In [ ]:

```
#IMPORTING BANK DATASET
import pandas as pd
bank = pd.read_csv(r'C:\Users\NJERI SHAWN\Desktop\python pdf\Bank.csv')
print(bank)
bank['Salary'].mean() #same as sal.mean() after assigning
sal=bank['salary']
sal=bank['Salary']
sal.mean()
sal.min()
sal.max()
sal.median()
sal.std()
#or we can get all statistical summary of all numeric data in the dataset
by using the describe() command
bank.describe()
```

```
     Employee  EducLev  JobGrade  YrHired  YrBorn  Gender  YrsPrior PCJob  \
0           1        3         1       92      69    Male         1    No
1           2        1         1       81      57  Female         1    No
2           3        1         1       83      60  Female         0    No
3           4        2         1       87      55  Female         7    No
4           5        3         1       92      67    Male         0    No
..        ...      ...       ...      ...     ...     ...       ...   ...
203       204        3         6       61      35    Male         0    No
204       205        5         6       59      34    Male         0    No
205       206        5         6       63      33    Male         0    No
206       207        5         6       60      36    Male         0    No
207       208        5         6       62      33  Female         0    No

     Salary      Mgmt
0      32.0  Non-Mgmt
1      39.1  Non-Mgmt
2      33.2  Non-Mgmt
3      30.6  Non-Mgmt
4      29.0  Non-Mgmt
..      ...       ...
203    95.0      Mgmt
204    97.0      Mgmt
205    88.0      Mgmt
206    94.0      Mgmt
207    30.0      Mgmt

[208 rows x 10 columns]
```

Out[ ]:

	Employee	EducLev	JobGrade	YrHired	YrBorn	YrsPrior	Salary
count	208.000000	208.000000	208.000000	208.000000	208.000000	208.000000	208.000000
mean	104.500000	3.158654	2.759615	85.326923	54.605769	2.375000	39.921923
std	60.188592	1.467464	1.566529	6.987832	10.318988	3.135237	11.256154
min	1.000000	1.000000	1.000000	56.000000	30.000000	0.000000	26.700000
25%	52.750000	2.000000	1.000000	82.000000	47.750000	0.000000	33.000000
50%	104.500000	3.000000	3.000000	87.000000	56.500000	1.000000	37.000000
75%	156.250000	5.000000	4.000000	90.000000	63.000000	4.000000	44.000000
max	208.000000	5.000000	6.000000	93.000000	73.000000	18.000000	97.000000

In [ ]:

```
sal.mean(),sal.min(),sal.max(), sal.std()
```

Out[ ]:

```
(39.92192307692307, 26.7, 97.0, 11.256153921958742)
```

In [ ]:

```
sal.describe()
```

Out[ ]:

```
count    208.000000
mean      39.921923
std       11.256154
min       26.700000
25%       33.000000
50%       37.000000
75%       44.000000
max       97.000000
Name: Salary, dtype: float64
```

In [ ]:

```
bank['Salary'].describe()
```

Out[ ]:

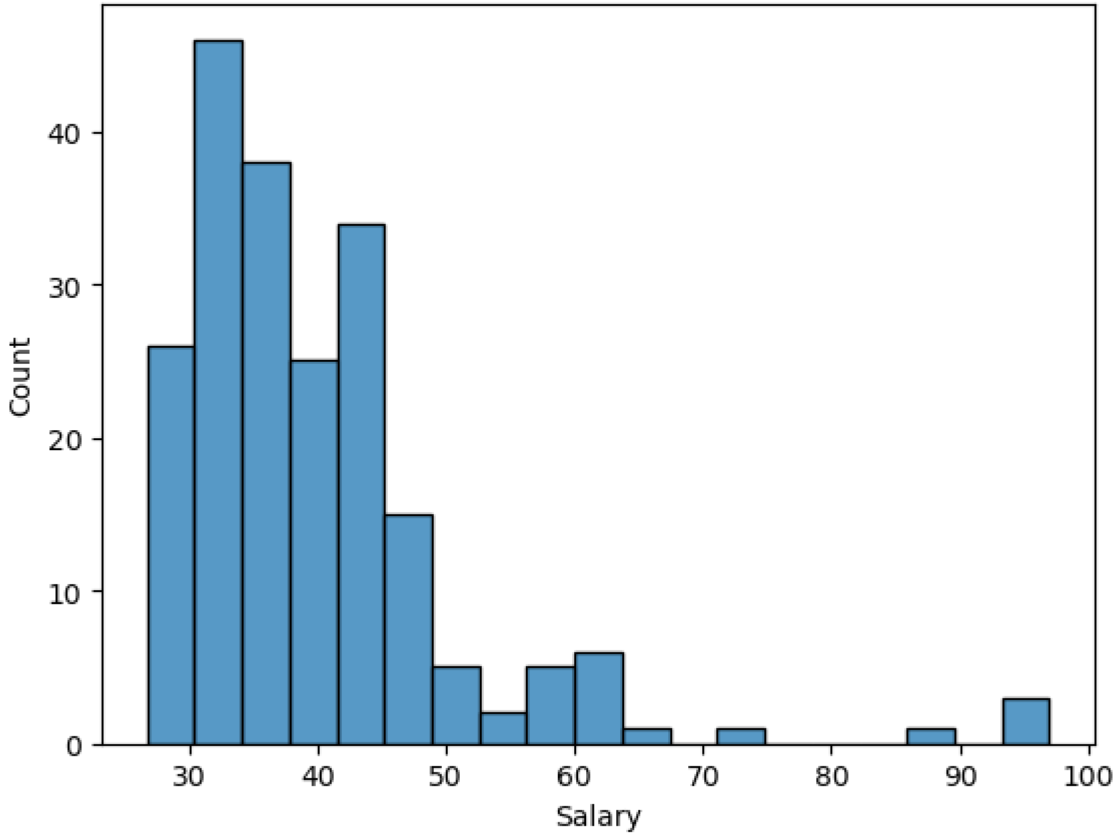
```
count    208.000000
mean      39.921923
std       11.256154
min       26.700000
25%       33.000000
50%       37.000000
75%       44.000000
max       97.000000
Name: Salary, dtype: float64
```

In [ ]:

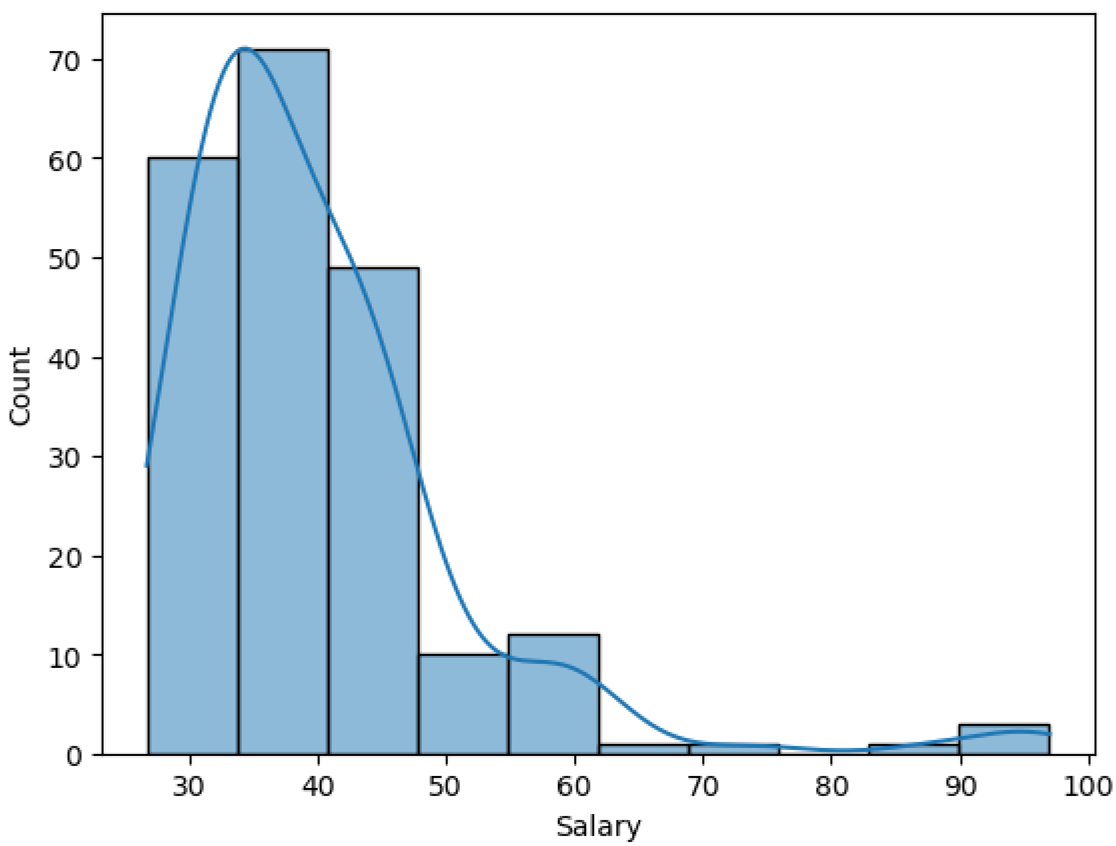
```
import seaborn as sns
sns.histplot(x=bank['Salary'])
```

Out[ ]:

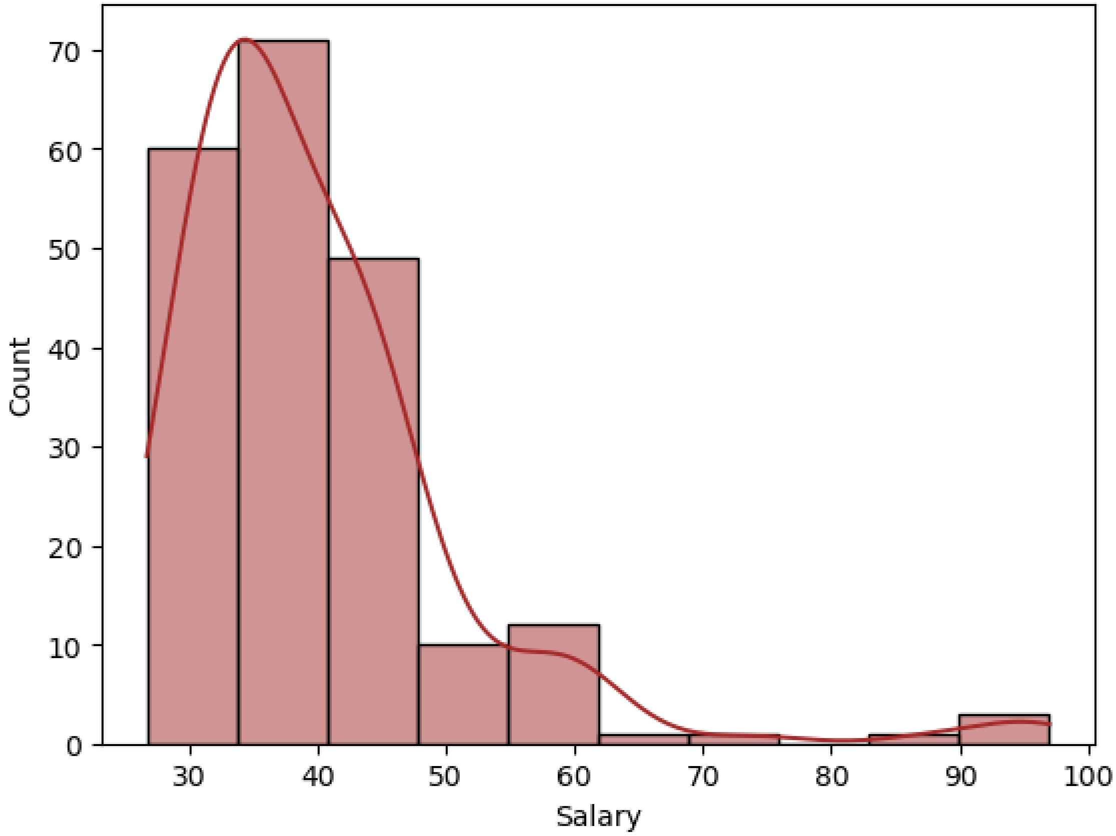
```
<Axes: xlabel='Salary', ylabel='Count'>
```



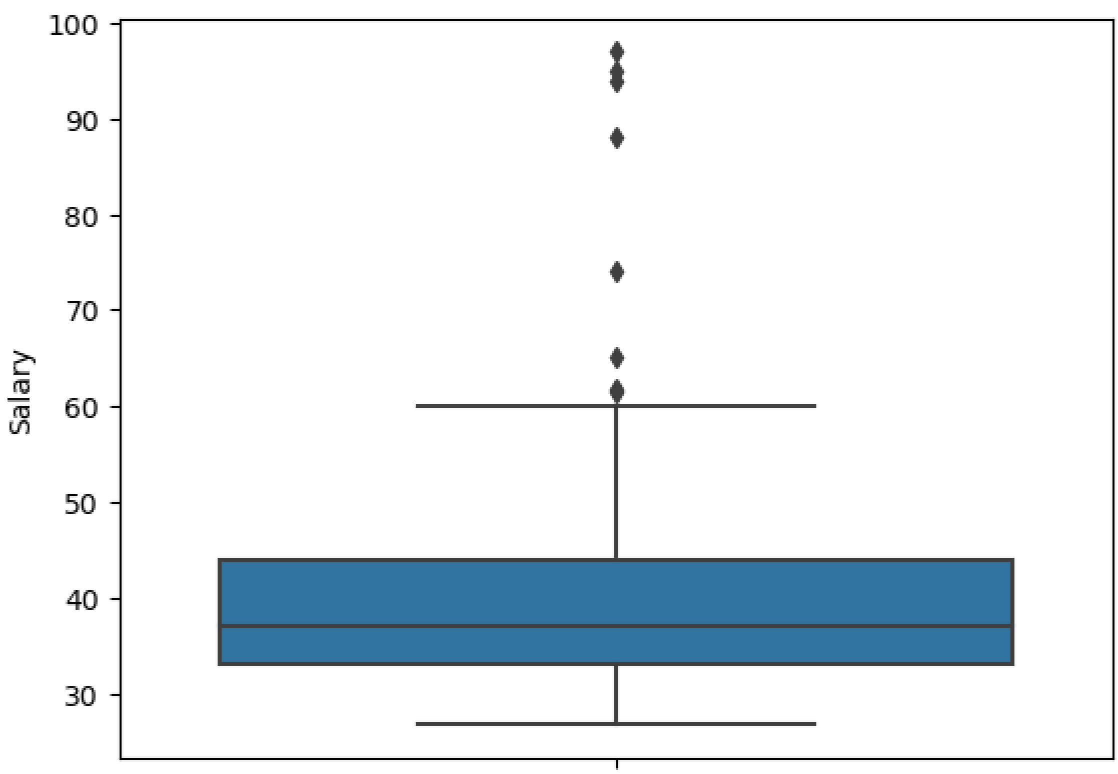
In [ ]:
```
sns.histplot(x=bank['Salary'], bins=10, kde=True);
```



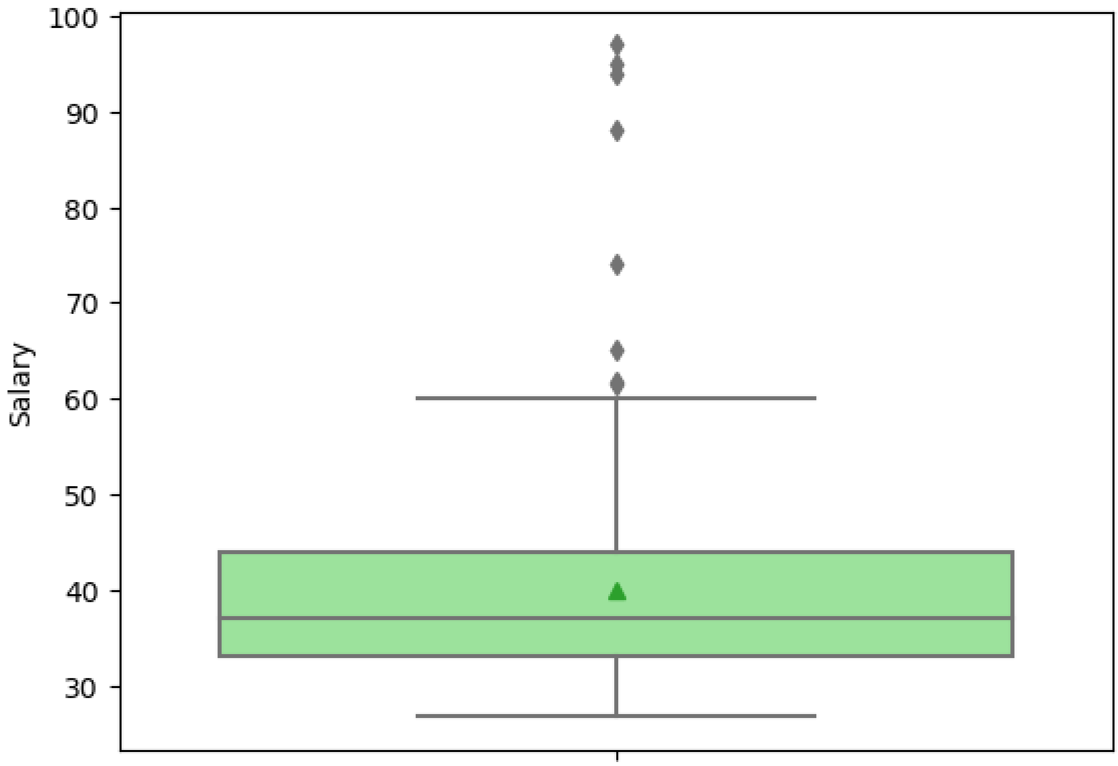
In [ ]:
```
sns.histplot(x=bank['Salary'],bins=10, kde=True,color='brown');
```



In [ ]:
```
sns.boxplot(y=bank['Salary']);
```



In [ ]:
```
sns.boxplot(y=bank['Salary'],color='lightgreen',showmeans=True);
```



In [ ]:
```
#Filtering Data in Python
```

In [ ]:
```
bank['Gender']=='Female'
```

Out[ ]:
```
0      False
1       True
2       True
3       True
4      False
       ...
203    False
204    False
205    False
206    False
207     True
Name: Gender, Length: 208, dtype: bool
```

In [ ]:
```
bank[bank['Gender']=='Female']
```

Out[ ]:

	Employee	EducLev	JobGrade	YrHired	YrBorn	Gender	YrsPrior	PCJob	Salary	Mgmt
1	2	1	1	81	57	Female	1	No	39.1	Non-Mgmt
2	3	1	1	83	60	Female	0	No	33.2	Non-Mgmt
3	4	2	1	87	55	Female	7	No	30.6	Non-Mgmt
5	6	3	1	92	71	Female	0	No	30.5	Non-Mgmt
6	7	3	1	91	68	Female	0	No	30.0	Non-Mgmt
...	...	...	...	...	...	...	...	...	...	...
186	187	5	5	86	58	Female	2	No	50.0	Mgmt
187	188	5	5	83	49	Female	2	No	61.8	Mgmt
188	189	4	5	79	52	Female	0	No	43.0	Mgmt
190	191	5	5	86	58	Female	6	No	58.5	Mgmt
207	208	5	6	62	33	Female	0	No	30.0	Mgmt

140 rows × 10 columns

In [ ]:

```
FemaleEmployees=bank[bank['Gender']=="Female"]
type(FemaleEmployees)
```

Out[ ]:

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

In [ ]:

```
round(FemaleEmployees['Salary'].mean(),2)
```

Out[ ]:

```
37.21
```

In [ ]:

```
(bank['Gender']=='Female')&(bank['JobGrade']==1)
```

Out[ ]:

```
0      False
1       True
2       True
3       True
4      False
       ...
203    False
204    False
205    False
206    False
207    False
Length: 208, dtype: bool
```

In [ ]:
```
bank[(bank['Gender']=='Female')&(bank['JobGrade']==1)].shape
```

Out[ ]:
```
(48, 10)
```

In [ ]:
```
bank[bank['JobGrade']>=4]
```

Out[ ]:

	Employee	EducLev	JobGrade	YrHired	YrBorn	Gender	YrsPrior	PCJob	Salary	Mgmt
145	146	5	4	90	62	Male	3	No	44.5	Non-Mgmt
146	147	5	4	91	65	Male	1	No	41.0	Non-Mgmt
147	148	5	4	89	58	Male	3	No	44.0	Non-Mgmt
148	149	5	4	89	65	Male	0	No	44.0	Non-Mgmt
149	150	5	4	90	63	Female	4	No	42.5	Non-Mgmt
...	...	...	...	...	...	...	...	...	...	...
203	204	3	6	61	35	Male	0	No	95.0	Mgmt
204	205	5	6	59	34	Male	0	No	97.0	Mgmt
205	206	5	6	63	33	Male	0	No	88.0	Mgmt
206	207	5	6	60	36	Male	0	No	94.0	Mgmt
207	208	5	6	62	33	Female	0	No	30.0	Mgmt

63 rows × 10 columns

In [ ]:
```
mgmt=[4,5,6]
bank[bank['JobGrade'].isin(mgmt)]


#isin() method checks if the dataframe contains the specified values
```

Out[ ]:

	Employee	EducLev	JobGrade	YrHired	YrBorn	Gender	YrsPrior	PCJob	Salary	Mgmt
145	146	5	4	90	62	Male	3	No	44.5	Non-Mgmt
146	147	5	4	91	65	Male	1	No	41.0	Non-Mgmt
147	148	5	4	89	58	Male	3	No	44.0	Non-Mgmt
148	149	5	4	89	65	Male	0	No	44.0	Non-Mgmt
149	150	5	4	90	63	Female	4	No	42.5	Non-Mgmt
...	...	...	...	...	...	...	...	...	...	...
203	204	3	6	61	35	Male	0	No	95.0	Mgmt
204	205	5	6	59	34	Male	0	No	97.0	Mgmt
205	206	5	6	63	33	Male	0	No	88.0	Mgmt
206	207	5	6	60	36	Male	0	No	94.0	Mgmt
207	208	5	6	62	33	Female	0	No	30.0	Mgmt

63 rows × 10 columns

In [ ]:

```
#Recoding Data in Python
```

In [ ]:

```
#Adding a Column in our Dataset
bank['Dummy']=0
bank.head()
```

Out[ ]:

	Employee	EducLev	JobGrade	YrHired	YrBorn	Gender	YrsPrior	PCJob	Salary	Mgmt	Dummy
0	1	3	1	92	69	Male	1	No	32.0	Non-Mgmt	0
1	2	1	1	81	57	Female	1	No	39.1	Non-Mgmt	0
2	3	1	1	83	60	Female	0	No	33.2	Non-Mgmt	0
3	4	2	1	87	55	Female	7	No	30.6	Non-Mgmt	0
4	5	3	1	92	67	Male	0	No	29.0	Non-Mgmt	0

In [ ]:

```
#Dropping a Column in our Dataset
bank.drop('Dummy',axis=1, inplace=True)
```

```
bank.head()
```

Out[ ]:

	Employee	EducLev	JobGrade	YrHired	YrBorn	Gender	YrsPrior	PCJob	Salary	Mgmt
0	1	3	1	92	69	Male	1	No	32.0	Non-Mgmt
1	2	1	1	81	57	Female	1	No	39.1	Non-Mgmt
2	3	1	1	83	60	Female	0	No	33.2	Non-Mgmt
3	4	2	1	87	55	Female	7	No	30.6	Non-Mgmt
4	5	3	1	92	67	Male	0	No	29.0	Non-Mgmt

In [ ]:

```
#Recoding using the numpy where method
import numpy as np
bank['GenderDummy_F'] = np.where(bank['Gender']=="Female",1,0)
bank.head()
```

Out[ ]:

	Employee	EducLev	JobGrade	YrHired	YrBorn	Gender	YrsPrior	PCJob	Salary	Mgmt	GenderD
0	1	3	1	92	69	Male	1	No	32.0	Non-Mgmt	
1	2	1	1	81	57	Female	1	No	39.1	Non-Mgmt	
2	3	1	1	83	60	Female	0	No	33.2	Non-Mgmt	
3	4	2	1	87	55	Female	7	No	30.6	Non-Mgmt	
4	5	3	1	92	67	Male	0	No	29.0	Non-Mgmt	

In [ ]:

```
#Recoding Using the apply() Function
#The easiest way to see how this works is to start with a parameterized
function that implements theif/then logic. What follows is a standard
function declaration in Python. The code defi nes a new function
called"my_recode" which takes a single parameter "gender". The function
returns a 1 or 0 depending on the value passed toit:

def my_recode(gender):
    if gender == "Female":
        return 1
    else:
        return 0
```

In [ ]:
```
my_recode("Female"), my_recode("Male")
```

Out[ ]:
```
(1, 0)
```

In [ ]:
```
bank['GenderDummy_F']=bank['Gender'].apply(my_recode)
bank.head()
```

Out[ ]:

	Employee	EducLev	JobGrade	YrHired	YrBorn	Gender	YrsPrior	PCJob	Salary	Mgmt	GenderD
0	1	3	1	92	69	Male	1	No	32.0	Non-Mgmt	
1	2	1	1	81	57	Female	1	No	39.1	Non-Mgmt	
2	3	1	1	83	60	Female	0	No	33.2	Non-Mgmt	
3	4	2	1	87	55	Female	7	No	30.6	Non-Mgmt	
4	5	3	1	92	67	Male	0	No	29.0	Non-Mgmt	

In [ ]:
```
#Recoding Using a Lambda Function
bank['GenderDummy_F']=bank['Gender'].apply(lambda x: 1 if x == "Female"
else 0)
bank.head()
```

Out[ ]:

	Employee	EducLev	JobGrade	YrHired	YrBorn	Gender	YrsPrior	PCJob	Salary	Mgmt	GenderD
0	1	3	1	92	69	Male	1	No	32.0	Non-Mgmt	
1	2	1	1	81	57	Female	1	No	39.1	Non-Mgmt	
2	3	1	1	83	60	Female	0	No	33.2	Non-Mgmt	
3	4	2	1	87	55	Female	7	No	30.6	Non-Mgmt	
4	5	3	1	92	67	Male	0	No	29.0	Non-Mgmt	

In [ ]:
```
#Replacing Values from a List
grades=[1,2,3,4,5,6]
status=["non-mgmt","non-mgmt","non-mgmt","non-mgmt","mgmt","mgmt"]
```

```
bank['Manager']=bank['JobGrade'].replace(grades,status)
bank.head()
```

Out[ ]:

	Employee	EducLev	JobGrade	YrHired	YrBorn	Gender	YrsPrior	PCJob	Salary	Mgmt	GenderD
0	1	3	1	92	69	Male	1	No	32.0	Non-Mgmt	
1	2	1	1	81	57	Female	1	No	39.1	Non-Mgmt	
2	3	1	1	83	60	Female	0	No	33.2	Non-Mgmt	
3	4	2	1	87	55	Female	7	No	30.6	Non-Mgmt	
4	5	3	1	92	67	Male	0	No	29.0	Non-Mgmt	

In [ ]:

```
genders=["Female", "Male"]
dummy_vars=[1,0]
bank['GenderDummy_F'] = bank['Gender'].replace(genders, dummy_vars)
bank.head()
```

Out[ ]:

	Employee	EducLev	JobGrade	YrHired	YrBorn	Gender	YrsPrior	PCJob	Salary	Mgmt	GenderD
0	1	3	1	92	69	Male	1	No	32.0	Non-Mgmt	
1	2	1	1	81	57	Female	1	No	39.1	Non-Mgmt	
2	3	1	1	83	60	Female	0	No	33.2	Non-Mgmt	
3	4	2	1	87	55	Female	7	No	30.6	Non-Mgmt	
4	5	3	1	92	67	Male	0	No	29.0	Non-Mgmt	

In [ ]:

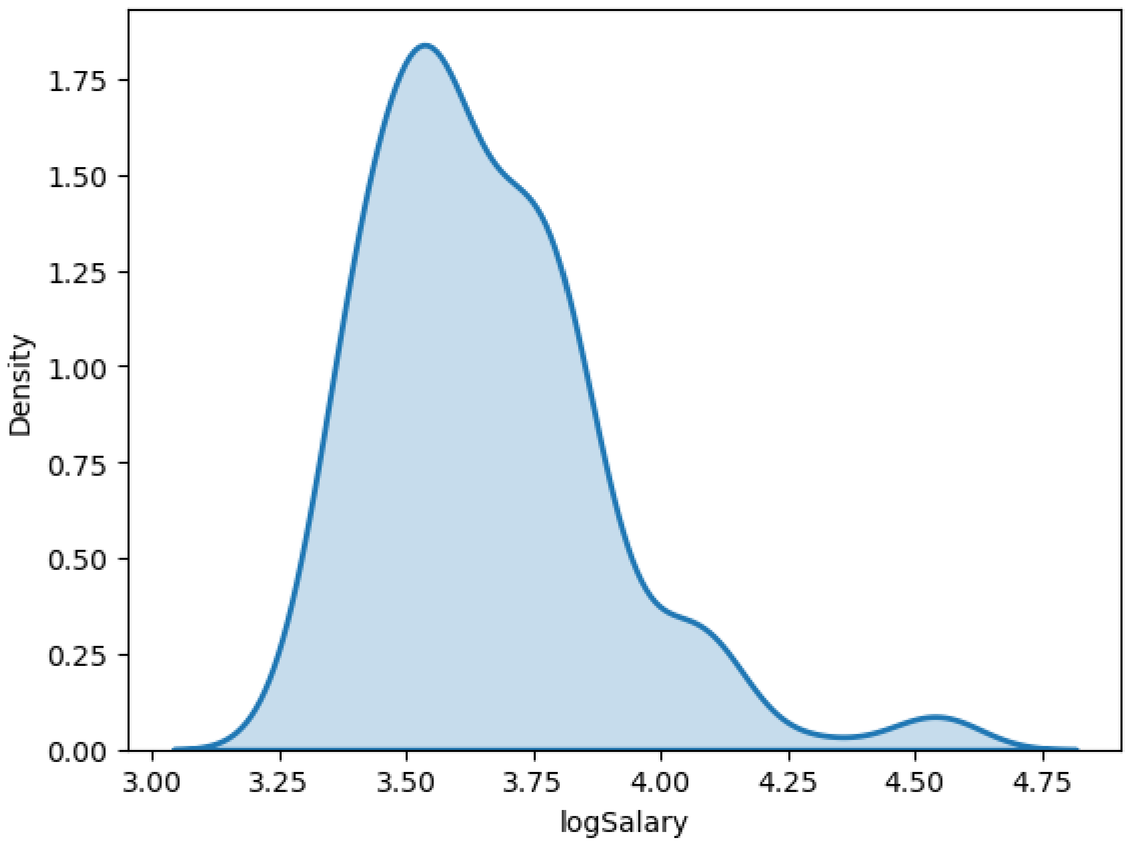
```
#Logging variables
bank['logSalary']=np.log(bank['Salary'])
bank.head()
```

Out[ ]:

	Employee	EducLev	JobGrade	YrHired	YrBorn	Gender	YrsPrior	PCJob	Salary	Mgmt	GenderD
0	1	3	1	92	69	Male	1	No	32.0	Non-Mgmt	
1	2	1	1	81	57	Female	1	No	39.1	Non-Mgmt	
2	3	1	1	83	60	Female	0	No	33.2	Non-Mgmt	
3	4	2	1	87	55	Female	7	No	30.6	Non-Mgmt	
4	5	3	1	92	67	Male	0	No	29.0	Non-Mgmt	

In [ ]:

```
import seaborn as sns
sns.kdeplot(x=bank['logSalary'],fill=True,linewidth=2);
```



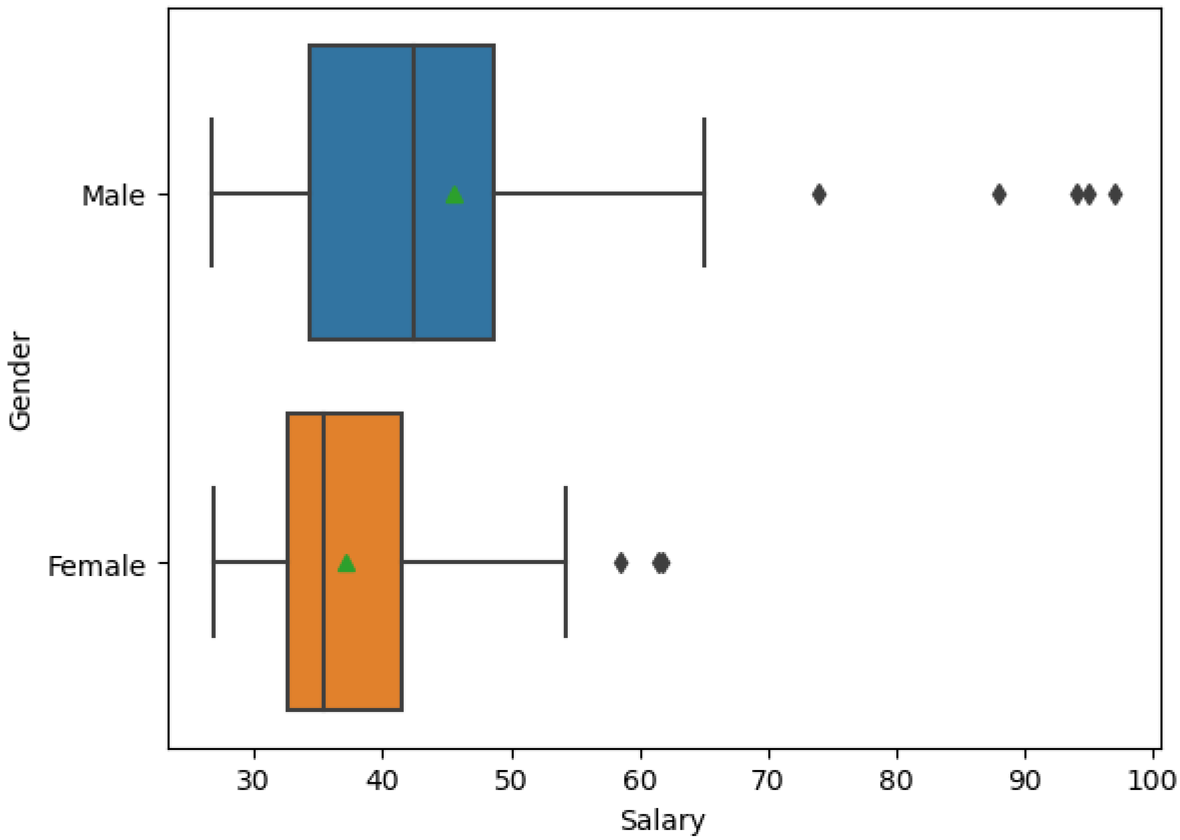
Gap analysis with Continuous Variables

In [ ]:

```
# Recall the purpose of Gap Analysis: determine whether two samples of
data are different. In our running example, we want to determine whether
Sample 1 (salaries of female employees in the bank) is different from
Sample 2 (salaries of male employees at the bank). We generally come at
```

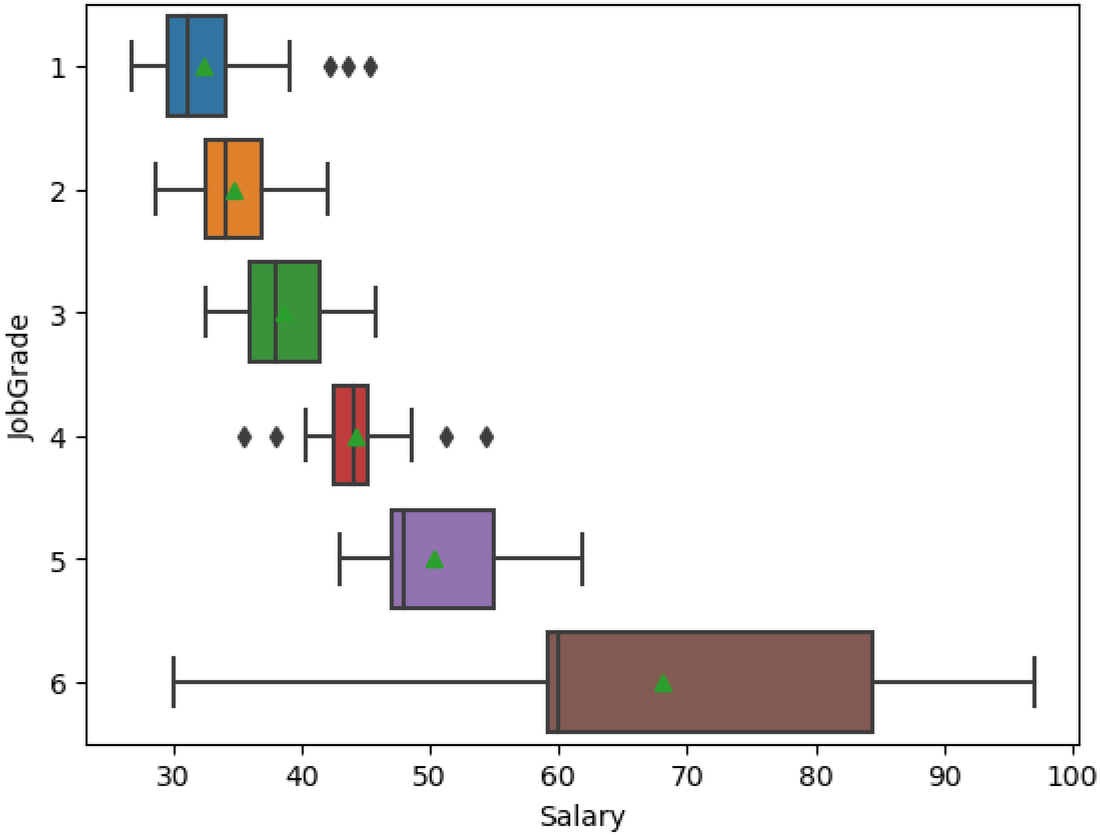
```
Gap Analysis in two steps:
        # 1. Plot the data in such a way that we can visually assess
whether a gap exists. These visualizations also come in handy later when
communicating the results of any formal analysis.
        # 2. Conduct a formal gap analysis using statistical techniques

#Using BoxPlots
#ensure Seaborn is loaded
import seaborn as sns
sns.boxplot(x=bank['Salary'], y=bank['Gender'], showmeans=True);
```



In [ ]:

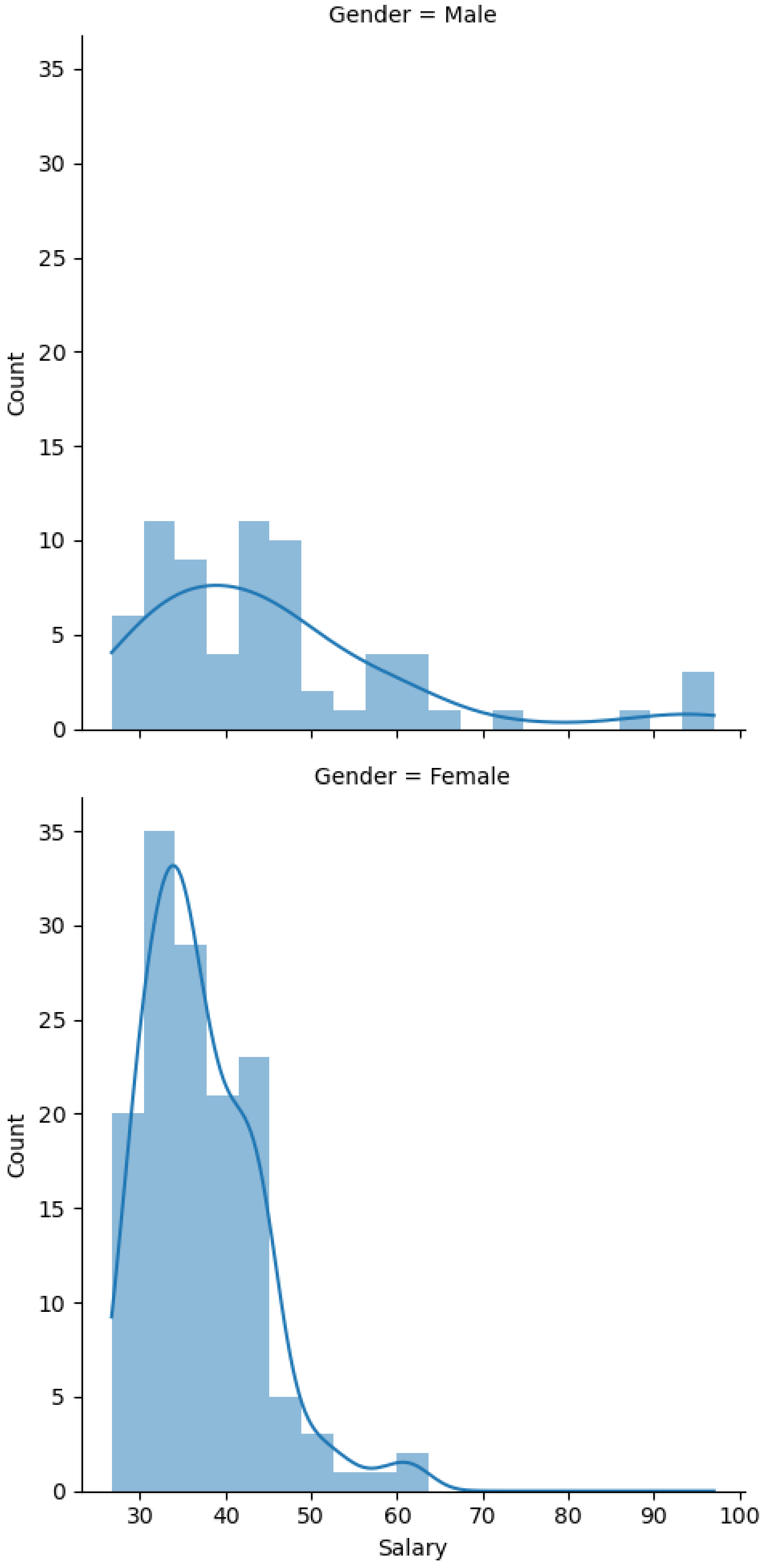
```
#As an aside: We can do the same kind of analysis by “JobGrade”. But
recall that we left JobGrade as an integer and did not convert it to a
category variable (as we did for Gender and PCJob). We can make this
conversion on the fly in order to get a boxplot:
sns.boxplot(x=bank['Salary'], y=bank['JobGrade'].astype('category'),
showmeans=True);
```



In [ ]:

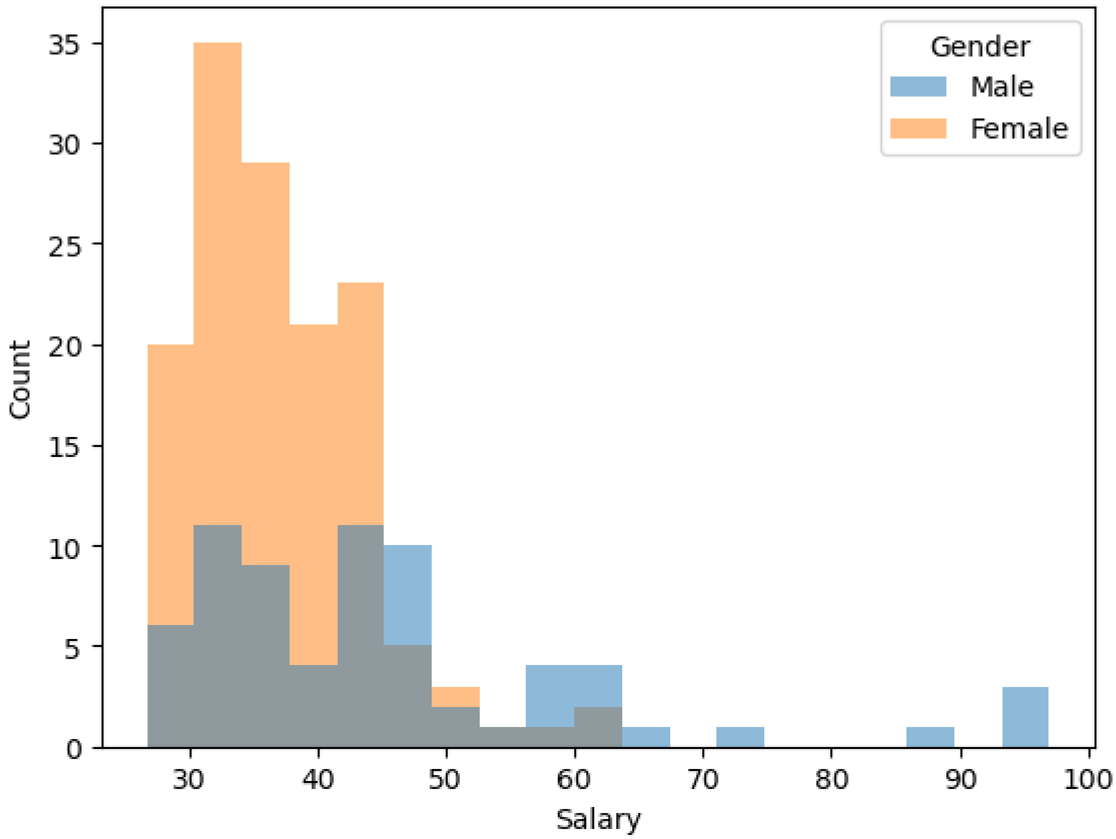
```
# Faceted histograms
# We used the notion of a “facet” in R to create a grid of histograms. In this case, we want a grid with one column and tworows. The rows correspond to different values of the “Gender” variable. This is a bit easier in Python with Seaborn’s displot function, which creates a faceted distribution plot. Here the row argument tells Seaborn to create one row foreach value of gender. I have also set the linewidth property to zero and added kernel density plots

sns.displot (x='Salary',row='Gender',data=bank,linewidth=0,kde=True);
```


Gender = Male
Gender = Female
Count
Count
Salary
0
5
10
15
20
25
30
35
30
40
50
60
70
80
90
100

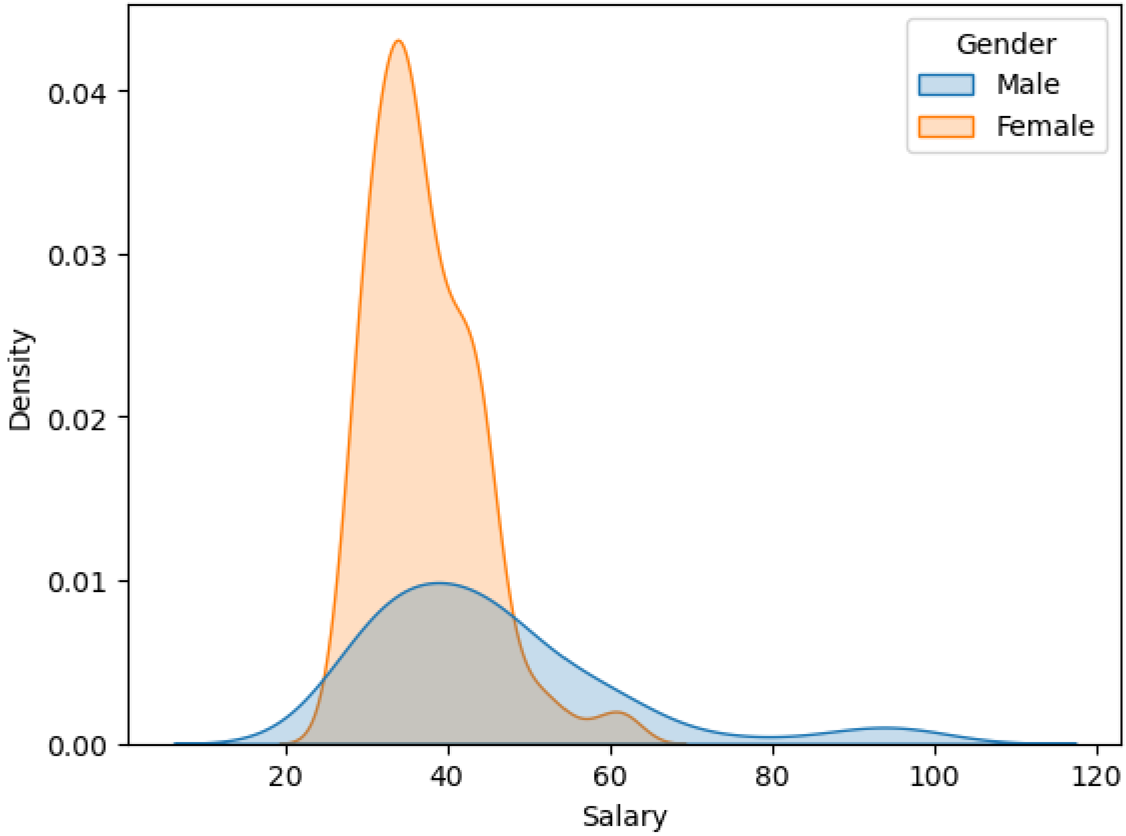
In [ ]:
```
#  As mentioned in our discussion of R, overlaying histograms almost never makes sense—the result is typically a mess,which is why SAS Enterprise Guide stacks them one on top of the other (as we just did above). Here is an overlayedhistogram for the bank salary data. Note that the color are added, so we get a third color in regions of overlap:

sns.histplot(x='Salary',hue='Gender',data=bank,linewidth=0);
```



In [ ]:
```
# A better approach is to stack kernel density plots. Note that I have also added some shading to make the result lookmarginally cooler

sns.kdeplot(x='Salary',hue='Gender',data=bank,fill=True);
```



T-tests

In [ ]:
```
# We use the t-test at this point to formally test the hypothesis that two
distributions have the same sample mean (and thus are "the same"—or at
least close enough). As in Excel and R, the two main preconditions to
running the test inPython are:
    # 1.Getting the data in the right format
    # 2.Determining which version of the t-test to run: equal variance or
unequal variance

female_sal=bank[bank['Gender']=="Female"]['Salary']
male_sal=bank[bank['Gender']=="Male"]['Salary']
female_sal

# Testing for equality of varianceUsing Levene's Test
# ensure the scipy stats module is loaded

from scipy import stats
stats.levene(female_sal,male_sal)
```

```
# from the  levene results below , we seen that the version of t-test to run is of unequal variance since p value has e-07, meaning it is smaller enough to treat as zero
```

Out[ ]:

```
LeveneResult(statistic=26.208558688866834, pvalue=7.013920590029544e-07)
```

Running the t-test to compare the means

In [ ]:

```
import statsmodels.stats.api as sms
model=sms.CompareMeans.from_data(bank[bank['Gender']=="Female"]['Salary'],bank[bank['Gender']=="Male"]['Salary'])
model.summary(usevar='unequal')
```

Out[ ]:

Test for equality of means

	coef	std err	t	P>|t|	[0.025	0.975]
subset #1	-8.2955	2.003	-4.141	0.000	-12.283	-4.308

In [ ]:

```
import statsmodels.stats.api as sms
model=sms.CompareMeans.from_data(female_sal,male_sal)
model.summary(usevar='unequal')
```

Out[ ]:

Test for equality of means

	coef	std err	t	P>|t|	[0.025	0.975]
subset #1	-8.2955	2.003	-4.141	0.000	-12.283	-4.308

In [ ]:

```
```