dataScience-with-answers

September 22, 2017

1 Python for Data Analysis

Research Computing Services Website: rcs.bu.edu Tutorial materials: http://rcs.bu.edu/examples/python/data_analysis

In [1]:
```
#Import Python Libraries
import numpy as np
import scipy as sp
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
#Read csv file
df = pd.read_csv("http://rcs.bu.edu/examples/python/data_analysis/Salaries.csv")
```

In [3]:
```
#Display a few first records
df.head()
```

Out[3]:

	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 [4]:
```
#Display first 10 records
# <your code goes here>
df.head(10)
```

Out[4]:

	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

	rank	discipline	phd	service	sex	salary
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

In [5]:
```
#Display first 20 records
# <your code goes here>
df.head(20)
```

Out[5]:

	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
10	Prof	B	39	33	Male	128250
11	Prof	B	23	23	Male	134778
12	AsstProf	B	1	0	Male	88000
13	Prof	B	35	33	Male	162200
14	Prof	B	25	19	Male	153750
15	Prof	B	17	3	Male	150480
16	AsstProf	B	8	3	Male	75044
17	AsstProf	B	4	0	Male	92000
18	Prof	A	19	7	Male	107300
19	Prof	A	29	27	Male	150500

In [6]:
```
#Display the last 5 records
# <your code goes here>
df.tail()
```

Out[6]:

	rank	discipline	phd	service	sex	salary
73	Prof	B	18	10	Female	105450
74	AssocProf	B	19	6	Female	104542
75	Prof	B	17	17	Female	124312
76	Prof	A	28	14	Female	109954
77	Prof	A	23	15	Female	109646

In [7]:
```
#Identify the type of df object
type(df)
```

Out[7]: pandas.core.frame.DataFrame

In [8]:
```
#Check the type of a column "salary"
df['salary'].dtype
```

Out[8]: dtype('int64')

In [9]:
```
#List the types of all columns
df.dtypes
```

Out[9]:
```
rank          object
discipline    object
phd            int64
service        int64
sex           object
salary         int64
dtype: object
```

In [10]:
```
#List the column names
df.columns
```

Out[10]: Index(['rank', 'discipline', 'phd', 'service', 'sex', 'salary'], dtype='object')

In [11]:
```
#List the row labels and the column names
df.axes
```

Out[11]:
```
[RangeIndex(start=0, stop=78, step=1),
 Index(['rank', 'discipline', 'phd', 'service', 'sex', 'salary'], dtype='object')]
```

In [12]:
```
#Number of dimensions
df.ndim
```

Out[12]: 2

In [13]:
```
#Total number of elements in the Data Frame
df.size
```

Out[13]: 468

In [14]:
```
#Number of rows and columns
df.shape
```

Out[14]: (78, 6)

In [15]:
```
#Output basic statistics for the numeric columns
df.describe()
```

Out[15]:

	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 [16]: #Calculate mean for all numeric columns
         df.mean()

Out[16]: phd             19.705128
         service         15.051282
         salary     108023.782051
         dtype: float64
```

Excersize

```
In [17]: #Calculate the standard deviation (std() method) for all numeric columns
         # <your code goes here>
         df.std()

Out[17]: phd            12.498425
         service        12.139768
         salary     28293.661022
         dtype: float64

In [18]: #Calculate average of the columns in the first 50 rows
         # <your code goes here>
         df.head(50).mean()

Out[18]: phd             21.52
         service         17.60
         salary     113789.14
         dtype: float64
```

1.0.1 Data slicing and grouping

```
In [19]: df_sex = df.groupby('sex')

In [20]: #Extract a column by name (method 1)
         df['sex'].head()

Out[20]: 0    Male
         1    Male
         2    Male
         3    Male
         4    Male
         Name: sex, dtype: object
```

In [21]:
```python
#Extract a column name (method 2)
df.sex.head()
```

Out[21]:
```
0    Male
1    Male
2    Male
3    Male
4    Male
Name: sex, dtype: object
```

Excersize

In [22]:
```python
#Calculate the basic statistics for the salary column (used describe() method)
# <your code goes here>
df['salary'].describe()
```

Out[22]:
```
count        78.000000
mean     108023.782051
std       28293.661022
min       57800.000000
25%       88612.500000
50%      104671.000000
75%      126774.750000
max      186960.000000
Name: salary, dtype: float64
```

In [23]:
```python
#Calculate how many values in the salary column (use count() method)
# <your code goes here>
df['salary'].count()
```

Out[23]: 78

In [24]:
```python
#Calculate the average salary
df['salary'].mean()
```

Out[24]: 108023.78205128205

In [25]:
```python
#Group data using rank
df_rank = df.groupby('rank')
```

In [26]:
```python
#Calculate mean of all numeric columns for the grouped object
df_rank.mean()
```

Out[26]:
```
                 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 [27]: #Calculate the mean salary for men and women. The following produce Pandas Series (sing
         df.groupby('sex')['salary'].mean()
```

```
Out[27]: sex
         Female    101002.410256
         Male      115045.153846
         Name: salary, dtype: float64
```

```
In [28]: # If we use double brackets Pandas will produce a DataFrame
         df.groupby('sex')[['salary']].mean()
```

```
Out[28]:                salary
         sex
         Female  101002.410256
         Male    115045.153846
```

```
In [29]: # Group using 2 variables - sex and rank:
         df.groupby(['sex','rank'], sort=False)[['salary']].mean()
```

```
Out[29]:                          salary
         sex    rank
         Male   Prof       124690.142857
                AssocProf  102697.666667
                AsstProf    85918.000000
         Female Prof       121967.611111
                AssocProf   88512.800000
                AsstProf    78049.909091
```

Excersize

```
In [30]: # Group data by the discipline and find the average salary for each group
         df.groupby('discipline')['salary'].mean()
```

```
Out[30]: discipline
         A     98331.111111
         B    116331.785714
         Name: salary, dtype: float64
```

1.0.2 Filtering

```
In [31]: #Select observation with the value in the salary column > 120K
         df_sub = df[ df['salary'] > 120000]
         df_sub.head()
```

Out[31]:
```
     rank discipline  phd  service   sex  salary
0    Prof          B   56       49  Male  186960
3    Prof          A   40       31  Male  131205
5    Prof          A   20       20  Male  122400
7    Prof          A   18       18  Male  126300
10   Prof          B   39       33  Male  128250
```

In [32]:
```
#Select data for female professors
df_w = df[ df['sex'] == 'Female']
df_w.head()
```

Out[32]:
```
         rank discipline  phd  service     sex  salary
39       Prof          B   18       18  Female  129000
40       Prof          A   39       36  Female  137000
41  AssocProf          A   13        8  Female   74830
42   AsstProf          B    4        2  Female   80225
43   AsstProf          B    5        0  Female   77000
```

Excersize

In [33]:
```
# Using filtering, find the mean value of the salary for the discipline A
df[df['discipline'] == 'A']['salary'].mean()
```

Out[33]:
```
98331.111111111109
```

In [34]:
```
# Challange:
# Extract (filter) only observations with high salary ( > 100K) and find how many femal
df[df['salary'] > 120000].groupby('sex')['salary'].count()
```

Out[34]:
```
sex
Female     9
Male      16
Name: salary, dtype: int64
```

1.0.3 More on slicing the dataset

In [35]:
```
#Select column salary
df1 = df['salary']
```

In [36]:
```
#Check data type of the result
type(df1)
```

Out[36]:
```
pandas.core.series.Series
```

In [37]:
```
#Look at the first few elements of the output
df1.head()
```

Out[37]:
```
0    186960
1     93000
2    110515
3    131205
4    104800
Name: salary, dtype: int64
```

In [38]:
```
#Select column salary and make the output to be a data frame
df2 = df[['salary']]
```

In [39]:
```
#Check the type
type(df2)
```

Out[39]:
```
pandas.core.frame.DataFrame
```

In [40]:
```
#Select a subset of rows (based on their position):
# Note 1: The location of the first row is 0
# Note 2: The last value in the range is not included
df[0:10]
```

Out[40]:

	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

In [41]:
```
#If we want to select both rows and columns we can use method .loc
df.loc[10:20,['rank', 'sex','salary']]
```

Out[41]:

	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 [42]:
```
#Let's see what we get for our df_sub data frame
# Method .loc subset the data frame based on the labels:
df_sub.loc[10:20,['rank','sex','salary']]
```

Out[42]:
```
    rank   sex  salary
10  Prof  Male  128250
11  Prof  Male  134778
13  Prof  Male  162200
14  Prof  Male  153750
15  Prof  Male  150480
19  Prof  Male  150500
```

In [43]:
```
#  Unlike method .loc, method iloc selects rows (and columns) by poistion:
df_sub.iloc[10:20, [0,3,4,5]]
```

Out[43]:
```
    rank  service     sex  salary
26  Prof       19    Male  148750
27  Prof       43    Male  155865
29  Prof       20    Male  123683
31  Prof       21    Male  155750
35  Prof       23    Male  126933
36  Prof       45    Male  146856
39  Prof       18  Female  129000
40  Prof       36  Female  137000
44  Prof       19  Female  151768
45  Prof       25  Female  140096
```

1.0.4 Sorting the Data

In [44]:
```
#Sort the data frame by yrs.service and create a new data frame
df_sorted = df.sort_values(by = 'service')
df_sorted.head()
```

Out[44]:
```
        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 [45]:
```
#Sort the data frame by yrs.service and overwrite the original dataset
df.sort_values(by = 'service', ascending = False, inplace = True)
df.head()
```

Out[45]:
```
    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 [46]:
```
# Restore the original order (by sorting using index)
df.sort_index(axis=0, ascending = True, inplace = True)
df.head()
```

```
Out[46]:    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 [47]: # Sort data frame by the salary (in descending order) and display the first few records
         df.sort_values(by='salary', ascending=False).head()
```

```
Out[47]:     rank discipline  phd  service     sex  salary
        0    Prof          B   56       49    Male  186960
        13   Prof          B   35       33    Male  162200
        72   Prof          B   24       15  Female  161101
        27   Prof          A   45       43    Male  155865
        31   Prof          B   22       21    Male  155750
```

```
In [48]: #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[48]:         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
```

1.0.5 Missing Values

```
In [49]: # Read a dataset with missing values
         flights = pd.read_csv("http://rcs.bu.edu/examples/python/data_analysis/flights.csv")
         flights.head()
```

```
Out[49]:    year  month  day  dep_time  dep_delay  arr_time  arr_delay carrier tailnum  \
        0   2013      1    1     517.0        2.0     830.0       11.0      UA  N14228
        1   2013      1    1     533.0        4.0     850.0       20.0      UA  N24211
        2   2013      1    1     542.0        2.0     923.0       33.0      AA  N619AA
        3   2013      1    1     554.0       -6.0     812.0      -25.0      DL  N668DN
        4   2013      1    1     554.0       -4.0     740.0       12.0      UA  N39463
```

```
   flight origin dest  air_time  distance  hour  minute
0    1545    EWR  IAH     227.0      1400   5.0    17.0
1    1714    LGA  IAH     227.0      1416   5.0    33.0
2    1141    JFK  MIA     160.0      1089   5.0    42.0
3     461    LGA  ATL     116.0       762   5.0    54.0
4    1696    EWR  ORD     150.0       719   5.0    54.0
```

In [50]:
```
# Select the rows that have at least one missing value
flights[flights.isnull().any(axis=1)].head()
```

Out[50]:
```
     year  month  day  dep_time  dep_delay  arr_time  arr_delay carrier  \
330  2013      1    1    1807.0       29.0    2251.0        NaN      UA
403  2013      1    1       NaN        NaN       NaN        NaN      AA
404  2013      1    1       NaN        NaN       NaN        NaN      AA
855  2013      1    2    2145.0       16.0       NaN        NaN      UA
858  2013      1    2       NaN        NaN       NaN        NaN      AA

    tailnum  flight origin dest  air_time  distance  hour  minute
330  N31412    1228    EWR  SAN       NaN      2425  18.0     7.0
403  N3EHAA     791    LGA  DFW       NaN      1389   NaN     NaN
404  N3EVAA    1925    LGA  MIA       NaN      1096   NaN     NaN
855  N12221    1299    EWR  RSW       NaN      1068  21.0    45.0
858     NaN     133    JFK  LAX       NaN      2475   NaN     NaN
```

In [51]:
```
# Filter all the rows where arr_delay value is missing:
flights1 = flights[ flights['arr_delay'].notnull( )]
flights1.head()
```

Out[51]:
```
   year  month  day  dep_time  dep_delay  arr_time  arr_delay carrier tailnum  \
0  2013      1    1     517.0        2.0     830.0       11.0      UA  N14228
1  2013      1    1     533.0        4.0     850.0       20.0      UA  N24211
2  2013      1    1     542.0        2.0     923.0       33.0      AA  N619AA
3  2013      1    1     554.0       -6.0     812.0      -25.0      DL  N668DN
4  2013      1    1     554.0       -4.0     740.0       12.0      UA  N39463

   flight origin dest  air_time  distance  hour  minute
0    1545    EWR  IAH     227.0      1400   5.0    17.0
1    1714    LGA  IAH     227.0      1416   5.0    33.0
2    1141    JFK  MIA     160.0      1089   5.0    42.0
3     461    LGA  ATL     116.0       762   5.0    54.0
4    1696    EWR  ORD     150.0       719   5.0    54.0
```

In [52]:
```
# Remove all the observations with missing values
flights2 = flights.dropna()
```

In [53]:
```
# Fill missing values with zeros
nomiss =flights['dep_delay'].fillna(0)
nomiss.isnull().any()
```

```
Out[53]: False
```

Excersize

```
In [54]: # Count how many missing data are in dep_delay and arr_delay columns
         flights[['dep_delay','arr_delay']].isnull().sum()

Out[54]: dep_delay    2336
         arr_delay    2827
         dtype: int64
```

1.0.6 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 [55]: # Find the number of non-missing values in each column
         flights.count()

Out[55]: year         160754
         month        160754
         day          160754
         dep_time     158418
         dep_delay    158418
         arr_time     158275
         arr_delay    157927
         carrier      160754
         tailnum      159321
         flight       160754
         origin       160754
         dest         160754
         air_time     157927
         distance     160754
```

```
hour          158418
minute        158418
dtype: int64
```

In [56]:
```
# Find mean value for all the columns in the dataset
flights.min()
```

Out[56]:
```
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 [57]:
```
# Let's compute summary statistic per a group':
flights.groupby('carrier')['dep_delay'].mean()
```

Out[57]:
```
carrier
AA     8.586016
AS     5.804775
DL     9.264505
UA    12.106073
US     3.782418
Name: dep_delay, dtype: float64
```

In [58]:
```
# We can use agg() methods for aggregation:
flights[['dep_delay','arr_delay']].agg(['min','mean','max'])
```

Out[58]:

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

In [59]:
```
# An example of computing different statistics for different columns
flights.agg({'dep_delay':['min','mean',max], 'carrier':['nunique']})
```

Out[59]:

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

1.0.7 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 [60]:
```
# Convinient describe() function computes a veriety of statistics
flights.dep_delay.describe()
```

Out[60]:
```
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 [61]:
```
# find the index of the maximum or minimum value
# if there are multiple values matching idxmin() and idxmax() will return the first mat
flights['dep_delay'].idxmin()  #minimum value
```

Out[61]:
```
54111
```

In [62]:
```
# Count the number of records for each different value in a vector
flights['carrier'].value_counts()
```

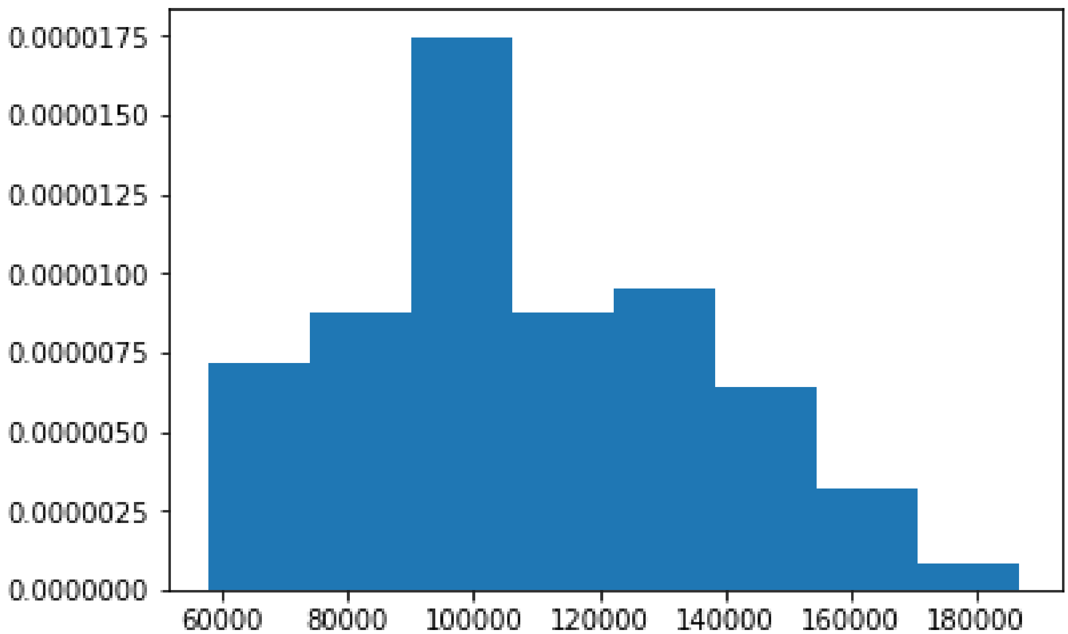
Out[62]:
```
UA    58665
DL    48110
AA    32729
US    20536
AS      714
Name: carrier, dtype: int64
```

1.0.8 Explore data using graphics

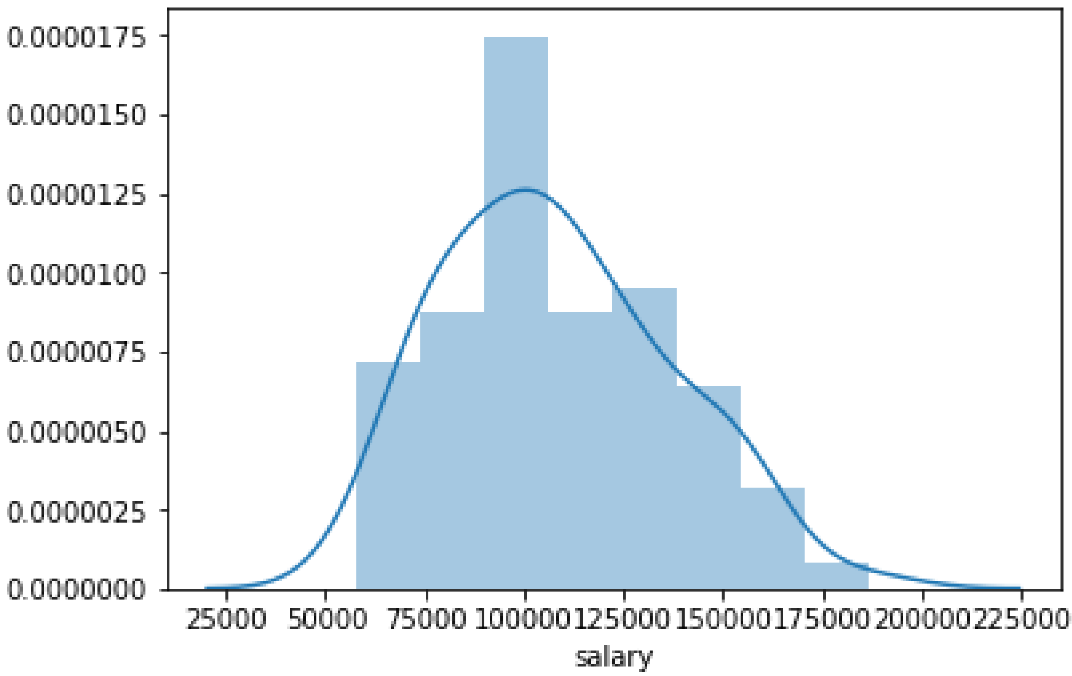
In [63]:
```
#Show graphs withint Python notebook
%matplotlib inline
```

In [64]:
```python
#Use matplotlib to draw a histogram of a salary data
plt.hist(df['salary'],bins=8, normed=1)
```

Out[64]:
```
(array([  7.14677085e-06,   8.73494215e-06,   1.74698843e-05,
          8.73494215e-06,   9.52902780e-06,   6.35268520e-06,
          3.17634260e-06,   7.94085650e-07]),
 array([  57800.,   73945.,   90090.,  106235.,  122380.,  138525.,
         154670.,  170815.,  186960.]),
 <a list of 8 Patch objects>)
```

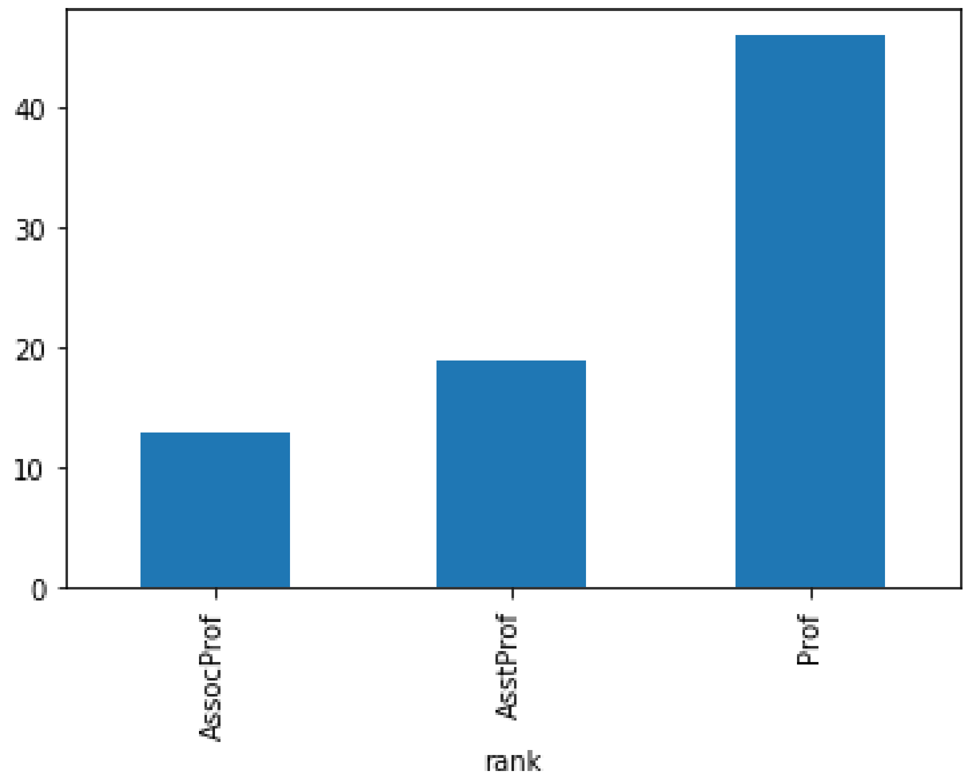


In [65]:
```python
#Use seaborn package to draw a histogram
sns.distplot(df['salary']);
```



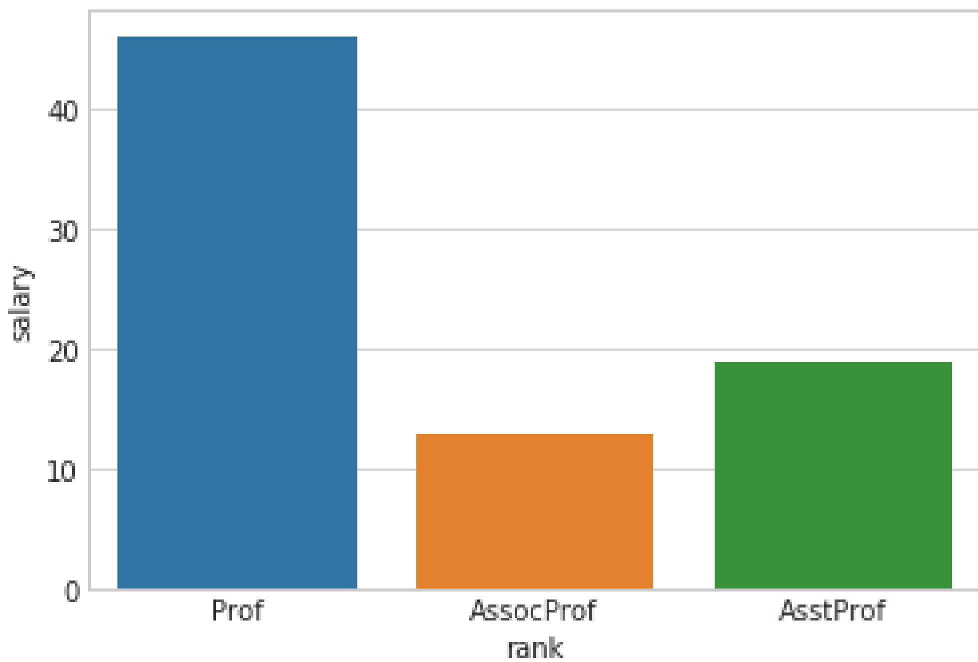

```
In [66]: # Use regular matplotlib function to display a barplot
         df.groupby(['rank'])['salary'].count().plot(kind='bar')

Out[66]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff58213f860>
```

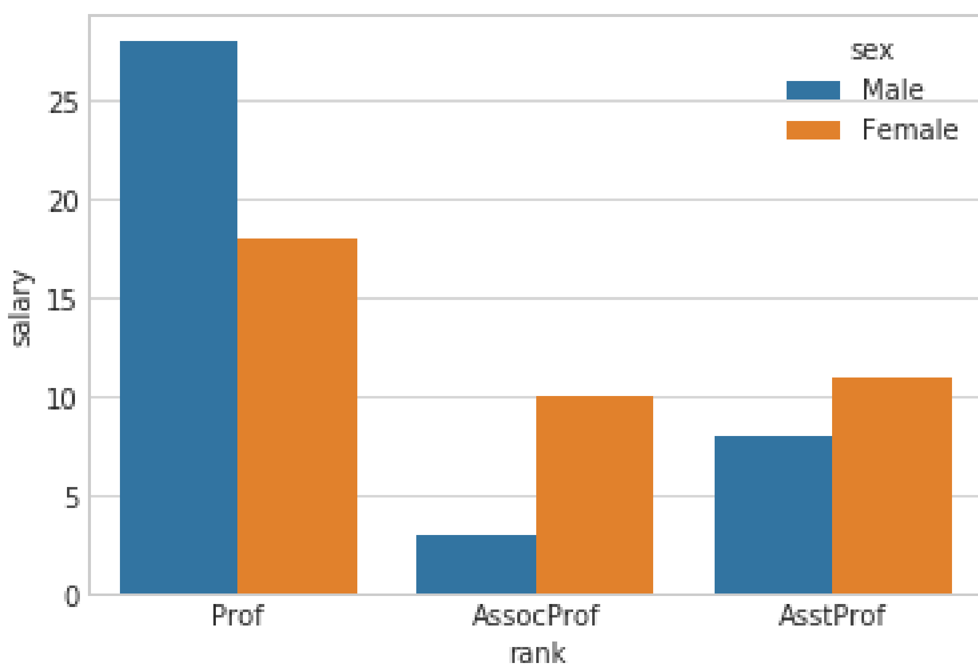



```
In [67]: # Use seaborn package to display a barplot
         sns.set_style("whitegrid")

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

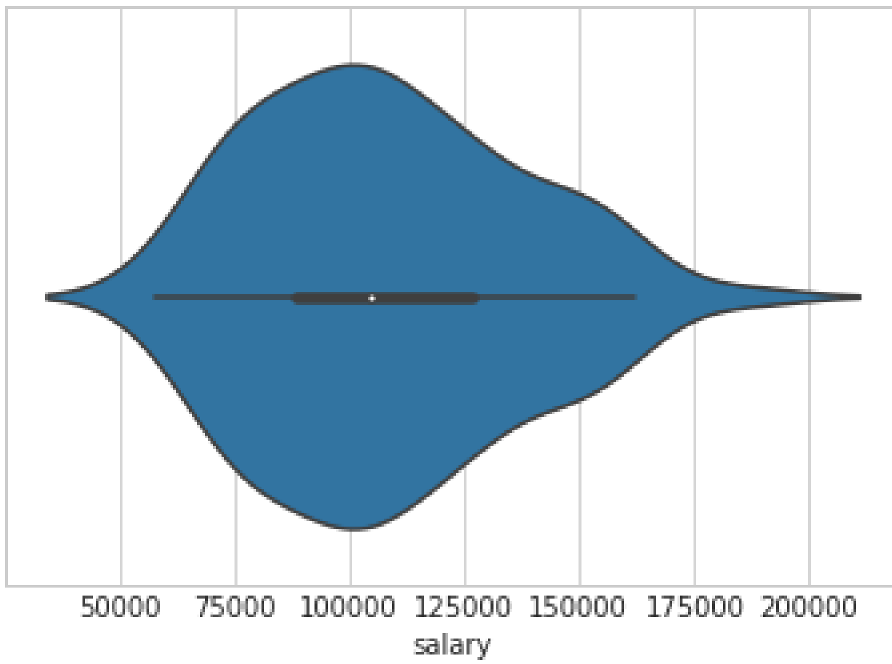


In [68]:
```
# Split into 2 groups:
ax = sns.barplot(x='rank',y ='salary', hue='sex', data=df, estimator=len)
```



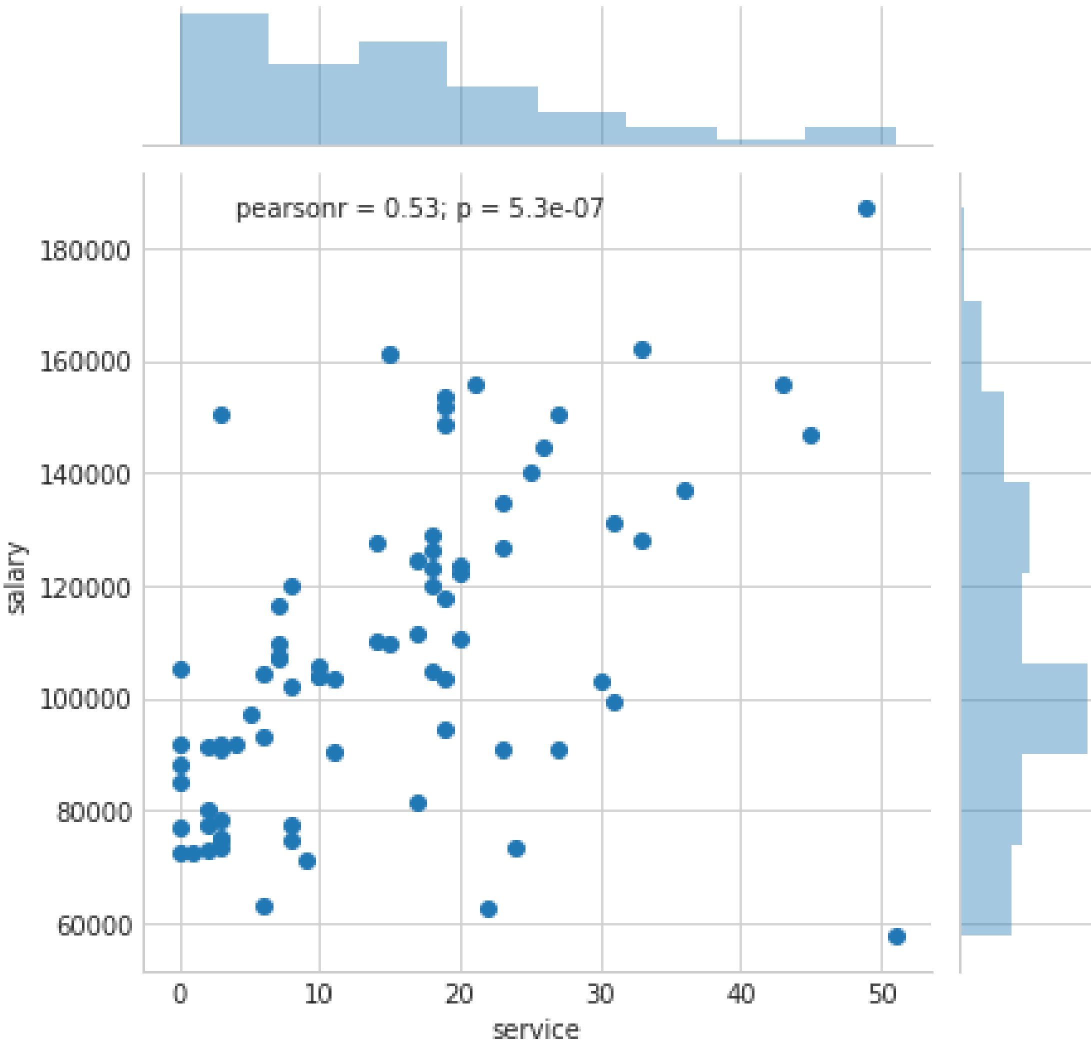
```
In [69]: #Violinplot
         sns.violinplot(x = "salary", data=df)

Out[69]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff5819b79e8>
```



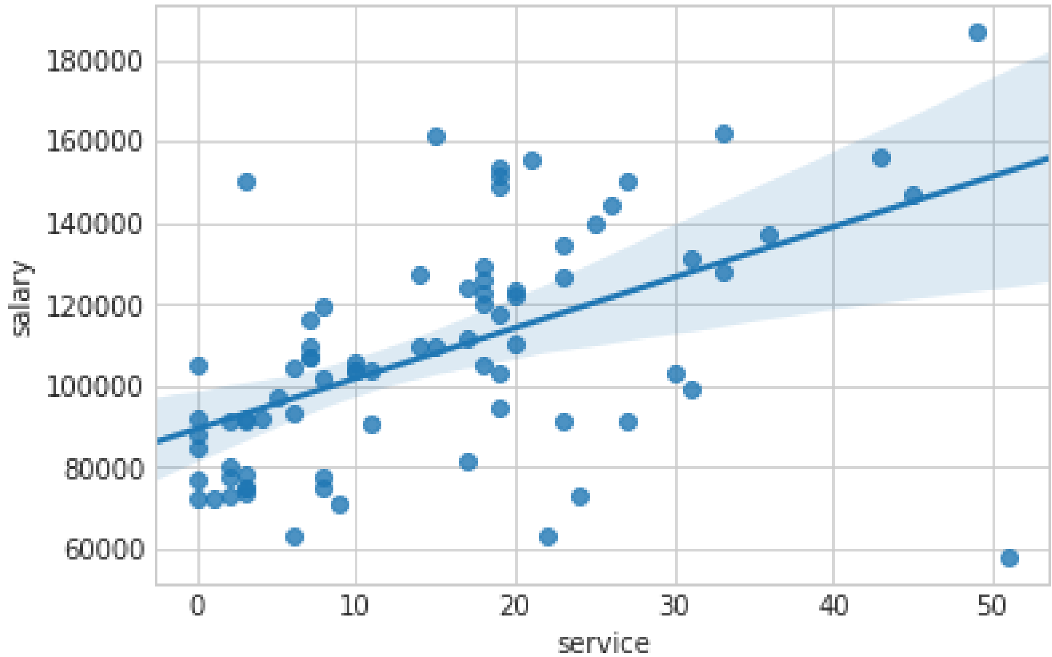

```
In [70]: #Scatterplot in seaborn
         sns.jointplot(x='service', y='salary', data=df)

Out[70]: <seaborn.axisgrid.JointGrid at 0x7ff581984550>
```



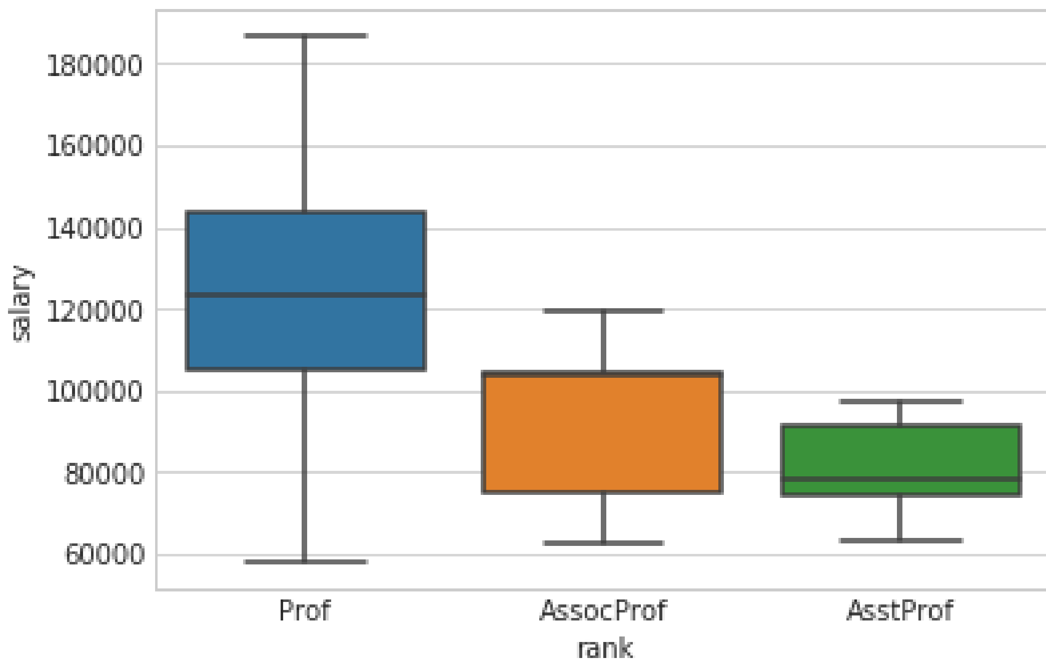

```
In [71]: #If we are interested in linear regression plot for 2 numeric variables we can use regp
         sns.regplot(x='service', y='salary', data=df)

Out[71]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff58184c470>
```



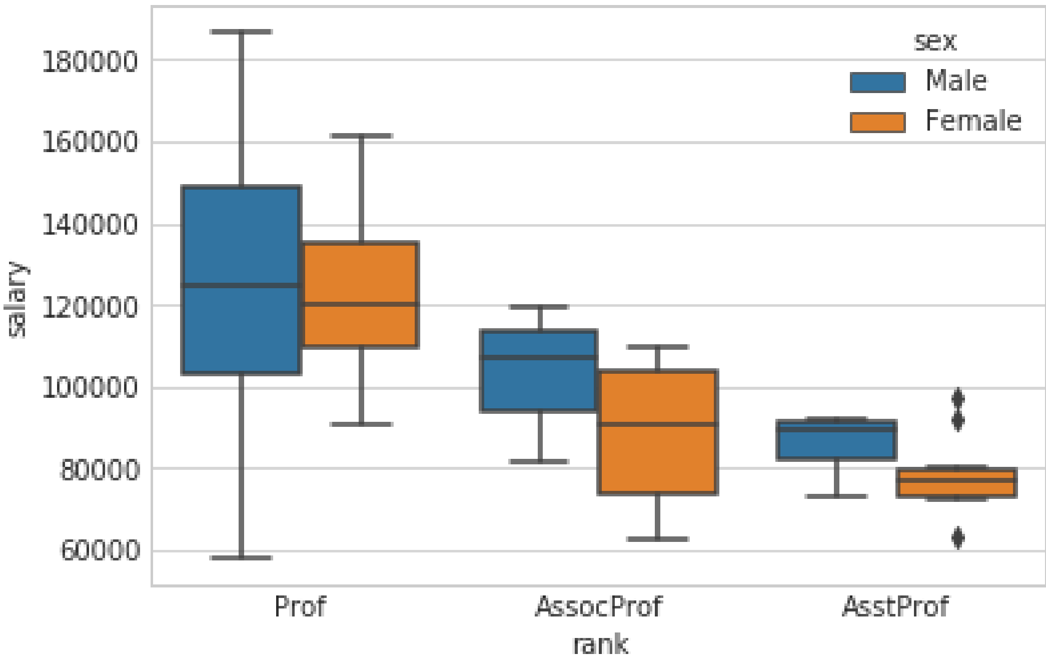
In [72]:
```
# box plot
sns.boxplot(x='rank',y='salary', data=df)
```

Out[72]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff58170de80>



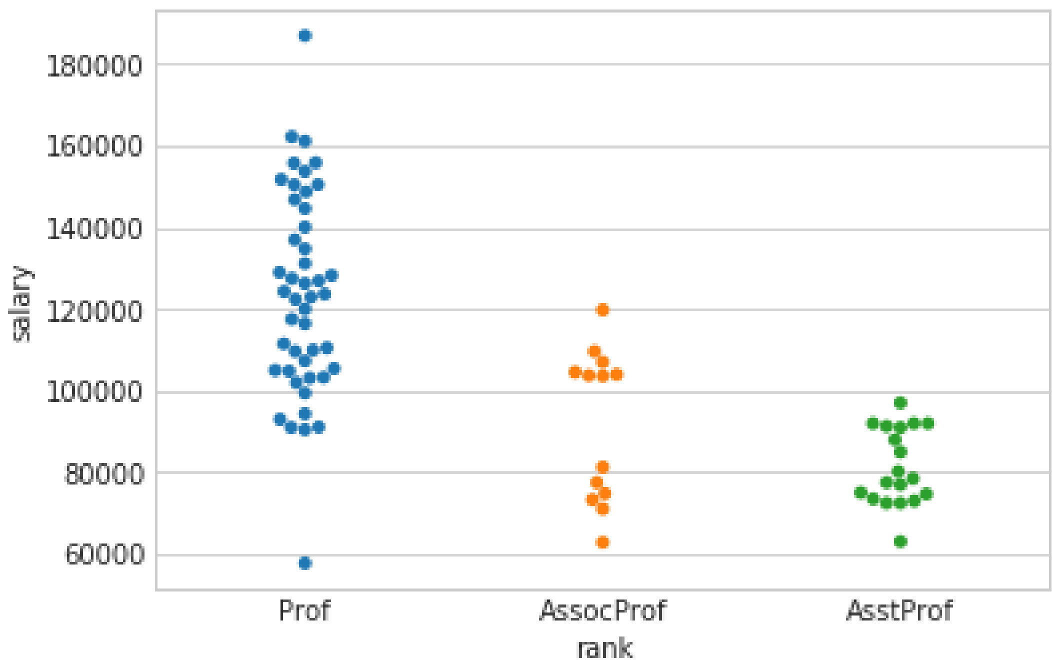
```
In [73]: # side-by-side box plot
         sns.boxplot(x='rank',y='salary', data=df, hue='sex')
```

```
Out[73]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff5818edc18>
```



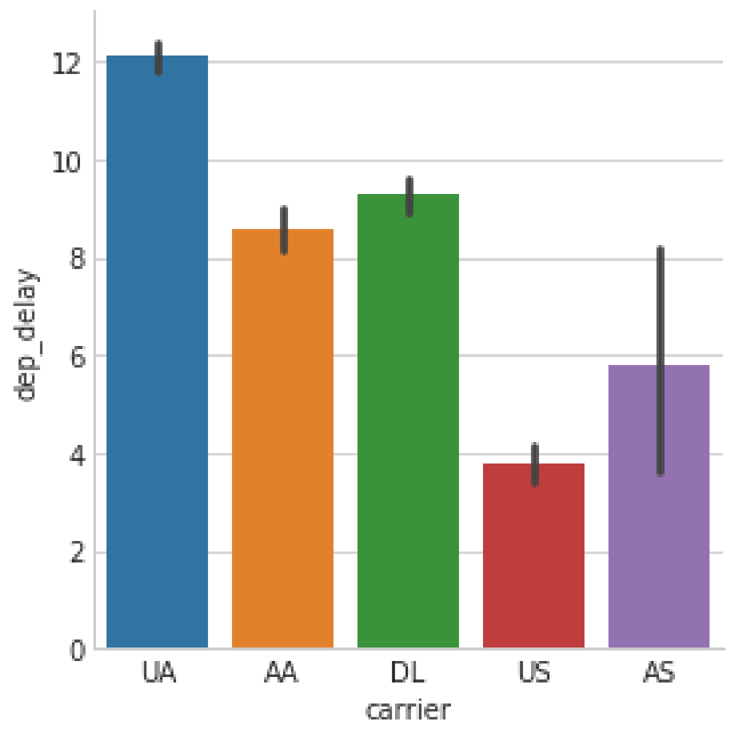

```
In [74]: # swarm plot
         sns.swarmplot(x='rank',y='salary', data=df)
```

```
Out[74]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff5814f75c0>
```



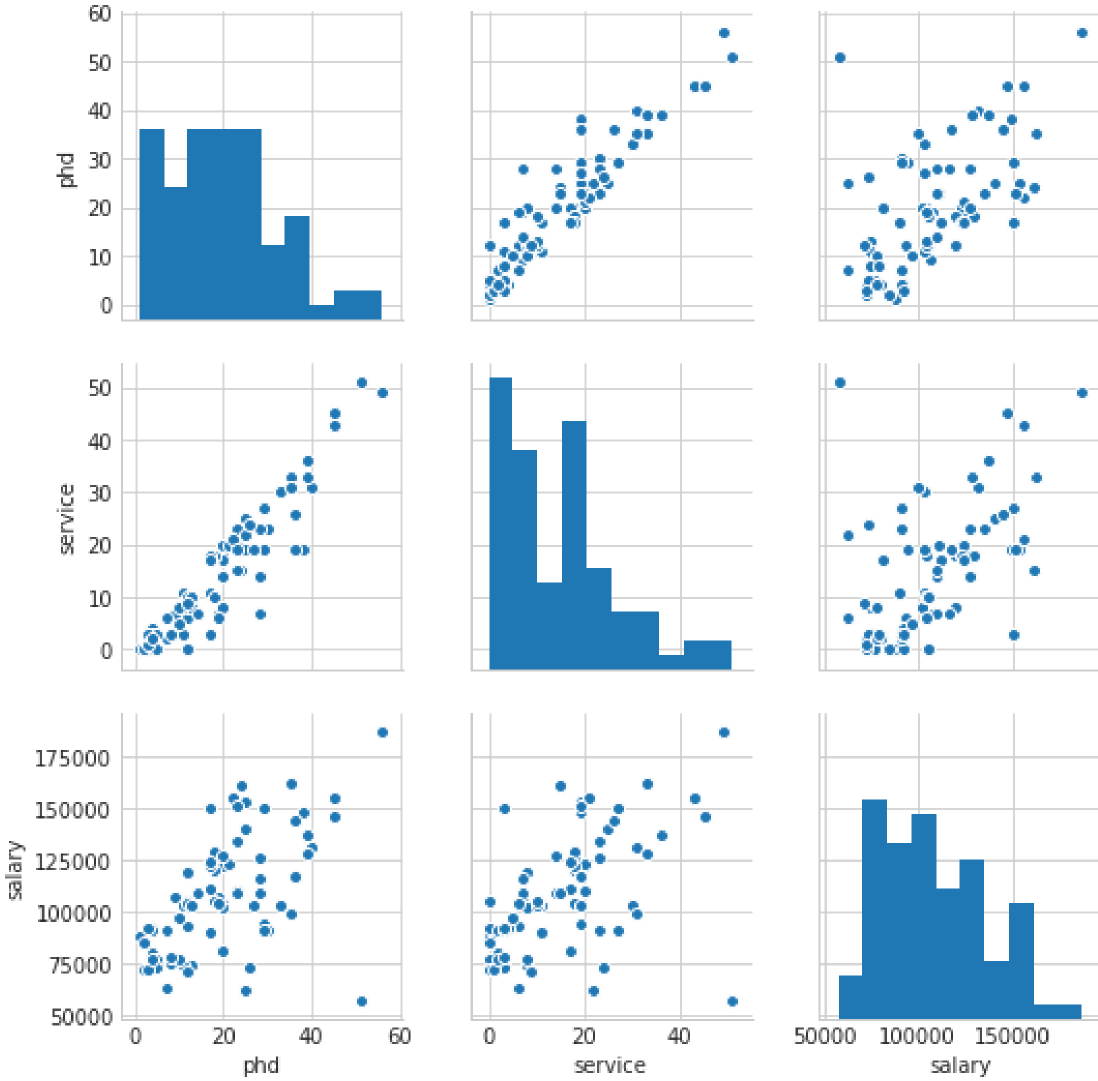

```
In [75]: #factorplot
         sns.factorplot(x='carrier',y='dep_delay', data=flights, kind='bar')

Out[75]: <seaborn.axisgrid.FacetGrid at 0x7ff58178a198>
```



```
In [76]: # Pairplot
         sns.pairplot(df)
```

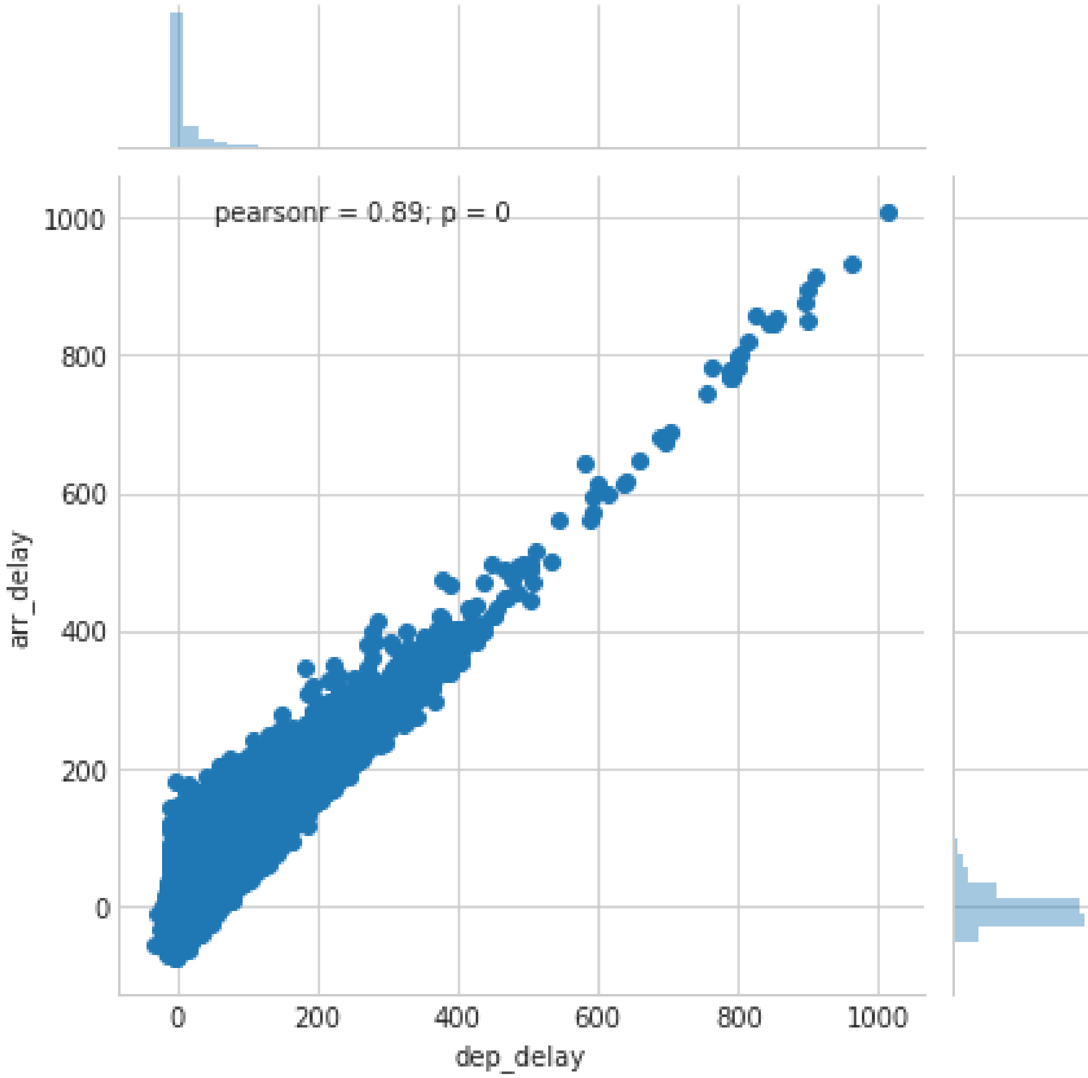
```
Out[76]: <seaborn.axisgrid.PairGrid at 0x7ff5822296a0>
```



Excersize

```
In [77]: #Using seaborn package explore the dependency of arr_delay on dep_delay (scatterplot or
         sns.jointplot(x='dep_delay', y='arr_delay', data=flights)
```

Out[77]: <seaborn.axisgrid.JointGrid at 0x7ff580cb7a20>



1.1 Basic statistical Analysis

1.1.1 Linear Regression

In [78]:
```
# Import Statsmodel functions:
import statsmodels.formula.api as smf
```

In [79]:
```
# 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:                     30.03
Date:                Fri, 15 Sep 2017   Prob (F-statistic):           5.31e-07
Time:                        14:11:46   Log-Likelihood:                -896.72
No. Observations:                  78   AIC:                             1797.
Df Residuals:                      76   BIC:                             1802.
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     9.8e+04
service     1240.3567    226.341      5.480      0.000     789.560    1691.153
==============================================================================
Omnibus:                       12.741   Durbin-Watson:                   1.630
Prob(Omnibus):                  0.002   Jarque-Bera (JB):               21.944
Skew:                          -0.576   Prob(JB):                     1.72e-05
Kurtosis:                       5.329   Cond. No.                         30.9
==============================================================================

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

```
In [80]: # print the coefficients
         lm.params
```

```
Out[80]: Intercept    89354.824215
         service       1240.356654
         dtype: float64
```

```
In [81]: #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 [82]:
```
# Build a linear model for arr_delay ~ dep_delay
lm = smf.ols(formula='arr_delay ~ dep_delay', data=flights).fit()

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

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              arr_delay   R-squared:                       0.794
Model:                            OLS   Adj. R-squared:                  0.794
Method:                 Least Squares   F-statistic:                 6.074e+05
Date:                Fri, 15 Sep 2017   Prob (F-statistic):               0.00
Time:                        14:11:47   Log-Likelihood:            -6.8778e+05
No. Observations:              157927   AIC:                         1.376e+06
Df Residuals:                  157925   BIC:                         1.376e+06
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     -7.4457      0.049   -152.050      0.000      -7.542      -7.350
dep_delay      1.0138      0.001    779.358      0.000       1.011       1.016
==============================================================================
Omnibus:                38155.693   Durbin-Watson:                   1.467
Prob(Omnibus):              0.000   Jarque-Bera (JB):           159178.104
Skew:                       1.141   Prob(JB):                         0.00
Kurtosis:                   7.357   Cond. No.                         38.9
==============================================================================

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

1.1.2 Student T-test

In [83]:
```
# 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[83]:
```
Ttest_indResult(statistic=-2.2486865976699053, pvalue=0.027429778657910103)
```