Assignment 1 - Probability, Linear Algebra, Programming, and Git

Zhenxing Xie

Netid: **zx94**

Instructions for all assignments can be found here (https://github.com/kylebradbury/ids705/blob/master/assignments/_Assignment%20Instructions.ipynb), which is also linked to from the course syllabus (https://kylebradbury.github.io/ids705/index.html).

Probability and Statistics Theory

Note: for all assignments, write out all equations and math using markdown and LaTeX (https://tobi.oetiker.ch/lshort/lshort.pdf). For this section of the assignment (Probability and Statistics Theory) show and type up ALL math work

1

[3 points]

Let $f(x) = \begin{cases} 0 & x < 0 \\ \alpha x^2 & 0 \leq x \leq 2 \\ 0 & 2 < x \end{cases}$

For what value of α is $f(x)$ a valid probability density function?

ANSWER

In essence, the function $f(x)$ should satisfy $\int_{-\infty}^{+\infty} f(x)dx = 1$, thus,

$$\int_{-\infty}^{+\infty} f(x)dx = \int_{-\infty}^{0} 0dx + \int_{0}^{2} \alpha x^2 dx + \int_{2}^{+\infty} 0dx = 1$$

therefore,

$$\int_{0}^{2} \alpha x^2 dx = 1$$

so we have,

$$\frac{8}{3}\alpha = 1$$

the value of α is:

$$\alpha = \frac{3}{8}$$

2

[3 points] What is the cumulative distribution function (CDF) that corresponds to the following probability distribution function? Please state the value of the CDF for all possible values of x.

$$f(x) = \begin{cases} \frac{1}{3} & 0 < x < 3 \\ 0 & \text{otherwise} \end{cases}$$

ANSWER

Set the CDF as $F(x)$, then,

when $x \leq 0$,

$$F(x) = \int_{-\infty}^{x} 0dx = 0$$

and then when $0 < x < 3$

$$F(x) = \int_{-\infty}^{x} 0dx + \int_{0}^{x} \frac{1}{3}dx = 0 + \frac{x}{3} = \frac{x}{3}$$

when $x \geq 3$,

$$F(x) = \int_{-\infty}^{x} 0dx + \int_{0}^{x} \frac{1}{3}dx + \int_{x}^{+\infty} 0dx = 0 + 3 * \frac{1}{3} + 0 = 1$$

so we have,

$$F(x) = \begin{cases} 0 & x \leq 0 \\ \frac{x}{3} & 0 < x < 3 \\ 1 & x \geq 3 \end{cases}$$

3

[6 points] For the probability distribution function for the random variable X,

$$f(x) = \begin{cases} \frac{1}{3} & 0 < x < 3 \\ 0 & \text{otherwise} \end{cases}$$

what is the (a) expected value and (b) variance of X. *Show all work*.

ANSWER

(a) The formula of Expected Value $E(X)$ is as following:

$$E(X) = \int_{-\infty}^{+\infty} xf(x)dx$$

So plug in the aforementioned $f(x)$ we get:

$$E(X) = \int_{0}^{3} \frac{x}{3}dx = \frac{3}{2}$$

(b) One formula for calculating Variance $Var(X)$ is as following:

$$Var(X) = E(X^2) - (E(X))^2$$

We can evaluate $E(X^2)$ and $E(X)^2$ by utilizing the $E(X)$ formula above:

$$(E(X))^2 = (\frac{3}{2})^2 = \frac{9}{4}$$

$$E(X^2) = \int_{-\infty}^{+\infty} x^2 f(x)dx = \int_{0}^{3} \frac{1}{3}x^2 dx = 3$$

Therefore, the $Var(X)$ is as following:

$$Var(X) = E(X^2) - (E(X))^2 = 3 - \frac{9}{4} = \frac{3}{4}$$

4

[6 points] Consider the following table of data that provides the values of a discrete data vector $\mathbf{x}$ of samples from the random variable X, where each entry in $\mathbf{x}$ is given as x_i.

Table 1. Dataset N=5 observations

	x_0	x_1	x_2	x_3	x_4
$\mathbf{x}$	2	3	10	-1	-1

What is the (a) mean, (b) variance, and the of the data?

Show all work. Your answer should include the definition of mean, median, and variance in the context of discrete data.

ANSWER

(a) Mean

$$E(X) = \sum_{i=1}^{n} \frac{x_i}{n} = \frac{1}{5}(2 + 3 + 10 - 1 - 1) = \frac{13}{5}$$

(b) Variance

$$(E(X))^2 = (\frac{13}{5})^2 = \frac{169}{25}$$

$$E(X^2) = \sum_{i=1}^{n} \frac{x_i^2}{n} = \frac{1}{5} \cdot 2^2 + \frac{1}{5} \cdot 3^2 + \frac{1}{5} \cdot 10^2 + \frac{1}{5} \cdot (-1)^2 + \frac{1}{5} \cdot (-1)^2 = 23$$

$$Var(X) = \frac{1}{n} \sum_{i=1}^{n} (x_i - E(X))^2 = E(X^2) - (E(X))^2 = 23 - \frac{169}{25} = \frac{406}{25} = 16.24$$

(c) Median

The median is the middle value that divides the higher half from the lower half of the dataset. In this context, the median is 2, since the value is in the middle of the ordered dataset {-1, -1, 2, 3, 10}.

5

[8 points] Review of counting from probability theory.

(a) How many different 7-place license plates are possible if the first 3 places only contain letters and the last 4 only contain numbers?

(b) How many different batting orders are possible for a baseball team with 9 players?

(c) How many batting orders of 5 players are possible for a team with 9 players total?

(d) Let's assume this class has 26 students and we want to form project teams. How many unique teams of 3 are possible?

Hint: For each problem, determine if order matters, and if it should be calculated with or without replacement.

ANSWER

(a) There are 176760000 possibilities for the 7-place license plates.

There are $26 \times 26 \times 26 \times 10 \times 10 \times 10 \times 10 = 175760000$ possibilities, since we randomly choose 3 letters out of 26 letters with replacement and the order of the letters matter. Furthermore, 4 numbers should be randomly chosen from 10 numbers (0-9) with replacement, and the order also matters. Finally, we multiply both the possibilities of the letters and the possibilities of the numbers together.

(b)There are 362880 possibilities for batting orders for a baseball team with 9 players.

Order matters. This is a permutation without replacement for 9 players. Therefore there are $9! = 362880$ different possibilities.

(c) There are 15120 possibilities for batting orders of 5 players for a team with 9 players.

Order matters. This is the permutation of 5 players from 9 players without replacement. Therefore, there are $9 \times 8 \times 7 \times 6 \times 5 = 15120$ different possibilities.

(d) There are 2600 possibilities for unique teams of 3.

Order does not matter. This is a combination problem without replacement, thus there are $\binom{26}{3} = 2600$ possibilities for unique teams of 3.

Linear Algebra

6

[7 points] Matrix manipulations and multiplication. Machine learning involves working with many matrices, so this exercise will provide you with the opportunity to practice those skills.

Let $\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{bmatrix}$, $\mathbf{b} = \begin{bmatrix} -1 \\ 3 \\ 8 \end{bmatrix}$, $\mathbf{c} = \begin{bmatrix} 4 \\ -3 \\ 6 \end{bmatrix}$, and $\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Compute the following or indicate that it cannot be computed:

1. $\mathbf{AA}$
2. $\mathbf{AA}^T$
3. $\mathbf{Ab}$
4. $\mathbf{Ab}^T$
5. $\mathbf{bA}$
6. $\mathbf{b}^T\mathbf{A}$
7. $\mathbf{bb}$
8. $\mathbf{b}^T\mathbf{b}$
9. $\mathbf{bb}^T$
10. $\mathbf{b} + \mathbf{c}^T$
11. $\mathbf{b}^T\mathbf{b}^T$
12. $\mathbf{A}^{-1}\mathbf{b}$
13. $\mathbf{A} \circ \mathbf{A}$
14. $\mathbf{b} \circ \mathbf{c}$

Note: The element-wise (or Hadamard) product is the product of each element in one matrix with the corresponding element in another matrix, and is represented by the symbol "$\circ$".

ANSWER

In [62]:

```
import numpy as np

#creating the arrays
A = np.array([
    [1,2,3],
    [2,4,5],
    [3,5,6]])
b = np.array([[-1],[3],[8]])
c = np.array([[4],[-3],[6]])
I = np.eye(3)
# 1
print("1. A @ A = \n\n",np.matmul(A, A), end = '\n\n\n')

# 2
print("2. A @ A.T = \n\n",np.matmul(A, A.T), end = '\n\n\n')

# 3
print("3. A @ b = \n\n",np.matmul(A, b), end = '\n\n\n')


# 4
try:
    print("4. A @ b.T = \n\n",np.matmul(A, b.T), end = '\n\n\n')
except ValueError:
    print("4.Cannot be calculated.\n\n", ValueError, end = '\n\n\n')

# 5
try:
    print("5. b @ A = \n\n",np.matmul(b, A), end = '\n\n\n')
except ValueError:
    print("5.Cannot be calculated.\n\n", ValueError, end = '\n\n\n')

# 6
print("6. b.T @ A = \n\n",np.matmul(b.T, A), end = '\n\n\n')

# 7
try:
    print("7. b @ b = \n\n",np.matmul(b, b), end = '\n\n\n')
except ValueError as VE:
    print("7.Cannot be calculated.\n\n", ValueError, end = '\n\n\n')

# 8
print("8. b.T @ b = \n\n",np.matmul(b.T, b), end = '\n\n\n')

# 9
print("9. b @ b.T = \n\n",np.matmul(b, b.T), end = '\n\n\n')

# 10
print("10.\n\n The vector with shapes (3,1) and The vector with (1,3) ca
nnot be added. \n")

# 11
try:
    print("11. b.T @ b.T = \n\n",np.matmul(b.T, b.T), end = '\n\n\n')
except ValueError:
    print("11.Cannot be calculated.\n\n", ValueError, end = '\n\n\n')
```

```
# 12
print("12. inv(A) @ b = \n\n", np.matmul(np.linalg.inv(A), b), end = '\n
\n\n')


# 13
print("13. A * A = \n\n", np.multiply(A,A), end = '\n\n\n')

# 14
print("14. b * c = \n\n", np.multiply(b,c), end = '\n\n\n')
```

```
1. A @ A =

 [[14 25 31]
 [25 45 56]
 [31 56 70]]


2. A @ A.T =

 [[14 25 31]
 [25 45 56]
 [31 56 70]]


3. A @ b =

 [[29]
 [50]
 [60]]


4.Cannot be calculated.

 <class 'ValueError'>


5.Cannot be calculated.

 <class 'ValueError'>


6. b.T @ A =

 [[29 50 60]]


7.Cannot be calculated.

 <class 'ValueError'>


8. b.T @ b =

 [[74]]


9. b @ b.T =

 [[ 1 -3 -8]
 [-3  9 24]
 [-8 24 64]]


10.

 The vector with shapes (3,1) and The vector with (1,3) cannot be adde
d.
```

```
11.Cannot be calculated.

 <class 'ValueError'>


12. inv(A) @ b =

 [[ 6.]
 [ 4.]
 [-5.]]


13. A * A =

 [[ 1  4  9]
 [ 4 16 25]
 [ 9 25 36]]


14. b * c =

 [[-4]
 [-9]
 [48]]
```

1.

$$\mathbf{AA} = \begin{bmatrix} 14 & 25 & 31 \\ 25 & 45 & 56 \\ 31 & 56 & 70 \end{bmatrix}$$

2.

$$\mathbf{AA^T} = \begin{bmatrix} 14 & 25 & 31 \\ 25 & 45 & 56 \\ 31 & 56 & 70 \end{bmatrix}$$

3.

$$\mathbf{Ab} = \begin{bmatrix} 29 \\ 50 \\ 60 \end{bmatrix}$$

4.

$\mathbf{Ab}^T$ cannot be calculated.

5.

$\mathbf{bA}$ cannot be calculated.

6.

$$\mathbf{b^T A} = \begin{bmatrix} 29 & 50 & 60 \end{bmatrix}$$

7.

$\mathbf{bb}$ cannot be calculated.

8.

$$\mathbf{b^T b} = \begin{bmatrix} 74 \end{bmatrix}$$

9.

$$\mathbf{bb^T} = \begin{bmatrix} 1 & -3 & -8 \\ -3 & 9 & 24 \\ -8 & 24 & 64 \end{bmatrix}$$

10.

$\mathbf{b} + \mathbf{c}^T$ cannot to be added.

11.

$\mathbf{b}^T \mathbf{b}^T$ cannot be calculated.

12.

$$\mathbf{A^{-1}b} = \begin{bmatrix} 6 \\ 4 \\ -5 \end{bmatrix}$$

13.

$$\mathbf{A} \circ \mathbf{A} = \begin{bmatrix} 1 & 4 & 9 \\ 4 & 16 & 25 \\ 9 & 25 & 36 \end{bmatrix}$$

14.

$$\mathbf{b} \circ \mathbf{c} = \begin{bmatrix} -4 \\ -9 \\ 48 \end{bmatrix}$$

7

[8 points] Eigenvectors and eigenvalues. Eigenvectors and eigenvalues are useful for some machine learning algorithms, but the concepts take time to solidly grasp. For an intuitive review of these concepts, explore this interactive website at Setosa.io (http://setosa.io/ev/eigenvectors-and-eigenvalues/). Also, the series of linear algebra videos by Grant Sanderson of 3Brown1Blue are excellent and can be viewed on youtube here (https://www.youtube.com/playlist?list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab).

1. Calculate the eigenvalues and corresponding eigenvectors of matrix $\mathbf{A}$ above, from the last question.
2. Choose one of the eigenvector/eigenvalue pairs, $\mathbf{v}$ and λ, and show that $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$. Also show that this relationship extends to higher orders: $\mathbf{A}\mathbf{A}\mathbf{v} = \lambda^2\mathbf{v}$
3. Show that the eigenvectors are orthogonal to one another (e.g. their inner product is zero). This is true for real, symmetric matrices.

ANSWER

1. Calculate the eigenvalues and corresponding eigenvectors of matrix $\mathbf{A}$

In [63]:
```
from numpy import linalg

A = np.array([
    [1,2,3],
    [2,4,5],
    [3,5,6]])

eigenvalues, eigenvectors = np.linalg.eig(A)

print("Eigenvalues are \n\n", eigenvalues, end = '\n\n')
print("Eigenvectors are \n\n", eigenvectors, end = '\n\n')
```

```
Eigenvalues are

 [11.34481428 -0.51572947  0.17091519]

Eigenvectors are

 [[-0.32798528 -0.73697623  0.59100905]
 [-0.59100905 -0.32798528 -0.73697623]
 [-0.73697623  0.59100905  0.32798528]]
```

2. Choose one of the eigenvector/eigenvalue pairs, $\mathbf{v}$ and λ, and show that $\mathbf{Av} = \lambda\mathbf{v}$. Also show that this relationship extends to higher orders: $\mathbf{AAv} = \lambda^2\mathbf{v}$

Show that $\mathbf{Av} = \lambda\mathbf{v}$

In [64]:
```
eigenvalue_0 = eigenvalues[0]
eigenvector_0 = eigenvectors[:,0]

print("A*v equals to \n\n", np.dot(A, eigenvector_0), end = '\n\n') #mat
rix by vector
print("lambda*v equals to \n\n", np.dot(eigenvalue_0, eigenvector_0), en
d = '\n\n') #value by vector

print("The above two show that A*v = lambda*v")
```

```
A*v equals to

 [-3.72093206 -6.70488789 -8.36085845]

lambda*v equals to

 [-3.72093206 -6.70488789 -8.36085845]

The above two show that A*v = lambda*v
```

Show that this relationship extends to higher orders: $\mathbf{AAv} = \lambda^2\mathbf{v}$

In [65]:
```
print("AA*v equals to \n\n", np.dot(np.dot(A,A), eigenvector_0), end = '
\n\n') #matrix by vector
print("(lambda^2)*v equals to \n\n", np.dot((eigenvalue_0**2), eigenvect
or_0), end = '\n\n') #value by vector

print("The above two show that AAv = (lambda^2)*v", end = '\n\n')
```

```
AA*v equals to

 [-42.2132832  -76.06570795 -94.85238636]

(lambda^2)*v equals to

 [-42.2132832  -76.06570795 -94.85238636]

The above two show that AAv = (lambda^2)*v
```

3. Show that the eigenvectors are orthogonal to one another

In [66]:
```
print(" The inner products of eigenvectors are: \n")

for i in range(3):
    for j in range(i+1, 3):
        print(np.dot(eigenvectors[:, i], eigenvectors[:, j]), end = '\n
\n')

print("The above shows the eigenvectors of Matrix A are orthogonal to on
e another.")
```

```
 The inner products of eigenvectors are:

-2.220446049250313e-16

-4.440892098500626e-16

-1.0547118733938987e-15

The above shows the eigenvectors of Matrix A are orthogonal to one anot
her.
```

Numerical Programming

8

[10 points] Loading data and gathering insights from a real dataset

Data. The data for this problem can be found in the `data` subfolder in the `assignments` folder on github (https://github.com/kylebradbury/ids705). The filename is `egrid2016.xlsx` . This dataset is the Environmental Protection Agency's (EPA) Emissions & Generation Resource Integrated Database (eGRID) (https://www.epa.gov/energy/emissions-generation-resource-integrated-database-egrid) containing information about all power plants in the United States, the amount of generation they produce, what fuel they use, the location of the plant, and many more quantities. We'll be using a subset of those data.

The fields we'll be using include:

field	description
SEQPLT16	eGRID2016 Plant file sequence number (the index)
PSTATABB	Plant state abbreviation
PNAME	Plant name
LAT	Plant latitude
LON	Plant longitude
PLPRMFL	Plant primary fuel
CAPFAC	Plant capacity factor
NAMEPCAP	Plant nameplate capacity (Megawatts MW)
PLNGENAN	Plant annual net generation (Megawatt-hours MWh)
PLCO2EQA	Plant annual CO2 equivalent emissions (tons)

For more details on the data, you can refer to the eGrid technical documents (https://www.epa.gov/sites/production/files/2018-02/documents/egrid2016_technicalsupportdocument_0.pdf). For example, you may want to review page 45 and the section "Plant Primary Fuel (PLPRMFL)", which gives the full names of the fuel types including WND for wind, NG for natural gas, BIT for Bituminous coal, etc.

There also are a couple of "gotchas" to watch out for with this dataset:

- The headers are on the second row and you'll want to ignore the first row (they're more detailed descriptions of the headers).
- NaN values represent blanks in the data. These will appear regularly in real-world data, so getting experience working with it will be important.

Your objective. For this dataset, your goal is answer the following questions about electricity generation in the United States:

(a) Which plant has generated the most energy (measured in MWh)?

(b) What is the name of the northern-most power plant in the United States?

(c) What is the state where the northern-most power plant in the United States is located?

(d) Plot a histogram of the amount of energy produced by each fuel for the plant.

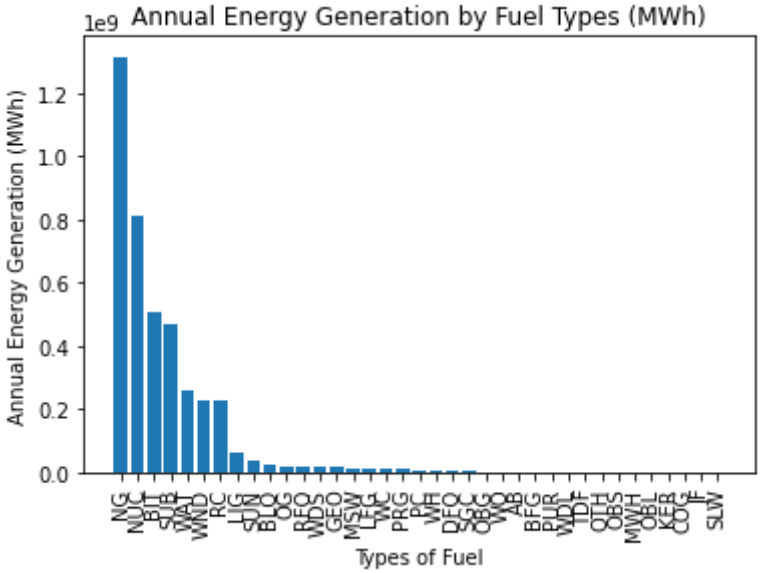
(e) From the plot in (e), which fuel for generation produces the most energy (MWh) in the United States?

ANSWER

In [67]:

```
import pandas as pd
import matplotlib.pyplot as plt

#Loading the dataset

df = pd.read_excel("/Users/Reinhard/egrid2016.xlsx", header=1)

#Locating the power plant generating with the most energy

max_energy_plant = df.loc[df['PLNGENAN'].idxmax()]
print('(a) The plant that produced the most energy is the {} plant.'.for
mat(str(max_energy_plant['PNAME'])))

#Locating the northern-most power plant

northern_most_plant = df.loc[df['LAT'].idxmax()]

print('(b) The northern-most plant is the {} plant.'.format(str(northern
_most_plant['PNAME'])))

print('(c) The northern-most plant is in {}.'.format(str(northern_most_p
lant['PSTATABB'])))

#Plot a histogram of the amount of energy produced by each fuel for the
 plant

# Grouped Energy Generation by fuel types.
grouped_by_fuel = df.groupby('PLPRMFL', as_index = False).sum().sort_val
ues('PLNGENAN', ascending=False)

#Retriving fuel types
fuel_types = grouped_by_fuel['PLPRMFL']

#Retriving amount of energy generation across different fuel types.
energy_generation = grouped_by_fuel['PLNGENAN']

#Plotting
plt.bar(fuel_types, energy_generation)
plt.xticks(rotation=90)
plt.title('Annual Energy Generation by Fuel Types (MWh)')
plt.xlabel('Types of Fuel')
plt.ylabel('Annual Energy Generation (MWh)')
plt.show()

#Locating the fuel produces the most energy.

max_fuel = grouped_by_fuel.loc[grouped_by_fuel['PLNGENAN'].idxmax()]

print('(e) The fuel producing the most energy in the United States is {}
.'.format(str(max_fuel['PLPRMFL'])))
```

```
(a) The plant that produced the most energy is the Palo Verde plant.
(b) The northern-most plant is the Barrow plant.
(c) The northern-most plant is in AK.
```




```
(e) The fuel producing the most energy in the United States is NG.
```

9

[8 points] Speed comparison between vectorized and non-vectorized code. Begin by creating an array of 10 million random numbers using the numpy random.randn module. Compute the sum of the squares first in a for loop, then using Numpy's `dot` module. Time how long it takes to compute each and report the results and report the output. How many times faster is the vectorized code than the for loop approach?

*Note: all code should be well commented, properly formatted, and your answers should be output using the `print()` function as follows (where the # represents your answers, to a reasonable precision):

```
Time [sec] (non-vectorized): ######

Time [sec] (vectorized):     ######

The vectorized code is ##### times faster than the vectorized code
```

ANSWER

In [68]:
```
import numpy as np
import time

# Generating the array asked in the question.
ran_arr = np.random.randn(10000000)

# Calculating the sum of squares by looping
start_time1 = time.time()
total1 = 0
for i in ran_arr:
    total1 += i**2
end_time1 = time.time()

# Calculating the sum of squares by using vectorization
start_time2 = time.time()
total2 = np.dot(ran_arr, ran_arr)
end_time2 = time.time()

# Print the results
print("Time [sec] (non-vectorized): %.4f" % (end_time1 - start_time1))
print("Time [sec] (vectorized):     %.4f" % (end_time2 - start_time2))
print("The vectorized code is %3.2f times faster than the non-vectorized
code" % ((end_time1 - start_time1)/(end_time2 - start_time2)))
```

```
Time [sec] (non-vectorized): 6.6930
Time [sec] (vectorized):     0.0053
The vectorized code is 1272.21 times faster than the non-vectorized cod
e
```

10

[10 points] One popular Agile development framework is Scrum (a paradigm recommended for data science projects). It emphasizes the continual evolution of code for projects, becoming progressively better, but starting with a quickly developed minimum viable product. This often means that code written early on is not optimized, and that's a good thing - it's best to get it to work first before optimizing. Imagine that you wrote the following code during a sprint towards getting an end-to-end system working. Vectorize the following code and show the difference in speed between the current implementation and a vectorized version.

The function below computes the function $f(x, y) = x^2 - 2y^2$ and determines whether this quantity is above or below a given threshold, `thresh=0` . This is done for $x, y \in \{-4, 4\}$, over a 2,000-by-2,000 grid covering that domain.

(a) Vectorize this code and demonstrate (as in the last exercise) the speed increase through vectorization and (b) plot the resulting data - both the function $f(x, y)$ and the thresholded output - using `imshow` (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.imshow.html?highlight=matplotlib%20pyplot%20imshow#matplotlib.pyplot.imshow) from `matplotlib` .

Hint: look at the `numpy` `meshgrid` *(https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.meshgrid.html) documentation*

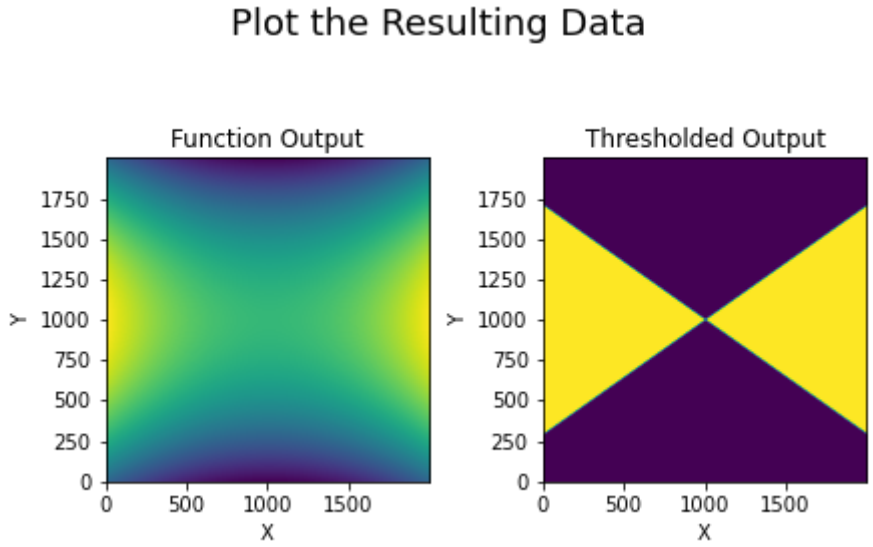
ANSWER

In [69]:

```
import numpy as np
import time
import matplotlib.pyplot as plt

nvalues = 2000
xvalues = np.linspace(-4,4,nvalues)
yvalues = np.linspace(-4,4,nvalues)
thresh  = 0

# Nonvectorized implementation

t0 = time.time()
f = np.zeros((nvalues,nvalues))
f_thresholded = np.zeros((nvalues,nvalues))
for ix, x in enumerate(xvalues):
    for iy, y in enumerate(yvalues):
        f[ix,iy]             = x**2 - 2 * y**2
        f_thresholded[ix,iy] = f[ix,iy] > thresh
t1 = time.time()
time_nonvectorized = t1 - t0

# Vectorized implementation

t0_vector = time.time()

#Create matrix by using meshgrid

X, Y = np.meshgrid(xvalues, yvalues)

# Calculating Z by matrices computation.

Z = np.multiply(X, X) - 2 * np.multiply(Y,Y)

Z_thresh = np.greater(Z,thresh)

t1_vector = time.time()

time_vectorized = t1_vector-t0_vector

# Vectorized speed performance results

print("Time [sec] (non-vectorized): %.4f" % time_nonvectorized)
print("Time [sec] (vectorized):     %.4f" % time_vectorized)
print("The vectorized code is %3.1f times faster than the non-vectorized
code" % (time_nonvectorized/time_vectorized))

#Plot the function f(x, y)

#Creating figure to hold both graphs.

fig, axs = plt.subplots(1, 2, constrained_layout=True)
axs[0].set_title('Function Output')
axs[0].imshow(Z, origin='lower') #set origin at 0,0
axs[0].set_xlabel('X')
axs[0].set_ylabel('Y')
fig.suptitle('Plot the Resulting Data', fontsize=18)
```

```
axs[1].set_title('Thresholded Output')
axs[1].imshow(Z_thresh, origin='lower')
axs[1].set_xlabel('X')
axs[1].set_ylabel('Y')

plt.show()
```

```
Time [sec] (non-vectorized): 11.0507
Time [sec] (vectorized):     0.1236
The vectorized code is 89.4 times faster than the non-vectorized code
```



11

[10 points] This exercise will walk through some basic numerical programming exercises.

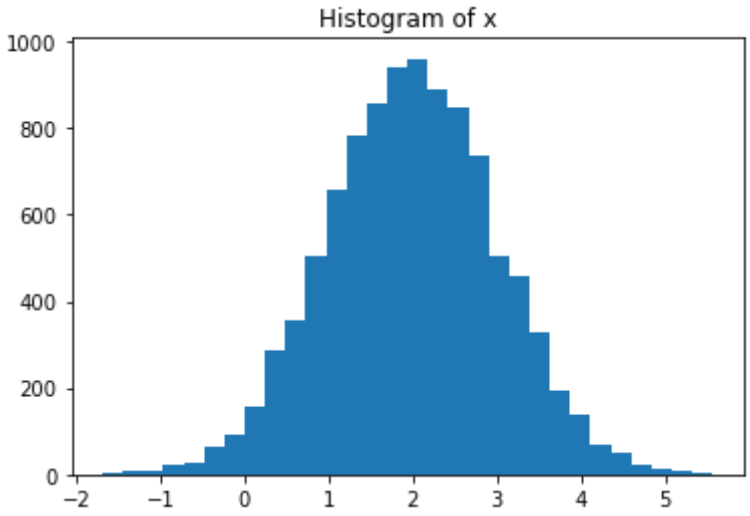
1. Synthesize $n = 10^4$ normally distributed data points with mean $\mu = 2$ and a standard deviation of $\sigma = 1$. Call these observations from a random variable X, and call the vector of observations that you generate, $\mathbf{x}$.
2. Calculate the mean and standard deviation of $\mathbf{x}$ to validate (1) and provide the result to a precision of four significant figures.
3. Plot a histogram of the data in $\mathbf{x}$ with 30 bins
4. What is the 90th percentile of $\mathbf{x}$? The 90th percentile is the value below which 90% of observations can be found.
5. What is the 99th percentile of $\mathbf{x}$?
6. Now synthesize $n = 10^4$ normally distributed data points with mean $\mu = 0$ and a standard deviation of $\sigma = 3$. Call these observations from a random variable Y, and call the vector of observations that you generate, $\mathbf{y}$.
7. Create a new figure and plot the histogram of the data in $\mathbf{y}$ on the same axes with the histogram of $\mathbf{x}$, so that both histograms can be seen and compared.
8. Using the observations from $\mathbf{x}$ and $\mathbf{y}$, estimate $E[XY]$

ANSWER

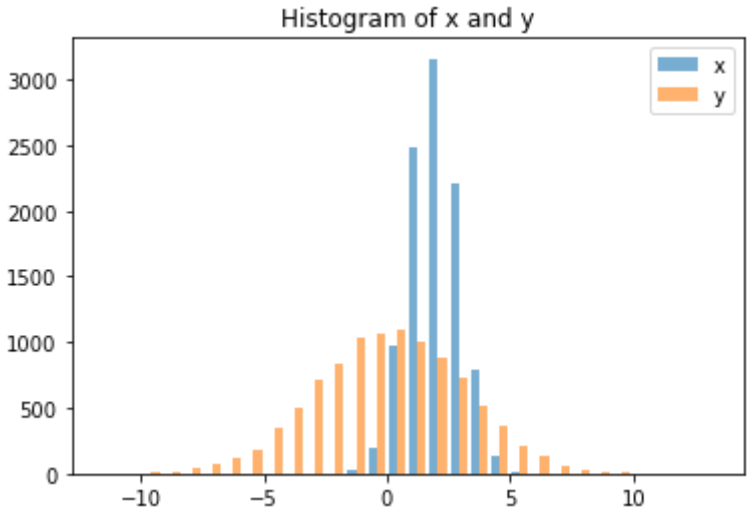
In [70]:

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# 1. Generating normally distributed data points

x = np.random.normal(2, 1, 10000)

# 2. Calculating mean and standard deviation with mean 2 and standard de
viation 1
print('The mean of x is:\t\t %1.3f' % np.mean(x))
print('The standard deviation of x is\t %1.3f' % np.std(x))

# 3. Plotting histogram
plt.hist(x, bins=30)
plt.title('Histogram of x')
plt.show()

# 4. Calculating 90th percentile of x

print('The 90th percentile of the data x is %1.3f.' % np.percentile(x, 9
0))


# 5.Calculating 99th percentile of x

print('The 99th percentile of the data x is %1.3f.' % np.percentile(x, 9
9))

# 6. Generating normally distributed data points with mean 0 and standar
d deviation 3.

y = np.random.normal(0, 3, 10000)

# 7. Plotting both histograms on the same figure.

plt.figure()
plt.title("Histogram of x and y")
#plt.hist(y, bins = 30, alpha = 0.6, color = 'orange', label = 'y')
#plt.hist(x, bins = 30, alpha = 0.6, label = 'x')
plt.hist([x, y], bins = 30, alpha = 0.6, label = ['x', 'y'])
plt.legend()
plt.show()

# 8. Evaluating E[XY].

print("The estimated value of XY is %1.3f " % np.mean(x*y))
```

```
The mean of x is:                1.991
The standard deviation of x is   1.004
```




```
The 90th percentile of the data x is 3.287.
The 99th percentile of the data x is 4.346.
```




```
The estimated value of XY is 0.008
```

Version Control via Git

12

[1 point] You will need to use Git to submit assignments and in the course projects and is generally a version control and collaboration tool. You can even use some Git repositories (e.g. Github) as hosts for website, such as with the course website (https://kylebradbury.github.io/ids705/index.html).

Complete the Atlassian Git tutorial (https://www.atlassian.com/git/tutorials/what-is-version-control), specifically the following listed sections. Try each concept that's presented. For this tutorial, instead of using BitBucket as your remote repository host, you may use your preferred platform such as Github (https://github.com/) or Duke's Gitlab (https://gitlab.oit.duke.edu/users/sign_in).

1. What is version control (https://www.atlassian.com/git/tutorials/what-is-version-control)
2. What is Git (https://www.atlassian.com/git/tutorials/what-is-git)
3. Install Git (https://www.atlassian.com/git/tutorials/install-git)
4. Setting up a repository (https://www.atlassian.com/git/tutorials/install-git)
5. Saving changes (https://www.atlassian.com/git/tutorials/saving-changes)
6. Inspecting a repository (https://www.atlassian.com/git/tutorials/inspecting-a-repository)
7. Undoing changes (https://www.atlassian.com/git/tutorials/undoing-changes)
8. Rewriting history (https://www.atlassian.com/git/tutorials/rewriting-history)
9. Syncing (https://www.atlassian.com/git/tutorials/syncing)
10. Making a pull request (https://www.atlassian.com/git/tutorials/making-a-pull-request)
11. Using branches (https://www.atlassian.com/git/tutorials/using-branches)
12. Comparing workflows (https://www.atlassian.com/git/tutorials/comparing-workflows)

I also have created two videos on the topic to help you understand some of these concepts: Git basics (https://www.youtube.com/watch?v=fBCwfoBr2ng) and a step-by-step tutorial (https://www.youtube.com/watch?v=nH7qJHx-h5s).

For your answer, affirm that you *either* completed the tutorial or have previous experience with all of the concepts above. Do this by typing your name below and selecting the situation that applies from the two options in brackets.

ANSWER

*I, [**Zhenxing Xie**], affirm that I have [**I have previous experience that covers all the content in this tutorial**]*

Exploratory Data Analysis

13

[20 points] Here you'll bring together some of the individual skills that you demonstrated above and create a Jupyter notebook based blog post on data analysis.

1. Find a dataset that interests you and relates to a question or problem that you find intriguing
2. Using a Jupyter notebook, describe the dataset, the source of the data, and the reason the dataset was of interest.
3. Check the data and see if they need to be cleaned: are there missing values? Are there clearly erroneous values? Do two tables need to be merged together? Clean the data so it can be visualized.
4. Plot the data, demonstrating interesting features that you discover. Are there any relationships between variables that were surprising or patterns that emerged? Please exercise creativity and curiosity in your plots.
5. What insights are you able to take away from exploring the data? Is there a reason why analyzing the dataset you chose is particularly interesting or important? Summarize this as if your target audience was the readership of a major news organization - boil down your findings in a way that is accessible, but still accurate.

Here your analysis will evaluated based on:

1. Data cleaning: did you look for and work to resolve issues in the data?
2. Quality of data exploration: did you provide plots demonstrating interesting aspects of the data?
3. Interpretation: Did you clearly explain your insights? Restating the data, alone, is not interpretation.
4. Professionalism: Was this work done in a way that exhibits professionalism through clarity, organization, high quality figures and plots, and meaningful descriptions?

ANSWER

Housing Prices Exploration on Beijing

Introduction

In this section, I will try to examine the potential factors which exerted influences on housing prices in Beijing. Specifically, I have scraped the data from the website of the largest housing agency which covers over 60% deals of housing in the megacity, and it includes several metrics regarding the housing transactions for the past 8 years with thousands of communities. And then I will preprocess the dataset and conduct exploratory data analysis to unveil what data is telling.

Data Preprocessing

Loading the dataset

In [71]:
```
import pandas as pd
import os
import numpy as np
from numpy import nan as NaN
import matplotlib.pyplot as plt
import matplotlib
import seaborn as sns

cwd = os.getcwd()
print(cwd)

df = pd.read_csv("/Users/Reinhard/Documents/Duke_Data Science/Spring 2021/705/Assignments/raw_haidian.csv")
df.head()
```

```
/Users/Reinhard/IDS705_Principles-of-Machine-Learning
```

Out[71]:

	Unnamed: 0	Housing ID	Livable Area	Price/Square Meter	Price	Transaction Date	Year Built	Floor Plan	Facing Direction	Floor
0	1	中关村南大街甲3号 2室1厅 56.3平米	56.3平米	38544.0	217.0	2015-05-08	1965年建板楼	2室1厅	南 北	低楼层(共3层)
1	2	中关村南大街甲3号 2室1厅 55.1平米	55.1平米	33667.0	185.5	2012-04-10	1965年建板楼	2室1厅	南 北	底层(共3层)
2	3	民族大学南路19号院 2室1厅 70平米	70平米	91429.0	640.0	2019-04-19	1993年建板塔结合	2室1厅	南	中楼层(共15层)
3	4	民族大学南路19号院 2室1厅 98.49平米	98.49平米	102752.0	1012.0	2019-04-07	2000年建塔楼	2室1厅	南	高楼层(共18层)
4	5	民族大学南路19号院 2室1厅 60.7平米	60.7平米	98847.0	600.0	2019-02-22	1991年建板塔结合	2室1厅	南	高楼层(共16层)

Checking Data Types

In [72]:
```
df.dtypes
```

Out[72]:
```
Unnamed: 0              int64
Housing ID             object
Livable Area           object
Price/Square Meter    float64
Price                 float64
Transaction Date       object
Year Built             object
Floor Plan             object
Facing Direction       object
Floor                  object
Decoration             object
dtype: object
```

Target Description

Price/Square Meter: Housing transaction price in RMB per square meter

Features Description

Housing ID: Community Name + Floor Plan + Living Area

Livable Area: Living area of a house in square meter

Price: Total transaction price for a house

Transaction Date: year-month-date

Year Built: The year of a house established

Floor Plan: Number of bedrooms and living rooms

Facing Direction: Which direction a house is facing

Floor: High/Medium/Low/Basement

Decoration: Simple/Refined

Checking Missing Data

In [73]:
```
df.isnull().sum(axis=0)
```

Out[73]:
```
Unnamed: 0              0
Housing ID              0
Livable Area            0
Price/Square Meter      0
Price                   0
Transaction Date        0
Year Built              0
Floor Plan              0
Facing Direction       62
Floor                   0
Decoration              0
dtype: int64
```

Taking a deeper look at Missing Data

In [74]:
```
df[df.isna().T.any()].head()
```

Out[74]:

	Unnamed: 0	Housing ID	Livable Area	Price/Square Meter	Price	Transaction Date	Year Built	Floor Plan	Facing Direction	Floo
620	621	万寿寺北里 3室1厅1卫	56.9平米	77153.0	439.0	2017-03-11	暂无信息	3室1厅	NaN	暂无信息
4222	4223	公交党校宿舍 2室1厅1卫	67.81平米	68869.0	467.0	2018-05-01	暂无信息	2室1厅	NaN	暂无信息
5097	5098	永泰东里 3室1厅1卫	70.98平米	62835.0	446.0	2018-08-13	暂无信息	3室1厅	NaN	暂无信息
5128	5129	永泰东里 1室1厅1卫	58.91平米	60771.0	358.0	2018-05-01	暂无信息	1室1厅	NaN	暂无信息
5848	5849	后屯路32号院 2室1厅1卫	60.49平米	67780.0	410.0	2017-07-17	暂无信息	2室1厅	NaN	暂无信息

Notes: From the above table, I found that there are unidentified missing data which are expressed as "暂无信息" in addition to "NaN". Therefore, I should replace "暂无信息" with "NaN", However, since I just had a snapshot for a few data points, I will go over columns one by one to examine what exact happen in those columns.

Checking Column: Decoration

In [75]:
```
# Checking value classes and values distribution

df['Decoration'].value_counts()
```

Out[75]:
```
精装      26202
简装      21363
其他      20922
暂无信息     2940
毛坯       1008
Name: Decoration, dtype: int64
```

In [76]:
```
# Rephrasing value classes and including unidentified missing data "暂无信息"

df.loc[df['Decoration'] == '毛坯' , 'Decoration'] = 'bare'
df.loc[df['Decoration'] == '简装' , 'Decoration'] = 'simple'
df.loc[df['Decoration'] == '精装' , 'Decoration'] = 'refined'
df.loc[df['Decoration'] == '其他' , 'Decoration'] = 'other'
df.loc[df['Decoration'] == '暂无信息' , 'Decoration'] = NaN
```

In [77]:
```
# Checking missing values in this column
df.isnull().sum()
```

Out[77]:
```
Unnamed: 0               0
Housing ID               0
Livable Area             0
Price/Square Meter       0
Price                    0
Transaction Date         0
Year Built               0
Floor Plan               0
Facing Direction        62
Floor                    0
Decoration            2940
dtype: int64
```

Notes: We see that there are 2940 missing values added by including the unidentified missing data "暂无信息".

Checking Column: Floor

In [78]:
```
# Checking value classes

df['Floor'].unique()
```

Out[78]:
```
array(['低楼层(共3层)', '底层(共3层)', '中楼层(共15层)', '高楼层(共18层)', '高楼层(共16层)',
       '低楼层(共18层)', '中楼层(共18层)', '高楼层(共6层)', '顶层(共15层)', '中楼层(共16层)',
       '低楼层(共15层)', '高楼层(共15层)', '底层(共15层)', '低楼层(共6层)', '中楼层(共6层)',
       '底层(共16层)', '顶层(共18层)', '顶层(共6层)', '低楼层(共16层)', '底层(共5层)',
       '顶层(共5层)', '中楼层(共5层)', '顶层(共16层)', '中楼层(共14层)', '低楼层(共14层)',
       '顶层(共14层)', '高楼层(共14层)', '底层(共6层)', '高楼层(共11层)', '低楼层(共11层)',
       '中楼层(共11层)', '底层(共11层)', '暂无信息', '底层(共18层)', '高楼层(共24层)',
       '顶层(共24层)', '低楼层(共24层)', '中楼层(共24层)', '底层(共12层)', '高楼层(共12层)',
       '低楼层(共12层)', '中楼层(共12层)', '顶层(共12层)', '中楼层(共20层)', '高楼层(共20层)',
       '低楼层(共20层)', '中楼层(共25层)', '高楼层(共25层)', '低楼层(共25层)', '低楼层(共10层)',
       '中楼层(共10层)', '底层(共14层)', '高楼层(共10层)', '底层(共9层)', '顶层(共9层)',
       '高楼层(共9层)', '低楼层(共9层)', '中楼层(共9层)', '中楼层(共23层)', '高楼层(共17层)',
       '低楼层(共22层)', '中楼层(共22层)', '高楼层(共23层)', '低楼层(共23层)', '高楼层(共22层)',
       '底层(共23层)', '顶层(共23层)', '顶层(共22层)', '底层(共4层)', '中楼层(共4层)',
       '中楼层(共7层)', '底层(共7层)', '低楼层(共7层)', '高楼层(共7层)', '高楼层(共21层)',
       '低楼层(共21层)', '高楼层(共19层)', '底层(共10层)', '中楼层(共21层)', '顶层(共11层)',
       '中楼层(共19层)', '地下室(共10层)', '地下室(共9层)', '顶层(共10层)', '顶层(共19层)',
       '低楼层(共19层)', '底层(共21层)', '顶层(共7层)', '底层(共19层)', '中楼层(共17层)',
       '低楼层(共17层)', '顶层(共17层)', '地下室(共17层)', '底层(共17层)', '顶层(共8层)',
       '未知(共10层)', '未知(共7层)', '地下室(共5层)', '顶层(共3层)', '地下室(共7层)',
       '底层(共2层)', '地下室(共2层)', '顶层(共20层)', '未知(共6层)', '顶层(共4层)',
       '高楼层(共27层)', '中楼层(共27层)', '底层(共22层)', '低楼层(共13层)', '中楼层(共13层)',
       '底层(共13层)', '高楼层(共13层)', '低楼层(共27层)', '未知(共18层)', '顶层(共21层)',
       '底层(共27层)', '顶层(共27层)', '低楼层(共30层)', '中楼层(共31层)', '高楼层(共31层)',
       '低楼层(共31层)', '高楼层(共30层)', '底层(共31层)', '中楼层(共30层)', '未知(共31层)',
       '顶层(共13层)', '中楼层(共28层)', '低楼层(共28层)', '高楼层(共28层)', '底层(共28层)',
       '顶层(共28层)', '地下室(共6层)', '中楼层(共8层)', '底层(共8层)', '低楼层(共8层)',
       '低楼层(共26层)', '中楼层(共26层)', '底层(共26层)', '顶层(共25层)', '高楼层(共26层)',
       '底层(共25层)', '顶层(共26层)', '底层(共20层)', '地下室(共15层)', '高
```

```
楼层(共8层)',
       '未知(共5层)', '地下室(共26层)', '未知(共16层)', '底层(共30层)', '顶层
(共30层)',
       '未知(共12层)', '顶层(共2层)', '地下室', '地下室(共8层)', '未知(共8
层)', '底层(共1层)',
       '底层(共24层)', '低楼层(共29层)', '顶层(共31层)', '中楼层(共29层)',
'未知(共30层)',
       '地下室(共21层)', '未知(共19层)', '地下室(共25层)', '未知(共23层)',
'地下室(共11层)',
       '地下室(共13层)', '地下室(共1层)', '地下室(共4层)', '地下室(共3层)',
'低楼层(共32层)',
       '中楼层(共32层)', '高楼层(共32层)', '顶层(共32层)', '底层(共32层)',
'未知(共4层)',
       '未知(共9层)', '未知(共17层)', '高楼层(共4层)', '地下室(共0层)', '低楼
层(共35层)',
       '中楼层(共35层)', '高楼层(共35层)', '顶层(共35层)', '高楼层(共29层)',
'地下室(共20层)',
       '地下室(共18层)', '地下室(共16层)', '地下室(共12层)'], dtype=object)
```

In [79]:
```
# Recategorizing the classes

df['Floor'] = df['Floor'].str[0:1]
```

In [80]:
```
# Rechecking the classes

df['Floor'].unique()
```

Out[80]:
```
array(['低', '底', '中', '高', '顶', '暂', '地', '未'], dtype=object)
```

In [81]:
```
# Checking values distribution

df['Floor'].value_counts()
```

Out[81]:
```
中    26273
高    13979
低    13448
顶     8630
底     6618
暂     2940
地      365
未      182
Name: Floor, dtype: int64
```

In [82]:
```
# Rephrasing value classes and including unidentified missing data "暂无
信息"

df.loc[df['Floor'] == '地' , 'Floor'] = 'basement'
df.loc[df['Floor'] == '底' , 'Floor'] = 'first'
df.loc[df['Floor'] == '低' , 'Floor'] = 'low'
df.loc[df['Floor'] == '中' , 'Floor'] = 'middle'
df.loc[df['Floor'] == '高' , 'Floor'] = 'high'
df.loc[df['Floor'] == '顶' , 'Floor'] = 'top'
df.loc[df['Floor'] == '暂' , 'Floor'] = NaN
df.loc[df['Floor'] == '未' , 'Floor'] = NaN
```

In [83]:
```
# Checking missing data again

df.isnull().sum()
```

Out[83]:
```
Unnamed: 0               0
Housing ID               0
Livable Area             0
Price/Square Meter       0
Price                    0
Transaction Date         0
Year Built               0
Floor Plan               0
Facing Direction        62
Floor                 3122
Decoration            2940
dtype: int64
```

Checking Column: Facing Direction

In [84]:
```
# Checking value classes

df['Facing Direction'].unique()
```

Out[84]:
```
array(['南 北', '南', '东北', '东 南 西北', '西南', '东 西 北', '西 南', '东
南 北', '东 西',
       '东 南', '东 南 西', '东', '北', '东南', '西北', '东 西南', '西 北',
'南 西', '东 北',
       '西', '东南 北', '西南 北', '东 西北', '南北', '南 西 北', '东西', na
n, '东南 西',
       '东南 西北', '东 南 西 北', '西 南 北', '北 西南', '西南 东北', '西 东
北', '东南 南 西南',
       '东 东北', '北 东北', '北 南', '南 东北', '南 北 东', '南 北 东北',
'西 西北 北',
       '西 西南 南', '东 东南 北', '东 东南 南', '南 北 西', '南 西南', '西 东
南', '南 西北',
       '东南 南', '东 东南', '南 东', '西南 西', '暂无数据', '东 西 南', '东南
东', '北 东南',
       '东 北 东北', '东 西 南 北', '东南 东北', '西 西北', '东南 西南', '东
南 西南', '东 北 西南',
       '西北 东北', '西南 西北', '南 北 东 西', '西 东', '南 东南', '西 北
南', '南 西南 西',
       '西北 北', '南 西 西北', '西 西南', '北 西北', '东 西南 北', '东南 南
北', '东 西北 北 东北',
       '东南 西 北', '西南 西 南', '南 东 西', '南 西南 北', '西 西北 南', '东
南 东南', '东北 南',
       '南 西南 东北', '西北 东南'], dtype=object)
```

In [85]:
```
# Checking values distribution

df['Facing Direction'].value_counts()
```

Out[85]:
```
南 北       29057
南          13541
东           3733
东 西         3229
西           2662
            ...
西 西南           1
东 北 东北         1
东 西南 北         1
南 西 西北         1
南 西南 北         1
Name: Facing Direction, Length: 86, dtype: int64
```

In [86]:
```
# Recategorizing the classes

df.loc[df['Facing Direction'].astype(str).str.contains('南'), 'Facing Dir
ection'] = 'South'
df.loc[df['Facing Direction'].astype(str).str.contains('北'), 'Facing Dir
ection'] = 'North'
df.loc[df['Facing Direction'].astype(str).str.contains('东'), 'Facing Dir
ection'] = 'East/West'
df.loc[df['Facing Direction'].astype(str).str.contains('西'), 'Facing Dir
ection'] = 'East/West'
```

In [87]:
```
# Rechecking the value classes and distribution

df['Facing Direction'].value_counts()
```

Out[87]:
```
South        56001
East/West     9773
North         6590
暂无数据             9
Name: Facing Direction, dtype: int64
```

In [88]:
```
# including unidentified missing data "暂无数据"

df.loc[df['Facing Direction'] == '暂无数据' , 'Facing Direction'] = NaN
```

In [89]:
```
# Rechecking missing data

df.isnull().sum()
```

Out[89]:
```
Unnamed: 0               0
Housing ID               0
Livable Area             0
Price/Square Meter       0
Price                    0
Transaction Date         0
Year Built               0
Floor Plan               0
Facing Direction        71
Floor                 3122
Decoration            2940
dtype: int64
```

Checking Column: Year Built

In [90]:
```
# Checking value classes

df['Year Built'].unique()
```

Out[90]:
```
array(['1965年建板楼', '1993年建板塔结合', '2000年建塔楼', '1991年建板塔结合',
'1985年建板楼',
       '1989年建塔楼', '1984年建板楼', '1982年建板楼', '1983年建板楼', '1988
年建板楼',
       '1979年建板楼', '1981年建板楼', '1980年建板楼', '1997年建塔楼', '1996
年建塔楼',
       '1995年建塔楼', '1997年建板塔结合', '1998年建塔楼', '1997年建板楼', '2
007年建板塔结合',
       '2007年建板楼', '1994年建塔楼', '暂无信息', '1990年建板楼', '1999年建
板楼', '1989年建板楼',
       '板楼', '1994年建板楼', '1993年建板楼', '2003年建塔楼', '2004年建塔
楼', '暂无数据',
       '2000年建板塔结合', '板塔结合', '2002年建塔楼', '1978年建板楼', '1986
年建板塔结合',
       '1985年建板塔结合', '2001年建塔楼', '1999年建板塔结合', '1995年建板塔结
合', '2001年建板塔结合',
       '2001年建板楼', '1990年建板塔结合', '1992年建板楼', '1991年建板楼', '1
995年建板楼',
       '2000年建板楼', '2003年建板楼', '1996年建板楼', '2012年建板楼', '2010
年建板楼',
       '2014年建板楼', '2011年建板楼', '2013年建板楼', '2016年建板楼', '2007
年建塔楼',
       '1987年建板楼', '1986年建板楼', '2002年建板楼', '1999年建塔楼', '1998
年建板楼',
       '2003年建板塔结合', '2005年建板塔结合', '2004年建板楼', '2005年建板楼',
'2008年建板塔结合',
       '2008年建板楼', '2005年建塔楼', '1993年建塔楼', '1996年建板塔结合', '2
002年建板塔结合',
       '2004年建板塔结合', '2004年建暂无数据', '2003年建暂无数据', '2014年建板
塔结合',
       '2013年建板塔结合', '2009年建板楼', '2002年建暂无数据', '2005年建暂无数
据', '1998年建板塔结合',
       '1976年建板楼', '1963年建板楼', '2010年建塔楼', '2015年建板楼', '塔
楼', '2006年建板楼',
       '2006年建板塔结合', '1975年建板楼', '2009年建板塔结合', '2012年建板塔结
合', '2011年建板塔结合',
       '2016年建塔楼', '2018年建暂无数据', '2016年建暂无数据', '2017年建板楼',
'1984年建塔楼',
       '1986年建塔楼', '1985年建塔楼', '1989年建板塔结合', '1988年建板塔结合',
'1983年建板塔结合',
       '1984年建板塔结合', '1987年建塔楼', '1980年建板塔结合', '1992年建塔楼',
'1992年建板塔结合',
       '1981年建塔楼', '1982年建塔楼', '1982年建板塔结合', '1979年建板塔结合',
'1994年建板塔结合',
       '1950年建板楼', '1970年建板楼', '1967年建板楼', '1960年建板楼', '1973
年建板楼',
       '1959年建板楼', '1962年建板楼', '1968年建板楼', '1958年建板楼', '1980
年建塔楼',
       '1981年建板塔结合', '1987年建板塔结合', '1977年建板楼', '1957年建板楼',
'1961年建板楼',
       '1990年建塔楼', '1991年建塔楼', '1974年建板楼', '1956年建板楼', '1963
年建板塔结合',
       '1970年建板塔结合', '1958年建板塔结合', '1964年建板楼', '1960年建板塔结
合', '2006年建塔楼',
       '1955年建板楼', '2009年建塔楼', '1988年建塔楼', '1983年建塔楼', '2010
年建板塔结合',
       '2018年建板楼', '2008年建暂无数据', '2012年建塔楼', '2010年建暂无数据',
```

```
'2011年建暂无数据',
       '1992年建暂无数据', '1966年建板楼', '2012年建暂无数据', '2000年建暂无数
据', '2001年建暂无数据',
       '底层(共6层)', '2008年建塔楼', '1982年建平房', '2009年建暂无数据', '19
78年建板塔结合',
       '2006年建暂无数据', '2007年建暂无数据', '1980年建暂无数据', '1993年建暂
无数据', '1954年建板楼',
       '2013年建塔楼', '1997年建暂无数据', '1999年建暂无数据', '1972年建板楼',
'2015年建板塔结合',
       '2011年建塔楼', '平房'], dtype=object)
```

In [91]:
```
# Keep the first 4 digits for 'Year Built'

df['Year Built'] = df['Year Built'].astype('str').str[0:4]
df['Year Built'].unique()
```

Out[91]:
```
array(['1965', '1993', '2000', '1991', '1985', '1989', '1984', '1982',
       '1983', '1988', '1979', '1981', '1980', '1997', '1996', '1995',
       '1998', '2007', '1994', '暂无信息', '1990', '1999', '板楼', '200
3',
       '2004', '暂无数据', '板塔结合', '2002', '1978', '1986', '2001', '19
92',
       '2012', '2010', '2014', '2011', '2013', '2016', '1987', '2005',
       '2008', '2009', '1976', '1963', '2015', '塔楼', '2006', '1975',
       '2018', '2017', '1950', '1970', '1967', '1960', '1973', '1959',
       '1962', '1968', '1958', '1977', '1957', '1961', '1974', '1956',
       '1964', '1955', '1966', '底层(共', '1954', '1972', '平房'], dtype=
object)
```

In [92]:
```
# including unidentified missing data

df.loc[df['Year Built'] == '暂无信息', 'Year Built'] = NaN
df.loc[df['Year Built'] == '板楼' , 'Year Built'] = NaN
df.loc[df['Year Built'] == '暂无数据', 'Year Built'] = NaN
df.loc[df['Year Built'] == '板塔结合', 'Year Built'] = NaN
df.loc[df['Year Built'] == '塔楼', 'Year Built'] = NaN
df.loc[df['Year Built'] == '底层(共' , 'Year Built'] = NaN
df.loc[df['Year Built'] == '平房', 'Year Built'] = NaN
```

In [93]:
```
# Rechecking the value classes and distribution

df['Year Built'].value_counts()
```

Out[93]:
```
1998    4522
2000    4084
2004    3656
1990    3327
1996    3279
        ...
2017       3
1961       3
2018       3
1950       2
1972       2
Name: Year Built, Length: 64, dtype: int64
```

In [94]:
```
# Rechecking missing data

df.isnull().sum()
```

Out[94]:
```
Unnamed: 0                0
Housing ID                0
Livable Area              0
Price/Square Meter        0
Price                     0
Transaction Date          0
Year Built             7159
Floor Plan                0
Facing Direction         71
Floor                  3122
Decoration             2940
dtype: int64
```

Checking Column: Floor Plan

In [95]:
```
# Checking value classes

df['Floor Plan'].unique()
```

Out[95]:
```
array(['2室1厅', '3室1厅', '3室2厅', '1室1厅', '2室0厅', '3室0厅', '1室0厅',
'4室1厅',
       '4室2厅', '2室2厅', '4室0厅', '5室1厅', '1室2厅', '3室3厅', '6室1厅',
'6室2厅',
       '5室2厅', '4室3厅', '5室3厅', '6室3厅', '2室3厅', '5室4厅', '4室4厅',
'0室0厅',
       '1房间1卫', '3房间1卫', '2房间1卫', '1房间0卫', '6室4厅', '4房间2卫',
'2房间0卫',
       '5房间3卫', '5房间2卫', '4室5厅', '7室3厅', '9室0厅', '8室4厅', '7室2
厅', '--室--厅',
       '2房间2卫', '5室0厅', '3房间2卫', '4室9厅'], dtype=object)
```

In [96]:
```
# Including unidentified missing data

df.loc[df['Floor Plan'] == '--室--厅', 'Floor Plan'] = NaN
df.loc[df['Floor Plan'] == '0室0厅', 'Floor Plan'] = NaN

df['Floor Plan'].unique()
```

Out[96]:
```
array(['2室1厅', '3室1厅', '3室2厅', '1室1厅', '2室0厅', '3室0厅', '1室0厅',
'4室1厅',
       '4室2厅', '2室2厅', '4室0厅', '5室1厅', '1室2厅', '3室3厅', '6室1厅',
'6室2厅',
       '5室2厅', '4室3厅', '5室3厅', '6室3厅', '2室3厅', '5室4厅', '4室4厅',
nan,
       '1房间1卫', '3房间1卫', '2房间1卫', '1房间0卫', '6室4厅', '4房间2卫',
'2房间0卫',
       '5房间3卫', '5房间2卫', '4室5厅', '7室3厅', '9室0厅', '8室4厅', '7室2
厅', '2房间2卫',
       '5室0厅', '3房间2卫', '4室9厅'], dtype=object)
```

In [97]:
```
# Checking the value classes and distribution

df['Floor Plan'].value_counts()
```

Out[97]:
```
2室1厅      32460
1室1厅      14321
3室1厅      10366
3室2厅       5593
2室2厅       4245
1室0厅       2620
4室2厅       1188
4室1厅        371
3室0厅        311
2室0厅        307
1室2厅        219
5室2厅        123
4室3厅         74
5室3厅         53
3室3厅         35
6室2厅         29
4室0厅         19
2房间1卫        16
5室1厅         15
6室3厅         12
3房间1卫         8
1房间1卫         6
5室4厅          5
4室4厅          3
8室4厅          3
2室3厅          3
2房间2卫         3
4房间2卫         3
5房间3卫         3
6室1厅          3
7室2厅          2
1房间0卫         2
5室0厅          1
2房间0卫         1
3房间2卫         1
7室3厅          1
5房间2卫         1
6室4厅          1
9室0厅          1
4室9厅          1
4室5厅          1
Name: Floor Plan, dtype: int64
```

In [98]:
```
# Recategorizing the classes

df.loc[df['Floor Plan'] == '2室1厅' , 'Floor Plan'] = '2B'
df.loc[df['Floor Plan'] == '3室1厅' , 'Floor Plan'] = '3B'
df.loc[df['Floor Plan'] == '3室2厅' , 'Floor Plan'] = '3B'
df.loc[df['Floor Plan'] == '1室1厅' , 'Floor Plan'] = '1B'
df.loc[df['Floor Plan'] == '2室0厅' , 'Floor Plan'] = '2B'
df.loc[df['Floor Plan'] == '3室0厅' , 'Floor Plan'] = '3B'
df.loc[df['Floor Plan'] == '1室0厅' , 'Floor Plan'] = '1B'
df.loc[df['Floor Plan'] == '4室1厅' , 'Floor Plan'] = '4B or above'

df.loc[df['Floor Plan'] == '4室2厅' , 'Floor Plan'] = '4B or above'
df.loc[df['Floor Plan'] == '2室2厅' , 'Floor Plan'] = '2B'
df.loc[df['Floor Plan'] == '4室0厅' , 'Floor Plan'] = '4B or above'
df.loc[df['Floor Plan'] == '5室1厅' , 'Floor Plan'] = '4B or above'
df.loc[df['Floor Plan'] == '1室2厅' , 'Floor Plan'] = '1B'
df.loc[df['Floor Plan'] == '3室3厅' , 'Floor Plan'] = '3B'
df.loc[df['Floor Plan'] == '6室1厅' , 'Floor Plan'] = '4B or above'
df.loc[df['Floor Plan'] == '6室2厅' , 'Floor Plan'] = '4B or above'

df.loc[df['Floor Plan'] == '5室2厅' , 'Floor Plan'] = '4B or above'
df.loc[df['Floor Plan'] == '4室3厅' , 'Floor Plan'] = '4B or above'
df.loc[df['Floor Plan'] == '5室3厅' , 'Floor Plan'] = '4B or above'
df.loc[df['Floor Plan'] == '6室3厅' , 'Floor Plan'] = '4B or above'
df.loc[df['Floor Plan'] == '2室3厅' , 'Floor Plan'] = '2B'
df.loc[df['Floor Plan'] == '5室4厅' , 'Floor Plan'] = '4B or above'
df.loc[df['Floor Plan'] == '4室4厅' , 'Floor Plan'] = '4B or above'
df.loc[df['Floor Plan'] == '0室0厅' , 'Floor Plan'] = '0B'

df.loc[df['Floor Plan'] == '6室4厅' , 'Floor Plan'] = '4B or above'
df.loc[df['Floor Plan'] == '4室5厅' , 'Floor Plan'] = '4B or above'
df.loc[df['Floor Plan'] == '7室3厅' , 'Floor Plan'] = '4B or above'
df.loc[df['Floor Plan'] == '9室0厅' , 'Floor Plan'] = '4B or above'
df.loc[df['Floor Plan'] == '8室4厅' , 'Floor Plan'] = '4B or above'
df.loc[df['Floor Plan'] == '7室2厅' , 'Floor Plan'] = '4B or above'
df.loc[df['Floor Plan'] == '5室0厅' , 'Floor Plan'] = '4B or above'
df.loc[df['Floor Plan'] == '4室9厅' , 'Floor Plan'] = '4B or above'

df.loc[df['Floor Plan'] == '1房间1卫', 'Floor Plan'] = '1B'
df.loc[df['Floor Plan'] == '3房间1卫', 'Floor Plan'] = '3B'
df.loc[df['Floor Plan'] == '2房间1卫', 'Floor Plan'] = '2B'
df.loc[df['Floor Plan'] == '1房间0卫', 'Floor Plan'] = '1B'
df.loc[df['Floor Plan'] == '4房间2卫', 'Floor Plan'] = '4B or above'
df.loc[df['Floor Plan'] == '2房间0卫', 'Floor Plan'] = '2B'
df.loc[df['Floor Plan'] == '5房间3卫', 'Floor Plan'] = '4B or above'
df.loc[df['Floor Plan'] == '5房间3卫', 'Floor Plan'] = '4B or above'
df.loc[df['Floor Plan'] == '5房间2卫', 'Floor Plan'] = '4B or above'
df.loc[df['Floor Plan'] == '2房间2卫', 'Floor Plan'] = '2B'
df.loc[df['Floor Plan'] == '3房间2卫', 'Floor Plan'] = '3B'
```

In [99]:
```
# Rechecking the value classes and distribution

df['Floor Plan'].value_counts()
```

Out[99]:
```
2B             37035
1B             17168
3B             16314
4B or above     1913
Name: Floor Plan, dtype: int64
```

In [100]:
```
# Rechecking missing data

df.isnull().sum()
```

Out[100]:
```
Unnamed: 0               0
Housing ID               0
Livable Area             0
Price/Square Meter       0
Price                    0
Transaction Date         0
Year Built            7159
Floor Plan               5
Facing Direction        71
Floor                 3122
Decoration            2940
dtype: int64
```

Dealing with missing data

In [101]:
```
# Filling the missing data with "pad" method

df['Year Built'] = df['Year Built'].fillna(method='pad',axis=0)
df.isnull().sum()
```

Out[101]:
```
Unnamed: 0               0
Housing ID               0
Livable Area             0
Price/Square Meter       0
Price                    0
Transaction Date         0
Year Built               0
Floor Plan               5
Facing Direction        71
Floor                 3122
Decoration            2940
dtype: int64
```

In [102]:
```
# Removing the oberservations with missing data

df.dropna( axis=0, how='any', thresh=None, subset=None, inplace=False)
```

Out[102]:

	Unnamed: 0	Housing ID	Livable Area	Price/Square Meter	Price	Transaction Date	Year Built	Floor Plan	Facing Direction	F
0	1	中关村南大街甲3号 2室1厅 56.3平米	56.3平米	38544.0	217.0	2015-05-08	1965	2B	South	
1	2	中关村南大街甲3号 2室1厅 55.1平米	55.1平米	33667.0	185.5	2012-04-10	1965	2B	South	
2	3	民族大学南路19号院 2室1厅 70平米	70平米	91429.0	640.0	2019-04-19	1993	2B	South	m
3	4	民族大学南路19号院 2室1厅 98.49平米	98.49平米	102752.0	1012.0	2019-04-07	2000	2B	South	
4	5	民族大学南路19号院 2室1厅 60.7平米	60.7平米	98847.0	600.0	2019-02-22	1991	2B	South	
...	...	...	...	...	...	...	...	...	...	
72430	72431	双榆树东里 1室1厅 46平米	46平米	31305.0	144.0	2012-04-15	1990	1B	South	
72431	72432	双榆树东里 2室1厅 53.1平米	53.1平米	33710.0	179.0	2012-03-04	1985	2B	South	
72432	72433	双榆树东里 3室1厅 70.2平米	70.2平米	30485.0	214.0	2012-02-26	1985	3B	South	
72433	72434	双榆树东里 3室1厅 71.9平米	71.9平米	33380.0	240.0	2012-02-26	1985	3B	South	m

	Unnamed: 0	Housing ID	Livable Area	Price/Square Meter	Price	Transaction Date	Year Built	Floor Plan	Facing Direction	F
72434	72435	双榆树东里 2室1厅 60.28 平米	60.28 平米	29861.0	180.0	2012-01-05	1990	2B	South	m

69300 rows × 11 columns

Simplifying the contents for a few columns

In [103]:
```
# Changing the content of Housing ID to community name

df['Housing ID'] = df['Housing ID'].str.split(pat = ' ')
for i in range(len(df['Housing ID'])):
    df['Housing ID'].iloc[i] = df['Housing ID'].iloc[i][0]


# Leaving only numbers for Livable Area and convert it into float

df['Livable Area'] = df['Livable Area'].astype('str').str[:-2]
df['Livable Area'] = df['Livable Area'].astype(float)

# Deleting useless columns such as 'Price' and 'Transaction_ID'

del df['Price']
del df['Unnamed: 0']
```

```
/Users/Reinhard/opt/anaconda3/lib/python3.8/site-packages/pandas/core/i
ndexing.py:671: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_with_indexer(indexer, value)
```

Renaming names for a few columns

In [104]:
```
df = df.rename(columns={"Transaction Date": "Date", "Housing ID": "Commu
nity Name"})

# getting a list of columns

cols = list(df)

# Moving the column of target variable to head of list using index, pop
 and insert

cols.insert(0, cols.pop(cols.index('Price/Square Meter')))
df = df[cols]
df.head()
```

Out[104]:

	Price/Square Meter	Community Name	Livable Area	Date	Year Built	Floor Plan	Facing Direction	Floor	Decoration
0	38544.0	中关村南大街甲3号	56.30	2015-05-08	1965	2B	South	low	simple
1	33667.0	中关村南大街甲3号	55.10	2012-04-10	1965	2B	South	first	other
2	91429.0	民族大学南路19号院	70.00	2019-04-19	1993	2B	South	middle	simple
3	102752.0	民族大学南路19号院	98.49	2019-04-07	2000	2B	South	high	simple
4	98847.0	民族大学南路19号院	60.70	2019-02-22	1991	2B	South	high	other

Setting Time Index

In [105]:
```
# Convert dtype of Transaction Date

df['Date'] = pd.to_datetime(df.Date)
```

In [106]:
```
# Setting Date as the Time Index

df = df.set_index(['Date'])
```

In [107]:
```
# Extracting Year, Month and Weekday from the time index

df['Year'] = df.index.year
df['Month'] = df.index.month
#df['weekday_name'] = df.index.day_name()

df.head()
```

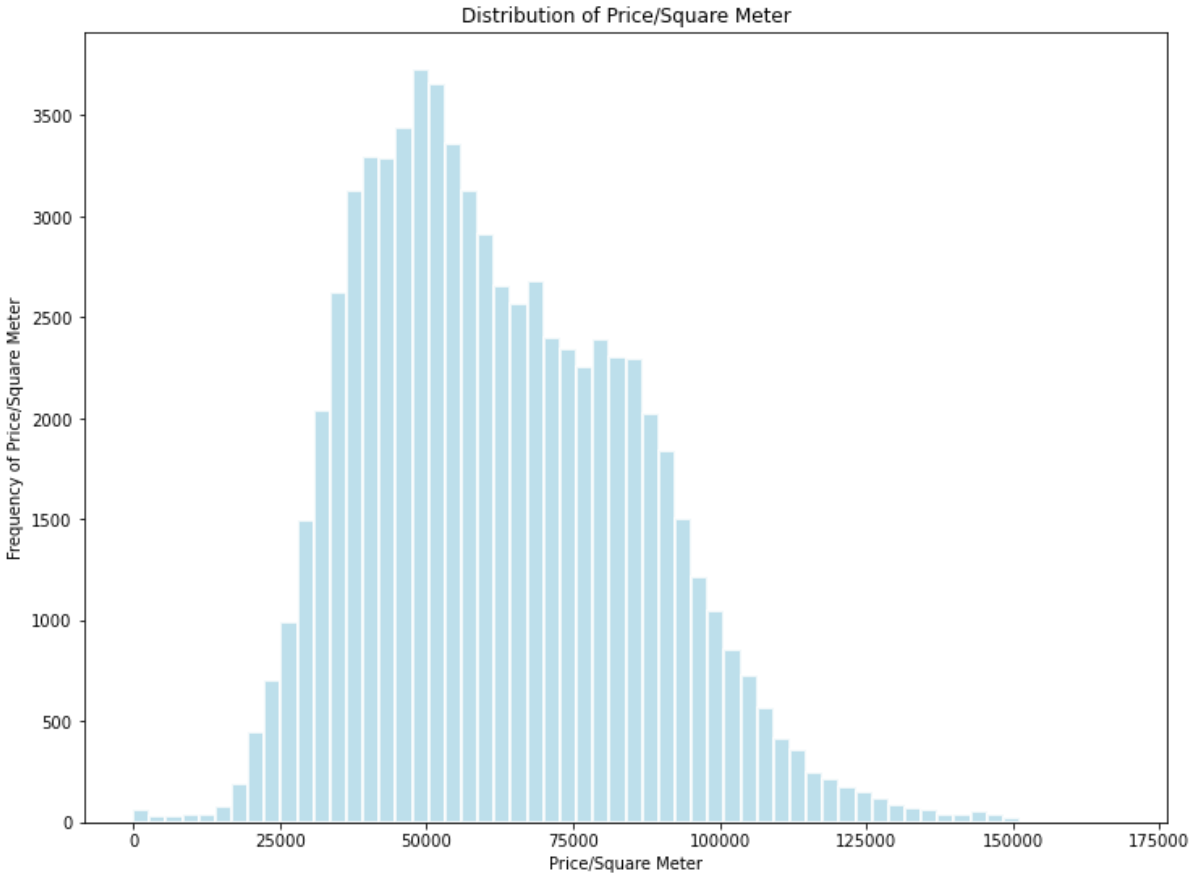
Out[107]:

Date	Price/Square Meter	Community Name	Livable Area	Year Built	Floor Plan	Facing Direction	Floor	Decoration	Year	Mont
2015-05-08	38544.0	中关村南大街甲3号	56.30	1965	2B	South	low	simple	2015	
2012-04-10	33667.0	中关村南大街甲3号	55.10	1965	2B	South	first	other	2012	
2019-04-19	91429.0	民族大学南路19号院	70.00	1993	2B	South	middle	simple	2019	
2019-04-07	102752.0	民族大学南路19号院	98.49	2000	2B	South	high	simple	2019	
2019-02-22	98847.0	民族大学南路19号院	60.70	1991	2B	South	high	other	2019	

Data Plotting and Interpretation

Checking the Distribution of the Target Variable

In [108]:
```
#Setting figure size

plt.figure(figsize=(12,9))

#Plotting the histogram of Price/Square Meter

plt.hist(df['Price/Square Meter'], bins=60, color='lightblue', alpha=0.8
, label='Value', edgecolor='white', linewidth=2)
plt.xlabel('Price/Square Meter')
plt.ylabel('Frequency of Price/Square Meter')
plt.title('Distribution of Price/Square Meter')
plt.show()
```



Key Takeways:

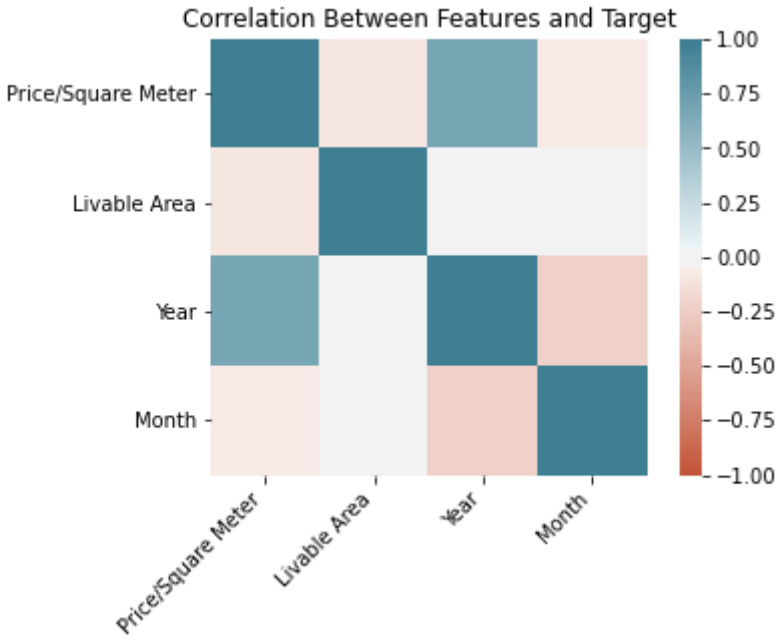
1. The target variable is not normally distributed.

2. The target variable is right skewed.

Plotting Heatmap

In [109]:
```
#Creating correlation matrix

corr = df.corr()

#Creating heatmap

hmp = sns.heatmap(
    corr,
    vmin=-1, vmax=1, center=0,
    cmap=sns.diverging_palette(20, 220, n=200),
    square=True)

hmp.set_xticklabels(
    hmp.get_xticklabels(),
    rotation=45,
    horizontalalignment='right');
plt.title('Correlation Between Features and Target')
```

Out[109]: Text(0.5, 1.0, 'Correlation Between Features and Target')



Key Takeways:

1. The target variable seems negatively correlated with livable area. This is indeed the case in Beijing, which means smaller the houses are coupled with higher unit housing prices.

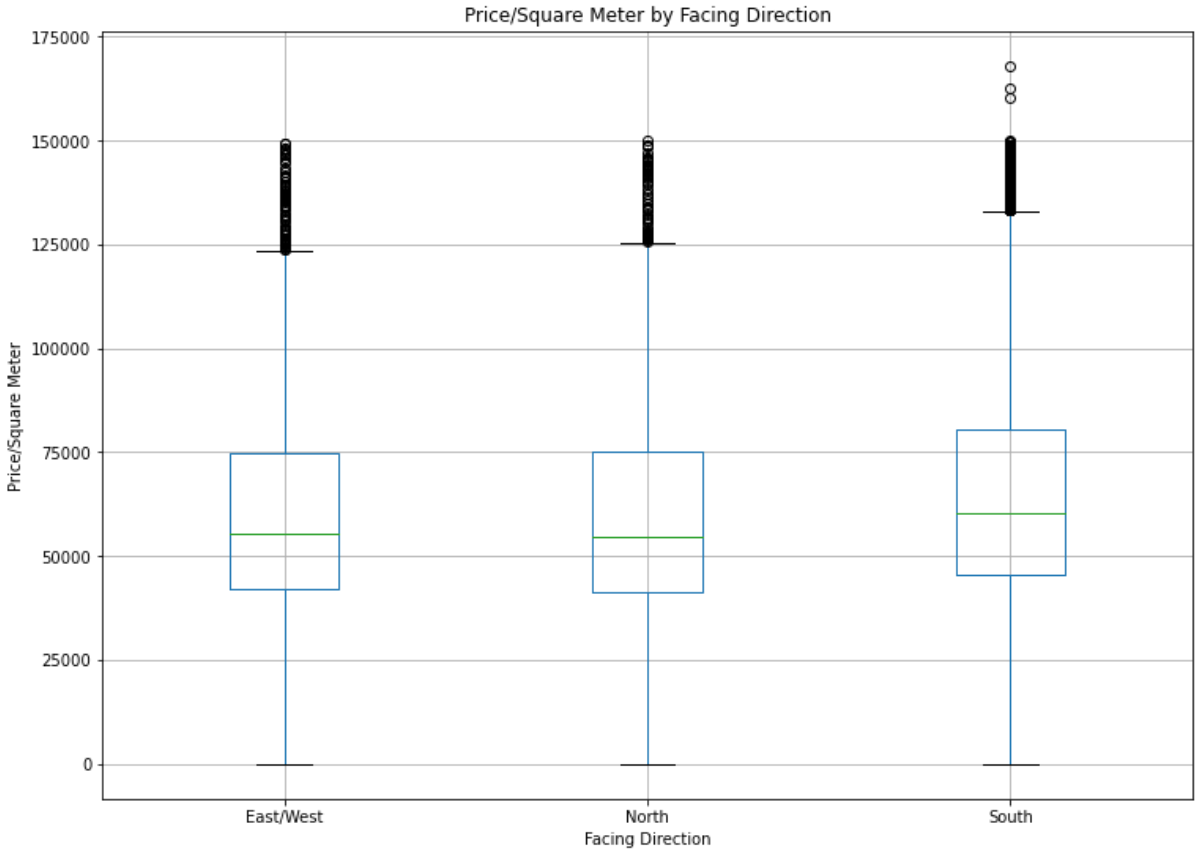
2. The correlations between target variable and time metrics such as year and month are not so useful due to autocorrelation.

3. The heatmap cannot include the correalations between the target variable and the categorical variables such as floor plan, decoration and facing direction.

Checking the relationships between the target variable and different facing directions

In [110]:
```
#Create boxplot

plt.figure()
df.boxplot(column= 'Price/Square Meter', by='Facing Direction', rot = 0,
figsize = (12,9))
plt.suptitle('')
plt.title('Price/Square Meter by Facing Direction')
plt.xlabel('Facing Direction')
plt.ylabel('Price/Square Meter')
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



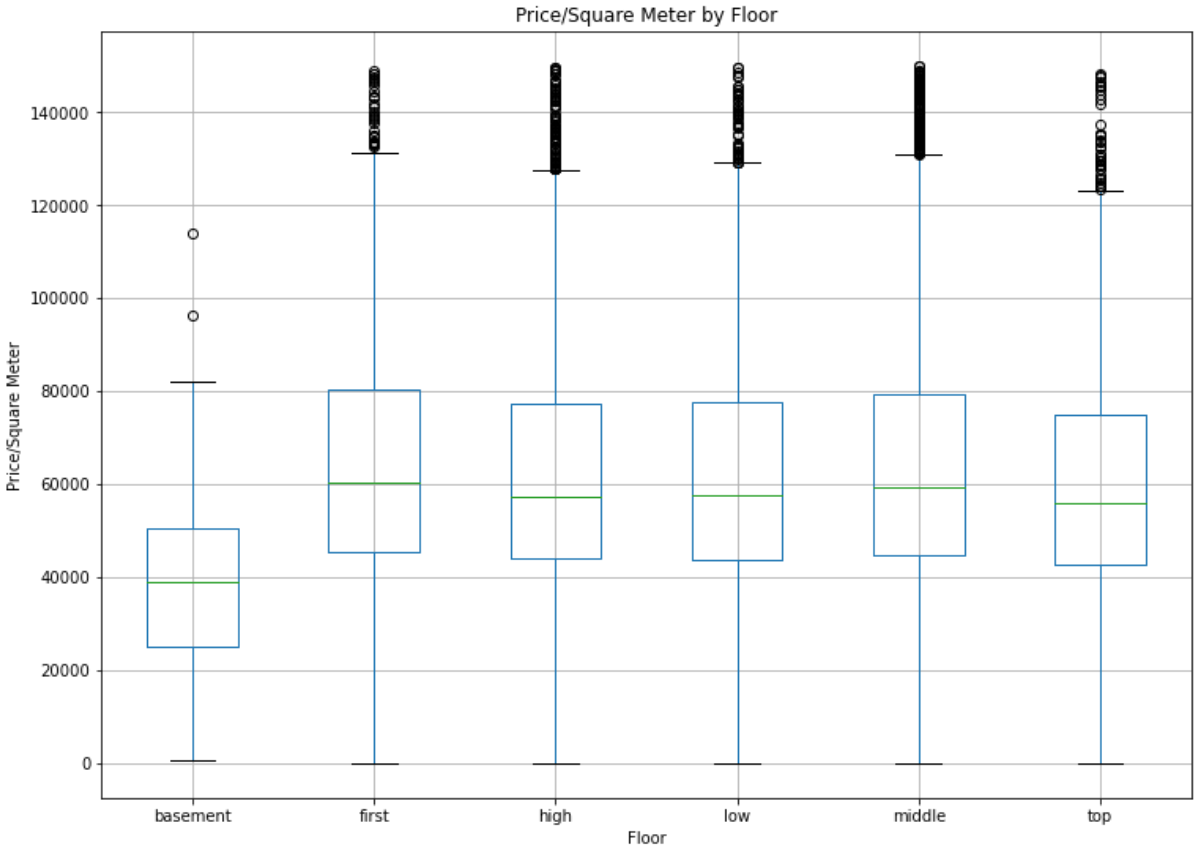
Key Takeways:

We see the houses facing south are with highest housing prices. This is in fact reflecting a strong preference for sunlight among people in the northern China.

Checking the relationships between the target variable and different floor level

In [111]:
```
#Creating boxplot

plt.figure()
df.boxplot(column= 'Price/Square Meter', by='Floor', rot = 0, figsize =
(12,9))
plt.suptitle('')
plt.title('Price/Square Meter by Floor')
plt.xlabel('Floor')
plt.ylabel('Price/Square Meter')
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



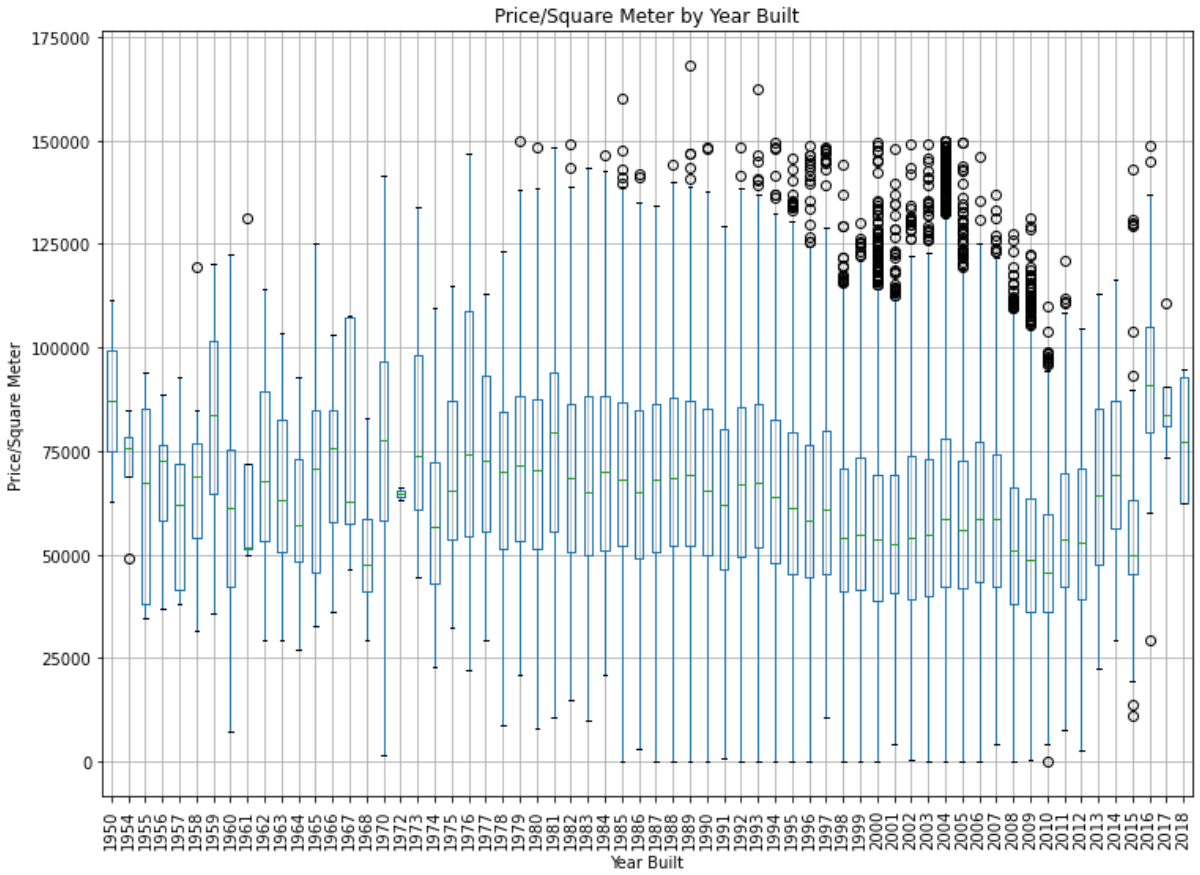
Key Takeways:

We see the houses on the first floor and the middle-level floor are with highest housing prices. This is actually revealing two preferences among people living in Beijing. First, families with the elder have strong preference for the first floor in order to provide convenience for the elder. Second, People who living in Beijing also value sunlight but not height for choosing their residences.

Checking the relationships between the target variable and year built (housing age)

In [112]:
```
#Creating boxplot

plt.figure()
df.boxplot(column= 'Price/Square Meter', by='Year Built', rot = 90, figsize = (12,9))
plt.suptitle('')
plt.title('Price/Square Meter by Year Built')
plt.xlabel('Year Built')
plt.ylabel('Price/Square Meter')
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



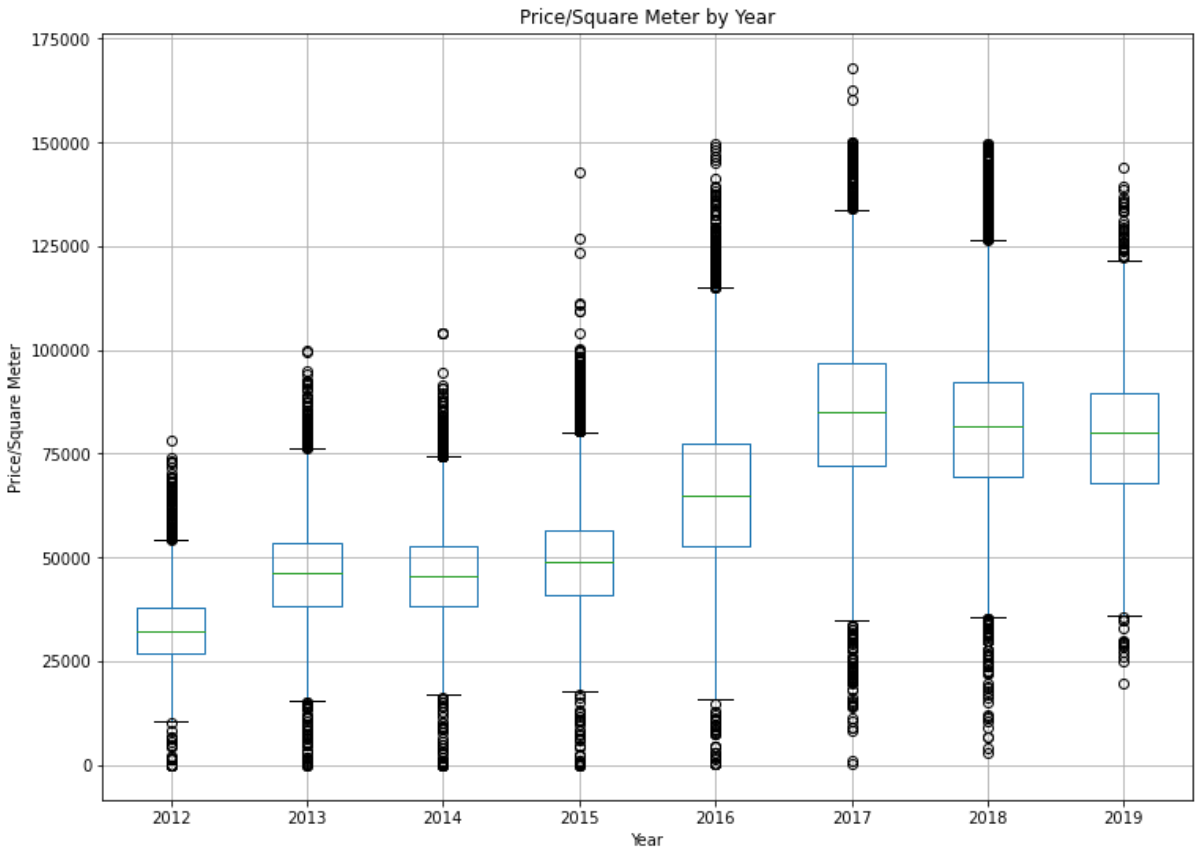
Key Takeways:

We see that there are old houses with higher housing prices on the left-hand side compared to their counterparts. There are two possible interpretations. The first is they are within good school districts which contribute a lot to their values. Second, the old buildings are mainly locating in the central areas of the city with major amenities of the cities which could also add values to their housing values.

Checking the variation of target variable along with years

In [113]:
```
#Create boxplot

plt.figure()
df.boxplot(column= 'Price/Square Meter', by='Year', rot = 0, figsize = (
12,9))
plt.suptitle('')
plt.title('Price/Square Meter by Year')
plt.xlabel('Year')
plt.ylabel('Price/Square Meter')
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



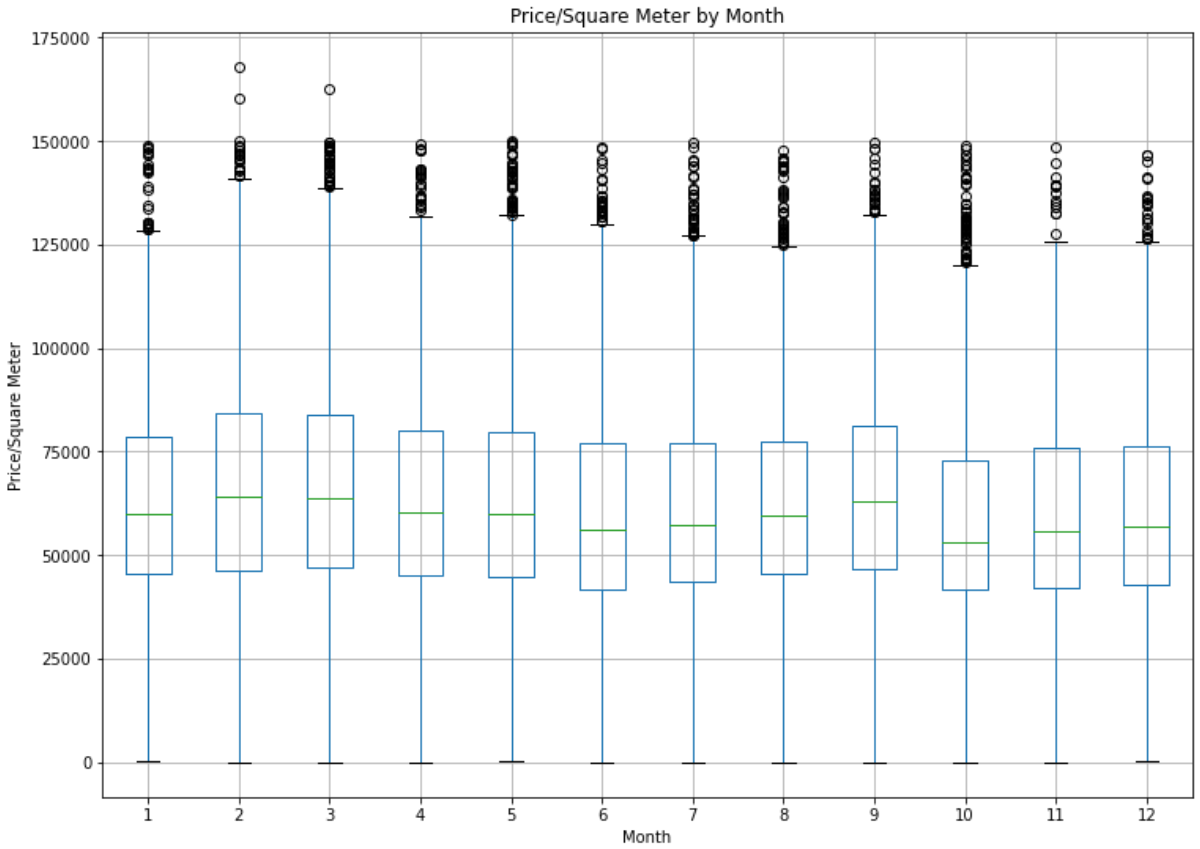
Key Takeways:

We see that there was an increasing trend for housing prices from 2012 to 2019 followed by a downward movements. There are two possible interpretations. The expansionary monetary policy is the main cause for housing prices booming during those years. However, in order to mitigate the risk of the collapse of housing market, the central bank collected liquidity startging from at the last quarter of 2016. Meanwhile, the local government conducted restrictions on housing purchase. These two policies cooled down the housing market after 2017.

Checking the variation of target variable across a year

In [114]:
```
#Create boxplot

plt.figure()
df.boxplot(column= 'Price/Square Meter', by='Month', rot = 0, figsize =
(12,9))
plt.suptitle('')
plt.title('Price/Square Meter by Month')
plt.xlabel('Month')
plt.ylabel('Price/Square Meter')
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



Key Takeways:

We see that housing prices were higher in Febrary and March compared with their peers. This is fact-reavling since the Luna New Year usually locating at end of Janary or the start of Febrary in each year, meaning people also receive their year-end bonus at that time creating purchasing momentum for housing during that time period.

Summary

1. The target variable is right skewed.

2. The target variable seems negatively correlated with livable area, which means the smaller the houses the higher unit housing prices.

3. The fact that the houses facing south are with highest housing prices potentially reflects a strong preference for sunlight among people in Beijing.

4. Families with the elder have strong preference for the first floor in order to provide convenience for the elder may cause the houses on first floor with higher prices relative to their peers.

5. The Old houses are usually within good school districts may partially explain the higher prices relative to their counterparts.

6. The expansionary monetary policy followed by collecting liquidty and housing purchase restrictions may explain the increasing trend of housing prices followed by a downward movement.

7. The fact year-end bonus granted around the Lunar New Year which locating in Janary or Febrary causing purchasing momentum may explain the higher housing prices in Febrary and March compared with other months.