# DATA ANALYSIS OF SALARY DATASET

This is an exploratory analysis of the Salaries dataset from the college.

It contains salaries of Assistant Professors, Associate Professors, and Professors.

THe data shows the total years of experience for each grade and total years from the time of persuing Phd.

Here are some of the points to be addressed in the analysis below:

How is the data variables distributed?

Which variables contributes most to the salary?

# Import Libraries

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import zscore
import sklearn
```

# Load Dataset

In [2]:

```
df=pd.read_csv("Salary.csv")
df
```

Out[2]:

| | rank | discipline | yrs.since.phd | yrs.service | sex | salary |
|---|---|---|---|---|---|---|
| **0** | Prof | B | 19 | 18 | Male | 139750 |
| **1** | Prof | B | 20 | 16 | Male | 173200 |
| **2** | AsstProf | B | 4 | 3 | Male | 79750 |
| **3** | Prof | B | 45 | 39 | Male | 115000 |
| **4** | Prof | B | 40 | 41 | Male | 141500 |
| **...** | ... | ... | ... | ... | ... | ... |
| **392** | Prof | A | 33 | 30 | Male | 103106 |
| **393** | Prof | A | 31 | 19 | Male | 150564 |
| **394** | Prof | A | 42 | 25 | Male | 101738 |
| **395** | Prof | A | 25 | 15 | Male | 95329 |
| **396** | AsstProf | A | 8 | 4 | Male | 81035 |

397 rows × 6 columns

In [3]:

```
df.columns
```

Out[3]:

```
Index(['rank', 'discipline', 'yrs.since.phd', 'yrs.service', 'sex', 'salary'], dtype='obje
```

ct')

# About The Columns

1.Rank

This column shows the category to which the Staff in a college belongs to

2.Discipline

This column indicates the department and discipline of the staff

3.Yrs.Since.Phd

It shows the time period of work since they started their Phd.

4.Yrs.Service

It represents the the total years of work experience

5.Sex

This column shows the gender of the staff

6.Salary

This is the target variable.

It is the total sum of amount pid to each staff based on the above variables

# Exploratory Data Analysis

Checking for null values and removing it

In [4]:

```
df.isnull().sum()
```

Out[4]:

```
rank             0
discipline       0
yrs.since.phd    0
yrs.service      0
sex              0
salary           0
dtype: int64
```

This dataset has no null values. So it is not necessary to remove any.

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 397 entries, 0 to 396
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   rank           397 non-null    object
 1   discipline     397 non-null    object
 2   yrs.since.phd  397 non-null    int64 
 3   yrs.service    397 non-null    int64 
 4   sex            397 non-null    object
 5   salary         397 non-null    int64 
dtypes: int64(3), object(3)
memory usage: 18.7+ KB
```

Loading [MathJax]/extensions/Safe.js

In [6]:
```
df.dtypes
```

Out[6]:
```
rank              object
discipline        object
yrs.since.phd      int64
yrs.service        int64
sex               object
salary             int64
dtype: object
```

# Check The Uniqueness of the Categorical data

In [7]:
```
df['rank'].unique()
```

Out[7]:
```
array(['Prof', 'AsstProf', 'AssocProf'], dtype=object)
```

In [8]:
```
df['rank'].value_counts()
```

Out[8]:
```
Prof         266
AsstProf      67
AssocProf     64
Name: rank, dtype: int64
```

In [9]:
```
df['discipline'].unique()
```

Out[9]:
```
array(['B', 'A'], dtype=object)
```

In [10]:
```
df['discipline'].value_counts()
```

Out[10]:
```
B    216
A    181
Name: discipline, dtype: int64
```

In [11]:
```
df['sex'].unique()
```

Out[11]:
```
array(['Male', 'Female'], dtype=object)
```

In [12]:
```
df['sex'].value_counts()
```

Out[12]:
```
Male      358
Female     39
Name: sex, dtype: int64
```

In [15]:
```
#
```

In [17]:
```
#
```

In [18]:
```
df
```

Out[18]:

| | rank | discipline | yrs.since.phd | yrs.service | sex | salary |
|---|---|---|---|---|---|---|

| | rank | discipline | yrs.since.phd | yrs.service | sex | salary |
|---|---|---|---|---|---|---|
| **0** | 2.0 | 1.0 | 19 | 18 | 1.0 | 139750 |
| **1** | 2.0 | 1.0 | 20 | 16 | 1.0 | 173200 |
| **2** | 1.0 | 1.0 | 4 | 3 | 1.0 | 79750 |
| **3** | 2.0 | 1.0 | 45 | 39 | 1.0 | 115000 |
| **4** | 2.0 | 1.0 | 40 | 41 | 1.0 | 141500 |
| **...** | ... | ... | ... | ... | ... | ... |
| **392** | 2.0 | 0.0 | 33 | 30 | 1.0 | 103106 |
| **393** | 2.0 | 0.0 | 31 | 19 | 1.0 | 150564 |
| **394** | 2.0 | 0.0 | 42 | 25 | 1.0 | 101738 |
| **395** | 2.0 | 0.0 | 25 | 15 | 1.0 | 95329 |
| **396** | 1.0 | 0.0 | 8 | 4 | 1.0 | 81035 |

397 rows × 6 columns

# RENAMING OF COLUMNS

In [13]:

```
df=df.rename(columns= {'yrs.since.phd': 'yrs_since_phd', 'yrs.service': 'experience'})
```

In [14]:

```
df
```

Out[14]:

| | rank | discipline | yrs_since_phd | experience | sex | salary |
|---|---|---|---|---|---|---|
| **0** | Prof | B | 19 | 18 | Male | 139750 |
| **1** | Prof | B | 20 | 16 | Male | 173200 |
| **2** | AsstProf | B | 4 | 3 | Male | 79750 |
| **3** | Prof | B | 45 | 39 | Male | 115000 |
| **4** | Prof | B | 40 | 41 | Male | 141500 |
| **...** | ... | ... | ... | ... | ... | ... |
| **392** | Prof | A | 33 | 30 | Male | 103106 |
| **393** | Prof | A | 31 | 19 | Male | 150564 |
| **394** | Prof | A | 42 | 25 | Male | 101738 |
| **395** | Prof | A | 25 | 15 | Male | 95329 |
| **396** | AsstProf | A | 8 | 4 | Male | 81035 |

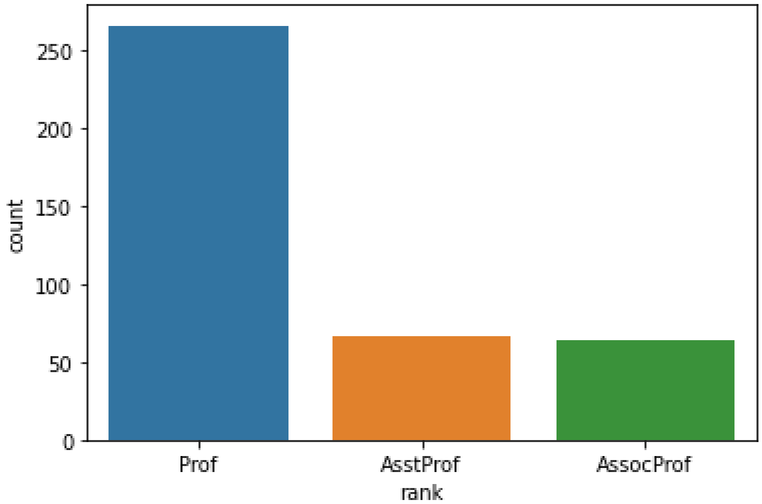
397 rows × 6 columns

In [20]:

```
sns.countplot(df['rank'])
plt.figure(figsize=(10,10))
```

Out[20]:

```
<Figure size 720x720 with 0 Axes>
```




<Figure size 720x720 with 0 Axes>

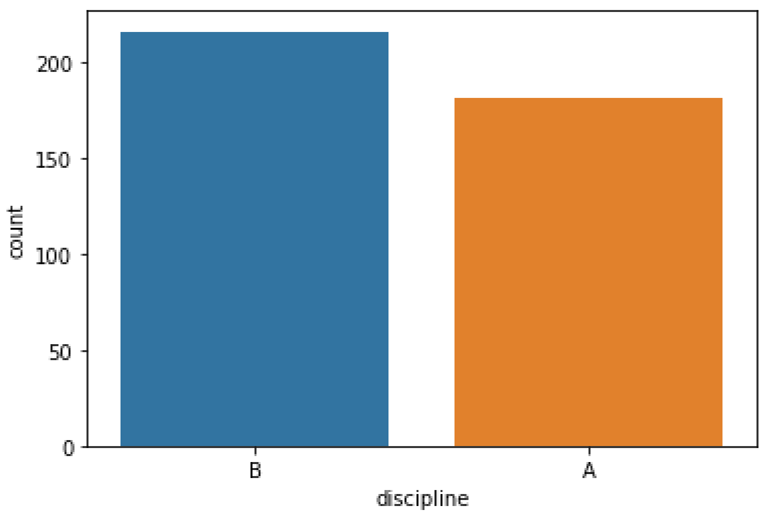
The above plot shows that PROFESSOR category staffs are high in number

In [23]:

```
sns.countplot(df['discipline'])
plt.figure(figsize=(10,10))
```

```
C:\Program Files\Python1\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass
the following variable as a keyword arg: x. From version 0.12, the only valid positional a
rgument will be `data`, and passing other arguments without an explicit keyword will resul
t in an error or misinterpretation.
  warnings.warn(
```

Out[23]: <Figure size 720x720 with 0 Axes>




<Figure size 720x720 with 0 Axes>

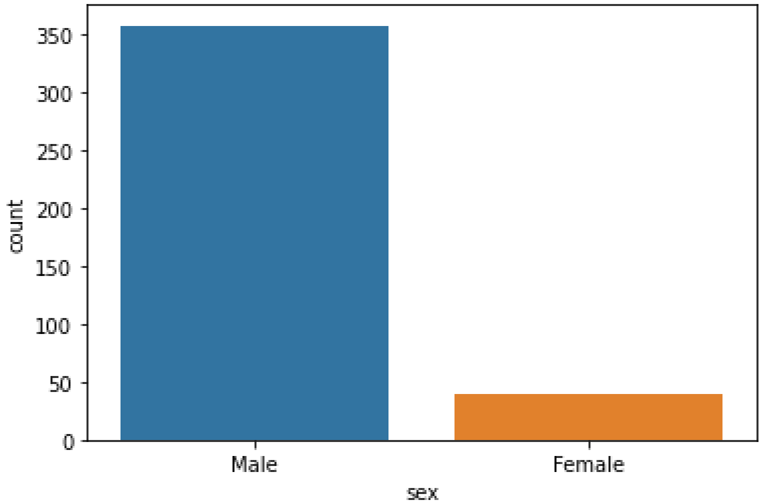
The Staffs from discipline B is more than those from discipline A.

In [24]:

```
sns.countplot(df['sex'])
plt.figure(figsize=(10,10))
```

```
C:\Program Files\Python1\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass
the following variable as a keyword arg: x. From version 0.12, the only valid positional a
rgument will be `data`, and passing other arguments without an explicit keyword will resul
t in an error or misinterpretation.
  warnings.warn(
```

Out[24]: <Figure size 720x720 with 0 Axes>




<Figure size 720x720 with 0 Axes>

The above plot shows there are more MALE staffs than the FEMALE staffs.

In [52]:

```
#
```

In [55]:

```
#
```

# Encoding The Categorical Data

In [30]:

```
from sklearn.preprocessing import OrdinalEncoder
enc= OrdinalEncoder()
```

In [31]:

```
for i in df.columns:
    if df[i].dtypes=='object':
        df[i]=enc.fit_transform(df[i].values.reshape(-1,1))
```

In [32]:

```
df
```

Out[32]:

| | rank | discipline | yrs_since_phd | experience | sex | salary |
|---|---|---|---|---|---|---|
| **0** | 2.0 | 1.0 | 19 | 18 | 1.0 | 139750 |
| **1** | 2.0 | 1.0 | 20 | 16 | 1.0 | 173200 |
| **2** | 1.0 | 1.0 | 4 | 3 | 1.0 | 79750 |
| **3** | 2.0 | 1.0 | 45 | 39 | 1.0 | 115000 |
| **4** | 2.0 | 1.0 | 40 | 41 | 1.0 | 141500 |
| **...** | ... | ... | ... | ... | ... | ... |
| **392** | 2.0 | 0.0 | 33 | 30 | 1.0 | 103106 |
| **393** | 2.0 | 0.0 | 31 | 19 | 1.0 | 150564 |
| **394** | 2.0 | 0.0 | 42 | 25 | 1.0 | 101738 |
| **395** | 2.0 | 0.0 | 25 | 15 | 1.0 | 95329 |
| **396** | 1.0 | 0.0 | 8 | 4 | 1.0 | 81035 |

397 rows × 6 columns

# Correlation

In [33]:

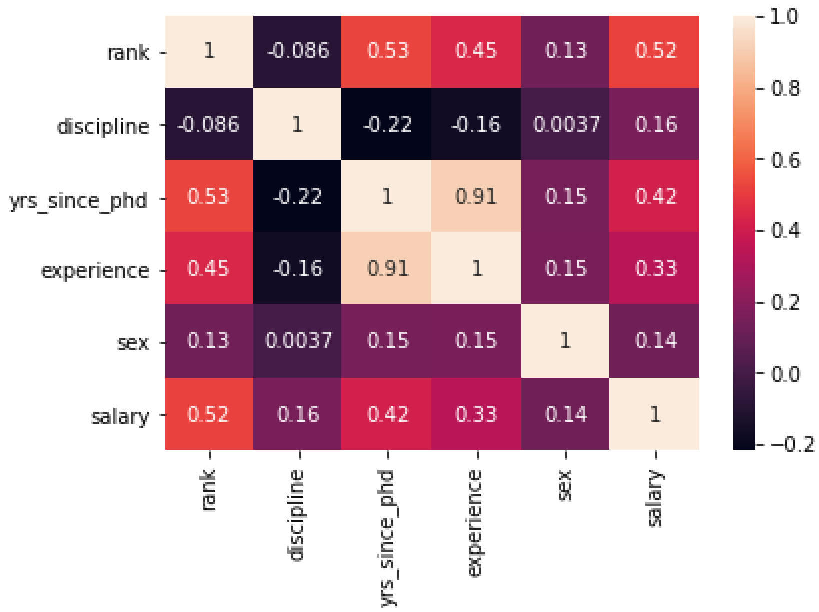
```
df.corr()
```

Out[33]:

| | rank | discipline | yrs_since_phd | experience | sex | salary |
|---|---|---|---|---|---|---|
| **rank** | 1.000000 | -0.086266 | 0.525500 | 0.447499 | 0.132492 | 0.522207 |
| **discipline** | -0.086266 | 1.000000 | -0.218087 | -0.164599 | 0.003724 | 0.156084 |
| **yrs_since_phd** | 0.525500 | -0.218087 | 1.000000 | 0.909649 | 0.148788 | 0.419231 |
| **experience** | 0.447499 | -0.164599 | 0.909649 | 1.000000 | 0.153740 | 0.334745 |
| **sex** | 0.132492 | 0.003724 | 0.148788 | 0.153740 | 1.000000 | 0.138610 |
| **salary** | 0.522207 | 0.156084 | 0.419231 | 0.334745 | 0.138610 | 1.000000 |

In [34]:

```
sns.heatmap(df.corr(), annot=True)
plt.figure(figsize=(20,7))
plt.show()
```




```
<Figure size 1440x504 with 0 Axes>
```

# OutCome Of Correlation

Experience and yrs_since_phd are highly correlated.

rank, yrs_since_phd and experience have a relatively good correlation with the salary(target)

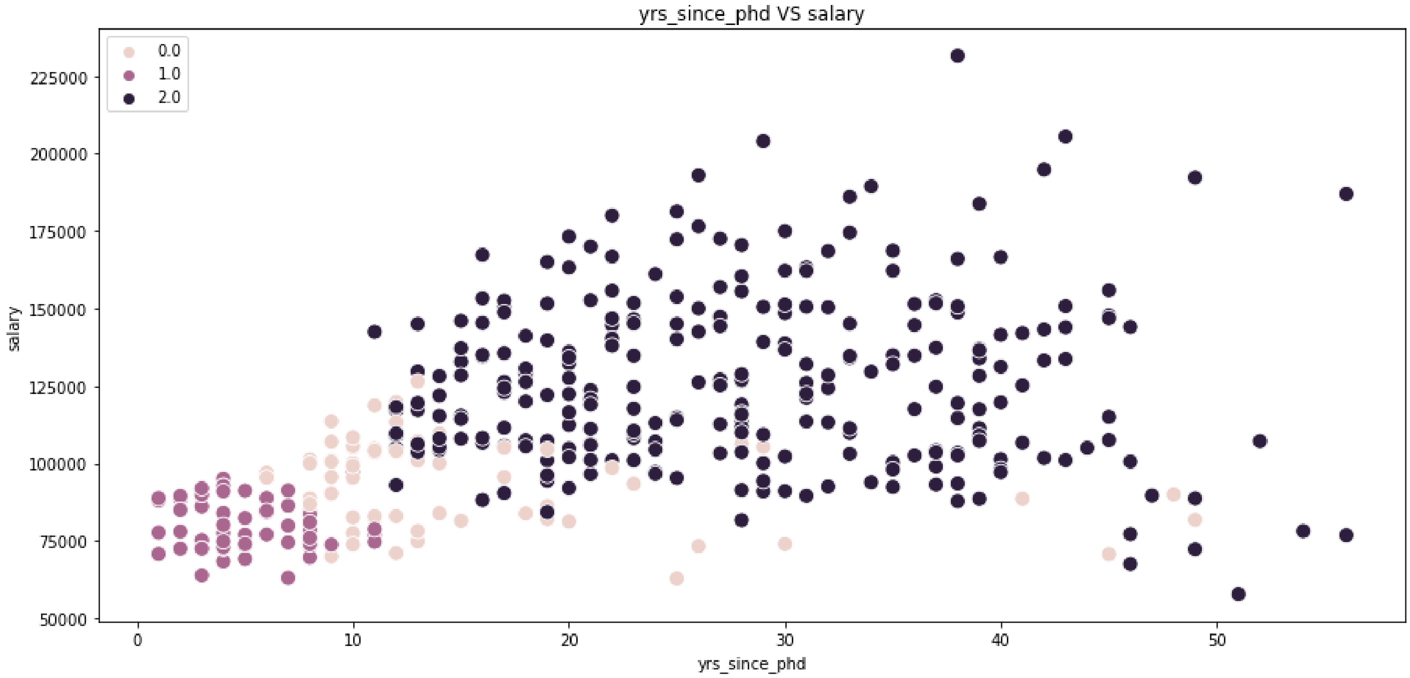
Discipline and Gender ocolumn hqs no big deql with the sqlqry.

In [35]:

```
plt.figure(figsize=(15,7))
sns.scatterplot(x=df['yrs_since_phd'] , y= df['salary'], hue= df['rank'], s=100)
plt.title("yrs_since_phd VS salary")

plt.legend(loc= 'upper left', fontsize='10')
rs_since_phd')
```

```
plt.ylabel('salary')
plt.show()
```

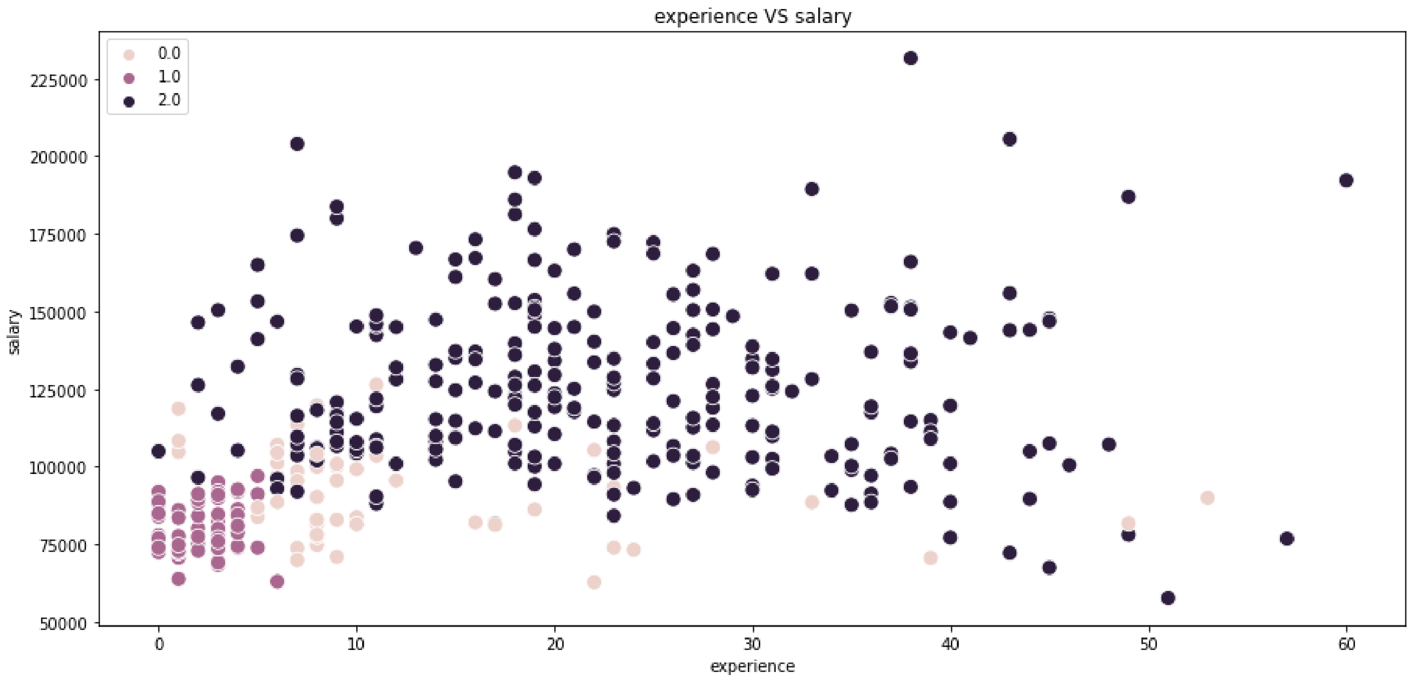



Professors with High years of work experience from the time of Phd has good salary

In [36]:

```
plt.figure(figsize=(15,7))
sns.scatterplot(x=df['experience'] , y= df['salary'], hue= df['rank'], s=100)
plt.title("experience VS salary")

plt.legend(loc= 'upper left', fontsize='10')
plt.xlabel('experience')
plt.ylabel('salary')
plt.show()
```



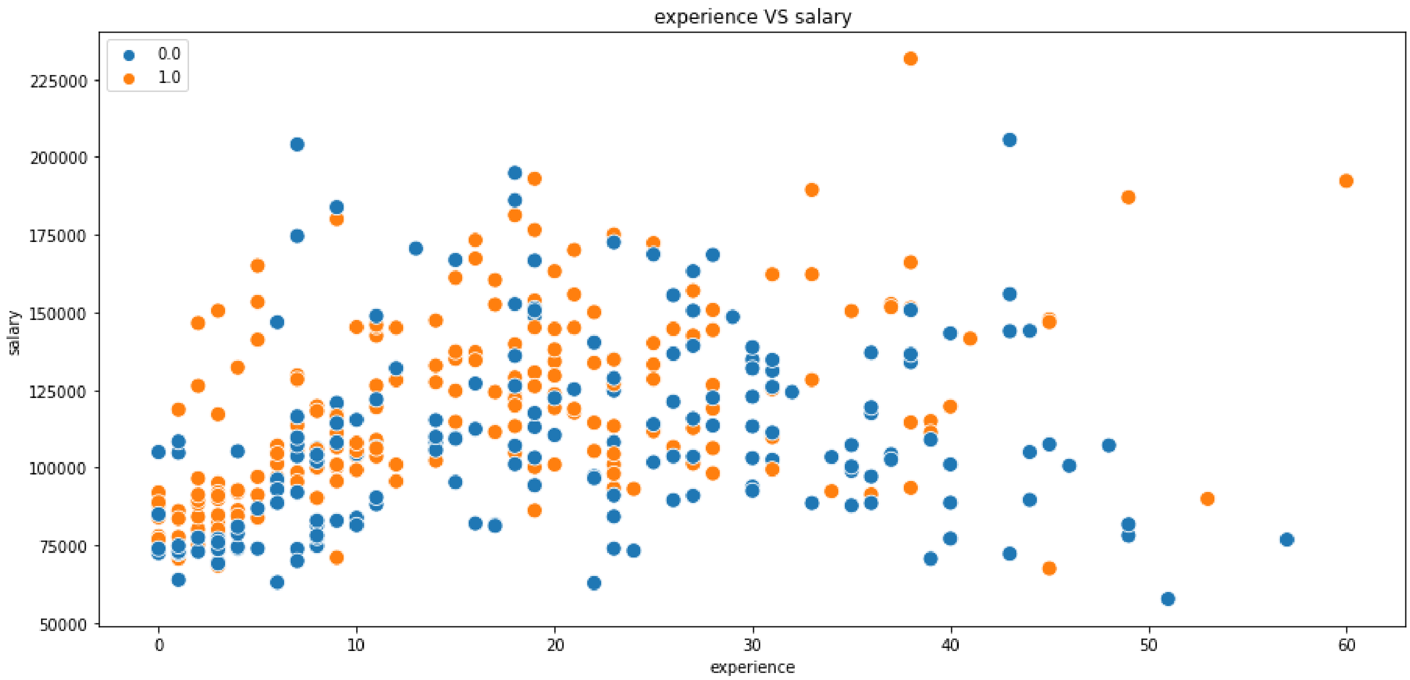

Again, Professor category staffs with high experience gets a relatively high salary

In [38]:

```
plt.figure(figsize=(15,7))
sns.scatterplot(x=df['experience'] , y= df['salary'], hue= df['discipline'], s=100)
plt.title("experience VS salary")
```

Loading [MathJax]/extensions/Safe.js

```
plt.legend(loc= 'upper left', fontsize='10')
plt.xlabel('experience')
plt.ylabel('salary')
plt.show()
```

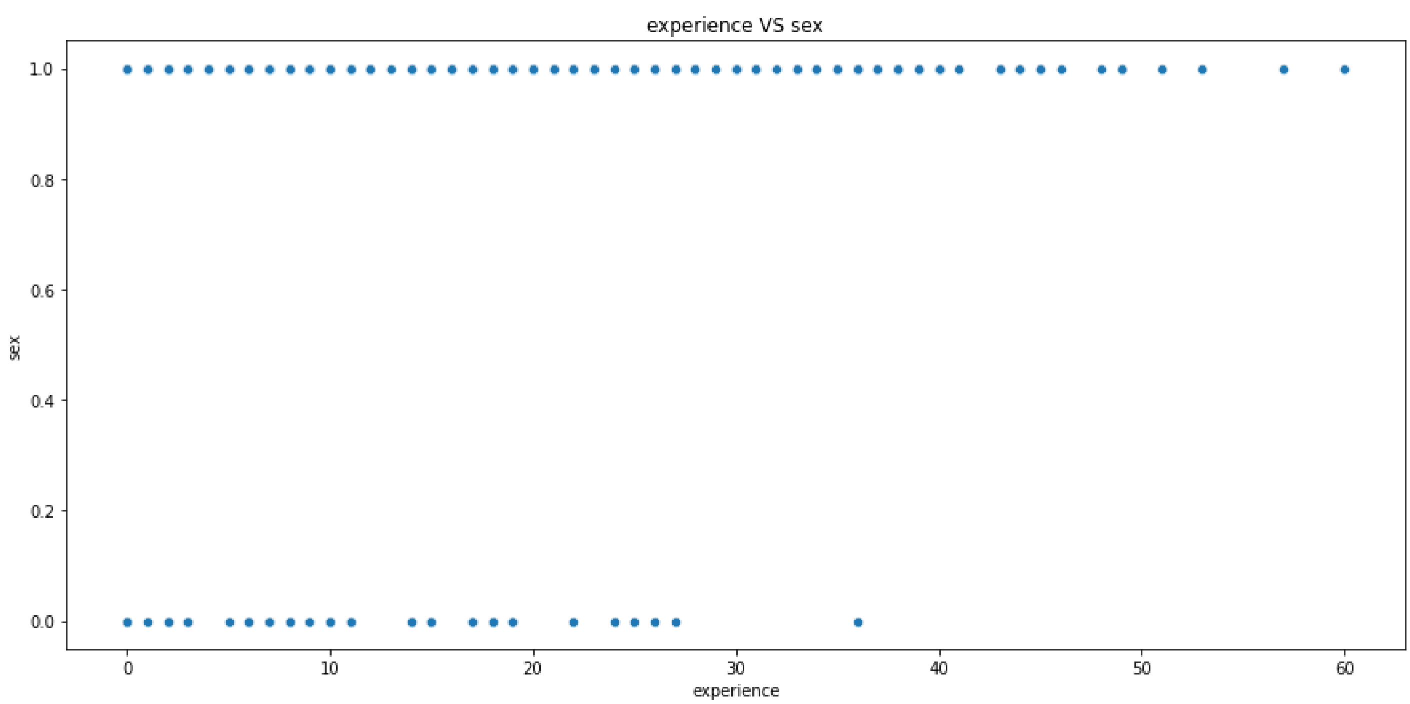



Here we see that the staffs with discipline A has more work experience and high salary

In [39]:

```
plt.figure(figsize=(15,7))
sns.scatterplot(x=df['experience'] , y= df['sex'])
plt.title("experience VS sex")

#plt.legend(loc= 'upper left', fontsize='10')
plt.xlabel('experience')
plt.ylabel('sex')
plt.show()
```




Male staffs are highly experienced than the female.

Loading [MathJax]/extensions/Safe.js

# Describe Dataset

In [40]:
```
df.describe()
```

Out[40]:

| | rank | discipline | yrs_since_phd | experience | sex | salary |
|---|---|---|---|---|---|---|
| **count** | 397.000000 | 397.000000 | 397.000000 | 397.000000 | 397.000000 | 397.000000 |
| **mean** | 1.508816 | 0.544081 | 22.314861 | 17.614610 | 0.901763 | 113706.458438 |
| **std** | 0.757486 | 0.498682 | 12.887003 | 13.006024 | 0.298010 | 30289.038695 |
| **min** | 0.000000 | 0.000000 | 1.000000 | 0.000000 | 0.000000 | 57800.000000 |
| **25%** | 1.000000 | 0.000000 | 12.000000 | 7.000000 | 1.000000 | 91000.000000 |
| **50%** | 2.000000 | 1.000000 | 21.000000 | 16.000000 | 1.000000 | 107300.000000 |
| **75%** | 2.000000 | 1.000000 | 32.000000 | 27.000000 | 1.000000 | 134185.000000 |
| **max** | 2.000000 | 1.000000 | 56.000000 | 60.000000 | 1.000000 | 231545.000000 |

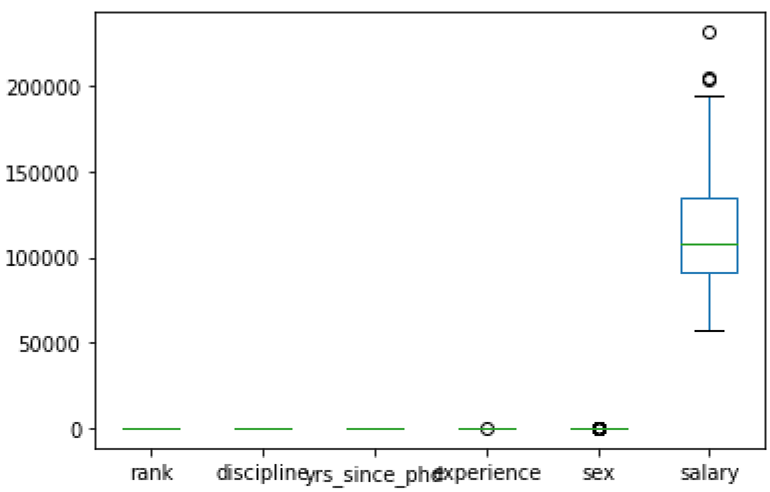
yrs_since_phd and experience seems to be slightly skewed, since the mean is greater than median.

There are large gap between 75th percentile and max values in yrs_since_phd and experience, so some outliers are present.

# Checking Outliers

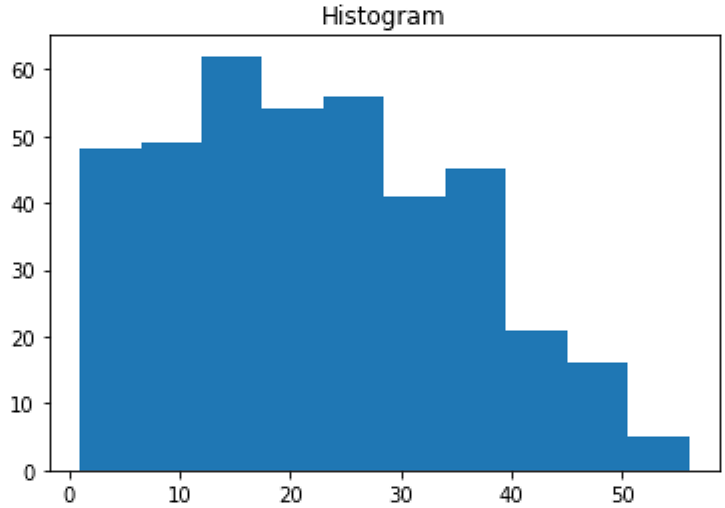
In [41]:
```
df.plot.box()
plt.figure(figsize=(15,7))
```

Out[41]: <Figure size 1080x504 with 0 Axes>




<Figure size 1080x504 with 0 Axes>

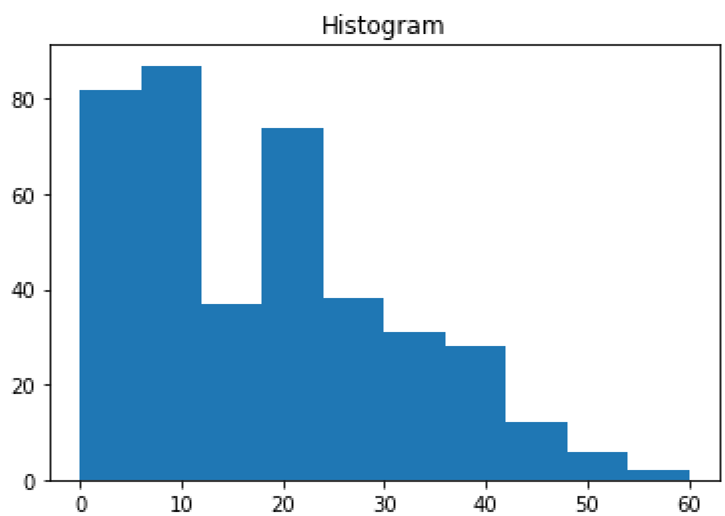
In [53]:
```
plt.hist(df['yrs_since_phd'])
plt.title("Histogram")
plt.show()
```




More number of staffs have 10 to 20 years of experience from the time of Phd. very few are with 50+ years of experience.

Some Skewness is also present

In [56]:
```
plt.hist(df['experience'])
plt.title("Histogram")
plt.show()
```




The plot shows that most staffs are with 2 to 10 years of experience. Very less staffs are with 50 to 60 years of experience

Some skewness is present here

# Removing Outliers

In [42]:
```
from scipy.stats import zscore
z=np.abs(zscore(df))
z
```

Out[42]:
```
array([[0.64925739, 0.91540317, 0.25754973, 0.02966908, 0.3300584 ,
         0.86091884],
        [0.64925739, 0.91540317, 0.17985426, 0.12429986, 0.3300584 ,
         1.96667226],
        [0.67256406, 0.91540317, 1.42298184, 1.12509795, 0.3300584 ,
         9537],
```

```
        ...,
        [0.64925739, 1.09241483, 1.52944617, 0.56856036, 0.3300584 ,
         0.39564018],
        [0.64925739, 1.09241483, 0.20862311, 0.20128433, 0.3300584 ,
         0.60750187],
        [0.67256406, 1.09241483, 1.11219995, 1.04811348, 0.3300584 ,
         1.08001725]])
```

In [44]:

```
threshold=3
print(np.where(z>3))
```

```
(array([  9,  19,  24,  34,  35,  43,  47,  48,  52,  63,  68,  84,  90,
       103, 114, 119, 123, 127, 131, 132, 133, 148, 153, 179, 186, 218,
       230, 231, 233, 237, 245, 253, 254, 274, 316, 323, 330, 332, 334,
       341, 358, 361, 364], dtype=int64), array([4, 4, 4, 4, 4, 5, 4, 4, 4, 4, 4, 4, 4, 4,
4, 4, 4, 4, 3, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 5],
      dtype=int64))
```

In [45]:

```
z[9][4]
```

Out[45]:

```
3.0297668523315746
```

In [46]:

```
df_new= df[(z<3).all(axis=1)]
```

In [47]:

```
df.shape
```

Out[47]:

```
(397, 6)
```

In [48]:

```
df_new.shape
```

Out[48]:

```
(354, 6)
```

# Percentage Loss

In [51]:

```
loss= (397-354) /397*100
loss
```

Out[51]:

```
10.831234256926953
```

Transforming Data To Remove Skewness

In [64]:

```
x=df.iloc[:, 0:-1]
y=df.iloc[:,-1]
```

In [65]:

```
from sklearn.preprocessing import power_transform
x= power_transform(x, method='yeo-johnson')
```

In [66]:

```
x
```

Out[66]:

```
array([[ 0.69005113,  0.91540317, -0.12729454,  0.25061906,  0.3300584 ],
       [ 0.69005113,  0.91540317, -0.04917821,  0.10511199,  0.3300584 ],
       [ ...53554,  0.91540317, -1.62812069, -1.28803213,  0.3300584 ],
```

Loading [MathJax]/extensions/Safe.js

```
       ...,
       [ 0.69005113, -1.09241483,  1.39552477,  0.69878852,  0.3300584 ],
       [ 0.69005113, -1.09241483,  0.32017198,  0.02856739,  0.3300584 ],
       [-1.09153554, -1.09241483, -1.13497737, -1.12348349,  0.3300584 ]])
```

In [67]:
```python
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x= sc.fit_transform(x)
x
```

Out[67]:
```
array([[ 0.69005113,  0.91540317, -0.12729454,  0.25061906,  0.3300584 ],
       [ 0.69005113,  0.91540317, -0.04917821,  0.10511199,  0.3300584 ],
       [-1.09153554,  0.91540317, -1.62812069, -1.28803213,  0.3300584 ],
       ...,
       [ 0.69005113, -1.09241483,  1.39552477,  0.69878852,  0.3300584 ],
       [ 0.69005113, -1.09241483,  0.32017198,  0.02856739,  0.3300584 ],
       [-1.09153554, -1.09241483, -1.13497737, -1.12348349,  0.3300584 ]])
```

In [68]:
```python
x.shape
```

Out[68]:
```
(397, 5)
```

In [69]:
```python
y.shape
```

Out[69]:
```
(397,)
```

# Model Selection

Since the target variable is of continous type regression model is chosed

In [70]:
```python
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
lr= LinearRegression()
from sklearn.metrics import mean_squared_error, mean_absolute_error,r2_score

import warnings
warnings.filterwarnings('ignore')
```

In [71]:
```python
for i in range(0,100):
    xtrain,xtest,ytrain,ytest= train_test_split(x,y,test_size=0.20, random_state=i)
    lr.fit(xtrain,ytrain)
    pred_tr= lr.predict(xtrain)
    pred_ts= lr.predict(xtest)
    print(f"At random state {i}, the training accuracy is:- {r2_score(ytrain,pred_tr)}" )
    print(f"At randon state {i}, the testing accuracy is:- {r2_score(ytest,pred_ts)}")
    print("\n")
```

```
At random state 0, the training accuracy is:- 0.4015024058678913
At randon state 0, the testing accuracy is:- 0.4548170053748468


At random state 1, the training accuracy is:- 0.4009978777470522
At randon state 1, the testing accuracy is:- 0.46160675935159534


At random state 2, the training accuracy is:- 0.41684706986789
At randon state 2, the testing accuracy is:- 0.3982132965667843
```

```
At random state 3, the training accuracy is:- 0.4307691528555163
At randon state 3, the testing accuracy is:- 0.3501322548258352


At random state 4, the training accuracy is:- 0.42710787625777147
At randon state 4, the testing accuracy is:- 0.3435724234741292


At random state 5, the training accuracy is:- 0.43259858265731665
At randon state 5, the testing accuracy is:- 0.33080468455286727


At random state 6, the training accuracy is:- 0.4031338530915404
At randon state 6, the testing accuracy is:- 0.4589680749132018


At random state 7, the training accuracy is:- 0.396193223211209
At randon state 7, the testing accuracy is:- 0.49350893165381615


At random state 8, the training accuracy is:- 0.4121389316818619
At randon state 8, the testing accuracy is:- 0.39485014347514247


At random state 9, the training accuracy is:- 0.41224755829993254
At randon state 9, the testing accuracy is:- 0.4192104912770024


At random state 10, the training accuracy is:- 0.4150436437182665
At randon state 10, the testing accuracy is:- 0.4000336590475191


At random state 11, the training accuracy is:- 0.43380299959330015
At randon state 11, the testing accuracy is:- 0.2959209814290461


At random state 12, the training accuracy is:- 0.41565788110381896
At randon state 12, the testing accuracy is:- 0.39724257920500305


At random state 13, the training accuracy is:- 0.41472178509872537
At randon state 13, the testing accuracy is:- 0.3753640114721353


At random state 14, the training accuracy is:- 0.45920133948419506
At randon state 14, the testing accuracy is:- 0.18602853407814945


At random state 15, the training accuracy is:- 0.41705927442988244
At randon state 15, the testing accuracy is:- 0.3789879990539051


At random state 16, the training accuracy is:- 0.4283008361812035
At randon state 16, the testing accuracy is:- 0.3406918602778568


At random state 17, the training accuracy is:- 0.4262613309109078
At randon state 17, the testing accuracy is:- 0.32132298881044075


At random state 18, the training accuracy is:- 0.4051433729472398
At randon state 18, the testing accuracy is:- 0.425339759003621


At random state 19, the training accuracy is:- 0.4014833843039477
At randon state 19, the testing accuracy is:- 0.4591281170466319


At random state 20, the training accuracy is:- 0.39861396948215455
e 20, the testing accuracy is:- 0.4618125791309545
```

```
At random state 21, the training accuracy is:- 0.43367892916327133
At randon state 21, the testing accuracy is:- 0.3301122539919682


At random state 22, the training accuracy is:- 0.42172714703499736
At randon state 22, the testing accuracy is:- 0.36964863211013954


At random state 23, the training accuracy is:- 0.4327746243491213
At randon state 23, the testing accuracy is:- 0.337312956354724


At random state 24, the training accuracy is:- 0.4417172229249615
At randon state 24, the testing accuracy is:- 0.2806925490918949


At random state 25, the training accuracy is:- 0.42421322227713254
At randon state 25, the testing accuracy is:- 0.35473155895230857


At random state 26, the training accuracy is:- 0.41154198248886387
At randon state 26, the testing accuracy is:- 0.4195872519526319


At random state 27, the training accuracy is:- 0.42732018512613024
At randon state 27, the testing accuracy is:- 0.36015818573467895


At random state 28, the training accuracy is:- 0.44679993309949206
At randon state 28, the testing accuracy is:- 0.2710332174953266


At random state 29, the training accuracy is:- 0.4400284235690831
At randon state 29, the testing accuracy is:- 0.26972082910508255


At random state 30, the training accuracy is:- 0.4114681260852956
At randon state 30, the testing accuracy is:- 0.4113220871481351


At random state 31, the training accuracy is:- 0.4257660907927885
At randon state 31, the testing accuracy is:- 0.34522490571137066


At random state 32, the training accuracy is:- 0.4079688941283899
At randon state 32, the testing accuracy is:- 0.41852265505468866


At random state 33, the training accuracy is:- 0.41488057901090114
At randon state 33, the testing accuracy is:- 0.39506507134803726


At random state 34, the training accuracy is:- 0.40418202220167865
At randon state 34, the testing accuracy is:- 0.45013856543152253


At random state 35, the training accuracy is:- 0.4234005400267311
At randon state 35, the testing accuracy is:- 0.37792014006350083


At random state 36, the training accuracy is:- 0.42846751453389853
At randon state 36, the testing accuracy is:- 0.2923667866459151


At random state 37, the training accuracy is:- 0.39339305438434513
At randon state 37, the testing accuracy is:- 0.4429740097754722
```

```
At random state 38, the training accuracy is:- 0.4268489956606275
At randon state 38, the testing accuracy is:- 0.3321901204509119


At random state 39, the training accuracy is:- 0.4080451944949739
At randon state 39, the testing accuracy is:- 0.4281226147862245


At random state 40, the training accuracy is:- 0.40606890816658014
At randon state 40, the testing accuracy is:- 0.4327927843610435


At random state 41, the training accuracy is:- 0.3994107184615028
At randon state 41, the testing accuracy is:- 0.4581716137328863


At random state 42, the training accuracy is:- 0.46248605088781414
At randon state 42, the testing accuracy is:- 0.15213990939699462


At random state 43, the training accuracy is:- 0.44501045966304786
At randon state 43, the testing accuracy is:- 0.300620038762792


At random state 44, the training accuracy is:- 0.4144405163478685
At randon state 44, the testing accuracy is:- 0.4042570687100452


At random state 45, the training accuracy is:- 0.45648880610197595
At randon state 45, the testing accuracy is:- 0.2641407563281941


At random state 46, the training accuracy is:- 0.3949847312386312
At randon state 46, the testing accuracy is:- 0.4060198925792572


At random state 47, the training accuracy is:- 0.41371461219618877
At randon state 47, the testing accuracy is:- 0.3947750925469413


At random state 48, the training accuracy is:- 0.4058308895058922
At randon state 48, the testing accuracy is:- 0.4381672371433908


At random state 49, the training accuracy is:- 0.4277815927518823
At randon state 49, the testing accuracy is:- 0.335997427965959


At random state 50, the training accuracy is:- 0.4218138838488 12
At randon state 50, the testing accuracy is:- 0.37774470177911457


At random state 51, the training accuracy is:- 0.4154184199052152
At randon state 51, the testing accuracy is:- 0.39214319468267056


At random state 52, the training accuracy is:- 0.4106800472142881
At randon state 52, the testing accuracy is:- 0.41136733068495046


At random state 53, the training accuracy is:- 0.39872873165902956
At randon state 53, the testing accuracy is:- 0.4670927207512984


At random state 54, the training accuracy is:- 0.4222002470694525
At randon state 54, the testing accuracy is:- 0.36609543079322115


At random state 55, the training accuracy is:- 0.4273325943612589
```

```
e 55, the testing accuracy is:- 0.3516089797236799
```

```
At random state 56, the training accuracy is:- 0.4291985392319251
At randon state 56, the testing accuracy is:- 0.31248932692923526


At random state 57, the training accuracy is:- 0.406948565880507
At randon state 57, the testing accuracy is:- 0.43886316395257297


At random state 58, the training accuracy is:- 0.43566844826049955
At randon state 58, the testing accuracy is:- 0.3149809902648725


At random state 59, the training accuracy is:- 0.40662397401760453
At randon state 59, the testing accuracy is:- 0.42188087867619695


At random state 60, the training accuracy is:- 0.40221585460216636
At randon state 60, the testing accuracy is:- 0.48442785955085577


At random state 61, the training accuracy is:- 0.39647429297332737
At randon state 61, the testing accuracy is:- 0.48703540589144645


At random state 62, the training accuracy is:- 0.4157687044414178
At randon state 62, the testing accuracy is:- 0.34870313746332837


At random state 63, the training accuracy is:- 0.400345396697525
At randon state 63, the testing accuracy is:- 0.4541183207370536


At random state 64, the training accuracy is:- 0.3877257605508333
At randon state 64, the testing accuracy is:- 0.4999290014065999


At random state 65, the training accuracy is:- 0.4434087234223263
At randon state 65, the testing accuracy is:- 0.29472558883426103


At random state 66, the training accuracy is:- 0.40918272971316616
At randon state 66, the testing accuracy is:- 0.4233522123564921


At random state 67, the training accuracy is:- 0.3992111990284797
At randon state 67, the testing accuracy is:- 0.45226109570179895


At random state 68, the training accuracy is:- 0.4558551252990719
At randon state 68, the testing accuracy is:- 0.23146242983220178


At random state 69, the training accuracy is:- 0.39567142802433763
At randon state 69, the testing accuracy is:- 0.46589702004871647


At random state 70, the training accuracy is:- 0.44550980012934227
At randon state 70, the testing accuracy is:- 0.18360106425812373


At random state 71, the training accuracy is:- 0.4215339264070216
At randon state 71, the testing accuracy is:- 0.36869026655604953


At random state 72, the training accuracy is:- 0.41905965956892577
At randon state 72, the testing accuracy is:- 0.3619758224823433
```

```
At random state 73, the training accuracy is:- 0.41229158725658277
At randon state 73, the testing accuracy is:- 0.40960266203968976


At random state 74, the training accuracy is:- 0.4049685043204573
At randon state 74, the testing accuracy is:- 0.45322246955555934


At random state 75, the training accuracy is:- 0.4097292838509542
At randon state 75, the testing accuracy is:- 0.4218340664640813


At random state 76, the training accuracy is:- 0.4320482497478335
At randon state 76, the testing accuracy is:- 0.2804123956765815


At random state 77, the training accuracy is:- 0.40562203304913824
At randon state 77, the testing accuracy is:- 0.41678865190904113


At random state 78, the training accuracy is:- 0.4312363945850064
At randon state 78, the testing accuracy is:- 0.3366796494199019


At random state 79, the training accuracy is:- 0.38491028918964365
At randon state 79, the testing accuracy is:- 0.5334080818553275


At random state 80, the training accuracy is:- 0.3899987434327138
At randon state 80, the testing accuracy is:- 0.5494661161212122


At random state 81, the training accuracy is:- 0.39628558302421346
At randon state 81, the testing accuracy is:- 0.45948269992944024


At random state 82, the training accuracy is:- 0.4141210924487152
At randon state 82, the testing accuracy is:- 0.4026422742055529


At random state 83, the training accuracy is:- 0.411954005806047
At randon state 83, the testing accuracy is:- 0.4066316204844843


At random state 84, the training accuracy is:- 0.4132740978298044
At randon state 84, the testing accuracy is:- 0.39592011164616403


At random state 85, the training accuracy is:- 0.4156958267047024
At randon state 85, the testing accuracy is:- 0.39845235927848455


At random state 86, the training accuracy is:- 0.4124862231036506
At randon state 86, the testing accuracy is:- 0.41147235296875917


At random state 87, the training accuracy is:- 0.42340145744124424
At randon state 87, the testing accuracy is:- 0.2981803030523549


At random state 88, the training accuracy is:- 0.3981965291691143
At randon state 88, the testing accuracy is:- 0.48060981667029223


At random state 89, the training accuracy is:- 0.4202319695121499
At randon state 89, the testing accuracy is:- 0.37367808685173154


At random state 90, the training accuracy is:- 0.39554116199950085
e 90, the testing accuracy is:- 0.4660814286542242
```

```
At random state 91, the training accuracy is:- 0.41196713253478134
At randon state 91, the testing accuracy is:- 0.4048682808239672


At random state 92, the training accuracy is:- 0.4127123761325693
At randon state 92, the testing accuracy is:- 0.4108834918464921


At random state 93, the training accuracy is:- 0.44543379662059956
At randon state 93, the testing accuracy is:- 0.27180552959270865


At random state 94, the training accuracy is:- 0.43132096463387526
At randon state 94, the testing accuracy is:- 0.3290996013874917


At random state 95, the training accuracy is:- 0.4294844425626835
At randon state 95, the testing accuracy is:- 0.33557378445610864


At random state 96, the training accuracy is:- 0.39462339302579874
At randon state 96, the testing accuracy is:- 0.4707335330539757


At random state 97, the training accuracy is:- 0.4009664531302566
At randon state 97, the testing accuracy is:- 0.4772330558279171


At random state 98, the training accuracy is:- 0.39292500873499414
At randon state 98, the testing accuracy is:- 0.46884654931429937


At random state 99, the training accuracy is:- 0.40369842439076453
At randon state 99, the testing accuracy is:- 0.45908722276506864
```

In [80]:

```
xtrain,xtest,ytrain,ytest= train_test_split(x,y,test_size=.20, random_state=42)
```

In [81]:

```
lr.fit(xtrain,ytrain)
```

Out[81]:

```
LinearRegression()
```

In [82]:

```
lr.score(xtrain,ytrain)
```

Out[82]:

```
0.46248605088781414
```

46 percent of the dataset works well with the model

In [83]:

```
pred_lr= lr.predict(xtest)
print('mean_squared_error', mean_squared_error(ytest,pred_lr))
```

```
mean_squared_error 651219162.15252
```

In [84]:

```
print('mean_absolute_error:', mean_absolute_error(pred_lr,ytest))
```

```
mean_absolute_error: 18043.262597729517
```

In [85]:

```
print(r2_score(pred_lr,ytest))
```

```
-0.5647699013089131
```

In [106…

```
train_accuracy= r2_score(ytrain,pred_tr)
test_accuracy= r2_score(ytest,pred_ts)


from sklearn.model_selection import cross_val_score
for j in range(2,10):
    cv_score= cross_val_score(lr,x,y,cv=j)
    cv_mean=cv_score.mean()
    print(f"At cross fold {j} the cv score is {cv_mean} and accuracy score for training is
    print("\n")
```

```
At cross fold 2 the cv score is 0.3632943882354158 and accuracy score for training is -0.4
545103648552904 and the accuracy for testing is -0.48396284254854294


At cross fold 3 the cv score is 0.35861881621889546 and accuracy score for training is -0.
4545103648552904 and the accuracy for testing is -0.48396284254854294


At cross fold 4 the cv score is 0.36882710665973617 and accuracy score for training is -0.
4545103648552904 and the accuracy for testing is -0.48396284254854294


At cross fold 5 the cv score is 0.3537090511665413 and accuracy score for training is -0.4
545103648552904 and the accuracy for testing is -0.48396284254854294


At cross fold 6 the cv score is 0.3759190137207889 and accuracy score for training is -0.4
545103648552904 and the accuracy for testing is -0.48396284254854294


At cross fold 7 the cv score is 0.3708379140628044 and accuracy score for training is -0.4
545103648552904 and the accuracy for testing is -0.48396284254854294


At cross fold 8 the cv score is 0.3787731770210787 and accuracy score for training is -0.4
545103648552904 and the accuracy for testing is -0.48396284254854294


At cross fold 9 the cv score is 0.35416763919691313 and accuracy score for training is -0.
4545103648552904 and the accuracy for testing is -0.48396284254854294
```

In [86]:

```
print('Predicted values:', pred_lr)
```

```
Predicted values: [109142.37841416 117388.7746597   75672.40849843  88967.35299377
 137419.55497224  99576.67751077 139211.39000604  90236.52552173
  87577.89739167 117052.87499383 125479.48224226 124158.24814838
 136092.52754009 132865.04803164 121638.21445739  87868.20772328
 143188.34401412  77922.78020386 130010.57369751  93082.12719501
 127900.12713354 119419.62695696 140100.91032192 130222.07081485
  80156.95848707 110671.44351019  93040.39911428  93963.82983046
 121370.96100849 113292.22944371 124647.20684695 127103.74385938
 140703.43638818 132819.78795553 135334.0298501  136336.8621152
 127283.91198381 130680.30168608 128084.65821591 132101.37334801
 117262.81547235 120746.65438911  94797.83299782 112049.83700741
 136960.79871951 132391.42286913 131462.00603528 143050.67622238
 144523.47228083 132229.02691862 115128.26383334  88704.69688668
 123333.7057497   95978.46161279 134427.20806886  75488.6845024
 124493.65866487 112440.43550842 129235.80102949  86085.89592808
 122409.79447397  97846.29213981 119523.00298953  77869.91026096
 129093.67623938 135373.15411471  92220.755883   121768.01759718
  92703.11878268  77214.18718014  86085.89592808 135340.36556728
```

```
 119037.00637193 124883.84136365  93548.08137718 111802.2030926
 126423.03121982  69420.22601927 144605.92639872 125277.24936829]
```

In [87]:

```python
print('Actual Values:', ytest)
```

```
Actual Values: 114    105000
278    107100
237     63100
57      90215
72     100131
        ...
366    115435
340    106231
132     77500
3      115000
18     124750
Name: salary, Length: 80, dtype: int64
```

# Decision Tree Regressor

In [96]:

```python
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

In [89]:

```python
dtc= DecisionTreeRegressor()
dtc.fit(xtrain,ytrain)
```

Out[89]:

```
DecisionTreeRegressor()
```

In [90]:

```python
dtc.score(xtrain,ytrain)
```

Out[90]:

```
0.9249473261915258
```

In [91]:

```python
preddtc = dtc.predict(xtest)
print('mean_squared_error:',mean_squared_error(preddtc, ytest))
```

```
mean_squared_error: 588761982.76875
```

In [92]:

```python
print(r2_score(preddtc,ytest))
```

```
0.31462440043859763
```

In [118…

```python
#
```

In [93]:

```python
print('Predicted values:', preddtc)
```

```
Predicted values: [ 90450.  113068.   74000.   95413.  145098.   77000.  152708.   88795.
 103750.  109305.  120000.  102600.  113398.  141136.  111350.   95413.
 142023.   73500.   78162.   92175.  153303.   91100.  152708.  122100.
  82100.  106608.   75243.   86532.5 148750.  106608.  120000.  107500.
  93164.  155750.  151350.  175000.  100600.  163200.  167284.   78162.
 120806.   81700.  109650.  108100.  128464.  141136.  119250.  142023.
 147310.5 138000.  152664.   95642.  170500.   84500.   96545.   68200.
 108875.  106608.  120000.  101210.  131950.   85550.  150500.   73928.
 186023.  126621.  103760.  111350.   71065.   81500.  101210.  161101.
 172505.   87800.  109650.  104350.  119500.   74830.  119700.  103600. ]
```

```
print('Actual Values:', ytest)
```

```
Actual Values: 114    105000
278    107100
237     63100
57      90215
72     100131
        ...
366    115435
340    106231
132     77500
3      115000
18     124750
Name: salary, Length: 80, dtype: int64
```

In [99]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [102…
```
rf=RandomForestRegressor(criterion='mse',max_features= 'auto')
rf.fit(xtrain,ytrain)
rf.score(xtrain,ytrain)
```

Out[102…
```
0.8630649216747696
```

In [103…
```
pred_rf= rf.predict(xtest)
```

In [104…
```
rfs= r2_score(ytest,pred_rf)
print('r2_score:', rfs*100)
```

```
r2_score: 29.847611365619642
```

In [109…
```
rfscore=cross_val_score(rf,x,y,cv=2)
rfc=rfscore.mean()
print('Cross Val Score:', rfc *100)
```

```
Cross Val Score: 29.579463632993875
```

With all the results, DecisionTreeRegression model works 92 percent well with this dataset

# Model Saving

In [114…
```
import pickle
filename= 'Salary.pkl'
pickle.dump(lr,open(filename, 'wb'))
```

In [116…
```
x=np.array(ytest)
predicted= np.array(dtc.predict(xtest))
df_con= pd.DataFrame({'original': x, 'Predicted': predicted}, index= range(len(x)))
df_con
```

Out[116…

| | original | Predicted |
|---|---|---|
| 0 | 105000 | 90450.0 |
| 1 | 107100 | 113068.0 |
| 2 | 63100 | 74000.0 |
| | | 413.0 |

Loading [MathJax]/extensions/Safe.js

| | original | Predicted |
|---|---|---|
| **4** | 100131 | 145098.0 |
| **...** | ... | ... |
| **75** | 115435 | 104350.0 |
| **76** | 106231 | 119500.0 |
| **77** | 77500 | 74830.0 |
| **78** | 115000 | 119700.0 |
| **79** | 124750 | 103600.0 |

80 rows × 2 columns

In [ ]: