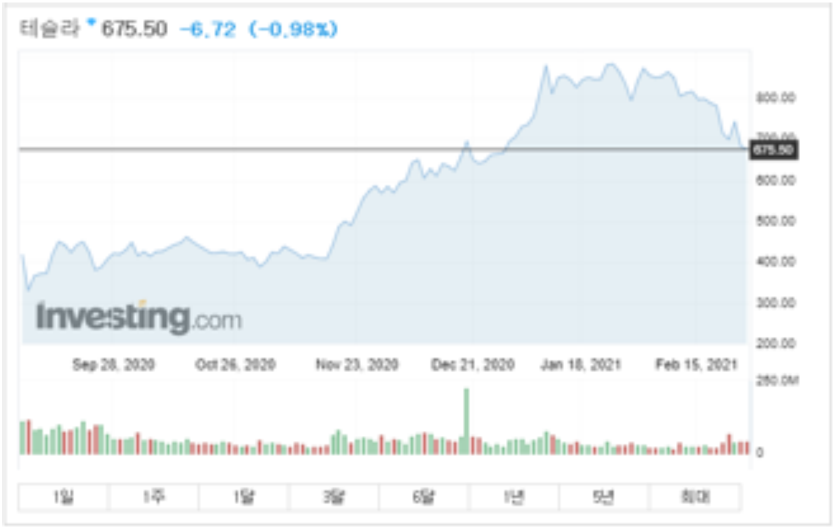
Data handling : Time-Series

ECE30007 Intro to AI Project

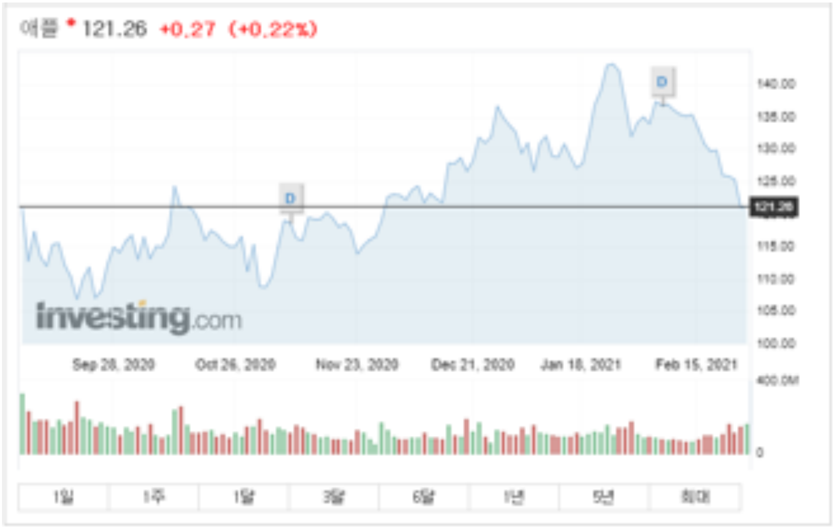
Contents

Time-Series example

- Stock Price



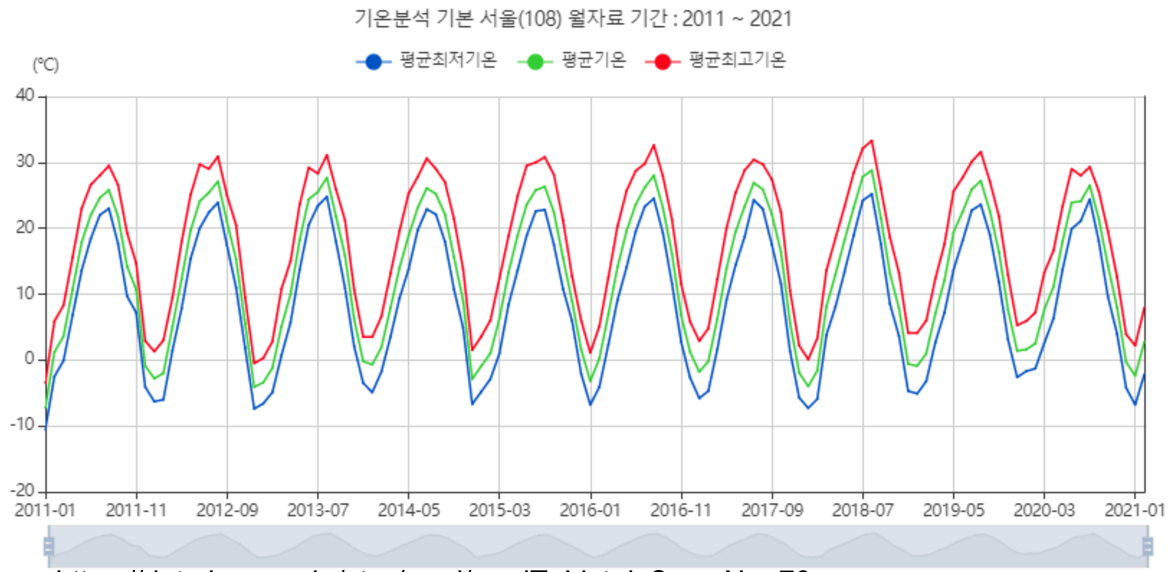
https://kr.investing.com/equities/tesla-motors



https://kr.investing.com/equities/apple-computer-inc

Time-Series example

- Temperature



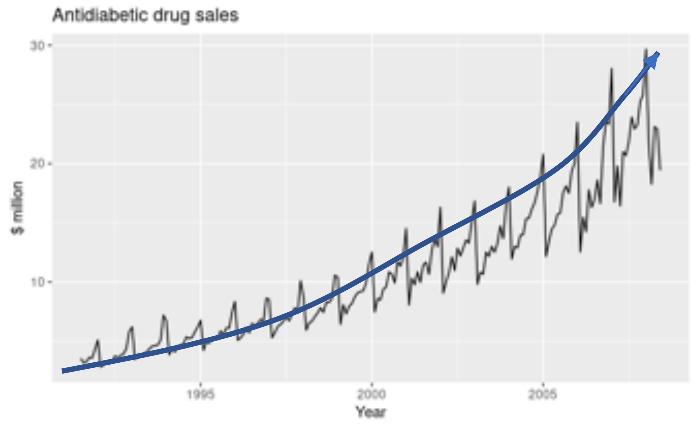
https://data.kma.go.kr/stcs/grnd/grndTaList.do?pgmNo=70

What is Time-Series?

- **Time-Series**
  - Time series is defined as a set of quantitative observation arranged in chronological order.
  - In order words, time series is a series of data points indexed in time order. It is a sequence of discrete-time data with same intervals.
  - A time series graph plots observed values on the y-axis against an increment of time on the x-axis.
- Components
  - **Trend**
  - **Seasonal**
  - **Cyclic**
  - **Irregular**

Time-Series

- Components
  - **Trend** : Long-term change in the mean level. General tendency of data to increase or decrease or stagnate over a long period of time.
    - Time series relating to Economics, Business, and Commerce may show an upward or increasing tendency.
    - Whereas, the time series relating to death rates, birth rates, share prices, etc. may show a downward or decreasing tendency.



- Here, there is a clear and increasing **trend**. There is also a strong **seasonal pattern** that increases in size as the level of the series increases.

https://otexts.com/fpp2/fpp_files/figure-html/a10-1.png

Time-Series

- Components
  - **Seasonal** : A *seasonal* pattern occurs when a time series is affected by seasonal factors such as the time of the year or the day of the week.
    - changes that take place due to the rhythmic forces which operate in a regular and periodic manner.
    - These forces usually have the same or most similar pattern year after year.
    - Seasonality is always of a fixed and known frequency.
    - These variations may be due to seasons, weather conditions, habits, customs or traditions.

Time-Series

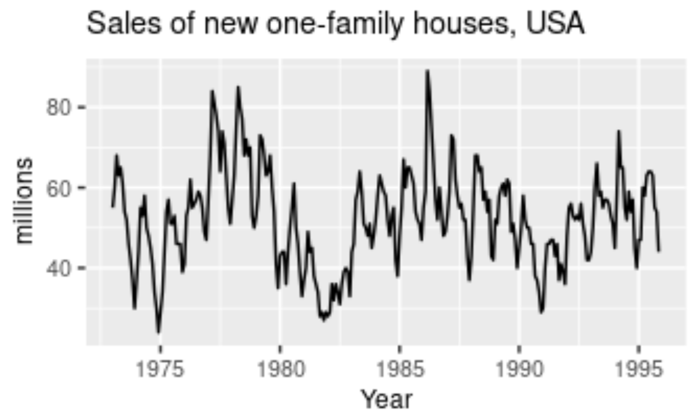
- Components
  - **Cyclic** : Apart from seasonal effects, some time-series exhibit variation at not a fixed period due to some other physical cause. Or some time-series exhibit oscillations, which do not have a fixed period but which are predictable to some extent.
    - These fluctuations are usually due to economic conditions, and are often related to the "business cycle."
      - The duration of these fluctuations is usually at least 2 years.
    - In general, the average length of cycles is longer than the length of a seasonal pattern, and the magnitude of cycles tends to be more variable than the magnitude of seasonal patterns.

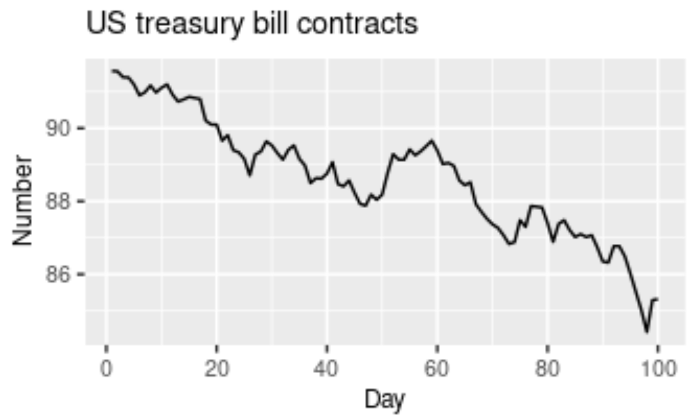
Time-Series

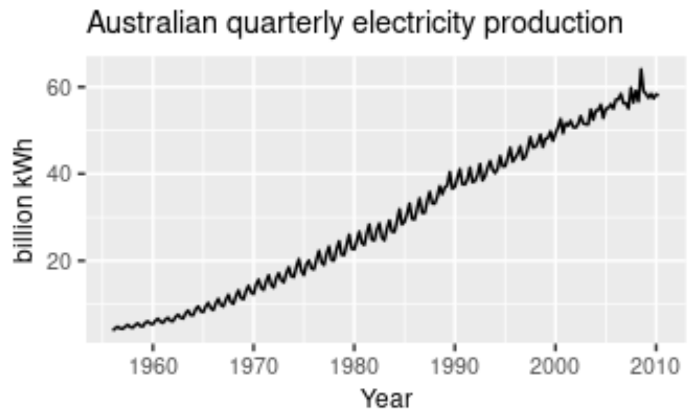
- Components
  - **Irregular** : after trend and cyclic variations have been removed from a data, then it’s left with residuals that may or may not be random.
    - Or random variations are fluctuations which are a result of unforeseen and unpredictable forces.
    - These forces operate in an absolutely random or erratic manner and do not have any definite pattern.
    - Thus, these variations may be due to floods, famines, earthquakes, strikes, etc.

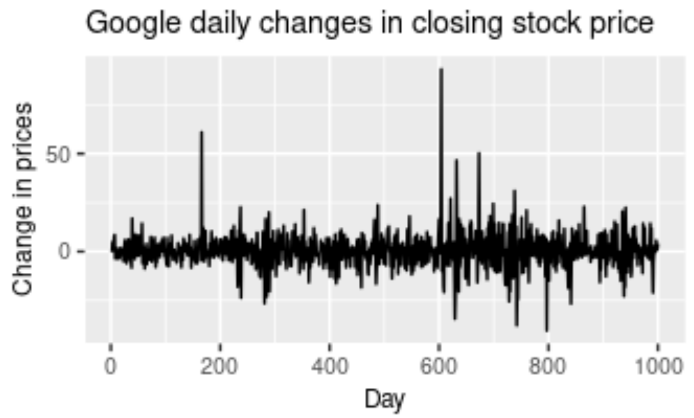
Temporal patterns

- Think what kinds of patterns each plot has.







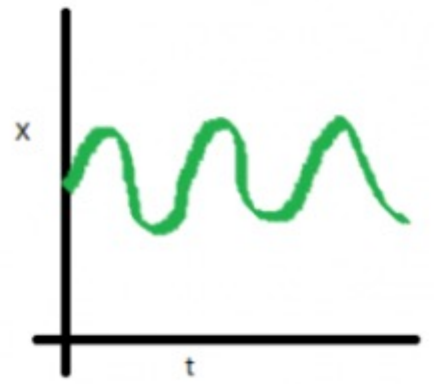


Stationary vs Non-Stationary

- As you can imagine, it is very difficult to forecast time-series data. We cannot predict with certainty what will occur in the future.
- Therefore, most statistical forecasting methods are based on the assumption that the time series can be rendered approximately stationary through the use of mathematical transformations.
- A stationarized series is relatively easy to predict.

Stationary Time-Series

- **Stationary Time Series**: data does not have any upward or downward trend or seasonal effects. Mean or variance are consistent over time
- A stationary time series is one whose properties do not depend on the time at which the series is observed. Thus, time series with trends, or with seasonality, are not stationary
  - the trend and seasonality will affect the value of the time series at different times.

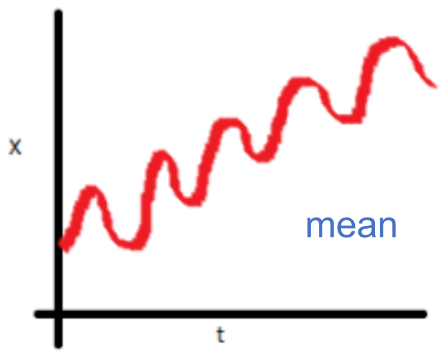


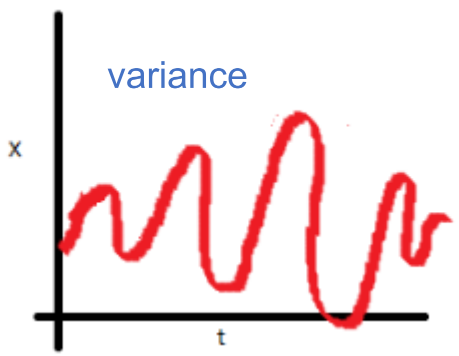
In this plot, variance and covariance are constant with time.
This is what a stationary time series looks like.

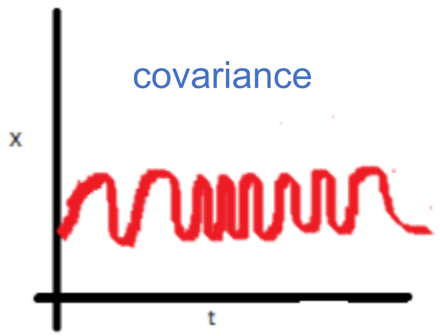
Non-Stationary Time-Series

- **Non-Stationary Time Series**: data show trends, seasonal effects, and other structures depend on time. Forecasting performance is dependent on the time of observation. Mean and variance change over time and a drift in the model is captured.
- A non-stationary time series can be converted to a stationary time series through a technique called differencing.

Non-Stationary Time-Series







- In the first plot, we can clearly see that the mean varies (increases) with time which results in an upward trend. Thus, this is a non-stationary series. For a series to be classified as stationary, it should not exhibit a trend.
- In the second plot, we certainly do not see a trend in the series, but the variance of the series is a function of time. As mentioned previously, a stationary series must have a constant variance.
- In the third plot, the spread becomes closer as the time increases, which implies that the covariance is a function of time.

Differencing

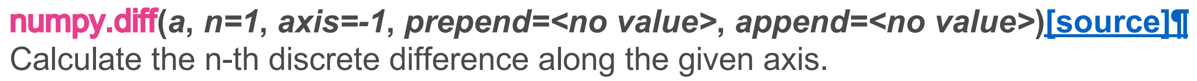
- Differencing is basically subtracting the previous value from the current value of your time series i.e.

$$Y_t' = Y_t - Y_{t-1}$$

- For n observations(data), the differenced series will only have n-1 observations as it is not possible to calculate a difference for the first element of a series.

Differencing

- Sometimes first order difference does not make it stationary, then we may go for higher orders like differencing the first order differenced series
- Differencing stabilizes the mean of the series which helps to eliminate the trend and seasonality. Other methods such as log transform stabilizes the variance.



numpy.diff(*a*, *n=1*, *axis=-1*, *prepend=<no value>*, *append=<no value>*)[source]¶

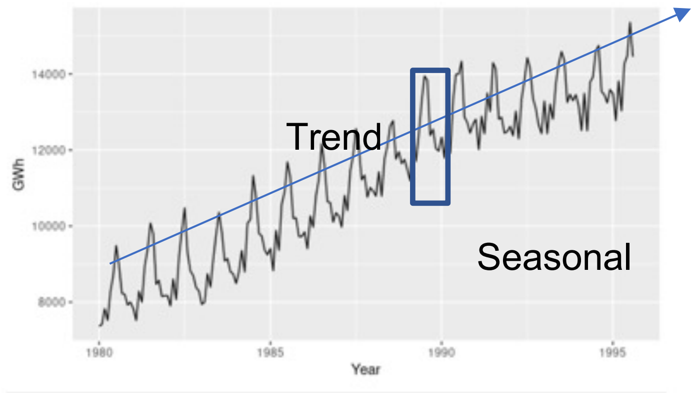
Calculate the n-th discrete difference along the given axis.

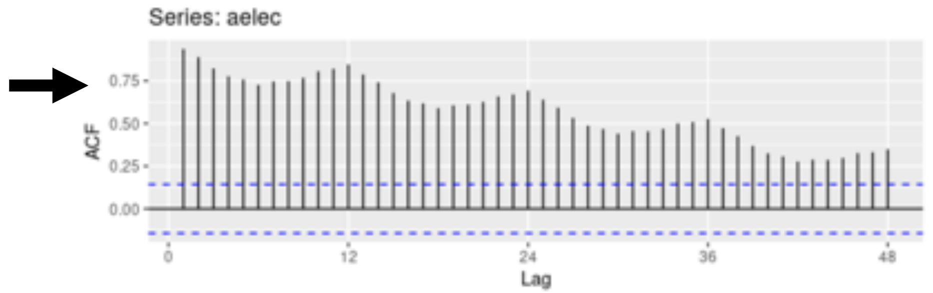
Autocorrelation function (ACF)

- ACF shows not only the lag-one autocorrelation, but the entire autocorrelation function for different lags.
- Any significant non-zero autocorrelations implies that the series can be forecasted from the past data.
- When data have a trend, the autocorrelations for small lags tend to be large and positive because observations nearby in time are also nearby in size. So the ACF of trended time series tends to have positive values that slowly decrease as the lags increase.
- When data are seasonal, the autocorrelations will be larger for the seasonal lags (at multiples of the seasonal frequency) than for other lags.

Autocorrelation function (ACF)

- When data are both trended and seasonal, you see a combination of these effects.





- Import

```
In [51]: from statsmodels.graphics.tsaplots import plot_acf, ~~plot_pacf~~
```

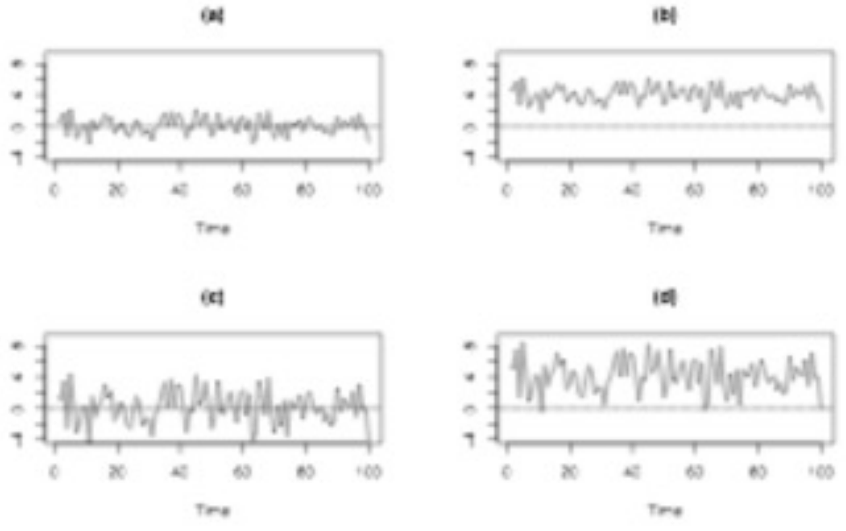
- method

statsmodels.graphics.tsaplots.plot_acf(x, ax=*None*, lags=*None*, *, alpha=*0.05*, use_vlines=*True*, adjusted=*False*, fft=*False*, missing=*'none'*, title=*'Autocorrelation'*, zero=*True*, vlines_kwargs=*None*, **kwargs)

You may need
>> pip install statsmodels

White Noise

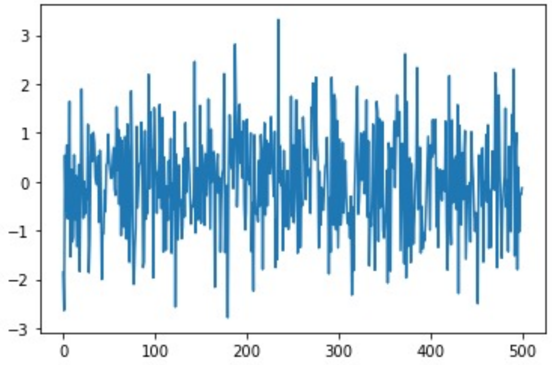
- Mean is constant with time
- Variance is also constant with time
- Zero autocorrelation at all lags.
- A white noise time series is simply a sequence of uncorrelated random variables that are identically distributed.



White Noise

- What does it look like

```
In [102]: noise=np.random.normal(loc= 0, scale = 1, size = 500)
          plt.plot(noise)
```

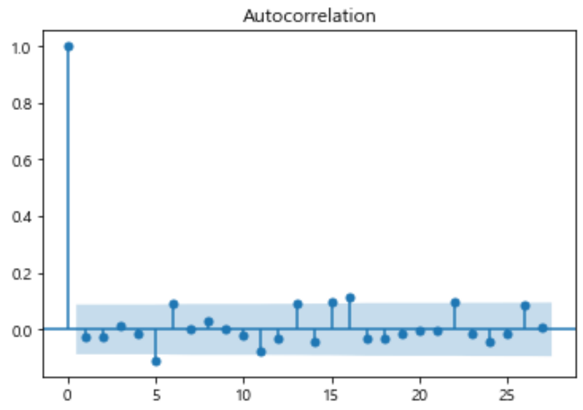


- NumPy random normal creates an array of normally distributed random numbers.
  - The loc argument is the mean.
  - The scale argument is the standard deviation.

White Noise

- What does it look like

```
In [101]: plot_acf(noise)
Out[101]:
```



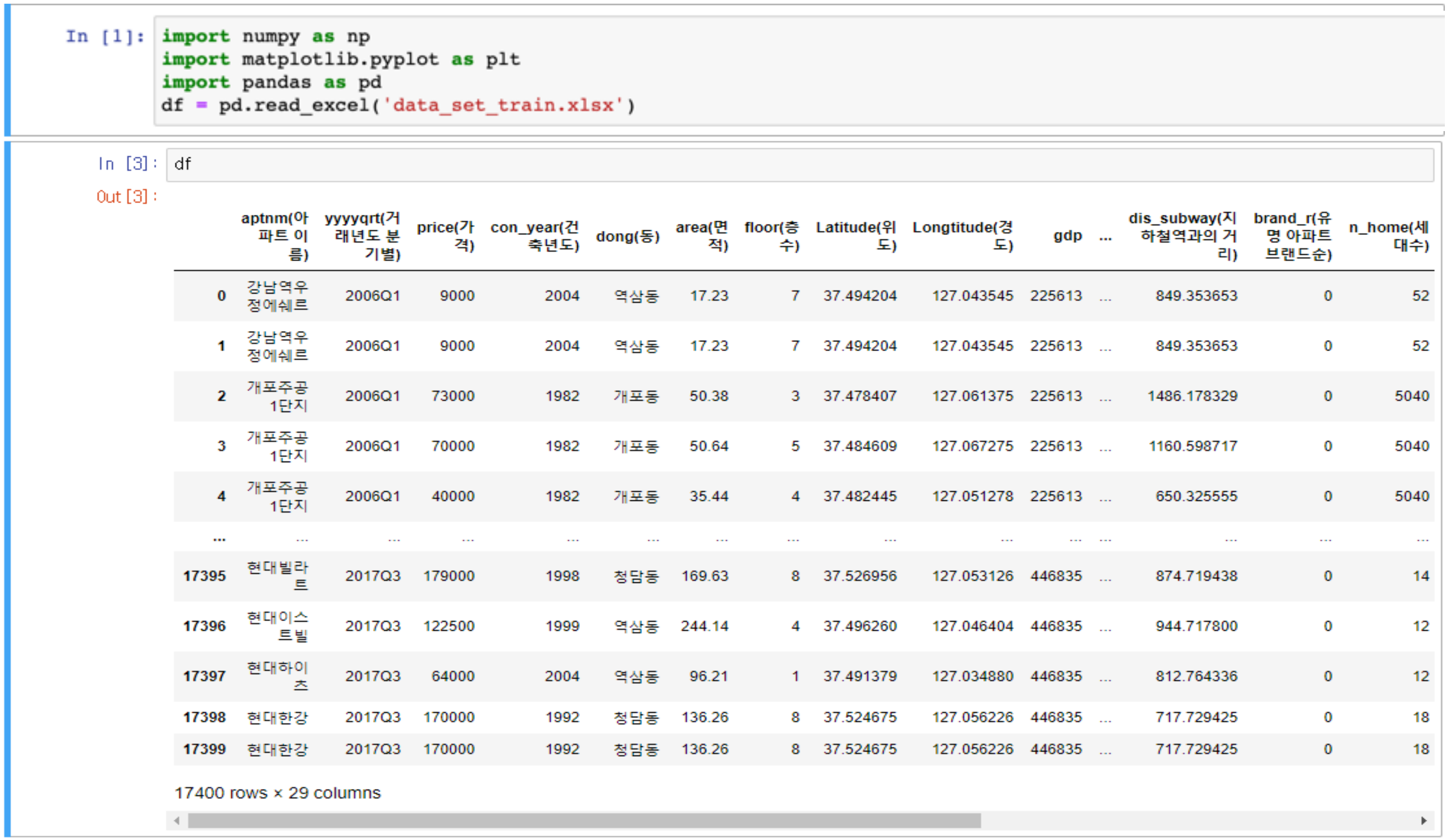
- There is no autocorrelations but itself.

You may need
>> pip install statsmodels

Data Handling

You may need
>> pip install xlrd
>> pip install openpyxl

- Read excel data



In [1]:
```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
df = pd.read_excel('data_set_train.xlsx')
```

In [3]:
```python
df
```

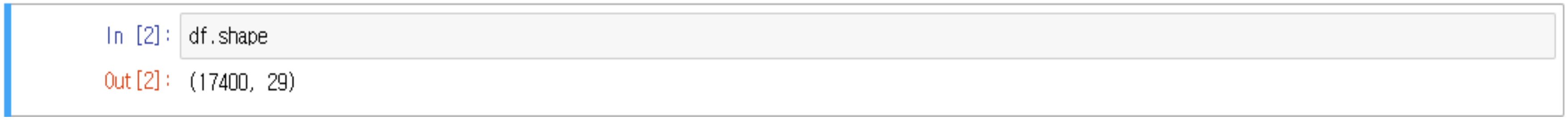
Out[3]:

	aptnm(아파트 이름)	yyyyqrt(거래년도 분기별)	price(가격)	con_year(건축년도)	dong(동)	area(면적)	floor(층수)	Latitude(위도)	Longtitude(경도)	gdp	...	dis_subway(지하철역과의 거리)	brand_r(유명 아파트 브랜드순)	n_home(세대수)
0	강남역우정에쉐르	2006Q1	9000	2004	역삼동	17.23	7	37.494204	127.043545	225613	...	849.353653	0	52
1	강남역우정에쉐르	2006Q1	9000	2004	역삼동	17.23	7	37.494204	127.043545	225613	...	849.353653	0	52
2	개포주공1단지	2006Q1	73000	1982	개포동	50.38	3	37.478407	127.061375	225613	...	1486.178329	0	5040
3	개포주공1단지	2006Q1	70000	1982	개포동	50.64	5	37.484609	127.067275	225613	...	1160.598717	0	5040
4	개포주공1단지	2006Q1	40000	1982	개포동	35.44	4	37.482445	127.051278	225613	...	650.325555	0	5040
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
17395	현대빌라트	2017Q3	179000	1998	청담동	169.63	8	37.526956	127.053126	446835	...	874.719438	0	14
17396	현대이스트빌	2017Q3	122500	1999	역삼동	244.14	4	37.496260	127.046404	446835	...	944.717800	0	12
17397	현대하이츠	2017Q3	64000	2004	역삼동	96.21	1	37.491379	127.034880	446835	...	812.764336	0	12
17398	현대한강	2017Q3	170000	1992	청담동	136.26	8	37.524675	127.056226	446835	...	717.729425	0	18
17399	현대한강	2017Q3	170000	1992	청담동	136.26	8	37.524675	127.056226	446835	...	717.729425	0	18

17400 rows × 29 columns

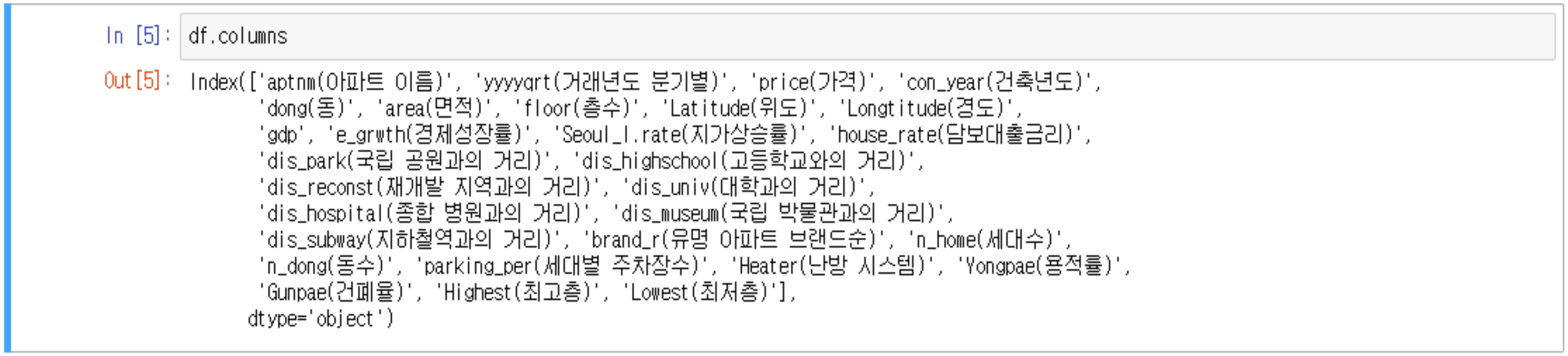
Data Handling

- Let’s see how data looks like
  - df.shape



```
In [2]: df.shape
Out[2]: (17400, 29)
```

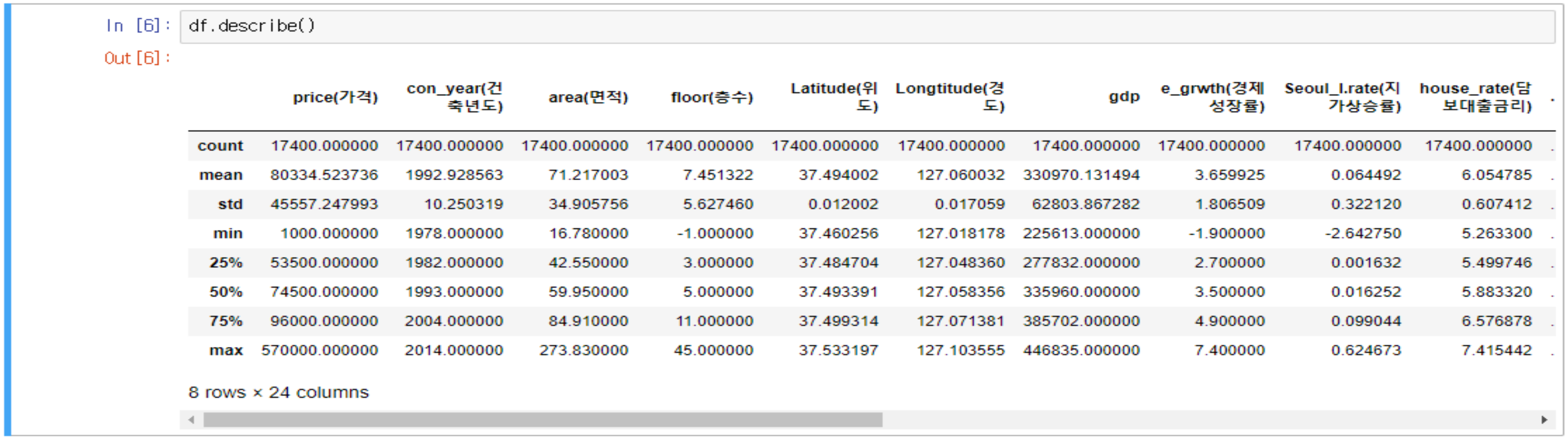
  - df.columns



```
In [5]: df.columns
Out[5]: Index(['aptnm(아파트 이름)', 'yyyyqrt(거래년도 분기별)', 'price(가격)', 'con_year(건축년도)',
               'dong(동)', 'area(면적)', 'floor(층수)', 'Latitude(위도)', 'Longtitude(경도)',
               'gdp', 'e_grwth(경제성장률)', 'Seoul_l.rate(지가상승률)', 'house_rate(담보대출금리)',
               'dis_park(국립 공원과의 거리)', 'dis_highschool(고등학교와의 거리)',
               'dis_reconst(재개발 지역과의 거리)', 'dis_univ(대학과의 거리)',
               'dis_hospital(종합 병원과의 거리)', 'dis_museum(국립 박물관과의 거리)',
               'dis_subway(지하철역과의 거리)', 'brand_r(유명 아파트 브랜드순)', 'n_home(세대수)',
               'n_dong(동수)', 'parking_per(세대별 주차장수)', 'Heater(난방 시스템)', 'Yongpae(용적률)',
               'Gunpae(건폐율)', 'Highest(최고층)', 'Lowest(최저층)'],
              dtype='object')
```

Data Handling

- Let’s see how data looks like
  - df.describe()



```
In [6]: df.describe()
```

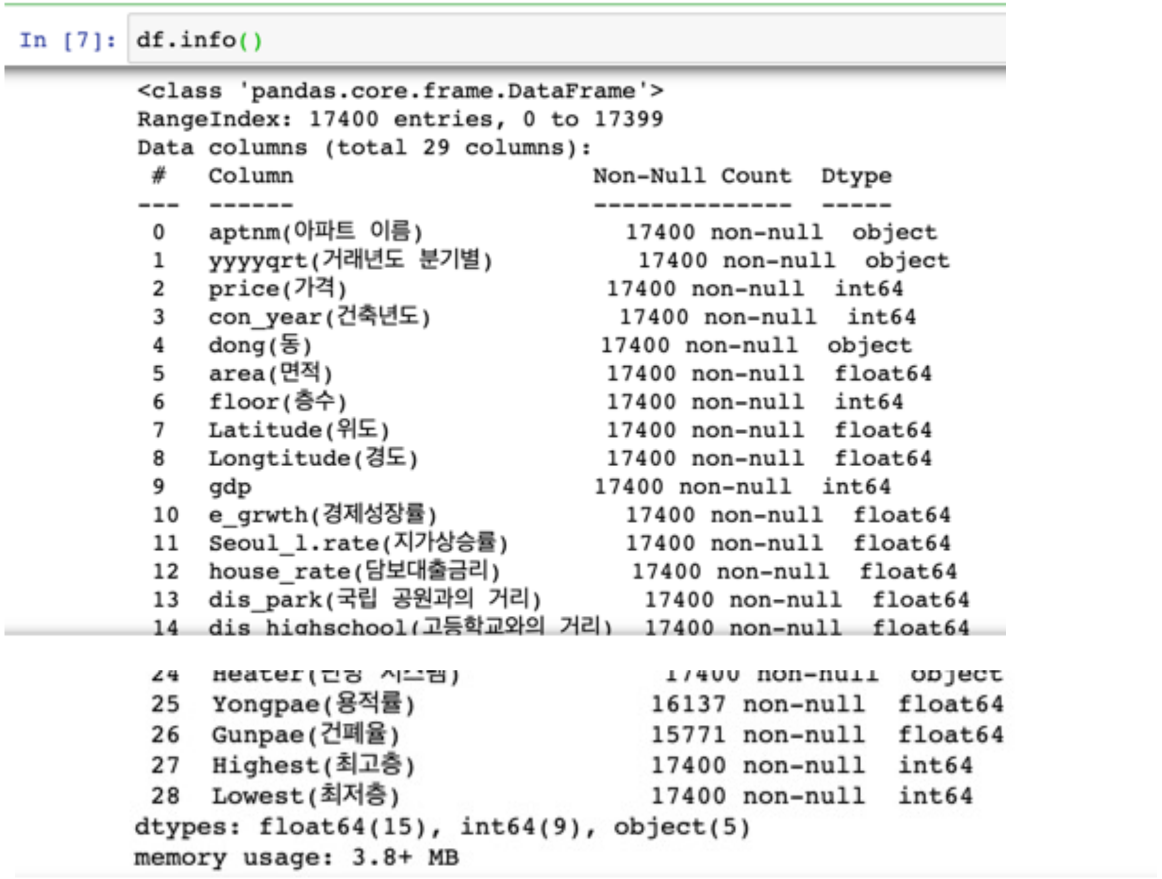
Out [6]:

	price(가격)	con_year(건축년도)	area(면적)	floor(층수)	Latitude(위도)	Longtitude(경도)	gdp	e_grwth(경제성장률)	Seoul_I.rate(지가상승률)	house_rate(담보대출금리)	.
count	17400.000000	17400.000000	17400.000000	17400.000000	17400.000000	17400.000000	17400.000000	17400.000000	17400.000000	17400.000000	.
mean	80334.523736	1992.928563	71.217003	7.451322	37.494002	127.060032	330970.131494	3.659925	0.064492	6.054785	.
std	45557.247993	10.250319	34.905756	5.627460	0.012002	0.017059	62803.867282	1.806509	0.322120	0.607412	.
min	1000.000000	1978.000000	16.780000	-1.000000	37.460256	127.018178	225613.000000	-1.900000	-2.642750	5.263300	.
25%	53500.000000	1982.000000	42.550000	3.000000	37.484704	127.048360	277832.000000	2.700000	0.001632	5.499746	.
50%	74500.000000	1993.000000	59.950000	5.000000	37.493391	127.058356	335960.000000	3.500000	0.016252	5.883320	.
75%	96000.000000	2004.000000	84.910000	11.000000	37.499314	127.071381	385702.000000	4.900000	0.099044	6.576878	.
max	570000.000000	2014.000000	273.830000	45.000000	37.533197	127.103555	446835.000000	7.400000	0.624673	7.415442	.

8 rows × 24 columns

Data Handling

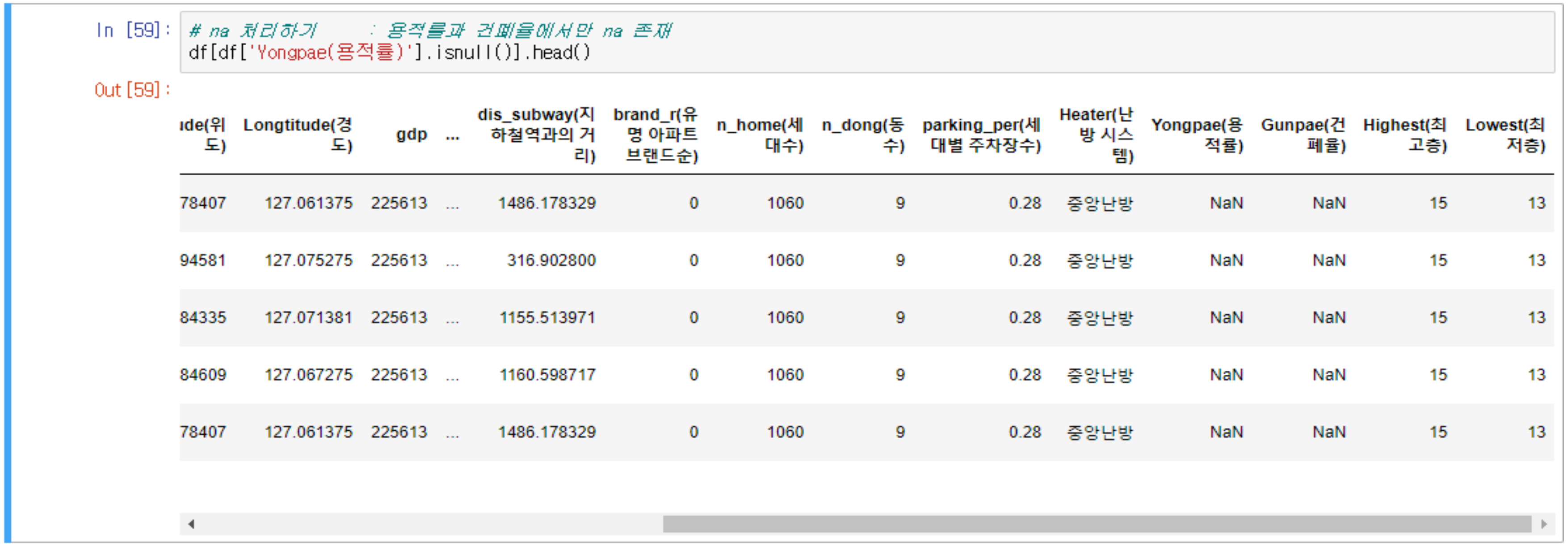
- Let’s see how data looks like
  - df.info()



```
In [7]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17400 entries, 0 to 17399
Data columns (total 29 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   aptnm(아파트 이름)              17400 non-null  object
 1   yyyyqrt(거래년도 분기별)          17400 non-null  object
 2   price(가격)                   17400 non-null  int64
 3   con_year(건축년도)              17400 non-null  int64
 4   dong(동)                      17400 non-null  object
 5   area(면적)                    17400 non-null  float64
 6   floor(층수)                   17400 non-null  int64
 7   Latitude(위도)                17400 non-null  float64
 8   Longtitude(경도)              17400 non-null  float64
 9   gdp                          17400 non-null  int64
 10  e_grwth(경제성장률)              17400 non-null  float64
 11  Seoul_l.rate(지가상승률)          17400 non-null  float64
 12  house_rate(담보대출금리)          17400 non-null  float64
 13  dis_park(국립 공원과의 거리)        17400 non-null  float64
 14  dis_highschool(고등학교와의 거리)    17400 non-null  float64
 24  heater(난방 시스템)                17400 non-null  object
 25  Yongpae(용적률)                  16137 non-null  float64
 26  Gunpae(건폐율)                   15771 non-null  float64
 27  Highest(최고층)                  17400 non-null  int64
 28  Lowest(최저층)                   17400 non-null  int64
dtypes: float64(15), int64(9), object(5)
memory usage: 3.8+ MB
```

Data Handling

- Let’s see how data looks like
  - Check missing value



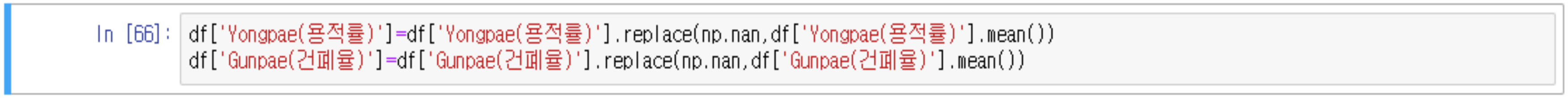
```
In [59]: # na 처리하기      : 용적률과 건폐율에서만 na 존재
         df[df['Yongpae(용적률)'].isnull()].head()
```

Out [59]:

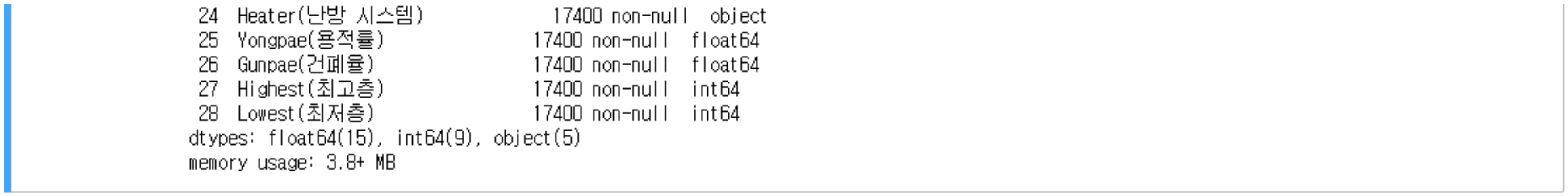
ıde(위 도)	Longtitude(경 도)	gdp	...	dis_subway(지 하철역과의 거 리)	brand_r(유 명 아파트 브랜드순)	n_home(세 대수)	n_dong(동 수)	parking_per(세 대별 주차장수)	Heater(난 방 시스 템)	Yongpae(용 적률)	Gunpae(건 폐율)	Highest(최 고층)	Lowest(최 저층)
78407	127.061375	225613	...	1486.178329	0	1060	9	0.28	중앙난방	NaN	NaN	15	13
94581	127.075275	225613	...	316.902800	0	1060	9	0.28	중앙난방	NaN	NaN	15	13
84335	127.071381	225613	...	1155.513971	0	1060	9	0.28	중앙난방	NaN	NaN	15	13
84609	127.067275	225613	...	1160.598717	0	1060	9	0.28	중앙난방	NaN	NaN	15	13
78407	127.061375	225613	...	1486.178329	0	1060	9	0.28	중앙난방	NaN	NaN	15	13

Handling NaN data

- Dealing with missing data
  - Replace NaN data with the average value of the column values.



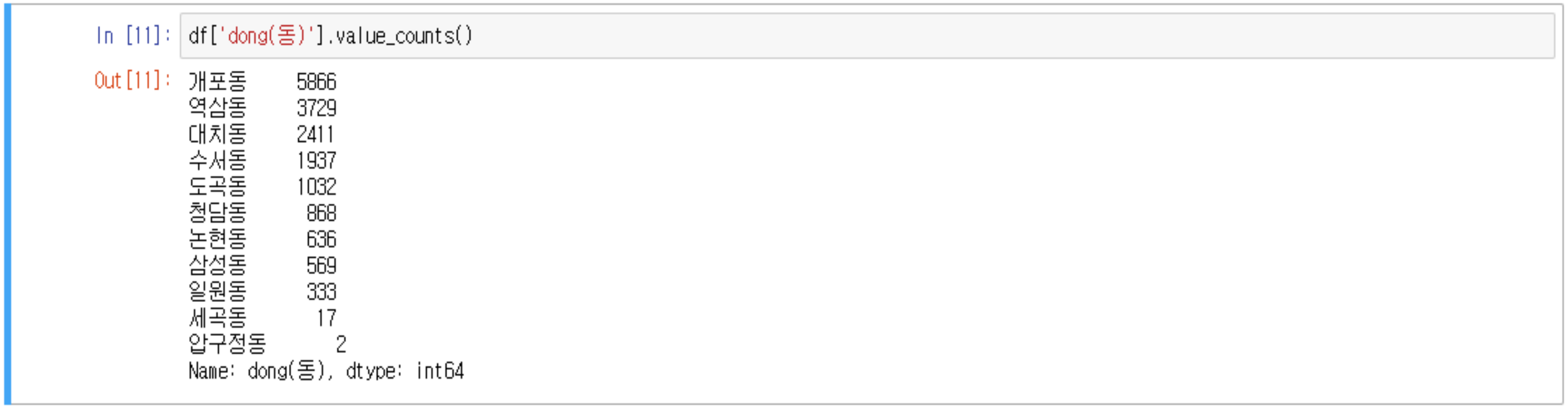
```
In [66]: df['Yongpae(용적률)']=df['Yongpae(용적률)'].replace(np.nan,df['Yongpae(용적률)'].mean())
         df['Gunpae(건폐율)']=df['Gunpae(건폐율)'].replace(np.nan,df['Gunpae(건폐율)'].mean())
```



```
 24  Heater(난방 시스템)          17400 non-null  object
 25  Yongpae(용적률)             17400 non-null  float64
 26  Gunpae(건폐율)              17400 non-null  float64
 27  Highest(최고층)             17400 non-null  int64
 28  Lowest(최저층)              17400 non-null  int64
dtypes: float64(15), int64(9), object(5)
memory usage: 3.8+ MB
```

Data Handling

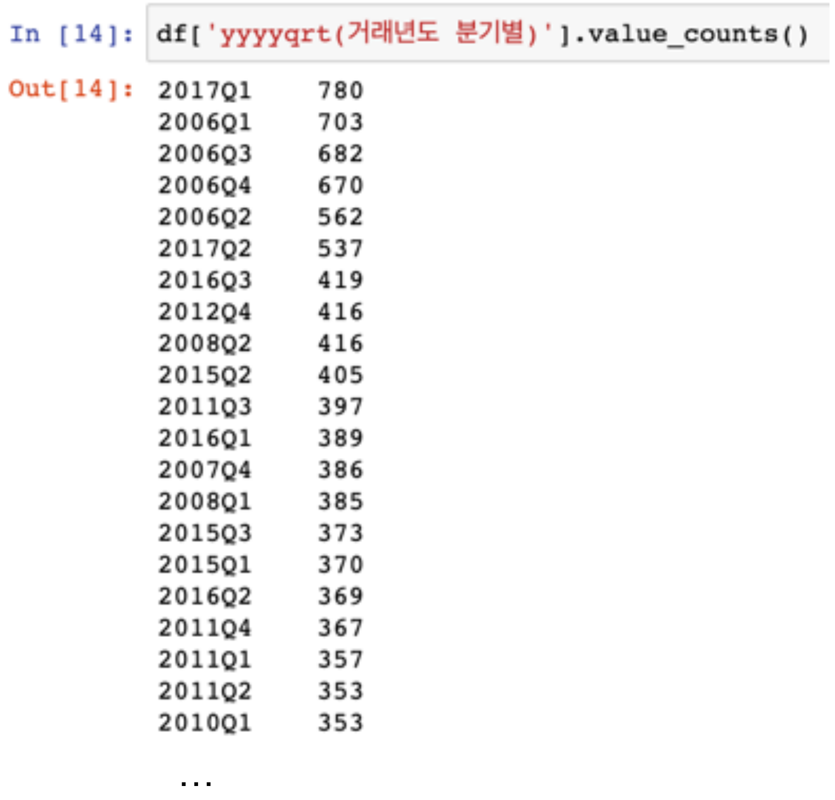
- Let’s see how data looks like



```
In [11]: df['dong(동)'].value_counts()

Out[11]: 개포동     5866
         역삼동     3729
         대치동     2411
         수서동     1937
         도곡동     1032
         청담동      868
         논현동      636
         삼성동      569
         일원동      333
         세곡동       17
         압구정동       2
         Name: dong(동), dtype: int64
```

Data Handling

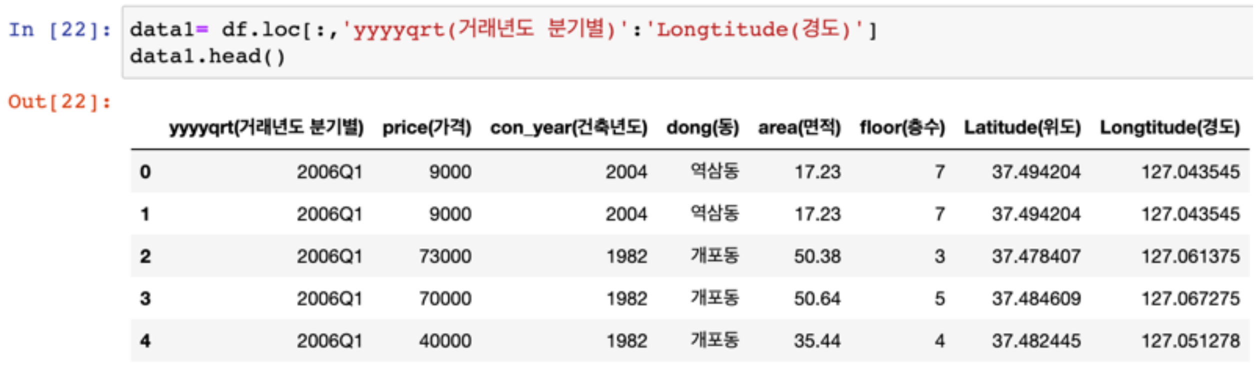
- Let’s see how data looks like



```
In [14]: df['yyyyqrt(거래년도 분기별)'].value_counts()

Out[14]: 2017Q1    780
         2006Q1    703
         2006Q3    682
         2006Q4    670
         2006Q2    562
         2017Q2    537
         2016Q3    419
         2012Q4    416
         2008Q2    416
         2015Q2    405
         2011Q3    397
         2016Q1    389
         2007Q4    386
         2008Q1    385
         2015Q3    373
         2015Q1    370
         2016Q2    369
         2011Q4    367
         2011Q1    357
         2011Q2    353
         2010Q1    353
           …
```

Data Handling

- Basic
  - Slice Dataframe(1)

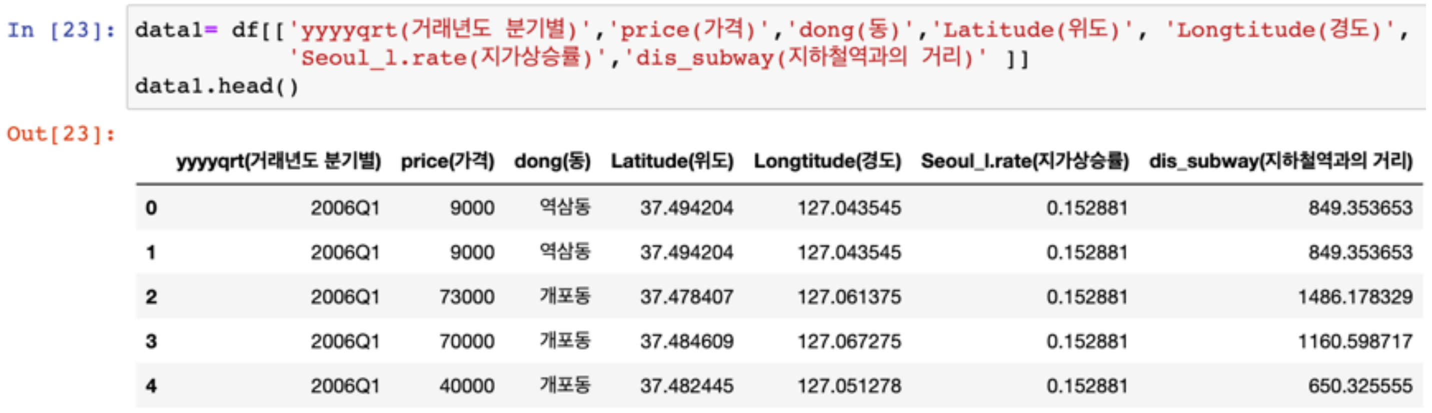


```
In [22]: data1= df.loc[:,'yyyyqrt(거래년도 분기별)':'Longtitude(경도)']
         data1.head()
```

Out[22]:

	yyyyqrt(거래년도 분기별)	price(가격)	con_year(건축년도)	dong(동)	area(면적)	floor(층수)	Latitude(위도)	Longtitude(경도)
0	2006Q1	9000	2004	역삼동	17.23	7	37.494204	127.043545
1	2006Q1	9000	2004	역삼동	17.23	7	37.494204	127.043545
2	2006Q1	73000	1982	개포동	50.38	3	37.478407	127.061375
3	2006Q1	70000	1982	개포동	50.64	5	37.484609	127.067275
4	2006Q1	40000	1982	개포동	35.44	4	37.482445	127.051278

  - Slice Dataframe(2)



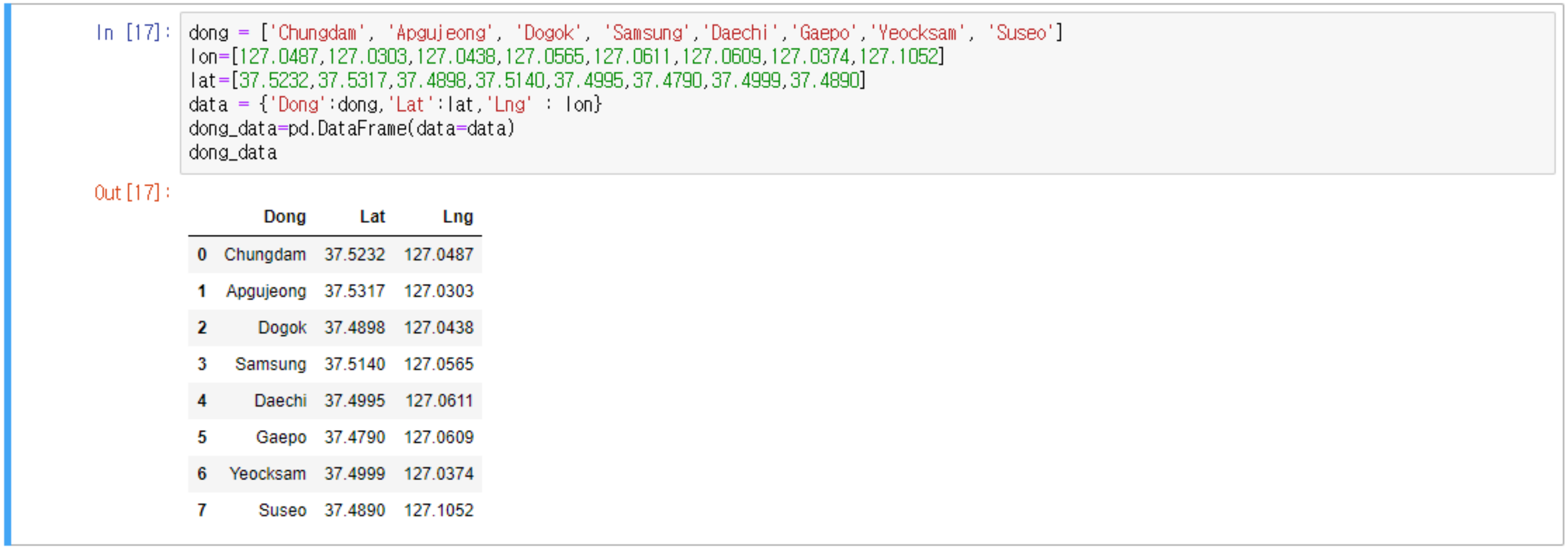
```
In [23]: data1= df[['yyyyqrt(거래년도 분기별)','price(가격)','dong(동)','Latitude(위도)', 'Longtitude(경도)',
                    'Seoul_l.rate(지가상승률)','dis_subway(지하철역과의 거리)' ]]
         data1.head()
```

Out[23]:

	yyyyqrt(거래년도 분기별)	price(가격)	dong(동)	Latitude(위도)	Longtitude(경도)	Seoul_l.rate(지가상승률)	dis_subway(지하철역과의 거리)
0	2006Q1	9000	역삼동	37.494204	127.043545	0.152881	849.353653
1	2006Q1	9000	역삼동	37.494204	127.043545	0.152881	849.353653
2	2006Q1	73000	개포동	37.478407	127.061375	0.152881	1486.178329
3	2006Q1	70000	개포동	37.484609	127.067275	0.152881	1160.598717
4	2006Q1	40000	개포동	37.482445	127.051278	0.152881	650.325555

Data Handling

- Visualize the graph considering the Location

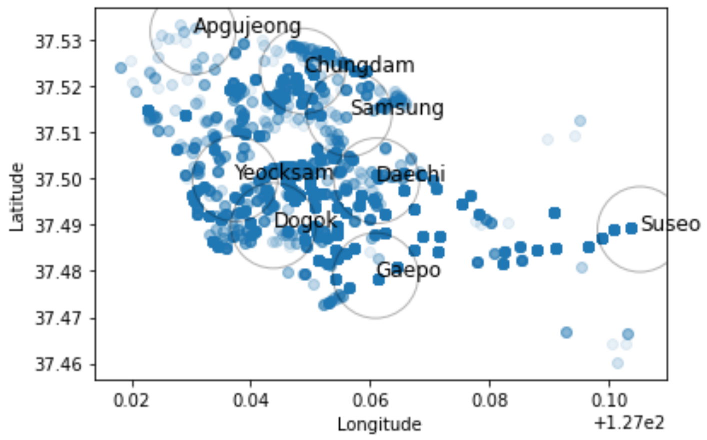


```
In [17]: dong = ['Chungdam', 'Apgujeong', 'Dogok', 'Samsung','Daechi','Gaepo','Yeocksam', 'Suseo']
         lon=[127.0487,127.0303,127.0438,127.0565,127.0611,127.0609,127.0374,127.1052]
         lat=[37.5232,37.5317,37.4898,37.5140,37.4995,37.4790,37.4999,37.4890]
         data = {'Dong':dong,'Lat':lat,'Lng' : lon}
         dong_data=pd.DataFrame(data=data)
         dong_data

Out[17]:
        Dong      Lat       Lng
0   Chungdam  37.5232  127.0487
1  Apgujeong  37.5317  127.0303
2      Dogok  37.4898  127.0438
3    Samsung  37.5140  127.0565
4     Daechi  37.4995  127.0611
5      Gaepo  37.4790  127.0609
6   Yeocksam  37.4999  127.0374
7      Suseo  37.4890  127.1052
```

Exercise(2) –Visualization

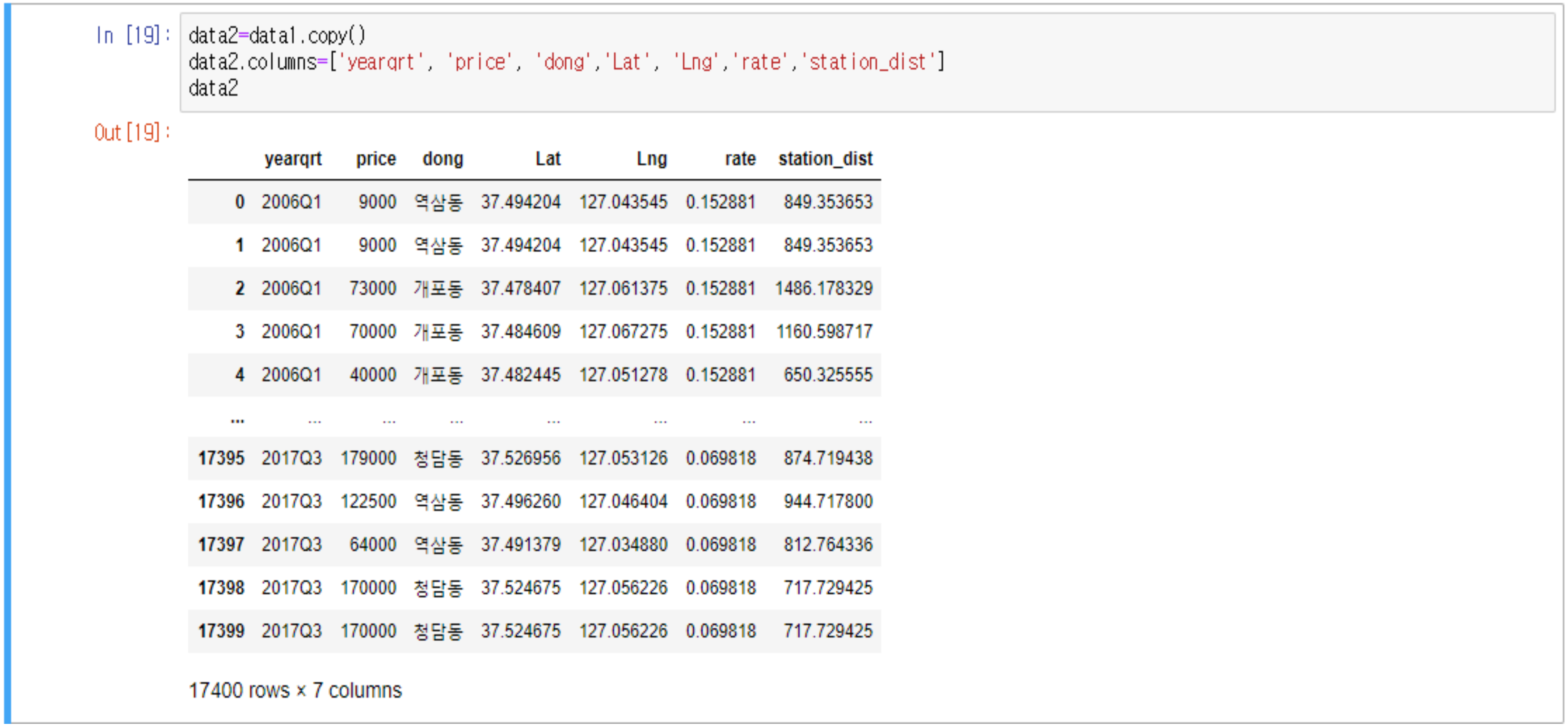
- Visualize the graph considering the Location

```
In [18]: [redacted]
         for i in range(8):
             plt.text(dong_data['Lng'][i], dong_data['Lat'][i],dong_data['Dong'][i],fontsize=12)
         plt.scatter(dong_data['Lng'], dong_data['Lat'],edgecolors="black",c="None", s=2500, alpha=0.3)

Out[18]: <matplotlib.collections.PathCollection at 0x22af38dda30>
```



Data Handling

- Basic
  - Since column names are too complicated, let’s change them to more readable names



```
In [19]: data2=data1.copy()
         data2.columns=['yearqrt', 'price', 'dong','Lat', 'Lng','rate','station_dist']
         data2
```

Out [19]:

	yearqrt	price	dong	Lat	Lng	rate	station_dist
0	2006Q1	9000	역삼동	37.494204	127.043545	0.152881	849.353653
1	2006Q1	9000	역삼동	37.494204	127.043545	0.152881	849.353653
2	2006Q1	73000	개포동	37.478407	127.061375	0.152881	1486.178329
3	2006Q1	70000	개포동	37.484609	127.067275	0.152881	1160.598717
4	2006Q1	40000	개포동	37.482445	127.051278	0.152881	650.325555
...	...	...	...	...	...	...	...
17395	2017Q3	179000	청담동	37.526956	127.053126	0.069818	874.719438
17396	2017Q3	122500	역삼동	37.496260	127.046404	0.069818	944.717800
17397	2017Q3	64000	역삼동	37.491379	127.034880	0.069818	812.764336
17398	2017Q3	170000	청담동	37.524675	127.056226	0.069818	717.729425
17399	2017Q3	170000	청담동	37.524675	127.056226	0.069818	717.729425

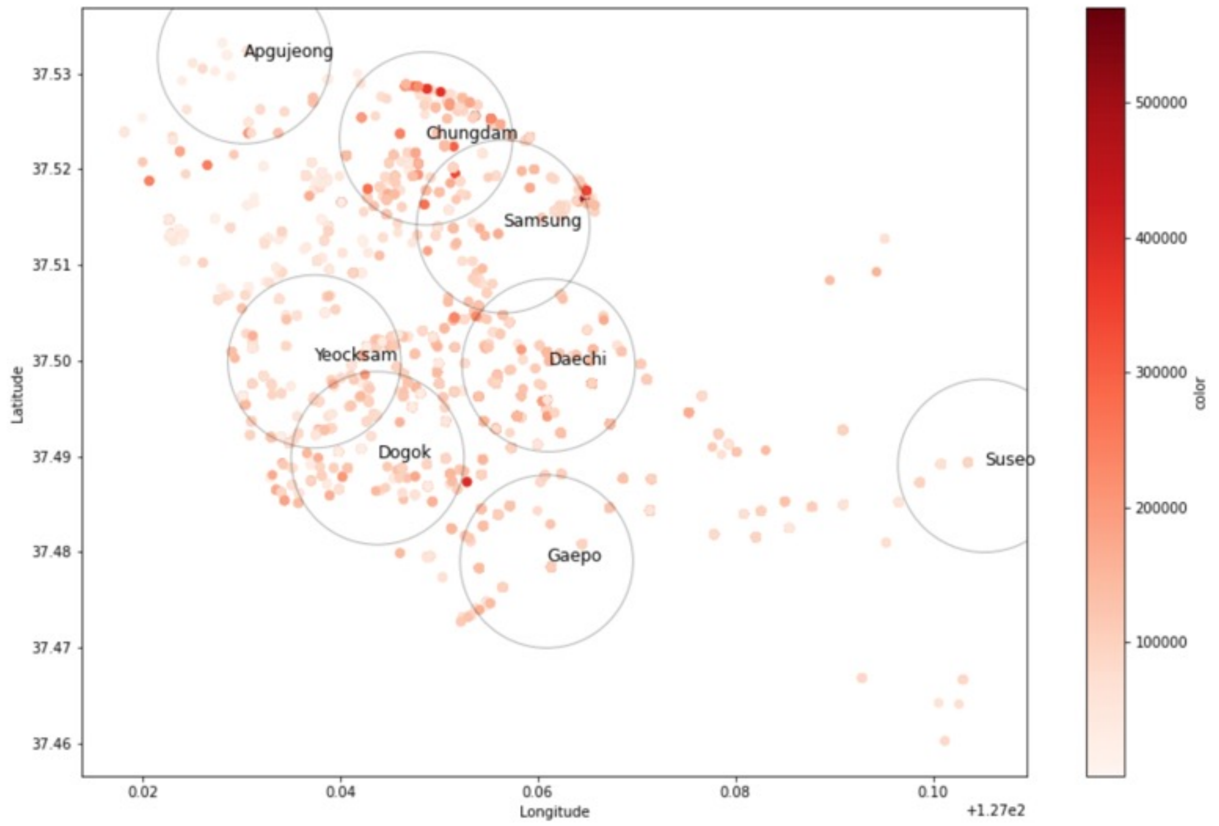
17400 rows × 7 columns

Data Handling

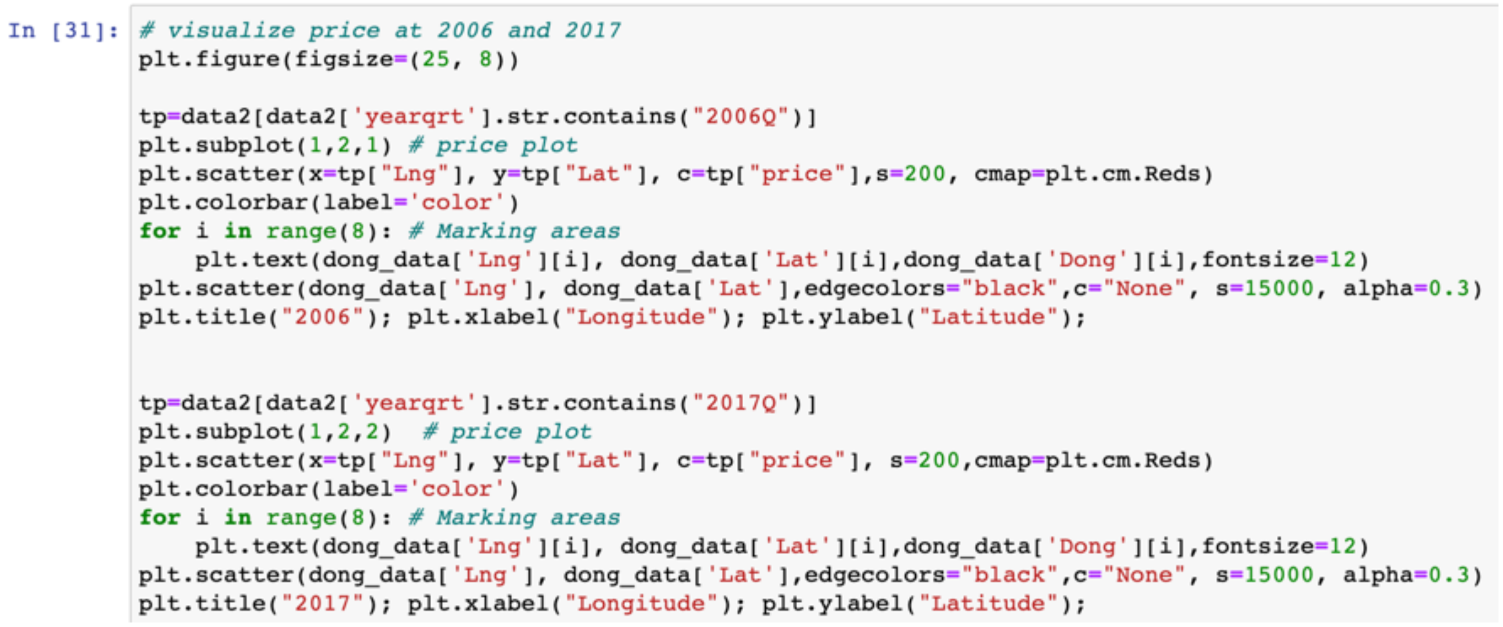
- Visualize the graph considering the Location & Price

```
plt.figure(figsize=(25, 15))
plt.scatter(x=data2["Lng"], y=data2["Lat"], c=data2["price"], cmap=plt.cm.Reds)
plt.colorbar(label='color')
for i in range(8):
    plt.text(dong_data['Lng'][i], dong_data['Lat'][i],dong_data['Dong'][i],fontsize=12)
plt.scatter(dong_data['Lng'], dong_data['Lat'],edgecolors="black",c="None", s=15000, alpha=0.3)
plt.xlabel("Longitude")
plt.ylabel("Latitude")
```

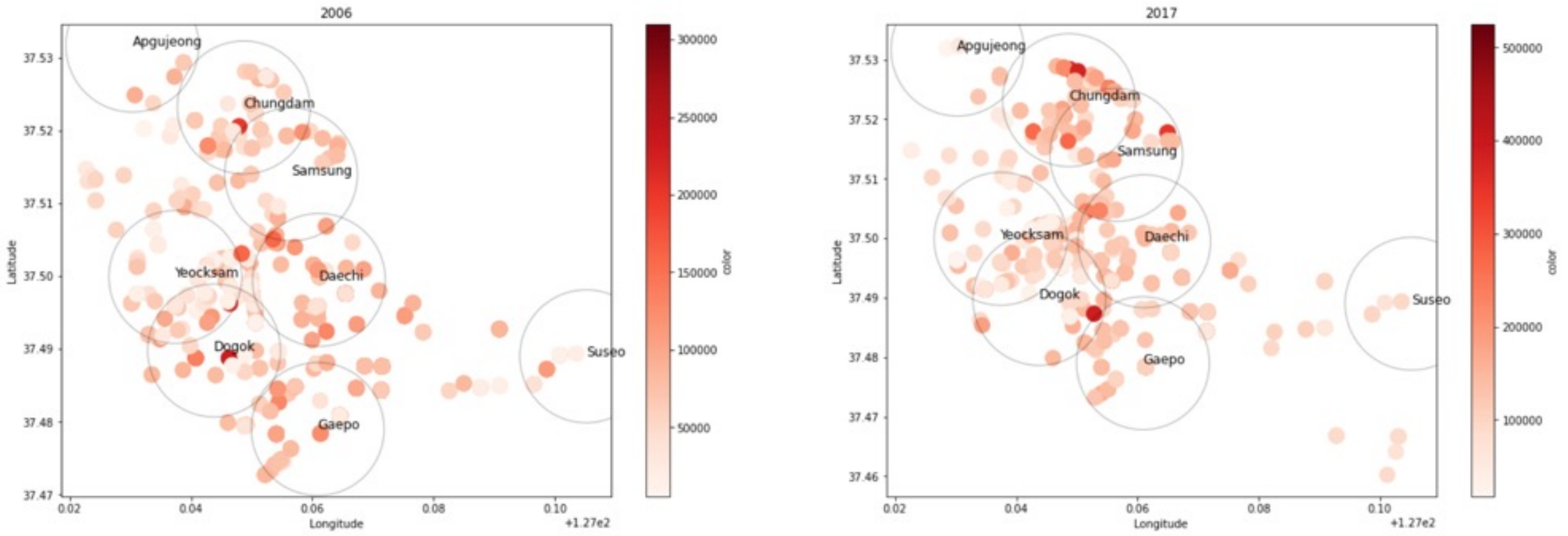
```
Text(0, 0.5, 'Latitude')
```



visualization of price in 2006 and 2017

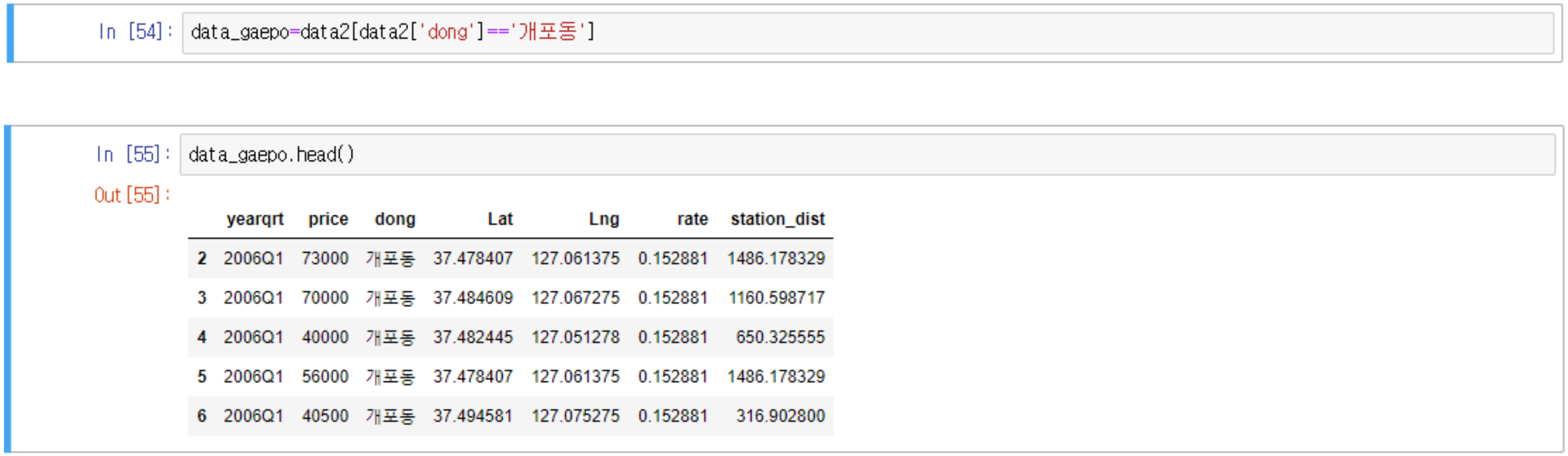


```
In [31]: # visualize price at 2006 and 2017
         plt.figure(figsize=(25, 8))

         tp=data2[data2['yearqrt'].str.contains("2006Q")]
         plt.subplot(1,2,1) # price plot
         plt.scatter(x=tp["Lng"], y=tp["Lat"], c=tp["price"],s=200, cmap=plt.cm.Reds)
         plt.colorbar(label='color')
         for i in range(8): # Marking areas
             plt.text(dong_data['Lng'][i], dong_data['Lat'][i],dong_data['Dong'][i],fontsize=12)
         plt.scatter(dong_data['Lng'], dong_data['Lat'],edgecolors="black",c="None", s=15000, alpha=0.3)
         plt.title("2006"); plt.xlabel("Longitude"); plt.ylabel("Latitude");


         tp=data2[data2['yearqrt'].str.contains("2017Q")]
         plt.subplot(1,2,2)  # price plot
         plt.scatter(x=tp["Lng"], y=tp["Lat"], c=tp["price"], s=200,cmap=plt.cm.Reds)
         plt.colorbar(label='color')
         for i in range(8): # Marking areas
             plt.text(dong_data['Lng'][i], dong_data['Lat'][i],dong_data['Dong'][i],fontsize=12)
         plt.scatter(dong_data['Lng'], dong_data['Lat'],edgecolors="black",c="None", s=15000, alpha=0.3)
         plt.title("2017"); plt.xlabel("Longitude"); plt.ylabel("Latitude");
```



Data Handling

- See how ‘개포동’ price changes over time



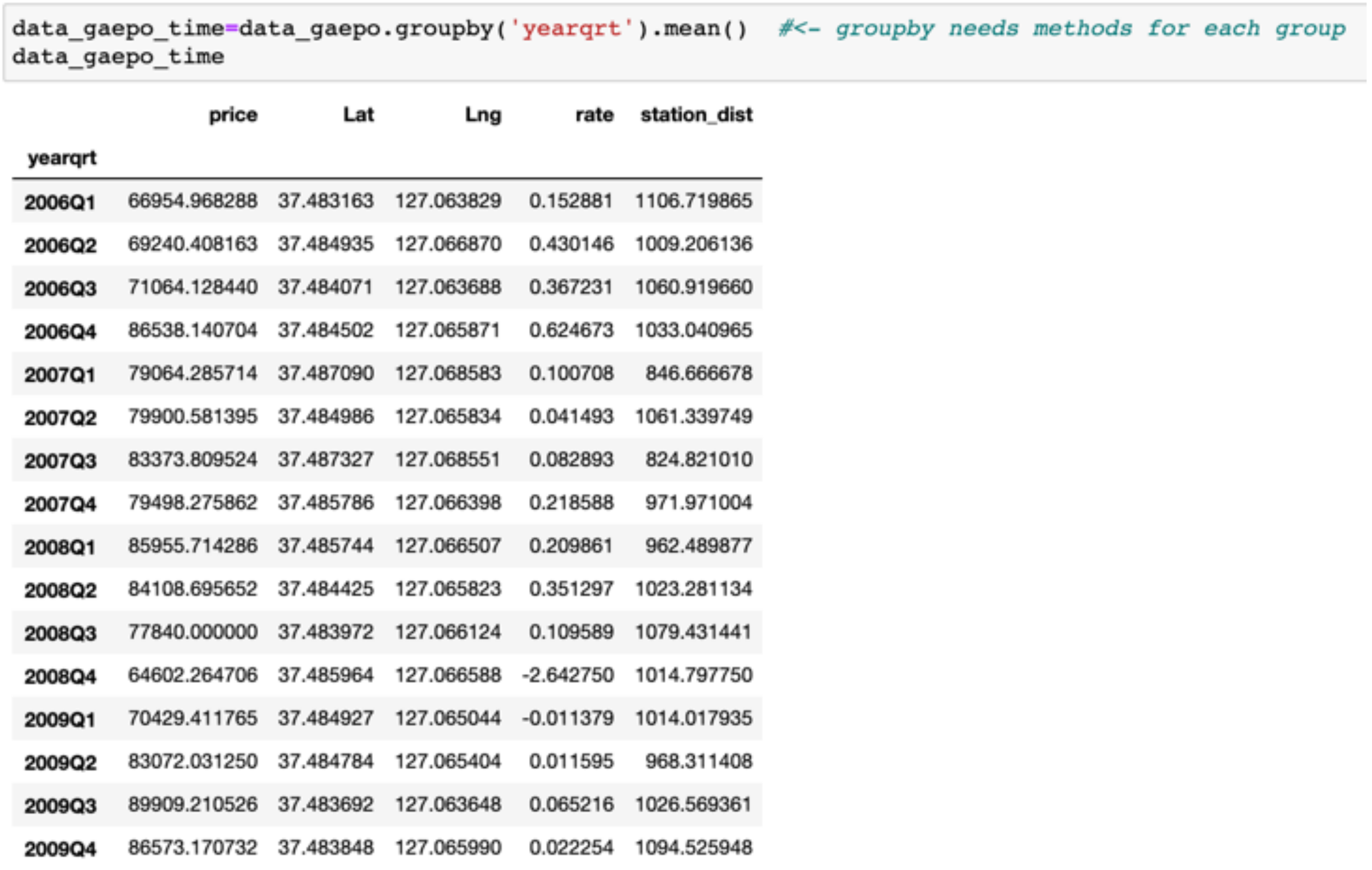
```
In [54]: data_gaepo=data2[data2['dong']=='개포동']
```

```
In [55]: data_gaepo.head()
Out[55]:
```

	yearqrt	price	dong	Lat	Lng	rate	station_dist
2	2006Q1	73000	개포동	37.478407	127.061375	0.152881	1486.178329
3	2006Q1	70000	개포동	37.484609	127.067275	0.152881	1160.598717
4	2006Q1	40000	개포동	37.482445	127.051278	0.152881	650.325555
5	2006Q1	56000	개포동	37.478407	127.061375	0.152881	1486.178329
6	2006Q1	40500	개포동	37.494581	127.075275	0.152881	316.902800

Data Handling

- See how '개포동' price changes over time



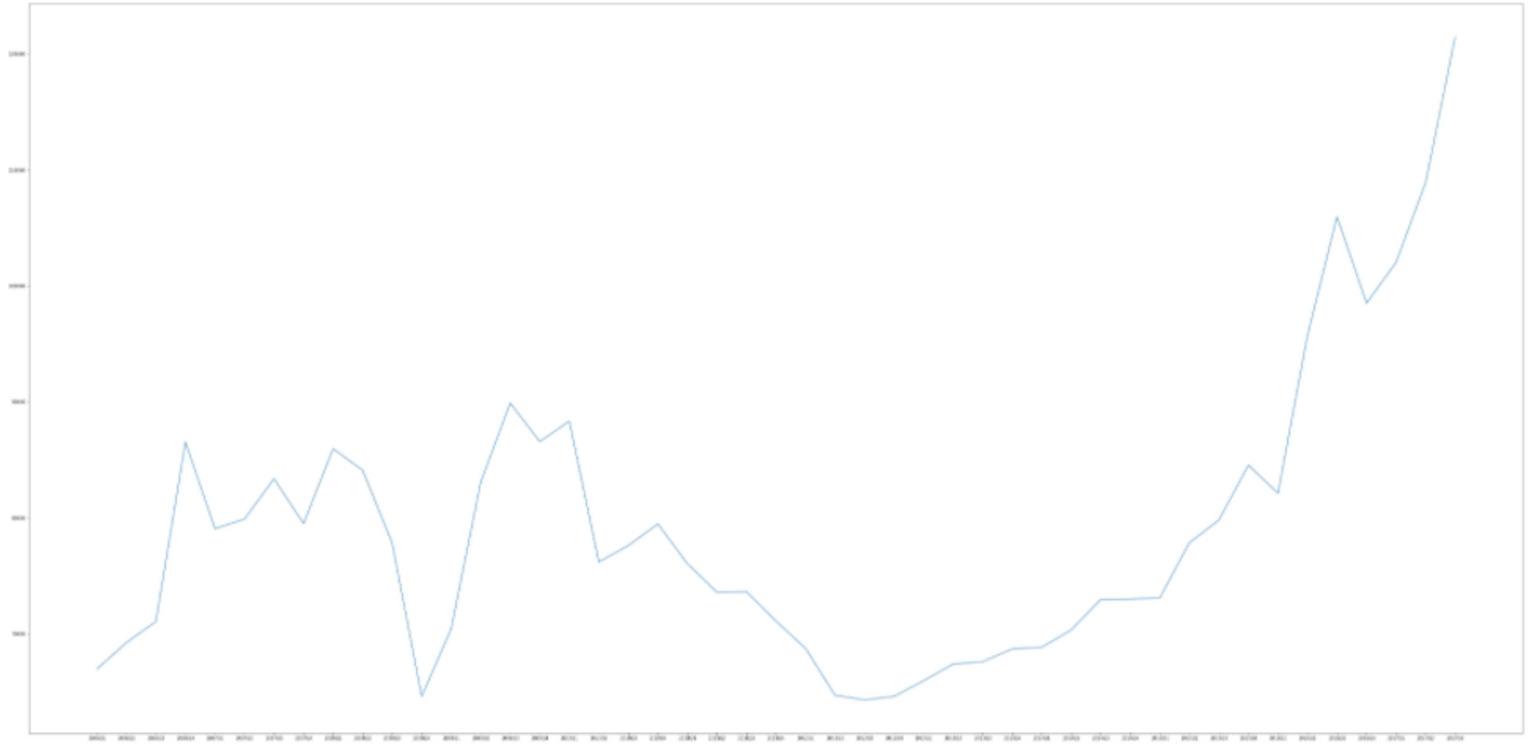
```
data_gaepo_time=data_gaepo.groupby('yearqrt').mean()  #<- groupby needs methods for each group
data_gaepo_time
```

yearqrt	price	Lat	Lng	rate	station_dist
2006Q1	66954.968288	37.483163	127.063829	0.152881	1106.719865
2006Q2	69240.408163	37.484935	127.066870	0.430146	1009.206136
2006Q3	71064.128440	37.484071	127.063688	0.367231	1060.919660
2006Q4	86538.140704	37.484502	127.065871	0.624673	1033.040965
2007Q1	79064.285714	37.487090	127.068583	0.100708	846.666678
2007Q2	79900.581395	37.484986	127.065834	0.041493	1061.339749
2007Q3	83373.809524	37.487327	127.068551	0.082893	824.821010
2007Q4	79498.275862	37.485786	127.066398	0.218588	971.971004
2008Q1	85955.714286	37.485744	127.066507	0.209861	962.489877
2008Q2	84108.695652	37.484425	127.065823	0.351297	1023.281134
2008Q3	77840.000000	37.483972	127.066124	0.109589	1079.431441
2008Q4	64602.264706	37.485964	127.066588	-2.642750	1014.797750
2009Q1	70429.411765	37.484927	127.065044	-0.011379	1014.017935
2009Q2	83072.031250	37.484784	127.065404	0.011595	968.311408
2009Q3	89909.210526	37.483692	127.063648	0.065216	1026.569361
2009Q4	86573.170732	37.483848	127.065990	0.022254	1094.525948

Data Handling

- See how Gaepo-dong price changes over time

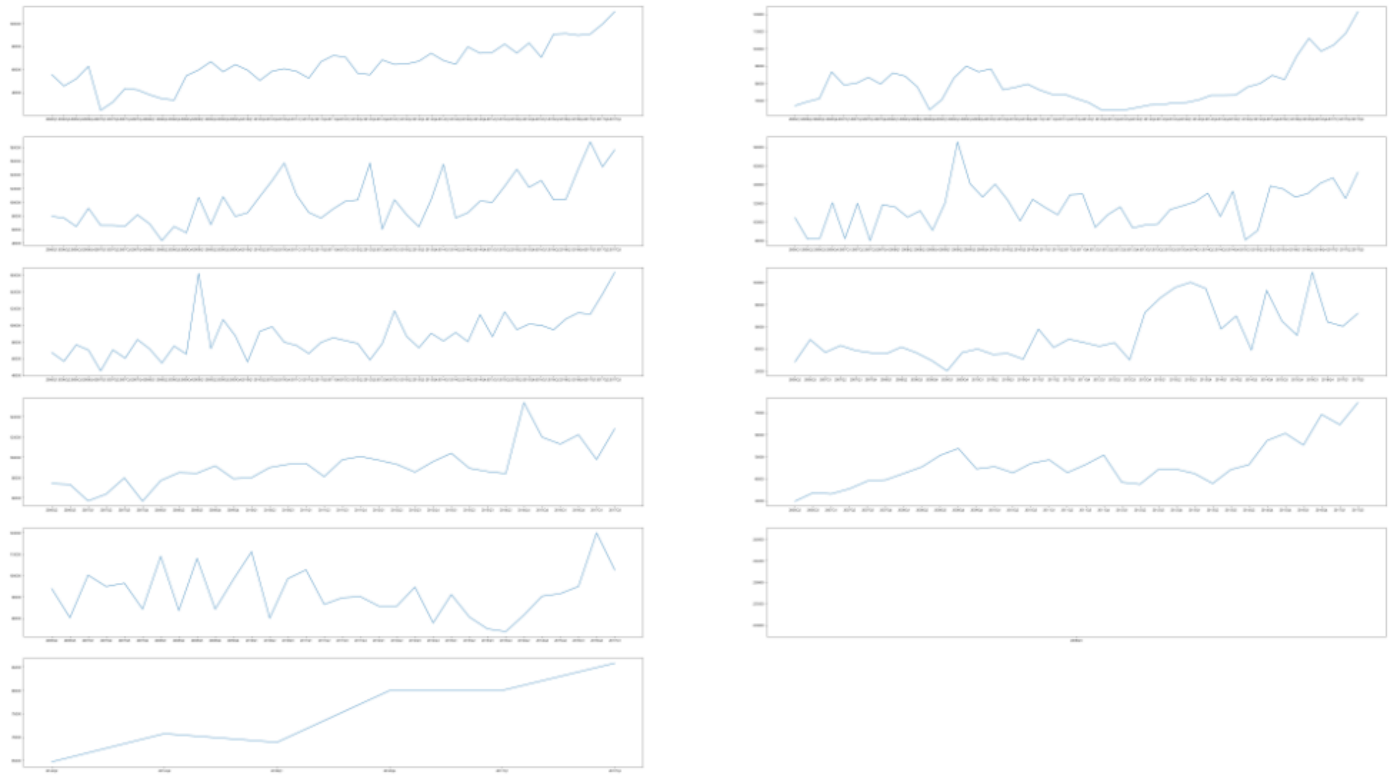
```
In [29]: plt.figure(figsize=(60,30))
         plt.plot(data_gaepo_time['price'])

Out[29]: [<matplotlib.lines.Line2D at 0x22932064340>]
```



Exercise(3) –Each dong's average price change over time

In [73]:

- Get unique values of dongs from the dataset.
- Iterate by each dong.
- Separate the data by dongs.
- Groupby 'yyyyqrt(거래년도 분기별)'.
- Plot each dong's graph by subplot (there will be 11 subplots)



Basic Decomposition

- Moving Average
  - The first step in a classical decomposition is to use a moving average method to estimate the trend-cycle.
    - remove noise and better expose the signal of the underlying causal processes.
    - a simple and common type of smoothing used in time series analysis and time series forecasting.

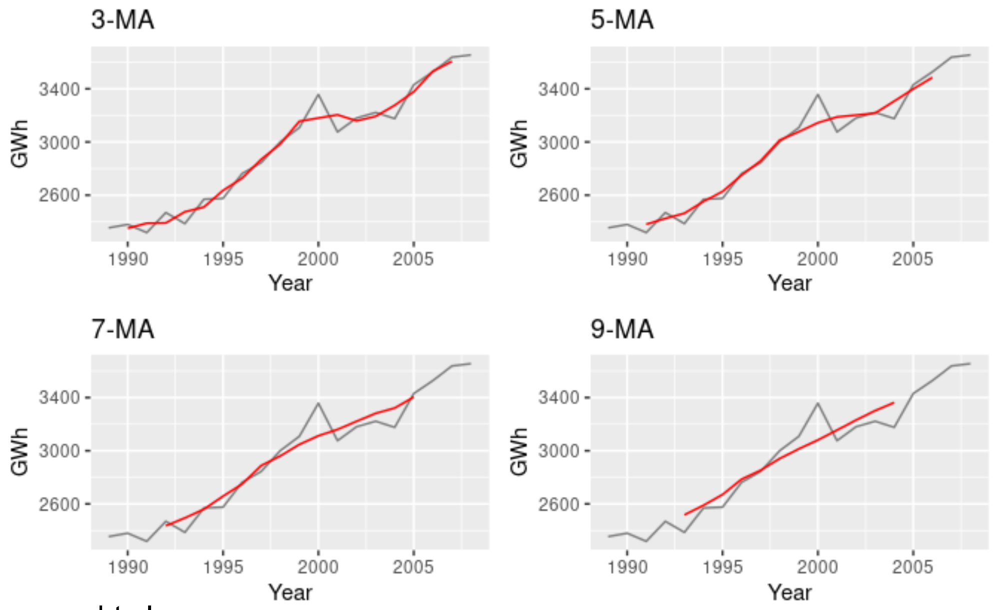
A moving average of order m can be written as

$$\hat{T}_t = \frac{1}{m} \sum_{j=-k}^{k} y_{t+j}, \qquad (6.1)$$

  - The average eliminates some of the randomness in the data, leaving a smooth trend-cycle component. We call this an m-MA, meaning a moving average of order m .

Basic Decomposition

- Moving Average



https://otexts.com/fpp2/moving-averages.html

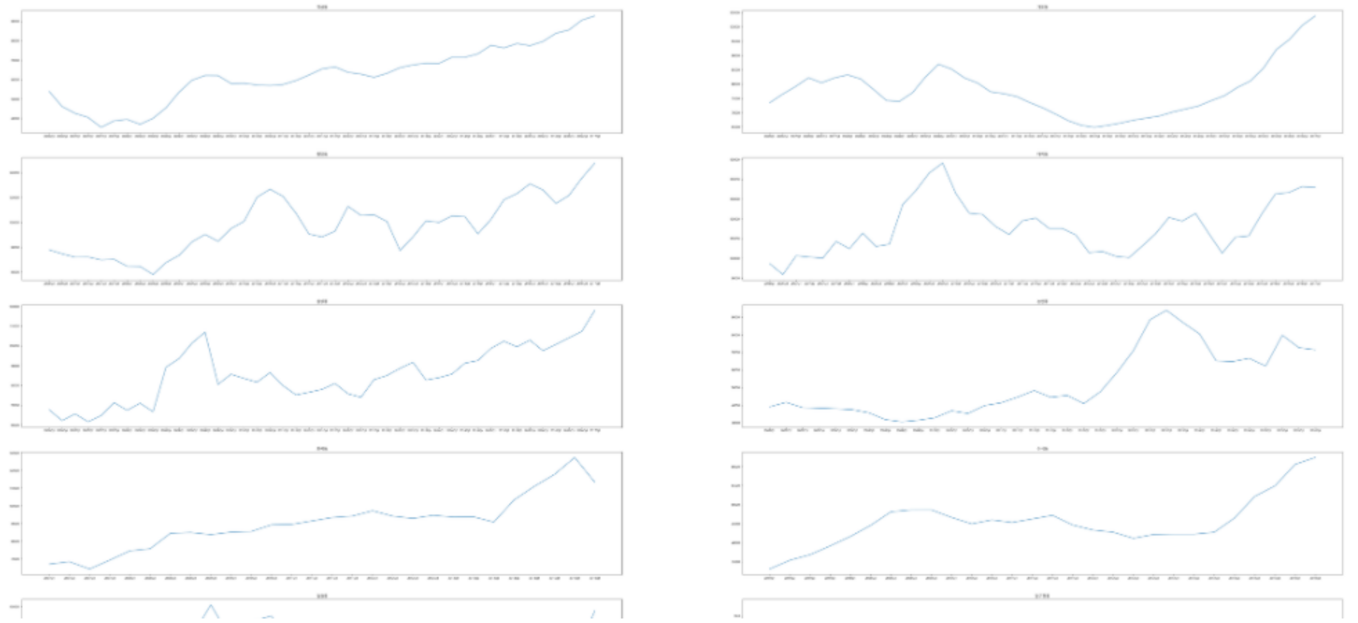
- Notice that the trend-cycle (in red) is smoother than the original data and captures the main movement of the time series without all of the minor fluctuations.
- The order of the moving average determines the smoothness of the trend-cycle estimate.
- Simple moving averages such as these are usually of an odd order (e.g., 3, 5, 7, etc.).

Exercise(4) –Simple Smoothing using 5-MA

In [74]:

- Bring codes from Exercise 3.
- Average every 5 price values
- Plot each dong's graph by subplot (there will be 11 subplots)

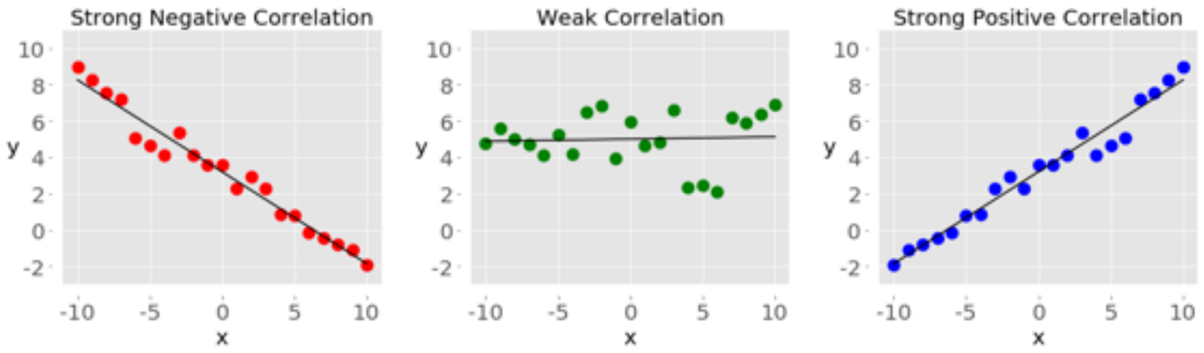
(if you don't want empty graphs, don't plot them)



```
년도 분기별)').mean()
```

Data Analysis

- Correlation coefficient: Quantifies the association between variables or features of a dataset.



https://realpython.com/numpy-scipy-pandas-correlation-python/

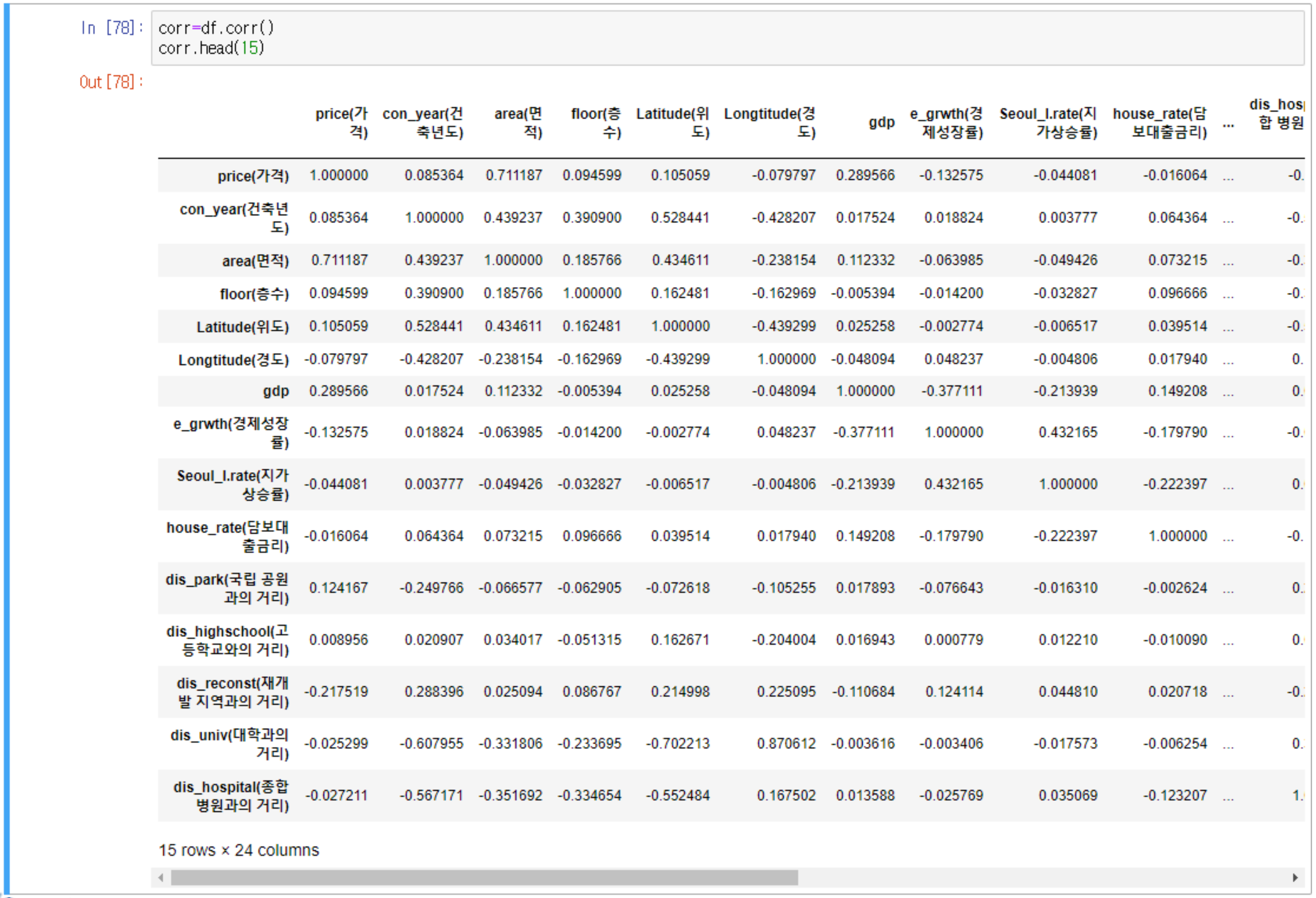
  - Negative correlation (red dots): In the plot on the left, the y values tend to decrease as the x values increase. This shows strong negative correlation, which occurs when large values of one feature correspond to small values of the other, and vice versa.

  - Weak or no correlation (green dots): The plot in the middle shows no obvious trend. This is a form of weak correlation, which occurs when an association between two features is not obvious or is hardly observable.

  - Positive correlation (blue dots): In the plot on the right, the y values tend to increase as the x values increase. This illustrates strong positive correlation, which occurs when large values of one feature correspond to large values of the other, and vice versa.

Data Analysis

- Let’s find if there is any correlations between variables



```
In [78]: corr=df.corr()
         corr.head(15)
```

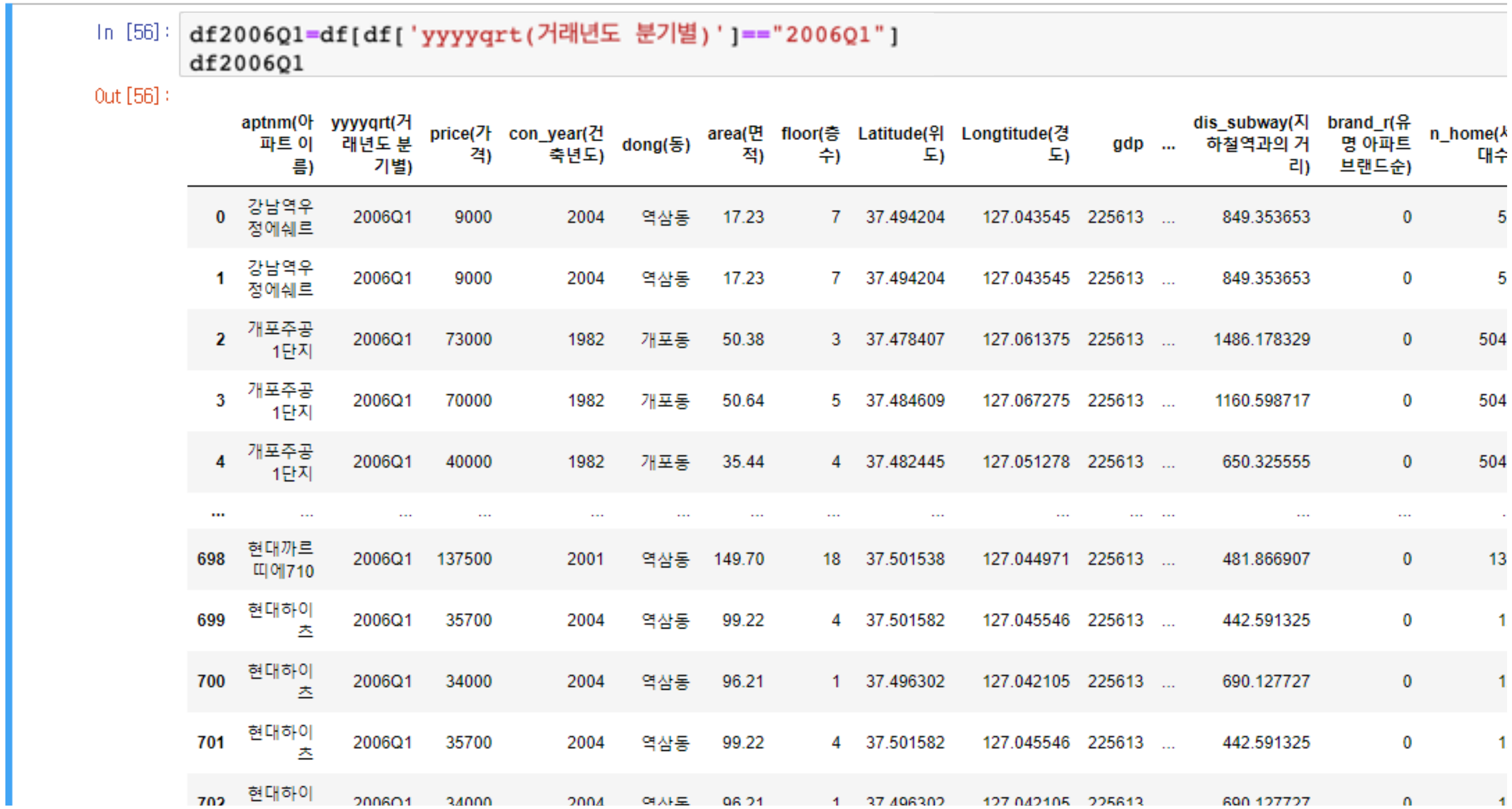
Out [78]:

	price(가격)	con_year(건축년도)	area(면적)	floor(층수)	Latitude(위도)	Longtitude(경도)	gdp	e_grwth(경제성장률)	Seoul_l.rate(지가상승률)	house_rate(담보대출금리)	...	dis_hos 합 병원
price(가격)	1.000000	0.085364	0.711187	0.094599	0.105059	-0.079797	0.289566	-0.132575	-0.044081	-0.016064	...	-0.
con_year(건축년도)	0.085364	1.000000	0.439237	0.390900	0.528441	-0.428207	0.017524	0.018824	0.003777	0.064364	...	-0.
area(면적)	0.711187	0.439237	1.000000	0.185766	0.434611	-0.238154	0.112332	-0.063985	-0.049426	0.073215	...	-0.
floor(층수)	0.094599	0.390900	0.185766	1.000000	0.162481	-0.162969	-0.005394	-0.014200	-0.032827	0.096666	...	-0.
Latitude(위도)	0.105059	0.528441	0.434611	0.162481	1.000000	-0.439299	0.025258	-0.002774	-0.006517	0.039514	...	-0.
Longtitude(경도)	-0.079797	-0.428207	-0.238154	-0.162969	-0.439299	1.000000	-0.048094	0.048237	-0.004806	0.017940	...	0.
gdp	0.289566	0.017524	0.112332	-0.005394	0.025258	-0.048094	1.000000	-0.377111	-0.213939	0.149208	...	0.
e_grwth(경제성장률)	-0.132575	0.018824	-0.063985	-0.014200	-0.002774	0.048237	-0.377111	1.000000	0.432165	-0.179790	...	-0.
Seoul_l.rate(지가상승률)	-0.044081	0.003777	-0.049426	-0.032827	-0.006517	-0.004806	-0.213939	0.432165	1.000000	-0.222397	...	0.
house_rate(담보대출금리)	-0.016064	0.064364	0.073215	0.096666	0.039514	0.017940	0.149208	-0.179790	-0.222397	1.000000	...	-0.
dis_park(국립 공원과의 거리)	0.124167	-0.249766	-0.066577	-0.062905	-0.072618	-0.105255	0.017893	-0.076643	-0.016310	-0.002624	...	0.
dis_highschool(고등학교와의 거리)	0.008956	0.020907	0.034017	-0.051315	0.162671	-0.204004	0.016943	0.000779	0.012210	-0.010090	...	0.
dis_reconst(재개발 지역과의 거리)	-0.217519	0.288396	0.025094	0.086767	0.214998	0.225095	-0.110684	0.124114	0.044810	0.020718	...	-0.
dis_univ(대학과의 거리)	-0.025299	-0.607955	-0.331806	-0.233695	-0.702213	0.870612	-0.003616	-0.003406	-0.017573	-0.006254	...	0.
dis_hospital(종합병원과의 거리)	-0.027211	-0.567171	-0.351692	-0.334654	-0.552484	0.167502	0.013588	-0.025769	0.035069	-0.123207	...	1.

15 rows × 24 columns

Exercise(5) –Correlations

- Let’s find if there is any correlations between variables
  - We can think that “Station Influence Area” can influence the price.
    - 1. Separates only “2006Q1”



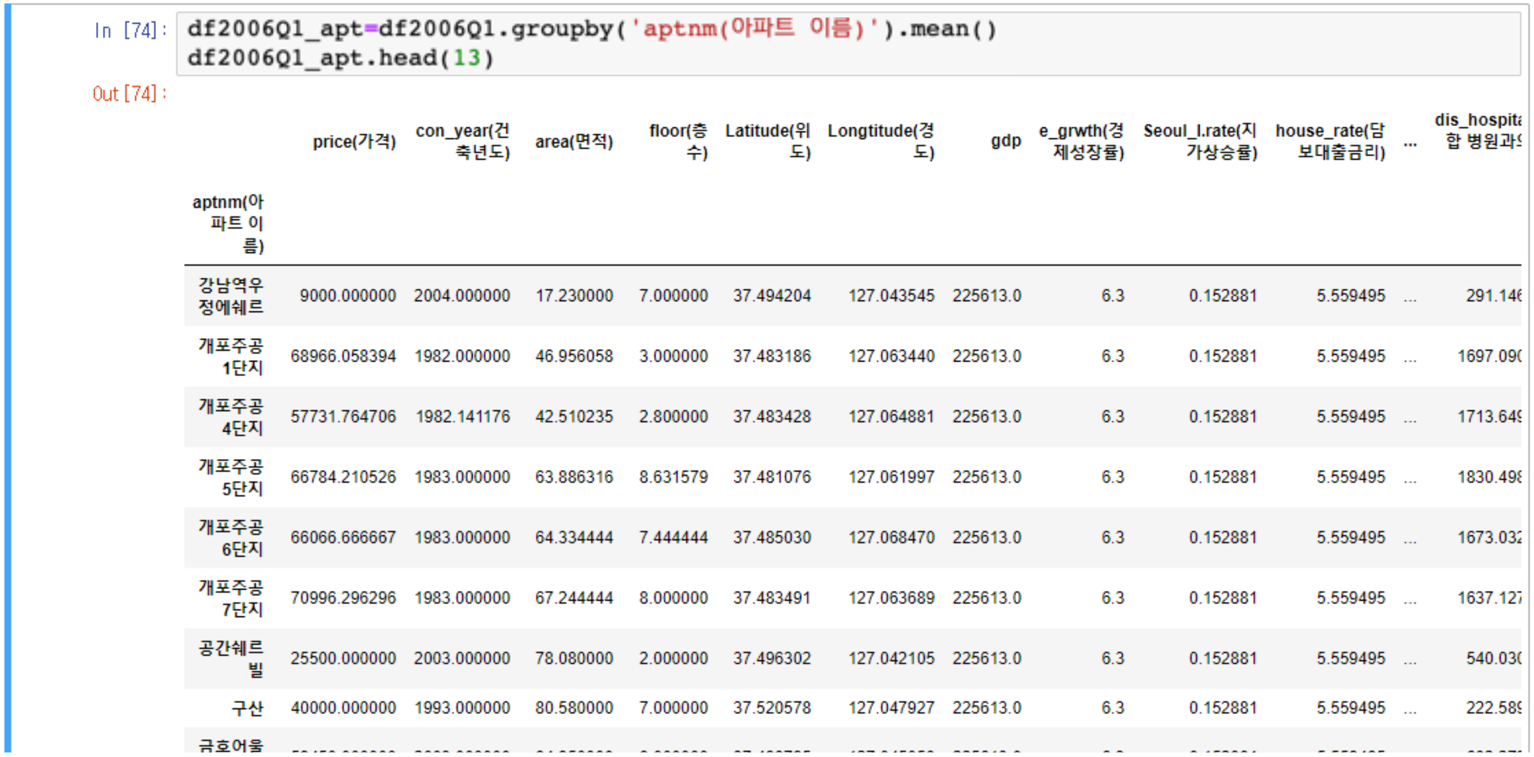
In [56]:
```
df2006Q1=df[df['yyyyqrt(거래년도 분기별)']=="2006Q1"]
df2006Q1
```

Out [56]:

	aptnm(아파트 이름)	yyyyqrt(거래년도 분기별)	price(가격)	con_year(건축년도)	dong(동)	area(면적)	floor(층수)	Latitude(위도)	Longtitude(경도)	gdp	...	dis_subway(지하철역과의 거리)	brand_r(유명 아파트 브랜드순)	n_home(대수
0	강남역우정에쉐르	2006Q1	9000	2004	역삼동	17.23	7	37.494204	127.043545	225613	...	849.353653	0	5
1	강남역우정에쉐르	2006Q1	9000	2004	역삼동	17.23	7	37.494204	127.043545	225613	...	849.353653	0	5
2	개포주공1단지	2006Q1	73000	1982	개포동	50.38	3	37.478407	127.061375	225613	...	1486.178329	0	504
3	개포주공1단지	2006Q1	70000	1982	개포동	50.64	5	37.484609	127.067275	225613	...	1160.598717	0	504
4	개포주공1단지	2006Q1	40000	1982	개포동	35.44	4	37.482445	127.051278	225613	...	650.325555	0	504
...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
698	현대까르띠에710	2006Q1	137500	2001	역삼동	149.70	18	37.501538	127.044971	225613	...	481.866907	0	13
699	현대하이츠	2006Q1	35700	2004	역삼동	99.22	4	37.501582	127.045546	225613	...	442.591325	0	1
700	현대하이츠	2006Q1	34000	2004	역삼동	96.21	1	37.496302	127.042105	225613	...	690.127727	0	1
701	현대하이츠	2006Q1	35700	2004	역삼동	99.22	4	37.501582	127.045546	225613	...	442.591325	0	1
702	현대하이	2006Q1	34000	2004	역삼동	96.21	1	37.496302	127.042105	225613		690.127727	0	1

Exercise(5) –Correlations

- Let’s find if there is any correlations between variables
  - We can think that “Station Influence Area” can influence the price.
    - 2. groupby “artnm(아파트 이름)”



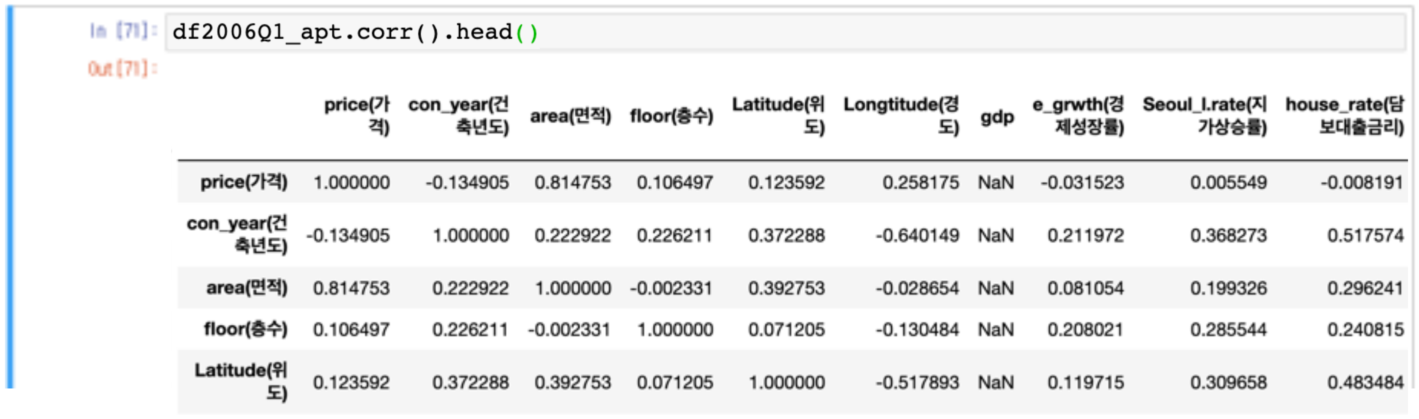
```
In [74]: df2006Q1_apt=df2006Q1.groupby('aptnm(아파트 이름)').mean()
         df2006Q1_apt.head(13)
```

Out [74]:

aptnm(아파트 이름)	price(가격)	con_year(건축년도)	area(면적)	floor(층수)	Latitude(위도)	Longtitude(경도)	gdp	e_grwth(경제성장률)	Seoul_l.rate(지가상승률)	house_rate(담보대출금리)	...	dis_hospit 합 병원과
강남역우정에쉐르	9000.000000	2004.000000	17.230000	7.000000	37.494204	127.043545	225613.0	6.3	0.152881	5.559495	...	291.14
개포주공1단지	68966.058394	1982.000000	46.956058	3.000000	37.483186	127.063440	225613.0	6.3	0.152881	5.559495	...	1697.09
개포주공4단지	57731.764706	1982.141176	42.510235	2.800000	37.483428	127.064881	225613.0	6.3	0.152881	5.559495	...	1713.64
개포주공5단지	66784.210526	1983.000000	63.886316	8.631579	37.481076	127.061997	225613.0	6.3	0.152881	5.559495	...	1830.49
개포주공6단지	66066.666667	1983.000000	64.334444	7.444444	37.485030	127.068470	225613.0	6.3	0.152881	5.559495	...	1673.03
개포주공7단지	70996.296296	1983.000000	67.244444	8.000000	37.483491	127.063689	225613.0	6.3	0.152881	5.559495	...	1637.12
공간쉐르빌	25500.000000	2003.000000	78.080000	2.000000	37.496302	127.042105	225613.0	6.3	0.152881	5.559495	...	540.03
구산	40000.000000	1993.000000	80.580000	7.000000	37.520578	127.047927	225613.0	6.3	0.152881	5.559495	...	222.58
금호어울	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	...	[illegible]

Data Analysis

- Let’s find if there is any correlations between variables
  - We can think that “Station Influence Area” can influence the price.



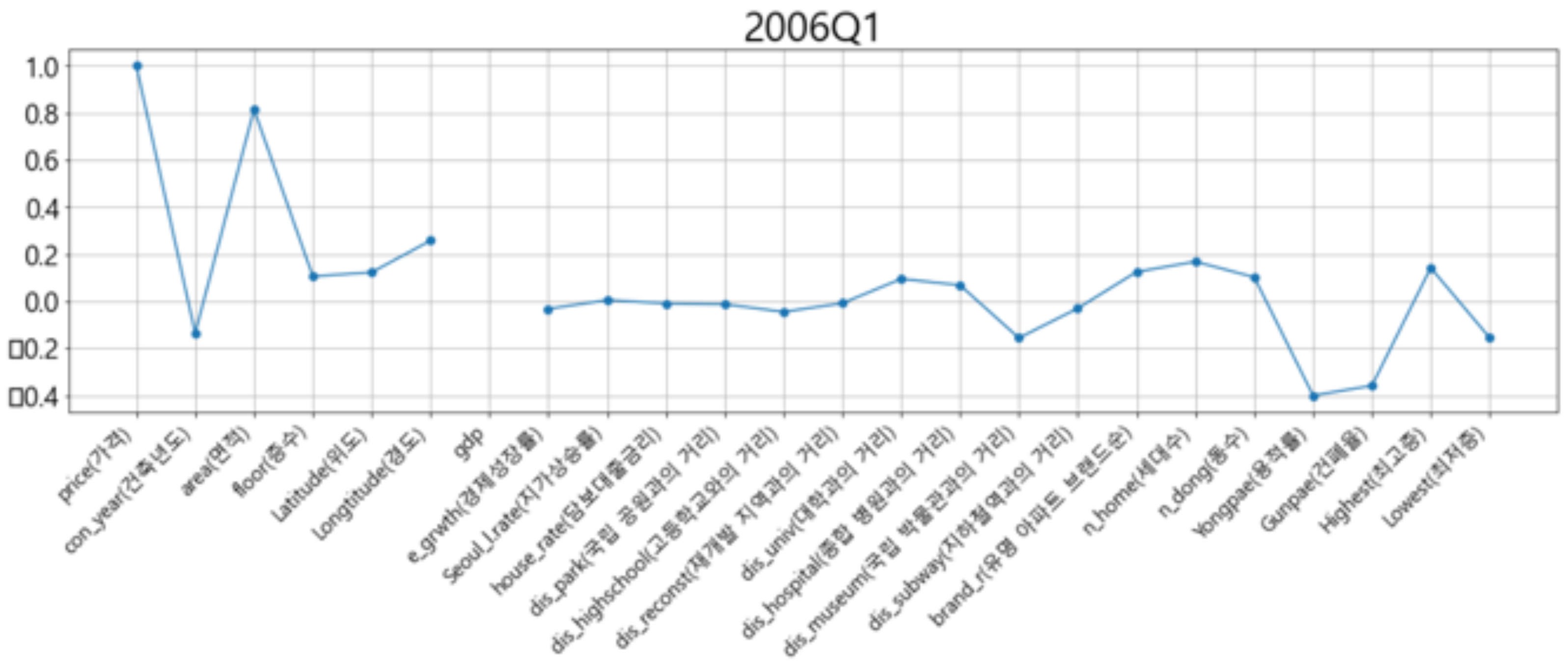
```
In [71]: df2006Q1_apt.corr().head()
Out[71]:
```

	price(가격)	con_year(건축년도)	area(면적)	floor(층수)	Latitude(위도)	Longtitude(경도)	gdp	e_grwth(경제성장률)	Seoul_l.rate(지가상승률)	house_rate(담보대출금리)
price(가격)	1.000000	-0.134905	0.814753	0.106497	0.123592	0.258175	NaN	-0.031523	0.005549	-0.008191
con_year(건축년도)	-0.134905	1.000000	0.222922	0.226211	0.372288	-0.640149	NaN	0.211972	0.368273	0.517574
area(면적)	0.814753	0.222922	1.000000	-0.002331	0.392753	-0.028654	NaN	0.081054	0.199326	0.296241
floor(층수)	0.106497	0.226211	-0.002331	1.000000	0.071205	-0.130484	NaN	0.208021	0.285544	0.240815
Latitude(위도)	0.123592	0.372288	0.392753	0.071205	1.000000	-0.517893	NaN	0.119715	0.309658	0.483484

We can see that in Gangnam, “distance to subway” doesn’t matter (corr = -0.029509)

Exercise(5) –Correlations

```
df2006Q1=df[df['yyyyqrt(거래년도 분기별)']=="2006Q1"] # extract data in 2006Q1
df2006Q1_apt=df2006Q1.groupby('aptnm(아파트 이름)').mean() #average price
r = df2006Q1_apt.corr()
r_price = r['price(가격)']

plt.figure(figsize=(20,5))
plt.plot(r_price,'o-')
plt.yticks(fontsize = 20); plt.xticks(rotation=45, fontsize = 15, ha = 'right')
plt.title('2006Q1',fontsize=30);
plt.grid(True)
plt.show()
```



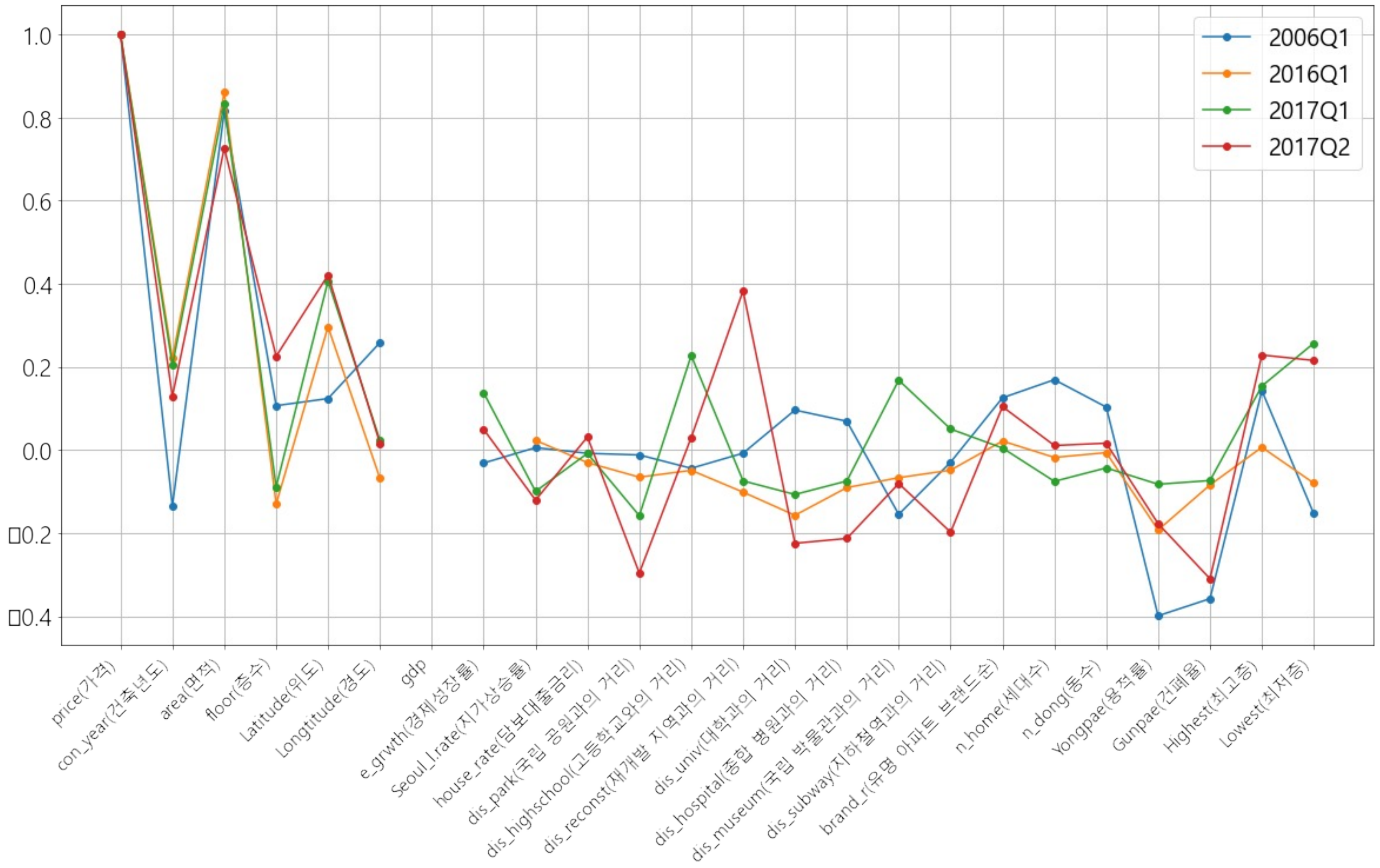
Exercise(5) –Correlations

- What about the results in 2016Q1, 2017Q1 and 2017Q2?

```
years = ['2006Q1', '2016Q1', '2017Q1', '2017Q2']

for y in years:
    df_y=df[df['yyyyqrt(거래년도 분기별)']==y] # extract data in 2006Q1
    df_y_apt=df_y.groupby('aptnm(아파트 이름)').mean() #average price
    df_r = df_y_apt.corr()
    df_r_price = df_r['price(가격)']
    if y==years[0]:
        df_r_price_all = df_r_price
    else:
        df_r_price_all = pd.concat([df_r_price_all,df_r_price],axis=1)
df_r_price_all.columns = years


plt.figure(figsize=(20,10))
plt.plot(df_r_price_all,'o-')
plt.yticks(fontsize = 20); plt.xticks(rotation=45, fontsize = 15, ha = 'right')
plt.grid(True)
plt.legend(years,fontsize=20)
plt.show()
```

Exercise(5) –Correlations

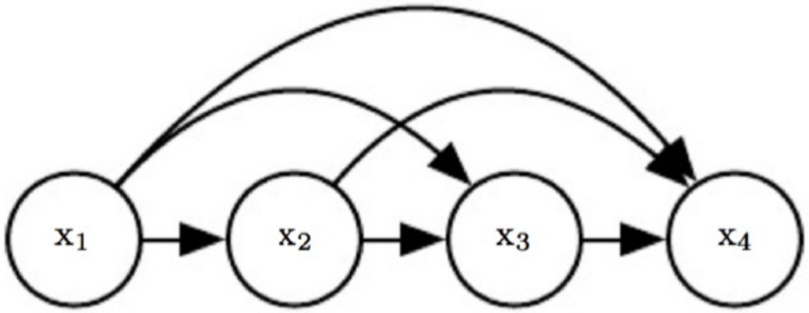
- What about the results in 2016Q1, 2017Q1 and 2017Q2?



AutoRegressive Model

- AR(p) : An autoregressive process of order p
- Present value of the series, X_t, by a function of p past values, $X_{t-1}, X_{t-2}, \dots, X_{t-p}$.

- $X_t = c + \varphi_1 X_{t-1} + \varphi_2 X_{t-2} + \varphi_3 X_{t-3} + \cdots + \varphi_p X_{t-p} + Z_t$
  - *Where $\{Z_t\}$ is white noise, i.e., $\{Z_t\} \sim WN(0, \sigma^2)$, and Z_t, is uncorrelated with X_s for each $s < t$.*



AR(1) model

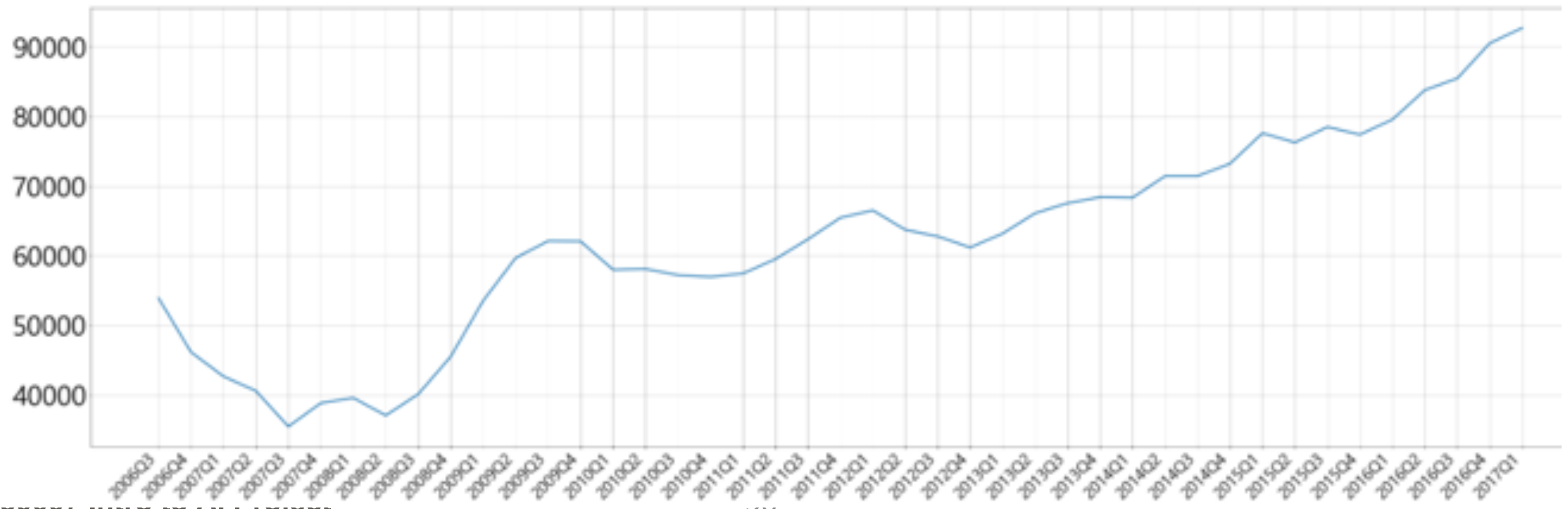
- AR(1) = $X_t = c + \varphi_1 X_{t-1} + Z_t$

- For an AR(1) model:
  - when φ_1 =0, X_t is equivalent to white noise;
  - when φ_1 =1 and c=0, X_t is equivalent to a random walk;
  - when φ_1 =1 and c≠0, X_t is equivalent to a random walk with drift;
  - when φ_1 <0, X_t tends to oscillate around the mean.

Exercise(6) – the price trend in YeockSam-dong

- First, let's check the price in YeockSam-dong

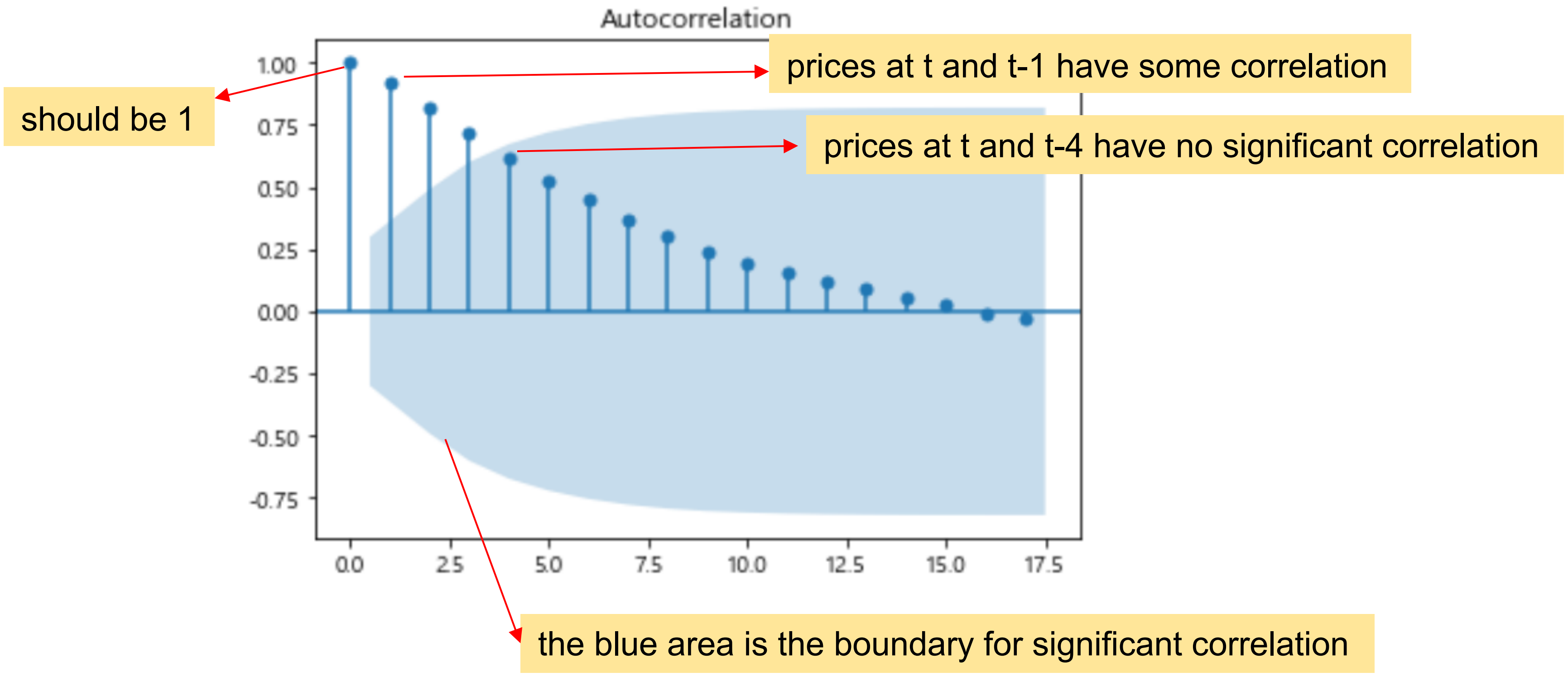
```
nMA = 2 # number of Pre(or post) samples to average

plt.figure(figsize=(50,15))
tmp=df[df['dong(동)']=="역삼동"]
tp=tmp.groupby('yyyyqrt(거래년도 분기별)').mean().copy()
ls=[]
for j in range(nMA,len(tp)-nMA):
    ls.append(tp['price(가격)'].iloc[j-nMA:j+nMA].mean())
df_mean5=pd.DataFrame({"Time" : tp.index.values[nMA:len(tp)-nMA], "Mean_Price" : ls })
plt.plot(df_mean5["Time"],df_mean5["Mean_Price"],linewidth=3.0)
plt.yticks(fontsize = 50); plt.xticks(fontsize = 30, rotation = 45, ha = 'right')
plt.grid(True)
df_mean5.set_index("Time", inplace=True)
```



Exercise(6) – Autocorrelation of the price trend

- Second, check autocorrelation

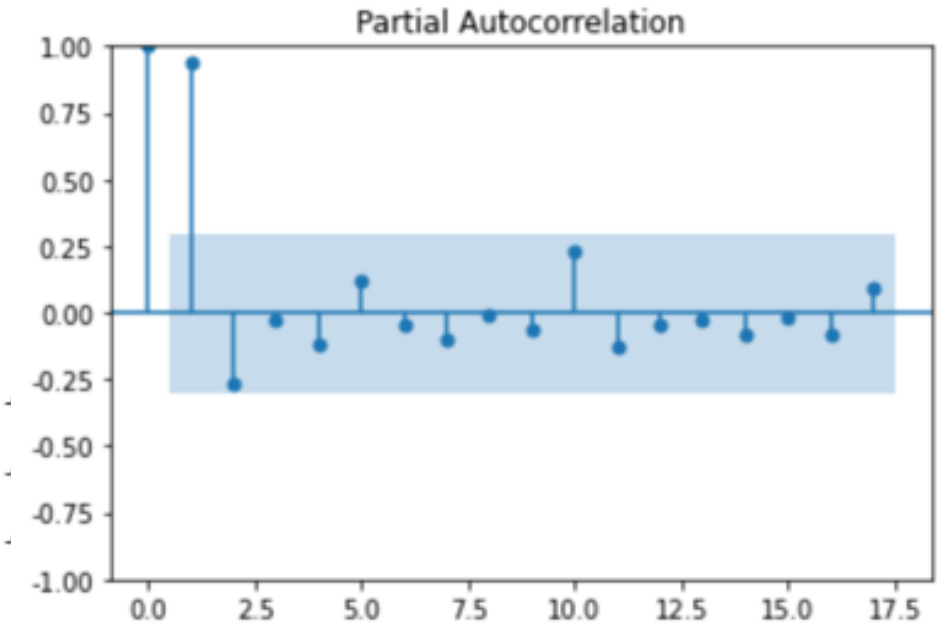
```
plot_acf(df_mean5['Mean_Price'])
plt.show()
```



Exercise(6) – Autocorrelation of the price trend

- Second, check autocorrelation

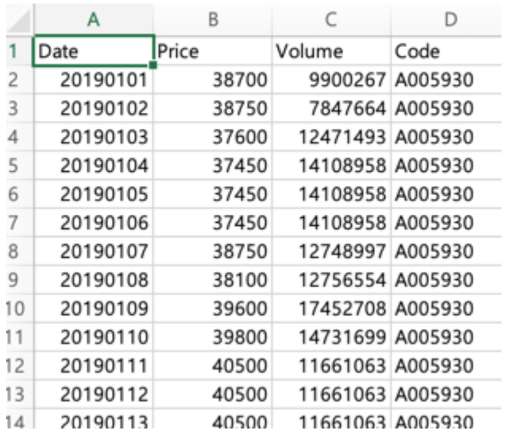
```
plot_pacf(df_mean5['Mean_Price'])
plt.show()
```



acf : Pearson correlation
pacf: partial acf
= direct correlation
= coefficient in AR model

Exercise(7) – Stock price and volume

- Given 3 companies (codes) stock price and volume for 2 years (2019 and 2020) in 'stockprice_3.xlsx', plot 3 figures, each with 2 subplots as follows.
  - one figure for one company
    - subplot1: price curve, price with 3-MA, price with 5-MA
    - subplot2: volume curve, with 3-MA, with 5-MA
  - For all subplots, x-axis should be 'Date'.
  - Submit your code and your docx file including the 3 figures



	A	B	C	D
1	Date	Price	Volume	Code
2	20190101	38700	9900267	A005930
3	20190102	38750	7847664	A005930
4	20190103	37600	12471493	A005930
5	20190104	37450	14108958	A005930
6	20190105	37450	14108958	A005930
7	20190106	37450	14108958	A005930
8	20190107	38750	12748997	A005930
9	20190108	38100	12756554	A005930
10	20190109	39600	17452708	A005930
11	20190110	39800	14731699	A005930
12	20190111	40500	11661063	A005930
13	20190112	40500	11661063	A005930
14	20190113	40500	11661063	A005930

References

- https://books.google.co.kr/books?hl=en&lr=&id=IlupDwAAQBAJ&oi=fnd&pg=PP1&dq=time+series&ots=wfdZ27Rn6c&sig=pWqrVvG8mndFUuvkfGZt4xNdZJs&redir_esc=y#v=onepage&q=time%20series&f=false
- Adhikari, R., & Agrawal, R. K. (2013). An introductory study on time series modeling and forecasting. *arXiv preprint arXiv:1302.6613*.
- Mingda, Z. (2018). Time Series: Auto regressive models AR MA ARMA ARIMA. *University of Pittsburgh*.
- https://books.google.co.kr/books?hl=en&lr=&id=o7jWV67165QC&oi=fnd&pg=PR5&dq=time+series&ots=mqfhWSFc0r&sig=9TjaCtQ-mBVuJ8xDtNIR5nq2Iko&redir_esc=y#v=onepage&q=time%20series&f=false
- https://realpython.com/numpy-scipy-pandas-correlation-python/
- https://otexts.com/fpp2/index.html
- https://www.analyticsvidhya.com/blog/2018/09/non-stationary-time-series-python/
- https://people.duke.edu/~rnau/411diff.htm
- https://www.itl.nist.gov/div898/handbook/pmc/section4/pmc442.htm