```
19IT006 - rutvik balar
```

PRACTICAL - 3

Linear Regression Select Dataset of your choice and respond to following questions.

- Why you want to apply regression on selected dataset? Discuss full story behind dataset.
- How many total observations in data?
- How many independent variables?
- Which is dependent variable?
- Which are most useful variable in estimation? Prove using correlation.
- Implement linear regression using OLS method.
- Implement linear regression using Gradient Descent from scratch.
- Implement linear regression using sklearn API.
- Quantify goodness of your model and discuss steps taken for improvement (RMSE, MSE, R2Score).
- Discuss comparison of different methods.
- Prepare presentation for this work in group of 5 For help: refer following free course on datacamp. Regression models: fitting them and evaluating their performancea

Double-click (or enter) to edit

```
from sklearn.datasets import load_iris
from sklearn import model_selection
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import seaborn as sns
from sklearn import preprocessing, svm
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

```
house_price = pd.read_csv("house_price.csv")
house_price
```

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfr
0	2014-05-02 00:00:00	3.130000e+05	3.0	1.50	1340	7912	1.5	
1	2014-05-02 00:00:00	2.384000e+06	5.0	2.50	3650	9050	2.0	
2	2014-05-02 00:00:00	3.420000e+05	3.0	2.00	1930	11947	1.0	
3	2014-05-02 00:00:00	4.200000e+05	3.0	2.25	2000	8030	1.0	
4	2014-05-02 00:00:00	5.500000e+05	4.0	2.50	1940	10500	1.0	
...	...	...	...	...	...	...	...	
4595	2014-07-09 00:00:00	3.081667e+05	3.0	1.75	1510	6360	1.0	
4596	2014-07-09 00:00:00	5.343333e+05	3.0	2.50	1460	7573	2.0	
4597	2014-07-09 00:00:00	4.169042e+05	3.0	2.50	3010	7014	2.0	
4598	2014-07-10 00:00:00	2.034000e+05	4.0	2.00	2090	6630	1.0	
4599	2014-07-10 00:00:00	2.206000e+05	3.0	2.50	1490	8102	2.0	

```
# handle the missing value
house_price.isnull().sum()
```

```
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
```

```
condition        0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
street           0
city             0
statezip         0
country          0
dtype: int64
```

How many total observations in data?

```
total_observations = np.shape(house_price)
total_observations
```

```
(4600, 18)
```

How many independent variables?

```
X = house_price.iloc[:, -1]
X
```

```
0       USA
1       USA
2       USA
3       USA
4       USA
       ...
4595    USA
4596    USA
4597    USA
4598    USA
4599    USA
Name: country, Length: 4600, dtype: object
```

Which is dependent variable?

```
Y = house_price.iloc[:, 3]
Y
```

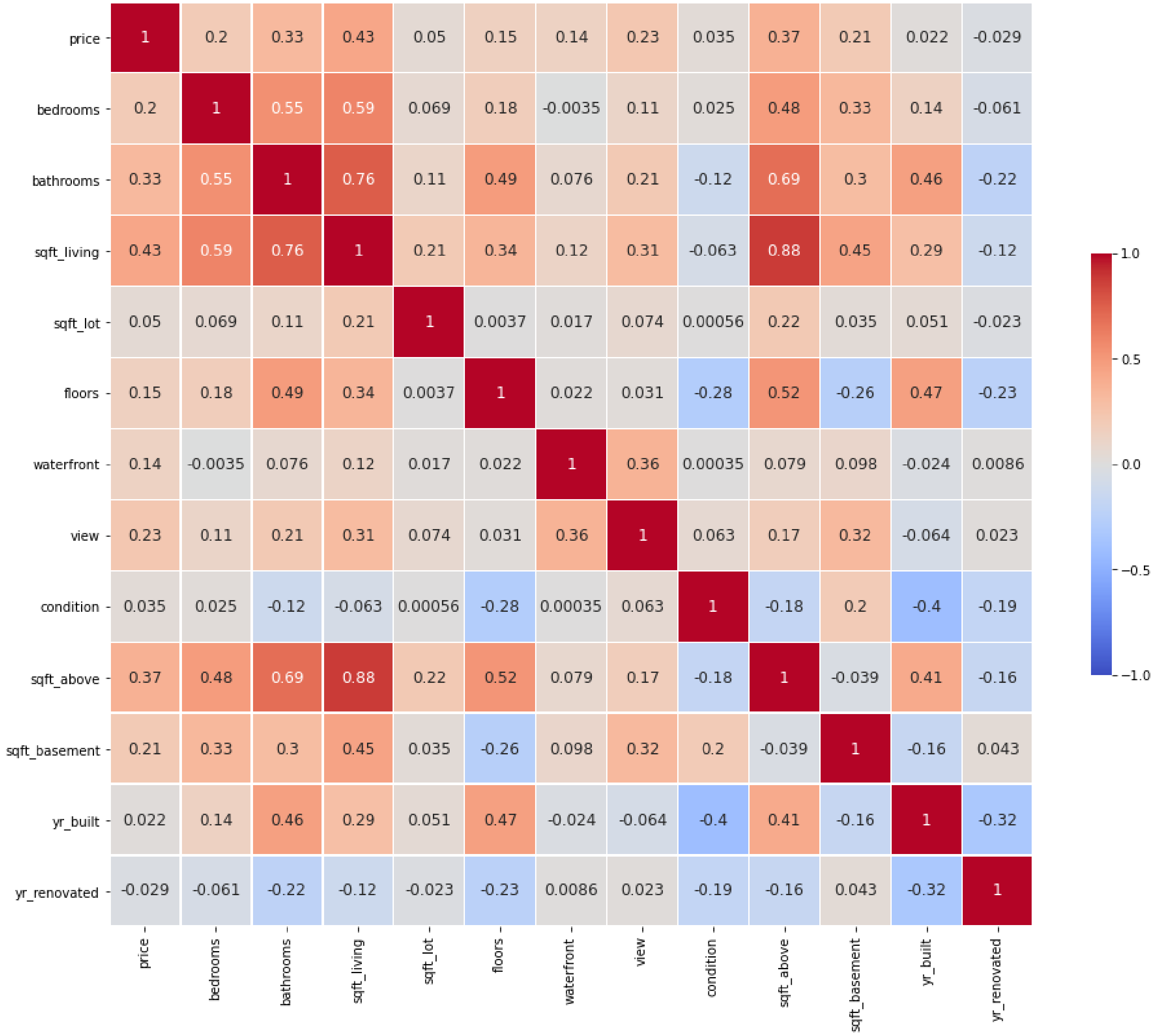
```
0       1.50
1       2.50
2       2.00
```

```
3       2.25
4       2.50
        ...
4595    1.75
4596    2.50
4597    2.50
4598    2.00
4599    2.50
Name: bathrooms, Length: 4600, dtype: float64
```

Which are most useful variable in estimation? Prove using correlation.

```
import seaborn as sns
import matplotlib.pyplot as plt

%matplotlib inline


# correlation Matrix
corr_matrix = house_price.corr()

# plotting headmap
pf, ax = plt.subplots(figsize=(16, 15))

heatmap = sns.heatmap(corr_matrix,
                      square = True,
                      linewidths = .5,
                      cmap = 'coolwarm',
                      cbar_kws = {'shrink': .4,
                                'ticks' : [-1, -.5, 0, 0.5, 1]},
                      vmin = -1,
                      vmax = 1,
                      annot = True,
                      annot_kws = {"size": 12})
#add the column names as labels
ax.set_yticklabels(corr_matrix.columns, rotation = 0)
ax.set_xticklabels(corr_matrix.columns)
```

```
[Text(0.5, 0, 'price'),
 Text(1.5, 0, 'bedrooms'),
 Text(2.5, 0, 'bathrooms'),
 Text(3.5, 0, 'sqft_living'),
 Text(4.5, 0, 'sqft_lot'),
 Text(5.5, 0, 'floors'),
 Text(6.5, 0, 'waterfront'),
 Text(7.5, 0, 'view'),
 Text(8.5, 0, 'condition'),
 Text(9.5, 0, 'sqft_above'),
 Text(10.5, 0, 'sqft_basement'),
 Text(11.5, 0, 'yr_built'),
 Text(12.5, 0, 'yr_renovated')]
```



	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_above	sqft_basement	yr_built	yr_renovated
price	1	0.2	0.33	0.43	0.05	0.15	0.14	0.23	0.035	0.37	0.21	0.022	-0.029
bedrooms	0.2	1	0.55	0.59	0.069	0.18	-0.0035	0.11	0.025	0.48	0.33	0.14	-0.061
bathrooms	0.33	0.55	1	0.76	0.11	0.49	0.076	0.21	-0.12	0.69	0.3	0.46	-0.22
sqft_living	0.43	0.59	0.76	1	0.21	0.34	0.12	0.31	-0.063	0.88	0.45	0.29	-0.12
sqft_lot	0.05	0.069	0.11	0.21	1	0.0037	0.017	0.074	0.00056	0.22	0.035	0.051	-0.023
floors	0.15	0.18	0.49	0.34	0.0037	1	0.022	0.031	-0.28	0.52	-0.26	0.47	-0.23
waterfront	0.14	-0.0035	0.076	0.12	0.017	0.022	1	0.36	0.00035	0.079	0.098	-0.024	0.0086
view	0.23	0.11	0.21	0.31	0.074	0.031	0.36	1	0.063	0.17	0.32	-0.064	0.023
condition	0.035	0.025	-0.12	-0.063	0.00056	-0.28	0.00035	0.063	1	-0.18	0.2	-0.4	-0.19
sqft_above	0.37	0.48	0.69	0.88	0.22	0.52	0.079	0.17	-0.18	1	-0.039	0.41	-0.16
sqft_basement	0.21	0.33	0.3	0.45	0.035	-0.26	0.098	0.32	0.2	-0.039	1	-0.16	0.043
yr_built	0.022	0.14	0.46	0.29	0.051	0.47	-0.024	-0.064	-0.4	0.41	-0.16	1	-0.32
yr_renovated	-0.029	-0.061	-0.22	-0.12	-0.023	-0.23	0.0086	0.023	-0.19	-0.16	0.043	-0.32	1

Implement linear regression using OLS method.

```
import statsmodels.api as sm


x = house_price['price'].tolist()
y = house_price['bedrooms'].tolist()

x
y
```

```
[3.0,
 5.0,
 3.0,
 3.0,
 4.0,
 2.0,
 2.0,
 4.0,
 3.0,
 4.0,
 3.0,
 4.0,
 3.0,
 3.0,
 5.0,
 3.0,
 3.0,
 4.0,
 3.0,
 3.0,
 3.0,
 4.0,
 3.0,
 4.0,
 4.0,
 3.0,
 3.0,
 4.0,
 5.0,
 3.0,
 4.0,
 4.0,
 4.0,
 4.0,
 4.0,
 3.0,
 2.0,
 3.0,
 3.0,
 3.0,
 3.0,
 4.0,
 2.0,
 3.0,
 2.0,
 3.0,
 3.0,
       3.0,
       3.0,
       4.0,
       4.0,
       3.0,
       3.0,
       3.0,
       5.0,
       4.0,
       4.0,
       5.0,
```

```
# x = sm.add_constant(x)
# x
```

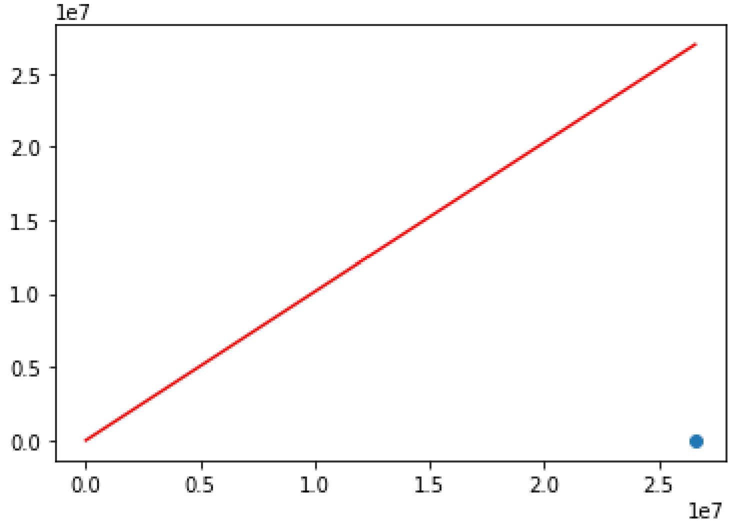
```
array([[1.00000000e+00, 3.13000000e+05],
       [1.00000000e+00, 2.38400000e+06],
       [1.00000000e+00, 3.42000000e+05],
       ...,
       [1.00000000e+00, 4.16904167e+05],
       [1.00000000e+00, 2.03400000e+05],
       [1.00000000e+00, 2.20600000e+05]])
```

```
result = sm.OLS(y, x).fit()
result.summary()
```

```
                          OLS Regression Results
     Dep. Variable:   y                        R-squared:   0.040
             Model:   OLS                 Adj. R-squared:   0.040
            Method:   Least Squares          F-statistic:   192.3
              Date:   Thu, 13 Jan 2022 Prob (F-statistic):  7.38e-43
              Time:   09:59:02            Log-Likelihood:   -5992.7
  No. Observations:   4600                           AIC:   1.199e+04
      Df Residuals:   4598                           BIC:   1.200e+04
          Df Model:   1
   Covariance Type:   nonrobust

            coef       std err     t        P>|t|  [0.025     0.975]
  const  3.2226     0.018     175.380  0.000  3.187      3.259
    x1   3.229e-07  2.33e-08  13.866   0.000  2.77e-07   3.69e-07

        Omnibus:     399.003  Durbin-Watson:     2.014
  Prob(Omnibus):     0.000    Jarque-Bera (JB):  2181.660
           Skew:     0.214         Prob(JB):     0.00
       Kurtosis:     6.346        Cond. No.      1.10e+06
```

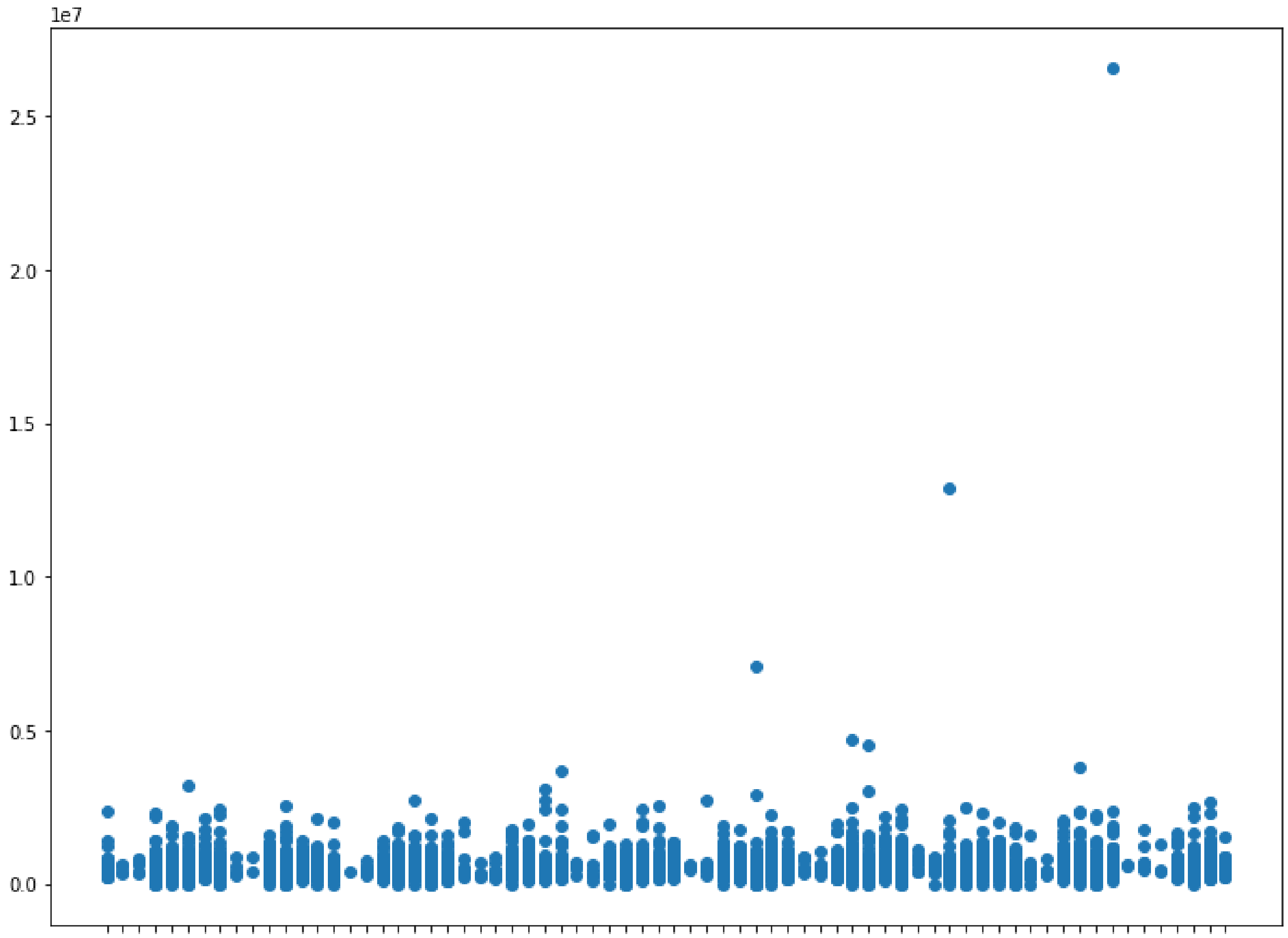
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.1e+06. This might indicate that there are
strong multicollinearity or other numerical problems.

```
max_x = house_price['price'].max()
min_x = house_price['bedrooms'].min()

plt.scatter(max_x, min_x)

# range of values for plotting
# the regression line
x = np.arange(min_x, max_x, 1)

# the substituted equation
y = 1.0143 * x - 0.4618

# plotting the regression line
plt.plot(y, 'r')
plt.show()
```



Implement linear regression using Gradient Descent from scratch

```
plt.rcParams['figure.figsize'] = (12.0, 9.0)

# Preprocessing Input data
data = pd.read_csv('house_price.csv')
X = data.iloc[:, 0]
Y = data.iloc[:, 1]
plt.scatter(X, Y)
plt.show()
```



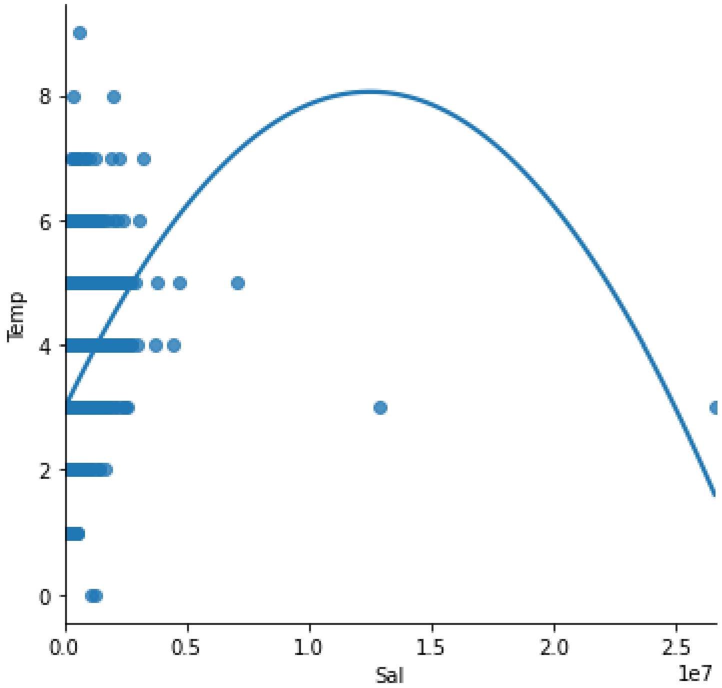
Implement linear regression using sklearn API.

```
df = pd.read_csv('house_price.csv')
df_binary = df[['price', 'bedrooms']]

# Taking only the selected two attributes from the dataset
df_binary.columns = ['Sal', 'Temp']

# Renaming the columns for easier writing of the code
df_binary.head()
```

	Sal	Temp
0	313000.0	3.0
1	2384000.0	5.0
2	342000.0	3.0
3	420000.0	3.0
4	550000.0	4.0

```
sns.lmplot(x ="Sal", y ="Temp", data = df_binary, order = 2, ci = None)
```

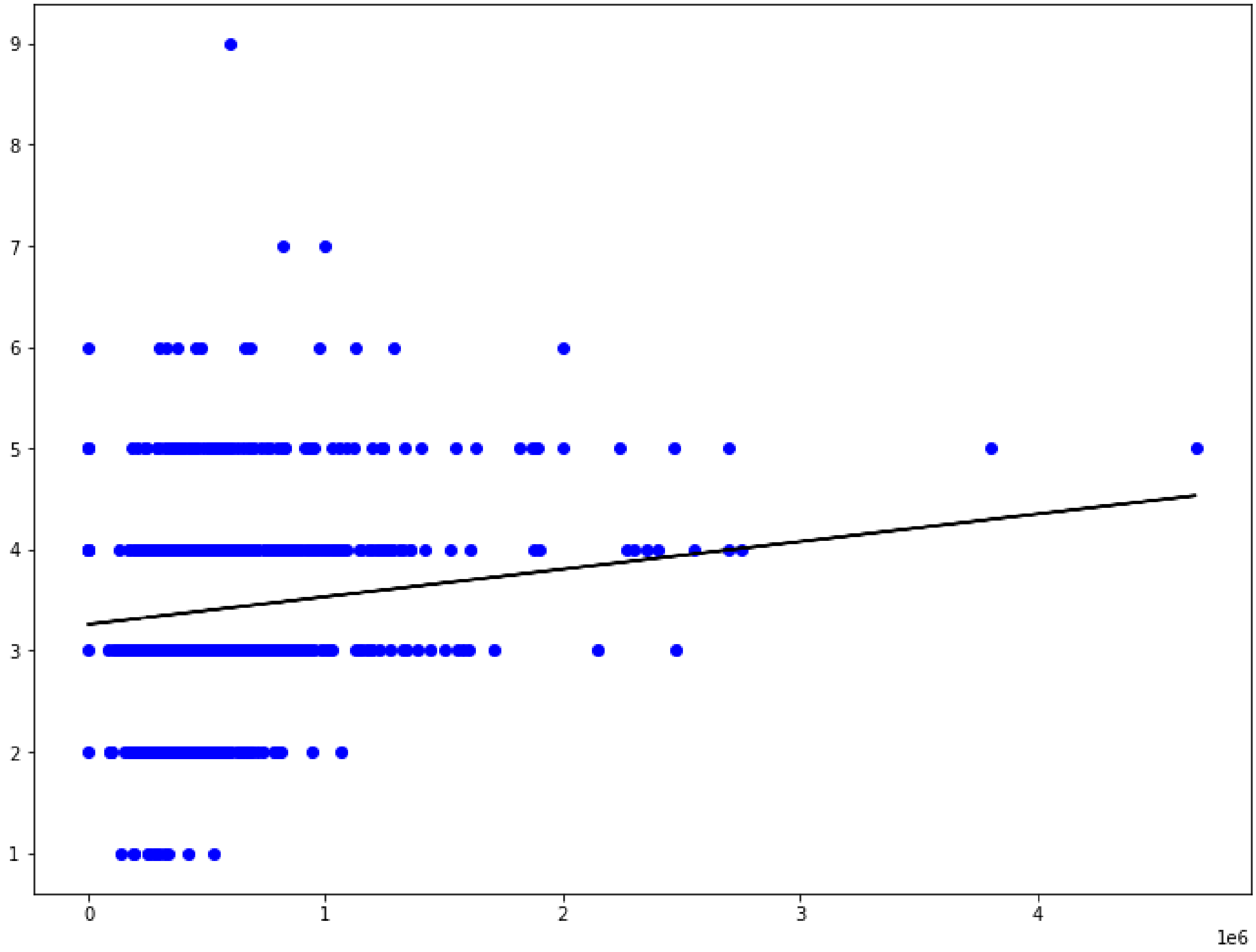
```
<seaborn.axisgrid.FacetGrid at 0x7fe63b194400>
```



```
import numpy as np
import pandas as pd

df = pd.read_csv('house_price.csv')
df_binary = df[['price', 'bedrooms']]

# Taking only the selected two attributes from the dataset
df_binary.columns = ['Sal', 'Temp']

# Renaming the columns for easier writing of the code
df_binary.head()

X = np.array(df_binary['Sal']).reshape(-1, 1)
y = np.array(df_binary['Temp']).reshape(-1, 1)

# Separating the data into independent and dependent variables
# Converting each dataframe into a numpy array
# since each dataframe contains only one column
df_binary.dropna(inplace = True)

# Dropping any rows with Nan values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25)

# Splitting the data into training and testing data
regr = LinearRegression()

regr.fit(X_train, y_train)
print(regr.score(X_test, y_test))
```

```
0.044428118614029066
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:19: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
```

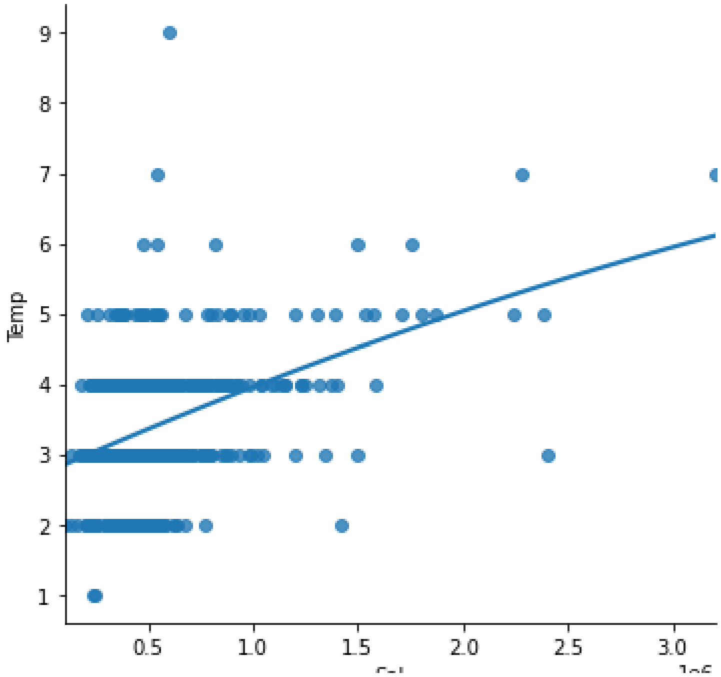
```
y_pred = regr.predict(X_test)
plt.scatter(X_test, y_test, color ='b')
plt.plot(X_test, y_pred, color ='k')

plt.show()
# Data scatt
```



```
df_binary500 = df_binary[:][:500]

# Selecting the 1st 500 rows of the data
```

```
sns.lmplot(x ="Sal", y ="Temp", data = df_binary500,
                               order = 2, ci = None)
```

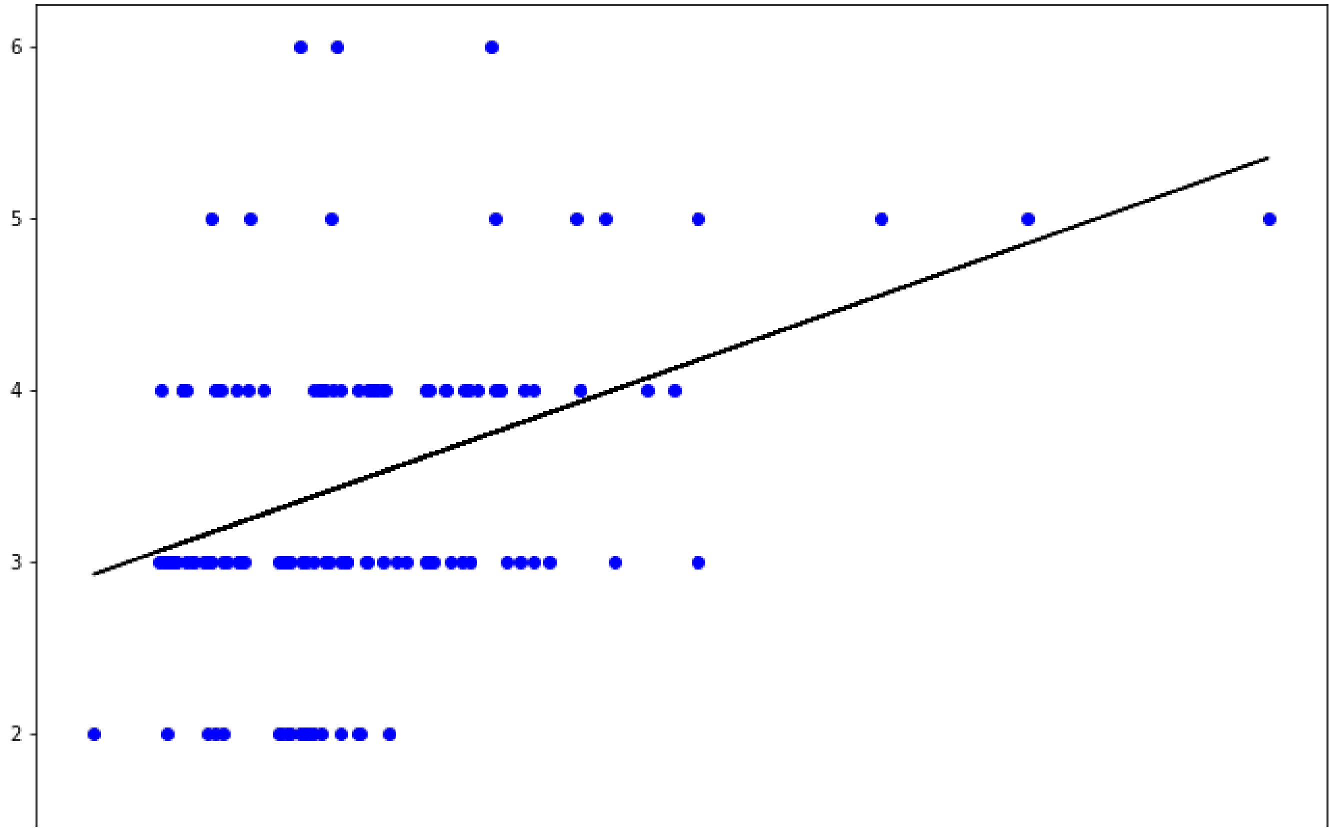
```
     <seaborn.axisgrid.FacetGrid at 0x7fe63b1956a0>
```



```
df_binary500.fillna(method ='ffill', inplace = True)

X = np.array(df_binary500['Sal']).reshape(-1, 1)
y = np.array(df_binary500['Temp']).reshape(-1, 1)

df_binary500.dropna(inplace = True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25)

regr = LinearRegression()
regr.fit(X_train, y_train)
print(regr.score(X_test, y_test))
```

```
     0.16156911072185953
```

```
y_pred = regr.predict(X_test)
plt.scatter(X_test, y_test, color ='b')
plt.plot(X_test, y_pred, color ='k')

plt.show()
```



Quantify goodness of your model and discuss steps taken for improvement (RMSE, MSE, R2Score)

RMSE

- Try to play with other input variables, and compare your RMSE values. The smaller the RMSE value, the better the model. Also, try to compare your RMSE values of both training and testing data. If they are almost similar, your model is good

MSE

- To minimize MSE, the model could be more accurate, which would mean the model is closer to actual data.

R2Score

- Adding more independent variables or predictors to a regression model tends to increase the R-squared value, which tempts makers of the model to add even more variables. This is called overfitting and can return an unwarranted high R-squared value.

Discuss comparison of different methods.

Unsupported Cell Type. Double-Click to inspect/edit the content.