In [1]:
```
import pandas
import numpy
```

In [2]:
```
data = pandas.read_csv(r'OneDrive\Desktop\cars.csv')
```

In [3]:
```
data
```

Out[3]:

	Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	EngineSize	Cylinders	Horsepow
0	Acura	MDX	SUV	Asia	AWD	36945	33337	3.5	6.0	2
1	Acura	RSX Type S 2dr	Sedan	Asia	FWD	23820	21761	2.0	4.0	2
2	Acura	TSX 4dr	Sedan	Asia	FWD	26990	24647	2.4	4.0	2
3	Acura	TL 4dr	Sedan	Asia	FWD	33195	30299	3.2	6.0	2
4	Acura	3.5 RL 4dr	Sedan	Asia	FWD	43755	39014	3.5	6.0	2
...	...	...	...	...	...	...	...	...	...	
423	Volvo	C70 LPT convertible 2dr	Sedan	Europe	FWD	40565	38203	2.4	5.0	1
424	Volvo	C70 HPT convertible 2dr	Sedan	Europe	FWD	42565	40083	2.3	5.0	2
425	Volvo	S80 T6 4dr	Sedan	Europe	FWD	45210	42573	2.9	6.0	2
426	Volvo	V40	Wagon	Europe	FWD	26135	24641	1.9	4.0	1
427	Volvo	XC70	Wagon	Europe	AWD	35145	33112	2.5	5.0	2

428 rows × 15 columns

Question 1

Cramer's V

In [15]:
```
col = ['Make', 'Origin', 'Type']
df = data[col]
df
```

Out[15]:

	Make	Origin	Type
0	Acura	Asia	SUV
1	Acura	Asia	Sedan
2	Acura	Asia	Sedan
3	Acura	Asia	Sedan
4	Acura	Asia	Sedan
...	...	...	...
423	Volvo	Europe	Sedan
424	Volvo	Europe	Sedan
425	Volvo	Europe	Sedan
426	Volvo	Europe	Wagon
427	Volvo	Europe	Wagon

428 rows × 3 columns

In [129]:
```
from itertools import combinations, product, chain
```

In [26]:
```
a = combinations(col, 2)
lst = [a for a in a]
lst
```

Out[26]:
```
[('Make', 'Origin'), ('Make', 'Type'), ('Origin', 'Type')]
```

In [48]:
```
def cramer_V(obsCount):
    xNCat = obsCount.shape[0]
    yNCat = obsCount.shape[1]
    cTotal = obsCount.sum(axis = 1)
    rTotal = obsCount.sum(axis = 0)
    nTotal = numpy.sum(rTotal)
    expCount = numpy.outer(cTotal, (rTotal / nTotal))
    # Calculate the Chi-Square statistics
    chiSq_stat = ((obsCount - expCount)**2 / expCount).to_numpy().sum()
    chiSq_DF = (xNCat - 1) * (yNCat - 1)
    if (chiSq_DF > 0):
        chiSq_pvalue = chi2.sf(chiSq_stat, chiSq_DF)
        cramerv = chiSq_stat / nTotal / (min(xNCat, yNCat) - 1.0)
        cramerv = numpy.sqrt(cramerv)
    else:
        chiSq_pvalue = numpy.NaN
        cramerv = numpy.NaN
    return cramerv
```

In [46]:
```
from scipy.stats import chi2, t
```

In [49]:
```
for a in lst:
    col = []
    col.append(a[0])
    col.append(a[1])
    df1 = df[col]
    cont_table = pandas.crosstab(index=df1[col[0]], columns=df[col[1]])
    c_stats = cramer_V(cont_table)
    print("The cramer's V of vaiables: ", col, " is: ", c_stats)
```

```
The cramer's V of vaiables:  ['Make', 'Origin']  is:  1.0
The cramer's V of vaiables:  ['Make', 'Type']  is:  0.37019054302046817
The cramer's V of vaiables:  ['Origin', 'Type']  is:  0.2041219966860013
```

Pearson Correlation

In [51]:
```
col = ['Horsepower', 'Length', 'Weight', 'Wheelbase']
a = combinations(col, 2)
lst = [a for a in a]
lst
```

Out[51]:
```
[('Horsepower', 'Length'),
 ('Horsepower', 'Weight'),
 ('Horsepower', 'Wheelbase'),
 ('Length', 'Weight'),
 ('Length', 'Wheelbase'),
 ('Weight', 'Wheelbase')]
```

In [60]:
```
df2 = data[col]
df2
```

Out[60]:

	Horsepower	Length	Weight	Wheelbase
0	265	189	4451	106
1	200	172	2778	101
2	200	183	3230	105
3	270	186	3575	108
4	225	197	3880	115
...	...	...	...	...
423	197	186	3450	105
424	242	186	3450	105
425	268	190	3653	110
426	170	180	2822	101
427	208	186	3823	109

428 rows × 4 columns

In [64]:
```
def pearson_corr(a):
    col = []
    col.append(a[0])
    col.append(a[1])
    df = df2[col]
    n = df.shape[0]
    xyCov = numpy.cov(df[col[0]], df[col[1]], ddof = 0)
    pCorr_sq = (xyCov[0,1] / xyCov[0,0]) *  (xyCov[0,1] / xyCov[1,1])
    pCorr = numpy.sqrt(pCorr_sq)
    return pCorr
```

In [66]:
```
for a in lst:
    p_corr = pearson_corr(a)
    print("The pearson correlation of vaiables: ", a, " is: ", p_corr)
```

```
The pearson correlation of vaiables:  ('Horsepower', 'Length')  is:  0.38155
388132658946
The pearson correlation of vaiables:  ('Horsepower', 'Weight')  is:  0.63079
58167406753
The pearson correlation of vaiables:  ('Horsepower', 'Wheelbase')  is:  0.38
739778269895586
The pearson correlation of vaiables:  ('Length', 'Weight')  is:  0.690020710
9097168
The pearson correlation of vaiables:  ('Length', 'Wheelbase')  is:  0.889194
6668509833
The pearson correlation of vaiables:  ('Weight', 'Wheelbase')  is:  0.760702
758886357
```

Eta-Squares

In [80]:
```
from scipy.stats import chi2, f, t
```

In [81]:
```
def AnalysisOfVarianceTest(xCat, yCont):
   df = pandas.DataFrame(columns = ['x', 'y'])
   df['x'] = xCat
   df['y'] = yCont
   # Total Count and Sum of Squares
   totalCount = df['y'].count()
   totalSSQ = df['y'].var(ddof = 0) * totalCount
   # Within Group Count and Sums of Squares
   groupCount = df.groupby('x').count()
   groupSSQ = df.groupby('x').var(ddof = 0) * groupCount
   nGroup = groupCount.shape[0]
   withinSSQ = numpy.sum(groupSSQ.values)
   betweenSSQ = max(0.0, (totalSSQ - withinSSQ))
   if (totalSSQ > 0.0):
      etasq = betweenSSQ / totalSSQ
   else:
      etasq = numpy.NaN

   # Compute F statistics
   fDf1 = (nGroup - 1)
   fDf2 = (totalCount - nGroup)
   if (fDf1 > 0 and fDf2 > 0 and withinSSQ > 0.0):
      fStat = (betweenSSQ / fDf1) / (withinSSQ / fDf2)
      fSig = f.sf(fStat, fDf1, fDf2)
   else:
      fStat = numpy.NaN
      fSig = numpy.NaN
   outlist = [nGroup, fStat, fDf1, fDf2, fSig, etasq]
   return outlist[5]
```

In [76]:
```
col_cont = ['Horsepower', 'Length', 'Weight', 'Wheelbase']
col_cat = ['Make', 'Origin', 'Type']
```

In [82]:
```
comb = list(product(col_cat, col_cont))
```

In [84]:
```
for a in comb:
    col = []
    col.append(a[0])
    col.append(a[1])
    xCat = data[col[0]]
    yCont = data[col[1]]
    eta_sq = AnalysisOfVarianceTest(xCat, yCont)
    print("The Eta statistics of variables ", a, " is: ",eta_sq)
```

```
The Eta statistics of variables  ('Make', 'Horsepower')  is:  0.379377713535
01754
The Eta statistics of variables  ('Make', 'Length')  is:  0.3332456993112077
3
The Eta statistics of variables  ('Make', 'Weight')  is:  0.2842022448303112
The Eta statistics of variables  ('Make', 'Wheelbase')  is:  0.3227504206101
0345
The Eta statistics of variables  ('Origin', 'Horsepower')  is:  0.1184777186
4192043
The Eta statistics of variables  ('Origin', 'Length')  is:  0.14727842041765
79
The Eta statistics of variables  ('Origin', 'Weight')  is:  0.07027988993772
898
The Eta statistics of variables  ('Origin', 'Wheelbase')  is:  0.11418806981
793002
The Eta statistics of variables  ('Type', 'Horsepower')  is:  0.166846258729
64987
The Eta statistics of variables  ('Type', 'Length')  is:  0.2395260109360784
5
The Eta statistics of variables  ('Type', 'Weight')  is:  0.2940396144123072
The Eta statistics of variables  ('Type', 'Wheelbase')  is:  0.3183338767676
6267
```

From the Cramer's V statistics, we can see the statistic of variables ('Make', 'Origin') equals to 1, which indicates the strong association between these two variables. For the Pearson Correlation, if we regard the correlation coefficient higher than 0.7 as high association, we can observe the strong association from two pairs of variables which are ('Length', 'Wheelbase') and ('Weight', 'Wheelbase'). From the Eta test, we can observe that the greatest statistic is 0.37937771 by variables ('Make', 'Horsepower'), which means about 38% of variance can be explained.

Question 2

In [101]:
```
from sklearn.preprocessing import LabelEncoder
```

In [112]:
```
#Transfer the categorical variables by LabelEncoder
labelencoder = LabelEncoder()
df['Make'] = labelencoder.fit_transform(data[['Make']])
df['Origin'] = labelencoder.fit_transform(data[['Origin']])
df['Type'] = labelencoder.fit_transform(data[['Type']])
df
```

```
E:\Anaconda\lib\site-packages\sklearn\preprocessing\_label.py:115: DataConve
rsionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
E:\Anaconda\lib\site-packages\sklearn\preprocessing\_label.py:115: DataConve
rsionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
E:\Anaconda\lib\site-packages\sklearn\preprocessing\_label.py:115: DataConve
rsionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[112]:

	Make	Origin	Type	Horsepower	Length	Weight	Wheelbase
0	0	0	1	265	189	4451	106
1	0	0	2	200	172	2778	101
2	0	0	2	200	183	3230	105
3	0	0	2	270	186	3575	108
4	0	0	2	225	197	3880	115
...	...	...	...	...	...	...	...
423	37	1	2	197	186	3450	105
424	37	1	2	242	186	3450	105
425	37	1	2	268	190	3653	110
426	37	1	5	170	180	2822	101
427	37	1	5	208	186	3823	109

428 rows × 7 columns

In [207]:
```
def SWEEPOperator (pDim, inputM, origDiag, sweepCol = None, tol = 1e-7):
    ''' Implement the SWEEP operator
    Parameter
    ---------
    pDim: dimension of matrix inputM, integer greater than one
    inputM: a square and symmetric matrix, numpy array
    origDiag: the original diagonal elements before any SWEEPing
    sweepCol: a list of columns numbers to SWEEP
    tol: singularity tolerance, positive real
    Return
    ------
    A: negative of a generalized inverse of input matrix
    aliasParam: a list of aliased rows/columns in input matrix
    nonAliasParam: a list of non-aliased rows/columns in input matrix
    '''
    if (sweepCol is None):
        sweepCol = range(pDim)
    aliasParam = []
    nonAliasParam = []
    A = numpy.copy(inputM)
    ANext = numpy.zeros((pDim,pDim))
    for k in sweepCol:
        Akk = A[k,k]
        pivot = tol * abs(origDiag[k])
        if (not numpy.isinf(Akk) and abs(Akk) >= pivot and pivot > 0.0):
            nonAliasParam.append(k)
            ANext = A - numpy.outer(A[:, k], A[k, :]) / Akk
            ANext[:, k] = A[:, k] / abs(Akk)
            ANext[k, :] = ANext[:, k]
            ANext[k, k] = -1.0 / Akk
        else:
            aliasParam.append(k)
            ANext[:,k] = numpy.zeros(pDim)
            ANext[k, :] = numpy.zeros(pDim)
        A = ANext
    return (A, aliasParam, nonAliasParam)
def LinearRegression (X, y):
    ''' Train a linear regression model
    Parameter
    ---------
    X: A Pandas DataFrame, rows are observations, columns are regressors
    y: A Pandas Series, rows are observations of the response variable
    Return
    ------
    A list of the following entities:
    1. b: an array of regression coefficient
    2. residual_SS: residual sum of squares
    3. XtX_Ginv: a generalized inverse of the XtX matrix
    4. aliasParam: a list of aliased rows/columns in input matrix
    5. nonAliasParam: a list of non-aliased rows/columns in input matrix
    '''
    # X: A Pandas DataFrame, rows are observations, columns are regressors
    # y: A Pandas Series, rows are observations of the response variable
    Z = X.join(y)
    n_sample = Z.shape[0]
    n_param = Z.shape[1] - 1
```

```
    ZtZ = Z.transpose().dot(Z)
    diag_ZtZ = numpy.diagonal(ZtZ)
    eps_double = numpy.finfo(numpy.float64).eps
    tol = numpy.sqrt(eps_double)
    ZtZ_transf, aliasParam, nonAliasParam = SWEEPOperator ((n_param+1), ZtZ,
diag_ZtZ, sweepCol = range(n_param), tol = tol)
    b = ZtZ_transf[0:n_param, n_param]
    b[aliasParam] = 0.0
    XtX_Ginv = - ZtZ_transf[0:n_param, 0:n_param]
    XtX_Ginv[:, aliasParam] = 0.0
    XtX_Ginv[aliasParam, :] = 0.0
    residual_SS = ZtZ_transf[n_param, n_param]
    return ([b, residual_SS, XtX_Ginv, aliasParam, nonAliasParam])
```

In [208]:

```
# The FSig is the sixth element in each row of the FTest
def takeFSig(s):
    return s[6]
n_sample = data.shape[0]
X = df[['Make', 'Origin', 'Type', 'Horsepower', 'Length', 'Weight', 'Wheelbas
X.insert(0, 'Intercept', 1.0)
y = data['MPG_City']
```

In [209]:

```
enter_threshold = 0.05
q_show_diary = True
step_diary = []
var_in_model = ['Intercept']
# Step 0: Enter Intercept
result_list = LinearRegression(X[var_in_model], y)
m0 = len(result_list[4])
SSE0 = result_list[1]
step_diary.append([0, 'Intercept', SSE0, m0] + 4 * [numpy.nan])
# Forward Selection Steps
candidate = X.columns
candidate = candidate.to_list()
for iStep in range(len(candidate)):
    FTest = []
    for pred in candidate:
        work_list = var_in_model.copy()
        work_list.append(pred)
        result_list = LinearRegression(X[work_list], y)
        m1 = len(result_list[4])
        SSE1 = result_list[1]
        df_numer = m1 - m0
        df_denom = n_sample - m1
        if (df_numer > 0 and df_denom > 0):
            FStat = ((SSE0 - SSE1) / df_numer) / (SSE1 / df_denom)
            FSig = f.sf(FStat, df_numer, df_denom)
            FTest.append([pred, SSE1, m1, FStat, df_numer, df_denom, FSig])
    # Show F Test results for the current step
    if (q_show_diary):
        print('\n===== F Test Results for the Current Forward Step =====')
        print('Step Number: ', iStep)
        print('Step Diary:')
        print('[Variable Candidate | Residual Sum of Squares | N Non-Aliased
        for row in FTest:
            print(row)
    FTest.sort(key = takeFSig, reverse = False)
    FSig = takeFSig(FTest[0])
    if (FSig <= enter_threshold):
        enter_var = FTest[0][0]
        SSE0 = FTest[0][1]
        m0 = FTest[0][2]
        step_diary.append([iStep+1] + FTest[0])
        var_in_model.append(enter_var)
        candidate.remove(enter_var)
    else:
        break
forward_summary = pandas.DataFrame(step_diary, columns = ['Step', 'Variable E
```

```
===== F Test Results for the Current Forward Step =====
Step Number:  0
Step Diary:
[Variable Candidate | Residual Sum of Squares | N Non-Aliased Parameters |
F Stat | F DF1 | F DF2 | F Sig]
['Make', 11402.98265660611, 2, 11.709615912372932, 1, 426, 0.0006818602217
303737]
```

```
['Origin', 11041.993129835388, 2, 26.01940448525872, 1, 426, 5.09833745584
2681e-07]
['Type', 11678.895069532251, 2, 1.3687818207627174, 1, 426, 0.242675420260
11696]
['Horsepower', 6351.214305686575, 2, 359.8647053382581, 1, 426, 1.28990637
68087322e-58]
['Length', 8769.404017767567, 2, 143.1601332047103, 1, 426, 1.210652858859
8324e-28]
['Weight', 5335.730401687142, 2, 509.4286635809614, 1, 426, 9.058801587355
108e-75]
['Wheelbase', 8701.353690708083, 2, 147.61134098116926, 1, 426, 2.27780644
26503926e-29]

===== F Test Results for the Current Forward Step =====
Step Number:  1
Step Diary:
[Variable Candidate | Residual Sum of Squares | N Non-Aliased Parameters |
F Stat | F DF1 | F DF2 | F Sig]
['Make', 5307.91733012613, 3, 2.2269667514112053, 1, 425, 0.13636177806849
098]
['Origin', 5299.655309969346, 3, 2.893002107367663, 1, 425, 0.089696490405
62495]
['Type', 5089.341487859362, 3, 20.575410124591777, 1, 425, 7.4684597893717
35e-06]
['Horsepower', 4467.790547901631, 3, 82.56305525156951, 1, 425, 3.91540892
560443e-18]
['Length', 5334.409397499521, 3, 0.10524628649650716, 1, 425, 0.7457820818
048477]
['Wheelbase', 5254.37459456236, 3, 6.580463080004762, 1, 425, 0.0106525908
73070503]

===== F Test Results for the Current Forward Step =====
Step Number:  2
Step Diary:
[Variable Candidate | Residual Sum of Squares | N Non-Aliased Parameters |
F Stat | F DF1 | F DF2 | F Sig]
['Make', 4454.955362135424, 4, 1.2215877203005074, 1, 424, 0.2696760513943
405]
['Origin', 4420.013385666142, 4, 4.583134714827137, 1, 424, 0.032857783961
42617]
['Type', 4339.303836586917, 4, 12.554632643628999, 1, 424, 0.0004391467021
512636]
['Length', 4464.98264470401, 4, 0.2666417880040734, 1, 424, 0.605862634762
4214]
['Wheelbase', 4454.288377946746, 4, 1.285260309865766, 1, 424, 0.257563847
963612]

===== F Test Results for the Current Forward Step =====
Step Number:  3
Step Diary:
[Variable Candidate | Residual Sum of Squares | N Non-Aliased Parameters |
F Stat | F DF1 | F DF2 | F Sig]
['Make', 4323.185901101629, 5, 1.5770514769072659, 1, 423, 0.2098791165733
5198]
['Origin', 4299.039610050952, 5, 3.9617610837763704, 1, 423, 0.04718810766
929923]
['Length', 4339.290196585499, 5, 0.0013296461721700016, 1, 423, 0.97092933
```

```
48689463]
['Wheelbase', 4308.727752871613, 5, 3.0017406885254467, 1, 423, 0.08390398
62368399]

===== F Test Results for the Current Forward Step =====
Step Number:  4
Step Diary:
[Variable Candidate | Residual Sum of Squares | N Non-Aliased Parameters |
F Stat | F DF1 | F DF2 | F Sig]
['Make', 4295.6146111921435, 6, 0.3364709475220512, 1, 422, 0.562182950019
6787]
['Length', 4297.546457343247, 6, 0.1466209728983586, 1, 422, 0.70197841821
45142]
['Wheelbase', 4253.191309341122, 6, 4.549050699190307, 1, 422, 0.033513490
28806664]

===== F Test Results for the Current Forward Step =====
Step Number:  5
Step Diary:
[Variable Candidate | Residual Sum of Squares | N Non-Aliased Parameters |
F Stat | F DF1 | F DF2 | F Sig]
['Make', 4246.696920024407, 7, 0.6438269445236372, 1, 421, 0.4227810984197
0224]
['Length', 4216.436869507971, 7, 3.6698330008584406, 1, 421, 0.05608279292
63456]
```

According to the final result, we have our final model MPG_City ~ Intercept + Origin + Type + Horsepower + Weight + Wheelbase.

In [210]:
```
def tss(a):
    m = a.mean()
    n = 0
    for i in a:
        n += ((i-m)**2)
    return (n)
```

In [211]:
```
X_1 = df[['Origin', 'Type', 'Horsepower', 'Weight', 'Wheelbase']]
X_1.insert(0, 'Intercept', 1.0)
X_1
```

Out[211]:

	Intercept	Origin	Type	Horsepower	Weight	Wheelbase
0	1.0	0	1	265	4451	106
1	1.0	0	2	200	2778	101
2	1.0	0	2	200	3230	105
3	1.0	0	2	270	3575	108
4	1.0	0	2	225	3880	115
...	...	...	...	...	...	...
423	1.0	1	2	197	3450	105
424	1.0	1	2	242	3450	105
425	1.0	1	2	268	3653	110
426	1.0	1	5	170	2822	101
427	1.0	1	5	208	3823	109

428 rows × 6 columns

In [212]:
```
result_list = LinearRegression(X_1, y)
```

In [213]:
```
residual_SS = result_list[1]
r_sq = 1 - (residual_SS/tss(y))
r_sq
```

Out[213]:
```
0.636988849342762
```

The R-squared is 0.636988849342762

Question 3

In [230]:
```
def all_subsets(v):
    return chain(*map(lambda x: combinations(v, x), range(0, len(v)+1)))
values = ['Make', 'Origin', 'Type', 'Horsepower', 'Length', 'Weight', 'Wheelb
subset = pandas.DataFrame(all_subsets(values))
subset
```

Out[230]:

	0	1	2	3	4	5	6
0	None	None	None	None	None	None	None
1	Make	None	None	None	None	None	None
2	Origin	None	None	None	None	None	None
3	Type	None	None	None	None	None	None
4	Horsepower	None	None	None	None	None	None
...	...	...	...	...	...	...	...
123	Make	Origin	Type	Length	Weight	Wheelbase	None
124	Make	Origin	Horsepower	Length	Weight	Wheelbase	None
125	Make	Type	Horsepower	Length	Weight	Wheelbase	None
126	Origin	Type	Horsepower	Length	Weight	Wheelbase	None
127	Make	Origin	Type	Horsepower	Length	Weight	Wheelbase

128 rows × 7 columns

In [231]:
```
len(subset.index)
```

Out[231]:
```
128
```

In [232]:
```
import math
```

In [233]:
```
aic_lst = []
r_sq_lst = []
permu_lst = []
for i in subset.index:
    lst_var = subset.loc[i].to_list()
    subset_var = []
    for val in lst_var:
        if val != None :
            subset_var.append(val)
    permu_lst.append(subset_var)
    X_2 = df[subset_var]
    n_sample1 = X_2.shape[0]
    y = data['MPG_City']
    X_2.insert(0, 'Intercept', 1.0)
    return_list1 = LinearRegression(X_2, y)
    residual_SS1 = return_list1[1]
    no_of_nonaliased = len(return_list1[4])
    pop_var = residual_SS1/n_sample1
    AIC = n_sample * math.log(pop_var) + 2.0 * no_of_nonaliased
    r_sq = 1 - (residual_SS1/tss(y))
    aic_lst.append(AIC)
    r_sq_lst.append(r_sq)
```

In [423]:
```
dff = pandas.DataFrame(aic_lst, columns=['AIC'])
dff['R-Squared'] = r_sq_lst
order_lst = [x for x in range(128)]
dff['Subset_order'] = order_lst
```

In [424]:
```
dff.sort_values(by=['AIC'])
```

Out[424]:

	AIC	R-Squared	Subset_order
126	993.102435	6.401259e-01	126
127	993.933856	6.411071e-01	127
125	994.564468	6.388944e-01	125
116	994.817119	6.369888e-01	116
119	995.684644	6.362523e-01	119
...	...	...	...
14	1395.852497	6.040325e-02	14
1	1408.913014	2.675202e-02	1
9	1409.092271	3.088349e-02	9
0	1418.518819	-2.664535e-15	0
3	1419.145815	3.202812e-03	3

128 rows × 3 columns

In [425]:
```
subset.loc[126]
```

Out[425]:
```
0        Origin
1          Type
2    Horsepower
3        Length
4        Weight
5     Wheelbase
6          None
Name: 126, dtype: object
```

Save to Excel

In [426]:
```
file_name = 'R-squared.xlsx'

# saving the excel
dff.to_excel(file_name)
```

Question 4

In [427]:
```
dff
```

Out[427]:

	AIC	R-Squared	Subset_order
0	1418.518819	-2.664535e-15	0
1	1408.913014	2.675202e-02	1
2	1395.144512	5.756258e-02	2
3	1419.145815	3.202812e-03	3
4	1158.435030	4.579220e-01	4
...	...	...	...
123	1049.644141	5.893008e-01	123
124	1006.274599	6.288781e-01	124
125	994.564468	6.388944e-01	125
126	993.102435	6.401259e-01	126
127	993.933856	6.411071e-01	127

128 rows × 3 columns

In [428]:
```
dff['permu'] = permu_lst
dff
```

Out[428]:

	AIC	R-Squared	Subset_order	permu
0	1418.518819	-2.664535e-15	0	[]
1	1408.913014	2.675202e-02	1	[Make]
2	1395.144512	5.756258e-02	2	[Origin]
3	1419.145815	3.202812e-03	3	[Type]
4	1158.435030	4.579220e-01	4	[Horsepower]
...	...	...	...	...
123	1049.644141	5.893008e-01	123	[Make, Origin, Type, Length, Weight, Wheelbase]
124	1006.274599	6.288781e-01	124	[Make, Origin, Horsepower, Length, Weight, Whe...
125	994.564468	6.388944e-01	125	[Make, Type, Horsepower, Length, Weight, Wheel...
126	993.102435	6.401259e-01	126	[Origin, Type, Horsepower, Length, Weight, Whe...
127	993.933856	6.411071e-01	127	[Make, Origin, Type, Horsepower, Length, Weigh...

128 rows × 4 columns

In [429]:
```
from itertools import permutations
p = permutations(['Make', 'Origin', 'Type', 'Horsepower', 'Length', 'Weight',
perm_index = []
for i in p:
    perm_index.append(list(i))
perm_index
```

```
e'],
 ['Make', 'Origin', 'Type', 'Length', 'Weight', 'Wheelbase', 'Horsepowe
r'],

 ['Make', 'Origin', 'Type', 'Length', 'Wheelbase', 'Horsepower', 'Weigh
t'],
 ['Make', 'Origin', 'Type', 'Length', 'Wheelbase', 'Weight', 'Horsepowe
r'],
 ['Make', 'Origin', 'Type', 'Weight', 'Horsepower', 'Length', 'Wheelbas
e'],
 ['Make', 'Origin', 'Type', 'Weight', 'Horsepower', 'Wheelbase', 'Lengt
h'],
 ['Make', 'Origin', 'Type', 'Weight', 'Length', 'Horsepower', 'Wheelbas
e'],
 ['Make', 'Origin', 'Type', 'Weight', 'Length', 'Wheelbase', 'Horsepowe
r'],
 ['Make', 'Origin', 'Type', 'Weight', 'Wheelbase', 'Horsepower', 'Lengt
h'],
 ['Make', 'Origin', 'Type', 'Weight', 'Wheelbase', 'Length', 'Horsepowe
r'],
```

In [430]:
```
dfff = pandas.DataFrame(perm_index)
dfff
```

Out[430]:

	0	1	2	3	4	5	6
0	Make	Origin	Type	Horsepower	Length	Weight	Wheelbase
1	Make	Origin	Type	Horsepower	Length	Wheelbase	Weight
2	Make	Origin	Type	Horsepower	Weight	Length	Wheelbase
3	Make	Origin	Type	Horsepower	Weight	Wheelbase	Length
4	Make	Origin	Type	Horsepower	Wheelbase	Length	Weight
...	...	...	...	...	...	...	...
5035	Wheelbase	Weight	Length	Horsepower	Make	Type	Origin
5036	Wheelbase	Weight	Length	Horsepower	Origin	Make	Type
5037	Wheelbase	Weight	Length	Horsepower	Origin	Type	Make
5038	Wheelbase	Weight	Length	Horsepower	Type	Make	Origin
5039	Wheelbase	Weight	Length	Horsepower	Type	Origin	Make

5040 rows × 7 columns

In [431]:
```
def SWEEPOperator (pDim, inputM, origDiag, sweepCol = None, tol = 1e-7):
    ''' Implement the SWEEP operator
    Parameter
    ---------
    pDim: dimension of matrix inputM, integer greater than one
    inputM: a square and symmetric matrix, numpy array
    origDiag: the original diagonal elements before any SWEEPing
    sweepCol: a list of columns numbers to SWEEP
    tol: singularity tolerance, positive real
    Return
    ------
    A: negative of a generalized inverse of input matrix
    aliasParam: a list of aliased rows/columns in input matrix
    nonAliasParam: a list of non-aliased rows/columns in input matrix
    '''
    if (sweepCol is None):
        sweepCol = range(pDim)
    aliasParam = []
    nonAliasParam = []
    A = numpy.copy(inputM)
    ANext = numpy.zeros((pDim,pDim))
    for k in sweepCol:
        Akk = A[k,k]
        pivot = tol * abs(origDiag[k])
        if (not numpy.isinf(Akk) and abs(Akk) >= pivot and pivot > 0.0):
            nonAliasParam.append(k)
            ANext = A - numpy.outer(A[:, k], A[k, :]) / Akk
            ANext[:, k] = A[:, k] / abs(Akk)
            ANext[k, :] = ANext[:, k]
            ANext[k, k] = -1.0 / Akk
        else:
            aliasParam.append(k)
            ANext[:,k] = numpy.zeros(pDim)
            ANext[k, :] = numpy.zeros(pDim)
        A = ANext
    return (A, aliasParam, nonAliasParam)
def LinearRegression (X, y):
    ''' Train a linear regression model
    Parameter
    ---------
    X: A Pandas DataFrame, rows are observations, columns are regressors
    y: A Pandas Series, rows are observations of the response variable
    Return
    ------
    A list of the following entities:
    1. b: an array of regression coefficient
    2. residual_SS: residual sum of squares
    3. XtX_Ginv: a generalized inverse of the XtX matrix
    4. aliasParam: a list of aliased rows/columns in input matrix
    5. nonAliasParam: a list of non-aliased rows/columns in input matrix
    '''
    # X: A Pandas DataFrame, rows are observations, columns are regressors
    # y: A Pandas Series, rows are observations of the response variable
    Z = X.join(y)
    n_sample = Z.shape[0]
    n_param = Z.shape[1] - 1
```

```
    ZtZ = Z.transpose().dot(Z)
    diag_ZtZ = numpy.diagonal(ZtZ)
    eps_double = numpy.finfo(numpy.float64).eps
    tol = numpy.sqrt(eps_double)
    ZtZ_transf, aliasParam, nonAliasParam = SWEEPOperator ((n_param+1), ZtZ,
diag_ZtZ, sweepCol = range(n_param), tol = tol)
    b = ZtZ_transf[0:n_param, n_param]
    b[aliasParam] = 0.0
    XtX_Ginv = - ZtZ_transf[0:n_param, 0:n_param]
    XtX_Ginv[:, aliasParam] = 0.0
    XtX_Ginv[aliasParam, :] = 0.0
    residual_SS = ZtZ_transf[n_param, n_param]
    return ([b, residual_SS, XtX_Ginv, aliasParam, nonAliasParam])
```

In [432]:

```
for i in range(len(dfff.index)):
    #set the varr as the list of each row, and the first 7 elements of each r
    varr = dfff.loc[i,:].to_list()
    for j in range(7):
        #select the related features
        list_para = varr[0:j+1]
        X12 = df[list_para]
        X12.insert(0, 'Intercept', 1.0)
        y = data['MPG_City']
        result_list12 = LinearRegression(X12, y)
        residual_SS12 = result_list12[1]
        #calculate the r-squared
        r_sq12 = 1 - (residual_SS12/tss(y))
        #name the predictor columns
        pred_name = "Predictor" + str(j)
        #if the column exists, add the value to the corresponding cell
        if pred_name in dfff.columns:
        #change the designated cell
            dfff.at[i, pred_name] = r_sq12
        #otherwise, add the new column and name it.
        else:
            dfff[pred_name] = r_sq12
```

In [433]:
```
dfff
```

Out[433]:

	0	1	2	3	4	5	6	Predictor0	Pre
0	Make	Origin	Type	Horsepower	Length	Weight	Wheelbase	0.026752	0.(
1	Make	Origin	Type	Horsepower	Length	Wheelbase	Weight	0.026752	0.(
2	Make	Origin	Type	Horsepower	Weight	Length	Wheelbase	0.026752	0.(
3	Make	Origin	Type	Horsepower	Weight	Wheelbase	Length	0.026752	0.(
4	Make	Origin	Type	Horsepower	Wheelbase	Length	Weight	0.026752	0.(
...	...	...	...	...	...	...	...	...	
5035	Wheelbase	Weight	Length	Horsepower	Make	Type	Origin	0.257337	0.!
5036	Wheelbase	Weight	Length	Horsepower	Origin	Make	Type	0.257337	0.!
5037	Wheelbase	Weight	Length	Horsepower	Origin	Type	Make	0.257337	0.!
5038	Wheelbase	Weight	Length	Horsepower	Type	Make	Origin	0.257337	0.!
5039	Wheelbase	Weight	Length	Horsepower	Type	Origin	Make	0.257337	0.!

5040 rows × 14 columns

In [434]:
```
dfff1 = dfff.iloc[:,0:7]
dfff2 = dfff.iloc[:,7:14]
```

In [435]:
```
dfff3 = pandas.DataFrame()
```

In [436]:
```
dfff3['Predictor0'] = dfff2['Predictor0']
dfff3['Predictor1'] = dfff2['Predictor1'] - dfff2['Predictor0']
dfff3['Predictor2'] = dfff2['Predictor2'] - dfff2['Predictor1']
dfff3['Predictor3'] = dfff2['Predictor3'] - dfff2['Predictor2']
dfff3['Predictor4'] = dfff2['Predictor4'] - dfff2['Predictor3']
dfff3['Predictor5'] = dfff2['Predictor5'] - dfff2['Predictor4']
dfff3['Predictor6'] = dfff2['Predictor6'] - dfff2['Predictor5']
dfff3
```

Out[436]:

	Predictor0	Predictor1	Predictor2	Predictor3	Predictor4	Predictor5	Predictor6
0	0.026752	0.036962	0.003338	0.414056	0.052861	0.099522	0.007615
1	0.026752	0.036962	0.003338	0.414056	0.052861	0.004982	0.102155
2	0.026752	0.036962	0.003338	0.414056	0.152259	0.000124	0.007615
3	0.026752	0.036962	0.003338	0.414056	0.152259	0.004175	0.003564
4	0.026752	0.036962	0.003338	0.414056	0.054693	0.003151	0.102155
...	...	...	...	...	...	...	...
5035	0.257337	0.294201	0.007312	0.065337	0.001991	0.012718	0.002213
5036	0.257337	0.294201	0.007312	0.065337	0.004146	0.000546	0.012229
5037	0.257337	0.294201	0.007312	0.065337	0.004146	0.011793	0.000981
5038	0.257337	0.294201	0.007312	0.065337	0.012066	0.002642	0.002213
5039	0.257337	0.294201	0.007312	0.065337	0.012066	0.003874	0.000981

5040 rows × 7 columns

In [437]:
```
col12 = ['Make', 'Origin', 'Type', 'Horsepower', 'Length', 'Weight', 'Wheelba
dict_pred = {'Make': 0, 'Origin': 0, 'Type': 0, 'Horsepower': 0, 'Length': 0,
dict_pred1 = dict_pred
```

In [438]:
```
for i in range(len(dfff1.index)):
    variables = dfff1.loc[i,:].to_list()
    pred_diff = dfff3.loc[i,:].to_list()
    for j in range(7):
        dict_pred[variables[j]] += pred_diff[j]
dict_pred
```

Out[438]:
```
{'Make': 31.84065274050079,
 'Origin': 73.40952453693774,
 'Type': 45.5482272707267,
 'Horsepower': 1050.9494893478136,
 'Length': 341.0860556706681,
 'Weight': 1323.6873757129083,
 'Wheelbase': 364.65838245209807}
```

Calculate the Shap Value for each feature

In [439]:
```
dic_keys = dict_pred.keys()
sum_rsq = 0
for i in dic_keys:
    #Calculate the average sum
    dict_pred[i] = dict_pred[i] / 5040
    sum_rsq += dict_pred[i]

dict_pred
```

Out[439]:
```
{'Make': 0.006317589829464443,
 'Origin': 0.014565381852567012,
 'Type': 0.009037346680699741,
 'Horsepower': 0.20852172407694713,
 'Length': 0.06767580469656113,
 'Weight': 0.2626363840700215,
 'Wheelbase': 0.07235285366113058}
```

In [440]:
```
sum_rsq
```

Out[440]:
```
0.6411070848673915
```

In [441]:
```
df_shapval = pandas.DataFrame.from_dict(dict_pred, orient='index', columns=['
df_shapval['Percent Shape'] = df_shapval['Shap Value'] / sum_rsq
df_shapval
```

Out[441]:

	Shap Value	Percent Shape
Make	0.006318	0.009854
Origin	0.014565	0.022719
Type	0.009037	0.014096
Horsepower	0.208522	0.325253
Length	0.067676	0.105561
Weight	0.262636	0.409661
Wheelbase	0.072353	0.112856