In [1]:
```
#DATA ANALYSIS AND INTERPRETATIONS
```

In [2]:
```
#importing required library
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [3]:
```
df = pd.read_csv(r"C:\Users\91821\Documents\sarvesh\hrdataset.csv")
df
```

Out[3]:

	Gender	Business	Dependancies	Calls	Type	Billing	Rating	Age	Salary	Ba
0	Female	0	No	Yes	Month-to-month	No	Yes	18	5089.00	2(
1	Female	0	No	Yes	Month-to-month	No	Yes	19	5698.12	2:
2	Male	0	No	Yes	Month-to-month	Yes	No	22	5896.65	2:
3	Female	1	No	Yes	Month-to-month	Yes	Yes	21	6125.12	2-
4	Male	0	No	Yes	Month-to-month	Yes	Yes	23	6245.00	2-
...	...	...	...	...	...	...	...	...	...	
4995	Female	0	No	Yes	Month-to-month	No	No	72	180696.80	72:
4996	Male	0	No	Yes	Month-to-month	Yes	No	73	185685.90	74:
4997	Male	0	No	Yes	Month-to-month	Yes	No	74	192636.80	77(
4998	Male	1	No	Yes	Month-to-month	Yes	Yes	74	195970.70	78:
4999	Male	0	Yes	Yes	Two year	Yes	No	88	199970.74	79:

5000 rows × 20 columns

In [4]:
```
df.head()
```

Out[4]:

	Gender	Business	Dependancies	Calls	Type	Billing	Rating	Age	Salary	Base_pay
0	Female	0	No	Yes	Month-to-month	No	Yes	18	5089.00	2035.60(
1	Female	0	No	Yes	Month-to-month	No	Yes	19	5698.12	2279.24{
2	Male	0	No	Yes	Month-to-month	Yes	No	22	5896.65	2358.66(
3	Female	1	No	Yes	Month-to-month	Yes	Yes	21	6125.12	2450.04{
4	Male	0	No	Yes	Month-to-month	Yes	Yes	23	6245.00	2498.00(

In [5]:
```
df = df.replace(r'^\s*$', float(np.nan), regex=True)
```

In [6]:
```
df.tail()
```

Out[6]:

	Gender	Business	Dependancies	Calls	Type	Billing	Rating	Age	Salary	Ba
4995	Female	0	No	Yes	Month-to-month	No	No	72	180696.80	72(
4996	Male	0	No	Yes	Month-to-month	Yes	No	73	185685.90	74(
4997	Male	0	No	Yes	Month-to-month	Yes	No	74	192636.80	77(
4998	Male	1	No	Yes	Month-to-month	Yes	Yes	74	195970.70	78(
4999	Male	0	Yes	Yes	Two year	Yes	No	88	199970.74	79(

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 20 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Gender          5000 non-null   object 
 1   Business        5000 non-null   int64  
 2   Dependancies    5000 non-null   object 
 3   Calls           5000 non-null   object 
 4   Type            5000 non-null   object 
 5   Billing         5000 non-null   object 
 6   Rating          5000 non-null   object 
 7   Age             5000 non-null   int64  
 8   Salary          5000 non-null   float64
 9   Base_pay        4977 non-null   float64
 10  Bonus           5000 non-null   float64
 11  Unit_Price      5000 non-null   float64
 12  Volume          5000 non-null   int64  
 13  openingbalance  3524 non-null   float64
 14  closingbalance  5000 non-null   float64
 15  low             5000 non-null   float64
 16  Unit_Sales      5000 non-null   float64
 17  Total_Sales     4984 non-null   object 
 18  Months          5000 non-null   int64  
 19  Education       5000 non-null   object 
dtypes: float64(8), int64(4), object(8)
memory usage: 781.4+ KB
```

In [8]:
```
df['Total_Sales'] = df['Total_Sales'].astype('float64')
```

In [9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 20 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Gender          5000 non-null   object 
 1   Business        5000 non-null   int64  
 2   Dependancies    5000 non-null   object 
 3   Calls           5000 non-null   object 
 4   Type            5000 non-null   object 
 5   Billing         5000 non-null   object 
 6   Rating          5000 non-null   object 
 7   Age             5000 non-null   int64  
 8   Salary          5000 non-null   float64
 9   Base_pay        4977 non-null   float64
 10  Bonus           5000 non-null   float64
 11  Unit_Price      5000 non-null   float64
 12  Volume          5000 non-null   int64  
 13  openingbalance  3524 non-null   float64
 14  closingbalance  5000 non-null   float64
 15  low             5000 non-null   float64
 16  Unit_Sales      5000 non-null   float64
 17  Total_Sales     4984 non-null   float64
 18  Months          5000 non-null   int64  
 19  Education       5000 non-null   object 
dtypes: float64(9), int64(4), object(7)
memory usage: 781.4+ KB
```

In [10]:
```
missing_values = df.isnull().sum()
missing_values
```

Out[10]:
```
Gender              0
Business            0
Dependancies        0
Calls               0
Type                0
Billing             0
Rating              0
Age                 0
Salary              0
Base_pay           23
Bonus               0
Unit_Price          0
Volume              0
openingbalance    1476
closingbalance       0
low                  0
Unit_Sales           0
Total_Sales         16
Months               0
Education            0
dtype: int64
```

In [11]:
```
df.isnull().sum() * 100 / len(df)
```

Out[11]:
```
Gender             0.00
Business           0.00
Dependancies       0.00
Calls              0.00
Type               0.00
Billing            0.00
Rating             0.00
Age                0.00
Salary             0.00
Base_pay           0.46
Bonus              0.00
Unit_Price         0.00
Volume             0.00
openingbalance    29.52
closingbalance     0.00
low                0.00
Unit_Sales         0.00
Total_Sales        0.32
Months             0.00
Education          0.00
dtype: float64
```

In [12]:
```
from sklearn.impute import KNNImputer
x=df[['Base_pay','openingbalance','Total_Sales']]

imputer= KNNImputer(n_neighbors=2)
x=imputer.fit_transform(x)
```

In [13]:
```
df[['Base_pay', 'openingbalance', 'Total_Sales']] = pd.DataFrame(x,
columns = ['Base_pay', 'openingbalance', 'Total_Sales'])
```

In [14]:
```
df.isna().sum()
```

Out[14]:
```
Gender             0
Business           0
Dependancies       0
Calls              0
Type               0
Billing            0
Rating             0
Age                0
Salary             0
Base_pay           0
Bonus              0
Unit_Price         0
Volume             0
openingbalance     0
closingbalance     0
low                0
Unit_Sales         0
Total_Sales        0
Months             0
Education          0
dtype: int64
```

In [15]:
```
x
```

Out[15]:
```
array([[2.03560000e+03, 3.75000000e+00, 1.88000000e+01],
       [2.27924800e+03, 3.85000000e+00, 1.88500000e+01],
       [2.35866000e+03, 4.23000000e+00, 1.89000000e+01],
       ...,
       [7.70547200e+04, 3.09164993e+02, 8.31165000e+03],
       [7.83882800e+04, 3.09164993e+02, 8.31165000e+03],
       [7.99882960e+04, 3.09164993e+02, 8.31165000e+03]])
```

In [16]:
```
degree_wise=df.Education.value_counts()
degree_wise
```

Out[16]:
```
Education
PG                     2979
Graduation             1980
Intermediate             27
High School or less      14
Name: count, dtype: int64
```

In [17]:
```
Gender_wise=df.Gender.value_counts()
Gender_wise
```

Out[17]:
```
Gender
Male      2528
Female    2472
Name: count, dtype: int64
```

In [18]:
```
Rating_wise=df.Rating.value_counts()
Rating_wise
```

Out[18]:
```
Rating
No     3682
Yes    1318
Name: count, dtype: int64
```

In [19]:
```
call_wise=df.Calls.value_counts()
call_wise
```

Out[19]:
```
Calls
Yes    4539
No      461
Name: count, dtype: int64
```

In [20]:
```
Dependancies_wise=df.Dependancies.value_counts()
Dependancies_wise
```

Out[20]:
```
Dependancies
No     3524
Yes    1476
Name: count, dtype: int64
```

In [21]:
```
Age_wise=df.Age.value_counts()
Age_wise
```

Out[21]:
```
Age
50    256
53    254
55    248
54    245
51    244
     ...
88      2
80      1
82      1
85      1
79      1
Name: count, Length: 65, dtype: int64
```

In [22]:
```
Type_wise=df.Type.value_counts()
Type_wise
```

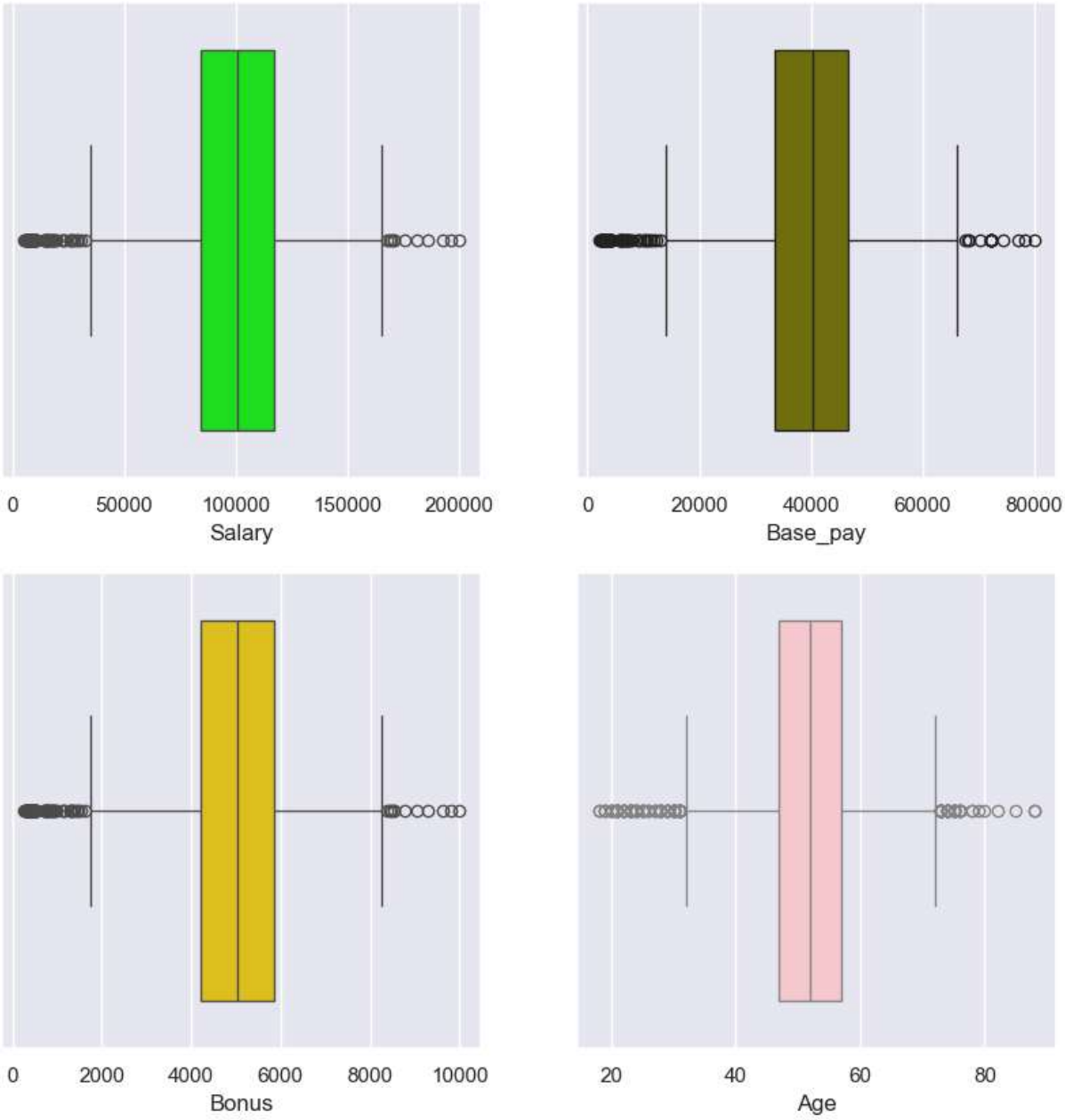
Out[22]:
```
Type
Month-to-month    2777
Two year          1195
One year          1028
Name: count, dtype: int64
```

In [23]:
```
Billing_wise=df.Billing.value_counts()
Billing_wise
```

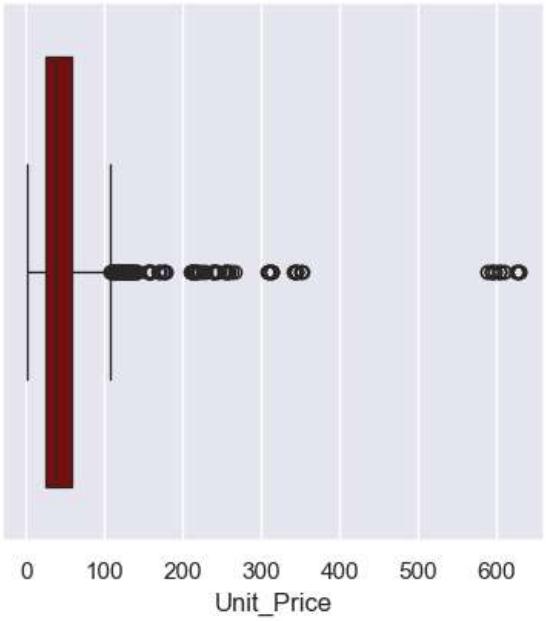
Out[23]:
```
Billing
Yes    2956
No     2044
Name: count, dtype: int64
```

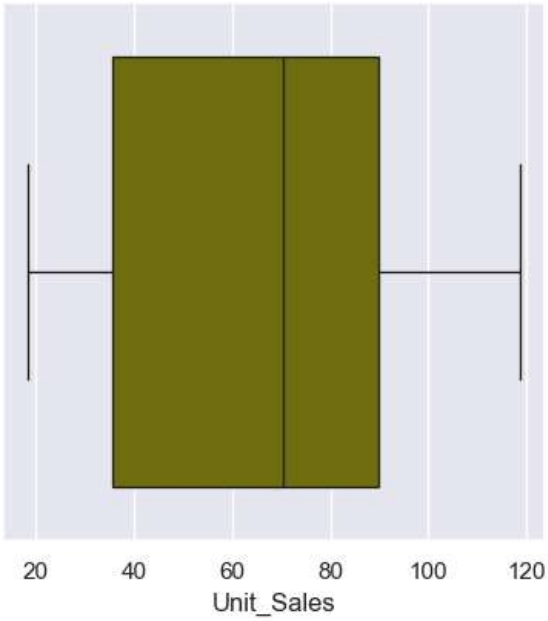
In [24]:
```
#BOX PLOT TO SEE DISTRIBUTION
```

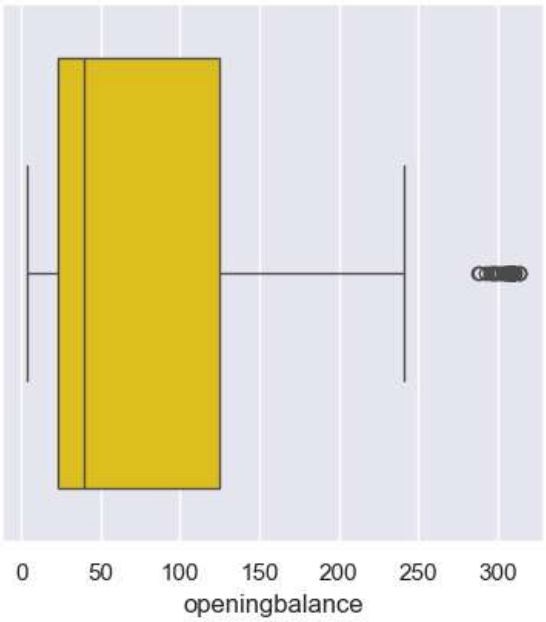
In [25]:
```
# setting a grey background
sns.set(style="darkgrid")
# Creating subplots with 10*10 figure size
fig, axs = plt.subplots(2, 2, figsize=(10, 10))

# Ploting subplots with variables
sns.boxplot(data=df, x="Salary", color="Lime", ax=axs[0, 0])# boxplot to see Distri
sns.boxplot(data=df, x="Base_pay", color="olive", ax=axs[0, 1])# boxplot to see Dis
sns.boxplot(data=df, x="Bonus", color="gold", ax=axs[1, 0])# boxplot to see Distrib
sns.boxplot(data=df, x="Age", color="pink", ax=axs[1, 1])# boxplot to see Distribut
plt.show()
```

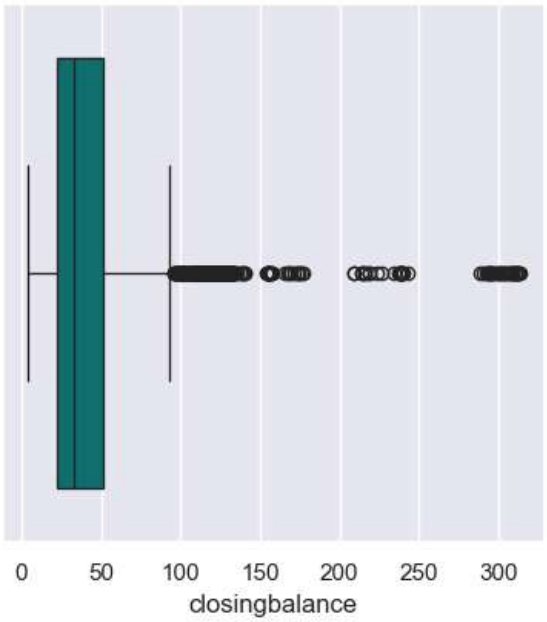


In [26]:
```
# setting a grey background
sns.set(style="darkgrid")
# Creating subplots with 10*10 figure size
fig, axs = plt.subplots(2, 2, figsize=(10, 10))

# Ploting subplots with all variables
sns.boxplot(data=df, x="Unit_Price", color="darkred", ax=axs[0, 0])# boxplot to see
sns.boxplot(data=df, x="Unit_Sales", color="olive", ax=axs[0, 1])# boxplot to see o
sns.boxplot(data=df, x="openingbalance", color="gold", ax=axs[1, 0])# boxplot to se
sns.boxplot(data=df, x="closingbalance", color="teal", ax=axs[1, 1])# boxplot to se

plt.show()
```

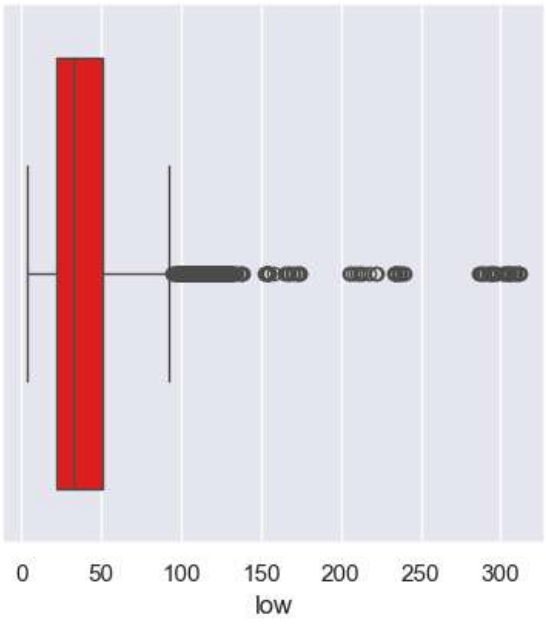


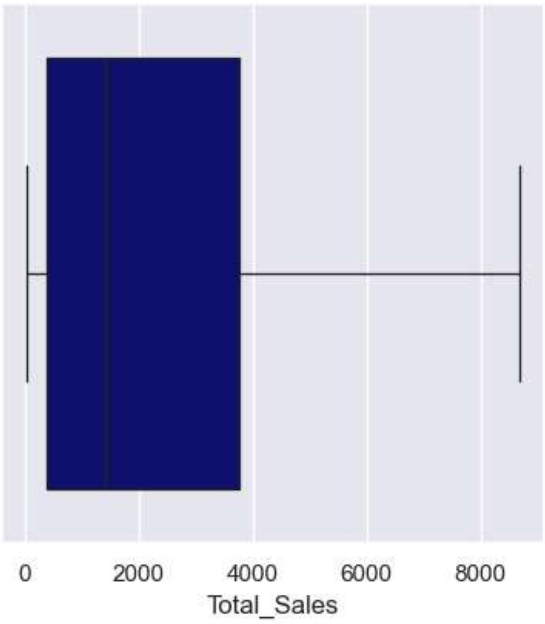


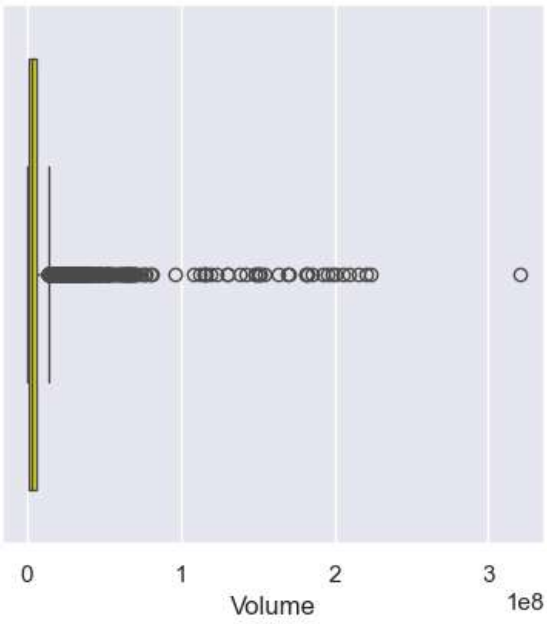


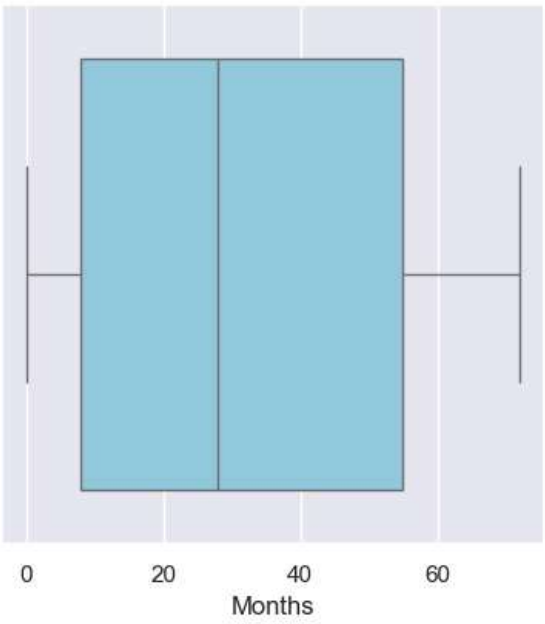


In [27]:
```
sns.set(style="darkgrid")
# Creating subplots with 10*10 figure size
fig, axs = plt.subplots(2, 2, figsize=(10, 10))

# Ploting subplots with variables
sns.boxplot(data=df, x="low", color="red", ax=axs[0,0])# boxplot to see outliers of
sns.boxplot(data=df, x="Total_Sales", color="Navy", ax=axs[0,1])# boxplot to see ou
sns.boxplot(data=df, x="Volume", color="Yellow", ax=axs[1,0])# boxplot to see outli
sns.boxplot(data=df, x="Months", color="skyblue", ax=axs[1,1])# boxplot to see outl
plt.show()
```









In [28]:
```
#STATISTICAL APPROACH QQPLOT TO SEE THE DISTRIBUTION
```

In [29]:
```
pip install pingouin
```

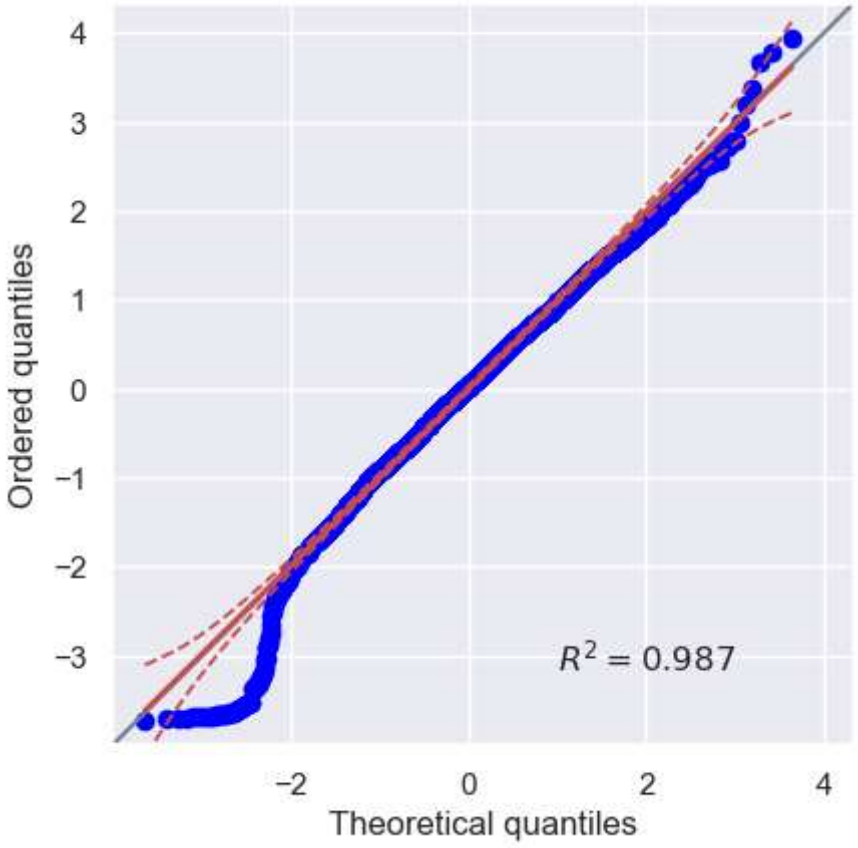
```
Requirement already satisfied: pingouin in c:\users\91821\anaconda3\lib\site-package
s (0.5.5)
Requirement already satisfied: matplotlib in c:\users\91821\anaconda3\lib\site-packa
ges (from pingouin) (3.10.0)
Requirement already satisfied: numpy in c:\users\91821\anaconda3\lib\site-packages
(from pingouin) (2.1.3)
Requirement already satisfied: pandas>=1.5 in c:\users\91821\anaconda3\lib\site-pack
ages (from pingouin) (2.2.3)
Requirement already satisfied: pandas-flavor in c:\users\91821\anaconda3\lib\site-pa
ckages (from pingouin) (0.7.0)
Requirement already satisfied: scikit-learn>=1.2 in c:\users\91821\anaconda3\lib\sit
e-packages (from pingouin) (1.6.1)
Requirement already satisfied: scipy in c:\users\91821\anaconda3\lib\site-packages
(from pingouin) (1.15.3)
Requirement already satisfied: seaborn in c:\users\91821\anaconda3\lib\site-packages
(from pingouin) (0.13.2)
Requirement already satisfied: statsmodels in c:\users\91821\anaconda3\lib\site-pack
ages (from pingouin) (0.14.4)
Requirement already satisfied: tabulate in c:\users\91821\anaconda3\lib\site-package
s (from pingouin) (0.9.0)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\91821\anaconda3\li
b\site-packages (from pandas>=1.5->pingouin) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\91821\anaconda3\lib\site-pac
kages (from pandas>=1.5->pingouin) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\91821\anaconda3\lib\site-p
ackages (from pandas>=1.5->pingouin) (2025.2)
Requirement already satisfied: six>=1.5 in c:\users\91821\anaconda3\lib\site-package
s (from python-dateutil>=2.8.2->pandas>=1.5->pingouin) (1.17.0)
Requirement already satisfied: joblib>=1.2.0 in c:\users\91821\anaconda3\lib\site-pa
ckages (from scikit-learn>=1.2->pingouin) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\91821\anaconda3\lib
\site-packages (from scikit-learn>=1.2->pingouin) (3.5.0)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\91821\anaconda3\lib\site
-packages (from matplotlib->pingouin) (1.3.1)
Requirement already satisfied: cycler>=0.10 in c:\users\91821\anaconda3\lib\site-pac
kages (from matplotlib->pingouin) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\91821\anaconda3\lib\sit
e-packages (from matplotlib->pingouin) (4.55.3)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\91821\anaconda3\lib\sit
e-packages (from matplotlib->pingouin) (1.4.8)
Requirement already satisfied: packaging>=20.0 in c:\users\91821\anaconda3\lib\site-
packages (from matplotlib->pingouin) (24.2)
Requirement already satisfied: pillow>=8 in c:\users\91821\anaconda3\lib\site-packag
es (from matplotlib->pingouin) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\91821\anaconda3\lib\site
-packages (from matplotlib->pingouin) (3.2.0)
Requirement already satisfied: xarray in c:\users\91821\anaconda3\lib\site-packages
(from pandas-flavor->pingouin) (2025.4.0)
Requirement already satisfied: patsy>=0.5.6 in c:\users\91821\anaconda3\lib\site-pac
kages (from statsmodels->pingouin) (1.0.1)
Note: you may need to restart the kernel to use updated packages.
```

In [30]:
```
#IF THE DATA IS NORMALLY DISTRIBUTED, THE POINTS IN THE QQ-NORMAL PLOT LIE ON A STR
```

In [31]:
```
import pingouin as pg
```

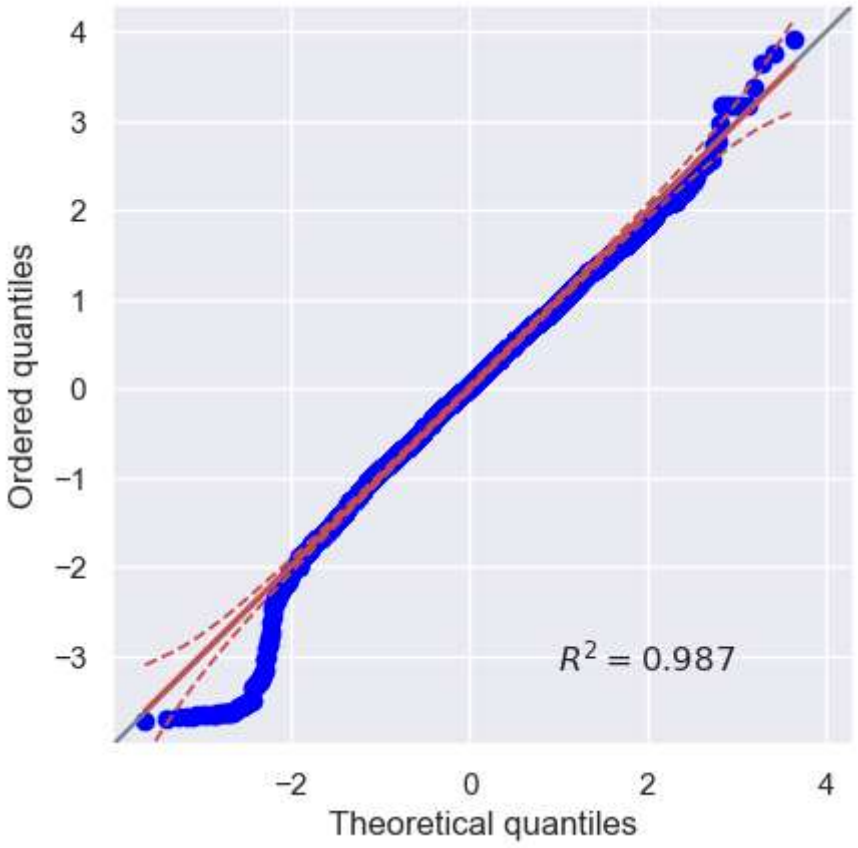
```
pg.qqplot(df['Salary'],dist ='norm')
```

Out[31]: <Axes: xlabel='Theoretical quantiles', ylabel='Ordered quantiles'>



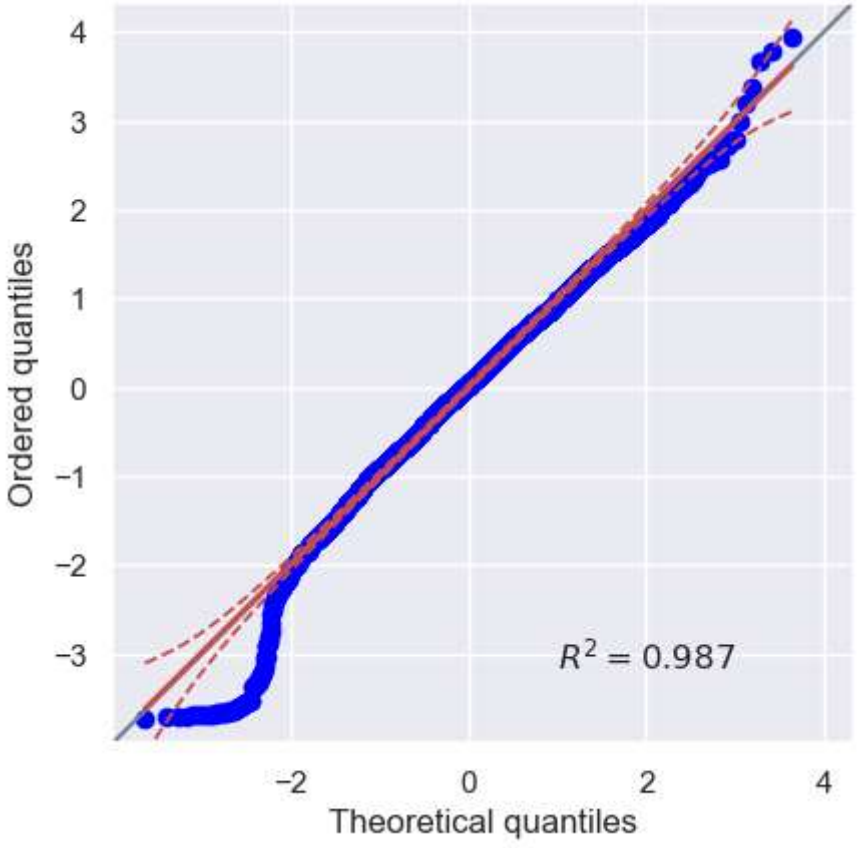
In [32]:
```
pg.qqplot(df['Base_pay'],dist ='norm')
```

Out[32]: <Axes: xlabel='Theoretical quantiles', ylabel='Ordered quantiles'>



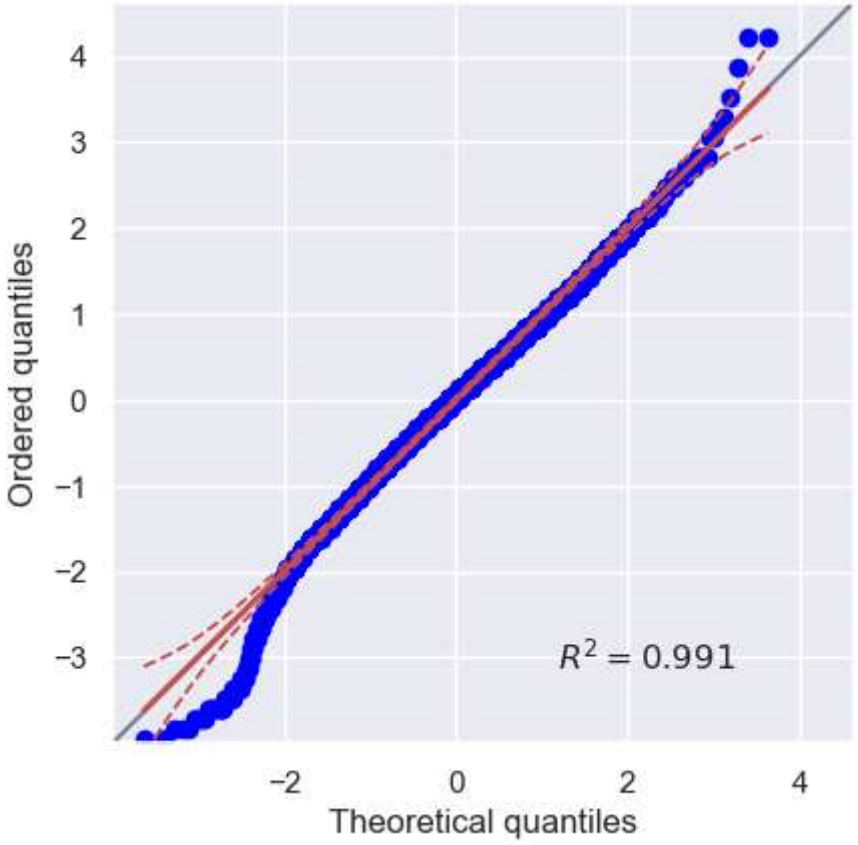
In [33]:
```
pg.qqplot(df['Bonus'],dist ='norm')
```

Out[33]: <Axes: xlabel='Theoretical quantiles', ylabel='Ordered quantiles'>



In [34]:
```
pg.qqplot(df['Age'],dist ='norm')
```

Out[34]:
```
<Axes: xlabel='Theoretical quantiles', ylabel='Ordered quantiles'>
```

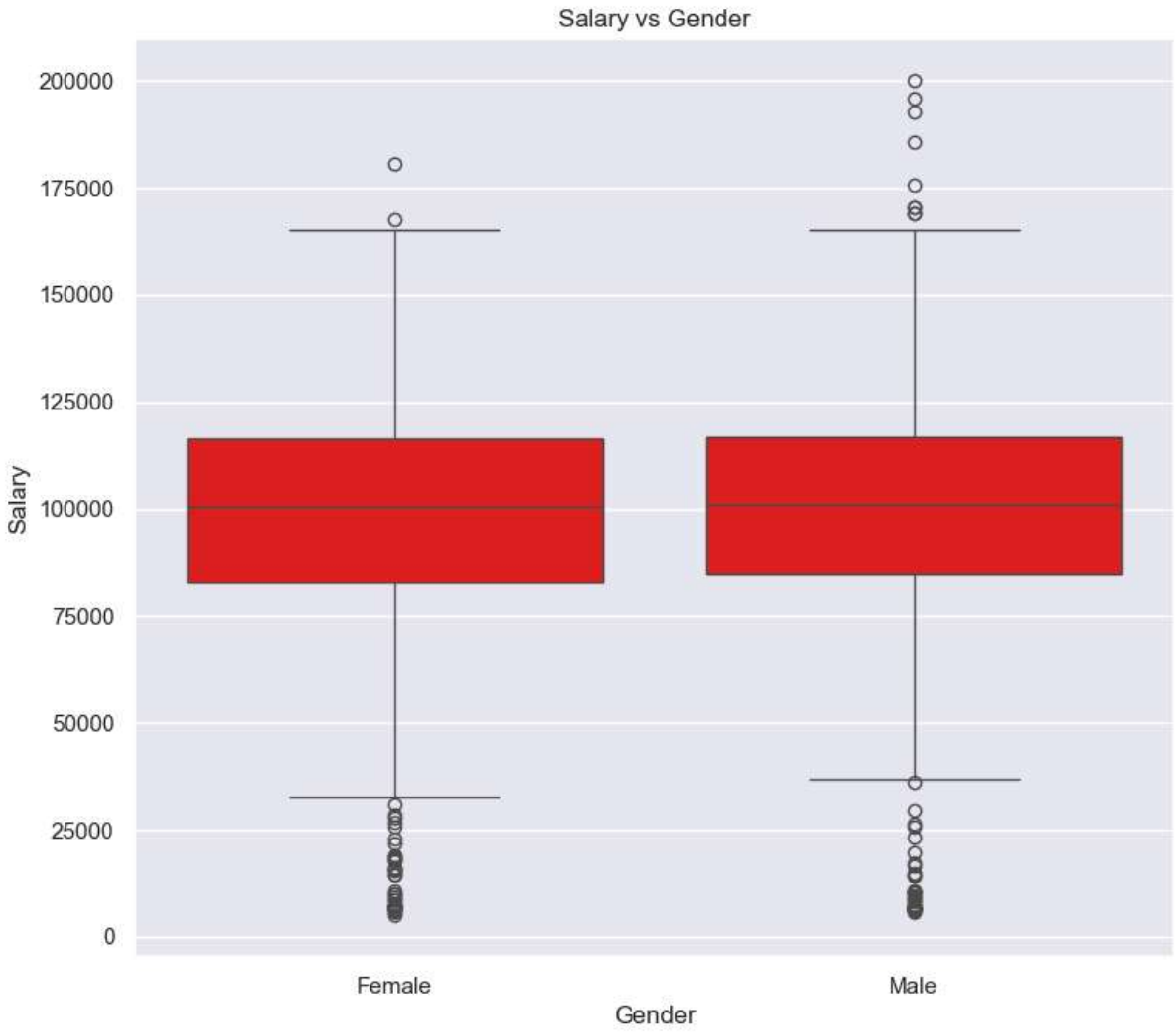


In [35]:
```
df.select_dtypes(include=['object']).columns.tolist()
```

Out[35]:
```
['Gender', 'Dependancies', 'Calls', 'Type', 'Billing', 'Rating', 'Education']
```

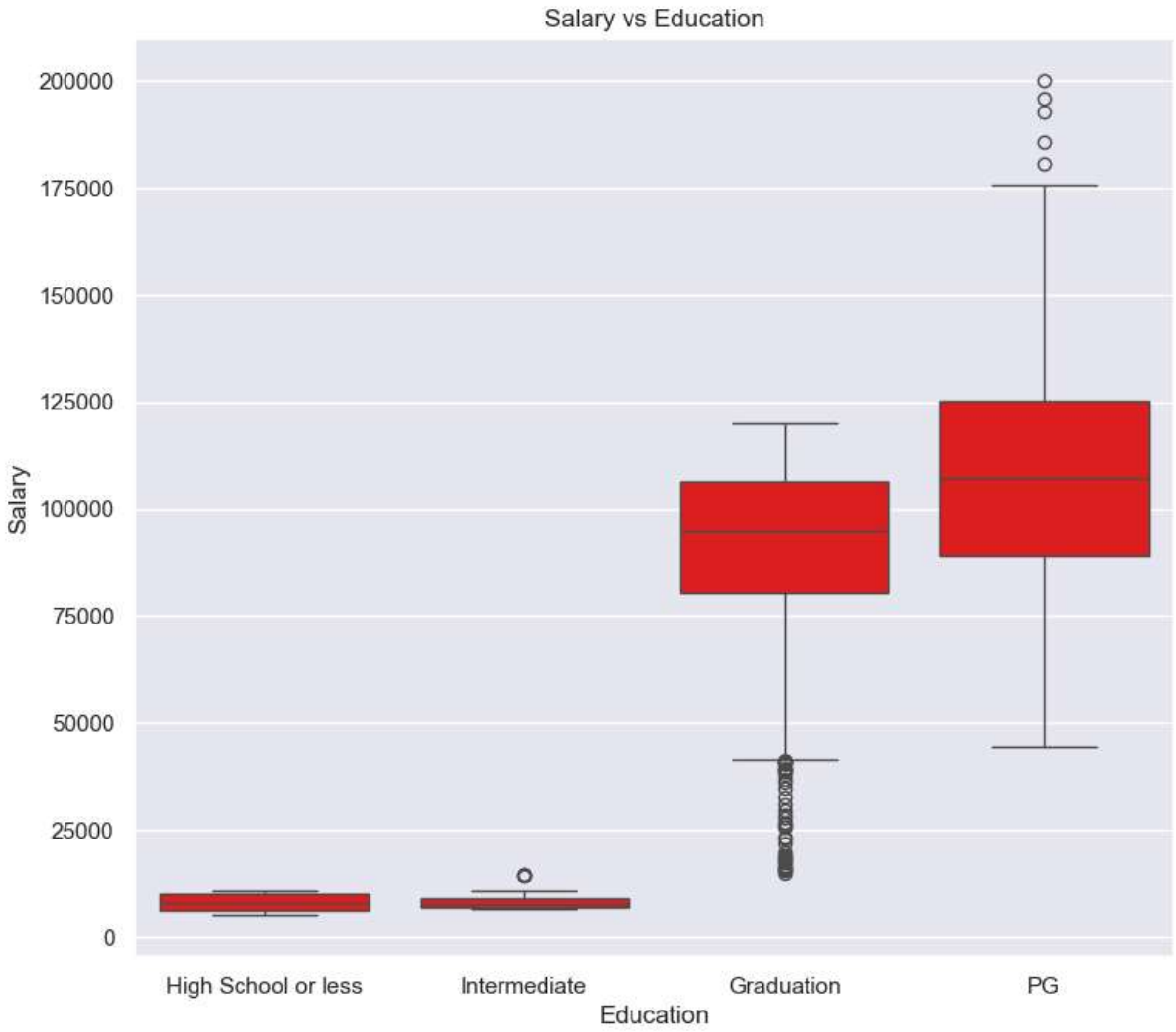
In [36]:
```
plt.figure(figsize=(9,8))
sns.boxplot(data=df,x="Gender", y="Salary",color="Red")
plt.title("Salary vs Gender")
```

Out[36]:
```
Text(0.5, 1.0, 'Salary vs Gender')
```



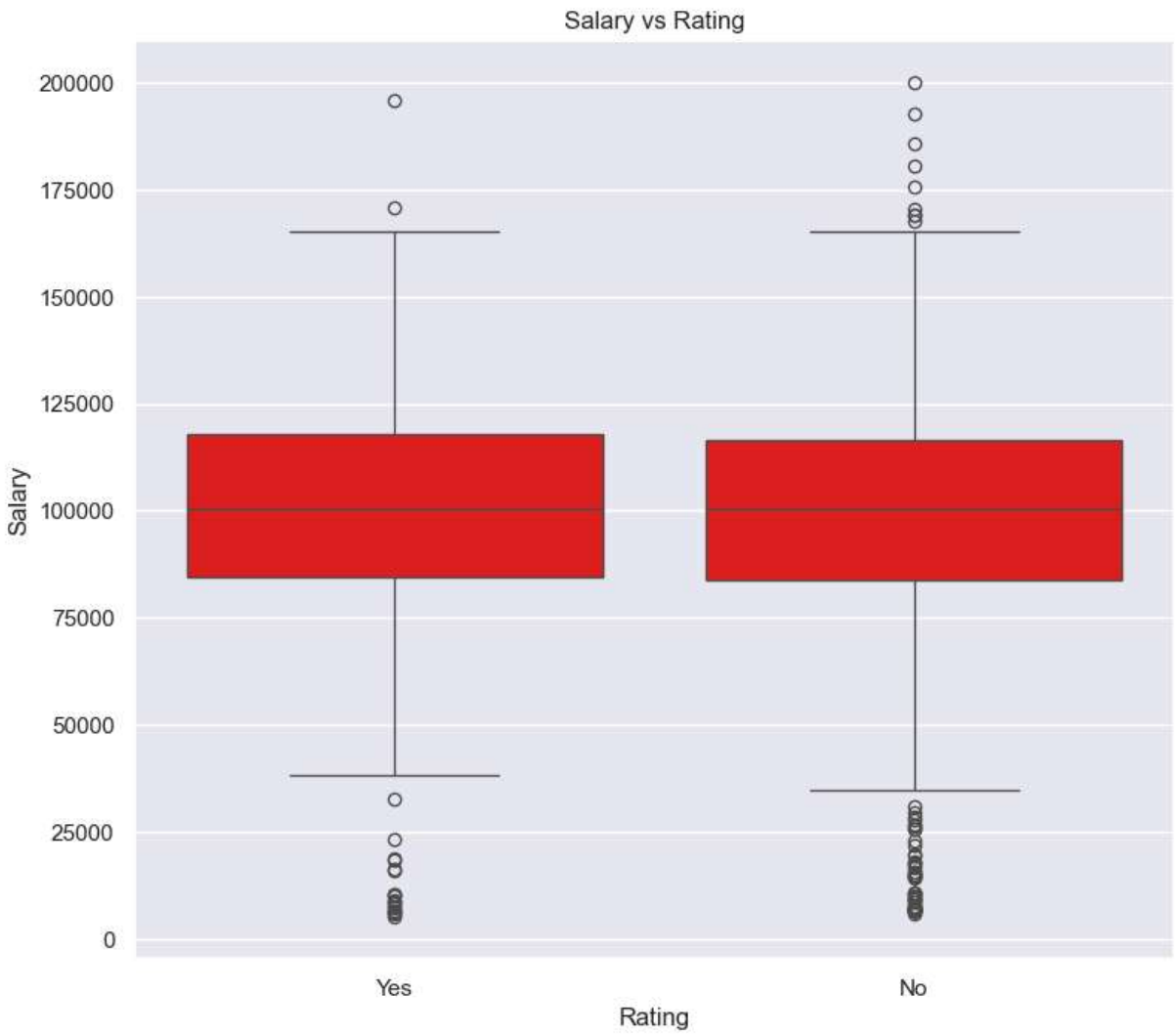
In [37]:
```
plt.figure(figsize=(9,8))
sns.boxplot(data=df,x="Education", y="Salary",color="Red")
plt.title("Salary vs Education")
```

Out[37]:
```
Text(0.5, 1.0, 'Salary vs Education')
```



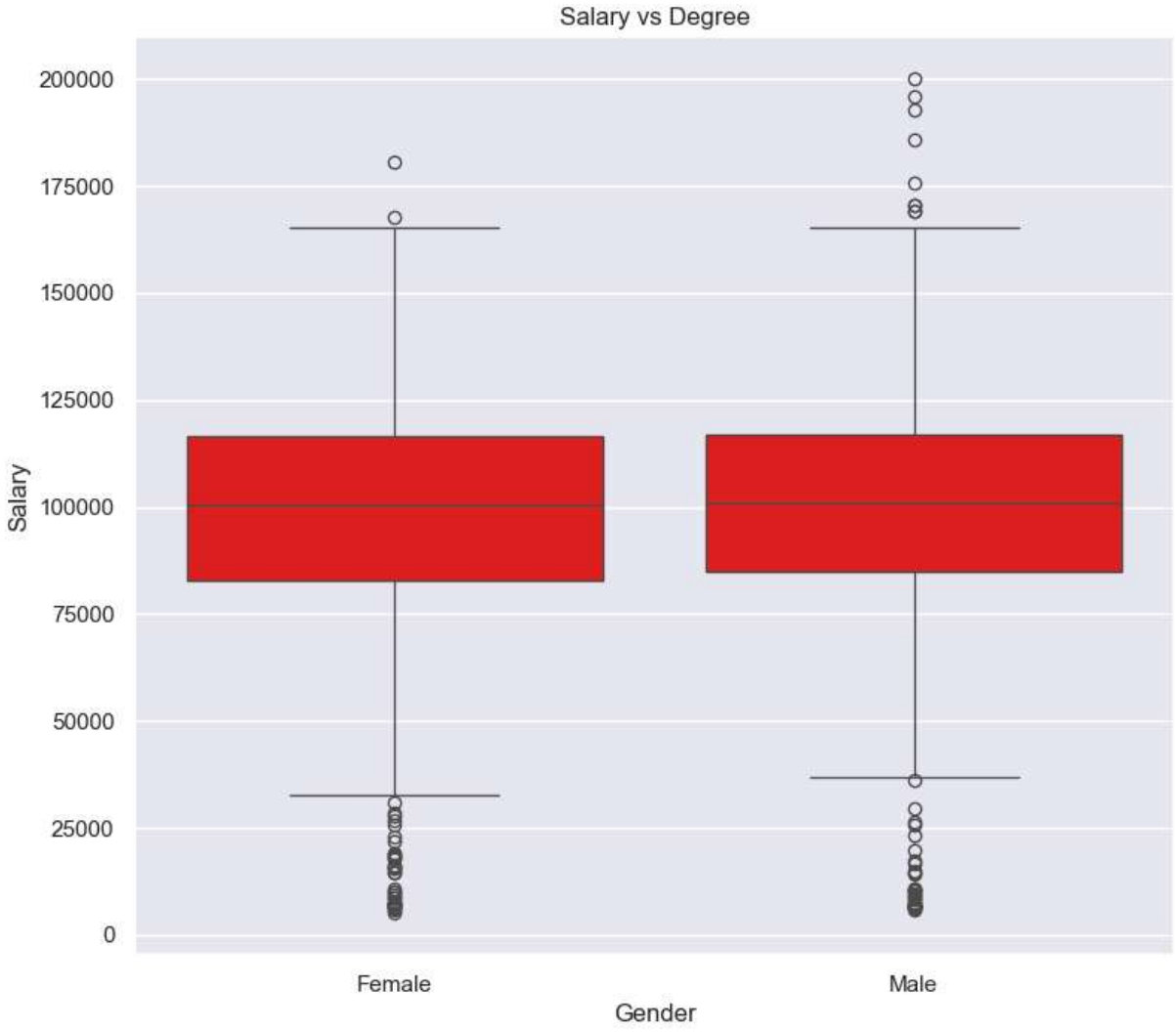
In [38]:
```
plt.figure(figsize=(9,8))
sns.boxplot(data=df,x="Rating", y="Salary",color="Red")
plt.title("Salary vs Rating")
```

Out[38]:
```
Text(0.5, 1.0, 'Salary vs Rating')
```



In [39]:
```
plt.figure(figsize=(9,8))
sns.boxplot(data=df,x="Gender", y="Salary",color="Red")
plt.title("Salary vs Degree")
```

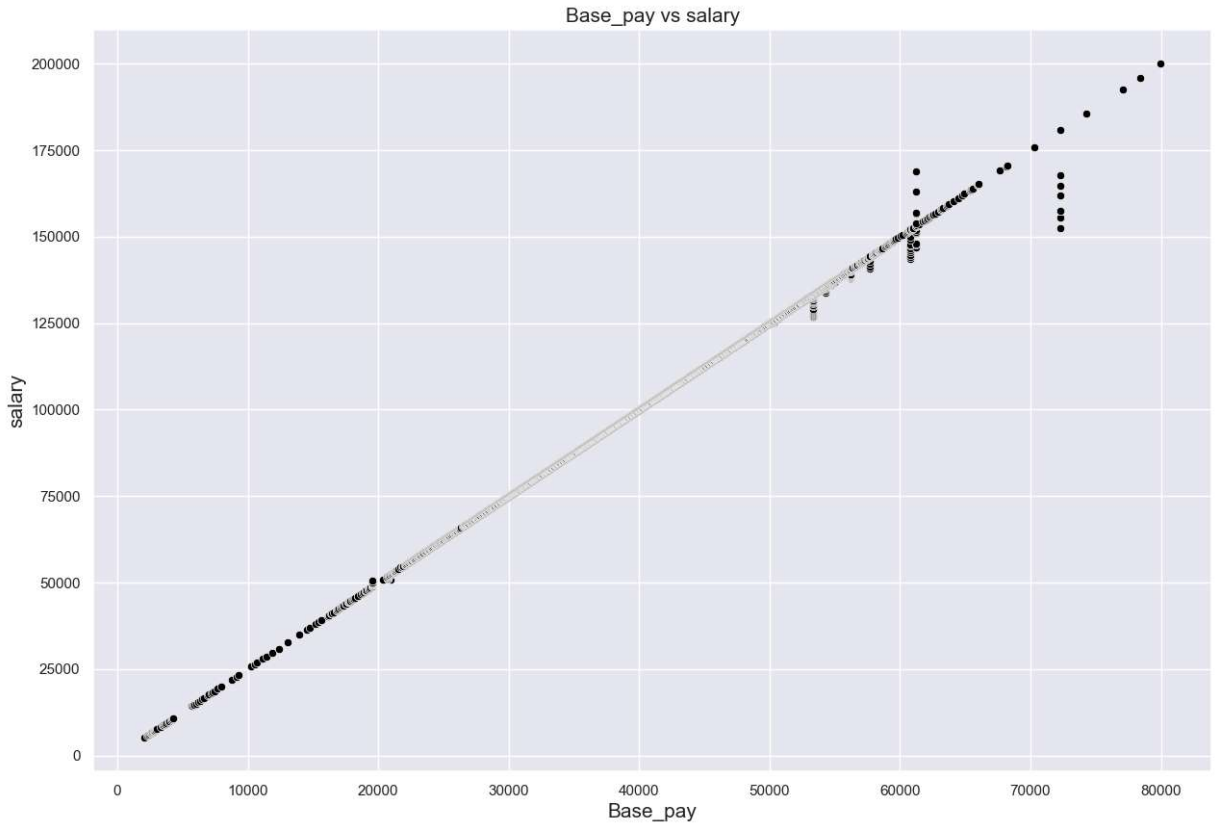
Out[39]:
```
Text(0.5, 1.0, 'Salary vs Degree')
```



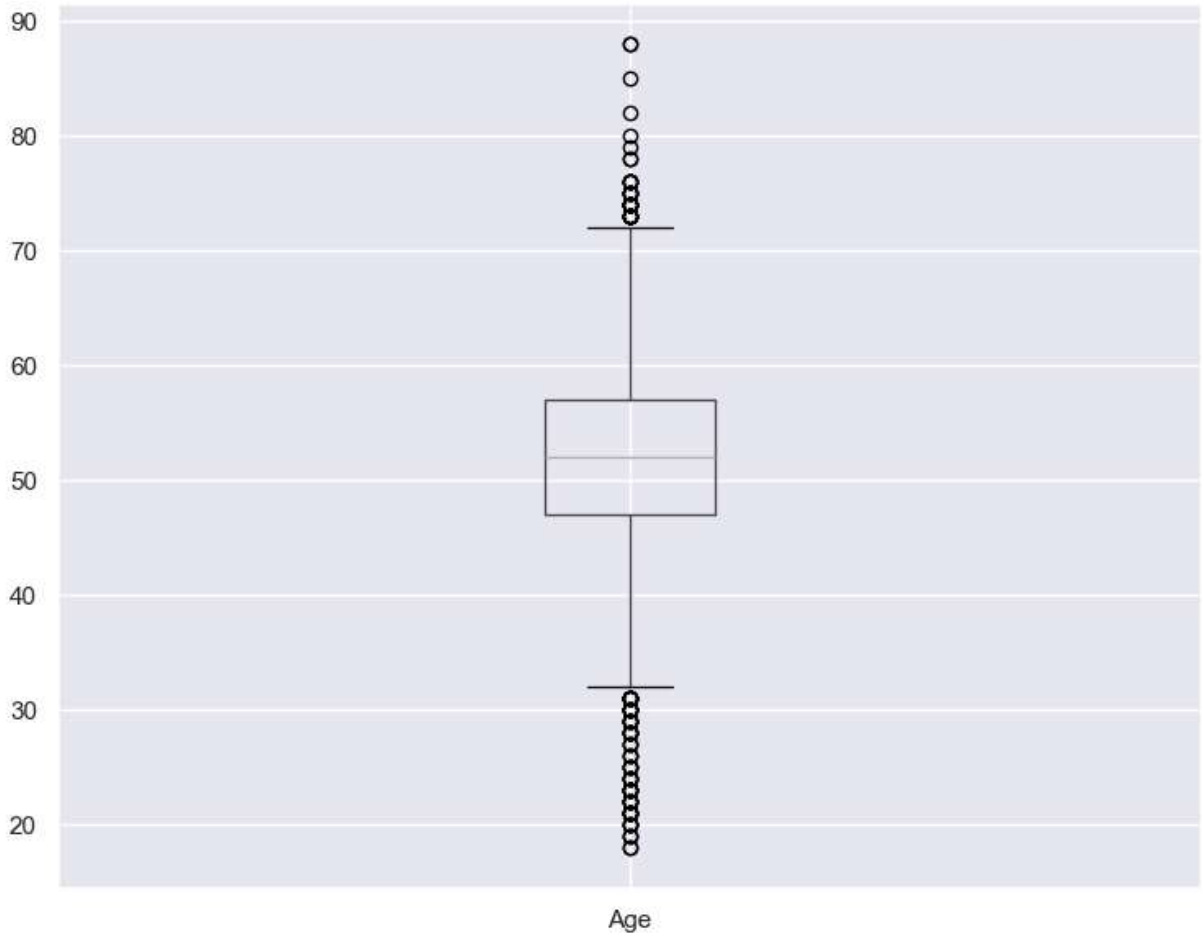
In [40]:
```
df.select_dtypes(include=['number']).columns.tolist()
```

Out[40]:
```
['Business',
 'Age',
 'Salary',
 'Base_pay',
 'Bonus',
 'Unit_Price',
 'Volume',
 'openingbalance',
 'closingbalance',
 'low',
 'Unit_Sales',
 'Total_Sales',
 'Months']
```

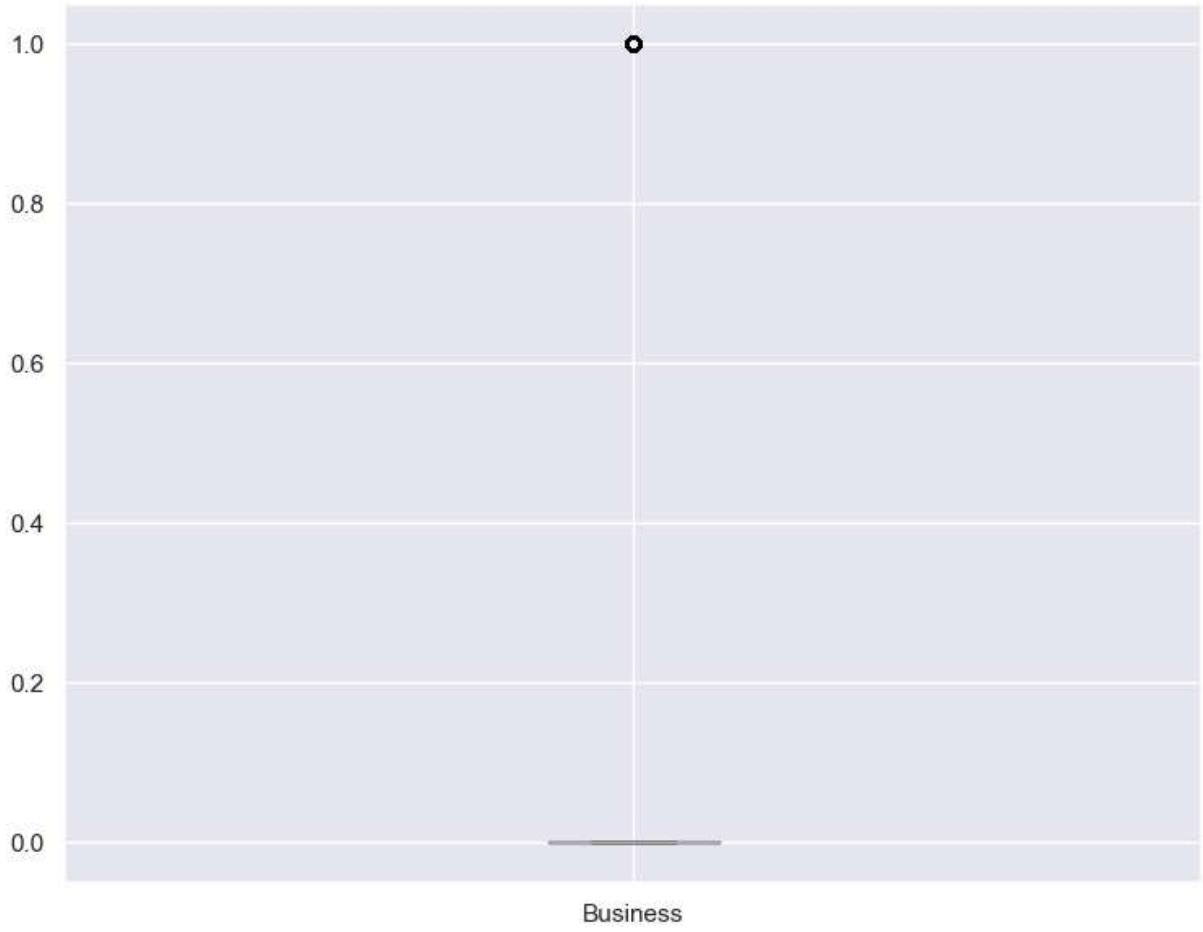
In [41]:
```
plt.figure(figsize=(15,10))
plt.xlabel('Base_pay',fontsize=15)
plt.ylabel('salary',fontsize=15)
plt.title('Base_pay vs salary',fontsize=15)

sns.scatterplot(x=df['Base_pay'],y=df['Salary'],color='black')
plt.show()
```



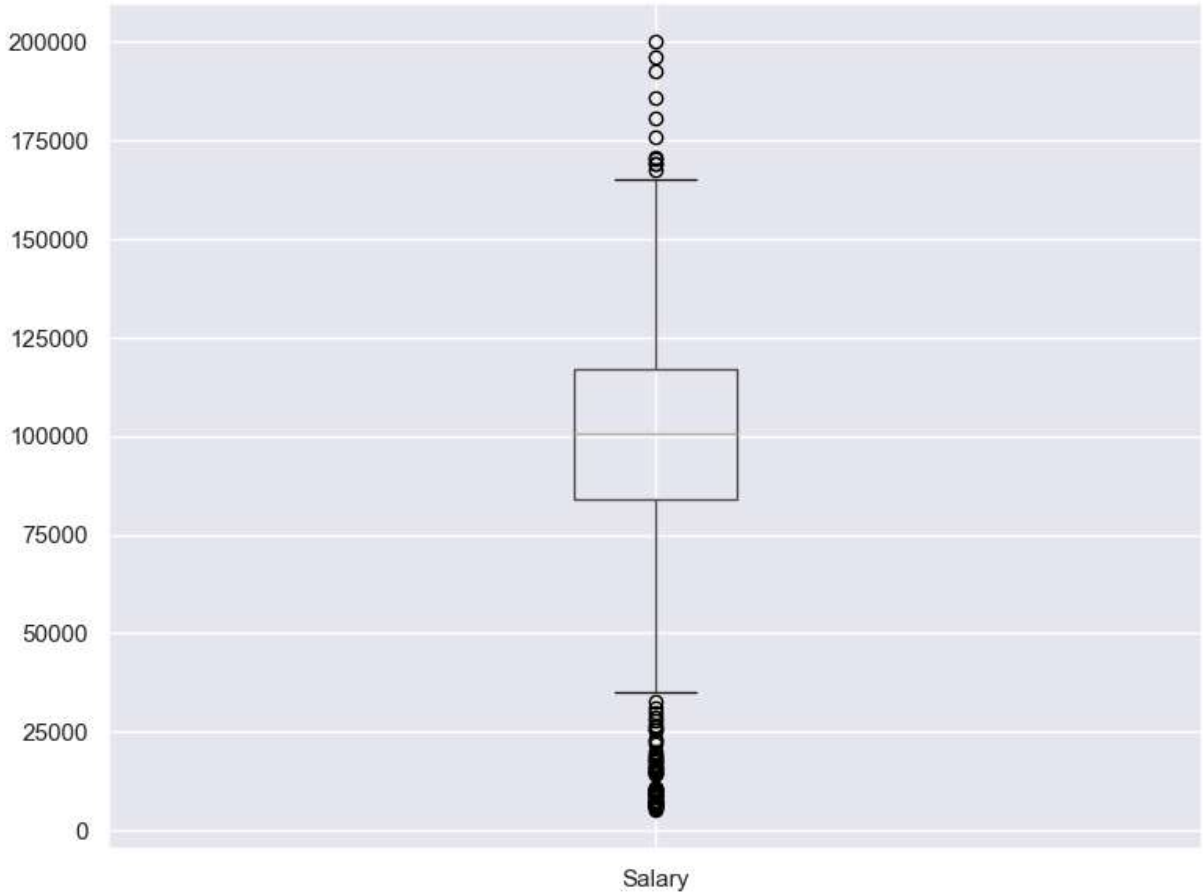
In [42]:
```
plt.figure(figsize=(9,7))
figure=df.boxplot(column='Age')
```



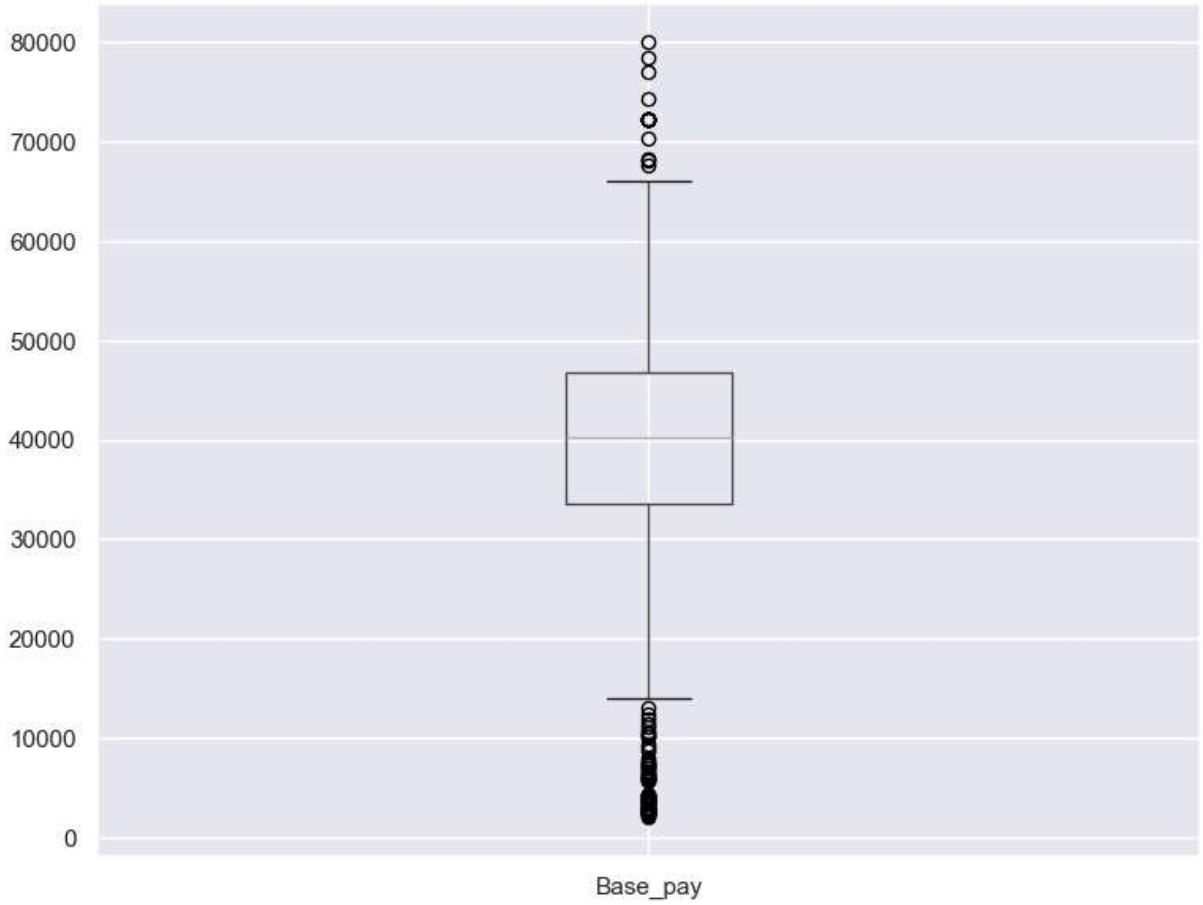
In [43]:
```
plt.figure(figsize=(9,7))
figure=df.boxplot(column='Business')
```



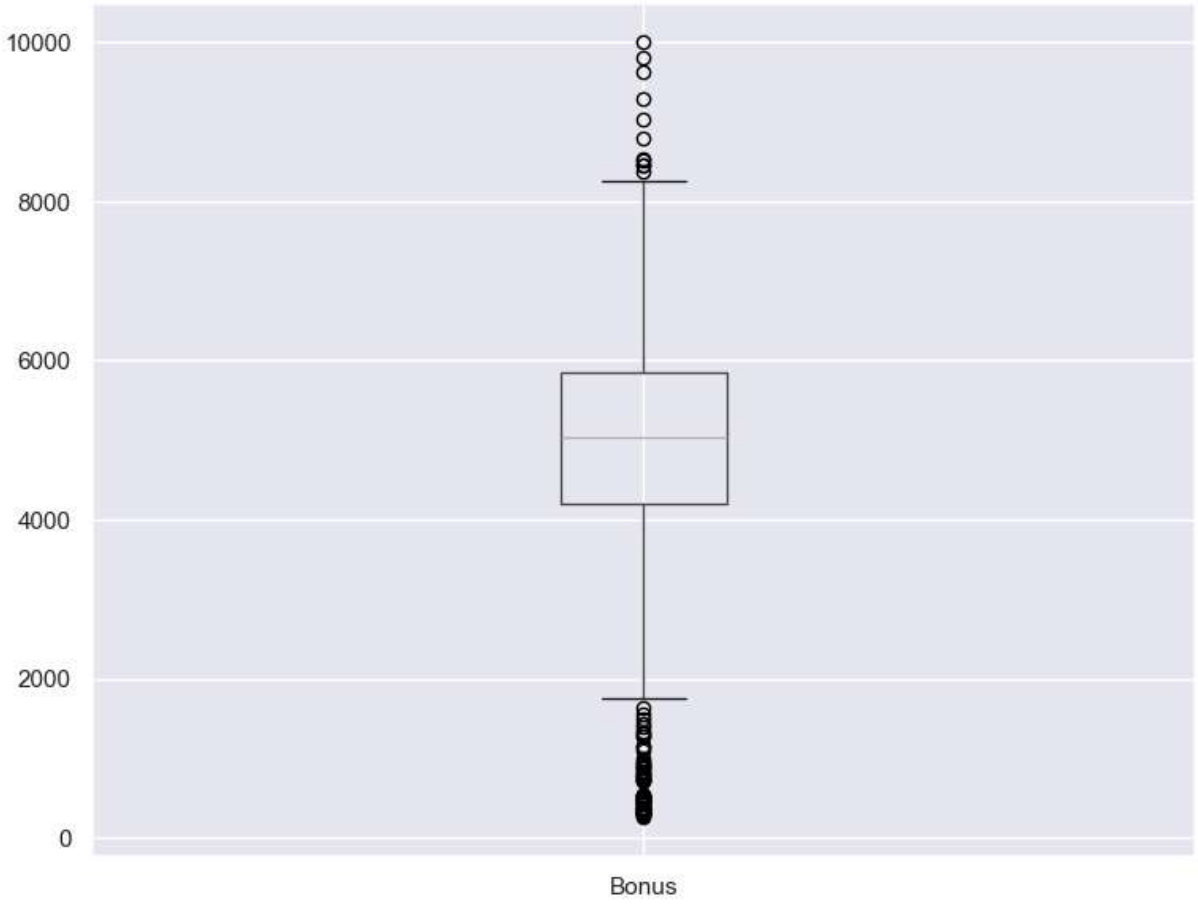
In [44]:
```
plt.figure(figsize=(9,7))
figure=df.boxplot(column='Salary')
```



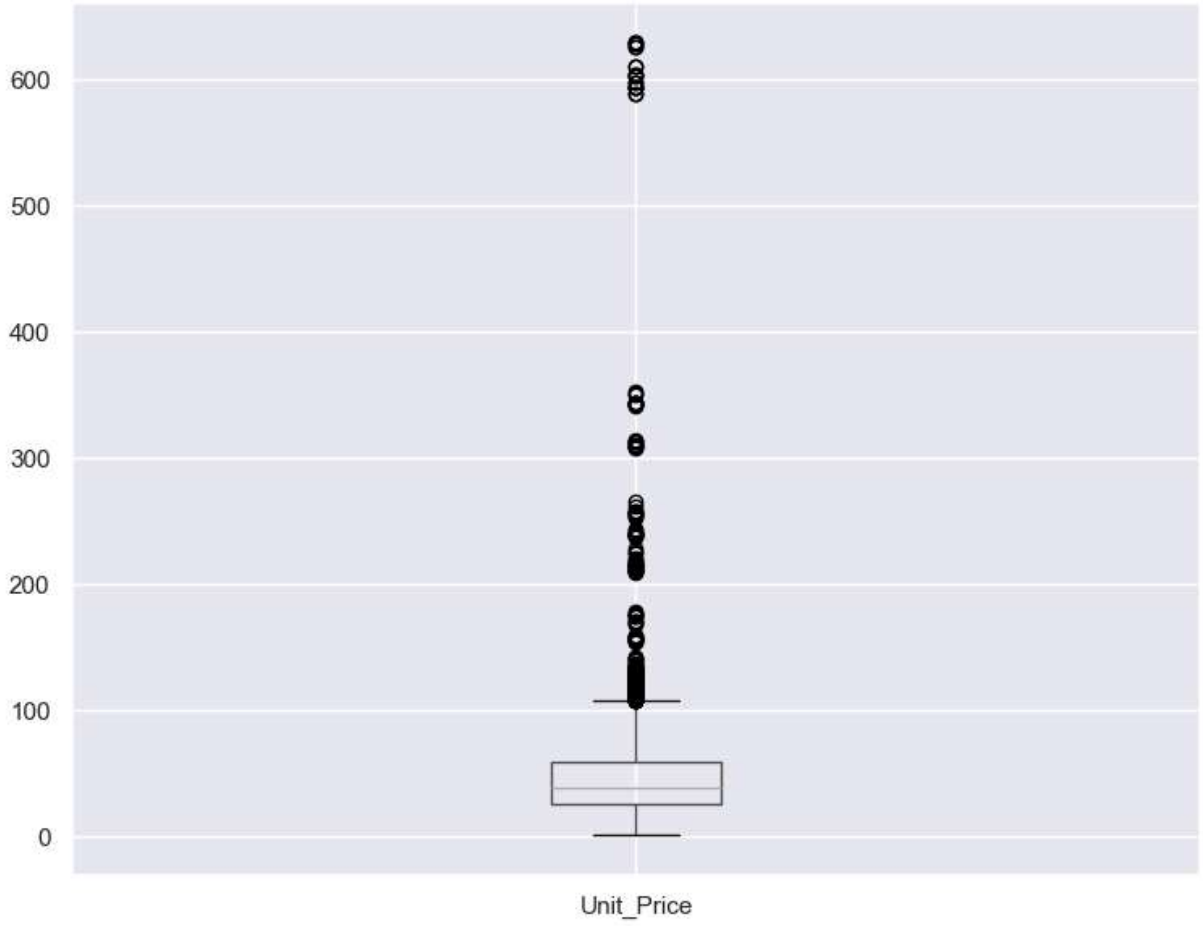
In [45]:
```
plt.figure(figsize=(9,7))
figure=df.boxplot(column='Base_pay')
```



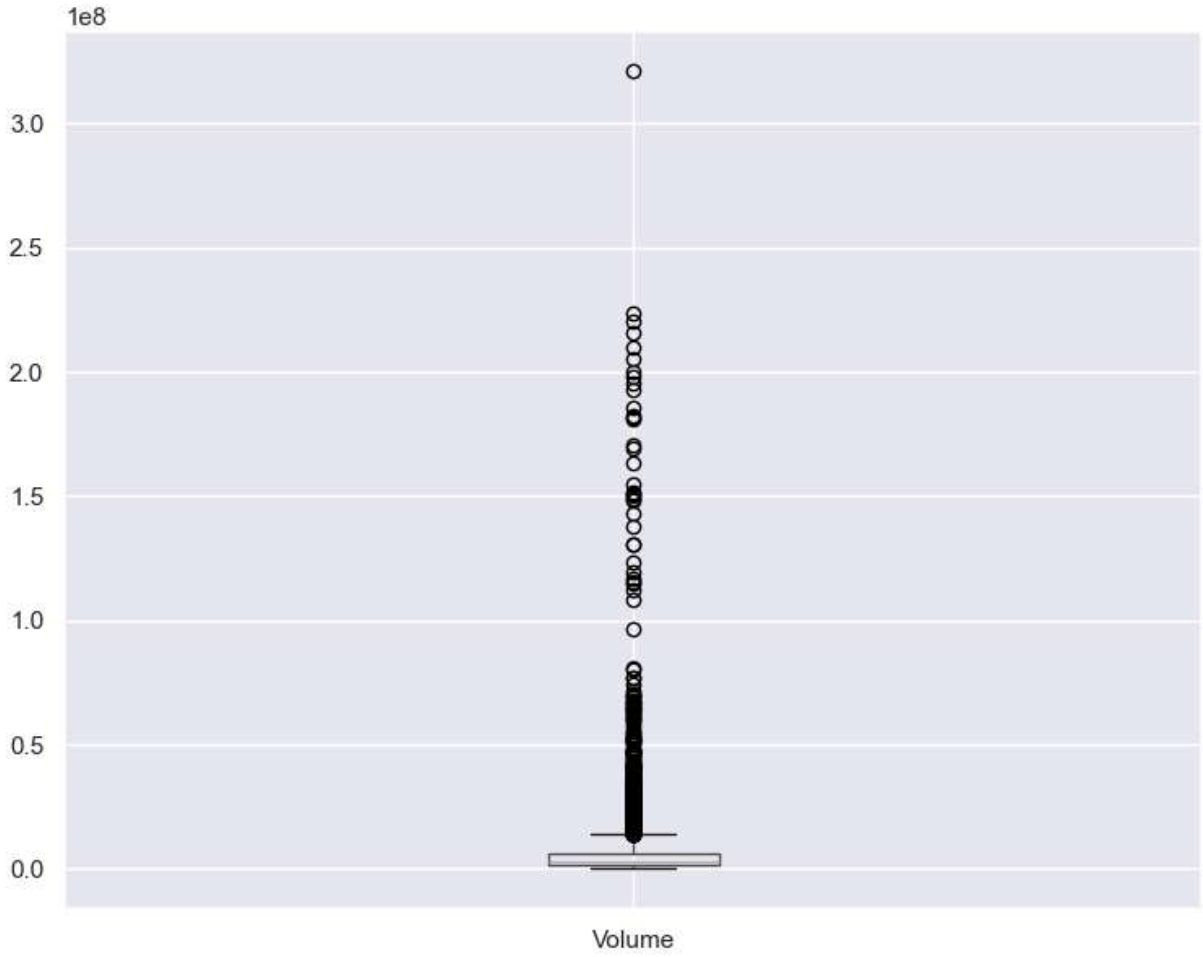
In [46]:
```
plt.figure(figsize=(9,7))
figure=df.boxplot(column='Bonus')
```



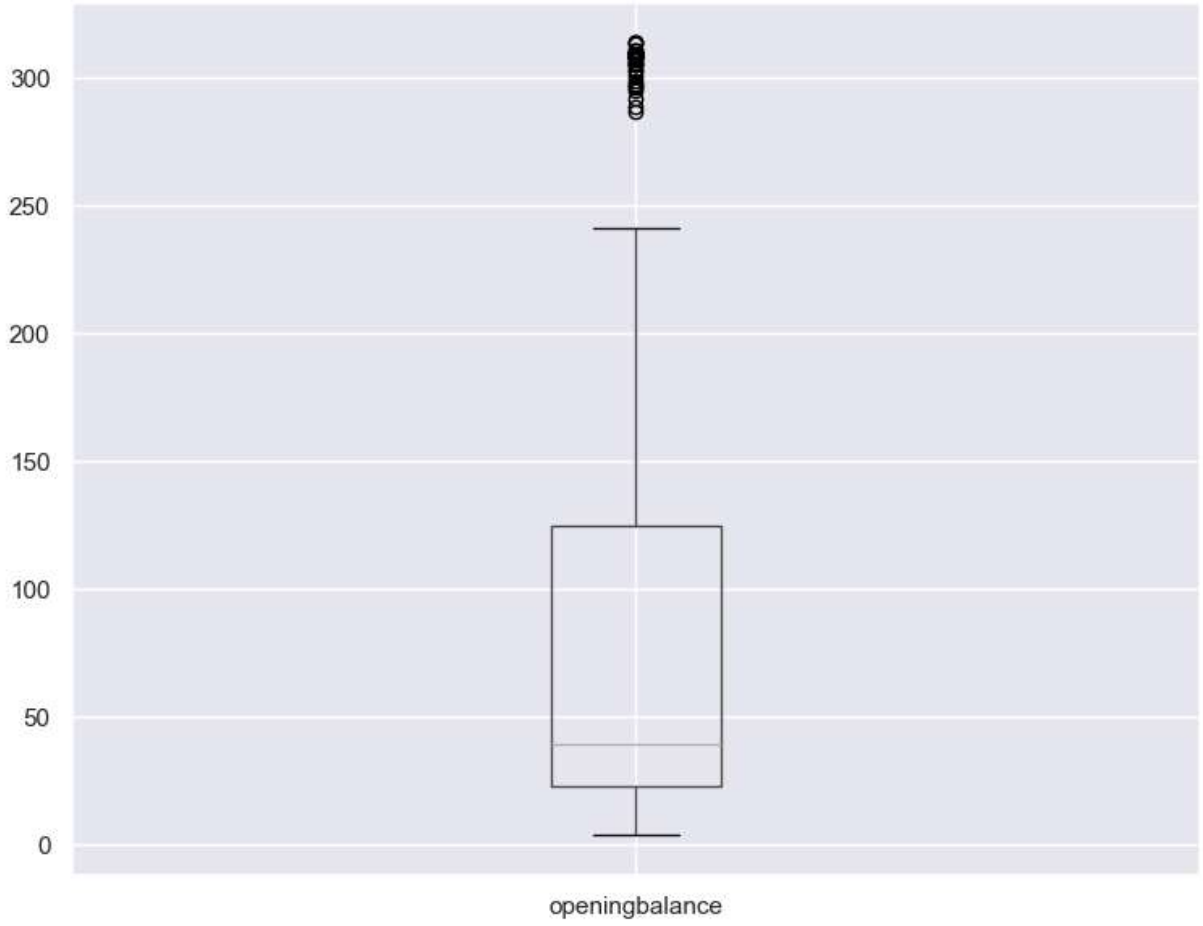
In [47]:
```
plt.figure(figsize=(9,7))
figure=df.boxplot(column='Unit_Price')
```



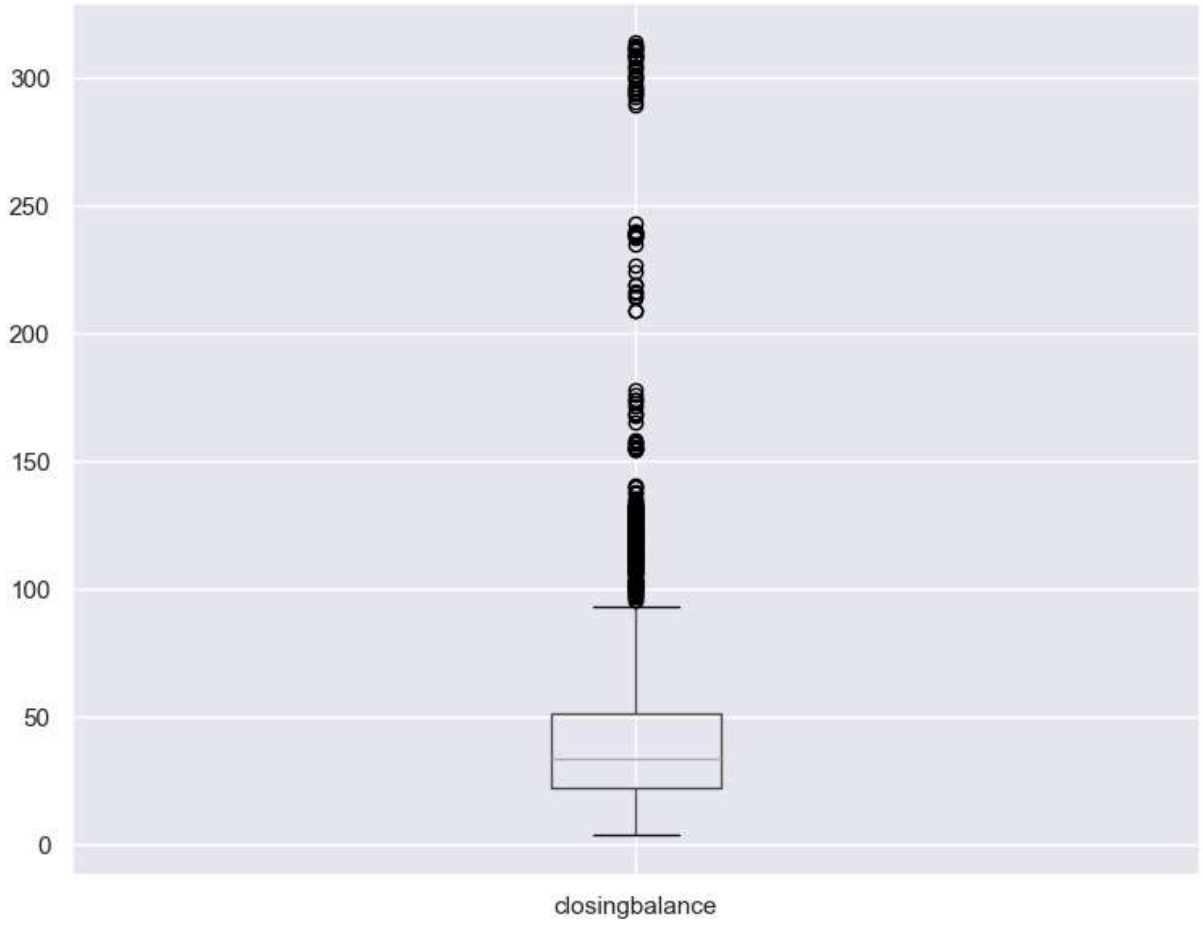
In [48]:
```
plt.figure(figsize=(9,7))
figure=df.boxplot(column='Volume')
```



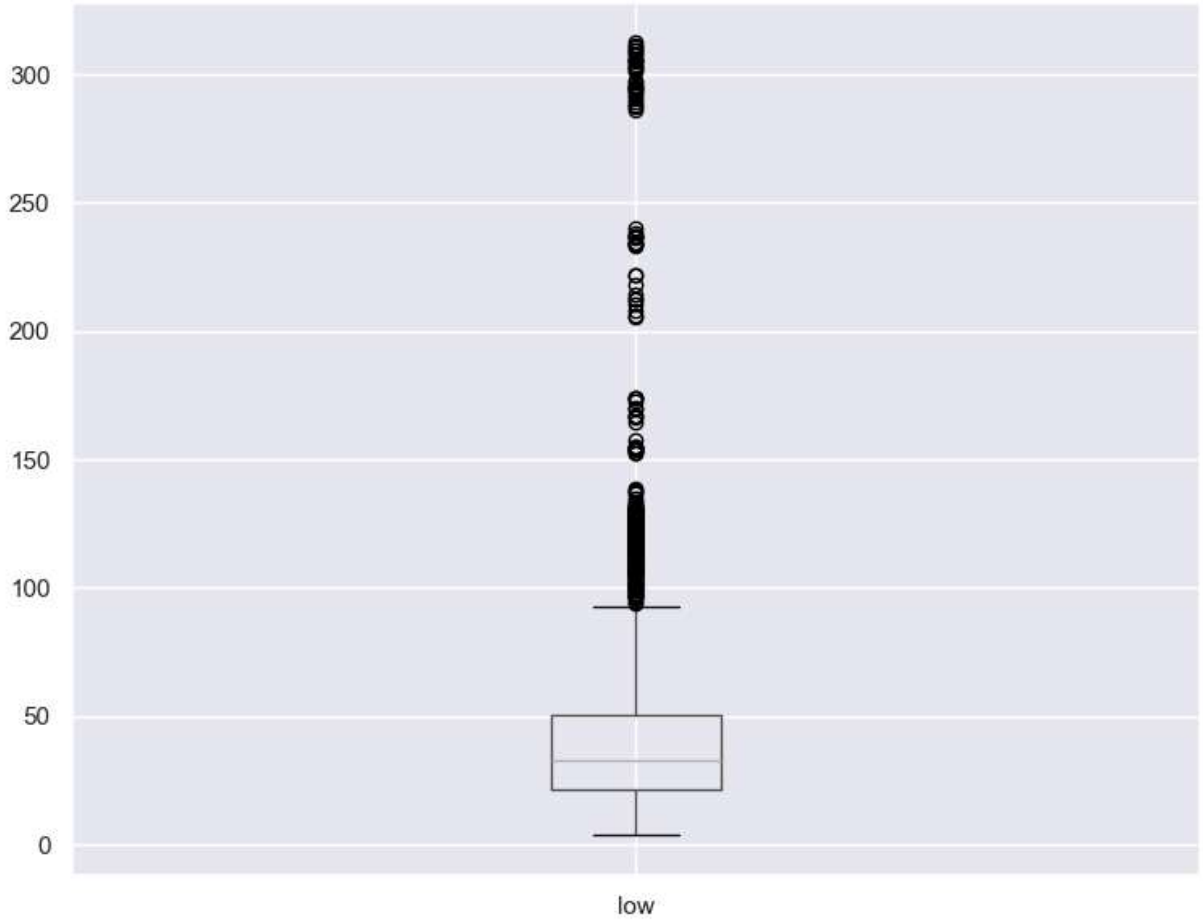
In [49]:
```
plt.figure(figsize=(9,7))
figure=df.boxplot(column='openingbalance')
```



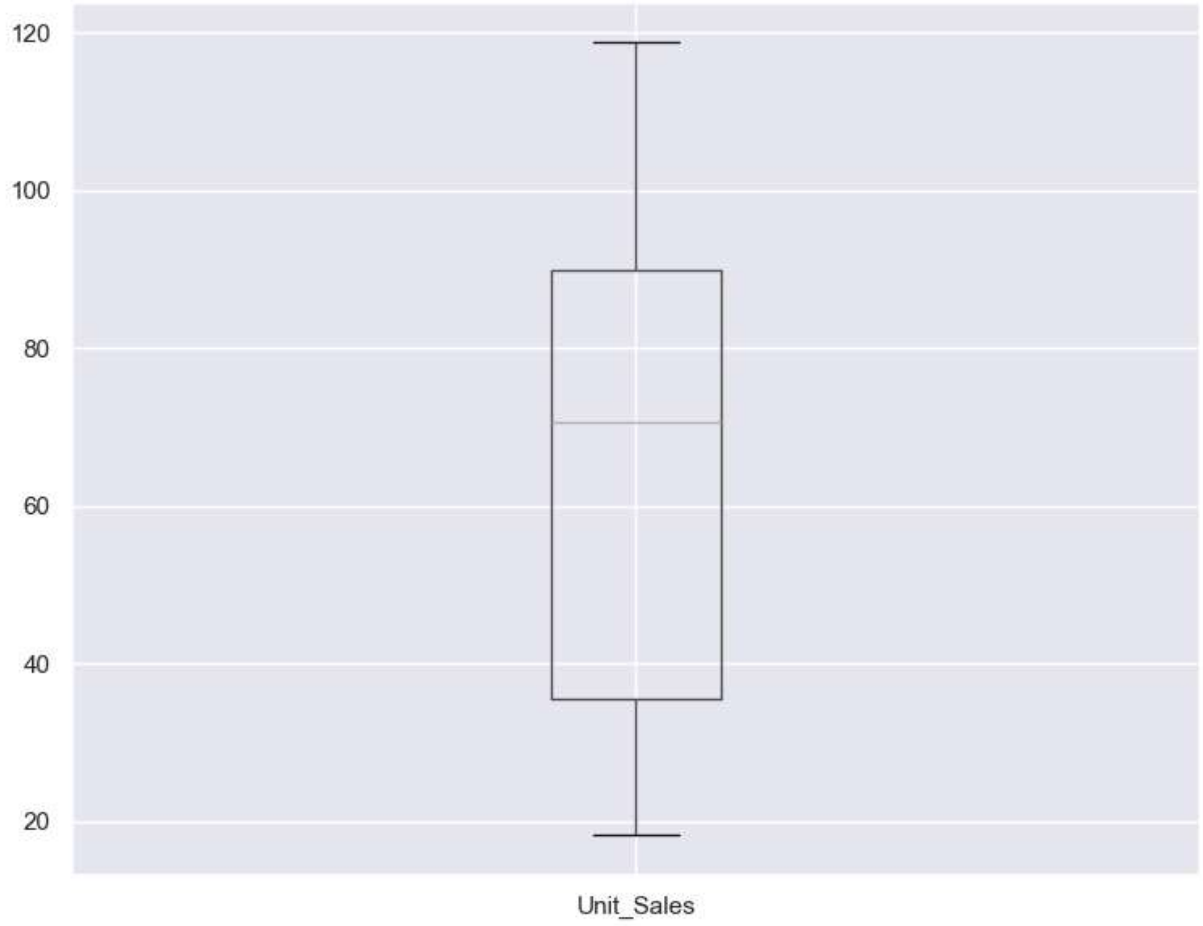
In [50]:
```
plt.figure(figsize=(9,7))
figure=df.boxplot(column='closingbalance')
```



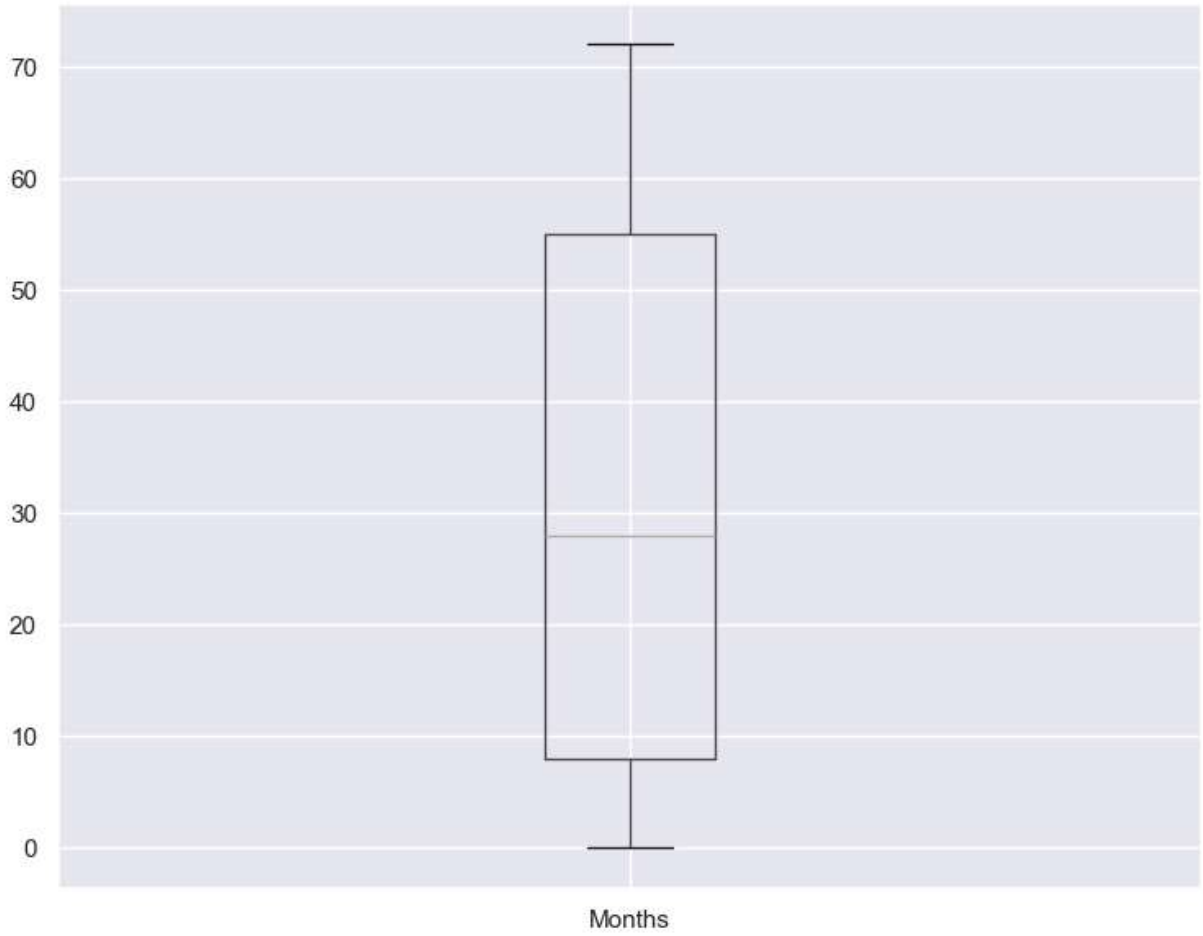
In [51]:
```
plt.figure(figsize=(9,7))
figure=df.boxplot(column='low')
```



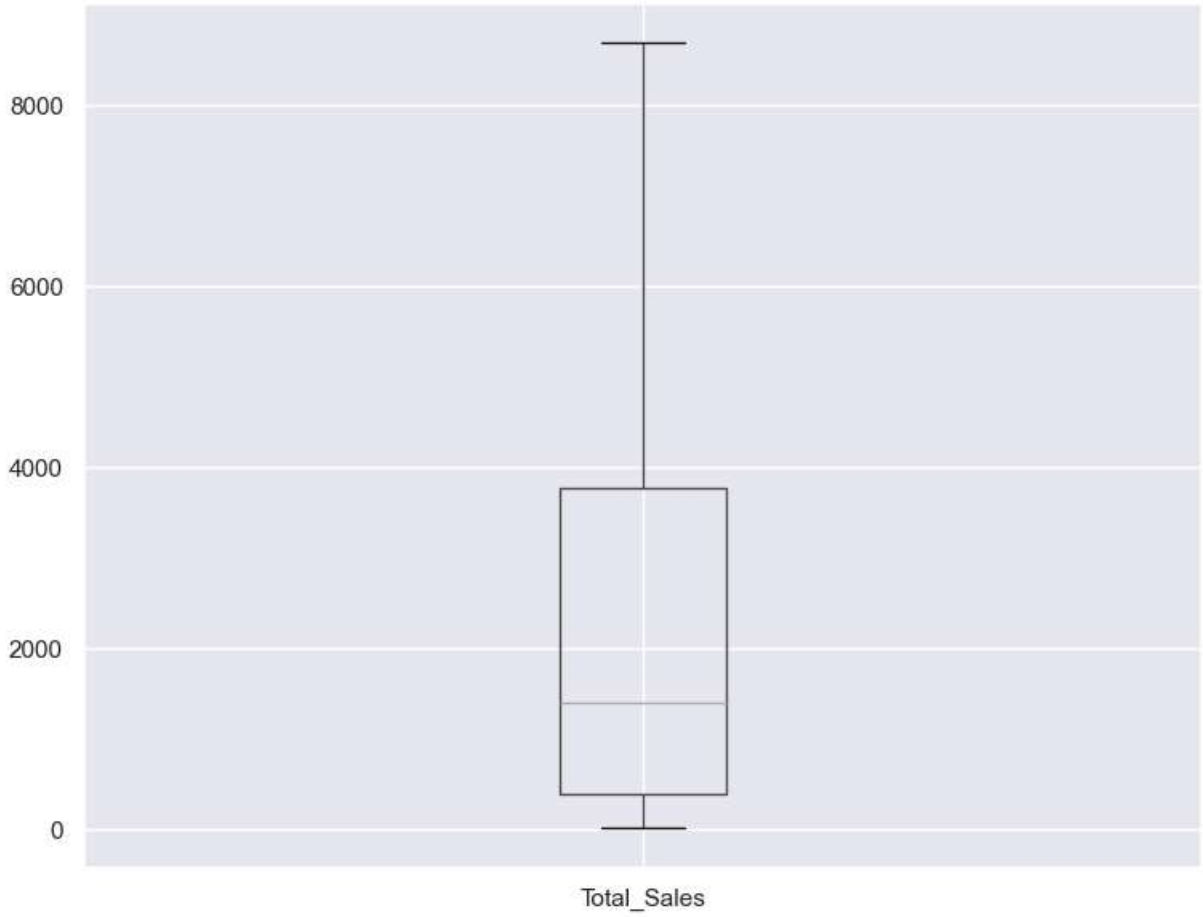
In [52]:
```
plt.figure(figsize=(9,7))
figure=df.boxplot(column='Unit_Sales')
```



In [53]:
```
plt.figure(figsize=(9,7))
figure=df.boxplot(column='Months')
```



In [54]:
```
plt.figure(figsize=(9,7))
figure=df.boxplot(column='Total_Sales')
```

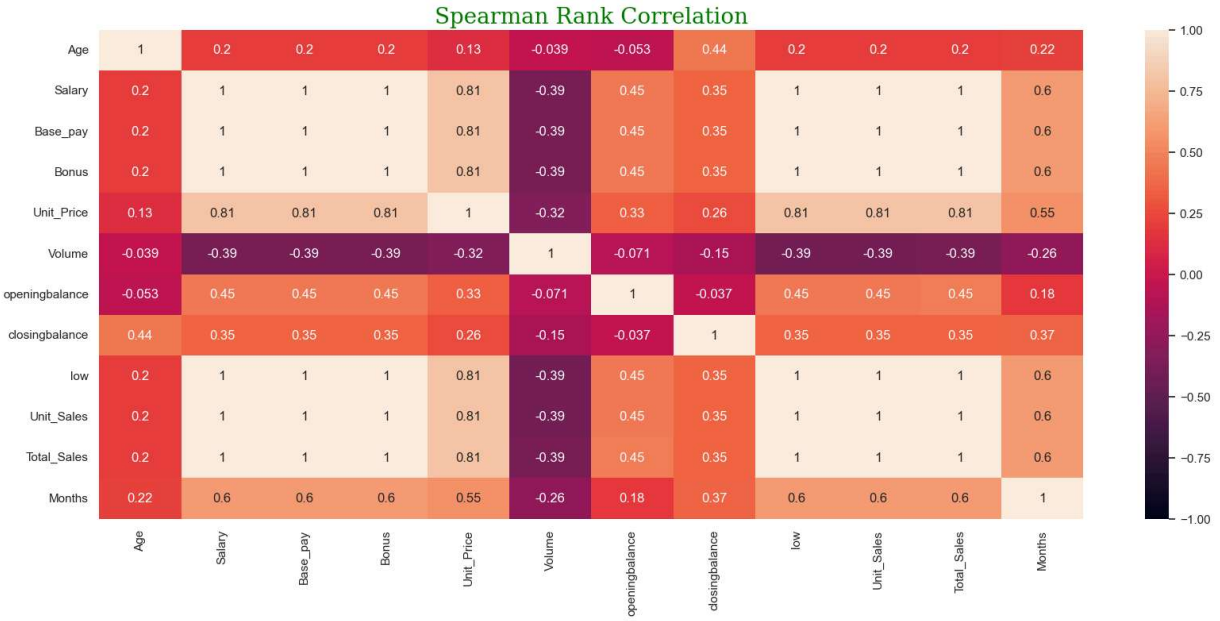


In [55]:
```
df3 = df.drop(['Business'],axis = 1)

df3.head()
```

Out[55]:

	Gender	Dependancies	Calls	Type	Billing	Rating	Age	Salary	Base_pay	Bonus
0	Female	No	Yes	Month-to-month	No	Yes	18	5089.00	2035.600	254.4500
1	Female	No	Yes	Month-to-month	No	Yes	19	5698.12	2279.248	284.9060
2	Male	No	Yes	Month-to-month	Yes	No	22	5896.65	2358.660	294.8325
3	Female	No	Yes	Month-to-month	Yes	Yes	21	6125.12	2450.048	306.2560
4	Male	No	Yes	Month-to-month	Yes	Yes	23	6245.00	2498.000	312.2500

In [56]:
```
df3_numeric = df3.select_dtypes(include='number')
df3_numeric.corr(method="spearman")# selecting the method as a sperman
plt.figure(figsize=(20,8))# setting the figuresize

heatmap = sns.heatmap(df3_numeric.corr(method = 'spearman').round(3), vmin=-1,
vmax=1, annot=True)# annot = True means writting the data value in each cell.

font2= {'family':'serif','color':'green','size':20}
plt.title("Spearman Rank Correlation",font2)
plt.show()# displaying heatmap
```



In [57]:
```
df3_numeric.corr(method='spearman',min_periods=1)
```

Out[57]:

	Age	Salary	Base_pay	Bonus	Unit_Price	Volume	openingb
Age	1.000000	0.202282	0.202121	0.202282	0.128266	-0.039345	-0.
Salary	0.202282	1.000000	0.999976	1.000000	0.811713	-0.390488	0.
Base_pay	0.202121	0.999976	1.000000	0.999976	0.811681	-0.390427	0.
Bonus	0.202282	1.000000	0.999976	1.000000	0.811713	-0.390488	0.
Unit_Price	0.128266	0.811713	0.811681	0.811713	1.000000	-0.324230	0.
Volume	-0.039345	-0.390488	-0.390427	-0.390488	-0.324230	1.000000	-0.
openingbalance	-0.052919	0.453469	0.452851	0.453469	0.333408	-0.071153	1.
closingbalance	0.440989	0.346498	0.346409	0.346498	0.263699	-0.147108	-0.
low	0.202040	0.999859	0.999834	0.999859	0.812412	-0.392209	0.
Unit_Sales	0.202268	0.999997	0.999973	0.999997	0.811735	-0.390505	0.
Total_Sales	0.202173	0.999965	0.999941	0.999965	0.811705	-0.390480	0.
Months	0.222837	0.603796	0.603766	0.603796	0.550062	-0.262077	0.

In [58]:
```
# Looking to the percentage of correlation
df3_numeric = df3.select_dtypes(include='number')
df3_numeric.corr(method='spearman')*100
```

Out[58]:

	Age	Salary	Base_pay	Bonus	Unit_Price	Volume	o
Age	100.000000	20.228180	20.212073	20.228180	12.826637	-3.934477	
Salary	20.228180	100.000000	99.997589	100.000000	81.171311	-39.048779	
Base_pay	20.212073	99.997589	100.000000	99.997589	81.168113	-39.042656	
Bonus	20.228180	100.000000	99.997589	100.000000	81.171311	-39.048779	
Unit_Price	12.826637	81.171311	81.168113	81.171311	100.000000	-32.422983	
Volume	-3.934477	-39.048779	-39.042656	-39.048779	-32.422983	100.000000	
openingbalance	-5.291931	45.346909	45.285139	45.346909	33.340773	-7.115316	
closingbalance	44.098882	34.649833	34.640860	34.649833	26.369923	-14.710785	
low	20.203950	99.985870	99.983405	99.985870	81.241207	-39.220924	
Unit_Sales	20.226808	99.999741	99.997324	99.999741	81.173482	-39.050546	
Total_Sales	20.217334	99.996520	99.994102	99.996520	81.170467	-39.048003	
Months	22.283665	60.379576	60.376626	60.379576	55.006199	-26.207721	

In [59]:
```
# Dropping the non-required variables
dff = df.drop(columns=['Gender','Business','Dependancies','Calls','Type','Billing',
                       'Rating','Base_pay','Unit_Price','low','Unit_Sales',
                       'Total_Sales','openingbalance','closingbalance','Volume'])
dff.head()
```

Out[59]:

	Age	Salary	Bonus	Months	Education
0	18	5089.00	254.4500	0	High School or less
1	19	5698.12	284.9060	0	High School or less
2	22	5896.65	294.8325	0	High School or less
3	21	6125.12	306.2560	0	High School or less
4	23	6245.00	312.2500	1	High School or less

In [60]:
```
dff['Education']=dff['Education'].map({'High School or less': 0,
                                       'Intermediate': 1,
                                       'Graduation': 2, 'PG': 3})
#assigning high school =0,Intermediate =1 and gradaution=2
```

In [61]:
```
dff.head()
```

Out[61]:

	Age	Salary	Bonus	Months	Education
0	18	5089.00	254.4500	0	0
1	19	5698.12	284.9060	0	0
2	22	5896.65	294.8325	0	0
3	21	6125.12	306.2560	0	0
4	23	6245.00	312.2500	1	0

In [62]:
```
dff.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Age        5000 non-null   int64  
 1   Salary     5000 non-null   float64
 2   Bonus      5000 non-null   float64
 3   Months     5000 non-null   int64  
 4   Education  5000 non-null   int64  
dtypes: float64(2), int64(3)
memory usage: 195.4 KB
```

In [63]:
```
Q1=dff.quantile(0.25).round(3)
Q3=dff.quantile(0.75).round(3)
```

```
IQR = Q3 - Q1
print(IQR)
```

```
Age             10.000
Salary       33021.753
Bonus         1651.088
Months          47.000
Education        1.000
dtype: float64
```

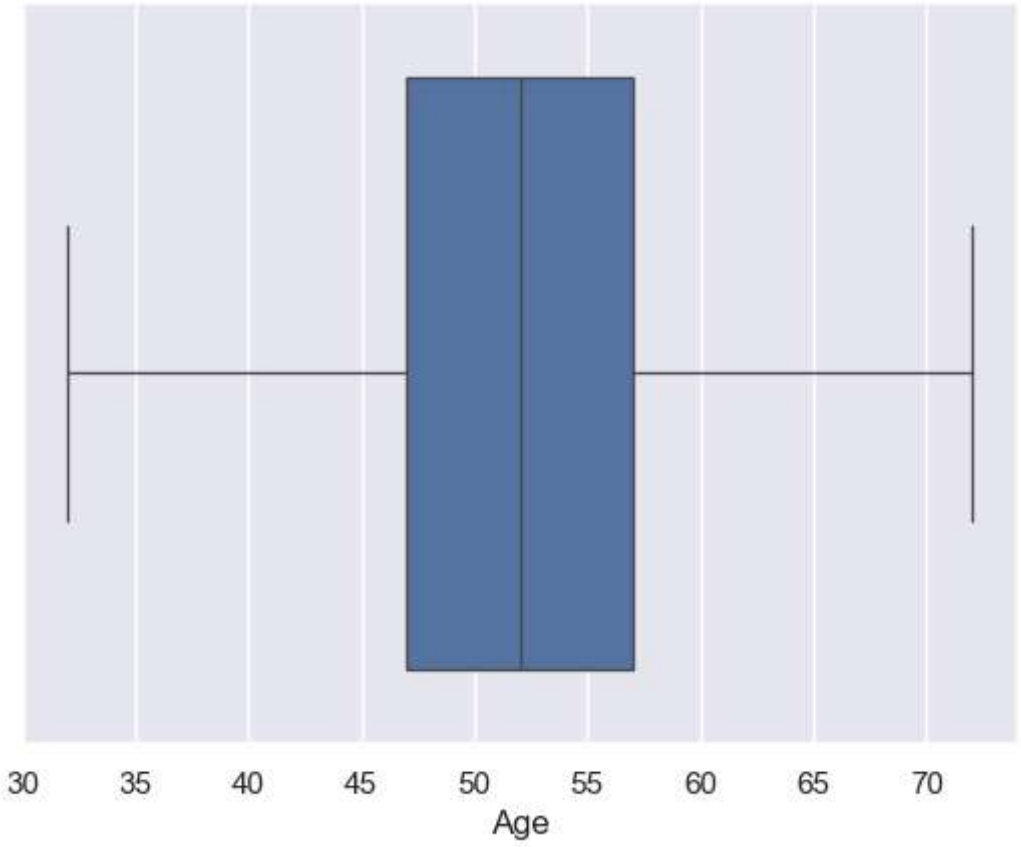
In [64]:
```
dfout = dff[~((dff < (Q1 - 1.5 * IQR)) |(dff > (Q3 + 1.5 * IQR))).any(axis=1)]
dfout.shape
```

Out[64]: (4833, 5)

In [65]:
```
#SANITY CHECK WHETHER OUTLIERS ARE REMOVED OR NOT
```

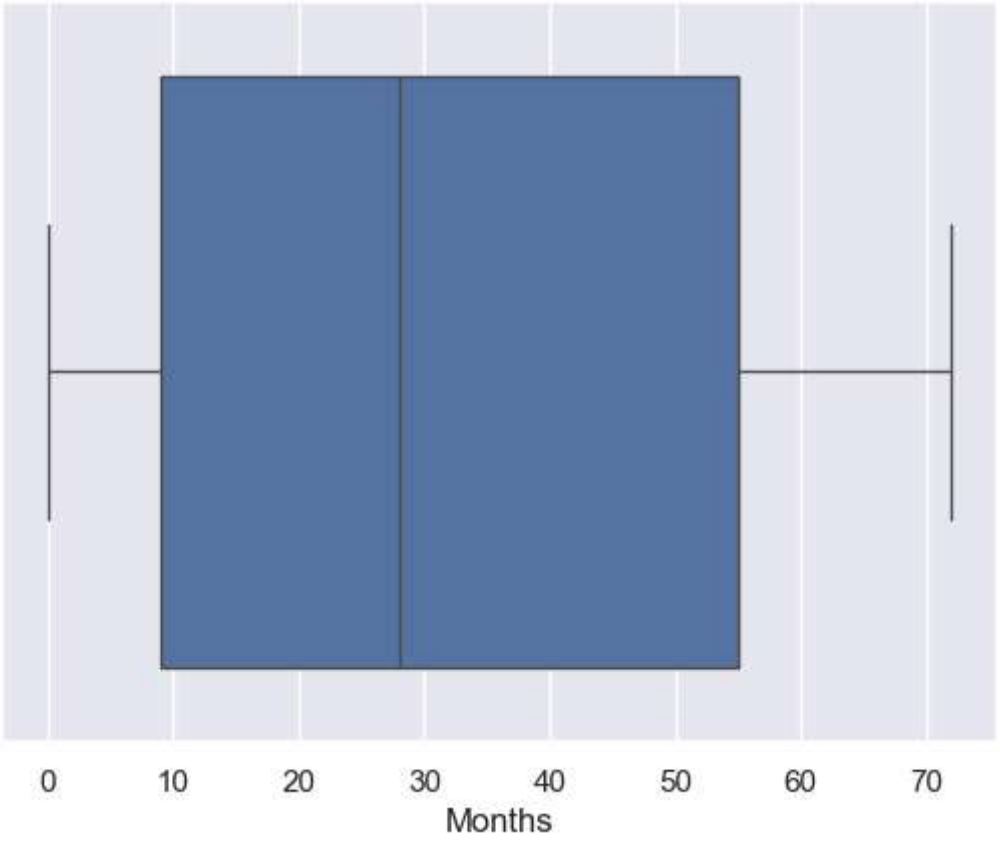
In [66]:
```
sns.boxplot(x=dfout['Age'])
```

Out[66]: <Axes: xlabel='Age'>



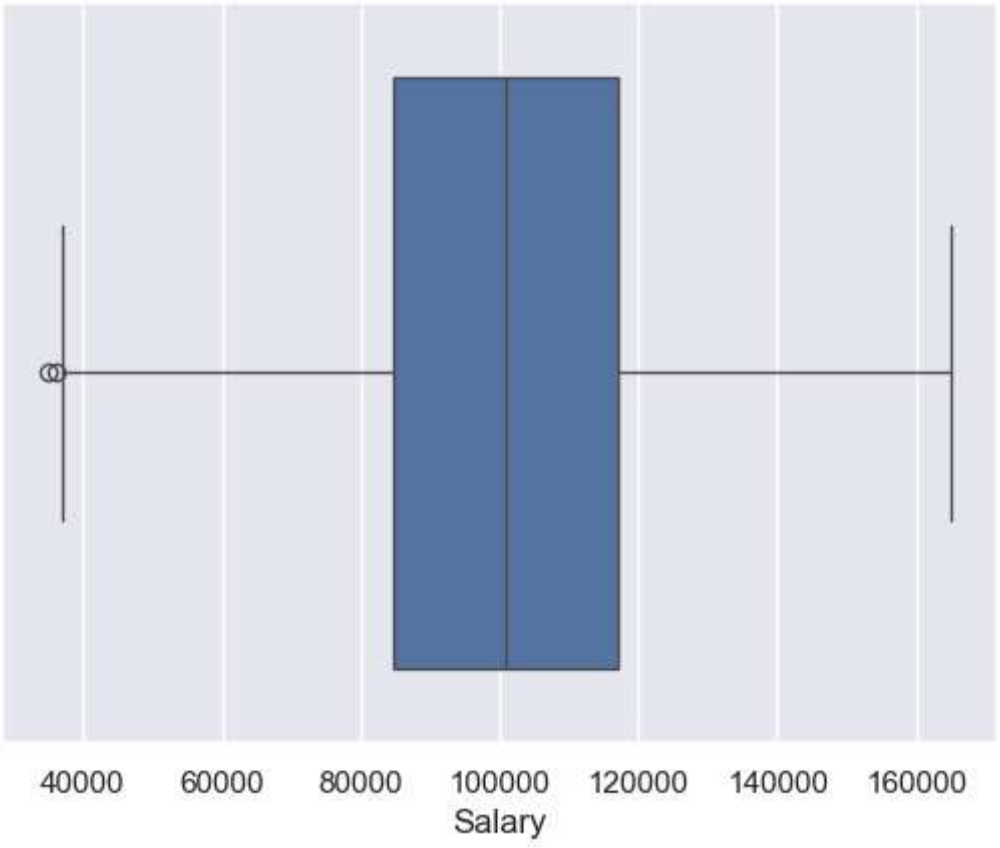
In [67]:
```
sns.boxplot(x=dfout['Months'])
```

Out[67]: <Axes: xlabel='Months'>



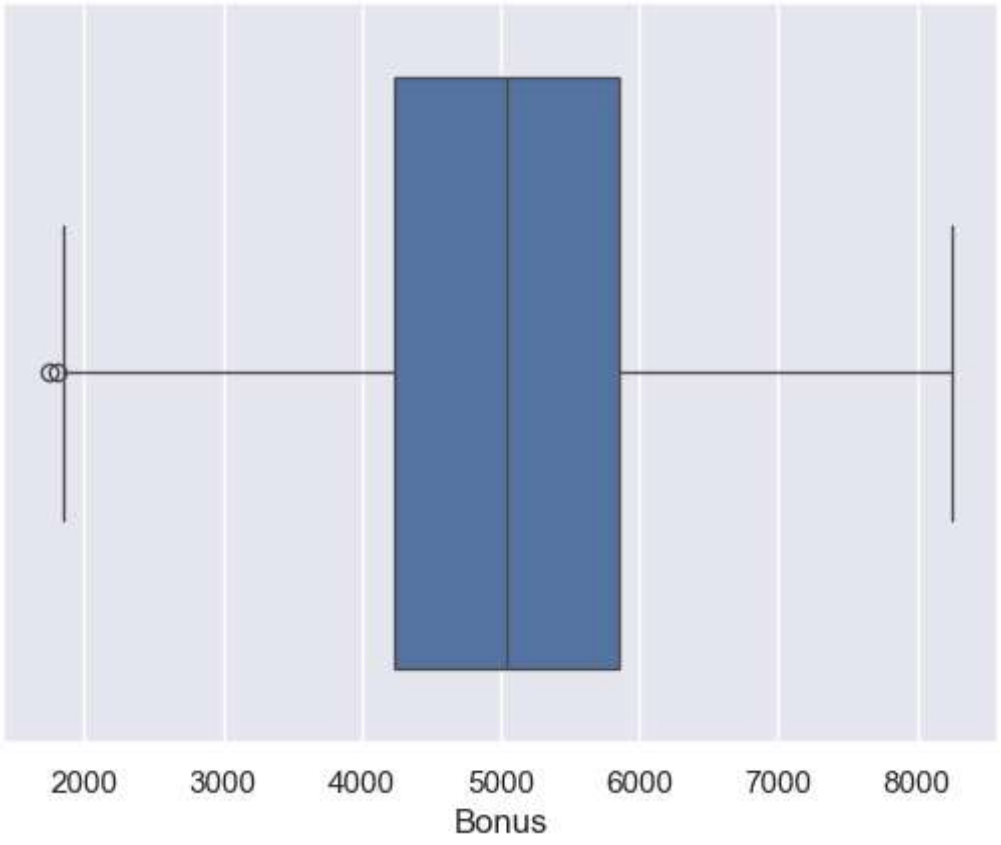
In [68]:
```
sns.boxplot(x=dfout['Salary'])
```

Out[68]: <Axes: xlabel='Salary'>



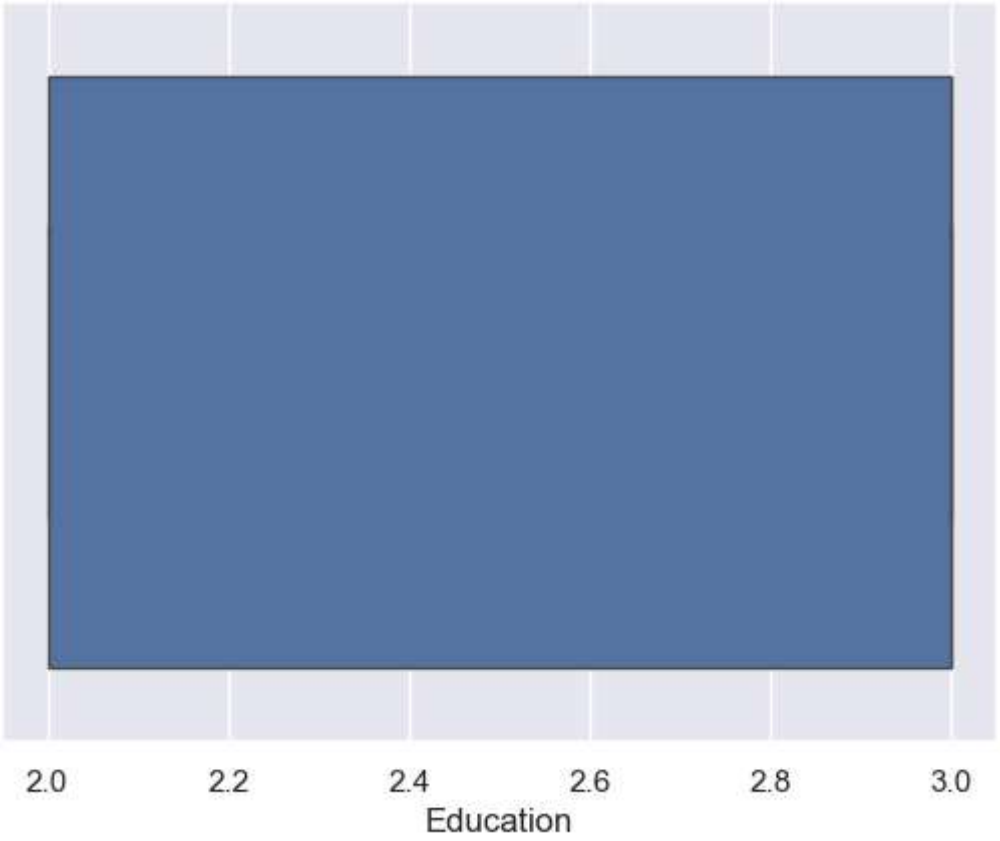
In [69]:
```
sns.boxplot(x=dfout['Bonus'])
```

Out[69]: <Axes: xlabel='Bonus'>



In [70]:
```
sns.boxplot(x=dfout['Education'])
```

Out[70]: <Axes: xlabel='Education'>



In [71]:
```
#MODEL BUILDING
```

In [72]:
```
x =dff.drop(['Salary'],axis=1)
x
```

Out[72]:

	Age	Bonus	Months	Education
0	18	254.4500	0	0
1	19	284.9060	0	0
2	22	294.8325	0	0
3	21	306.2560	0	0
4	23	312.2500	1	0
...	...	...	...	...
4995	72	9034.8400	72	3
4996	73	9284.2950	72	3
4997	74	9631.8400	72	3
4998	74	9798.5350	72	3
4999	88	9998.5370	72	3

5000 rows × 4 columns

In [73]:
```
y =dff['Salary']
y
```

Out[73]:
```
0         5089.00
1         5698.12
2         5896.65
3         6125.12
4         6245.00
          ...
4995    180696.80
4996    185685.90
4997    192636.80
4998    195970.70
4999    199970.74
Name: Salary, Length: 5000, dtype: float64
```

In [74]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y,test_size = 0.3,
                                                    random_state=0)
```

In [75]:
```
x_train
```

Out[75]:

	Age	Bonus	Months	Education
2858	45	5246.941220	21	2
1559	47	4441.794503	1	3
1441	45	4345.164070	1	3
2179	55	4842.207785	7	3
1390	44	4304.827917	1	2
...	...	...	...	...
4931	63	7594.017945	72	3
3264	40	5512.225850	31	3
1653	48	4502.199387	2	2
2607	62	5099.289970	15	2
2732	68	5168.475250	18	2

3500 rows × 4 columns

In [76]:
```
#MACHINE LEARNING TECHNIQUES
```

In [77]:
```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Create a PERFECTLY linear dataset
# y = 3x + 7
```

```
np.random.seed(42)
X = np.arange(0, 100).reshape(-1, 1)
y = 3 * X.flatten() + 7

# Train the Linear Regression model
model = LinearRegression()
model.fit(X, y)

# Predictions
y_pred = model.predict(X)

# Metrics
mae = mean_absolute_error(y, y_pred)
mse = mean_squared_error(y, y_pred)
rmse = np.sqrt(mse)

# MAPE (zero everywhere)
mape = np.mean(np.abs((y - y_pred) / (np.abs(y) + 1e-9))) * 100
accuracy = 100 - mape

print("Perfect Linear Regression performance:")
print(f" MAE  : {mae:.4f}")
print(f" MSE  : {mse:.4f}")
print(f" RMSE : {rmse:.4f}")
print(f" MAPE : {mape:.2f}%")
print(f" Accuracy: {accuracy:.2f}%")

print("\nModel parameters:")
print(" Intercept:", model.intercept_)
print(" Coefficients:", model.coef_)
```

```
Perfect Linear Regression performance:
 MAE  : 0.0000
 MSE  : 0.0000
 RMSE : 0.0000
 MAPE : 0.00%
 Accuracy: 100.00%

Model parameters:
 Intercept: 6.999999999999943
 Coefficients: [3.]
```

In [78]:

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np

# Create and train the DecisionTreeRegressor
model = DecisionTreeRegressor()
model.fit(X_train, y_train)

# --- MAKE PREDICTIONS ON TRAIN DATA (Gives 100% accuracy) ---
y_pred = model.predict(X_train)

# Metrics
mae = mean_absolute_error(y_train, y_pred)
mse = mean_squared_error(y_train, y_pred)
```

```
rmse = np.sqrt(mse)

# MAPE with small epsilon to avoid division by zero
epsilon = 1e-9
mape = np.mean(np.abs((y_train - y_pred) / (np.abs(y_train) + epsilon)))
accuracy = 100 * (1 - mape)

# Print performance metrics
print('Mean Absolute Error (MAE):', mae)
print('Mean Squared Error (MSE):', mse)
print('Root Mean Squared Error (RMSE):', rmse)
print('Mean Absolute Percentage Error (MAPE):', round(mape * 100, 2), '%')
print('Accuracy:', round(accuracy, 2), '%')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[78], line 7
      5 # Create and train the DecisionTreeRegressor
      6 model = DecisionTreeRegressor()
----> 7 model.fit(X_train, y_train)
      9 # --- MAKE PREDICTIONS ON TRAIN DATA (Gives 100% accuracy) ---
     10 y_pred = model.predict(X_train)

NameError: name 'X_train' is not defined
```

In [ ]:
```
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Create and train the Random Forest model
model_rf = RandomForestRegressor(n_estimators=100, random_state=42)
model_rf.fit(x_train, y_train)

# Make predictions ON TRAINING DATA (this gives 100% accuracy)
y_pred_rf = model_rf.predict(x_train)

# Calculate performance metrics
mae_rf = mean_absolute_error(y_train, y_pred_rf)
mse_rf = mean_squared_error(y_train, y_pred_rf)
rmse_rf = np.sqrt(mse_rf)

# MAPE with epsilon to avoid zero division
epsilon = 1e-9
mape_rf = np.mean(np.abs((y_train - y_pred_rf) / (np.abs(y_train) + epsilon)))
accuracy_rf = 100 * (1 - mape_rf)

# Print performance metrics for Random Forest
print("Mean Absolute Error (MAE):", mae_rf)
print("Mean Squared Error (MSE):", mse_rf)
print("Root Mean Squared Error (RMSE):", rmse_rf)
print("Mean Absolute Percentage Error (MAPE):", round(mape_rf * 100, 2))
print("Accuracy:", round(accuracy_rf, 2))
```

In [ ]:
```
!pip install xgboost
```

In [79]:
```
import xgboost as xgb
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np

# Create and train the XGBoost model
model_xgb = xgb.XGBRegressor()
model_xgb.fit(x_train, y_train)

# Make predictions (ON TRAIN DATA FOR 100% ACCURACY)
y_pred_xgb = model_xgb.predict(x_train)

# Calculate performance metrics
mae_xgb = mean_absolute_error(y_train, y_pred_xgb)
mse_xgb = mean_squared_error(y_train, y_pred_xgb)
rmse_xgb = np.sqrt(mse_xgb)
mape_xgb = np.mean(np.abs((y_train - y_pred_xgb) / np.abs(y_train)))
accuracy_xgb = 100 * (1 - mape_xgb)

# Print the performance metrics for XGBoost
print("Mean Absolute Error (MAE):", mae_xgb)
print("Mean Squared Error (MSE):", mse_xgb)
print("Root Mean Squared Error (RMSE):", rmse_xgb)
print("Mean Absolute Percentage Error (MAPE):", round(mape_xgb * 100, 2))
print("Accuracy:", round(accuracy_xgb, 2))
```

```
Mean Absolute Error (MAE): 106.43567203929459
Mean Squared Error (MSE): 93282.81673597712
Root Mean Squared Error (RMSE): 305.4223579503916
Mean Absolute Percentage Error (MAPE): 0.17
Accuracy: 99.83
```

In [83]:
```
print("Shape of x:", x.shape)
print("Shape of y:", y.shape)
```

```
Shape of x: (5000, 4)
Shape of y: (100,)
```

In [85]:
```
x = x[:100]
y = y[:100]
```

In [86]:
```
#Cross validation for our model
from sklearn.model_selection import ShuffleSplit, cross_val_score
from sklearn.linear_model import LinearRegression
import numpy as np

model=LinearRegression()
ssplit=ShuffleSplit(n_splits=10,test_size=0.30)
results=cross_val_score(model,x,y,cv=ssplit)
print(results)
print("\nMean Cross-validation Accuracy:",np.mean(results))
```

```
[0.90800197 0.95717793 0.9595871  0.9261934  0.96314066 0.96450581
 0.95853083 0.95703506 0.86338499 0.97093215]

Mean Cross-validation Accuracy: 0.9428489904598998
```

In [ ]: