INF-552 Lab0

September 8, 2023

INF-552 Lab 0

Introduction to Basic Development Tools

0.0.1 Rules

1. Please read the instructions and problem prompts **carefully**.
2. This lab is to give you some basic APIs of numpy, pandas and scikit-learn. Besides, some topics such as how to make your jupyter notebook be a more efficient developing tools, how to use git and GitHub will also be covered. The lab is to be done individually. You may talk to your fellow classmates about general issues ("Remind me again: Which API should I used for doing group by operation to a data set") but about the specifies of how to do these exercises.
3. Along with a similar vein, you can ask the TA for help, but ask questions about **concepts** but not ask the TA to help you debug your code. The TA is here to help, but not to do the work for you.
4. You are welcome to use the class resources and the Internet.
5. Playing with variations. Solve one problems, and then copy the code to a new cell and play around with it. Doing this is the single most important thing when learning programming.
6. This lab will not be graded but the content is highly related to your future programming assignments. So, treat it wisely.
7. All the content having been gone though in the week 1 discussion is just a snapshot of the most basic concepts. **You need to keep study more about Git, GitHub, Pandas, Numpy and Scikit-Learn in order to finish your programming assignments successfully.**
8. Have fun!

0.0.2 Setup Development Environment

There are many ways to setup the environment. But, I do recommend a simple idea that is using the Anaconda, which is a pre-build python environment with bundles of useful packages.

To download the Anaconda, go to the following website: https://www.anaconda.com/distribution/. Download the correct version based on your operating system and install it step by step.

Then, **configure your PATH environment variable** to make the conda command work. The following command is an easy way to test whether your configuration is correct. If it is, you will see something as like as the sample output.

command:

conda –version

sample output:

conda 4.6.12

Finally, download this jupyter notebook file, then change the working directory to where its location in terminal, and type the following command to open the jupter notebook and finish the lab.

command: jupyter notebook

[3]:
```
import pandas as pd
import numpy as np
```

0.0.3 Pandas

The read_csv() Method First, read the documentation about the *read_csv()* method in Pandas (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html). Then, try to read data from file Salaries.csv to a dataframe, make the column playerID in the csv file as the index column and the first row as the header. Also, skip the second row when reading the file.

[4]:
```
salaries_data = pd.read_csv('Salaries.csv', index_col='playerID', header=0,␣
 ↪skiprows=[1])
# salaries_data = pd.read_csv('Salaries.csv', index_col=3, header=0,␣
 ↪skiprows=[1])
salaries_data
print(salaries_data)
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[4], line 1
----> 1 salaries_data =␣
 ↪pd.read_csv('Salaries.csv', index_col='playerID', header=0, skiprows=[1])
      2 # salaries_data = pd.read_csv('Salaries.csv', index_col=3, header=0,␣
 ↪skiprows=[1])
      3 salaries_data

File ~/anaconda3/lib/python3.10/site-packages/pandas/io/parsers/readers.py:948,␣
 ↪in read_csv(filepath_or_buffer, sep, delimiter, header, names, index_col,␣
 ↪usecols, dtype, engine, converters, true_values, false_values,␣
 ↪skipinitialspace, skiprows, skipfooter, nrows, na_values, keep_default_na,␣
 ↪na_filter, verbose, skip_blank_lines, parse_dates, infer_datetime_format,␣
 ↪keep_date_col, date_parser, date_format, dayfirst, cache_dates, iterator,␣
 ↪chunksize, compression, thousands, decimal, lineterminator, quotechar,␣
 ↪quoting, doublequote, escapechar, comment, encoding, encoding_errors, dialect␣
 ↪on_bad_lines, delim_whitespace, low_memory, memory_map, float_precision,␣
 ↪storage_options, dtype_backend)
    935 kwds_defaults = _refine_defaults_read(
    936     dialect,
    937     delimiter,
   (…)
```

```
    944     dtype_backend=dtype_backend,
    945 )
    946 kwds.update(kwds_defaults)
--> 948 return _read(filepath_or_buffer, kwds)

File ~/anaconda3/lib/python3.10/site-packages/pandas/io/parsers/readers.py:611,
 ↪in _read(filepath_or_buffer, kwds)
    608 _validate_names(kwds.get("names", None))
    610 # Create the parser.
--> 611 parser = TextFileReader(filepath_or_buffer, **kwds)
    613 if chunksize or iterator:
    614     return parser

File ~/anaconda3/lib/python3.10/site-packages/pandas/io/parsers/readers.py:1448
 ↪in TextFileReader.__init__(self, f, engine, **kwds)
   1445     self.options["has_index_names"] = kwds["has_index_names"]
   1447 self.handles: IOHandles | None = None
-> 1448 self._engine = self._make_engine(f, self.engine)

File ~/anaconda3/lib/python3.10/site-packages/pandas/io/parsers/readers.py:1705
 ↪in TextFileReader._make_engine(self, f, engine)
   1703     if "b" not in mode:
   1704         mode += "b"
-> 1705 self.handles = get_handle(
   1706     f,
   1707     mode,
   1708     encoding=self.options.get("encoding", None),
   1709     compression=self.options.get("compression", None),
   1710     memory_map=self.options.get("memory_map", False),
   1711     is_text=is_text,
   1712     errors=self.options.get("encoding_errors", "strict"),
   1713     storage_options=self.options.get("storage_options", None),
   1714 )
   1715 assert self.handles is not None
   1716 f = self.handles.handle

File ~/anaconda3/lib/python3.10/site-packages/pandas/io/common.py:863, in␣
 ↪get_handle(path_or_buf, mode, encoding, compression, memory_map, is_text,␣
 ↪errors, storage_options)
    858 elif isinstance(handle, str):
    859     # Check whether the filename is to be opened in binary mode.
    860     # Binary mode does not support 'encoding' and 'newline'.
    861     if ioargs.encoding and "b" not in ioargs.mode:
    862         # Encoding
--> 863         handle = open(
    864             handle,
    865             ioargs.mode,
    866             encoding=ioargs.encoding,
```

```
    867             errors=errors,
    868             newline="",
    869         )
    870     else:
    871         # Binary mode
    872         handle = open(handle, ioargs.mode)

FileNotFoundError: [Errno 2] No such file or directory: 'Salaries.csv'
```

Indexing and Selecting Data Select the id of the players who are registered in ATL and HOU and whose salary is higher than one million.

[3]:
```
# salaries_data[(salaries_data["salary"]>1000000) &␣
 ↪(salaries_data["teamID"]=="ATL")|
#               (salaries_data["salary"]>1000000) &␣
 ↪(salaries_data["teamID"]=="HOU")].index
salaries_data[(salaries_data['teamID'].isin(['ATL','HOU'])) &␣
 ↪(salaries_data['salary']>1000000)].index
```

[3]:
```
Index(['hornebo01', 'murphda05', 'suttebr01', 'ryanno01', 'hornebo01',
       'murphda05', 'suttebr01', 'ryanno01', 'griffke01', 'murphda05',
       …
       'feldmsc01', 'gonzama01', 'gregelu01', 'lowrije01', 'neshepa01',
       'quallch01', 'rasmuco01', 'singljo02', 'sippto01', 'valbulu01'],
      dtype='object', name='playerID', length=595)
```

The describe() Method Calculate the standard Deviation, first quartile, medium, third quartile, mean, maximum, minimum of the salary in team ATL.

[4]:
```
salaries_data[salaries_data['teamID']=='ATL']['salary'].describe()
```

[4]:
```
count    8.850000e+02
mean     2.207749e+06
std      3.434320e+06
min      6.000000e+04
25%      3.000000e+05
50%      6.000000e+05
75%      2.400000e+06
max      1.606180e+07
Name: salary, dtype: float64
```

The iterrows() Method Create a Python dictionary object whose keys are the headers of the dataframe created in the read_csv() exercise and values are Python list objects that contain data corresponding to the headers. (Here, use the iterrows method to iterate each row of the dataframe and copy it to a dictionary. However, there is a easier way. Learn how the to_dict() method works by yourself later)

[5]:
```
my_dict = {'playerID':[],'yearID':[], 'teamID':[], 'lgID':[], 'salary':[]}

for row in salaries_data.iterrows():
    my_dict['playerID'].append(row[0])
    my_dict['teamID'].append(row[1].teamID)
    my_dict['yearID'].append(row[1].yearID)
    my_dict['lgID'].append(row[1].lgID)
    my_dict['salary'].append(row[1].salary)

# print(my_dict)
```

Create Dataframe Using the Constructor Read the documentation: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html#pandas.DataFrame and create a dataframe using pd.DataFrame from the dictionary created in the iterrows() exercise. Change the header to "a", "b", "c", ... at creation time.

[6]:
```
# df = pd.DataFrame(data=my_dict)

# # df

# new_col = {'playerID': 'a', 'yearID': 'b', 'teamID': 'c', 'lgID':'d',␣
  ↪'salary':'e'}
# df.rename(columns=new_col, inplace=True)

# df

pd.DataFrame(salaries_data.to_dict())
```

[6]:

	yearID	teamID	lgID	salary
bedrost01	1995	ATL	NL	750000
benedbr01	1989	ATL	NL	395000
campri01	1986	ATL	NL	600000
ceronri01	1992	MON	NL	230000
chambch01	1986	ATL	NL	769452
…	…	…	…	…
barreaa01	2015	WAS	NL	514200
dendema01	2015	WAS	NL	512972
robincl01	2015	WAS	NL	525000
taylomi02	2015	WAS	NL	508700
treinbl01	2015	WAS	NL	512800

[4963 rows x 4 columns]

0.0.4 Numpy

Quick start: https://www.numpy.org/devdocs/user/quickstart.html

Numpy axes explaination: https://www.sharpsightlabs.com/blog/numpy-axes-explained/

The np.array Method Example 1:

```
ls = [1, 2, 3]
arr = np.array(ls)
```

Example 2:

```
>>> np.array([[1, 2], [3, 4]])
array([[1, 2],
       [3, 4]])
```

Now, create a 2-dimensional Python list object, then convert it to a Numpy array object.

[7]:
```
import numpy as np


lst = [[1,2,3],[4,5,6],[7,8,9]]
arr = np.array(lst)
arr
```

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

ndarray Objects' Attributes Play with the **ndim, shape, size, dtype, itemsize and data** attribute.

Example:

```
>>> arr = np.array([[1, 2], [3, 4]])
>>> arr.ndim
2
```

[8]:
```
print('ndim', arr.ndim) #no. of axes (dimension) of the array
print('shape', arr.shape) #(n,m): n=row, m=column
print('size', arr.size) #total no. of elements of the array
print('itemsize', arr.itemsize) #the size in bytes of each element of the array␣
 ↪np.float64 np.int32 np.float32
print('data', arr.data) #the buffer containing the actual elements of the array
```

```
ndim 2
shape (3, 3)
size 9
itemsize 8
data <memory at 0x1217b8110>
```

Dimension of ndarray Ojects Play with the reshape() and flatten() method.

Example:

```
>>> arr = np.array([[1, 2], [3, 4]])
>>> arr.flatten()
```

```
array([1, 2, 3, 4])
```

[9]:
```
arr.flatten()
```

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

The Slice Operation of ndarray Objects Understand how the slice operation works for 1-D array and 2-D array.

Example:

```
>>> arr = np.array([[1, 2, 3], [3, 4, 6], [7, 8, 9]])
>>> arr[1:]
array([[3, 4, 6],
       [7, 8, 9]])
>>> arr[1:, 0:2]
array([[3, 4],
       [7, 8]])
```

[10]:
```
lst = [0,1,2,3,4]
lst[0]
lst[:2]

print(arr)
print("\n2 row")
print(arr[1])
print("\n2,3 rows")
print(arr[1:])
print("\n2 column")
print(arr[:, 1])
print("\n2,3 column")
print(arr[:, 1:])
print("\nintersection of 2,3 rows and 0,1 columns")
print(arr[1:,0:2])
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]

2 row
[4 5 6]

2,3 rows
[[4 5 6]
 [7 8 9]]

2 column
[2 5 8]
```

```
2,3 column
[[2 3]
 [5 6]
 [8 9]]

intersection of 2,3 rows and 0,1 columns
[[4 5]
 [7 8]]
```

The Calculation of ndarray Objects Play with the **argmin(), argmax(), min(), max(), mean(), sum(), std(), dot(), square(), sqrt(), abs(). exp(), sign(), mod()** method.

Example:

```
>>> np.square(array)
array([[ 1,  4,  9],
       [ 9, 16, 36],
       [49, 64, 81]])
```

[11]:
```
arr = np.array([[1,2,3,4],[5,6,7,8],[7,8,9,0]])
np.square(arr) #returns square of each object
np.argmin(arr) ## indicies of minimum values
print(np.argmax(arr)) ##indicies of max value
np.min(arr) ##smallest object
np.max(arr) ##largest object
np.mean(arr) ##avg of objects
np.sum(arr) ##adds up objects
np.std(arr) ##standard deviation

a=[[1,2], [3,4]]
b=[[3,5], [4,5]]
print(np.dot(a, b)) ## matrix product
print(np.sqrt(arr)) ## square root

arr = np.array([[1,2,-3,4],[5,-6,-7,8],[-7,8,-9,0]])
np.abs(arr) #absoulte value
np.exp(arr) #exponentiate
print(np.sign(arr)) #gives sign

arr2 = np.array([[1,4,-6,2],[2,2,2,2],[-7,8,-9,1]])
print(np.mod(arr, arr2))
```

```
10
[[11 15]
 [25 35]]
[[1.         1.41421356 1.73205081 2.        ]
 [2.23606798 2.44948974 2.64575131 2.82842712]
 [2.64575131 2.82842712 3.         0.        ]]
[[ 1  1 -1  1]
```

```
 [ 1 -1 -1  1]
 [-1  1 -1  0]]
[[ 0  2 -3  0]
 [ 1  0  1  0]
 [ 0  0  0  0]]
```

Other Important Methods Inside Module Numpy Play with the arange(), ones(), zeros(), eye(), linspace(), concatenate() method.

Example:

```
>>> np.eye(3)
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

[12]:
```
#arange() create an array of range 0-9
x = np.arange(10)
print('arange(10)')
print(x)

#ones() creates an array full of 1s
ones = np.ones(3)
print('\nones(3)')
print(ones)

#zero() creates an array full of 0s
zero = np.zeros(5)
print('\nzeros(5)')
print(zero)

#eye() Return a 2-D array with ones on the diagonal and zeros elsewhere.
eye = np.eye(4)
print('\neye(4)')
print(eye)

#linspace() adds the amount of evenly spaced elements you want from the range
linspace =  np.linspace(0,9,11)
print('\nlinspace(0,9,10)')
print(linspace)

#concatenate()
print('\nconcat(ones, zero)')
con = np.concatenate((ones,zero), axis=None)
print(con)
```

```
arange(10)
[0 1 2 3 4 5 6 7 8 9]
```

```
ones(3)
[1. 1. 1.]

zeros(5)
[0. 0. 0. 0. 0.]

eye(4)
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]

linspace(0,9,10)
[0.  0.9 1.8 2.7 3.6 4.5 5.4 6.3 7.2 8.1 9. ]

concat(ones, zero)
[1. 1. 1. 0. 0. 0. 0. 0.]
```

0.0.5 Scikit-Learn

The followings are packages (or methods) in Python (Scikit-Learn and Scipy) that will be frequently used in your programming assignment. So, please read carefully.

- Data Preprocessing (https://scikit-learn.org/stable/modules/classes.html#module-sklearn.preprocessing)
  - Standardization: StandardScaler
  - Normalization: MinMaxScaler
  - Quantifing Categorical Features: LabelEncoder. OneHotEncoder
  - Construct Train and Test Set: model_selection.train_test_split
- KNN: KNeighborsClassifier
- Linear Regression: LinearRegression
- Logistic Regression: LogisticRegression, LogisticRegressionCV
- Feature Selection / Model Selection
  - L1 Penalized Regression (Lasso Regression) with Cross-Validation: LassoCV
  - L2 Penalized Regression (Ridge Regression) with Cross-Validation: RidgeCV
  - Cross-Validation: StratifiedKFold, RepeatedKFold, LeaveOneOut, KFold, model_selection.cross_validate, model_selection.cross_val_predict, model_selection.cross_val_score
  - Model Metrics (https://scikit-learn.org/stable/modules/classes.html#sklearn-metrics-metrics): accuracy_score, auc, f1_score, hamming_loss, precision_score, recall_score, roc_auc_score
- Decision Tree: DecisionTreeClassifier, DecisionTreeRegressor
- Bootstrap, Ensemble Methods
  - Bootstrap: bootstrapped (https://pypi.org/project/bootstrapped/)
  - Bagging: RandomForestClassifier, RandomForestRegressor
  - Boosting: AdaBoostClassifier, AdaBoostRegressor
- Support Vector Machines (https://scikit-learn.org/stable/modules/svm.html#svm): LinearSVC, LinearSVR

- Multiclass and Multilabel Classification (https://scikit-learn.org/stable/modules/classes.html#module-sklearn.multiclass)
  - One-vs-one Multiclass Strategy: OneVsOneClassifier
  - One-vs-the-rest (OvR) multiclass/multilabel strategy / OneVsRestClassifier
- Unsupervised Learning
  - K-means Clustering: KMeans
  - Hierarchical Clustering: scipy.cluster.hierarchy (not scikit-learn)
- Semisupervised Learning (https://scikit-learn.org/stable/modules/label_propagation.html)

0.0.6 Matplotlib

Quick start: https://matplotlib.org/3.1.1/tutorials/introductory/pyplot.html

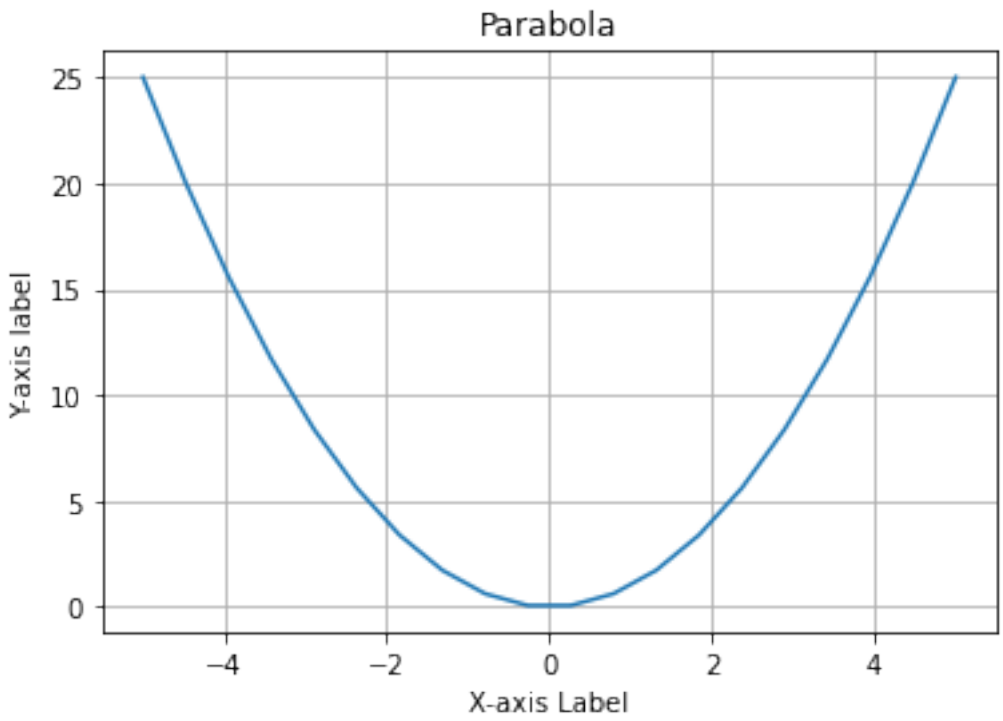
Exercises:

(a) Create two one dimensional arrays x and y and plot y vs x, add title, xlabel, ylabel, grid.

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


x = np.linspace(-5, 5, num=20)
y = np.array([j ** 2 for j in x])
```

copy the code above to the following cell and add code for plotting the parabola.

[13]:
```
import numpy as np
import matplotlib.pyplot as plt


x = np.linspace(-5, 5, num=20)
y = np.array([j ** 2 for j in x])

plt.plot(x,y)
plt.title("Parabola")
plt.xlabel("X-axis Label")
plt.ylabel("Y-axis label")
plt.grid()
plt.show()
```

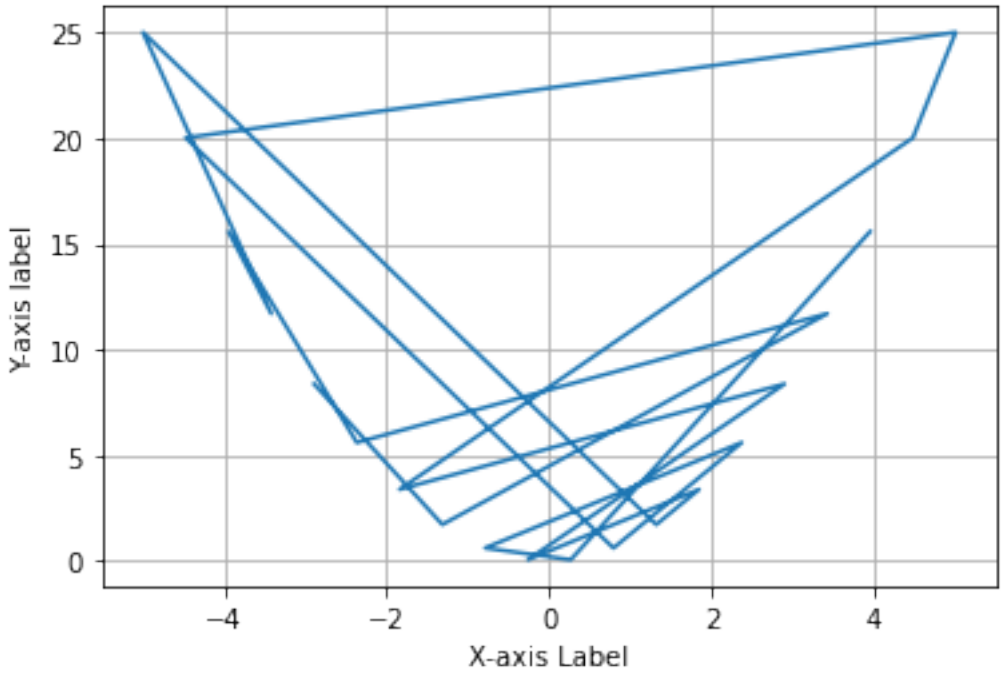


What happens if the independent variable is not sorted before plotting? Try plotting directly using the following defined array.

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


x = np.linspace(-5, 5, num=20)
np.random.shuffle(x)
y = np.array([j ** 2 for j in x])
```

[14]:
```
import numpy as np
import matplotlib.pyplot as plt


x = np.linspace(-5, 5, num=20)
np.random.shuffle(x)
y = np.array([j ** 2 for j in x])

plt.plot(x,y)
# plt.scatter(x,y)
plt.xlabel("X-axis Label")
plt.ylabel("Y-axis label")
plt.grid()
```

```
plt.show()
```



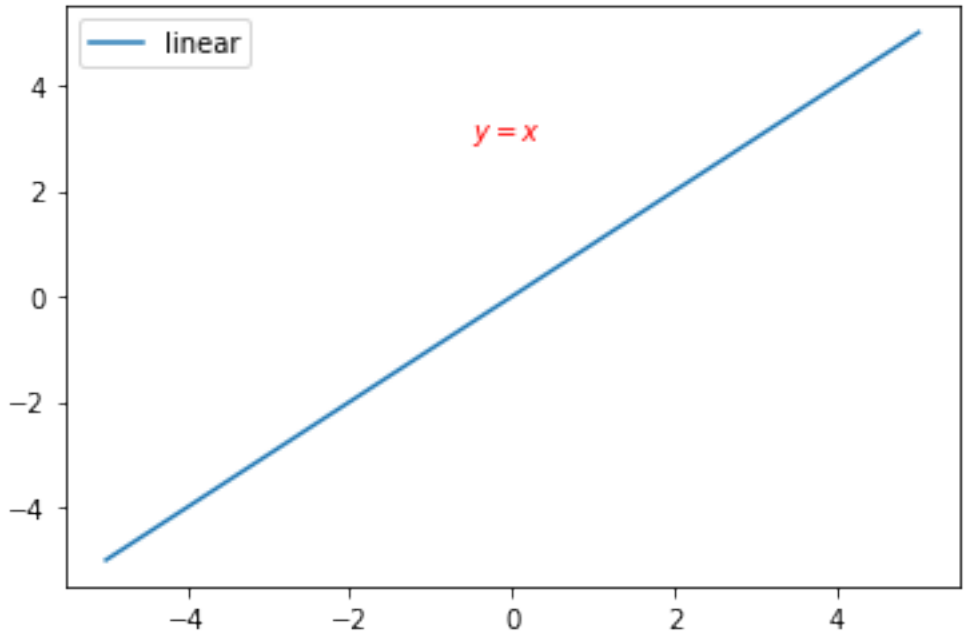
(b) Create multiple arrays and plot them with different styles, add legends, add text/mathematical equations on the plot.

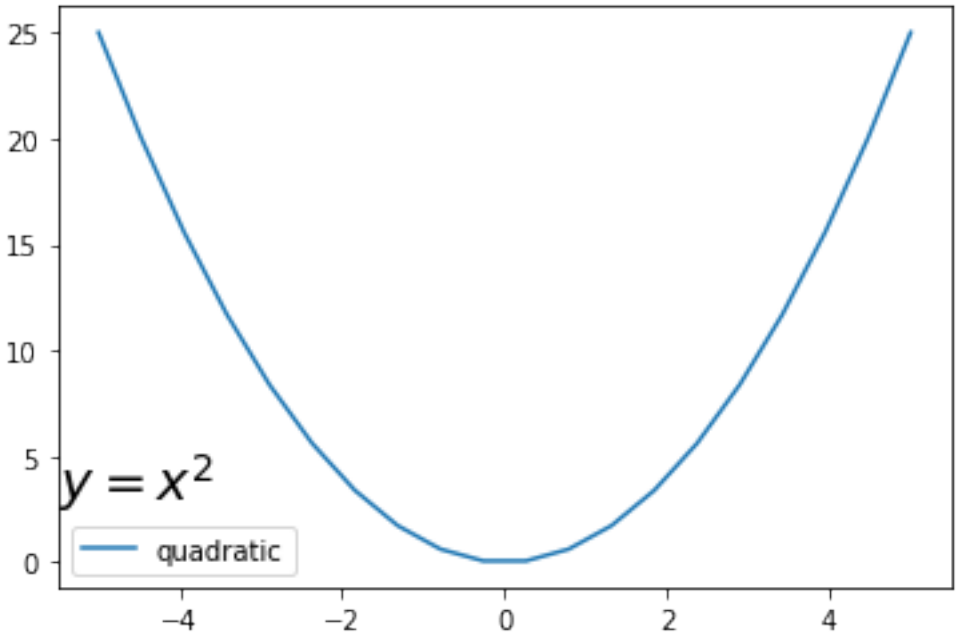
```
import numpy as np
import matplotlib.pyplot as plt


x = np.linspace(-5, 5, num=20)
y1 = np.array([j for j in x])
y2 = np.array([j ** 2 for j in x])
y3 = np.array([j ** 3 for j in x])
```

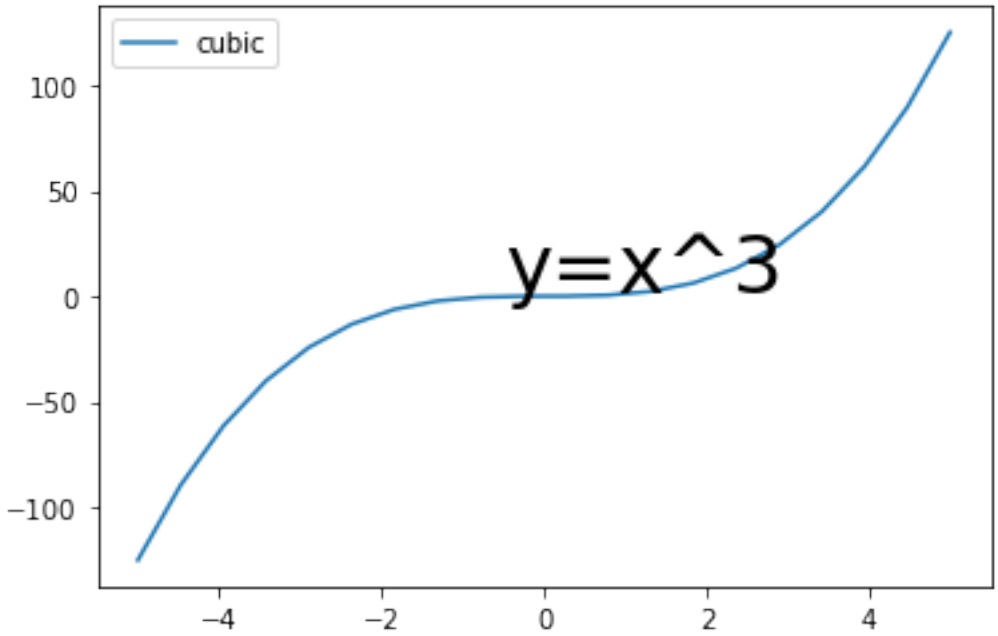
copy the code above to the following cell and add code for plotting curve $(x, y1)$, $(x, y2)$ and $(x, y3)$.

[15]:
```
import numpy as np
import matplotlib.pyplot as plt


x = np.linspace(-5, 5, num=20)
y1 = np.array([j for j in x])
y2 = np.array([j ** 2 for j in x])
y3 = np.array([j ** 3 for j in x])

plt.plot(x, y1, label='linear')
```

```
plt.legend(loc="best")
plt.text(-0.5, 3, '$y=x$',color='red')
plt.show()

plt.plot(x, y2, label='quadratic')
plt.legend(loc="best")
plt.text(-5.5, 3, '$y=x^2$', fontsize=20)
plt.show()

plt.plot(x, y3, label='cubic')
plt.legend(loc="best")
plt.text(-0.5, 3, 'y=x^3', fontsize=30)
plt.show()
```







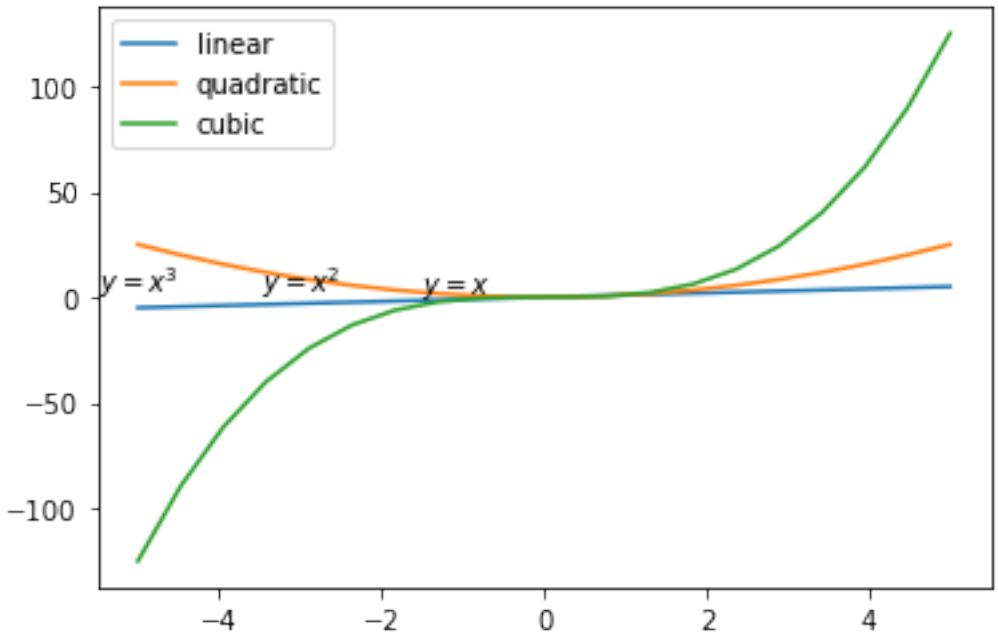
(c) Create multiple arrays and plot them into one figure **(No multiple figure and no subplot is allowed in this question)**.

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

```
x = np.linspace(-5, 5, num=20)
y1 = np.array([j for j in x])
y2 = np.array([j ** 2 for j in x])
y3 = np.array([j ** 3 for j in x])
```

copy the code above to the following cell and add code for plotting curve $(x, y1)$, $(x, y2)$ and $(x, y3)$.

[16]:
```
import numpy as np
import matplotlib.pyplot as plt


x = np.linspace(-5, 5, num=20)
y1 = np.array([j for j in x])
y2 = np.array([j ** 2 for j in x])
y3 = np.array([j ** 3 for j in x])

plt.plot(x,y1, label='linear')
plt.plot(x,y2, label='quadratic')
plt.plot(x,y3, label='cubic')
plt.text(-1.5, 3, '$y=x$', fontsize=10)
plt.text(-3.5, 3, '$y=x^2$', fontsize=10)
plt.text(-5.5, 3, '$y=x^3$', fontsize=10)
plt.legend(loc='best')
plt.show()
```



(d) Create multiple subplots, play around with the figure size, figure title, and its font style and font size **(One curve is plotted in one subplot in this question)**.

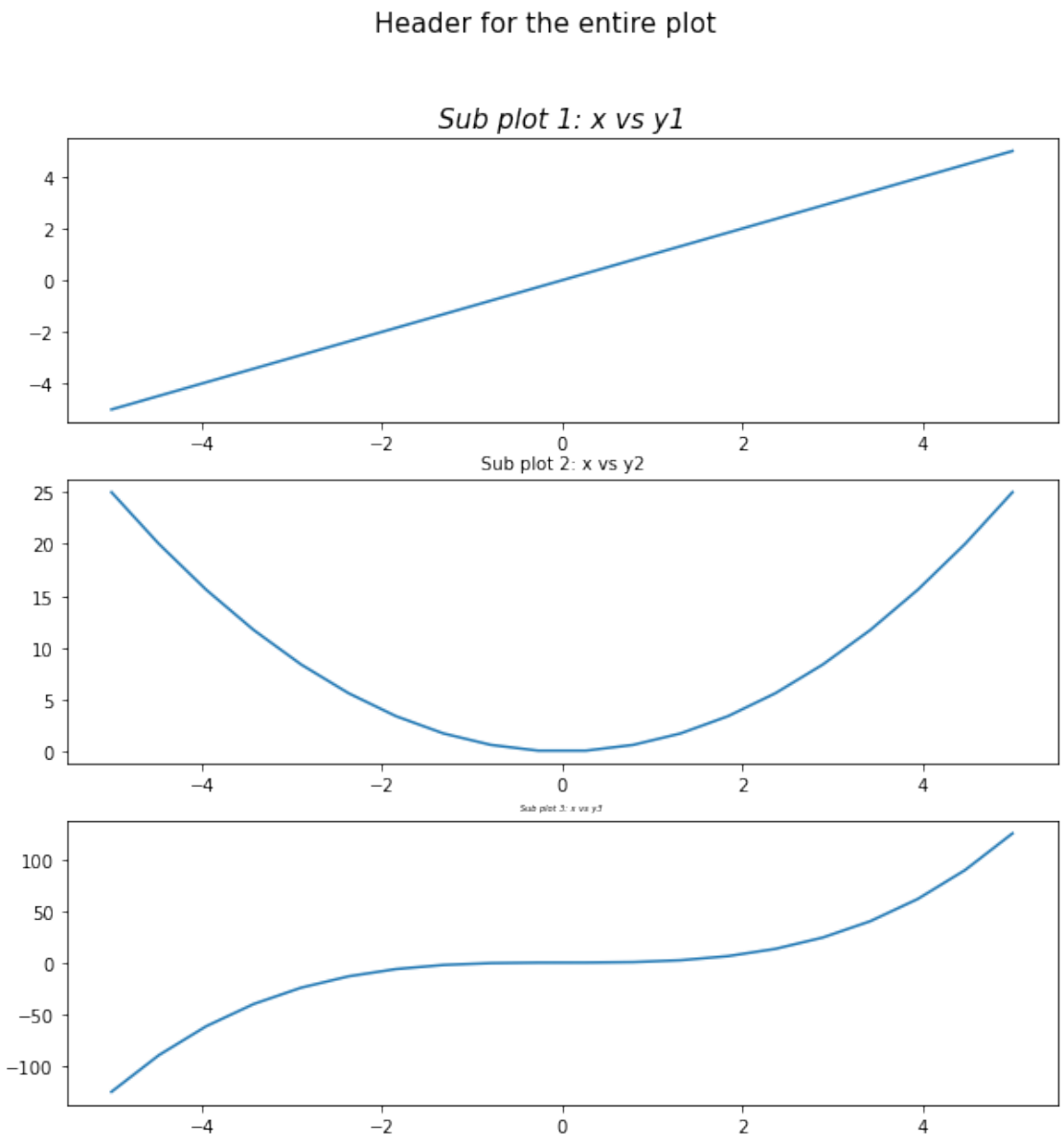
```
import numpy as np
import matplotlib.pyplot as plt


x = np.linspace(-5, 5, num=20)
y1 = np.array([j for j in x])
y2 = np.array([j ** 2 for j in x])
y3 = np.array([j ** 3 for j in x])
```

copy the code above to the following cell and add code for plotting curve $(x, y1)$, $(x, y2)$ and $(x, y3)$.

[17]:
```
import numpy as np
import matplotlib.pyplot as plt


x = np.linspace(-5, 5, num=20)
y1 = np.array([j for j in x])
y2 = np.array([j ** 2 for j in x])
y3 = np.array([j ** 3 for j in x])

fig,axes=plt.subplots(nrows=3,ncols=1,figsize=(10,10))
print(fig)
print(axes)


# Title for the entire plot
fig.suptitle("Header for the entire plot",fontsize=15)

# Edit the subplots
axes[0].set_title("Sub plot 1: x vs y1",fontsize=15,fontstyle='oblique')
axes[0].plot(x,y1)


axes[1].set_title("Sub plot 2: x vs y2",fontsize=10,fontstyle='normal')
axes[1].plot(x,y2)


axes[2].set_title("Sub plot 3: x vs y3",fontsize=5,fontstyle='italic')
axes[2].plot(x,y3)
```

```
Figure(720x720)
[<AxesSubplot:> <AxesSubplot:> <AxesSubplot:>]
```

[17]:
```
[<matplotlib.lines.Line2D at 0x132b493d0>]
```

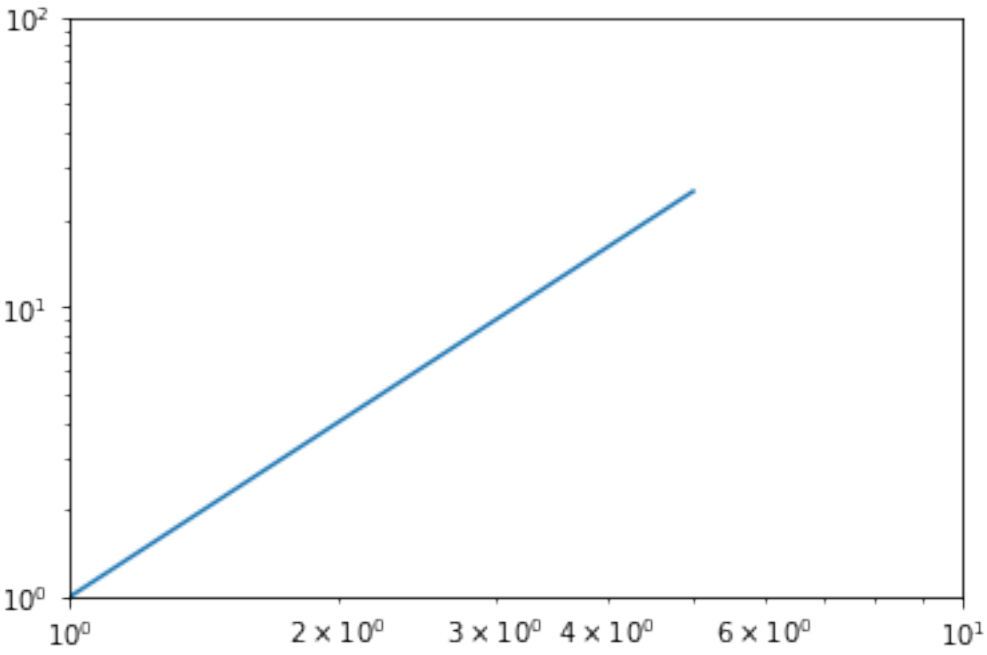


(e) Change the limits on x and y axes, **use logarithmic axes to plot**.

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


x = np.linspace(-5, 5, num=20)
y = np.array([j ** 2 for j in x])
```

copy the code above to the following cell and add code for plotting the parabola.

[18]:
```
import numpy as np
import matplotlib.pyplot as plt


x = np.linspace(-5, 5, num=20)
y = np.array([j ** 2 for j in x])


plt.xscale('log')
plt.yscale('log')
plt.ylim(ymin=1,ymax=100)
plt.xlim(xmin=1,xmax=10)

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



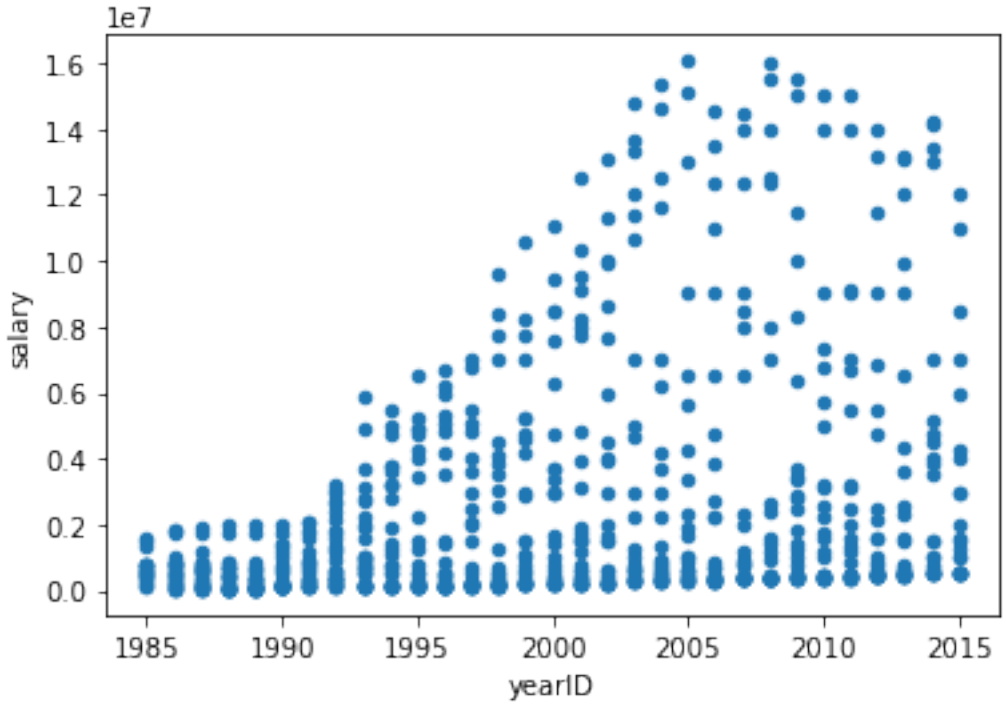
0.0.7 Pandas's DataFrame.plot and Seaborn

Pandas's DataFrame.plot Use the Salaries.csv again (You can use the dataframe object loaded from section 3.1).

(a) For team 'ATL', plot a scatter plot between feature yearID and salary.

[19]:
```
ATL = (salaries_data[salaries_data["teamID"]=="ATL"])

ATL.plot(x='yearID', y='salary', kind='scatter')
```

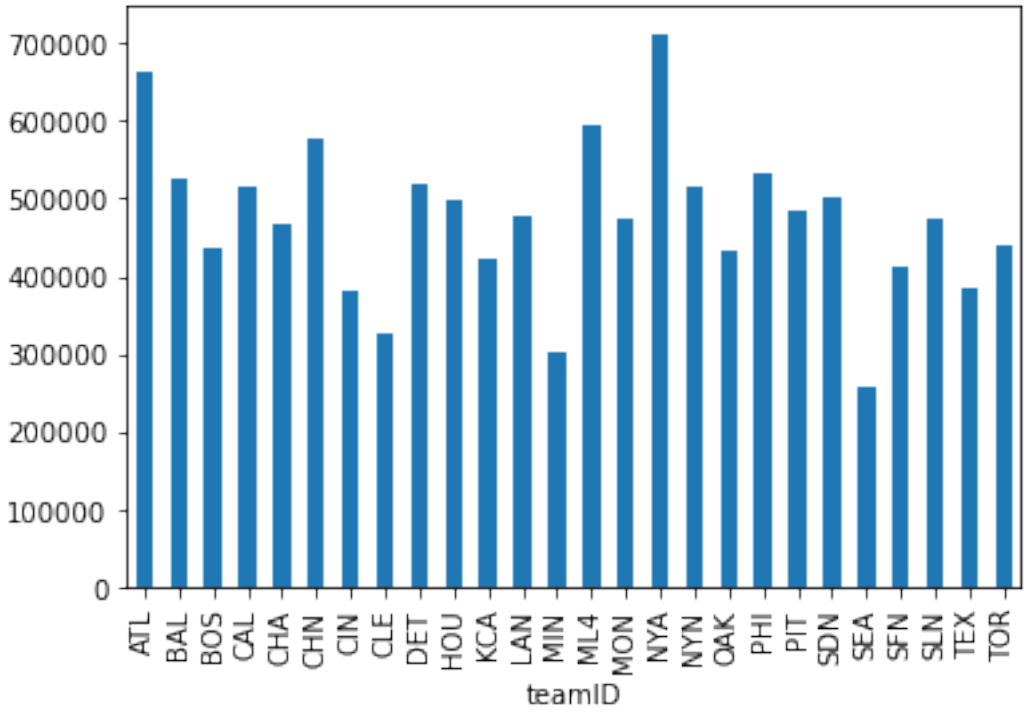
[19]: <AxesSubplot:xlabel='yearID', ylabel='salary'>



(b) For year 1985, plot a bar chart to show the average salary for each team.

[20]:
```
salaries_1985 = (salaries_data[salaries_data["yearID"]==1985])
avgsal=salaries_1985.groupby('teamID')['salary'].mean()
avgsal.plot(kind='bar')
```

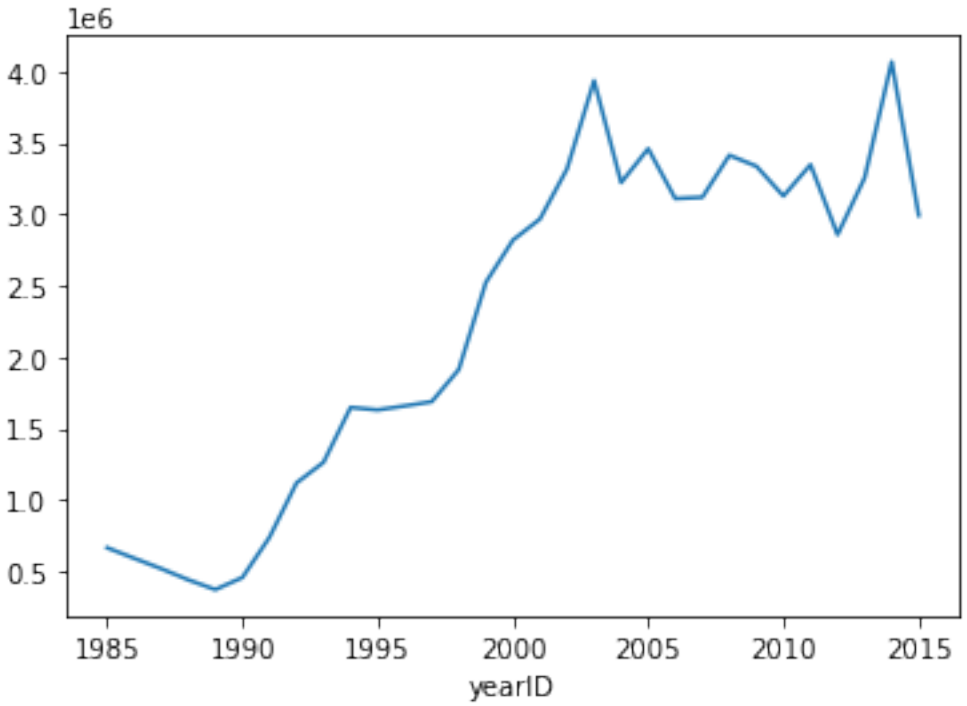
[20]: <AxesSubplot:xlabel='teamID'>



(c) For team 'ATL', plot a line chart to show how the annual average salary change by years.

[21]:
```
ATL = (salaries_data[salaries_data["teamID"]=="ATL"])
avgsal=ATL.groupby('yearID')['salary'].mean()

avgsal.plot(kind='line')
# avgsal.plot()
```

[21]:
```
<AxesSubplot:xlabel='yearID'>
```

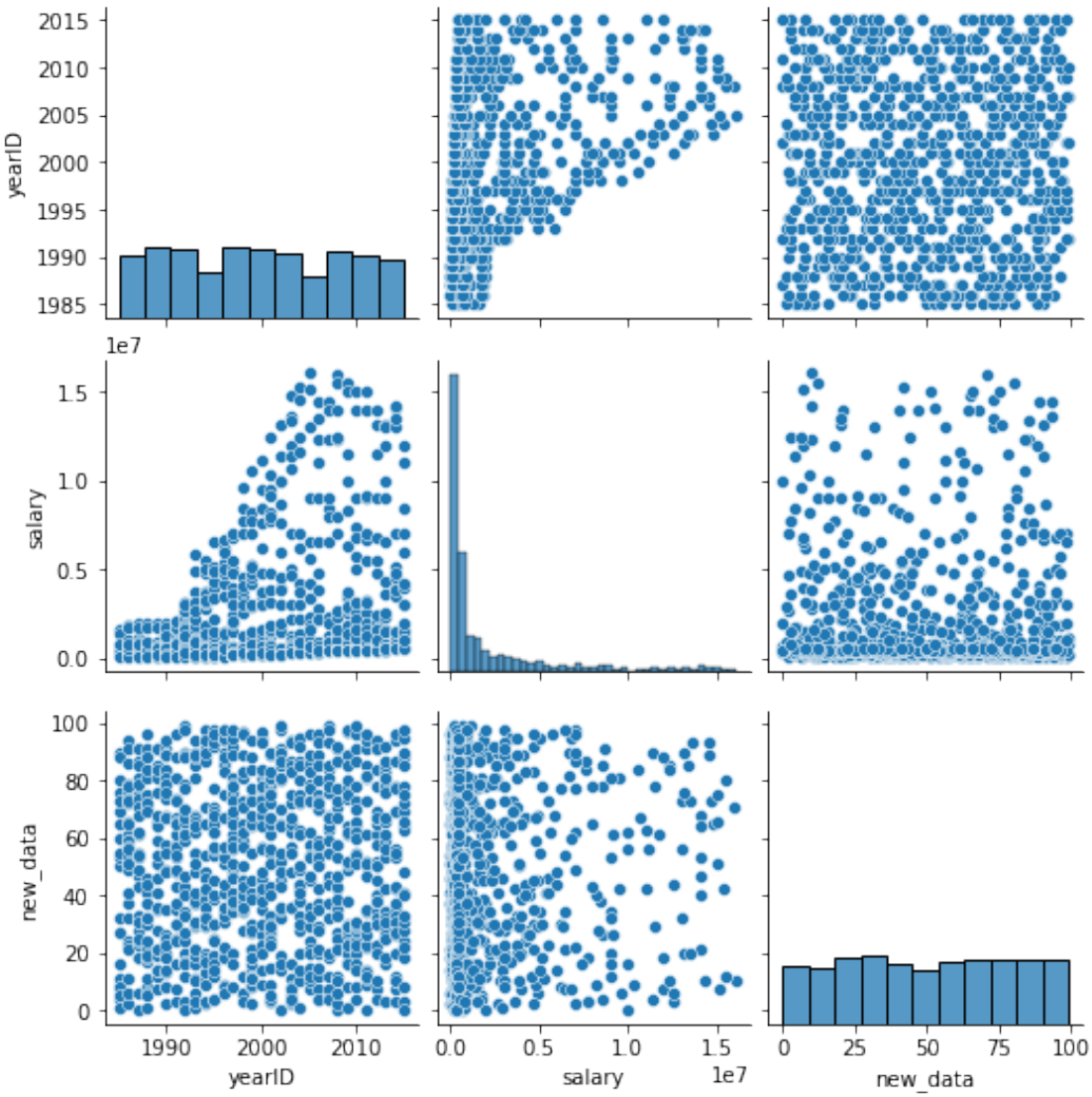


Seaborn

(a) Append one more numeric feature to the data frame (can be generated randomly), then for team 'ATL', use the seaborn.pairplot to plot scatter plots among all numeric features in the data frame for team.

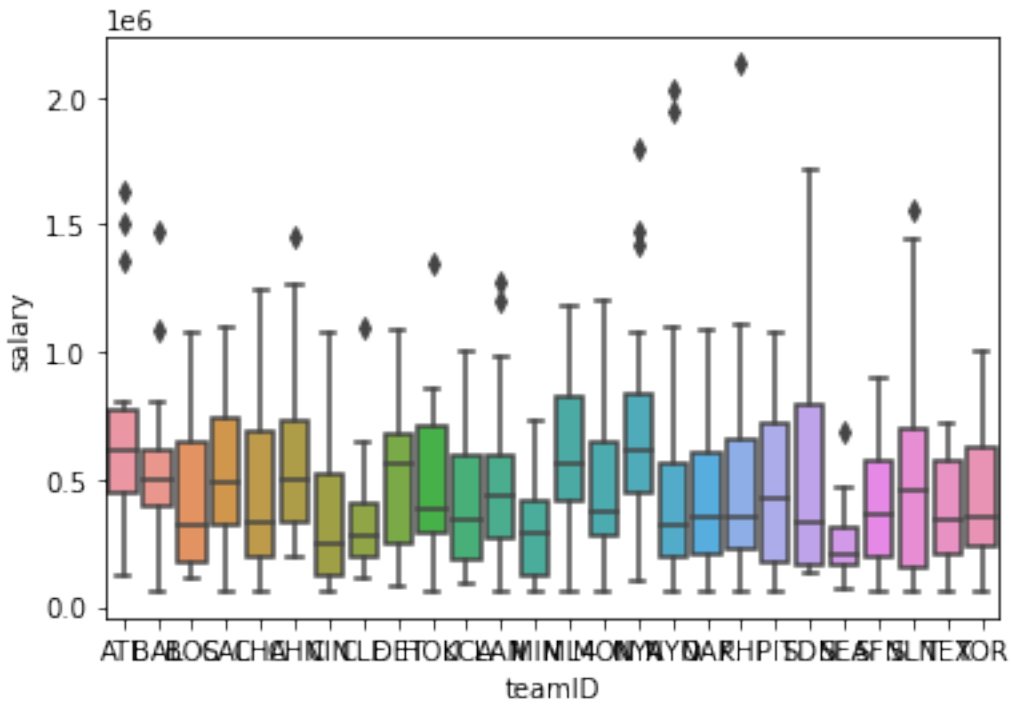
[22]:
```
salaries_data['new_data'] = np.random.randint(0,100, size=salaries_data.
 ↪shape[0])

ATL = (salaries_data[salaries_data["teamID"]=="ATL"])
# ATL['Arandom'] = np.random.randint(0,5, size=len(ATL))

import seaborn as sns
sns.pairplot(ATL)
```

[22]:
```
<seaborn.axisgrid.PairGrid at 0x168222850>
```



(b) For year 1985 and for each team, plot a boxplot to show how the salary distribute within a team.

[23]:
```
salaries_1985 = (salaries_data[salaries_data["yearID"]==1985])
bp = sns.boxplot(x="teamID", y="salary", data=salaries_1985)
```



(c) Read the offical documentation (https://seaborn.pydata.org/) to understand how lmplot, catplot, relplot, and jointplot works.

0.0.8 Jupyter Notebook

Jupyter Notebook Extensions Extensions such as the code formatter, table of content is to make your development more efficient. To explore it, please refer to https://github.com/ipython-contrib/jupyter_contrib_nbextensions.

Jupyter Visual Debugger The Pixie Debugger is a visual debugger for debugging on Jupyter Notebook. To explore it, please refer to https://medium.com/codait/the-visual-python-debugger-for-jupyter-notebooks-youve-always-wanted-761713babc62.

0.0.9 Git and GitHub

1. In the directory that where this jupyter notebook file locates in, init a Git repository.
2. Checkout a new branch called dev and commit the current notebook within this branch.
3. Merge the dev branch to the master branch (the default branch).
4. Create a temporary repository (just for practicing and you can delete it later) in GitHub.
5. Push new changes in the master branch to the remote repository created in step 4.
6. Checkout the dev branch again and do some changes to your notebook, and then repeat step 3 and step 5.

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: