Thasina Tabashum

1. Test Points

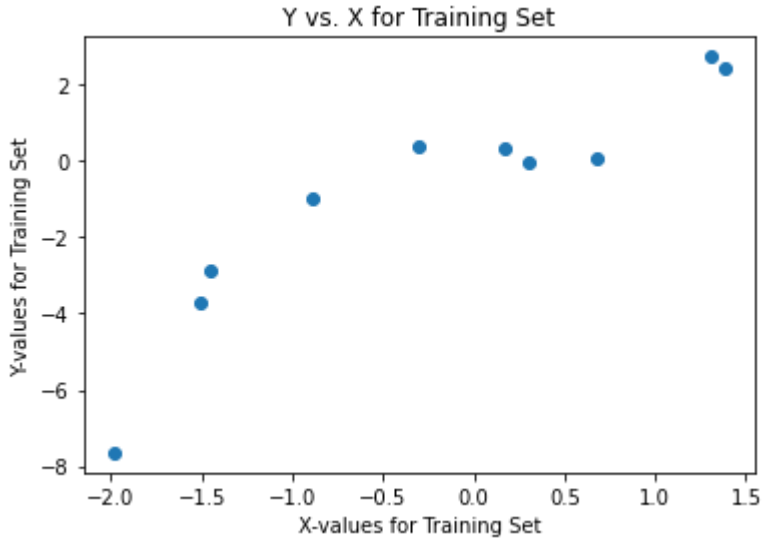
```
from random import randint
import numpy as np
import matplotlib.pyplot as plt
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn import linear_model
from sklearn.metrics import mean_squared_error
import math
from math import sqrt


def y_generator(x):
    return x**3 + 0.5 * np.random.normal(0,1,1)

print(y_generator(10))
```

```
[999.76083334]
```

```
f2 = np.vectorize(y_generator)

np.random.seed(100)

#Getting 10 x-values from the x data set
#X_training = np.random.uniform(0,3,10)
X_training = np.sort(np.random.uniform(-2,2,10))
#Getting 10 y-values from the y data set that has been computed
y_training = f2(X_training)

print("----X values for training set----")
print(X_training)

print()
print("----Y values for training set----")
print(y_training)

%matplotlib inline
#Plots the data points in the training set
#plt.xlim(0,3)
#plt.ylim(0,1.3)
plt.plot(X_training,y_training,'o')
plt.xlabel('X-values for Training Set')
plt.ylabel('Y-values for Training Set')
plt.title('Y vs. X for Training Set')
```

```
plt.show()
```

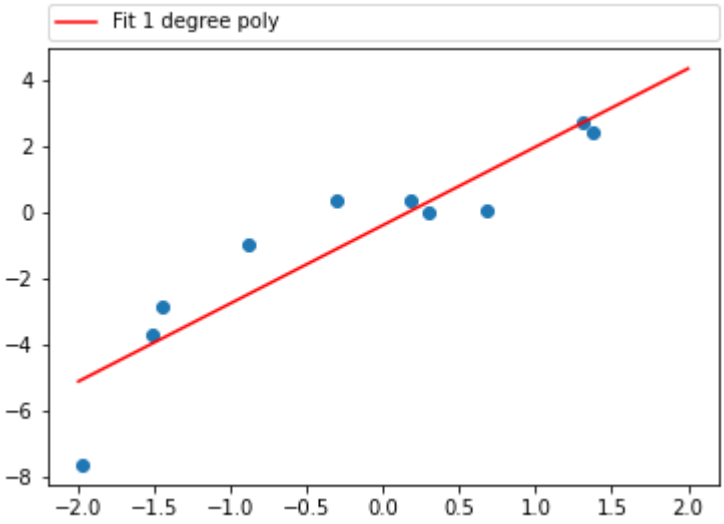
```
----X values for training set----
[-1.98112458 -1.51372352 -1.45317364 -0.88652246 -0.30192964  0.17361977
  0.30037332  0.68299634  1.30341102  1.37910453]

----Y values for training set----
[-7.64812515 -3.69749732 -2.85110484 -0.98853509  0.38089918  0.34159397
 -0.02510465  0.05296668  2.72920563  2.4038915 ]
```



2. Create Graphs

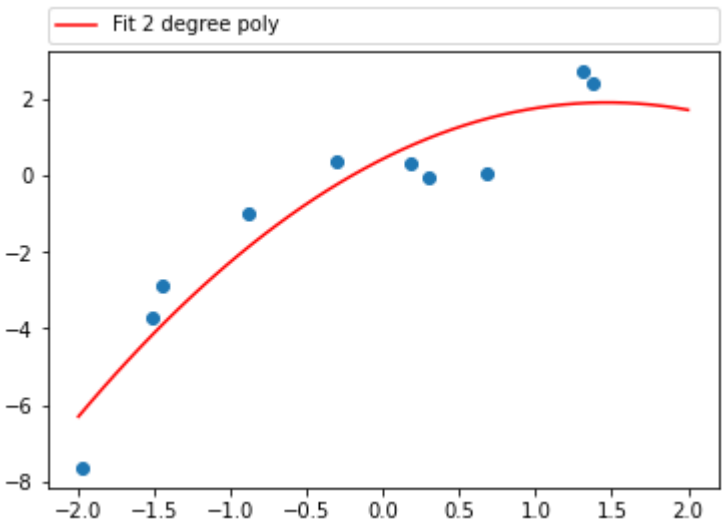
a degree 1 (linear) regression model,

```
X_new = np.linspace(-2,2)
#print(X_new)

model1 = Pipeline([('poly', PolynomialFeatures(degree=1)),('linear', linear_model.LinearRegre
model1 =model1.fit(X_training[:,np.newaxis], y_training[:,np.newaxis])

y_new1 = model1.predict(X_new[:, np.newaxis])


#Plotting the degree 1 fit on the test data
plt.scatter(X_training, y_training)
plt.plot(X_new, y_new1, 'r', label="Fit "+str(1)+ " degree poly")

plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc=3,
               ncol=2, mode="expand", borderaxespad=0.)
#plt.xlim(0,3)
#plt.ylim(0,1.3)

plt.show()
```



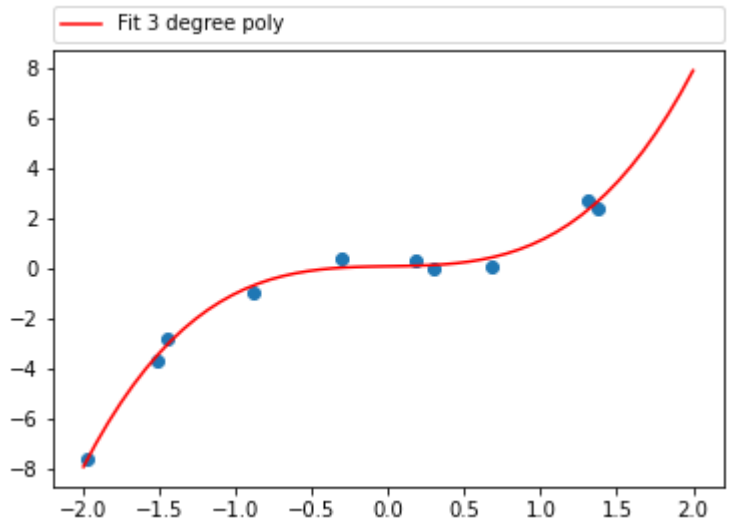
degree 2 (quadratic) regression model,

```
model2 = Pipeline([('poly', PolynomialFeatures(degree=2)),('linear', linear_model.LinearRegre
model2 =model2.fit(X_training[:,np.newaxis], y_training[:,np.newaxis])

y_new2 = model2.predict(X_new[:, np.newaxis])

#Plotting the degree 1 fit on the test data
plt.scatter(X_training, y_training)
plt.plot(X_new, y_new2, 'r', label="Fit "+str(2)+ " degree poly")
plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc=3,
               ncol=2, mode="expand", borderaxespad=0.)
#plt.xlim(0,3)
#plt.ylim(0,1.3)

plt.show()
```



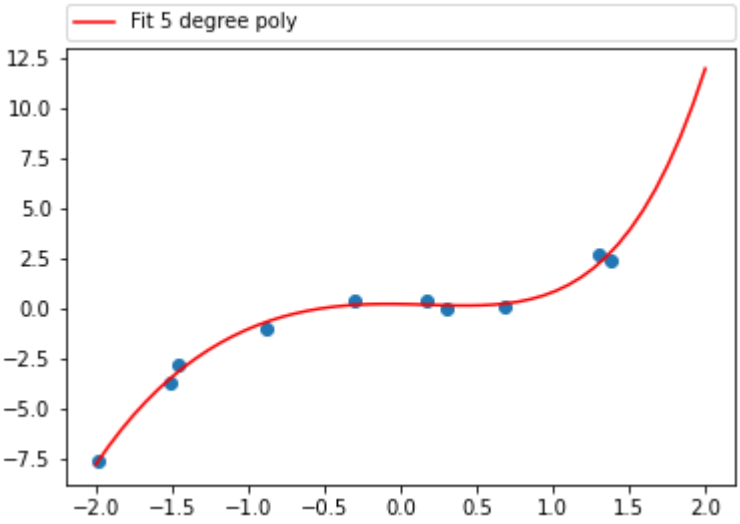
degree 3 (cubic) regression model,

```
model3 = Pipeline([('poly', PolynomialFeatures(degree=3)),('linear', linear_model.LinearRegre
model3 =model3.fit(X_training[:,np.newaxis], y_training[:,np.newaxis])

y_new3 = model3.predict(X_new[:, np.newaxis])

#Plotting the degree 1 fit on the test data
plt.scatter(X_training, y_training)
plt.plot(X_new, y_new3, 'r', label="Fit "+str(3)+ " degree poly")
plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc=3,
               ncol=2, mode="expand", borderaxespad=0.)
#plt.xlim(0,3)
#plt.ylim(0,1.3)

plt.show()
```



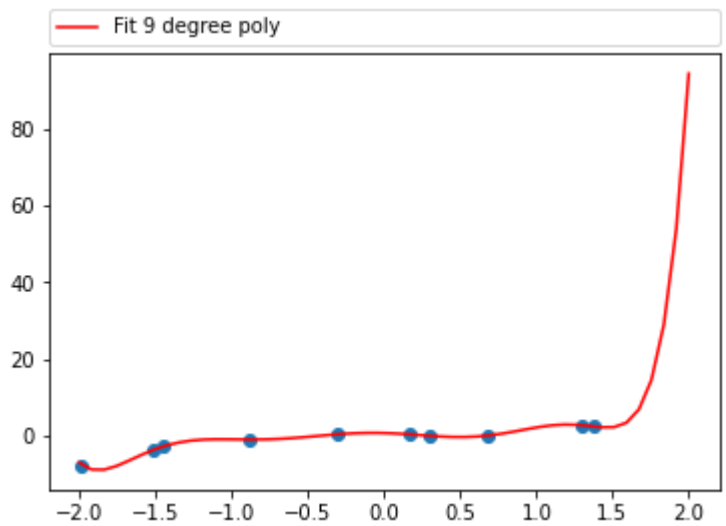
degree 5,

```
model5 = Pipeline([('poly', PolynomialFeatures(degree=5)),('linear', linear_model.LinearRegre
model5 =model5.fit(X_training[:,np.newaxis], y_training[:,np.newaxis])

y_new5 = model5.predict(X_new[:, np.newaxis])

#Plotting the degree 1 fit on the test data
plt.scatter(X_training, y_training)
plt.plot(X_new, y_new5, 'r', label="Fit "+str(5)+ " degree poly")
plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc=3,
               ncol=2, mode="expand", borderaxespad=0.)
#plt.xlim(0,3)
#plt.ylim(0,1.3)

plt.show()
```



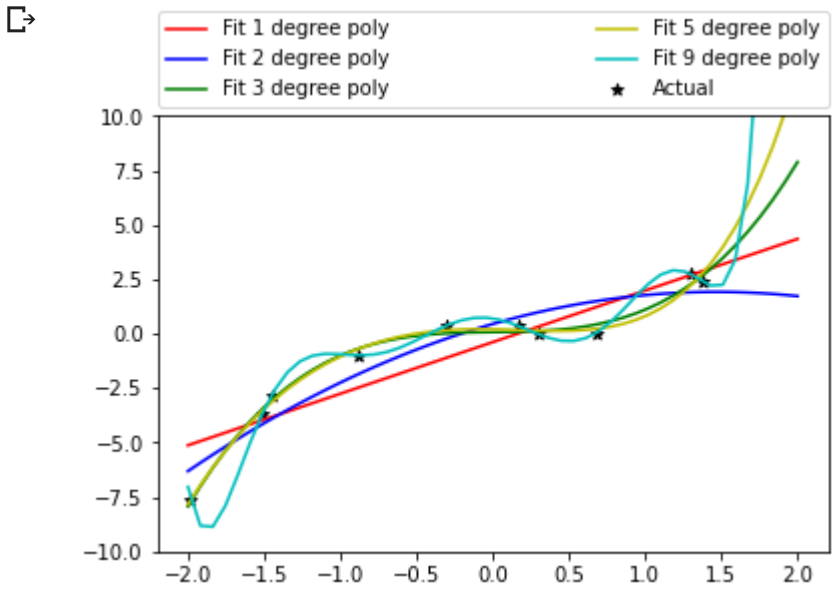
degree 9.

```
model9 = Pipeline([('poly', PolynomialFeatures(degree=9)),('linear', linear_model.LinearRegre
model9 =model9.fit(X_training[:,np.newaxis], y_training[:,np.newaxis])

y_new9 = model9.predict(X_new[:, np.newaxis])

#Plotting the degree 1 fit on the test data
plt.scatter(X_training, y_training)
plt.plot(X_new, y_new9, 'r', label="Fit "+str(9)+ " degree poly")
plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc=3,
               ncol=2, mode="expand", borderaxespad=0.)
#plt.xlim(0,3)
#plt.ylim(0,1.3)

plt.show()
```



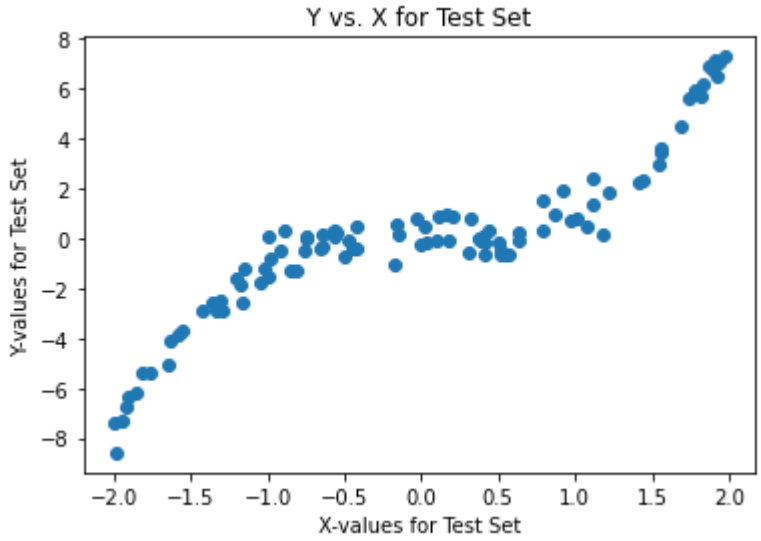
3. Combine Graphs

```
plot_col=['r', 'b', 'g', 'y', 'c']
plt.scatter(X_training, y_training, color = 'black', marker = '*', label="Actual")
d_degree = [1,2,3,5,9]
i = 0
for degree in d_degree:
    model = Pipeline([('poly', PolynomialFeatures(degree=degree)),('linear', linear_model.Lin
    model=model.fit(X_training[:,np.newaxis], y_training[:,np.newaxis])

    predict_sk=model.predict(X_new[:,np.newaxis])

    plt.plot(X_new, predict_sk, plot_col[i], label="Fit "+str(degree)+ " degree poly")
    plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc=3,
               ncol=2, mode="expand", borderaxespad=0.)
    #plt.xlim(-3,3)
    plt.ylim(-10,10)
    i = i+1

plt.show()
```



4. Test the Models

```
X_test = np.random.uniform(-2,2,100)
y_test = f2(X_test)

plt.plot(X_test,y_test,'o')
plt.xlabel('X-values for Test Set')
plt.ylabel('Y-values for Test Set')
plt.title('Y vs. X for Test Set')
plt.show()
```



5. The Results

```
y_predictions1 = model1.predict(X_test[:, np.newaxis])
rms1 = sqrt(mean_squared_error(y_test,y_predictions1))
print('Root mean square error for Degree 1')
print(rms1)


#Root mean squared error
y_predictions2 = model2.predict(X_test[:, np.newaxis])
rms2 = sqrt(mean_squared_error(y_test,y_predictions2))
print('Root mean square error for Degree 2')
print(rms2)

#Root mean squared error
y_predictions3 = model3.predict(X_test[:, np.newaxis])
rms3 = sqrt(mean_squared_error(y_test,y_predictions3))
print('Root mean square error for Degree 3')
print(rms3)

#Root mean squared error
y_predictions5 = model5.predict(X_test[:, np.newaxis])
rms5 = sqrt(mean_squared_error(y_test,y_predictions5))
print('Root mean square error for Degree 5')
print(rms5)


#Root mean squared error
y_predictions9 = model9.predict(X_test[:, np.newaxis])
rms9 = sqrt(mean_squared_error(y_test,y_predictions9))
print('Root mean square error for Degree 9')
print(rms9)
```

```
Root mean square error for Degree 1
1.4902410023308106
Root mean square error for Degree 2
1.8292871801491588
Root mean square error for Degree 3
0.5385454583069076
Root mean square error for Degree 5
1.0665111079926226
Root mean square error for Degree 9
12.374312194587194
```