Project #1

a)

Aim of the project: Learning Linear and Non-Linear Model; Analysis of Uncertainty.

b)

We have implemented models in Python 2.

We have used the following packages in our program -

- Numpy - To perform all mathematical and array operations.
- Matplotlib - To plot the graph of Root mean square error vs Polynomial order no.
- Pandas - Load data from dataset
- Scikit-learn - To create Linear/Non Linear models, and do cross validation, fitting, prediction etc.
- Math - To perform square root operations.

c)

Mathematical equation in Linear model

idx = Tm(-0.541374251478) + Pr(-0.00153607697961) + Th(3610.9360164) + Sv(0.0545256082025) + -77.5364799032

i.e, In Linear model, The X input consists of the 4 independent columns of our dataset.

Mathematical equation in Non - Linear model for Polynomial Order 2

idx = 46.4115174128 + (46.4115174128)Tm^1 + (1.29645685902)Tm^2 + (0.00244645092515)Pr^1 + (-5301.98595622)Pr^2 + (-0.0562387236544)Th^1 + (0.00569571850555)Th^2 + (1.01270161247e-05)Sv^1 + (-69.8202008068)Sv^2

i.e in Polynomial order 2, the X input consists of each independent column X_i^0, X_i, X_i^2

We created this input array using Scikit-learn PolynomialFeatures.fit_transform() method.

For both the models, the Normal Equation to calculate coefficients are as follows -

$$\hat{w} = (X^T X)^{-1} X^T t$$

We calculated coefficients using Scikit-learn, by fitting the model with LinearRegression.fit() method which calculates the above equation, and then obtaining coefficients from LinearRegression.coeff_.

i) <u>Process for developing the models</u>

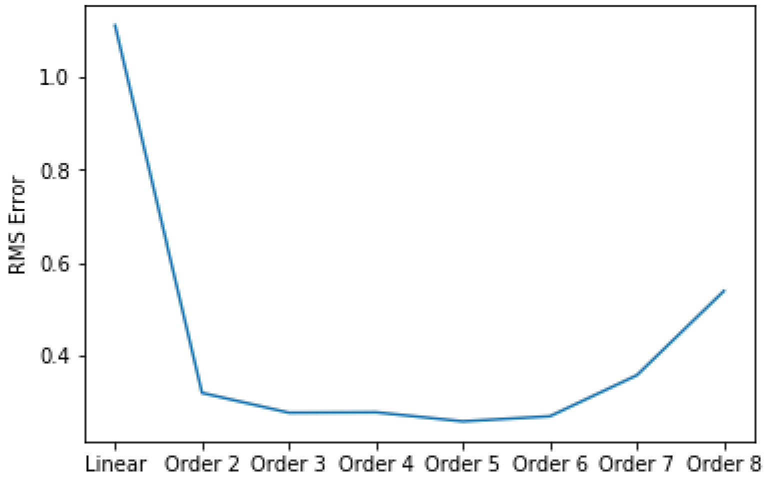
- Read values from data set.
- Separate independent columns into X matrix, and dependent column into Y.
- Create a separate training and test set from X and Y.
- Use scikit-learn to create a Linear model, Fit the training data, get predicted values of Y.
- Calculate Root Mean Square Error, by comparing predicted and test set of Y.
- Get coefficients of the linear model.
- Use scikit-learn to create a Non-Linear model for Polynomial orders 2-8, Fit the training data, get predicted values of Y.
- For each Polynomial order, calculate Root Mean Square Error, by comparing predicted and test set of Y.
- Get coefficients of the non-linear model.

ii) <u>Analyze the uncertainty</u>

We have used Root Mean Square Error (RMSE) to calculate the uncertainties of the model.

$$RMSErrors = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y_i} - y_i)^2}{n}}$$

We have plotted the graph of RMSE for each of the Linear model and Polynomial orders.



We can see that the error is highest for Linear model.
RMSE keeps on decreasing uptill Order 5, and after that increases for Order 6, 7, 8 due to overfitting.

iii) Justification
Linear model is optimal, as it has been calculated using Least Squares approach.

Non-linear model with Polynomial order 5, is the best as it has the least RMSE.

d)
Python 2 Code script

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error

from math import sqrt

# Read dataset from file
dataset = pd.read_csv('Dataset.csv')
numpyData = np.array(dataset)

# Separate independent and dependent columns
X = numpyData[:,:4]
Y = numpyData[:,-1]

# Splitting dataset into training  set and test set
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state
= 0)



'''

LINEAR REGRESSION
```

```
'''

#Fitting multi-variate Linear regression to the Training set
linear_regressor = LinearRegression()
linear_regressor.fit(X_train, Y_train)

print "The Linear model in mathematical equation -"
print "idx = Tm({0}) + Pr({1}) + Th({2}) + Sv({3}) +
{4}".format(linear_regressor.coef_[0],linear_regressor.coef_[1],linear_regressor.coef
_[2],linear_regressor.coef_[3],linear_regressor.intercept_)


#Prediction for Test Set
Y_pred = linear_regressor.predict(X_test)

#Root mean square for Linear regression
rms_linear = sqrt(mean_squared_error(Y_test, Y_pred))

# Arrays to store the RMSE for each method
X_plot = np.array(["Linear","Order 2", "Order 3", "Order 4", "Order 5", "Order 6",
"Order 7", "Order 8"])
Y_plot = np.array([])
Y_plot = np.append(Y_plot,rms_linear)



'''

Non linear regression using Polynomial Order 2 to 8

'''

for deg in range(2,9):
   poly_regressor = PolynomialFeatures(degree = deg)
   X_poly = poly_regressor.fit_transform(X_train)
```

```
    linear_regressor_poly = LinearRegression()
    linear_regressor_poly.fit(X_poly, Y_train)

    # Prediction for test set
    Y_pred = linear_regressor_poly.predict(poly_regressor.fit_transform(X_test))

    #Root mean square for Linear regression
    rms_non_linear = sqrt(mean_squared_error(Y_test, Y_pred))
    print rms_non_linear

    #Append to Plotting array Y axis
    Y_plot = np.append(Y_plot,rms_non_linear)




'''

Uncertainty analysis

'''

plt.ylabel('RMS Error')
plt.plot(X_plot,Y_plot)

# Above graph shows that the Uncertainity decreases uptill Order 5, but increases
after due to Overfitting


# To get Mathematical model for Non linear equation with degree 2

poly_regressor = PolynomialFeatures(degree = 2)
X_poly = poly_regressor.fit_transform(X_train)

linear_regressor_poly = LinearRegression()
linear_regressor_poly.fit(X_poly, Y_train)
```

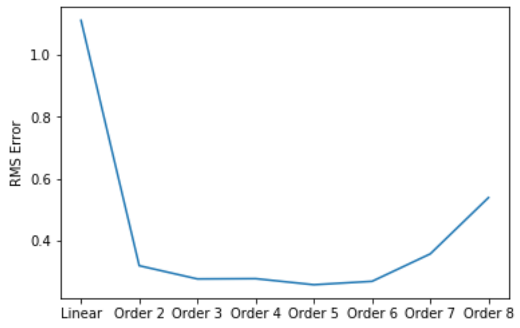
```
print "The Polynomial Order 2 in mathematical equation -"
print "idx = {0} + ({0})Tm^1 + ({1})Tm^2 + ({2})Pr^1 + ({3})Pr^2 + ({4})Th^1 + ({5})Th^2 + ({6})Sv^1 + ({7})Sv^2".format(linear_regressor_poly.intercept_,linear_regressor_poly.coef_[1],linear_regressor_poly.coef_[2],linear_regressor_poly.coef_[3],linear_regressor_poly.coef_[4],linear_regressor_poly.coef_[5],linear_regressor_poly.coef_[6],linear_regressor_poly.coef_[7],linear_regressor_poly.coef_[8],linear_regressor_poly.coef_[9],linear_regressor_poly.coef_[10],linear_regressor_poly.coef_[11],linear_regressor_poly.coef_[12],linear_regressor_poly.coef_[13],linear_regressor_poly.coef_[14])
```

Major screenshot

Uncertainity analysis

```
In [14]: plt.ylabel('RMS Error')
         plt.plot(X_plot,Y_plot)

Out[14]: [<matplotlib.lines.Line2D at 0x7ff06f00c990>]
```



```
# Above graph shows that the Uncertainity decreases uptill Order 5, but increases after due to Overfitting
```