Python for Data Analysis

Research Computing Services

Website: rcs.bu.edu (http://www.bu.edu/tech/support/research/)
Tutorial materials: http://rcs.bu.edu/examples/python/data_analysis (http://rcs.bu.edu/examples/python/data_analysis)

In [1]:

```
#Import Python Libraries
import numpy as np
import pandas as pd
```

Pandas is a python package that deals mostly with :

- **Series** (1d homogeneous array)
- **DataFrame** (2d labeled heterogeneous array)
- **Panel** (general 3d array)

Pandas Series

Pandas *Series* is one-dimentional labeled array containing data of the same type (integers, strings, floating point numbers, Python objects, etc.). The axis labels are often referred to as *index*.

In [22]:

```
# Example of creating Pandas series :
s1 = pd.Series(np.random.randn(5) )
print(s1)
```

```
0   -1.066232
1   -0.389708
2    0.026830
3    0.502178
4    0.877520
dtype: float64
```

We did not pass any index, so by default, it assigned the indexes ranging from 0 to len(data)-1

In [3]:

```
# View index values
print(s1.index)
```

```
RangeIndex(start=0, stop=5, step=1)
```

In [4]:

```
# Creating Pandas series with index:
s2 = pd.Series( np.random.randn(5), index=['a', 'b', 'c', 'd', 'e'] )
print(s2)
```

```
a    0.339969
b    0.174370
c   -0.696714
d   -0.198310
e    0.616652
dtype: float64
```

In [5]:

```
# View index values
print(s2.index)
```

```
Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
```

In [6]:

```
# Create a Series from dictionary
data = {'pi': 3.1415, 'e': 2.71828}  # dictionary
print(data)
s3 = pd.Series ( data )
print(s3)
```

```
{'pi': 3.1415, 'e': 2.71828}
pi    3.14150
e     2.71828
dtype: float64
```

In [7]:

```
# reordering the elements
s4 = pd.Series ( data, index = ['e', 'pi', 'tau'])
print(s4)
```

```
e      2.71828
pi     3.14150
tau        NaN
dtype: float64
```

NAN (non a number) - is used to specify a missing value in Pandas.

In [8]:

```
# Creating a Pandas Series object from a single number:
s5 = pd.Series( 1, index = range(10), name='Ones')
print(s5)
```

```
0    1
1    1
2    1
3    1
4    1
5    1
6    1
7    1
8    1
9    1
Name: Ones, dtype: int64
```

In [9]:

```
s1
```

Out[9]:

```
0   -1.052986
1    0.411027
2    0.688874
3    0.931932
4    0.373640
dtype: float64
```

In [10]:

```
# Many ways to "slice" Pandas series (series have zero-based index by default):
print(s1)
s1[3]  # returns 4th element
```

```
0   -1.052986
1    0.411027
2    0.688874
3    0.931932
4    0.373640
dtype: float64
```

Out[10]:

```
0.9319317254255365
```

In [11]:

```
s1[:2] # First 2 elements
```

Out[11]:

```
0   -1.052986
1    0.411027
dtype: float64
```

In [12]:

```
print( s1[[2,1,0]])  # Elements out of order
```

```
2    0.688874
1    0.411027
0   -1.052986
dtype: float64
```

In [13]:

```
#Slicing series using index label (access series like a dictionary)

s4['pi']
```

Out[13]:

```
3.1415
```

In [ ]:

```
dir(s4)
```

In [15]:

```
# Series can be used as ndarray:
print("Median:" , s4.median())
```

```
Median: 2.9298900000000003
```

In [16]:

```
s1[s1 > 0]
```

Out[16]:

```
1    0.411027
2    0.688874
3    0.931932
4    0.373640
dtype: float64
```

In [17]:

```
# numpy functions can be used on series as usual:
s4[s4 > s4.median()]
```

Out[17]:

```
pi    3.1415
dtype: float64
```

In [23]:

```
# vector operations:
np.exp(s1)
```

Out[23]:

```
0    0.344303
1    0.677254
2    1.027193
3    1.652315
4    2.404927
dtype: float64
```

Exercise

In [29]:

```
# Create a series （mys） of your choice and explore it
# <your code goes here >
```

Popular Attributes and Methods:

Attribute/Method	Description
dtype	data type of values in series
empty	True if series is empty
size	number of elements
values	Returns values as ndarray
head()	First n elements
tail()	Last n elements

Pandas DataFrame

Pandas *DataFrame* is two-dimensional, size-mutable, heterogeneous tabular data structure with labeled rows and columns (axes). Can be thought of a dictionary-like container to store python Series objects.

In [22]:

```
d =  pd.DataFrame({ 'Name': pd.Series(['Alice','Bob','Chris']),
                  'Age': pd.Series([ 21,25,23]) } )
print(d)
```

```
    Name  Age
0  Alice   21
1    Bob   25
2  Chris   23
```

In [23]:

```
#Add a new column:
d['height'] = pd.Series([5.2,6.0,5.6])
d
```

Out[23]:

	Name	Age	height
0	Alice	21	5.2
1	Bob	25	6.0
2	Chris	23	5.6

In [52]:

```
#Read csv file
df = pd.read_csv("http://rcs.bu.edu/examples/python/data_analysis/Salaries.csv")
```

In [25]:

```
#Display a few first records
df.head(10)
```

Out[25]:

	rank	discipline	phd	service	sex	salary
0	Prof	B	56	49	Male	186960
1	Prof	A	12	6	Male	93000
2	Prof	A	23	20	Male	110515
3	Prof	A	40	31	Male	131205
4	Prof	B	20	18	Male	104800
5	Prof	A	20	20	Male	122400
6	AssocProf	A	20	17	Male	81285
7	Prof	A	18	18	Male	126300
8	Prof	A	29	19	Male	94350
9	Prof	A	51	51	Male	57800

Excersize

In [47]:

```
#Display first 10 records
```

In [50]:

```
#Display the last 5 records
```

In [28]:

```
#Identify the type of df object
type(df)
```

Out[28]:

```
pandas.core.frame.DataFrame
```

In [29]:

```
#Check the type of a column "salary"
df['salary'].dtype
```

Out[29]:

```
dtype('int64')
```

In [30]:

```
#List the types of all columns
df.dtypes
```

Out[30]:

```
rank          object
discipline    object
phd            int64
service        int64
sex           object
salary         int64
dtype: object
```

In [31]:

```
#List the column names
df.columns
```

Out[31]:

```
Index(['rank', 'discipline', 'phd', 'service', 'sex', 'salary'], dtype='ob
ject')
```

In [32]:

```
#List the row labels and the column names
df.axes
```

Out[32]:

```
[RangeIndex(start=0, stop=78, step=1),
 Index(['rank', 'discipline', 'phd', 'service', 'sex', 'salary'], dtype='o
bject')]
```

In [33]:

```
#Number of dimensions
df.ndim
```

Out[33]:

```
2
```

In [34]:

```
#Total number of elements in the Data Frame
df.size
```

Out[34]:

```
468
```

In [35]:

```
#Number of rows and columns
df.shape
```

Out[35]:

```
(78, 6)
```

In [44]:

```
#Output basic statistics for the numeric columns
df.describe()
```

Out[44]:

	phd	service	salary
count	78.000000	78.000000	78.000000
mean	19.705128	15.051282	108023.782051
std	12.498425	12.139768	28293.661022
min	1.000000	0.000000	57800.000000
25%	10.250000	5.250000	88612.500000
50%	18.500000	14.500000	104671.000000
75%	27.750000	20.750000	126774.750000
max	56.000000	51.000000	186960.000000

In [53]:

```
#Calculate mean for all numeric columns
df.mean()
```

Out[53]:

```
phd             19.705128
service         15.051282
salary      108023.782051
dtype: float64
```

Excersize

In [55]:

```
#Calculate the standard deviation (std() method) for all numeric columns
# <your code goes here>
```

In [57]:

```
#Calculate average of the columns in the first 50 rows
# <your code goes here>
```

Data slicing and grouping

In [58]:

```
#Extract a column by name (method 1)
df['sex'].head()
```

Out[58]:

```
0    Male
1    Male
2    Male
3    Male
4    Male
Name: sex, dtype: object
```

In [60]:

```
#Extract a column name (method 2)
df.loc[:,'sex'].head()
```

Out[60]:

```
0    Male
1    Male
2    Male
3    Male
4    Male
Name: sex, dtype: object
```

Excersize

In [61]:

```
#Calculate the basic statistics for the salary column (used describe() method)
```

In [62]:

```
#Calculate how many values in the salary column (use count() method)
```

In [63]:

```
#Calculate the average salary (use mean() method)
```

In [64]:

```
#Group data using rank
df_rank = df.groupby('rank')
```

In [65]:

```
#Calculate mean of all numeric columns for the grouped object
df_rank.mean()
```

Out[65]:

	phd	service	salary
rank			
AssocProf	15.076923	11.307692	91786.230769
AsstProf	5.052632	2.210526	81362.789474
Prof	27.065217	21.413043	123624.804348

In [77]:

```
df.groupby('sex').mean()
```

Out[77]:

	phd	service	salary
sex			
Female	16.512821	11.564103	101002.410256
Male	22.897436	18.538462	115045.153846

In [66]:

```
#Calculate the mean salary for men and women. The following produce Pandas Series (sing
le brackets around salary)
df.groupby('sex')['salary'].mean()
```

Out[66]:

```
sex
Female    101002.410256
Male      115045.153846
Name: salary, dtype: float64
```

In [67]:

```
# If we use double brackets Pandas will produce a DataFrame
df.groupby('sex')[['salary']].mean()
```

Out[67]:

	salary
sex	
Female	101002.410256
Male	115045.153846

In [68]:

```
# Group using 2 variables - sex and rank:
df.groupby(['rank','sex'], sort=True)[['salary']].mean()
```

Out[68]:

		salary
rank	**sex**	
AssocProf	**Female**	88512.800000
	Male	102697.666667
AsstProf	**Female**	78049.909091
	Male	85918.000000
Prof	**Female**	121967.611111
	Male	124690.142857

Excersize

In [80]:

```
# Group data by the discipline and find the average salary for each group
```

Filtering

In [307]:

```
#Select observation with the value in the salary column > 120K
df_sub = df[ df['salary'] > 120000]
df_sub.head()
```

Out[307]:

	rank	discipline	phd	service	sex	salary
0	Prof	B	56	49	Male	186960
36	Prof	B	45	45	Male	146856
27	Prof	A	45	43	Male	155865
40	Prof	A	39	36	Female	137000
10	Prof	B	39	33	Male	128250

In [190]:

```
df_sub.axes
```

Out[190]:

```
[Int64Index([ 0,  3,  5,  7, 10, 11, 13, 14, 15, 19, 26, 27, 29, 31, 35, 3
6, 39,
             40, 44, 45, 49, 51, 58, 72, 75],
            dtype='int64'),
 Index(['rank', 'discipline', 'phd', 'service', 'sex', 'salary'], dtype='o
bject')]
```

In [297]:

```
#Select data for female professors
df_w = df[ df['sex'] == 'Female']
df_w.head()
```

Out[297]:

	rank	discipline	phd	service	sex	salary
40	Prof	A	39	36	Female	137000
63	Prof	A	29	27	Female	91000
58	Prof	B	36	26	Female	144651
45	Prof	B	25	25	Female	140096
64	AssocProf	A	26	24	Female	73300

Excersize

In [84]:

```
# Using filtering, find the mean value of the salary for the discipline A
```

In [85]:

```
# Challange:
# Extract (filter) only observations with high salary ( > 100K) and find how many femal
e and male professors in each group
```

More on slicing the dataset

In [93]:

```
#Let's see what we get for our df_sub data frame
# Method .loc subset the data frame based on the labels:
df_sub = df.loc[10:20,['rank','sex','salary']]
df_sub
```

Out[93]:

	rank	sex	salary
10	Prof	Male	128250
11	Prof	Male	134778
12	AsstProf	Male	88000
13	Prof	Male	162200
14	Prof	Male	153750
15	Prof	Male	150480
16	AsstProf	Male	75044
17	AsstProf	Male	92000
18	Prof	Male	107300
19	Prof	Male	150500
20	AsstProf	Male	92000

In [94]:

```
#  Unlike method .loc, method iloc selects rows (and columns) by poistion:
df.iloc[10:20, [0,4,5]]
```

Out[94]:

	rank	sex	salary
10	Prof	Male	128250
11	Prof	Male	134778
12	AsstProf	Male	88000
13	Prof	Male	162200
14	Prof	Male	153750
15	Prof	Male	150480
16	AsstProf	Male	75044
17	AsstProf	Male	92000
18	Prof	Male	107300
19	Prof	Male	150500

Sorting the Data

In [96]:

```
#Sort the data frame by yrs.service and create a new data frame
df_sorted = df.sort_values(by = 'service')
df_sorted.head()
```

Out[96]:

	rank	discipline	phd	service	sex	salary
55	AsstProf	A	2	0	Female	72500
23	AsstProf	A	2	0	Male	85000
43	AsstProf	B	5	0	Female	77000
17	AsstProf	B	4	0	Male	92000
12	AsstProf	B	1	0	Male	88000

In [290]:

```
#Sort the data frame by yrs.service and overwrite the original dataset
df.sort_values(by = 'service', ascending = False, inplace = True)
df.head()
```

Out[290]:

	rank	discipline	phd	service	sex	salary
9	Prof	A	51	51	Male	57800
0	Prof	B	56	49	Male	186960
36	Prof	B	45	45	Male	146856
27	Prof	A	45	43	Male	155865
40	Prof	A	39	36	Female	137000

In [206]:

```
# Restore the original order (by sorting using index)
df.sort_index(axis=0, ascending = True, inplace = True)
df.head()
```

Out[206]:

	rank	discipline	phd	service	sex	salary
0	Prof	B	56	49	Male	186960
1	Prof	A	12	6	Male	93000
2	Prof	A	23	20	Male	110515
3	Prof	A	40	31	Male	131205
4	Prof	B	20	18	Male	104800

Excersize

In [207]:

```
# Sort data frame by the salary (in descending order) and display the first few records of the output (head)
```

In [208]:

```
#Sort the data frame using 2 or more columns:
df_sorted = df.sort_values(by = ['service', 'salary'], ascending = [True,False])
df_sorted.head(10)
```

Out[208]:

	rank	discipline	phd	service	sex	salary
52	Prof	A	12	0	Female	105000
17	AsstProf	B	4	0	Male	92000
12	AsstProf	B	1	0	Male	88000
23	AsstProf	A	2	0	Male	85000
43	AsstProf	B	5	0	Female	77000
55	AsstProf	A	2	0	Female	72500
57	AsstProf	A	3	1	Female	72500
28	AsstProf	B	7	2	Male	91300
42	AsstProf	B	4	2	Female	80225
68	AsstProf	A	4	2	Female	77500

Missing Values

In [321]:

```
# Read a dataset with missing values
flights = pd.read_csv("http://rcs.bu.edu/examples/python/data_analysis/flights.csv")
flights.head()
```

Out[321]:

	year	month	day	dep_time	dep_delay	arr_time	arr_delay	carrier	tailnum	fligh
0	2013	1	1	517.0	2.0	830.0	11.0	UA	N14228	1545
1	2013	1	1	533.0	4.0	850.0	20.0	UA	N24211	1714
2	2013	1	1	542.0	2.0	923.0	33.0	AA	N619AA	1141
3	2013	1	1	554.0	-6.0	812.0	-25.0	DL	N668DN	461
4	2013	1	1	554.0	-4.0	740.0	12.0	UA	N39463	1696

In [322]:

```
# Select the rows that have at least one missing value
flights[flights.isnull().any(axis=1)].head()
```

Out[322]:

	year	month	day	dep_time	dep_delay	arr_time	arr_delay	carrier	tailnum	fli
330	2013	1	1	1807.0	29.0	2251.0	NaN	UA	N31412	12
403	2013	1	1	NaN	NaN	NaN	NaN	AA	N3EHAA	79
404	2013	1	1	NaN	NaN	NaN	NaN	AA	N3EVAA	19
855	2013	1	2	2145.0	16.0	NaN	NaN	UA	N12221	12
858	2013	1	2	NaN	NaN	NaN	NaN	AA	NaN	13

In [211]:

```
# Filter all the rows where arr_delay value is missing:
flights1 = flights[ flights['arr_delay'].notnull( )]
flights1.head()
```

Out[211]:

	year	month	day	dep_time	dep_delay	arr_time	arr_delay	carrier	tailnum	fligh
0	2013	1	1	517.0	2.0	830.0	11.0	UA	N14228	1545
1	2013	1	1	533.0	4.0	850.0	20.0	UA	N24211	1714
2	2013	1	1	542.0	2.0	923.0	33.0	AA	N619AA	1141
3	2013	1	1	554.0	-6.0	812.0	-25.0	DL	N668DN	461
4	2013	1	1	554.0	-4.0	740.0	12.0	UA	N39463	1696

In [323]:

```
# Remove all the observations with missing values
flights2 = flights.dropna()
```

In [213]:

```
# Fill missing values with zeros
nomiss =flights['dep_delay'].fillna(0)
nomiss.isnull().any()
```

Out[213]:

```
False
```

Excersize

In [214]:

```
# Count how many missing data are in dep_delay and arr_delay columns
```

Common Aggregation Functions:

Function	Description
min	minimum
max	maximum
count	number of non-null observations
sum	sum of values
mean	arithmetic mean of values
median	median
mad	mean absolute deviation
mode	mode
prod	product of values
std	standard deviation
var	unbiased variance

In [215]:

```
# Find the number of non-missing values in each column
flights.describe()
```

Out[215]:

	year	month	day	dep_time	dep_delay	
count	160754.0	160754.000000	160754.000000	158418.000000	158418.000000	1582
mean	2013.0	6.547395	15.716567	1316.146006	9.463773	1517
std	0.0	3.410001	8.762794	470.823715	36.545109	510.6
min	2013.0	1.000000	1.000000	1.000000	-33.000000	1.000
25%	2013.0	4.000000	8.000000	855.000000	-5.000000	1112.
50%	2013.0	7.000000	16.000000	1345.000000	-2.000000	1541
75%	2013.0	10.000000	23.000000	1725.000000	7.000000	1944
max	2013.0	12.000000	31.000000	2400.000000	1014.000000	2400

In [216]:

```
# Find mean value for all the columns in the dataset
flights.min()
```

Out[216]:

```
year         2013
month           1
day             1
dep_time        1
dep_delay     -33
arr_time        1
arr_delay     -75
carrier        AA
flight          1
origin        EWR
dest          ANC
air_time       21
distance       17
hour            0
minute          0
dtype: object
```

In [217]:

```
# Let's compute summary statistic per a group':
flights.groupby('carrier')['dep_delay'].mean()
```

Out[217]:

```
carrier
AA     8.586016
AS     5.804775
DL     9.264505
UA    12.106073
US     3.782418
Name: dep_delay, dtype: float64
```

In [218]:

```
# We can use agg() methods for aggregation:
flights[['dep_delay','arr_delay']].agg(['min','mean','max'])
```

Out[218]:

	dep_delay	arr_delay
min	-33.000000	-75.000000
mean	9.463773	2.094537
max	1014.000000	1007.000000

In [219]:

```
# An example of computing different statistics for different columns
flights.agg({'dep_delay':['min','mean',max], 'carrier':['nunique']})
```

Out[219]:

	dep_delay	carrier
max	1014.000000	NaN
mean	9.463773	NaN
min	-33.000000	NaN
nunique	NaN	5.0

Basic descriptive statistics

Function	Description
min	minimum
max	maximum
mean	arithmetic mean of values
median	median
mad	mean absolute deviation
mode	mode
std	standard deviation
var	unbiased variance
sem	standard error of the mean
skew	sample skewness
kurt	kurtosis
quantile	value at %

In [220]:

```
# Convinient describe() function computes a veriety of statistics
flights.dep_delay.describe()
```

Out[220]:

```
count    158418.000000
mean          9.463773
std          36.545109
min         -33.000000
25%          -5.000000
50%          -2.000000
75%           7.000000
max        1014.000000
Name: dep_delay, dtype: float64
```

In [221]:

```
# find the index of the maximum or minimum value
# if there are multiple values matching idxmin() and idxmax() will return the first mat
ch
flights['dep_delay'].idxmin()  #minimum value
```

Out[221]:

```
54111
```

In [222]:

```
# Count the number of records for each different value in a vector
flights['carrier'].value_counts()
```

Out[222]:

```
UA    58665
DL    48110
AA    32729
US    20536
AS      714
Name: carrier, dtype: int64
```

Explore data using graphics

In [104]:

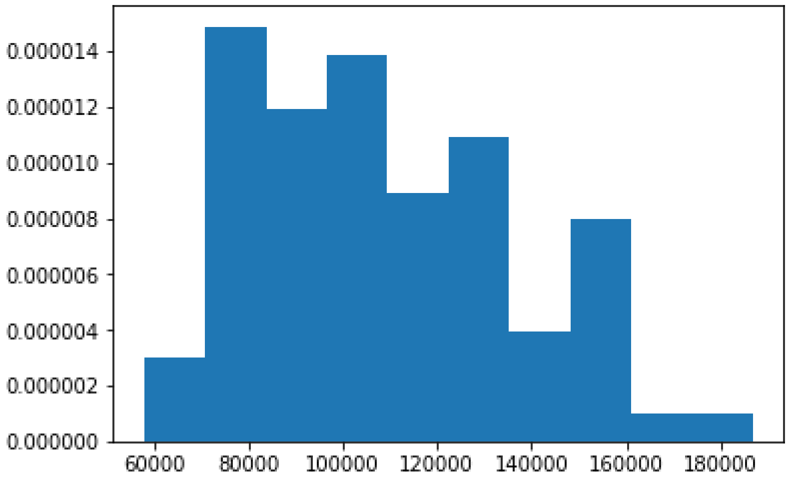
```
#Show graphs withint Python notebook
%matplotlib inline
import matplotlib.pyplot as plt
```

In [105]:

```
#Use matplotlib to draw a histogram of a salary data
plt.hist(df['salary'],bins=10, density=True)
```

Out[105]:

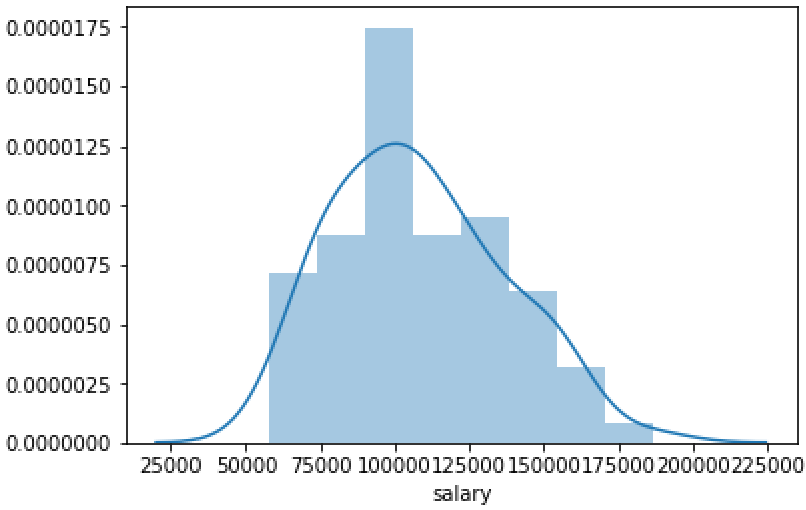
```
(array([2.97782119e-06, 1.48891059e-05, 1.19112848e-05, 1.38964989e-05,
        8.93346356e-06, 1.09186777e-05, 3.97042825e-06, 7.94085650e-06,
        9.92607063e-07, 9.92607063e-07]),
 array([ 57800.,  70716.,  83632.,  96548., 109464., 122380., 135296.,
        148212., 161128., 174044., 186960.]),
 <a list of 10 Patch objects>)
```



In [108]:

```
#Use seaborn package to draw a histogram
import seaborn as sns
sns.distplot(df['salary']);
```

```
C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462:
UserWarning: The 'normed' kwarg is deprecated, and has been replaced by th
e 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```

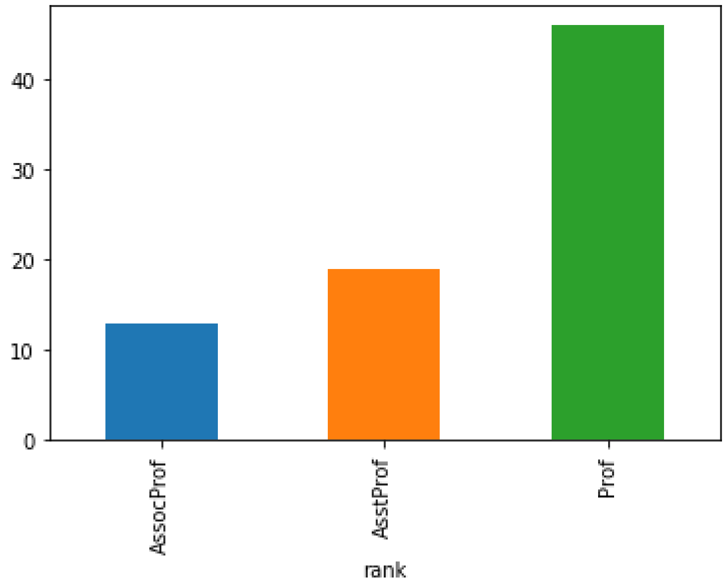


In [109]:

```
# Use regular matplotlib function to display a barplot
df.groupby(['rank'])['salary'].count().plot(kind='bar')
```

Out[109]:

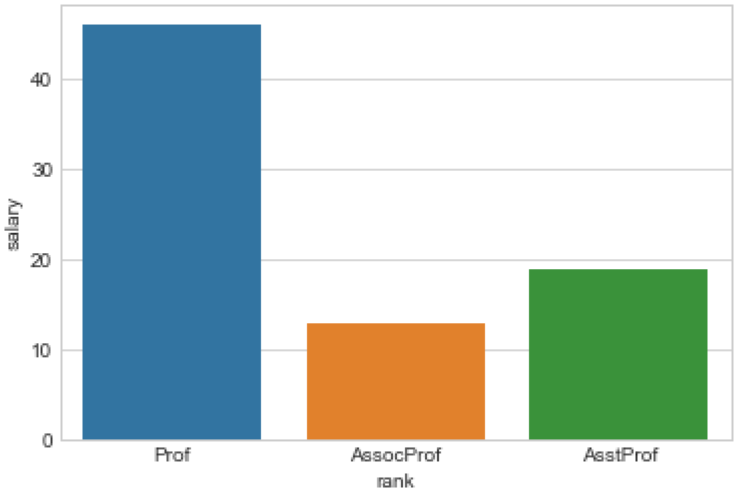
```
<matplotlib.axes._subplots.AxesSubplot at 0x1690f238eb8>
```



In [110]:

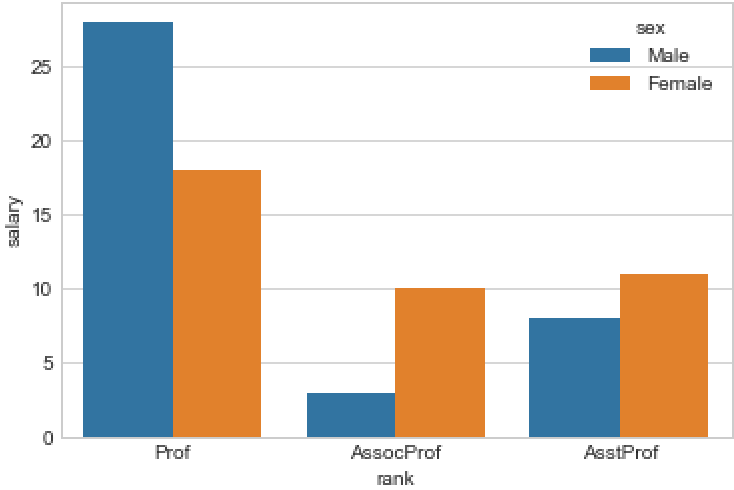
```
# Use seaborn package to display a barplot
sns.set_style("whitegrid")

ax = sns.barplot(x='rank',y ='salary', data=df, estimator=len)
```



In [111]:

```
# Split into 2 groups:
ax = sns.barplot(x='rank',y ='salary', hue='sex', data=df, estimator=len)
```

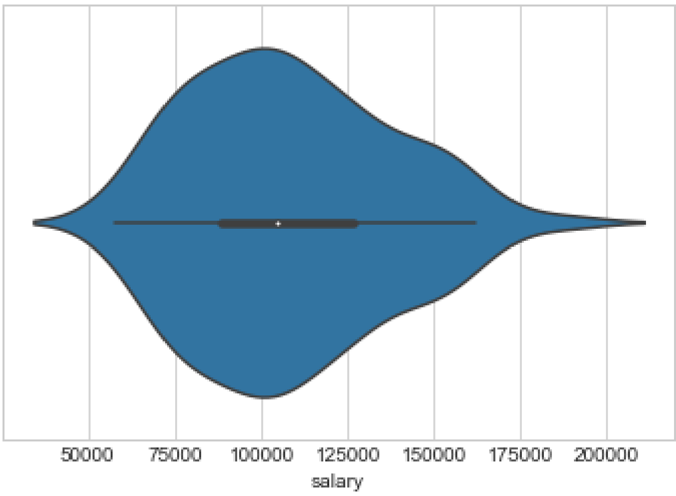


In [112]:

```
#Violinplot
sns.violinplot(x = "salary", data=df)
```

Out[112]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1690f216cf8>
```



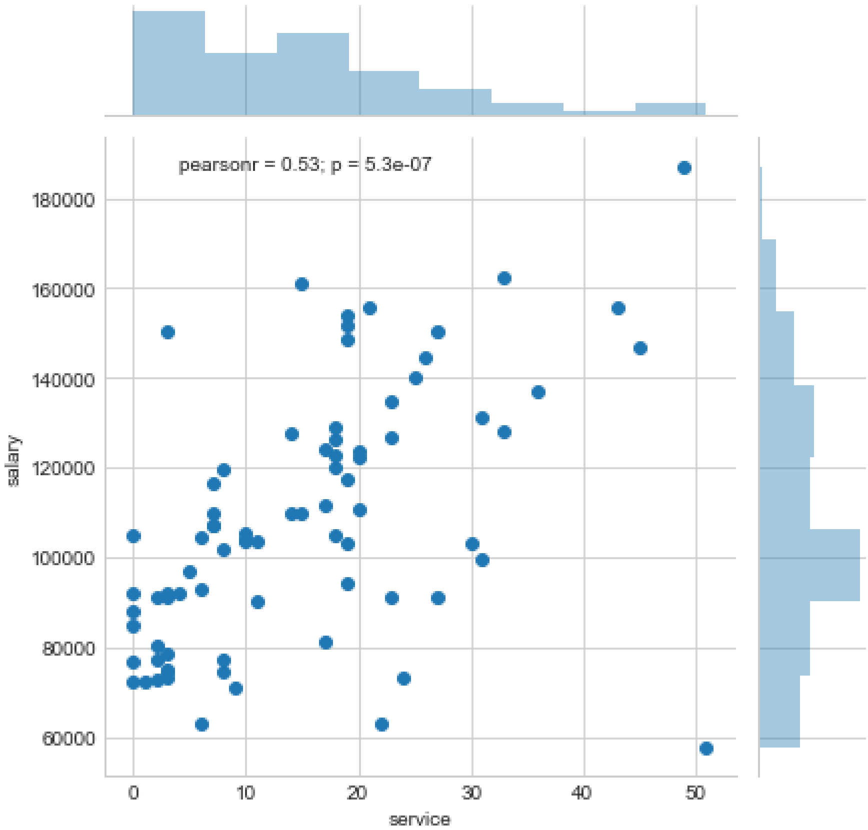
In [113]:

```
#Scatterplot in seaborn
sns.jointplot(x='service', y='salary', data=df)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462:
UserWarning: The 'normed' kwarg is deprecated, and has been replaced by th
e 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462:
UserWarning: The 'normed' kwarg is deprecated, and has been replaced by th
e 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```

Out[113]:

```
<seaborn.axisgrid.JointGrid at 0x1690f18d908>
```

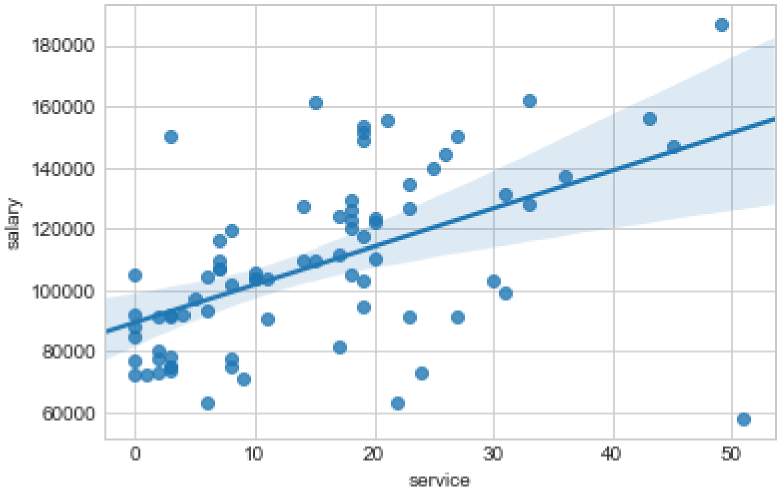


In [114]:

```
#If we are interested in linear regression plot for 2 numeric variables we can use regp
lot
sns.regplot(x='service', y='salary', data=df)
```

Out[114]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1690f2ca080>
```

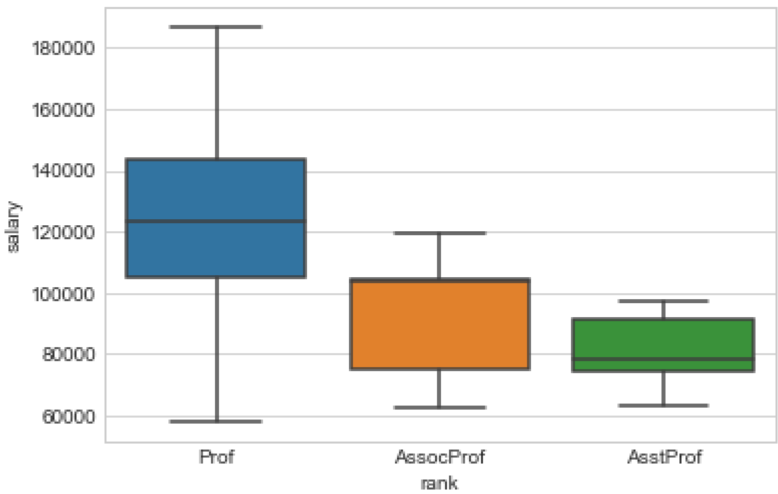


In [232]:

```
# box plot
sns.boxplot(x='rank',y='salary', data=df)
```

Out[232]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2d681bb4c18>
```

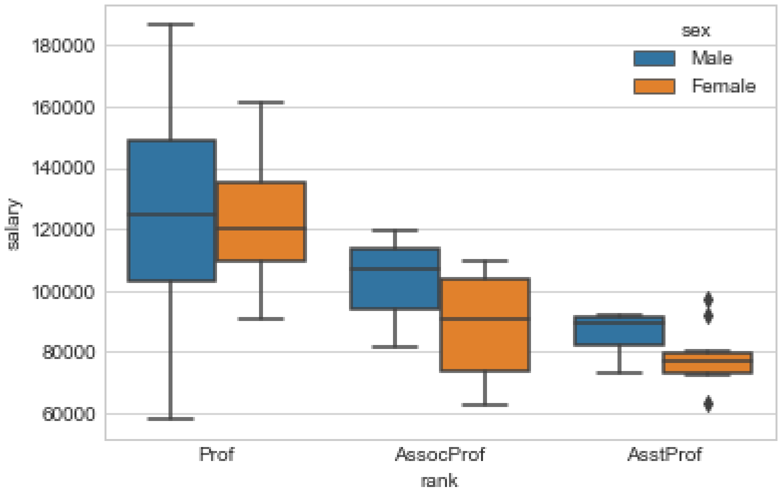


In [233]:

```
# side-by-side box plot
sns.boxplot(x='rank',y='salary', data=df, hue='sex')
```

Out[233]:

<matplotlib.axes._subplots.AxesSubplot at 0x2d681c32d30>

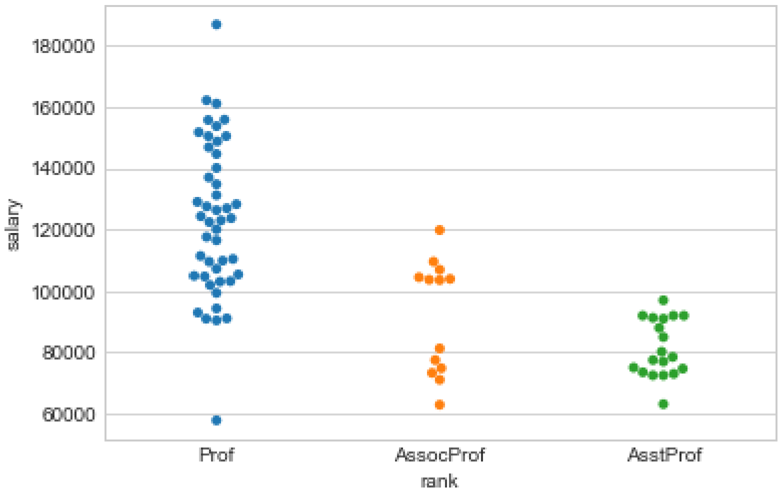


In [234]:

```
# swarm plot
sns.swarmplot(x='rank',y='salary', data=df)
```

Out[234]:

<matplotlib.axes._subplots.AxesSubplot at 0x2d681cf55c0>

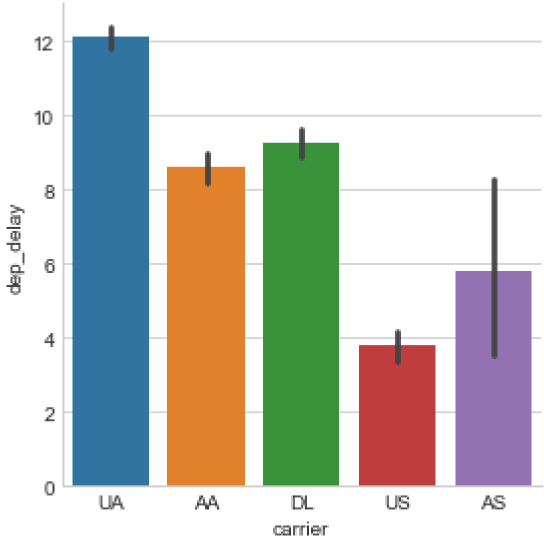


In [235]:

```
#factorplot
sns.factorplot(x='carrier',y='dep_delay', data=flights, kind='bar')
```

Out[235]:

<seaborn.axisgrid.FacetGrid at 0x2d681be5748>

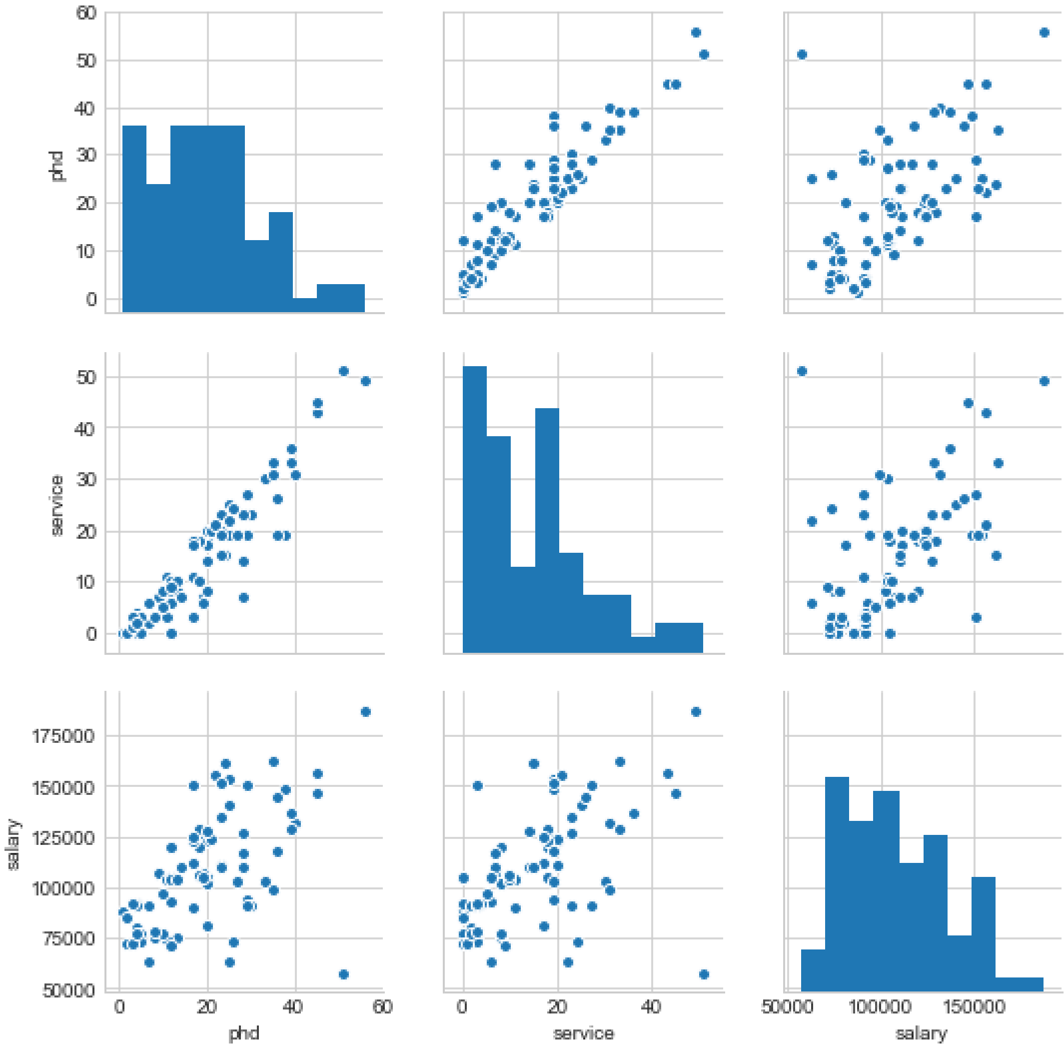


In [236]:

```
# Pairplot
sns.pairplot(df)
```

Out[236]:

```
<seaborn.axisgrid.PairGrid at 0x2d687153ef0>
```



Excersize

In [237]:

```
#Using seaborn package explore the dependency of arr_delay on dep_delay (scatterplot or
regplot) using flights dataset
```

Basic statistical Analysis

Linear Regression

In [115]:

```
# Import Statsmodel functions:
import statsmodels.formula.api as smf
```

In [116]:

```
# create a fitted model
lm = smf.ols(formula='salary ~ service', data=df).fit()

#print model summary
print(lm.summary())
```

```
                            OLS Regression Results
==========================================================================
====
Dep. Variable:                 salary   R-squared:
0.283
Model:                            OLS   Adj. R-squared:
0.274
Method:                 Least Squares   F-statistic:                     3
0.03
Date:                Fri, 25 Jan 2019   Prob (F-statistic):           5.31
e-07
Time:                        16:16:51   Log-Likelihood:                -89
6.72
No. Observations:                  78   AIC:                             1
797.
Df Residuals:                      76   BIC:                             1
802.
Df Model:                           1
Covariance Type:            nonrobust
==========================================================================
====
                 coef    std err          t      P>|t|      [0.025      0.
975]
--------------------------------------------------------------------------
----
Intercept   8.935e+04   4365.651     20.468      0.000    8.07e+04     9.8
e+04
service     1240.3567    226.341      5.480      0.000     789.560    169
1.153
==========================================================================
====
Omnibus:                       12.741   Durbin-Watson:
1.630
Prob(Omnibus):                  0.002   Jarque-Bera (JB):               2
1.944
Skew:                          -0.576   Prob(JB):                     1.72
e-05
Kurtosis:                       5.329   Cond. No.
30.9
==========================================================================
====

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is cor
rectly specified.
```

In [117]:

```
# print the coefficients
lm.params
```

Out[117]:

```
Intercept    89354.824215
service       1240.356654
dtype: float64
```

In [118]:

```
#using scikit-learn:
from sklearn import linear_model
est = linear_model.LinearRegression(fit_intercept = True)   # create estimator object
est.fit(df[['service']], df[['salary']])

#print result
print("Coef:", est.coef_, "\nIntercept:", est.intercept_)
```

```
Coef: [[1240.3566535]]
Intercept: [89354.82421525]
```

Excersize

In [242]:

```
# Build a linear model for arr_delay ~ dep_delay


#print model summary
```

Student T-test

In [119]:

```
# Using scipy package:
from scipy import stats
df_w = df[ df['sex'] == 'Female']['salary']
df_m = df[ df['sex'] == 'Male']['salary']
stats.ttest_ind(df_w, df_m)
```

Out[119]:

```
Ttest_indResult(statistic=-2.2486865976699053, pvalue=0.02742977865791010
3)
```