Task:

Understand Lasso and Ridge regression using the shared book ISLR and apply Lasso and Ridge regression on given dataset for finding the salary of the player.

The data

The given data is in .xlsx format. Its various columns are the parameters that describe a player and which affect the target variable(Salary). The rows are unique records for different players with values for each parameter as well as their Salary.

Approach

1. Check the data to see if it needs any cleaning and preprocessing followed by visualization of the data
2. Perform some sort of dimensionality reduction, maybe by using the correlation matrix.
3. Train both the regressors for various values of alpha(tuning parameter) to find the best value of alpha. We then use these models to perform the predictions on the missing values.
4. Checking the R2 scores and the Mean Squared Error for each case to find out the improvement(if any).

Importing the required libraries

```
# -*- coding: utf-8 -*-
"""Untitled0.ipynb


Automatically generated by Colaboratory

Original file is located at

    https://colab.research.google.com/drive/1-eHqIVm9tVt-tKG6qxWln58Xq4VIpfHH

# Commented out IPython magic to ensure Python compatibility.

#For data manipulation and preprocessing

import pandas as pd

import numpy as np

from scipy.stats import norm

# For data visualisation
```

```
import matplotlib.pyplot as plt
import seaborn as sns
# %matplotlib inline
# For Regression
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.linear_model import LinearRegression
from sklearn.utils.testing import ignore_warnings
from sklearn.exceptions import ConvergenceWarning
from sklearn.linear_model import Lasso
# Metrics for the models
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
# For importing data
from google.colab import files
files.upload()
```

Reading and Exploring the data

```
[ ] data = pd.read_excel('Hitters.xlsx')
```

```
data
```

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	CRuns	CRBI	CWalks	League	Division	PutOuts	Assists	Errors	Salary	NewLeague
0	293	66	1	30	29	14	1	293	66	1	30	29	14	A	E	446	33	20	NaN	A
1	315	81	7	24	38	39	14	3449	835	69	321	414	375	N	W	632	43	10	475.0	N
2	479	130	18	66	72	76	3	1624	457	63	224	266	263	A	W	880	82	14	480.0	A
3	496	141	20	65	78	37	11	5628	1575	225	828	838	354	N	E	200	11	3	500.0	N
4	321	87	10	39	42	30	2	396	101	12	48	46	33	N	E	805	40	4	91.5	N
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
317	497	127	7	65	48	37	5	2703	806	32	379	311	138	N	E	325	9	3	700.0	N
318	492	136	5	76	50	94	12	5511	1511	39	897	451	875	A	E	313	381	20	875.0	A
319	475	126	3	61	43	52	6	1700	433	7	217	93	146	A	W	37	113	7	385.0	A
320	573	144	9	85	60	78	8	3198	857	97	470	420	332	A	E	1314	131	12	960.0	A
321	631	170	9	77	44	31	11	4908	1457	30	775	357	249	A	W	408	4	3	1000.0	A

322 rows × 20 columns

```
"""Reading and Exploring the data"""


data = pd.read_excel('Hitters.xlsx')

data

"""Shape of data"""

data.shape

"""Checking for Null values"""

data.isnull().sum()
```

Shape of data

```
[ ] data.shape
```

```
(322, 20)
```

Checking for Null values

```
[ ] data.isnull().sum()
```

```
AtBat         0
Hits          0
HmRun         0
Runs          0
RBI           0
Walks         0
Years         0
CAtBat        0
CHits         0
CHmRun        0
CRuns         0
CRBI          0
CWalks        0
League        0
Division      0
PutOuts       0
Assists       0
Errors        0
Salary       59
NewLeague     0
dtype: int64
```

```
"""Creating test dataset"""

test = data[data['Salary'].isnull()]

test.shape
```

Creating test dataset

```
[ ] test = data[data['Salary'].isnull()]
```

```
[ ] test.shape
```

```
(59, 20)
```

Creating training dataset

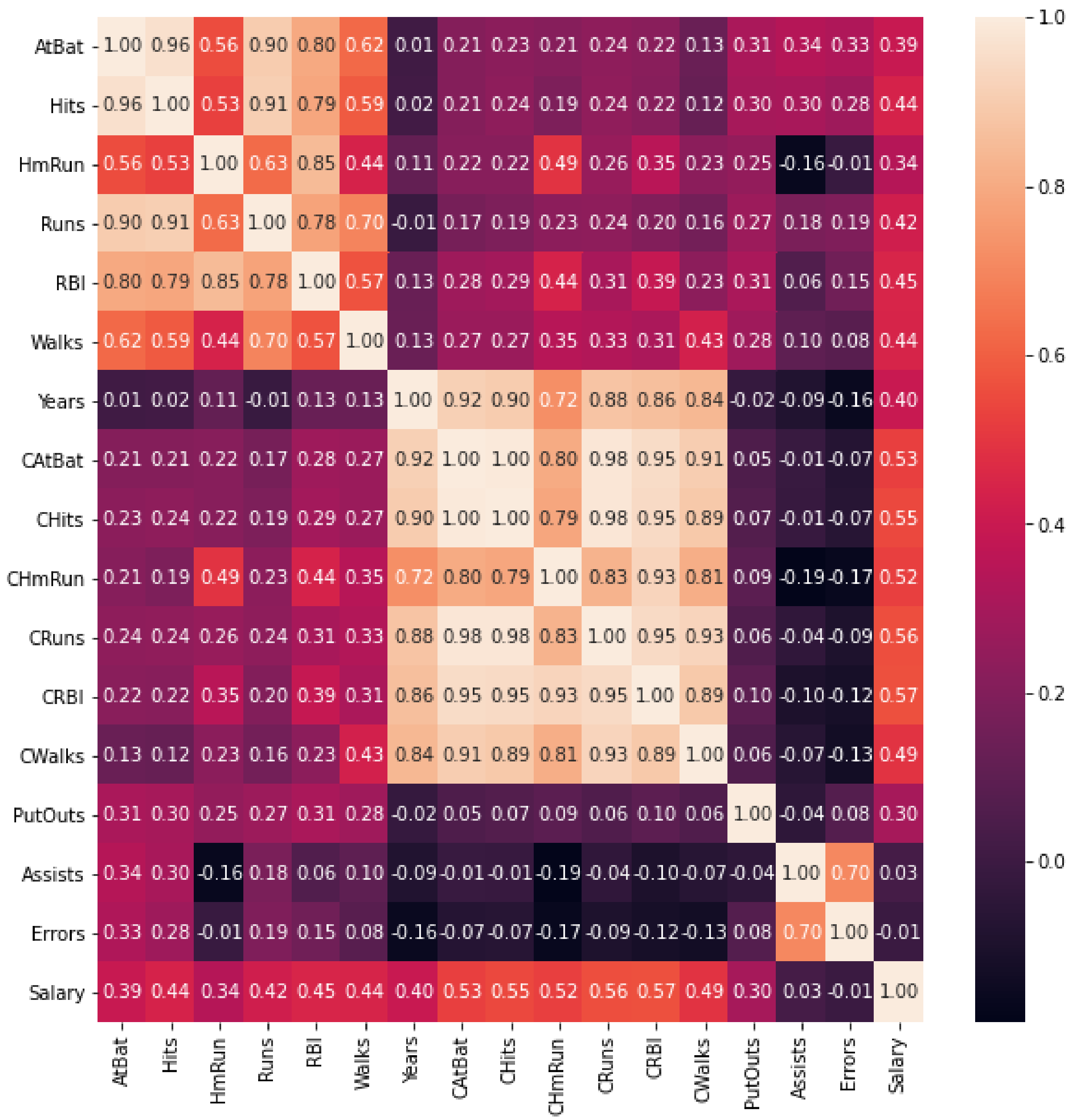
```
[ ] data = data[data['Salary'].notnull()]
```

```
[ ] data.shape
```

```
(263, 20)
```

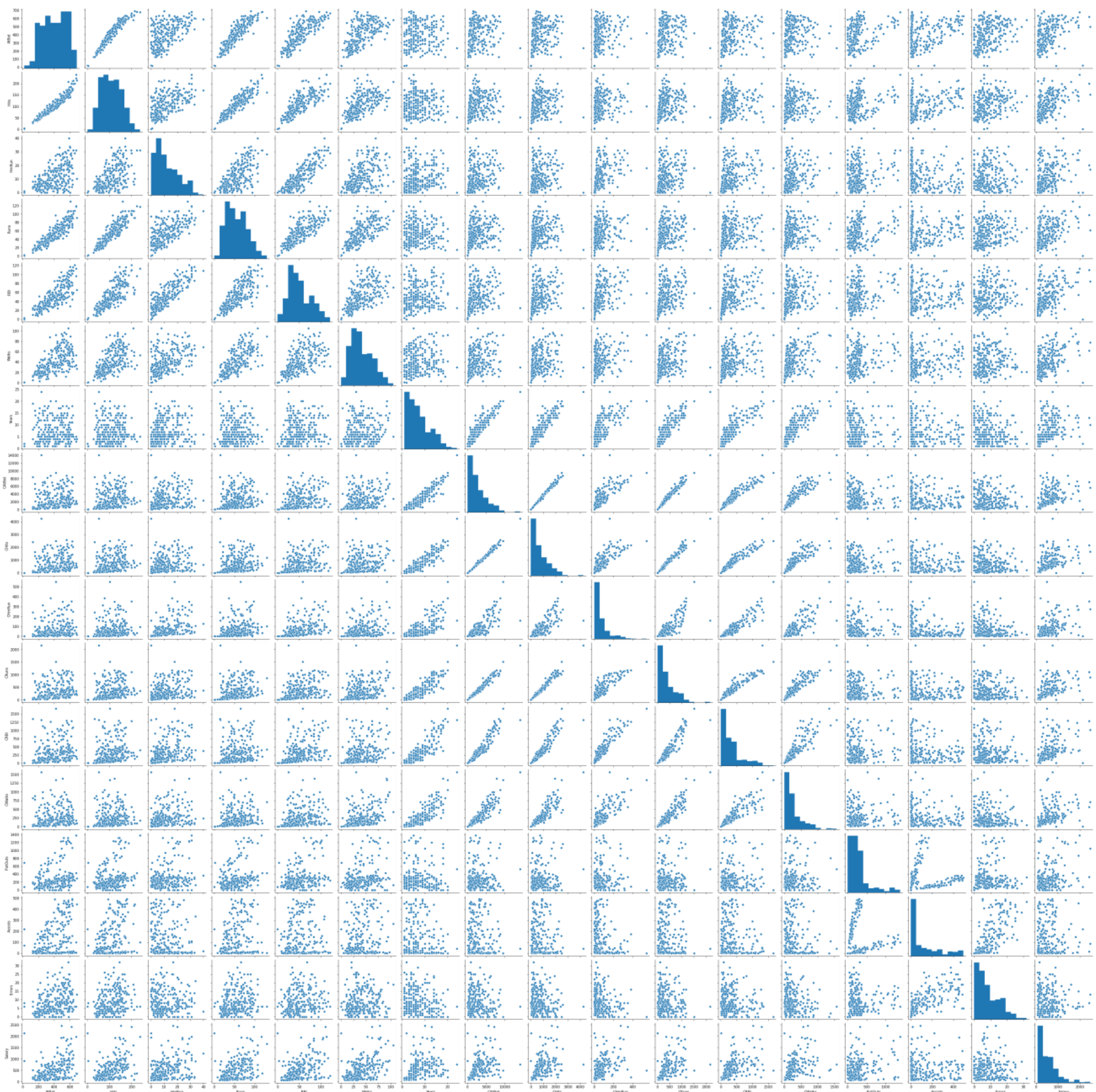
```
"""Creating training dataset"""

data = data[data['Salary'].notnull()]

data.shape

"""**Visualization**

Visualizing the correlation among different features
using a heatmap
"""
```

```
cor_mat = data.corr()

plt.figure(figsize=(10,10))

sns.heatmap(cor_mat, annot=True, fmt='.2f')
```



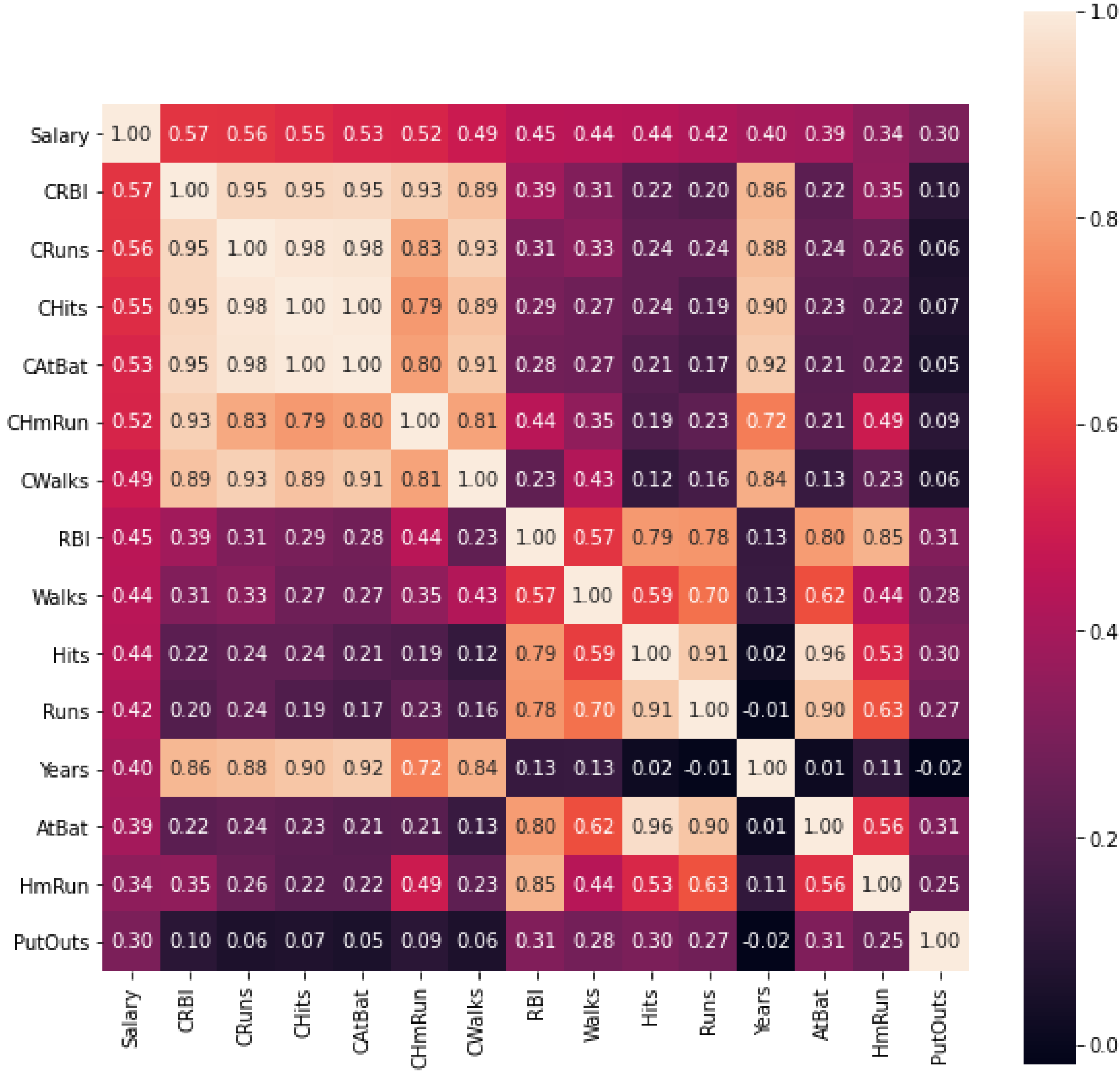
```
"""Pairplots for all features"""
```

```
sns.pairplot(data)
```

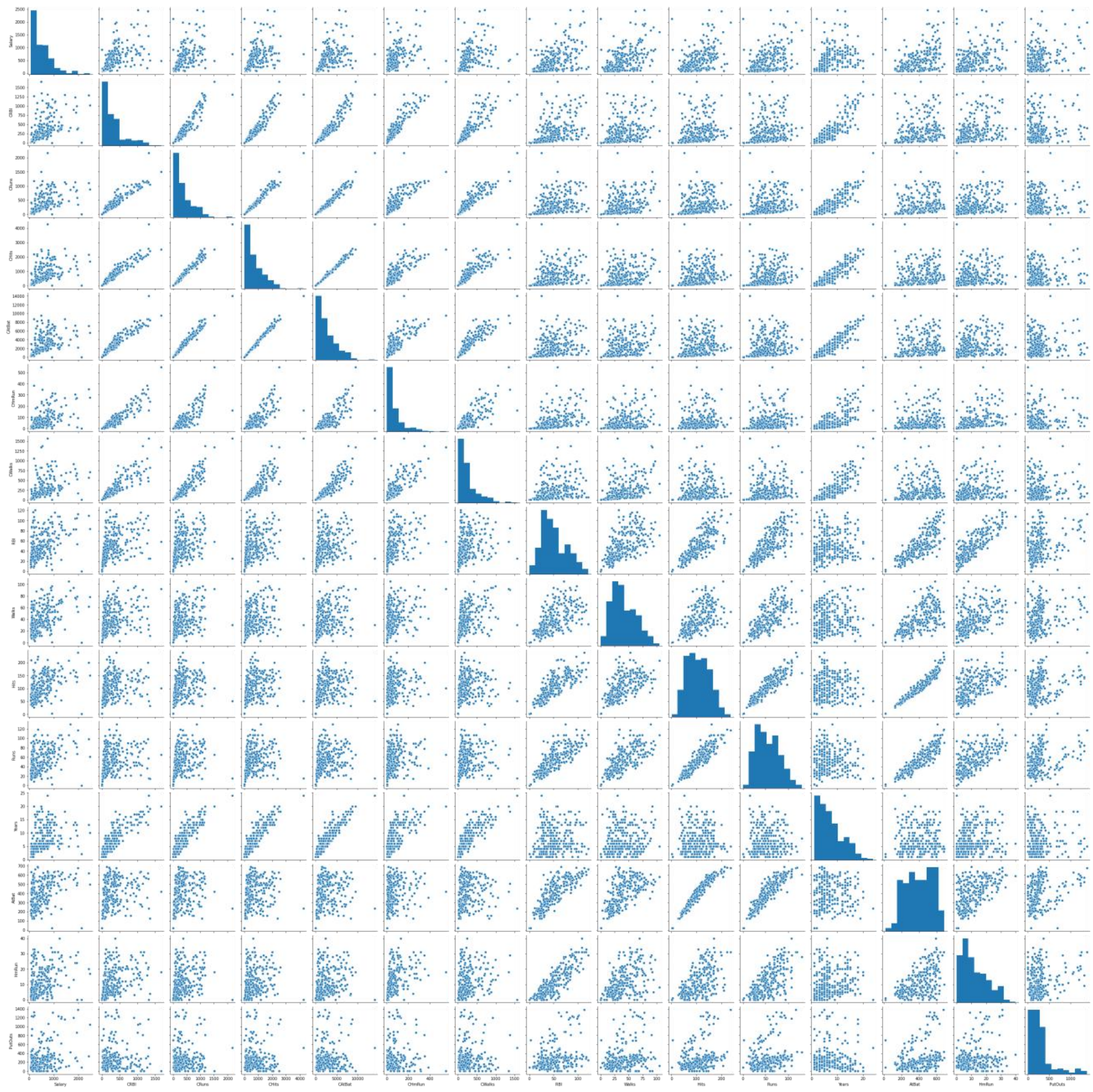


```
"""Selecting the features that highly affect the target
variable and creating a heatmap for them"""
```

```
feat = 15

fields = cor_mat.nlargest(feat,
'Salary')['Salary'].index

cm = np.corrcoef(data[fields].values.T)

plt.figure(figsize= (10, 10))

hm = sns.heatmap(cm, annot = True, square = True, fmt =
'.2f', yticklabels=fields.values

, xticklabels=fields.values)

plt.show()
```




```
"""Pairplots for the most important features"""

sns.pairplot(data[fields])
```



```
field = list(fields)

field.remove('Salary')
```

```
print(field)
```

```
[ ] field = list(fields)
    field.remove('Salary')
    print(field)
```

```
['CRBI', 'CRuns', 'CHits', 'CAtBat', 'CHmRun', 'CWalks', 'RBI', 'Walks', 'Hits', 'Runs', 'Years', 'AtBat', 'HmRun', 'PutOuts']
```

Exploring the output

```
[ ] y = data['Salary'].values
    y.shape
    pd.DataFrame(y).describe()
```

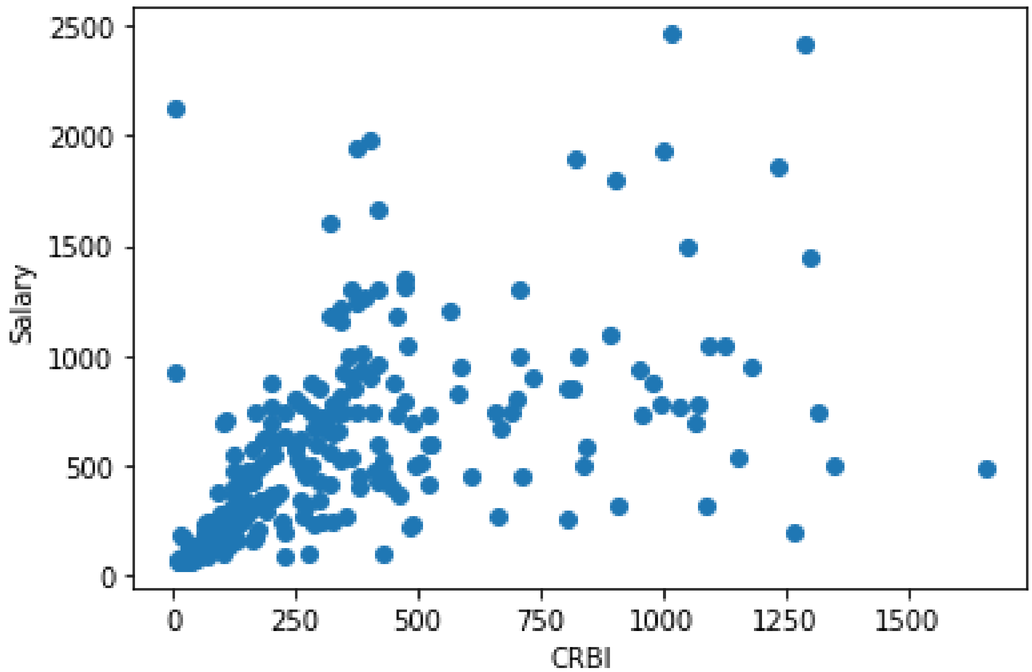
	0
count	263.000000
mean	535.925882
std	451.118681
min	67.500000
25%	190.000000
50%	425.000000
75%	750.000000
max	2460.000000

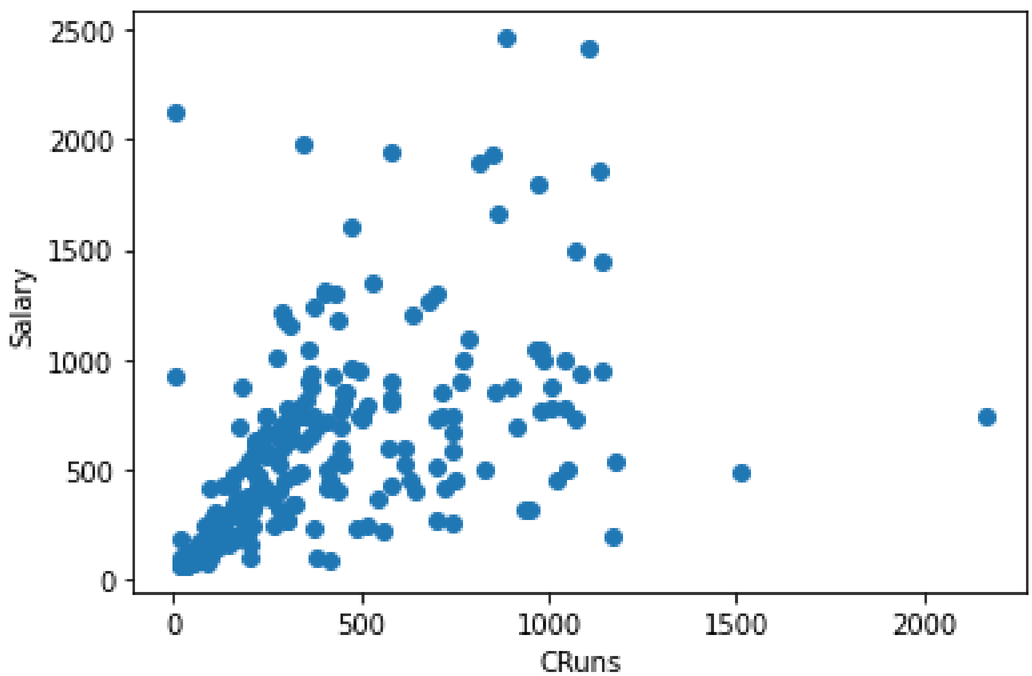
```
"""Exploring the output"""


y = data['Salary'].values

y.shape

pd.DataFrame(y).describe()


x = data[field]

x.shape
```

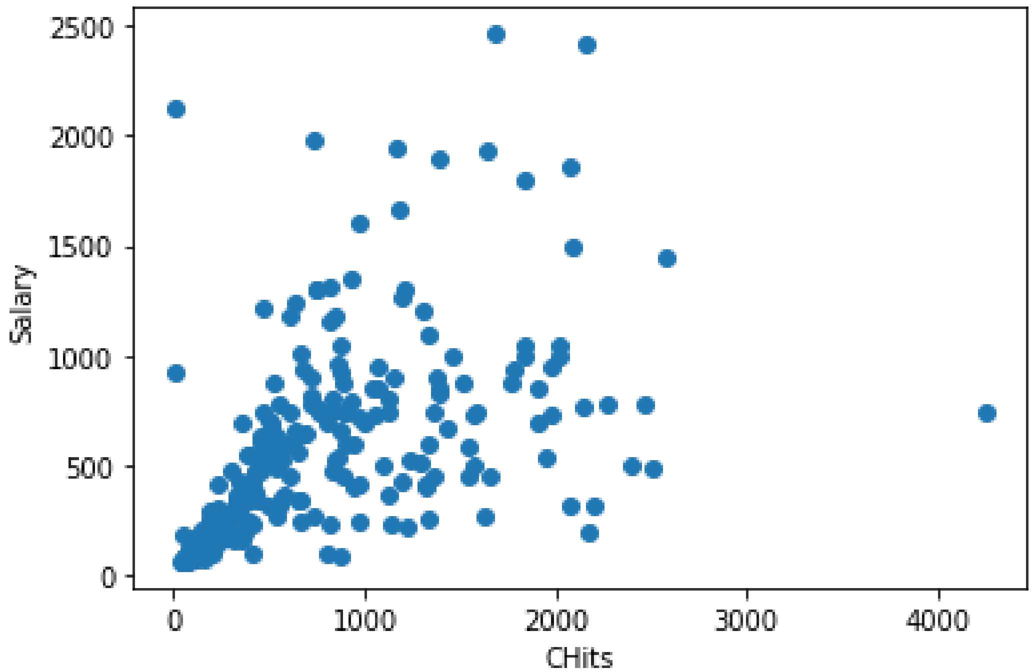
```
[ ] x = data[field]
    x.shape
```

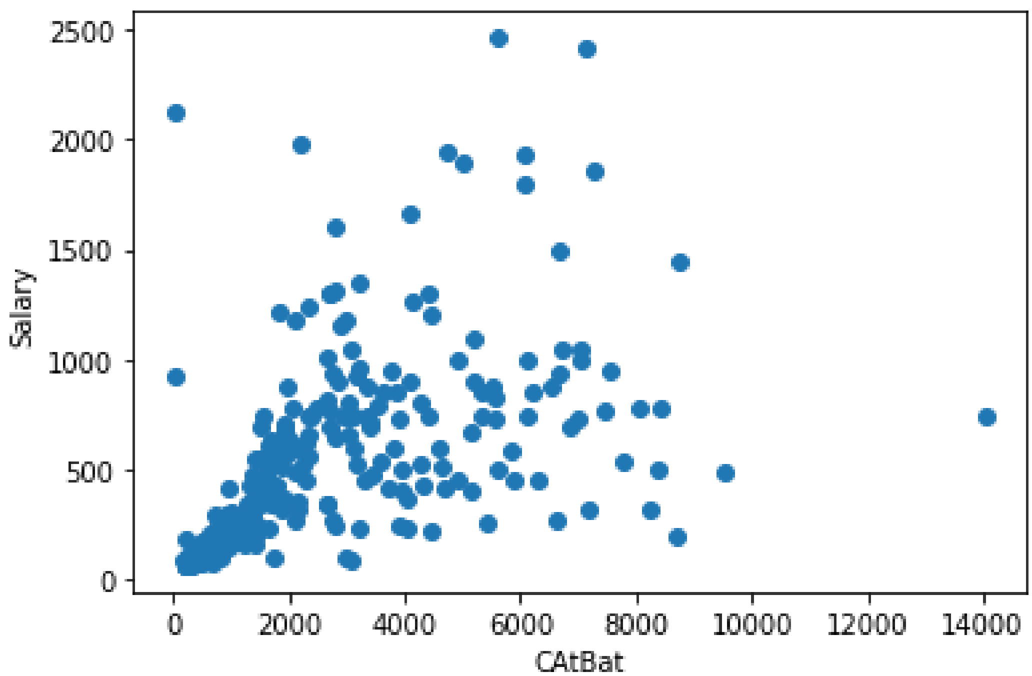
```
(263, 14)
```

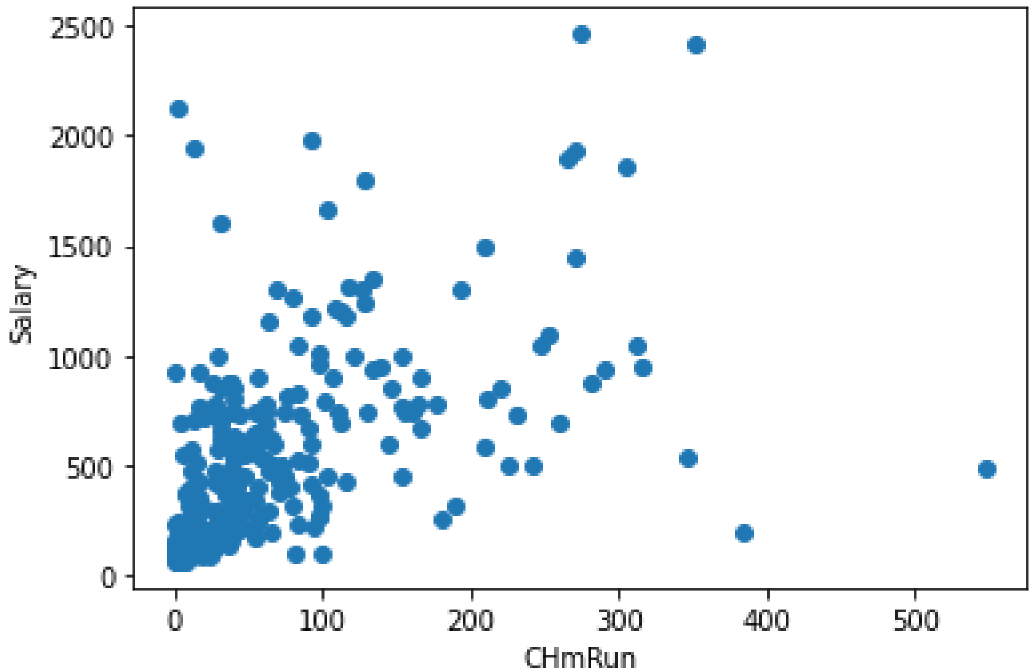
```
"""Scatterplots for the features vs Salary"""

for col in x.columns:
  plt.scatter(x[col][::], y[::])
  plt.xlabel(col)
  plt.ylabel('Salary')
  plt.show()
```


Salary
CRBI
2500
2000
1500
1000
500
0
250
750
1250

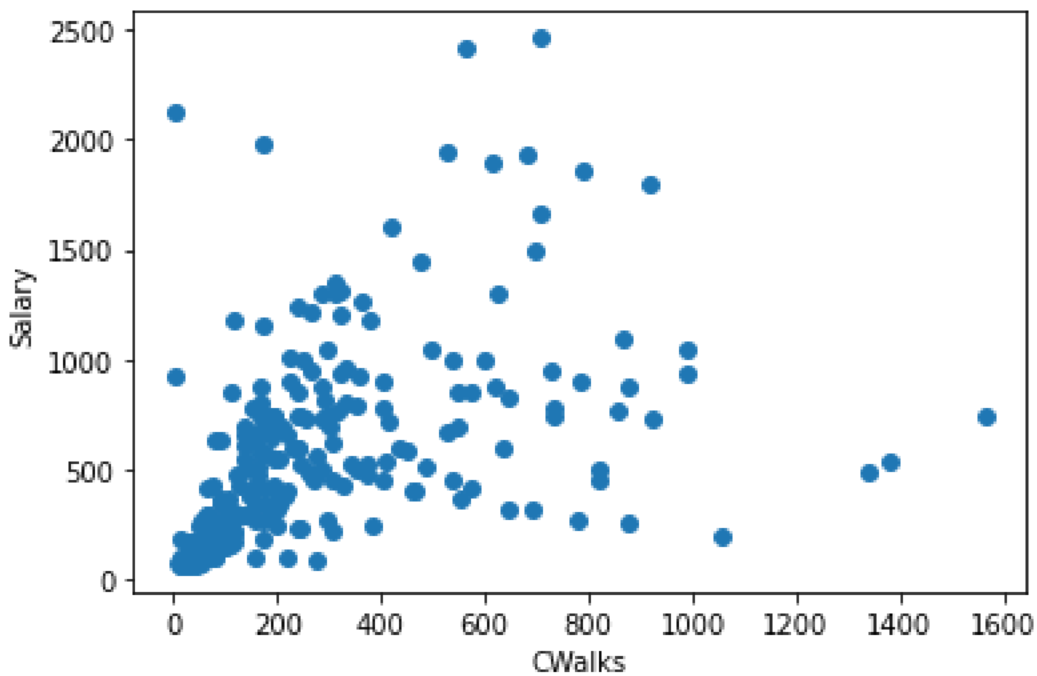

Salary
CRuns
2500
2000
1500
1000
500
0


Salary
CHits

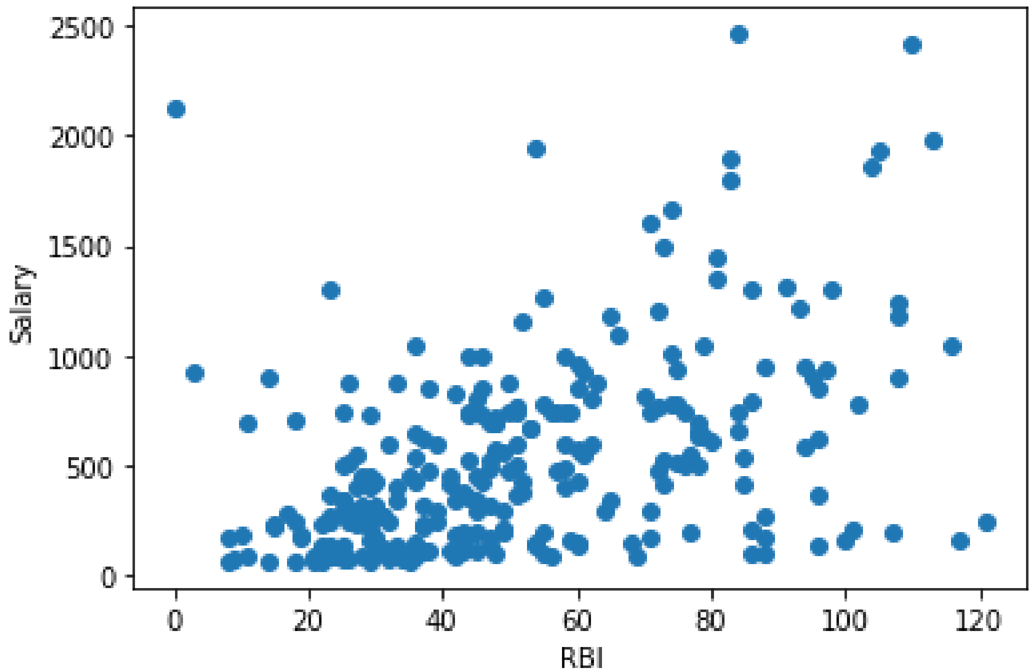

Salary
CAtBat

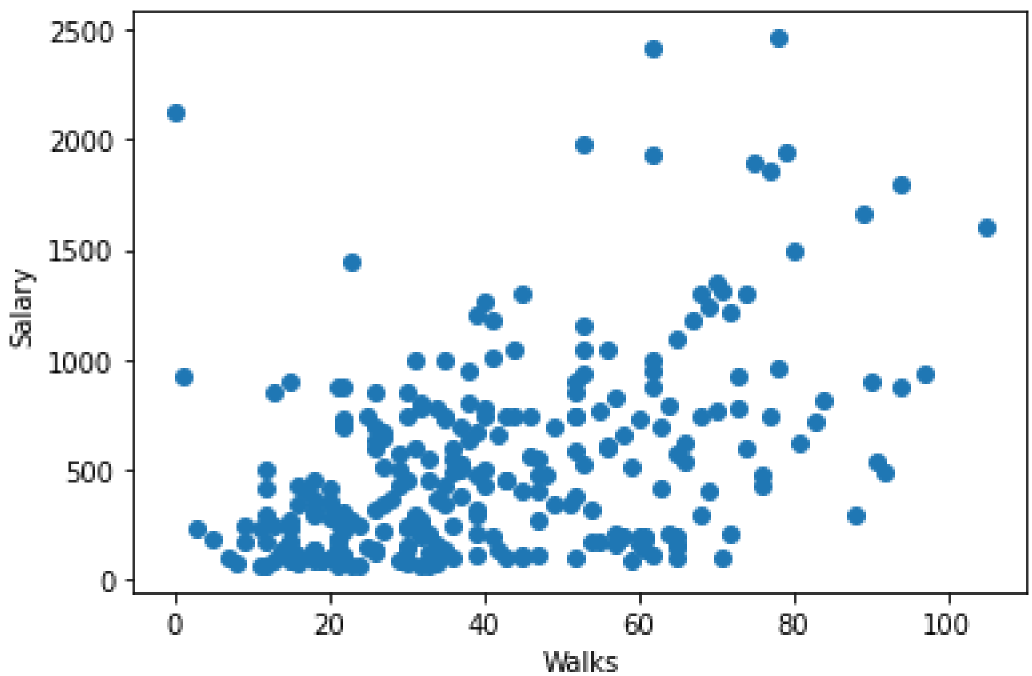


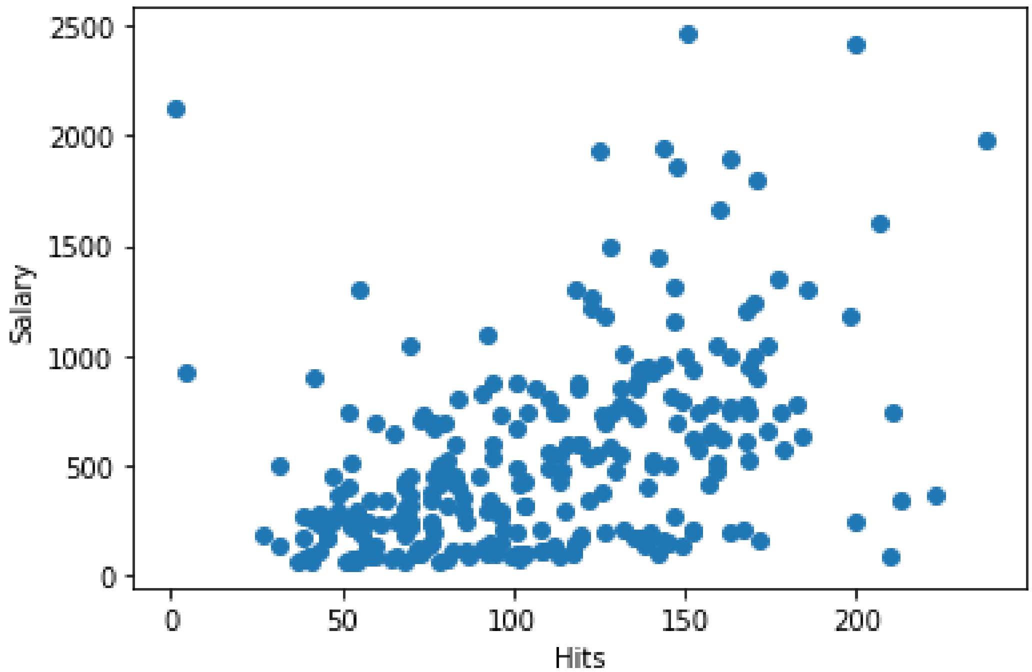
Salary
2500
2000
1500
1000
500
0
0
100
200
300
400
500
CHmRun



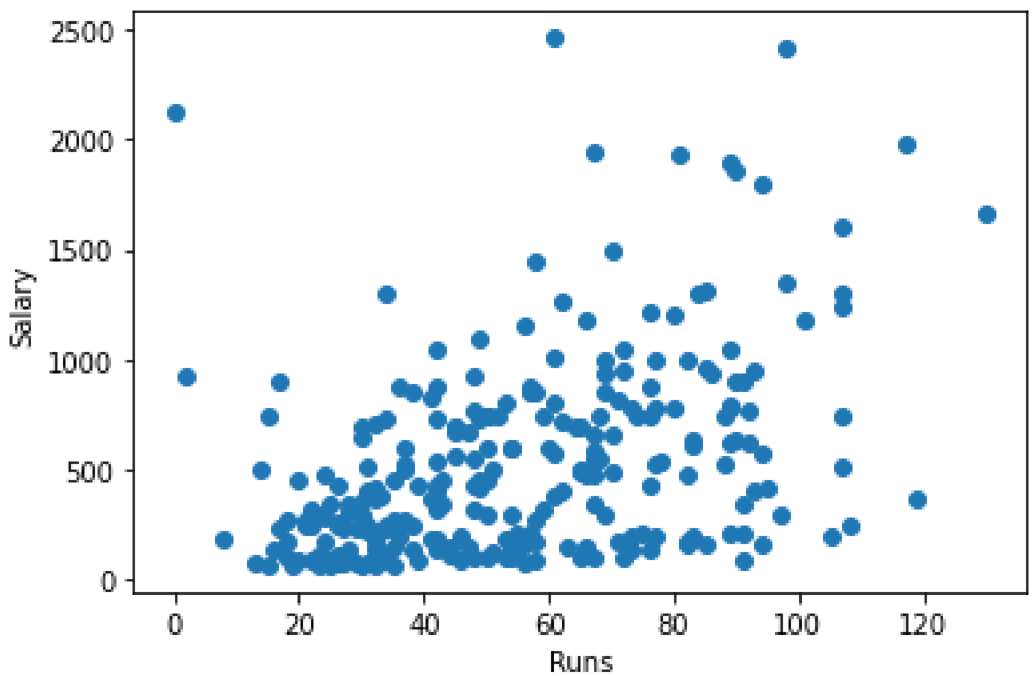
Salary
2500
2000
1500
1000
500
0
0
200
400
600
800
1000
1200
1400
1600
CWalks


2500
2000
1500
1000
500
0
Salary
0
20
40
60
80
100
120
RBI

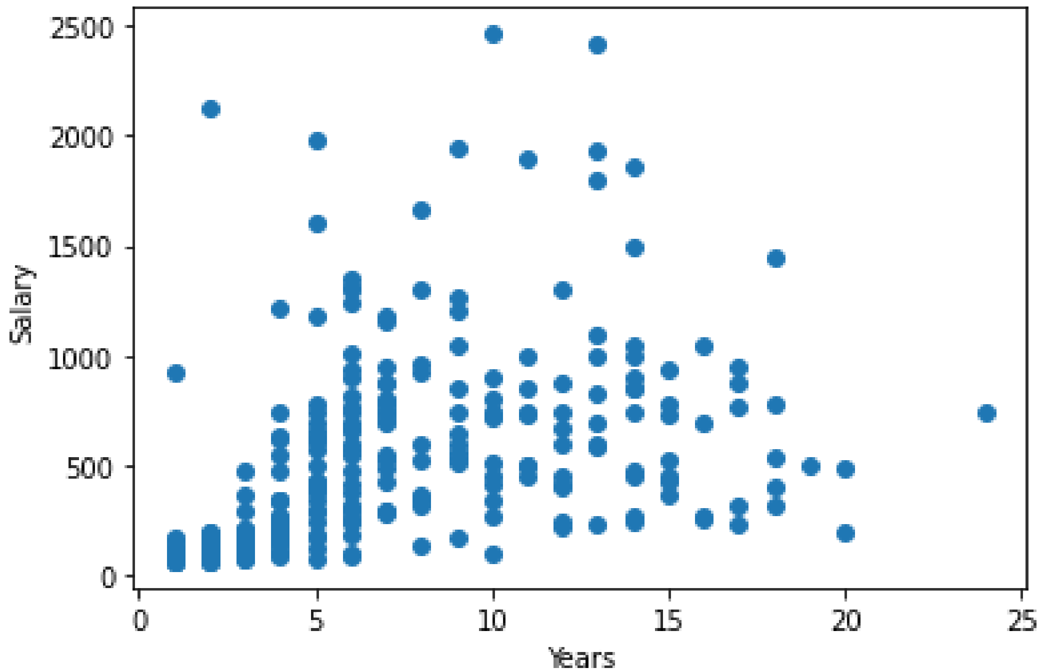

2500
2000
1500
1000
500
0
Salary
0
20
40
60
80
100
Walks

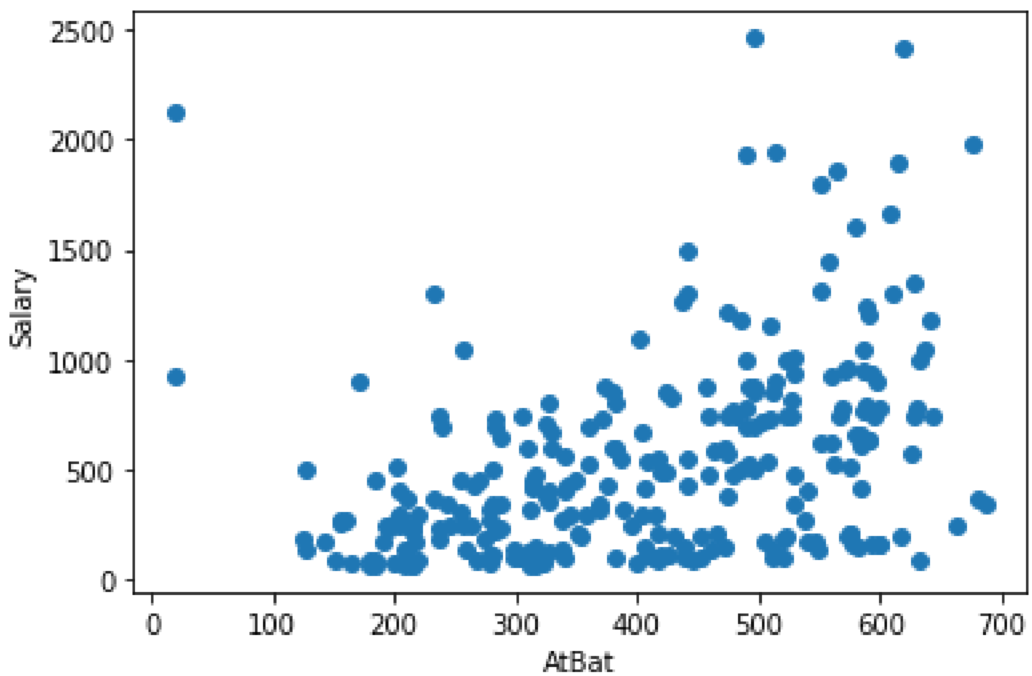


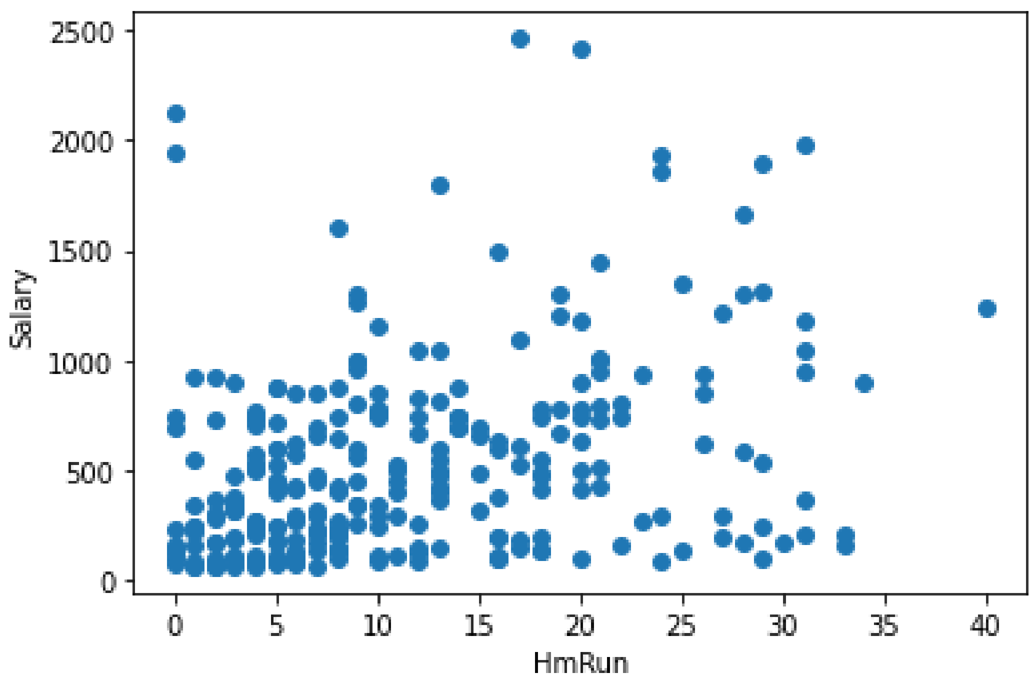
2500
2000
1500
1000
500
0
Salary
0
50
100
150
200
Hits

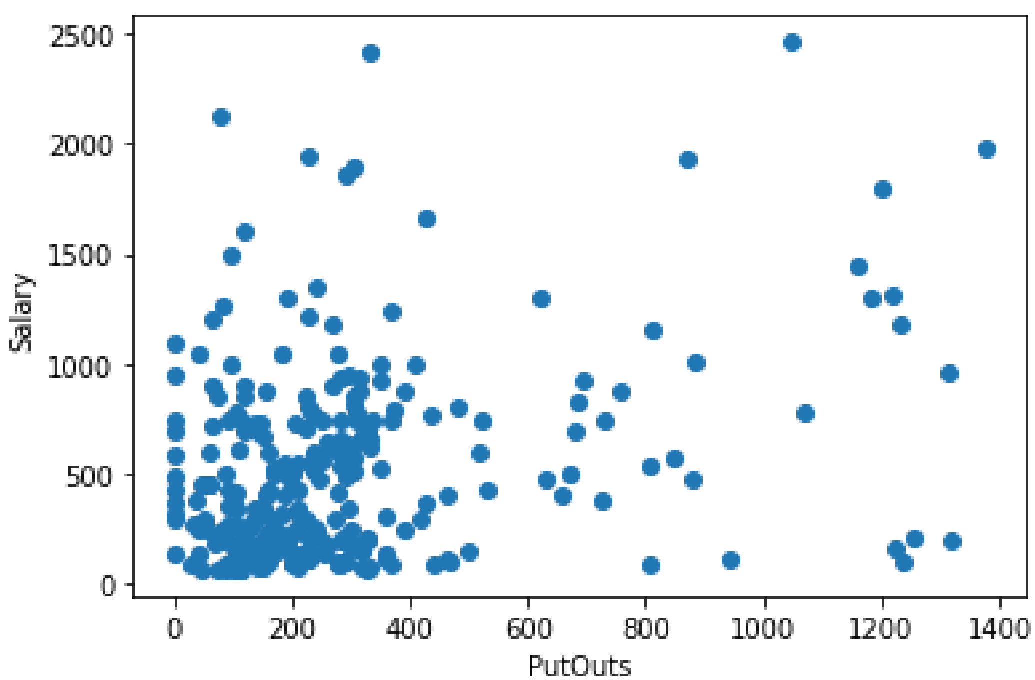


2500
2000
1500
1000
500
0
Salary
0
20
40
60
80
100
120
Runs


Salary
Years
0
500
1000
1500
2000
2500
5
10
15
20
25


Salary
AtBat
0
500
1000
1500
2000
2500
100
200
300
400
500
600
700






```
"""Creating training and validation sets"""
```

```
x_train, x_val, y_train, y_val = train_test_split(x, y, test_size = 0.3, random_state =40)
```

Creating training and validation sets

```
[ ] x_train, x_val, y_train, y_val = train_test_split(x, y, test_size = 0.3, random_state =40)
```

Linear Regression

```
[ ] lin_reg = LinearRegression()
    lin_reg.fit(x_train, y_train)
    y_val_lr = lin_reg.predict(x_val)
    r2 = r2_score(y_val, y_val_lr)
    mse = mean_squared_error(y_val, y_val_lr)
```

```
[ ] print('LR r2:', r2)
    print('LR mse:', mse)
```

```
LR r2: 0.4938380797244203
LR mse: 117743.38499844934
```

```
"""Linear Regression"""


lin_reg = LinearRegression()

lin_reg.fit(x_train, y_train)

y_val_lr = lin_reg.predict(x_val)

r2 = r2_score(y_val, y_val_lr)

mse = mean_squared_error(y_val, y_val_lr)


print('LR r2:', r2)

print('LR mse:', mse)
```

```
R r2: 0.49383807972455795
R mse: 117743.38499841733

alpha = 0.0001
R r2: 0.4938380808741073
R mse: 117743.38473100918

alpha = 0.001
R r2: 0.4938380912210767
R mse: 117743.38232409715

alpha = 0.01
R r2: 0.49383819468871304
R mse: 117743.3582554558

alpha = 1
R r2: 0.4938495537102757
R mse: 117740.71591987998

alpha = 5
R r2: 0.4938950021240246
R mse: 117730.14370995418

alpha = 10
R r2: 0.4939508312640499
R mse: 117717.15673550032

alpha = 20
R r2: 0.4940593718047622
R mse: 117691.90803511198

alpha = 40
R r2: 0.49426500457358724
R mse: 117644.07374079194

alpha = 100
R r2: 0.4948070809990702
R mse: 117517.9759236557
```

```
"""Ridge Regression"""

params = [1e-15, 1e-10, 1e-8, 1e-4, 1e-3, 1e-2, 1,5
,10, 20, 40, 100]

def RIDGE(params):
  least = 0
  b = 0
```

```
  for i in params:
    ridge = Ridge(alpha=i)
    ridge.fit(x_train, y_train)
    y_val_r = ridge.predict(x_val)
    mse = mean_squared_error(y_val, y_val_r)
    print('alpha =', i)
    print('R r2:', r2_score(y_val, y_val_r))
    print('R mse:', mse, '\n')
    if least == 0:
      least = mse
      b = i
    if mse < least:
      least = mse
      b = i
  ridge = Ridge(alpha=b)
  ridge.fit(x_train, y_train)
  return ridge
best_alpha_r_model = RIDGE(params)
```

Lasso Regression

```
[ ] @ignore_warnings(category=ConvergenceWarning)
    def LASSO(params):
      least = 0
      b = 0
      for i in params:
        lasso = Lasso(alpha= i)
        lasso.fit(x_train, y_train)
        y_val_l = lasso.predict(x_val)
        r2 = r2_score(y_val, y_val_l)
        mse = mean_squared_error(y_val, y_val_l)
        print('alpha =', i)
        print('L r2:', r2)
        print('L mse:', mse, '\n')
        if least == 0:
          least = mse
          b = i
        if mse < least:
          least = mse
          b = i
      lasso = Lasso(alpha= b)
      lasso.fit(x_train, y_train)
      return lasso
    best_alpha_l_model = LASSO(params)
```

```
alpha = 1e-15
L r2: 0.49607780006356916
L mse: 117222.38125712151

alpha = 1e-10
L r2: 0.4960778000636362
L mse: 117222.3812571059

alpha = 1e-08
L r2: 0.49607780007036595
L mse: 117222.38125554046
```

```
L r2: 0.49607780007036595
L mse: 117222.38125554046

alpha = 0.0001
L r2: 0.49607786802402554
L mse: 117222.36544816026

alpha = 0.001
L r2: 0.4960784796625811
L mse: 117222.22316880056

alpha = 0.01
L r2: 0.49608459549822614
L mse: 117220.80050312362

alpha = 1
L r2: 0.49675417411048706
L mse: 117065.04312752484

alpha = 5
L r2: 0.49923829927831664
L mse: 116487.18593538196

alpha = 10
L r2: 0.5006041737842244
L mse: 116169.4562102357

alpha = 20
L r2: 0.5025627283519967
L mse: 115713.85725014775

alpha = 40
L r2: 0.5062834156946681
L mse: 114848.35096708225

alpha = 100
L r2: 0.5148004448700553
L mse: 112867.11965538132
```

```
"""Lasso Regression"""

@ignore_warnings(category=ConvergenceWarning)
def LASSO(params):
  least = 0
  b = 0
  for i in params:
    lasso = Lasso(alpha= i)
    lasso.fit(x_train, y_train)
```

```
    y_val_l = lasso.predict(x_val)
    r2 = r2_score(y_val, y_val_l)
    mse = mean_squared_error(y_val, y_val_l)
    print('alpha =', i)
    print('L r2:', r2)
    print('L mse:', mse, '\n')
    if least == 0:
      least = mse
      b = i
    if mse < least:
      least = mse
      b = i
  lasso = Lasso(alpha= b)
  lasso.fit(x_train, y_train)
  return lasso
best_alpha_l_model = LASSO(params)

"""Lasso on Test Set"""
```

Lasso on Test Set

```
[ ] x_test = test[field]
    y_test_l = best_alpha_l_model.predict(x_test)
    pd.DataFrame(y_test_l).describe()
```

	0
count	59.000000
mean	482.311806
std	336.645030
min	166.403359
25%	235.888629
50%	332.797227
75%	617.421863
max	1451.356950

```
x_test = test[field]

y_test_l = best_alpha_l_model.predict(x_test)

pd.DataFrame(y_test_l).describe()


"""Ridge on Test Set"""
```

Ridge on Test Set

```
[ ] y_test_r = best_alpha_r_model.predict(x_test)
    pd.DataFrame(y_test_r).describe()
```

	0
count	59.000000
mean	489.352350
std	349.119685
min	174.963115
25%	233.011502
50%	349.071671
75%	619.382315
max	1473.836164


```
y_test_r = best_alpha_r_model.predict(x_test)

pd.DataFrame(y_test_r).describe()
```


```
"""Linear Regression on Test Set"""
```

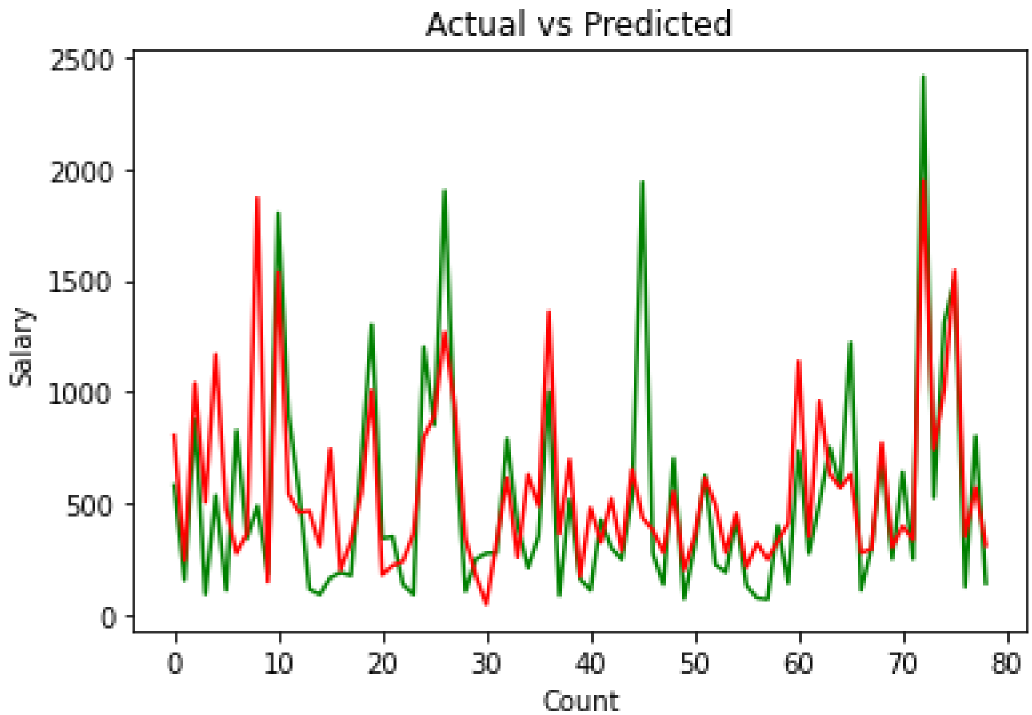
Linear Regression on Test Set

```
[ ] y_test_lr = lin_reg.predict(x_test)
    pd.DataFrame(y_test_lr).describe()
```

	0
count	59.000000
mean	489.683559
std	349.636341
min	174.411652
25%	231.992653
50%	348.919382
75%	620.148050
max	1473.703232

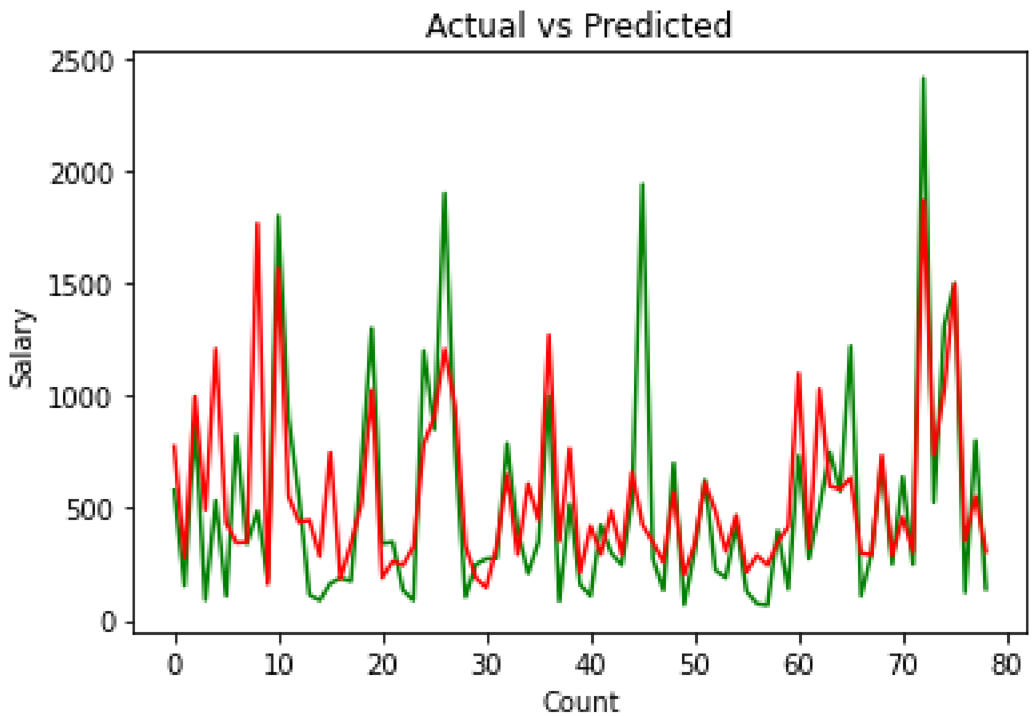
```
y_test_lr = lin_reg.predict(x_test)

pd.DataFrame(y_test_lr).describe()
```

```
"""**Visualising the Results**


Linear Regression

"""
```



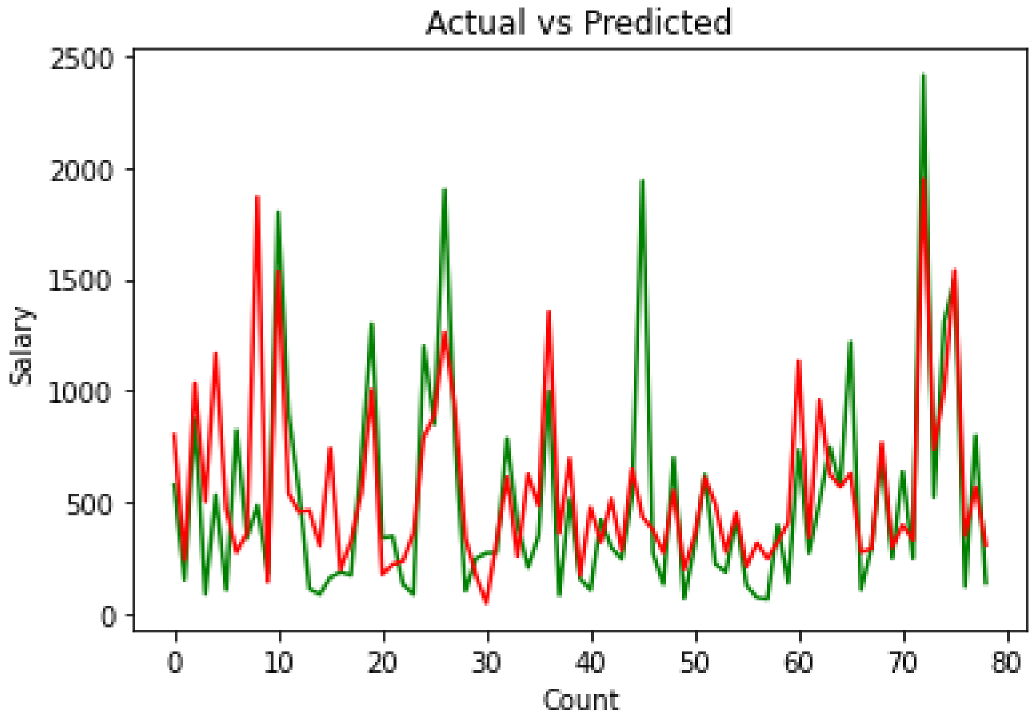

```
plt.plot(range(len(y_val)), y_val, c='g')

plt.plot(range(len(y_val)), lin_reg.predict(x_val),
c='r')

plt.xlabel('Count')

plt.ylabel('Salary')

plt.title('Actual vs Predicted')

plt.show()


"""Lasso"""


plt.plot(range(len(y_val)), y_val, c='g')

plt.plot(range(len(y_val)),
best_alpha_l_model.predict(x_val), c='r')
```

```
plt.xlabel('Count')

plt.ylabel('Salary')

plt.title('Actual vs Predicted')

plt.show()
```




```
"""Ridge"""


plt.plot(range(len(y_val)), y_val, c='g')

plt.plot(range(len(y_val)),
best_alpha_r_model.predict(x_val), c='r')

plt.xlabel('Count')

plt.ylabel('Salary')

plt.title('Actual vs Predicted')
```

```
plt.show()
```



#Conclusion

#It is pretty evident from the results that ridge and lasso regression both improve upon the results of linear

#regression. However, the impact is not that significant on this dataset due to its limited size. The best performing

#model out of the three was LASSO, with and R2 score of ~52