In this file, I answered some questions as Homework.

Question 1

In [1]:

```
import seaborn as sns
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Questions

1. Load the dataset as a dataframe titanic using the following code:

- import seaborn as sns
- titanic = sns.load_dataset('titanic')

2. Use the apply method to count the missing values for each column.
3. Drop the row if age=NaN.
4. Plot an histogram for age.
5. Divide age into 4 bins： [0,20], (20,40], (40,60],(60,80]. Then, use concat/merge to join it to the titanic DataFrame. (Hint: after merging, rename the column of age bins into age_bin)
6. Generate a dummy variables for age_bin. Drop one column of dummy variables.
7. Compute the number of passangers by category. Compute average age for survivors.

In [2]:

```
titanic = sns.load_dataset("titanic")
titanic
```

Out[2]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	S	Second	man	True	NaN	Southampton	no	True
887	1	1	female	19.0	0	0	30.0000	S	First	woman	False	B	Southampton	yes	True
888	0	3	female	NaN	1	2	23.4500	S	Third	woman	False	NaN	Southampton	no	False
889	1	1	male	26.0	0	0	30.0000	C	First	man	True	C	Cherbourg	yes	True
890	0	3	male	32.0	0	0	7.7500	Q	Third	man	True	NaN	Queenstown	no	True

891 rows × 15 columns

In [3]:

```
missing_values = titanic.isna().sum()
missing_values
```

Out[3]:

```
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
deck           688
embark_town      2
alive            0
alone            0
dtype: int64
```

In [4]:

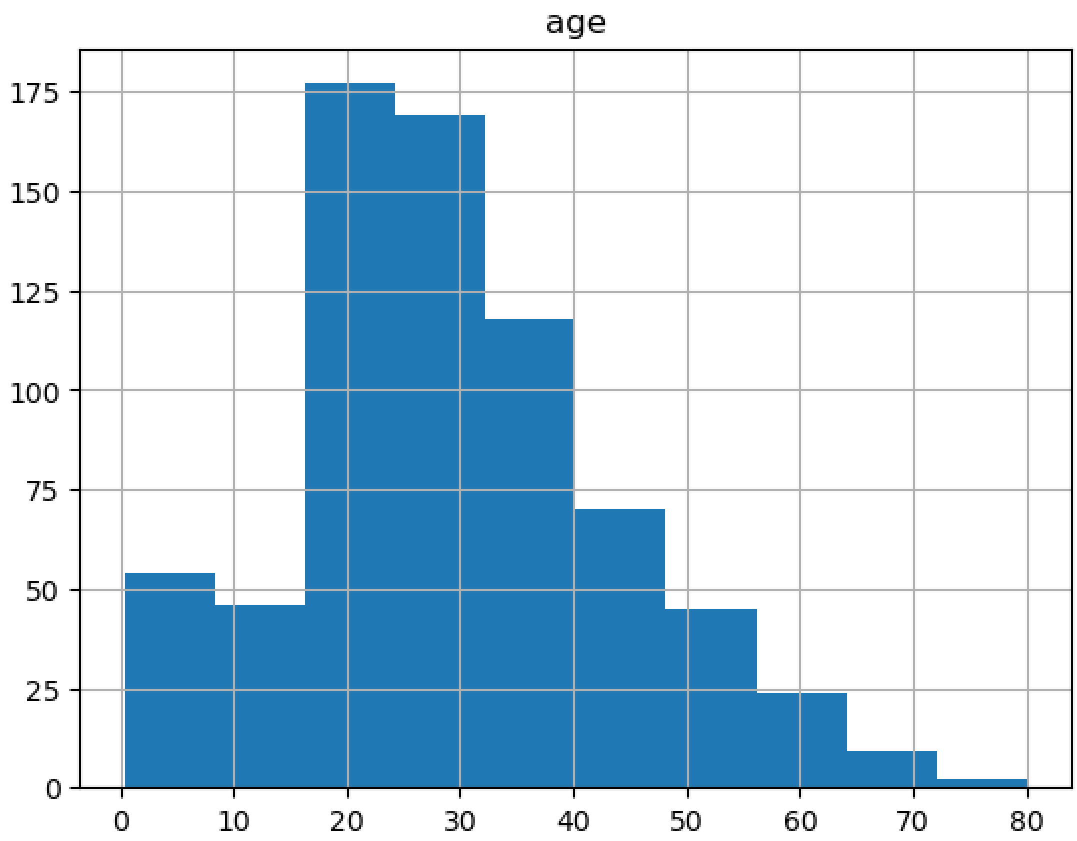
```
# Dropping the rows with Nan in the Age column
df = titanic.dropna(subset="age")
df
```

Out[4]:

```
     survived  pclass     sex   age  sibsp  parch     fare  embarked   class    who  adult_male  deck  embark_town  alive  alone
  0         0       3    male  22.0      1      0   7.2500         S   Third    man        True   NaN  Southampton     no  False
  1         1       1  female  38.0      1      0  71.2833         C   First  woman       False     C    Cherbourg    yes  False
  2         1       3  female  26.0      0      0   7.9250         S   Third  woman       False   NaN  Southampton    yes   True
  3         1       1  female  35.0      1      0  53.1000         S   First  woman       False     C  Southampton    yes  False
  4         0       3    male  35.0      0      0   8.0500         S   Third    man        True   NaN  Southampton     no   True
...       ...     ...     ...   ...    ...    ...      ...       ...     ...    ...         ...   ...          ...    ...    ...
885         0       3  female  39.0      0      5  29.1250         Q   Third  woman       False   NaN   Queenstown     no  False
886         0       2    male  27.0      0      0  13.0000         S  Second    man        True   NaN  Southampton     no   True
887         1       1  female  19.0      0      0  30.0000         S   First  woman       False     B  Southampton    yes   True
889         1       1    male  26.0      0      0  30.0000         C   First    man        True     C    Cherbourg    yes   True
890         0       3    male  32.0      0      0   7.7500         Q   Third    man        True   NaN   Queenstown     no   True

714 rows × 15 columns
```

```
In [5]: df.hist(column="age")
```

```
Out[5]: array([[<Axes: title={'center': 'age'}>]], dtype=object)
```



In [6]:

```
bins = [0, 20, 40, 60, 80]
labels = ["[0,20]", "[20,40]", "[40,60]", "[60,80]"]
# We use the cut function from Pandas in order to divide the ages into these bins
df["age_bin"] = pd.cut(df["age"], bins=bins, labels=labels, right= True)
```

```
C:\Users\henrik.knudsen\AppData\Local\Temp\ipykernel_16012\1896458843.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#return
ing-a-view-versus-a-copy)
  df["age_bin"] = pd.cut(df["age"], bins=bins, labels=labels, right= True)
```

In [7]:

```
age_bin_dummies = pd.get_dummies(df["age_bin"], prefix="age_bin")

# Dropping on column to avoid dummy variable trap
age_bin_dummies = age_bin_dummies.drop("age_bin_[0,20]", axis=1)

# Concatenate the dummy variables to the original dataframe
df = pd.concat([df, age_bin_dummies], axis=1)
df
```

Out[7]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone	a
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False	
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False	
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True	
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False	
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
885	0	3	female	39.0	0	5	29.1250	Q	Third	woman	False	NaN	Queenstown	no	False	
886	0	2	male	27.0	0	0	13.0000	S	Second	man	True	NaN	Southampton	no	True	
887	1	1	female	19.0	0	0	30.0000	S	First	woman	False	B	Southampton	yes	True	
889	1	1	male	26.0	0	0	30.0000	C	First	man	True	C	Cherbourg	yes	True	
890	0	3	male	32.0	0	0	7.7500	Q	Third	man	True	NaN	Queenstown	no	True	

714 rows × 19 columns

In [8]:

```
passenger_count_by_age_bin = df["age_bin"].value_counts().sort_index()
print(passenger_count_by_age_bin)
```

```
[0,20]     179
[20,40]    385
[40,60]    128
[60,80]     22
Name: age_bin, dtype: int64
```

In [9]:
```
average_age_survivors = df[df.alive=="yes"]["age"].mean()
print(f"The average age of survivors is: {round(average_age_survivors,2)}")
```

```
The average age of survivors is: 28.34
```

Question 2

Questions:

1. The class must define instance data coefficients. The coefficients are input to the class (as array).
2. Create methods associated to the class:

2.1 Evaluate the polynomial at x. For this part try to use 'enumerate()' in your loop to calculate p(x). Hint: use the built-in method call.

2.2 Differentiate the polynomial, replace the coefficients by those of its derivative p'. Note that you will need to eliminate the first coefficient.

3. Use the clas assuming the instance data is: a=(3, 5, 7, 9)
4. Evaluate P(x) for values x{1, 15}
5. Calculate the coefficents of p' and evaluate it at x=10

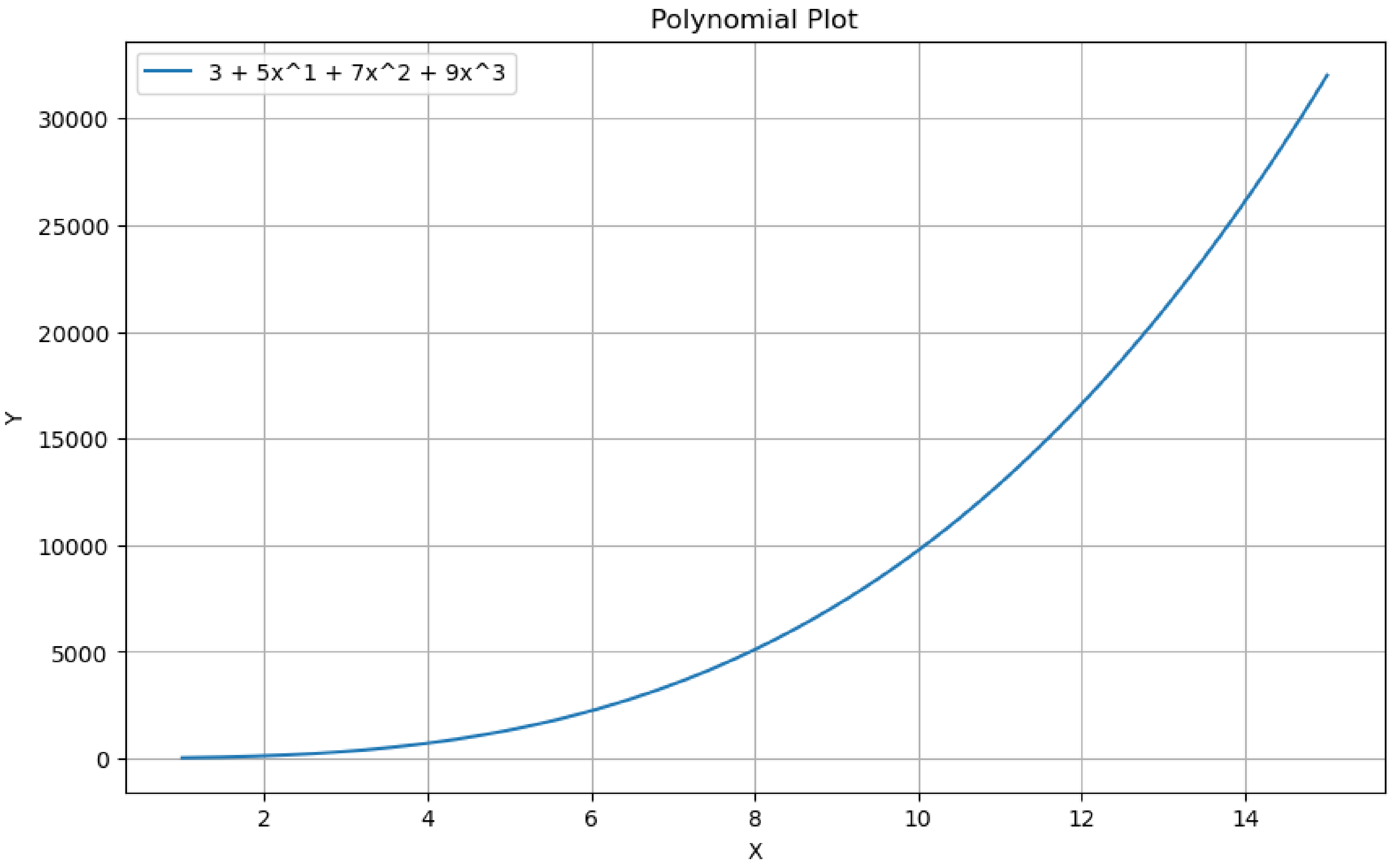
In [10]:

```
class Polynomial:
    def __init__(self, coefficients):
        # Initialize the Polynomial with given coefficients
        self.coefficients = coefficients

    def evaluate(self, x):
            # Evaluate the polynomial for a given value of x
            return sum([a*(x**i) for i, a in enumerate(self.coefficients)])

    def differentiate(self):
            # Differentiate the coefficients with those of its derivative p'(x)
            self.coefficients = [i*a for i,a in enumerate(self.coefficients)][1:]

    def __str__(self):
            # Return a string representation of the polynomial
            terms = []
            for i, a in enumerate(self.coefficients):
                if a:
                    if i == 0:
                        terms.append(str(a))
                    elif i==1:
                        terms.append(f"{a}x^{i}")
                    else:
                        terms.append(f"{a}x^{i}")
            return " + ".join(terms)


# Testing
a = [3, 5, 7, 9]
p = Polynomial(a)
print("Originial Polynomial: ", p)
```

```
Originial Polynomial:  3 + 5x^1 + 7x^2 + 9x^3
```

In [11]:

```
# Evaluate the polynomial
x_values = np.linspace(1, 15, 500)
y = [p.evaluate(x) for x in x_values]

# Plot the result
plt.figure(figsize = (10, 6))
plt.plot(x_values, y, label = f"{p}")
plt.xlabel("X")
plt.ylabel("Y")
plt.title("Polynomial Plot")
plt.legend()
plt.grid(True)
plt.show()
```


Polynomial Plot
3 + 5x^1 + 7x^2 + 9x^3
30000
25000
20000
15000
10000
5000
0
Y
2
4
6
8
10
12
14
X

In [12]:

```
#Originial
print("Originial Polynomial: ", p)

# Differentiate the Polynomial
p.differentiate()

# Display the differentiated polynomial
print(f"Differentiated: {p}")

# Evaluate the differentiated polynomial at x = 10
at_10 = p.evaluate(10)
print(f"Evaluated at p'(10)= {at_10}")
```

```
Originial Polynomial:  3 + 5x^1 + 7x^2 + 9x^3
Differentiated: 5 + 14x^1 + 27x^2
Evaluated at p'(10)= 2845
```

Question 4

1. Define a function that takes the arguments x_0 and n to simulate the difference equation.
2. Simulate for n= 20 000 000. How long did it take?
3. Now repeat the simulation, but use the Numba's JIT decorator. How long does it take? Run it again, how long does it take the second time, what about a third time?

In [30]:

```
# Import the time-module for time-taking
import time

def Non_DE(x_0, n):
    x = x_0
    for _ in range (n):
        x = (3*x + x**2)/(1+x**2)
    return x

x_0 = 0.5
n = 200000000

# Getting the start time
st = time.time()

result = Non_DE(x_0=x_0, n = n)

# Get the end time
et = time.time()

print(f"The result was: {result} and the time it took: {round(et-st, 3)}")
```

```
The result was: 2.0 and the time it took: -71.015
```

In [26]:

```
# Import the JIT from Numba
from numba import jit
import quantecon as qe
```

In [32]:

```
@jit(nopython = True)
def Non_DE_JIT(x_0, n):
    x = x_0
    for _ in range (n):
        x = (3*x + x**2)/(1+x**2)
    return x

x_0 = 0.5
n = 200000000
# Warm up the JIT-accelerated function
Non_DE_JIT(x_0=x_0, n = 1)

# Taking the time and running the function
start_time = time.time()
Non_DE_JIT(x_0=x_0, n = n)
end_time = time.time()

# Printing out the result
print(f"The JIT time was: {round(end_time - start_time, 3)}")

# Using Quantecon tic and toc for benchmarking
qe.tic()
Non_DE_JIT(x_0=x_0, n = n)
time_jit = qe.toc()

print(f"Using the quantecon, time it took:{time_jit}")
```

```
The JIT time was: 1.735
TOC: Elapsed: 0:00:1.71
Using the quantecon, time it took:1.7101550102233887
```