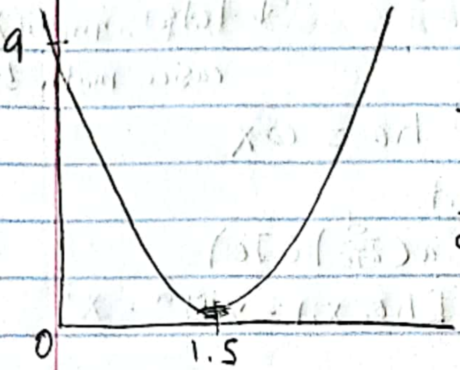
3

Find the minimum of $J(x) = (2x-3)^2$



When slope of $J(x) = (2x-3)^2 = 0$
is when the function is minimized

$\frac{d}{dx} = (2x-3)^2 \rightarrow$ Chain rule

$\frac{d}{dx} = 2(2x-3)\cdot 2$

$\frac{d}{dx} = 4(2x-3)$

$$\frac{0 = 4(2x-3)}{4}$$

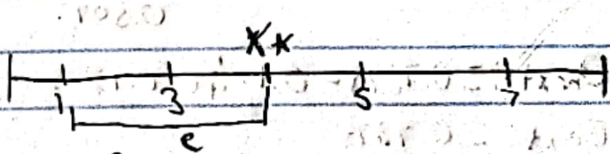
$0 = 2x-3$

$3 = 2x$

$\boxed{1.5 = x}$

4A Find AVG of $[1,3,5,7] = [x_1, x_2, x_3, x_4]$

$e = (x_i - x_*)^2$



$J = (x_1 - x_*)^2 + (x_2 - x_*)^2 + (x_3 - x_*)^2 + (x_4 - x_*)^2$

$\frac{dJ}{dx_*} = 2(x_1 - x_*)\cdot -1 + 2(x_2 - x_*)\cdot -1 + 2(x_3 - x_*)\cdot -1 + 2((x_4 - x_*)\cdot -1$

$$\frac{0 = -2(x_1 - x_*) + (x_2 - x_*) + (x_3 - x_*) + (x_4 - x_*)}{-2}$$

$0 = x_1 + x_2 + x_3 + x_4 - 4x_*$

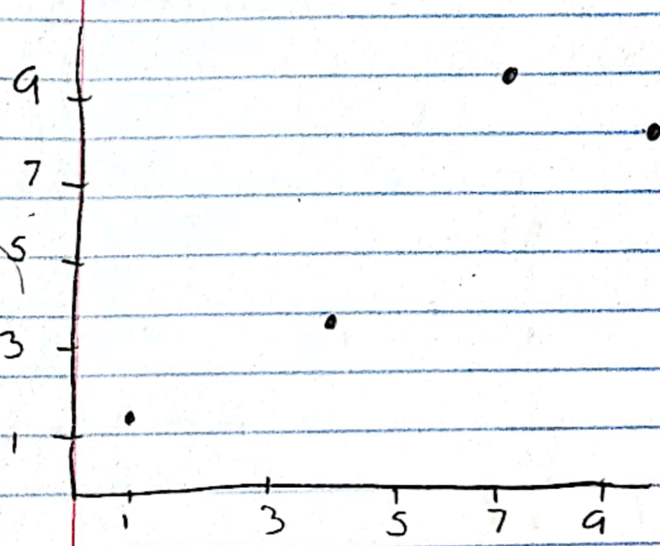
$\frac{4x_* = x_1 + x_2 + x_3 + x_4}{4}$ – $x_* = \frac{x_1 + x_2 + x_3 + x_4}{x4}$ = mean = $\frac{1+3+5+7}{4}$

Mean = 4

4B

$(X, Y) = (1, 1.5), (4, 3.5), (7, 9), (10, 8)$

hypothesis $h_\theta(X) = \theta X$



$\theta_{next} := \theta_{current} - a \cdot \left(\frac{-1}{m} \sum x_i (h_\theta(x_i) - y)\right)$

$\theta = 1 \quad a = 0.01 \qquad -0.5 \qquad -2 \qquad 1.4 \qquad -2.0$

1) $\theta_{next} := \theta_{current} - a \cdot \left(-\frac{1}{4}\left(((1.5-1.5)\cdot 1) + ((3.5-4.5)\cdot 4) + ((9-7)\cdot 7) + ((8-10)\cdot 10)\right)\right)$

$\theta_{next} := 1 - 0.01 \cdot 1.875$

$\theta_{next} = 0.98125$

2) $\theta_{next} := 0.98125 - 0.01 \cdot 1.097$

$\theta_{next} = 0.9703$

3) $\theta_{next} := 0.9703 - 0.01 \cdot 0.6416$

$\theta_{next} = 0.9639$

Repeat until $\frac{d}{dX} \approx 0$

$\downarrow$

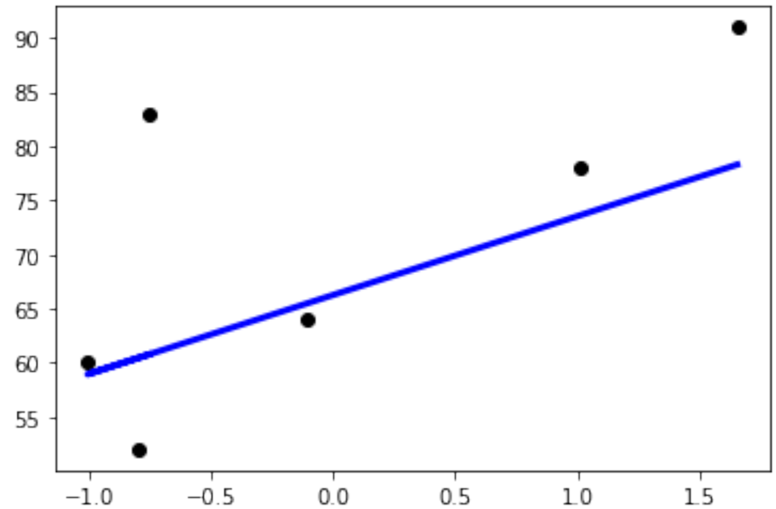
$\boxed{\theta = 0.955}$

Problem 2 Linear regression notebook

In [1]:

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
%matplotlib inline

np.random.seed(123)

df = pd.read_csv("MizzouGameData.csv")

df

X = df[['3FGPCT']]

y = df['PTS'] # target is what we are trying to predict

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20) #

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)

regression = LinearRegression().fit(X_train, y_train) # train model
y_pred = regression.predict(X_test)


# The coefficients
print("Coefficients: \n", regression.coef_)
# The mean squared error
print("Mean squared error: %.2f" % mean_squared_error(y_test, y_pred))
# The coefficient of determination: 1 is perfect prediction
print("Coefficient of determination: %.2f" % r2_score(y_test, y_pred))

plt.scatter(X_test, y_test, color="black")
plt.plot(X_test, y_pred, color="blue", linewidth=3)
plt.show()
```

```
Coefficients:
 [7.28544392]
Mean squared error: 124.95
Coefficient of determination: 0.33
```



Fit of model is not great but we are only using one feature so that is kinda expected

In [ ]:

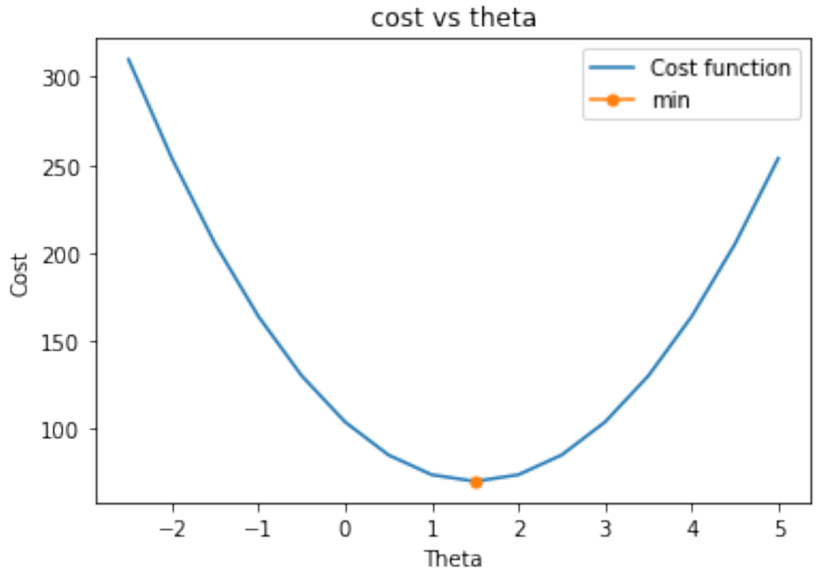
```

```

Problem 3

In [1]:

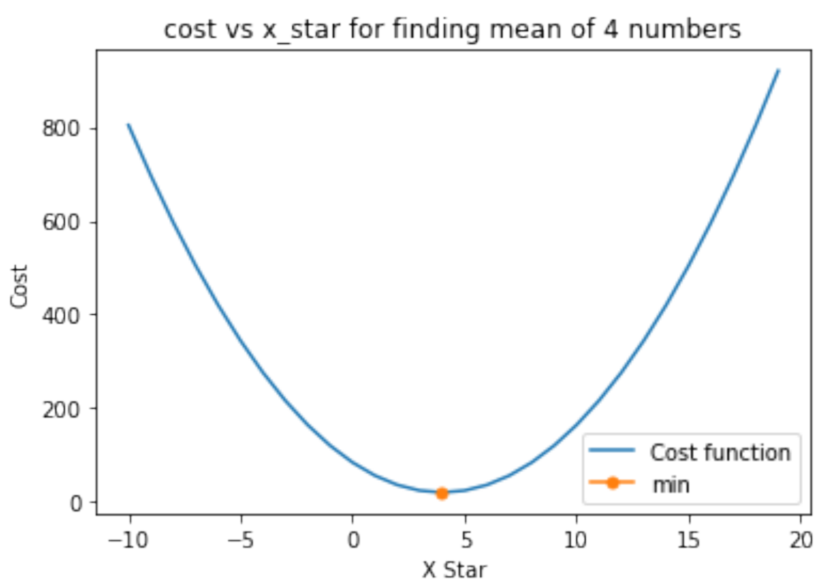
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# Function is y = (2x-3)^2
X = np.arange(-2,5.5,0.5)
x_star_guess = np.arange(-2.5, 5.5, 0.5)
Y = []
e = []
J = []

#generate data for to do gradent desent on
for i in range(len(X)):
    y = (2*X[i]-3)**2
    Y.append(y)

def error_function(x, x_star):
    error = (x-x_star)**2
    return error

for i in range(len(x_star_guess)):
    x_star_error = []
    for j in range(len(X)):
        x_star_error.append(error_function(X[j], x_star_guess[i]))
    cost = sum(x_star_error)
    J.append(cost)

min_index = (np.where(J == min(J)))
min_J = min(J)
min_x_star = x_star_guess[min_index]

plt.xlabel("Theta")
plt.ylabel("Cost")
plt.title("cost vs theta")
plt.plot(x_star_guess,J,label='Cost function')
plt.plot(min_x_star, min_J , marker="o", markersize=5, label='min')
plt.legend()
print("The min cost of the function 2x-3^2 is found at theta", min_x_star)
```

```
The min cost of the function 2x-3^2 is found at theta [1.5]
```



Problem 4a

In [2]:
```
nums = [1, 3, 5, 7]
x_star_guess = np.arange(-10, 20, 1)
e = []
J = []

def error_function(x, x_star):
    error = (x-x_star)**2
    return error

for i in range(len(x_star_guess)):
    x_star_error = []
    for j in range(len(nums)):
        x_star_error.append(error_function(nums[j], x_star_guess[i]))
    cost = sum(x_star_error)
    J.append(cost)

min_index = (np.where(J == min(J)))
min_J = min(J)
min_x_star = x_star_guess[min_index]

plt.xlabel("X Star")
plt.ylabel("Cost")
plt.title("cost vs x_star for finding mean of 4 numbers")
plt.plot(x_star_guess,J,label='Cost function')
plt.plot(min_x_star, min_J , marker="o", markersize=5, label='min')
plt.legend()
plt.show()

print("The min aka the mean is %d" %min_x_star)
```

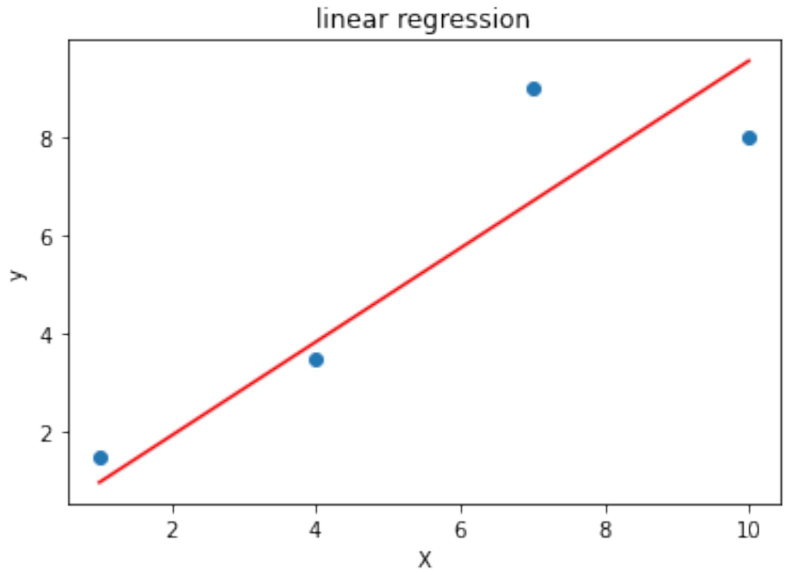



```
The min aka the mean is 4
```

Problem 4b

In [3]:

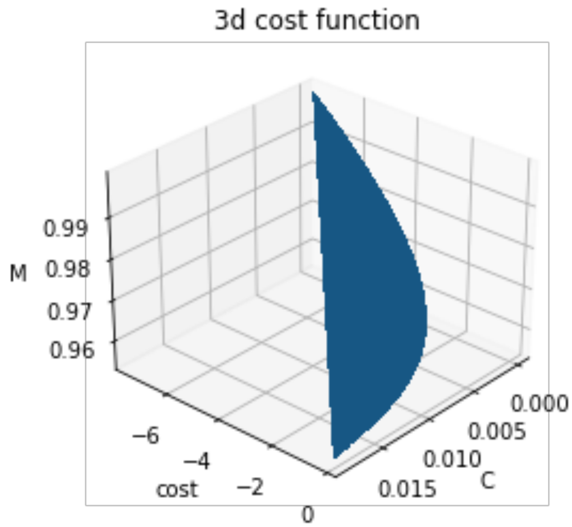
```
from mpl_toolkits import mplot3d


X = np.array([1, 4, 7, 10])

Y = np.array([1.5, 3.5, 9, 8])

m = 1
c = 0

M = []
C = []
J = []

L = 0.0001  # The learning Rate
epochs = 1000  # The number of iterations to perform gradient descent

n = float(len(X)) # Number of elements in X

def cost_fun(y,y_hat,x):
    J = ((y-y_hat)*x)
    return J

# Performing Gradient Descent
for i in range(epochs):
    Y_pred = m*X + c  # The current predicted value of Y
    D_m = (-1/n) * sum(X * (Y - Y_pred))  # Derivative wrt m
    D_c = (-1/n) * sum(Y - Y_pred)  # Derivative wrt c
    m = m - L * D_m  # Update m
    c = c - L * D_c  # Update c
    M.append(m)
    C.append(c)
    total_cost = cost_fun(Y,Y_pred,X)
    total_cost = sum(total_cost)
    J.append(total_cost)


Y_pred = m*X + c
#print (m, c)
plt.scatter(X, Y)
plt.plot([min(X), max(X)], [min(Y_pred), max(Y_pred)], color='red')  # regres
plt.xlabel('X')
plt.ylabel('y')
plt.title('linear regression')
plt.show()
print("slope equals", m)
print("bias equals", c)
```




```
slope equals 0.9537790462397807
bias equals 0.01851484803581841
```

In [4]:

```
%matplotlib notebook

ax = plt.axes(projection='3d')

print("J is minimized when m is {} and when c is {}".format(m,c))

surf = ax.plot_trisurf(C, J, M, linewidth=0, antialiased=False)
ax.set_xlabel('C', fontsize=10)
ax.set_ylabel('cost', fontsize=10)
ax.set_zlabel('M', fontsize=10)
ax.set_title("3d cost function")
```




```
J is minimized when m is 0.9537790462397807 and when c is 0.01851484803581841
```

Out[4]:
```
Text(0.5, 0.92, '3d cost function')
```

In [ ]:

In [ ]:

In [ ]: