

EDA 3 AAA Project Martin George mgeorgevienna@gmail.com

```
In [1]: %matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
df = pd.read_csv('member_sample.csv', index_col = 0)
```

Application of linear regressoin model on AAA data

```
In [2]: df.head()
df.info()
df.columns

<class 'pandas.core.frame.DataFrame'>
Int64Index: 21344 entries, 0 to 99998
Columns: 112 entries, Individual Key to Was Towed To AAR Referral
dtypes: float64(35), object(77)
memory usage: 18.4+ MB
```

```
Out[2]: Index(['Individual Key', 'Household Key', 'Member Flag', 'City',
              'State - Grouped', 'ZIP5', 'ZIP9', 'FSV CMSI Flag',
              'FSV Credit Card Flag', 'FSV Deposit Program Flag',
              ...,
              'SC Vehicle Manufacturer Name', 'SC Vehicle Model Name',
              'SVC Facility Name', 'SVC Facility Type', 'Total Cost',
              'Tow Destination Latitude', 'Tow Destination Longitude',
              'Tow Destination Name', 'Was Duplicated', 'Was Towed To AAR Referral'],
              dtype='object', length=112)
```

In [70]: df.head()

Out[70]:

	Individual Key	Household Key	Member Flag	City	State - Grouped	ZIP5	ZIP9	FSV CMSI Flag	FSV Credit Card Flag	FSV Deposit Program Flag	...	SC Vehicle Manufacturer Name	SC Vehicle Model Name	SVC Facility Name	F
0	10000003.0	10462590.0	Y	NEW HAVEN	CT	6511.0	65111349.0	N	N	N	...	NaN	NaN	NaN	
1	52211550.0	4500791.0	Y	WEST WARWICK	RI	2893.0	28933850.0	N	Y	N	...	TOYOTA	CAMRY	ASTRO WRECKER SERVICE	indep
2	52211550.0	4500791.0	Y	WEST WARWICK	RI	2893.0	28933850.0	N	Y	N	...	TOYOTA	CAMRY	Astro Wrecker Service	indep
3	52211550.0	4500791.0	Y	WEST WARWICK	RI	2893.0	28933850.0	N	Y	N	...	TOYOTA	CAMRY	ASTRO WRECKER SERVICE	indep
4	52211550.0	4500791.0	Y	WEST WARWICK	RI	2893.0	28933850.0	N	Y	N	...	TOYOTA	CAMRY	ASTRO WRECKER SERVICE	indep

5 rows × 112 columns

Testing the function of get_dummies of panda

```
In [3]: pd.get_dummies(df['SC Vehicle Manufacturer Name'])
```

Out[3]:

	ACURA	ALFA ROMEO	AMERICAN AUSTIN	APRILIA	AUDI	AUSTIN HEALEY	Audi	BICYCLE	BMW	BUICK	...	TOYOTA	TRIUMPH	Toyota	UNK	VOLKSWAGE
0	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	0 ...	1	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	0 ...	1	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	0 ...	1	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	0 ...	1	0	0	0	
5	0	0	0	0	0	0	0	0	0	0	0 ...	1	0	0	0	
6	0	0	0	0	0	0	0	0	0	0	0 ...	1	0	0	0	
7	0	0	0	0	0	0	0	0	0	0	0 ...	1	0	0	0	
8	0	0	0	0	0	0	0	0	0	0	0 ...	1	0	0	0	
9	0	0	0	0	0	0	0	0	0	0	0 ...	1	0	0	0	
10	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
11	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
12	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
13	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
14	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
15	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
16	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
17	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
18	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
19	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
20	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
21	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
22	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
23	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
24	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
25	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
26	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
27	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
28	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	

	ACURA	ALFA ROMEO	AMERICAN AUSTIN	APRILIA	AUDI	AUSTIN HEALEY	Audi	BICYCLE	BMW	BUICK	...	TOYOTA	TRIUMPH	Toyota	UNK	VOLKSWAGE
29	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
...	
99968	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
99969	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
99970	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
99971	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
99972	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
99973	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
99974	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
99975	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
99976	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
99977	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
99979	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
99980	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
99981	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
99982	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
99983	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
99984	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
99985	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
99986	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
99987	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
99988	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
99989	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
99990	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
99991	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
99992	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
99993	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
99994	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	
99995	0	0	0	0	0	0	0	0	0	0	0 ...	0	0	0	0	

	ACURA	ALFA ROMEO	AMERICAN AUSTIN	APRILIA	AUDI	AUSTIN HEALEY	Audi	BICYCLE	BMW	BUICK	...	TOYOTA	TRIUMPH	Toyota	UNK	VOLKSWAGE
99996	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
99997	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	
99998	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	

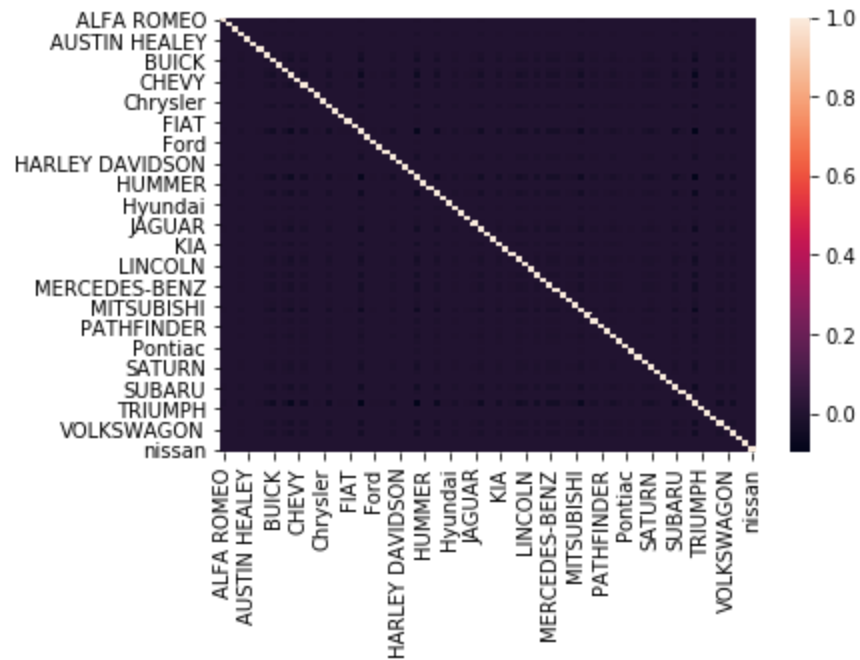
21344 rows × 86 columns



```
In [77]: get_dummied = pd.get_dummies(df['SC Vehicle Manufacturer Name'],drop_first = True)
```

```
In [78]: import seaborn as sns
sns.heatmap(get_dummied.corr())
```

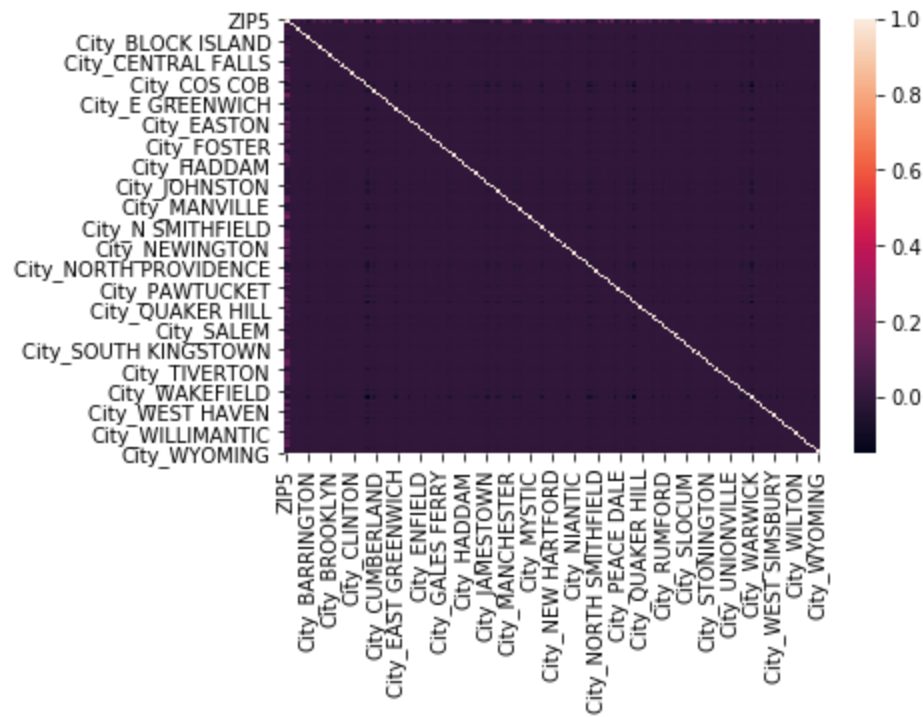
```
Out[78]: <matplotlib.axes._subplots.AxesSubplot at 0x1b45bfe8d68>
```



```
In [5]: dummied = pd.get_dummies(df[['Member Flag', 'City', 'ZIP5']])
```

```
In [6]: import seaborn as sns
sns.heatmap(dummied.corr())
```

```
Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x1b459a559e8>
```



Heatmap shows less correlation between features

```
In [75]: df[['ERS Member Cost Year 1', 'ERS Member Cost Year 2', 'Total Cost']]
```


Out[75]:

	ERS Member Cost Year 1	ERS Member Cost Year 2	Total Cost
0	NaN	NaN	NaN
1	0.00	0.0	32.50
2	0.00	0.0	30.00
3	0.00	0.0	32.50
4	0.00	0.0	30.00
5	0.00	0.0	53.00
6	0.00	0.0	30.00
7	0.00	0.0	32.00
8	0.00	0.0	32.00
9	0.00	0.0	32.50
10	NaN	NaN	NaN
11	58.85	117.7	58.85
12	58.85	117.7	53.00
13	58.85	117.7	53.00
14	NaN	NaN	NaN
15	NaN	NaN	NaN
16	0.00	0.0	53.00
17	0.00	0.0	53.00
18	0.00	0.0	53.00
19	0.00	0.0	29.00
20	0.00	0.0	28.00
21	0.00	0.0	53.00
22	0.00	0.0	53.00
23	0.00	0.0	53.00
24	0.00	0.0	53.00
25	0.00	0.0	53.00
26	58.85	0.0	58.85
27	0.00	0.0	53.00
28	0.00	0.0	53.00

	ERS Member Cost Year 1	ERS Member Cost Year 2	Total Cost
29	0.00	0.0	53.00
...
99968	NaN	NaN	NaN
99969	164.70	117.7	47.00
99970	164.70	117.7	58.85
99971	164.70	117.7	58.85
99972	164.70	117.7	53.00
99973	164.70	117.7	53.00
99974	164.70	117.7	58.85
99975	NaN	NaN	NaN
99976	0.00	0.0	NaN
99977	0.00	34.5	34.00
99979	NaN	NaN	NaN
99980	NaN	NaN	NaN
99981	NaN	NaN	NaN
99982	NaN	NaN	NaN
99983	0.00	0.0	29.00
99984	0.00	0.0	44.00
99985	0.00	0.0	29.00
99986	0.00	0.0	47.00
99987	0.00	0.0	44.00
99988	0.00	0.0	82.00
99989	0.00	0.0	29.43
99990	NaN	NaN	NaN
99991	NaN	NaN	NaN
99992	0.00	0.0	NaN
99993	0.00	0.0	53.00
99994	0.00	0.0	36.00
99995	0.00	0.0	53.00
99996	117.70	0.0	58.85

	ERS Member Cost Year 1	ERS Member Cost Year 2	Total Cost
99997	117.70	0.0	58.85
99998	117.70	0.0	NaN

21344 rows × 3 columns

Instead of get_dummies function, also we can use OneHotEncoder of sklearn

```
In [7]: from sklearn.preprocessing import OneHotEncoder
```

```
In [8]: ohe = OneHotEncoder(handle_unknown='ignore')
```

```
In [9]: ohe.fit(df[['Member Flag', 'City', 'ZIP5']])
```

```
Out[9]: OneHotEncoder(categorical_features=None, categories=None,
dtype=<class 'numpy.float64'>, handle_unknown='ignore',
n_values=None, sparse=True)
```

```
In [10]: ohe.transform(df[['Member Flag', 'City', 'ZIP5']]).toarray()
```

```
Out[10]: array([[1., 0., 0., ..., 0., 0., 0.],
[1., 0., 0., ..., 0., 0., 0.],
[1., 0., 0., ..., 0., 0., 0.],
...,
[1., 0., 0., ..., 0., 0., 0.],
[1., 0., 0., ..., 0., 0., 0.],
[1., 0., 0., ..., 0., 0., 0.]])
```

```
In [11]: ohe.transform(df[['Member Flag', 'City', 'ZIP5']])
```

```
Out[11]: <21344x350 sparse matrix of type '<class 'numpy.float64'>'
with 64032 stored elements in Compressed Sparse Row format>
```

```
In [12]: df[['Member Flag', 'City', 'ZIP5']]
```

Out[12]:

	Member Flag	City	ZIP5
0	Y	NEW HAVEN	6511.0
1	Y	WEST WARWICK	2893.0
2	Y	WEST WARWICK	2893.0
3	Y	WEST WARWICK	2893.0
4	Y	WEST WARWICK	2893.0
5	Y	WEST WARWICK	2893.0
6	Y	WEST WARWICK	2893.0
7	Y	WEST WARWICK	2893.0
8	Y	WEST WARWICK	2893.0
9	Y	WEST WARWICK	2893.0
10	Y	TIVERTON	2878.0
11	Y	WARWICK	2889.0
12	Y	WARWICK	2889.0
13	Y	WARWICK	2889.0
14	Y	WARWICK	2889.0
15	Y	WARWICK	2889.0
16	Y	WARWICK	2889.0
17	Y	WARWICK	2889.0
18	Y	WARWICK	2889.0
19	Y	WARWICK	2889.0
20	Y	CENTRAL FALLS	2863.0
21	Y	WARWICK	2888.0
22	Y	WARWICK	2888.0
23	Y	WARWICK	2888.0
24	Y	WARWICK	2888.0
25	Y	WARWICK	2888.0
26	Y	WARWICK	2888.0
27	Y	BARRINGTON	2806.0
28	Y	BARRINGTON	2806.0

	Member Flag	City	ZIP5
29	Y	BARRINGTON	2806.0
...
99968	Y	WARWICK	2886.0
99969	Y	PROVIDENCE	2906.0
99970	Y	PROVIDENCE	2906.0
99971	Y	PROVIDENCE	2906.0
99972	Y	PROVIDENCE	2906.0
99973	Y	PROVIDENCE	2906.0
99974	Y	PROVIDENCE	2906.0
99975	Y	PROVIDENCE	2906.0
99976	Y	BRISTOL	2809.0
99977	Y	BRISTOL	2809.0
99979	Y	PORTSMOUTH	2871.0
99980	Y	PORTSMOUTH	2871.0
99981	Y	PORTSMOUTH	2871.0
99982	Y	PORTSMOUTH	2871.0
99983	Y	BRISTOL	2809.0
99984	Y	BRISTOL	2809.0
99985	Y	BRISTOL	2809.0
99986	Y	BRISTOL	2809.0
99987	Y	BRISTOL	2809.0
99988	Y	BRISTOL	2809.0
99989	Y	WAKEFIELD	2879.0
99990	Y	WAKEFIELD	2879.0
99991	Y	WAKEFIELD	2879.0
99992	Y	COVENTRY	2816.0
99993	Y	WARWICK	2886.0
99994	Y	WARWICK	2886.0
99995	Y	WARWICK	2886.0
99996	Y	WARWICK	2886.0

	Member Flag	City	ZIP5
99997	Y	WARWICK	2886.0
99998	Y	WARWICK	2886.0

21344 rows × 3 columns

Using Linear Regression to test the correlation between features 'City','Motorcycle Indicator','FSV Credit Card Flag' to 'Total Cost'

```
In [13]: X = df[['City', 'Motorcycle Indicator', 'FSV Credit Card Flag', 'Total Cost']].dropna()
```

```
In [14]: y= X['Total Cost']
```

```
In [ ]:
```

```
In [33]: #X.drop('Total Cost', axis =1)
X = X.drop('Total Cost', axis =1)
```

```
In [34]: X.head()
```

Out[34]:

	City	Motorcycle Indicator	FSV Credit Card Flag
1	WEST WARWICK	N	Y
2	WEST WARWICK	N	Y
3	WEST WARWICK	N	Y
4	WEST WARWICK	N	Y
5	WEST WARWICK	N	Y

Different transforms can be used together using Pipeline of sklearn.

```
In [35]: from sklearn.linear_model import LinearRegression
```

```
In [36]: from sklearn.preprocessing import OneHotEncoder
```

```
In [37]: from sklearn.pipeline import Pipeline
```

```
In [38]: ohe= OneHotEncoder()
```

```
In [39]: X_numbers = ohe.fit_transform(X)
```

```
In [40]: lr = LinearRegression()
```

```
In [41]: lr.fit(X_numbers,y)
```

```
Out[41]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,  
        normalize=False)
```

```
In [42]: pipe = Pipeline([('ohe', OneHotEncoder(handle_unknown='ignore')), ('lr', LinearRegression())])
```

```
In [43]: pipe.fit(X,y)
```

```
Out[43]: Pipeline(memory=None,  
        steps=[('ohe', OneHotEncoder(categorical_features=None, categories=None,  
        dtype=<class 'numpy.float64'>, handle_unknown='ignore',  
        n_values=None, sparse=True)), ('lr', LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,  
        normalize=False))])
```

Check the score of the Pipe which involves two transformers, ohe and lr

```
In [44]: pipe.score(X,y)
```

```
Out[44]: 0.0055248751735678114
```

```
In [45]: pipe.named_steps
```

```
Out[45]: {'ohe': OneHotEncoder(categorical_features=None, categories=None,  
        dtype=<class 'numpy.float64'>, handle_unknown='ignore',  
        n_values=None, sparse=True),  
        'lr': LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,  
        normalize=False)}
```



```
In [46]: ohe.get_feature_names()
```

```
Out[46]: array(['x0_ADAMSVILLE', 'x0_ALBION', 'x0_ASHAWAY', 'x0_BARRINGTON',  
               'x0_BLOCK ISLAND', 'x0_BRADFORD', 'x0_BRIDGEPORT', 'x0_BRISTOL',  
               'x0_BROOKLYN', 'x0_CAROLINA', 'x0_CENTRAL FALLS', 'x0_CHARLESTOWN',  
               'x0_CHEPACHET', 'x0_CHESHIRE', 'x0_COLCHESTER', 'x0_COS COB',  
               'x0_COVENTRY', 'x0_CRANSTON', 'x0_CUMBERLAND', 'x0_DANIELSON',  
               'x0_DAYVILLE', 'x0_E GREENWICH', 'x0_E PROVIDENCE',  
               'x0_EAST GREENWICH', 'x0_EAST PROVIDENCE', 'x0_EASTON',  
               'x0_EXETER', 'x0_FISKEVILLE', 'x0_FOSTER', 'x0_GREENE',  
               'x0_GREENVILLE', 'x0_GROTON', 'x0_GUILFORD', 'x0_HARMONY',  
               'x0_HARRISVILLE', 'x0_HOPE', 'x0_HOPE VALLEY', 'x0_HOPKINTON',  
               'x0_JAMESTOWN', 'x0_JOHNSTON', 'x0_KINGSTON', 'x0_LINCOLN',  
               'x0_LITTLE COMPTON', 'x0_MANCHESTER', 'x0_MANSFIELD CENTER',  
               'x0_MANVILLE', 'x0_MAPLEVILLE', 'x0_MIDDLETOWN', 'x0_MILFORD',  
               'x0_MOOSUP', 'x0_MYSTIC', 'x0_N KINGSTOWN', 'x0_N PROVIDENCE',  
               'x0_N SMITHFIELD', 'x0_NARRAGANSETT', 'x0_NEW CANAAN',  
               'x0_NEW HARTFORD', 'x0_NEW LONDON', 'x0_NEWINGTON', 'x0_NEWPORT',  
               'x0_NEWTOWN', 'x0_NORFOLK', 'x0_NORTH KINGSTOWN',  
               'x0_NORTH PROVIDENCE', 'x0_NORTH SCITUATE', 'x0_NORTH SMITHFIELD',  
               'x0_NORTH STONINGTON', 'x0_PASCOAG', 'x0_PAWCATUCK',  
               'x0_PAWTUCKET', 'x0_PEACE DALE', 'x0_PORTSMOUTH', 'x0_PROSPECT',  
               'x0_PROVIDENCE', 'x0_PUTNAM', 'x0_RIVERSIDE', 'x0_RUMFORD',  
               'x0_SAUNDERSTOWN', 'x0_SCITUATE', 'x0_SLATERSVILLE',  
               'x0_SMITHFIELD', 'x0_SOUTH KINGSTOWN', 'x0_STRATFORD',  
               'x0_TIVERTON', 'x0_TOLLAND', 'x0_TRUMBULL', 'x0_UNIONVILLE',  
               'x0_VOLUNTOWN', 'x0_W GREENWICH', 'x0_WAKEFIELD', 'x0_WALLINGFORD',  
               'x0_WARREN', 'x0_WARWICK', 'x0_WATERBURY', 'x0_WATERFORD',  
               'x0_WEST GREENWICH', 'x0_WEST HARTFORD', 'x0_WEST KINGSTON',  
               'x0_WEST WARWICK', 'x0_WESTBROOK', 'x0_WESTERLY', 'x0_WESTPORT',  
               'x0_WOOD RIVER JUNCTION', 'x0_WOODBURY', 'x0_WOODSTOCK',  
               'x0_WOONSOCKET', 'x0_WYOMING', 'x1_N', 'x1_Y', 'x2_N', 'x2_Y'],  
              dtype=object)
```

The classes in the `sklearn.feature_selection` module can be used for feature selection/dimensionality reduction on sample sets, either to improve estimators' accuracy scores or to boost their performance on very high-dimensional datasets.

```
In [47]: from sklearn.feature_selection import SelectKBest
```

```
In [48]: pipe2 = Pipeline([('ohe', OneHotEncoder(handle_unknown='ignore')), ('kbest', SelectKBest()), ('lr', LinearRegression())])
```

```
In [49]: pipe2.fit(X,y)
```

```
Out[49]: Pipeline(memory=None,
      steps=[('ohe', OneHotEncoder(categorical_features=None, categories=None,
      dtype=<class 'numpy.float64'>, handle_unknown='ignore',
      n_values=None, sparse=True)), ('kbest', SelectKBest(k=10, score_func=<function f_classif at 0x000001B45BA91D08
      >)), ('lr', LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
      normalize=False))])
```

```
In [50]: pipe2.score(X,y)
```

```
Out[50]: 0.0009818896899872476
```

But there is no performance improvement as the selection of initial features were not appropriate

Clustering of AAA data

```
In [56]: X = df[['Mosaic Household', 'Total Cost']]
```

```
In [57]: y = X['Total Cost']
X = X.drop('Total Cost', axis = 1)
```

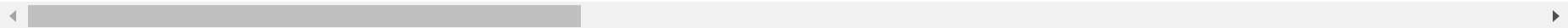
```
In [58]: X= pd.get_dummies(X)
```

```
In [59]: X.head()
```

```
Out[59]:
```

	Mosaic Household_Aging in Place	Mosaic Household_Aging of Aquarius	Mosaic Household_American Royalty	Mosaic Household_Babies and Bliss	Mosaic Household_Balance and Harmony	Mosaic Household_Birkenstocks and Beemers	Mosaic Household_Blue Collar Comfort	Ho
0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	

5 rows × 66 columns



```
In [60]: from sklearn.cluster import KMeans
```

Application of KMeans Clustering

```
In [61]: kmeans = KMeans(n_clusters = 5)
```

```
In [62]: kmeans.fit(X,y)
```

```
Out[62]: KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,  
n_clusters=5, n_init=10, n_jobs=None, precompute_distances='auto',  
random_state=None, tol=0.0001, verbose=0)
```

```
In [63]: X['cluster'] = kmeans.labels_
```

In [64]:

X

Out[64]:

	Mosaic Household_Aging in Place	Mosaic Household_Aging of Aquarius	Mosaic Household_American Royalty	Mosaic Household_Babies and Bliss	Mosaic Household_Balance and Harmony	Mosaic Household_Birkenstocks and Beemers	Mosaic Household_Blue Collar Comfort
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0
10	1	0	0	0	0	0	0
11	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0
21	1	0	0	0	0	0	0
22	1	0	0	0	0	0	0
23	1	0	0	0	0	0	0
24	1	0	0	0	0	0	0
25	1	0	0	0	0	0	0
26	1	0	0	0	0	0	0
27	0	0	1	0	0	0	0

	Mosaic Household_Aging in Place	Mosaic Household_Aging of Aquarius	Mosaic Household_American Royalty	Mosaic Household_Babies and Bliss	Mosaic Household_Balance and Harmony	Mosaic Household_Birkenstocks and Beemers	Mosaic Household_Blue Collar Comfort
28	0	0	1	0	0	0	0
29	0	0	1	0	0	0	0
...
99968	0	0	0	0	0	0	0
99969	0	0	0	0	0	0	0
99970	0	0	0	0	0	0	0
99971	0	0	0	0	0	0	0
99972	0	0	0	0	0	0	0
99973	0	0	0	0	0	0	0
99974	0	0	0	0	0	0	0
99975	0	0	0	0	0	0	0
99976	0	0	0	0	0	0	0
99977	0	0	0	0	0	0	0
99979	0	0	0	0	0	0	0
99980	0	0	0	0	0	0	0
99981	0	0	0	0	0	0	0
99982	0	0	0	0	0	0	0
99983	0	0	0	0	0	0	0
99984	0	0	0	0	0	0	0
99985	0	0	0	0	0	0	0
99986	0	0	0	0	0	0	0
99987	0	0	0	0	0	0	0
99988	0	0	0	0	0	0	0
99989	0	0	0	0	0	0	0
99990	0	0	0	0	0	0	0
99991	0	0	0	0	0	0	0
99992	0	0	0	0	0	0	0
99993	1	0	0	0	0	0	0
99994	1	0	0	0	0	0	0

	Mosaic Household_Aging in Place	Mosaic Household_Aging of Aquarius	Mosaic Household_American Royalty	Mosaic Household_Babies and Bliss	Mosaic Household_Balance and Harmony	Mosaic Household_Birkenstocks and Beemers	Mosaic Household_Blue Collar Comfort
99995	1	0	0	0	0	0	0
99996	1	0	0	0	0	0	0
99997	1	0	0	0	0	0	0
99998	1	0	0	0	0	0	0

21344 rows × 67 columns

Grouping based on clusters

```
In [65]: X.groupby('cluster').mean()
```

Out[65]:

	Mosaic Household_Aging in Place	Mosaic Household_Aging of Aquarius	Mosaic Household_American Royalty	Mosaic Household_Babies and Bliss	Mosaic Household_Balance and Harmony	Mosaic Household_Birkenstocks and Beemers	Mosaic Household_Blue Collar Comfort
cluster							
0	1.0	0.000000	0.0000	0.000000	0.000000	0.000000	0.000000
1	0.0	0.046139	0.0177	0.002718	0.001724	0.013457	0.023268
2	0.0	0.000000	0.0000	0.000000	0.000000	0.000000	0.000000
3	0.0	0.000000	0.0000	0.000000	0.000000	0.000000	0.000000
4	0.0	0.000000	0.0000	0.000000	0.000000	0.000000	0.000000

5 rows × 66 columns

Clustering didn't work here for Mosaic Household and Total Cost

Try with SC Vehicle Model name to see a possible clustering

```
In [66]: X= df[['SC Vehicle Model Name', 'Total Cost']].dropna()
```

```
In [67]: y = X['Total Cost']
```

```
In [68]: X = pd.get_dummies( X.drop('Total Cost', axis = 1 ), drop_first = True)
```

```
In [69]: kmeans.fit(X,y)
```

```
Out[69]: KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,  
               n_clusters=5, n_init=10, n_jobs=None, precompute_distances='auto',  
               random_state=None, tol=0.0001, verbose=0)
```

```
In [73]: X['cluster'] = kmeans.labels_
```



```
In [74]: X.groupby('cluster').mean().T
```

Out[74]:

	cluster	0	1	2	3	4
SC Vehicle Model Name_135I	0.001075	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_140	0.000083	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_150	0.000083	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_1500	0.001819	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_155	0.000083	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_190D	0.000165	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_2	0.000413	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_200	0.001488	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_2000	0.000083	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_2002	0.000083	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_2019-09-03 00:00:00	0.002563	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_2019-09-05 00:00:00	0.003886	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_220I	0.000083	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_23	0.000083	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_230I	0.000083	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_230SLK	0.000083	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_240	0.001323	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_244	0.000083	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_245	0.000083	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_2500	0.000165	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_3	0.004630	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_300	0.003555	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_3000	0.000083	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_300C	0.000083	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_300CE	0.000083	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_300D	0.000083	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_300E	0.000083	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_300M	0.002067	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_300SDL	0.000083	0.0	0.0	0.0	0.0	0.0

	cluster	0	1	2	3	4
SC Vehicle Model Name_300ZX	0.000248	0.0	0.0	0.0	0.0	0.0
...
SC Vehicle Model Name_XB	0.000827	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_XC60	0.000579	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_XC70	0.002646	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_XC90	0.001075	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_XD	0.000248	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_XE	0.000083	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_XF	0.000248	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_XG300	0.000083	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_XG350	0.000248	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_XJ	0.000083	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_XJ12	0.000083	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_XJ8	0.000248	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_XJS	0.000331	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_XK	0.000248	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_XK8	0.000331	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_XL-7	0.000165	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_XTERRA	0.002067	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_XTS	0.000331	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_XV	0.000083	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_XV CROSSTREK	0.000331	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_YARIS	0.002067	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_YUKON	0.001654	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_YUKON XL	0.000083	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_YUKON XL 1500	0.000248	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_YUKON XL 2500	0.000083	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_Yukon XL 1500	0.000083	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_Z3	0.000661	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_Z4	0.000248	0.0	0.0	0.0	0.0	0.0

	cluster	0	1	2	3	4
SC Vehicle Model Name_ZEPHYR	0.000248	0.0	0.0	0.0	0.0	0.0
SC Vehicle Model Name_murano	0.000083	0.0	0.0	0.0	0.0	0.0

784 rows × 5 columns

```
In [76]: #kmeans.score(X,y)
```

```
In [102]: pd.get_dummies(df['SC Vehicle Manufacturer Name'],drop_first = True).shape
```

```
Out[102]: (21344, 85)
```

```
In [79]: X= df[['SC Vehicle Manufacturer Name', 'Total Cost']].dropna()
```

```
In [80]: y = X['Total Cost']
```

```
In [81]: X = pd.get_dummies( X.drop('Total Cost', axis = 1 ), drop_first = True)
```

```
In [82]: kmeans.fit(X,y)
```

```
Out[82]: KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
               n_clusters=5, n_init=10, n_jobs=None, precompute_distances='auto',
               random_state=None, tol=0.0001, verbose=0)
```

```
In [83]: X['cluster'] = kmeans.labels_
```

```
In [84]: X.groupby('cluster').mean().T
```

Out[84]:

	cluster	0	1	2	3	4
SC Vehicle Manufacturer Name_ALFA ROMEO	0.001726	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_AMERICAN AUSTIN	0.000123	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_APRILIA	0.000123	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_AUDI	0.008259	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_AUSTIN HEALEY	0.000123	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_Audi	0.000123	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_BICYCLE	0.000740	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_BMW	0.021080	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_BUICK	0.060528	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_Buick	0.000123	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_CADILLAC	0.023792	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_CHEVROLET	0.115015	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_CHEVY	0.000370	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_CHRYSLER	0.047707	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_CRYSLER	0.000247	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_Chevrolet	0.000493	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_Chrysler	0.000123	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_DODGE	0.047830	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_EAGLE	0.000123	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_FERRARI	0.000616	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_FIAT	0.000863	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_FLINT	0.000123	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_FORD	0.000000	0.0	1.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_FWD CORPORATION	0.000247	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_Ford	0.000986	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_GENESIS	0.000370	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_GEO	0.000863	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_GMC	0.021820	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_HARLEY DAVIDSON	0.000123	0.0	0.0	0.0	0.0	0.0

	cluster	0	1	2	3	4
SC Vehicle Manufacturer Name_HHKHKJHKJHKJHKJ	0.000123	0.0	0.0	0.0	0.0	0.0
...
SC Vehicle Manufacturer Name_MINI	0.004931	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_MITSUBISHI	0.006903	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_NISSAN	0.095661	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_Nissan	0.000370	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_OLDSMOBILE	0.010848	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_PATHFINDER	0.000123	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_PLYMOUTH	0.003205	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_PONTIAC	0.020340	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_PORSCHE	0.003328	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_Pontiac	0.000123	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_RAM	0.002589	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_ROLLS ROYCE	0.000123	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_SAAB	0.012574	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_SATURN	0.022436	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_SCION	0.003945	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_SEMI	0.000123	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_SMART	0.000123	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_SUBARU	0.000000	0.0	0.0	1.0	0.0	0.0
SC Vehicle Manufacturer Name_SUZUKI	0.002342	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_Subaru	0.000123	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_TOYOTA	0.000000	1.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_TRIUMPH	0.000123	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_Toyota	0.000123	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_UNK	0.000123	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_VOLKSWAGEN	0.043886	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_VOLKSWAGON	0.000247	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_VOLVO	0.041667	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_Volkswagen	0.000247	0.0	0.0	0.0	0.0	0.0

	cluster	0	1	2	3	4
SC Vehicle Manufacturer Name_Volvo	0.000123	0.0	0.0	0.0	0.0	0.0
SC Vehicle Manufacturer Name_nissan	0.000123	0.0	0.0	0.0	0.0	0.0

85 rows × 5 columns

We can see two clusters one with Vechicle Manufacturer name like TOYOTA, FORD and the other for remaining Vechicle names. Value count also justify this clustering.


```
In [103]: df['SC Vehicle Manufacturer Name'].value_counts().nlargest(50)
```

```
Out[103]:
```

TOYOTA	2289
FORD	1580
HONDA	1494
CHEVROLET	935
NISSAN	780
HYUNDAI	630
BUICK	491
SUBARU	480
JEEP	468
DODGE	391
CHRYSLER	387
VOLKSWAGEN	357
VOLVO	340
LEXUS	325
MERCURY	278
MERCEDES-BENZ	276
MAZDA	264
KIA	246
CADILLAC	195
SATURN	182
GMC	179
BMW	171
ACURA	165
PONTIAC	165
LINCOLN	160
SAAB	102
INFINITI	90
OLDSMOBILE	88
AUDI	68
MITSUBISHI	58
MINI	40
JAGUAR	37
SCION	33
PORSCHE	27
PLYMOUTH	26
LAND ROVER	26
RAM	21
SUZUKI	19
ALFA ROMEO	14
ISUZU	10
FIAT	8
Ford	8
GEO	7
BICYCLE	6
FERRARI	5
MG	5
Honda	4

```
Chevrolet      4
CHEVY          3
Hyundai        3
Name: SC Vehicle Manufacturer Name, dtype: int64
```

In []:

In []:

In []:

In []:

In []:

In []: