Final mark: 96%. Regrets, this pdf is corrupt and my comments on it were lost when I saved.
Come to office hour to go through the document. -LTE

STAT 440, FALL 2021
TEAMNAME: LTR

THE COVID-19 WASTEWATER SURVEILLANCE KAGGLE COMPETITION

Lakshay Sethi (301386559)
Tarunjeev Juneja (301329078)
Rajvir Sangha (301301767)

October 9, 2021

1 Introduction

This module studies the case counts of the current COVID-19 pandemic using various machine learning methodologies. Data about the COVID-19 cases in B.C. and wastewaster surveillance data is obtained from the following **Kaggle competition**. **The goal of this module is to use 550 x 14 case data points and 258 x 3 wasterwater data points to predict 2060 case counts for intervals of 1, 3, 5 or 7 days before the specified date.**

Before conducting any prediction modeling, pre-processing of the data was done to handle missing-data and irregular data points. This is an important step in the project and it likely had a substantial impact on our RMSE scores. After prepossessing the data, we were able to use machine learning techniques such as Multiple Linear Regression, Ada-Boost Regression, Ensemble Regression, and XG Boost Regression to choose the method with the lowest RMSE score. **The method that provided our team with the lowest RMSE score, equal to 103.76218, is XG Boost.**

2 Data Exploration

Firstly, we explored both of the datasets in an attempt to find patterns and peculiarities [Ghouzam, 2017]. Some of the exploratory methods that we used include:

- dataframe.info(): datatype for each column and see if any column has non-null values
- (dataframe.isnull().sum() / len(dataframe)) * 100: To see the percentage of missing values
- sns.heatmap(dataframe.corr()): Heatmap to check the correlation
- dataframe.agg(['skew']).transpose(): To check the skewness of the data

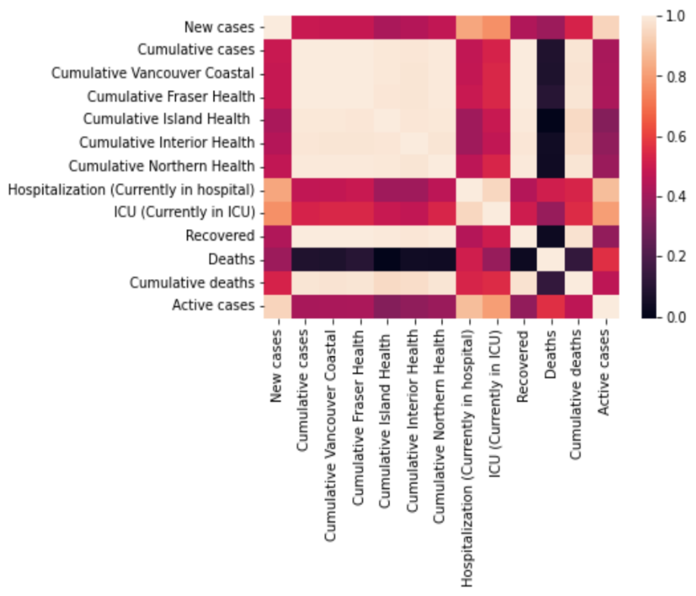
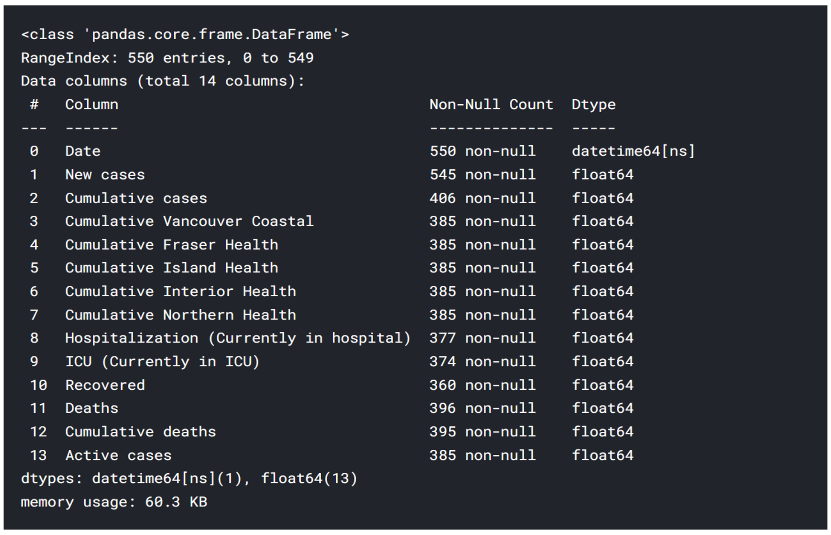


Figure 1: Correlation heatmap of the variables



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 550 entries, 0 to 549
Data columns (total 14 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   Date                                  550 non-null    datetime64[ns]
 1   New cases                             545 non-null    float64
 2   Cumulative cases                      406 non-null    float64
 3   Cumulative Vancouver Coastal          385 non-null    float64
 4   Cumulative Fraser Health              385 non-null    float64
 5   Cumulative Island Health              385 non-null    float64
 6   Cumulative Interior Health            385 non-null    float64
 7   Cumulative Northern Health            385 non-null    float64
 8   Hospitalization (Currently in hospital)  377 non-null    float64
 9   ICU (Currently in ICU)                374 non-null    float64
 10  Recovered                             360 non-null    float64
 11  Deaths                                396 non-null    float64
 12  Cumulative deaths                     395 non-null    float64
 13  Active cases                          385 non-null    float64
dtypes: datetime64[ns](1), float64(13)
memory usage: 60.3 KB
```

Figure 2: info() on the data

Through the exploratory methods, we found insightful patterns in the data-set:

- Seasonality in missing rows (NA values were present at every 7 day interval).
- The data in cases_data file had mostly slightly right skewed columns (between 0.5 and 1) and some highly right skewed columns(value > 1).
- Many columns were highly correlated.

This process allowed us to understand the dataset better and guided us on what to do in our next step, data-preprocessing.

3 PREPROCESSING

In order to combine both data sets and answer the challenge question, we first transformed the Plant Column in waste_water dataset into 5 different columns containing the RNA count in respect to that station to make it compatible with the dates in the cases_datset (solving the 700+ rows issue).

To establish a backtesting framework, we subset the entire dataset by rows that were before the target date. Then, we transformed the data column into 5 additional columns of 'Date year', 'Date month', 'Date weekday', 'Date day', and 'Date day of the week' to use dates in modeling.

Also, we corrected the skewness of the data by using a log transformation. Lastly, NaN values were handled by an K-means based imputation method [Belloni, 2018]. This method normalized the data.

4 Models

Method	RMSE
Multiple Linear Regression	107
AdaBoost regression	120
Ensemble regression	107
XG boost regression	103

Table 1: Table of various methods and their RMSE.

4.1 Multiple Linear Regression

Firstly, based on the lectures we tried various forms of Linear regression. Starting from regressing New cases with row numbers. This served as a base line for all further approaches. Next, we began Linear regression on basis of multiple variables. We observed a decrease in score from the baseline RMSE score obtained from the aforementioned method on the Kaggle Leaderboard. Within the testing of our linear regression model, we made sure that the distribution and the variance of the dependent variable was normalized and constant for all values of the independent variable.

During our exploration process, we tried to determine a mathematical relationship among several random variables which further helped us examine how multiple independent variables were related to our dependent variable 'New cases'. Once all of the independent factors were determined, the information on the multiple variables was used to create prediction of the dependent variable on the level of effect that they have on the outcome variable. Notably, the information on the multiple variables can be used to create an accurate prediction based on the impact they have on the outcome variable.

4.2 AdaBoost regression

The second approach was AdaBoost regression. Transitioning from linear regression to Adaboost allows us to secure non-linear relationships, which produces a better prediction accuracy. Weak models are inserted sequentially and trained using the weighted training data. The main reason to shift to a tree based model like AdaBoost is that it allows us to handle outliers and unexpected changes in the data points.

The RMSE score received from non hyper parameterized model was 120, which is significantly worse than the baseline. But still we believed a tree based model would be the best answer after plotting the RNA counts in the wastewater data set.

4.3 Ensemble regression

The third approach was Ensemble methods. Ensemble learning combines various base algorithms to construct one optimized predictive algorithm. We took advantage of the Parallel Learning technique of ensembles. In essence, generate base models parallel to each other and take an advantage of the independence between models by averaging the known mistakes.

Subsequently, we used the three models GBoost, Light GBoost, and XG Boost. We trained the three models separately and averaged the mean. The initial RMSE scores obtained from the mean of all three models surpassed the threshold. However, we observed that the score of XG Boost alone surpassed the mean value of all three models. Furthermore, GBoost and Light GBoost don't have the innate NaN value imputation capability. We observed a similar trend even after a small round of Cross validation with n estimators. The RMSE score obtained here was adequate to some extent but it is unacceptable as the final score.

4.4 XG boost regression

XGBoost is a decision tree based ensemble algorithm that uses a gradient boosting framework. It utilizes decision trees, bagging and boosting, which is the process of minimizing the errors of the previous models while boosting the influence of better performing models. For this module, we trained a XGBoost model using a RMSE loss function. It outperformed all other attempted models. Consequently, we began hyperparamter tuning it on max_depth, learning_rate, n_estimators, colsample_bytree using a Grid search CV.

5 CONCLUSION

5.1 Results

Nevertheless, various methods were considered and implemented to minimize the RMSE score using the B.C. COVID-19 cases information and wastewater dataset. We explored various pre-processing methodologies together as a team and our best model proved to be a XG Boost algorithm, providing us an RMSE of 103.76218 on Kaggle.

5.2 Lessons Learned

Through the course of this project, we learned about data prepossessing and fully comprehend it's significance on the RMSE score. We got the opportunity to work with time-series data for the first time in a real world setting and to research more about prediction and data imputation methods unique to time series problems. Moreover, we also learned about the effects of collinearity and skewness in data, and how it affects the RMSE score acquired from a regression model.

6 The Challenge Question

Method	RMSE with wasterwater data	RMSE without wasterwater data
Multilpe Linear Regression	108.11102	107.66323
AdaBoost regression	120.80162	119.34533
Ensemble method	110.14771	107.66323
XG boost regression	104.83795	103.76218

Table 2: Table of various methods of with and without wastewater data and their RMSE.

The challenge question is to determine whether the wastewater data improved the prediction of case counts or not. From table 2, it can be observed that the RMSE score decreases marginally when the wastewater data is used. As shown in the figure 3 below, this conclusion's rationale is that the data present in the wastewater dataset is highly scattered and does not match with the combined dataset because it is skewed. Thus, the various timestamps do not allow regression to accurately fit the model.

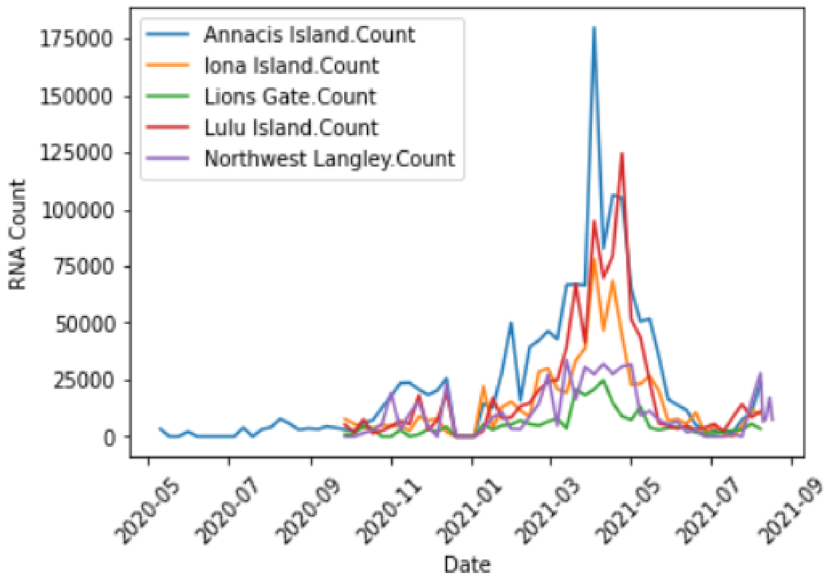


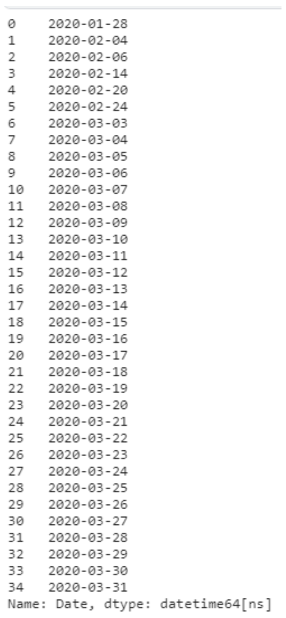
Figure 3: RNA Count vs Dates

Proof of Back testing

The backtesting method is implemented by training a model using Total Dates (D) before a certain day interval (T). In essence, the difference of D - T dates. Moreover, we created a function called 'training_set_creater' that outputs all data T dates before the target date. For instance, if the target date is 1st of April and T is 16 , then the function which return all the rows which have dates 31st March or before.

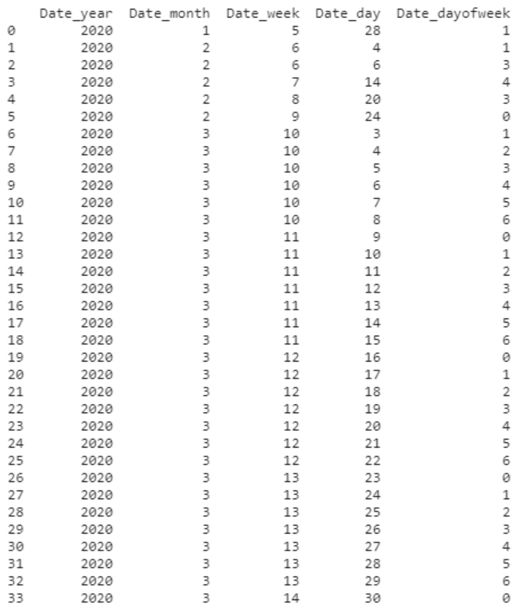
Additionally, the following code and the 4 figures below illustrate how the code transforms the data.

```
# Function to retrieve the data for dates before the target date.
def training_set_creater(target_date, days_to_subtract, data):
    train_data = data[target_date - timedelta(days=days_to_subtract)  >= data['Date']]
    return train_data

# Function to predict the number of new cases using the test and train data.
def predict_due_dates(target_date, days_to_subtract, data,sign):
    target_date = datetime.fromisoformat(target_date)
    train_data  = training_set_creater(target_date, int(days_to_subtract), data)
    print(train_data["Date"])  #===================================> A
    ################################################
    # Conversion of Date for usage in the model#

    train_data.loc[:, 'Date_year'] = train_data['Date'].dt.year
    train_data.loc[:, 'Date_month'] = train_data['Date'].dt.month
    train_data.loc[:, 'Date_week'] = train_data['Date'].dt.isocalendar().week.astype('
    int64')
    train_data.loc[:, 'Date_day'] = train_data['Date'].dt.day
    train_data.loc[:, 'Date_dayofweek'] = train_data['Date'].dt.dayofweek
    train_data = train_data.loc[:, train_data.columns != 'Date']
    ################################################################
    lakshay_imputer(cases_data.iloc[0:34,:],"New cases",'Cumulative cases',1)
    lakshay_imputer(cases_data.iloc[0:34,:],"Deaths",'Cumulative deaths',1)
    ################################################################
    # using KNN based imputation stratergy (data auto normalized)
    knnmv_imp = KNNMVImputer(strategy="mean", k=3, l=0.25)
    sample = train_data.copy()
    sample = sample.drop(['New cases','Date_year', 'Date_month', 'Date_week', 'Date_day
    ', 'Date_dayofweek'], axis = 1)
    sample = sample.values
    ab = knnmv_imp.fit_transform(sample)
    sample1 = train_data.copy()
    sample1 = sample1.drop(['New cases','Date_year', 'Date_month', 'Date_week', '
    Date_day', 'Date_dayofweek'], axis = 1)
    df = pd.DataFrame(ab,columns=sample1.columns)
    df[['New cases','Date_year', 'Date_month', 'Date_week', 'Date_day', 'Date_dayofweek
    ']] = train_data[['New cases','Date_year', 'Date_month', 'Date_week', 'Date_day', '
    Date_dayofweek']]
    ##########################################
    # Creating Training and testing sets
    X = df.loc[:, df.columns != 'New cases']
    X_= X.iloc[-1,:]
    X = X.iloc[:-int(days_to_subtract),:]
    y = df['New cases'].shift(periods= -int(days_to_subtract)) # -1 * days_to_subtract
    y = y.iloc[:-int(days_to_subtract)]
    # Row of data used for the final prediction
    # To see if the date being passed is correct or not
    test_data = X_
    print(X[['Date_year', 'Date_month', 'Date_week', 'Date_day', 'Date_dayofweek']])
    #============================================> B
    print(y)#====================================================> C
    print(test_data) #============================================> D
    test_data = X.iloc[-1,:].values.reshape(1,test_data.shape[0]) # -1 *
    days_to_subtract

    # To get the final predicted result
    result = predictor(X,y,test_data,sign)

    return result
```



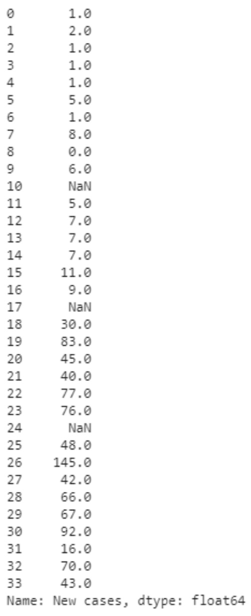
```
0     2020-01-28
1     2020-02-04
2     2020-02-06
3     2020-02-14
4     2020-02-20
5     2020-02-24
6     2020-03-03
7     2020-03-04
8     2020-03-05
9     2020-03-06
10    2020-03-07
11    2020-03-08
12    2020-03-09
13    2020-03-10
14    2020-03-11
15    2020-03-12
16    2020-03-13
17    2020-03-14
18    2020-03-15
19    2020-03-16
20    2020-03-17
21    2020-03-18
22    2020-03-19
23    2020-03-20
24    2020-03-21
25    2020-03-22
26    2020-03-23
27    2020-03-24
28    2020-03-25
29    2020-03-26
30    2020-03-27
31    2020-03-28
32    2020-03-29
33    2020-03-30
34    2020-03-31
Name: Date, dtype: datetime64[ns]
```

(a) Data after removing T days



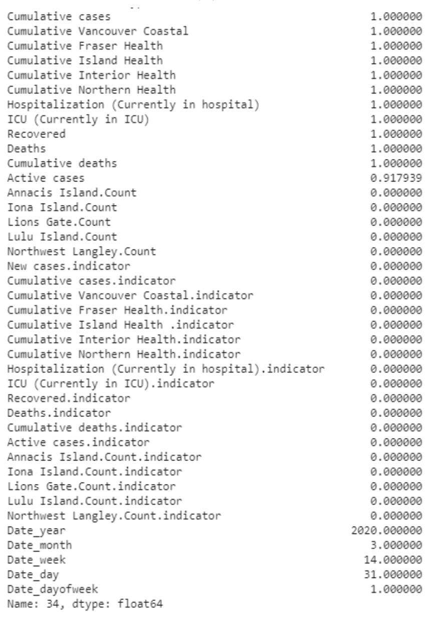
```
    Date_year  Date_month  Date_week  Date_day  Date_dayofweek
0        2020           1          5        28               1
1        2020           2          6         4               1
2        2020           2          6         6               3
3        2020           2          7        14               4
4        2020           2          8        20               3
5        2020           2          9        24               0
6        2020           3         10         3               1
7        2020           3         10         4               2
8        2020           3         10         5               3
9        2020           3         10         6               4
10       2020           3         10         7               5
11       2020           3         10         8               6
12       2020           3         11         9               0
13       2020           3         11        10               1
14       2020           3         11        11               2
15       2020           3         11        12               3
16       2020           3         11        13               4
17       2020           3         11        14               5
18       2020           3         11        15               6
19       2020           3         12        16               0
20       2020           3         12        17               1
21       2020           3         12        18               2
22       2020           3         12        19               3
23       2020           3         12        20               4
24       2020           3         12        21               5
25       2020           3         12        22               6
26       2020           3         13        23               0
27       2020           3         13        24               1
28       2020           3         13        25               2
29       2020           3         13        26               3
30       2020           3         13        27               4
31       2020           3         13        28               5
32       2020           3         13        29               6
33       2020           3         14        30               0
```

(b) X data frame with train values



```
0       1.0
1       2.0
2       1.0
3       1.0
4       1.0
5       5.0
6       1.0
7       8.0
8       0.0
9       6.0
10      NaN
11      5.0
12      7.0
13      7.0
14      7.0
15     11.0
16      9.0
17      NaN
18     30.0
19     83.0
20     45.0
21     40.0
22     77.0
23     76.0
24      NaN
25     48.0
26    145.0
27     42.0
28     66.0
29     67.0
30     92.0
31     16.0
32     70.0
33     43.0
Name: New cases, dtype: float64
```

(c) y data frame with labels

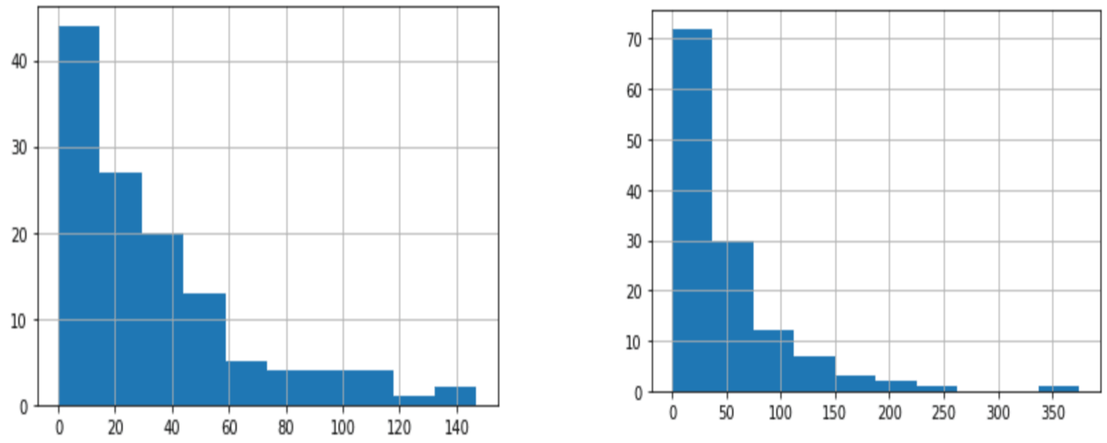


```
Cumulative cases                                       1.000000
Cumulative Vancouver Coastal                           1.000000
Cumulative Fraser Health                               1.000000
Cumulative Island Health                               1.000000
Cumulative Interior Health                             1.000000
Cumulative Northern Health                             1.000000
Hospitalization (Currently in hospital)                1.000000
ICU (Currently in ICU)                                 1.000000
Recovered                                              1.000000
Deaths                                                 1.000000
Cumulative deaths                                      1.000000
Active cases                                           0.917939
Annacis Island.Count                                   0.000000
Iona Island.Count                                      0.000000
Lions Gate.Count                                       0.000000
Lulu Island.Count                                      0.000000
Northwest Langley.Count                                0.000000
New cases.indicator                                    0.000000
Cumulative cases.indicator                             0.000000
Cumulative Vancouver Coastal.indicator                 0.000000
Cumulative Fraser Health.indicator                     0.000000
Cumulative Island Health .indicator                    0.000000
Cumulative Interior Health.indicator                   0.000000
Cumulative Northern Health.indicator                   0.000000
Hospitalization (Currently in hospital).indicator      0.000000
ICU (Currently in ICU).indicator                       0.000000
Recovered.indicator                                    0.000000
Deaths.indicator                                       0.000000
Cumulative deaths.indicator                            0.000000
Active cases.indicator                                 0.000000
Annacis Island.Count.indicator                         0.000000
Iona Island.Count.indicator                            0.000000
Lions Gate.Count.indicator                             0.000000
Lulu Island.Count.indicator                            0.000000
Northwest Langley.Count.indicator                      0.000000
Date_year                                           2020.000000
Date_month                                             3.000000
Date_week                                             14.000000
Date_day                                              31.000000
Date_dayofweek                                         1.000000
Name: 34, dtype: float64
```

(d) Data for prediction

The Bonus Question

The bonus question is to determine whether rainfall affects the ratio between wasterwater RNA counts and reported case counts. The historical rainfall data is collected from **Weather Stats**. The selected two columns are dates and an indicator variable for whether it rained or not. The rainfall data is then combined with wastewater data and columns such as Plant and index are dropped. Similarly, two columns are selected from the case dataset, specficially, dates and New cases that are combined to form the merged dataset between wastewater and rain datasets where they are joined on the basis of their dates. Subsequently, we computed the ratio between RNA counts and reported case counts.



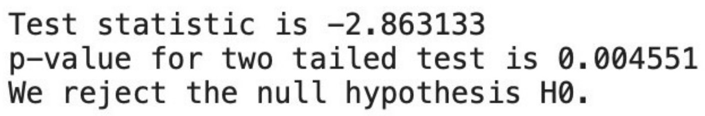
(a) Ratio with rain

(b) Ratio with no rain

Figure 5: Histogram of RNA counts Vs. Case counts

We hypothesize from Figure 5 that there is an effect of rain on the ratio between RNA counts and case counts and in order to further explore it we apply a t-test [R., 2021]. Note that μ_0 and μ_1 are means of the ratio with rain and no rain, respectively.

$$H_0 : \mu_0 = \mu_1,$$
$$H_a : \mu_0 \neq \mu_1$$



```
Test statistic is -2.863133
p-value for two tailed test is 0.004551
We reject the null hypothesis H0.
```

Figure 6: Results of the hypothesis test

An alpha level of 0.05 is selected for the hypothesis testing. The p-value for the two tailed test is equal to 0.004551. This p-value is less than 0.05 and thus, we reject the null hypothesis and conclude that the rainfall affects the ratio between wastewater RNA counts and reported case counts.

References

[Belloni, 2018] Belloni, M. (2018). Xgboost is not black magic. https://towardsdatascience.com/xgboost-is-not-black-magic-56ca013144b4.

[Ghouzam, 2017] Ghouzam, Y. (2017). Eda, introduction to ensemble regression. https://www.kaggle.com/yassineghouzam/eda-introduction-to-ensemble-regression.

[R., 2021] R., L. (2021). T-test - performing hypothesis testing with python. https://www.analyticsvidhya.com/blog/2021/07/t-test-performing-hypothesis-testing-with-python/.

Project 1 STAT 440

Authored by Lakshay Sethi

Double-click (or enter) to edit

```
import pandas as pd
import numpy as np
import  datetime
from datetime import datetime, timedelta
! pip install openpyxl
! pip install xgboost
! pip install knnmv
! pip install lightgbm
import xgboost as xg
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import AdaBoostRegressor, GradientBoostingRegressor
from sklearn.ensemble import RandomForestRegressor
import lightgbm as lgb
from knnmv.impute import KNNMVImputer
from sklearn.metrics import mean_squared_error
```

```
import warnings
warnings.filterwarnings("ignore")
```

```
# Reading data
pred_data = pd.read_csv('../input/stat440-21-module1/predictions.csv')
cases_data = pd.read_excel(open('../input/stat440-21-module1/BC COVID CASES.xlsx', 'rb'), she
waste_water_data = pd.read_excel(open('../input/stat440-21-module1/BC COVID CASES.xlsx', 'rb'
```

```
cases_data.info()
```

```
# Heatmap
corrmat = cases_data.corr()
g = sns.heatmap(cases_data.corr())
```

```
# most correlated features
top_corr_features = corrmat.index[abs(corrmat['New cases'])>0.5]
g = sns.heatmap(cases_data[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```

```
# To see the missing percentage of values
all_data_na = (cases_data.isnull().sum() / len(cases_data)) * 100
all_data_na = (cases_data.isnull().sum() / len(cases_data)) * 100
all_data_na = all_data_na.drop(all_data_na[all_data_na == 0].index).sort_values(ascending=Fal
missing_data = pd.DataFrame({'Missing Ratio' :all_data_na})
missing_data


cases_data.agg(['skew']).transpose()


import matplotlib.pyplot as plt


# preparing waste water dataset for combining
df = waste_water_data.copy()
df = df.groupby(['Date', 'Plant']).size().unstack(fill_value=0).reset_index().rename_axis(Non
tmp = df.Date
fg = pd.DataFrame(tmp,columns = ['Date'])
for x in waste_water_data.Plant.unique():
    Annacis_df = waste_water_data.copy()
    Annacis_df = Annacis_df[Annacis_df['Plant'] == x].rename(columns = {"Count": x + ".Count"
    fg = fg.join(Annacis_df.set_index('Date'), on='Date').reset_index()
    fg.drop('index', axis=1, inplace=True)

for x in fg.iloc[:,fg.columns != "Date"]:
    plt.plot(fg.Date,fg[x], label = x)
    plt.xticks(rotation=45)
    plt.xlabel('Date')
    plt.ylabel('RNA Count')
    plt.legend(loc="upper left")

plt.show()


# Attempt to check skewness of waste water data
# fg.agg(['skew', 'kurtosis']).transpose()
# # skewed data
# skew1 = fg.copy()
# #skew = skew.loc[:, cases_data.columns != 'New cases']
# skew1 = skew1.loc[:, skew1.columns != 'Date']

# for feature in skew1.columns:
#     skew1[feature] = np.log1p(skew1[feature])

# #skew['New cases'] = cases_data['New cases']
# skew1['Date'] = fg['Date']
# skew1.agg(['skew', 'kurtosis']).transpose()


# Dateframe created from combining cases and wastewater pages of the BC COVID CASES excel fil
combined_covid_date = cases_data.join(fg.set_index('Date'), on='Date').reset_index()
combined_covid_date.drop('index', axis=1, inplace=True)
from sklearn.impute import MissingIndicator
indicator = MissingIndicator()
for x in combined_covid_date.iloc[:,combined_covid_date.columns!='Date'].columns:
```

```
for x in combined_covid_date.iloc[:,combined_covid_date.columns!='Date'].columns:
    combined_covid_date[x + '.indicator'] = indicator.fit_transform(combined_covid_date[['Dat
combined_covid_date[['Annacis Island.Count', 'Iona Island.Count', 'Lions Gate.Count',
       'Lulu Island.Count', 'Northwest Langley.Count']] = combined_covid_date[['Annacis Islan
       'Lulu Island.Count', 'Northwest Langley.Count']].fillna(0)
combined_covid_date.head()


# Retrieving the unique target dates from the prediction.csv file
target_dates = pred_data['Date:Delay'].str.split(pat=':', expand=True)[0]
day_diff = pred_data['Date:Delay'].str.split(pat=':', expand=True)[1]
#day_diff = day_diff.astype(int)
print("Unique target dates:", target_dates.unique().size)


# Code to add indicator variables for rows with nan values
from sklearn.impute import MissingIndicator
indicator = MissingIndicator()
for x in cases_data.iloc[:,cases_data.columns!='Date'].columns:
    cases_data[x + '.indicator'] = indicator.fit_transform(cases_data[['Date',x]].values)*1


#  Code to create a real data subset for intermal rmse testing
# test_data = list()
# temp_impute = target_dates
# for i in range(target_dates.size):
#      #val = (test_data[test_data['Date'] == temp_impute.iloc[i]]['Cumulative cases'].iloc[0
#     test_data.append(cases_data.iloc[cases_data[cases_data['Date'] == temp_impute[i]].index


# # #print(test_data)
# pd.DataFrame(list(zip(pred_data['Date:Delay'], test_data)),columns =['Date:Delay', 'Count']


# correcting skewness for cases_data
skew = cases_data.copy()
skew = skew.loc[:, cases_data.columns != 'New cases']
skew = skew.loc[:, skew.columns != 'Date']

for feature in skew.columns:
    skew[feature] = np.log1p(skew[feature])



skew['New cases'] = cases_data['New cases']
skew['Date'] = cases_data['Date']


# Function to retrieve the data for dates before the target date.
def training_set_creater(target_date, days_to_subtract, data):
    train_data = data[target_date - timedelta(days=days_to_subtract)  >= data['Date']]
    return train_data


def lakshay_imputer(data,col_change, col_used, days):
    # new cases imputation using formula
    # For example:
    # New cases[t] = Cumulative cases[t + 1] -  Cumulative cases[t - 1]
    # beacuse Cumulative cases is the total number of cases
    iu = data.copy()

    temp_impute = data[data[col_change].isnull()]['Date']

    for i in range(temp_impute.size):
        val = (iu[iu['Date'] == (temp_impute.iloc[i] + timedelta(days=days))][col_used].iloc[
        data.at[iu[iu['Date'] == temp_impute.iloc[i]].index , col_change ] = val

        val = (data[data['Date'] == temp_impute.iloc[i]][col_change].iloc[0] + iu[iu['Date']
        data.at[iu[iu['Date'] == temp_impute.iloc[i]].index , col_used ] = val

    return data


# Function to predict the number of new cases using the test and train data.
def predict_due_dates(target_date, days_to_subtract, data,sign):
    target_date = datetime.fromisoformat(target_date)
    train_data  = training_set_creater(target_date, int(days_to_subtract), data)
    print(train_data["Date"])
    ##################################################
    # Conversion of Date for usage in the model#

    train_data.loc[:, 'Date_year'] = train_data['Date'].dt.year
    train_data.loc[:, 'Date_month'] = train_data['Date'].dt.month
    train_data.loc[:, 'Date_week'] = train_data['Date'].dt.isocalendar().week.astype('int64')
    train_data.loc[:, 'Date_day'] = train_data['Date'].dt.day
    train_data.loc[:, 'Date_dayofweek'] = train_data['Date'].dt.dayofweek
    train_data = train_data.loc[:, train_data.columns != 'Date']
    #########################################################################
    lakshay_imputer(cases_data.iloc[0:34,:],"New cases",'Cumulative cases',1)
    lakshay_imputer(cases_data.iloc[0:34,:],"Deaths",'Cumulative deaths',1)
    #######################################################################
    # using KNN based imputation stratergy (data auto normalized)
    knnmv_imp = KNNMVImputer(strategy="mean", k=3, l=0.25)
    sample = train_data.copy()
    sample = sample.drop(['New cases','Date_year', 'Date_month', 'Date_week', 'Date_day', 'Da
    sample = sample.values
    ab = knnmv_imp.fit_transform(sample)
    sample1 = train_data.copy()
    sample1 = sample1.drop(['New cases','Date_year', 'Date_month', 'Date_week', 'Date_day', '
    df = pd.DataFrame(ab,columns=sample1.columns)
    df[['New cases','Date_year', 'Date_month', 'Date_week', 'Date_day', 'Date_dayofweek']] =
    ##########################################
    # Creating Training and testing sets
    X = df.loc[:, df.columns != 'New cases']
    X_= X.iloc[-1,:]
    X = X.iloc[:-int(days_to_subtract),:]
    y = df['New cases'].shift(periods= -int(days_to_subtract)) # -1 * days_to_subtract
    y = y.iloc[:-int(days_to_subtract)]
    # Row of data used for the final prediction
    # To see if the date being passed is correct or not
    test_data = X_
    print(X[['Date_year', 'Date_month', 'Date_week', 'Date_day', 'Date_dayofweek']])
    print(y)
    print(test_data)
    test_data = X.iloc[-1,:].values.reshape(1,test_data.shape[0]) # -1 * days_to_subtract

    # To get the final predicted result
    result = predictor(X,y,test_data,sign)

    return result



def predictor(X,y,test_data,sign):
# create an xgboost regression model
    regr = xg.XGBRegressor(colsample_bytree=0.5, gamma=0.0468,
                             learning_rate=0.05, max_depth=7,
                             min_child_weight=0.5, n_estimators=2200,
                             reg_alpha=0.8, reg_lambda=0.7,
                             subsample=0.5213,
                             random_state =7, nthread = -1)

    regr.fit(X, y)
    res = regr.predict(test_data)[0]

    return res




input_data = combined_covid_date
pred_list2 = list()
for i in range(target_dates.size): #target_dates.size
    #print(i)
    pred_list2.append(predict_due_dates(target_dates[i], day_diff[i],input_data,1 ))
pd.DataFrame(list(zip(pred_data['Date:Delay'], pred_list2)),columns =['Date:Delay', 'Count'])



# skewed data for combined dataset
skew1 = combined_covid_date.copy()
#skew = skew.loc[:, cases_data.columns != 'New cases']
skew1 = skew1.loc[:, skew1.columns != 'Date']

for feature in skew1.columns:
    skew1[feature] = np.log1p(skew1[feature])

#skew['New cases'] = cases_data['New cases']
skew1['Date'] = fg['Date']
```

```
# # hyper parameter tuning

# def para_tuning(target_date, days_to_subtract, data,sign):
#     target_date = datetime.fromisoformat(target_date)
#     train_data  = training_set_creater(target_date, int(days_to_subtract), data)

#     #################################################
#     #train_data = train_data.iloc[:-int(14),:]
#     #################################################
#     # Conversion of Date for usage in the model

#     ############### Temporary decision to remove date column ###################

#     train_data.loc[:, 'Date_year'] = train_data['Date'].dt.year
#     train_data.loc[:, 'Date_month'] = train_data['Date'].dt.month
#     train_data.loc[:, 'Date_week'] = train_data['Date'].dt.isocalendar().week.astype('int64
#     train_data.loc[:, 'Date_day'] = train_data['Date'].dt.day
#     train_data.loc[:, 'Date_dayofweek'] = train_data['Date'].dt.dayofweek
#     train_data = train_data.loc[:, train_data.columns != 'Date']
#     ########################################################################

#     #knnmv_imp = KNNMVImputer(strategy="median", k=5, l=0.25)

#     sample = train_data.copy()
#     sample = sample.drop(['New cases'], axis = 1)
#     sample = sample.values
#     #print((sample))
#     ab = knnmv_imp.fit_transform(sample)
#     sample1 = train_data.copy()
#     sample1 = sample1.drop(['New cases'], axis = 1)
#     df = pd.DataFrame(ab,columns=sample1.columns)
#     # filled na's in new cases column with 0 , not sure
#     df['New cases'] = train_data['New cases'].fillna(0)
#     #print(df.head())
#     ##########################################
#     X = df.loc[:, df.columns != 'New cases']
#     X = X.iloc[:-int(days_to_subtract),:]

#     y = df['New cases'].shift(periods= -int(days_to_subtract)) # -1 * days_to_subtract
#     y = y.iloc[:-int(days_to_subtract)]

#     # Row of data used for the final prediction
#     test_data = X.iloc[-1,:]
#     test_data = test_data.values.reshape(1,17) # -1 * days_to_subtract
#     from sklearn.model_selection import GridSearchCV
#     # To get the final predicted result


#     params = {
#             'objective':['reg:squarederror']
#             'max_depth': [3,6,7],
#             'learning_rate': [0.01, 0.05, 0.1],
#             'n_estimators': [100, 500, 1000],
#             'colsample_bytree': [0.3, 0.7 ,0.8]}

#     xgbr = xg.XGBRegressor(seed = 20)
#     clf = GridSearchCV(estimator=xgbr,
#                        param_grid=params,
#                        scoring='neg_mean_squared_error',
#                        verbose=1)
#     clf.fit(X, y)
#     print("Best parameters:", clf.best_params_)
#     print("Lowest RMSE: ", (-clf.best_score_)**(1/2.0))

#     return clf.best_params_
```

BONUS QUESTION

```
# rainfall_data = pd.read_csv('../input/rainstatdata/weatherstats_vancouver_daily_stat.csv')
# # rainfall_data.head()
# # converting date to datetime format
# # loaded the rain data and converted date into datetime object
# rainfall_data['date'] = pd.to_datetime(rainfall_data['date'])
# rainfall_data.columns
# #filtering two columns to rain_data
# # Filtered the columns from the rain data
# # And renamed the date to Date
# rain_data = rainfall_data[['date','rain']]
# rain_data.rename(columns = {'date' : 'Date'}, inplace = True)
# rain_data.head()
# # #joining wastewater and rain on date
# # #combined_rain_waste = pd.merge(waste_water_data, rain_data, on='Date', how='inner')
# # combining the waste water data with the rain data and dropping plant and index column
# combined_rain_waste = waste_water_data.join(rain_data.set_index('Date'), on='Date').reset_i
# combined_rain_waste.drop('index', axis=1, inplace=True)
# combined_rain_waste.drop('Plant', axis=1, inplace=True)
# combined_rain_waste
# # #combined_rain_waste[combined_rain_waste.rain == 0]
# # # df2 = combined_rain_waste[combined_rain_waste.rain > 0.0]
# # # df2.head()
# # #combined_rain_waste.head()
# # making new dataframe by taking Date and New cases column from cases data and
# # combining with the dataframe combined_rain_waste
# # i.e from combining wastewater and rain data
# # now spliting into two dataframe, one with
# bonus_data_set = cases_data.copy()
# bonus_data_set = bonus_data_set[['Date','New cases']]
# #Annacis_df = Annacis_df[Annacis_df['Plant'] == x].rename(columns = {"Count": x + ".Count"}
# bonus_data_set = bonus_data_set.join(combined_rain_waste.set_index('Date'), on='Date').rese
# bonus_data_set.drop('index', axis=1, inplace=True)

# bonus_data_set = bonus_data_set.dropna().reset_index()
# bonus_data_set.drop('index', axis=1, inplace=True)
# print(bonus_data_set.shape)
# bonus_data_set_no_rain = bonus_data_set[ bonus_data_set['rain'] == 0.0 ]
# bonus_data_set_rain = bonus_data_set[ bonus_data_set['rain'] != 0.0 ]
# bonus_data_set_no_rain.head()
# bonus_data_set_rain.head()
# bonus_data_set_rain["ratio"] = bonus_data_set_rain["Count"] / bonus_data_set_rain["New case
# bonus_data_set_no_rain["ratio"] = bonus_data_set_no_rain["Count"] / bonus_data_set_no_rain[
# bonus_data_set_rain["ratio"].hist()
# bonus_data_set_rain["ratio"]
# bonus_data_set_no_rain["ratio"].hist()
# from scipy import stats
# t_value,p_value=stats.ttest_ind(bonus_data_set_rain["ratio"],bonus_data_set_no_rain["ratio"
# print('Test statistic is %f'%float("{:.6f}".format(t_value)))
# print('p-value for two tailed test is %f'%p_value)
# alpha = 0.05
# if p_value<=alpha:
#     print('We reject the null hypothesis H0.')
# else:
#     print('We do not reject the null hypothesis H0')
```