Data Analysis using Statistical Methods: Case Study of Predicting Price of Automobile.

Abstract

With the aid of statistical analysis, enormous amounts of data may be gathered, analyzed, and turned into useful information by spotting common patterns and trends. The idea is to employ a dataset on which different statistical techniques can be applied in order to make precise predictions. I try to find out the best suitable model of predicting price of automobiles.

CHAPTER 1: Introduction

The goal of this project is to put into practice the statistical techniques that were learnt during the Statistical Methods course to analyze actual data. This includes and is not limited to Z-test, Shapiro-Wilk Test, Chi Squared Tests of Independence various distributions, multivariate normality test, categorical analysis of data, Kruskal-Wallis H-test and regression models. To achieve this, we will use the 1985 Ward's Automotive Yearbook dataset. First, I will implement various statistical methods to observe correlations and dependency between the various features of the data, both numerical and categorical. Going ahead, I will be implementing Basic Linear Regression, Forward and Backward elimination methods to select the best features from our data to fit a linear regression model, Principal Component Regression and Polynomial regression. I will test the accuracy of these linear regression models and other parameterized regression models on our test data.

CAHPTER 2: Methodology

- Shapiro-Wilk Test: It tests the null hypothesis that a sample x1, ..., xn came from a normally distributed population.
- Z test: A z-test is a statistical test used to determine whether two population means are different when the variances are known and the sample size is large.
- Chi-Square Test: It is a nonparametric independence hypothesis test. We can use it to test whether two categorical variables are related to each other.
- Kruskal-Wallis H-test: The Kruskal–Wallis test by ranks or one-way ANOVA on ranks is a non-parametric method for testing whether samples originate from the same distribution.
- Nemenyni Test: It is a post-hoc test intended to find the groups of data that differ after a global statistical test has rejected the null hypothesis that the performance of the comparisons on the groups of data is similar. The test makes pair-wise tests of performance.
- Linear Regression: It is a linear approach for modelling the relationship between a scalar response and one or more explanatory variables (also known as dependent and independent variables). The case of one explanatory variable is called simple linear regression; for more than one, the process is called multiple linear regression. This term is distinct from multivariate linear regression, where multiple correlated dependent variables are predicted, rather than a single scalar variable.
- Forward/Backward Sampling: Forward selection starts with model containing null predictions, then add features to the model one at a time, and then predict and compare the performances with each added features, upto the point all the features are into the model. Backward selection starts with the model containing all the features and the performance being compared after taking the least useful feature out one at a time.
- Ridge Regression: Ridge regression is a method of estimating the coefficients of multiple-regression models in scenarios where the independent variables are highly correlated.
- Principal Component Analysis: Principal component analysis (PCA) is a popular technique for analyzing large datasets containing a high number of dimensions/features per observation, increasing the interpretability of data while preserving the maximum amount of information, and enabling the visualization of multidimensional data. Formally, PCA is a statistical technique for reducing the dimensionality of a dataset.
- Polynomial Regression: It is a form of regression analysis in which the relationship between the independent variable x and the dependent variable y is modelled as an nth degree polynomial in x. Polynomial regression fits a nonlinear relationship between the value of x and the corresponding conditional mean of y

CHAPTER 3: Data Description

Dataset link: https://www.kaggle.com/datasets/toramky/automobile-dataset?resource=download&sort=votes

This dataset consist of data From 1985 Ward's Automotive Yearbook. Here are the sources

Sources:

1. 1985 Model Import Car and Truck Specifications, 1985 Ward's Automotive Yearbook.
2. Personal Auto Manuals, Insurance Services Office, 160 Water Street, New York, NY 10038
3. Insurance Collision Report, Insurance Institute for Highway Safety, Watergate 600, Washington, DC 20037

The Data contains the following columns:

1. symboling: This indicates the risk factor associated with respective car. A value of +3 indicates that the auto is risky, -3 that it is probably pretty safe.
2. normalized-losses: It is the relative average loss of payment per insured vehicle year. This value is normalized for all autos within a particular size classification (two-door small, station wagons, sports/speciality, etc...), and represents the average loss per car per year.
3. make: It denotes the company that has manufactured the car.
4. fuel-type: It denotes the type of fuel on which the car runs, either gas or diesel.
5. aspiration: It shows the type of internal combustion engine being used in the car. Is it a naturally aspirated engine denoted by "std" or a turbo-charged engine denoted by "turbo".
6. num-of-doors: It shows how many doors does the car have, 2 or 4 doors.
7. body-style: The body style that is followed by the car, e.g: Sedan, Hatchback, etc.
8. drive-wheels: The type of wheel drive used by the car, whether it uses front wheel axel drive, rear wheel drive or 4 wheel drive.
9. engine-location: This determines the placement of engine with respect to the car, either front or back.
10. wheel-base: It is the horizontal distance between the centers of the front and rear wheels.
11. length: The length of the car.
12. width: The width of the car.
13. height: The height of the car.
14. curb-weight: It is the weight of the vehicle including a full tank of fuel and all standard equipment.
15. engine-type: The engine used in the car like overhead camshaft(ohc), dual overhead camshaft(dohc), overhead valve(ohv) etc.
16. num-of-cylinders: The number of cylinders used by the engine of a car.
17. engine-size: It is the volume of fuel and air that can be pushed through a car's cylinders and is measured in cubic centimetres (cc).
18. fuel-system: The method used by which the fuel is stored and supplied to the cylinder chambers.
19. bore: It is the diameter of each cylinder.
20. stroke: The stroke length is how far the piston travels in the cylinder, which is determined by the cranks on the crankshaft.
21. compression-ratio: It is the ratio between the volume of the cylinder and combustion chamber in an internal combustion engine at their maximum and minimum values.
22. horsepower: It is a unit of measurement of power, or the rate at which work is done, usually in reference to the output of engines or motors.
23. peak-rpm: The maximum revolution per minute that can be attained by the car engine.
24. city-mpg: It is the mileage given by the car within city, in miles per gallon.
25. highway-mpg: It is the mileage given by the car on highway, in miles per gallon.
26. price: It is the price of the car.

Importing the required libraries

In [1]:
```
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objects as go
import seaborn as sns
import plotly.offline as pyo
import warnings
from scipy import stats
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler, OrdinalEncoder, PolynomialFeatures
```

```
from statsmodels.stats.weightstats import ztest
from pingouin import multivariate_normality
from sklearn.decomposition import PCA
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
import scikit_posthocs as sp

warnings.filterwarnings("ignore")
pyo.init_notebook_mode()
```

In [2]:
```
automotive_data = pd.read_csv("/Users/subhammoda/Documents/Stevens/MA 541/Project/Automobile_data.csv")
automotive_data.head()
```

Out[2]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	strok
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.6
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.6
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.4
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.

5 rows × 26 columns

In [3]:
```
automotive_data.shape
```

Out[3]:
```
(205, 26)
```

In [4]:
```
automotive_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   symboling          205 non-null    int64  
 1   normalized-losses  205 non-null    object 
 2   make               205 non-null    object 
 3   fuel-type          205 non-null    object 
 4   aspiration         205 non-null    object 
 5   num-of-doors       205 non-null    object 
 6   body-style         205 non-null    object 
 7   drive-wheels       205 non-null    object 
 8   engine-location    205 non-null    object 
 9   wheel-base         205 non-null    float64
 10  length             205 non-null    float64
 11  width              205 non-null    float64
 12  height             205 non-null    float64
 13  curb-weight        205 non-null    int64  
 14  engine-type        205 non-null    object 
 15  num-of-cylinders   205 non-null    object 
 16  engine-size        205 non-null    int64  
 17  fuel-system        205 non-null    object 
 18  bore               205 non-null    object 
 19  stroke             205 non-null    object 
 20  compression-ratio  205 non-null    float64
 21  horsepower         205 non-null    object 
 22  peak-rpm           205 non-null    object 
 23  city-mpg           205 non-null    int64  
 24  highway-mpg        205 non-null    int64  
 25  price              205 non-null    object 
dtypes: float64(5), int64(5), object(16)
memory usage: 41.8+ KB
```

Since we saw that some of the rows have missing values in the form of '?', we find out which columns have such missing values.

In [5]:
```
automotive_data.isin(['?']).any()
```

Out[5]:
```
symboling            False
normalized-losses     True
make                 False
fuel-type            False
aspiration           False
num-of-doors          True
body-style           False
drive-wheels         False
engine-location      False
wheel-base           False
length               False
width                False
height               False
curb-weight          False
engine-type          False
num-of-cylinders     False
engine-size          False
fuel-system          False
bore                  True
stroke                True
compression-ratio    False
horsepower            True
peak-rpm              True
city-mpg             False
highway-mpg          False
price                 True
dtype: bool
```

We can see that there are missing values which are valued as "?". Removing and handling those missing values. Handling the datatypes.

In [6]:
```
automotive_data[['bore','stroke','horsepower','peak-rpm','price','normalized-losses']] = automotive_data[['bo
automotive_data.describe()
```

Out[6]:

	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	bore	str
count	205.000000	164.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	201.000000	201.0000
mean	0.834146	122.000000	98.756585	174.049268	65.907805	53.724878	2555.565854	126.907317	3.329751	3.255
std	1.245307	35.442168	6.021776	12.337289	2.145204	2.443522	520.680204	41.642693	0.273539	0.316
min	-2.000000	65.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	2.540000	2.070
25%	0.000000	94.000000	94.500000	166.300000	64.100000	52.000000	2145.000000	97.000000	3.150000	3.110
50%	1.000000	115.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000	3.310000	3.290
75%	2.000000	150.000000	102.400000	183.100000	66.900000	55.500000	2935.000000	141.000000	3.590000	3.410
max	3.000000	256.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.000000	3.940000	4.170

In [7]:
```
automotive_data[automotive_data['price'].isna()]
```

Out[7]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	str
9	0	NaN	audi	gas	turbo	two	hatchback	4wd	front	99.5	...	131	mpfi	3.13	3
44	1	NaN	isuzu	gas	std	two	sedan	fwd	front	94.5	...	90	2bbl	3.03	
45	0	NaN	isuzu	gas	std	four	sedan	fwd	front	94.5	...	90	2bbl	3.03	
129	1	NaN	porsche	gas	std	two	hatchback	rwd	front	98.4	...	203	mpfi	3.94	

4 rows × 26 columns

Dropping rows where price is missing.

In [8]:
```
automotive_data.drop(automotive_data[automotive_data['price'].isna()].index,inplace=True)
```

Replacing missing values with respective mean/mode values.

In [9]:
```
automotive_data['bore'] = automotive_data['bore'].fillna(automotive_data['bore'].mean())
automotive_data['stroke'] = automotive_data['stroke'].fillna(automotive_data['stroke'].mean())
automotive_data['horsepower'] = automotive_data['horsepower'].fillna(automotive_data['horsepower'].median())
```

```
automotive_data['peak-rpm'] = automotive_data['peak-rpm'].fillna(automotive_data['peak-rpm'].median())
automotive_data['normalized-losses'] = automotive_data['normalized-losses'].fillna(automotive_data['normalize
automotive_data.describe()
```

Out[9]:

	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	bore	strol
count	201.000000	201.00000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	201.00000
mean	0.840796	122.00000	98.797015	174.200995	65.889055	53.766667	2555.666667	126.875622	3.330711	3.25690
std	1.254802	31.99625	6.066366	12.322175	2.101471	2.447822	517.296727	41.546834	0.268072	0.31604
min	-2.000000	65.00000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	2.540000	2.07000
25%	0.000000	101.00000	94.500000	166.800000	64.100000	52.000000	2169.000000	98.000000	3.150000	3.11000
50%	1.000000	122.00000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000	3.310000	3.29000
75%	2.000000	137.00000	102.400000	183.500000	66.600000	55.500000	2926.000000	141.000000	3.580000	3.41000
max	3.000000	256.00000	120.900000	208.100000	72.000000	59.800000	4066.000000	326.000000	3.940000	4.17000

There are fields that are not exactly categorical variable and can be replaced/transformed into a numerical variable. Here, num-of-doors and fuel-type are such variables. Converting them into numerical variable and transforming the same.

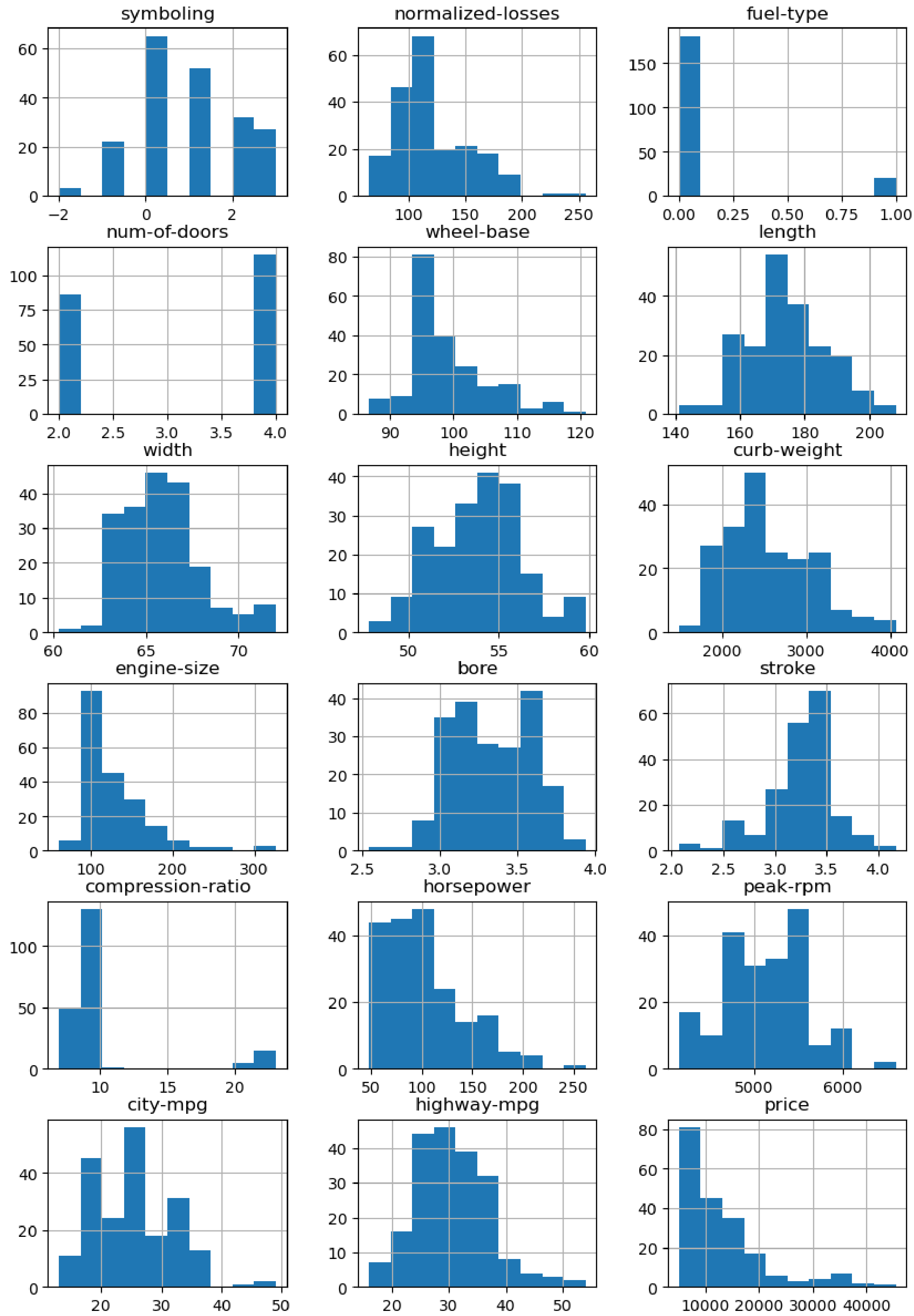
In [10]:
```
automotive_data['num-of-doors'] = automotive_data['num-of-doors'].replace({'four':'4','two':'2'})
automotive_data[['num-of-doors']] = automotive_data[['num-of-doors']].apply(pd.to_numeric, errors='coerce')
automotive_data['num-of-doors'] = automotive_data['num-of-doors'].fillna(automotive_data['num-of-doors'].mode
automotive_data['fuel-type'] = automotive_data['fuel-type'].replace({'gas':'0','diesel':'1'})
automotive_data[['fuel-type']] = automotive_data[['fuel-type']].apply(pd.to_numeric, errors='coerce')
automotive_data['fuel-type'] = automotive_data['fuel-type'].fillna(automotive_data['num-of-doors'].mode()[0]
automotive_data.describe()
```

Out[10]:

	symboling	normalized-losses	fuel-type	num-of-doors	wheel-base	length	width	height	curb-weight	engin si:
count	201.000000	201.00000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	201.00000
mean	0.840796	122.00000	0.099502	3.144279	98.797015	174.200995	65.889055	53.766667	2555.666667	126.87562
std	1.254802	31.99625	0.300083	0.992008	6.066366	12.322175	2.101471	2.447822	517.296727	41.54683
min	-2.000000	65.00000	0.000000	2.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.00000
25%	0.000000	101.00000	0.000000	2.000000	94.500000	166.800000	64.100000	52.000000	2169.000000	98.00000
50%	1.000000	122.00000	0.000000	4.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.00000
75%	2.000000	137.00000	0.000000	4.000000	102.400000	183.500000	66.600000	55.500000	2926.000000	141.00000
max	3.000000	256.00000	1.000000	4.000000	120.900000	208.100000	72.000000	59.800000	4066.000000	326.00000

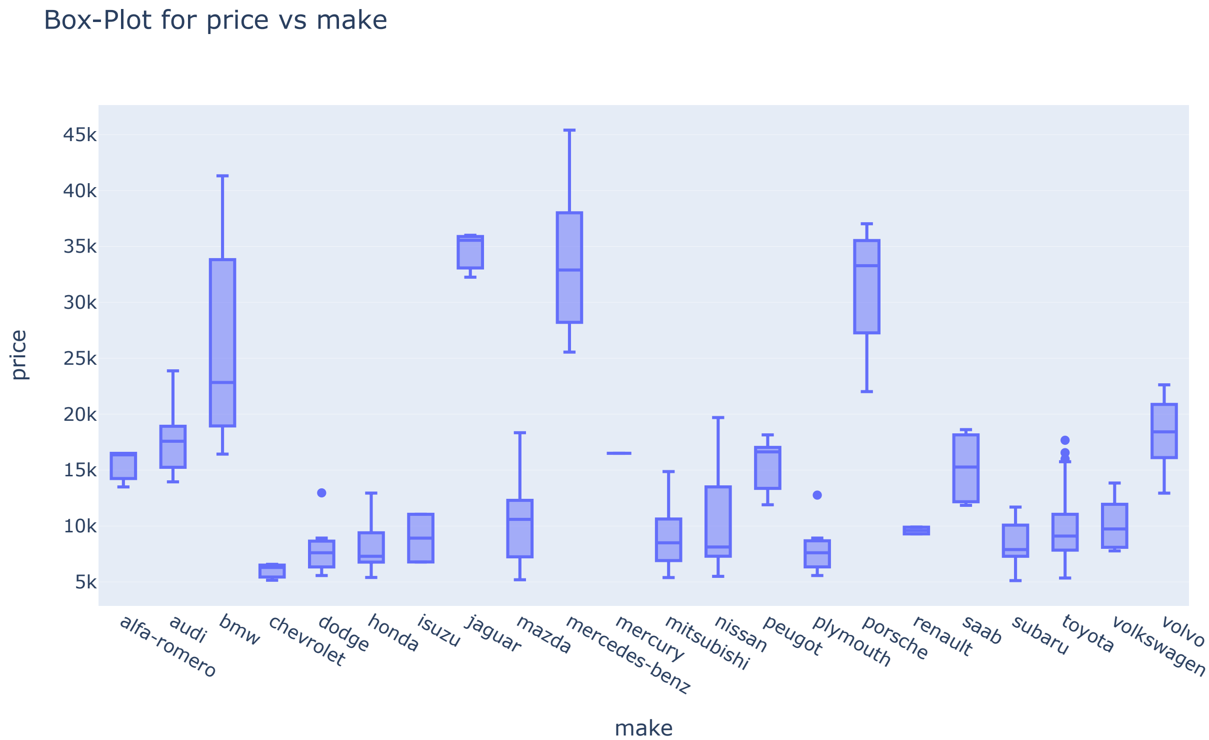
The histogram below shows how the data for each variable is distributed.

In [11]:
```
automotive_data.hist(layout=(10,3),figsize=(10,25))
plt.show()
```



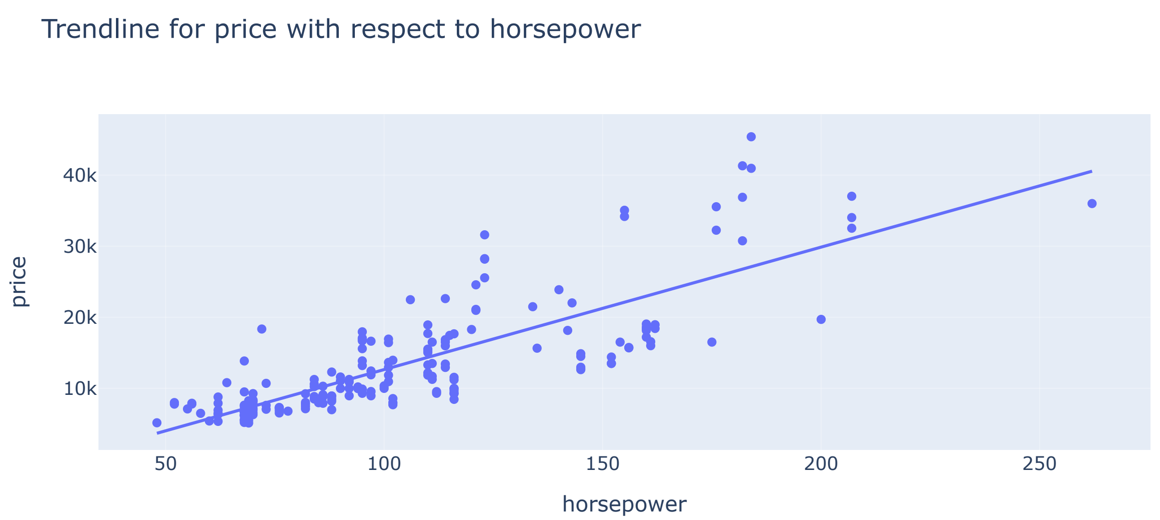
The box plot below shows how the price of a car is varied based on the make of the car.

In [12]:
```
fig = px.box(data_frame = automotive_data,x="make", y = 'price', title="Box-Plot for price vs make", width=8
fig.show()
```

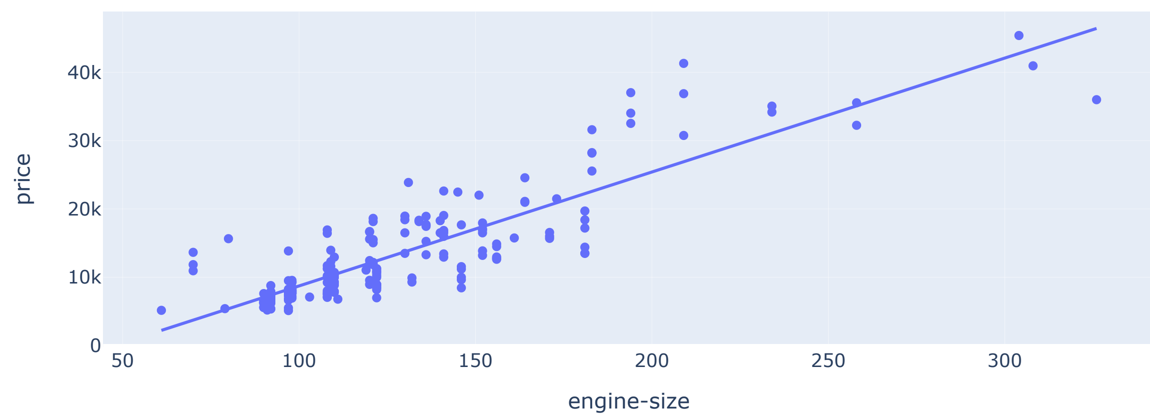


We can see how various feature affect the price and respective trendlines in the following scatter plots.

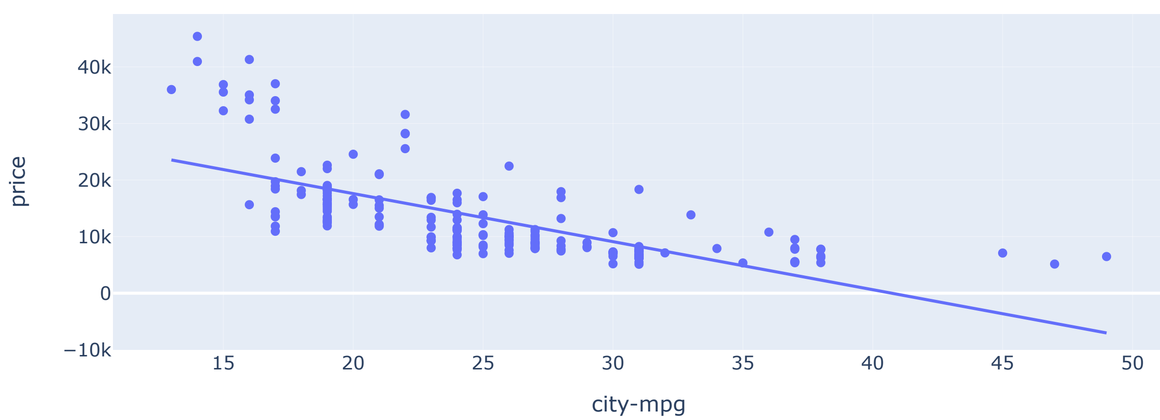
In [13]:
```
trend_li = ['horsepower','engine-size','city-mpg','highway-mpg','length','width','curb-weight','wheel-base']
for trend in trend_li:
    fig = px.scatter(data_frame = automotive_data, y="price", x=trend, trendline="ols", title="Trendline for
    fig.show()
```



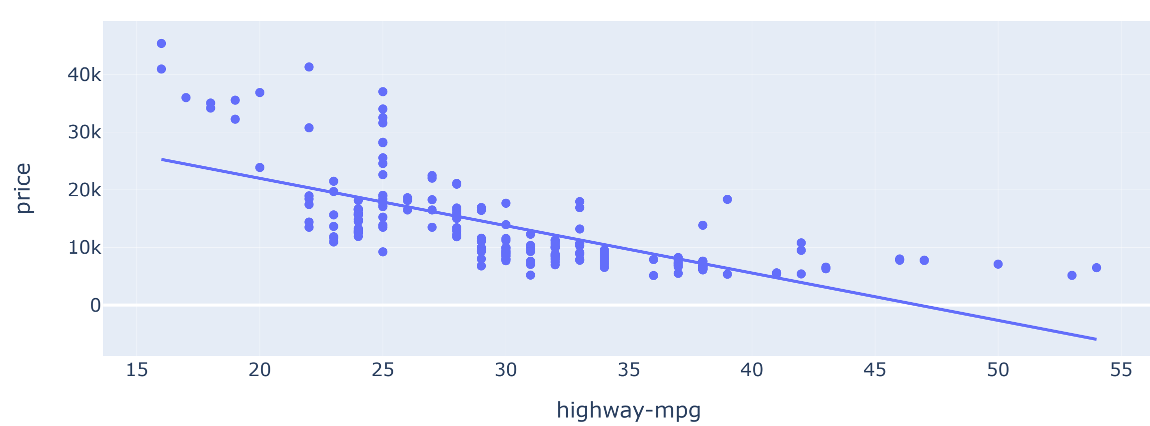
Trendline for price with respect to engine-size

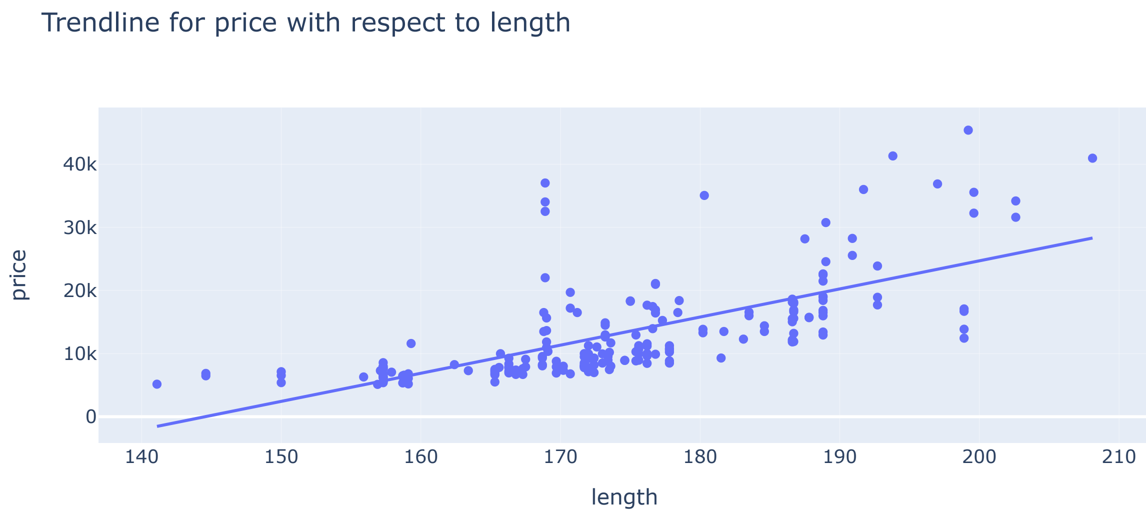


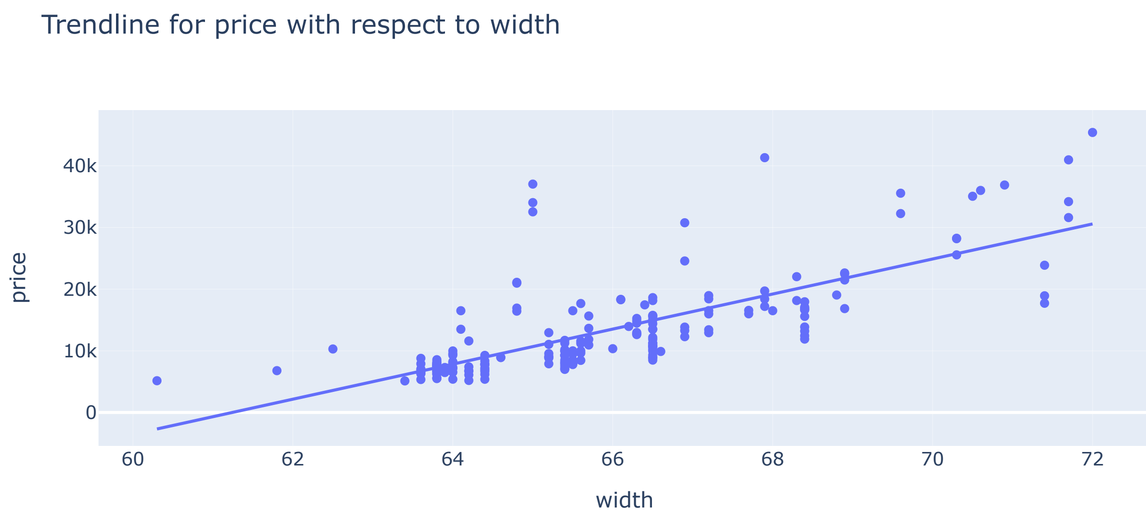
Trendline for price with respect to city-mpg



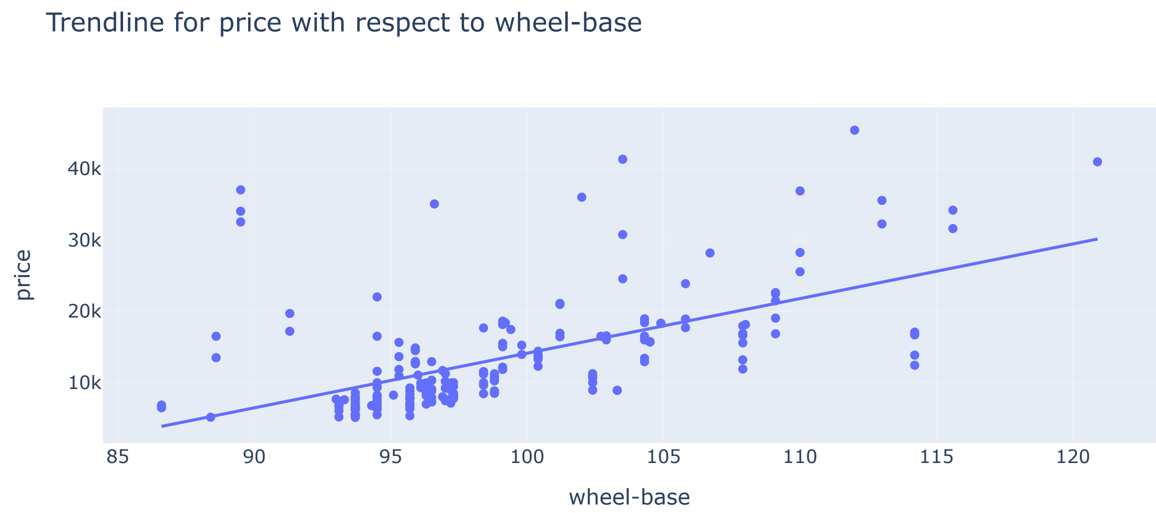
Trendline for price with respect to highway-mpg




Trendline for price with respect to length
price
40k
30k
20k
10k
0
140
150
160
170
180
190
200
210
length

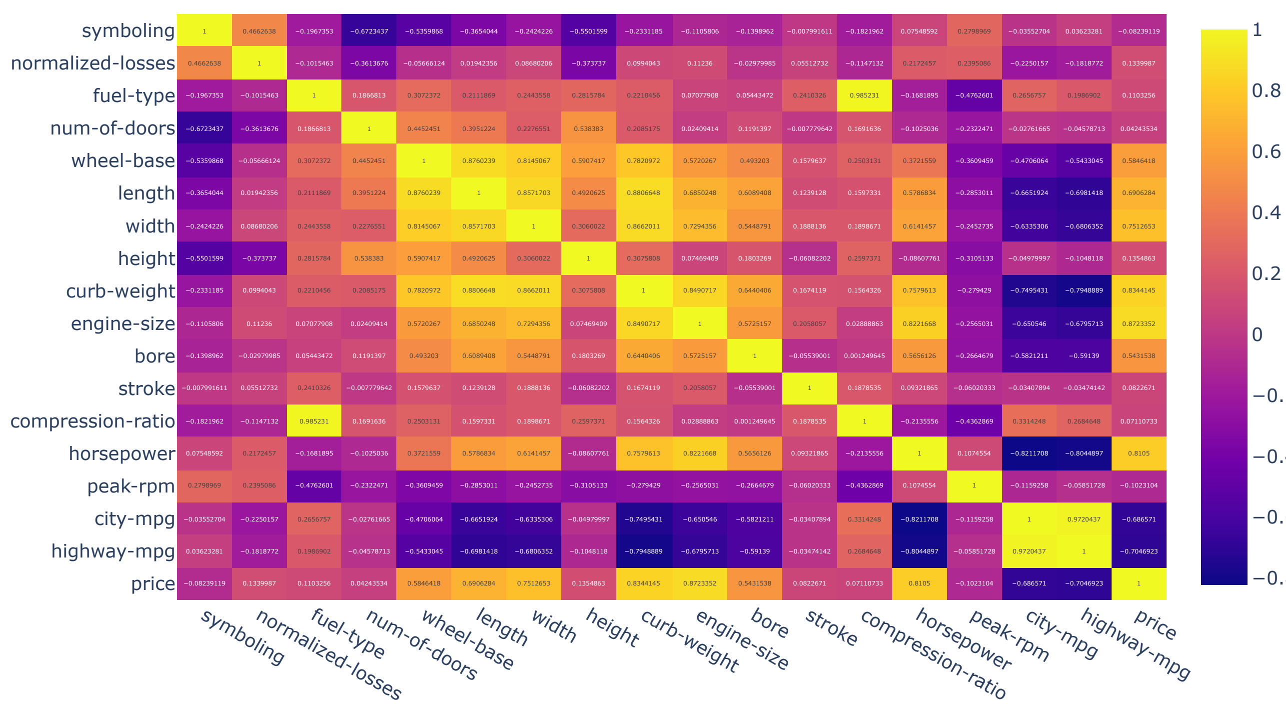

Trendline for price with respect to width
price
40k
30k
20k
10k
0
60
62
64
66
68
70
72
width


Trendline for price with respect to curb-weight
price
40k
30k
20k
10k
0
1500
2000
2500
3000
3500
4000
curb-weight



The correlation heat-map shows how the data are correlated with respect to other features and we can confirm that the features selected above for scatterplot have massive correlation with price.

In [14]:
```
fig = px.imshow(automotive_data.corr(), text_auto=True, aspect="auto", width=875)
fig.show()
```



In [15]:
```
automotive_data.corr()['price'][:]
```

Out[15]:

```
symboling            -0.082391
normalized-losses     0.133999
fuel-type             0.110326
num-of-doors          0.042435
wheel-base            0.584642
length                0.690628
width                 0.751265
height                0.135486
curb-weight           0.834415
engine-size           0.872335
bore                  0.543154
stroke                0.082267
compression-ratio     0.071107
horsepower            0.810500
peak-rpm             -0.102310
city-mpg             -0.686571
highway-mpg          -0.704692
price                 1.000000
Name: price, dtype: float64
```

CHAPTER 4: Analysis and Results

Univariate Analysis

Test to check if the dataset is normally distributed. I also check how likely it is for a random variable in the dataset to be normally distributed. There are multiple numerical features, I check on all of these features.

Hypothesis -

H_0 = The Sample has Gaussian distribution.

H_1 = The Sample does not have Gaussian distribution.

I perform Shapiro-Wilk Test, since N<5000

In [16]:

```
for feature in automotive_data._get_numeric_data().columns:
    print(feature,":")
    stat, pval = stats.shapiro(automotive_data[feature])
    print("Statistics for {}: %.3f".format(feature) %stat, "p-value for {}: %.3f".format(feature) %pval)
    print("Sample does not look Gaussian (reject H0)") if pval<0.05 else print("Sample looks Gaussian (fail 
    print("================================================================================")
```

```
symboling :
Statistics for symboling: 0.918 p-value for symboling: 0.000
Sample does not look Gaussian (reject H0)
==============================================================================
normalized-losses :
Statistics for normalized-losses: 0.951 p-value for normalized-losses: 0.000
Sample does not look Gaussian (reject H0)
==============================================================================
fuel-type :
Statistics for fuel-type: 0.341 p-value for fuel-type: 0.000
Sample does not look Gaussian (reject H0)
==============================================================================
num-of-doors :
Statistics for num-of-doors: 0.629 p-value for num-of-doors: 0.000
Sample does not look Gaussian (reject H0)
==============================================================================
wheel-base :
Statistics for wheel-base: 0.913 p-value for wheel-base: 0.000
Sample does not look Gaussian (reject H0)
==============================================================================
length :
Statistics for length: 0.982 p-value for length: 0.010
Sample does not look Gaussian (reject H0)
==============================================================================
width :
Statistics for width: 0.924 p-value for width: 0.000
Sample does not look Gaussian (reject H0)
==============================================================================
height :
Statistics for height: 0.984 p-value for height: 0.022
Sample does not look Gaussian (reject H0)
==============================================================================
curb-weight :
Statistics for curb-weight: 0.953 p-value for curb-weight: 0.000
Sample does not look Gaussian (reject H0)
==============================================================================
engine-size :
Statistics for engine-size: 0.827 p-value for engine-size: 0.000
Sample does not look Gaussian (reject H0)
==============================================================================
bore :
Statistics for bore: 0.966 p-value for bore: 0.000
Sample does not look Gaussian (reject H0)
==============================================================================
stroke :
Statistics for stroke: 0.937 p-value for stroke: 0.000
Sample does not look Gaussian (reject H0)
==============================================================================
compression-ratio :
Statistics for compression-ratio: 0.497 p-value for compression-ratio: 0.000
Sample does not look Gaussian (reject H0)
==============================================================================
horsepower :
Statistics for horsepower: 0.904 p-value for horsepower: 0.000
Sample does not look Gaussian (reject H0)
==============================================================================
peak-rpm :
Statistics for peak-rpm: 0.970 p-value for peak-rpm: 0.000
Sample does not look Gaussian (reject H0)
==============================================================================
city-mpg :
Statistics for city-mpg: 0.957 p-value for city-mpg: 0.000
Sample does not look Gaussian (reject H0)
==============================================================================
highway-mpg :
Statistics for highway-mpg: 0.972 p-value for highway-mpg: 0.001
Sample does not look Gaussian (reject H0)
==============================================================================
price :
Statistics for price: 0.799 p-value for price: 0.000
Sample does not look Gaussian (reject H0)
==============================================================================
```

CHAPTER 4.1: Comparing two samples

Statistical Analysis with various hypothesis testing on mulitple paired data types in order to know whether the pair is dependent or indepoendent

I perform Z test since the datasets are not normally distributed and the sample size is greater than 30.

A z-test is a statistical test used to determine whether two population means are different when the variances are known and the sample size is large. If a z-score is 0, it indicates that the data point's score is identical to the mean score. A z-score of 1.0 would indicate a value that is one standard deviation from the mean. Z-scores may be positive or negative, with a positive value indicating the score is above the mean and a negative score indicating it is below the mean. **I use Z test between Numerical & Numerical.**

Hypothesis -

H_0 = There is no difference between two samples.

H_1 = There is difference between the two samples.

In [17]:
```
def perform_ztest(feature1, feature2):
    z, pval = ztest(automotive_data[feature1], automotive_data[feature2])
    print("Correlation between {} and {}: %.3f".format(feature1, feature2) %z, "p-value: %.3f" %pval)
    print("There is difference between the two samples. (reject H0)") if pval<0.05 else print("There is no d
```

In [18]:
```
numeric_features = automotive_data._get_numeric_data().columns
temp = []
for i in range(len(numeric_features)-1):
    if [numeric_features[i],numeric_features[i+1]] not in temp:
        temp.append([numeric_features[i],numeric_features[i+1]])
for i in temp:
    perform_ztest(i[0],i[1])
    print("================================================================================")
```

```
Correlation between symboling and normalized-losses: -53.644 p-value: 0.000
There is difference between the two samples. (reject H0)
==============================================================================
Correlation between normalized-losses and fuel-type: 54.011 p-value: 0.000
There is difference between the two samples. (reject H0)
==============================================================================
Correlation between fuel-type and num-of-doors: -41.651 p-value: 0.000
There is difference between the two samples. (reject H0)
==============================================================================
Correlation between num-of-doors and wheel-base: -220.616 p-value: 0.000
There is difference between the two samples. (reject H0)
==============================================================================
Correlation between wheel-base and length: -77.836 p-value: 0.000
There is difference between the two samples. (reject H0)
==============================================================================
Correlation between length and width: 122.846 p-value: 0.000
There is difference between the two samples. (reject H0)
==============================================================================
Correlation between width and height: 53.272 p-value: 0.000
There is difference between the two samples. (reject H0)
==============================================================================
Correlation between height and curb-weight: -68.568 p-value: 0.000
There is difference between the two samples. (reject H0)
==============================================================================
Correlation between curb-weight and engine-size: 66.352 p-value: 0.000
There is difference between the two samples. (reject H0)
==============================================================================
Correlation between engine-size and bore: 42.158 p-value: 0.000
There is difference between the two samples. (reject H0)
==============================================================================
Correlation between bore and stroke: 2.525 p-value: 0.012
There is difference between the two samples. (reject H0)
==============================================================================
Correlation between stroke and compression-ratio: -24.376 p-value: 0.000
There is difference between the two samples. (reject H0)
==============================================================================
Correlation between compression-ratio and horsepower: -35.133 p-value: 0.000
There is difference between the two samples. (reject H0)
==============================================================================
Correlation between horsepower and peak-rpm: -148.238 p-value: 0.000
There is difference between the two samples. (reject H0)
==============================================================================
Correlation between peak-rpm and city-mpg: 150.993 p-value: 0.000
There is difference between the two samples. (reject H0)
==============================================================================
Correlation between city-mpg and highway-mpg: -8.338 p-value: 0.000
There is difference between the two samples. (reject H0)
==============================================================================
Correlation between highway-mpg and price: -23.507 p-value: 0.000
There is difference between the two samples. (reject H0)
==============================================================================
```

Multivariate Normality Test

I perform the multivariate normality test on the complete data before performing the multivariate analysis.

H_0 = The dataset follows multivariate normal distribution.

H_1 = The dataset does not follow multivariate normal distribution.

In [19]:
```
hz, pval, normality = multivariate_normality(automotive_data._get_numeric_data(),alpha=0.05)
print("H-Z test statistics for the dataset: %.3f" %hz, "p-value for the dataset: %.3f" %pval)
print("The dataset does not follow multivariate normal distribution. (reject H0)") if pval<0.05 else print("
```

```
H-Z test statistics for the dataset: 1.707 p-value for the dataset: 0.000
The dataset does not follow multivariate normal distribution. (reject H0)
```

CHAPTER 4.2: The Analysis of Variance (ONE-WAY ANOVA)

I perform Kruskal-Wallis H-test for multivariate analysis for more than 2 categorical or numerical data.

The Kruskal-Wallis H-test tests the null hypothesis that the population mean of all of the groups are equal. It is a non-parametric version of ANOVA. The test works on 2 or more independent samples, which may have different sizes.

I also perform the Nemenyni Test in order to know which two pairs of data have difference in metrics.

Hypothesis -

H_0 = The mean is equal across groups.

H_1 = The mean is not equal across more than one pair.

In [20]:
```
stat, pval= stats.kruskal(automotive_data['symboling'],automotive_data['normalized-losses'],automotive_data[
print("Correlation between all numerical variables: %.3f" %stat, "p-value: %.3f" %pval)
print("The mean is not equal across more than one pair. (reject H0)") if pval<0.05 else print("The mean is e
```

```
Correlation between all numerical variables: 3537.271 p-value: 0.000
The mean is not equal across more than one pair. (reject H0)
```

In [21]:
```
result = round(sp.posthoc_nemenyi([automotive_data['symboling'],automotive_data['normalized-losses'],automoti
result.rename(columns={1:'symboling',2:'normalized-losses',3:'fuel-type',4:'num-of-doors',5:'wheel-base',6:'
result.rename(index={1:'symboling',2:'normalized-losses',3:'fuel-type',4:'num-of-doors',5:'wheel-base',6:'le
result
```

Out[21]:

	symboling	normalized-losses	fuel-type	num-of-doors	wheel-base	length	width	height	curb-weight	engine-size	bore	stroke	compression-ratio	h
symboling	1.00	0.00	1.00	0.35	0.00	0.00	0.00	0.00	0.00	0.00	0.43	0.54	0.00	
normalized-losses	0.00	1.00	0.00	0.00	1.00	0.82	0.02	0.00	0.01	1.00	0.00	0.00	0.00	
fuel-type	1.00	0.00	1.00	0.05	0.00	0.00	0.00	0.00	0.00	0.00	0.07	0.11	0.00	
num-of-doors	0.35	0.00	0.05	1.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	1.00	0.61	
wheel-base	0.00	1.00	0.00	0.00	1.00	0.05	0.68	0.01	0.00	1.00	0.00	0.00	0.00	
length	0.00	0.82	0.00	0.00	0.05	1.00	0.00	0.00	0.99	0.87	0.00	0.00	0.00	
width	0.00	0.02	0.00	0.00	0.68	0.00	1.00	1.00	0.00	0.01	0.00	0.00	0.00	
height	0.00	0.00	0.00	0.00	0.01	0.00	1.00	1.00	0.00	0.00	0.00	0.00	0.01	
curb-weight	0.00	0.01	0.00	0.00	0.00	0.99	0.00	0.00	1.00	0.01	0.00	0.00	0.00	
engine-size	0.00	1.00	0.00	0.00	1.00	0.87	0.01	0.00	0.01	1.00	0.00	0.00	0.00	
bore	0.43	0.00	0.07	1.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	1.00	0.52	
stroke	0.54	0.00	0.11	1.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	1.00	0.41	
compression-ratio	0.00	0.00	0.00	0.61	0.00	0.00	0.00	0.01	0.00	0.00	0.52	0.41	1.00	
horsepower	0.00	1.00	0.00	0.00	1.00	0.05	0.69	0.01	0.00	1.00	0.00	0.00	0.00	
peak-rpm	0.00	0.00	0.00	0.00	0.00	0.33	0.00	0.00	1.00	0.00	0.00	0.00	0.00	
city-mpg	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.82	0.00	0.00	0.00	0.00	1.00	
highway-mpg	0.00	0.00	0.00	0.00	0.00	0.00	0.21	0.99	0.00	0.00	0.00	0.00	0.88	
price	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.62	0.00	0.00	0.00	0.00	

From the table above we can see the column pairs with value greater than equal to 0.05 have equal mean, while with value less than 0.05 have unequal means. For example: symboling and num-of-doors have p-value of 0.35, that means they have equal means, while wheel-base and height have p-value of 0.01, that means they have unequal means

In [22]:
```
stat, pval= stats.kruskal(automotive_data['fuel-system'], automotive_data['engine-type'], automotive_data['nu
print("Correlation between all categorical variables: %.3f" %stat, "p-value: %.3f" %pval)
print("The mean is not equal across more than one pair. (reject H0)") if pval<0.05 else print("The mean is e
```

```
Correlation between all categorical variables: 816.025 p-value: 0.000
The mean is not equal across more than one pair. (reject H0)
```

In [23]:
```
result = round(sp.posthoc_nemenyi([automotive_data['fuel-system'], automotive_data['engine-type'], automotive
result.rename(columns={1:'fuel-system',2:'engine-type',3:'num-of-cylinders',4:'aspiration',5:'body-style',6:
result.rename(index={1:'fuel-system',2:'engine-type',3:'num-of-cylinders',4:'aspiration',5:'body-style',6:'ma
result
```

Out[23]:

	fuel-system	engine-type	num-of-cylinders	aspiration	body-style	make	engine-location	drive-wheels
fuel-system	1.00	0.00	0.16	0.0	0.00	0.00	0.42	0.00
engine-type	0.00	1.00	0.00	0.0	0.63	0.98	0.00	0.09
num-of-cylinders	0.16	0.00	1.00	0.0	0.00	0.00	1.00	0.00
aspiration	0.00	0.00	0.00	1.0	0.00	0.00	0.00	0.00
body-style	0.00	0.63	0.00	0.0	1.00	0.99	0.00	0.00
make	0.00	0.98	0.00	0.0	0.99	1.00	0.00	0.00
engine-location	0.42	0.00	1.00	0.0	0.00	0.00	1.00	0.00
drive-wheels	0.00	0.09	0.00	0.0	0.00	0.00	0.00	1.00

From the table above we can see the column pairs with value greater than equal to 0.05 have equal mean, while with value less than 0.05 have unequal means. For example: fuel-system and num-of-cylinders have p-value of 0.16, that means they have equal means, while aspiration and engine-type have p-value of 0.00, that means they have unequal means

CHAPTER 4.3: The Analysis of Categorical Data

I perform Chi-Square Test for the independence of different categories of a population.

The Chi-Square Test computes the chi-square statistic and p-value for the hypothesis test of independence of the observed frequencies in the contingency table observed. The expected frequencies are computed based on the marginal sums under the assumption of independence.

Hypothesis -

H_0 = The two samples are not related to each other.

H_1 = The two samples are related to each other.

In [24]:
```
def find_chisqaure(feature1, feature2):
    crosstab = pd.crosstab(index = automotive_data[feature1], columns = automotive_data[feature2])
    chi, pval, _, _ = stats.chi2_contingency(crosstab)
    print("Correlation between {} and {}: %.3f".format(feature1, feature2) %chi, "p-value: %.3f" %pval)
    print("The two samples are related to each other. (reject H0)") if pval<0.05 else print("The two samples
```

In [25]:
```
cols = automotive_data.columns
numerical_cols = automotive_data._get_numeric_data().columns
categorical_cols = list(set(cols)-set(numerical_cols))
categorical_cols.remove('make')

temp = []
for i in range(len(categorical_cols)-1):
    if [categorical_cols[i],categorical_cols[i+1]] not in temp:
        temp.append([categorical_cols[i],categorical_cols[i+1]])

for i in temp:
    find_chisqaure(i[0],i[1])
    print("==============================================================================")
```

```
Correlation between drive-wheels and aspiration: 3.685 p-value: 0.158
The two samples are not related to each other. (fail to reject H0)
==============================================================================
Correlation between aspiration and engine-type: 10.471 p-value: 0.063
The two samples are not related to each other. (fail to reject H0)
==============================================================================
Correlation between engine-type and num-of-cylinders: 355.831 p-value: 0.000
The two samples are related to each other. (reject H0)
==============================================================================
Correlation between num-of-cylinders and fuel-system: 204.058 p-value: 0.000
The two samples are related to each other. (reject H0)
==============================================================================
Correlation between fuel-system and body-style: 46.839 p-value: 0.014
The two samples are related to each other. (reject H0)
==============================================================================
Correlation between body-style and engine-location: 42.298 p-value: 0.000
The two samples are related to each other. (reject H0)
==============================================================================
```

CHAPTER 4.4: Linear Regression

Getting a list of the categorical columns.

In [26]:
```
def MAPE(Y_actual,Y_Predicted):
    mape = round(np.mean(np.abs((Y_actual - Y_Predicted)/Y_actual))*100,2)
    return mape
```

In [27]:
```
cols = automotive_data.columns
numerical_cols = automotive_data._get_numeric_data().columns
categorical_cols = list(set(cols)-set(numerical_cols))
print("The categorical columns are: ", categorical_cols)
```

```
The categorical columns are:  ['drive-wheels', 'aspiration', 'engine-type', 'make', 'num-of-cylinders', 'fue
l-system', 'body-style', 'engine-location']
```

Converting the categorical data to numerical data using Ordinal Encoder in order use these columns for prediction in linear regressor.

In [28]:
```
oe = OrdinalEncoder(handle_unknown='use_encoded_value',unknown_value=-1)
oe.fit(automotive_data[categorical_cols])
automotive_data[categorical_cols] = oe.transform(automotive_data[categorical_cols])
automotive_data.head()
```

Out[28]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	co
0	3	122.0	0.0	0	0.0	2.0	0.0	2.0	0.0	88.6	...	130	5.0	3.47	2.68	
1	3	122.0	0.0	0	0.0	2.0	0.0	2.0	0.0	88.6	...	130	5.0	3.47	2.68	
2	1	122.0	0.0	0	0.0	2.0	2.0	2.0	0.0	94.5	...	152	5.0	2.68	3.47	
3	2	164.0	1.0	0	0.0	4.0	3.0	1.0	0.0	99.8	...	109	5.0	3.19	3.40	
4	2	164.0	1.0	0	0.0	4.0	3.0	0.0	0.0	99.4	...	136	5.0	3.19	3.40	

5 rows × 26 columns

Splitting the Data into X and y, the features and target.

In [29]:
```
X = automotive_data.drop('price', axis=1)
y = automotive_data['price']
```

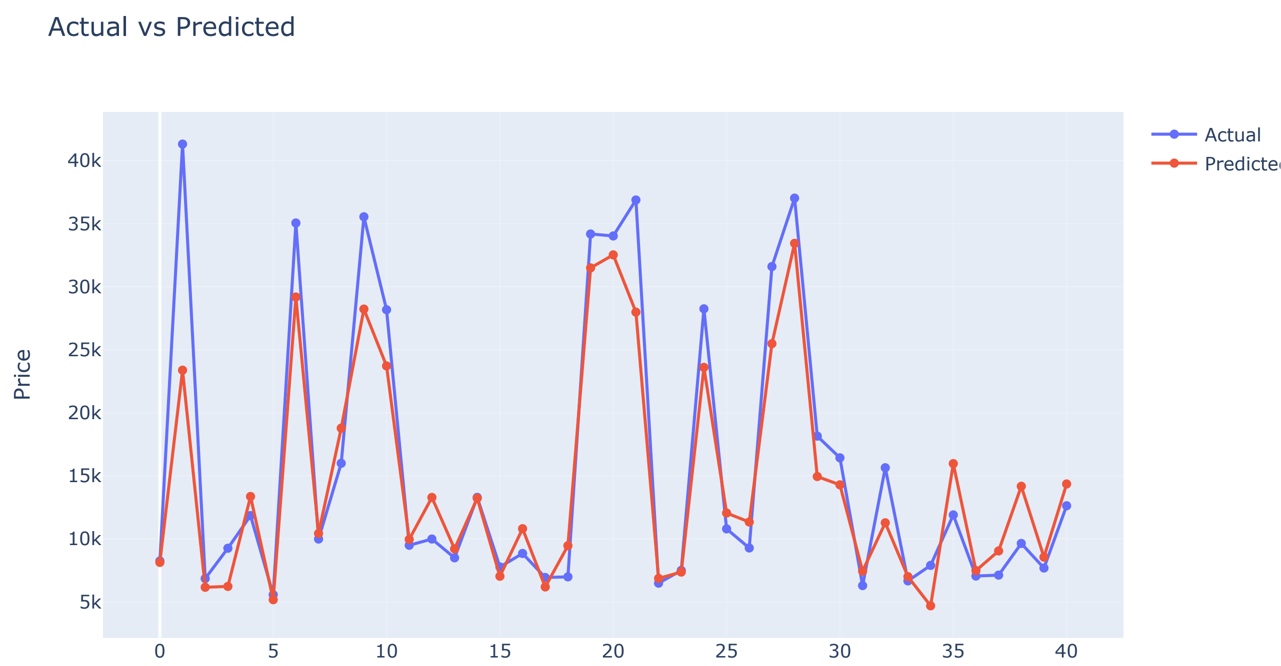
Splitting the data into train and test data in order to check the score of the model on the training data and testing data.

In [30]:
```
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.2, random_state=42)
train_mape = []
test_mape = []
```

In [31]:
```
lr_model = LinearRegression()
lr_model.fit(X_train,y_train)
print("MAPE for linear regression on train data: ", MAPE(y_train,lr_model.predict(X_train)))
print("MAPE for linear regression on test data: ", MAPE(y_test,lr_model.predict(X_test)))
train_mape.append(MAPE(y_train,lr_model.predict(X_train)))
test_mape.append(MAPE(y_test,lr_model.predict(X_test)))
```

```
MAPE for linear regression on train data:  14.44
MAPE for linear regression on test data:  16.88
```

In [32]:
```
y_pred = lr_model.predict(X_test)

data = {'y_pred':y_pred,'y_actual':y_test}
plot_df = pd.DataFrame(data)

fig = go.Figure()
fig.add_trace(go.Scatter(y=plot_df['y_actual'],mode='lines+markers',name='Actual'))
fig.add_trace(go.Scatter(y=plot_df['y_pred'],mode='lines+markers',name='Predicted'))
fig.update_layout(width=875, title_text="Actual vs Predicted", yaxis=dict(title_text="Price"))
fig.show()
```



We can see that there is minor difference between the actual and predicted values. At places the prices have been predicted on point as well.

CHAPTER 4.5: Resampling Methods

Forward Selection: It is an iterative method in which I start with having no feature in the model. In each iteration, I keep adding the feature which best improves the model till an addition of a new variable does not improve the performance of the model.

In [33]:
```
k_features = [10,15]
cv_s = [0,10,15]

for k in k_features:
    for cv in cv_s:
        print("Forward Selection with {} features and k-fold = {}".format(k,cv))
        lr_model = LinearRegression()
        forward_selection = SFS(lr_model, k_features=k, forward=True, floating=False, verbose=0, scoring='r2
        forward_selection = forward_selection.fit(X, y)
        forward_features = list(forward_selection.k_feature_names_)
        print("Forward features selected: ", forward_features)
        X_forward =  automotive_data[forward_features]
        y_forward = automotive_data['price']
        X_train_for, X_test_for, y_train_for, y_test_for=train_test_split(X_forward, y_forward, test_size=0.
        lr_for_model = LinearRegression()
        lr_for_model.fit(X_train_for,y_train_for)
        train_mape.append(MAPE(y_train_for,lr_for_model.predict(X_train_for)))
        test_mape.append(MAPE(y_test_for, lr_for_model.predict(X_test_for)))
        print("==============================================================================")
```

```
Forward Selection with 10 features and k-fold = 0
Forward features selected:  ['make', 'fuel-type', 'drive-wheels', 'engine-location', 'width', 'height', 'cur
b-weight', 'engine-size', 'stroke', 'peak-rpm']
==============================================================================
Forward Selection with 10 features and k-fold = 10
Forward features selected:  ['symboling', 'aspiration', 'body-style', 'engine-location', 'width', 'height',
'engine-size', 'bore', 'stroke', 'peak-rpm']
==============================================================================
Forward Selection with 10 features and k-fold = 15
Forward features selected:  ['symboling', 'aspiration', 'num-of-doors', 'engine-location', 'width', 'heigh
t', 'curb-weight', 'engine-size', 'city-mpg', 'highway-mpg']
==============================================================================
Forward Selection with 15 features and k-fold = 0
Forward features selected:  ['make', 'fuel-type', 'num-of-doors', 'body-style', 'drive-wheels', 'engine-loca
tion', 'width', 'height', 'curb-weight', 'engine-type', 'engine-size', 'bore', 'stroke', 'horsepower', 'peak
-rpm']
==============================================================================
Forward Selection with 15 features and k-fold = 10
Forward features selected:  ['symboling', 'aspiration', 'num-of-doors', 'body-style', 'engine-location', 'le
ngth', 'width', 'height', 'curb-weight', 'engine-size', 'bore', 'stroke', 'compression-ratio', 'peak-rpm',
'highway-mpg']
==============================================================================
Forward Selection with 15 features and k-fold = 15
Forward features selected:  ['symboling', 'aspiration', 'num-of-doors', 'engine-location', 'wheel-base', 'wi
dth', 'height', 'curb-weight', 'engine-size', 'fuel-system', 'bore', 'stroke', 'compression-ratio', 'city-mp
g', 'highway-mpg']
==============================================================================
```

Backward Selection: It is an iterative method in which I start with having all the feature in the model. In each iteration, I keep removing the feature which least impact the model till removal of a new variable does not improve the performance of the model.

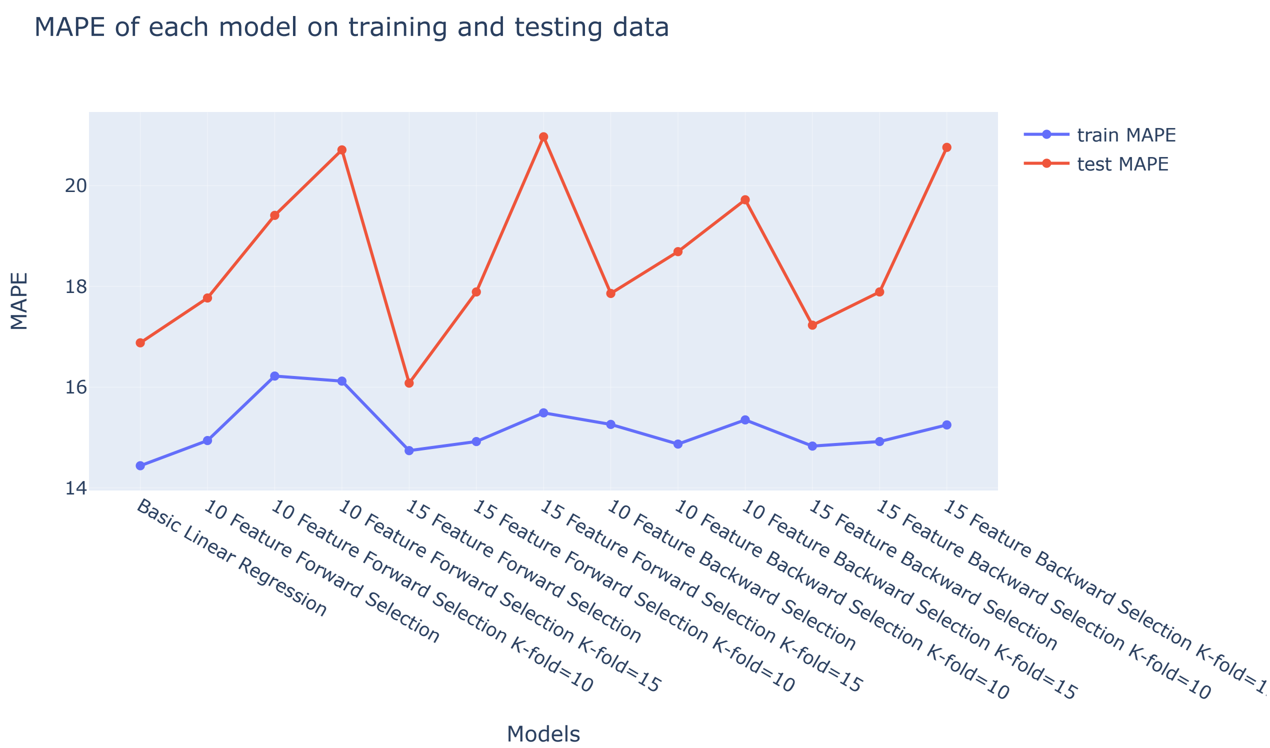
In [34]:
```
k_features = [10,15]
cv_s = [0,10,15]

for k in k_features:
    for cv in cv_s:
        print("Backward Selection with {} features and k-fold = {}".format(k,cv))
        lr_model = LinearRegression()
        backward_selection = SFS(lr_model, k_features=k, forward=False, floating=False, verbose=0, scoring='
        backward_selection = backward_selection.fit(X, y)
        backward_features = list(backward_selection.k_feature_names_)
        print("Backward features selected: ", backward_features)
        X_backward =  automotive_data[backward_features]
        y_backward = automotive_data['price']
        X_train_back, X_test_back, y_train_back, y_test_back = train_test_split(X_backward, y_backward, test_
        lr_back_model = LinearRegression()
        lr_back_model.fit(X_train_back,y_train_back)
        train_mape.append(MAPE(y_train_back,lr_back_model.predict(X_train_back)))
        test_mape.append(MAPE(y_test_back,lr_back_model.predict(X_test_back)))
        print("==============================================================================")
```

```
Backward Selection with 10 features and k-fold = 0
Backward features selected:  ['make', 'body-style', 'drive-wheels', 'engine-location', 'width', 'height', 'c
urb-weight', 'engine-size', 'stroke', 'peak-rpm']
==============================================================================
Backward Selection with 10 features and k-fold = 10
Backward features selected:  ['symboling', 'aspiration', 'body-style', 'engine-location', 'width', 'curb-wei
ght', 'engine-size', 'bore', 'stroke', 'peak-rpm']
==============================================================================
Backward Selection with 10 features and k-fold = 15
Backward features selected:  ['aspiration', 'drive-wheels', 'engine-location', 'wheel-base', 'curb-weight',
'engine-size', 'bore', 'stroke', 'compression-ratio', 'city-mpg']
==============================================================================
Backward Selection with 15 features and k-fold = 0
Backward features selected:  ['make', 'fuel-type', 'body-style', 'drive-wheels', 'engine-location', 'wheel-b
ase', 'width', 'height', 'curb-weight', 'engine-type', 'engine-size', 'bore', 'stroke', 'horsepower', 'peak-
rpm']
==============================================================================
Backward Selection with 15 features and k-fold = 10
Backward features selected:  ['symboling', 'aspiration', 'num-of-doors', 'body-style', 'engine-location', 'l
ength', 'width', 'height', 'curb-weight', 'engine-size', 'bore', 'stroke', 'compression-ratio', 'peak-rpm',
'highway-mpg']
==============================================================================
Backward Selection with 15 features and k-fold = 15
Backward features selected:  ['symboling', 'aspiration', 'num-of-doors', 'drive-wheels', 'engine-location',
'wheel-base', 'height', 'curb-weight', 'engine-size', 'fuel-system', 'bore', 'stroke', 'compression-ratio',
'city-mpg', 'highway-mpg']
==============================================================================
```

In [35]:

```
model = ['Basic Linear Regression', '10 Feature Forward Selection', '10 Feature Forward Selection K-fold=10'
data = {'model':model,'train_mape':train_mape,'test_mape':test_mape}
plot_df = pd.DataFrame(data)
print(plot_df)

fig = go.Figure()
fig.add_trace(go.Scatter(x=plot_df['model'], y=plot_df['train_mape'],mode='lines+markers',name='train MAPE')
fig.add_trace(go.Scatter(x=plot_df['model'], y=plot_df['test_mape'],mode='lines+markers',name='test MAPE'))
fig.update_layout(width=875, title_text="MAPE of each model on training and testing data", yaxis=dict(title_
fig.show()
```

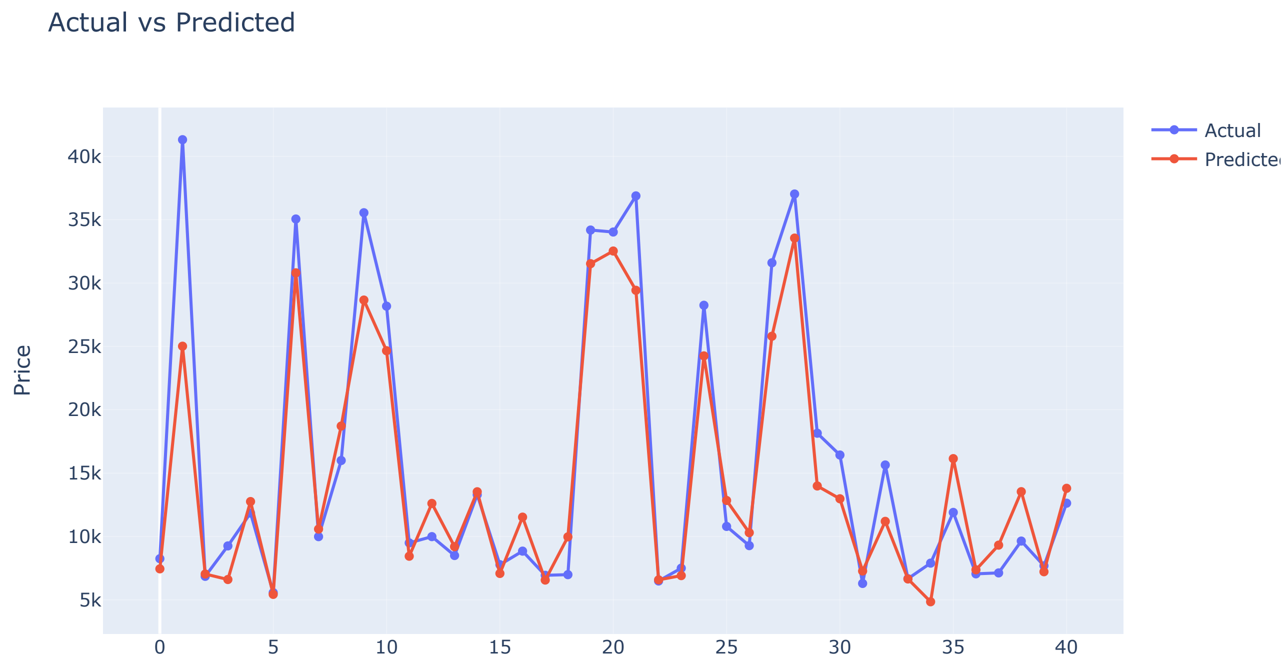
	model	train_mape	test_mape
0	Basic Linear Regression	14.44	16.88
1	10 Feature Forward Selection	14.94	17.77
2	10 Feature Forward Selection K-fold=10	16.22	19.41
3	10 Feature Forward Selection K-fold=15	16.12	20.71
4	15 Feature Forward Selection	14.74	16.08
5	15 Feature Forward Selection K-fold=10	14.92	17.89
6	15 Feature Forward Selection K-fold=15	15.49	20.97
7	10 Feature Backward Selection	15.26	17.86
8	10 Feature Backward Selection K-fold=10	14.87	18.69
9	10 Feature Backward Selection K-fold=15	15.35	19.72
10	15 Feature Backward Selection	14.83	17.23
11	15 Feature Backward Selection K-fold=10	14.92	17.89
12	15 Feature Backward Selection K-fold=15	15.25	20.76



We can see from the above chart that the basic linear regression model scored the best in training data, while the 15 feature forward selection scored the best on testing data. The model that can be selected from the above models should be 15 feature forward selection as it has the best overall MAPE value on training and testing data.

The Predictions based on the best model selected above.

In [36]:
```
lr_model = LinearRegression()
forward_selection = SFS(lr_model, k_features=15, forward=True, floating=False, verbose=0, scoring='r2', cv=0
forward_selection = forward_selection.fit(X, y)
forward_features = list(forward_selection.k_feature_names_)

X_forward = automotive_data[forward_features]
y_forward = automotive_data['price']
X_train_for, X_test_for, y_train_for, y_test_for=train_test_split(X_forward, y_forward, test_size=0.2, randor
lr_for_model = LinearRegression()
lr_for_model.fit(X_train_for,y_train_for)

y_pred = lr_for_model.predict(X_test_for)

data = {'y_pred':y_pred,'y_actual':y_test_for}
plot_df = pd.DataFrame(data)

fig = go.Figure()
fig.add_trace(go.Scatter(y=plot_df['y_actual'],mode='lines+markers',name='Actual'))
fig.add_trace(go.Scatter(y=plot_df['y_pred'],mode='lines+markers',name='Predicted'))
fig.update_layout(width=875, title_text="Actual vs Predicted", yaxis=dict(title_text="Price"))
fig.show()
```



We can see that the results are slightly better than the basic linear regression using all the input features. Using 15 feature forward Selection, more number of data points have been predicted with closer accuracy.

CAHPTER 4.6: Linear Model Selection and Regularization

I perform Ridge Regression that uses Linear least squares with l2 regularization. It estimates the coefficients of multiple-regression models in scenarios where the independent variables are highly correlated.

In [37]:
```
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.2, random_state=42)
alpha_values = [0.5,0.1,0.05,0.01,0.005,0.001]
train_mape = []
test_mape = []

for alpha in alpha_values:
    ridge_model = Ridge(alpha=alpha)
    ridge_model.fit(X_train, y_train)
    train_mape.append(MAPE(y_train,ridge_model.predict(X_train)))
    test_mape.append(MAPE(y_test,ridge_model.predict(X_test)))

data = {'alpha_values':alpha_values,'train_mape':train_mape,'test_mape':test_mape}
plot_df = pd.DataFrame(data)
print(plot_df)

fig = go.Figure()
fig.add_trace(go.Scatter(x=plot_df['alpha_values'], y=plot_df['train_mape'],mode='lines+markers',name='train
fig.add_trace(go.Scatter(x=plot_df['alpha_values'], y=plot_df['test_mape'],mode='lines+markers',name='test ma
fig.update_layout(width=875, title_text="MAPE of model on training and testing data with different alpha valu
fig.show()
```

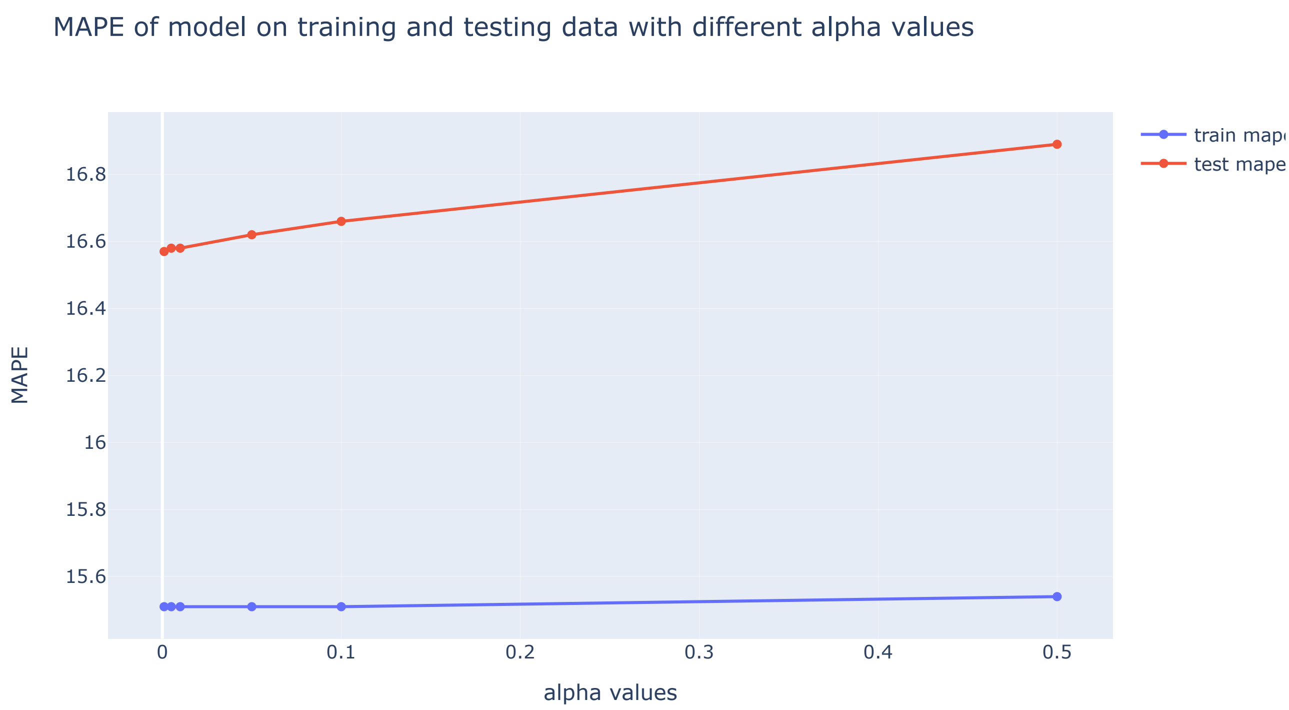
```
   alpha_values  train_mape  test_mape
0         0.500       14.54      17.15
1         0.100       14.43      16.92
2         0.050       14.43      16.89
3         0.010       14.43      16.88
4         0.005       14.44      16.88
5         0.001       14.44      16.88
```

We can see from the above graph that as we keep decreasing the alpha values, the MAPE results keep on improving till the alpha value of 0.01. The accuracy flattens and becomes consistent for lower values of alpha. But this MAPE values is still not better compared to the result of 15 features Linear Regression from forward selection.

I run Ridge Regression on same set of values of alpha as above, on the 15 best feature selected to compare their performance.

In [38]:
```
best_feature = ['symboling', 'make', 'aspiration', 'num-of-doors', 'body-style', 'engine-location', 'width',
X_best =  automotive_data[best_feature]
y_best = automotive_data['price']

X_train, X_test, y_train, y_test=train_test_split(X_best, y_best, test_size=0.2, random_state=42)
alpha_values = [0.5,0.1,0.05,0.01,0.005,0.001]
train_mape = []
test_mape = []

for alpha in alpha_values:
    ridge_model = Ridge(alpha=alpha)
    ridge_model.fit(X_train, y_train)
    train_mape.append(MAPE(y_train,ridge_model.predict(X_train)))
    test_mape.append(MAPE(y_test,ridge_model.predict(X_test)))

data = {'alpha_values':alpha_values,'train_mape':train_mape,'test_mape':test_mape}
plot_df = pd.DataFrame(data)
print(plot_df)

fig = go.Figure()
fig.add_trace(go.Scatter(x=plot_df['alpha_values'], y=plot_df['train_mape'],mode='lines+markers',name='train
fig.add_trace(go.Scatter(x=plot_df['alpha_values'], y=plot_df['test_mape'],mode='lines+markers',name='test ma
fig.update_layout(width=875, title_text="MAPE of model on training and testing data with different alpha valu
fig.show()
```

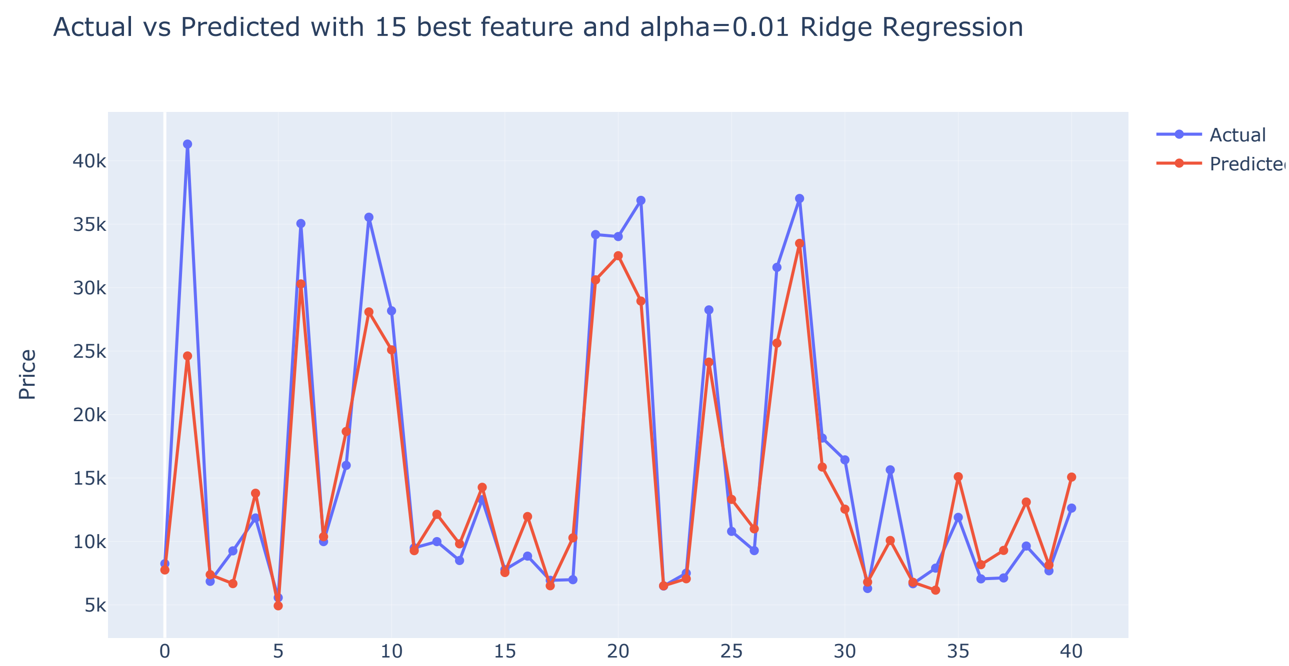
```
   alpha_values  train_mape  test_mape
0         0.500       15.54      16.89
1         0.100       15.51      16.66
2         0.050       15.51      16.62
3         0.010       15.51      16.58
4         0.005       15.51      16.58
5         0.001       15.51      16.57
```



We can see that the accuracy increases with decrease in alpha values. The accuracy becomes constant for alpha lower than 0.01. This form of Ridge regression with 15 best features perfomes better than Ridge Regression model comprising of all the input features.

The Predictions based on the best model selected above, i.e., for Ridge Regression with 15 best features and alpha = 0.01.

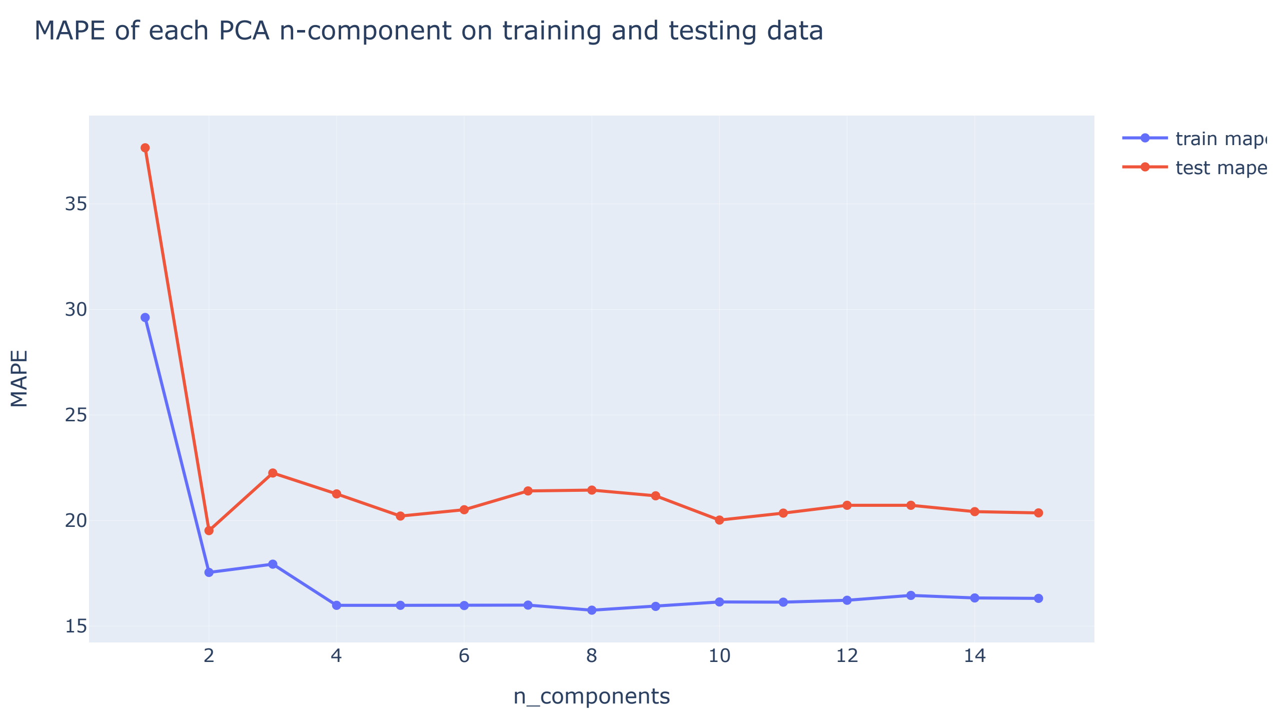
In [39]:
```
X_train, X_test, y_train, y_test=train_test_split(X_best, y_best, test_size=0.2, random_state=42)
ridge_model = Ridge(alpha=alpha)
ridge_model.fit(X_train, y_train)
y_pred = ridge_model.predict(X_test)

data = {'y_pred':y_pred,'y_actual':y_test}
plot_df = pd.DataFrame(data)

fig = go.Figure()
fig.add_trace(go.Scatter(y=plot_df['y_actual'],mode='lines+markers',name='Actual'))
fig.add_trace(go.Scatter(y=plot_df['y_pred'],mode='lines+markers',name='Predicted'))
fig.update_layout(width=875, title_text="Actual vs Predicted with 15 best feature and alpha=0.01 Ridge Regres
fig.show()
```



Moving forward, I perform Principal component analysis (PCA) in order to find how many features gather the maximum amount of information and how they perform on the regression model. PCA is a popular technique for analyzing large datasets containing a high number of dimensions/features per observation, increasing the interpretability of data while preserving the maximum amount of information, and enabling the visualization of multidimensional data.

In [40]:
```
n_components = []
train_mape = []
test_mape = []

for n in range(1,16):
    pca = PCA(n_components=n)
    pca.fit(X)
    x_pca = pca.transform(X)
    X_train, X_test, y_train, y_test = train_test_split(x_pca, y, test_size=0.2, random_state=42)
    lr_model = LinearRegression()
    lr_model.fit(X_train,y_train)
    n_components.append(n)
    train_mape.append(MAPE(y_train, lr_model.predict(X_train)))
    test_mape.append(MAPE(y_test, lr_model.predict(X_test)))

data = {'n_components':n_components,'train_mape':train_mape,'test_mape':test_mape}
plot_df = pd.DataFrame(data)
print(plot_df)

fig = go.Figure()
fig.add_trace(go.Scatter(x=plot_df['n_components'], y=plot_df['train_mape'],mode='lines+markers',name='train
fig.add_trace(go.Scatter(x=plot_df['n_components'], y=plot_df['test_mape'],mode='lines+markers',name='test ma
fig.update_layout(width=875, title_text="MAPE of each PCA n-component on training and testing data", yaxis=di
fig.show()
```

```
    n_components  train_mape  test_mape
0              1       29.62      37.66
1              2       17.54      19.52
2              3       17.93      22.25
3              4       15.98      21.26
4              5       15.98      20.21
5              6       15.97      20.51
6              7       15.99      21.40
7              8       15.75      21.44
8              9       15.93      21.17
9             10       16.14      20.02
10            11       16.13      20.35
11            12       16.22      20.72
12            13       16.45      20.72
13            14       16.33      20.42
14            15       16.31      20.36
```

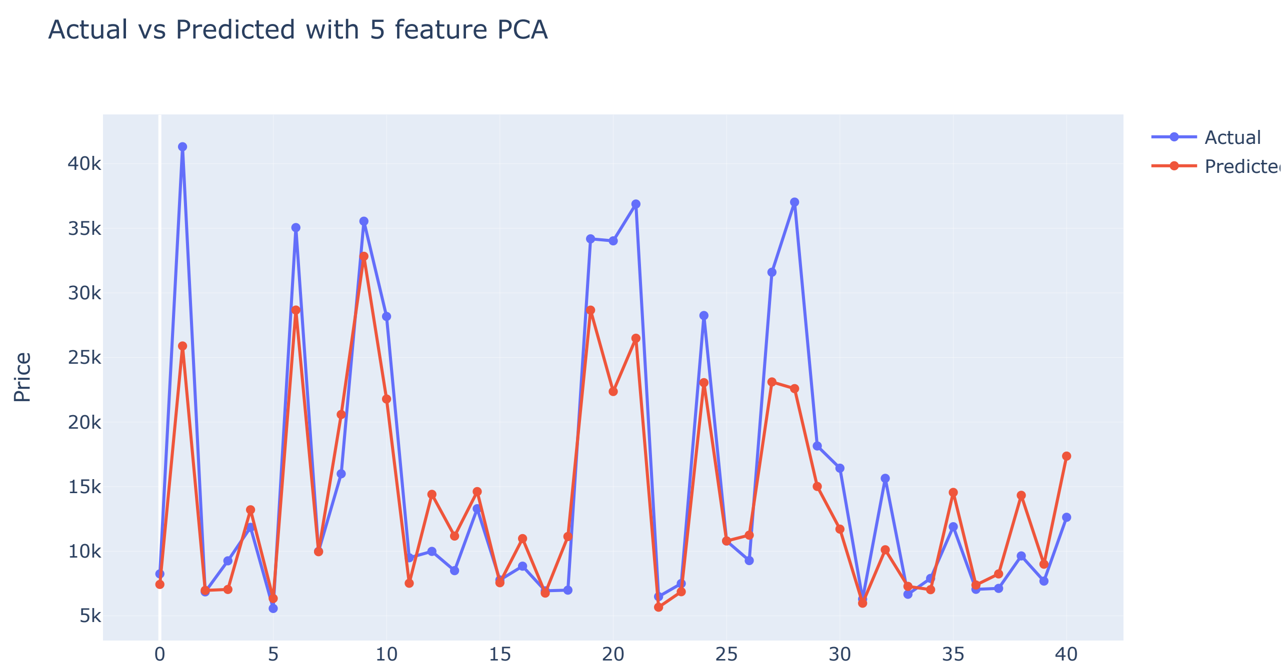


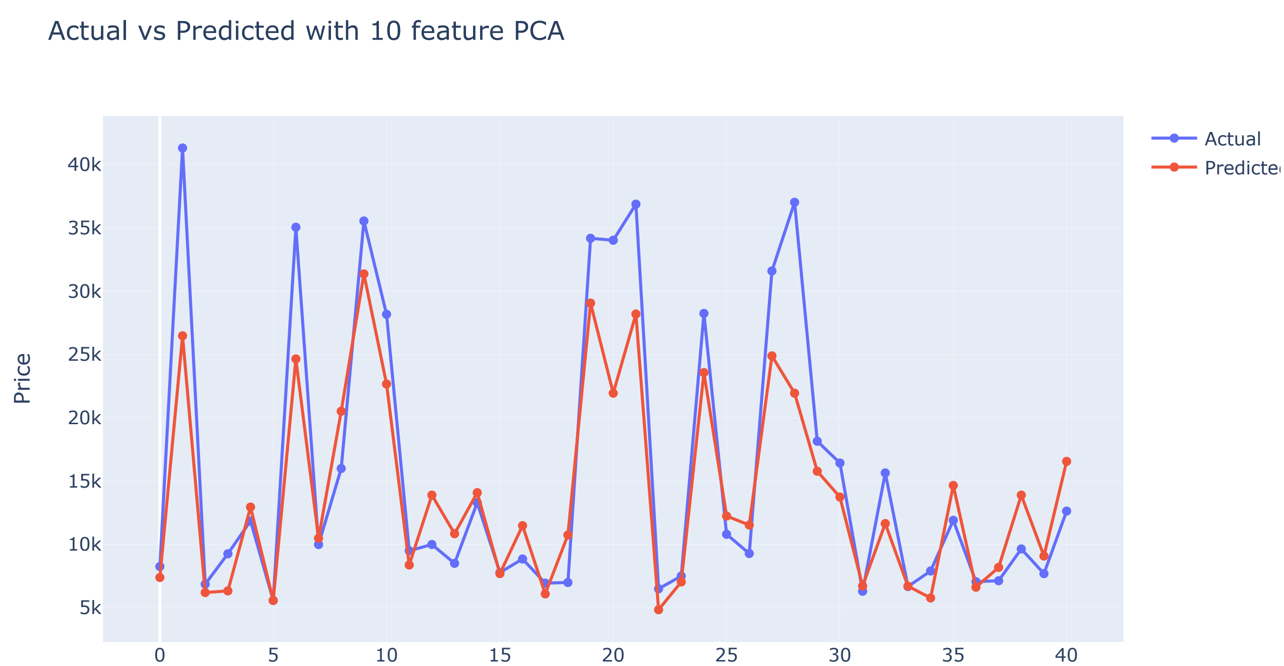
From the above chart we can infer that PCA with n-components greater than 5, starts to overfit the data on training dataset, and later improves at n-components equal to 10 and again starts to overfit for components larger than 10. The best result will be obtained for n-component value of 5 and 10 both as they have comparable MAPE values for both train and test data.

The Predictions based on the best model selected above, i.e., for PCA with n-components=[5,10].

In [41]:
```
pca = PCA(n_components=5)
pca.fit(X)
x_pca = pca.transform(X)
X_train, X_test, y_train, y_test = train_test_split(x_pca, y, test_size=0.2, random_state=42)
lr_model = LinearRegression()
lr_model.fit(X_train,y_train)

y_pred_5 = lr_model.predict(X_test)

pca = PCA(n_components=10)
pca.fit(X)
x_pca = pca.transform(X)
X_train, X_test, y_train, y_test = train_test_split(x_pca, y, test_size=0.2, random_state=42)
lr_model = LinearRegression()
lr_model.fit(X_train,y_train)

y_pred_10 = lr_model.predict(X_test)

data = {'y_pred_pca5':y_pred_5,'y_actual':y_test, 'y_pred_pca10':y_pred_10}
plot_df = pd.DataFrame(data)
```

```
fig = go.Figure()
fig.add_trace(go.Scatter(y=plot_df['y_actual'],mode='lines+markers',name='Actual'))
fig.add_trace(go.Scatter(y=plot_df['y_pred_pca5'],mode='lines+markers',name='Predicted'))
fig.update_layout(width=875, title_text="Actual vs Predicted with 5 feature PCA", yaxis=dict(title_text="Pric
fig.show()

fig = go.Figure()
fig.add_trace(go.Scatter(y=plot_df['y_actual'],mode='lines+markers',name='Actual'))
fig.add_trace(go.Scatter(y=plot_df['y_pred_pca10'],mode='lines+markers',name='Predicted'))
fig.update_layout(width=875, title_text="Actual vs Predicted with 10 feature PCA", yaxis=dict(title_text="Pri
fig.show()
```





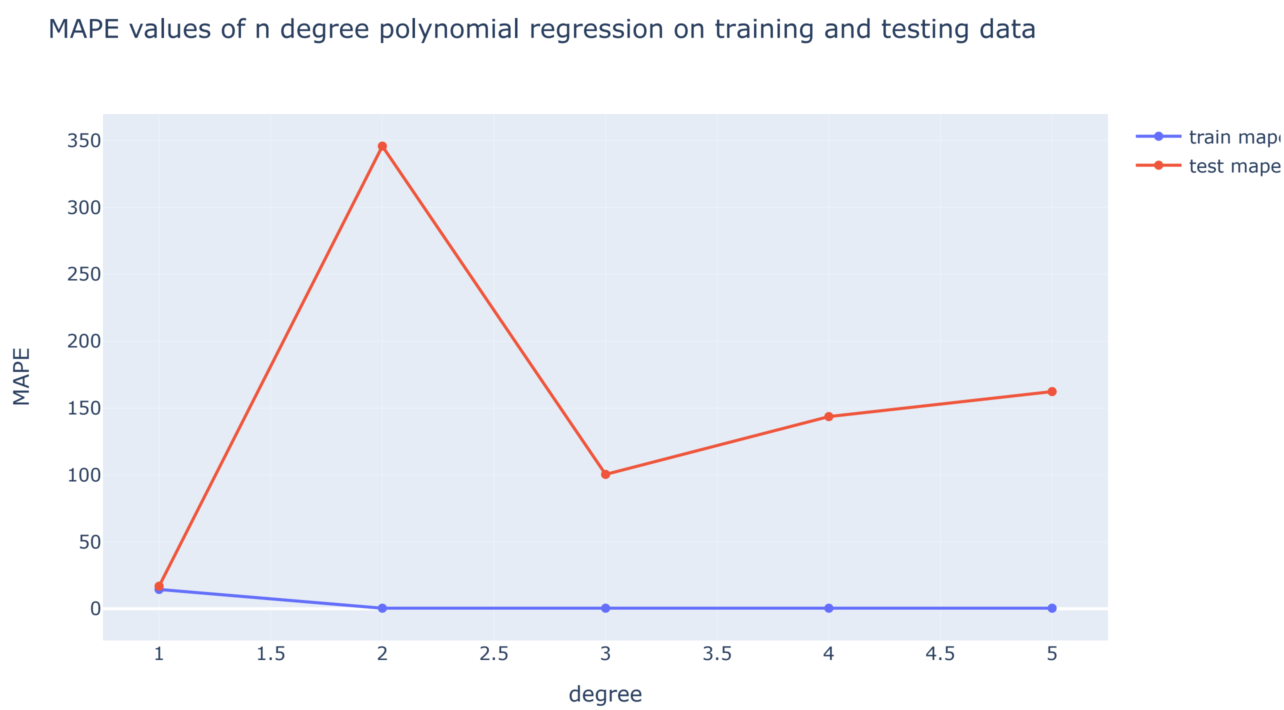
We can see that both n-componenets equal to 5 and 10 perform equally, and have most of the points with the same impact and accuracy, so it would be better to use 5 components as it would reduce the processing time and improve the performance while maintaining the accuracy. However, with increased performance over the basic linear regression, we end up losing a bit of accuracy compared to the basic linear regression.

CHAPTER 4.7: Moving beyond Linearity

I perform poynomial regression, in order to check the model scores compared to just linear regression

In [42]:
```
degree = []
train_mape = []
test_mape = []

for deg in range(1,6):
    poly = PolynomialFeatures(degree=deg, include_bias=False)
    poly_features = poly.fit_transform(X)
    y = automotive_data['price']
    X_train, X_test, y_train, y_test=train_test_split(poly_features, y, test_size=0.2, random_state=42)
    poly_regress_model = LinearRegression()
    poly_regress_model.fit(X_train, y_train)
    degree.append(deg)
    train_mape.append(MAPE(y_train,poly_regress_model.predict(X_train)))
    test_mape.append(MAPE(y_test,poly_regress_model.predict(X_test)))

data = {'polynomial_degree':degree,'train_mape':train_mape,'test_mape':test_mape}
plot_df = pd.DataFrame(data)
print(plot_df)

fig = go.Figure()
fig.add_trace(go.Scatter(x=plot_df['polynomial_degree'], y=plot_df['train_mape'],mode='lines+markers',name='t
fig.add_trace(go.Scatter(x=plot_df['polynomial_degree'], y=plot_df['test_mape'],mode='lines+markers',name='te
fig.update_layout(width=875, title_text="MAPE values of n degree polynomial regression on training and testin
fig.show()
```

```
   polynomial_degree  train_mape  test_mape
0                  1       14.44      16.88
1                  2        0.41     345.80
2                  3        0.41     100.44
3                  4        0.41     143.65
4                  5        0.41     162.33
```



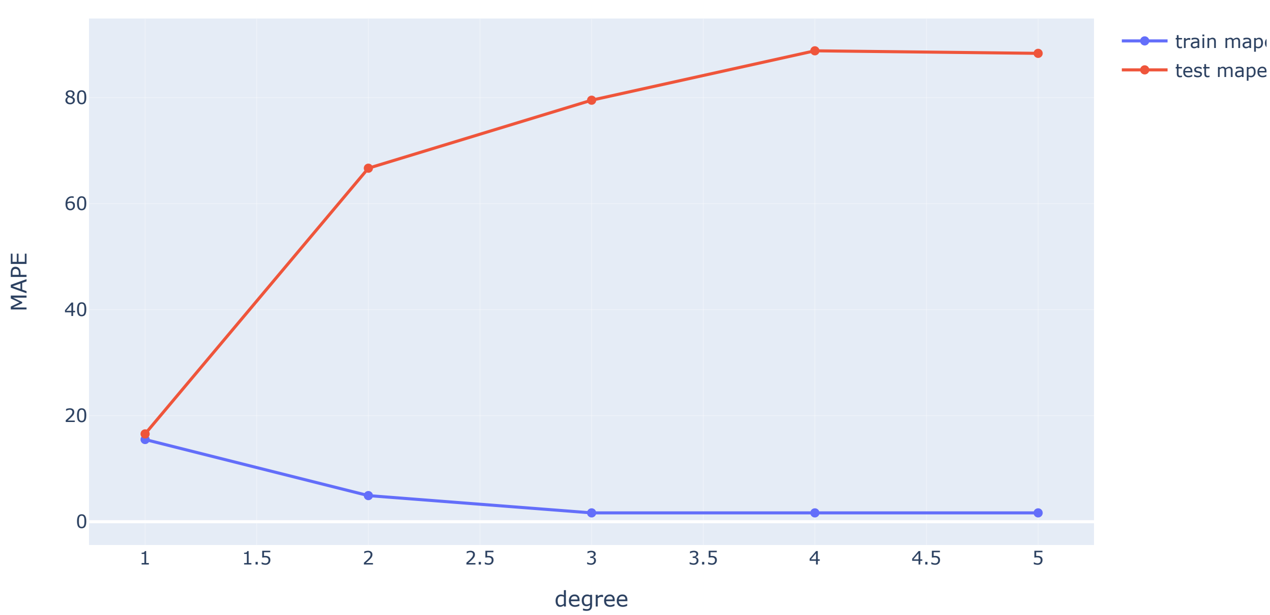
We can observe that the result is best with degree 1 that is linear regression as compared to any other degree in polynomial regression. These are the results with all the features taken into consideration.

Further we work with the best 15 features selected above using forward selection and polynomial regression.

In [43]:
```
best_feature = ['symboling', 'make', 'aspiration', 'num-of-doors', 'body-style', 'engine-location', 'width',
X_best =  automotive_data[best_feature]
y_best = automotive_data['price']

degree = []
train_mape = []
test_mape = []

for deg in range(1,6):
    poly = PolynomialFeatures(degree=deg, include_bias=False)
    poly_features = poly.fit_transform(X_best)
    X_train, X_test, y_train, y_test=train_test_split(poly_features, y_best, test_size=0.2, random_state=42)
    poly_regress_model = LinearRegression()
    poly_regress_model.fit(X_train, y_train)
    degree.append(deg)
    train_mape.append(MAPE(y_train,poly_regress_model.predict(X_train)))
    test_mape.append(MAPE(y_test,poly_regress_model.predict(X_test)))

data = {'polynomial_degree':degree,'train_mape':train_mape,'test_mape':test_mape}
plot_df = pd.DataFrame(data)
print(plot_df)

fig = go.Figure()
fig.add_trace(go.Scatter(x=plot_df['polynomial_degree'], y=plot_df['train_mape'],mode='lines+markers',name='
fig.add_trace(go.Scatter(x=plot_df['polynomial_degree'], y=plot_df['test_mape'],mode='lines+markers',name='te
fig.update_layout(width=875, title_text="MAPE values of n degree polynomial regression on training and testi
fig.show()
```

```
   polynomial_degree  train_mape  test_mape
0                  1       15.51      16.57
1                  2        4.93      66.69
2                  3        1.67      79.52
3                  4        1.67      88.84
4                  5        1.67      88.36
```

MAPE values of n degree polynomial regression on training and testing data with best features



Over here we again observe that the Linear Regression model has the least MAPE value for test data and incresing polynomial degree overfits the data to training dataset.

Conclusion

I have successfully used statistical methods for the analysis of our automotive dataset. I used non-parametric tests to identify the relation (check if their medians are the same) between the numeric input columns and the categorical output columns. Then we did categorical data analysis using the chi-squared test of independence to test the relation between input categorical variables and output categorical variables. I even performed Kruskal-Wallis H-test to compare multiple numerical and categorical data for analysis of variance, followed by Nemenyi Test in order to identify which 2 groups were not following similar distribution.

Further, I ran the basic Linear Regression model in order to predict the price of an automobile on the test split data. I perform various resampling methods, including both forward and backward resampling. I run ridge regression on complete data as well as the best selected features. I perform model selection and regularization in order to fit the best model based on principal component analysis. I even moved beyond linearity by using numerous polynomial degree regression for predicting the price. By implementing all these models and checking for the best result, I come to the conclusion that the best model results were obtained from 15 forward feature selected Linear Regression Model, and it had the best MAPE values on both training and testing dataset.

References

1. https://www.kaggle.com/datasets/toramky/automobile-dataset?resource=download&sort=votes
2. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.shapiro.html
3. https://www.statsmodels.org/dev/generated/statsmodels.stats.weightstats.ztest.html
4. https://pingouin-stats.org/generated/pingouin.multivariate_normality.html
5. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.kruskal.html
6. https://scikit-posthocs.readthedocs.io/en/latest/generated/scikit_posthocs.posthoc_nemenyi/
7. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.chi2_contingency.html
8. https://scikit-learn.org/0.20/modules/generated/sklearn.linear_model.LinearRegression.html
9. https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SequentialFeatureSelector.html
10. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html
11. https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
12. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html