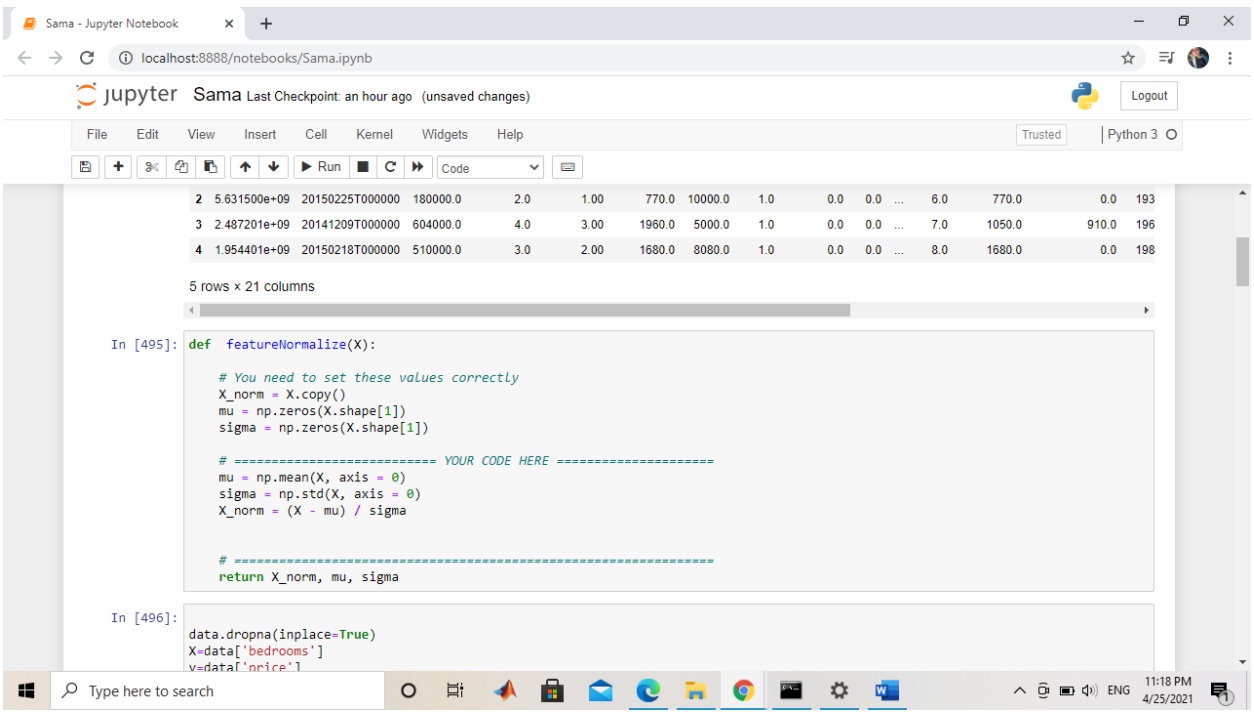
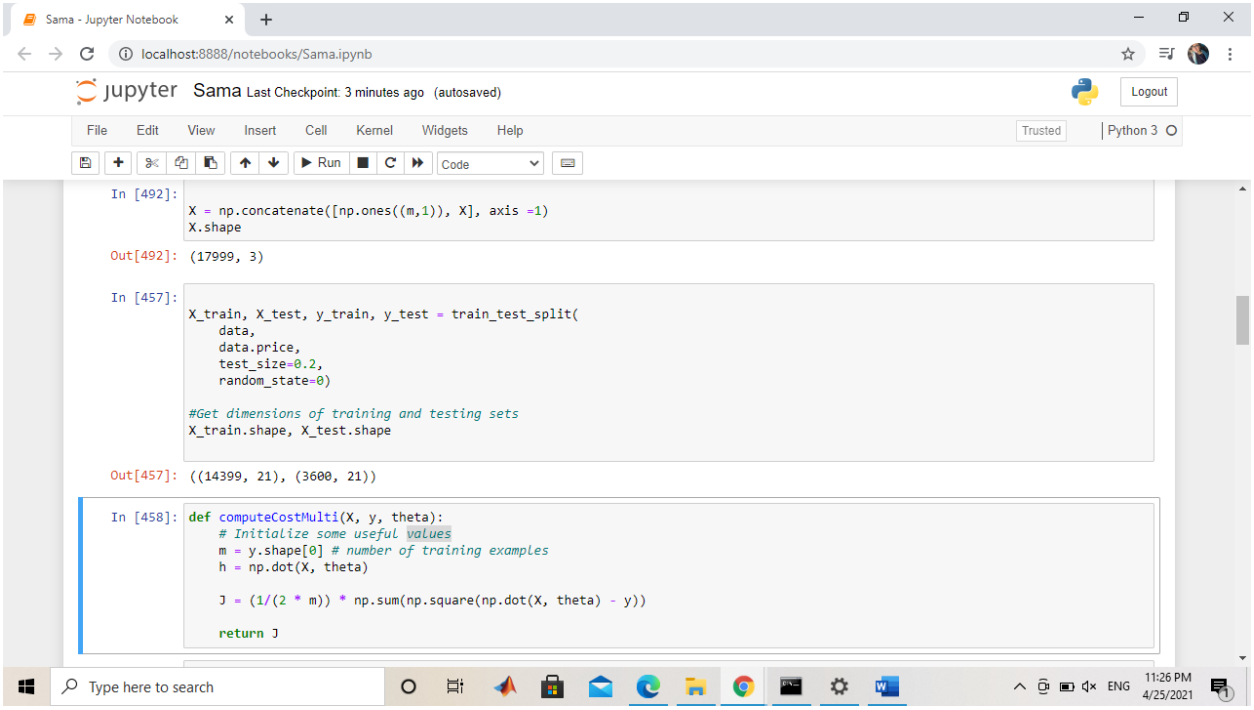
Report

First, we normalize to make sure that the different features take on similar ranges of values so that gradient descents can converge more quickly.



Then we split the dataset into training set and testing set to understand the parameters in the training set better to minimize the error in the testing set



In [492]:
```
X = np.concatenate([np.ones((m,1)), X], axis =1)
X.shape
```
Out[492]: (17999, 3)

In [457]:
```
X_train, X_test, y_train, y_test = train_test_split(
    data,
    data.price,
    test_size=0.2,
    random_state=0)

#Get dimensions of training and testing sets
X_train.shape, X_test.shape
```
Out[457]: ((14399, 21), (3600, 21))

In [458]:
```
def computeCostMulti(X, y, theta):
    # Initialize some useful values
    m = y.shape[0] # number of training examples
    h = np.dot(X, theta)

    J = (1/(2 * m)) * np.sum(np.square(np.dot(X, theta) - y))

    return J
```

Here we are calculating the cost and the gradient descent then plotting the results with different alpha values and different number of iterations. When we increase the alpha(the learning rate) the cost decrease.

Sama - Jupyter Notebook

localhost:8888/notebooks/Sama.ipynb

jupyter Sama Last Checkpoint: 10 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Out[457]: ((14399, 21), (3600, 21))

In [458]:
```
def computeCostMulti(X, y, theta):
    # Initialize some useful values
    m = y.shape[0] # number of training examples
    h = np.dot(X, theta)

    J = (1/(2 * m)) * np.sum(np.square(np.dot(X, theta) - y))

    return J
```

In [459]:
```
def gradientDescentMulti(X, y, theta, alpha, num_iters):
 # Peform a single gradient step on the parameter vector theta.

  # Initialize some useful values
  m = y.shape[0] # number of training examples

  # make a copy of theta, which will be updated by gradient descent
  theta = theta.copy()

  J_history = []

  for i in range(num_iters):

      theta = theta - (alpha / m) * (np.dot(X, theta) - y).dot(X)

      # save the cost J in every iteration
      J_history.append(computeCostMulti(X, y, theta))
```

Type here to search

ENG 11:33 PM 4/25/2021

Sama - Jupyter Notebook

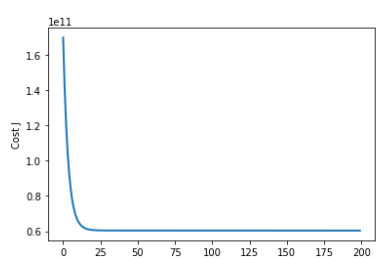
localhost:8888/notebooks/Sama.ipynb

jupyter Sama Last Checkpoint: 12 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

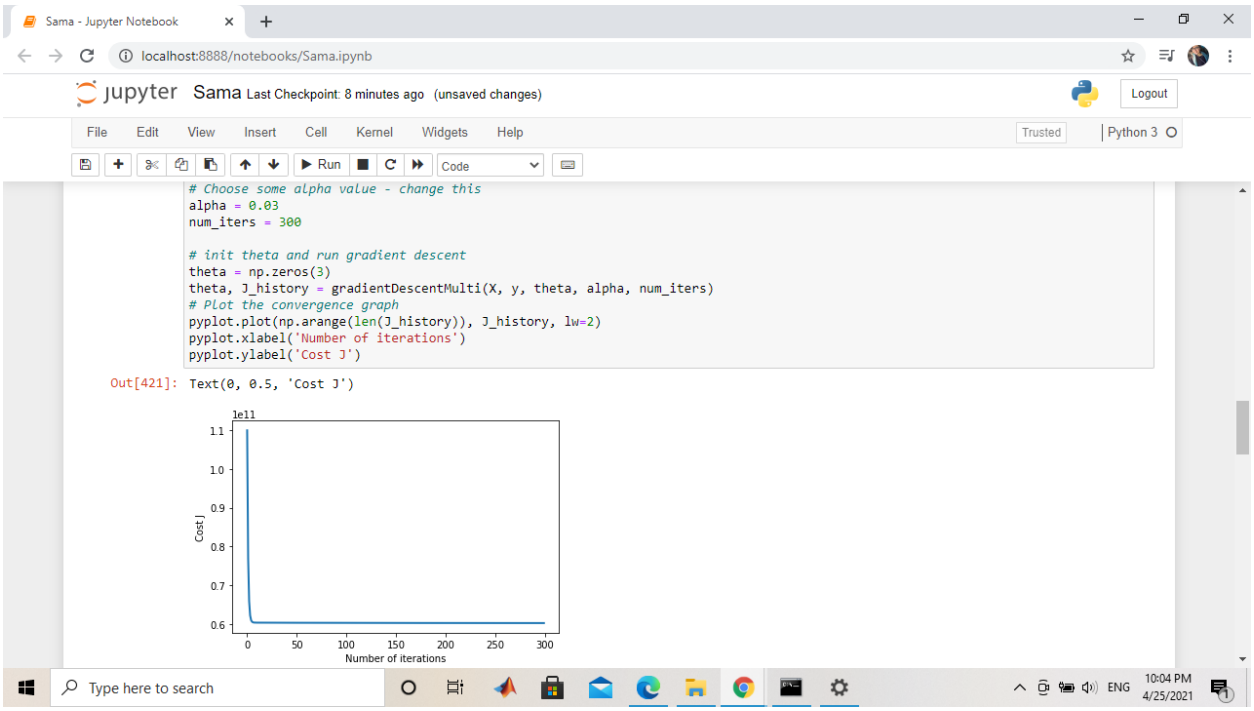
In [460]:
```
# Choose some alpha value - change this
alpha = 0.01
num_iters = 200

# init theta and run gradient descent
theta = np.zeros(3)
theta, J_history = gradientDescentMulti(X, y, theta, alpha, num_iters)
# Plot the convergence graph
pyplot.plot(np.arange(len(J_history)), J_history, lw=2)
pyplot.xlabel('Number of iterations')
pyplot.ylabel('Cost J')
```

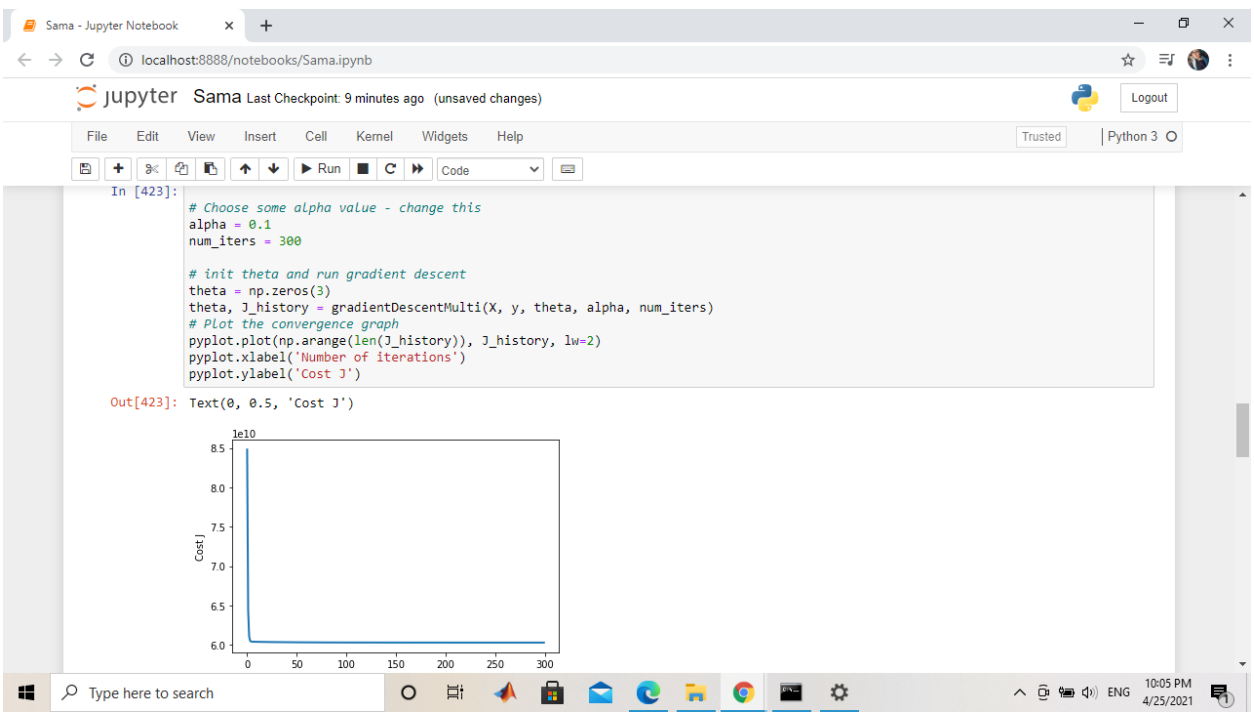
Out[460]: Text(0, 0.5, 'Cost J')



Type here to search

ENG 11:35 PM 4/25/2021





Here we calculate the cost and the gradient descent with different polynomial degree to increases the number of input features. To know how much this impacts the number of features, we can perform the transform with a range of different degrees and compare the number of features in the dataset.

Sama - Jupyter Notebook

localhost:8888/notebooks/Sama.ipynb

jupyter Sama Last Checkpoint: 17 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Notebook saved Trusted Python 3

In [463]:
```
def computeCostMulti3(X, y, theta):
    #  initialize some useful values

    m = y.shape[0]  #number of training examples
    h = np.dot(X, theta)
    J = (1/(2 * m)) * np.sum(np.dot(np.power(X,3), theta) - y)
#==========================
    return J
```

In [464]:
```
def gradientDescentMulti3(X, y, theta, alpha, num_iters):
   # Peform a single gradient step on the parameter vector theta.

    # Initialize some useful values
    m = y.shape[0] # number of training examples

    # make a copy of theta, which will be updated by gradient descent
    theta = theta.copy()

    J_history = []

    for i in range(num_iters):

        theta = theta - (alpha / m) * (np.dot(np.power(X,3), theta) - y).dot(np.power(X,3))

        # save the cost J in every iteration
        J_history.append(computeCostMulti3(X, y, theta))
    return theta, J_history
```

Type here to search ENG 11:40 PM 4/25/2021

The graphs with the 3rd polynomial with different alpha values

Sama - Jupyter Notebook

localhost:8888/notebooks/Sama.ipynb

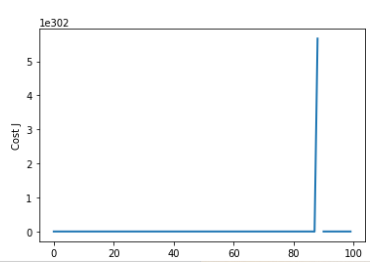
jupyter Sama Last Checkpoint: 23 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

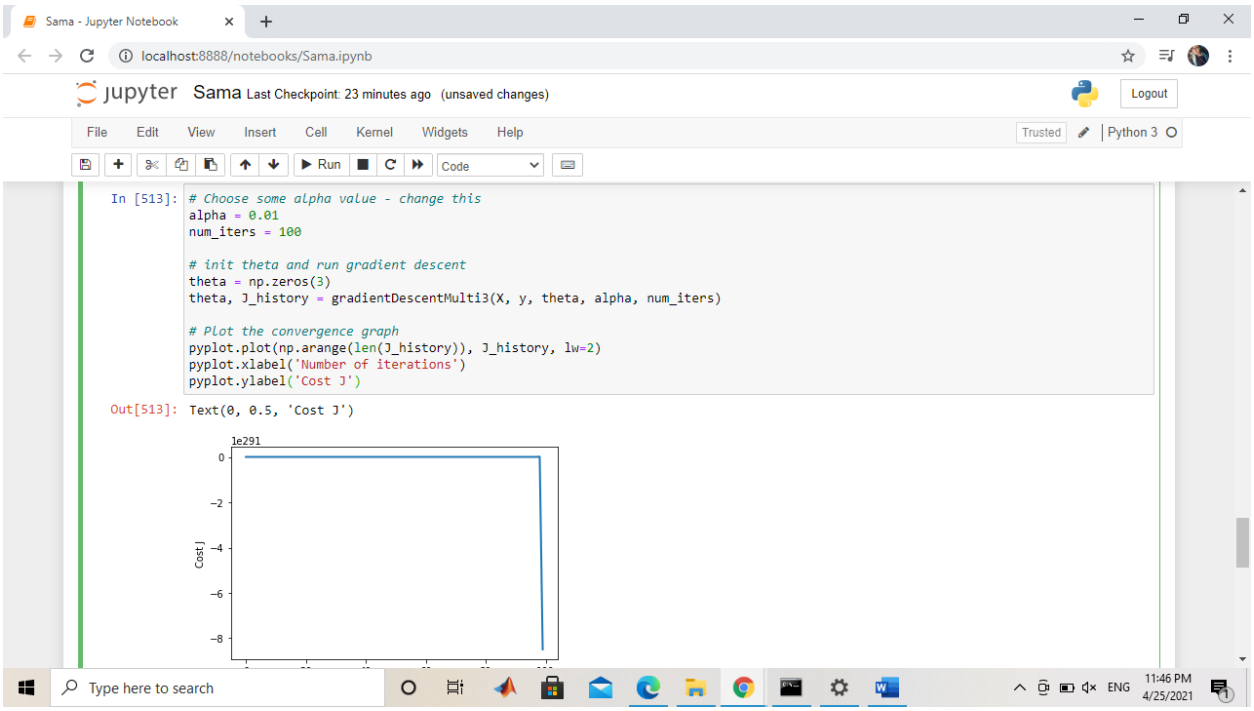
Trusted Python 3

In [514]:
```
# Choose some alpha value - change this
alpha = 0.03
num_iters = 100

# init theta and run gradient descent
theta = np.zeros(3)
theta, J_history = gradientDescentMulti3(X, y, theta, alpha, num_iters)

# Plot the convergence graph
pyplot.plot(np.arange(len(J_history)), J_history, lw=2)
pyplot.xlabel('Number of iterations')
pyplot.ylabel('Cost J')
```

Out[514]: Text(0, 0.5, 'Cost J')



Type here to search ENG 11:47 PM 4/25/2021



Here we are using the K-fold sampling to have less biased dataset because it ensures that every observation from the original dataset has the chance of appearing in training and test set.