Exercise #1 - Pandas

1. Init from dict

In [2]:

```
import pandas as pd
```

In [3]:

```
#1.     Create a new dictionary, name it your firstname where firstname _fruits is you
dongheun_fruits = {}
```

In [4]:

```
#2.     Add four items to the dictionary with names of your favorite fruits as keys ar
dongheun_fruits['orange'] = 'orange';
dongheun_fruits['apple'] = 'red';
dongheun_fruits['banana'] = 'yellow';
dongheun_fruits['grapes'] = 'purple';

print(dongheun_fruits)
```

```
{'orange': 'orange', 'apple': 'red', 'banana': 'yellow', 'grapes': 'purple'}
```

In [5]:

```
#3.     Convert the dictionary into a pandas series named firstname_f.
dongheun_f = pd.Series(dongheun_fruits)
```

In [6]:

```
#4.     Print out the second and third items.
print("Second and third items from the series:")
print(dongheun_f[1:3])
```

```
Second and third items from the series:
apple        red
banana    yellow
dtype: object
```

In [7]:

```
#5.     Create a sub series named firstname_f2 containing the second and third items.
dongheun_f2 = dongheun_f[1:3]
```

In [8]:

```
#6.     Printout from the sub series the last item using iloc.
print(dongheun_f2.iloc[-1])
```

```
yellow
```

2. Handling time

In [9]:

```
import matplotlib.pyplot as plt
```

In [10]:

```
#1.     Create a list containing four  rainfall amounts  of values 10, 23,24,30 name t
dongheun_amounts = [10, 23, 24, 30]
```

In [11]:

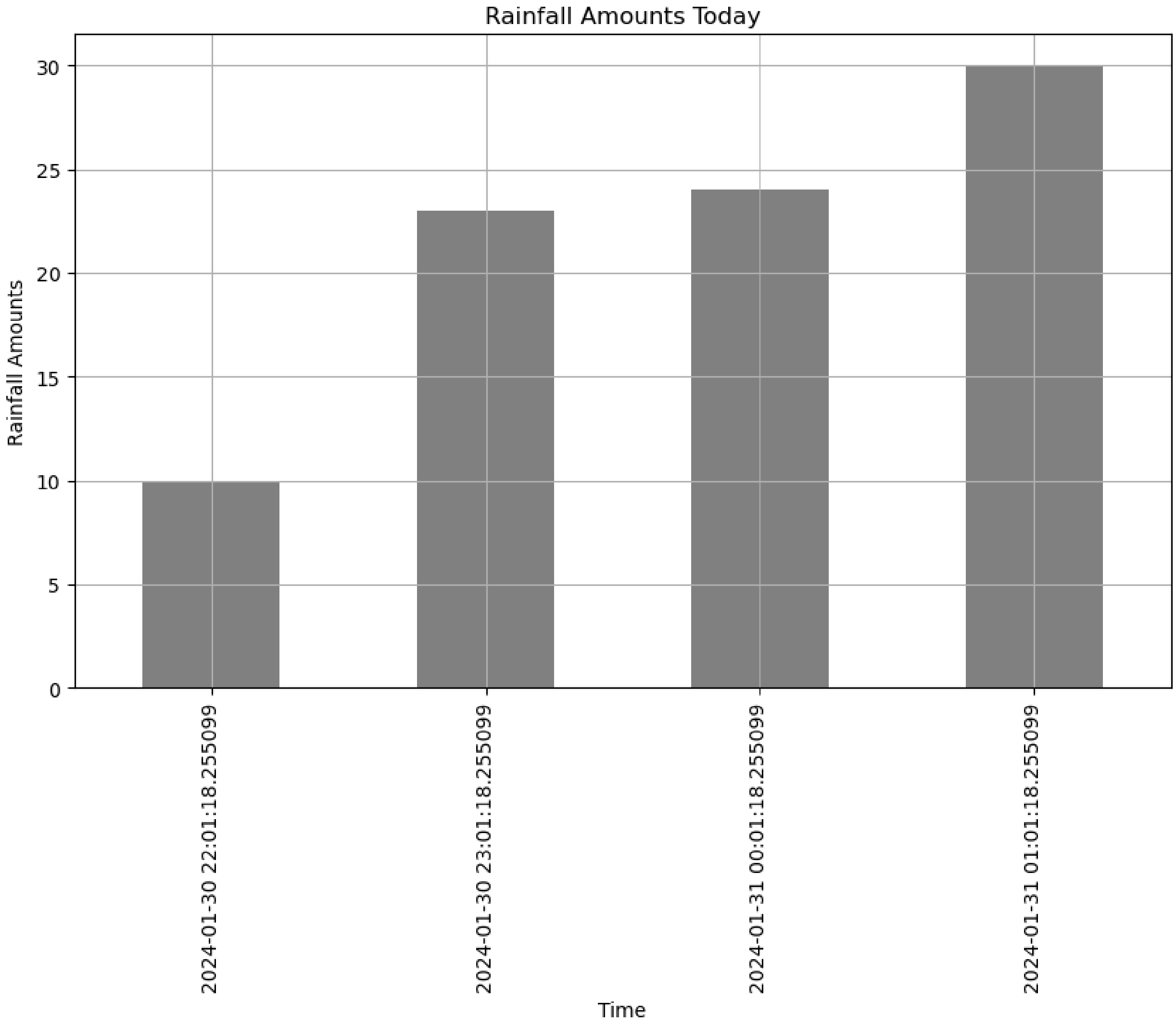
```
#2.     Using pandas create a date_range for todays date/time (you can set any time) w
date_range = pd.date_range(start=pd.Timestamp.today(), periods=4, freq='H')
```

In [12]:

```
#3.     Create a series that combines both the list and date range name it firstname_r
dongheun_rainfall_amounts_today = pd.Series(dongheun_amounts, index=date_range)
```

In [13]:

```
#4.     Plot as bar chart.
plt.figure(figsize=(10, 6))
dongheun_rainfall_amounts_today.plot(kind='bar', color='gray')
plt.title('Rainfall Amounts Today')
plt.xlabel('Time')
plt.ylabel('Rainfall Amounts')
plt.grid(True)
plt.show()
```



3. Pandas Multi - indexing

dongheun_d5 = dataframe

In [14]:

```
data = {
    'private_name': ['John', 'Anna', 'Peter', 'alice'],
    'private_age': [28, 22, 34, 40],
    'public_city': ['New York', 'Los Angeles', 'Chicago', 'Houston'],
    'public_job': ['Engineer', 'Doctor', 'Teacher', 'Nurse']
}

d5 = pd.DataFrame(data)
```

In [15]:

```python
#Make a copy of the dataframe d5 and name it fristname_d5, carryout the following:
dongheun_d5 = d5.copy()
```

In [16]:

```python
#1.     print out a dataframe containing all “private” columns
private_column = dongheun_d5.filter(like='private')
print(private_column)
```

```
  private_name  private_age
0         John           28
1         Anna           22
2        Peter           34
3        alice           40
```

In [17]:

```python
#2.     Swap the columns and rows (hint: look at transpose)
dongheun_d5_transposed = dongheun_d5.transpose()
print(dongheun_d5_transposed)
```

```
                     0            1        2        3
private_name      John         Anna    Peter    alice
private_age         28           22       34       40
public_city   New York  Los Angeles  Chicago  Houston
public_job    Engineer       Doctor  Teacher    Nurse
```

4. Querying

In [18]:

```python
alice_data = dongheun_d5.query("private_name == 'alice'")
alice_data
```

Out[18]:

	private_name	private_age	public_city	public_job
3	alice	40	Houston	Nurse

5. Operations on dataframes

In [25]:

```python
students = ['Alice', 'Bob', 'Carol', 'David']
months = ['April', 'May', 'June', 'July']
grades_data = {
    'April': [45, 70, 35, 90],
    'May': [55, 10, 90, 15],
    'June': [65, 85, 75, 60],
    'July': [70, 25, 95, 70]
}
dongheun_grades = pd.DataFrame(grades_data, index=students, columns=months)
print(dongheun_grades)
```

```
       April  May  June  July
Alice     45   55    65    70
Bob       70   10    85    25
Carol     35   90    75    95
David     90   15    60    70
```

In [26]:

```python
#1.     Print out the average for the month of April
ave_april = dongheun_grades['April'].mean()
print("Average for the month of April:", ave_april)
```

```
Average for the month of April: 60.0
```

In [27]:

```
#2.     Adjust all the grades by 2% (i.e. increase)
dongheun_grades *= 1.02
print("Adjusted grades (increased by 2%):")
print(dongheun_grades)
```

```
Adjusted grades (increased by 2%):
       April   May  June  July
Alice   45.9  56.1  66.3  71.4
Bob     71.4  10.2  86.7  25.5
Carol   35.7  91.8  76.5  96.9
David   91.8  15.3  61.2  71.4
```

In [29]:

```
#3.     Printout the grades for the month of may that are higher than 50%
may_grades_above_50 = dongheun_grades[dongheun_grades['May'] > 50]['May']
print("Grades for the month of May that are higher than 50%:")
print(may_grades_above_50)
```

```
Grades for the month of May that are higher than 50%:
Alice    56.1
Carol    91.8
Name: May, dtype: float64
```

In [30]:

```
#4.     Group the failing students i.e. the students with average over four month belc
average_grades = dongheun_grades.mean(axis=1)
failed_students = average_grades[average_grades < 50]
print("Failed Students (average over four months below 50%)")
print(failed_students)
```

```
Failed Students (average over four months below 50%)
Bob    48.45
dtype: float64
```

Exercise #2 - Numpy

1. function name

In [33]:

```
import numpy as np

#Add a cell to create a function and name it  my_function_firstname, where firstname i

#Let the function return an integer value stored in one byte i.e. ‘int8’ of (4x)*(3y).
def my_function_dongheun(x, y, z):
    return (4 * x) * (3 * y)

#Use np.fromfunction() to generate  three elements each are two by six using the  my_f
result_task1 = np.fromfunction(my_function_dongheun, (3,2,6), dtype = int)
result_task1 = result_task1.astype(np.int8)

result_task1
```

Out[33]:

```
array([[[ 0,  0,  0,  0,  0,  0],
        [ 0,  0,  0,  0,  0,  0]],

       [[ 0,  0,  0,  0,  0,  0],
        [12, 12, 12, 12, 12, 12]],

       [[ 0,  0,  0,  0,  0,  0],
        [24, 24, 24, 24, 24, 24]]], dtype=int8)
```

2.Multi-dimensinal arrays

In [34]:

```
#Inspect the code under this section copy it, add a cell to extract values 16,17,18
multi_dim_array = np.array([[[10,11,12,13,14,15],
                            [16,17,18,19,20,21]],
                            [[22,23,24,25,26,27],
                            [28,29,30,31,32,33]]])

extracted_values = multi_dim_array[multi_dim_array == 16]
extracted_values = np.append(extracted_values, multi_dim_array[multi_dim_array == 17])
extracted_values = np.append(extracted_values, multi_dim_array[multi_dim_array == 18])

extracted_values
```

Out[34]:

```
array([16, 17, 18])
```

3.Iterating

In [35]:

```
#Inspect the code under this section copy it, then add a cell to iterate over c
#and print the Boolean values for items equivalent to zeros.
c_array = np.array([[0, 1, 2], [3, 0, 4], [5, 6, 0]])

print("Boolean values for items equivalent to zeros:")
for row in c_array:
    for item in row:
        print(item == 0)
```

```
Boolean values for items equivalent to zeros:
True
False
False
False
True
False
False
False
True
```

4.vstack

In [37]:

```
#Inspect the code under this section copy it, then add a cell to create a variable
#name it q5_firstname where firstname is your firstname and vertically stack q1
#and q2 and print the output.

q1 = np.array([[1, 2, 3], [4, 5, 6]])
q2 = np.array([[7, 8, 9], [10, 11, 12]])

q5_dongheun = np.vstack((q1, q2))
q5_dongheun
```

Out[37]:

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

5.concatenate

In [38]:

```
#Inspect the code under this section copy it, then add a cell to create a
#variable name it q8_firstname where firstname is your firstname , concatenate
#q1 and q3 and print the results.
q3 = np.array([[13, 14, 15], [16, 17, 18]])

q8_dongheun = np.concatenate((q1, q3))

q1, q3, q8_dongheun
```

Out[38]:

```
(array([[1, 2, 3],
        [4, 5, 6]]),
 array([[13, 14, 15],
        [16, 17, 18]]),
 array([[ 1,  2,  3],
        [ 4,  5,  6],
        [13, 14, 15],
        [16, 17, 18]]))
```

6. Transpose

In [41]:

```
#Inspect the code under this section copy it, then add a cell and create a
#variable named t_firstname where firstname is your name, let the variable
#hold any ndaray size 2 by 7 with zero values, print the result then transpose
#and print the result.

t_dongheun = np.zeros((2, 7))
print("Original array (2x7):")
print(t_dongheun)

t_dongheun_transposed = t_dongheun.transpose()
print("\nTransposed array (7x2):")
print(t_dongheun_transposed)
```

```
Original array (2x7):
[[0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]]

Transposed array (7x2):
[[0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]]
```

7.Matrix multiplication

In [43]:

```
#Inspect the code under this section copy it, then  add a cell to create 2nd arrays
#name the first a1 and the second a2. Both arrays should contain numbers in the
#range 0 to 7, inclusive . Print a1 and a2. Reshape a1 to a 2 by 4. Reshape a2
```

```
#to a 4 by 2. Create a new variable a3 _first name where firstname is your first
#name which holds the dot product  of a1 and a2 name it a3 and print the output
#of a3_firstname, then the shape of a3_first name.
a1 = np.arange(8).reshape(2, 4)
a2 = np.arange(8).reshape(4, 2)
a3_dongheun = np.dot(a1, a2)

print("\na1:")
print(a1)
print("\na2:")
print(a2)
print("\na3_dongheun (dot product):")
print(a3_dongheun)
print("\nShape of a3_dongheun:")
print(a3_dongheun.shape)
```

```
a1:
[[0 1 2 3]
 [4 5 6 7]]

a2:
[[0 1]
 [2 3]
 [4 5]
 [6 7]]

a3_dongheun (dot product):
[[28 34]
 [76 98]]

Shape of a3_dongheun:
(2, 2)
```

8.Matrix inverse and pseudo-inverse

In [53]:

```
#Add a cell to create a new 4 by 4 ndaray with values between 0 and 15, name the
#variable that holds the array your first name, print the array and the inverse
#of the array.
import numpy.linalg as linalg
dongheun_matrix = np.random.rand(4,4)
dongheun_matrix_inverse = np.linalg.inv(dongheun_matrix)

print("\ndongheun_matrix (4x4):")
print(dongheun_matrix)
print("\nInverse of dongheun_matrix:")
print(dongheun_matrix_inverse)
```

```
dongheun_matrix (4x4):
[[0.64136465 0.00132161 0.44936247 0.14003732]
 [0.22205295 0.49398358 0.8603945  0.57545264]
 [0.74694078 0.30126964 0.31165628 0.20060802]
 [0.07870461 0.9381349  0.5488288  0.70785214]]

Inverse of dongheun_matrix:
[[  3.1566103   -2.27686796  -0.84937155   1.46722337]
 [-15.36161665   8.3225094   11.45082455  -6.9719896 ]
 [-11.16869227   8.68132461   7.7515507   -7.04480569]
 [ 28.66775114 -17.50789021 -21.09175374  15.95189333]]
```

9.Identity matrix

In [54]:

```
#Add a cell to create a 4 by 4 identity array.
identity_matrix = np.eye(4)
print(identity_matrix)
```

```
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
```

10.Determinant

In [56]:

```
#Add a cell to create a 3 by 3 matrix with values generated randomly then
#printout the determinant of the matrix.
random_matrix = np.random.rand(3,3)
determinant = np.linalg.det(random_matrix)

print("\nRandom 3x3 Matrix:")
print(random_matrix)
print("\nDeterminant of the Matrix:")
print(determinant)
```

```
Random 3x3 Matrix:
[[0.55713269 0.35888985 0.76968487]
 [0.27088665 0.0118426  0.67108737]
 [0.47146034 0.2930713  0.99326639]]

Determinant of the Matrix:
-0.02922850142863991
```

11.Eigenvalues and eigenvectors

In [58]:

```
#Add a cell to create a 4 by 4 matrix with values generated randomly, assign the
#matrix to a variable named e_firstname. Printout the Eigenvalue and eigenvectors
#of the matrix.

e_dongheun = np.random.rand(4,4)
eigenvalues, eigenvectors = np.linalg.eig(e_dongheun)

print("e_dongheun (4x4 Random Matrix):")
print(e_dongheun)
print("Eigenvalues:")
print(eigenvalues)
print("Eigenvectors:")
print(eigenvectors)
```

```
e_dongheun (4x4 Random Matrix):
[[0.06657611 0.65783264 0.15114237 0.1275392 ]
 [0.38536234 0.19105431 0.55095612 0.47663628]
 [0.1872434  0.21449032 0.2461605  0.69902933]
 [0.30803185 0.10880256 0.6987738  0.58771316]]
Eigenvalues:
[ 1.46117947  0.24674265 -0.40870431 -0.20771373]
Eigenvectors:
[[-0.36086362 -0.83486157 -0.59213493  0.75610126]
 [-0.53990709 -0.35299449  0.44666957 -0.2310236 ]
 [-0.48695809  0.26090649 -0.47859081 -0.56527291]
 [-0.58408013  0.33215788  0.46990781  0.23538377]]
```

12. Solving a system of linear scalar equations

In [59]:

```
#Add a cell to solve the following linear equations:
#2x+4y+z =12
#3x+8y+2z =16
#X+2y+3z = 3
#Check the results using the allcolse method.

coefficients = np.array([[2,4,1], [3,8,2], [1,2,3]])
constants = np.array([12,16,3])
solution = np.linalg.solve(coefficients, constants)

print("Solution to the linear equation:\n")
print(solution)

check = np.allclose(np.dot(coefficients, solution), constants)
print("\nCheck with allclose method:", check)
```

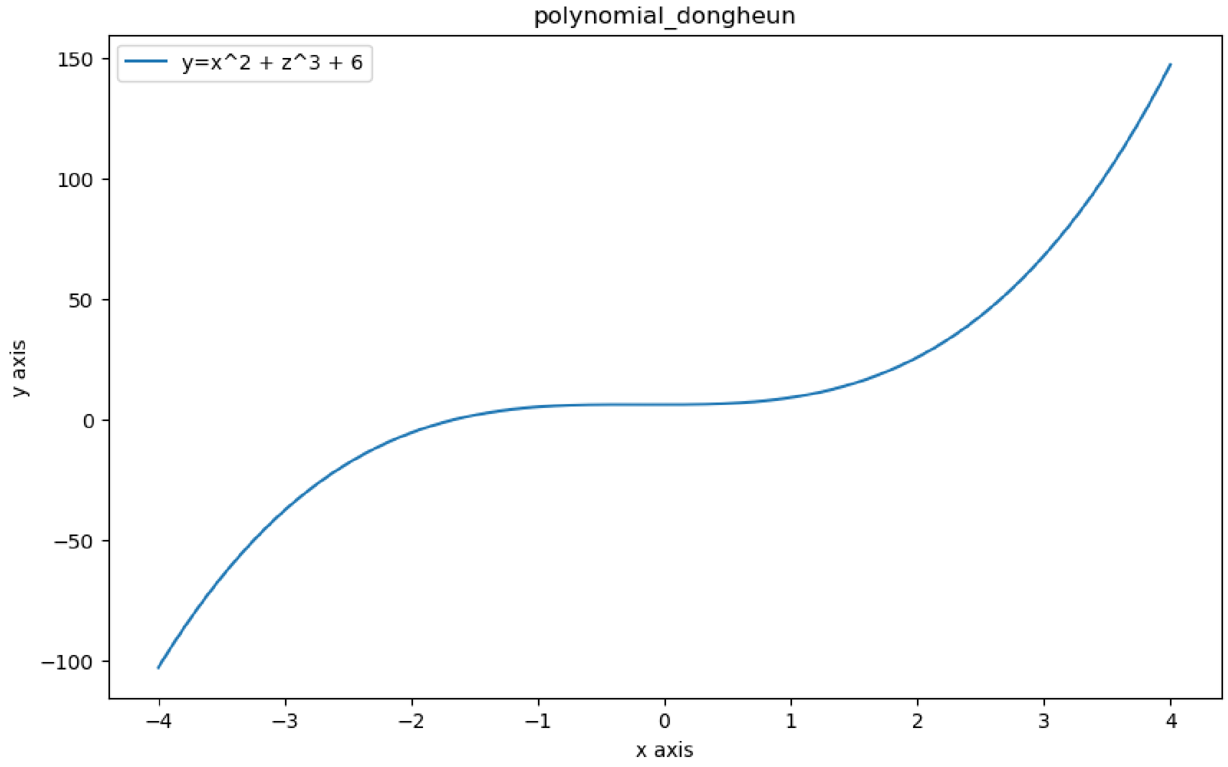
```
Solution to the linear equation:

[ 8.  -0.7 -1.2]

Check with allclose method: True
```

Exercise #3 Matplotlib

1.Plotting your first graph

In [61]:

```
import matplotlib.pyplot as plt

x = np.linspace(-4, 4, 1000)
z = np.linspace(-5, 5, 1000)
y = x**2 + z**3 + 6

plt.figure(figsize = (10, 6))
plt.plot(x, y, label='y=x^2 + z^3 + 6')
plt.title("polynomial_dongheun")
plt.xlabel("x axis")
plt.ylabel("y axis")
plt.legend()
plt.show()
```

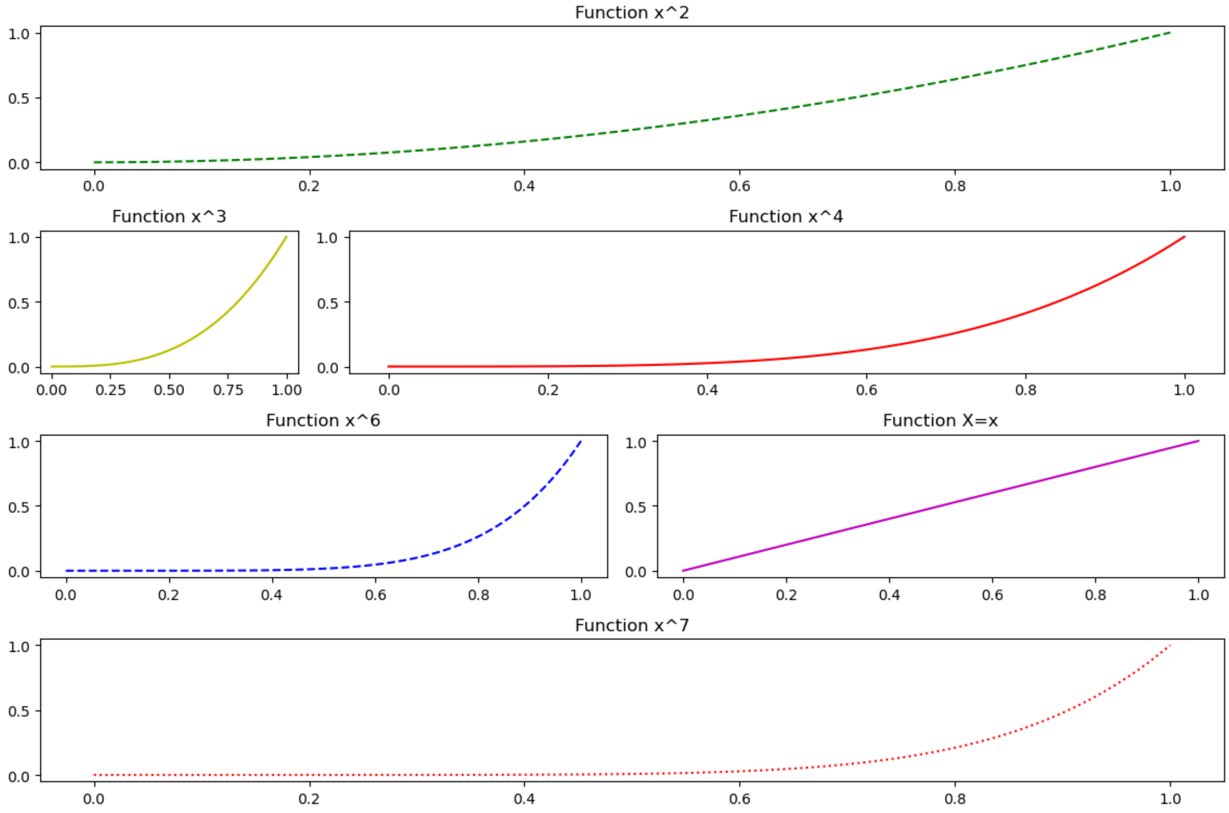


2.Subplots

In [62]:

```
# Creating the figure
fig = plt.figure(figsize=(12, 8))

# First row: function x^2 in a dashed green line
ax1 = plt.subplot2grid((4, 4), (0, 0), colspan=4)
x = np.linspace(0, 1, 100)
ax1.plot(x, x**2, 'g--')  # Dashed green line
ax1.set_title("Function x^2")

# Second row: function x^3 in yellow and x^4 in red spanning three columns
ax2 = plt.subplot2grid((4, 4), (1, 0))
ax2.plot(x, x**3, 'y')  # Yellow line
ax2.set_title("Function x^3")

ax3 = plt.subplot2grid((4, 4), (1, 1), colspan=3)
ax3.plot(x, x**4, 'r')  # Red line
ax3.set_title("Function x^4")

# Third row: X^6 in a dashed blue and X=x in magenta spanning two columns
ax4 = plt.subplot2grid((4, 4), (2, 0), colspan=2)
ax4.plot(x, x**6, 'b--')  # Dashed blue line
ax4.set_title("Function x^6")

ax5 = plt.subplot2grid((4, 4), (2, 2), colspan=2)
ax5.plot(x, x, 'm')  # Magenta line
ax5.set_title("Function X=x")

# Fourth row: function x^7 in dotted red spanning all columns
ax6 = plt.subplot2grid((4, 4), (3, 0), colspan=4)
ax6.plot(x, x**7, 'r:')  # Dotted red line
ax6.set_title("Function x^7")
```

```
# Adjust layout
plt.tight_layout()
plt.show()
```

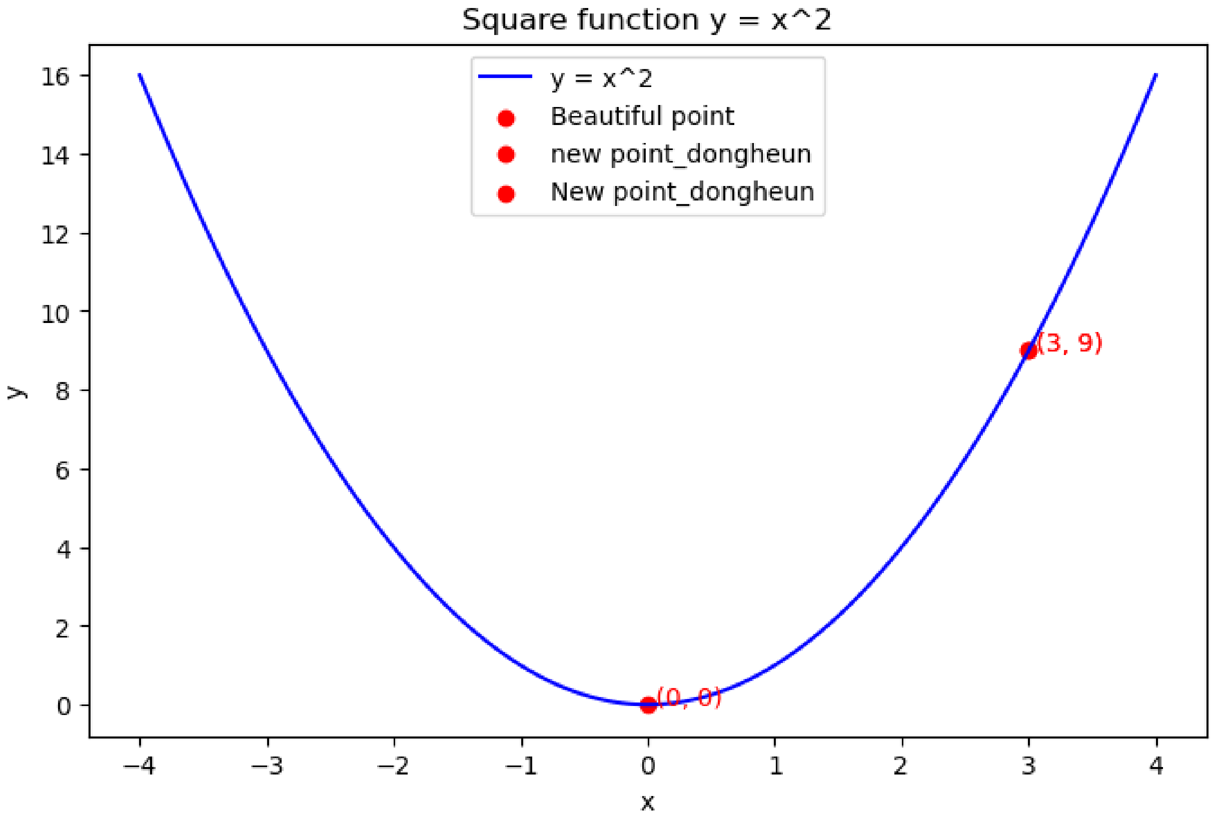


3. Drawing text

In [65]:

```
fig = plt.figure(figsize=(8, 5))
x = np.linspace(-4,4,1000)
y = x**2
plt.plot(x, y, label='y = x^2', color='blue')

beautiful_x = 0
beautiful_y = beautiful_x**2
plt.scatter(beautiful_x, beautiful_y, color='red', label='Beautiful point')
plt.text(beautiful_x, beautiful_y, f' ({beautiful_x}, {beautiful_y})', color='red')

new_point_x = 3
new_point_y = new_point_x**2
plt.scatter(new_point_x, new_point_y, color='red', label='new point_dongheun')
plt.text(new_point_x, new_point_y, f' ({new_point_x}, {new_point_y})', color='red')

new_point_x = 3
new_point_y = new_point_x**2
plt.scatter(new_point_x, new_point_y, color='red', label='New point_dongheun')
plt.text(new_point_x, new_point_y, f' ({new_point_x}, {new_point_y})', color='red')

plt.title('Square function y = x^2')
plt.xlabel('x')
plt.ylabel('y')

# Adding a legend
plt.legend()
```

```
# Showing the plot
plt.show()
```



```
In [ ]:
```