Table of Contents

Estimated Time Needed: **60 min**

Import all required Libraries

In [2]:
```
#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 [7]:
```
#Read csv file
df = pd.read_csv("Salaries.csv")
```

In [8]:
```
df.head(20)
```

Out[8]:

	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 [13]:

```
#Count the number of unique values in our data
#df.sex.value_counts(dropna=False)
#df.salary.value_counts(dropna=False)
#df.discipline.value_counts(dropna=False)
```

```
# Try for the rank field. what do you think?
df["rank"].value_counts(dropna=False)
```

Out[13]:
```
Prof         46
AsstProf     19
AssocProf    13
Name: rank, dtype: int64
```

In [14]:
```
#Data type of each column
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 78 entries, 0 to 77
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   rank        78 non-null     object
 1   discipline  78 non-null     object
 2   phd         78 non-null     int64
 3   service     78 non-null     int64
 4   sex         78 non-null     object
 5   salary      78 non-null     int64
dtypes: int64(3), object(3)
memory usage: 3.8+ KB
```

Part 2: Summary statistics and filtering

Summary statistics

In [15]:
```
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

Filtering

In [16]:
```
#Select observation with the value in the salary column > 120K
df_sub = df[ df['salary'] > 120000]
df_sub.head()
```

Out[16]:

	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 [17]:
```
# It give indexes and columns names
df_sub.axes
```

Out[17]:
```
[Int64Index([ 0,  3,  5,  7, 10, 11, 13, 14, 15, 19, 26, 27, 29, 31, 35, 36, 39,
             40, 44, 45, 49, 51, 58, 72, 75],
            dtype='int64'),
 Index(['rank', 'discipline', 'phd', 'service', 'sex', 'salary'], dtype='object')]
```

In [18]:
```
#Select data for female professors
df_female = df[ df['sex'] == 'Female']
df_female.head()
```

Out[18]:

	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

In [19]:
```
# Using filtering, find the mean value of the salary for the discipline A
df[ df['discipline'] =='A'].salary.mean()
```

Out[19]:
```
98331.11111111111
```

Part 3: Slicing the dataset

In [22]:
```
#Select column salary
df1 = df['salary']
```

In [23]:
```
#Check data type of the result
type(df1)
```

Out[23]:
```
pandas.core.series.Series
```

In [24]:
```
#Look at the first few elements of the output
df1.head()
```

Out[24]:
```
0    186960
1     93000
2    110515
3    131205
4    104800
Name: salary, dtype: int64
```

In [25]:
```
#Select column salary and make the output to be a data frame
df2 = df[['salary']]
```

In [26]:
```
#Check the type
type(df2)
```

Out[26]:
```
pandas.core.frame.DataFrame
```

In [27]:
```
#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_first10_rows=df[0:11]
df_first10_rows.head(5)
```

Out[27]:

	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

In [28]:
```
#If we want to select both rows and columns we can use method .loc
df_sub=df.loc[10:20,['rank', 'sex','salary']]
```

In [29]:
```
df_sub.head(3)
```

Out[29]:

	rank	sex	salary
10	Prof	Male	128250
11	Prof	Male	134778
12	AsstProf	Male	88000

In [30]:
```
#  Unlike method .loc, method iloc selects rows (and columns) by poistion:
#df_sub.iloc[10:20, [0,3,4,5]]
df.iloc[0:10,[0,4]]
```

Out[30]:

	rank	sex
0	Prof	Male
1	Prof	Male
2	Prof	Male
3	Prof	Male
4	Prof	Male
5	Prof	Male
6	AssocProf	Male
7	Prof	Male
8	Prof	Male
9	Prof	Male

Part 4: Sorting the Data

In [31]:
```
#Sort the data frame by yrs.service and create a new data frame
df_sorted = df.sort_values(by = 'service')
df_sorted.head()
```

Out[31]:

	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 [32]:
```
#Sort the data frame by yrs.service and overwrite the original dataset
df.sort_values(by = 'service', ascending = False, inplace = True)
df.head()
```

Out[32]:

	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 [33]:
```
# Restore the original order (by sorting using index)
df.sort_index(axis=0, ascending = True, inplace = True)
df.head()
```

Out[33]:

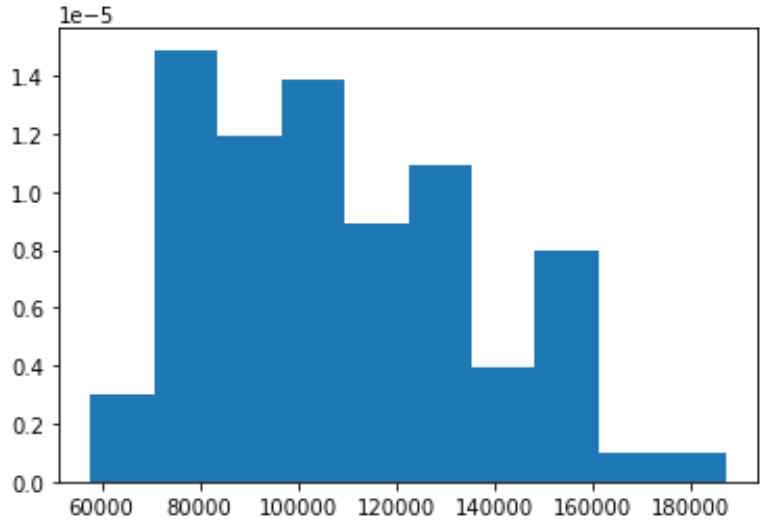
	rank	discipline	phd	service	sex	salary
0	Prof	B	56	49	Male	186960
1	Prof	A	12	6	Male	93000
2	Prof	A	23	20	Male	110515
3	Prof	A	40	31	Male	131205
4	Prof	B	20	18	Male	104800

Part 4: Explore data using graphics

In [34]:
```
#Show graphs withint Python notebook
%matplotlib inline
```

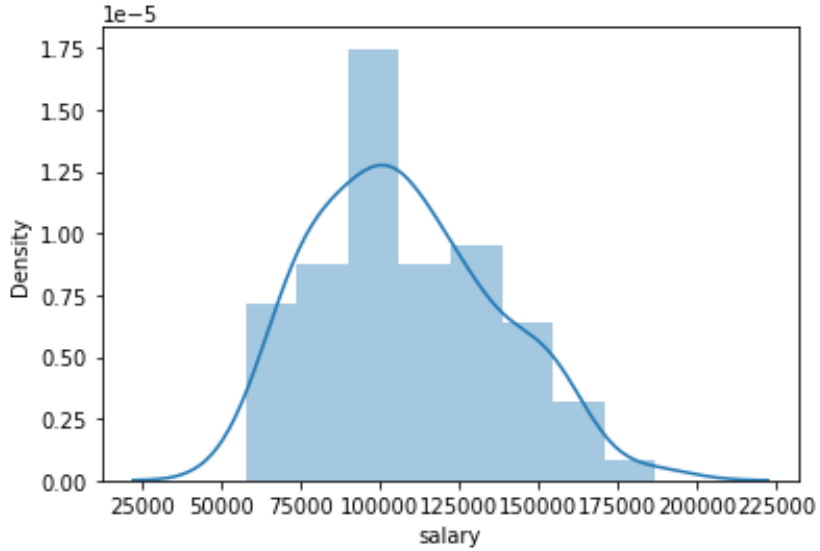
In [35]:
```
#Use matplotlib to draw a histogram of a salary data
plt.hist(df['salary'],bins=10, density= True)
```

Out[35]:
```
(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.]),
 <BarContainer object of 10 artists>)
```



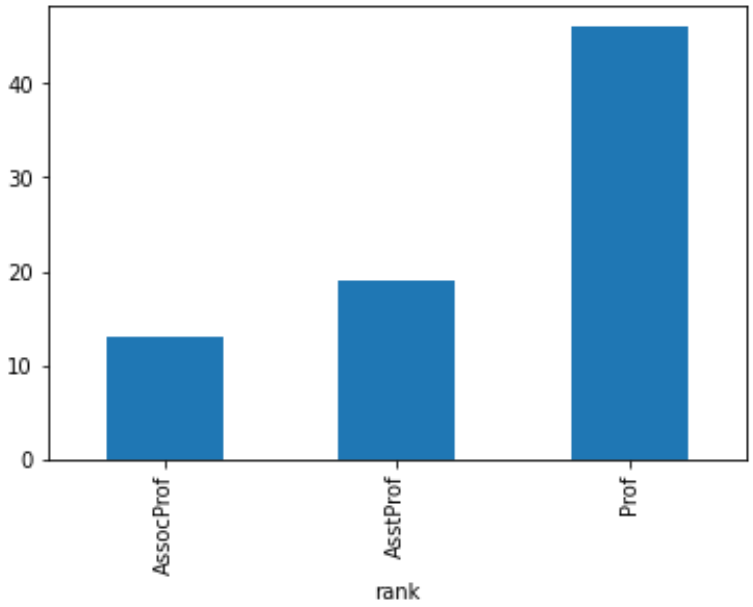
In [36]:
```
#Use seaborn package to draw a histogram
sns.distplot(df['salary']);
```

```
C:\Anaconda\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will b
e removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexib
ility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



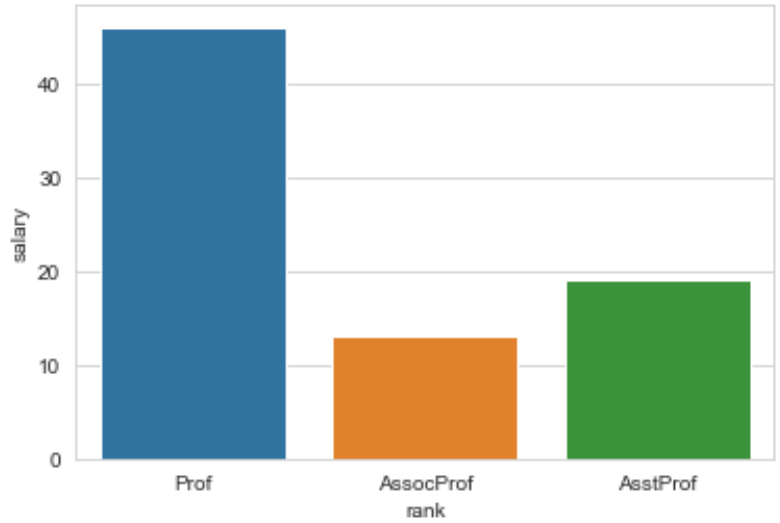
In [37]:
```
# Use regular matplotlib function to display a barplot
df.groupby(['rank'])['salary'].count().plot(kind='bar')
```

Out[37]:
```
<AxesSubplot:xlabel='rank'>
```

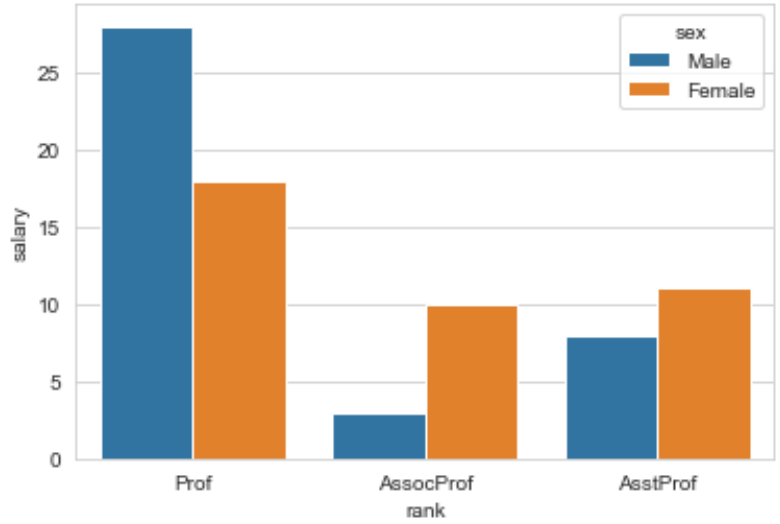


In [38]:
```
# Use seaborn package to display a barplot
sns.set_style("whitegrid")

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

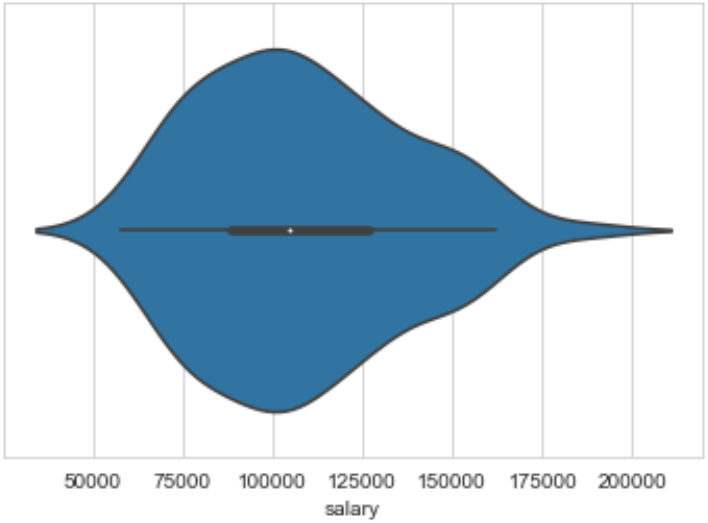


In [39]:
```
# Split into 2 groups:
ax = sns.barplot(x='rank',y ='salary', hue='sex', data=df, estimator=len)
```



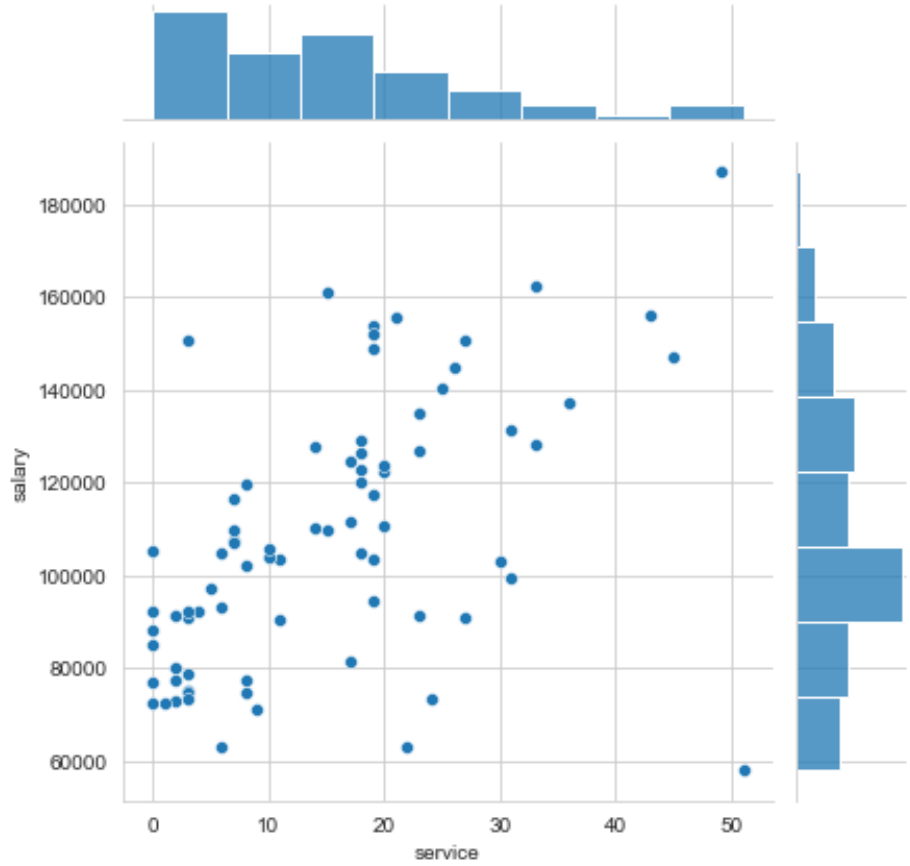
In [40]:
```
#Violinplot
sns.violinplot(x = "salary", data=df)
```

Out[40]:
```
<AxesSubplot:xlabel='salary'>
```



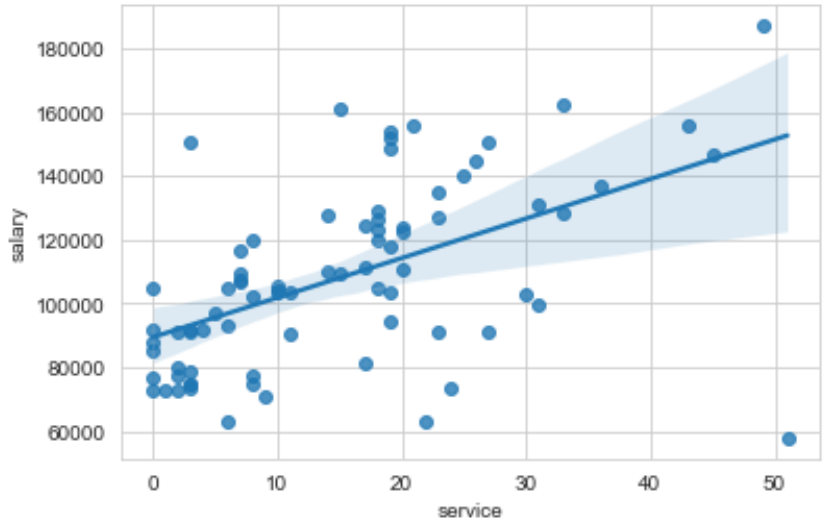
In [41]:
```
#Scatterplot in seaborn
sns.jointplot(x='service', y='salary', data=df)
```

Out[41]:
```
<seaborn.axisgrid.JointGrid at 0x1f8457f61f0>
```



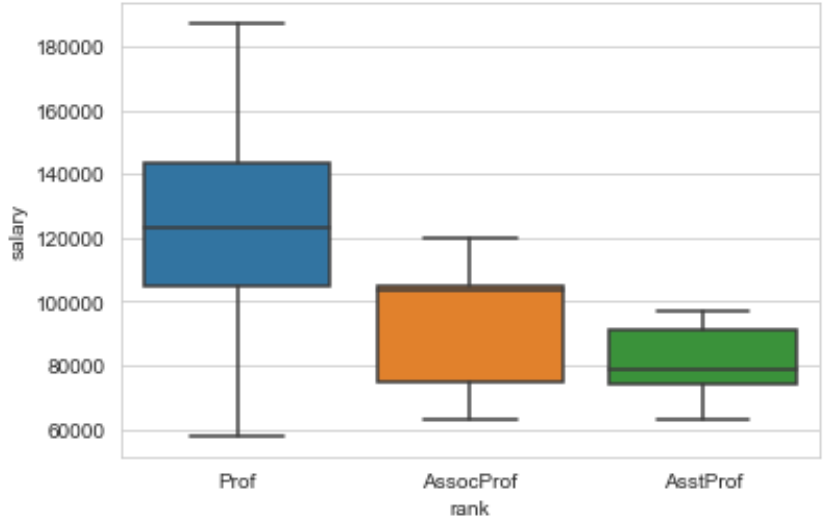
In [42]:
```
#If we are interested in linear regression plot for 2 numeric variables we can use regplot
sns.regplot(x='service', y='salary', data=df)
```

Out[42]:
```
<AxesSubplot:xlabel='service', ylabel='salary'>
```



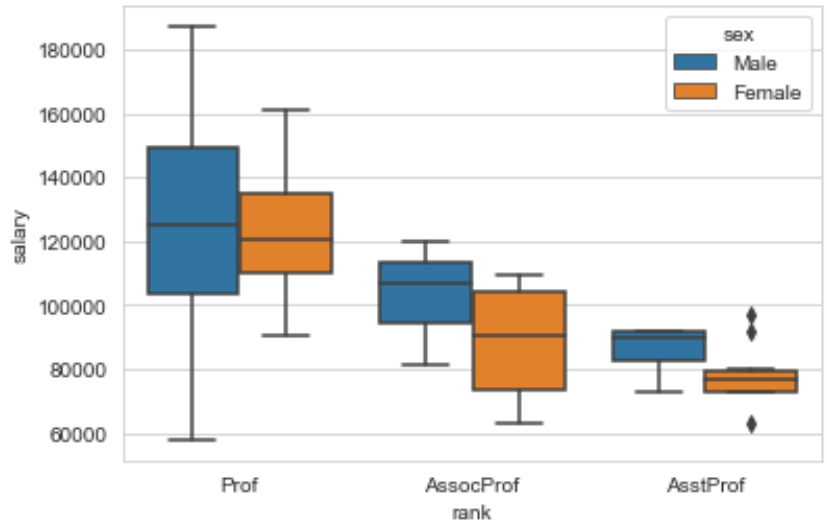
In [43]:
```
# box plot
sns.boxplot(x='rank',y='salary', data=df)
```

Out[43]:
```
<AxesSubplot:xlabel='rank', ylabel='salary'>
```



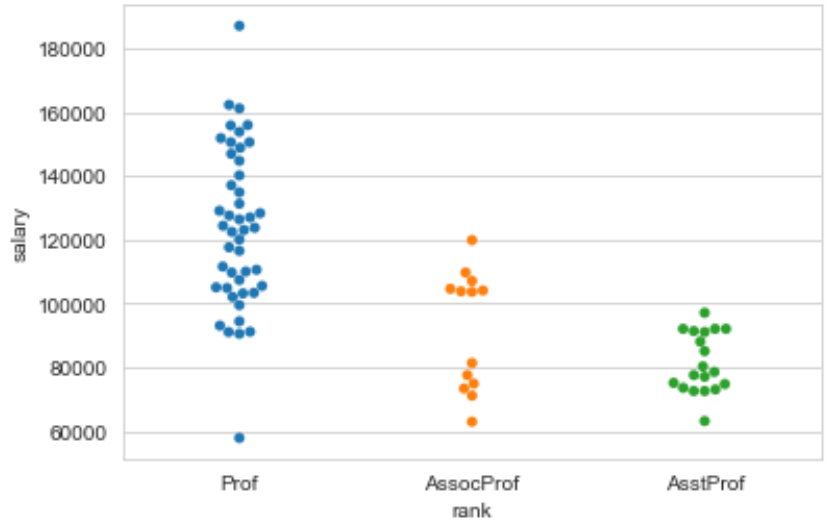
In [44]:
```
# side-by-side box plot
sns.boxplot(x='rank',y='salary', data=df, hue='sex')
```

Out[44]: <AxesSubplot:xlabel='rank', ylabel='salary'>



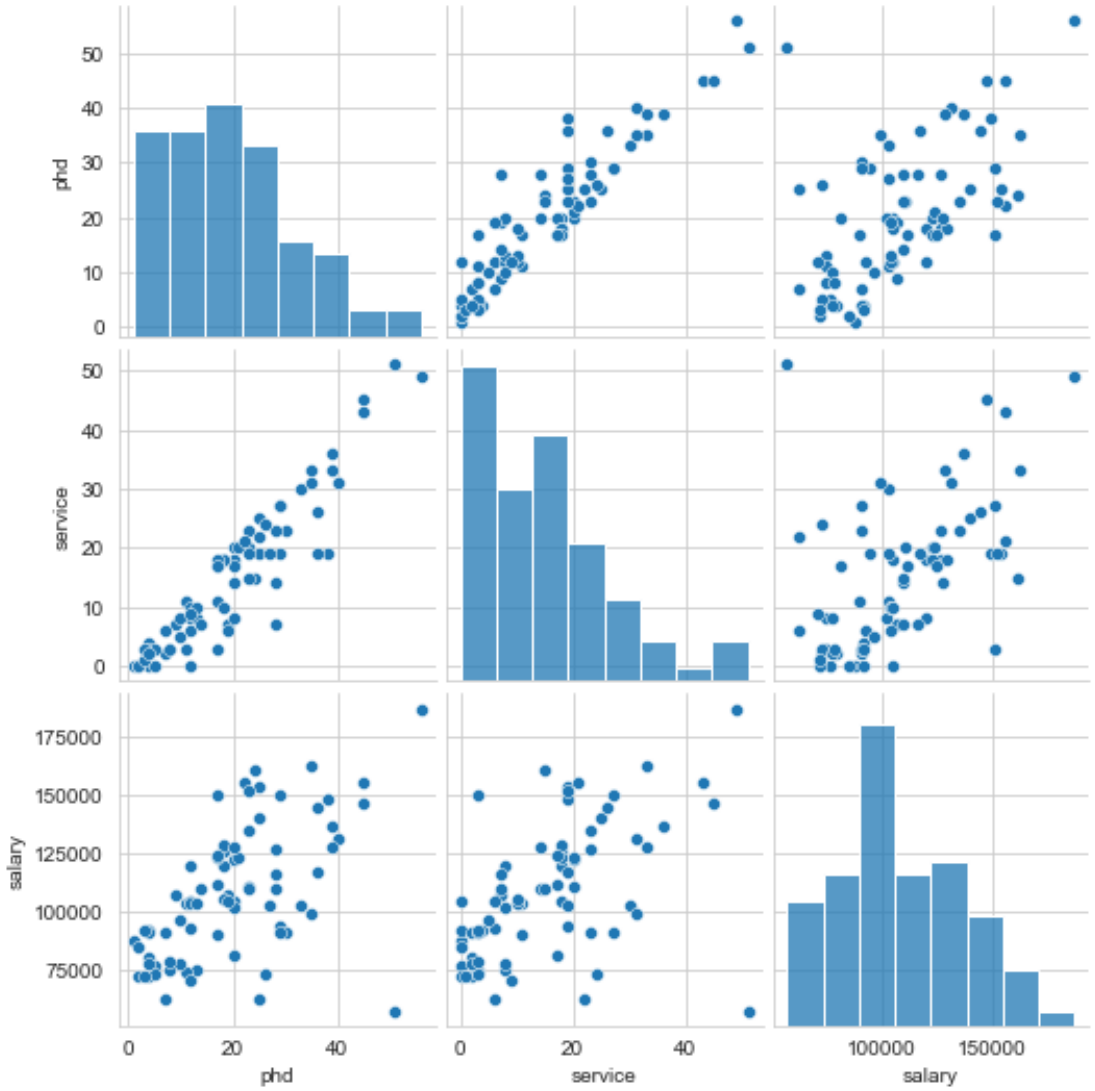
In [45]:
```
# swarm plot
sns.swarmplot(x='rank',y='salary', data=df)
```

Out[45]: <AxesSubplot:xlabel='rank', ylabel='salary'>



In [46]:
```
# Pairplot
sns.pairplot(df)
```

Out[46]:
```
<seaborn.axisgrid.PairGrid at 0x1f8469f4790>
```



Question #1:

Calculate average of the columns in the first 50 rows

In [75]:
```
#df_head= df.head(50)
#df_head.mean(axis=0)#can be used if specific numeric columns are not known.
df_head= df.head(50)
df_head[["phd", "service","salary"]].mean(axis=0)
```

Out[75]:
```
phd            21.52
service        17.60
salary     113789.14
dtype: float64
```

Question #2:

Calculate the basic statistics for the salary column

In [76]:
```
df["salary"].describe()
```

Out[76]:
```
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
```

Question #3:

Find how many values in the salary column (use count method)

In [80]:
```
df["salary"].count()
```

Out[80]:
```
78
```

Question #4:

Calculate the average salary

In [87]:
```
df["salary"].mean()
```

Out[87]:
```
108023.78205128205
```

In [86]:
```
df.groupby(["sex","rank","discipline"])["salary"].mean()
```

Out[86]:
```
sex     rank       discipline
Female  AssocProf  A              72128.500000
                   B              99435.666667
        AsstProf   A              72933.333333
                   B              84189.800000
        Prof       A             109631.875000
                   B             131836.200000
Male    AssocProf  A              81285.000000
                   B             113404.000000
        AsstProf   A              79000.000000
                   B              88224.000000
        Prof       A             113164.400000
                   B             137989.076923
Name: salary, dtype: float64
```

Question #5:

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

In [93]:
```
df.groupby("discipline")["salary"].mean()
```

Out[93]:
```
discipline
A     98331.111111
B    116331.785714
Name: salary, dtype: float64
```

Question #6:

Extract (filter) only observations with high salary (> 100K) and find how many female and male professors in each group

In [104…
```
df_sub = df[ df['salary'] > 100000]
df_sub.groupby(["sex","rank","discipline"]).count()
```

Out[104]:

sex	rank	discipline	phd	service	salary
Female	**AssocProf**	**B**	5	5	5
	Prof	**A**	6	6	6
		B	10	10	10
Male	**AssocProf**	**B**	2	2	2
	Prof	**A**	11	11	11
		B	12	12	12

Question #7:

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

In [109…

```
df.sort_values(by = 'salary', ascending = False)
df.head()
```

Out[109]:

	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

Question #8:

Sort the data frame using 2 or more columns

In [119…]
```
df.sort_values(by =["salary","phd","service"], ascending = True)
```

Out[119]:

	rank	discipline	phd	service	sex	salary
9	Prof	A	51	51	Male	57800
54	AssocProf	A	25	22	Female	62884
66	AsstProf	A	7	6	Female	63100
71	AssocProf	B	12	9	Female	71065
55	AsstProf	A	2	0	Female	72500
...	...	...	...	...	...	...
31	Prof	B	22	21	Male	155750
27	Prof	A	45	43	Male	155865
72	Prof	B	24	15	Female	161101
13	Prof	B	35	33	Male	162200
0	Prof	B	56	49	Male	186960

78 rows × 6 columns

Toluwalope Eunice David

1. Import the file addresses.txt

In [123…

```
# Importing Packages
import pandas as pd

# Importing file addresses.txt
addresses = pd.read_csv("addresses.txt", sep = r",\b", engine = "python")
addresses.head()
```

Out[123]:

	ID	LastName	Address	City	State
0	1	O'Higgins	48 Grant Rd.	Des Moines	IA
1	11	Macina	401 1st Ave., Apt 13G	New York	NY
2	242	Roeder	71 Quebec Ave.	E. Thetford	VT
3	146	Stephens	1234 Smythe St., #5	Detroit	MI
4	241	Ishikawa	986 OceanView Dr.	Pacific Grove	CA

2. Read in the SAS file “medical.sas7bdat”. Export the data frame to a CSV file. Call your file “medicalexport.csv”.

In [124…

```
# Importing the sas file with 'utf8' encoding
sas_file = pd.read_sas("medical.sas7bdat", encoding = "utf8")
print("This is the first 5 rows of imported sas file")
print(sas_file.head())

# Exporting dataframe as a csv file
sas_file.to_csv("medicalexport.csv", index = False, header = True)
print("\nThe sas file is exported as a csv file.")

# Importing the exported csv file to show that csv file was created
print("\n Following is the first 5 rows of converted csv file.")
```

```
sas_to_csv = pd.read_csv("medicalexport.csv", index_col = False)
print(sas_to_csv.head())
```

```
This is the first 5 rows of imported sas file
    ID  YEAR  MEDEXP   INC   AGE  INSUR
0  1.0   1.0     9.0  49.0  51.0    1.0
1  1.0   2.0     9.0  51.0  52.0    1.0
2  1.0   3.0     9.0  55.0  53.0    1.0
3  1.0   4.0    10.0  58.0  54.0    1.0
4  1.0   5.0    11.0  61.0  55.0    1.0

The sas file is exported as a csv file.

 Following is the first 5 rows of converted csv file.
    ID  YEAR  MEDEXP   INC   AGE  INSUR
0  1.0   1.0     9.0  49.0  51.0    1.0
1  1.0   2.0     9.0  51.0  52.0    1.0
2  1.0   3.0     9.0  55.0  53.0    1.0
3  1.0   4.0    10.0  58.0  54.0    1.0
4  1.0   5.0    11.0  61.0  55.0    1.0
```

3. The msleep data set contains the sleep times and weights for various mammals.

a. Import the csv data from the following website: https://raw.githubusercontent.com/genomicsclass/dagdata/master/inst/extdata/msleep_ggplot2

In [125…

```
# Importing the csv data from the website
sleep = pd.read_csv("https://raw.githubusercontent.com/genomicsclass/dagdata/master/inst/extdata/msleep_ggplot2.csv")
sleep.head()
```

Out[125]:

	name	genus	vore	order	conservation	sleep_total	sleep_rem	sleep_cycle	awake	brainwt	bodywt
0	Cheetah	Acinonyx	carni	Carnivora	lc	12.1	NaN	NaN	11.9	NaN	50.000
1	Owl monkey	Aotus	omni	Primates	NaN	17.0	1.8	NaN	7.0	0.01550	0.480
2	Mountain beaver	Aplodontia	herbi	Rodentia	nt	14.4	2.4	NaN	9.6	NaN	1.350
3	Greater short-tailed shrew	Blarina	omni	Soricomorpha	lc	14.9	2.3	0.133333	9.1	0.00029	0.019
4	Cow	Bos	herbi	Artiodactyla	domesticated	4.0	0.7	0.666667	20.0	0.42300	600.000

b. Create a new data frame called subset to manipulate the data in the following ways:
i. Group according to order and genus
ii. Add a new variable for the proportion of REM sleep as a fraction of the total hours of sleep
iii. Summarize the average sleep_total, average rem_prop, minimum sleep_total, and maximum sleep_total
iv. Filter the data so that the average sleep hours are greater than 5 v. Arrange the data from lowest to highest by average sleep hours

In [126…

```
# Creating new dataframe from sleep dataframe
subset = sleep.copy()
subset.head()
```

Out[126]:

	name	genus	vore	order	conservation	sleep_total	sleep_rem	sleep_cycle	awake	brainwt	bodywt
0	Cheetah	Acinonyx	carni	Carnivora	lc	12.1	NaN	NaN	11.9	NaN	50.000
1	Owl monkey	Aotus	omni	Primates	NaN	17.0	1.8	NaN	7.0	0.01550	0.480
2	Mountain beaver	Aplodontia	herbi	Rodentia	nt	14.4	2.4	NaN	9.6	NaN	1.350
3	Greater short-tailed shrew	Blarina	omni	Soricomorpha	lc	14.9	2.3	0.133333	9.1	0.00029	0.019
4	Cow	Bos	herbi	Artiodactyla	domesticated	4.0	0.7	0.666667	20.0	0.42300	600.000

In [127…

```
# i. Grouping according to order and genus
grouped_subset = subset.groupby(["order", "genus"])
grouped_subset.first() # Show the first entries in all groups
```

Out[127]:

order	genus	name	vore	conservation	sleep_total	sleep_rem	sleep_cycle	awake	brainwt	bodywt
Afrosoricida	**Tenrec**	Tenrec	omni	None	15.6	2.3	NaN	8.4	0.00260	0.900
Artiodactyla	**Bos**	Cow	herbi	domesticated	4.0	0.7	0.666667	20.0	0.42300	600.000
	Capreolus	Roe deer	herbi	lc	3.0	NaN	NaN	21.0	0.09820	14.800
	Capri	Goat	herbi	lc	5.3	0.6	NaN	18.7	0.11500	33.500
	Giraffa	Giraffe	herbi	cd	1.9	0.4	NaN	22.1	NaN	899.995
...	**...**	...	...	...	...	...	...	...	...	...
Soricomorpha	**Blarina**	Greater short-tailed shrew	omni	lc	14.9	2.3	0.133333	9.1	0.00029	0.019
	Condylura	Star-nosed mole	omni	lc	10.3	2.2	NaN	13.7	0.00100	0.060
	Cryptotis	Lesser short-tailed shrew	omni	lc	9.1	1.4	0.150000	14.9	0.00014	0.005
	Scalopus	Eastern american mole	insecti	lc	8.4	2.1	0.166667	15.6	0.00120	0.075
	Suncus	Musk shrew	None	None	12.8	2.0	0.183333	11.2	0.00033	0.048

77 rows × 9 columns

In [128…

```
# ii. Add a new variable for the proportion of REM sleep as a fraction of the total hours of sleep
# Replacing NA data with 0
subset["sleep_rem"] = subset["sleep_rem"].fillna(0)
# Adding new column to the dataframe
subset["sleep_rem_prop"] = subset["sleep_rem"] / subset["sleep_total"]
subset.head()
```

Out[128]:

	name	genus	vore	order	conservation	sleep_total	sleep_rem	sleep_cycle	awake	brainwt	bodywt	sleep_rem_prop
0	Cheetah	Acinonyx	carni	Carnivora	lc	12.1	0.0	NaN	11.9	NaN	50.000	0.000000
1	Owl monkey	Aotus	omni	Primates	NaN	17.0	1.8	NaN	7.0	0.01550	0.480	0.105882
2	Mountain beaver	Aplodontia	herbi	Rodentia	nt	14.4	2.4	NaN	9.6	NaN	1.350	0.166667
3	Greater short-tailed shrew	Blarina	omni	Soricomorpha	lc	14.9	2.3	0.133333	9.1	0.00029	0.019	0.154362
4	Cow	Bos	herbi	Artiodactyla	domesticated	4.0	0.7	0.666667	20.0	0.42300	600.000	0.175000

In [129…

```
# iii. Summarize the average sleep_total, average rem_prop, minimum sleep_total, and maximum sleep_total
print(f"Average sleep_total \t=\t{subset['sleep_total'].mean()}")
print(f"Average rem_prop \t=\t{subset['sleep_rem_prop'].mean()}")
print(f"Minimum sleep_total \t=\t{subset['sleep_total'].min()}")
print(f"Maximum sleep_total \t=\t{subset['sleep_total'].max()}")
```

```
Average sleep_total     =       10.433734939759034
Average rem_prop        =       0.1278961493989131
Minimum sleep_total     =       1.9
Maximum sleep_total     =       19.9
```

In [130…

```
# iv. Filter the data so that the average sleep hours are greater than 5 v.
# Arrange the data from lowest to highest by average sleep hours
# First filtering the data then sorting them in ascending order
subset[subset["sleep_total"] > 5].sort_values("sleep_total", ascending = True)
```

Out[130]:

	name	genus	vore	order	conservation	sleep_total	sleep_rem	sleep_cycle	awake	brainwt	bodywt	sleep_rem_prop
79	Bottle-nosed dolphin	Tursiops	carni	Cetacea	NaN	5.2	0.0	NaN	18.80	NaN	173.330	0.000000
18	Tree hyrax	Dendrohyrax	herbi	Hyracoidea	lc	5.3	0.5	NaN	18.70	0.01230	2.950	0.094340
10	Goat	Capri	herbi	Artiodactyla	lc	5.3	0.6	NaN	18.70	0.11500	33.500	0.113208
62	Rock hyrax	Procavia	NaN	Hyracoidea	lc	5.4	0.5	NaN	18.60	0.02100	3.600	0.092593
59	Common porpoise	Phocoena	carni	Cetacea	vu	5.6	0.0	NaN	18.45	NaN	53.180	0.000000
...	...	...	...	...	...	...	...	...	...	...	...	...
19	North American Opossum	Didelphis	omni	Didelphimorphia	lc	18.0	4.9	0.333333	6.00	0.00630	1.700	0.272222
61	Giant armadillo	Priodontes	insecti	Cingulata	en	18.1	6.1	NaN	5.90	0.08100	60.000	0.337017
36	Thick-tailed opposum	Lutreolina	carni	Didelphimorphia	lc	19.4	6.6	NaN	4.60	NaN	0.370	0.340206
21	Big brown bat	Eptesicus	insecti	Chiroptera	lc	19.7	3.9	0.116667	4.30	0.00030	0.023	0.197970
42	Little brown bat	Myotis	insecti	Chiroptera	NaN	19.9	2.0	0.200000	4.10	0.00025	0.010	0.100503

72 rows × 12 columns

4. Export the msleep data set to a workbook called “mammal_sleep.xlsx” on a sheet named “Original”.

In [133…
```
# Exporting the dataset to an excel file with sheet named Original
sleep.to_excel("mammal_sleep.xlsx", sheet_name = "Original", index = False)
print("Data Succesfully Exported to Excel file")
```

```
Data Succesfully Exported to Excel file
```

5. Export the subset data frame, you have created in Q3.b, to the SAME excel workbook (“mammal_sleep.xlsx”) on a sheet called “Subset”.

In [134…
```
# Importing openpyxl package
import openpyxl
# Andding new sheet to excel file
with pd.ExcelWriter("mammal_sleep.xlsx", mode="a", engine="openpyxl") as writer:
    subset.to_excel(writer, sheet_name="Subset", index=False)
print("Data Succesfully Exported to Excel file")
```

```
Data Succesfully Exported to Excel file
```

In [135…
```
# Importing the exported excel file with sheet name Original
excel_file = pd.read_excel("mammal_sleep.xlsx", sheet_name = "Original")
excel_file.head()
```

Out[135]:

	name	genus	vore	order	conservation	sleep_total	sleep_rem	sleep_cycle	awake	brainwt	bodywt
0	Cheetah	Acinonyx	carni	Carnivora	lc	12.1	NaN	NaN	11.9	NaN	50.000
1	Owl monkey	Aotus	omni	Primates	NaN	17.0	1.8	NaN	7.0	0.01550	0.480
2	Mountain beaver	Aplodontia	herbi	Rodentia	nt	14.4	2.4	NaN	9.6	NaN	1.350
3	Greater short-tailed shrew	Blarina	omni	Soricomorpha	lc	14.9	2.3	0.133333	9.1	0.00029	0.019
4	Cow	Bos	herbi	Artiodactyla	domesticated	4.0	0.7	0.666667	20.0	0.42300	600.000

In [136…
```
# Reading the Subset sheet from the excel file
excel_file_2 = pd.read_excel("mammal_sleep.xlsx", sheet_name = "Subset")
```

```
excel_file_2.head()
```

Out[136]:

	name	genus	vore	order	conservation	sleep_total	sleep_rem	sleep_cycle	awake	brainwt	bodywt	sleep_rem_prop
0	Cheetah	Acinonyx	carni	Carnivora	lc	12.1	0.0	NaN	11.9	NaN	50.000	0.000000
1	Owl monkey	Aotus	omni	Primates	NaN	17.0	1.8	NaN	7.0	0.01550	0.480	0.105882
2	Mountain beaver	Aplodontia	herbi	Rodentia	nt	14.4	2.4	NaN	9.6	NaN	1.350	0.166667
3	Greater short-tailed shrew	Blarina	omni	Soricomorpha	lc	14.9	2.3	0.133333	9.1	0.00029	0.019	0.154362
4	Cow	Bos	herbi	Artiodactyla	domesticated	4.0	0.7	0.666667	20.0	0.42300	600.000	0.175000

In [ ]:

Toluwalope Eunice David

In [5]:
```
#Import Python Libraries
import pandas as pd
from scipy.stats import pearsonr
import matplotlib.pyplot as plt
```

1. Read the "FuelEfficiency.csv" dataset

In [6]:
```
##Read csv file
FuelEfficiency = pd.read_csv("FuelEfficiency.csv")

#showing first few rows for the object
FuelEfficiency.head()
```

Out[6]:

	Model	Eng Size	Cylinders	MSRP	City_L/100km	Highway_L/100km	Weight_in_Pounds	Type	Country
0	A4	1.8	4	25550	12.8	9.1	3252	Sedan	Germany
1	3_Series	2.5	6	28100	14.1	9.7	3219	Sedan	Germany
2	G35	3.5	6	28150	15.7	10.9	3336	Sedan	Japan
3	X-Type	2.5	6	29330	14.9	10.1	3428	Sedan	England
4	C-class	1.8	4	29250	12.8	9.4	3250	Sedan	Germany

2. Which pair of the variables in the dataset are most strongly correlated?

In [7]:
```
#Using the pearson standard correlation coefficient and correlation plot
FuelEfficiency.corr(method='pearson', min_periods=1)
corr = FuelEfficiency.corr()
corr.style.background_gradient(cmap='coolwarm')
```

Out[7]:

	Eng Size	Cylinders	MSRP	City_L/100km	Highway_L/100km	Weight_in_Pounds
Eng Size	1.000000	0.910581	0.691404	0.848351	0.744533	0.758818
Cylinders	0.910581	1.000000	0.752577	0.795642	0.671216	0.720889
MSRP	0.691404	0.752577	1.000000	0.654737	0.554326	0.588448
City_L/100km	0.848351	0.795642	0.654737	1.000000	0.866586	0.858554
Highway_L/100km	0.744533	0.671216	0.554326	0.866586	1.000000	0.906166
Weight_in_Pounds	0.758818	0.720889	0.588448	0.858554	0.906166	1.000000

Justification & Answer

From the plot, it shows that none of the variables have a negative correlation. Also none of the variables have a weak correlation. '0.0 – 0.4' (Weak linear correlation)

Most have a strong positive correlation. However, Variables pairs with high linear correlation '0.8 – 1.0' include; 1) Cylinders and Engine size

2) Engine size and City_L/100km

3) Highway_L/100km and City_L/100km

4) Weight_in_Pounds and City_L/100km

5) Weight_in_Pounds and Highway_L/100km

3. What are the minimum and maximum weights (in pounds) in the dataset?

In [8]:
```
Weight_in_Pounds_min_max = FuelEfficiency["Weight_in_Pounds"].agg(['min','max'])
Weight_in_Pounds_min_max
```

Out[8]:
```
min    2469
max    6400
Name: Weight_in_Pounds, dtype: int64
```

4. Add a column to the dataset to filter the weight in pounds variable.

In [9]:
```
FuelEfficiency['Weight_Value'] = FuelEfficiency["Weight_in_Pounds"].apply(lambda x: 1 if x < 4000 else 0)
FuelEfficiency.sort_values(by='Weight_Value', ascending=False)
```

Out[9]:

	Model	Eng Size	Cylinders	MSRP	City_L/100km	Highway_L/100km	Weight_in_Pounds	Type	Country	Weight_Value
0	A4	1.8	4	25550	12.8	9.1	3252	Sedan	Germany	1
29	Grand_Marquis_GS	4.6	8	25520	16.6	11.3	3957	Sedan	US	1
31	LS_430	4.3	8	55375	15.7	11.3	3990	Sedan	Japan	1
34	XJ-Series	4.2	8	69330	15.7	10.1	3803	Sedan	England	1
36	TL_3.2	3.2	6	32650	14.1	10.1	3575	Sedan	Japan	1
...	...	...	...	...	...	...	...	...	...	...
53	Odyssey_EX	3.5	6	26990	15.7	11.3	4365	Minivan	Japan	0
52	Astro_LS	4.3	6	25230	17.7	14.1	4309	Minivan	US	0
51	Freestar_SE	3.9	6	26390	16.6	12.3	4275	Minivan	US	0
48	Sedona_EX	3.5	6	22085	18.8	14.1	4802	Minivan	Korea	0
28	Crown_Victoria_LX	4.6	8	27220	16.6	11.3	4057	Sedan	US	0

84 rows × 10 columns

5. Determine the correlation between weight in pounds and litres of fuel consumed per 100 km on the highway only for cars with a weight less than 4000 pounds. How does this value compare to the value obtained when considering all cars

In [16]:
```
#correlation between weight in pounds and litres of fuel consumed per 100 km on the highway
#only for cars with a weight less than 4000 pounds
data_less_than_4000 = FuelEfficiency[FuelEfficiency["Weight_Value"] == 1]
data_1 = data_less_than_4000['Weight_in_Pounds']
data_2 = data_less_than_4000['Highway_L/100km']
```

```
correlation = data_1.corr(data_2)
print(f"Correlation between Weight in Pounds and Highway_L/100km for car less than 4000 pounds is {correlation.round(2)}.")
```

```
Correlation between Weight in Pounds and Highway_L/100km for car less than 4000 pounds is 0.72.
```

In [15]:
```
#correlation between weight in pounds and litres of fuel consumed per 100 km on the highway
#considering all cars
data_1 = FuelEfficiency['Weight_in_Pounds']
data_2 = FuelEfficiency['Highway_L/100km']
correlation = data_1.corr(data_2)
print(f"Correlation between Weight in Pounds and Highway_L/100km for all car is {correlation.round(2)}.")
```

```
Correlation between Weight in Pounds and Highway_L/100km for all car is 0.91.
```

For cars with a weight less than 4000 pounds, there is a strong positive correlation between weight in pounds and litres of fuel consumed per 100 km on the highway. However,when considering all cars, there seems to be a very strong positive correlation between the two variables.

6. Suppose I created a subset of the data to restrict the weight to between 3000 and 4000 pounds. Would the correlation between highway fuel consumption and weight subject to this restriction be greater than or less than the value you obtained in Question 7? Explain

In [18]:
```
# Creating a subset dataframe with weight of vehicle between 3000 & 4000
data_between_3000_and_4000 = FuelEfficiency[(FuelEfficiency["Weight_in_Pounds"] > 3000) & (FuelEfficiency["Weight_in_Pounds
# Calculating the correlation for Highway_L/100km and Weight_in_Pounds between 3000 & 4000
corr = data_between_3000_and_4000['Highway_L/100km'].corr(data_between_3000_and_4000['Weight_in_Pounds'])
print(f"Correlation between Weight in Pounds and Highway_L/100km for car weighing between 3000 & 4000 pounds using a subset
# The Correlation for both the condition will be same.
# Because the data for which the correlation is being calculated will remain same weather you calculate the correlation fro
```

```
Correlation between Weight in Pounds and Highway_L/100km for car weighing between 3000 & 4000 pounds using a subset is 0.5
3
```

7. Calculate the correlation coefficient between highway fuel consumption and weight for vehicles with a weight between 3000 and 4000 pounds.

In [20]:
```
# Calculation the correlation coefficient between highway fuel consumption and weight for vehicles with a weight between 30
corr = FuelEfficiency['Highway_L/100km'].corr(FuelEfficiency["Weight_in_Pounds"][(FuelEfficiency["Weight_in_Pounds"] > 3000
print(f"Correlation between Weight in Pounds and Highway_L/100km for car weighing between 3000 & 4000 pounds is {corr.round
```

Correlation between Weight in Pounds and Highway_L/100km for car weighing between 3000 & 4000 pounds is 0.53

In [ ]:

Toluwalope Eunice David

In [68]:
```
#Import Python Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

1. Read the “airquality.csv data set” dataset

In [79]:
```
##Read csv file
airquality = pd.read_csv("airquality.csv")

#showing first few rows for the object
airquality.head(10)
#Shows we have some missing values (NaN)
```

Out[79]:

	Ozone	Solar.R	Wind	Temp	Month	Day
0	41.0	190.0	7.4	67.0	5	1
1	36.0	118.0	8.0	72.0	5	2
2	12.0	149.0	12.6	74.0	5	3
3	18.0	313.0	11.5	62.0	5	4
4	NaN	NaN	14.3	56.0	5	5
5	28.0	NaN	14.9	66.0	5	6
6	23.0	299.0	8.6	65.0	5	7
7	19.0	99.0	13.8	59.0	5	8
8	8.0	19.0	NaN	NaN	5	9
9	NaN	194.0	8.6	69.0	5	10

1a. How many missing values are present for each variable?

In [80]:
```
print('These are the number of missing values for each variable:')

display(airquality.isnull().sum())


print("The total number of missing values in the air quality DataFrame is " + str(airquality.isnull().sum().sum()))
                                                                        #(turn the integer to a str value for the final stateme

#Showing the missing values present for each of the six variables in the "airquality" object
```

```
These are the number of missing values for each variable:
Ozone      37
Solar.R     7
Wind       16
Temp        5
Month       0
Day         0
dtype: int64
The total number of missing values in the air quality DataFrame is 65
```

1b. Create a data frame of complete cases and find the mean temperature using listwise deletion

In [89]:
```
airquality_listwise = airquality.copy() #make a copy of the dataframe to work on
```

In [90]:
```
airquality_listwise.dropna(inplace=True)
```

In [91]:
```
#Drop rows that containing missing values
print("The average temperature in the airquality object using the listwise deletion is "+
      str(round(airquality_listwise["Temp"].mean(),2)))
    #used round method to round off the values

#Because there is a missing completely at Random case, we will use the listwise deletion method for the NaN missing values.
#In Listwise deletion entire rows(which hold the missing values) are deleted.
```

```
The average temperature in the airquality object using the listwise deletion is 77.99
```

1c. Find the mean temperature using pairwise deletion.

In [6]:
```
airquality_pairwise = airquality.copy() #make a copy of the dataframe to work on


print("The average temperature in the airquality object using the pairwise deletion is "+
      str(round(airquality_pairwise.mean(),2)))
      #used round method to round off the values


#Using the pairwise skips the missing values and calculates mean of the remaining values.
```

```
The average temperature in the airquality object using the pairwise deletion is 77.91
```

1d. Which rows contain missing temperature values?

In [92]:
```
missing_data_temp = airquality[airquality['Temp'].isnull()]
#saving null values in Temp column to a new variable :missing_data_temp
```

In [93]:
```
print("The number of rows with missing values for 'Temp' in the airquality object is "+
      str(missing_data_temp['Temp'].isna().sum()))
      #counting the number of missing values in Temp coloumn for the final statement
```

```
The number of rows with missing values for 'Temp' in the airquality object is 5
```

In [94]:
```
missing_data_temp #showing rows with  missing values in the "temp" coloum
```

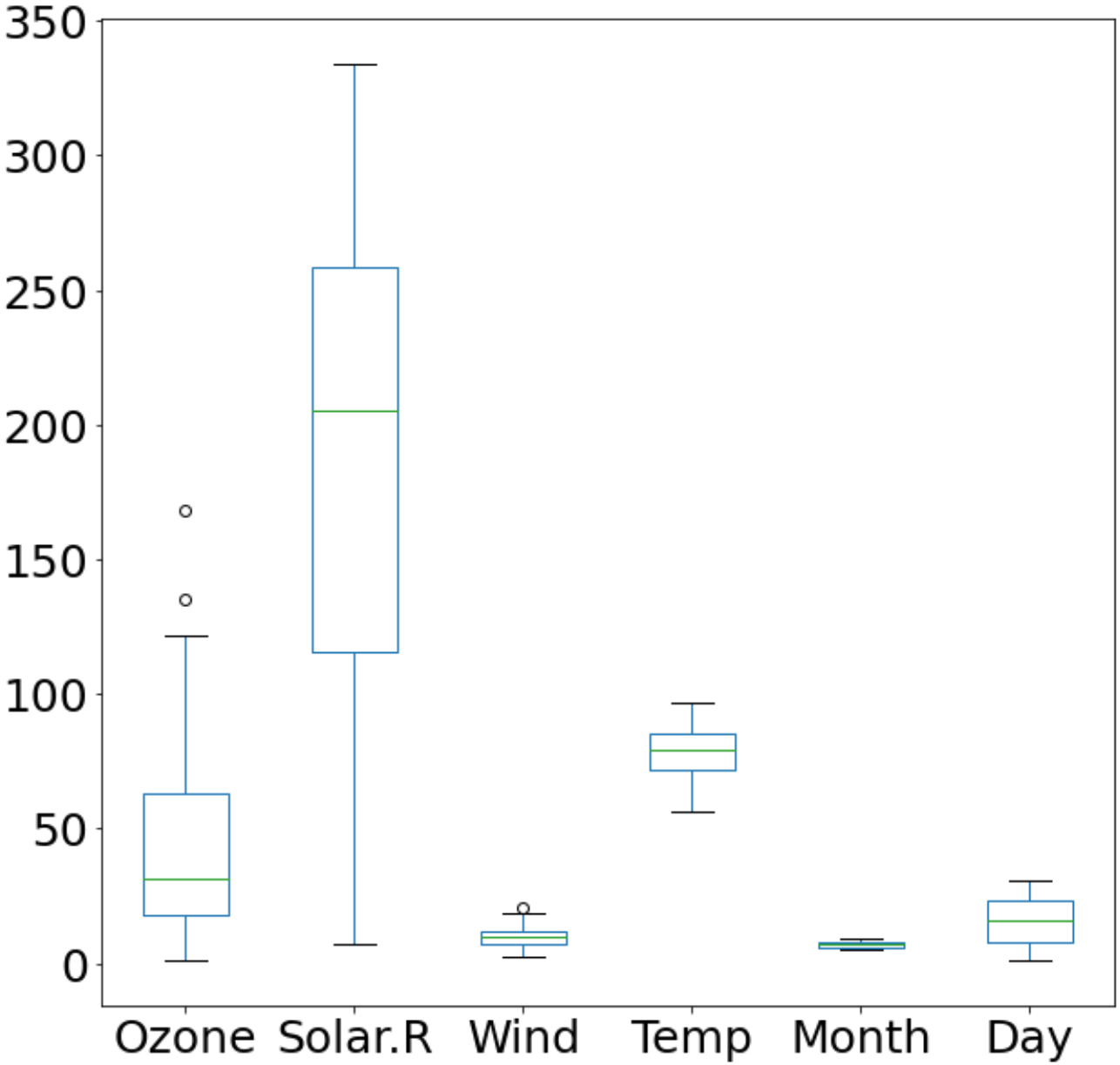
Out[94]:

	Ozone	Solar.R	Wind	Temp	Month	Day
8	8.0	19.0	NaN	NaN	5	9
60	NaN	138.0	8.0	NaN	6	30
81	16.0	7.0	6.9	NaN	7	21
89	50.0	275.0	7.4	NaN	7	29
145	36.0	139.0	NaN	NaN	9	23

1e. Create a box plot for the air quality data. Interpret the plot to explain why there are differences in the mean temperature.

In [97]:
```
boxplot = airquality.boxplot(grid=False,fontsize=25,figsize=(10,10))
boxplot
#Temp distribution does not appear to be symmetric, it seems to have a left-shew kind of symmetry.
```

Out[97]:
```
<AxesSubplot:>
```



1f. How many of the ozone values are outliers (using the default 1.5 IQR setting)?

What are the ozone outlier values?

Create a new data frame called ozone_complete that has all rows with ozone outliers removed.

In [9]:
```
#Outlier Detection and Treatment in Python Using 1.5 IQR rule
#Arranging the data in ascending order.
airquality["Ozone"].sort_values(ascending=True)
```

Out[9]:
```
20     1.0
22     4.0
17     6.0
146    7.0
75     7.0
      ...
102    NaN
106    NaN
114    NaN
118    NaN
149    NaN
Name: Ozone, Length: 153, dtype: float64
```

In [10]:
```
Q1=airquality["Ozone"].quantile(.25)
Q1
#Tne common technique to detect outliers is using IQR (interquartile range).
#In specific, IQR is the middle 50% of data, which is Q3-Q1. Q1 is the first quartile, Q3 is the third quartile
```

Out[10]:
```
18.0
```

In [11]:
```
Q3=airquality["Ozone"].quantile(.75)
Q3
```

Out[11]:
```
63.25
```

In [12]:
```
IQR=Q3-Q1
IQR
```

Out[12]:
```
45.25
```

In [13]:
```
upper_limit=  Q3+1.5*IQR
lower_limit=  Q1-1.5*IQR
#The outlier base value is defined above and below datasets normal range namely Upper and Lower limits
```

In [22]:
```
outliers_ozone = airquality["Ozone"][(airquality["Ozone"] > upper_limit) | (airquality["Ozone"] < lower_limit)]
outliers_ozone

#There are two outliers in the "Ozone" variable.
#135.0 and #168.0 at row 61 and row 116 #confirmed values even with the boxplot
```

Out[22]:
```
61     135.0
116    168.0
Name: Ozone, dtype: float64
```

In [52]:
```
#Creating a new data frame called ozone_complete that has all rows with ozone outliers removed.



                                #specifying the condition for dropping the rows with the outlier values
ozone_complete = airquality.drop(airquality.index[airquality['Ozone'] == 135.0])
ozone_complete = airquality.drop(airquality.index[airquality['Ozone'] == 168.0])
ozone_complete
```

Out[52]:

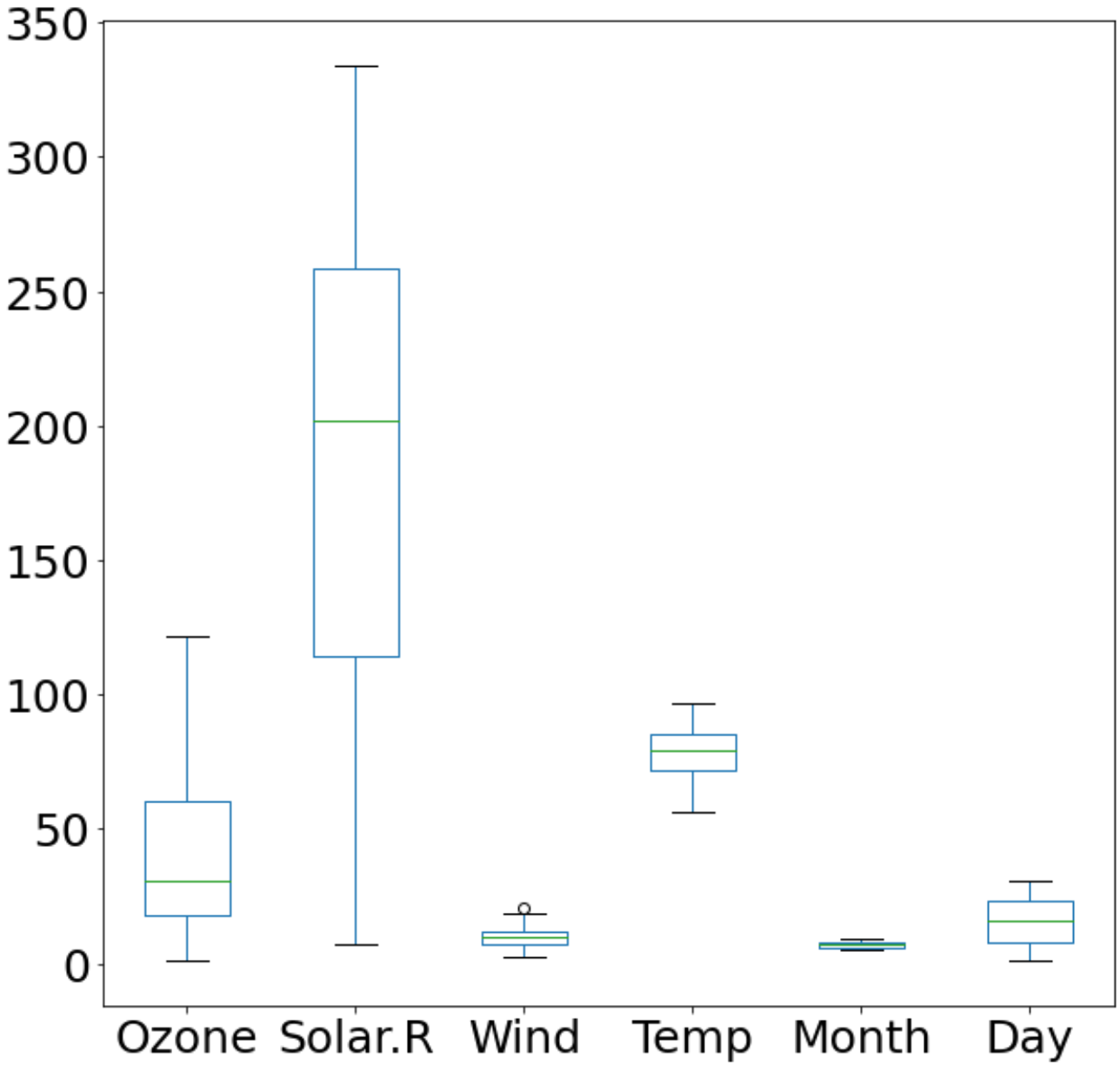
	Ozone	Solar.R	Wind	Temp	Month	Day
0	41.0	190.0	7.4	67.0	5	1
1	36.0	118.0	8.0	72.0	5	2
2	12.0	149.0	12.6	74.0	5	3
3	18.0	313.0	11.5	62.0	5	4
4	NaN	NaN	14.3	56.0	5	5
...	...	...	...	...	...	...
148	30.0	193.0	6.9	70.0	9	26
149	NaN	145.0	13.2	77.0	9	27
150	14.0	191.0	14.3	75.0	9	28
151	18.0	131.0	8.0	76.0	9	29
152	20.0	223.0	11.5	68.0	9	30

151 rows × 6 columns

In [54]:
```
#Showing a new boxplot without the outlier values  for Ozone (135,168)

boxplot = ozone_complete.boxplot(grid=False,fontsize=25,figsize=(10,10))
boxplot
```

Out[54]:
```
<AxesSubplot:>
```


350
300
250
200
150
100
50
0
Ozone
Solar.R
Wind
Temp
Month
Day