In [1]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn import linear_model
```

Assignment5

Ground Cricket Chirps

In *The Song of Insects* (1948) by George W. Pierce, Pierce mechanically measured the frequency (the number of wing vibrations per second) of chirps (or pulses of sound) made by a striped ground cricket, at various ground temperatures. Since crickets are ectotherms (cold-blooded), the rate of their physiological processes and their overall metabolism are influenced by temperature. Consequently, there is reason to believe that temperature would have a profound effect on aspects of their behavior, such as chirp frequency.

In general, it was found that crickets did not sing at temperatures colder than 60º F. or warmer than 100º F.

In [2]:
```python
ground_cricket_data = {"Chirps/Second": [20.0, 16.0, 19.8, 18.4, 17.1, 15.5, 14.7,
                                         15.7, 15.4, 16.3, 15.0, 17.2, 16.0, 17.0,
                                         14.4],
                       "Ground Temperature": [88.6, 71.6, 93.3, 84.3, 80.6, 75.2, 69.7,
                                              71.6, 69.4, 83.3, 79.6, 82.6, 80.6, 83.5,
                                              76.3]}
df = pd.DataFrame(ground_cricket_data)
```

Tasks

1. Find the linear regression equation for this data.
2. Chart the original data and the equation on the chart.
3. Find the equation's R^2 score (use the `.score` method) to determine whether the

equation is a good fit for this data. (0.8 and greater is considered a strong correlation.) 4. Extrapolate data: If the ground temperature reached 95, then at what approximate rate would you expect the crickets to be chirping? 5. Interpolate data: With a listening device, you discovered that on a particular morning the crickets were chirping at a rate of 18 chirps per second. What was the approximate ground temperature that morning?

In [3]:
```python
df.shape
```

Out[3]:
```
(15, 2)
```

In [4]:
```python
df
```

Out[4]:

	Chirps/Second	Ground Temperature
0	20.0	88.6
1	16.0	71.6
2	19.8	93.3
3	18.4	84.3
4	17.1	80.6
5	15.5	75.2
6	14.7	69.7
7	15.7	71.6
8	15.4	69.4
9	16.3	83.3
10	15.0	79.6
11	17.2	82.6
12	16.0	80.6
13	17.0	83.5
14	14.4	76.3

In [5]:

```
df.describe() # to clean the data
```

Out[5]:

	Chirps/Second	Ground Temperature
count	15.000000	15.000000
mean	16.566667	79.346667
std	1.712837	7.020467
min	14.400000	69.400000
25%	15.450000	73.400000
50%	16.000000	80.600000
75%	17.150000	83.400000
max	20.000000	93.300000

In [6]:

```
df.isnull().sum() #Checking any null values
```

Out[6]:

```
Chirps/Second         0
Ground Temperature    0
dtype: int64
```

In [7]:

```
df.dtypes #check data types
```

Out[7]:

```
Chirps/Second         float64
Ground Temperature    float64
dtype: object
```

In [8]:

```
df = df.drop_duplicates() #check duplicates
df.shape
```

Out[8]: (15, 2)

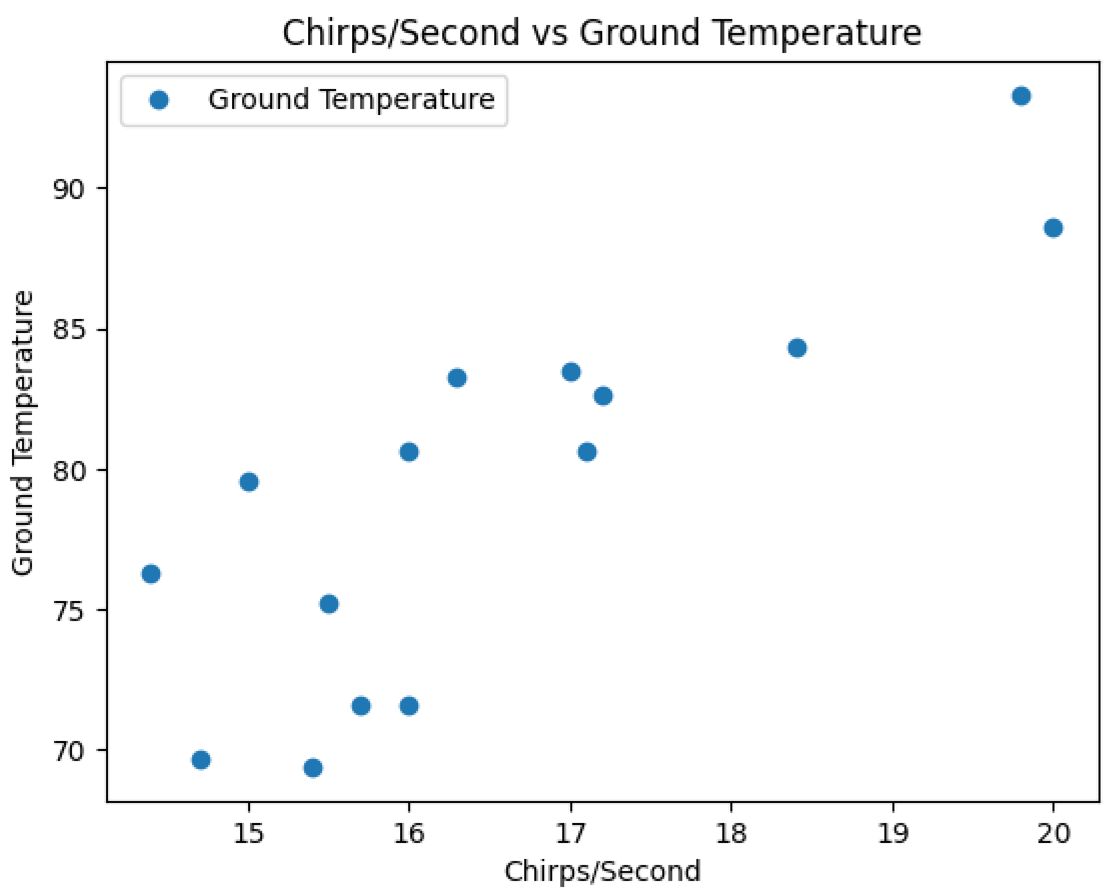
In [9]:
```
iqr = df['Chirps/Second'].quantile(0.75) - df['Chirps/Second'].quantile(0.25) #check outliers
upper_threshold = df['Chirps/Second'].quantile(0.75) + (1.5 * iqr)
lower_threshold = df['Chirps/Second'].quantile(0.25) - (1.5 * iqr)
upper_threshold, lower_threshold
```

Out[9]: (19.699999999999996, 12.9)

In [10]:
```
iqr = df['Ground Temperature'].quantile(0.75) - df['Ground Temperature'].quantile(0.25)
upper_threshold = df['Ground Temperature'].quantile(0.75) + (1.5 * iqr)
lower_threshold = df['Ground Temperature'].quantile(0.25) - (1.5 * iqr)
upper_threshold, lower_threshold
```

Out[10]: (98.4, 58.400000000000006)

In [11]:
```
df.plot(x='Chirps/Second', y='Ground Temperature', style='o')
plt.title('Chirps/Second vs Ground Temperature')
plt.xlabel('Chirps/Second')
plt.ylabel('Ground Temperature')
plt.show()
```



In [12]:
```
df.corr() # correlation
```

Out[12]:

	Chirps/Second	Ground Temperature
Chirps/Second	1.000000	0.832042
Ground Temperature	0.832042	1.000000

In [13]:
```
X = df.loc[:, ['Chirps/Second']].values # select all rows and select all columns except the last
y = df.loc[:, 'Ground Temperature'].values # target as arrays
# Syntax : dataset.loc[:, :-1]
```

```
from sklearn.model_selection import train_test_split #import the required function
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

In [14]:
```
y_test
```

Out[14]: `array([83.3, 69.7, 79.6, 83.5])`

In [15]:
```
X_train.shape, X_test.shape
```

Out[15]: `((11, 1), (4, 1))`

In [16]:
```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression() # spredicted score = m * hours + c
"Symtax : varName = ModelName(modelHyperParams)"
regressor.fit(X_train, y_train)
```

Out[16]:
```
▾ LinearRegression
LinearRegression()
```

In [17]:
```
print(regressor.intercept_) # c
```

```
19.114733483483498
```

In [18]:
```
print(regressor.coef_) # slope - m
```

```
[3.57864114]
```

In [19]:
```
regressor.predict([[7]])
```

Out[19]: `array([44.16522147])`

In [20]:
```
y_pred = regressor.predict(X_test)
"Syntax : varName.predict(test_features)"
y_pred
```

Out[20]: `array([77.44658408, 71.72075826, 72.7943506 , 79.95163288])`

In [21]:
```
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df
```

Out[21]:

	Actual	Predicted
0	83.3	77.446584
1	69.7	71.720758
2	79.6	72.794351
3	83.5	79.951633

In [22]:
```
regressor.predict([[12]]) # perils of extrapolation
```

Out[22]: `array([62.05842718])`

In [23]:
```
from sklearn import metrics # metrics will contain all the evaluation metrics
print('R2- SCORE:', metrics.r2_score(y_test,y_pred))
regressor.score(X_test,y_test)
```

```
R2- SCORE: 0.22560991525112128
```

Out[23]: 0.22560991525112128

Assignment6

Brain vs. Body Weight

In the file `brain_body.txt`, the average brain and body weight for a number of mammal species are recorded. Load this data into a Pandas data frame.

Tasks

1. Find the linear regression equation for this data for brain weight to body weight.
2. Chart the original data and the equation on the chart.
3. Find the equation's R^2 score (use the `.score` method) to determine whether the

equation is a good fit for this data. (0.8 and greater is considered a strong correlation.)

In [45]:
```
df = pd.read_fwf("C:/Users/Karthi/Downloads/brain_body.txt")
```

In [46]:
```
df
```

Out[46]:

	Brain	Body
0	3.385	44.5
1	0.480	15.5
2	1.350	8.1
3	465.000	423.0
4	36.330	119.5
...	...	...
57	160.000	169.0
58	0.900	2.6
59	1.620	11.4
60	0.104	2.5
61	4.235	50.4

62 rows × 2 columns

In [47]:
```
print(df.shape)
df.head()
```

```
(62, 2)
```

Out[47]:

	Brain	Body
0	3.385	44.5
1	0.480	15.5
2	1.350	8.1
3	465.000	423.0
4	36.330	119.5

In [48]:
```
df.describe()
```

Out[48]:

	Brain	Body
count	62.000000	62.000000
mean	198.789984	283.134194
std	899.158011	930.278942
min	0.005000	0.140000
25%	0.600000	4.250000
50%	3.342500	17.250000
75%	48.202500	166.000000
max	6654.000000	5712.000000

In [49]:
```
df.isnull().sum() #Checking any null values
```

Out[49]:
```
Brain    0
Body     0
dtype: int64
```

In [50]:
```
df.dtypes #check data types
```

Out[50]:
```
Brain    float64
Body     float64
dtype: object
```

In [51]:
```
df = df.drop_duplicates() #check duplicates
df.shape
```

Out[51]:
```
(62, 2)
```

In [52]:
```
iqr = df['Brain'].quantile(0.75) - df['Brain'].quantile(0.25) #check outliers
upper_threshold = df['Brain'].quantile(0.75) + (1.5 * iqr)
lower_threshold = df['Brain'].quantile(0.25) - (1.5 * iqr)
upper_threshold, lower_threshold
```

Out[52]:
```
(119.60625, -70.80375000000001)
```

In [63]:
```
df.Brain = df.Brain.clip(-70.80,119.60)
```

In [53]:
```
iqr = df['Body'].quantile(0.75) - df['Body'].quantile(0.25) #check outliers
upper_threshold = df['Body'].quantile(0.75) + (1.5 * iqr)
lower_threshold = df['Body'].quantile(0.25) - (1.5 * iqr)
upper_threshold, lower_threshold
```

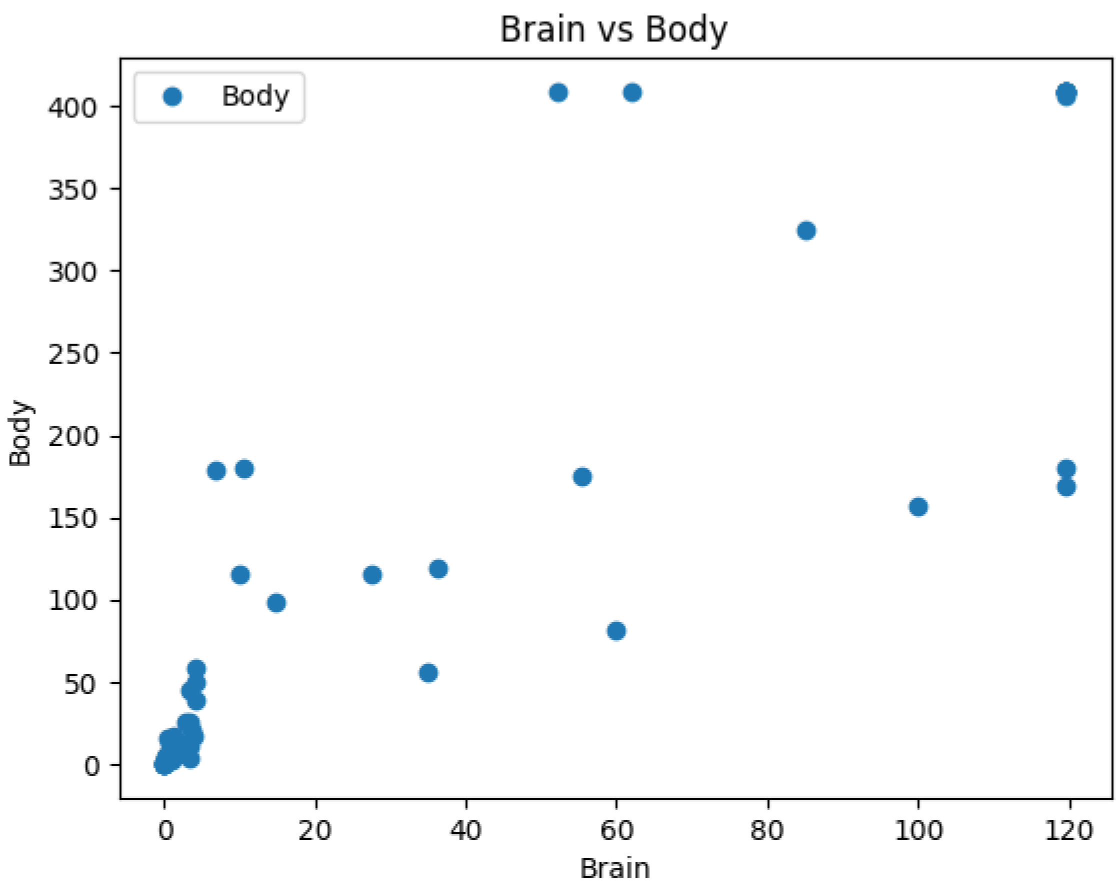
Out[53]: (408.625, -238.375)

In [64]:
```python
df.Body = df.Body.clip(-238.375,408.625)
```

In [65]:
```python
df.shape
```

Out[65]: (62, 2)

In [66]:
```python
df.plot(x='Brain', y='Body', style='o')
plt.title('Brain vs Body')
plt.xlabel('Brain')
plt.ylabel('Body')
plt.show()
```



In [57]:
```python
df.corr() # correlation
```

Out[57]:

	Brain	Body
Brain	1.000000	0.934164
Body	0.934164	1.000000

In [67]:
```python
X = df.loc[:, ['Brain']].values # select all rows and select all columns except the last column
y = df.loc[:, 'Body'].values # target as arrays
# Syntax : dataset.loc[:, :-1]
from sklearn.model_selection import train_test_split #import the required function
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
```

In [68]:
```python
y_test
```

Out[68]:
```
array([1.20000e+00, 1.57000e+02, 5.70000e+00, 4.08625e+02, 1.79000e+02,
       6.30000e+00, 1.14000e+01, 2.50000e-01, 5.60000e+01, 4.08625e+02,
       4.08625e+02, 3.92000e+01, 2.50000e+01, 1.79500e+02, 3.90000e+00,
       4.00000e-01])
```

In [69]:
```
X_train.shape, X_test.shape
```

Out[69]:
```
((46, 1), (16, 1))
```

In [70]:
```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression() # spredicted score = m * hours + c
"Symtax : varName = ModelName(modelHyperParams)"
regressor.fit(X_train, y_train)
```

Out[70]:
```
▾ LinearRegression
LinearRegression()
```

In [71]:
```
print(regressor.intercept_) # c
```

```
17.785739646339067
```

In [72]:
```
print(regressor.coef_) # slope - m
```

```
[2.94799362]
```

In [73]:
```
regressor.predict([[7]])
```

Out[73]:
```
array([38.42169496])
```

In [74]:
```
y_pred = regressor.predict(X_test)
"Syntax : varName.predict(test_features)"
y_pred
```

Out[74]:
```
array([ 18.00683917, 312.58510127,  20.49789377, 370.36577615,
        37.83209624,  22.79732879,  22.5614893 ,  17.81521958,
       120.96551622, 370.36577615, 370.36577615,  30.42673627,
        26.6297205 ,  48.8870723 ,  28.1037173 ,  17.8535435 ])
```

In [75]:
```
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df
```

Out[75]:

	Actual	Predicted
0	1.200	18.006839
1	157.000	312.585101
2	5.700	20.497894
3	408.625	370.365776
4	179.000	37.832096
5	6.300	22.797329
6	11.400	22.561489
7	0.250	17.815220
8	56.000	120.965516
9	408.625	370.365776
10	408.625	370.365776
11	39.200	30.426736
12	25.000	26.629720
13	179.500	48.887072
14	3.900	28.103717
15	0.400	17.853543

In [76]:

```
regressor.predict([[12]]) # perils of extrapolation
```

Out[76]:

```
array([53.16166304])
```

In [77]:

```
from sklearn import metrics # metrics will contain all the evaluation metrics
print('R2- SCORE:', metrics.r2_score(y_test,y_pred))
regressor.score(X_test,y_test)
```

```
R2- SCORE: 0.8064557642758013
```

Out[77]:

```
0.8064557642758013
```

Assignment7

Salary Discrimination

The file `salary.txt` contains data for 52 tenure-track professors at a small Midwestern college. This data was used in legal proceedings in the 1980s about discrimination against women in salary.

The data in the file, by column:

1. Sex. 1 for female, 0 for male.
2. Rank. 1 for assistant professor, 2 for associate professor, 3 for full professor.
3. Year. Number of years in current rank.
4. Degree. Highest degree. 1 for doctorate, 0 for master's.
5. YSdeg. Years since highest degree was earned.
6. Salary. Salary/year in dollars.

Tasks

1. Find the linear regression equation for this data using columns 1-5 to column 6.
2. Find the selection of columns with the best R^2 score.
3. Report whether sex is a factor in salary.

In [79]:
```
df = pd.read_fwf("C:/Users/Karthi/Downloads/salary.txt", header=None,
                 names=["Sex", "Rank", "Year", "Degree", "YSdeg", "Salary"])
```

In [85]:
```
df.head()
```

Out[85]:

	Sex	Rank	Year	Degree	YSdeg	Salary
0	0	3	25	1	35	36350
1	0	3	13	1	22	35350
2	0	3	10	1	23	28200
3	1	3	7	1	27	26775
4	0	3	19	0	30	33696

In [86]:
```
df.shape
```

Out[86]:
```
(52, 6)
```

In [88]:
```
df.isnull().sum()
```

Out[88]:
```
Sex       0
Rank      0
Year      0
Degree    0
YSdeg     0
Salary    0
dtype: int64
```

In [89]:
```
df.dtypes
```

Out[89]:
```
Sex       int64
Rank      int64
Year      int64
Degree    int64
YSdeg     int64
Salary    int64
dtype: object
```

In [90]:
```
df = df.drop_duplicates()
```

In [91]:
```
df.describe()
```

Out[91]:

	Sex	Rank	Year	Degree	YSdeg	Salary
count	52.000000	52.000000	52.000000	52.000000	52.000000	52.000000
mean	0.269231	2.038462	7.480769	0.653846	16.115385	23797.653846
std	0.447888	0.862316	5.507536	0.480384	10.222340	5917.289154
min	0.000000	1.000000	0.000000	0.000000	1.000000	15000.000000
25%	0.000000	1.000000	3.000000	0.000000	6.750000	18246.750000
50%	0.000000	2.000000	7.000000	1.000000	15.500000	23719.000000
75%	1.000000	3.000000	11.000000	1.000000	23.250000	27258.500000
max	1.000000	3.000000	25.000000	1.000000	35.000000	38045.000000

In [95]:

```
iqr = df['Sex'].quantile(0.75) - df['Sex'].quantile(0.25)
upper_threshold = df['Sex'].quantile(0.75) + (1.5 * iqr)
lower_threshold = df['Sex'].quantile(0.25) - (1.5 * iqr)
upper_threshold, lower_threshold
```

Out[95]: (2.5, -1.5)

In [96]:

```
iqr = df['Rank'].quantile(0.75) - df['Rank'].quantile(0.25)
upper_threshold = df['Rank'].quantile(0.75) + (1.5 * iqr)
lower_threshold = df['Rank'].quantile(0.25) - (1.5 * iqr)
upper_threshold, lower_threshold
```

Out[96]: (6.0, -2.0)

In [97]:

```
iqr = df['Year'].quantile(0.75) - df['Year'].quantile(0.25)
upper_threshold = df['Year'].quantile(0.75) + (1.5 * iqr)
lower_threshold = df['Year'].quantile(0.25) - (1.5 * iqr)
upper_threshold, lower_threshold
```

Out[97]: (23.0, -9.0)

In [98]:

```
iqr = df['Degree'].quantile(0.75) - df['Degree'].quantile(0.25)
upper_threshold = df['Degree'].quantile(0.75) + (1.5 * iqr)
lower_threshold = df['Degree'].quantile(0.25) - (1.5 * iqr)
upper_threshold, lower_threshold
```

Out[98]: (2.5, -1.5)

In [99]:

```
iqr = df['YSdeg'].quantile(0.75) - df['YSdeg'].quantile(0.25)
upper_threshold = df['YSdeg'].quantile(0.75) + (1.5 * iqr)
lower_threshold = df['YSdeg'].quantile(0.25) - (1.5 * iqr)
upper_threshold, lower_threshold
```

Out[99]: (48.0, -18.0)

In [100…

```
df.YSdeg = df.YSdeg.clip(18.0,48.0)
```

In [101…

```
df.shape
```

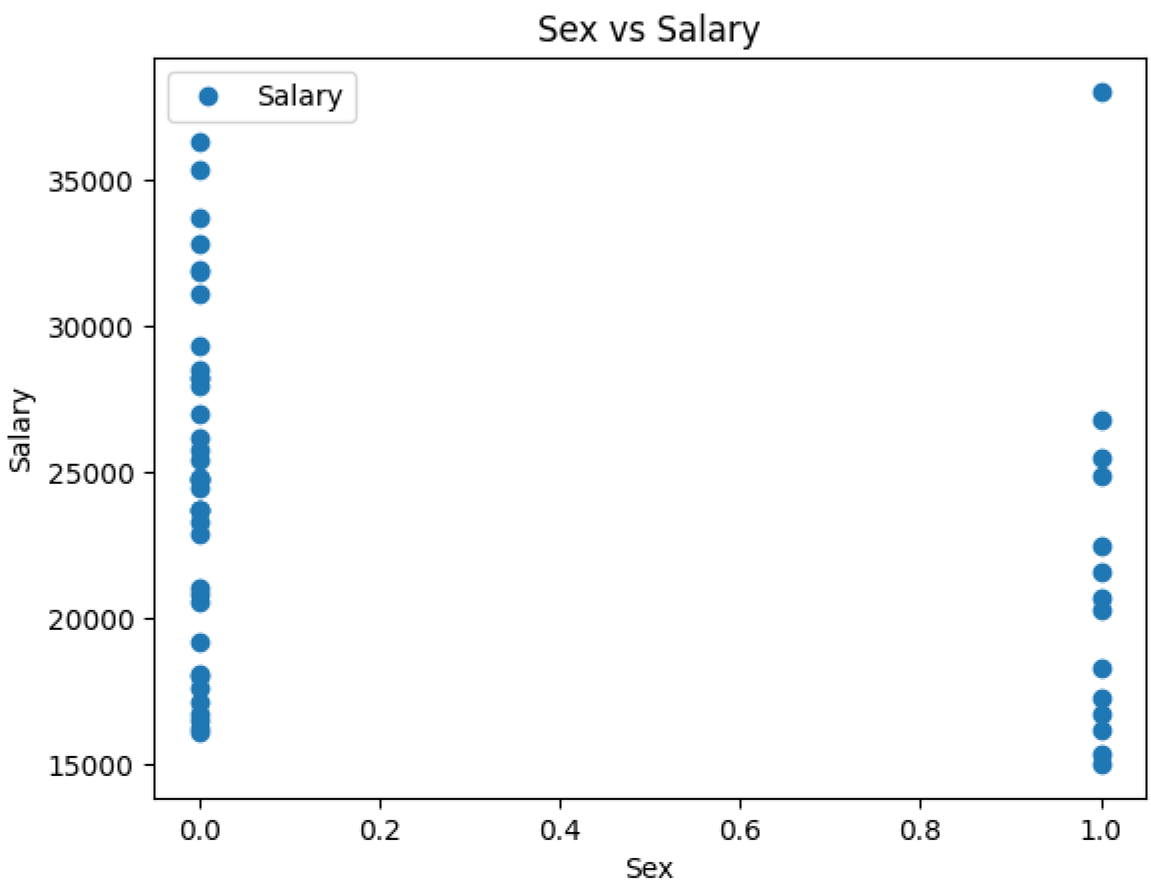
Out[101]: (52, 6)

In [103…

```
df.groupby('Sex')['Salary'].mean()
```

Out[103]:
```
Sex
0    24696.789474
1    21357.142857
Name: Salary, dtype: float64
```

In [104…
```
df.plot(x='Sex', y='Salary', style='o')
plt.title('Sex vs Salary')
plt.xlabel('Sex')
plt.ylabel('Salary')
plt.show()
```

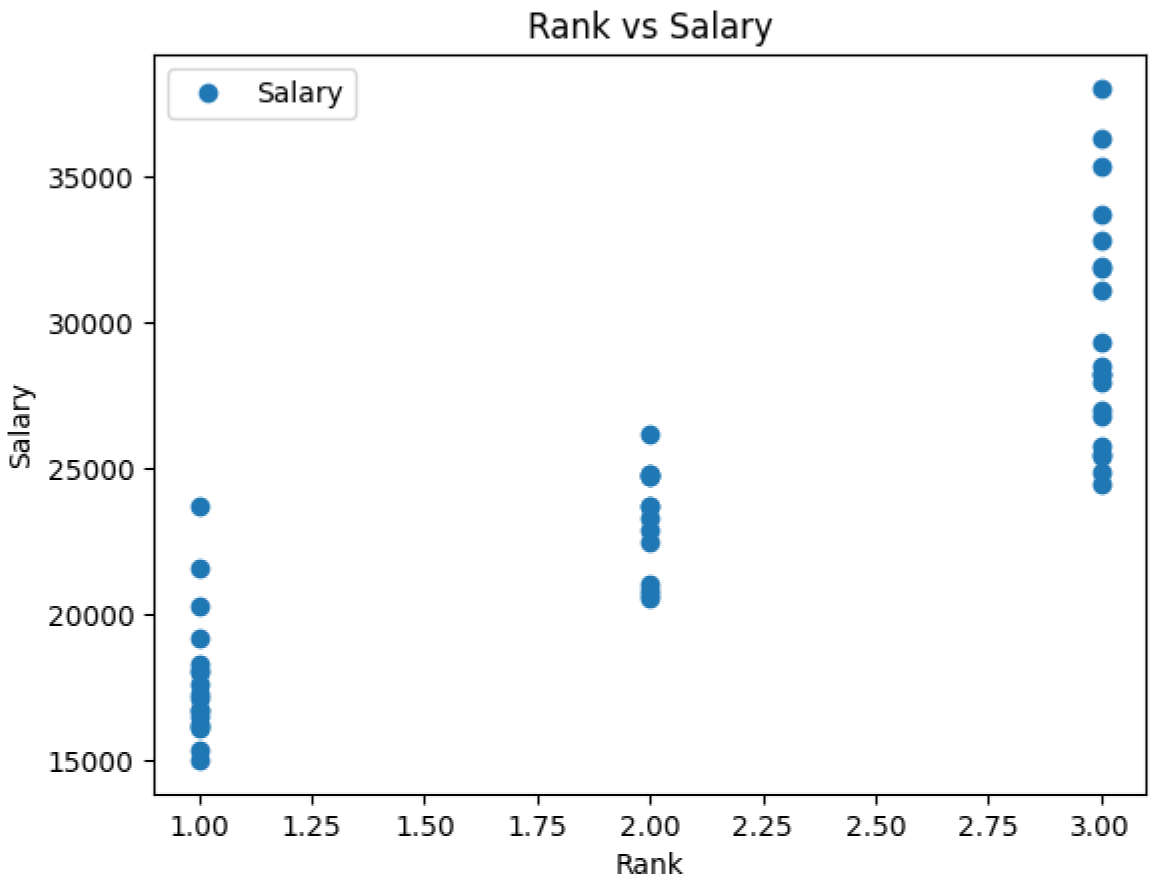


In [105…
```
df[['Sex','Salary']].corr()
```

Out[105]:

	Sex	Salary
Sex	1.000000	-0.252782
Salary	-0.252782	1.000000

In [106…
```
df.plot(x='Rank', y='Salary', style='o')
plt.title('Rank vs Salary')
plt.xlabel('Rank')
plt.ylabel('Salary')
plt.show()
```



In [107…

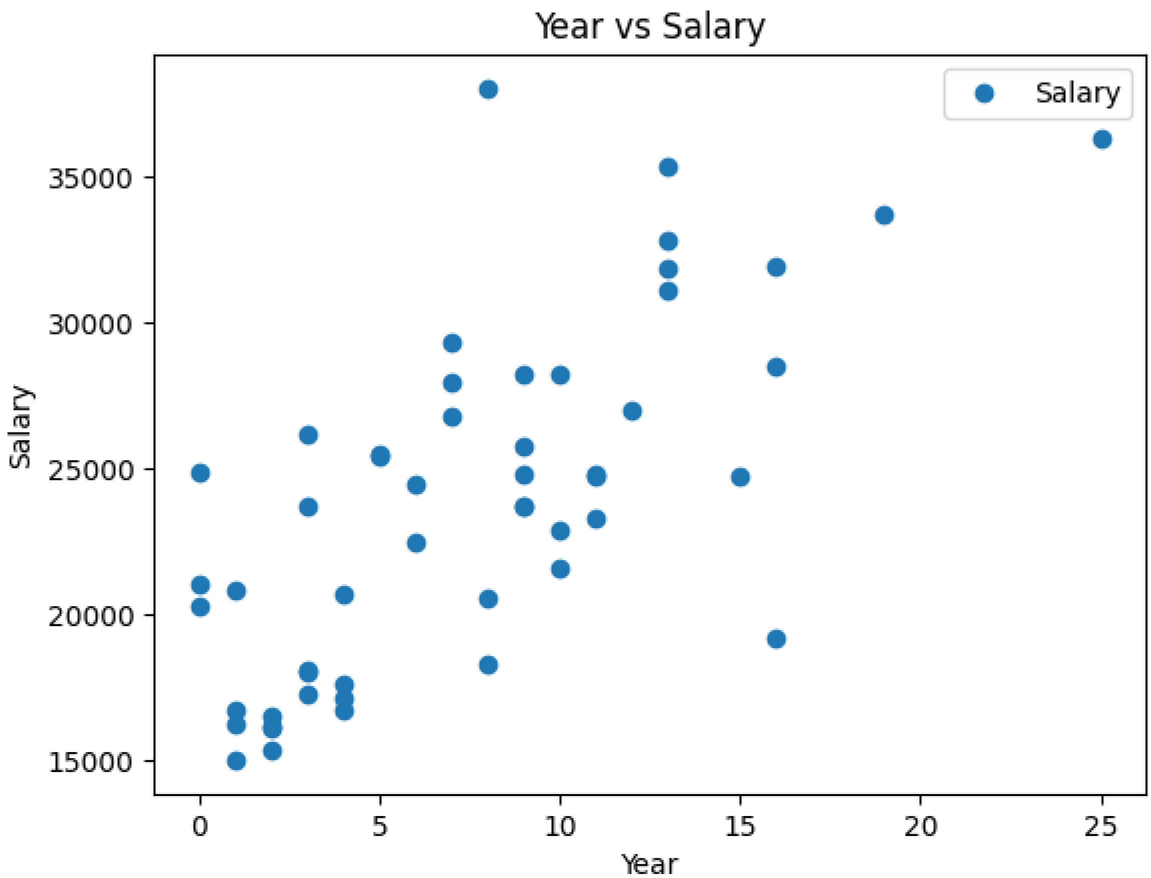
```
df[['Rank','Salary']].corr()
```

Out[107]:

	Rank	Salary
Rank	1.000000	0.867488
Salary	0.867488	1.000000

In [108…

```
df.plot(x='Year', y='Salary', style='o')
plt.title('Year vs Salary')
plt.xlabel('Year')
plt.ylabel('Salary')
plt.show()
```

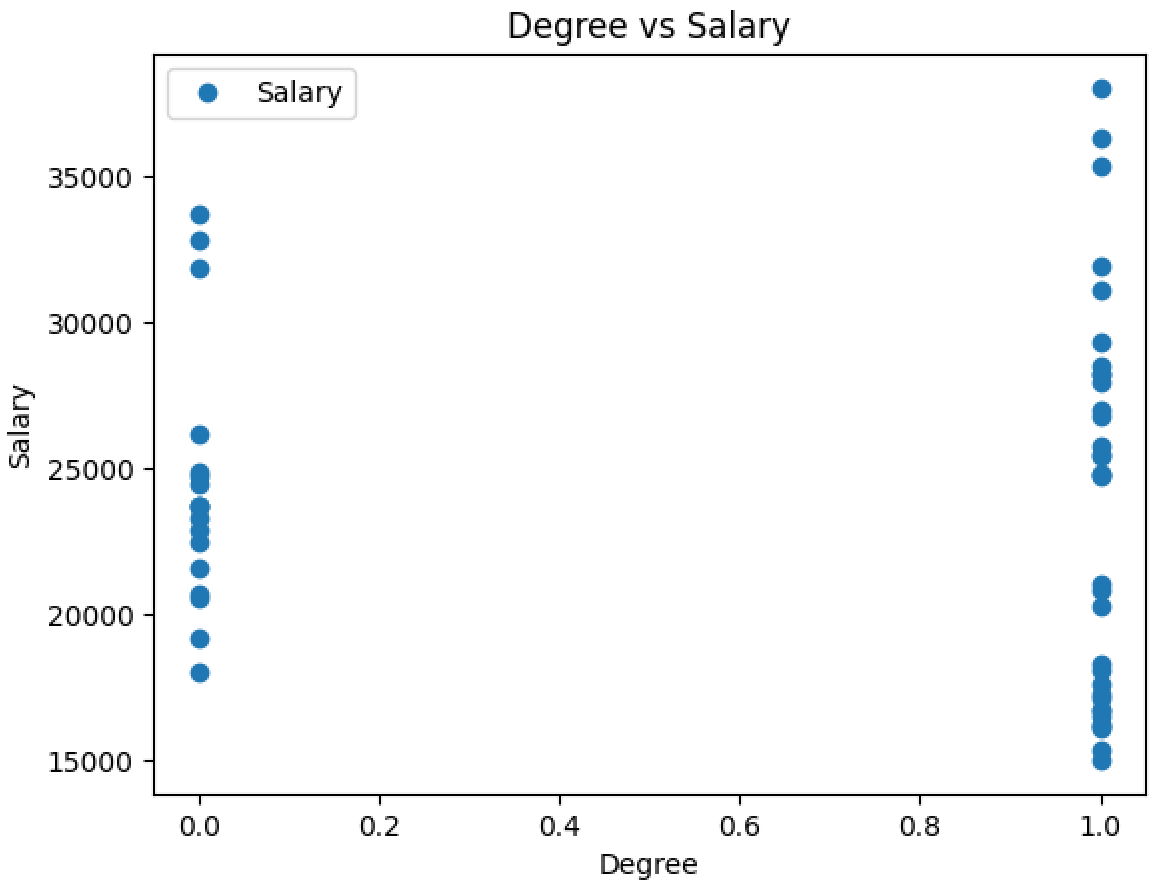


In [109…
```
df[['Year','Salary']].corr()
```

Out[109]:

	Year	Salary
Year	1.000000	0.700669
Salary	0.700669	1.000000

In [110…
```
df.plot(x='Degree', y='Salary', style='o')
plt.title('Degree vs Salary')
plt.xlabel('Degree')
plt.ylabel('Salary')
plt.show()
```



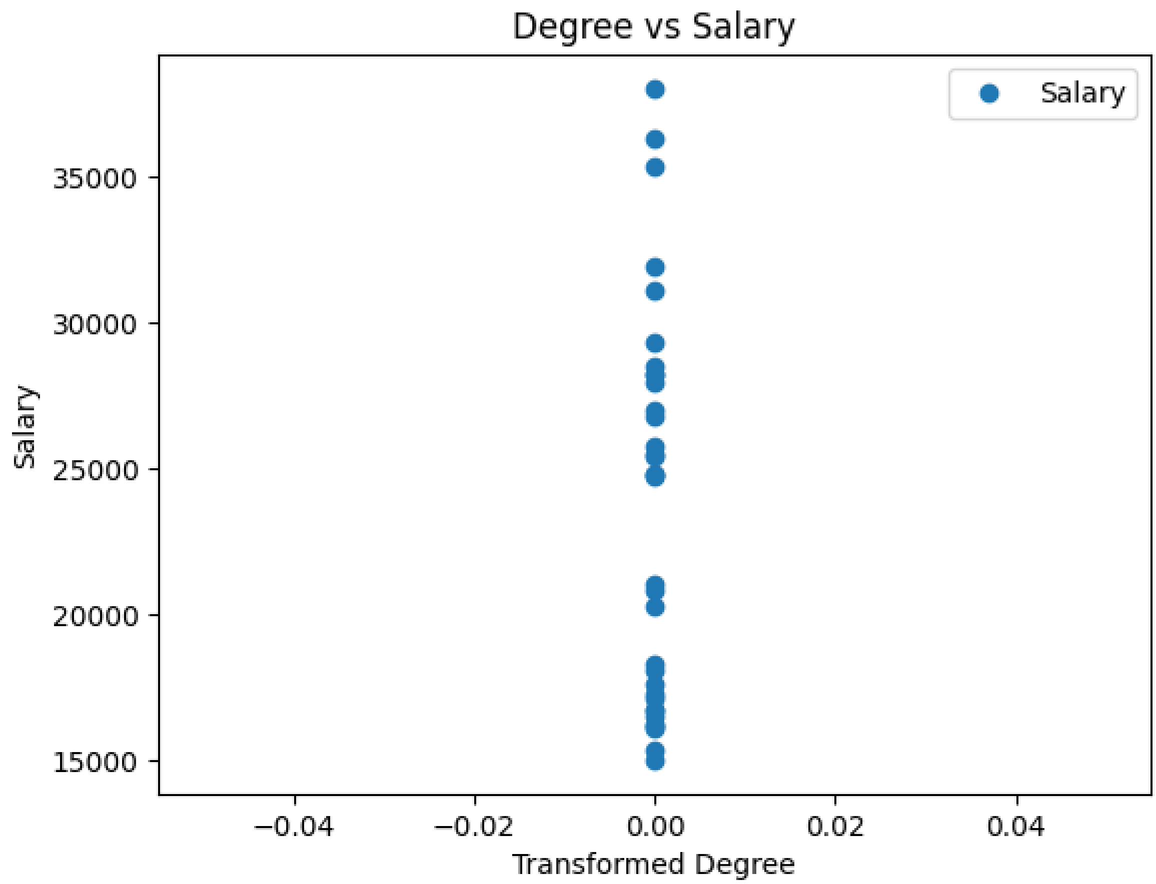
In [111…
```
df[['Degree','Salary']].corr()
```

Out[111]:

	Degree	Salary
Degree	1.000000	-0.069726
Salary	-0.069726	1.000000

In [115…
```
df['transformed'] = np.log(df['Degree']) # transformation
df.plot(x='transformed', y='Salary', style='o')
plt.title('Degree vs Salary')
plt.xlabel('Transformed Degree')
plt.ylabel('Salary')
plt.show()
df[['transformed','Salary']].corr()
```

```
C:\Users\Karthi\PycharmProjects\pythonProject\venv\lib\site-packages\pandas\core\arraylike.py:40
2: RuntimeWarning: divide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

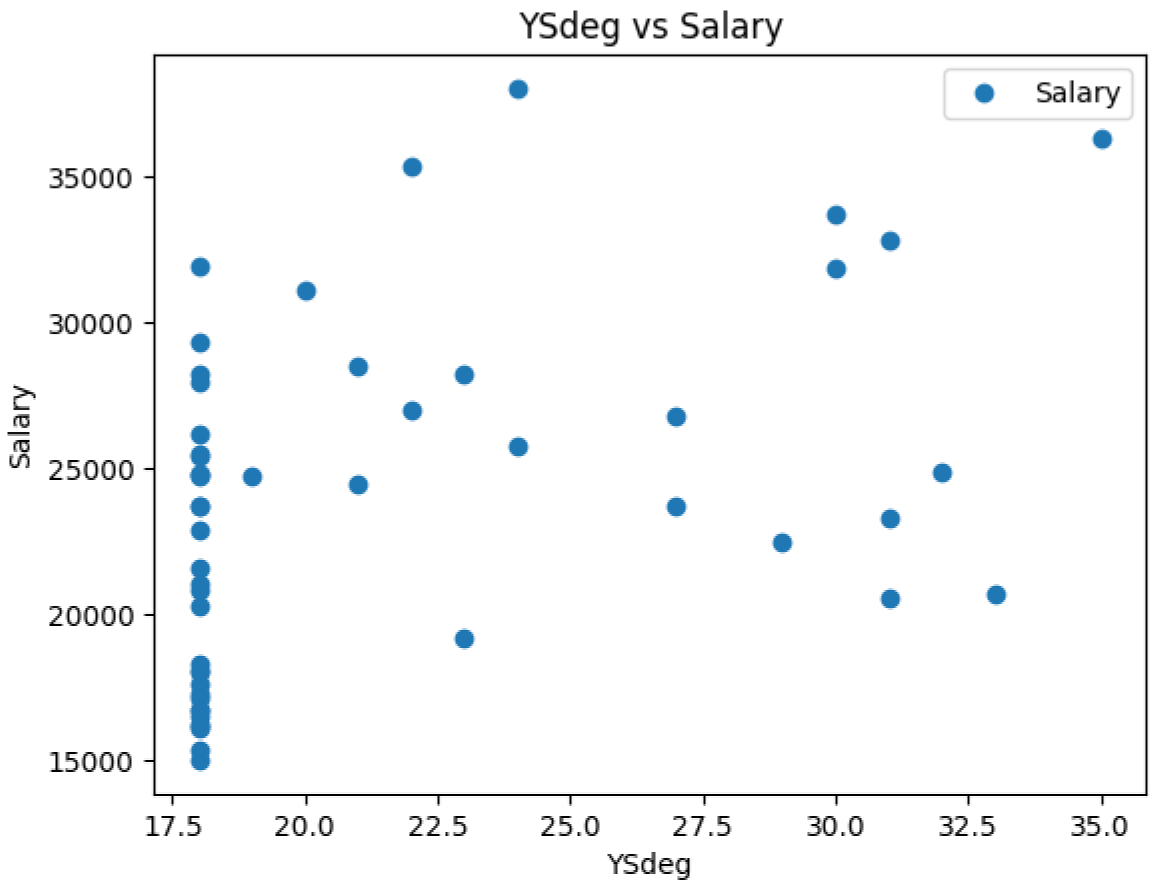


Out[115]:

	transformed	Salary
transformed	NaN	NaN
Salary	NaN	1.0

In [112…

```
df.plot(x='YSdeg', y='Salary', style='o')
plt.title('YSdeg vs Salary')
plt.xlabel('YSdeg')
plt.ylabel('Salary')
plt.show()
```



In [113…
```
df[['YSdeg','Salary']].corr()
```

Out[113]:

	YSdeg	Salary
YSdeg	1.000000	0.425932
Salary	0.425932	1.000000

In [116…
```
X = df[['Sex', 'Rank', 'Year','Degree','YSdeg']].values #array of features
y = df['Salary'].values #array of targets
```

In [117…
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

In [118…
```
from sklearn.preprocessing import StandardScaler ## standrard scalig
scaler = StandardScaler() #initialise to a variable
scaler.fit(X_train) # we are finding the values of mean and sd from the td
X_train_scaled = scaler.transform(X_train) # fit (mean, sd) and then transform the training data
X_test_scaled = scaler.transform(X_test) # transform the test data
```

In [119…
```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train_scaled, y_train)
```

Out[119]:

▾ LinearRegression

LinearRegression()

In [124…
```
coeff_df = pd.DataFrame(regressor.coef_,['Sex', 'Rank',
        'Year','Degree','YSdeg'], columns=['Coefficient'])
y_pred = regressor.predict(X_test_scaled)
coeff_df
```

Out[124]:

	Coefficient
Sex	575.834058
Rank	4284.543201
Year	2192.665752
Degree	-140.935024
YSdeg	-432.991839

In [125…

```
regressor.predict(scaler.transform(np.array([[1,5,3,4,2]])))
```

Out[125]: array([38642.66788006])

In [126…

```
regressor.intercept_
```

Out[126]: 23840.756097560974

In [127…

```
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df
```

Out[127]:

	Actual	Predicted
0	25748	28725.230737
1	31114	30656.863413
2	20525	23017.084135
3	16150	17729.725379
4	16500	16430.081892
5	32850	30016.273613
6	25500	28941.593561
7	20999	20642.760596
8	31909	32020.752719
9	15000	17331.652411
10	23712	23754.497909

In [129…

```
from sklearn import metrics
print('R2- SCORE:', metrics.r2_score(y_test,y_pred))
```

R2- SCORE: 0.8959110210516464

In [ ]: