

CSGY 6513 Big Data - Academic Project

Analyzing Temperatures based on Covid Lockdown

Done By: Team Meteorology

Members: Prateek Sridhar Kumar (ps4146), Mithra Shanmugasundaram(ms12292),
Divya Gupta(dg3483), Aayushi Sikligar (aps6962), Rashi Dhir(rd2808)

Problem Statement

Just as a small amount of sugar or salt can make a cake taste very different, a very short lockdown can trigger a chain reaction of interesting atmospheric effects and cause the everyday temperatures to plummet or rise. That is exactly our problem statement: We have made an attempt to analyze the effect of these lockdowns on everyday temperatures.

As we all know the pandemic has caused a lot of busy cities around the world to enforce lockdowns. In 2020, New York City went into a major lockdown, forcing thousands of people to stay back at home, which triggered a chain reaction, causing the traffic system of NYC to see an unbelievable reduction in the flow of vehicles in the city. Naturally the result of this is clear, cleaner and unpolluted air. There are proofs for this in newspaper articles about how this impacted the Air Quality Index. The good consequence that resulted from this is lowering of everyday temperatures as well.

In this project, we aim to use the techniques taught to us in this class to analyze the temperature data of NYC, and use this data to train a machine learning model and predict the future temperatures and how the future temperature varies based on whether the city is under a lockdown or not.

Why is this a big data problem?

Big Data has three main characteristics: Volume (amount of data), Velocity (speed of data in and out), Variety (range of data types and sources).

- **Volume:** Volume describes the amount of data generated by organizations or individuals. Our data is voluminous as it spans across years.
- **Velocity:** Velocity describes the frequency at which data is generated, captured and shared. Temperature keeps on changing rapidly and thus it has velocity.
- **Variety:** The data sources are varied in our case as it comprises sensor data, meteorological data amongst others.

All the above criteria are met in our case.

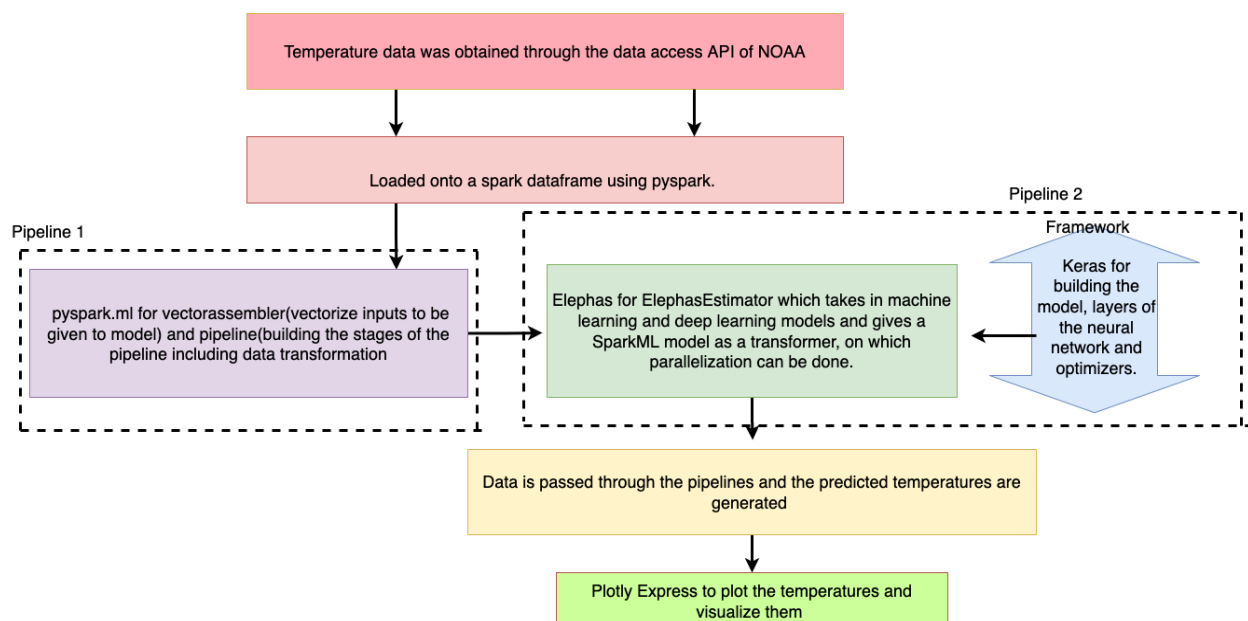
Technology Stack

Our main tool is Apache Spark: Few of the reasons that we went for pyspark is because it's a robust tool to load big data, and this helps us work with semi-structured data. This is great for performing exploratory data analysis at scale, building machine learning pipelines, and creating ETLs for data platforms. Other equally important reasons would be familiarity with the tool - since all of us have experience working with this, it was the easier choice.

We are mainly using three libraries:

- **Keras:** As we are aware, Keras is a deep learning API mainly focussing on Neural networks. Keras was relatively easy to learn and work with because it provides a python frontend with a high level of abstraction while having the option of multiple back-ends for computation purposes. We can define a neural network, compile it, fit it, evaluate it and finally make predictions.
- **Elephas:** Elephas is an extension of Keras, which allows you to run distributed deep learning models at scale with Spark. Elephas implements a class of data-parallel algorithms on top of Keras, using Spark's data frames.
- **Plotly:** Plotly is a open source python plotting library that can make different kinds of interactive graphs. We have used that to create a line graph to plot temperature predictions over a time range.

Architecture:



Overall flow and sequence of steps in the project:

We are obtaining the data from National Oceanic and Atmospheric Administration(NOAA) through a simple API call and we are loading the data into a spark dataframe using pyspark

- There are two pipelines that we have built: In pipeline 1, we are using sparkml's vector assembler to vectorize the inputs to give it to the model and build the stages of the pipeline including the data transformation.
- The result of pipeline 1 is then passed on to pipeline 2, where we are using Keras to define and build the neural network model, layers of the model and define optimizers. Elephas, as I said previously, is an extension of Keras, which has predefined ML and deep learning models, we can use this to perform data parallelization.
- We run our data through this model and pipeline to train the model and later we use this to predict the temperature trend in 2022.
- We are finally using plotly to plot and visualize the data

Execution of the project

- Data loading

We are getting our weather data from the most reliable source, NOAA (National Oceanic and Atmospheric Administration). First we had to request an access token, which basically affirms that we are a trusted source asking for data.

Next, we found some weather stations in New York State to gather data from. We had used <https://www.ncdc.noaa.gov/cdo-web/datatools/findstation> website for that. In this, We need to fill out preferred location, dataset preference, time range and data categories. Number of weather stations will be retrieved on map as pin-point locations. We can get all the information of the required station by clicking that pin-point location. We have chosen New York as location, Daily Summaries as dataset preference, time range as 1 January 2015 to 31st December 2019 and Air temperature as data category. We got the list of stations out of which we have chosen JFK INTERNATIONAL AIRPORT, NY US USW00094789 weather station.

Now we can request the data from NOAA using Rest API. We have used Python's requests library for this task.

The base request is <https://www.ncdc.noaa.gov/cdo-web/api/v2/data?>

We need to specify all the options for the data we are looking for. These will include:

- **datasetid**, which for us is GHCND (Global Historical Climatology Network Daily).
- **datatypeid**, which is a list of the variables we are after. For us, that is TAVG (average temperature)
- **limit**, specifying the maximum number of items to include in the response. The default is 25 and the maximum is **1000**, which is what we have set.

- **stationid**, specifying which station(s) we would like data from. Our station id is **GHCND:USW00094789**.
- **startdate** and **enddate**, specifying the date range we wish to get data from. We will be calling the API once for each year because 1000 items are not enough for our full 7 years date range from 1st January 2015 until 12th December 2021.

Full documentation of API can be found here-:

<https://www.ncdc.noaa.gov/cdo-web/webservices/v2#data>

Our API call to NOAA is-:

```
requests.get('https://www.ncdc.noaa.gov/cdo-web/api/v2/data?datasetid=GHCND&datatypeid=TAVG&limit=1000&stationid=GHCND:USW00094789&startdate='+year+'-01-01&enddate='+year+'-12-31', headers={'token':Token})
```

We are creating two list **dates_temp** and **temps** to store the data from api. After applying transformations like casting string date to datetime and converting temperature from tenths of Celsius to Fahrenheit, we store this data in Pandas data frame **df_temp** as date and average temperature fields. We have approximately 2538 rows × 2 columns amount of data. We further need to transform data using py-spark. We converted pandas data-frame into spark data frame using command “**spark.createDataFrame(df_temp)**”. Then we used spark.sql functions like split, withcolumn etc functions to split date columns into year, month, day columns.

After transformations are data frame looks like -:

```
df = sparkDF_temp

df.printSchema()

root

|-- avgTemp: double (nullable = true)

|-- year: integer (nullable = true)

|-- month: integer (nullable = true)

|-- day: integer (nullable = true)
```

After that we assembled Lockdown information from the news articles and add it in our data frame.

df.show()

avgTemp	year	month	day	lockdown
32.54	2015	1	1	0
39.38	2015	1	2	0
36.5	2015	1	3	0
47.3	2015	1	4	0
42.980000000000004	2015	1	5	0
22.1	2015	1	6	0
21.92	2015	1	7	0
13.82	2015	1	8	0
26.060000000000002	2015	1	9	0
22.46	2015	1	10	0
25.7	2015	1	11	0
36.14	2015	1	12	0
32.18	2015	1	13	0
24.439999999999998	2015	1	14	0
30.92	2015	1	15	0
34.34	2015	1	16	0
23.18	2015	1	17	0
36.14	2015	1	18	0
39.74	2015	1	19	0
39.019999999999996	2015	1	20	0

only showing top 20 rows

Based on this information we calculated no of lockdown days.

df.filter((df.year == 2020) & (df.month == 3) & (df.day > 20)) \
.show(truncate=False)

avgTemp	year	month	day	lockdown	lockdown_days
54.14	2020	3	21	0	0
39.74	2020	3	22	0	0
38.66	2020	3	23	1	1
43.16	2020	3	24	1	2
43.879999999999995	2020	3	25	1	3
43.7	2020	3	26	1	4
53.42	2020	3	27	1	5
48.74	2020	3	28	1	6
47.84	2020	3	29	1	7
45.32	2020	3	30	1	8
43.16	2020	3	31	1	9

These are data loading and pre-processing steps.

- Building Pipelines

Now we create the pipeline using PySpark. This essentially takes your data and, per the feature lists you pass, will do the transformations and vectorizing so it is ready for modeling.

The first thing to do is create stages. This will contain each step that the data pipeline needs to to complete all transformations within our pipeline. We have vectorized those features using VectorAssembler. we can simply pipeline the stages and fit our data to the pipeline by using fit().

Then we actually transform the data by using transform. This will help to preview our newly transformed PySpark dataframe with all the original and transformed features.

```
df_transform.show()
```

	avgTemp	year	month	day	lockdown	lockdown_days	assembled_inputs
	32.54	2015	1	1	0	0	[2015.0,1.0,1.0,0.0,0.0]
	39.38	2015	1	2	0	0	[2015.0,1.0,2.0,0.0,0.0]
	36.5	2015	1	3	0	0	[2015.0,1.0,3.0,0.0,0.0]
	47.3	2015	1	4	0	0	[2015.0,1.0,4.0,0.0,0.0]
	42.98	2015	1	5	0	0	[2015.0,1.0,5.0,0.0,0.0]
	22.1	2015	1	6	0	0	[2015.0,1.0,6.0,0.0,0.0]
	21.92	2015	1	7	0	0	[2015.0,1.0,7.0,0.0,0.0]
	13.82	2015	1	8	0	0	[2015.0,1.0,8.0,0.0,0.0]
	26.06	2015	1	9	0	0	[2015.0,1.0,9.0,0.0,0.0]
	22.46	2015	1	10	0	0	[2015.0,1.0,10.0,0.0,0.0]
	25.7	2015	1	11	0	0	[2015.0,1.0,11.0,0.0,0.0]
	36.14	2015	1	12	0	0	[2015.0,1.0,12.0,0.0,0.0]
	32.18	2015	1	13	0	0	[2015.0,1.0,13.0,0.0,0.0]
	24.43	2015	1	14	0	0	[2015.0,1.0,14.0,0.0,0.0]
	30.92	2015	1	15	0	0	[2015.0,1.0,15.0,0.0,0.0]
	34.34	2015	1	16	0	0	[2015.0,1.0,16.0,0.0,0.0]
	23.18	2015	1	17	0	0	[2015.0,1.0,17.0,0.0,0.0]
	36.14	2015	1	18	0	0	[2015.0,1.0,18.0,0.0,0.0]
	39.74	2015	1	19	0	0	[2015.0,1.0,19.0,0.0,0.0]
	39.01	2015	1	20	0	0	[2015.0,1.0,20.0,0.0,0.0]

only showing top 20 rows

```
df_transform_fin = df_transform.select('assembled_inputs','avgTemp')
df_transform_fin.limit(5).toPandas()
```

	assembled_inputs	avgTemp
0	[2015.0, 1.0, 1.0, 0.0, 0.0]	32.54
1	[2015.0, 1.0, 2.0, 0.0, 0.0]	39.38
2	[2015.0, 1.0, 3.0, 0.0, 0.0]	36.50
3	[2015.0, 1.0, 4.0, 0.0, 0.0]	47.30
4	[2015.0, 1.0, 5.0, 0.0, 0.0]	42.98

- Splitting data into test and train data

Finally, we want to shuffle our dataframe and then split the data into train and test sets. You always want to shuffle the data prior to modeling to avoid any bias from how the data may be sorted or otherwise organized and specifically shuffling prior to splitting the data.

```
# Shuffle Data
```

```
df_transform_fin = df_transform_fin.orderBy(rand())
```

```
# Split Data into Train / Test Sets
```

```
train_data, test_data = df_transform_fin.randomSplit([.8, .2], seed=1234)
```

We'll now build a basic deep learning model using Keras. We need to determine the number of inputs from our data so we can plug those values into our Keras deep learning model. In our case it is

```
input_dim = 5
```

We created a basic deep learning model. Using the model = Sequential() feature from Keras, it's easy to simply add layers and build a deep learning model the all the desired settings (# of units, dropout %, regularization - l2, activation functions, etc.) .

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 256)	1536
activation (Activation)	(None, 256)	0
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 256)	65792
activation_1 (Activation)	(None, 256)	0
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 1)	257
Total params: 67,585		
Trainable params: 67,585		
Non-trainable params: 0		

Using Elephas:

The first thing we do with Elephas is create an estimator similar to some of the PySpark pipeline items above. We can set the optimizer settings right from Keras optimizer function and then pass that to our Elephas estimator. We explicitly use Adam optimizer with a set learning rate. Then within the Elephas estimator you specify a variety of items: features column, label column, # of epochs, batch size for training, validation split of your training data, loss function, metric, etc.

```

# Set and Serialize Optimizer
optimizer_conf = Adam(learning_rate=0.01)
opt_conf = optimizers.serialize(optimizer_conf)

# Initialize SparkML Estimator and Get Settings
estimator = ElephasEstimator()
estimator.setFeaturesCol("assembled_inputs")
estimator.setLabelCol("avgTemp")
estimator.set_keras_model_config(model.to_json())
estimator.set_categorical_labels(False)
#estimator.set_nb_classes(nb_classes)
estimator.set_num_workers(1)
estimator.set_epochs(25)
estimator.set_batch_size(64)
estimator.set_verbosity(1)
estimator.set_validation_split(0.10)
estimator.set_optimizer_config(opt_conf)
estimator.set_mode("synchronous")
estimator.set_loss("mse")
estimator.set_metrics(['acc'])

ElephasEstimator_e7caa163d1ed

```

we run the estimator; the output, `ElephasEstimator_e7caa163d1ed`, looks similar to one of our pipeline stages list items. This can be passed directly into our PySpark pipeline to fit and transform our data .

Our deep learning model is to be run on Spark, using Elephas, we can pipeline it exactly how we did above using `Pipeline()`. You could append this to our stages list and do all of this with one pass with a new dataset now that it's all built out now.

The helper function called **`dl_pipeline_fit_score_results`** takes the deep learning pipeline `dl_pipeline` and then does all the fitting, transforming, and prediction on both the train and test data sets.

Our training set predictions:-

```
#from elephas.spark_model import SparkMLlibModel  
a.show()
```

assembled_inputs	avgTemp	prediction
[2015.0,1.0,1.0,0.0,...]	32.54	46.12065124511719
[2015.0,1.0,3.0,0.0,...]	36.5	45.956199645996094
[2015.0,1.0,4.0,0.0,...]	47.3	45.91920471191406
[2015.0,1.0,6.0,0.0,...]	22.1	45.9461669921875
[2015.0,1.0,7.0,0.0,...]	21.92	45.98291778564453
[2015.0,1.0,8.0,0.0,...]	13.82	46.01304244995117
[2015.0,1.0,9.0,0.0,...]	26.060000000000002	46.065834045410156
[2015.0,1.0,10.0,...]	22.46	46.12055206298828
[2015.0,1.0,11.0,...]	25.7	46.16706466674805
[2015.0,1.0,12.0,...]	36.14	46.19855499267578
[2015.0,1.0,13.0,...]	32.18	46.19550323486328
[2015.0,1.0,14.0,...]	24.439999999999998	46.18675994873047
[2015.0,1.0,15.0,...]	30.92	46.14360809326172
[2015.0,1.0,16.0,...]	34.34	46.098350524902344
[2015.0,1.0,17.0,...]	23.18	46.051055908203125
[2015.0,1.0,18.0,...]	36.14	46.04096984863281
[2015.0,1.0,19.0,...]	39.74	46.05299758911133
[2015.0,1.0,20.0,...]	39.019999999999996	46.06438064575195
[2015.0,1.0,21.0,...]	32.36	46.06938934326172
[2015.0,1.0,23.0,...]	33.26	46.064849853515625

only showing top 20 rows

Our test set predictions-:

```
[ ] b.show()
```

```
+-----+-----+-----+
| assembled_inputs | avgTemp | prediction |
+-----+-----+-----+
|[2015.0,1.0,2.0,0...| 39.38 | 46.036136627197266 |
|[2015.0,1.0,5.0,0...| 42.980000000000004 | 45.91567611694336 |
|[2015.0,1.0,22.0,...| 35.42 | 46.06187057495117 |
|[2015.0,1.0,25.0,...| 38.66 | 46.1011848449707 |
|[2015.0,1.0,26.0,...| 28.4 | 46.109901428222656 |
|[2015.0,1.0,27.0,...| 24.8 | 46.100433349609375 |
|[2015.0,1.0,28.0,...| 23.54 | 46.09267807006836 |
|[2015.0,2.0,6.0,0...| 18.68 | 47.52458190917969 |
|[2015.0,2.0,15.0,...| 21.38 | 47.51694107055664 |
|[2015.0,2.0,17.0,...| 20.479999999999997 | 47.40709686279297 |
|[2015.0,2.0,22.0,...| 34.16 | 47.2611083984375 |
|[2015.0,2.0,26.0,...| 27.86 | 47.14141082763672 |
|[2015.0,2.0,27.0,...| 24.98 | 47.12146759033203 |
|[2015.0,2.0,28.0,...| 23.18 | 47.101158142089844 |
|[2015.0,3.0,5.0,0...| 32.72 | 49.00448226928711 |
|[2015.0,3.0,8.0,0...| 38.84 | 49.1517219543457 |
|[2015.0,3.0,13.0,...| 37.04 | 48.90959167480469 |
|[2015.0,3.0,14.0,...| 42.620000000000005 | 48.84087371826172 |
|[2015.0,3.0,18.0,...| 35.24 | 48.519798278808594 |
|[2015.0,3.0,26.0,...| 45.32 | 48.05818557739258 |
+-----+-----+-----+
only showing top 20 rows
```

Now we are testing our model on the year 2022 dates by creating two data frames with lockdown and without lockdown information as input and trying to predict the average temperature in both the situations.

```
sparkDFpred1.show()
```



lockdown	year	month	day	lockdown_days
1	2022	1	1	1
1	2022	1	2	2
1	2022	1	3	3
1	2022	1	4	4
1	2022	1	5	5
1	2022	1	6	6
1	2022	1	7	7
1	2022	1	8	8
1	2022	1	9	9
1	2022	1	10	10
1	2022	1	11	11
1	2022	1	12	12
1	2022	1	13	13
1	2022	1	14	14
1	2022	1	15	15
1	2022	1	16	16
1	2022	1	17	17
1	2022	1	18	18
1	2022	1	19	19
1	2022	1	20	20


sparkDFpred2.show()

lockdown	lockdown_days	year	month	day
0	0	2022	1	1
0	0	2022	1	2
0	0	2022	1	3
0	0	2022	1	4
0	0	2022	1	5
0	0	2022	1	6
0	0	2022	1	7
0	0	2022	1	8
0	0	2022	1	9
0	0	2022	1	10
0	0	2022	1	11
0	0	2022	1	12
0	0	2022	1	13
0	0	2022	1	14
0	0	2022	1	15
0	0	2022	1	16
0	0	2022	1	17
0	0	2022	1	18
0	0	2022	1	19
0	0	2022	1	20

only showing top 20 rows

Send both these data frames to our pipeline 1. We got the output as below:-

```
df1.show()
```



lockdown	year	month	day	lockdown_days	assembled_inputs
1	2022	1	1	1	[2022.0,1.0,1.0,1...
1	2022	1	2	2	[2022.0,1.0,2.0,1...
1	2022	1	3	3	[2022.0,1.0,3.0,1...
1	2022	1	4	4	[2022.0,1.0,4.0,1...
1	2022	1	5	5	[2022.0,1.0,5.0,1...
1	2022	1	6	6	[2022.0,1.0,6.0,1...
1	2022	1	7	7	[2022.0,1.0,7.0,1...
1	2022	1	8	8	[2022.0,1.0,8.0,1...
1	2022	1	9	9	[2022.0,1.0,9.0,1...
1	2022	1	10	10	[2022.0,1.0,10.0,...
1	2022	1	11	11	[2022.0,1.0,11.0,...
1	2022	1	12	12	[2022.0,1.0,12.0,...
1	2022	1	13	13	[2022.0,1.0,13.0,...
1	2022	1	14	14	[2022.0,1.0,14.0,...
1	2022	1	15	15	[2022.0,1.0,15.0,...
1	2022	1	16	16	[2022.0,1.0,16.0,...
1	2022	1	17	17	[2022.0,1.0,17.0,...
1	2022	1	18	18	[2022.0,1.0,18.0,...
1	2022	1	19	19	[2022.0,1.0,19.0,...
1	2022	1	20	20	[2022.0,1.0,20.0,...

only showing top 20 rows

```
df2.show()
```

lockdown	lockdown_days	year	month	day	assembled_inputs
0	0	2022	1	1	[2022.0,1.0,1.0,0...
0	0	2022	1	2	[2022.0,1.0,2.0,0...
0	0	2022	1	3	[2022.0,1.0,3.0,0...
0	0	2022	1	4	[2022.0,1.0,4.0,0...
0	0	2022	1	5	[2022.0,1.0,5.0,0...
0	0	2022	1	6	[2022.0,1.0,6.0,0...
0	0	2022	1	7	[2022.0,1.0,7.0,0...
0	0	2022	1	8	[2022.0,1.0,8.0,0...
0	0	2022	1	9	[2022.0,1.0,9.0,0...
0	0	2022	1	10	[2022.0,1.0,10.0,...
0	0	2022	1	11	[2022.0,1.0,11.0,...
0	0	2022	1	12	[2022.0,1.0,12.0,...
0	0	2022	1	13	[2022.0,1.0,13.0,...
0	0	2022	1	14	[2022.0,1.0,14.0,...
0	0	2022	1	15	[2022.0,1.0,15.0,...
0	0	2022	1	16	[2022.0,1.0,16.0,...
0	0	2022	1	17	[2022.0,1.0,17.0,...
0	0	2022	1	18	[2022.0,1.0,18.0,...
0	0	2022	1	19	[2022.0,1.0,19.0,...
0	0	2022	1	20	[2022.0,1.0,20.0,...

only showing top 20 rows

We only select the assembled_input column from the output dataframe and send it to pipeline 2 to get the predictions.



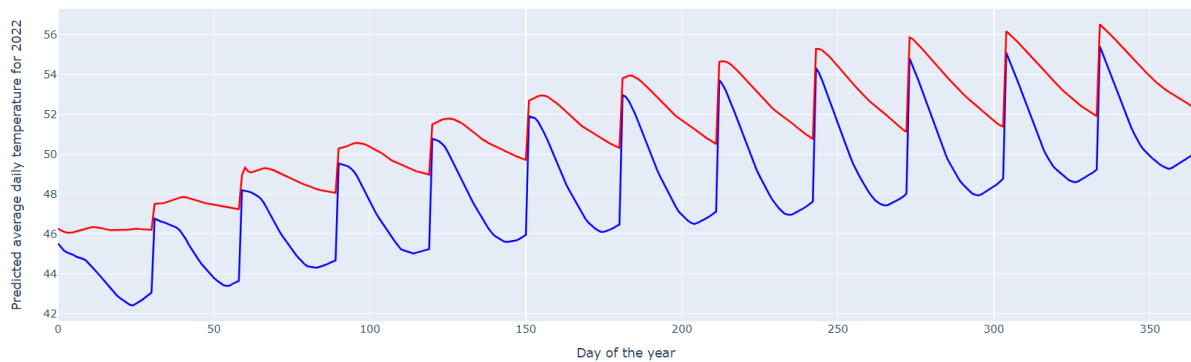
assembled_inputs	prediction
[2022.0,1.0,1.0,1.0,...]	45.51806640625
[2022.0,1.0,2.0,1.0,...]	45.32127380371094
[2022.0,1.0,3.0,1.0,...]	45.14875411987305
[2022.0,1.0,4.0,1.0,...]	45.055397033691406
[2022.0,1.0,5.0,1.0,...]	45.01095199584961
[2022.0,1.0,6.0,1.0,...]	44.93840026855469
[2022.0,1.0,7.0,1.0,...]	44.84392547607422
[2022.0,1.0,8.0,1.0,...]	44.796180725097656
[2022.0,1.0,9.0,1.0,...]	44.743743896484375
[2022.0,1.0,10.0,...]	44.660160064697266
[2022.0,1.0,11.0,...]	44.500213623046875
[2022.0,1.0,12.0,...]	44.31977844238281
[2022.0,1.0,13.0,...]	44.152976989746094
[2022.0,1.0,14.0,...]	43.98039245605469
[2022.0,1.0,15.0,...]	43.798072814941406
[2022.0,1.0,16.0,...]	43.60568618774414
[2022.0,1.0,17.0,...]	43.43058776855469
[2022.0,1.0,18.0,...]	43.255096435546875
[2022.0,1.0,19.0,...]	43.070404052734375
[2022.0,1.0,20.0,...]	42.88470458984375

only showing top 20 rows

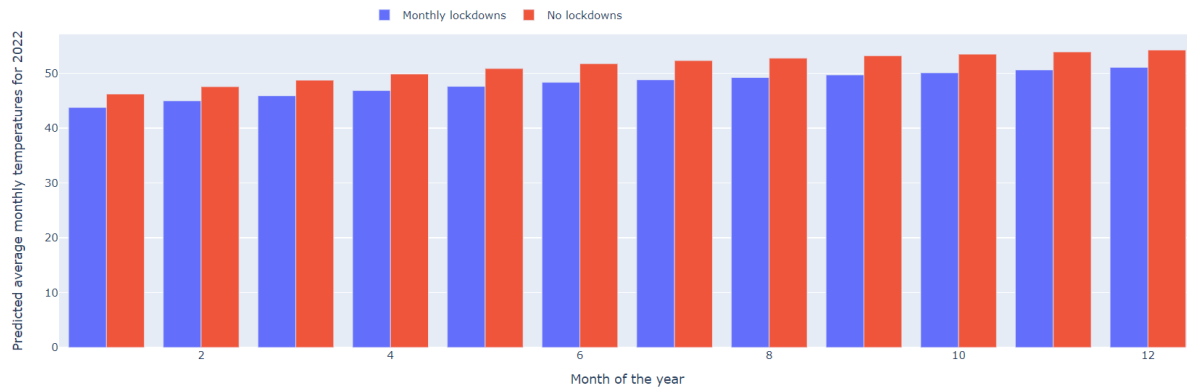
assembled_inputs	prediction
[2022.0,1.0,1.0,0.0,...]	46.26041030883789
[2022.0,1.0,2.0,0.0,...]	46.175628662109375
[2022.0,1.0,3.0,0.0,...]	46.09564208984375
[2022.0,1.0,4.0,0.0,...]	46.05713653564453
[2022.0,1.0,5.0,0.0,...]	46.05221176147461
[2022.0,1.0,6.0,0.0,...]	46.08230972290039
[2022.0,1.0,7.0,0.0,...]	46.11885452270508
[2022.0,1.0,8.0,0.0,...]	46.14863586425781
[2022.0,1.0,9.0,0.0,...]	46.200408935546875
[2022.0,1.0,10.0,0.0,...]	46.255332946777344
[2022.0,1.0,11.0,0.0,...]	46.302330017089844
[2022.0,1.0,12.0,0.0,...]	46.336158752441406
[2022.0,1.0,13.0,0.0,...]	46.333683013916016
[2022.0,1.0,14.0,0.0,...]	46.324951171875
[2022.0,1.0,15.0,0.0,...]	46.283843994140625
[2022.0,1.0,16.0,0.0,...]	46.23857116699219
[2022.0,1.0,17.0,0.0,...]	46.191349029541016
[2022.0,1.0,18.0,0.0,...]	46.177608489990234
[2022.0,1.0,19.0,0.0,...]	46.189640045166016
[2022.0,1.0,20.0,0.0,...]	46.201168060302734

only showing top 20 rows

And compare the results of average temperature in both the scenarios using plotly.



The blue line depicts daily average temperature with lockdowns and red line depicts daily average temperature without lockdown.



We can see a marked difference in temperatures with lockdown and without lockdown.

Conclusion:

As observed from our predictive model, there is variation in temperature impacted by monthly lockdowns. The temperature has significantly decreased during monthly lockdowns which implies how the pandemic impacted the global temperature in general. Thus, we have successfully analyzed the temperature variation during the pandemic using a pipelining approach in ML.

- What we learned:

From this project, we learned how to deal with a real world problem in the domain of Big Data and how to apply the tools and techniques learned during the course of the class in order to resolve it and obtain effective solutions.

- Future Scope:

The scope of the project can be extended by incorporating more parameters into this project like Air Quality information, precipitation information to get a detailed analysis of the overall effects of the lockdown on the atmosphere. This analysis can also be extended to help predict the climatic conditions post pandemic scenario.