## Summary

I found two online datasets,

- X: Coal usage data of UK for 1913-2018
- Y: Temperature data of cities in the world for 17..-2013

And checked the correlation between coal usage and avg temperature of UK between 1913-2013

## Data

# Y

This is not my data, unfortunately I didn't save the source, and now can not find it..

https://www.kaggle.com/berkeleyearth/climate-change-earth-surface-temperature-data

This data is nearly the same with what I had.

What I had: Yearly temperature of most cities in the world between 17.. – 2013

I only needed United kingdom

Between the years 1913 and 2013

I took the 'year' and 'avg temperature' information

Then took the average of these citywise-temperatures for each year to find the average of whole UK (for each year).

# X

Source: https://www.gov.uk/government/statistical-data-sets/historical-coal-data-coal-production-availability-and-consumption

This data has many interesting parameters relating coal production. Which I correlated with my Y data,

Again only between 1913 and 2013

Code

```
class model:
    def __init__(self):
        pass

    def getData(self, xDataFile, yDataFile):
        self.X = pd.read_excel(xDataFile).values
        self.X_years_label = self.X[25:-13,0]
        self.Y = pd.read_csv(yDataFile)
        self.X = self.X[25:-13,1:]     # The rest is description strings
        self.Y = self.Y.values[30000:60000, :]

    def fixX(self):  # Changing NaN's with the column avgs
        avgs = np.ones(self.X.shape[1])
        for i in range(0,self.X.shape[1]):
            for j in range(0,self.X.shape[0]):
                if self.X[j,i] == '..':
                    self.X[j,i] = -1
        self.X = self.X.astype('float64')
        nans = np.argwhere(self.X!=self.X)
        for i in nans:
            self.X[i] = -1
        avg = 0
        count = 0
        avgs = np.ones(self.X.shape[1])
        for i in range(0,self.X.shape[1]):
            avg = 0
            count = 0
            for j in range(0,self.X.shape[0]):
                if self.X[j,i] != -1:
                    avg = avg + self.X[j,i]
                    count = count + 1
            avgs[i] = avg / count
            avg = 0
            count = 0
        for i in range(0,self.X.shape[1]):
            for j in range(0,self.X.shape[0]):
                if(self.X[j,i] == -1):
                    self.X[j,i] = avgs[i]
        self.X = self.X[:-6,:].T
        self.X_train = self.X[:,:80]
        self.X_test = self.X[:,20:]
```

Description in next page:

Function: getData

I get the data from my files, discard the unnecessary columns and rows

Put them into numpy arrays

Function: fixX

I see there are NaN values in the data that aren't convertible to float directly, null fields written as '..' by the UK government workers.

I detected these anomalies and replaced them with the average of their column. Thinking this would be the safest way to go about it. (Given I didn't have much time left)

Lastly, separate the X data into train and test sets, %80, %20

Function: fixY

I had to get the average temperature of the whole UK by averaging the temperatures of cities for each year.

Lastly, separate the Y data into train and test sets, %80, %20

```
def fixY(self): # Getting the avgd temp data of UK between 1913-2013
    count = 0
    for i in range(self.Y.shape[1]):
        for j in range(self.Y.shape[0]):
            if self.Y[j,4] == 'United Kingdom':
                count = count + 1

    y = np.empty([count, 2])
    c = 0
    for i in range(0, self.Y.shape[1]):
        for j in range(0, self.Y.shape[0]):
            if self.Y[j,4] == 'United Kingdom':
                y[c,0] = int(self.Y[j,0].split('-')[0])
                y[c,1] = self.Y[j,1]
                c = c + 1

    y_avgD = np.zeros([2013-1743+1,2])
    start_year = 1743
    index = 0
    for i in range(y.shape[0]):
        index = int(y[i,0]-1743)
        y_avgD[index,1] = y_avgD[index,1] + y[i, 1]
        y_avgD[index,0] = y[i,0]
    y_avgD[:,1] = y_avgD[:,1]/(2013-1743)
    self.Y = y_avgD[170:-1,1]
    self.Y_train = self.Y[:80]
    self.Y_test = self.Y[20:]


def fit(self):
    self.coefficients = alg.inv(self.X_train @ self.X_train.T) @ self.X_trai
    return self.coefficients
```

Function: fit

This is a single line matrix calculation that returns the coefficients for a linear multivariate approximation

```
    def predict(self):
        self.y_pred = self.coefficients @ self.X_test
        return self.y_pred

    def plot(self):
        plt.plot(np.linspace(1,self.Y_test.shape[0],self.Y_test.shape[0]), self.
        plt.plot(np.linspace(1,self.y_pred.shape[0],self.y_pred.shape[0]), self.
        plt.legend()
        plt.show()                                                    # Plot

    def error(self):
        self.error = np.abs(self.y_pred - self.Y_test).mean()/self.Y_test.mean()
        return self.error
```

In [149]:
```
m = model()

m.getData('coal.xls', 'temp.csv')
m.fixX()
m.fixY()
```

In [150]:
```
m.fit()
m.predict()
m.plot()
print('Error: ', m.error())

print('I had 100 years, used the first 80 to create coefficients, the rest to te
```

## Function: predict

Simple matrix multiplication of X_test and coefficients. Results in y_prediction matrix.

## Other points of interests,

Matplotlib helps to visualize the data in Cartesian coordinate system.

Error is the percentage of the difference between our prediction and the y_test values to mean of y_test values.

Then invoking the written functions to demonstrate.

Thank you for your interest!