```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

df =
pd.read_csv('https://raw.githubusercontent.com/dsrscientist/dataset3/
main/Salaries.csv')

df.head()
```

	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

```
df.tail()
```

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

```
df.sample(2)
```

	rank	discipline	yrs.since.phd	yrs.service	sex	salary
348	AsstProf	B	4	3	Male	80139
109	Prof	A	40	31	Male	131205

```
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

df['sex'] = df['sex'].replace ({'Male':1,'Female':2}) #encoding string
value to numeric values

df.head()
```

```
       rank discipline  yrs.since.phd  yrs.service  sex  salary
0      Prof          B             19           18    1  139750
1      Prof          B             20           16    1  173200
2  AsstProf          B              4            3    1   79750
3      Prof          B             45           39    1  115000
4      Prof          B             40           41    1  141500
```

```
df['discipline'].value_counts()
```

```
B    216
A    181
Name: discipline, dtype: int64
```

```
df['rank'].value_counts()
```

```
Prof         266
AsstProf      67
AssocProf     64
Name: rank, dtype: int64
```

```
df['discipline'] = df['discipline'].replace ({'B':2,'A':1}) #encoding
string value to numeric values
```

```
df['discipline'].value_counts()
```

```
2    216
1    181
Name: discipline, dtype: int64
```

```
df['rank'] = df['rank'].replace
({'Prof':3,'AsstProf':2,'AssocProf':1}) #encoding string value to
numeric values
```

```
df.head()
```

```
   rank  discipline  yrs.since.phd  yrs.service  sex  salary
0     3           2             19           18    1  139750
1     3           2             20           16    1  173200
2     2           2              4            3    1   79750
3     3           2             45           39    1  115000
4     3           2             40           41    1  141500
```

```
df.info()
```

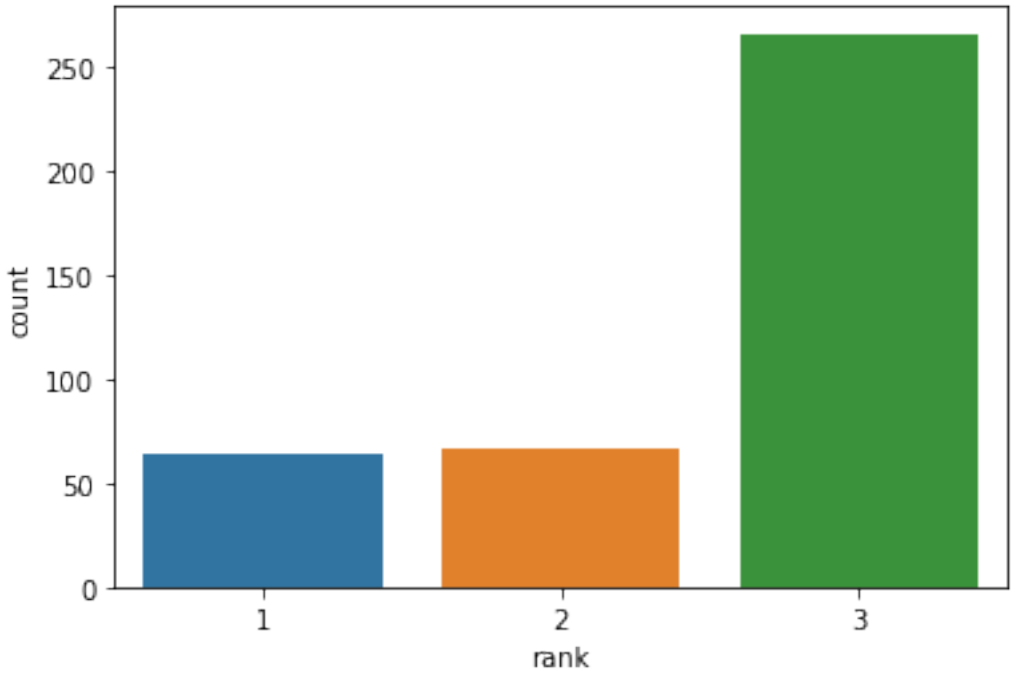
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 397 entries, 0 to 396
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   rank           397 non-null    int64
 1   discipline     397 non-null    int64
 2   yrs.since.phd  397 non-null    int64
 3   yrs.service    397 non-null    int64
```

```
 4   sex            397 non-null    int64
 5   salary         397 non-null    int64
dtypes: int64(6)
memory usage: 18.7 KB
```

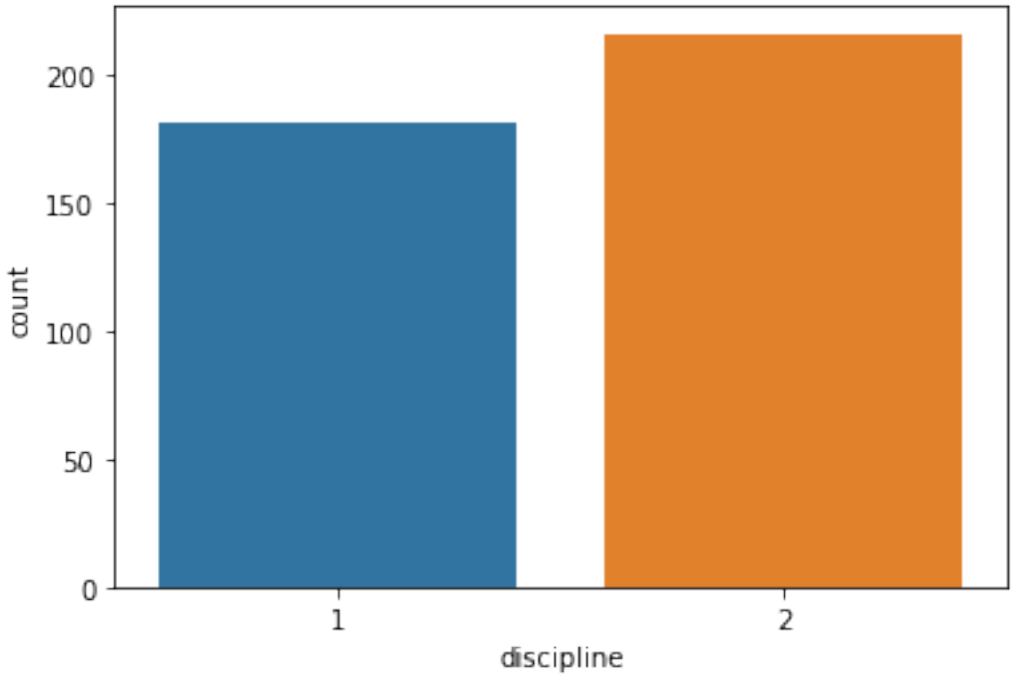
```
df.describe().transpose()
```

```
               count      mean      std      min      25%       50%
75%  \
rank           397.00      2.51     0.76     1.00     2.00      3.00
3.00
discipline     397.00      1.54     0.50     1.00     1.00      2.00
2.00
yrs.since.phd  397.00     22.31    12.89     1.00    12.00     21.00
32.00
yrs.service    397.00     17.61    13.01     0.00     7.00     16.00
27.00
sex            397.00      1.10     0.30     1.00     1.00      1.00
1.00
salary         397.00 113706.46 30289.04 57800.00 91000.00 107300.00
134185.00

                    max
rank               3.00
discipline         2.00
yrs.since.phd     56.00
yrs.service       60.00
sex                2.00
salary        231545.00
```

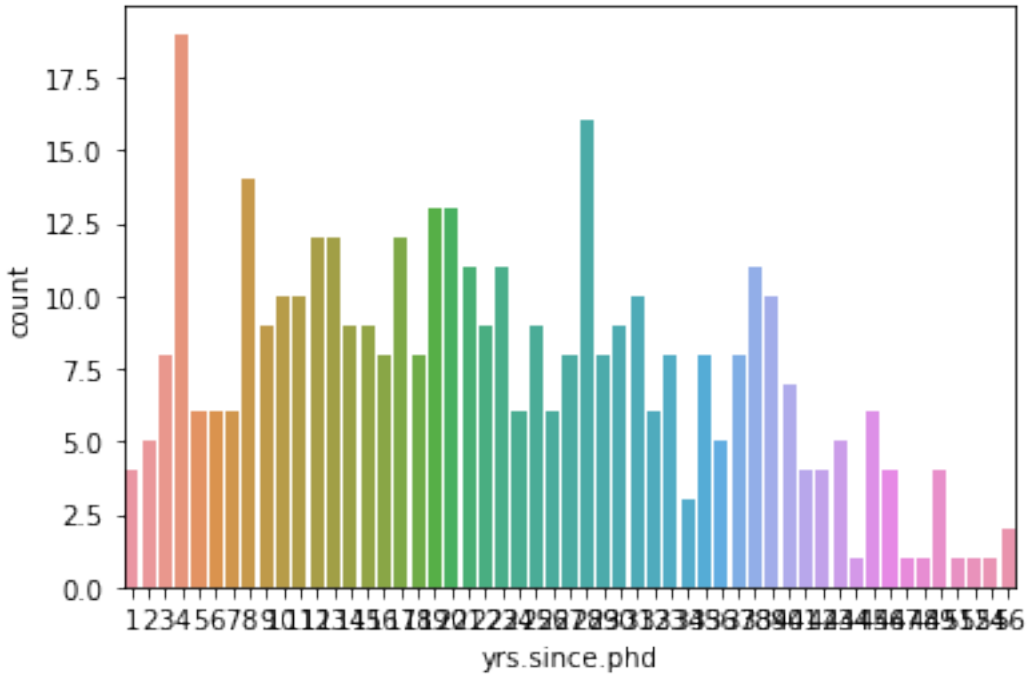
```
sns.countplot (x='rank', data = df)
plt.show()
```



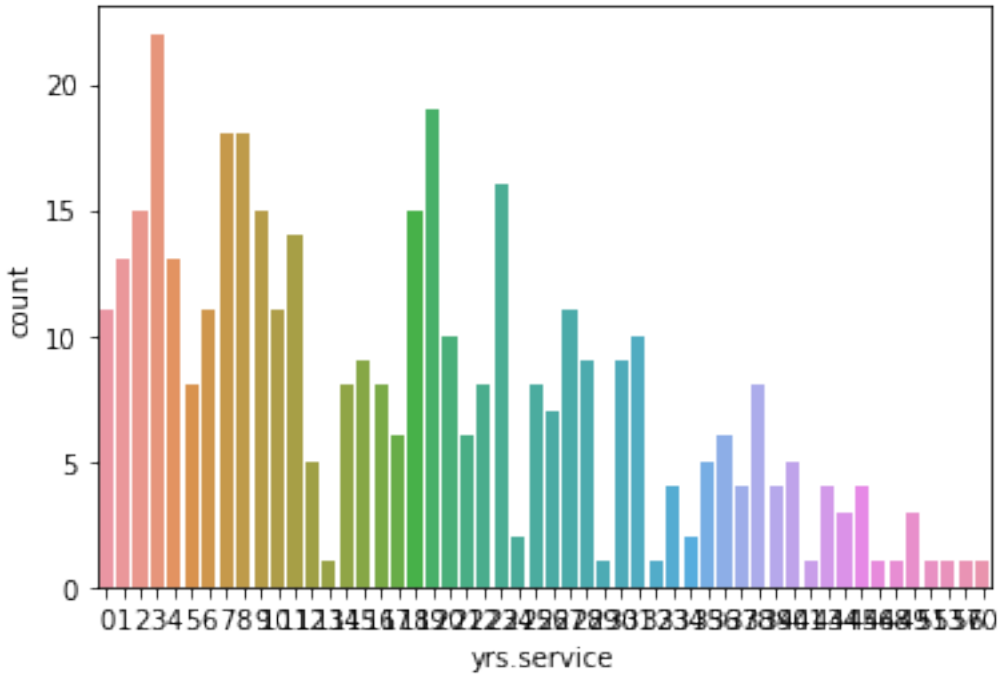

```
sns.countplot (x='discipline', data = df)
plt.show()
```



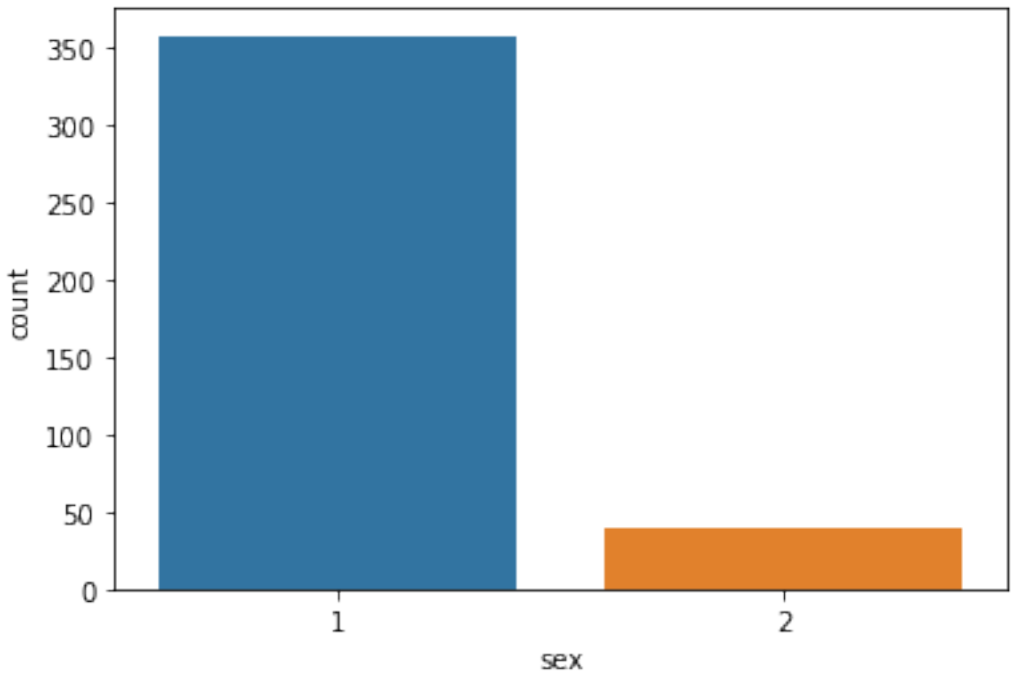

```
sns.countplot (x='yrs.since.phd', data = df)
plt.show()
```




```
sns.countplot (x='yrs.service', data = df)
plt.show()
```

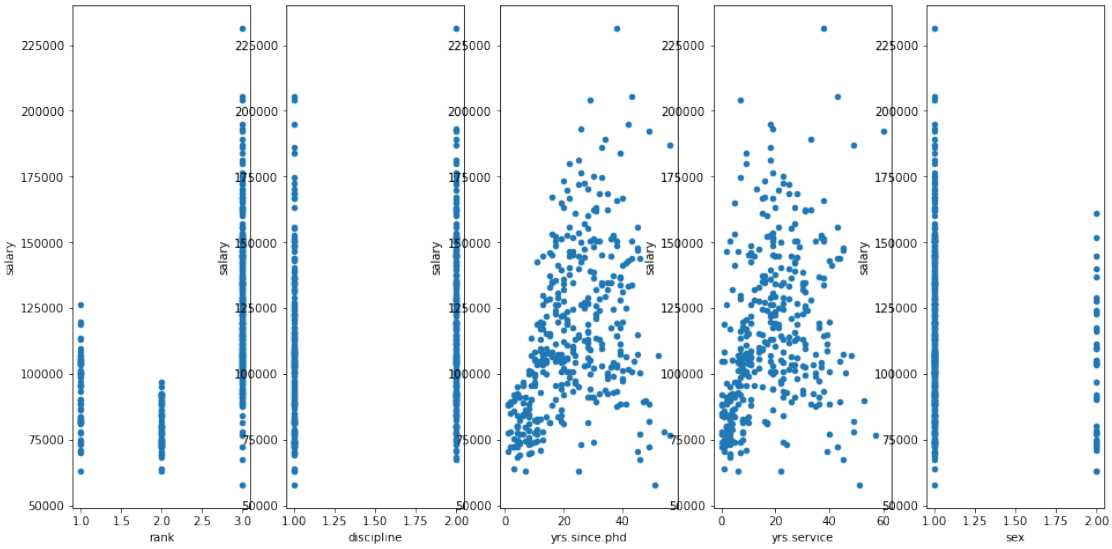



```
sns.countplot (x='sex', data = df)
plt.show()
```



NOTES : 1. Imbalance present in ranks and sex columns

```
#Checking for relationships

fig, axs = plt.subplots(1,5) #1 row and 3 columns
df.plot(kind='scatter', x ='rank', y='salary', ax=axs[0], figsize =
(16,8))
df.plot(kind='scatter', x ='discipline', y='salary', ax=axs[1])
df.plot(kind='scatter', x ='yrs.since.phd', y='salary', ax=axs[2])
df.plot(kind='scatter', x ='yrs.service', y='salary', ax=axs[3])
df.plot(kind='scatter', x ='sex', y='salary', ax=axs[4])
fig.savefig('test_name.png')
```



```
df.corr().sort_values('salary')
```

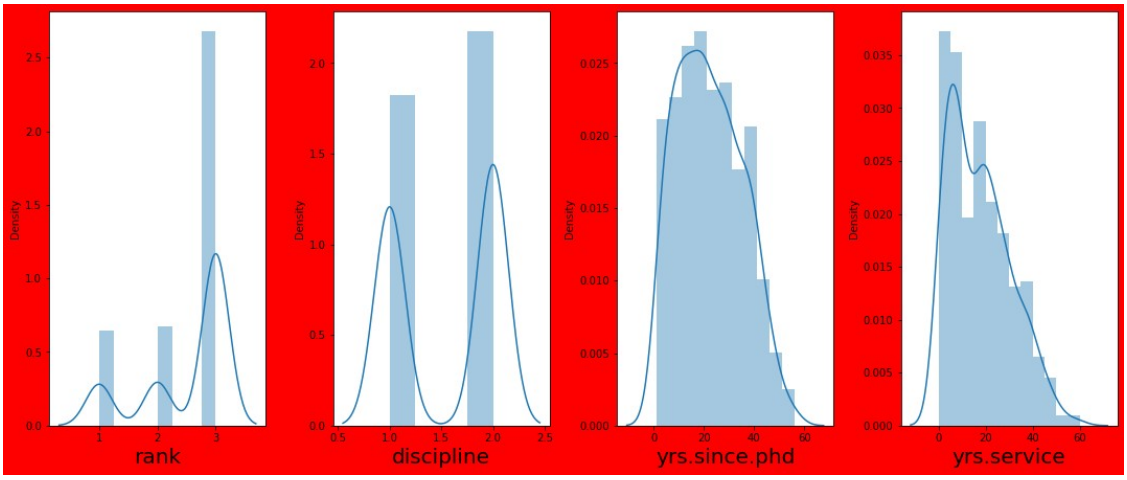
```
                rank  discipline  yrs.since.phd  yrs.service   sex
salary
sex            -0.13       -0.00          -0.15        -0.15  1.00   -
0.14
discipline     -0.09        1.00          -0.22        -0.16 -0.00
0.16
yrs.service     0.45       -0.16           0.91         1.00 -0.15
0.33
yrs.since.phd   0.53       -0.22           1.00         0.91 -0.15
0.42
rank            1.00       -0.09           0.53         0.45 -0.13
0.52
salary          0.52        0.16           0.42         0.33 -0.14
1.00
```

```
plt.figure(figsize = (12,8))
sns.heatmap(df.corr(), annot = True, linewidths=0.5, linecolor  =
"white", fmt='.2f',cbar=True) #upto 2 places of decimals
```

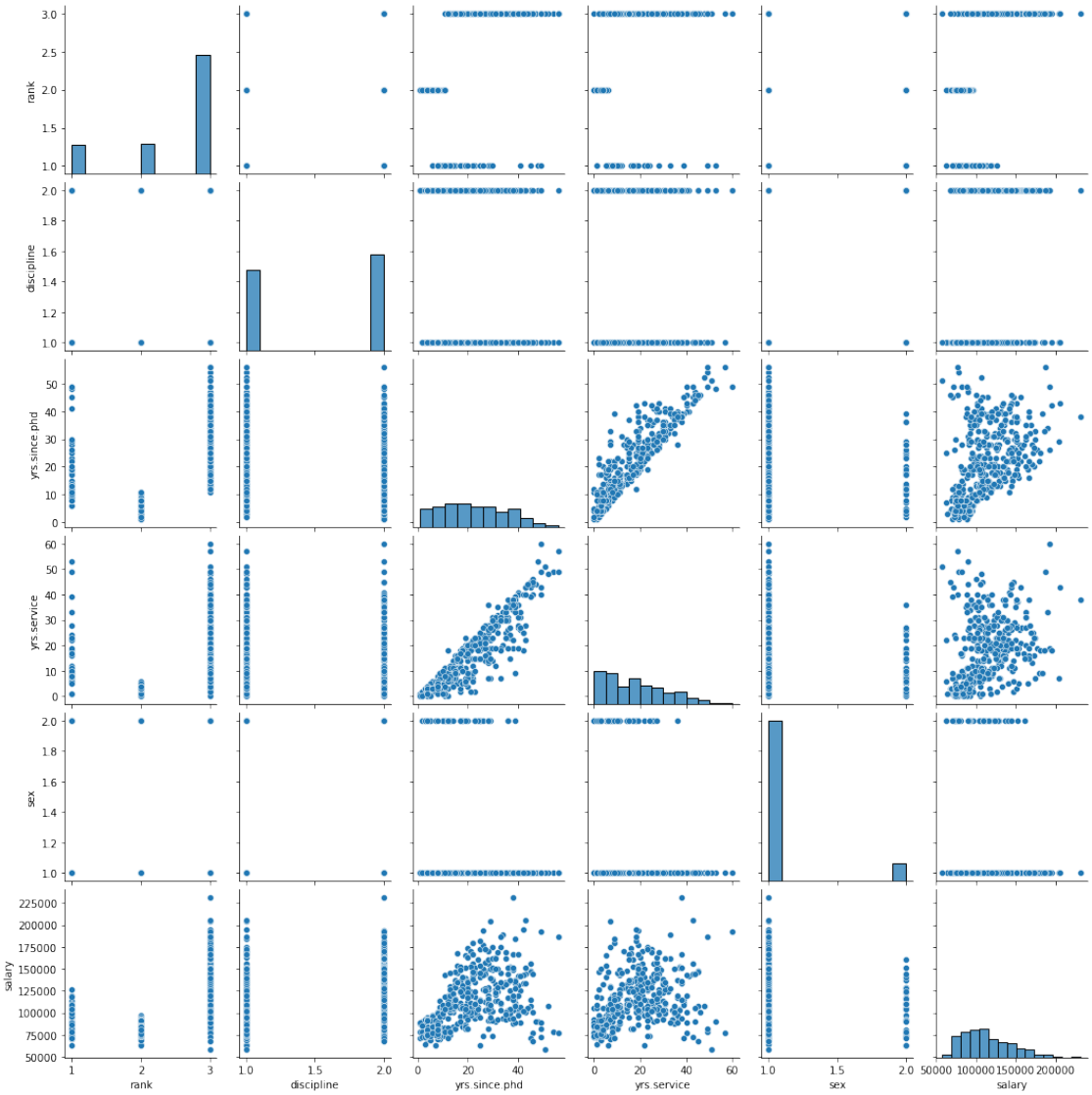
```
<AxesSubplot:>
```


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

```
#checking the ditribution
plt.figure(figsize = (15,12), facecolor = 'red')
plotnumber = 1 #this is an initiator

for column in df:
    if plotnumber <5: #here 5 is the number of features
        ax = plt.subplot (2,4, plotnumber)
        sns.distplot (df[column])
        plt.xlabel (column, fontsize =20)
    plotnumber +=1
plt.tight_layout()
```




```
#using pairplot
sns.pairplot(df)
plt.savefig ('pairplot.png')
plt.show()
```




```
df1 = df
```

```
df1
```

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

```
[397 rows x 6 columns]
```

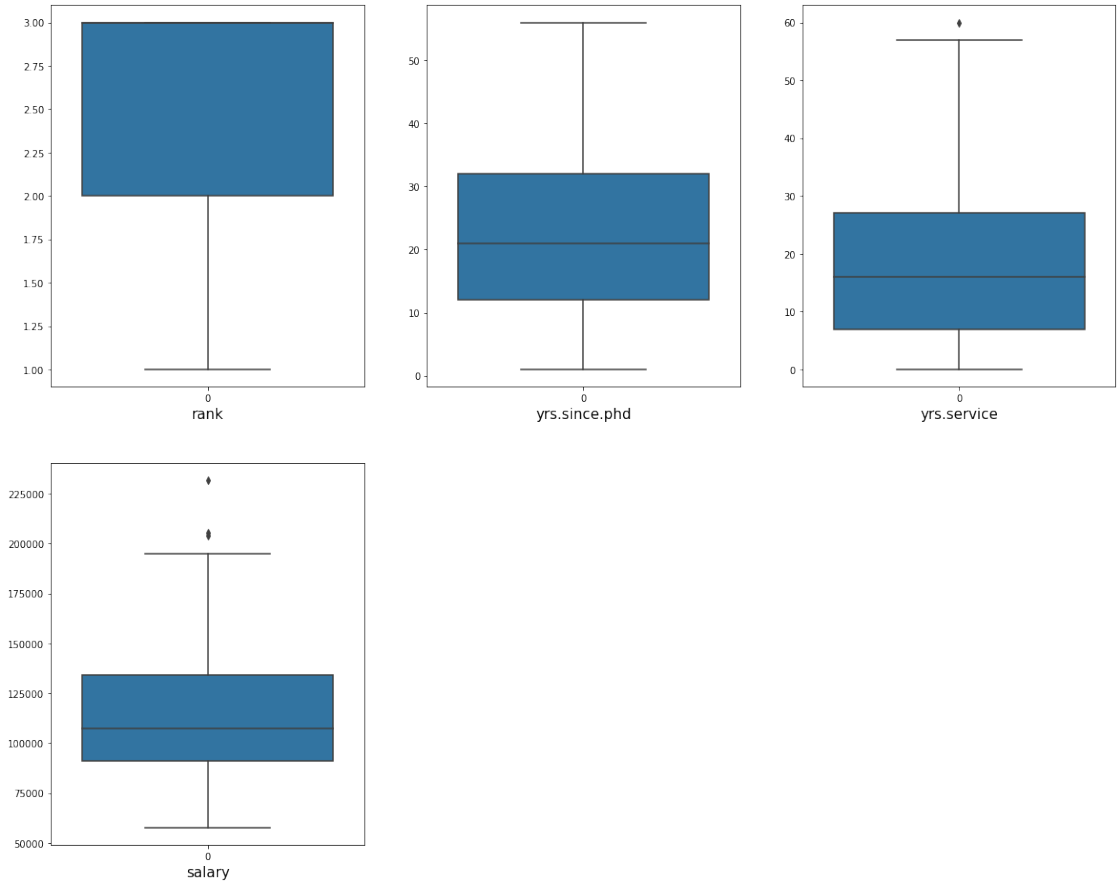
```
df1 = df1.drop (columns = ['discipline'])
df1 = df1.drop (columns = ['sex'])
```

```
df1
```

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

```
[397 rows x 4 columns]
```

```
plt.figure (figsize = (20,25))
graph = 1 #Initiator

for column in df1:
    if graph <=6: #here 8 is the number of features
        plt.subplot (3,3, graph)
        ax = sns.boxplot (data = df1[column])
        plt.xlabel (column, fontsize = 15)
    graph +=1
plt.show()
```




```
#NOTES: Outliers present in higher end of yrs.service

df1.rename(columns = {'yrs.since.phd':'yrs_since_phd',
'yrs.service':'yrs_service'}, inplace = True)

df1.head()
```

	rank	yrs_since_phd	yrs_service	salary
0	3	19	18	139750
1	3	20	16	173200
2	2	4	3	79750
3	3	45	39	115000
4	3	40	41	141500

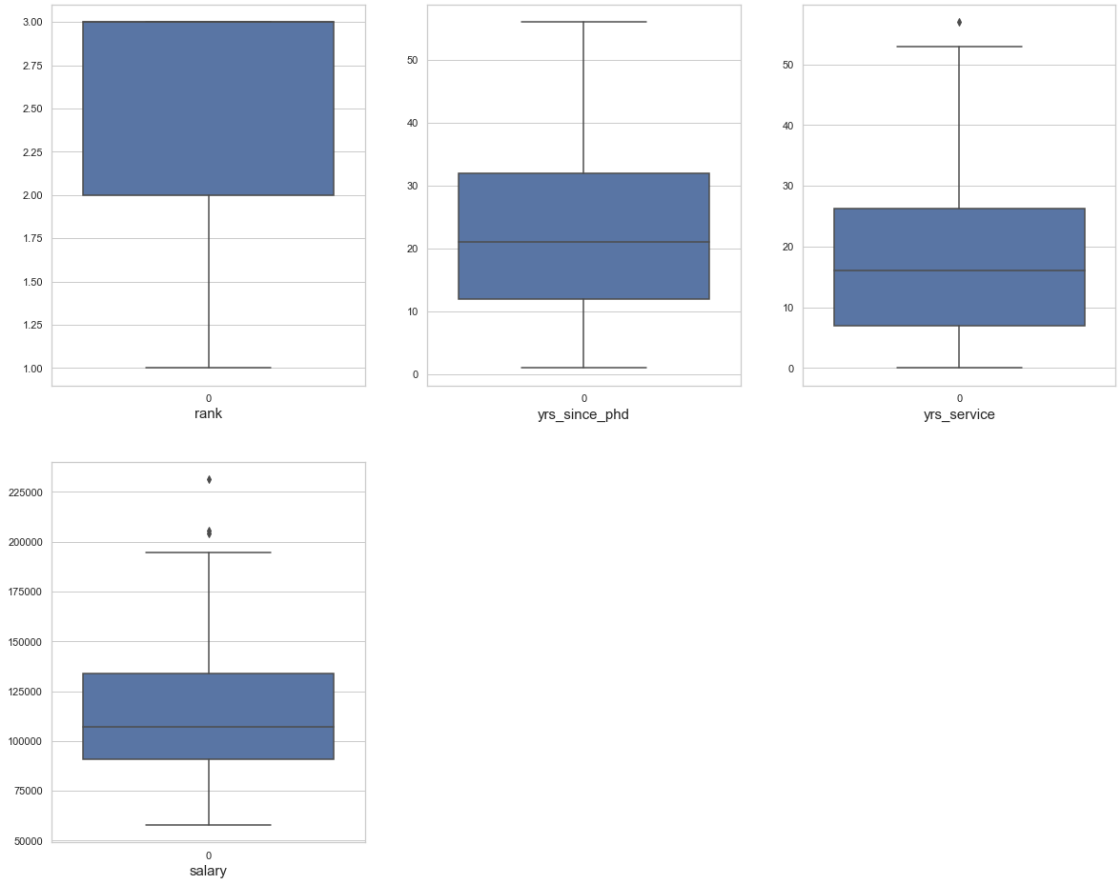
```
#removing outliers
q1 =df1.quantile (0.25) #first quantile
q3 =df1.quantile (0.75) #third quantile
iqr=q3-q1
var_high = (q3.yrs_service + (1.5 * iqr.yrs_service)) #for outliers on
the higher end of data
var_high
```

```
57.0

index_high = np.where (df1['yrs_service']>var_high)
```

```
df1 = df1.drop(df.index[index_high])
df1.shape
df1.reset_index() #to reset the dropped index
df1.shape
```

```
(396, 4)
```

```
plt.figure (figsize = (20,25))
graph = 1 #Initiator

for column in df1:
    if graph <=6: #here 8 is the number of features
        plt.subplot (3,3, graph)
        ax = sns.boxplot (data = df1[column])
        plt.xlabel (column, fontsize = 15)
    graph +=1
plt.show()
```




```
df1.describe().transpose()
```

```
                count      mean       std      min       25%       50%
75%  \
rank           396.00      2.51      0.76     1.00      2.00      3.00
3.00
```

```
yrs_since_phd 396.00     22.25    12.83     1.00    12.00     21.00
32.00
yrs_service   396.00     17.51    12.85     0.00     7.00     16.00
26.25
salary        396.00 113508.11 30068.09 57800.00 91000.00 107250.00
134046.25

                    max
rank               3.00
yrs_since_phd     56.00
yrs_service       57.00
salary        231545.00

df1.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 396 entries, 0 to 396
Data columns (total 4 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   rank           396 non-null    int64
 1   yrs_since_phd  396 non-null    int64
 2   yrs_service    396 non-null    int64
 3   salary         396 non-null    int64
dtypes: int64(4)
memory usage: 35.5 KB

#separating features and labels
x = df1.drop(columns = ['salary'])
y = df1['salary']

#Standardization of features (standard scaler)

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
x_scaled

array([[ 0.65042291, -0.25337508,  0.03838073],
       [ 0.65042291, -0.17535289, -0.11750409],
       [-0.67043593, -1.42370785, -1.13075543],
       ...,
       [ 0.65042291,  1.54113518,  0.58397761],
       [ 0.65042291,  0.21475803, -0.1954465 ],
       [-0.67043593, -1.11161911, -1.05281302]])

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split
(x_scaled,y,test_size = 0.25, random_state = 348)
y_test.head()
```

```
275     93000
282     57800
281    103600
23     113068
88     172272
Name: salary, dtype: int64
```

```
#Model Training
from sklearn.linear_model import LinearRegression
reg = LinearRegression()
reg.fit(x_train,y_train)
```

```
LinearRegression()
```

```
#Accuracy of the model
```

```
reg.score(x_train, y_train) #training score
```

```
0.2986666020857056
```

```
reg.score(x_test, y_test) #testing score
```

```
0.3321731540977363
```

```
y_pred = reg.predict(x_test)
```

```
y_pred
```

```
array([119529.44159448, 129928.0882534 , 129902.50956507,
123128.82868314,
       121105.67695948, 125890.81852518,  99592.33217157,
121711.34354216,
        84259.79102707, 102972.77794045,  96817.55298537,
140031.05752755,
        99253.00866027,  92222.9820111 , 120560.20147256,
96817.55298537,
       101482.31235955, 123068.63758738, 133622.27884526,
97290.04803236,
       120074.9170814 , 128691.1763997 , 130907.69075482,
118863.58391603,
       122802.294516  , 130362.2152679 , 130447.98505199,
123407.96109868,
       118730.41238034, 130727.11746753, 123893.24548984,
128811.55859123,
        81957.50688787, 100197.99875425, 122510.37275629,
85883.42814367,
        97290.04803236,  82430.00193486, 125370.92172658,
121226.059151  ,
       124486.12272836,  96138.90596276, 100537.32226556,
121492.40222238,
       132084.41151275,  86501.88407052, 128837.13727956,
134906.59245055,
```

```
       121299.03959093,  88185.71228288, 127321.09301032,
138214.05777951,
       124765.2551439 , 121359.23068669,  97083.89605675,
123249.21087467,
       118730.41238034,  81351.84030519, 119808.57401002,
123601.32373013,
       127926.759593  , 120074.9170814 ,  81351.84030519,
134932.17113888,
        83447.97246878,  96478.22947407, 120414.2405927 ,
125504.09326227,
       118923.7750118 , 126328.70116474, 100537.32226556,
98101.86659066,
       118524.26040473, 101215.96928817, 119056.94654748,
132870.65138272,
        96402.45699412, 126788.40686757, 123614.1130743 ,
127892.14718557,
        96950.72452106, 122995.65714745, 125031.59821528,
126109.75984496,
        86162.56055921, 119735.59357009,  84599.11453838,
118184.93689342,
       125856.20611774, 127819.16674564,  98101.86659066,
98235.03812635,
        84259.79102707, 125504.09326227,  82296.83039917,
83447.97246878,
       118597.24084465, 117918.59382205, 128085.50981702])
```

```
from sklearn.metrics import r2_score
y_pred = reg.predict(x_test)
r2_score(y_test, y_pred)*100
```

```
33.217315409773626
```

We got accuracy score of 33.21%

Trying out lazypredict

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

```
!pip install pathlib
```

```
Requirement already satisfied: pathlib in c:\users\admin\anaconda3\
lib\site-packages (1.0.1)
```

```
import lazypredict
```

```
from lazypredict.Supervised import LazyRegressor
```

```
reg = LazyRegressor(verbose=0,ignore_warnings=False,
custom_metric=None)
```

```
models_train,predictions_train = reg.fit(x_train, x_train, y_train,
y_train)

models_test,predictions_test = reg.fit(x_train, x_test, y_train,
y_test)
```

```
100%|██████████| 42/42 [00:01<00:00, 32.12it/s]
100%|██████████| 42/42 [00:00<?, ?it/s]
```

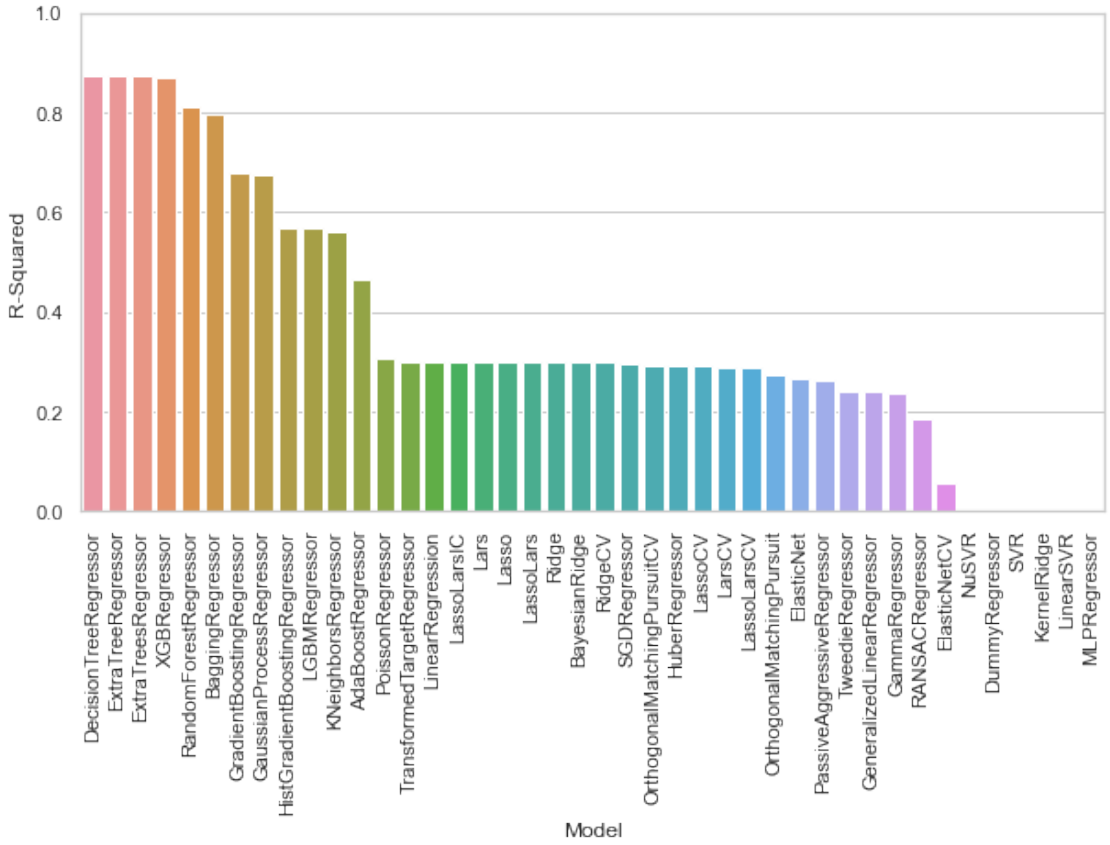
```
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
```

```
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
tuple model failed to execute
'tuple' object is not callable
```

```
plt.figure(figsize=(10, 5))
sns.set_theme(style="whitegrid")
ax = sns.barplot(x=models_train.index, y="R-Squared",
data=models_train)
```

```
ax.set(ylim=(0, 1))
plt.xticks(rotation=90)
```

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14,
15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31,
32, 33,
        34, 35, 36, 37, 38, 39, 40, 41]),
 [Text(0, 0, 'DecisionTreeRegressor'),
  Text(1, 0, 'ExtraTreeRegressor'),
  Text(2, 0, 'ExtraTreesRegressor'),
  Text(3, 0, 'XGBRegressor'),
  Text(4, 0, 'RandomForestRegressor'),
  Text(5, 0, 'BaggingRegressor'),
  Text(6, 0, 'GradientBoostingRegressor'),
  Text(7, 0, 'GaussianProcessRegressor'),
  Text(8, 0, 'HistGradientBoostingRegressor'),
  Text(9, 0, 'LGBMRegressor'),
  Text(10, 0, 'KNeighborsRegressor'),
  Text(11, 0, 'AdaBoostRegressor'),
  Text(12, 0, 'PoissonRegressor'),
  Text(13, 0, 'TransformedTargetRegressor'),
  Text(14, 0, 'LinearRegression'),
  Text(15, 0, 'LassoLarsIC'),
  Text(16, 0, 'Lars'),
  Text(17, 0, 'Lasso'),
  Text(18, 0, 'LassoLars'),
  Text(19, 0, 'Ridge'),
  Text(20, 0, 'BayesianRidge'),
  Text(21, 0, 'RidgeCV'),
  Text(22, 0, 'SGDRegressor'),
  Text(23, 0, 'OrthogonalMatchingPursuitCV'),
  Text(24, 0, 'HuberRegressor'),
  Text(25, 0, 'LassoCV'),
  Text(26, 0, 'LarsCV'),
  Text(27, 0, 'LassoLarsCV'),
  Text(28, 0, 'OrthogonalMatchingPursuit'),
  Text(29, 0, 'ElasticNet'),
  Text(30, 0, 'PassiveAggressiveRegressor'),
  Text(31, 0, 'TweedieRegressor'),
  Text(32, 0, 'GeneralizedLinearRegressor'),
  Text(33, 0, 'GammaRegressor'),
  Text(34, 0, 'RANSACRegressor'),
  Text(35, 0, 'ElasticNetCV'),
  Text(36, 0, 'NuSVR'),
  Text(37, 0, 'DummyRegressor'),
  Text(38, 0, 'SVR'),
  Text(39, 0, 'KernelRidge'),
  Text(40, 0, 'LinearSVR'),
  Text(41, 0, 'MLPRegressor')])
```




```
models_train
```

Model	Adjusted R-Squared	R-Squared	RMSE
DecisionTreeRegressor	0.87	0.87	10716.23
ExtraTreeRegressor	0.87	0.87	10716.23
ExtraTreesRegressor	0.87	0.87	10716.23
XGBRegressor	0.87	0.87	10796.56
RandomForestRegressor	0.81	0.81	13025.36
BaggingRegressor	0.79	0.80	13556.95
GradientBoostingRegressor	0.67	0.68	17019.82
GaussianProcessRegressor	0.67	0.67	17119.11
HistGradientBoostingRegressor	0.57	0.57	19654.56

LGBMRegressor	0.56	0.57	19693.96
KNeighborsRegressor	0.56	0.56	19849.33
AdaBoostRegressor	0.46	0.46	21953.33
PoissonRegressor	0.30	0.31	24971.02
TransformedTargetRegressor	0.29	0.30	25086.33
LinearRegression	0.29	0.30	25086.33
LassoLarsIC	0.29	0.30	25086.33
Lars	0.29	0.30	25086.33
Lasso	0.29	0.30	25086.33
LassoLars	0.29	0.30	25086.46
Ridge	0.29	0.30	25086.64
BayesianRidge	0.29	0.30	25096.46
RidgeCV	0.29	0.30	25105.50
SGDRegressor	0.29	0.30	25112.17
OrthogonalMatchingPursuitCV	0.28	0.29	25215.45
HuberRegressor	0.28	0.29	25223.37
LassoCV	0.28	0.29	25239.30
LarsCV	0.28	0.29	25244.94
LassoLarsCV	0.28	0.29	25244.94
OrthogonalMatchingPursuit	0.27	0.27	25510.65
ElasticNet	0.26	0.27	25633.28
PassiveAggressiveRegressor	0.25	0.26	25739.95
TweedieRegressor	0.23	0.24	26130.14
GeneralizedLinearRegressor	0.23	0.24	26130.14

GammaRegressor	0.23	0.24	26165.76
RANSACRegressor	0.18	0.19	27017.61
ElasticNetCV	0.05	0.06	29115.51
NuSVR	-0.01	0.00	29952.07
DummyRegressor	-0.01	0.00	29955.39
SVR	-0.06	-0.05	30632.88
KernelRidge	-14.36	-14.21	116813.78
LinearSVR	-14.59	-14.43	117668.02
MLPRegressor	-14.65	-14.49	117911.22

Model	Time Taken
DecisionTreeRegressor	0.01
ExtraTreeRegressor	0.01
ExtraTreesRegressor	0.10
XGBRegressor	0.05
RandomForestRegressor	0.13
BaggingRegressor	0.02
GradientBoostingRegressor	0.03
GaussianProcessRegressor	0.02
HistGradientBoostingRegressor	0.14
LGBMRegressor	0.03
KNeighborsRegressor	0.01
AdaBoostRegressor	0.02
PoissonRegressor	0.01
TransformedTargetRegressor	0.01
LinearRegression	0.01
LassoLarsIC	0.00
Lars	0.01
Lasso	0.01
LassoLars	0.01
Ridge	0.02
BayesianRidge	0.01
RidgeCV	0.01
SGDRegressor	0.01
OrthogonalMatchingPursuitCV	0.02
HuberRegressor	0.02
LassoCV	0.05
LarsCV	0.02
LassoLarsCV	0.02

OrthogonalMatchingPursuit	0.01
ElasticNet	0.01
PassiveAggressiveRegressor	0.02
TweedieRegressor	0.01
GeneralizedLinearRegressor	0.02
GammaRegressor	0.00
RANSACRegressor	0.03
ElasticNetCV	0.04
NuSVR	0.02
DummyRegressor	0.01
SVR	0.01
KernelRidge	0.01
LinearSVR	0.01
MLPRegressor	0.35

Thank You