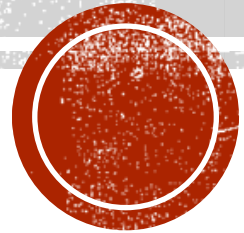
MARCHINE LEARNING – FLOW ANALYSIS

Analyse Boston Hubway trip



Xin ZHAO
03-22-2020

Unsupervised learning clustering

- **Clustering the flow by weekend and week days**

  Data mining to get OD flows in hourly basis.

2017-10 Hubway trips

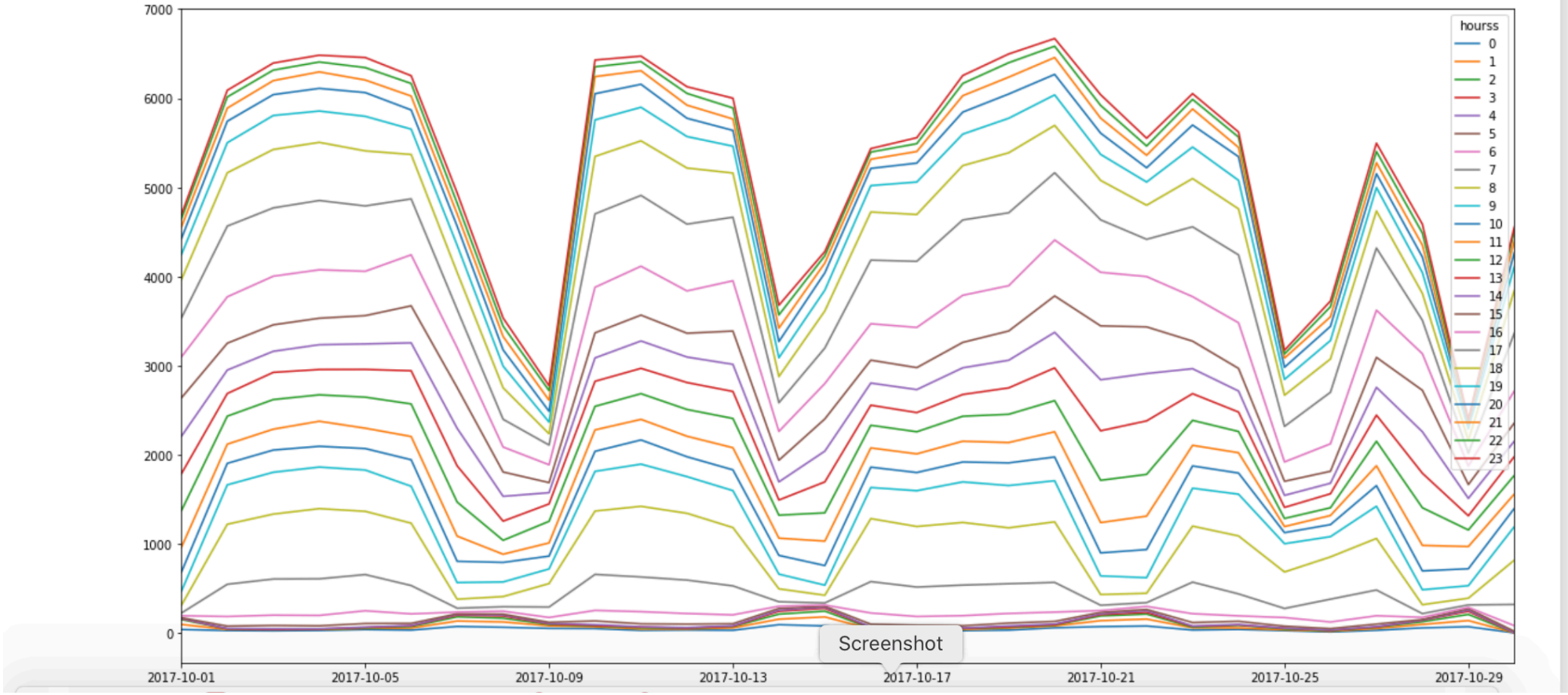
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 163662 entries, 0 to 163661
Data columns (total 15 columns):
tripduration               163662 non-null int64
starttime                  163662 non-null object
stoptime                   163662 non-null object
start station id           163662 non-null int64
start station name         163662 non-null object
start station latitude     163662 non-null float64
start station longitude    163662 non-null float64
end station id             163662 non-null int64
end station name           163662 non-null object
end station latitude       163662 non-null float64
end station longitude      163662 non-null float64
bikeid                     163662 non-null int64
usertype                   163662 non-null object
birth_year                 163662 non-null object
gender                     163662 non-null int64
dtypes: float64(4), int64(5), object(6)
memory usage: 20.0+ MB
```

hourss	0	1	2	3	4	5	6	7	8	9	...	14	15	16	17	18	19	20	21	22	23
Date																					
2017-10-01	43.0	57.0	55.0	10.0	3.0	7.0	22.0	30.0	86.0	157.0	...	420.0	431.0	459.0	436.0	430.0	280.0	179.0	130.0	93.0	47.0
2017-10-02	31.0	12.0	6.0	1.0	4.0	27.0	108.0	361.0	673.0	442.0	...	263.0	301.0	520.0	793.0	598.0	337.0	241.0	147.0	125.0	75.0
2017-10-03	28.0	13.0	4.0	0.0	6.0	37.0	116.0	405.0	728.0	469.0	...	236.0	297.0	545.0	766.0	655.0	381.0	234.0	157.0	118.0	79.0
2017-10-04	32.0	11.0	3.0	2.0	3.0	33.0	117.0	410.0	787.0	466.0	...	278.0	297.0	543.0	777.0	652.0	351.0	254.0	185.0	111.0	76.0
2017-10-05	42.0	13.0	5.0	6.0	4.0	40.0	143.0	406.0	709.0	463.0	...	286.0	317.0	497.0	733.0	617.0	388.0	266.0	141.0	139.0	113.0
2017-10-06	36.0	23.0	17.0	8.0	4.0	24.0	106.0	318.0	699.0	414.0	...	313.0	415.0	572.0	627.0	498.0	284.0	215.0	156.0	141.0	87.0
2017-10-07	77.0	61.0	47.0	19.0	8.0	7.0	19.0	44.0	101.0	186.0	...	421.0	457.0	447.0	428.0	416.0	305.0	213.0	138.0	121.0	115.0
2017-10-08	68.0	60.0	42.0	24.0	11.0	10.0	33.0	50.0	114.0	164.0	...	278.0	273.0	280.0	311.0	353.0	243.0	182.0	150.0	116.0	92.0
2017-10-09	55.0	29.0	15.0	7.0	5.0	13.0	54.0	117.0	263.0	164.0	...	126.0	113.0	200.0	221.0	126.0	131.0	121.0	124.0	108.0	55.0
2017-10-10	53.0	20.0	10.0	5.0	10.0	41.0	118.0	404.0	710.0	445.0	...	262.0	281.0	511.0	823.0	644.0	410.0	293.0	192.0	110.0	76.0
2017-10-11	33.0	16.0	14.0	8.0	2.0	34.0	136.0	389.0	792.0	473.0	...	307.0	290.0	548.0	794.0	612.0	376.0	258.0	152.0	102.0	62.0
2017-10-12	37.0	12.0	8.0	3.0	5.0	39.0	118.0	375.0	749.0	411.0	...	286.0	267.0	474.0	749.0	629.0	353.0	205.0	147.0	132.0	75.0
2017-10-13	34.0	27.0	15.0	3.0	6.0	23.0	99.0	325.0	654.0	414.0	...	304.0	373.0	564.0	713.0	495.0	301.0	177.0	128.0	124.0	109.0
2017-10-14	97.0	62.0	56.0	32.0	13.0	19.0	25.0	50.0	145.0	165.0	...	203.0	243.0	323.0	322.0	293.0	211.0	182.0	150.0	148.0	109.0
2017-10-15	83.0	101.0	65.0	31.0	13.0	11.0	13.0	24.0	86.0	113.0	...	345.0	361.0	397.0	398.0	416.0	233.0	194.0	110.0	82.0	47.0
2017-10-16	34.0	15.0	9.0	5.0	6.0	33.0	126.0	352.0	706.0	349.0	...	248.0	257.0	408.0	713.0	540.0	296.0	193.0	103.0	80.0	39.0

Visualization

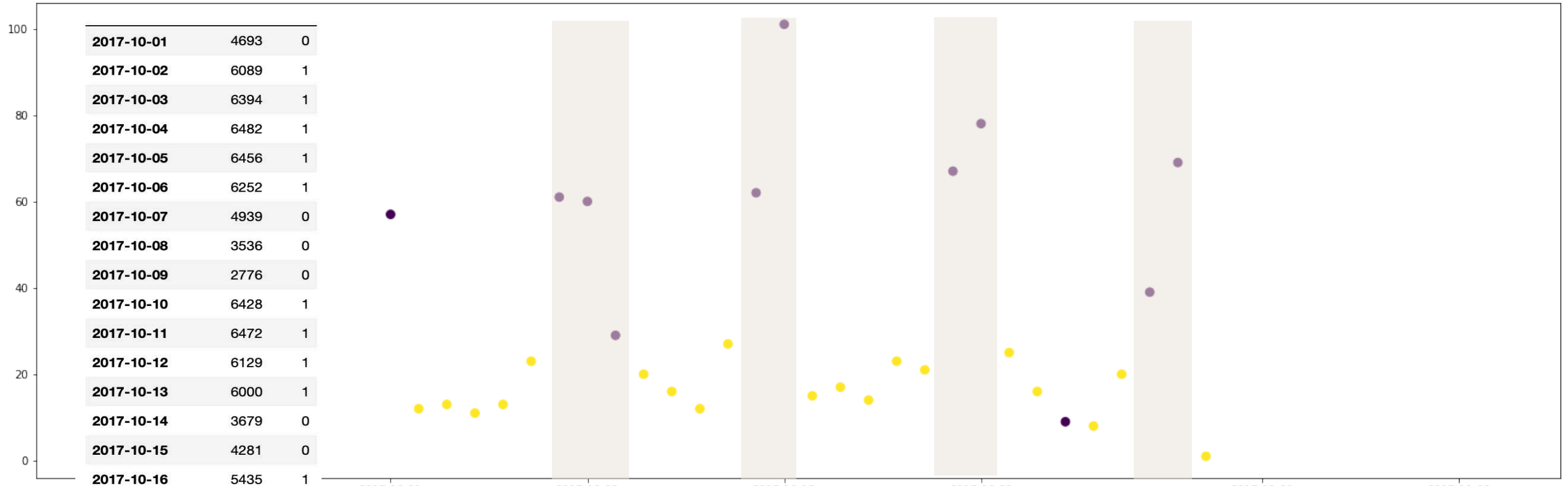
The evolution of OD flow in hourly basis of Oct-2017

Out[88]: <matplotlib.axes._subplots.AxesSubplot at 0x12ed83280>



A clustering without considering the days feature

2017-10-01	4693	0
2017-10-02	6089	1
2017-10-03	6394	1
2017-10-04	6482	1
2017-10-05	6456	1
2017-10-06	6252	1
2017-10-07	4939	0
2017-10-08	3536	0
2017-10-09	2776	0
2017-10-10	6428	1
2017-10-11	6472	1
2017-10-12	6129	1
2017-10-13	6000	1
2017-10-14	3679	0
2017-10-15	4281	0
2017-10-16	5435	1
2017-10-17	5557	1
2017-10-18	6254	1



Saturday Sunday

Saturday Sunday

Saturday Sunday

Saturday Sunday

By the Elbow method to find out the optimal number of clusters

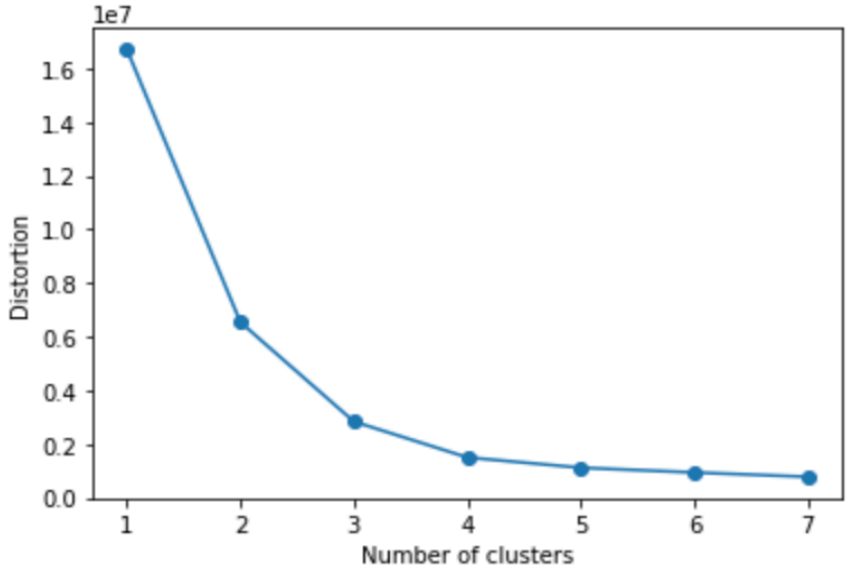
Algorithm theory :

$$D_k = \sum_{i=1}^{K} \sum dist(x, c_i)^2$$

The best clustering number is

k = 3 or 4

```
In [246]: #
distortions = []
for i in range(1, 8):
    km = KMeans(n_clusters=i,
                random_state=100)
    km.fit(df_new)
    distortions.append(km.inertia_)

plt.plot(range(1, 8), distortions, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Distortion')
plt.show()
```

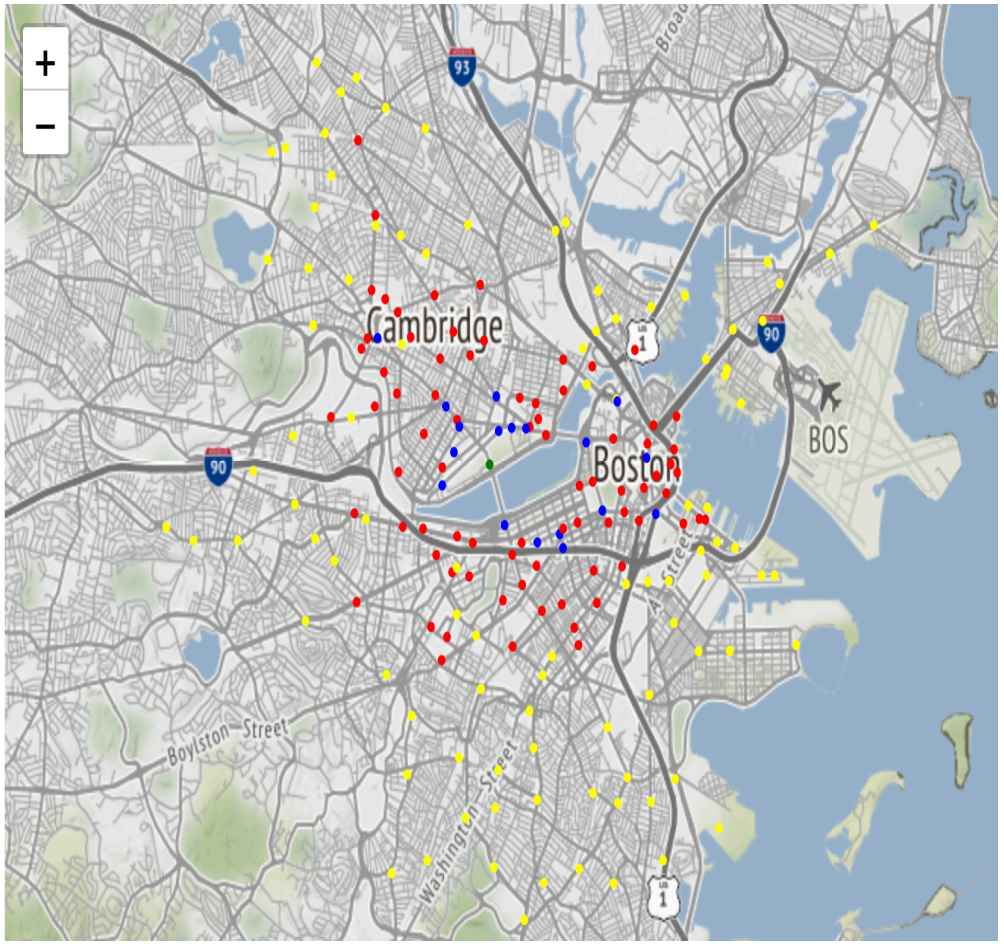


Clustering Station similar temporal patterns according hourly flow

0	141
1	1
2	6
4	4
8	8
9	54
10	20
11	11
12	24
13	13

1	1240
5	15
6	12
7	35
8	64
9	36
10	70
11	132
12	144
13	130
14	140
15	60
16	80
17	170
18	36
19	95
21	21

2	829
2	2
3	3
5	5
8	40
9	108
10	170
11	209
12	108
13	65
14	56
15	15
16	48
3	**3**
3	3
Grand Total	**2213**

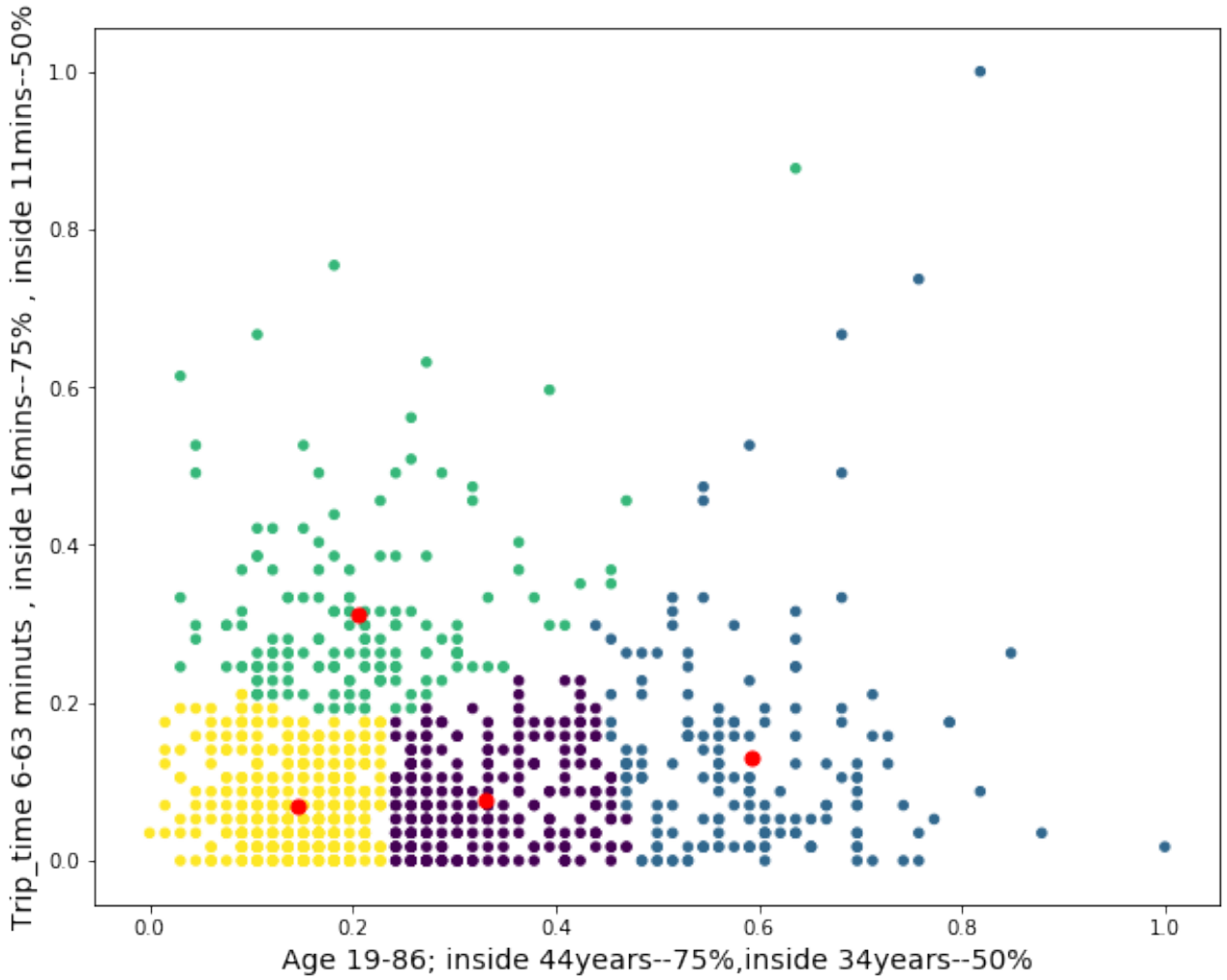


Clustering Station similar temporal patterns according hourly flow

	gender	birth_year	trip_time	age
0	0	1982	3	38
1	1	1997	6	23
2	1	1977	4	43
3	1	1991	18	29
4	1	1963	43	57
...	...	...	...	...
55067	1	1965	4	55
55068	1	1955	4	65
55069	1	1987	16	33
55070	1	1989	29	31
55071	2	1984	26	36

52952 rows × 4 columns

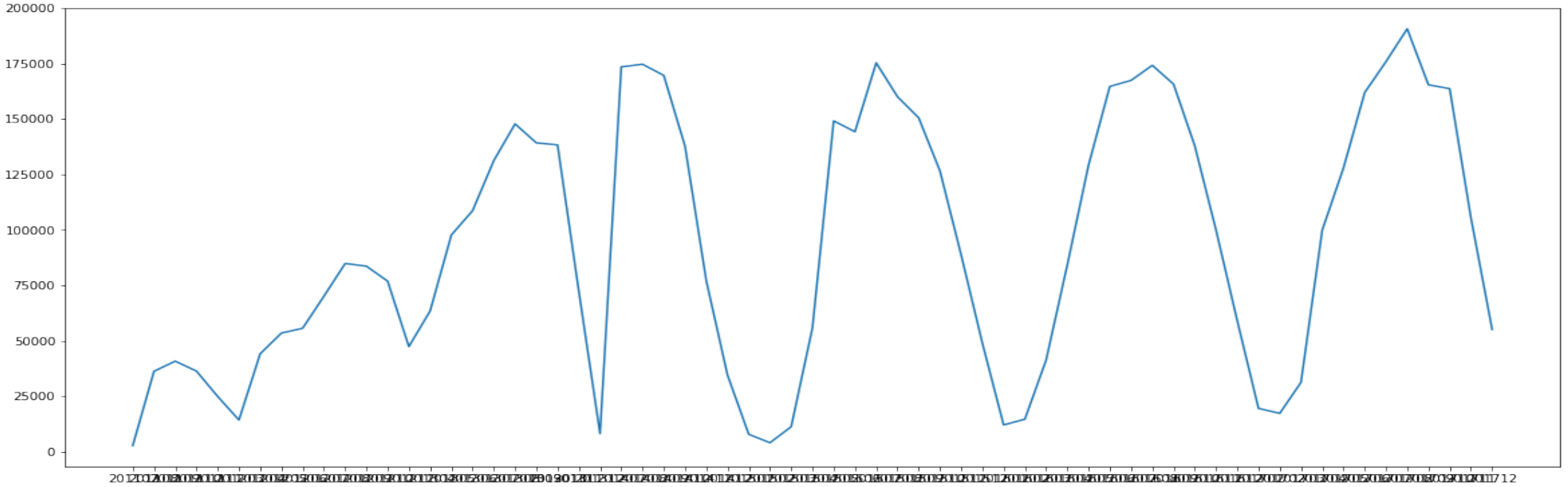
Try to cluster different profiles of users based on their information (age, flow locations, etc) and show how the clustering result present different types of profiles.



Supervised learning

- The idea is to predict bike flow based on history of rideships.
- Split your date into Train and Test set. You can learn on 1 year data and test on 1 month. (considering the yearly growth of users)
- Feel free to add features from other sources (eg. types of days, school holidays, weather information, sociodemographic data, etc)
- Try a supervised learning model (ANN, XGBoost, SVM, etc) on your choice.
- Predict the flow for your test data and compare your results with the real data.
- Calculate the errors and explain how you can improve your results.

PLOT MONTH- TRIP VARIABLE



Visualization the data to observer : not simple line regression, but have the regularity
Considering the serval features , we should make sure features are on a similar scale

LINEAR_MODEL.RIDGE REGRESSION

Line regression + Regularization

Line regression fitted function :

$$f(x) = w^\top x + b$$

Cost function :

$$J = \frac{1}{n}\sum_{i=1}^{n}(f(x_i) - y_i)^2$$

$$= \frac{1}{n}\sum_{i=1}^{n}(w^\top x_i + b - y_i)^2$$

line Regression

$$J = \frac{1}{n}\sum_{i=1}^{n}(f(x_i) - y_i)^2 + \lambda\|w\|_2^2$$

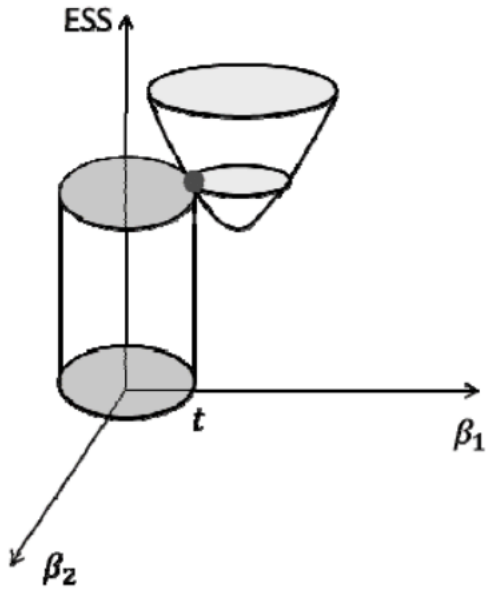
Ridge Regression

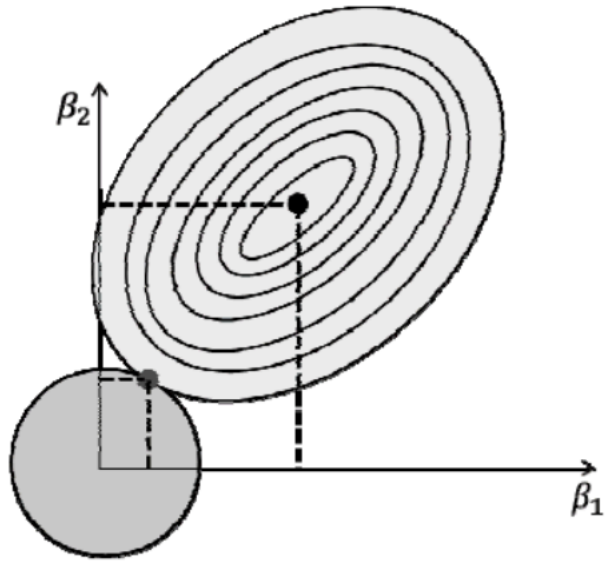
$$J(\beta) = \sum (y - X\beta)^2 + \lambda \|\beta\|_2^2 = \sum (y - X\beta)^2 + \sum \lambda \beta^2$$

$$\begin{aligned} J(\beta) &= (y - X\beta)'(y - X\beta) + \lambda\beta'\beta \\ &= (y' - \beta'X')(y - X\beta) + \lambda\beta'\beta \\ &= y'y - y'X\beta - \beta'X'y + \beta'X'X\beta + \lambda\beta'\beta \end{aligned}$$

$$\begin{aligned} \frac{\partial J(\beta)}{\partial \beta} &= 0 - X'y - X'y + 2X'X\beta + 2\lambda\beta \\ &= 2(X'X + \lambda I)\beta - 2X'y \end{aligned}$$

$$\begin{aligned} &2(X'X + \lambda I)\beta - 2X'y = 0 \\ &\therefore \beta = (X'X + \lambda I)^{-1}X'y \end{aligned}$$





$$\begin{cases} argmin\left\{\sum (y - x\beta)^2\right\} \\ \sum \beta^2 \le t \end{cases}$$

LINEAR_MODEL.RIDGE REGRESSION

Features

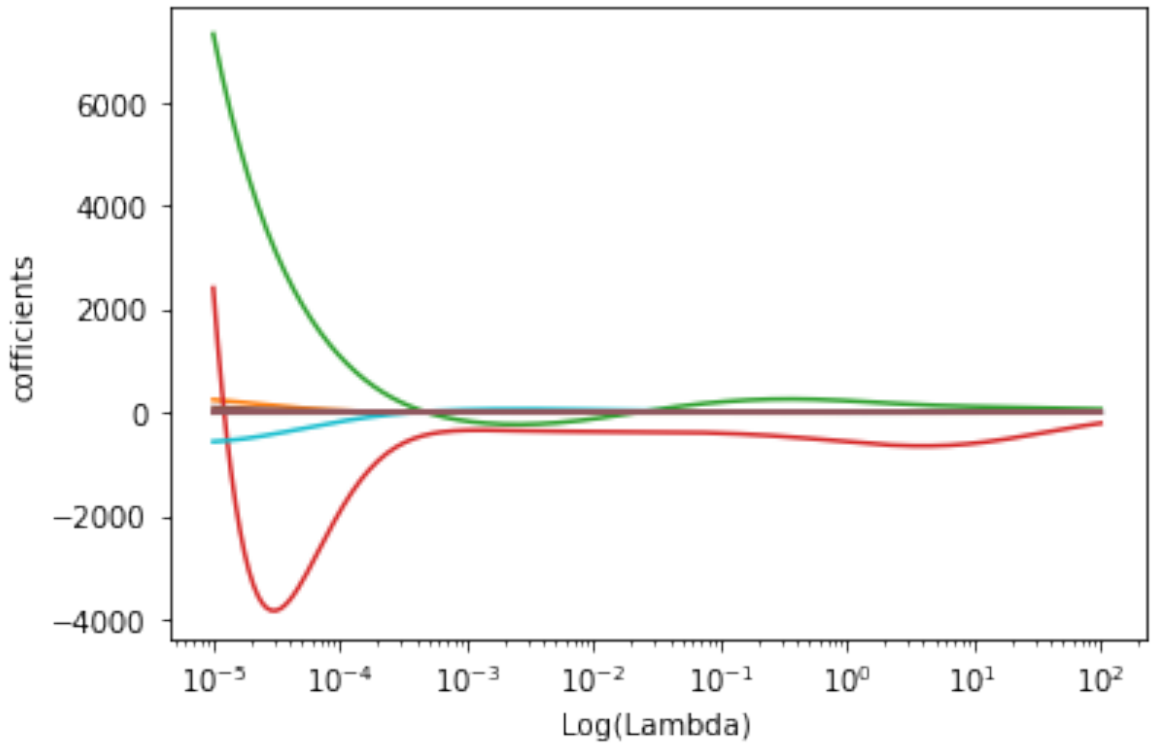
Traget

	year	month	yr	user_sub	od_flow
0	201501	1	2015	5484	7840
1	201502	2	2015	3071	4020
2	201503	3	2015	8228	11271
3	201504	4	2015	37298	55843
4	201505	5	2015	99413	149143
5	201506	6	2015	94866	144232
6	201507	7	2015	122283	175321
7	201508	8	2015	110407	160065

1-find out the optimal lambdas

ridge_best_lamb

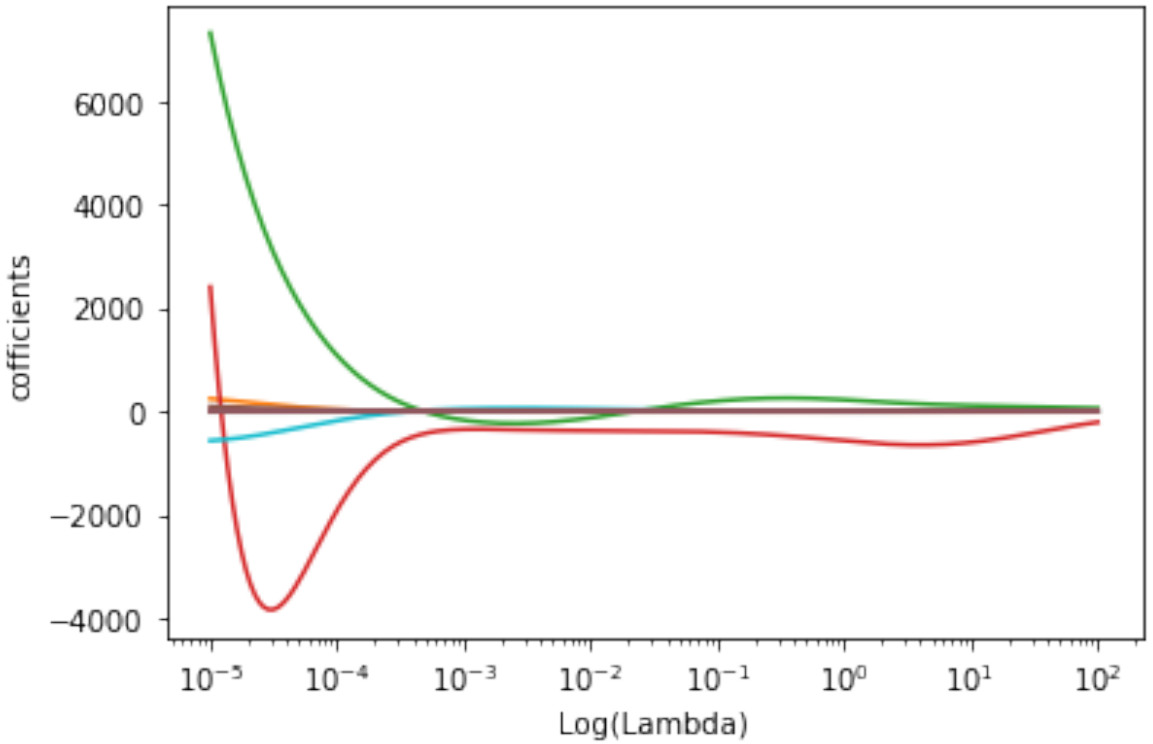
0.011489510001873086



LINEAR_MODEL.RIDGE REGRESSION

1-find out the optimal lambdas ridge_best_lamb

0.011489510001873086



#step2- build model to predict with the best lambdas

```
In [520]: ridge.intercept_
Out[520]: 2962228.1448225877

In [528]: ridge.coef_
Out[528]: array([ 0.00000000e+00, -1.91178786e+00,  6.57856960e+01,
          -2.04326612e+02,
```

```
clf.score(test_set_X,test_set_y)
```

```
0.6202919307446314
```

```
clf.score(X,y)
```

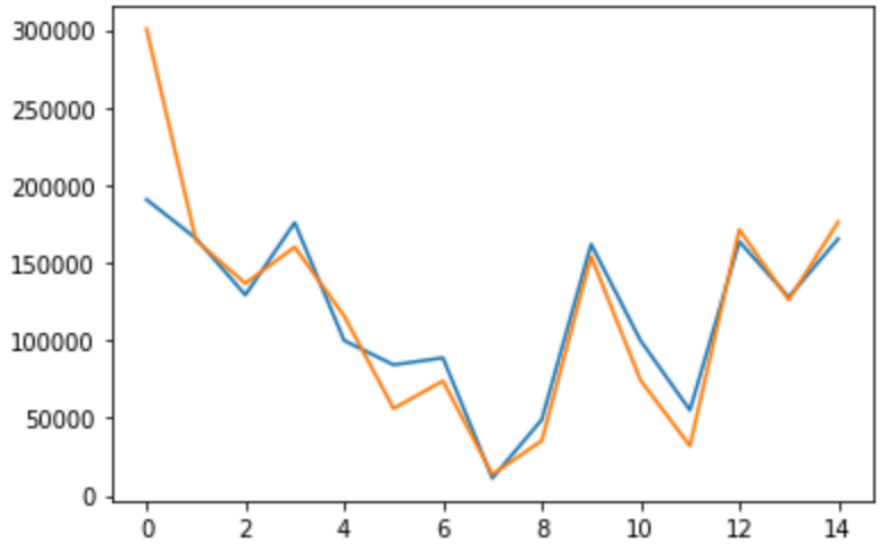
```
0.8857613306825324
```

LINEAR_MODEL.RIDGE REGRESSION

step 3 predict the test-data

```
#plot test set
plt.plot(test_set_y,label='real')
plt.plot(test_y_pre,label='prediction')
```

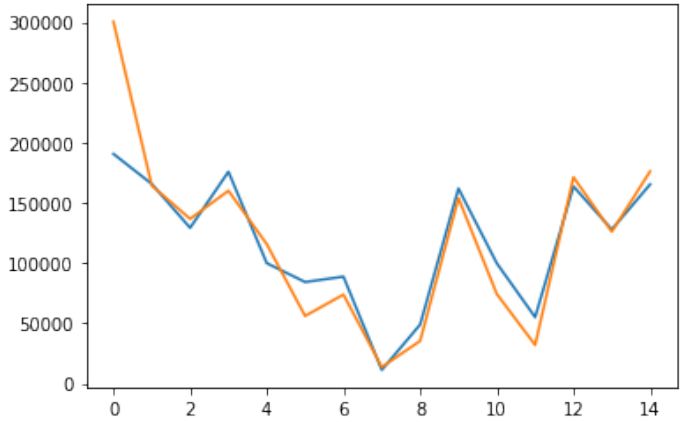
```
[<matplotlib.lines.Line2D at 0x16063d8b0>]
```



	origine data	predict data
0	190651	300471.0
1	165690	164378.0
2	129260	136703.0
3	175833	160058.0
4	99910	115709.0
5	84262	56094.0
6	88735	73859.0
7	11271	13654.0
8	48916	35331.0
9	161910	153958.0
10	99860	74322.0
11	55072	32020.0
12	163662	171299.0
13	127894	126056.0
14	165386	176230.0

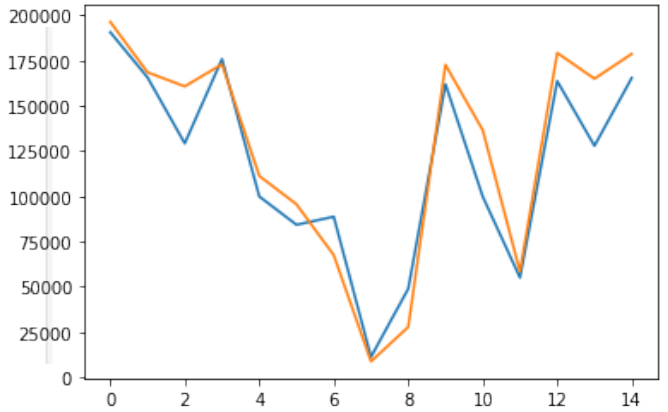
CALCULATE THE ERRORS AND EXPLAIN HOW YOU CAN IMPROVE YOUR RESULTS

```
In [527]: #Calculate the errors
          from sklearn.metrics import mean_squared_error
          rsme=np.sqrt(mean_squared_error(test_set_y,test_y_pre))
          rsme

Out[527]: 31882.18068447046
```

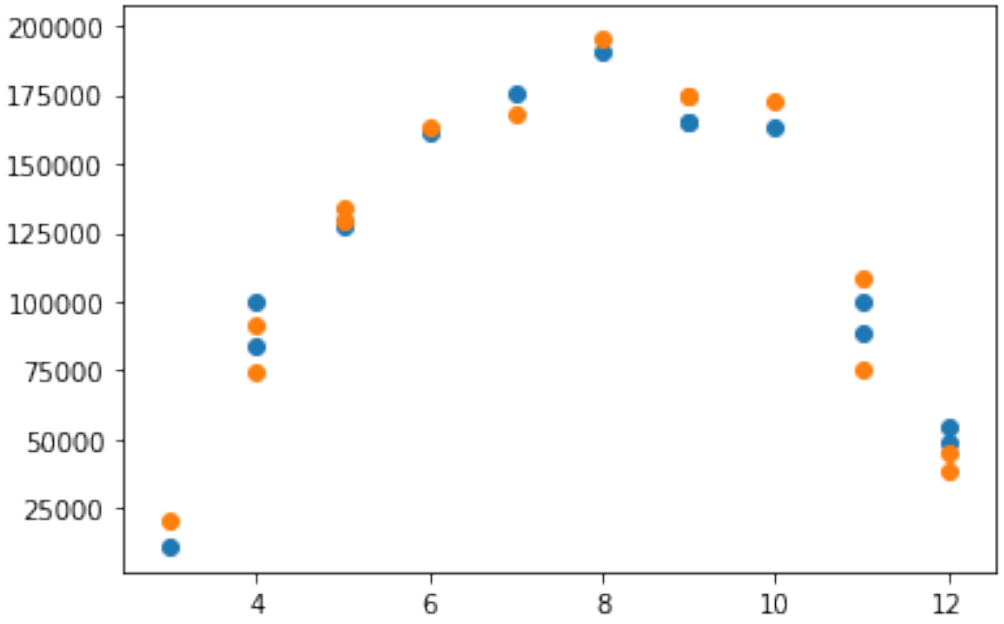


Propose the solution to improve the results: Regard the data, redefine the used features

```
In [588]: #Calculate the errors
          from sklearn.metrics import mean_squared_error
          rsme=np.sqrt(mean_squared_error(test_set_y,test_y_pre))
          rsme

Out[588]: 19092.640826594245
```



MULTIPLE LINEAR REGRESSION



Intercept 2.418590e+07
month -1.530116e+03
yr -1.199612e+04
user_sub 1.393367e+00
dtype: float64

```
from sklearn.metrics import mean_squared_error
rsme=np.sqrt(mean_squared_error(test.od_flow,pred))
rsme
```

```
8363.541698984936
```

	test	predict
31	190651	195341.776947
20	165690	174914.052048
16	129260	134408.281121
30	175833	167829.948034
22	99910	108388.748513
15	84262	74620.503141
10	88735	75901.632758
2	11271	20587.955205
11	48916	38964.672681
29	161910	162871.154504
27	99860	91338.886570
35	55072	45336.682386
33	163662	172877.519062
28	127894	129213.184539
32	165386	174399.274621