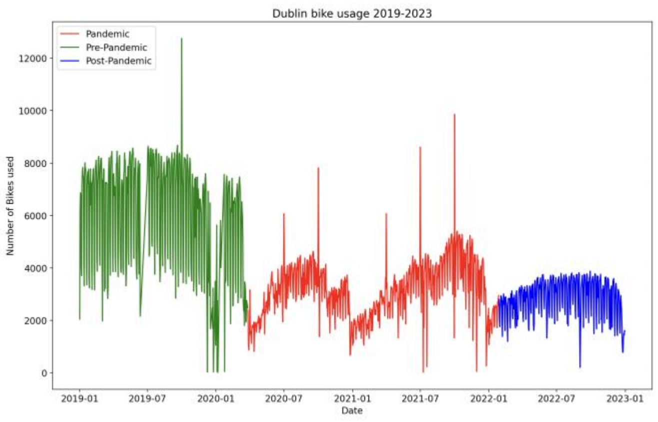
Machine Learning Final Report Declan Quinn 20334565

Data files

My approach involves a lot of pre-processing of the original data files before running. Therefore I have done this pre-processing in advance and have included the final pre-processed files with my code which will work without needing to preproccess titled 'PrePandemicdata.csv', 'Pandemicdata.csv' and ‚PostPandemicdata.csv'. I have included the final files used in my code and have left the option to reprocess the files in my Data Cleaning.py file. To reprocess the csv files from scratch run Data Cleaning.py the program with my named existing files and they should overwrite, this takes a long time. Otherwise run FinalAssignment20334565.py with the pre-loaded files.
Data Pre-processing and feature engineering:

Pre-processing Approach

When presented with the problem of cleaning the data from the Dublin bikes dataset, I decided on a way to estimate the average number of bikes used each day. The logic was to first combine the data from each csv file into each year. Then I would want to perform a count of the number of times the 'Available Bikes' parameter changed throughout each day. In order to do this I had to reduce the data I needed to process. I started by dropping columns I did not need. I then created a loop that for each available date in the dataset drop any adjacent values where the bike availability had not changed for each station. This cleared out a lot of noise and allowed me to approximate a count of the change in the number of available bikes per day. To avoid double counting bikes being returned to stands increasing the number of available bikes or not returned by the end of the day (after 23:59), I divided the total count by 2. This allowed me to have a good estimate of the daily bike usage in Dublin city. I calculated the date and stored them in csv files for the pre pandemic period, the pandemic period and the post pandemic period as shown below.
Issues I faced with this approach for pre-processing was that a large number of date times are not included in the original source files, this lead to certain days not being counted as evident from the graph below for example the 2019-06 period. Some extreme datapoints were extreme and would skew the data. Finally the biggest problem with this approach was that from 2022 onward data collection changed from every 5 minutes to every 30 minutes. This negatively affected my average predictions for the post pandemic period as I lost a lot of data regarding bikes taken and returned within that period. Hence why it shows the least amount of bike usage. With that in mind we can still expect that the Dublin bike usage has reduced since before and after the pandemic

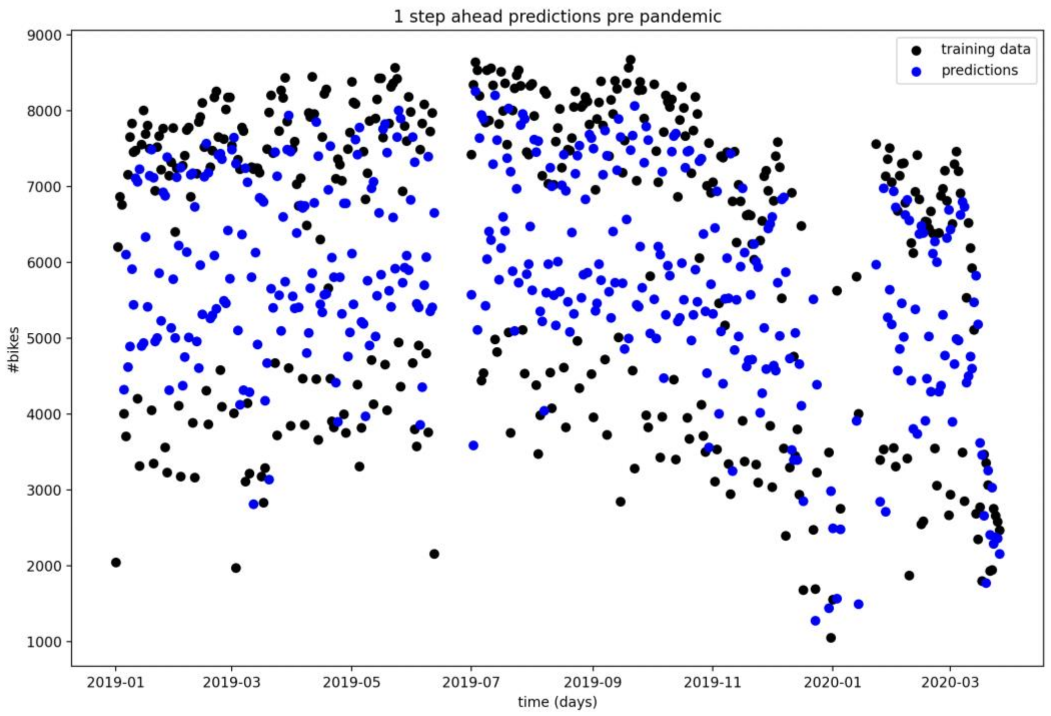


<u>ML approach</u>

Predicting bike usage if the pandemic had never happened:

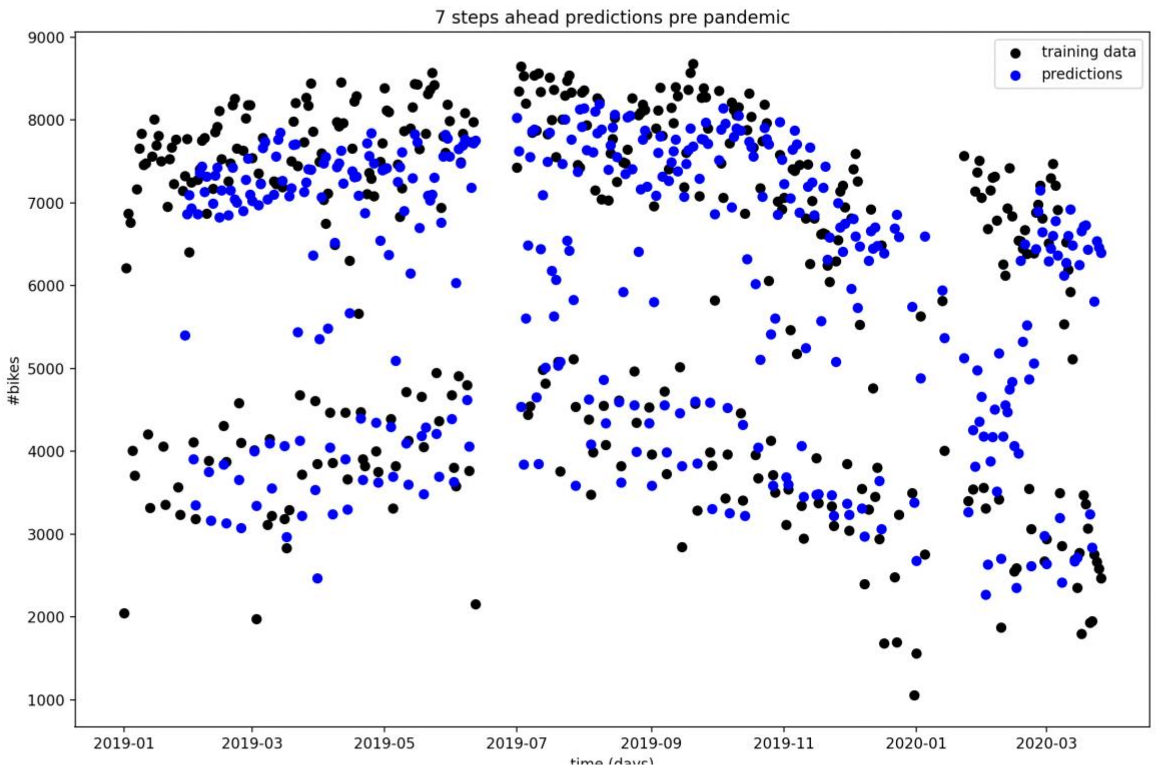
This is a time series problem as we are given data over a period of time and must make predictions based off the data for the future. The data has clear seasonality, for the summer months bike usage is the highest and for the winter months we can see it is at the lowest. Also there is weekly seasonality as we see that on weekdays bike usage is higher in the city than on weekdays which could be correlated to commuter use. Ridge regression was useful for this dataset as it can be robust to extreme outliers by assigning low parameter weights to them. It can also handle a large number of coefficients by being able to effectively manage their contribution to the predictions by using regularisation to lower their weights. I modified the ridge regression model from the notes to predict q step ahead training on the training data and plotted the results. I used a lag of three and used the past three days as features.

1 day ahead minimal feature engineering (last three days)



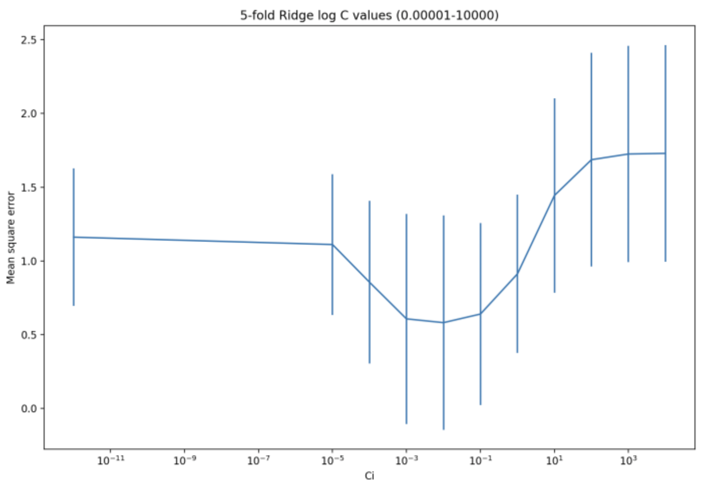
As you can see by the data the predictions have a high variance within the data suggesting that the weekend seasonality is not getting effectively captured.

1 day ahead better features (last three weeks)



Trying to account for higher seasonality I found that checking each value by week provided better results for our training data as we can observe our predictions closer to our training points.

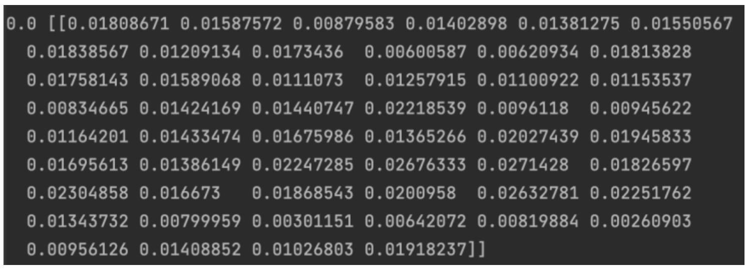
Choosing a C value for ridge regression
I performed K fold cross validation to test different C penalty terms to use on my model, Ridge regression uses an L2 penalty term to the data, this adds a penalty to the cost function of the regression model. By choosing the Ci value that gives the lowest mean square error I can reduce overfitting of model parameters. From the graph it looks like C = 0.01 is the best value to use.



Model Rationale

I decided on a multi-step time series prediction approach for my model. The rationale was that each day of the week would be correlated with the most previous seven days, so as there is less correlation with the previous days we could have a dataframe of previous entries of that day and try to predict that day next week. This allowed me to predict the next seven days in advance without losing too much data often lost in Multi step prediction farther into the future. This left me flexible to then train a new model with the next weeks data included and repeat for the length of the pandemic.

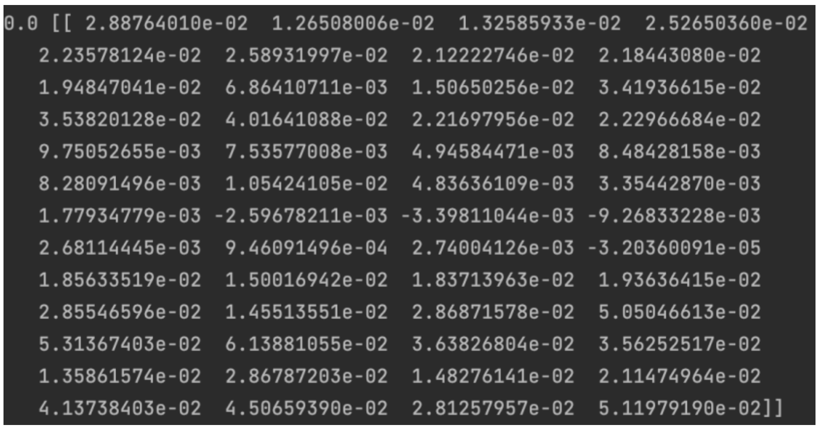
Coefficcient values initial model



```
0.0 [[0.01808671 0.01587572 0.00879583 0.01402898 0.01381275 0.01550567
  0.01838567 0.01209134 0.0173436  0.00600587 0.00620934 0.01813828
  0.01758143 0.01589068 0.0111073  0.01257915 0.01100922 0.01153537
  0.00834665 0.01424169 0.01440747 0.02218539 0.0096118  0.00945622
  0.01164201 0.01433474 0.01675986 0.01365266 0.02027439 0.01945833
  0.01695613 0.01386149 0.02247285 0.02676333 0.0271428  0.01826597
  0.02304858 0.016673   0.01868543 0.0200958  0.02632781 0.02251762
  0.01343732 0.00799959 0.00301151 0.00642072 0.00819884 0.00260903
  0.00956126 0.01408852 0.01026803 0.01918237]]
```

I chose to have 52 parameters for my model by inputting a feature vector for each weekday in a year. The idea behind this was to tackle the yearly seasonality of the data so that the model had values back to approximately the same day of the week last year.

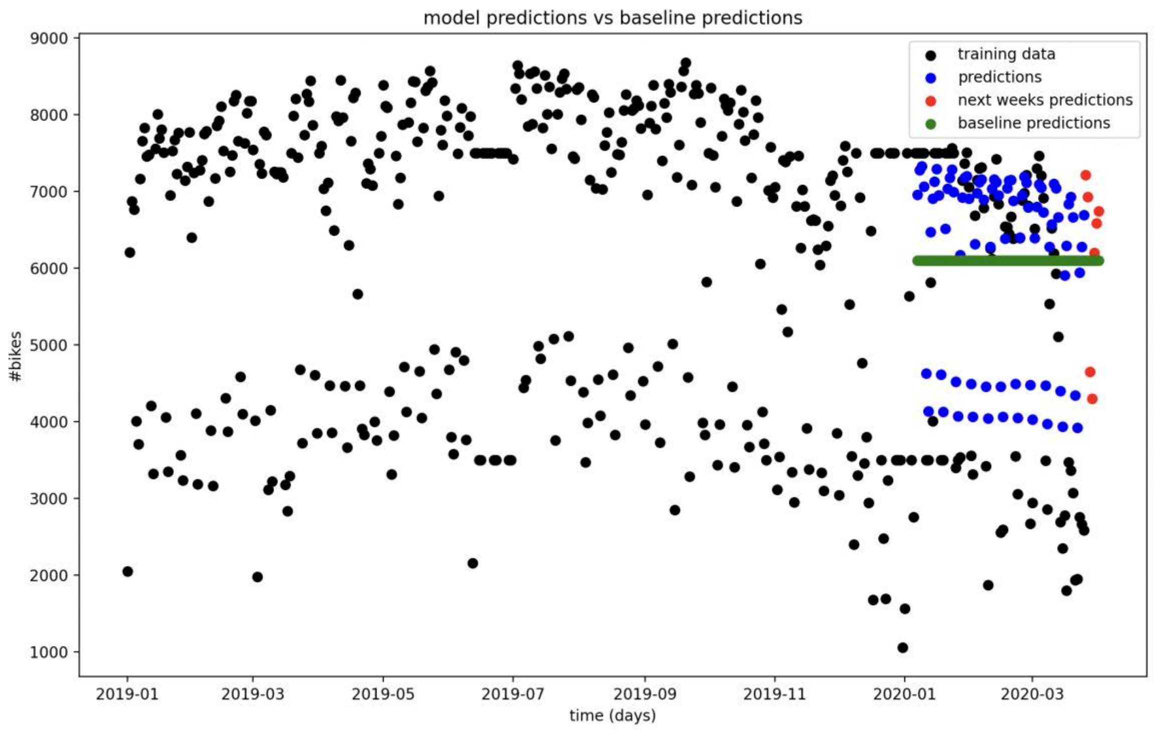
Coefficcient values on the final trained model



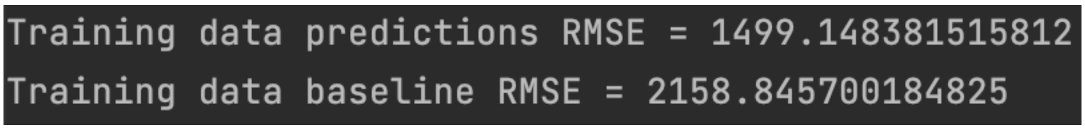
```
0.0 [[ 2.88764010e-02  1.26508006e-02  1.32585933e-02  2.52650360e-02
   2.23578124e-02  2.58931997e-02  2.12222746e-02  2.18443080e-02
   1.94847041e-02  6.86410711e-03  1.50650256e-02  3.41936615e-02
   3.53820128e-02  4.01641088e-02  2.21697956e-02  2.22966684e-02
   9.75052655e-03  7.53577008e-03  4.94584471e-03  8.48428158e-03
   8.28091496e-03  1.05424105e-02  4.83636109e-03  3.35442870e-03
   1.77934779e-03 -2.59678211e-03 -3.39811044e-03 -9.26833228e-03
   2.68114445e-03  9.46091496e-04  2.74004126e-03 -3.20360091e-05
   1.85633519e-02  1.50016942e-02  1.83713963e-02  1.93636415e-02
   2.85546596e-02  1.45513551e-02  2.86871578e-02  5.05046613e-02
   5.31367403e-02  6.13881055e-02  3.63826804e-02  3.56252517e-02
   1.35861574e-02  2.86787203e-02  1.48276141e-02  2.11474964e-02
   4.13738403e-02  4.50659390e-02  2.81257957e-02  5.11979190e-02]]
```

As I kept retraining my model ridge regression made my 52 parameters more conservative over time as the data began to average itself out the further I predicted into the future.

Model predictions and baseline after 1 week if no pandemic happened

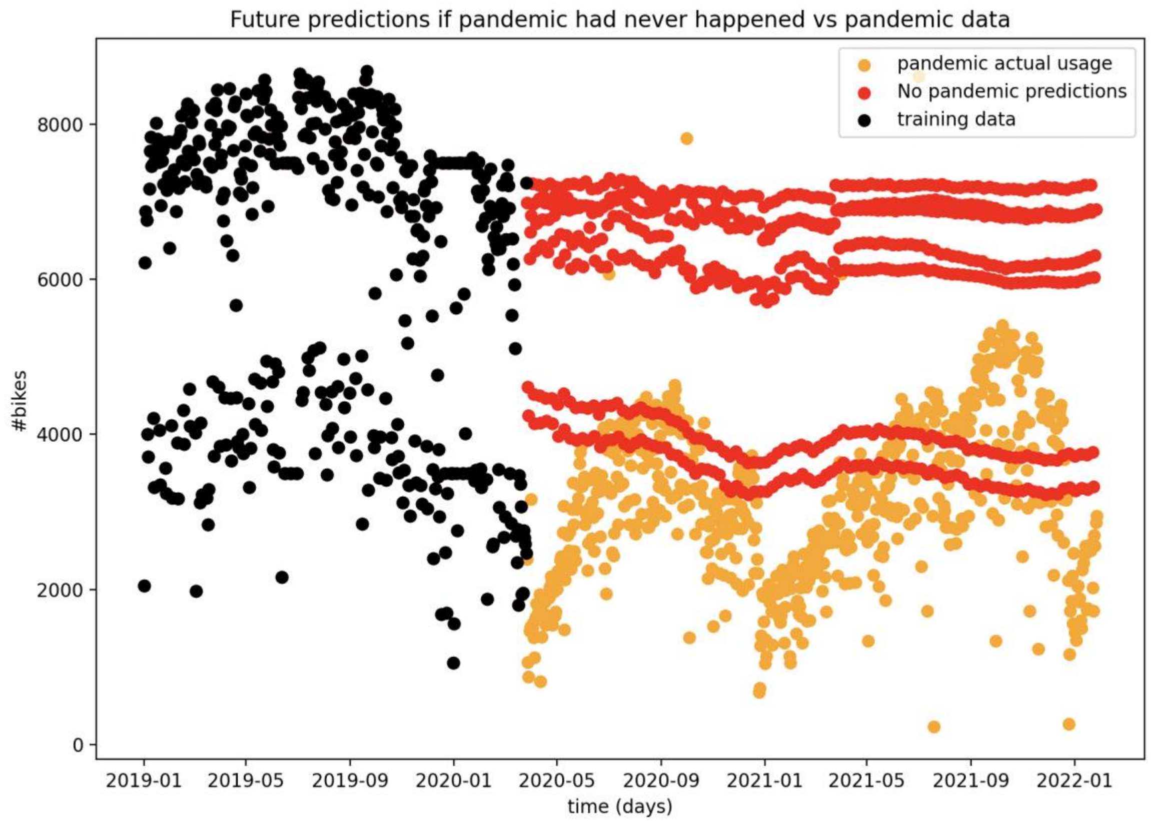


We can visually see above that the model fared better than the baseline that only predicted the prepandemic average value. Difficulties lied with not separating the weekdays from the weekends as we can see two distinct separate trends in the predictions. Other issues include missing data that I had to fill in with average values in the training data as indicated above as series of horizontal black lines.



```
Training data predictions RMSE = 1499.148381515812
Training data baseline RMSE = 2158.845700184825
```

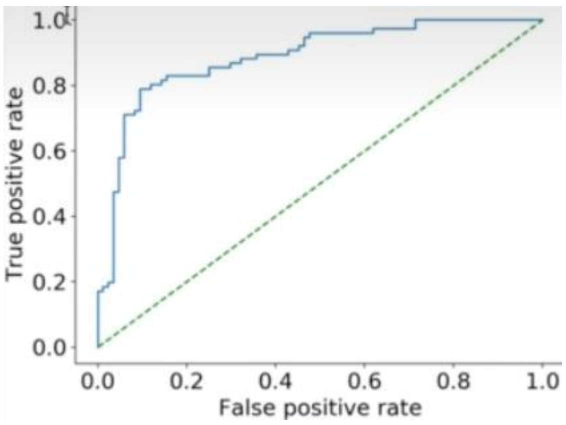
If we consider the performance of our model vs the baseline predictor by incorporating a root mean square error we can see that our training data predictions have a smaller root mean square error. Therefore we can say that our models performance is superior to that of the baseline predictor and that our model is doing something useful.



By expanding out our feature predictions we can identify that our predictions have identified the difference between the days of the week with seven separate lines approximating the usage on that given day. We can see that the weekday predictions approximate the training data. The weekend predictions start off much higher than anticipated but level out over time, this could be to having gaps in the training data close to the start of the pandemic identified by the horizontal black lines in January. Also some annual seasonality was identified by the bumps in the graphs. We can see from the final coefficient values that as the model progressed less weight was put on the values of the parameters. While this worked for the current time period I the data might plateau for future predictions going into 2022.

(i) What is a ROC curve? How can it be used to evaluate the performance of a classifier compared with a baseline classifier? Why would you use an ROC curve instead of a classification accuracy metric?

An ROC curve is a tool we can use to evaluate the performance of our model by comparing performance against other factors. It considers the tradeoff between true and false positives of our model by plotting the decision probability. Most classifiers output a confidence intervals between 0 and one, we can map our parameters to a function that will compute these confidence values. By changing our confidence value we can change the tradeoff between true and false positives which can be mapped on our ROC curve. We can observe our ROC curve to find out our false and true positive rate for each point in our classifier. Ideally we would want a true positive rate of 1 and false positive rate of 0. In the worst case we would havew a false positive rate of 1 and true positive rate of 0. We can plot multiple classifiers onto the same graph.



In the example above we can see that our classifier in blue is superior to the baseline classifier. We can tell this as the blue curve is always greater and equal to the baseline which appears to have an even rate between true and false positives. The further the distance from the baseline indicates that our classifier is performing much better than the baseline at predicting. We would use an ROC curve as it allows us to have an easy visual way of determining how our classifier compares against the other.

(ii) Give two examples of situations where a linear regression would give inaccurate predictions. Explain your reasoning and what possible solutions you would adopt in each situation. [5 marks]

Linear regression depends on some form of linear relationship between our parameters and our data. It works best when we have data structured linearly or can be arranged linearly to best create a classifier that can make predictions on how to label the data. If we have data that follows a specific pattern like a repeating cycle linear regression will struggle with this. As a solution I might group the occurring patterns per cycle and try to identify a trend between the grouped patterns.
Linear regression does not offer a penalty term to prevent overfitting of data. This will also cause issues with outliers effecting our data, a solution would be to incorporate a penalty term. We could consider Lasso or Ridge regression to incorporate a penalty term to incorporate L1 or L2 regularization respectively. This will allow us to change the weights on the coefficient values.

(iii) The term 'kernel' has different meanings in SVM and CNN models. Explain the two different meanings. Discuss why and when the use of SVM kernels and CNN kernels is useful, as well as mentioning different types of kernels. [10 marks]

The kernel in a support vector machine is a function that is used to calculate the distance between an input x and a training point xi. Kernelized SVM operates similarly to a kNN model. Kernels in SVM are useful for attaching more weight to training data that are close to the input x than further away points. When x(i) and x two datapoints are close the kernel should return a value close to 1 when they are far apart the kernel should return a value close to 0. The idea is consider these to learn the theta parameter values that minimise an SVM cost function. For example if we had a classifier outputting 0 or 1 and for a given input X if there were a lot of datapoints predicting 1 close to x vs some datapoints predicting 0 further away from X our kernel would weight the nearby points higher similar to how kNN works. These types of kernel can be applied to any type of linear model. The most typically used type of Kernel is the Gaussian kernel.

$$\textit{Gaussian}\ \text{kernel}\ K(x^{(i)}, x) = e^{-\gamma d(x^{(i)}, x)^2}$$

The Kernel in a CNN model is used for computing convolutions. The kernel sets an KxK map that is used to calculate output values given an input array like structure. A kernel can also have a number of channels to account for different features or different dimensionality of input data. Using matrix calculations the values of the input matrix are convolved with the kernel matrix to determine the outputs this is done by applying our kernel to the top left of our input matrix and sliding it through our inputs to calculate our output values. CNN Kernels are useful for identifying features in images, we could use a CNN kernel to identify edges in images for example. Two types of CNN Kernels are hand crafted kernels where we create the kernel values ourselves and learning kernels where we create a cost function and use gradient descent to learn the kernel values by minimising cost.

(iv) In k-fold cross-validation, a dataset is resampled multiple times. What is the idea behind this resampling i.e. why does resampling allow us to evaluate the generalisation performance of a machine learning model. Give a small example to illustrate. Discuss when it is and it is not appropriate to use k-fold cross-validation. [5 marks]

We resample our data so that we can get different subsets of training data to train our model giving us models that have slightly different parameters and will give different predictions. We can use this as a basis to test if our model will give similar or vastly different outputs based of variations of training data. This gives us an idea of how strong or weak our model is to different datasets. Resampling the training data should keep identifiable trends in the data relatively similar to allow us to compare models more easily.
Lets say we are performing a 5-fold cross validation of our model on our data. We could split our data into five equal parts. In case 1 we would use our testing data as the first part and use the rest as training data. In case 2 we can choose the second part to be training data and the rest to be testing data. We could continue this process for the 5 folds. This should give us different values of our parameters that we can compare how our models compare with the different sets of training and testing data. It would be inappropriate to use K-fold cross validation on very small datasets. We want to use as much data as possible to train our models, if we had a very small dataset we would run into the issue of having a poor representation of our different trained models. Ideally for a dataset of n elements we want to set k such that (k-1)/n is large to reduce fluctuations. A small number of fluctuations in a small dataset would upset this.

Resources

https://stackoverflow.com/questions/19463985/pandas-drop-consecutive-duplicates

Code:

Data cleaning.py (does not need to be run)

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime

# Read CSV
#df1 = pd.read_csv("2019Q1.csv", delimiter=',')
#df2 = pd.read_csv("2019Q2.csv", delimiter=',')
#df3 = pd.read_csv("2019Q3.csv", delimiter=',')
#df4 = pd.read_csv("2019Q4.csv", delimiter=',')

#data2019 = pd.concat([df1,df2,df3,df4])
#del data2019['LAST UPDATED']
#del data2019['STATUS']
#del data2019['LATITUDE']
#del data2019['LONGITUDE']
#del data2019['ADDRESS']
#del data2019['NAME']
#data2019.to_csv('2019Full.csv')
'''
KEEP ALL OF ABOVE FOR PREPROCESSING!!!
'''

df2019 = pd.read_csv("2019Full.csv")

new_df2019 = df2019[["STATION ID", "TIME", "AVAILABLE BIKE STANDS",
"AVAILABLE BIKES"]]

new_df2019['TIME'] = pd.to_datetime(new_df2019['TIME'], format='%Y-%m-%d
%H:%M:%S')




# Get weekday and date
new_df2019['DAY'] = new_df2019['TIME'].dt.day_name()
new_df2019['DATE'] = new_df2019['TIME'].dt.date

#remove conecutive duplicate values
#https://stackoverflow.com/questions/19463985/pandas-drop-consecutive-
duplicates
df = new_df2019[new_df2019["AVAILABLE BIKES"] != new_df2019["AVAILABLE
BIKES"].shift(-1)].dropna()
df = df.reset_index(drop=True)

date_pairs = {}
unique_dates = df['DATE'].unique()

for date in unique_dates:
    temp_df = df[(df['DATE'] == date)]
    diff = 0
    for index in range(1,len(temp_df)):
        last_val = temp_df["AVAILABLE BIKES"].iloc[index-1]
        current_val = temp_df["AVAILABLE BIKES"].iloc[index]
        if temp_df["STATION ID"].iloc[index-1] == temp_df["STATION
ID"].iloc[index]:
            diff = diff + abs(last_val - current_val)
    # avoid double counting bikes taken and returned as separate journeys
    diff = diff/2
    date_pairs[date] = diff

# Create dataframe of dict values
df2019_data = pd.DataFrame(list(date_pairs.items()), columns=['Date', 'Bike
Usage'])
df2019_data.to_csv('2019data.csv')





# 2020 Data SAVED IN 2022data.csv NO NEED TO RERUN


# Read CSV
df1 = pd.read_csv("2020Q1.csv", delimiter=',')
df2 = pd.read_csv("2020Q2.csv", delimiter=',')
df3 = pd.read_csv("2020Q3.csv", delimiter=',')
df4 = pd.read_csv("2020Q4.csv", delimiter=',')

data2020 = pd.concat([df1,df2,df3,df4])
del data2020['LAST UPDATED']
del data2020['STATUS']
del data2020['LATITUDE']
del data2020['LONGITUDE']
del data2020['ADDRESS']
del data2020['NAME']
data2020.to_csv('2020Full.csv')
'''
KEEP ALL OF ABOVE FOR PREPROCESSING!!!
'''
df2020 = pd.read_csv("2020Full.csv")

new_df2020 = df2020[["STATION ID", "TIME", "AVAILABLE BIKE STANDS",
"AVAILABLE BIKES"]]

new_df2020['TIME'] = pd.to_datetime(new_df2020['TIME'], format='%Y-%m-%d
%H:%M:%S')


# Get weekday and date
new_df2020['DAY'] = new_df2020['TIME'].dt.day_name()
new_df2020['DATE'] = new_df2020['TIME'].dt.date

#remove conecutive duplicate values
#https://stackoverflow.com/questions/19463985/pandas-drop-consecutive-
duplicates
df = new_df2020[new_df2020["AVAILABLE BIKES"] != new_df2020["AVAILABLE
BIKES"].shift(-1)].dropna()
df = df.reset_index(drop=True)

date_pairs = {}
unique_dates = df['DATE'].unique()

for date in unique_dates:
    temp_df = df[(df['DATE'] == date)]
    diff = 0
    for index in range(1,len(temp_df)):
        last_val = temp_df["AVAILABLE BIKES"].iloc[index-1]
        current_val = temp_df["AVAILABLE BIKES"].iloc[index]
        if temp_df["STATION ID"].iloc[index-1] == temp_df["STATION
```

```python
ID"].iloc[index]:
            diff = diff + abs(last_val - current_val)
    # avoid double counting bikes taken and returned as separate journeys
    diff = diff/2
    date_pairs[date] = diff

# Create dataframe of dict values
df2020_data = pd.DataFrame(list(date_pairs.items()), columns=['Date', 'Bike
Usage'])
df2020_data.to_csv('2020data.csv')






# 2021 SAVED IN 2021data.csv NO NEED TO RERUN

# Read CSV
#df1 = pd.read_csv("2021Q1.csv", delimiter=',')
#df2 = pd.read_csv("2021Q2.csv", delimiter=',')
#df3 = pd.read_csv("2021Q3.csv", delimiter=',')
#df4 = pd.read_csv("2021Q4.csv", delimiter=',')

#data2021 = pd.concat([df1,df2,df3,df4])
#del data2021['LAST UPDATED']
#del data2021['STATUS']
#del data2021['LATITUDE']
#del data2021['LONGITUDE']
#del data2021['ADDRESS']
#del data2021['NAME']
#data2021.to_csv('2021Full.csv')
'''
KEEP ALL OF ABOVE FOR PREPROCESSING!!!
'''

df2021 = pd.read_csv("2021Full.csv")

new_df2021 = df2021[["STATION ID", "TIME", "AVAILABLE BIKE STANDS",
"AVAILABLE BIKES"]]

new_df2021['TIME'] = pd.to_datetime(new_df2021['TIME'], format='%Y-%m-%d
%H:%M:%S')


# Get weekday and date
new_df2021['DAY'] = new_df2021['TIME'].dt.day_name()
new_df2021['DATE'] = new_df2021['TIME'].dt.date

#remove conecutive duplicate values
#https://stackoverflow.com/questions/19463985/pandas-drop-consecutive-
duplicates
df = new_df2021[new_df2021["AVAILABLE BIKES"] != new_df2021["AVAILABLE
BIKES"].shift(-1)].dropna()
df = df.reset_index(drop=True)

date_pairs = {}
unique_dates = df['DATE'].unique()

for date in unique_dates:
    temp_df = df[(df['DATE'] == date)]
    diff = 0
    for index in range(1,len(temp_df)):
        last_val = temp_df["AVAILABLE BIKES"].iloc[index-1]
        current_val = temp_df["AVAILABLE BIKES"].iloc[index]
        if temp_df["STATION ID"].iloc[index-1] == temp_df["STATION
ID"].iloc[index]:
            diff = diff + abs(last_val - current_val)
    # avoid double counting bikes taken and returned as separate journeys
    diff = diff/2
    date_pairs[date] = diff

# Create dataframe of dict values
df2021_data = pd.DataFrame(list(date_pairs.items()), columns=['Date', 'Bike
Usage'])
df2021_data.to_csv('2021data.csv')






# 2022 SAVED IN 2022data.csv NO NEED TO RERUN

# Read CSV
#df1 = pd.read_csv("2022Jan.csv", delimiter=',')
#df2 = pd.read_csv("2022Feb.csv", delimiter=',')
#df3 = pd.read_csv("2022Mar.csv", delimiter=',')
#df4 = pd.read_csv("2022Apr.csv", delimiter=',')
#df5 = pd.read_csv("2022May.csv", delimiter=',')
#df6 = pd.read_csv("2022Jun.csv", delimiter=',')
#df7 = pd.read_csv("2022Jul.csv", delimiter=',')
#df8 = pd.read_csv("2022Aug.csv", delimiter=',')
#df9 = pd.read_csv("2022Sept.csv", delimiter=',')
#df10 = pd.read_csv("2022Oct.csv", delimiter=',')
#df11 = pd.read_csv("2022Nov.csv", delimiter=',')
#df12 = pd.read_csv("2022Dec.csv", delimiter=',')

#data2022 = pd.concat([df1,df2,df3,df4,df5,df6,df7,df8,df9,df10,df11,df12])
#del data2022['LAST UPDATED']
#del data2022['STATUS']
#del data2022['LATITUDE']
#del data2022['LONGITUDE']
#del data2022['ADDRESS']
#del data2022['NAME']
#data2022.to_csv('2022Full.csv')
'''
KEEP ALL OF ABOVE FOR PREPROCESSING!!!
'''

df2022 = pd.read_csv("2022Full.csv")
#df2022 = pd.read_csv("2022debug.csv")

new_df2022 = df2022[["STATION ID", "TIME", "AVAILABLE_BIKE_STANDS",
"AVAILABLE_BIKES"]]

new_df2022['TIME'] = pd.to_datetime(new_df2022['TIME'], format='%Y-%m-%d
%H:%M:%S')


# Get weekday and date
new_df2022['DAY'] = new_df2022['TIME'].dt.day_name()
```

```
new_df2022['DATE'] = new_df2022['TIME'].dt.date

#remove conecutive duplicate values
#https://stackoverflow.com/questions/19463985/pandas-drop-consecutive-
duplicates
df = new_df2022[new_df2022["AVAILABLE_BIKES"] !=
new_df2022["AVAILABLE_BIKES"].shift(-1)].dropna()
df = df.reset_index(drop=True)




date_pairs = {}
unique_dates = df['DATE'].unique()
unique_times = df['TIME'].unique()

# same as before but we sort
for date in unique_dates:
    temp_df = df[(df['DATE'] == date)]
    temp_df = temp_df.sort_values(['STATION ID', 'TIME'])
    # Remove duplicates
    temp_df = temp_df.reset_index(drop=True)
    temp_df = temp_df[temp_df["AVAILABLE_BIKES"] !=
temp_df["AVAILABLE_BIKES"].shift(-1)].dropna()
    temp_df = temp_df.reset_index(drop=True)
    diff = 0
    for index in range(1,len(temp_df)):
        last_val = temp_df["AVAILABLE_BIKES"].iloc[index-1]
        current_val = temp_df["AVAILABLE_BIKES"].iloc[index]
        # If last station ID == Station ID
        if temp_df["STATION ID"].iloc[index-1] == temp_df["STATION
ID"].iloc[index]:
            diff = diff + abs(last_val - current_val)
    # avoid double counting bikes taken and returned as separate journeys
    diff = diff/2
    date_pairs[date] = diff

# Create dataframe of dict values
df2022_data = pd.DataFrame(list(date_pairs.items()), columns=['Date', 'Bike
Usage'])
df2022_data.to_csv('2022data.csv')


# Concatenate into one final dataset

df1 = pd.read_csv("2019data.csv", delimiter=',')
df2 = pd.read_csv("2020data.csv", delimiter=',')
df3 = pd.read_csv("2021data.csv", delimiter=',')
df4 = pd.read_csv("2022data.csv", delimiter=',')

bike_usage_total = pd.concat([df1,df2,df3,df4])
bike_usage_total = bike_usage_total.reset_index(drop=True)
bike_usage_total.to_csv('full_cleaned_data_merged.csv')


# Plot dataset
# Read CSV
df = pd.read_csv("full_cleaned_data_merged.csv")
# Drop unnamed columns
df.drop(df.columns[df.columns.str.contains('unnamed',case = False)],axis =
1, inplace = True)
```

```python
# Drop duplicate dates
df = df.drop_duplicates(subset='Date')
print(df.head())
# Drop extreme outlier values
df = df.drop(df[df['Bike Usage'] < 100.0].index)
df = df.drop(df[df['Bike Usage'] > 9000.0].index)
df = df.reset_index(drop=True)
# Pandemic times
pand_start = pd.to_datetime("27-03-2020")
pand_end = pd.to_datetime("28-01-2022")

# add day of the week feature
weekdays = {
    'Monday': 1,
    'Tuesday': 2,
    'Wednesday': 3,
    'Thursday': 4,
    'Friday': 5,
    'Saturday': 6,
    'Sunday': 7
}

# For each value calculate weekdays value
for index in range(0, len(df)):
    datetimeString = df['Date'].iloc[index]
    datetimeObject = datetime.strptime(datetimeString, '%Y-%m-%d')
    day = weekdays.get(datetimeObject.strftime('%A'))
    df.at[index, 'Day'] = int(day)


# Calculate pandemic values
df['Date'] = pd.to_datetime(df['Date'])
pre_pandemic_df = df[df['Date'] < pand_start]
pandemic_df = df[(df['Date'] < pand_end) & (df['Date'] >= pand_start)]
post_pandemic_df = df[df['Date'] >= pand_end]

pre_pandemic_df.to_csv('PrePandemicdata.csv')
pandemic_df.to_csv('Pandemicdata.csv')
post_pandemic_df.to_csv('PostPandemicdata.csv')

print('Done')
```

FinalAssignment20334565.py

```python
import numpy as np
import pandas as pd
import time
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from datetime import datetime, date, timedelta
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import math
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error

weekdays = {
    'Monday': 1,
    'Tuesday': 2,
    'Wednesday': 3,
    'Thursday': 4,
    'Friday': 5,
    'Saturday': 6,
    'Sunday': 7
}

# Plot dataset
# Read CSV
pre_pand_df = pd.read_csv('PrePandemicdata.csv')
pand_df = pd.read_csv('Pandemicdata.csv')
post_pand_df = pd.read_csv('PostPandemicdata.csv')


# Calculate pandemic values
pand_start = pd.to_datetime("27-03-2020")
pand_end = pd.to_datetime("28-01-2022")
pre_pandemic_X = pd.to_datetime(pre_pand_df['Date'])
post_pandemic_X = pd.to_datetime(post_pand_df['Date'])
pandemic_X = pd.to_datetime(pand_df['Date'])
pre_pandemic_y = pre_pand_df['Bike Usage']
post_pandemic_y = post_pand_df['Bike Usage']
pandemic_y = pand_df['Bike Usage']

# Missing Data filler fill in missing values with yearly average
def fill_missing_data(df):
    usage = df.get("Bike Usage")
    total = usage.sum()
    dataset_avg = total/len(df)
    dates = pd.to_datetime(df['Date'])
    filled_dataset = pd.DataFrame()
    temp_dataset = pd.DataFrame()
    last_date = dates.iloc[0]
    last_start = 0
    for index in range(1,len(df)):
        current_date = dates.iloc[index]
        # If missing date
        if (current_date - last_date) != timedelta(days=1):
            filled_dataset = pd.concat([filled_dataset,
df[last_start:index]], ignore_index=True)
            diff = current_date - last_date
            last_recorded_date = last_date
            while diff > timedelta(days=1):
                last_recorded_date = last_recorded_date + timedelta(days=1)
                is_weekend = last_recorded_date.dayofweek > 4
                if is_weekend:
                    data = {'Date': [last_recorded_date], 'Bike Usage':
[3500], 'Day': [0]}
                else:
                    data = {'Date': [last_recorded_date], 'Bike Usage':
[7500], 'Day': [0]}
                temp_dataset = pd.DataFrame.from_dict(data)
                filled_dataset = pd.concat([filled_dataset, temp_dataset],
ignore_index=True)
                diff = diff - timedelta(days=1)
            last_start = index
        last_date = current_date
    filled_dataset = pd.concat([filled_dataset, df[last_start:index]],
ignore_index=True)
```

```python
    return filled_dataset, dataset_avg

pre_pand_df_filled, pre_pand_dataset_avg = fill_missing_data(pre_pand_df)
pand_df_filled, pand_dataset_avg = fill_missing_data(pand_df)



# UNCOMMENT WHEN FINISHED !!!
# Plot data
plt.rc('font', size=12)
plt.plot(pandemic_X, pandemic_y, color='red', label='Pandemic')
plt.plot(pre_pandemic_X,pre_pandemic_y, color='green', label='Pre-
Pandemic')
plt.plot(post_pandemic_X,post_pandemic_y, color='blue', label='Post-
Pandemic')
plt.xlabel("Date")
plt.ylabel("Number of Bikes used")
plt.title('Dublin bike usage 2019-2023')
plt.legend(loc='upper left')
plt.show()
plt.clf()


# Assess what bike usage might have been for the pandemic period if the
pandemic had not happened
# split into training and testing data
# q = n step ahead prediction
# dd = number of samples per unit of time
# lag = number of features we want
# y = data points for time
# t = time in required units
# dt = difference in time

def test_preds(q,dd,lag,plot,y, t, dt, title, Kfold_ridge):
    #q-step ahead prediction
    stride=1
    XX=y[0:y.size-q-lag*dd:stride]
    # computes the time series feature values for each datapoint
    for i in range(1,lag):
        X=y[i*dd:y.size-q-(lag-i)*dd:stride]
        XX=np.column_stack((XX,X))
    yy=y[lag*dd+q::stride]; tt=t[lag*dd+q::stride]
    # reset index so data is alligned
    yy = yy.reset_index(drop=True)
    tt = tt.reset_index(drop=True)

    train, test = train_test_split(np.arange(0,yy.size),test_size=0.2)
    model = Ridge(fit_intercept=False).fit(XX[train], yy[train])
    print(model.intercept_, model.coef_)
    if plot:
        y_pred = model.predict(XX)
        plt.scatter(t, y, color='black', label= 'training data')
        plt.scatter(tt, y_pred, color='blue', label= 'predictions')
        plt.xlabel("time (days)"); plt.ylabel("#bikes")
        plt.legend(["training data","predictions"],loc='upper right')
        plt.title(title)
        plt.show()
        plt.clf()
    if Kfold_ridge:
        mean_error = [];
        std_error = []
        Ci_range = [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000,
10000]
        ###
        scaler = StandardScaler()
        XX = scaler.fit_transform(XX)
        yy = yy.to_numpy()
        yy = yy.reshape(-1, 1)
        yy = scaler.fit_transform(yy)
        ###
        for Ci in Ci_range:
            model = Ridge(alpha=1 / (2 * Ci))
            temp = []
            kf = KFold(n_splits=5)
            for train, test in kf.split(X):


                model.fit(XX[train], yy[train])
                ypred = model.predict(XX[test])
                from sklearn.metrics import mean_squared_error
                tmp = y[test]
                temp.append(mean_squared_error(yy[test], ypred))
            mean_error.append(np.array(temp).mean())
            std_error.append(np.array(temp).std())

        plt.errorbar(Ci_range, mean_error, yerr=std_error)
        plt.xscale('log')
        plt.xlabel('Ci');
        plt.ylabel('Mean square error')
        plt.title("5-fold Ridge log C values (0.00001-10000)")
        plt.show()
        plt.clf()

# pre pandemic
y = pre_pandemic_y
t = pre_pandemic_X
dt = 86400 #1 day interval (seconds)
# prediction using short-term trend
plot=True
# prediction using daily seasonality
d=math.floor(24*60*60/dt) # number of samples per day
# prediction using weekly seasonality

# Test PREDS FIRST
w=math.floor(7*24*60*60/dt) # number of samples per day
test_preds(q=1,dd=1,lag=3,plot=plot, y=y, t=t, dt=dt, title="1 step ahead
predictions pre pandemic", Kfold_ridge=True)
test_preds(q=w,dd=w,lag=3,plot=plot, y=y, t=t, dt=dt, title="7 steps ahead
predictions pre pandemic", Kfold_ridge=True)


# Predict values for the pandemic period
"""
def predict_future(q,dd,lag,plot,y, t, dt, title, Kfold_ridge):
    #q-step ahead prediction
    stride=1
    y_cpy = y   # REMOVE AT END
    XX=y[0:y.size-q-lag*dd:stride]
    # computes the time series feature values for each datapoint
    for i in range(1,lag):
        X=y[i*dd:y.size-q-(lag-i)*dd:stride]
        XX=np.column_stack((XX,X))
    yy=y[lag*dd+q::stride]; tt=t[lag*dd+q::stride]
    # reset index so data is alligned
    yy = yy.reset_index(drop=True)
    tt = tt.reset_index(drop=True)

    ### Fit to smaller scale
    scaler = StandardScaler()
    XX = scaler.fit_transform(XX)
    yy = yy.to_numpy()
    yy = yy.reshape(-1, 1)
    yy = scaler.fit_transform(yy)
    y = y.to_numpy()
    y = y.reshape(-1, 1)
    y = scaler.fit_transform(y)

    ###

    train, test = train_test_split(np.arange(0,yy.size),test_size=0.2)
    model = Ridge(fit_intercept=False, alpha=1 / (2 *
0.001)).fit(XX[train], yy[train])
    print(model.intercept_, model.coef_)

    # for the length of the week
    next_week = pd.Series(index=np.arange(7))
    last_date = t.iloc[-1]
    for index in range(0,7):
        last_date = last_date + timedelta(days=1)
        next_week.iloc[index] = last_date

   # next_week = pd.to_datetime(next_week)

    #XXT = y[y.size - q - lag * dd:]
    #XXT = y[-((1 * dd) + q):]
    XXT = y[-dd:]
    for i in range(1, lag):
        X2 = y[-((i * dd) + q):-((i * dd))]
        XXT = np.column_stack((XXT, X2))

    y_pred_extra = model.predict(XXT)
    y_pred = model.predict(XX)

    # Convert back to correct scale
    y_pred = scaler.inverse_transform(y_pred.reshape(-1, 1)).ravel()
    y_pred_extra = scaler.inverse_transform(y_pred_extra.reshape(-1,
1)).ravel()
    y = scaler.inverse_transform(y.reshape(-1, 1)).ravel()
    y =  pd.Series(y)
    y_pred = pd.Series(y_pred)
    #

    y_pred_extra = pd.Series(y_pred_extra)
    y_out = pd.concat([y,y_pred_extra], ignore_index=True)

    new_t = pd.concat([t,next_week], ignore_index=True)

    if plot:
        plt.scatter(t, y, color='black', label= 'training data')
        plt.scatter(tt, y_pred, color='blue', label= 'predictions')
        plt.scatter(next_week, y_pred_extra, color='red', label= 'next week
predictions')
        plt.xlabel("time (days)"); plt.ylabel("#bikes")
        plt.legend(["training data","predictions","next weeks
predictions"],loc='upper right')
        plt.title(title)
        plt.show()

        if Kfold_ridge:
            mean_error = [];
            std_error = []
            Ci_range = [0.000000000001, 0.00001, 0.0001, 0.001, 0.01, 0.1,
1, 10, 100, 1000, 10000]
            ###
            scaler = StandardScaler()
            XX = scaler.fit_transform(XX)
            yy = yy.to_numpy()
            yy = yy.reshape(-1, 1)
            yy = scaler.fit_transform(yy)
            ###
            for Ci in Ci_range:
                model = Ridge(alpha=1 / (2 * Ci))
                temp = []
                kf = KFold(n_splits=5)
                for train, test in kf.split(X):
                    model.fit(XX[train], yy[train])
                    ypred = model.predict(XX[test])
                    from sklearn.metrics import mean_squared_error
                    tmp = y[test]
                    temp.append(mean_squared_error(yy[test], ypred))
                mean_error.append(np.array(temp).mean())
                std_error.append(np.array(temp).std())

                ypred_original_format =
scaler.inverse_transform(ypred.reshape(-1, 1)).ravel()  # REMOVE THIS AT
END!!!

            plt.errorbar(Ci_range, mean_error, yerr=std_error)
            plt.xscale('log')
            plt.xlabel('Ci');
            plt.ylabel('Mean square error')
            plt.title("5-fold Ridge log C values (0.00001-10000)")
            plt.show()
            plt.clf()

    return (y_out, new_t) #!!!
"""

def predict_future(q,dd,lag,plot,y, t, dt, title, Kfold_ridge):
    #q-step ahead prediction
    stride=1
    y_cpy = y   # REMOVE AT END
    XX=y[0:y.size-q-lag*dd:stride]
    # computes the time series feature values for each datapoint
    for i in range(1,lag):
        X=y[i*dd:y.size-q-(lag-i)*dd:stride]
        XX=np.column_stack((XX,X))
    yy=y[lag*dd+q::stride]; tt=t[lag*dd+q::stride]
    # reset index so data is alligned
    yy = yy.reset_index(drop=True)
    tt = tt.reset_index(drop=True)
    yy_compare = yy

    ### Fit to smaller scale
    scaler = StandardScaler()
    XX = scaler.fit_transform(XX)
    yy = yy.to_numpy()
    yy = yy.reshape(-1, 1)
    yy = scaler.fit_transform(yy)
    y = y.to_numpy()
    y = y.reshape(-1, 1)
    y = scaler.fit_transform(y)
    ###

    train, test = train_test_split(np.arange(0,yy.size),test_size=0.2)
    model = Ridge(fit_intercept=False, alpha=1 / (2 *
0.001)).fit(XX[train], yy[train])
    print(model.intercept_, model.coef_)

    # for the length of the week
    next_week = pd.Series(index=np.arange(7))
    last_date = t.iloc[-1]
    for index in range(0,7):
        last_date = last_date + timedelta(days=1)
        next_week.iloc[index] = last_date

    XXT = y[-dd:]
    for i in range(1, lag):
        X2 = y[-((i * dd) + q):-((i * dd))]
        XXT = np.column_stack((XXT, X2))

    y_pred_extra = model.predict(XXT)
    y_pred = model.predict(XX)


    # Convert back to correct scale
    y_pred = scaler.inverse_transform(y_pred.reshape(-1, 1)).ravel()
    y_pred_extra = scaler.inverse_transform(y_pred_extra.reshape(-1,
1)).ravel()
    y = scaler.inverse_transform(y.reshape(-1, 1)).ravel()
    y =  pd.Series(y)
    y_pred = pd.Series(y_pred)
    #

    y_pred_extra = pd.Series(y_pred_extra)
    y_out = pd.concat([y,y_pred_extra], ignore_index=True)

    new_t = pd.concat([t,next_week], ignore_index=True)

    if plot:

        # compare with baseline predictor (always predicts the average)
        arr = np.zeros(shape=(len(y_pred_extra), 1))
        for index in range(0, len(arr)):
            arr[index] = pre_pand_dataset_avg
        arr = arr.flatten()
        baseline_pred = pd.Series(arr)
        arr2 = np.zeros(shape=(len(y_pred), 1))
        for index in range(0, len(arr2)):
            arr2[index] = pre_pand_dataset_avg
        arr2 = arr2.flatten()
        baseline_pred2 = pd.Series(arr2)

        plt.scatter(t, y, color='black', label= 'training data')
        plt.scatter(tt, y_pred, color='blue', label= 'predictions')
        plt.scatter(next_week, y_pred_extra, color='red', label= 'next week
predictions')
        plt.scatter(next_week, baseline_pred, color='green',
label='baseline predictions')
        plt.scatter(tt, baseline_pred2, color='green')
        plt.xlabel("time (days)"); plt.ylabel("#bikes")
        plt.legend(["training data","predictions","next weeks predictions",
"baseline predictions"],loc='upper right')
        plt.title("model predictions vs baseline predictions")
        plt.show()
        plt.clf()
        # MSE comparison of plotted data with a simple baseline
        predictions_error = mean_squared_error(yy_compare,y_pred)
        predictions_error = math.sqrt(predictions_error)
        print("Training data predictions RMSE = " + str(predictions_error))
        baseline_error = mean_squared_error(yy_compare, baseline_pred2)
        baseline_error = math.sqrt(baseline_error)

        print("Training data baseline RMSE = " + str(baseline_error))

    if Kfold_ridge:
        mean_error = [];
        std_error = []
        Ci_range = [0.000000000001, 0.00001, 0.0001, 0.001, 0.01, 0.1, 1,
10, 100, 1000, 10000]
        ###
        scaler = StandardScaler()
        XX = scaler.fit_transform(XX)
        #yy = yy.to_numpy()
        yy = yy.reshape(-1, 1)
        yy = scaler.fit_transform(yy)
        ###
        for Ci in Ci_range:
            model = Ridge(alpha=1 / (2 * Ci))
            temp = []
            kf = KFold(n_splits=5)
            for train, test in kf.split(X):
                model.fit(XX[train], yy[train])
                ypred = model.predict(XX[test])
                tmp = y[test]
                temp.append(mean_squared_error(yy[test], ypred))
            mean_error.append(np.array(temp).mean())
            std_error.append(np.array(temp).std())

        plt.errorbar(Ci_range, mean_error, yerr=std_error)
        plt.xscale('log')
        plt.xlabel('Ci');
        plt.ylabel('Mean square error')
        plt.title("5-fold Ridge log C values (0.00001-10000)")
        plt.show()
        plt.clf()


    return (y_out, new_t) #!!!

# Predict next 7 days, loop for pandemic period.
PANDEMIC_LENGTH = len(pand_df)
index = 0
plot = True
original_t = t
original_y = y
```

```
# Calculate pandemic values
pre_pandemic_X = pd.to_datetime(pre_pand_df_filled['Date'])
pre_pandemic_y = pre_pand_df_filled['Bike Usage']
y = pre_pandemic_y
t = pre_pandemic_X

# perform ridge regression on the model to determine C value
predict_future(q=w, dd=w, lag=52, plot=plot, y=y, t=t, dt=dt, title="",
Kfold_ridge=True)

plot = False
while index < PANDEMIC_LENGTH:
    if index == 21:
        print("debug")
    (y, t) = predict_future(q=w, dd=w, lag=52, plot=plot, y=y, t=t, dt=dt,
title="", Kfold_ridge=False)
    index = index + 7

# Plot data
plt.scatter(t, y, color='red', label='No pandemic predictions')
Pre_pandemic_time = t[t< datetime.strptime('27-03-2020', "%d-%m-%Y")]
plt.scatter(t[t< datetime.strptime('27-03-2020', "%d-%m-%Y")],
y[:len(Pre_pandemic_time)], color='black')
plt.scatter(original_t, original_y, color='black', label='training data')

plt.xlabel("time (days)");
plt.ylabel("#bikes")
plt.legend(loc='upper right')
plt.title("Future predictions if pandemic had never happened")
plt.show()
plt.clf()

# Plot comparison of actual pandemic data, predicted data and baseline data
# pandemic data not filled as we do not predict off this data
plt.scatter(pandemic_X, pandemic_y, label='pandemic actual usage',
color='orange')
plt.scatter(t, y, color='red', label='No pandemic predictions')
plt.scatter(t[t< datetime.strptime('27-03-2020', "%d-%m-%Y")],
y[:len(Pre_pandemic_time)], color='black')
plt.scatter(original_t, original_y, color='black', label='training data')

plt.xlabel("time (days)");
plt.ylabel("#bikes")
plt.legend(loc='upper right')
plt.title("Future predictions if pandemic had never happened vs pandemic
data")
plt.show()
```