CPSC 340 Machine Learning Take-Home Midterm Exam (Fall 2020)

Question 2

1 Team

Team Members	*Ali Seyfi (aliseyfi), Mohammad Amin Mohammadi(lemohama), Farnoosh Hashemi (Farsh)*
Kaggle Team Name	*Farnoosh didn't let me in*

2 Solution Summary

In order to solve the problem, we first go through the data set and plot different figures to understand the existing features and their trends and correlation with each other. We find similar countries to Canada, by computing correlation between their features and also applying the KNN algorithm on the data set.

Then we start to think about how we can use the data from other countries to help us predict the deaths in Canada. Although it seems that using other countries' data will help us to predict Canada's deaths' trend - as they may have a similar trend to Canada and are prior in time, we figured out that it is not helpful that much. In the next section, we will discuss the pros and cons of using other countries' data.

We then used an Autoregressive model as described in the assignment description. In order to get the best result, we tried different methods, like VAR, and features, like cases, deaths, daily death, and daily cases. Due to our expectations, by having daily death we will have better results, but it showed up that because of the daily noise, the accuracy will fall dramatically. We will discuss more in the next part.

Among all these methods and features, by performing a Cross-Validation method and looking at the validation and test errors, and search among our feature space, we decide to pick multi-variable autoregression models that predict deaths of a certain day using deaths of n past days, and cases of m past days where n and m come from minimizing validation error.

The final approach is that using the validation data, we can find the hyper-parameters that give us the minimum validation error. Then we store all models that have a validation error less than 5. After that, we fit all these models with all of our data set and report the median of their predictions.

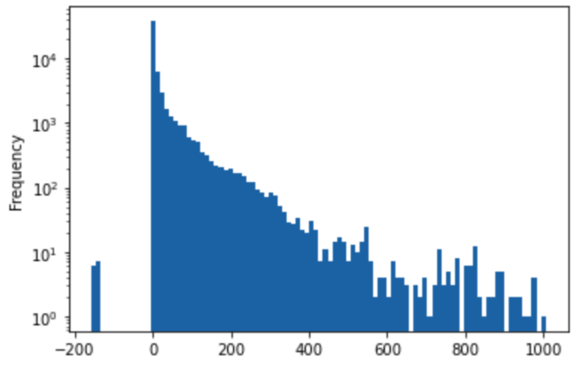
The reason that we use the median of predictions is that we wanted to have a meta-regressor that fits with the data of models' predictions. But as the system of our equations is under-determined, we can not use a linear regression model to fit the first stage predictions. So in order to use the results of our models, one thing we can do is to average their results. But as we learned in the class, the average is sensitive to outliers and if a model has a bad prediction (because of over-fitting), we will get a bad result, So we compute the median instead of the mean.

By using this approach, we get a validation error of 6.2 and a test error of 12. In the following pages, we discussed more different methods we used and also our feature and hyper-parameter selection approach. The code of the class is enclosed in appendix.

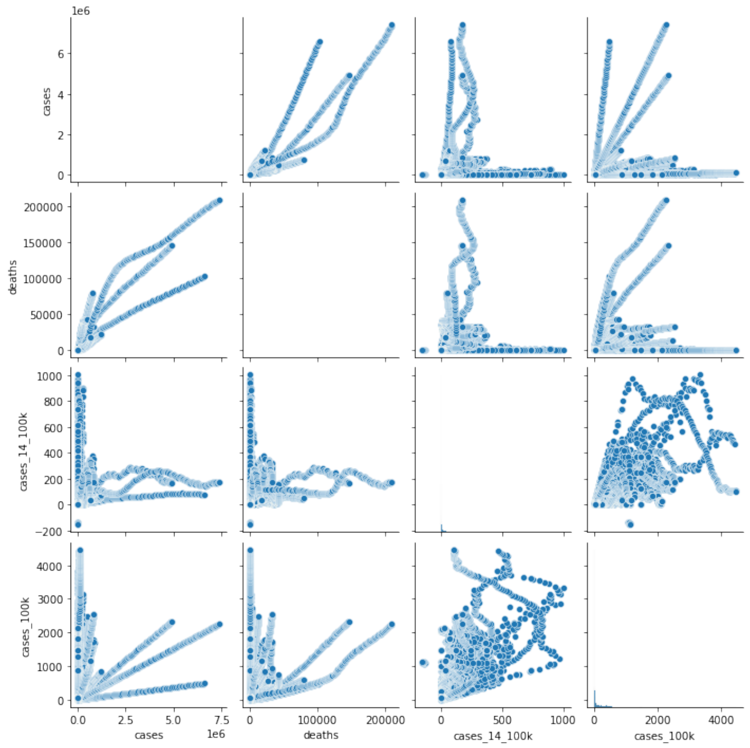
3 Experiments

3.1 Exploratory Data Analysis

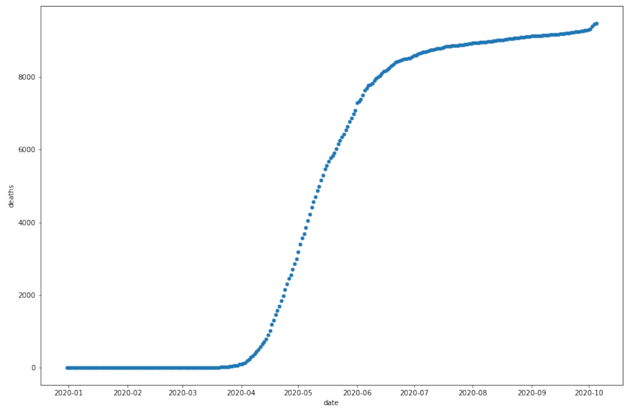
First of all, we went through some general exploratory data analysis in the training data set, to see what we can find and grasp a general understanding of the data. Going through the data, we found some outliers, which had negative 'cases_14_100k' values, that are visible in the plot below. This is the histogram of 'cases_14_100k' values for all countries where the y-axis is scaled to the log scale.

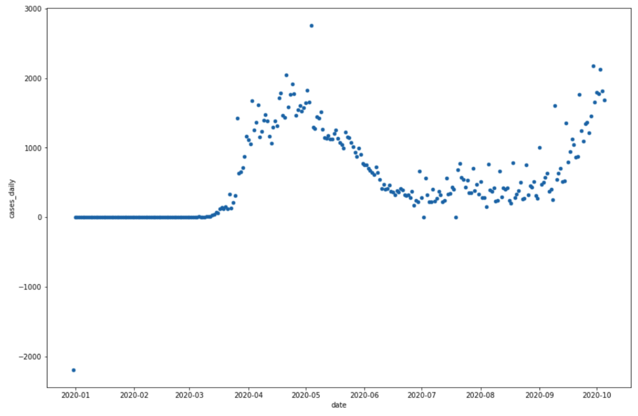


Moreover, we found a country that didn't have a country id and the id was missing, so we filled its 'country_id' value as unknown, to remove any kind of missing data in our training data set. A natural task that comes next, was to plot the variables of the data set against each other after sorting the data set based on '(country_id, time)' to be able to recognize the data better. Through this, we came to the conclusion that the data of the 'cases', 'deaths' and 'cases_100k' columns were cumulative.



Next, we moved on to explore the data of the specific country, "Canada", further. Plotting the diagrams of 'deaths', 'cases' and 'daily_cases' (which can be retrieved from calculating the difference of consecutive 'cases' values), we understood that until a certain point in time, there has been no case or no death regarding the virus in the country.





Two thoughts that came into our minds here, just looking at these plots were that:

- We might want to not include the data in the model training phase from the beginning of the data set, since obviously there are some data without any patterns at the beginning.
- We might want to use the daily-transformation of some features in the data that are originally cumulative since they are more like normal distributions which lead to better results in linear regression.

We removed some of the first data from data sets since in the first days the number of deaths in the data set is 0.(More precisely, deaths for the first 70 days in Canada are 0). Moreover, the death trend for the appearance of Covid-19 is completely different from the Covid-19 prevalence.

Towards having a better understanding of the inherent patterns between cases and deaths in different countries, and trying to figure out if there are any other countries with patterns similar to Canada that actually make sense, we computed the correlation of 'daily_deaths' and 'daily_cases' between Canada and all other countries, from the point of time that there was at least one case in Canada. Using Pearson's correlation coefficient on the only countries with a value of more than 0.45 were "AE, DE, JP". Clearly, these countries are not that much related to Canada, in respect of travel patterns and virus distribution. Thus, this might suggest that we should not be using other countries in our final prediction model.

To predict the death s for Canada is not appropriate to consider all countries over the world. Since we want to predict over only one group of data. There are many different countries that are very different from Canada and they look like outliers. So it is reasonable that we consider countries that their trend is similar to Canada and exclude other countries. To find countries that are similar to Canada, we used spatial.KDTree(). At first, we group the data by countries then we built this tree with all countries except Canada, then this tree was queried by Canada and we could find that' 'DE' is the most similar country to Canada.

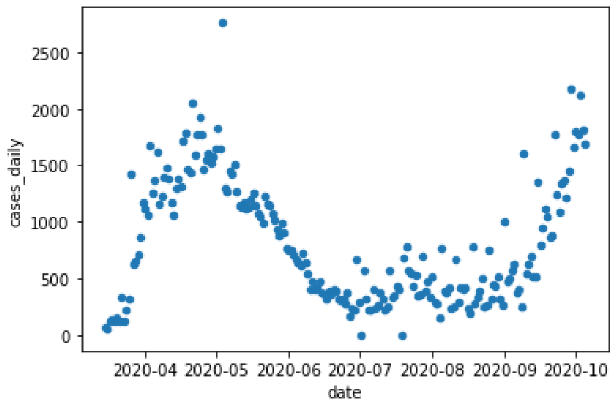
Moreover, we used the correlation between death per 100 between countries and we find that "CA and CM, DE, FI, HU, IE, JE, LT, LV, ML" have a correlation more than 0.90 and $p\_value < 0.05$.

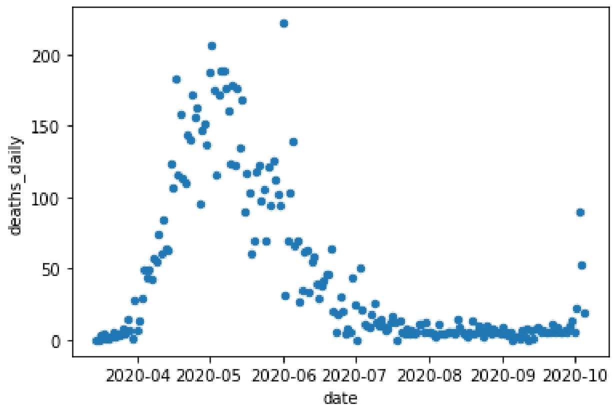
In addition to all this, death in countries related to many factors, for example, Hospital filling, sudden privilege due to frequent visits to annual celebrations, experimental drug testing on a group of people, etc. that are related to only that particular country. So due to this reason, we consider only Canada as a study country.

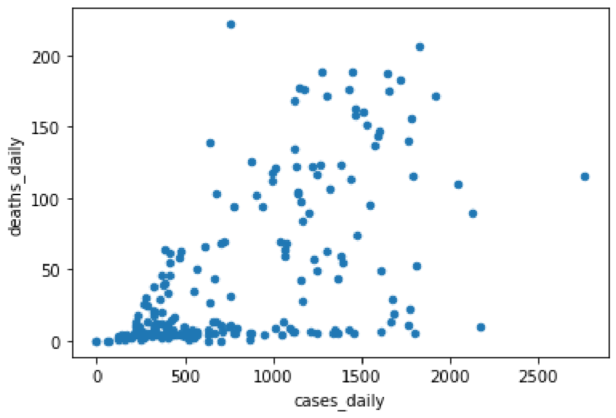
3.2 Training our first model

Moving onto the training phase, the first model that we tested on the data was the simple AutoRegressive model, which after a few tries of different parameter values, led us to a test error of ~ 60 in the phase 1 competition. First, we implemented to AutoRegression model ourselves using only one feature (deaths), and made a few tryouts using it on the phase 1 competition. Then, we compared our model to the 'statsmodels' package's 'AutoReg' model, to see if we are doing everything right in our implementation or not. Thankfully, using the same model parameters, we achieved the same results using both models, which was promising.

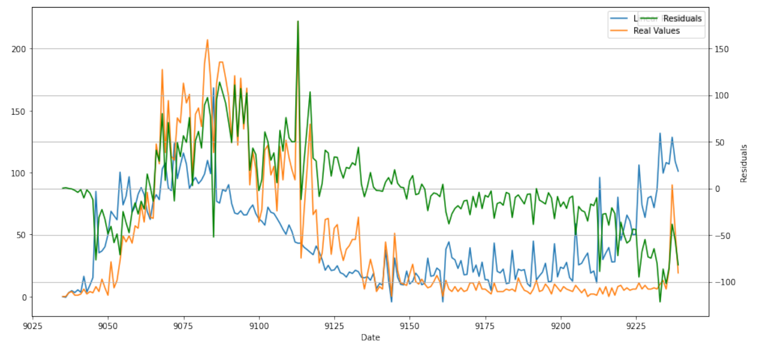
In order to utilize more than one feature in our model, we tried to dive deeper into understanding the relation between daily cases and daily deaths in Canada. Plotting both variables and then making a scatter plot of these two features against each other, suggested that there is almost no linear relation between these two.







Fitting a simple linear model on these data also resulted in the following residual vs data plot, which is clearly indicating that the residuals do not have the same mean, and probably, do not follow a normal distribution too.



Based on these results, we can confidently say that the normal (non-time series related) approach will most probably not be a good idea to try, and the mentioned features probably do not have any classic linear relation together. But the question of whether a time series model would capture any relation between different features remains unanswered.

To answer the previous question, we tried to use the Vector Autoregression model from the 'statsmodels' package to see if it helps in getting better results and then implementing it ourselves if so. After reading a few articles about this model, we came to the conclusion that if we're going to use VAR on some data, the data must have the following characteristics (In fact, these are the assumptions of VAR):

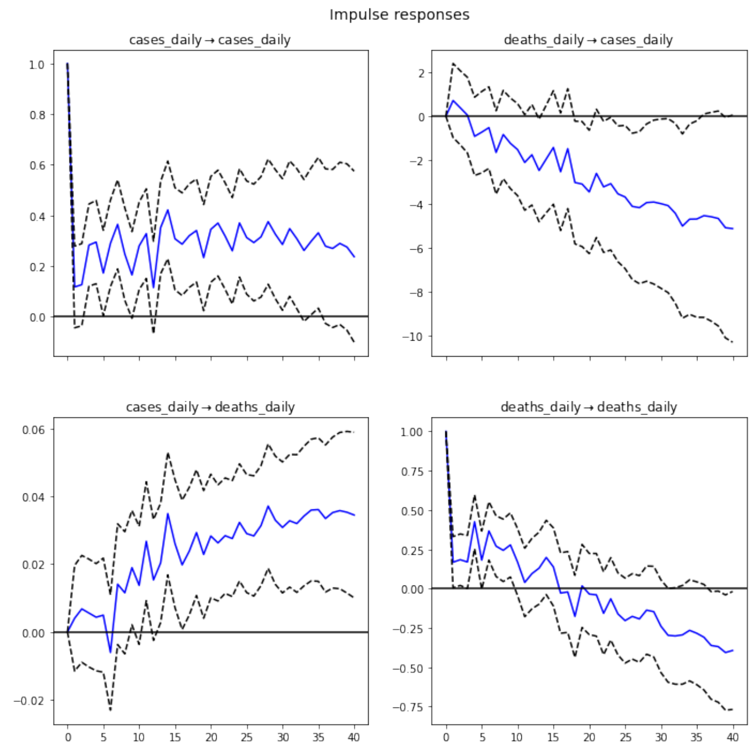
- The time series used as VAR input must affect each other in a bi-directional way.
- The timer series used in VAR must be stationary.

Using the 'adfuller' test, we understood that all of our variables (both cumulative and daily variants) are confidently (with a p-value of about zero) stationary. But the problem is, although we know that the 'cases' time series clearly affects the 'deaths', we are not sure if this fact holds true in reverse order, both in cumulative and daily variants. With this in mind, we tried to see if we could achieve any good result using VAR on (cases, deaths) for prediction of the following dates (Phase 1 competition).

In order to check for the best lag order values and proper lags that we should include in the model, we utilized the analysis provided by the 'statsmodels' package on the VAR model and selected the variables corresponding to the best AIC, BIC values regarding the model. Also, to sanity check our initial conditions and if VAR is usable on the input data or not, we checked if the final residuals follow a normal distribution and tested the normality of them too.

Unfortunately, fitting the VAR model on these two columns yielded pretty bad (extremely larger than what we expected) death results for the following 11 days, which was kind of a disappointment, but at the same time, kind of expected since we knew death time series might not affect case time series.

Moving on to using the daily features in the VAR model, we achieved way better results than using the cumulative features. We plotted the impulse response plots of the VAR model to check if what it has learned makes sense or not.



Although the test error of the VAR model on daily data was rather good (~ 30), as the impulse response plots suggest, the effect of 'deaths_daily' towards 'cases_daily' and also vice versa, is not as expected, which again is a point of doubt in the model. Normally, we would expect that as the cases count grow, the deaths grow too, until a point where the case has either passed out or is cured and the effect is gone (on average, almost 18 days), but the plot suggests that the effect is strictly ascending or descending even after 40 days of the case/death. Clearly, this model can not be trusted because of this, as the relations learned by it are not that true. According to these results and understandings, we gave up on VAR and tried to improve the normal autoregressive model and add a few extensions to it instead of using VAR.

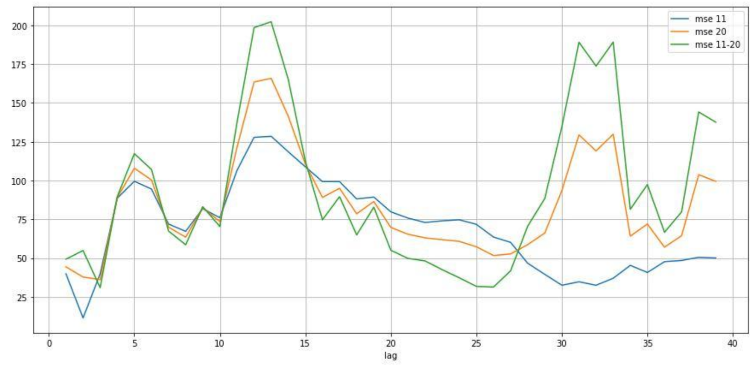
3.3 Optimizing the naive autoregressive model

Using the data released after phase 1 competition, we tried to optimize the simple AutoRegressive model with one feature (deaths) to see if it is possible to improve our best results or not. Using the following code (The class code is in the Appendix section):

```python
from Model.AutoRegressionModel import AutoRegressionModel as arm

errs = []
for lag in range(1, 40):
    model = arm(k=lag, bias=True)
    model.fit(deaths)
    predictions = pd.DataFrame({
        "deaths": validation_df_ca['deaths'].values,
        "deaths_pred": model.predict(205, 205+10)
    })
    err1 = np.sqrt(np.mean((predictions['deaths'] - predictions['deaths_pred']).values ** 2)
    )
    err2 = np.sqrt(np.mean((res.predict(205, 205+19) - second_val_df['deaths'].values) ** 2)
    )
    err3 = np.sqrt(np.mean((res.predict(205+11, 205+19) - second_val_df['deaths'].values
    [-9:]) ** 2))
    errs.append((lag, err1, err2, err3))

pd.DataFrame(errs, columns=['lag', 'mse 11', 'mse 20', 'mse 11-20']).plot\
    .line(x='lag', y=['mse 11', 'mse 20', 'mse 11-20'], figsize=(15, 7), grid=True)
```

Listing 1: simple AutoRegressive model

Which resulted in the following plot:



According to this optimization, we were able to achieve a test error of ~ 11 in phase 1 competition using parameters $k = 2$ and using data from 15th March onward, only using one feature, 'deaths'. Also, there is a hint that the k value around $2 - 5$ is a good parameter since it leads to a good test error on the data for days $16 - 25$ October too (which were not part of the phase 1 competition test set).

3.4 Moving towards our Multivariate AutoRegression Model

Next, in order to improve our test results and using more features, but not using VAR, we came up with an idea to use the 'cases' feature in 'X' feature space alongside 'deaths', but not in 'Y'. The structure of our 'X' matrix and 'Y' vector is as shown below:

$$y = \begin{bmatrix} d_K \\ d_{K+1} \\ \vdots \\ d_T \end{bmatrix} \qquad X = \begin{bmatrix} x_K^T \\ x_{K+1}^T \\ \vdots \\ x_T^T \end{bmatrix} = \begin{bmatrix} 1 & d_1 & d_2 & \cdots & d_{K-1} & c_i & c_j & \cdots & c_k \\ 1 & d_2 & d_3 & \cdots & d_K & c_j & c_m & \cdots & c_l \\ \vdots & \cdots & \cdots & \cdots & \vdots & \vdots & \cdots & \cdots & \vdots \\ 1 & d_{T-K} & d_{T-K+1} & \cdots & d_{T-1} & c_n & c_o & \cdots & c_p \end{bmatrix}.$$

In which d_i is the 'deaths' time series and c_{ind} is the 'cases' with selected lags.

3.5 Choosing the validation set

In the process of choosing our hyper-parameters for the model, like the beginning index of the training data set to start the time series from, the lag (k) value, or the cases from previous days to include in the feature space, we need to first choose an evaluation set to search for the best hyper-parameters in it. Here, the question was if we should use the last 10 or the last 5 days as the validation set. Based on two reasoning, we chose to have the last 5 days as the validation set:

- According to an article, the average case confirmation to death time for people is 14 days, which for old (60+ yo) people, this average is about 11.5, with a range starting from 6, and for young people this average is about 19, starting from 14. Based on the fact that Canada does not have any issues with providing enough health care for infected people, we guess that most death cases are the old people in Canada and most young people are cured. So, the number of cases in the previous 6 days is probably useful in the final model. On the other hand, to be able to use the case data from 6 days ago in our model, we need the validation set to be smaller than 6 in size.
- In the end, we are going to predict 5 days, so let's choose the validation set that is as close as possible to our test set. In other words, we are optimizing hyper-parameters for tuning a model that is best for predicting 5 days onward.

3.6 Final hyper-parameter optimization

Finally, we used the data until 20th October to learn the model and used 21-25th October for the validation set. We trained the model on different hyper-parameters, namely:

- The beginning index of the data
- The lag of deaths used in feature space
- The lag of cases used in the feature space
- The number of consecutive cases used starting from the lag of cases values selected onward (only consecutive cases dates are tested)

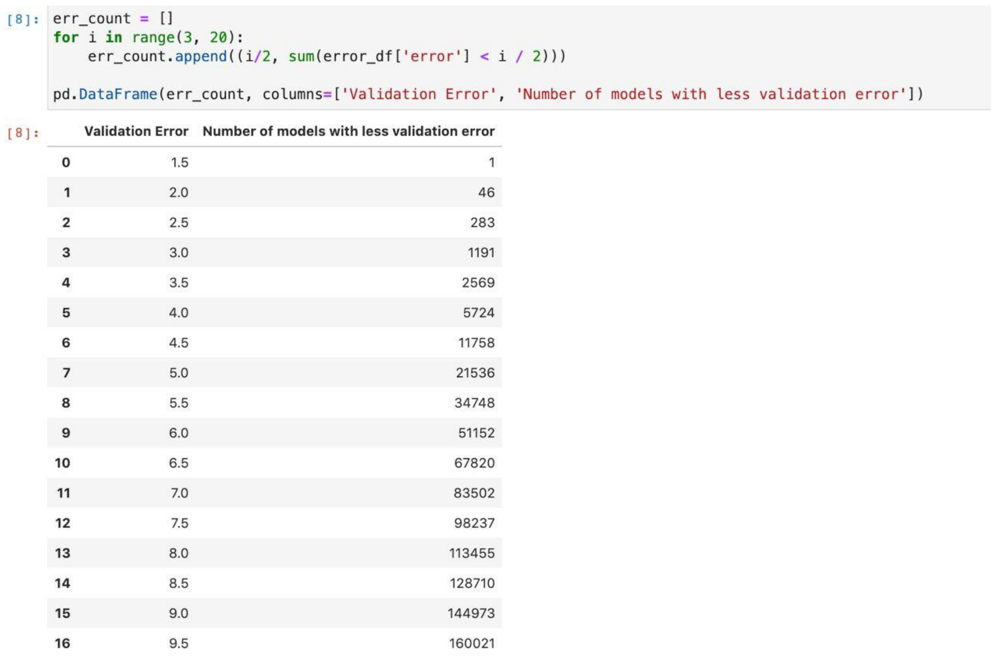
The code is provided in the following:

```python
from AutoReg_MultiVar import AutoRegressionModelMultiVar as armv

errs = []

i = 0
for begining_ind in range(70, 180):
    data = new_ca_train_df[begining_ind:]
    deaths = data['deaths'].values
    cases = data['cases'].values
    for k_lag in range(1, 20):
        for seq_case_count in range(20):
            for case_lag in range(5, 30):
                lag_indices = list(np.arange(case_lag, min(case_lag+seq_case_count, 30)))
                model = armv(k=k_lag, bias=True)
                if len(lag_indices) == 0:
                    model.fit(deaths, np.empty(0), np.empty(0))
                else:
                    model.fit(deaths, [cases], [lag_indices])
                error = calculate_error(ca_val_df['deaths'][-5:].values, model.predict(len(
    data), len(data)+4))
                errs.append((begining_ind, k_lag, seq_case_count, case_lag, error))
                i += 1
                if i % 1000 == 0:
                    print("i is:", i)
error_df = pd.DataFrame(errs, columns=['begining_index', 'k_lag', 'case_sequence_count', '
    case_lag', 'error'])
```

Listing 2: Final Hyper-Parameter Optimization

Finally, the results of validation were saved in 'error_df'. According to the result, the lowest validation error we achieved was '1.445660'.

The following plot shows the cumulative number of models with validation error less than the validation error mentioned in the "Validation Error" column.



```
[8]: err_count = []
     for i in range(3, 20):
         err_count.append((i/2, sum(error_df['error'] < i / 2)))

     pd.DataFrame(err_count, columns=['Validation Error', 'Number of models with less validation error'])
```

[8]:

	Validation Error	Number of models with less validation error
0	1.5	1
1	2.0	46
2	2.5	283
3	3.0	1191
4	3.5	2569
5	4.0	5724
6	4.5	11758
7	5.0	21536
8	5.5	34748
9	6.0	51152
10	6.5	67820
11	7.0	83502
12	7.5	98237
13	8.0	113455
14	8.5	128710
15	9.0	144973
16	9.5	160021

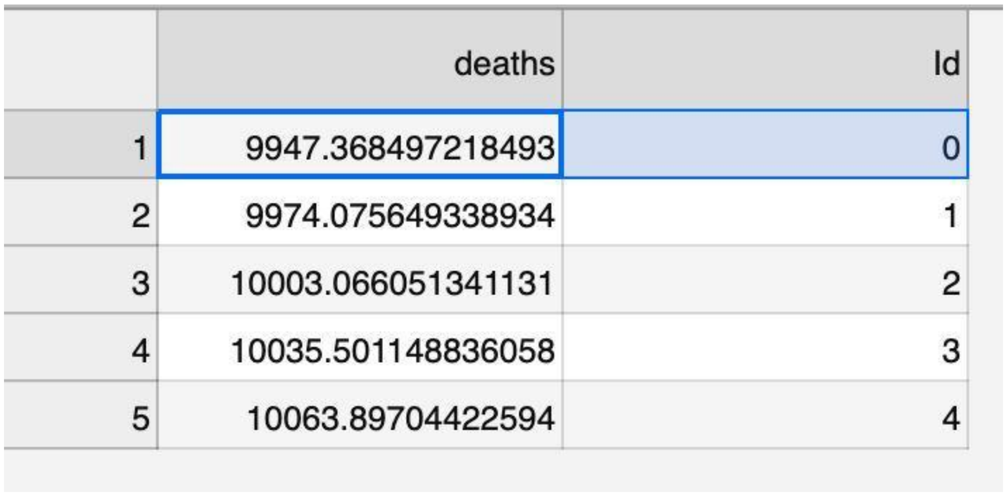
3.7 Final Model

Finally, to select the final model for prediction, for each day, we used the median value of predictions of the best 200 models sorted according to their validation error. We use the median here to avoid being mistaken by the outliers predicted from some of these models. The results are predicted and uploaded to the Kaggle competition phase 2.

4 Results

Team Name	Kaggle Phase 1 Score	Kaggle Phase 2 Score
Farnoosh didn't let me in	*6.29670*	*N.A.*

Our prediction is:



	deaths	Id
1	9947.368497218493	0
2	9974.075649338934	1
3	10003.066051341131	2
4	10035.501148836058	3
5	10063.89704422594	4

5 Conclusion

Working with time series data sets is indeed one of most interesting and difficult area in data science. Forecasting future is a very demanding and mysterious task in this area. In this take-home exam, we learned about time series data set and how we can work with and how we can use our machine learning knowledge, specially linear regression to solve a problem and particularly predict the future.

If we had more time, we would go over new creative features, like multiplication of existing features, polynomial functions of features and etc. Also one another thing that we would definitely try if we had time was examining different lags of 'cases' and different combinations of them to find the best lags of cases for predicting future of our deaths vector.

Appendix

Codes

```python
from numpy.linalg import solve
import numpy as np


def gradient_descent(X, y, theta, alpha=0.0005, num_iters=1000):
    '''Gradient descent for linear regression'''
    # Initialisation of useful values
    m = np.size(y)

    for i in range(num_iters):
        # Grad function in vectorized form
        h = X @ theta
        gradient = (1 / m) * (X.T @ (h - y))
        theta = theta - alpha * gradient
    return theta


def gradient_descent_reg(X, y, theta, alpha=0.0005, lamda=10, num_iters=1000):
    '''Gradient descent for ridge regression'''
    # Initialisation of useful values
    m = np.size(y)

    for i in range(num_iters):
        # Hypothesis function
        h = np.dot(X, theta)

        # Grad function in vectorized form
        theta = theta - alpha * (1 / m) * ((X.T @ (h - y)) + lamda * theta)

    return theta


def gradient_descent_las(X, y, theta, alpha=0.0005, lamda=10, num_iters=1000):
    '''Gradient descent for ridge regression'''
    # Initialisation of useful values
    m = np.size(y)

    for i in range(num_iters):
        # Hypothesis function
        h = np.dot(X, theta)

        # Grad function in vectorized form
        theta = theta - alpha * (1 / m) * ((X.T @ (h - y)) + lamda)

    return theta


# Least Squares with a bias added
class AutoRegressionModel:

    def __init__(self, k, bias=False, method="normal", alpha=0.0005, num_iters=1000, lamda
    =10):
        self.k = k
        self.bias = bias
        self.method = method
        self.alpha = alpha
        self.num_iters = num_iters
        self.lamda = lamda
        self.Z = None

    def fit(self, X):
        Z, y = self.__ZY_creator(X)
        self.Z = Z
        if self.method=="normal":
            self.w = solve(Z.T @ Z, Z.T @ y)
        elif self.method=="normal-L2":
            self.w = solve(Z.T @ Z + self.lamda*np.eye(Z.shape[1], dtype=float), Z.T @ y)
        elif self.method=="grad":
            w = np.zeros((Z.shape[1],1))
            w = gradient_descent(Z, y, w, alpha=self.alpha, num_iters=self.num_iters)
            self.w = w
        elif self.method=="grad-L2":
            w = np.zeros((Z.shape[1],1))
            w = gradient_descent_reg(Z, y, w, alpha=self.alpha, num_iters=self.num_iters,
    lamda=self.lamda)
            self.w = w
        elif self.method=="grad-L1":
            w = np.zeros((Z.shape[1],1))
            w = gradient_descent_las(Z, y, w, alpha=self.alpha, num_iters=self.num_iters,
    lamda=self.lamda)
            self.w = w

    def predict(self, X):
        Z = X[-self.k:]
        if self.bias:
            Z = np.insert(Z, 0, 1)
        return Z @ self.w

    def __ZY_creator(self, X):
        k = self.k

        num_samples = X.shape[0]

        temp_X = []
        y = []
        for i in range(num_samples - k):
            # Slice a window of features
            temp_X.append(X[i:i + k])
            y.append(X[i + k])

        Z = np.vstack(temp_X)
        y = np.array(y).reshape(-1, 1)

        n, d = Z.shape

        if self.bias:
            bias = np.ones((n, 1))
            Z = np.append(bias, Z, axis=1)

        return Z, y
```

Listing 3: AutoRegressionMultiVar Class