Question 2

1 Team

Team Members	*Shuyang Ye*	*Jinlin Zhu*
Student ID	*96481163*	*96057468*
CS ID	*l8n8s*	*q6j8i*

Kaggle Team Name	*TESNO.1*

2 Solution Summary

Three different methods are adopted individually. The most intuitive and simplest model is to use only Canada's past deaths to predict future deaths. The predicted future deaths curve is a straight line which means the daily deaths is a constant. The second method calculates the daily deaths and adopts Auto Regression model to predict the future daily deaths and then the future total deaths. In the end, I stack this two method together to predict the final result.

Third method is to consider the impact from other countries. Given that case density is more informative than the total number of cases for national outbreak correlation analysis, we used the total national death toll density as an indicator to calculate the **Cross Correlation** between different countries, thus selecting countries with similar outbreaks. Thus we can establish a linear map between deaths density of other countries and Canada. Using AR model own each other countries to predict the future deaths, then using the linear map to predict the deaths of Canada.

Finally we adopted the third method that is to combine factors of impact from other countries and the spec country "Canada" together.

3 Experiments

The formula we used to predict the future deaths of Canada is Deaths(CA,Future) = Num_people_100k * Deaths_100k(CA,Future) and

3.1 Auto Regression Model

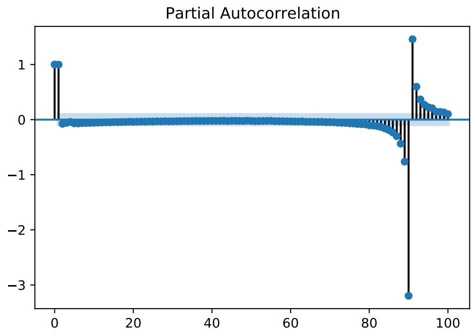
As for the final method, we used this model to predict the short term future of feature "deaths_100k" of all countries. We coded two classes called My_AutoReg and AutoReg1 in "AutoRegression.py" file based on the same algorithm provided by instructions of the Question 2.

3.2 Select lags

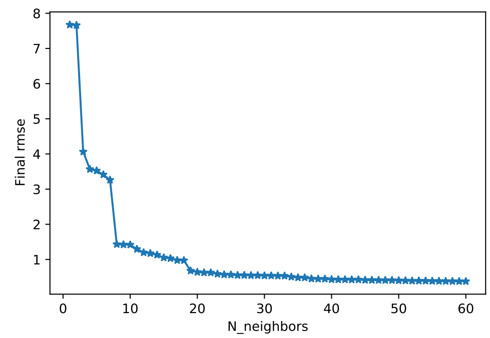
The main hyper-parameter of AR model is the lags we used to generate the Matrix X, by which we predict the value of next day based on the past "num=lags" days. We choose lags by plotting the Partial Autocorrelation Plot (Figure (a)), we can see that only first two past days have significant impact on the value of the present day. Thus we choose **Lags = 2**. We adopted the same lags for all countires future predicitons.

3.3 Multivariable Linear Regression Model

We adopted Linear Regression for feature "deaths_100k" of all NearestNeighbors Countries as X and the feature "deaths_100k" of Canada as y. Thus we could get a Linear relationship and could make "Parallel" Predictions to the death density of Canada.



(a) PACF plot of feature "deaths_100k" of Canada



(b) Training Error vs N_Nearest_Neighbors plot

3.4 Select N Nearest Neighbors

We finally used the **Cross Correlation** to measure "distances" between other countries. This distances is actually the similarities. We noticed that the deaths density between two selected countries have significantly higher correlation. However, the starting date of the pandemic outbreak of each country is different by days. So we created a loop to shift the deaths_100k data from other countries by "num=lag_country" days until they has their best correlation with the spec country "CA".

We used these nearest countries to implement a multivariable linear regression from their feature "deaths_100k" as Input Matrix X to the feature "deaths_100k" of the spec country "CA" as y vector. We selected the period of 269 days as training set and newest 11 days as validation set. We ploted training error plots vs N_Nearest_Neighbors as Figure (b). For training phase, we adopted "elbow" method and find approximate best num is 7 or 8 or 9. We chose **N_Nearest_Neighbors = 7** in the end.

4 Results

Team Name	Kaggle Phase 1 Score	Kaggle Phase 2 Score
TESNO.1	*10.865*	*459.16159*

5 Conclusion

The final result is good enough according to our very simple model. We have learned how to code the Auto Regression model, how to select features and hyper-parameters of our models in practice. We could try more features and created more sophisticated modesl if given more time.

6 Code

```
import argparse
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import time
from pandas.plotting import autocorrelation_plot
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf
#from statsmodels.tsa.ar_model import AutoReg

from statsmodels.tsa import stattools
from sklearn.metrics import mean_squared_error
from scipy.stats import pearsonr
from math import sqrt
import seaborn as sns
#from sklearn.neighbors import NearestNeighbors
import linear_model
import utils
import country
import warnings
from AutoRegression import My_AutoReg
from AutoRegression import AutoReg1

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('-q','--question', required=True)
    parser.add_argument('-d','--test_day', required=False)
    parser.add_argument('-l','--test_lag', required=False)
    parser.add_argument('-n','--test_num', required=False)
    parser.add_argument('-m','--test_method', required=False)
    parser.add_argument('-p','--test_par', required=False)

    io_args = parser.parse_args()
    question = io_args.question


    def read_dataset(filename):
        with open(os.path.join("..", "data", filename), "rb") as f:
            dataset = pd.read_csv(f,keep_default_na=False)
        return dataset

    if question == 'find_nan_ct':
        dataset = read_dataset("phase1_training_data.csv")
        n_ct, ct_key, ct_mapper, ct_inv_mapper, ct_ind =utils.make_mapper(dataset)
        dataset=utils.get_all(dataset)
        dataset_CA = dataset[dataset[ct_key]=="CA"]

        print("Input Country Number:",n_ct)
        n_ct1=0
        i=0
        ct_nan_ind=n_ct*[None]
        for nct in range(n_ct):
            if type(ct_inv_mapper[nct]) == str:
                n_ct1 +=1
            else:
                ct_nan_ind[i] = nct
                i+=1
        print("Output Country Number:",n_ct1)
        if n_ct1 == n_ct:
            print("No 'nan' named country found!")

```

```python
    if question == 'lagplot':
        dataset = read_dataset("phase1_training_data.csv")
        dataset=utils.get_all(dataset)
        n_ct, ct_key, ct_mapper, ct_inv_mapper, ct_ind =utils.make_mapper(dataset)

        dataset_CA=dataset[dataset[ct_key]=='CA']
        #cases_CA = dataset_CA["cases"]
        #deaths_CA = dataset_CA["deaths"]

        testname ="deaths_100k"
        test_CA = dataset_CA[testname]
        #autocorrelation_plot(cases_CA)

        #pacf analysis
        nlags = 300
        #conf_CA = np.zeros(2)
        pacf_CA, conf_CA = stattools.pacf(test_CA, nlags=nlags,alpha=0.05)
        #pacf1 = pacf_CA[pacf_CA > conf_CA[1] or pacf_CA < conf_CA[0]]
        conf0 = conf_CA[:,0]
        conf1 = conf_CA[:,1]
        ind_lags = np.argsort(pacf_CA)
        ind_lags = ind_lags[::-1]

        #pacf_lag_plot for test dataset with testname

        nlags_max =100
        #pacf output
        plot_pacf(test_CA,lags=nlags_max)
        #plotname = testname+"_pacf_lag_plot_CA.pdf"
        plotname = testname+"_pacf_lag_plot_CA.pdf"
        fname = os.path.join("..", "figs", plotname)
        #plt.xlim(left = 80)
        plt.savefig(fname)
        plt.clf()

        plt.plot(conf0)
        plt.plot(conf1)
        plotname = "conf.pdf"
        fname = os.path.join("..", "figs", plotname)
        plt.xlim(left = 80)
        plt.savefig(fname)
        plt.clf()

        testname=testname
        refname="date"
        sns.scatterplot(dataset_CA[refname],dataset_CA[testname])
        plotname = testname+"_CA.pdf"
        fname = os.path.join("..", "figs", plotname)
        plt.xlim(left = 50)
        plt.savefig(fname)
        plt.clf()

    if question == 'ar_ca':
        dataset = read_dataset("phase1_training_data.csv")
        #Input arg parameters
        test_lag = io_args.test_lag
        test_lag = int(test_lag)
        test_day= io_args.test_day
        test_day = int(test_day)

        n_ct, ct_key, ct_mapper, ct_inv_mapper, ct_ind =utils.make_mapper(dataset)

        dataset_CA=dataset[dataset[ct_key]=='CA']
        cases_CA = dataset_CA["cases"]
        deaths_CA = dataset_CA["deaths"]
```

```
        #Choose experiment set
        exp_set = deaths_CA
        exp_set = exp_set.reset_index(drop = True)

        X = exp_set
        train_set, test_set = X[:len(X)-test_day], X[len(X)-test_day:]

        # train autoregression
        lags=test_lag
        model1 = AutoReg1(lags=lags)
        model1.fit(train_set)

        model3 = My_AutoReg(lag=lags)
        model3.fit(train_set)
        #print('Coefficients: %s' % model_fit.params)

        # make predictions
        predictions_train= model1.predict(start=lags, end=len(train_set)-1)
        # predictions_train= model3.predict(train_set,start=lags, end=len(train_set)-1)
        # predictions_test= model.predict(start=len(train_set), end=len(train_set)+len(
    test_set))
        # predictions_all= model.predict(start=lags, end=len(train_set)+len(test_set))
        #Train Error
        rmse_train = sqrt(mean_squared_error(train_set[lags:], predictions_train))
        print('Train RMSE: %.3f' % rmse_train)
        # rmse_test = sqrt(mean_squared_error(test_set, predictions_test))
        # print('Test RMSE: %.3f' % rmse_test)
        # plot results
        plt.plot(train_set[lags:])
        plt.plot(predictions_train, color='red')
        fname = os.path.join("..", "figs", "prediction_train_plot_CA.pdf")
        plt.savefig(fname)
        plt.clf()

        # plt.plot(test_set)
        # plt.plot(predictions_test, color='red')
        # fname = os.path.join("..", "figs", "prediction_test_plot_CA.pdf")
        # plt.savefig(fname)
        # plt.clf()


        # plt.plot(exp_set[lags:])
        # plt.plot(predictions_all, color='red')
        # fname = os.path.join("..", "figs", "prediction_all_plot_CA.pdf")
        # plt.xlim(left = len(exp_set[lags:]) -2*len(test_set))
        # plt.savefig(fname)
        # plt.clf()
        # plt.close('all')

    # if question == 'knn':
    #     dataset = read_dataset("phase1_training_data.csv")
    #     test_num= io_args.test_num
    #     test_num= int(test_num)
    #     test_method= io_args.test_method
    #     #test_method=int(test_method)


    #     n_ct, ct_key, ct_mapper, ct_inv_mapper, ct_ind =utils.make_mapper(dataset)

    #     dataset_CA=dataset[dataset[ct_key]=='CA']
    #     cases_CA = dataset_CA["cases"]
    #     cases14_CA =dataset_CA["cases_14_100k"]
    #     day_n = len(cases_CA)

```

```python
    #     exp_name = "cases_14_100k"
    #     exp_dataset = np.zeros((n_ct,day_n))
    #     exp_ct_name ='CA'

    #     #Choose the dataset regarding the name interested
    #     cases14_exp = dataset[dataset[ct_key]==exp_ct_name][exp_name]

    #     for nct in range(1,n_ct):
    #         #fillter out nan set
    #         dataset_ct = dataset[dataset[ct_key]==ct_inv_mapper[nct]]
    #         dataset1 = dataset_ct[exp_name]
    #         exp_dataset[nct] = dataset1

    #     exp_dataset = exp_dataset[1:]
    #     target= np.zeros((1,day_n))
    #     target[0]= cases14_exp
    #     #!!!!!!remember the true country index is ind_ct = indices+1
    #     #Using KNN model to find K nearest countries
    #     n_neigh = test_num

    #     metric_method =test_method
    #     model = NearestNeighbors(metric=metric_method)
    #     model.fit(exp_dataset)
    #     distances, indices=model.kneighbors(target,n_neighbors=n_neigh)
    #     #!!!!!!remember the true country index is ind_ct = indices+1
    #     indices = indices +1
    #     indices = indices[0]
    #     distances = distances[0]
    #     ct_neigh =n_neigh*[None]
    #     for i in range(n_neigh):
    #         ct_neigh[i] = ct_inv_mapper[indices[i]]
    #     #ct_neigh = [ct_inv_mapper[i] for i in range(len(indices))]
    #     ct_neigh_dict=dict(zip(ct_neigh,list(distances)))
    #     print(ct_neigh_dict)

    #     #Distance Plot of N_nearest countreis
    #     plt.plot(distances[1:])
    #     plt.xlabel("Neighbor_Country")
    #     plt.ylabel("Distance")
    #     plotname="Distance_plot_n"+str(n_neigh)+"_"+exp_ct_name+"_"+metric_method+".pdf"
    #     fname = os.path.join("..", "figs", plotname)
    #     plt.savefig(fname)
    #     plt.clf()
    #     plt.close('all')

    if question == "linear":
        test_par = io_args.test_par

        dataset = read_dataset("phase1_training_data.csv")
        dataset = utils.get_all(dataset)
        n_ct, ct_key, ct_mapper, ct_inv_mapper, ct_ind =utils.make_mapper(dataset)

        dataset_CA=dataset[dataset[ct_key]=='CA']
        cases_CA = dataset_CA["cases"]
        cases14_CA =dataset_CA["cases_14_100k"]
        day_n = len(cases_CA)

        ref_name = "deaths_100k"
        exp_name = "deaths_100k"

        #exp_dataset = np.zeros((n_ct,day_n))
        exp_ct_name ='CA'
        ref_ct_name ='UK'

        exp_dataset= dataset[dataset[ct_key]==exp_ct_name][exp_name]
```

```
        ref_dataset= dataset[dataset[ct_key]==ref_ct_name][ref_name]
        exp_dataset = exp_dataset.reset_index(drop = True)
        ref_dataset = ref_dataset.reset_index(drop = True)

        y=exp_dataset

        test_ratio=11/day_n

        lag = 0

        test_day = int(day_n*test_ratio)
        train_day = day_n-test_day
        train_ref_dataset=ref_dataset[:train_day]
        train_exp_dataset=exp_dataset[:train_day]


        if exp_name == ref_name:
            maxcorr,lag=country.cross_corr(train_ref_dataset,train_exp_dataset,60)
            print("Best Lag =",lag)
            print("Cross correlation =",maxcorr)
        else:
            maxcorr,_=pearsonr(train_ref_dataset,train_exp_dataset)

        print("Correlation =",maxcorr)


        train_ref_dataset=utils.shift_fill0(train_ref_dataset,lag)


        X=utils.s2v(train_ref_dataset)
        y=utils.s2v(train_exp_dataset)
        Xtest=utils.s2v(ref_dataset[train_day:])
        ytest=utils.s2v(exp_dataset[train_day:])

        poly_par =1
        if test_par !=None:
            poly_par = int(test_par)
        model = linear_model.LeastSquaresPoly(p=poly_par)
        model.fit(X,y)
        titlename = exp_name+"_"+exp_ct_name+"_vs_"+ref_name+"_"+ref_ct_name+"_lag"+str(lag)+"_p"+str(poly_par)
        filename = "LRplot_" + titlename + ".pdf"
        utils.test_and_plot(model,X,y,Xtest,ytest,title=titlename,
                            filename=filename)


    if question == "corr":

        dataset = read_dataset("phase1_training_data.csv")
        dataset = utils.get_all(dataset)
        n_ct, ct_key, ct_mapper, ct_inv_mapper, ct_ind =utils.make_mapper(dataset)

        day_n = len(dataset[dataset[ct_key]=='CA']["cases"])

        exp_name = "cases_14_100k"
        exp_ct_name ='CA'

        n_neigh = 100
        cutoff = 0.5
        n_neigh, n_neigh_indices, n_neigh_distances, n_neigh_lags, n_neigh_dict, n_neigh_dict_lags = country.select_country(
            dataset,exp_name,exp_ct_name,n_neigh=n_neigh,method="cross",cutoff=cutoff)
        print("N_neighbor = %d with cutoff = %.3f"%(n_neigh,cutoff))
        print(n_neigh_dict)
        print(n_neigh_dict_lags)
```

```python

        plt.plot(n_neigh_distances)
        plt.xlabel("Neighbor_Country")
        plt.ylabel("Correlation")
        plotname="Ncorr_plot_n"+str(n_neigh)+"_"+exp_ct_name+".pdf"
        fname = os.path.join("..", "figs", plotname)
        plt.savefig(fname)
        plt.clf()
        plt.close('all')

#------------AMLR: Main prediciton model based on  AutoReg and normal Multi-variable Linear Regressions-------

    if question == "AMLR":
        warnings.filterwarnings("ignore")
        test_par = io_args.test_par
        test_day = io_args.test_day
        test_num = io_args.test_num


        dataset = read_dataset("phase1_training_data.csv")
        dataset = utils.get_all(dataset)
        n_ct, ct_key, ct_mapper, ct_inv_mapper, ct_ind =utils.make_mapper(dataset)

        dataset_CA=dataset[dataset[ct_key]=='CA']
        cases_CA = dataset_CA["cases"]
        cases14_CA =dataset_CA["cases_14_100k"]
        day_n = len(cases_CA)

        exp_fea_name = "deaths_100k"
        ref_fea_name = "deaths_100k"
        #exp_dataset = np.zeros((n_ct,day_n))
        exp_ct_name ='CA'
        #ref_ct_name ='UK'

        #Find nearest neighbors of CA,return indices, get a combined all neighbor country dataset
        n_neigh = 100
        cutoff = 0.9
        if test_num != None:
            test_num = int(test_num)
            n_neigh = test_num
        if test_par != None:
            test_par = float(test_par)
            cutoff = test_par
        n_neigh, n_neigh_indices, n_neigh_distances,n_neigh_lags, n_neigh_dict, n_neigh_dict_lags = country.select_country(
            dataset,exp_fea_name,exp_ct_name,n_neigh=n_neigh,
            method="cross",cutoff=cutoff,R=30,ref_fea_name=ref_fea_name)
        print("y is '%s' from country '%s'"%(exp_fea_name,exp_ct_name))
        print("N_neighbor = %d with cutoff = %.3f"%(n_neigh,cutoff))
        #print(n_neigh_dict)

        # ------------------------------------------------------------------------------------------------

        #   Combine the lags list and modify each country, return a dataset of reference
        #   the reference is used to be a X input matrix
        #   all subset of dataset are shifted based on the best lag according to the cross correlation
        #   we are fitting all the correlated countries with the same time-shift
        #   so that the total linear correlation coefficient is significantly improved
        #   we just considered the case that all countries don't have the same date of outbreak of pandemic
        #   this is an important factor to fit our data
```

```
    # ----------------------------------------------------------------------------------------------------
    ref_dataset=country.crosscombine_ct(dataset,ref_fea_name,n_neigh_indices,n_neigh_lags)
    exp_dataset= dataset[dataset[ct_key]==exp_ct_name][exp_fea_name]
    exp_dataset=exp_dataset.reset_index(drop = True)

    test_ratio=11/day_n
    if test_day != None:
        test_day = float(test_day)
        test_ratio = test_day

    test_day = int(day_n*test_ratio)
    train_day = day_n-test_day

    X=ref_dataset[:train_day]
    y=utils.s2v(exp_dataset[:train_day])

    Xtest=ref_dataset[train_day:]
    ytest=utils.s2v(exp_dataset[train_day:])

    self_AR_X=exp_dataset[:train_day]
    self_AR_y=exp_dataset[train_day:]

    #k is the lag chosen as parameter in AutoReg model
    #optimal lags is 2 as simplist optimal lag
    #optimal lags is 12 as second best lag
    k=2
    #Implement auto regression on the self feature of the selected country
    self_auto_model = AutoReg1(lags=k)
    self_auto_model.fit(self_AR_X)
    self_train_predictions = self_auto_model.predict(start=k,end=train_day-1)
    self_train_rmse = sqrt(mean_squared_error(self_AR_X[k:], self_train_predictions))
    self_test_predictions = self_auto_model.predict(start=train_day, end=train_day+test_day-1)
    self_test_rmse = sqrt(mean_squared_error(self_AR_y, self_test_predictions))
    # ----------------------------------------------------------------------------------------------------
    #Implement the parallel Linear Regression on the neighbors of selected country:
    #
    #   Feature(selected country,past)= LR (Feature(other countries,past) )
    #
    #So we can train a model to fit the past feature of selected country
    #Later, we will use this model to parallelly predict the Future Feature of the spec country:
    #
    #   Feature(selected country,future)= a*LR (Feature(other countries,future) ) + b*AR (Feature(spec country,past)) + c
    #
    #Note that we have already shifted the training set so that all outbreaks happend at the same time
    #Based on this we are able to say, the correlation decides whether we can predict the spec country
    # ----------------------------------------------------------------------------------------------------
    parallel_model = linear_model.LeastSquaresPoly(p=1)
    parallel_model.fit(X,y)
    parallel_predictions=parallel_model.predict(X)
    train_parallel_rmse = sqrt(mean_squared_error(parallel_predictions,y))
    print("MultiLinear train Process rmse is %.3f"%train_parallel_rmse)

```

```
        #Implement auto_regression for all neighbors for feature "cases_100k" except selected country
        test_rmse_ct = np.zeros(n_neigh)
        train_rmse_ct = np.zeros(n_neigh)
        test_predictions = np.zeros((n_neigh,test_day))
        for nct in range(n_neigh):
            dataset_ct = dataset[dataset[ct_key]==ct_inv_mapper[n_neigh_indices[nct]]]
            X = dataset_ct[ref_fea_name]
            X = X.reset_index(drop = True)
            train, test = X[:(len(X)-test_day)], X[(len(X)-test_day):]

            auto_model=AutoReg1(lags=k)
            auto_model.fit(train)
            train_predictions = auto_model.predict(start=k,end=len(train)-1)
            train_rmse_ct[nct] = sqrt(mean_squared_error(train[k:], train_predictions))
            test_predictions[nct] = auto_model.predict(start=len(train), end=len(train)+len(test)-1)
            test_rmse_ct[nct] = sqrt(mean_squared_error(test, test_predictions[nct]))

        #Compute all errors happend in the AutoReg process
        #Add up and get the mean value of rmse of each phase of the regression process of each country:
        test_auto_rmse = np.mean(test_rmse_ct)
        train_auto_rmse = np.mean(train_rmse_ct)

        #This is the matrix of the auto predicted value of all other countries
        #This can be considered as an X of a LR model, then we can predict the future feature of spec country
        auto_predictions = test_predictions.T

        #Prediction phase and calculate the final result of the selected feature of the spec country
        MLR_predictions = parallel_model.predict(auto_predictions)
        test_exp_dataset = exp_dataset[train_day:]
        final_rmse = sqrt(mean_squared_error(test_exp_dataset, MLR_predictions))
        baseline = sqrt(mean_squared_error(test_exp_dataset, np.zeros(test_day)))

#------------------Output Phase ----------------------------------------------------------------------------------------------

        print("AutoReg train Process  mean_rmse is %.3f"%train_auto_rmse)
        print("AutoReg test Process(validation process)  mean_rmse is %.3f"%test_auto_rmse)
        print("Final test rmse is %.3f"%final_rmse)
        print("Final test percentage error is %.3f%%"%(final_rmse/baseline*100))
        #Predict the final deaths feature value based on the population information of the spec country
        #Theres almost no errors added here regarding the defination of deaths_100k which is sctrictly proportional to deaths

#------------------deaths feature ----------------------------------------------------------------------------------------------

        if exp_fea_name == ref_fea_name and exp_fea_name =="deaths_100k":
            deaths_test_dataset=dataset[dataset[ct_key]==exp_ct_name]["deaths"][train_day:]
            num_people_100k = utils.get_num_people_100k(dataset,exp_ct_name)

            MLR_deaths_predictions = num_people_100k * MLR_predictions

            self_deaths_predictions = num_people_100k * self_test_predictions

            MLR_deaths_rmse = sqrt(mean_squared_error(MLR_deaths_predictions.T[0], deaths_test_dataset))

            print("Without Self_AR, MultiLR deaths test rmse is %.3f"%MLR_deaths_rmse)

```

```
            # predict the deaths by combining two models together : stacked accroding to the correlation coefficients

            AMLR_deaths_predictions = utils.estimator(MLR_deaths_predictions.T[0], self_deaths_predictions,n_neigh_distances)

            AMLR_deaths_rmse = sqrt(mean_squared_error(AMLR_deaths_predictions, deaths_test_dataset))

            print("With Self_AR, Final AutoMultiLR deaths test rmse is %.3f"% AMLR_deaths_rmse)

            utils.predictions_plot(AMLR_deaths_predictions,deaths_test_dataset,
                                   exp_ct_name,exp_fea_name,"test",n_neigh=n_neigh
                                   )
#-------------------AMLR model training and prediction-----------------Completed--------------------------------------------------------------



    if question == "find_n": #I found that N_neighbors = 7 is best
        warnings.filterwarnings("ignore")
        test_par = io_args.test_par
        test_day = io_args.test_day
        test_num = io_args.test_num


        dataset = read_dataset("phase1_training_data.csv")
        n_ct, ct_key, ct_mapper, ct_inv_mapper, ct_ind =utils.make_mapper(dataset)

        dataset_CA=dataset[dataset[ct_key]=='CA']
        cases_CA = dataset_CA["cases"]
        cases14_CA =dataset_CA["cases_14_100k"]
        day_n = len(cases_CA)

        exp_fea_name = "cases_100k"
        ref_fea_name = "cases_100k"
        #exp_dataset = np.zeros((n_ct,day_n))
        exp_ct_name ='CA'
        #ref_ct_name ='UK'

        #Find nearest neighbors of CA,return indices, get a combined all neighbor country dataset
        # n_neigh = 100
    #N_Neighbor Loop to find N optimal

        n_neigh_max = 60
        n_neigh_min = 1
        cutoff = 0.9
        if test_num != None:
            test_num = int(test_num)
            n_neigh_max=test_num
        if test_par != None:
            test_par = float(test_par)
            cutoff = test_par

        test_ratio=11/day_n
        if test_day != None:
            test_day = float(test_day)
            test_ratio = test_day

        test_day = int(day_n*test_ratio)
        train_day = day_n-test_day
        neigh_final_rmse=np.zeros(n_neigh_max+1)
```

```
        train_parallel_rmse = np.zeros(n_neigh_max+1)
        report_rmse = np.zeros(n_neigh_max+1)
        for n in range(n_neigh_min,n_neigh_max+1):
            n_neigh = n

            n_neigh, n_neigh_indices, n_neigh_distances,n_neigh_lags, n_neigh_dict, n_neigh_dict_lags = country.select_country(
                dataset,exp_fea_name,exp_ct_name,n_neigh=n,
                method="cross",cutoff=cutoff,R=30,ref_fea_name=ref_fea_name)

            #print(n_neigh_dict)

            #ref_dataset=country.combine_ct(dataset,ref_fea_name,n_neigh_indices)
            ref_dataset=country.crosscombine_ct(dataset,ref_fea_name,n_neigh_indices, n_neigh_lags)
            exp_dataset= dataset[dataset[ct_key]==exp_ct_name][exp_fea_name]

            X=ref_dataset[:train_day]
            y=utils.s2v(exp_dataset[:train_day])
            Xtest=ref_dataset[train_day:]
            ytest=utils.s2v(exp_dataset[train_day:])

            # print("y is '%s' from country '%s'"%(exp_fea_name,exp_ct_name))
            # print("N_neighbor = %d with cutoff = %.3f"%(n_neigh,cutoff))
            parallel_model = linear_model.LeastSquaresPoly(p=1)
            parallel_model.fit(X,y)
            parallel_predictions=parallel_model.predict(X)
            train_parallel_rmse[n] = sqrt(mean_squared_error(parallel_predictions,y))
            print("MultiLinear train Process rmse is %.3f"%train_parallel_rmse[n])
            #optimal lags is 2 as simplist optimal lag
            #optimal lags is 12 as second best lag
            k=2
            #Implement auto_regression for all neighbors for feature "cases_100k"
            test_rmse_ct = np.zeros(n_neigh)
            train_rmse_ct = np.zeros(n_neigh)
            test_predictions = np.zeros((n_neigh,test_day))
            for nct in range(n_neigh):
                dataset_ct = dataset[dataset[ct_key]==ct_inv_mapper[n_neigh_indices[nct]]]
                X = dataset_ct[ref_fea_name]
                X = X.reset_index(drop = True)
                train, test = X[:(len(X)-test_day)], X[(len(X)-test_day):]

                auto_model=AutoReg1(lags=k)
                auto_model.fit(train)
                train_predictions = auto_model.predict(start=k,end=len(train)-1)
                train_rmse_ct[nct] = sqrt(mean_squared_error(train[k:], train_predictions))
                test_predictions[nct] = auto_model.predict(start=len(train), end=len(train)+len(test)-1)
                test_rmse_ct[nct] = sqrt(mean_squared_error(test, test_predictions[nct]))

            test_auto_rmse = np.mean(test_rmse_ct)
            train_auto_rmse = np.mean(train_rmse_ct)
            auto_predictions = test_predictions.T

            final_predictions = parallel_model.predict(auto_predictions)
            test_exp_dataset = exp_dataset[train_day:]
            final_rmse = sqrt(mean_squared_error(test_exp_dataset, final_predictions))
            baseline = sqrt(mean_squared_error(test_exp_dataset, np.zeros(test_day)))


            # print("AutoReg train Process  mean_rmse is %.3f"%train_auto_rmse)
            # print("AutoReg test Process(validation process)  mean_rmse is %.3f"%test_auto_rmse)
            print("N_neighbor= %d, Final test rmse is %.3f"%(n_neigh,final_rmse))
            # print("Final test percentage error is %.3f%%"%(final_rmse/baseline*100))
```

```
            neigh_final_rmse[n]=final_rmse
            # edit to report training phase error or test phase error
            #report_rmse[n]=train_parallel_rmse[n]
            report_rmse[n]=neigh_final_rmse[n]
        plt.plot(np.arange(1,n_neigh_max+1),report_rmse[1:n_neigh_max+1],marker="*")
        #plt.plot(np.arange(1,n_neigh_max+1),neigh_final_rmse[1:n_neigh_max+1],marker="*")
        #plt.plot(n_neigh_distances)
        plt.xlabel("N_neighbors")
        plt.ylabel("Final rmse")

        plotname="Find_N_plot_d"+str(test_day)+"_"+exp_ct_name+".pdf"
        fname = os.path.join("..", "figs", plotname)
        plt.savefig(fname)
        plt.clf()
        plt.close('all')


    if question == "AMLR_future":
        warnings.filterwarnings("ignore")
        test_par = io_args.test_par
        test_day = io_args.test_day
        test_num = io_args.test_num


        dataset = read_dataset("phase1_training_data.csv")
        dataset = utils.get_all(dataset)
        n_ct, ct_key, ct_mapper, ct_inv_mapper, ct_ind =utils.make_mapper(dataset)

        dataset_CA=dataset[dataset[ct_key]=='CA']
        cases_CA = dataset_CA["cases"]
        cases14_CA =dataset_CA["cases_14_100k"]
        day_n = len(cases_CA)

        exp_fea_name = "deaths_100k"
        ref_fea_name = "deaths_100k"

        exp_ct_name ='CA'
        #ref_ct_name ='UK'

        #Find nearest neighbors of CA,return indices, get a combined all neighbor country dataset
        n_neigh = 100
        cutoff = 0.9
        if test_num != None:
            test_num = int(test_num)
            n_neigh = test_num
        if test_par != None:
            test_par = float(test_par)
            cutoff = test_par
        if test_day != None:
            test_day = int(test_day)

        train_day = day_n

        n_neigh, n_neigh_indices, n_neigh_distances,n_neigh_lags, n_neigh_dict, n_neigh_dict_lags = country.select_country(
            dataset,exp_fea_name,exp_ct_name,n_neigh=n_neigh,
            method="cross",cutoff=cutoff,R=30,ref_fea_name=ref_fea_name)
        print("y is '%s' from country '%s'"%(exp_fea_name,exp_ct_name))
        print("N_neighbor = %d with cutoff = %.3f"%(n_neigh,cutoff))
        #print(n_neigh_dict)

        # ---------------------------------------------------------------------------------------------------
```

```
    #   Combine the lags list and modify each country, return a dataset of reference
    #   the reference is used to be a X input matrix
    #   all subset of dataset are shifted based on the best lag according to the cross correlation
    #   we are fitting all the correlated countries with the same time-shift
    #   so that the total linear correlation coefficient is significantly improved
    #   we just considered the case that all countries don't have the same date of outbreak of pandemic
    #   this is an important factor to fit our data
    # ---------------------------------------------------------------------------------------------------
    ref_dataset=country.crosscombine_ct(dataset,ref_fea_name,n_neigh_indices,n_neigh_lags)
    exp_dataset= dataset[dataset[ct_key]==exp_ct_name][exp_fea_name]
    exp_dataset=exp_dataset.reset_index(drop = True)




    X=ref_dataset[:train_day]
    y=utils.s2v(exp_dataset[:train_day])

    AR_X=exp_dataset[:train_day]
    AR_y=exp_dataset[train_day:]

    #k is the lag chosen as parameter in AutoReg model
    #optimal lags is 2 as simplist optimal lag
    k=2
    #Implement auto regression on the self feature of the selected country

    AR_model = AutoReg1(lags=k)
    AR_model.fit(AR_X)
    AR_train_predictions = AR_model.predict(start=k,end=train_day-1)
    AR_test_predictions = AR_model.predict(start=train_day, end=train_day+test_day-1)
    AR_train_rmse = sqrt(mean_squared_error(AR_X[k:], AR_train_predictions))

    # ----------------------------------------------------------------------------------------------------
    #Implement the parallel Linear Regression on the neighbors of selected country:
    #
    #   Feature(selected country,past)= LR (Feature(other countries,past) )
    #
    #So we can train a model to fit the past feature of selected country
    #Later, we will use this model to parallelly predict the Future Feature of the spec country:
    #
    #   Feature(selected country,future)= a*LR (Feature(other countries,future) ) + b*AR (Feature(spec country,past)) + c
    #
    #Note that we have already shifted the training set so that all outbreaks happend at the same time
    #Based on this we are able to say, the correlation decides whether we can predict the spec country
    # -----------------------------------------------------------------------------------------------------
    parallel_model = linear_model.LeastSquaresPoly(p=1)
    parallel_model.fit(X,y)
    parallel_predictions=parallel_model.predict(X)
    train_parallel_rmse = sqrt(mean_squared_error(parallel_predictions,y))
    print("MultiLinear train Process rmse is %.3f"%train_parallel_rmse)

```

```python
        #Implement auto_regression for all neighbors for feature "cases_100k" except selected country

        train_rmse_ct = np.zeros(n_neigh)
        test_predictions = np.zeros((n_neigh,test_day))
        for nct in range(n_neigh):
            dataset_ct = dataset[dataset[ct_key]==ct_inv_mapper[n_neigh_indices[nct]]]
            X = dataset_ct[ref_fea_name]
            X = X.reset_index(drop = True)

            auto_model=AutoReg1(lags=k)
            auto_model.fit(X)
            train_predictions = auto_model.predict(start=k,end=train_day-1)

            train_rmse_ct[nct] = sqrt(mean_squared_error(X[k:], train_predictions))
            test_predictions[nct] = auto_model.predict(start=train_day, end=train_day+test_day-1)


        #Compute all errors happend in the AutoReg process
        #Add up and get the mean value of rmse of each phase of the regression process of each country:

        train_auto_rmse = np.mean(train_rmse_ct)

        #This is the matrix of the auto predicted value of all other countries
        #This can be considered as an X of a LR model, then we can predict the future feature of spec country
        auto_predictions = test_predictions.T

        #Prediction phase and calculate the final result of the selected feature of the spec country
        MLR_predictions = parallel_model.predict(auto_predictions)


#------------------Output Phase ------------------------------------------------------------------------------------------

        print("AutoReg train Process  mean_rmse is %.3f"%train_auto_rmse)

#------------------deaths feature ----------------------------------------------------------------------------------------

        if exp_fea_name == ref_fea_name and exp_fea_name =="deaths_100k":
            num_people_100k = utils.get_num_people_100k(dataset,exp_ct_name)

            MLR_deaths_predictions = num_people_100k * MLR_predictions

            AR_deaths_predictions = num_people_100k * AR_test_predictions

            # predict the deaths by combining two models together : stacked accroding to the correlation coefficients

            AMLR_deaths_predictions = utils.estimator(MLR_deaths_predictions.T[0], AR_deaths_predictions,n_neigh_distances)
            with open(os.path.join("..", "data", "sample_phase1_submission_origin.csv"), "rb") as f:
                sample_submission_dataset=pd.read_csv(f)
                sample_submission_dataset["deaths"]=AMLR_deaths_predictions
            sample_submission_dataset.to_csv(r'..\data\sample_phase1_submission.csv', index=False, header=True)
                #TrainingSet.to_csv(r'.\data\Modified_data.csv', index=False, header=True)
```

```
            # y_test = np.asarray(y_test)
            # y_pred = AMLR_deaths_predictions
            # utils.predictions_plot(y_pred,y_test,"CA","deaths","Test")



#-------------------AMLR model training and prediction-----------------Completed
    --------------------------------------------------------------
```

```
import numpy as np
from linear_model import LeastSquares


class My_AutoReg:
    def __init__(self,lag):
        self.lag = lag

    def fit(self,X):
        self.X = self.BuildX(X,self.lag)
        self.y = self.Buildy(X,self.lag)

        model = LeastSquares()
        #print(self.X.shape)
        #print(self.y.shape)
        model.fit(self.X,self.y)
        self.my_model = model



    def BuildX(self,X,lag):
        if type(self.lag) == type(1):
            lag = np.arange(1,self.lag+1)

        length = len(X)
        A = np.empty((0,len(lag)+1),dtype=np.int)
        for i in range(length-lag[-1]):
            xi = np.ones(len(lag)+1,dtype=np.int)

            for j,term in enumerate(lag):
                xi[j+1] = X[i+term-1]
            A = np.vstack([A, xi])
        #print(A)

        return A

    def Buildy(self,X,lag):
        if type(self.lag) == type(1):
            lag = np.arange(1,self.lag+1)

        maxlag = lag[-1]
        y=np.empty(int(0),dtype=np.int)
        length = len(X)
        for i in range(length-lag[-1]):
            y = np.append(y, X[i+lag[-1]])
        #print(y)
        return y

    def predict(self, Z, start = None, end = None):
        n_time_topre = end - start + 1
        start = start - self.lag
        model = self.my_model
        w = model.w
        y_pred = [0]*n_time_topre
        latest_X = self.X[start-1]
        y_pred[0] = np.dot(latest_X,w)
```

```python
        i=1
        while(i<n_time_topre):
            if(start + i >= len(Z)):
                latest_X = np.roll(latest_X,-1)
                latest_X[0] = 1
                latest_X[-1] = y_pred[i-1]
                y_pred[i] = model.predict(latest_X)
                i += 1
            else:
                latest_X = self.X[start+i-1]
                y_pred[i] = model.predict(latest_X)
                i += 1
        return y_pred

class AutoReg1():
    def __init__(self,lags):
        self.lags = lags

    def fit(self,y):
        self.y = np.zeros(len(y)) #train set
        self.y = y
        self.X = self.BuildX() #matrix constructed from train set by shifting
        model = LeastSquares()
        #print(self.X.shape)
        #print(self.y.shape)
        model.fit(self.X,self.y[self.lags:].T)
        self.w = model.w
        #print(self.w)
        # self.my_model = model

    def BuildX(self):
        k = self.lags
        y = self.y
        T = len(y)
        X = np.ones((T-k,k+1))
        for t in range(k,T):
            X[t-k][1:] = y[t-k:t]
        #print(X)
        #print(y)
        return X

    def predict(self,start,end):
        y=self.y
        k=self.lags
        T= len(y)
        y_pred = np.zeros(end-start+1)

        if start <= T:
            ylag = y[start-k:start]
        else:
            ylag = y[T-k:]
            #predict from y[T] for each y[t] up to y[start-1]
            for t in range(T,start):
                y_pred1 = ylag@self.w[1:]+self.w[0]
                ylag = np.roll(ylag,-1)
                ylag[-1] =y_pred1
        #scroll data into y_pred
        for t in range(start,end+1):
            y_pred[t-start] = ylag@self.w[1:]+self.w[0]
            ylag = np.roll(ylag,-1)
            ylag[-1] =y_pred[t-start]
        return y_pred
```

```python
import numpy as np
import utils
```

```python
from scipy.stats import pearsonr
import pandas as pd

def select_country(dataset,exp_fea_name,exp_ct_name,ref_fea_name=None,n_neigh=10,method="
    corr",cutoff=0.5,itself=False,R=30):
    n_ct, ct_key, ct_mapper, ct_inv_mapper, ct_ind =utils.make_mapper(dataset)
    nan_num = 0
    if method == "corr" or "cross":
        day_n = len(dataset[dataset[ct_key]=='CA']["cases"])
        if ref_fea_name == None:
            ref_fea_name = exp_fea_name

        exp_dataset = np.zeros((n_ct,day_n))

        #Choose the dataset regarding the name interested
        target= np.zeros((1,day_n))
        target[0] = dataset[dataset[ct_key]==exp_ct_name][exp_fea_name]
        corr=np.zeros(n_ct)
        lags=np.zeros(n_ct)
        indices=np.zeros(n_ct)
        if method == "corr":
            corr_pair = np.zeros((2,n_ct))
        elif method == "cross":
            corr_pair = np.zeros((3,n_ct))



        #Construct Experiment Dataset (Matrix) =feature(country,day)
        for nct in range(nan_num,n_ct):  #fillter out nan set
            exp_dataset[nct] = dataset[dataset[ct_key]==ct_inv_mapper[nct]][ref_fea_name]

        #Compute Corr between N_CT countries
        if method == "corr":
            for nct in range(nan_num,n_ct):  #fillter out nan set
                corr[nct]=pearsonr(target[0],exp_dataset[nct])[0]
                indices[nct]=nct
        elif method == "cross":
            for nct in range(nan_num,n_ct):  #fillter out nan set
                series1=pd.Series(exp_dataset[nct])
                series2=pd.Series(target[0])
                corr[nct],lags[nct]=cross_corr(series1,series2,R)
                indices[nct]=nct

        corr_pair[0] = indices
        corr_pair[1] = corr
        if method == "cross":
            corr_pair[2] = lags

        #remove the nan set
        #sort the corr_pair
        corr_pair1 = corr_pair.T[1:]
        corr_pair= corr_pair1[np.argsort(corr_pair1[:,1],axis =0)]
        corr_pair= corr_pair[::-1]

        #remove itself from the correlation list
        if itself == False:
            corr_pair = corr_pair[1:]
        #choose the neigh distance vector
        corr_n_neigh =corr_pair[:,1][:n_neigh]
        n_neigh_distances = corr_n_neigh

        if cutoff != None:
            n_neigh_distances =n_neigh_distances[n_neigh_distances >= cutoff]

        #update n_neigh after cutoff
```

```
        n_neigh = len(n_neigh_distances)
        n_neigh_indices = corr_pair[:,0][:n_neigh]
        if method == "cross":
            n_neigh_lags = corr_pair[:,2][:n_neigh]

        #create dict for n neigh countries with descent order of distances
        ct_neigh = [ct_inv_mapper[n_neigh_indices[i]] for i in range(n_neigh)]
        n_neigh_dict=dict(zip(ct_neigh,list(corr_n_neigh)))
        n_neigh_dict_lags=dict(zip(ct_neigh,list(n_neigh_lags)))
        if method == "corr":
            return n_neigh, n_neigh_indices, n_neigh_distances, n_neigh_dict
        elif method =="cross":
            return n_neigh, n_neigh_indices, n_neigh_distances, n_neigh_lags, n_neigh_dict,
    n_neigh_dict_lags


def combine_ct(dataset,fea_name,ind_ct):
    n_ct, ct_key, ct_mapper, ct_inv_mapper, ct_ind =utils.make_mapper(dataset)
    day_n = len(dataset[dataset[ct_key]=='CA']["cases"])
    nmax = len(ind_ct)
    exp_dataset = np.zeros((nmax,day_n))
    exp_name = fea_name
    for n in range(nmax):
        exp_dataset[n] = dataset[dataset[ct_key]==ct_inv_mapper[ind_ct[n]]][exp_name]
    return exp_dataset.T #dim=(cases14,country) fea=country

def cross_corr(ref_s,exp_s,R):

    ref_s1 = ref_s.reset_index(drop = True)
    exp_s1 = exp_s.reset_index(drop = True)
    maxcorr =0.0
    lag_best = 0
    for lag in range(R):
        ref_s1 = ref_s1.shift(lag)
        ref_s1[:lag]=0.0
        corr= pearsonr(ref_s1, exp_s1)[0]
        if corr>maxcorr:
            maxcorr = corr
            lag_best = lag
    return maxcorr,lag_best


def crosscombine_ct(dataset,fea_name,ind_ct,lags):
    n_ct, ct_key, ct_mapper, ct_inv_mapper, ct_ind =utils.make_mapper(dataset)
    day_n = len(dataset[dataset[ct_key]=='CA']["cases"])
    nmax = len(ind_ct)
    exp_dataset = np.zeros((nmax,day_n))
    exp_name = fea_name
    for n in range(nmax):
        exp_dataset[n] = dataset[dataset[ct_key]==ct_inv_mapper[ind_ct[n]]][exp_name]
        exp_dataset[n] = utils.shift_fill0_array(exp_dataset[n],lags[n])
    return exp_dataset.T #dim=(cases14,country) fea=country
```

```
import pickle
import os
import sys
import numpy as np
from scipy.optimize import approx_fprime
import matplotlib.pyplot as plt
import pandas as pd
from scipy.sparse import csr_matrix as sparse_matrix
from sklearn.metrics import mean_squared_error
import datetime
import math

```

```python
def create_user_item_matrix(ratings,user_key="user",item_key="item"):

    n = len(set(ratings[user_key]))
    d = len(set(ratings[item_key]))

    user_mapper = dict(zip(np.unique(ratings[user_key]), list(range(n))))
    item_mapper = dict(zip(np.unique(ratings[item_key]), list(range(d))))

    user_inverse_mapper = dict(zip(list(range(n)), np.unique(ratings[user_key])))
    item_inverse_mapper = dict(zip(list(range(d)), np.unique(ratings[item_key])))

    user_ind = [user_mapper[i] for i in ratings[user_key]]
    item_ind = [item_mapper[i] for i in ratings[item_key]]

    X = sparse_matrix((ratings["rating"], (user_ind, item_ind)), shape=(n,d))

    return X, user_mapper, item_mapper, user_inverse_mapper, item_inverse_mapper, user_ind,
    item_ind

def standardize_cols(X, mu=None, sigma=None):
    # Standardize each column with mean 0 and variance 1
    n_rows, n_cols = X.shape

    if mu is None:
        mu = np.mean(X, axis=0)

    if sigma is None:
        sigma = np.std(X, axis=0)
        sigma[sigma < 1e-8] = 1.

    return (X - mu) / sigma, mu, sigma


def check_gradient(model, X, y):
    # This checks that the gradient implementation is correct
    w = np.random.rand(model.w.size)
    f, g = model.funObj(w, X, y)

    # Check the gradient
    estimated_gradient = approx_fprime(w,
                                       lambda w: model.funObj(w,X,y)[0],
                                       epsilon=1e-6)

    implemented_gradient = model.funObj(w, X, y)[1]

    if np.max(np.abs(estimated_gradient - implemented_gradient) > 1e-4):
        raise Exception('User and numerical derivatives differ:\n%s\n%s' %
             (estimated_gradient[:5], implemented_gradient[:5]))
    else:
        print('User and numerical derivatives agree.')

def classification_error(y, yhat):
    return np.mean(y!=yhat)


def test_and_plot(model,X,y,Xtest=None,ytest=None,title=None,filename=None):

    # Compute training error
    yhat = model.predict(X)
    #trainError = np.mean((yhat - y)**2)
    trainError = mean_squared_error(y,yhat)
    print("Training mse = %.10f" % trainError)

    # Compute test error
    if Xtest is not None and ytest is not None:
```

```
        yhat = model.predict(Xtest)
        # testError = np.mean((yhat - ytest)**2)
        testError = mean_squared_error(ytest,yhat)
        print("Test mse    = %.10f" % testError)

    # Plot model
    plt.figure()
    plt.plot(X,y,'b.')

    # Choose points to evaluate the function
    if(len(X[0])==1):
        Xgrid = np.linspace(np.min(X),np.max(X),1000)[:,None]
        ygrid = model.predict(Xgrid)
        plt.plot(Xgrid, ygrid, 'g')

        if title is not None:
            plt.title(title)

        if filename is not None:
            filename = os.path.join("..", "figs", filename)
            print("Saving", filename)
            plt.savefig(filename)

def linear_model_test(model,X,y,Xtest=None,ytest=None):

    # Compute training error
    yhat = model.predict(X)
    #trainError = np.mean((yhat - y)**2)
    trainError = mean_squared_error(y,yhat)
    print("Training mse = %.10f" % trainError)

    # Compute test error
    if Xtest is not None and ytest is not None:
        yhat = model.predict(Xtest)
        # testError = np.mean((yhat - ytest)**2)
        testError = mean_squared_error(ytest,yhat)
        print("Test mse    = %.10f" % testError)

def linear_model_test_with_plot(model,X,y,Xtest=None,ytest=None):

    # Compute training error
    yhat = model.predict(X)
    #trainError = np.mean((yhat - y)**2)
    trainError = mean_squared_error(y,yhat)
    print("Training mse = %.10f" % trainError)

    # Compute test error
    if Xtest is not None and ytest is not None:
        yhat = model.predict(Xtest)
        # testError = np.mean((yhat - ytest)**2)
        testError = mean_squared_error(ytest,yhat)
        print("Test mse    = %.10f" % testError)

def predictions_plot(y_pred,y_real,exp_ct_name,exp_fea_name,phase,n_neigh=None):
    rmse = np.sqrt(mean_squared_error(y_pred,y_real))
    l=len(y_pred)
    plt.plot(np.arange(l),y_pred,marker="*",label="pred")
    plt.plot(np.arange(l),y_real,marker=".",label="real")
    plt.xlabel("Date")
    plt.ylabel("Deaths")
    title="Pred_"+exp_fea_name+"_rmse is "+str(rmse)
    plt.title(title)
    plt.legend()
    plotname="Pred_"+phase+"_plot_"+exp_fea_name+"_d"+str(l)+"_"+exp_ct_name+".pdf"
    if n_neigh !=None:
```

```
        plotname="Pred_"+phase+"_plot_"+exp_fea_name+"_d"+str(l)+"_n"+str(n_neigh)+"_"+exp_ct_name+".pdf"
    fname = os.path.join("..", "figs", plotname)
    if fname is not None:
        fname = os.path.join("..", "figs", fname)
        print("Saving", fname)
    plt.savefig(fname)
    plt.clf()
    plt.close('all')



def dif_series(x,day,refine=None):
    y=x-x.shift(day)
    if refine == 1:
        y=y[day:]
    return y

def s2m(series):
    X = np.zeros((1,len(series)))
    X[0] = series
    return X

def s2v(series):
    return s2m(series).T

def make_mapper(dataset):

    ct_key="country_id"
    dataset_ct = dataset[ct_key]
    n_ct=len(set(dataset_ct))
    ct_mapper = dict(zip(list(set(dataset_ct)),list(range(n_ct))))
    ct_inv_mapper = dict(zip(list(range(n_ct)),list(set(dataset_ct))))
    ct_ind = [ct_mapper[i] for i in dataset_ct]
    return n_ct, ct_key, ct_mapper, ct_inv_mapper, ct_ind


def shift_fill0(series,lag):
    series=series.shift(lag)
    series[:lag]=0.0
    return series

def shift_fill0_array(array,lag):
    int_lag =int(lag)
    array=np.roll(array,int_lag)
    array[:int_lag]=0.0
    return array
# from original dataset generate the death_100k column and return the new dataset
def get_death_100k(dataset):
    temp = dataset["cases"]
    temp[temp == 0] = 1
    deaths_100k_dataset=dataset["deaths"]*dataset["cases_100k"]/temp
    #print(deaths_100k_dataset)
    dataset["deaths_100k"]=pd.Series(deaths_100k_dataset, index=dataset.index)
    return dataset


def test_death_100k(dataset,n):
    deaths=dataset["deaths"]
    cases=dataset["cases"]
    cases_100k=dataset["cases_100k"]
    cases[cases==0]=1
    return deaths[n]*cases_100k[n]/cases[n]

def get_num_people_100k(dataset,ct_name):
```

```
    #choose the newest data to compute population
    #Here's some assumption that population wont change during the whole process
    n_row=dataset[dataset["country_id"]==ct_name]["deaths"].index[-1]
    cases_100k = dataset["cases_100k"]
    cases_100k[cases_100k ==0] =1
    num_people_100k = dataset["cases"][n_row]/cases_100k[n_row]
    return num_people_100k

def get_all(dataset):
    n_ct, ct_key, ct_mapper, ct_inv_mapper, ct_ind = make_mapper(dataset)
    dataset=get_death_100k(dataset)
    dataset["daily_cases"] = 0.0
    dataset["daily_deaths"] = 0.0
    dataset["daily_deaths_100k"] = 0.0
    for nct in range(1,n_ct):
        dataset1=dataset[dataset[ct_key]==ct_inv_mapper[nct]]
        l=len(dataset1)
        daily_cases=np.zeros(l)
        daily_deaths=np.zeros(l)
        daily_deaths_100k=np.zeros(l)
        for i in range(1,l):
            daily_deaths[i]= dataset1.iloc[i,3] - dataset1.iloc[i-1,3]
            daily_cases[i] = dataset1.iloc[i,2] - dataset1.iloc[i-1,2]
            daily_deaths_100k[i] =dataset1.iloc[i,6] - dataset1.iloc[i-1,6]
        dataset1["daily_cases"]=daily_cases
        dataset1["daily_deaths"]=daily_deaths
        dataset1["daily_deaths_100k"]=daily_deaths_100k
        dataset[dataset[ct_key]==ct_inv_mapper[nct]] = dataset1
    return dataset
#Estimator to average the value between ensamble predictions from other coutries and auto regression
def estimator(MLR_pred,self_pred,distances,method="cross"):
    if type(self_pred) == pd.core.series.Series:
        self_pred=self_pred.reset_index(drop = True)
    if type(MLR_pred) == pd.core.series.Series:
        MLR_pred=MLR_pred.reset_index(drop = True)

    if method == "cross":
        #The maximum of weight depends on the mean correlation of n_neighbor countreis
        #The n_neighbors are selected strictly to provide long term information to the prediction
        #Especially those countries with earlier outbreaks
        weight_corr = np.mean(distances)
        weight_self = 1.0
        w_max = weight_corr
        w_min = 0.0
        #The weight of the MLR part is made from calculating the mean correlations
        #Between spec country
        #They are selected N_neighbor countries
        #so they have higher weight if they are more similar to spec coutnrey
        predictions=np.zeros(len(MLR_pred))
        for day in range(len(MLR_pred)):
            #The error of auto regression increase as time increases
            #The curve will tend to go to the averaged predictions made by other countries
            #This is an linear scaling over the weights of other countries according to time
            weight_MLR=w_function(day,len(MLR_pred),w_min,w_max)
            w_MLR = weight_MLR / (weight_corr + weight_self)
            predictions[day] = w_MLR * MLR_pred[day] + (1-w_MLR) * self_pred[day]
        return predictions
    else:
        print("Error: Method not defined! ")

def w_function(x,length,w_min,w_max):
    return w_min+(w_max-w_min)/length*x

```

```python


# def read_data(filename):
#     with open(os.path.join("data", filename), "rb") as f:
#         OriginDataSet = pd.read_csv(f,keep_default_na=False, na_values='_')
#     TrainingSet = OriginDataSet.copy()
#     data = TrainingSet["date"]
#     print(data)
#     refer_date = datetime.datetime.timestamp(datetime.datetime.strptime(data[0], "%m/%d/%Y"))
#     for i in range(len(data)):
#         TrainingSet["date"][i] = math.ceil((datetime.datetime.timestamp(datetime.datetime.strptime(data[i], "%m/%d/%Y")) - refer_date) / 86400)
#     TrainingSet.to_csv(r'.\data\Modified_data.csv', index=False, header=True)
```

```python
import numpy as np
from numpy.linalg import solve
from findMin import findMin
from scipy.optimize import approx_fprime
import utils

# Ordinary Least Squares
class LeastSquares:
    def fit(self,X,y):
        self.w = solve(X.T@X, X.T@y)

    def predict(self, X):
        return X@self.w

# Least squares where each sample point X has a weight associated with it.
class WeightedLeastSquares(LeastSquares): # inherits the predict() function from LeastSquares
    def fit(self,X,y,z):
        ''' YOUR CODE HERE '''
        n, d = X.shape
        V = np.diag(z)
        self.w = solve(X.T@V@X, X.T@V@y)

class LinearModelGradient(LeastSquares):

    def fit(self,X,y):
        n, d = X.shape

        # Initial guess
        self.w = np.zeros((d, 1))

        # check the gradient
        estimated_gradient = approx_fprime(self.w.T[0], lambda w: self.funObj(w,X,y)[0], epsilon=1e-6)
        implemented_gradient = self.funObj(self.w,X,y)[1]
        if np.max(np.abs(estimated_gradient - implemented_gradient) > 1e-4):
            print('User and numerical derivatives differ: %s vs. %s' % (estimated_gradient, implemented_gradient));
        else:
            print('User and numerical derivatives agree.')

        self.w, f = findMin(self.funObj, self.w, 100, X, y)

    def funObj(self,w,X,y):
        n,d = X.shape
        ''' MODIFY THIS CODE '''
        # Calculate the function value
        #f = 0.5*np.sum((X@w - y)**2)
        f=0
        g=np.zeros((d,1))
```

```
        for i in range(n):
            ri = w.T@X[i]-y[i]
            f += np.log(2*np.cosh(ri))
            g += X[i]*np.tanh(ri)
        # Calculate the gradient value
        #g = X.T@(X@w-y)


        return (f,g)


# Least Squares with a bias added
class LeastSquaresBias:

    def fit(self,X,y):
        n, d = X.shape
        Z=np.hstack((np.ones((n,1)),X))

        self.v = solve(Z.T@Z, Z.T@y)
        self.w = self.v[1:]

    def predict(self, X):
        n, d = X.shape

        return X@self.w + np.ones((n,1))*self.v[0]

# Least Squares with polynomial basis
class LeastSquaresPoly:
    def __init__(self, p):
        self.leastSquares = LeastSquares()
        self.p = p

    def fit(self,X,y):

        Z=self.__polyBasis(X)
        self.v = solve(Z.T@Z, Z.T@y)
        #self.w = self.v[1:]

    def predict(self, X):

        Z=self.__polyBasis(X)
        return Z@self.v

    # A private helper function to transform any matrix X into
    # the polynomial basis defined by this class at initialization
    # Returns the matrix Z that is the polynomial basis of X.
    def __polyBasis(self, X):
        ''' YOUR CODE HERE '''
        n, d = X.shape
        Z=np.ones((n,1))
        if self.p !=0:
            for k in range(1,self.p+1):
                Z=np.hstack((Z,X**k))
        return Z
```

```
import numpy as np
from numpy.linalg import norm

def findMin(funObj, w, maxEvals, *args, verbose=0):
    """
    Uses gradient descent to optimize the objective function

    This uses quadratic interpolation in its line search to
    determine the step size alpha
    """
```

```python
    # Parameters of the Optimization
    optTol = 1e-2
    gamma = 1e-4

    # Evaluate the initial function value and gradient
    f, g = funObj(w,*args)
    funEvals = 1

    alpha = 1.
    while True:
        # Line-search using quadratic interpolation to find an acceptable value of alpha
        gg = g.T@g

        while True:
            w_new = w - alpha * g
            f_new, g_new = funObj(w_new, *args)

            funEvals += 1

            if f_new <= f - gamma * alpha*gg:
                break

            if verbose > 1:
                print("f_new: %.3f - f: %.3f - Backtracking..." % (f_new, f))

            # Update step size alpha
            alpha = (alpha**2) * gg/(2.*(f_new - f + alpha*gg))

        # Print progress
        if verbose > 0:
            print("%d - loss: %.3f" % (funEvals, f_new))

        # Update step-size for next iteration
        y = g_new - g
        alpha = -alpha*(y.T@g) / (y.T@y)

        # Safety guards
        if np.isnan(alpha) or alpha < 1e-10 or alpha > 1e10:
            alpha = 1.

        if verbose > 1:
            print("alpha: %.3f" % (alpha))

        # Update parameters/function/gradient
        w = w_new
        f = f_new
        g = g_new

        # Test termination conditions
        optCond = norm(g, float('inf'))

        if optCond < optTol:
            if verbose:
                print("Problem solved up to optimality tolerance %.3f" % optTol)
            break

        if funEvals >= maxEvals:
            if verbose:
                print("Reached maximum number of function evaluations %d" % maxEvals)
            break

    return w, f
```