Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

CSU44061 Machine Learning
Final Project

Adriana Hrabowych, 19304296

January 4, 2024

1 Part 1

1.1 (i) Data Preprocessing

In total there are 38 csv files as part of the dublin bikes dataset, all over 100,000 lines and spanning from mid 2018 to the end of 2023. The files are gone through either chronologically by station or just chronologically. The header for each of these files is:

STATION ID,TIME,LAST UPDATED,NAME,BIKE STANDS,AVAILABLE BIKE STANDS,AVAILABLE BIKES,STATUS,ADDRESS,LATITUDE,LONGITUDE

For these models, 'bike usage' is defined as the number of bikes taken from a station per day. Bikes being put back are ignored as its assumed that 'usage' has been logged previously when the bike was first taken. Before starting any training, files are gone through one by one line by line in order to reduce and refine the data. First the data is separated into different dates, each day is represented by a class that contains its date, an array of stations, an array of station usages, and an array of # of bikes at each station.

```
class combinedStationTime:
  def __init__(self, date, station, bikes):
    self.date = date
    self.stations = [station]
    self.bikes = [bikes]
    self.usages = [0]
```

An array of these 'combinedStationTime's are maintained and each row of data either adds a new day or updates an existing one. If the current row's date already exists in the list, it will then check if that station is already in the list of station, if not it appends the station id, number of available bikes, and a usage of zero to each of the appropriate class lists. If that day already has that station within its list, it will then check if any bikes have been taken and increase the stations usage by 1 for each bike. Once all rows have been gone through, the average usage of all stations is calculated for each individual date.
The end result is a file called 'condenseddata.csv' which has a header of:

DATE,AVG USAGE

When this data is read in and plotted in the main file the output is as shown in figure 1. There are a few gaps in the data, shown by horizontal connections over the gaps, as some dates weren't included in the files at all. There are also clear trends in the data, average usage drops significantly over holidays like new years and Christmas each year. It is obvious to see that usage fell drastically over the pandemic and has not recovered since.

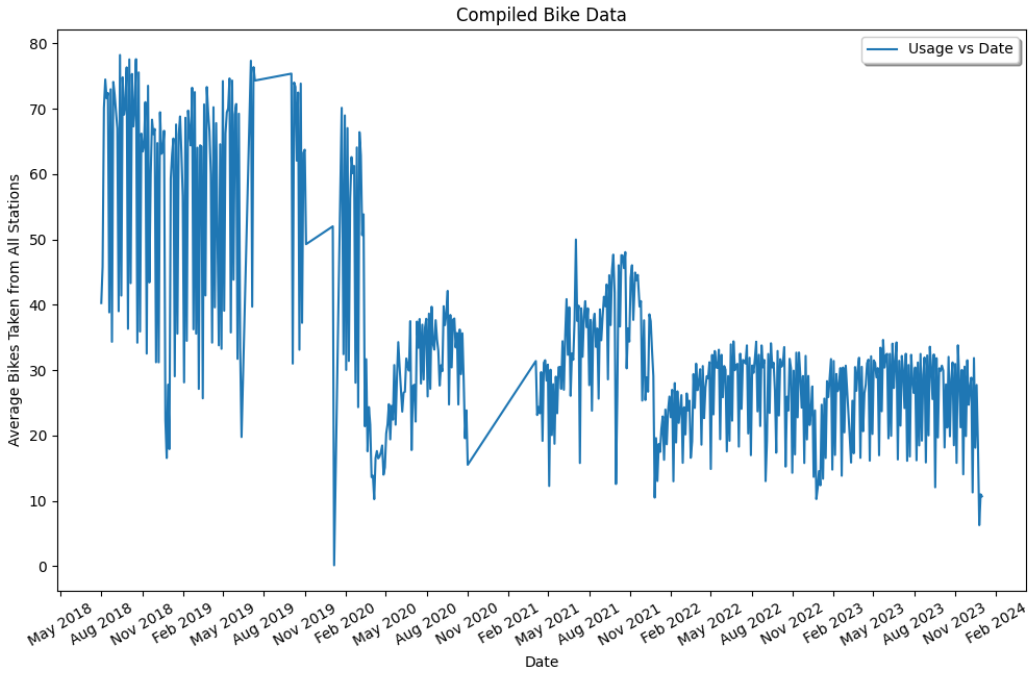


Figure 1: A single line plot representing the average bike usage per day over the course of several years.

1.2 (ii) Machine Learning Methodology and Feature Engineering

In order to see the impact of Covid on bike usage, a machine learning model will be trained only on pre-covid data and will then predict data over covid and postcovid periods. These predictions will be compared to the actual usage over those periods to see the full impact.
In this project a Linear Regression model is used. The first feature engineering that happened was to figure out the best date related variables to use. All of the date variables are in int format. The month and the type of weekday (Monday - Sunday) were the first variables considered as there are trends over months and weekdays with holidays and weekends. The year can be useful as there are trends as years go by, though with only 2 and a bit years of precovid data to train on the usefulness of this variable was unsure. Additionally the day was considered to potentially not be too useful as there are holidays on different days in every month so getting useful information by what day of the month it is would be hard. Four different Linear Regression models were trained and tested: One using weekdays and months (w+m), one with w+m+years, one with w+m+days, and one with w+m+years+days. A dummy model with each set of variables was also trained and tested.
The results were that the r2 scores of the dummy models were all zero as it would just predict the mean value for each input. The r2 scores of the models pre-covid period predictions vs actual pre-covid period predictions were all similar, between 0.32 and 0.34. Although in both the models that used the day variable its coefficient was below 0.06 and so its influence is negligable. Though the inclusion of the year variable did raise the r2 score by 0.02, its inclusion when predicting over a larger course of time causes the model to overly rely on it with a large coefficient. This can be seen in figure two. So from this the only variables we will use for our final model is the month and weekday.
The next bit of feature engineering was to decide on the degree of the polynomial the model should have. This might seem strange as the features are categorical and not continuous but as they are represented by ints they can still be transformed. Additionally as the data is not linear, increasing the degree will help the model predict more accurately. The degrees chosen to test were 1, 2, 5, 7, these were chosen as they represent a good spread and represent the typical range of degrees used in linear regression models.

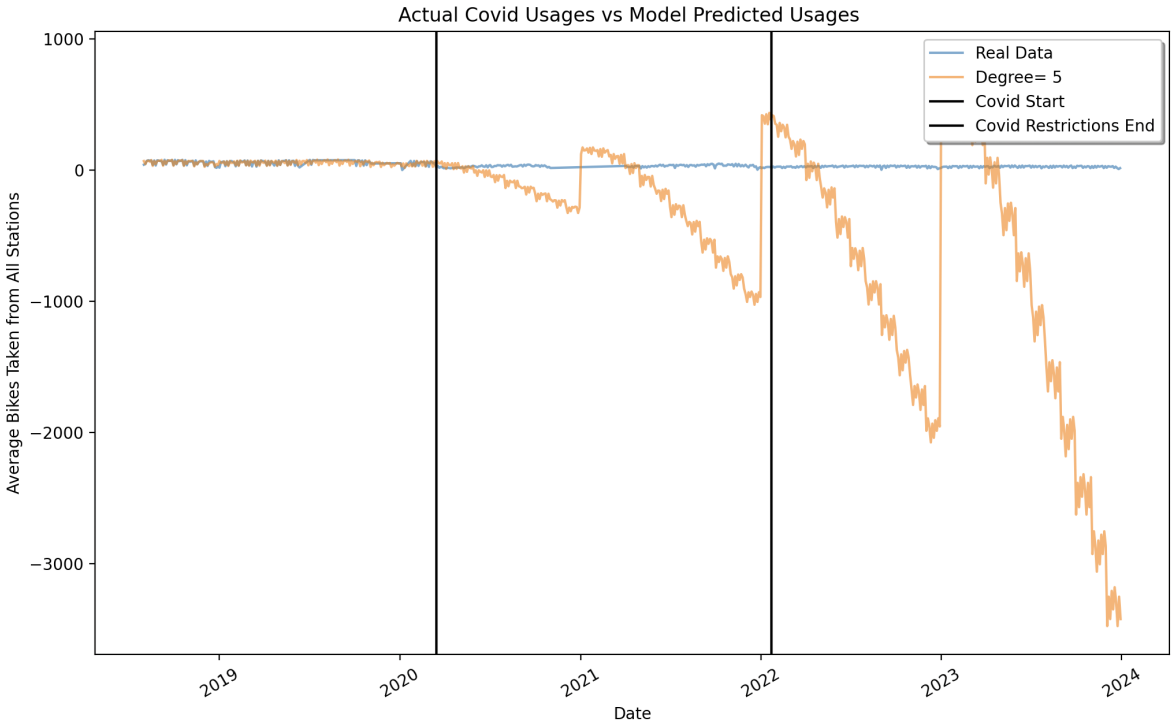


Figure 2: A graph of actual data vs predicted data over the entire time period. The predicted data is very inaccurate.

The process of testing the different polynomial degrees was the same as feature engineering for the different variables, the graphs can be seen in figure 3. The r2 scores of each model were as follows:

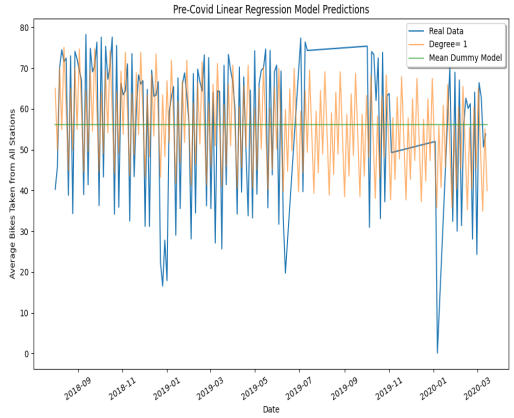
Degree 1 - 0.345
Degree 2 - 0.635
Degree 5 - 0.683
Degree 7 - 0.706

Though the model with a degree of 7 performed the best, a model with that many parameters can be easily overfit, so instead for the final model a degree of five will be used.

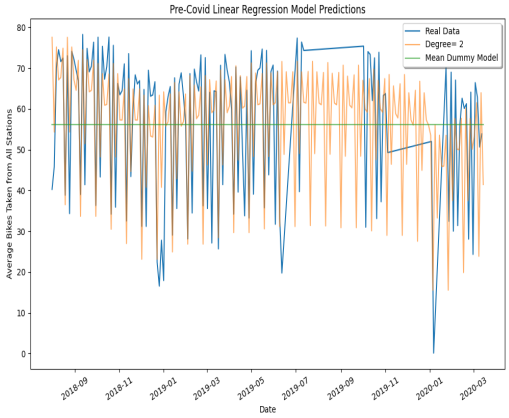
1.3 (iii) Evaluation

The final model was a linear regression model that took in two variables (months and weekdays) to a polynomial degree of 5. This model was trained on pre-covid data and then was told to predict the entire timerange (including covid and post-covid time), this was plotted against the real values. This graph can be seen in figure 4. The model predicts the same values for each year, causing a repeating pattern. It can be seen from the graph that the actual pandemic usages and post pandemic usages were much lower than the predicted values. This shows that the pandemic clearly had a noticeable and still present impact on Dublin Bike usage.

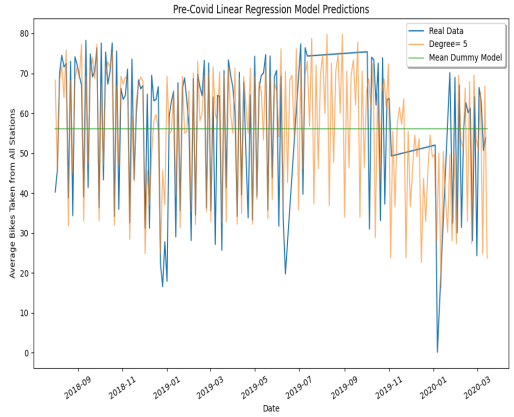
There were many problems with the methodologies used this project. A Linear Regression model was not the best model to choose for this scenario, as it isn't complex enough and the variables are categorial which does not engage well with linear regression. An ARIMA model or any model that can work well with time series data would have been more effective at accurately predicting the usage. Originally, an ARIMA model was going to be used but unfortunately the writer could not get the packages to work on their Mac. Additionally, the models may have been more accurate if more variables were introduced other than time. This was not done due to time constraints.



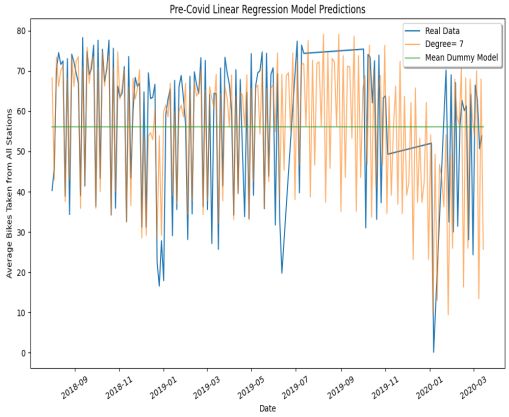
(a) Model with polynomial degree = 1 vs actual.



(b) Model with polynomial degree = 2 vs actual.



(c) Model with polynomial degree = 5 vs actual.



(d) Model with polynomial degree = 7 vs actual.

Figure 3: A figure with four subfigures, each containing two plots. One is the plot of the actual average bike usage over pre covid dates and the other is a model prediction the average bike usage over pre covid dates. Each subfigure differs in the polynomial degree the variables in the models use.

2 Part 2

2.1 (i)

An ROC curve is line that plots the true positive rate of a model against the false positive rate of a model at different classification thresholds. A classification threshold is the threshold that must be reached for a model to classify a point as positive. The formula for each rate are as follows:

$$\text{TPR} = \text{TP} \ / \ (\text{TP} + \text{FN})$$
$$\text{FPR} = \text{FP} \ / \ (\text{FP} + \text{TN})$$

An ROC curve is considered better the closer the line is to an upside down L as it would mean that when the classification threshold is 1 the false positive rate is 0. A perfect model would have this L shape. The ROC curve for a baseline model is usually close to a perfectly diagonal line as the TPR usually equals the FPR. You would want to use an ROC curve over a classification accuracy metric to evaluate your models when its known that the data classifications are unequal in size. If the data is 99 percent class 1, then a baseline model that always predicts class 1 will have a very high accuracy, potentially better than a well fit model. Obviously, using this baseline model isn't good as it will have a lot of false class 1s. Using an ROC

curve to analyze models can make it easier to see which models have the lowest number of false positives and highest true positives.

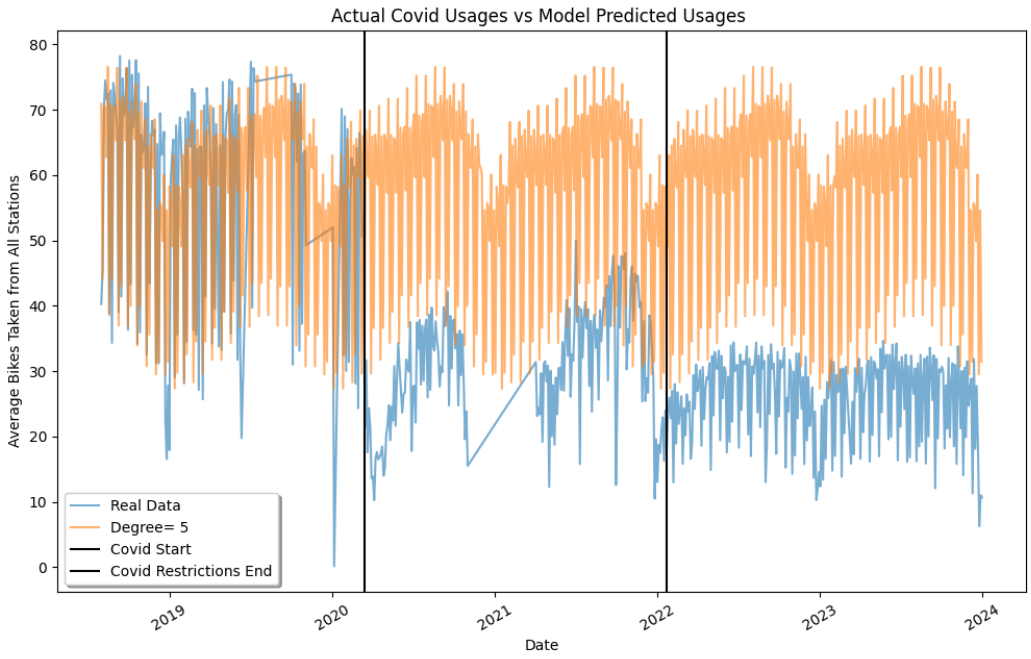


Figure 4: A graph of actual data vs predicted data over the entire time period. The mode used was a linear regression model with a polynomial degree of five

2.2 (ii)

Linear Regression models, as the name suggests, are meant to model linear relationships. If the data has a non-linear relationship (ex. a parabola shape) then a LR model will not be able to capture it accurately and would not give the most accurate predictions. This can be mitigated by increasing the number of features through polynomial transformations of existing features. With a polynomial degree of 2 for the features, the equation changes from a linear to a quadratic equation and can be used in a non-linear dataset.
This solution however can lead to a different problem, where too little data and too many parameters used in a LR model can cause the model to overfit the data and the outliers of the data become exaggerated in predictions. This can be solved in two ways, firstly by carefully selecting the number of parameters so that the model doesn't under or over fit. Secondly, before training normalize the data and potentially remove outliers so that theres less noise the begin with.

2.3 (iii)

Both types of kernels transform the data in some way. The kernel in a kernelized SVM is a function that transforms the training data into different forms depending on whats needed. There are several different types of kernels, the 'kernel trick' kernel for example will transform the data to a higher plane so that non-linear relationships can be represented linearly. In Convolutional Neural Networks a kernel is a filter that is applied to an input matrix to produce an output matrix, a typical CNN will use many different kernels to transform the data over many layers. A kernel can be any size smaller than the input matrix, and different kernel sizes can lead to the size of the input changing in the output. In image processing, kernels can be used to detect edges in an image.

2.4 (iv)

In K-Fold Cross Validation the data is sampled over multiple times, this resampling is done to allow us to evaluate how the model performs generally. This is because if we were to only split and train and test once we'd only get one score, this score might be affected by an odd splitting of training/testing or some other bias. But, in K-fold our final score is the K# of score's from each fold / K, aka the average scores over K folds. As this is an average its easier to trust, even if one those folds got a really good or bad score by chance its evened out by the other scores and so therefore the end score is more representative of the model. For example, if you trained and tested you model and got a score of 0.80 accuracy you would assume the model is good and would use it over other models/hyper parameter combinations. But that score was an outlier, now every time you use the model the score is around 0.40. If you had used K-fold cross validation to begin with, with a K of five in this example, the score for that model would have been (0.4 * 4 + 0.8 / 5 = 0.48) which is quite low and would have given you the chance to explore different models.
It is not appropriate to use K-fold to cross validate when the datasets are very large as K-fold increases the computational time and power to train/test by K. Another scenario where K-fold isn't appropriate is in time-series data (like this project) as the order of the data is important and so the arbitrary K-splits and potentially predicting past dates than what was trained on would cause the scores to be inaccurate.

3 Appendix

```
import datetime
import os.path
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from sklearn import preprocessing
from sklearn.dummy import DummyRegressor
from sklearn.metrics import r2_score
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from readfiles import readFiles

plt.rcParams['figure.figsize'] = [12, 7]

# Check if the condensed data file exists
# If it does, use that data, if not reread all of the csv files and condense them per day

# WARNING this process can take up to two hours!!
# The file already exists, if you want to test the data preposessing you can delete it but expect to be waiting a long time.
if not os.path.exists("condenseddata.csv"):
    readFiles()

# Read in condensed data
df = pd.read_csv("condenseddata.csv")
dates=df.iloc[:,0]
dates = pd.to_datetime(dates)
avgUsages=df.iloc[:,1]

# Remove duplicates, theres just a few as some dates are in two files. They all have the same avgusage for both so its fine to just delete one.
duplicateBA = dates.duplicated(keep="first")
dates = dates[~duplicateBA].reset_index(drop=True)
avgUsages = avgUsages[~duplicateBA].reset_index(drop=True)

## Create and show graphs of the data
fig, ax = plt.subplots(figsize=(12, 7))

ax.xaxis.set_major_locator(mdates.MonthLocator(interval=3))
fmt = mdates.DateFormatter('%b %Y')
ax.xaxis.set_major_formatter(fmt)
plt.plot(dates[::3],avgUsages[::3]) #Plot every 3rd day to be able to read graph
```

```
42
43 plt.xticks(rotation=30)
44 plt.xlabel("Date")
45 plt.ylabel("Average Bikes Taken from All Stations")
46 plt.legend(["Usage vs Date"], loc='upper right',fancybox=True, shadow=True)
47 plt.title("Compiled Bike Data")
48
49 plt.savefig('./Report/CompiledData.png')
50 plt.show()
51
52 ## Reorganize data for training
53 months = []
54 weekDays = []
55 years = []
56 days = []
57 for date in dates:
58     months.append(date.month)
59     weekDays.append(date.day_of_week)
60     days.append(date.day)
61     years.append(date.year)
62 months = pd.Series(months, index = None)
63 weekDays = pd.Series(weekDays, index = None)
64 days = pd.Series(days, index = None)
65 years = pd.Series(years, index = None)
66
67 # Split into only precovid data
68 precovidBA = dates < "2020-03-15"
69 pcDates = dates[precovidBA]
70 pcUsages = avgUsages[precovidBA]
71 pcMonths = months[precovidBA]
72 pcWeekDays = weekDays[precovidBA]
73 pcDays = days[precovidBA]
74 pcYears = years[precovidBA]
75
76 # X's: month, weekday
77 # Y: Usage
78 pcX = np.column_stack((pcWeekDays,pcMonths))
79 pcXY = np.column_stack((pcWeekDays,pcMonths, pcYears))
80 pcXD = np.column_stack((pcDays, pcWeekDays,pcMonths))
81 pcXYD = np.column_stack((pcDays, pcWeekDays,pcMonths, pcYears))
82
83 rangedate = pd.date_range(start="2018-08-01",end="2020-03-15")
84 monthsTest = []
85 weekDaysTest = []
86 daysTest = []
87 yearsTest = []
88 for date in rangedate:
89     monthsTest.append(date.month)
90     weekDaysTest.append(date.day_of_week)
91     daysTest.append(date.day)
92     yearsTest.append(date.year)
93 Xtest = np.column_stack((weekDaysTest,monthsTest))
94 XtestY = np.column_stack((weekDaysTest,monthsTest, yearsTest))
95 XtestD = np.column_stack((daysTest, weekDaysTest,monthsTest))
96 XtestYD = np.column_stack((daysTest, weekDaysTest,monthsTest, yearsTest))
97
98 #Remove dates that arent in training for scoring
99 dateisInTestingBA = rangedate.isin(pcDates)
100
101 #Train and test various models to find best variables
102 pcXs = [pcX, pcXY, pcXD, pcXYD]
103 Xtests = [Xtest, XtestY, XtestD, XtestYD]
104 names = [["Weekdays", "Months"], ["Weekdays", "Months", "Years"], ["Days", "Weekdays", "Months"], ["Days", "
      Weekdays", "Months", "Years"]]
105
106 for i in range(4):
107     #Train and Test
108     modelname = "Variables: "
109     for j in range(len(names[i])):
```

```
        modelname = modelname + names[i][j] + " "
    plt.plot(pcDates[::3], pcUsages[::3])

    model = LinearRegression()
    model.fit(pcXs[i], pcUsages)

    ypred = model.predict(Xtests[i])

    #Dummy model comparison
    dr = DummyRegressor(strategy="mean")
    dr.fit(pcXs[i], pcUsages)
    ydummy = dr.predict(Xtests[i])

    #Plot
    plt.plot(rangedate[::3],ypred[::3], alpha=0.6) #Plot every 3rd day to be able to read graph
    plt.plot(rangedate[::3],ydummy[::3], alpha=0.6) #Plot every 3rd day to be able to read graph

    plt.xticks(rotation=30)
    plt.xlabel("Date")
    plt.ylabel("Average Bikes Taken from All Stations")
    plt.legend(["Real Data", modelname, "Mean Dummy Model"], loc='upper right',fancybox=True, shadow=True)
    plt.title("Pre-Covid Linear Regression Model Predictions different Variables")
    stringname = "variablemodel" + int.__str__(i)
    plt.savefig('./Report/' + stringname)
    plt.show()

    #Score
    print("Linear model" + modelname + ":")
    print(r2_score(pcUsages, ypred[dateisInTestingBA]))
    print("Variable Coefficient")
    for j in range(len(model.coef_)):
        print("Coefficient theta -", names[i][j], ": ", model.coef_[j])

    print("Dummy model" + modelname + ":")
    print(r2_score(pcUsages, ydummy[dateisInTestingBA]))

## Train and test various models to find best degree parameter

degreePoly = [1, 2, 5, 7]
for degree in degreePoly:
    #Train and Test
    plt.plot(pcDates[::3], pcUsages[::3])
    polyX = PolynomialFeatures(degree).fit_transform(np.array(pcXY))

    model = LinearRegression()
    model.fit(polyX, pcUsages)

    polyXTest = PolynomialFeatures(degree).fit_transform(np.array(XtestY))
    ypred = model.predict(polyXTest)

    #Dummy model comparison
    dr = DummyRegressor(strategy="mean")
    dr.fit(polyX, pcUsages)
    ydummy = dr.predict(polyXTest)

    #Plot
    plt.plot(rangedate[::3],ypred[::3], alpha=0.6) #Plot every 3rd day to be able to read graph
    plt.plot(rangedate[::3],ydummy[::3], alpha=0.6) #Plot every 3rd day to be able to read graph

    plt.xticks(rotation=30)
    plt.xlabel("Date")
    plt.ylabel("Average Bikes Taken from All Stations")
    stringdegree = "Degree= " + int.__str__(degree)
    plt.legend(["Real Data", stringdegree, "Mean Dummy Model"], loc='upper right',fancybox=True, shadow=True)
    plt.title("Pre-Covid Linear Regression Model Predictions")
    stringname = "featuremodel" + int.__str__(degree)
    plt.savefig('./Report/' + stringname)
    plt.show()

```

```
    #Score
    print("Linear model (Degree = ", degree, "):")
    print(r2_score(pcUsages, ypred[dateisInTestingBA]))

    print("Mean Dummy model (Degree = ", degree, "):")
    print(r2_score(pcUsages, ydummy[dateisInTestingBA]))

#Best model was the one with degree five
#Use model trained on pre-covid data to predict

#Train

polyX = PolynomialFeatures(5).fit_transform(np.array(pcX))
model = LinearRegression()
model.fit(polyX, pcUsages)

#Create predict X's
rangedate = pd.date_range(start="2018-08-01",end="2024-01-01")
monthsTest = []
weekDaysTest = []
for date in rangedate:
    monthsTest.append(date.month)
    weekDaysTest.append(date.day_of_week)
Xpredict = np.column_stack((weekDaysTest,monthsTest))
polyXp = PolynomialFeatures(5).fit_transform(np.array(Xpredict))

ypred = model.predict(polyXp)

#Plot
plt.plot(dates[::3],avgUsages[::3], alpha=0.6) #Plot every 3rd day to be able to read graph
plt.plot(rangedate[::3],ypred[::3], alpha=0.6) #Plot every 3rd day to be able to read graph
plt.axvline(x = datetime.date(2020, 3, 15), color = 'black')
plt.axvline(x = datetime.date(2022, 1, 22), color = 'black')

plt.xticks(rotation=30)
plt.xlabel("Date")
plt.ylabel("Average Bikes Taken from All Stations")
stringdegree = "Degree= " + int.__str__(5)
plt.legend(["Real Data", stringdegree, "Covid Start", "Covid Restrictions End"], loc='lower left',fancybox=
    True, shadow=True)
plt.title("Actual Covid Usages vs Model Predicted Usages ")
stringname = "covidDif" + int.__str__(5)
plt.savefig('./Report/' + stringname)
plt.show()

#Score
#Remove dates that arent in training for scoring
dateisInTestingBA1 = rangedate.isin(dates)

print("Linear model r2 score")
print(r2_score(avgUsages, ypred[dateisInTestingBA1]))
```

```
import pandas as pd
import os.path
from datetime import datetime

def readFiles():
    filenames = ["csv/yearly/2018-1.csv", "csv/yearly/2018-2.csv", "csv/yearly/2019-1.csv", "csv/yearly
    /2019-2.csv", "csv/yearly/2019-3.csv", "csv/yearly/2019-4.csv", "csv/yearly/2020-1.csv", "csv/yearly
    /2020-2.csv", "csv/yearly/2020-3.csv", "csv/yearly/2020-4.csv",
                "csv/yearly/2021-1.csv", "csv/yearly/2021-2.csv", "csv/yearly/2021-3.csv", "csv/yearly
    /2021-4.csv", "csv/monthly/2022-1.csv", "csv/monthly/2022-2.csv", "csv/monthly/2022-3.csv", "csv/monthly
    /2022-4.csv", "csv/monthly/2022-5.csv", "csv/monthly/2022-6.csv", "csv/monthly/2022-7.csv", "csv/monthly
    /2022-8.csv", "csv/monthly/2022-9.csv", "csv/monthly/2022-10.csv", "csv/monthly/2022-11.csv", "csv/
    monthly/2022-12.csv",
                "csv/monthly/2023-1.csv", "csv/monthly/2023-2.csv", "csv/monthly/2023-3.csv", "csv/monthly
    /2023-4.csv", "csv/monthly/2023-5.csv", "csv/monthly/2023-6.csv", "csv/monthly/2023-7.csv", "csv/monthly
    /2023-8.csv", "csv/monthly/2023-9.csv", "csv/monthly/2023-10.csv", "csv/monthly/2023-11.csv", "csv/
```

```
     monthly/2023-12.csv"]

    num = 1
    for filename in filenames:
        readFile(filename)
        print("Done ", num ," of 38")
        num = num + 1

def readFile(filename):
    df = pd.read_csv(filename)

    #Compress data to track each stations usage per date
    stations=df.iloc[:,0]
    times=df.iloc[:,1]
    bikes=df.iloc[:,6]

    compressed = []
    row = 0
    while row < len(times)-1:
       rowDate = datetime.strptime(times[row], '%Y-%m-%d %H:%M:%S').date()
       current = combinedStationTime(rowDate, stations[row], bikes[row])

       for c in compressed:
          if c.isDateEqual(current):
             c.addUsage(current)
             current = None
             break

       if current != None:
          compressed.append(current)

       print(row)
       row = row + 1

    #Write compressed data
    date = []
    avgusage = []
    for c in compressed:
       date.append(c.date)
       avgusage.append(c.getAvgUsage())

    toWrite = pd.DataFrame({'DATE': date, 'AVG USAGE':avgusage})
    if not os.path.exists("condenseddata.csv"):
        toWrite.to_csv('condenseddata.csv', mode='a', index=False)
    else:
       toWrite.to_csv('condenseddata.csv', mode='a', header=False, index=False)


class combinedStationTime:
  def __init__(self, date, station, bikes):
    self.date = date
    self.stations = [station]
    self.bikes = [bikes]
    self.usages = [0]

  def isDateEqual(self, other):
     if (self.date == other.date):
        return True
     return False

  def getAvgUsage(self):
     total = 0
     for usage in self.usages:
        total = total + usage
     return total / len(self.stations)


  def addUsage(self, other):
     if self.stations.count(other.stations[0]) > 0:
```

```
77          index = self.stations.index(other.stations[0])
78          if(other.bikes[0] < self.bikes[index]):
79             self.usages[index] = self.usages[index] + (self.bikes[index] - other.bikes[0])
80          self.bikes[index] = other.bikes[0]
81       else:
82         self.stations.append(other.stations[0])
83         self.bikes.append(other.bikes[0])
84         self.usages.append(0)
```