y_test, predicted_values_CART)
```

```
# Random forest
# Instantiate model with 1000 decision trees
rf = RandomForestRegressor(n_estimators = 1000, oob_score=True)
# Train the model on training data
forest=rf.fit(X_train, y_train);
# Use the forest's predict method on the test data
rf_predictions = rf.predict(X_test)
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, rf_predictions))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, rf_predictions))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, rf_predictions)))
predR2_RF=metrics.r2_score(y_test, rf_predictions)
```

```
# Get numerical feature importances
importances = list(rf.feature_importances_)
# List of tuples with variable and importance
feature_importances = [(feature, round(importance, 2))
for feature, importance in zip(data2.columns, importances)]
# Sort the feature importances by most important first
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse = True)
# Print out the feature and importances
[print('Variable: {:20} Importance: {}'.format(*pair)) for pair in feature_importances];
```

```
# Ridge Regression
mm=Ridge()
parameters={'alpha': [1e-15, 1e-10, 1e-8, 1e-4, 1e-3, 1e-2, 1, 2, 5, 10]}
ridge_regressor=GridSearchCV(mm, parameters, scoring='neg_mean_squared_error', cv=5)
ridge_regressor.fit(X_train, y_train)
print(ridge_regressor.best_params_) # Best alpha
print(ridge_regressor.best_score_)
rr=Ridge(alpha=ridge_regressor.best_params_['alpha'])
result_ridge=rr.fit(X_train, y_train)
print(result_ridge.coef_)
ridge_predict=result_ridge.predict(X_test)
# fit measurements with ridge regression
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, ridge_predict))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, ridge_predict))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, ridge_predict)))
```

```
predR2_ridge=metrics.r2_score(y_test, ridge_predict)
```

```
# Lasso Regression
lasso=Lasso()
parameters={'alpha': [1e-15, 1e-10, 1e-8, 1e-4, 1e-3, 1e-2, 1, 2, 5, 10]}
lasso_regressor=GridSearchCV(lasso, parameters, scoring='neg_mean_squared_error', cv=5)
lasso_regressor.fit(X_train, y_train)
print(lasso_regressor.best_params_) # Best alpha
print(lasso_regressor.best_score_) # Lowest possible MSE with Lasso Regression and the best alpha
ll=Lasso(alpha=lasso_regressor.best_params_['alpha'])
result_lasso=ll.fit(X_train, y_train)
lasso_predict=result_lasso.predict(X_test)
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, lasso_predict))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, lasso_predict))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, lasso_predict)))
predR2_lasso=metrics.r2_score(y_test, lasso_predict)
```