In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
sns.set()
from sklearn.linear_model import LinearRegression
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
from sklearn import metrics
from scipy.stats import linregress
from numpy.linalg import inv
```

In [2]:

```
HW1data=pd.read_csv("Q1_data.csv")
```

Question 1a)

In [3]:

```
print(np.mean(HW1data['x1']),np.mean(HW1data['y1']),np.std(HW1data['x1']),np.std
(HW1data['y1']),(linregress(HW1data['x1'], HW1data['y1'])))
```

```
9.0 7.500909090909093 3.1622776601683795 1.937024215108669 Linregres
sResult(slope=0.5000909090909914, intercept=3.0000909090909103, rva
lue=0.8164205163448399 2, pvalue=0.002169628873078790 1, stderr=0.1179
0550059563408)
```

mean of X1: 9.0; mean of Y1: 7.5009; standard deviation of X1: 3.162; standard deviation of y1: 1.937; coefficient of correlation: 0.8164

In [4]:

```
print(np.mean(HW1data['x2']),np.mean(HW1data['y2']),np.std(HW1data['x2']),np.std
(HW1data['y2']),(linregress(HW1data['x2'], HW1data['y2'])))
```

```
9.0 7.500909090909091 3.1622776601683795 1.93710869148962 Linregress
Result(slope=0.5000000000000011, intercept=3.0009090909090892, rval
ue=0.816236506000243, pvalue=0.002178816236910784 5, stderr=0.1179637
4596764074)
```

mean of X2: 9.0; mean of Y2: 7.5009; standard deviation of X2: 3.162; standard deviation of y2: 1.937; coefficient of correlation: 0.8162

In [5]:

```
print(np.mean(HW1data['x3']),np.mean(HW1data['y3']),np.std(HW1data['x3']),np.std
(HW1data['y3']),(linregress(HW1data['x3'], HW1data['y3'])))
```

```
9.0 7.500000000000001 3.1622776601683795 1.9359329439927313 Linregre
ssResult(slope=0.49972727272727291, intercept=3.0024545454545439, rv
alue=0.8162867394895984, pvalue=0.0021763052792280152, stderr=0.1178
7766222100221)
```

mean of X3: 9.0; mean of Y3: 7.5; standard deviation of X3: 3.162; standard deviation of y3: 1.936; coefficient of correlation: 0.8163

In [6]:

```
print(np.mean(HW1data['x4']),np.mean(HW1data['y4']),np.std(HW1data['x4']),np.std
(HW1data['y4']),(linregress(HW1data['x4'], HW1data['y4'])))
```

```
9.0 7.50090909090909 3.1622776601683795 1.9360806451340837 Linregres
sResult(slope=0.49990909090909091, intercept=3.0017272727272726, rva
lue=0.81652143688850276, pvalue=0.0021646023471972222, stderr=0.1178
1894172968553)
```

mean of X4: 9.0; mean of Y4: 7.5; standard deviation of X4: 3.162; standard deviation of y4: 1.936; coefficient of correlation: 0.8165

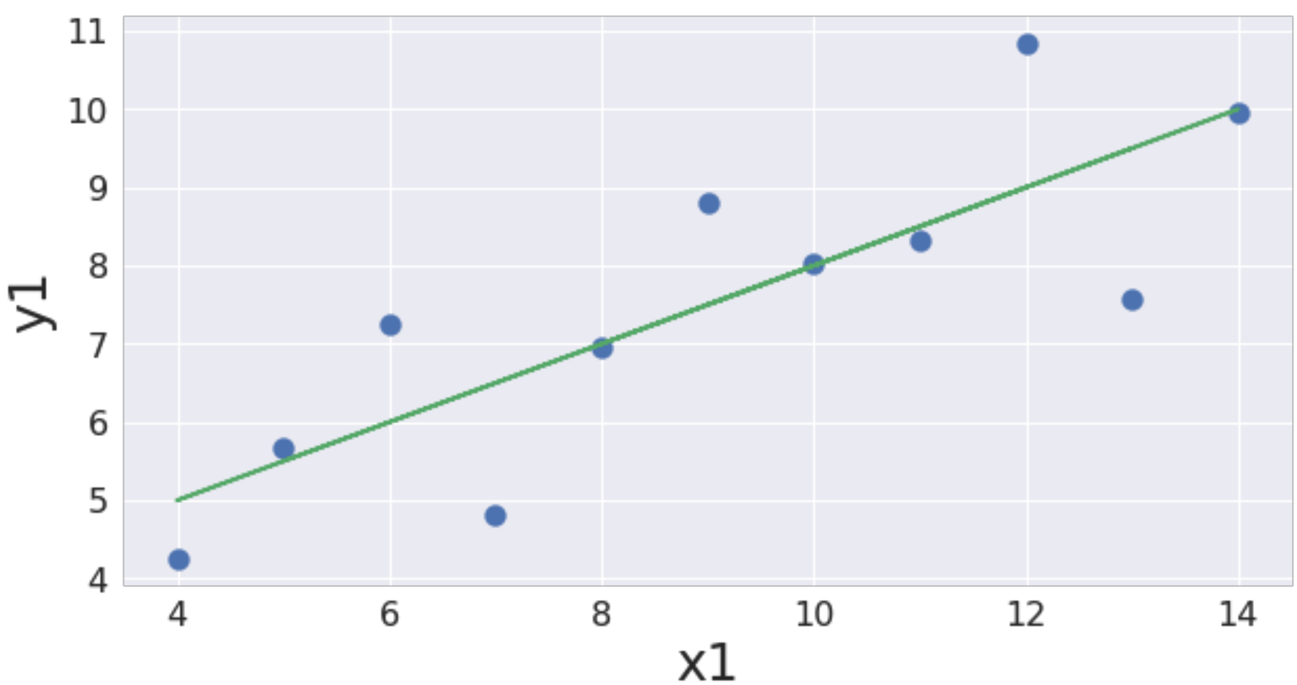
Question 1b)

In [7]:

```
X = np.array([[10, 0], [8, 0], [13, 0], [9, 0],[11, 0],[14, 0],[6, 0],[4, 0],[12
, 0],[7, 0],[5, 0]])
LinearRegression1 = LinearRegression().fit(X, HW1data['y1'].values)
Slope1=LinearRegression1.coef_[0]
Intercept1=LinearRegression1.intercept_
fig=plt.figure(figsize=(10,5))
plt.scatter(HW1data['x1'], HW1data['y1'], s=100)
plt.ylabel('y1', fontsize = 25)
plt.yticks(fontsize=16)
plt.xlabel('x1', fontsize=25)
plt.xticks(fontsize=16)
Line1 = Slope1*HW1data['x1']+Intercept1
plt.plot(HW1data['x1'],HW1data['y1'],'o',HW1data['x1'],Line1)
```

Out[7]:

```
[<matplotlib.lines.Line2D at 0x1a1bb43be0>,
 <matplotlib.lines.Line2D at 0x1a1bb4a2e8>]
```

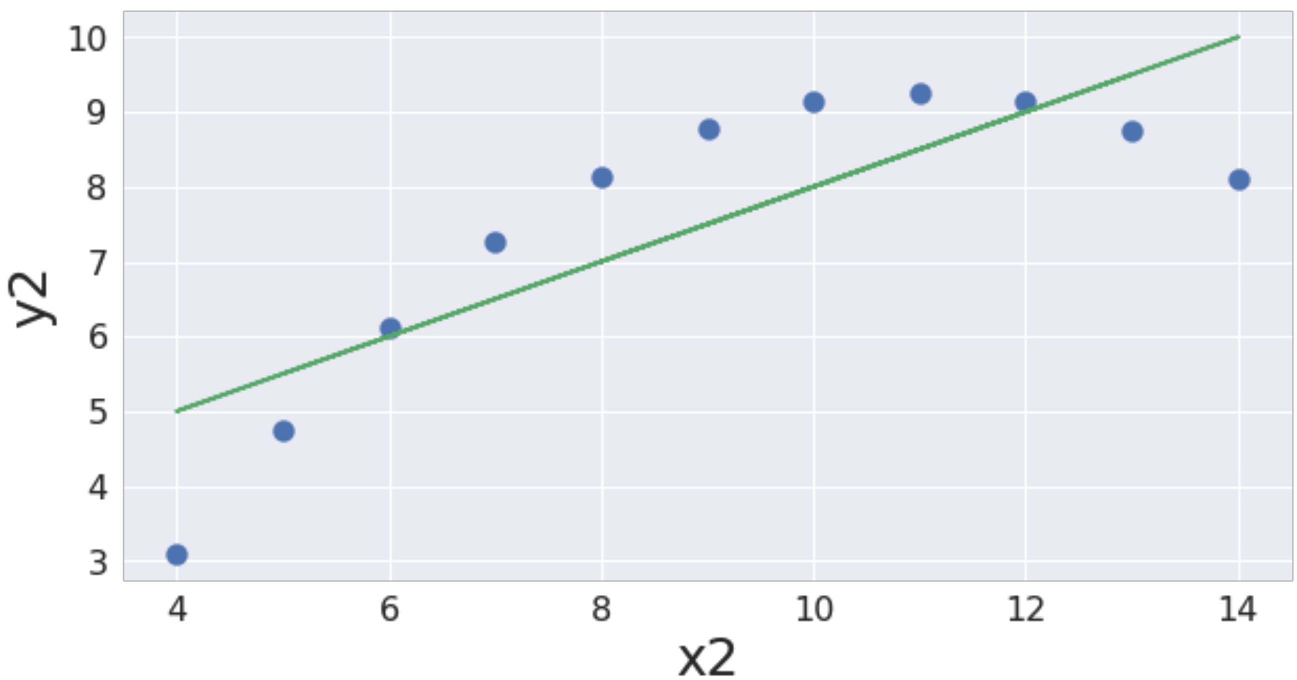


In [8]:

```
LinearRegression2 = LinearRegression().fit(X, HW1data['y2'].values)
Slope2=LinearRegression2.coef_[0]
Intercept2=LinearRegression2.intercept_
fig=plt.figure(figsize=(10,5))
plt.scatter(HW1data['x2'], HW1data['y2'], s=100)
plt.ylabel('y2', fontsize = 25)
plt.yticks(fontsize=16)
plt.xlabel('x2', fontsize=25)
plt.xticks(fontsize=16)
Line2 = Slope2*HW1data['x2']+Intercept2
plt.plot(HW1data['x2'],HW1data['y2'],'o',HW1data['x2'],Line2)
```

Out[8]:

```
[<matplotlib.lines.Line2D at 0x1a1bd78978>,
 <matplotlib.lines.Line2D at 0x1a1bd78cc0>]
```

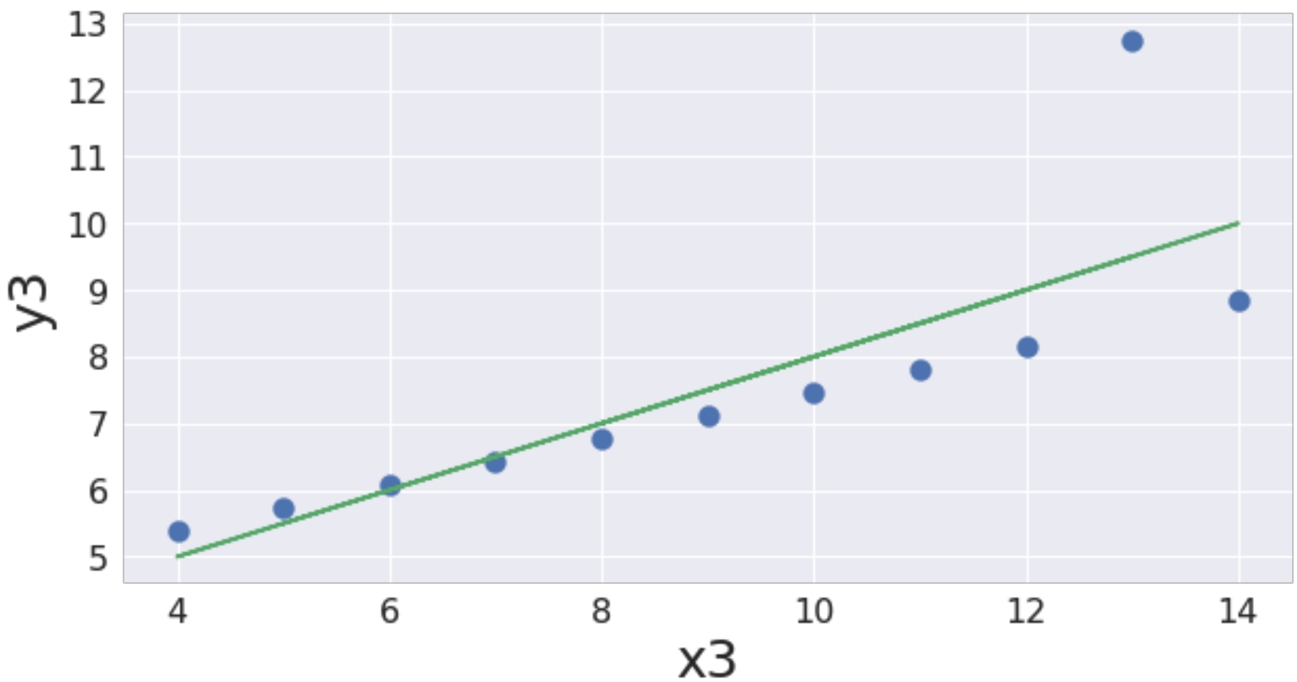


In [9]:

```
LinearRegression3 = LinearRegression().fit(X, HW1data['y3'].values)
Slope3=LinearRegression3.coef_[0]
Intercept3=LinearRegression3.intercept_
fig=plt.figure(figsize=(10,5))
plt.scatter(HW1data['x3'], HW1data['y3'], s=100)
plt.ylabel('y3', fontsize = 25)
plt.yticks(fontsize=16)
plt.xlabel('x3', fontsize=25)
plt.xticks(fontsize=16)
Line3 = Slope3*HW1data['x3']+Intercept3
plt.plot(HW1data['x3'],HW1data['y3'],'o',HW1data['x3'],Line3)
```

Out[9]:

[<matplotlib.lines.Line2D at 0x1a1beb66a0>,
 <matplotlib.lines.Line2D at 0x1a1beb69e8>]

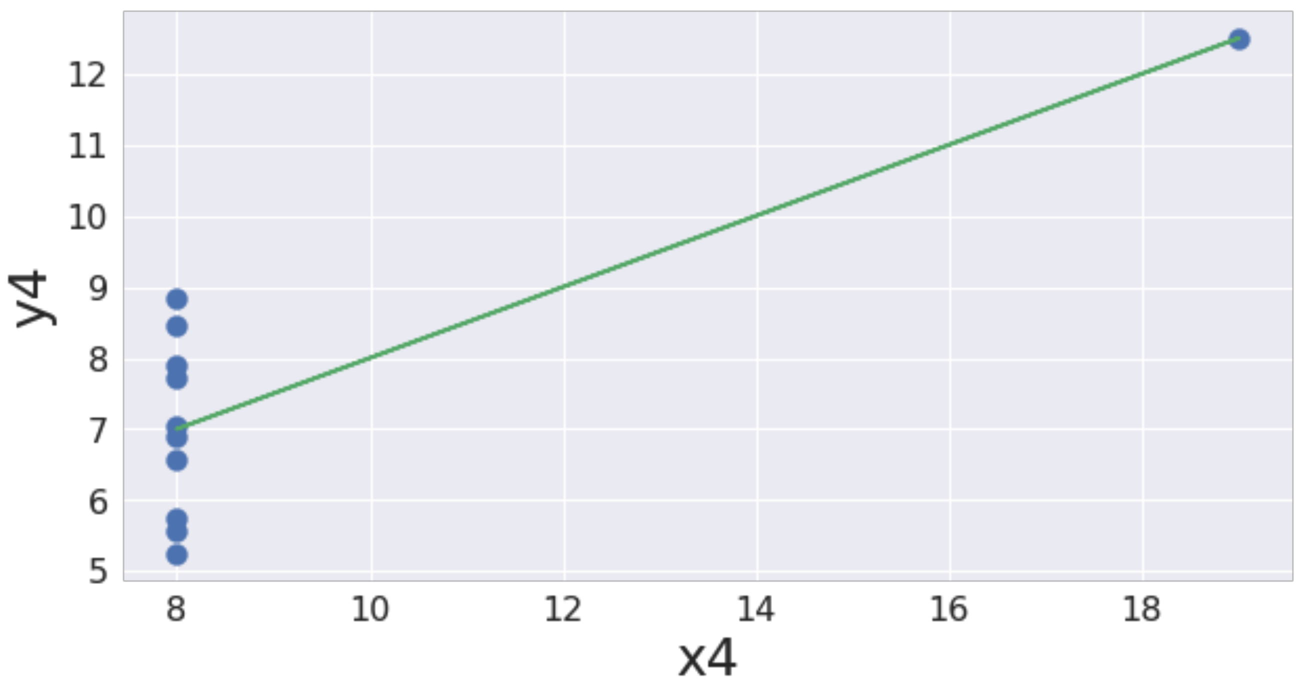


In [10]:

```
X4=np.array([[8, 0], [8, 0], [8, 0], [8, 0],[8, 0],[8, 0],[8, 0],[19, 0],[8, 0],
[8, 0],[8, 0]])
LinearRegression4 = LinearRegression().fit(X4, HW1data['y4'].values)
Slope4=LinearRegression4.coef_[0]
Intercept4=LinearRegression3.intercept_
fig=plt.figure(figsize=(10,5))
plt.scatter(HW1data['x4'], HW1data['y4'], s=100)
plt.ylabel('y4', fontsize = 25)
plt.yticks(fontsize=16)
plt.xlabel('x4', fontsize=25)
plt.xticks(fontsize=16)
Line4 = Slope4*HW1data['x4']+Intercept4
plt.plot(HW1data['x4'],HW1data['y4'],'o',HW1data['x4'],Line4)
```

Out[10]:

```
[<matplotlib.lines.Line2D at 0x1a1bff0e10>,
 <matplotlib.lines.Line2D at 0x1a1bff0470>]
```



question 1c)

In [12]:

```
y1 = []
y1.append(HW1data['y1'])
y1_pred = LinearRegression1.predict(X)
print(Slope1)
print(Intercept1)
print(np.sum(np.square(np.subtract(y1,np.mean(HW1data['y1'])))))
print(np.sum(np.square(np.subtract(y1,y1_pred))))
print(np.sum(np.square(np.subtract(y1_pred,np.mean(HW1data['y1'])))))
print(np.sum(np.square(np.subtract(y1_pred,np.mean(HW1data['y1']))))/np.sum(np.s
quare(np.subtract(y1,np.mean(HW1data['y1'])))))
```

```
0.500090909091
3.00009090909
41.2726909091
13.76269
27.5100009091
0.666542459509
```

For (x1,y1): slope = 0.50009, intercept = 3.00009, SSTO = 41.2727, SSR = 13.763, SSE = 27.51, coefficient of multiple determination = 0.6665

In [13]:

```
y2 = []
y2.append(HW1data['y2'])
y2_pred = LinearRegression2.predict(X)
print(Slope2)
print(Intercept2)
print(np.sum(np.square(np.subtract(y2,np.mean(HW1data['y2'])))))
print(np.sum(np.square(np.subtract(y2,y2_pred))))
print(np.sum(np.square(np.subtract(y2_pred,np.mean(HW1data['y2'])))))
print(np.sum(np.square(np.subtract(y2_pred,np.mean(HW1data['y2']))))/np.sum(np.s
quare(np.subtract(y2,np.mean(HW1data['y2'])))))
```

```
0.5
3.00090909091
41.2762909091
13.7762909091
27.5
0.666242033727
```

For (x2,y2): slope = 0.5, intercept = 3.0009, SSTO = 41.2763, SSR = 13.776, SSE = 27.5, coefficient of multiple determination = 0.6662

In [14]:

```
y3 = []
y3.append(HW1data['y3'])
y3_pred = LinearRegression3.predict(X)
print(Slope3)
print(Intercept3)
print(np.sum(np.square(np.subtract(y3,np.mean(HW1data['y3'])))))
print(np.sum(np.square(np.subtract(y3,y3_pred))))
print(np.sum(np.square(np.subtract(y3_pred,np.mean(HW1data['y3'])))))
print(np.sum(np.square(np.subtract(y3_pred,np.mean(HW1data['y3']))))/np.sum(np.s
quare(np.subtract(y3,np.mean(HW1data['y3'])))))
```

```
0.499727272727
3.00245454545
41.2262
13.7561918182
27.4700081818
0.666324041067
```

For (x3,y3): slope = 0.4997, intercept = 3.0025, SSTO = 41.2262, SSR = 13.756, SSE = 27.47, coefficient of multiple determination = 0.6663

In [15]:

```
y4 = []
y4.append(HW1data['y4'])
y4_pred = LinearRegression4.predict(X4)
print(Slope4)
print(Intercept4)
print(np.sum(np.square(np.subtract(y4,np.mean(HW1data['y4'])))))
print(np.sum(np.square(np.subtract(y4,y4_pred))))
print(np.sum(np.square(np.subtract(y4_pred,np.mean(HW1data['y4'])))))
print(np.sum(np.square(np.subtract(y4_pred,np.mean(HW1data['y4']))))/np.sum(np.s
quare(np.subtract(y4,np.mean(HW1data['y4'])))))
```

```
0.499909090909
3.00245454545
41.2324909091
13.74249
27.4900009091
0.666707256898
```

For (x4,y4): slope = 0.4999, intercept = 3.0025, SSTO = 41.2325, SSR = 13.7425, SSE = 27.49, coefficient of multiple determination = 0.6667

1d) The summary statistics, coefficient of correlation, slope or regression and coefficient of multiple determination are NOT sufficient to judge the quality of fit of linear regression. I made this conclusion because I observed that for all these 4 datasets, all of these values are roughly the same even though the data are very different.

2a)

In [16]:

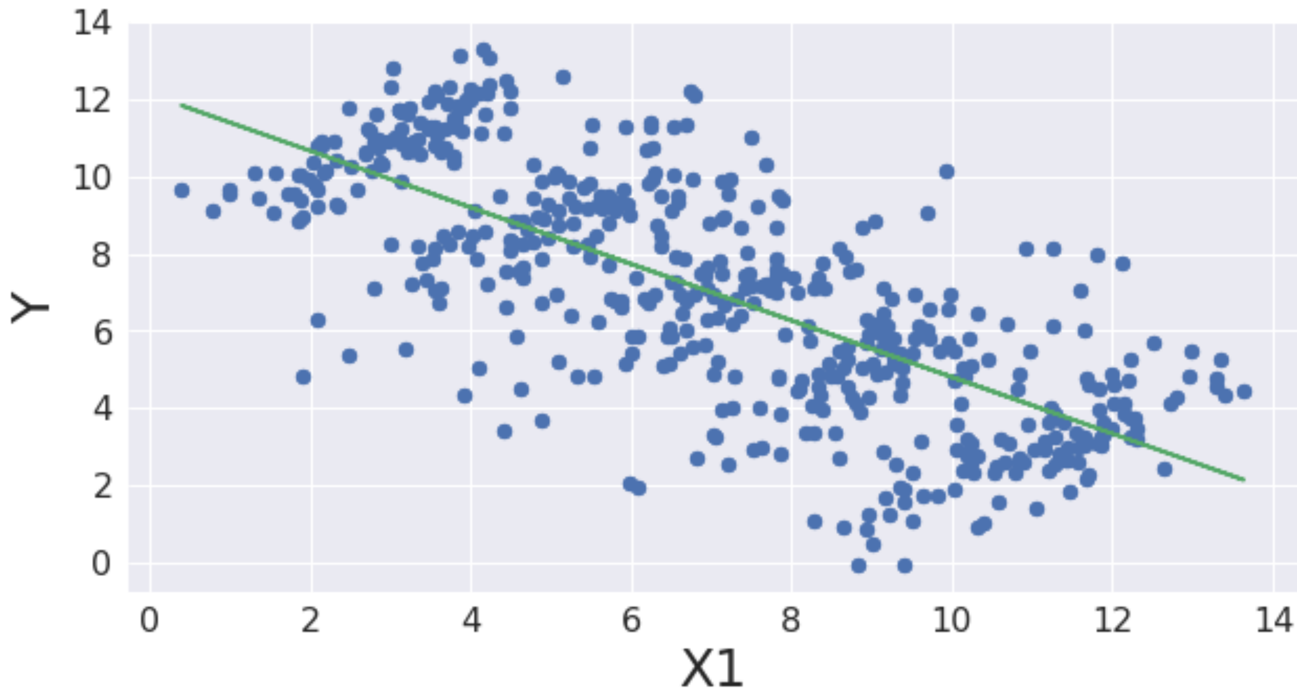
```
HW2data=pd.read_csv("Q2_data.csv")
```

In [17]:

```
Q2_X1 = np.zeros((500,2))
for i in range(500):
    Q2_X1[i][0] = HW2data['X1'][i]
LinearRegression_2a = LinearRegression().fit(Q2_X1, HW2data['Y'].values)
Slope_2a=LinearRegression_2a.coef_[0]
Intercept_2a=LinearRegression_2a.intercept_
fig=plt.figure(figsize=(10,5))
plt.scatter(HW2data['X1'], HW2data['Y'], s=50)
plt.ylabel('Y', fontsize = 25)
plt.yticks(fontsize=16)
plt.xlabel('X1', fontsize=25)
plt.xticks(fontsize=16)
Line_2a = Slope_2a*HW2data['X1']+Intercept_2a
plt.plot(HW2data['X1'],HW2data['Y'],'o',HW2data['X1'],Line_2a)
```

Out[17]:

```
[&lt;matplotlib.lines.Line2D at 0x1a278a5a20&gt;,
 &lt;matplotlib.lines.Line2D at 0x1a278a5da0&gt;]
```



In [18]:

```
y_2a = []
y_2a.append(HW2data['Y'])
y_2a_pred = LinearRegression_2a.predict(Q2_X1)
print(Slope_2a)
print(Intercept_2a)
print(np.sum(np.square(np.subtract(y_2a,np.mean(HW2data['Y'])))))
print(np.sum(np.square(np.subtract(y_2a,y_2a_pred))))
print(np.sum(np.square(np.subtract(y_2a_pred,np.mean(HW2data['Y'])))))
print(np.sum(np.square(np.subtract(y_2a_pred,np.mean(HW2data['Y']))))/np.sum(np.
square(np.subtract(y_2a,np.mean(HW2data['Y'])))))
```

```
-0.732087336713
12.1301609462
4667.38580111
2151.42326294
2515.96253817
0.539051761603
```

The linear model: linear regression. Slope = -0.7321, intercept = 12.1302, SSTO = 4667.39, SSR = 2151.42, SSE = 2515.96, coefficient of multiple determination = 0.53905. The model does NOT fit well.

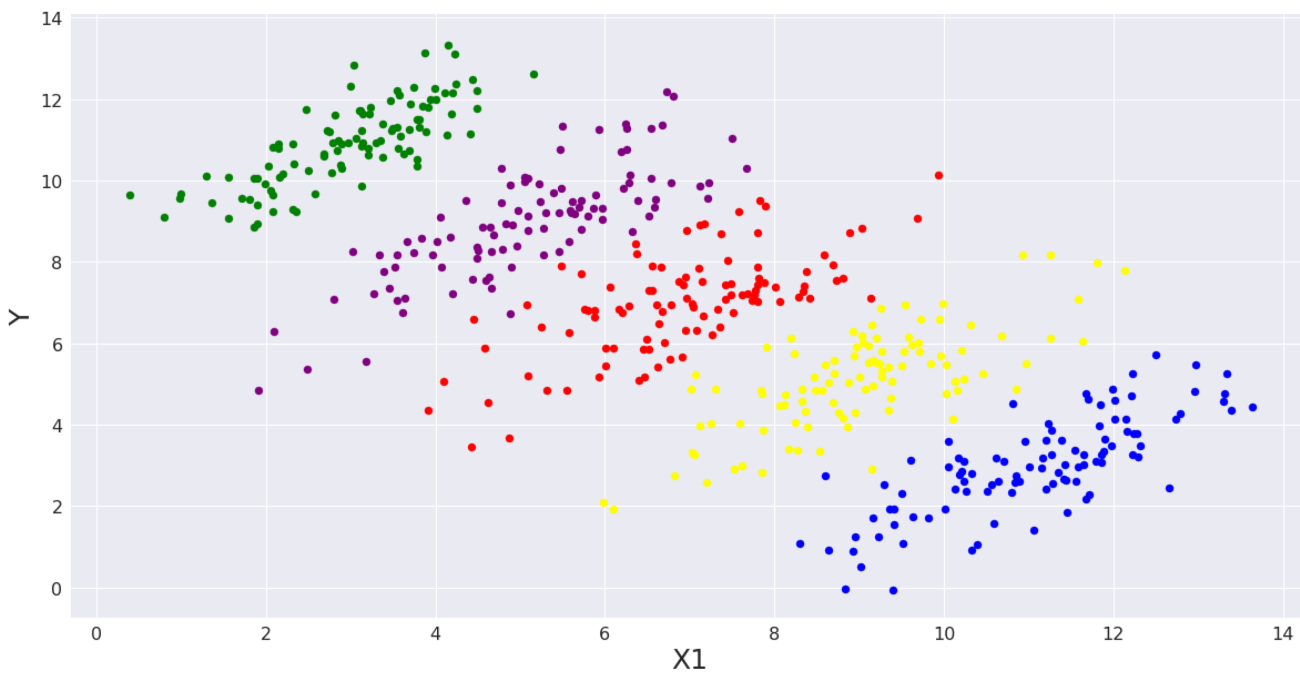
2b)

In [19]:

```
dataSet1x = []
dataSet1y = []
dataSet2x = []
dataSet2y = []
dataSet3x = []
dataSet3y = []
dataSet4x = []
dataSet4y = []
dataSet5x = []
dataSet5y = []
for i in range(500):
    if HW2data['X2'][i] == 1:
        dataSet1x.append(HW2data['X1'][i])
for i in range(100):
    dataSet1y.append(HW2data['Y'][i])
for i in range(500):
    if HW2data['X2'][i] == 2:
        dataSet2x.append(HW2data['X1'][i])
for i in range(100,200):
    dataSet2y.append(HW2data['Y'][i])
for i in range(500):
    if HW2data['X2'][i] == 3:
        dataSet3x.append(HW2data['X1'][i])
for i in range(200,300):
    dataSet3y.append(HW2data['Y'][i])
for i in range(500):
    if HW2data['X2'][i] == 4:
        dataSet4x.append(HW2data['X1'][i])
for i in range(300,400):
    dataSet4y.append(HW2data['Y'][i])
for i in range(500):
    if HW2data['X2'][i] == 5:
        dataSet5x.append(HW2data['X1'][i])
for i in range(400,500):
    dataSet5y.append(HW2data['Y'][i])
fig=plt.figure(figsize=(20,10))
plt.scatter(dataSet1x,dataSet1y, s=50, color='blue')
plt.scatter(dataSet2x,dataSet2y, s=50, color='yellow')
plt.scatter(dataSet3x,dataSet3y, s=50, color='red')
plt.scatter(dataSet4x,dataSet4y, s=50, color='purple')
plt.scatter(dataSet5x,dataSet5y, s=50, color='green')
plt.ylabel('Y', fontsize = 25)
plt.yticks(fontsize=16)
plt.xlabel('X1', fontsize=25)
plt.xticks(fontsize=16)
```

Out[19]:

```
(array([ -2.,   0.,   2.,   4.,   6.,   8.,  10.,  12.,  14.,  16.])
,
 <a list of 10 Text xticklabel objects>)
```



2c)

In [20]:

```
dataSet_2c = np.zeros((500,2))
```

In [21]:

```
for i in range(500):
    dataSet_2c[i][0] = HW2data['X1'][i]
```

In [22]:

```
for i in range(500):
    dataSet_2c[i][1] = HW2data['X2'][i]
```

In [25]:

```
LinearRegression_2c = LinearRegression().fit(dataSet_2c, HW2data['Y'].values)
```

In [26]:

```
Intercept_2c=LinearRegression_2c.intercept_
Slope_2c=LinearRegression_2c.coef_
```

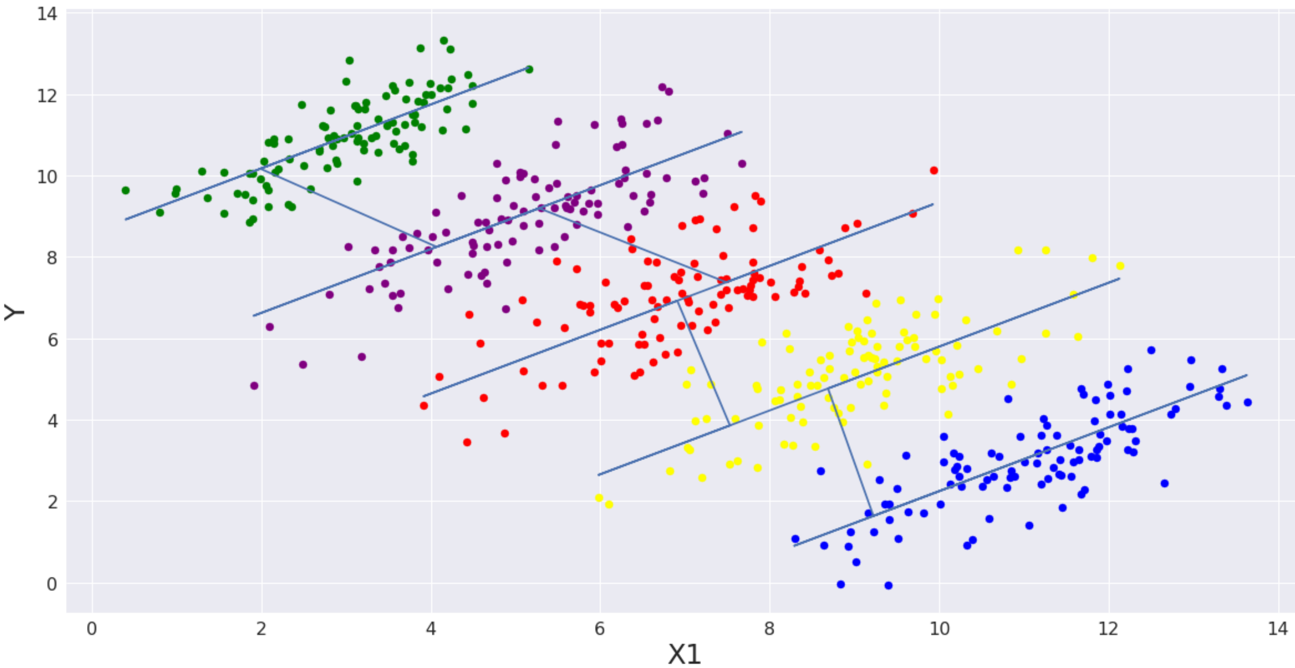
In [117]:

```
dataSet1x = []
```

```
dataSet1y = []

dataSet2x = []
dataSet2y = []
dataSet3x = []
dataSet3y = []
dataSet4x = []
dataSet4y = []
dataSet5x = []
dataSet5y = []
for i in range(500):
    if HW2data['X2'][i] == 1:
        dataSet1x.append(HW2data['X1'][i])
for i in range(100):
    dataSet1y.append(HW2data['Y'][i])
for i in range(500):
    if HW2data['X2'][i] == 2:
        dataSet2x.append(HW2data['X1'][i])
for i in range(100,200):
    dataSet2y.append(HW2data['Y'][i])
for i in range(500):
    if HW2data['X2'][i] == 3:
        dataSet3x.append(HW2data['X1'][i])
for i in range(200,300):
    dataSet3y.append(HW2data['Y'][i])
for i in range(500):
    if HW2data['X2'][i] == 4:
        dataSet4x.append(HW2data['X1'][i])
for i in range(300,400):
    dataSet4y.append(HW2data['Y'][i])
for i in range(500):
    if HW2data['X2'][i] == 5:
        dataSet5x.append(HW2data['X1'][i])
for i in range(400,500):
    dataSet5y.append(HW2data['Y'][i])
fig=plt.figure(figsize=(20,10))
plt.ylabel('Y', fontsize = 25)
plt.yticks(fontsize=16)
plt.xlabel('X1', fontsize=25)
plt.xticks(fontsize=16)
Intercept_2c=LinearRegression_2c.intercept_
plt.ylabel('Y', fontsize = 25)
plt.yticks(fontsize=16)
plt.xlabel('X1', fontsize=25)
plt.xticks(fontsize=16)
Line_2c = np.dot(LinearRegression_2c.coef_,dataSet_2c.T)+Intercept_2c
plt.scatter(dataSet1x,dataSet1y, s=50, color='blue')
plt.scatter(dataSet2x,dataSet2y, s=50, color='yellow')
plt.scatter(dataSet3x,dataSet3y, s=50, color='red')
plt.scatter(dataSet4x,dataSet4y, s=50, color='purple')
plt.scatter(dataSet5x,dataSet5y, s=50, color='green')
plt.plot(HW2data['X1'],Line_2c)
```

Out[117]:

[<matplotlib.lines.Line2D at 0x1a29f3a470>]



In [118]:

```
y_2c = []
y_2c.append(HW2data['Y'])
y_2c_pred = LinearRegression_2c.predict(dataSet_2c)
print(Slope_2c)
print(Intercept_2c)
print(np.sum(np.square(np.subtract(y_2c,np.mean(HW2data['Y'])))))
print(np.sum(np.square(np.subtract(y_2c,y_2c_pred))))
print(np.sum(np.square(np.subtract(y_2c_pred,np.mean(HW2data['Y'])))))
print(np.sum(np.square(np.subtract(y_2c_pred,np.mean(HW2data['Y']))))/np.sum(np.
square(np.subtract(y_2c,np.mean(HW2data['Y'])))))
```

```
[ 0.78441638  3.55191332]
-9.15645044033
4667.38580111
331.396183473
4335.98961764
0.928997473619
```

The linear model: multivariate linear regression. Slope = [0.78441638 3.55191332], intercept = -9.1565, SSTO = 4667.4, SSR = 331.4, SSE = 4335.99, coefficient of multiple determination = 0.928997. The model DOES fit well.

2d) The Simpson's paradox: The nature of association between two variables changes when conditioned on another variable. This is true because when we introduced a new predictor X2, we can see that the model fitted five lines instead of one.

Question 3

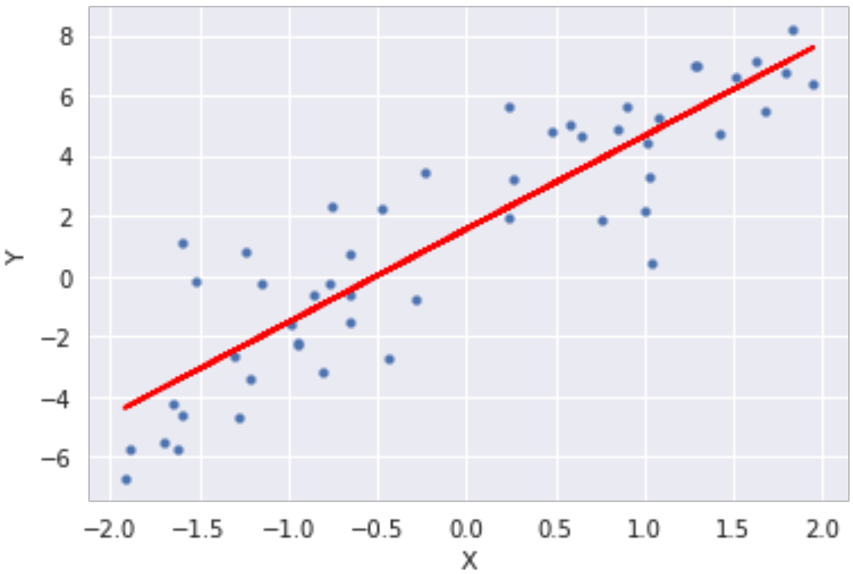
In [11]:

```
import operator
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import PolynomialFeatures
```

In [59]:

```
x=np.random.uniform(low=-2, high=2, size=(50,))
y=2+3*x+np.random.normal(0,2,50)

x = x[:, np.newaxis]
y = y[:, np.newaxis]

reg_1 = LinearRegression().fit(x,y)

y_pred = reg_1.predict(x)

coef_1 = reg_1.score(x,y)
print(coef_1)

plt.ylabel('Y')
plt.xlabel('X')
plt.scatter(x, y, s=15)
plt.plot(x, y_pred, color='r')
plt.show()
print()
```

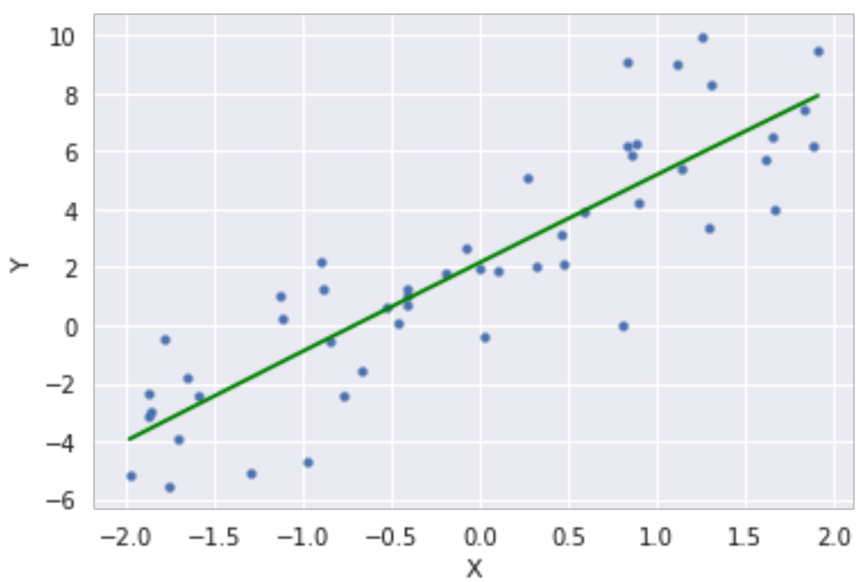
```
0.796351224925
```



In [60]:

```
x=np.random.uniform(low=-2, high=2, size=(50,))
y=2+3*x+np.random.normal(0,2,50)

x = x[:, np.newaxis]
y = y[:, np.newaxis]

x_degree_2 = PolynomialFeatures(degree=2).fit_transform(x)

reg_2 = LinearRegression().fit(x_degree_2,y)
y_pred_degree_2 = reg_2.predict(x_degree_2)

coef_2 = reg_2.score(x_degree_2,y)
print(coef_2)

plt.scatter(x, y, s=15)
plt.ylabel('Y')
plt.xlabel('X')
sort_axis = operator.itemgetter(0)
sorted_zip = sorted(zip(x,y_pred_degree_2), key=sort_axis)
x, y_pred_degree_2 = zip(*sorted_zip)
plt.plot(x, y_pred_degree_2, color='green')
plt.show()
```

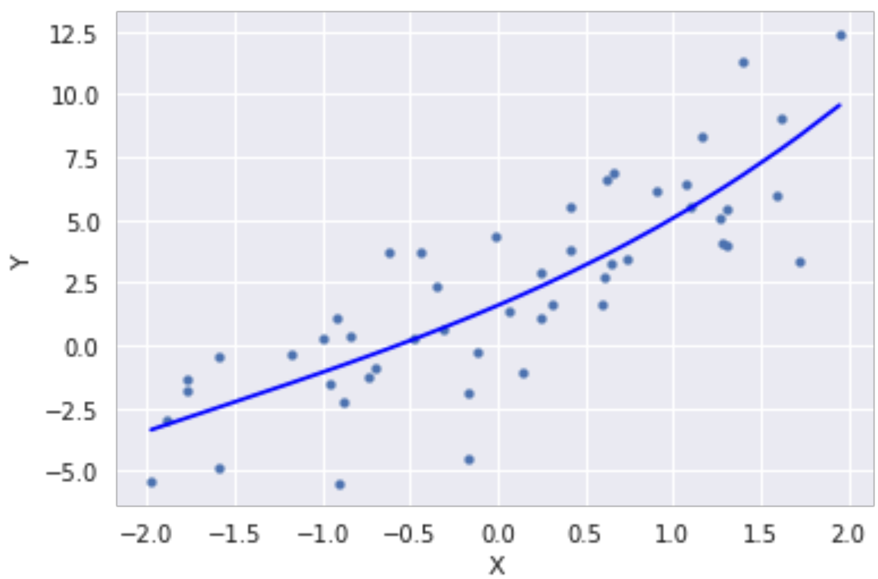
```
0.771719275592
```



In [61]:

```
x=np.random.uniform(low=-2, high=2, size=(50,))
y=2+3*x+np.random.normal(0,2,50)

x = x[:, np.newaxis]
y = y[:, np.newaxis]

x_degree_3 = PolynomialFeatures(degree=3).fit_transform(x)

reg_3 = LinearRegression().fit(x_degree_3,y)

y_pred_degree_3 = reg_3.predict(x_degree_3)

coef_3 = reg_3.score(x_degree_3,y)
print(coef_3)

plt.scatter(x, y, s=15)
plt.ylabel('Y')
plt.xlabel('X')
sort_axis = operator.itemgetter(0)
sorted_zip = sorted(zip(x,y_pred_degree_3), key=sort_axis)
x, y_pred_degree_3 = zip(*sorted_zip)
plt.plot(x, y_pred_degree_3, color='blue')
plt.show()
```

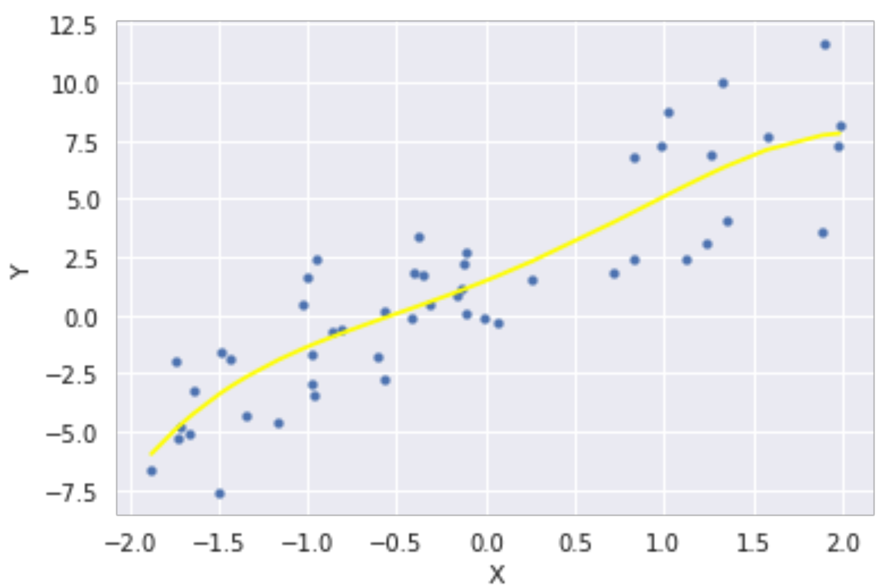
```
0.686463602766
```



In [62]:

```
x=np.random.uniform(low=-2, high=2, size=(50,))
y=2+3*x+np.random.normal(0,2,50)

x = x[:, np.newaxis]
y = y[:, np.newaxis]

x_degree_4 = PolynomialFeatures(degree=4).fit_transform(x)

reg_4 = LinearRegression().fit(x_degree_4,y)

y_pred_degree_4 = reg_4.predict(x_degree_4)

coef_4 = reg_4.score(x_degree_4,y)
print(coef_4)

plt.scatter(x, y, s=15)
plt.ylabel('Y')
plt.xlabel('X')
sort_axis = operator.itemgetter(0)
sorted_zip = sorted(zip(x,y_pred_degree_4), key=sort_axis)
x, y_pred_degree_4 = zip(*sorted_zip)
plt.plot(x, y_pred_degree_4, color='yellow')
plt.show()
```

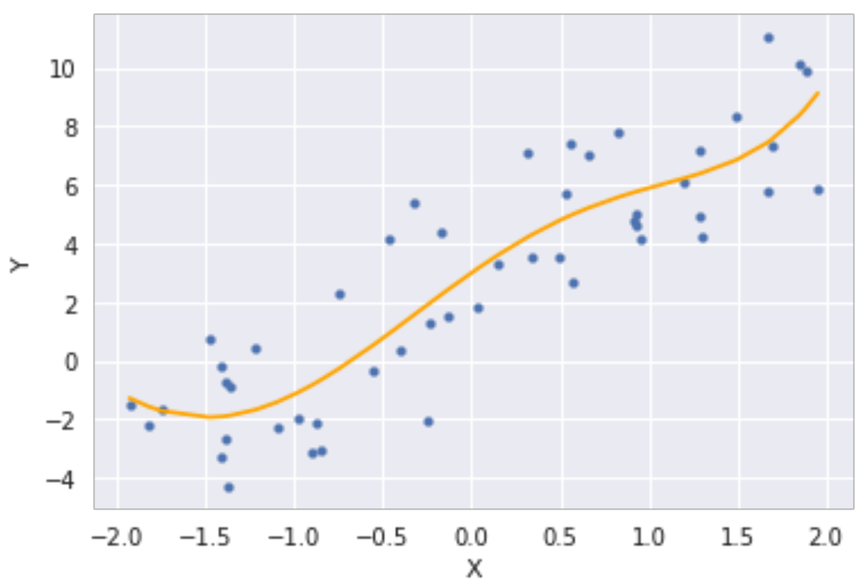
```
0.775644277324
```



In [63]:

```
x=np.random.uniform(low=-2, high=2, size=(50,))
y=2+3*x+np.random.normal(0,2,50)

x = x[:, np.newaxis]
y = y[:, np.newaxis]

x_degree_5 = PolynomialFeatures(degree=5).fit_transform(x)

reg_5 = LinearRegression().fit(x_degree_5,y)

y_pred_degree_5 = reg_5.predict(x_degree_5)

coef_5=reg_5.score(x_degree_5,y)
print(coef_5)

plt.scatter(x, y, s=15)
plt.ylabel('Y')
plt.xlabel('X')
sort_axis = operator.itemgetter(0)
sorted_zip = sorted(zip(x,y_pred_degree_5), key=sort_axis)
x, y_pred_degree_5 = zip(*sorted_zip)
plt.plot(x, y_pred_degree_5, color='orange')
plt.show()
Slope=LinearRegression().fit(x_degree_5,y).coef_
print(Slope)
```

```
0.785880116742
```



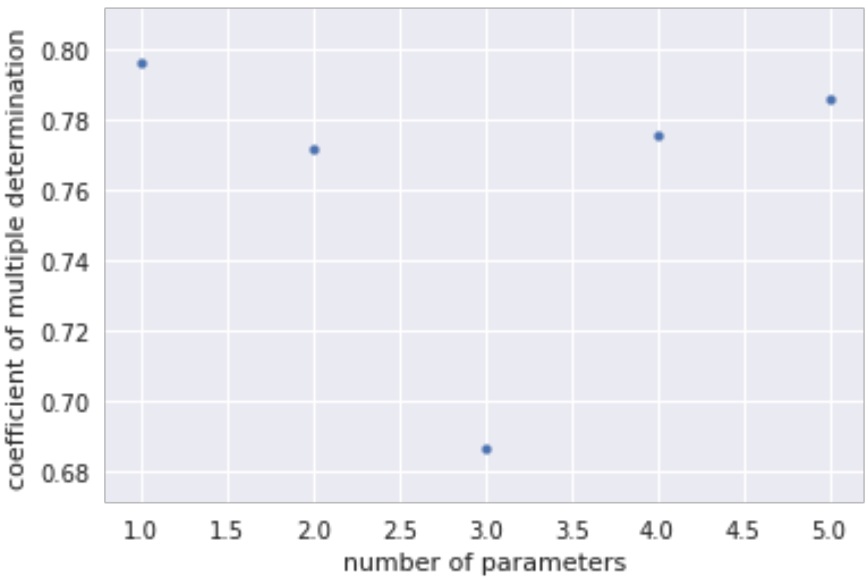

```
[[ 0.          4.21250153 -0.94042119 -0.79545496  0.3117489   0.100
75729]]
```

In [64]:

```
plt.scatter(x=[1,2,3,4,5], y=[coef_1,coef_2,coef_3,coef_4,coef_5], s=15)
plt.ylabel('coefficient of multiple determination')
plt.xlabel('number of parameters')
```

Out[64]:

```
Text(0.5, 0, 'number of parameters')
```

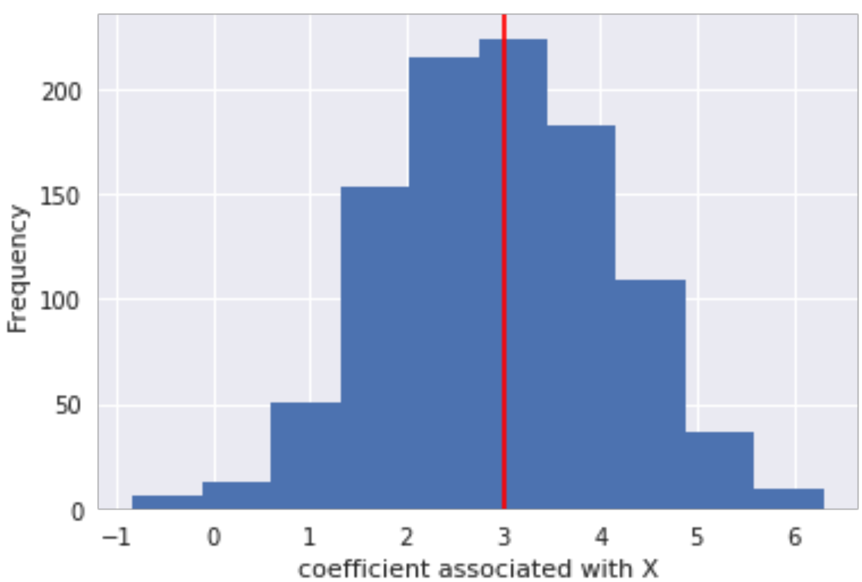


There is no clear correlation between the two.

question 3b)

In [ ]:

```
coef_x = []
slopes=[]
predictions_y=[]
x_vals = [1,1.5,2.25,3.375,5.0625,7.59375]
for w in range(1000):
    x=np.random.uniform(low=-2, high=2, size=(50,))
    y=2+3*x+np.random.normal(0,2,50)
    x = x[:, np.newaxis]
    y = y[:, np.newaxis]
    x_degree_5 = PolynomialFeatures(degree=5).fit_transform(x)
    y_pred_degree_5 = LinearRegression().fit(x_degree_5,y).predict(x_degree_5)
    Slope=LinearRegression().fit(x_degree_5,y).coef_
    slopes.append(Slope)

    coef_x.append(slopes[w][0][1])

    predictions_y.append(np.dot(slopes[-1],x_vals))

plt.hist(coef_x)
plt.xlabel('coefficient associated with X')
plt.ylabel('Frequency')
plt.axvline(x=3.0, color='red')
plt.show()

plt.hist(predictions_y)
plt.xlabel('predictions of Y with X=1.5')
plt.ylabel('Frequency')
plt.axvline(x=6.5, color='red')
plt.show()
```




```
---------------------------------------------------------------------
-------
KeyboardInterrupt                         Traceback (most recent cal
l last)
<ipython-input-12-8ee5e963a282> in <module>()
```

```
     23 plt.show()
     24 
---> 25 plt.hist(predictions_y)
     26 plt.xlabel('predictions of Y with X=1.5')
     27 plt.ylabel('Frequency')

/anaconda3/lib/python3.6/site-packages/matplotlib/pyplot.py in hist(
x, bins, range, density, weights, cumulative, bottom, histtype, alig
n, orientation, rwidth, log, color, label, stacked, normed, data, **
kwargs)
   2617         align=align, orientation=orientation, rwidth=rwidth,
log=log,
   2618         color=color, label=label, stacked=stacked, normed=no
rmed,
-> 2619         data=data, **kwargs)
   2620 
   2621 # Autogenerated by boilerplate.py.  Do not edit as changes w
ill be lost.

/anaconda3/lib/python3.6/site-packages/matplotlib/__init__.py in inn
er(ax, data, *args, **kwargs)
   1783                         "the Matplotlib list!)" % (label_nam
er, func.__name__),
   1784                         RuntimeWarning, stacklevel=2)
-> 1785             return func(ax, *args, **kwargs)
   1786 
   1787         inner.__doc__ = _add_data_doc(inner.__doc__,

/anaconda3/lib/python3.6/site-packages/matplotlib/axes/_axes.py in h
ist(self, x, bins, range, density, weights, cumulative, bottom, hist
type, align, orientation, rwidth, log, color, label, stacked, normed
, **kwargs)
   6643                 patch = _barfunc(bins[:-1]+boffset, height,
width,
   6644                                  align='center', log=log,
-> 6645                                  color=c, **{bottom_kwarg: b
ottom})
   6646                 patches.append(patch)
   6647                 if stacked:

/anaconda3/lib/python3.6/site-packages/matplotlib/__init__.py in inn
er(ax, data, *args, **kwargs)
   1783                         "the Matplotlib list!)" % (label_nam
er, func.__name__),
   1784                         RuntimeWarning, stacklevel=2)
-> 1785             return func(ax, *args, **kwargs)
   1786 
   1787         inner.__doc__ = _add_data_doc(inner.__doc__,

/anaconda3/lib/python3.6/site-packages/matplotlib/axes/_axes.py in b
ar(self, x, height, width, bottom, align, **kwargs)
   2332             ymin = max(ymin * 0.9, 1e-100)
   2333             self.dataLim.intervaly = (ymin, ymax)
```

```
-> 2334         self.autoscale_view()
   2335
   2336         bar_container = BarContainer(patches, errorbar,
label=label)

/anaconda3/lib/python3.6/site-packages/matplotlib/axes/_base.py in a
utoscale_view(self, tight, scalex, scaley)
   2402             stickies = [artist.sticky_edges for artist in
self.get_children()]
   2403             x_stickies = sum([sticky.x for sticky in
stickies], [])
-> 2404             y_stickies = sum([sticky.y for sticky in
stickies], [])
   2405             if self.get_xscale().lower() == 'log':
   2406                 x_stickies = [xs for xs in x_stickies if xs
> 0]

KeyboardInterrupt:
```

Both of these two graphs above have big variance. They are also slightly biased to the left of the expected values (this means there is an underestimation for both of them). This means that polynomial of degree 5 is not a very good fit. The coefficient of multiple determination, however, told us it is a good fit (0.785880116742). This is why it's misleading.

Si Wu Supervised ML

HOMEWORK 2

4) Ridge regression shrinks the regression coefficients by imposing a penalty on their size. The ridge coefficients minimize a penalized residual sum of squares

$$\hat{\theta}_{Ridge} = \underset{\theta}{\arg\min} \left\{ \sum_{i=1}^{N} \Big(y_i - \theta_0 - \sum_{j=1}^{p} x_{ij}\theta_j\Big)^2 + \lambda \sum_{j=1}^{p} \theta_j^2 \right\}$$

$\lambda \geq 0$ is a complexity parameter that controls the amount of shrinkage :

$$\hat{\theta}_{Ridge} = \underset{\theta}{\arg\min} \left\{ \sum_{i=1}^{N} (y_i - x_i^T\theta)^2 \right\} + \lambda \|\theta_j\|_2^2$$

$$= \|Y - X\theta\|_2^2 + \lambda \|\theta_j\|_2^2$$

So, $RSS(\lambda) = (Y - X\theta)^T(Y - X\theta) + \lambda\theta^T\theta$

$$= (Y^T - \theta^T X^T)(Y - X\theta) + \lambda\theta^T\theta = Y^TY - Y^TX\theta - \theta^TX^TY + \theta^TX^TX\theta + \lambda\theta^T\theta$$

So $\nabla_\theta RSS(\lambda) = -(Y^TX)^T - X^TY + 2X^TX\theta + 2\lambda\theta$

$$= -2X^TY + 2X^TX\theta + 2\lambda\theta \ \underset{\text{set to}}{=} \ 0$$

so, $2X^TX\theta + 2\lambda\theta = 2X^TY$

$$X^TX\theta + \lambda\theta = X^TY$$

$$\theta(X^TX + \lambda I) = X^TY$$

So, $\hat{\theta}_{Ridge} = (X^TX + \lambda I)^{-1}X^TY$

where I is the $p \times p$ identity matrix

5) $\mathrm{Var}(\hat{\theta}_{LS}) = \sigma^2 (X^T X)^{-1}$

$\mathrm{Var}(\hat{\theta}_{Ridge}) = \sigma^2 (X^T X + \lambda I)^{-1} X^T X \, (X^T X + \lambda I)^{-1}$

Let $(X^T X)\, \mathrm{Var}(\hat{\theta}_{LS}) = A$

$(X^T X)\, \mathrm{Var}(\hat{\theta}_{Ridge}) = B$

so $A = \sigma^2 (X^T X)(X^T X)^{-1} = \sigma^2 I$

$$B = \sigma^2 \underbrace{(X^T X)(X^T X + \lambda I)^{-1}}_{C} \; \underbrace{(X^T X)(X^T X + \lambda I)^{-1}}_{D}$$

Let $C = (X^T X)(X^T X + \lambda I)^{-1} = (X^T X + \lambda I - \lambda I)(X^T X + \lambda I)^{-1}$

so $C = (X^T X + \lambda I)(X^T X + \lambda I)^{-1} - \lambda I (X^T X + \lambda I)^{-1}$

$= I - \lambda I (X^T X + \lambda I)^{-1}$

Because $C = D$

So we have the same thing for D.

So, $C \leq I$, and $D \leq I$

So, $\sigma^2 C D \leq \sigma^2 I$

so $B \leq A$

so $\mathrm{Var}(\hat{\theta}_{Ridge}) \leq \mathrm{Var}(\hat{\theta}_{LS})$.

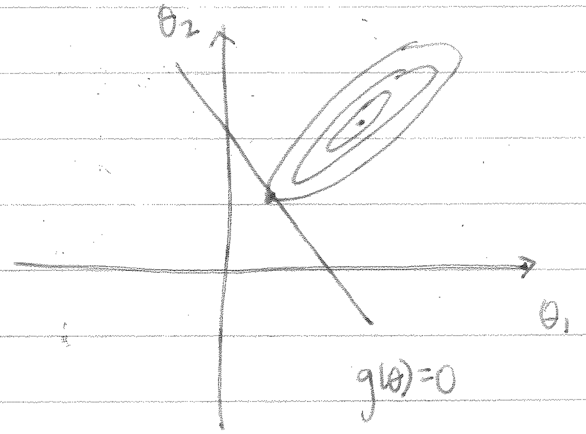
6) $Y = X\theta + \varepsilon$

Constrained optimization

$\hat{\theta} = \text{argmin}_{\theta} \| Y - X\theta \|_2^2$ such that $W^T\theta = b$

equivalently, $W^T\theta - b = 0$

let $g(\theta) = W^T\theta - b$



At any point on the constraint surfa : $g(\theta)$

$\nabla g(\theta) \perp g(\theta)$ (i.e., gradient is $\perp$ to surface)

From Taylor approximation around θ:

$$g(\theta+\varepsilon) \approx g(\theta) + \underbrace{(\theta+\varepsilon-\theta)^T}_{\varepsilon} \nabla g(\theta)$$

$g(\theta+\varepsilon)$ ↓ another point on the surface

$g(\theta)$ ⇓ on the surface $=0$

ε ⇓ approx. on the surface for small ε

$\Rightarrow \nabla g(\theta)$ is orthogonal to the surface

Next, look for point $\hat{\theta}$ on the surface to minimize $J(\theta)$

At $\hat{\theta}$, $\nabla J(\theta)$ is orthogonal to the surface (i.e., cannot decrease $J(\theta)$ by moving along the surface)

So, $\nabla J(\theta)$ and $\nabla g(\theta)$ are (anti) parallel vectors.

So there exists λ such that $\nabla J(\theta) + \lambda \nabla g(\theta) = 0$

Lagrange multiplier, $\lambda \neq 0$ (any sign)

Lagrangian function: $L(\theta, \lambda) = J(\theta) + \lambda g(\theta)$

Stationary point: $\dfrac{dL(\theta,\lambda)}{d\theta} = \nabla J(\theta) + \lambda \nabla g(\theta) \overset{\text{set to}}{=} 0$

and $\dfrac{dL(\theta,\lambda)}{d\lambda} = g(\theta) \overset{\text{set to}}{=} 0$

$$L(\theta,\lambda) = \|Y - X\theta\|_2^2 + \lambda(W^T\theta - b)$$

$$\frac{dL(\theta,\lambda)}{d\theta} = 2X^T(X\theta - Y) + \lambda W \overset{\text{set to}}{=} 0 \quad (1)$$

$$\frac{dL(\theta,\lambda)}{d\lambda} = W^T\theta - b \overset{\text{set to}}{=} 0 \quad (2)$$

from (1), $2X^TX\theta - 2X^TY + \lambda W = 0$

$$2X^TX\theta - 2X^TY = -\lambda W$$

$$2X^TX\theta = 2X^TY - \lambda W$$

$$X^TX\theta = X^TY - \tfrac{1}{2}\lambda W$$

$$\hat{\theta} = (X^TX)^{-1}(X^TY - \tfrac{1}{2}\lambda W) = X^TY - \tfrac{1}{2}\lambda W$$

because $W^T\hat{\theta} = b$

so $W^T(X^TY - \frac{1}{2}\lambda W) = b$

$W^TX^TY - \frac{1}{2}\lambda W^T W = b$

$\frac{1}{2}\lambda W^T W = W^T X^T Y - b$

$\lambda = 2(W^TW)^{-1}(W^TX^TY - b)$

So, $\hat{\theta} = X^TY - \frac{1}{2}2(W^TW)^{-1}(W^TX^TY - b)\,W$

$= X^TY - (W^TW)^{-1}(W^TX^TY - b)W$

In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
sns.set()
from sklearn.linear_model import LinearRegression
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
from sklearn import metrics
from scipy.stats import linregress
from numpy.linalg import inv
import operator
import matplotlib.pyplot as plt
import matplotlib
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import PolynomialFeatures
```

Question 7ai)

In [ ]:

```
theta_7ai= np.dot(inv(np.dot(X_7ai.T,X_7ai)),np.dot(X_7ai.T,Y_7ai))
```

Question 7a)ii)

In [ ]:

```
theta_7aii = np.dot(inv(np.dot(X_7aii.T,X_7aii)+regularization_parameter*np.iden
tity(number_j)),np.dot(X_7aii.T,Y_7aii))
```

Question 7a)iii)

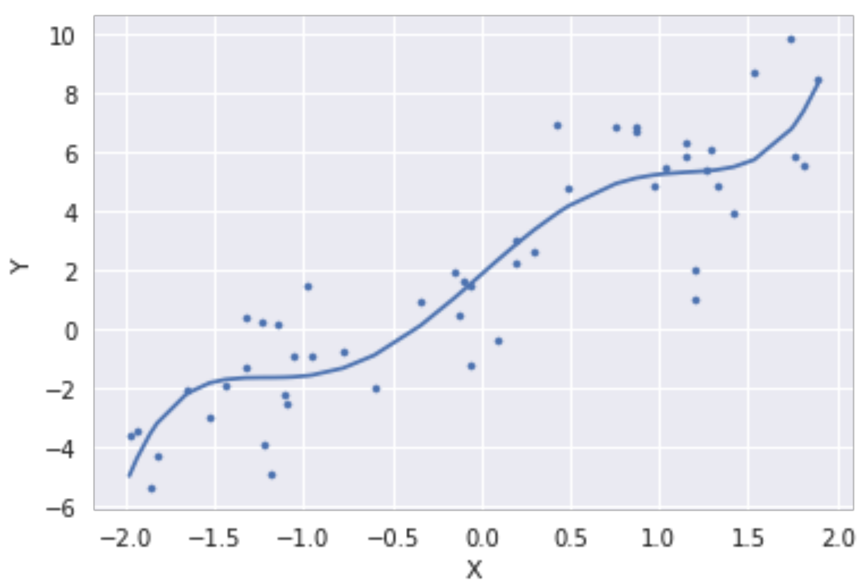
In [2]:

```
def gradient_descent(x, y, initial_theta, regularization_parameter, loss_func, a
lpha=0.01, precision=0.001):
    loss = []
    theta = initial_theta
    all_thetas = [] # to store all thetas
    predictions = [] # to store all predictions
    number_of_steps = 0
    previous_loss = 0
    prediction = np.dot(x,theta) #dot product
    error = prediction - y
    current_loss = loss_func(error,theta,regularization_parameter)
    predictions.append(prediction)
    loss.append(current_loss)
    all_thetas.append(theta)
    number_of_steps+=1
    while abs(current_loss - previous_loss) > precision: #if the difference betw
een current and previous values of loss function is bigger than the precision we
set
        previous_loss = current_loss #we update the value of the loss function t
o be the current value of loss function
        gradient = np.dot(x.T,error/np.size(y)) #new gradient
        theta = theta - alpha * gradient  #update theta
        all_thetas.append(theta)

        prediction = np.dot(x,theta)
        error = prediction - y
        current_loss = loss_func(error,theta,regularization_parameter)
        loss.append(current_loss)

    return all_thetas, loss, predictions

def loss_func(error,theta,regularization_parameter): #lasso loss
    return np.sum(error**2)+regularization_parameter*np.sum(np.abs(theta))
```

Question 7b)i) - analytical solution for least squares regression

In [3]:

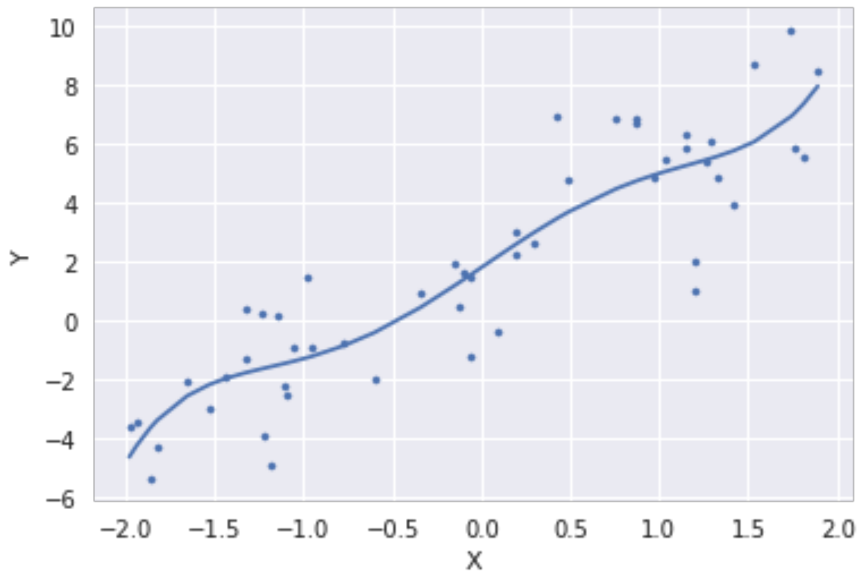
```
x=np.random.uniform(low=-2, high=2, size=(50,))
y=2+3*x+np.random.normal(0,2,50)

A=np.ones(50)
B=np.array((x))
C=np.array((x**2))
D=np.array((x**3))
E=np.array((x**4))
F=np.array((x**5))
matrix_x = np.column_stack((A,B,C,D,E,F))

theta_7bi= np.dot(inv(np.dot(matrix_x.T,matrix_x)),np.dot(matrix_x.T,y))
y_analytical_solution=np.dot(matrix_x,theta_7bi)

plt.scatter(x, y, s=10)
sort_axis = operator.itemgetter(0)
sorted_zip = sorted(zip(x,y_analytical_solution), key=sort_axis)
x_copy, y_analytical_zip = zip(*sorted_zip)
plt.plot(x_copy, y_analytical_zip)
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```



Question 7b)ii) - analytical solution for ridge regression as function of the regularization parameter

In [4]:

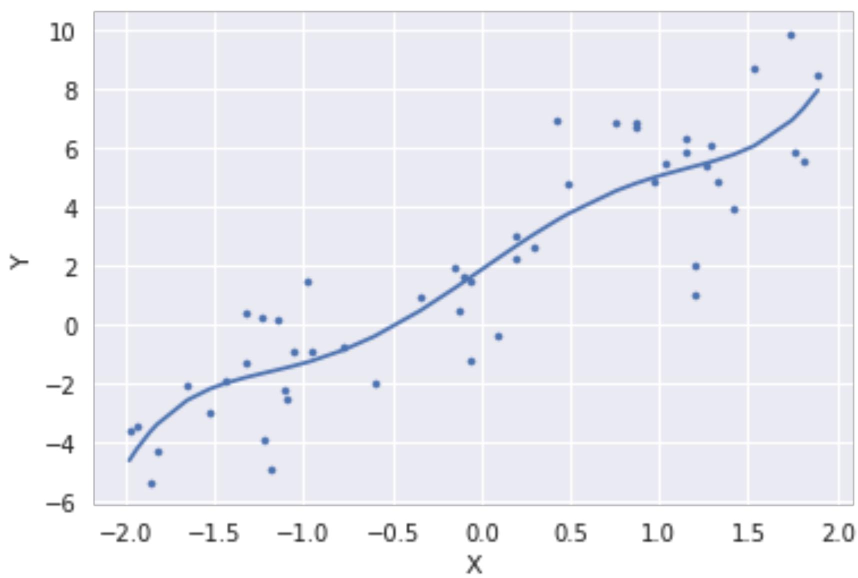
```
regularization_parameter=0.5

theta_7bii = np.dot(inv(np.dot(matrix_x.T,matrix_x)+regularization_parameter*np.
identity(6)),np.dot(matrix_x.T,y))
y_ridge_regression=np.dot(matrix_x,theta_7bii)

plt.scatter(x, y, s=10)

sort_axis = operator.itemgetter(0)
sorted_zip = sorted(zip(x,y_ridge_regression), key=sort_axis)
x_copy, y_ridge_zip = zip(*sorted_zip)
plt.plot(x_copy, y_ridge_zip)
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```



Question 7b)iii) - batch gradient descent for lasso regression

In [5]:

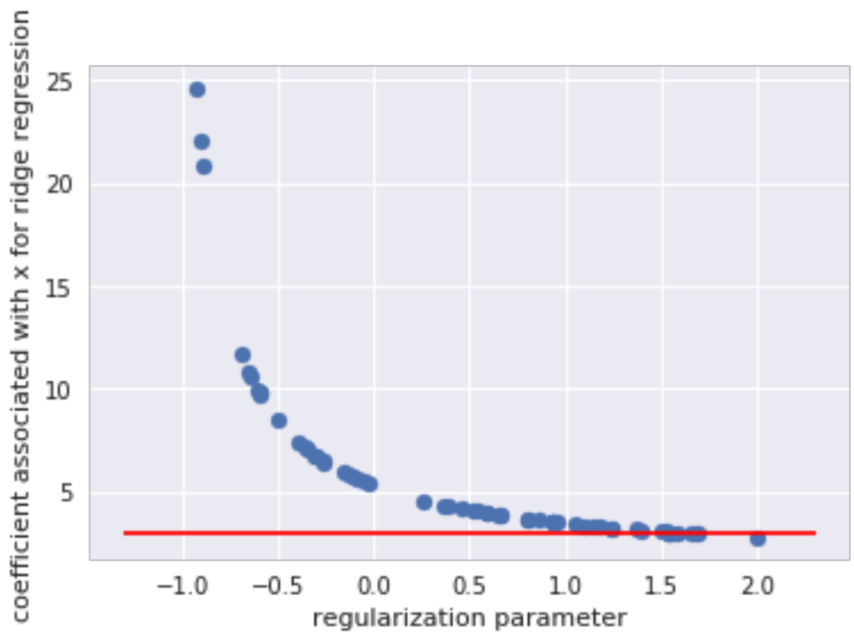
```
regularization_parameter=0.5

theta_i = np.random.rand(6)
all_thetas, loss, predictions = gradient_descent(matrix_x, y, theta_i, regulariz
ation_parameter, loss_func)

y_lasso_regression=np.dot(matrix_x,all_thetas[-1])
plt.scatter(x, y, s=10)
sort_axis = operator.itemgetter(0)
sorted_zip = sorted(zip(x,y_lasso_regression), key=sort_axis)
x_copy, y_lasso_zip = zip(*sorted_zip)
plt.plot(x_copy, y_lasso_zip)
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```

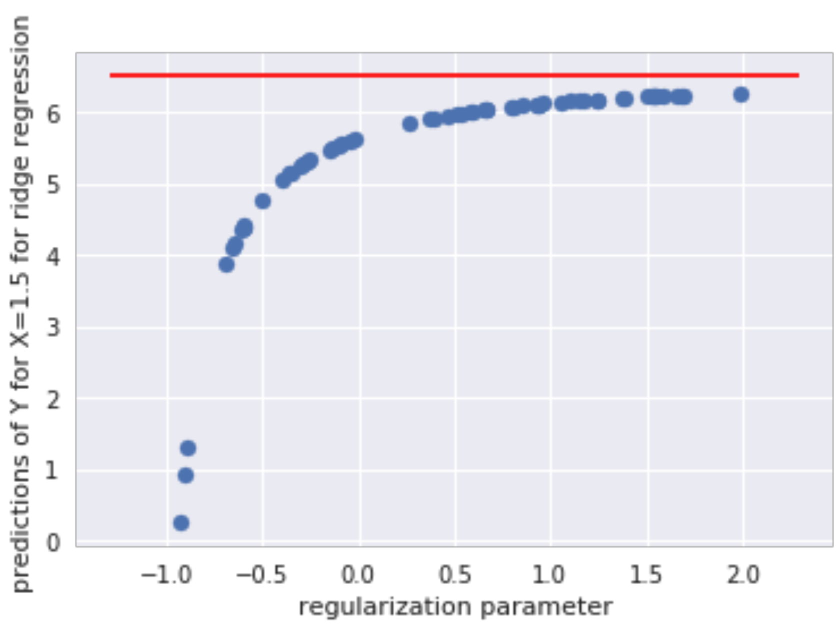


Question 7c) – analytical solution for ridge regression

In [12]:

```
regularization_parameter=np.random.uniform(low=-1, high=2, size=(60,))
all_theta_7c_ridge=[]
coef_associated_with_x_ridge=[]
predictions_y_ridge=[]
x_vals = [1,1.5,2.25,3.375,5.0625,7.59375]
for q in range(60):
    theta_7c_ridge = np.dot(inv(np.dot(matrix_x.T,matrix_x)+regularization_param
eter[q]*np.identity(6)),np.dot(matrix_x.T,y))
    all_theta_7c_ridge.append(theta_7c_ridge)

    coef_associated_with_x_ridge.append(all_theta_7c_ridge[q][1])

    predictions_y_ridge.append(np.dot(all_theta_7c_ridge[-1],x_vals))

plt.scatter(regularization_parameter,coef_associated_with_x_ridge)
plt.xlabel('regularization parameter')
plt.ylabel('coefficient associated with x for ridge regression')
plt.hlines(y=3,xmin=-1.3,xmax=2.3,color='red')
plt.show()

plt.scatter(regularization_parameter,predictions_y_ridge)
plt.xlabel('regularization parameter')
plt.ylabel('predictions of Y for X=1.5 for ridge regression')
plt.hlines(y=6.5,xmin=-1.3,xmax=2.3,color='red')
plt.show()
```

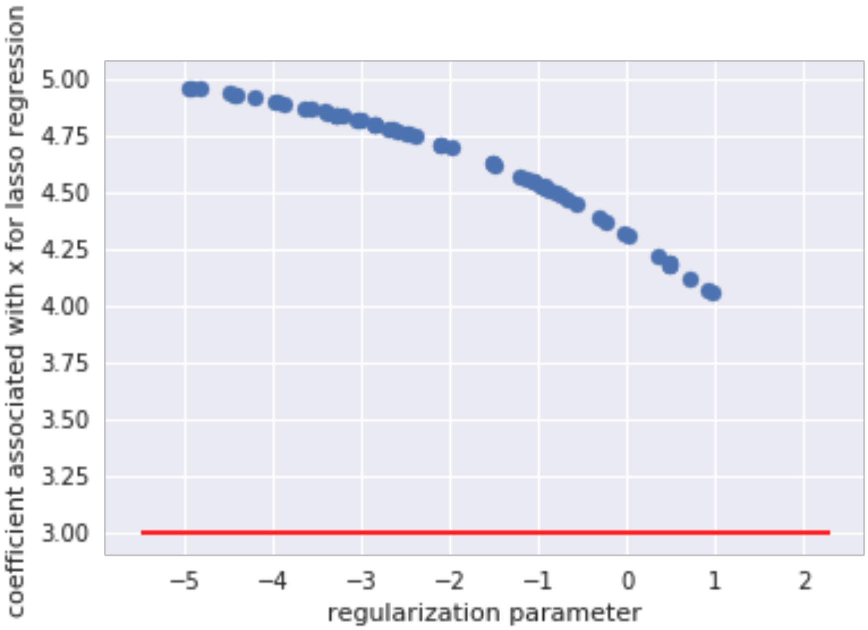




Question 7c) batch gradient descent optimization for lasso regression

In [13]:

```
regularization_parameter=np.random.uniform(low=-5, high=1, size=(60,))
all_theta_7c_lasso=[]
coef_associated_with_x_lasso=[]
predictions_y_lasso=[]
x_vals = [1,1.5,2.25,3.375,5.0625,7.59375]
for q in range(60):
    theta_i = np.random.rand(6)
    all_thetas, loss, predictions = gradient_descent(matrix_x, y, theta_i, regul
arization_parameter[q], loss_func)
    coef_associated_with_x_lasso.append(all_thetas[-1][1])

    predictions_y_lasso.append(np.dot(all_thetas[-1],x_vals))

plt.scatter(regularization_parameter,coef_associated_with_x_lasso)
plt.xlabel('regularization parameter')
plt.ylabel('coefficient associated with x for lasso regression')
plt.hlines(y=3,xmin=-5.5,xmax=2.3,color='red')
plt.show()

plt.scatter(regularization_parameter,predictions_y_lasso)
plt.xlabel('regularization parameter')
plt.ylabel('predictions of Y for X=1.5 for lasso regression')
plt.hlines(y=6.5,xmin=-5.5,xmax=2.3,color='red')
plt.show()
```



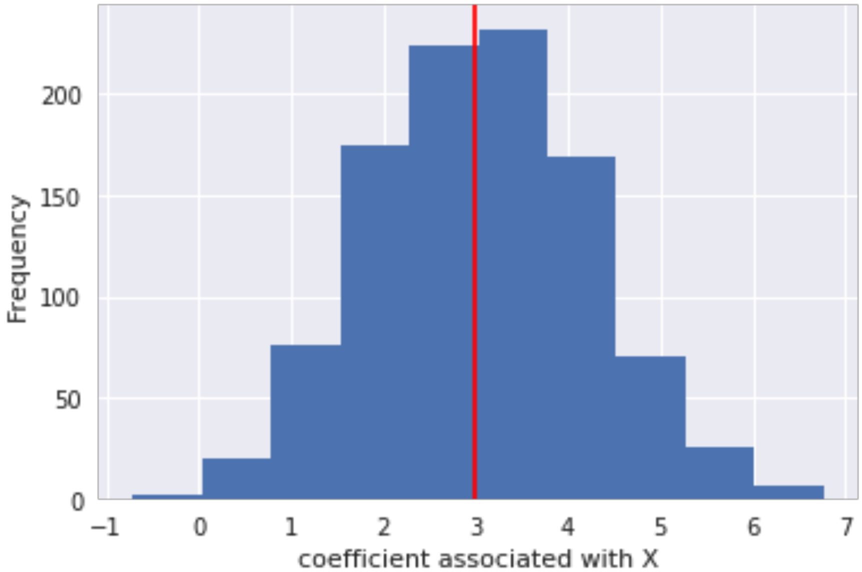
For analytical solution for ridge regression, parameters are bigger when regularization parameter is negative (overestimation). When regularization parameter gets bigger or positive, parameters get exponentially smaller and asymptotically approach the true value.

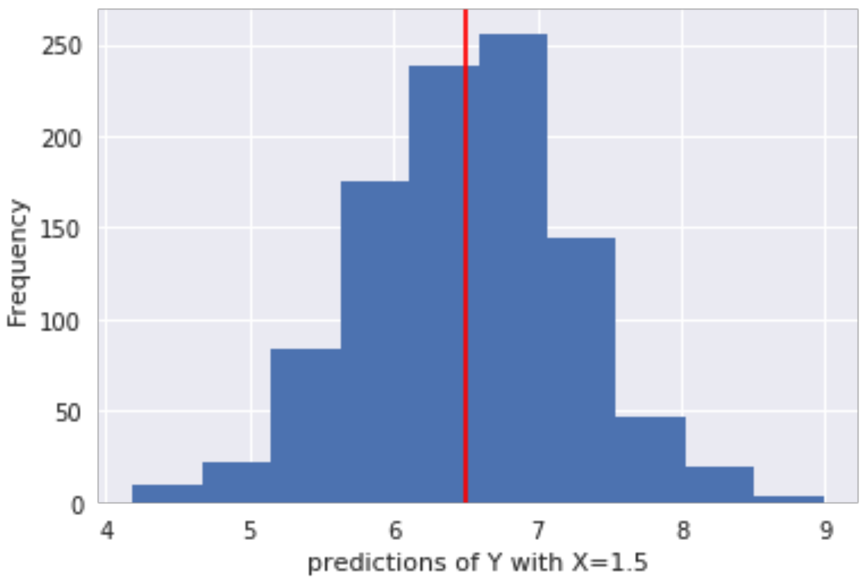
For batch gradient descent using Lasso regression, the parameters are overestimated for negative/small positive regularization parameter. As the regularization parameter gets bigger, the parameter and slope approach the true value.

Question 7d)i – analytical solution for least squares

In [15]:

```
coef_with_x_7di = []
all_theta_7d_least_sq=[]
predictions_y_7di=[]
x_vals = [1,1.5,2.25,3.375,5.0625,7.59375]
for w in range(1000):
    x=np.random.uniform(low=-2, high=2, size=(50,))
    y=2+3*x+np.random.normal(0,2,50)
    A=np.ones(50)
    B=np.array((x))
    C=np.array((x**2))
    D=np.array((x**3))
    E=np.array((x**4))
    F=np.array((x**5))
    matrix_x = np.column_stack((A,B,C,D,E,F))
    theta_7di= np.dot(inv(np.dot(matrix_x.T,matrix_x)),np.dot(matrix_x.T,y))
    all_theta_7d_least_sq.append(theta_7di)

    coef_with_x_7di.append(all_theta_7d_least_sq[w][1])

    predictions_y_7di.append(np.dot(all_theta_7d_least_sq[-1],x_vals))

plt.hist(coef_with_x_7di)
plt.xlabel('coefficient associated with X')
plt.ylabel('Frequency')
plt.axvline(x=3.0, color='red')
plt.show()

plt.hist(predictions_y_7di)
plt.xlabel('predictions of Y with X=1.5')
plt.ylabel('Frequency')
plt.axvline(x=6.5, color='red')
plt.show()
```



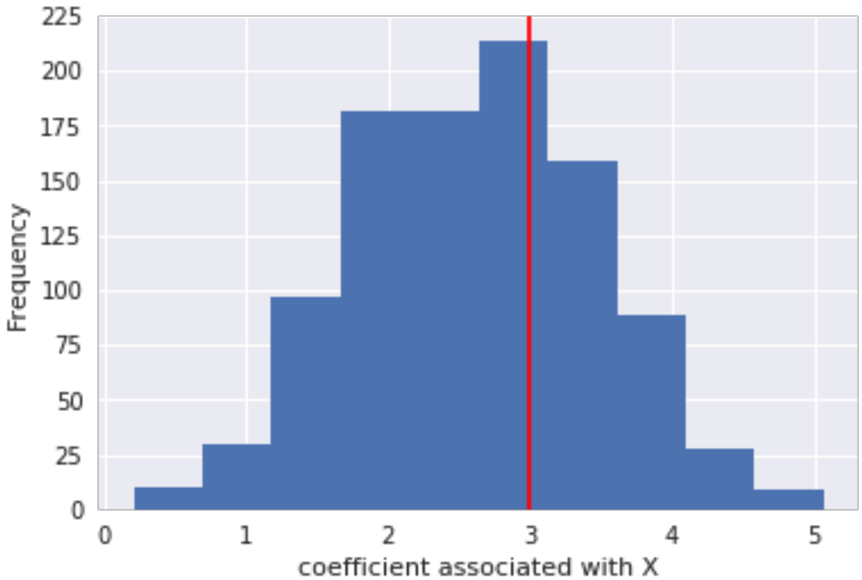


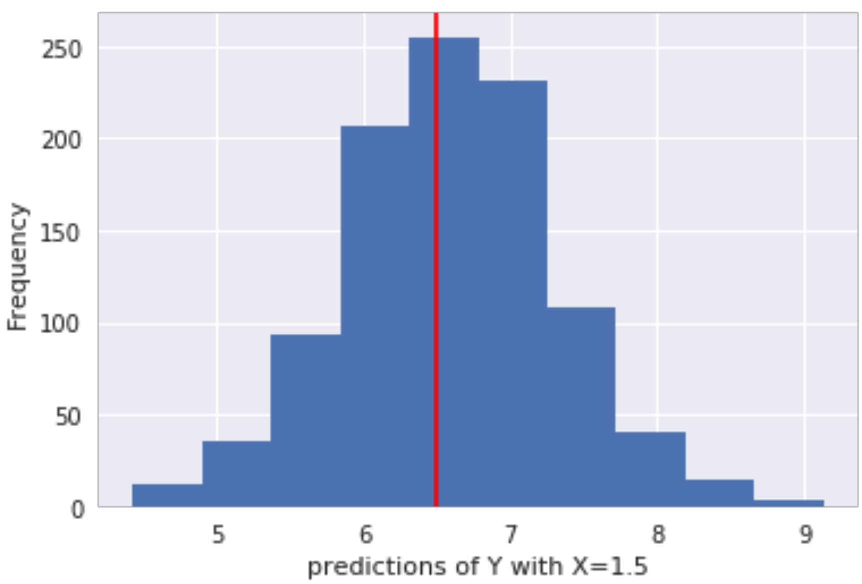
Unbiased but large variance.

Question 7d)ii – analytical solution for ridge regression

In [17]:

```
coef_with_x_7dii = []
all_theta_7d_ridge=[]
predictions_y_7dii=[]
x_vals = [1,1.5,2.25,3.375,5.0625,7.59375]
for w in range(1000):
    x=np.random.uniform(low=-2, high=2, size=(50,))
    y=2+3*x+np.random.normal(0,2,50)
    A=np.ones(50)
    B=np.array((x))
    C=np.array((x**2))
    D=np.array((x**3))
    E=np.array((x**4))
    F=np.array((x**5))
    matrix_x = np.column_stack((A,B,C,D,E,F))
    theta_7d_ridge = np.dot(inv(np.dot(matrix_x.T,matrix_x)+0.5*np.identity(6)),
np.dot(matrix_x.T,y))
    all_theta_7d_ridge.append(theta_7d_ridge)

    coef_with_x_7dii.append(all_theta_7d_ridge[w][1])

    predictions_y_7dii.append(np.dot(all_theta_7d_ridge[-1],x_vals))

plt.hist(coef_with_x_7dii)
plt.xlabel('coefficient associated with X')
plt.ylabel('Frequency')
plt.axvline(x=3.0, color='red')
plt.show()

plt.hist(predictions_y_7dii)
plt.xlabel('predictions of Y with X=1.5')
plt.ylabel('Frequency')
plt.axvline(x=6.5, color='red')
plt.show()
```





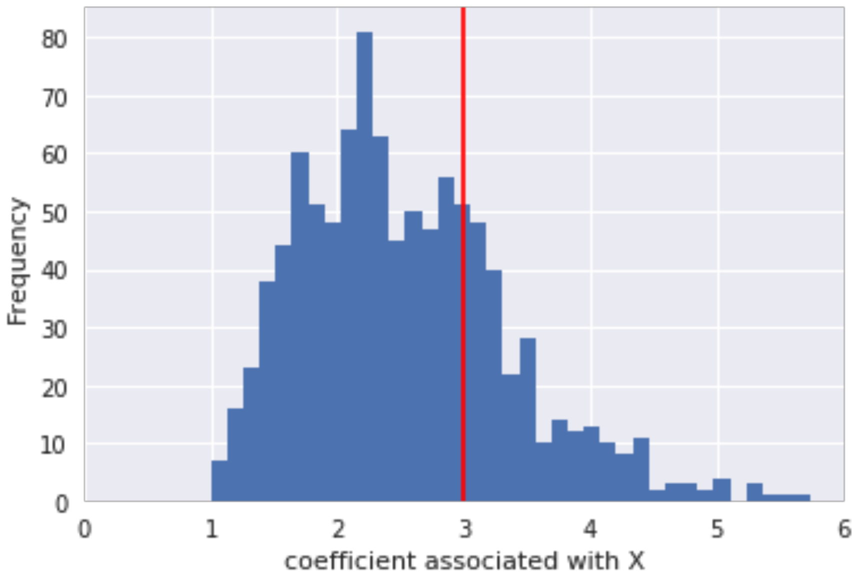
A bit biased to the left (underestimation) and big variance.

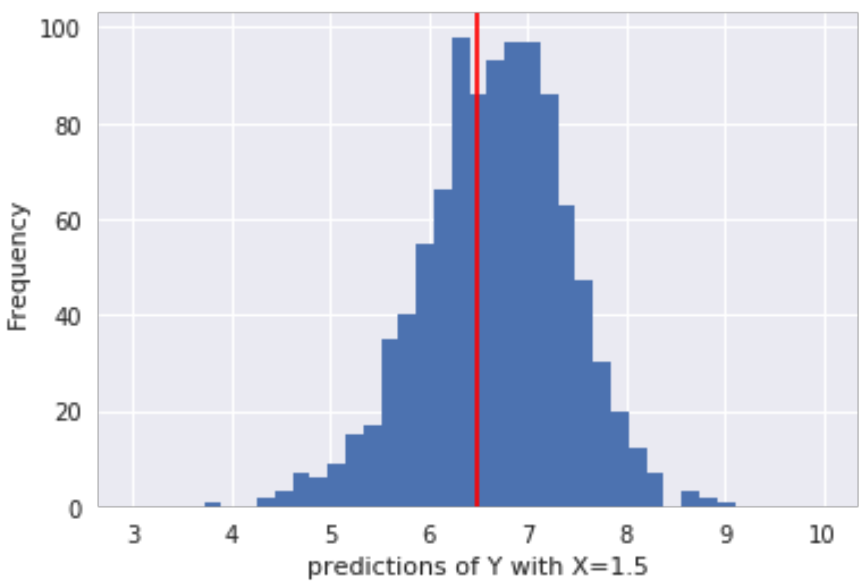
Question 7diii bath gradient descent for Lasso

In [18]:

```
coef_with_x_7diii = []
all_theta_7d_lasso=[]
predictions_y_7diii=[]
x_vals = [1,1.5,2.25,3.375,5.0625,7.59375]
for w in range(1000):
    x=np.random.uniform(low=-2, high=2, size=(50,))
    y=2+3*x+np.random.normal(0,2,50)
    A=np.ones(50)
    B=np.array((x))
    C=np.array((x**2))
    D=np.array((x**3))
    E=np.array((x**4))
    F=np.array((x**5))
    matrix_x = np.column_stack((A,B,C,D,E,F))
    theta_i = np.random.rand(6)
    all_thetas, loss, predictions = gradient_descent(matrix_x, y, theta_i, 0.5,
loss_func)
    coef_with_x_7diii.append(all_thetas[-1][1])

    predictions_y_7diii.append(np.dot(all_thetas[-1],x_vals))

plt.hist(coef_with_x_7diii, bins=np.linspace(1,6,40))
plt.xlabel('coefficient associated with X')
plt.ylabel('Frequency')
plt.xlim([0,6])
plt.axvline(x=3.0, color='red')
plt.show()

plt.hist(predictions_y_7diii, bins=np.linspace(3,10,40))
plt.xlabel('predictions of Y with X=1.5')
plt.ylabel('Frequency')
plt.axvline(x=6.5, color='red')
plt.show()
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:15: Run
timeWarning: invalid value encountered in double_scalars
  from ipykernel import kernelapp as app
```





The graphs show a large variance. It is also biased towards underestimating both the slope of the X term.