INTRODUCING AI-DS-ML SKILLS

PYTHON PROGRAMMING

CHAPTER 10: MATPLOTLIB AND PANDAS

BY FRED ANNEXSTEIN, PHD

PYTHON PROGRAMMING

10. MATPLOTLIB AND PANDAS

BY FRED ANNEXSTEIN, PHD

CHAPTER 10 OVERVIEW

In this module we will be programming in a style known as declarative programming, where we abstract away the control flow for the logic required for the software to perform an action. In declarative programming we use statements and commands for determining what the task or desired outcome is. In this module we will be using a collection of popular python libraries for working with table data including, Numpy, Pandas, and Matplotlib.

CHAPTER 10 OBJECTIVES

By the end of this chapter, you will be able to...

1. Understand the syntax and usage of a declarative programming style using Python.
2. Understand how to work with Matplotlib library for basic plotting, including line plots, bar plots, and scatterplots. More advanced plotting such as trig functions and fractals.
3. Understand how to work with Pandas series and data frames.

4. Understand how to create and inspect a data frame object.
5. Understand how to aggregate and visualize statistical data with Pandas and Matplotlib.
6. Understand how to generate a scatterplot of two normalized variables and analyze their correlation.

BASIC PLOTS IN MATPLOTLIB

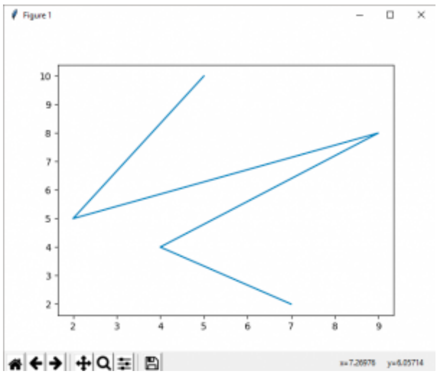
Matplotlib comes with a wide variety of plots. Plots can help people to understand trends and patterns in datasets, especially correlations. Some example code for matplotlib plots are covered here, including line plots, bar plots, and scatterplots. We start with two lists in each case, call a command to build the plot, and finally call a command to show the plot.

LINE PLOT

```
from matplotlib import
pyplot as plt

x = [5, 2, 9, 4, 7]
y = [10, 5, 8, 4, 2]

plt.plot(x,y)
plt.show()
```

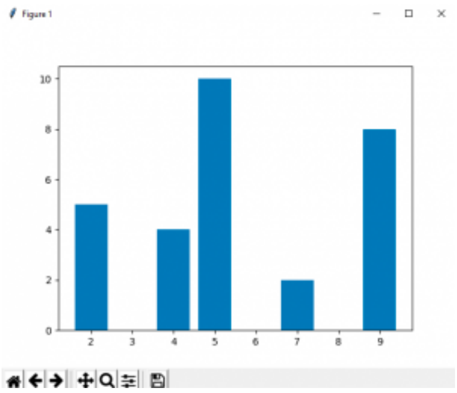


BAR PLOT

```
from matplotlib
import pyplot as
plt

x = [5, 2, 9, 4, 7]
y = [10, 5, 8, 4,
2]

plt.bar(x,y)
plt.show()
```

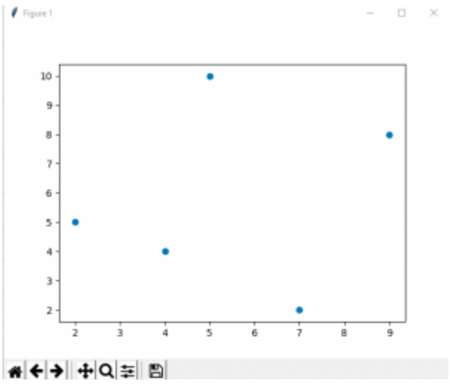


SCATTERPLOTS

```
from matplotlib import
pyplot as plt

x = [5, 2, 9, 4, 7]
y = [10, 5, 8, 4, 2]


plt.scatter(x, y)
plt.show()
```



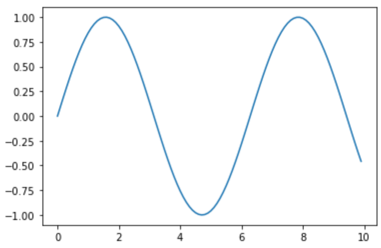
LINE PLOT FOR A TRIG FUNCTION

We can plot the curve of a trig function using Numpy arange command to obtain an array of numbers over an interval. We then use the ordinary plot to get a graph of the sine function, for example, as follows:

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

x = np.arange(0,10,.1)
y = np.sin(x)

plt.plot(x,y)
plt.show()
```



GENERATING A FRACTAL

In this next example we will use a scatterplot to visualize a fractal image known as Sierpiński triangle. The fractal is generated by choosing 3 corners of a triangle at random, as well as a random starting point. We then iteratively choose a random corner of the triangle and move to the midpoint between the current random point and the random corner. The image generated gives a recursive pattern with infinitely nested similar triangle.

```
import numpy as np
import matplotlib.pyplot as plt
from random import random, randint

# Three corners of an random triangle
corner = [(random(), random()),
(random(),random()), (random(),random())]
```

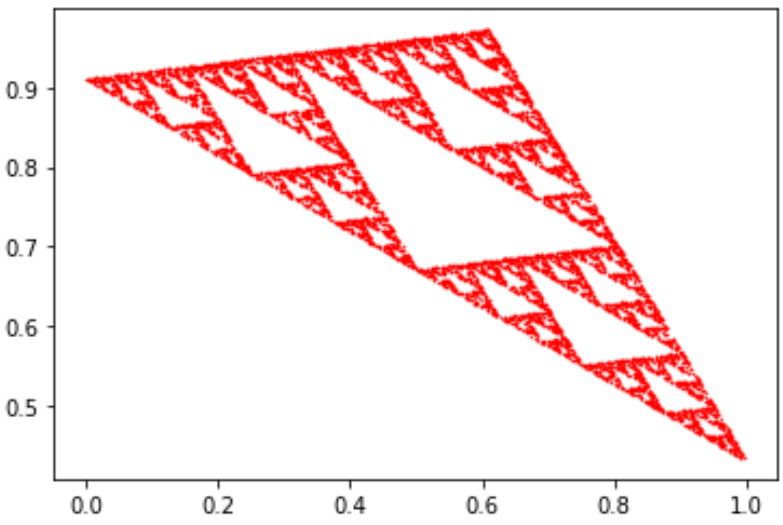
```
def midpoint(p, q):
    return (0.5*(p[0] + q[0]),
            0.5*(p[1] + q[1]))

N = 10000
x = np.zeros(N)
y = np.zeros(N)

x[0] = random()
y[0] = random()

for i in range(1, N):
    k = randint(0, 2)
    x[i], y[i] = midpoint( corner[k],
                    (x[i-1], y[i-1]) )

plt.scatter(x, y, s=0.01, c="red")
plt.show()
```



PANDAS

Pandas is an extremely popular Python library for data table creation and manipulations. Pandas makes abundant use of NumPy's ndarray, which was a data class introduced in the last chapter.

Pandas has two key data structures: the Series, which is one dimensional, and DataFrames, which is two dimensional. We will use DataFrame to provide a size-mutable, two-dimensional structure for data tables made up of three components: rows and columns (which are labeled) and data values.

PANDAS DATAFRAME

A DataFrame is an enhanced two-dimensional array. DataFrames can have custom row and column indices, and offer additional operations and capabilities that make them more convenient for many data-science oriented tasks. DataFrames also support missing data.

DataFrames let you organize tabular information, allowing you to view it more easily than nested lists. Generally speaking, in the DataFrame grid, each row corresponds with an instance or data sample, and each column contains variable data for one attribute. The data in the columns can contain numeric, alphanumerical characters or logical data and typically are of the same type, although they do not have to be.

Each column in a DataFrame is called a series, an object of the class pandas.core.series.Series. The Series representing each column may contain different element types, however we will only be storing series of numbers.

To create a Python Pandas DataFrame, we can load existing datasets using a CSV file. However, next we show how to create a Dataframe grid directly from a dictionary using multiple lists. Each key/value pair of the dictionary will become a new column, see the example that follows:

```
import pandas as pd

temps_dict = {
    'St Thomas': [87, 96, 70],
    'Key West': [100, 87, 90],
    'San Juan': [94, 77, 90],
    'Havana': [100, 81, 82],
    'Miami': [83, 65, 85]}

temps = pd.DataFrame(temps_dict)


>>> type(temps)
pandas.core.frame.DataFrame
```

```
>>> temps
  St Thomas  Key West  San Juan  Havana  Miami
0        87       100        94     100     83
1        96        87        77      81     65
2        70        90        90      82     85

>>> type(temps.'Miami')
pandas.core.series.Series

>>> temps.Miami
0    83
1    65
2    85
Name: Miami, dtype: int64
```

THE INDEX ATTRIBUTE

We can use the index attribute to change the DataFrame's row indices from sequential integers to strings, which helps readability.

```
>>> temps.index=['Friday', 'Saturday', 'Sunday']

>>> temps
            St Thomas  Key West  San Juan  Havana  Miami
Friday             87       100        94     100     83
Saturday           96        87        77      81     65
Sunday             70        90        90      82     85
```

LOGICAL OPERATORS

We can use the logical operators on column values to filter rows.

```
>>> temps[temps.SanJuan > 80]
         StThomas  KeyWest  SanJuan  Havana  Miami
Friday         87      100       94     100     83
Sunday         70       90       90      82     85
```

We have selected all the rows in which the value in "SanJuan" column is greater than 80.

Pandas allows for combining multiple logical operators. For instance, we can apply conditions on both val and val2 columns as below.

```
>>> temps[(temps.SanJuan > 80) & (temps.Miami < 84)]

        StThomas  KeyWest  SanJuan  Havana  Miami
Friday        87      100       94     100     83

>>> temps[(temps.SanJuan > 80) | (temps.Miami < 84)]
          StThomas  KeyWest  SanJuan  Havana  Miami
Friday          87      100       94     100     83
Saturday        96       87       77      81     65
Sunday          70       90       90      82     85
```

THE ISIN METHOD

The isin method is another way of applying multiple condition for filtering. For instance, we can filter the names that exist in a given list.

```
>>> days = ['Friday','Sunday']
>>> temps[temps.index.isin(days)]

StThomas  KeyWest  SanJuan  Havana  Miami
Friday        87      100       94     100     83
Sunday        70       90       90      82     85
```

NLARGEST OR NSMALLEST

In some cases, we do not have a specific range for filtering but just need the largest or smallest values. The nlargest and nsmallest functions allow for selecting rows that have the largest or smallest values in a column, respectively.

```
>>> temps.nlargest(1,'Miami')

        StThomas  KeyWest  SanJuan  Havana  Miami
Sunday        70       90       90      82     85

>>> temps.nlargest(2,'KeyWest')

        StThomas  KeyWest  SanJuan  Havana  Miami
Friday        87      100       94     100     83
Sunday        70       90       90      82     85

>>> temps.nsmallest(2,'KeyWest')

          StThomas  KeyWest  SanJuan  Havana  Miami
Saturday        96       87       77      81     65
Sunday          70       90       90      82     85
```

.ILOC AND .LOC INDEXERS

The loc and iloc methods are used to select rows or columns based on index or label.

loc: select rows or columns using labels
iloc: select rows or columns using indices

Thus, they can be used for filtering. However, we can only select a particular part of the dataframe without specifying a condition. To select one column, place the column’s name between brackets. To select one row, place the row’s name as in index to the .loc object in class pandas.core.indexing._LocIndexer. The code would look similar to this:

```
>>> temps['Miami']
0    83
1    65
2    85
Name: Miami, dtype: int64

>>> temps.loc['Friday']

St Thomas     87
Key West     100
San Juan      94
Havana       100
Miami         83
Name: Friday, dtype: int64
```

Use DataFrame.loc[] or pass the integer's location as an index to the iloc[] object to select multiple rows and columns. Here is what the code might look like:

```
>>> temps.loc[['Friday','Sunday']]

        St Thomas  Key West  San Juan  Havana  Miami
Friday         87       100        94     100     83
Sunday         70        90        90      82     85


# Return first 3 columns of 2 outer rows
>>> temps.iloc[[0, 2], 0:3]

        St Thomas  Key West  San Juan
Friday         87       100        94
Sunday         70        90        90
```

One of pandas' more powerful selection capabilities is Boolean indexing. For example, we can create a table selecting all the high temperatures —that is, those that are greater than or equal to 90, as follows:

```
>>> temps[temps >= 90]

          St Thomas  Key West  San Juan  Havana  Miami
Friday          NaN     100.0      94.0   100.0    NaN
Saturday       96.0       NaN       NaN     NaN    NaN
Sunday          NaN      90.0      90.0     NaN    NaN
```

We can locate an individual attribute by specifying the row and column indexes, as follows:

```
>>temps.at['Saturday', 'San Juan']
77
```

CHAPTER 10 LAB

For Lab 10 we will load data from the **Superhero Movie Dataset —** you will find this dataset at the end of the chapter. You will then perform some simple manipulations and plot the data as a means of exploring the features in this dataset. Let us create the lab program from scratch, as follows:

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from pandas import DataFrame, Series
```

Next, download the dataset and save it as a .csv file. You should be able to do this using Numbers on Mac or Excel on Windows. Save the data set as 'dataset.csv'.

We will read the data from the .csv file using the Pandas pd.read_csv() method. Since this particular data set does not include a header row, we must name each column series in the file. Try to execute this code:

```
sh_raw =
  pd.read_csv('/Users/fred/Desktop
     dataset.csv',  header=None,
     names=
['Year','Title','Comic','IMDB','RT',
'CompositeRating','OpeningWeekendBoxOffice',
'AvgTicketPriceThatYear','EstdOpeningAttenda
nce','USPopThatYear'])

print(sh_raw.head(5))
```

When you execute it, you should see output like this:

```
 Year         Title  ... EstdOpeningAttendance  USPopThatYear
NaN  1978.0      Superman  ...           3190317.521    222584545.0
NaN  1980.0   Superman II  ...           5241830.112    227224681.0
NaN  1982.0   Swamp Thing  ...                   NaN    231664458.0
NaN  1983.0  Superman III  ...           4238843.492    233791994.0

[5 rows x 10 columns]
```

Since we will be analyzing box office numbers, we need to clean up the dataset and exclude movies where this data is missing. those columns have **Not a Number** `NaN` in the **OpeningWeekendBoxOffice** Series. We will use the Numpy function `isfinite()` to check for value numbers in this column. Try to execute the following:

```
sh = sh_raw[np.isfinite(
            sh_raw.OpeningWeekendBoxOffice)]
print(sh.head(5))

Output:

Year            Title  ... EstdOpeningAttendance  USPopThatYear
NaN  1978.0        Superman  ...           3190317.521    222584545.0
NaN  1980.0     Superman II  ...           5241830.112    227224681.0
NaN  1983.0    Superman III  ...           4238843.492    233791994.0
NaN  1984.0       Supergirl  ...           1707812.202    235824902.0
NaN  1986.0  Howard the Duck  ...           1366613.477    240132887.0
```

You should notice that Swamp Thing has been omitted from the output. That happened because we did not have the OpeningWeekendBoxOffice value for this title.

With our dataset cleaned, we now add the calculated columns required to perform our analysis.

We wish to compare Rotten Tomatoes ratings to IMDB ratings. To do this meaningfully, we have to normalize them first, since they have different scoring ranges. Normalization is done by dividing the original score by the maximum possible value, and thereby obtaining a normalized score between 0.0 and 1.0. Once we produce two normalized series of numbers, we can visualize there correlation by plotting a scatterplot. Enter the following code and execute it.

```
# Normalize the scores

imdb_normalized = sh.IMDB / 10

sh.insert(10,'IMDBNormalized',imdb_normalize
d)

rt_normalized = sh.RT/100

sh.insert(11, 'RTNormalized', rt_normalized)
```

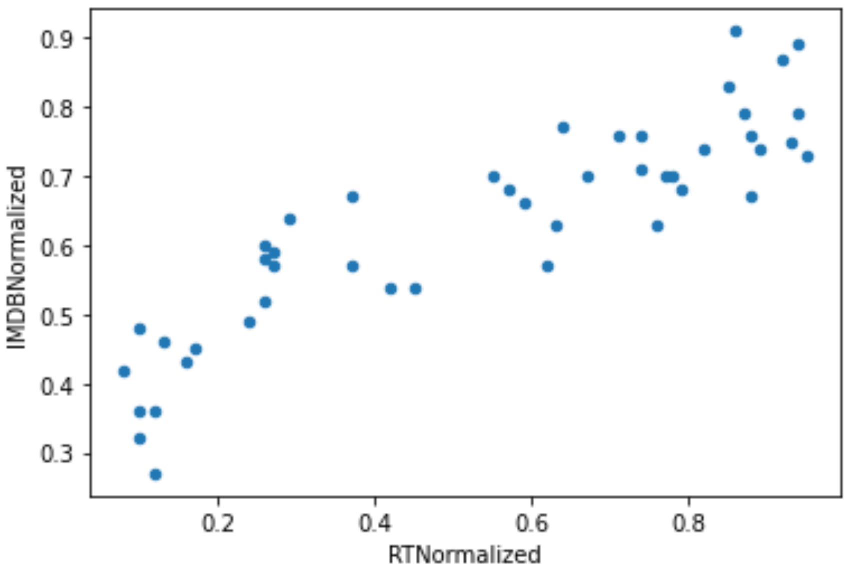
With our scores normalized, let's make our first scatter plot, and explore the relationship between Rotten Tomatoes and IMDB ratings for each movie. Try to execute the following:

```
sh.plot.scatter(x ='RTNormalized',
                y ='IMDBNormalized')
plt.show()
```

Here is the output you should see:

At a glance you can see there is a positive correlation — a trending of points up and to the right, which one should expect from two different movie ratings services.

We now want to calculate the correlation coefficient and verify how strong of a correlation this is. And, lucky for us Pandas



provides a corr() method to calculate correlations. Rather than do this to the entire DataFrame, we select the two normalized columns in question. The
Pandas describe() method makes it easy to get summary statistics for our data, including mean, standard deviation,
and percentiles. Try to execute the following two commands:

```
print(sh[[‘RTNormalized’,’IMDBNormalized']].
corr())

print(sh[['RTNormalized','IMDBNormalized']].
describe())

# Output
               RTNormalized  IMDBNormalized
RTNormalized        1.00000         0.88836
```

```
IMDBNormalized       0.88836          1.00000
       RTNormalized  IMDBNormalized
count     46.000000       46.000000
mean       0.539783        0.630000
std        0.302129        0.152898
min        0.080000        0.270000
25%        0.260000        0.540000
50%        0.605000        0.665000
75%        0.812500        0.740000
max        0.950000        0.910000
```

We find that the correlation is 0.88836, which, indeed, is a high positive correlation. We see also that IMDB has a higher mean score and a lower standard deviation as compared to RT scores.

The 25th percentile is the value at which 25% of the answers lie below that value, and 75% of the answers lie above that value.

From the scatterplot output we note that in the 25th percentile for Rotten Tomatoes there are more lower ratings for the same movies than IMDB. See if you can verify that from the output.

Required Lab Questions:

There are no doctests for this lab. Please answer the following questions. As a deliverable, upload your modified code that includes answers to each of the following questions:

1. Define a command to print the Series

records for the 'DC' comic movies from the sh DataFrame.

2. Define a command to show the Year and Title of only 'DC' movies from the sh DataFrame.
3. Define a command to show the Year and Title of only 'Marvel' movies from the sh DataFrame.
4. Define a command to plot a scatterplot for the AvgTicketPriceThatYear on the y-axis and with Year on the x axis.
5. Define a command to print the correlation for the Year versus the AvgTicketPriceThatYear.
6. Define a command to print the summary statistics for OpeningWeekendBoxOffice.

Superhero Movie Dataset to use for the Lab:

```
1978,Superman,DC,7.3,95,84,7465343,2.34,3190317.521,222584545
1980,Superman II,DC,6.7,88,77.5,14100523,2.69,5241830.112,227224681
1982,Swamp Thing,DC,5.3,60,56.5,,2.94,,231664458
1983,Superman III,DC,4.9,24,36.5,13352357,3.15,4238843.492,233791994
1984,Supergirl,DC,4.2,8,25,5738249,3.36,1707812.202,235824902
1986,Howard the Duck,Marvel,4.3,16,29.5,5070136,3.71,1366613.477,240132887
1987,Superman IV: The Quest for Peace,DC,3.6,10,23,5683122,3.91,1453483.887,242288918
1989,Batman,DC,7.6,71,73.5,40489746,3.97,10198928.46,246819230
1989,The Return of Swamp Thing,DC,3.9,40,39.5,,3.97,,246819230
1989,The Punisher,Marvel,5.4,24,39,,3.97,,246819230
1992,Batman Returns,DC,7,78,74,45687711,4.15,11009086.99,255029699
1995,Batman Forever,DC,5.4,42,48,52784433,4.35,12134352.41,262803276
1997,Batman & Robin,DC,3.6,12,24,42872605,4.59,9340436.819,267783607
1997,Steel,DC,2.7,12,19.5,870068,4.59,189557.2985,267783607
1998,Blade,Marvel,7,55,62.5,17073856,4.69,3640481.023,270248003
2000,X-Men,Marvel,7.4,82,78,54471475,5.39,10106025.05,282171957
2002,Blade II,Marvel,6.6,59,62.5,32528016,5.81,5598625.818,287803914
2002,Spider-Man,Marvel,7.4,89,81.5,114844116,5.81,19766629.26,287803914
2003,Daredevil,Marvel,5.4,45,49.5,40310419,6.03,6684978.275,290326418
2003,Hulk,Marvel,5.7,62,59.5,62128420,6.03,10303220.56,290326418
2003,X2,Marvel,7.6,88,82,85558731,6.03,14188844.28,290326418
2004,Blade: Trinity,Marvel,5.8,26,42,16061271,6.21,2586356.039,293045739
2004,Catwoman,DC,3.2,10,21,16728411,6.21,2693785.99,293045739
2004,Spider-Man 2,Marvel,7.5,93,84,88156227,6.21,14195849.76,293045739
2004,The Punisher,Marvel,6.4,29,46.5,13834527,6.21,2227782.126,293045739
2005,Batman Begins,DC,8.3,85,84,48745440,6.41,7604592.824,295753151
2005,Elektra,Marvel,4.8,10,29,12804793,6.41,1997627.613,295753151
2005,Fantastic Four,Marvel,5.7,27,42,56061504,6.41,8745944.462,295753151
2006,Superman Returns,DC,6.3,76,69.5,52535096,6.55,8020625.344,298593212
2006,X-Men: The Last Stand,Marvel,6.8,57,62.5,102750665,6.55,15687124.43,298593212
2007,Fantastic Four: Rise of the Silver Surfer,Marvel,5.7,37,47,58051684,6.88,8437744.767,301579895
2007,Ghost Rider,Marvel,5.2,26,39,45388836,6.88,6597214.535,301579895
2007,Spider-Man 3,Marvel,6.3,63,63,151116516,6.88,21964609.88,301579895
2008,The Dark Knight,DC,8.9,94,91.5,158411483,7.18,22062880.64,304374846
2008,The Incredible Hulk,Marvel,7,67,68.5,55414050,7.18,7717834.262,304374846
```

```
2008,Iron Man,Marvel,7.9,94,86.5,98618668,7.18,13735190.53,304374846
2008,Punisher: War Zone,Marvel,6,26,43,4271451,7.18,594909.61,304374846
2009,Watchmen,DC,7.7,64,70.5,55214334,7.5,7361911.2,307006550
2009,X-Men Origins: Wolverine,Marvel,6.7,37,52,85058003,7.5,11341067.07,307006550
2010,Iron Man 2,Marvel,7.1,74,72.5,128122480,7.89,16238590.62,308745538
2010,Jonah Hex,DC,4.6,13,29.5,5379365,7.89,681795.3105,308745538
2011,Captain America: The First Avenger,Marvel,6.8,79,73.5,65058524,7.93,8204101.387,311591917
2011,Green Lantern,DC,5.9,27,43,53174303,7.93,6705460.656,311591917
2011,Thor,Marvel,7,77,73.5,65723338,7.93,8287936.696,311591917
2011,X-Men: First Class,Marvel,7.9,87,83,55101604,7.93,6948499.874,311591917
2012,Marvel's The Avengers,Marvel,8.7,92,89.5,207438708,7.92,26191756.06,314055984
2012,The Dark Knight Rises,DC,9.1,86,88.5,160887295,7.92,20314052.4,314055984
2012,Ghost Rider: Spirit of Vengeance,Marvel,4.5,17,31,22115334,7.92,2792340.152,314055984
2012,The Amazing Spider-Man,Marvel,7.6,74,75,62004688,7.92,7828874.747,314055984
```