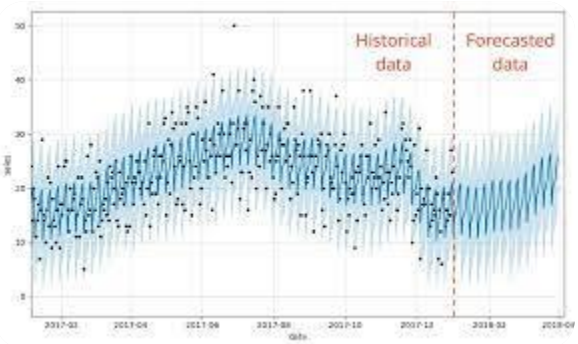


PHASE 6

ABSTRACT

Phase 6: Project Report

Soumen Chatterjee

Phase 6 – 06-Mar-2022

Phase 1: Literature survey & Data Acquisition

Objectives

Client “Walmart” reached out, to get help on predicting stocks, required to get inventory up to date.

As per the briefing, in 27 countries Walmart operates 11,450 stores, manages inventory across varying climates and cultures.

- Issue-

  Extreme weather events, like hurricanes, blizzards, & floods, have huge impact on sales at the stores and product level. Walmart need a solution which accurately predict the sales of 111 potentially weather-sensitive products (like umbrellas, bread, and milk) around the time of major weather events at 45 of their retail locations which are spread across 20 weather stations. Intuitively, we may expect an uptick in the sales of umbrellas before a big thunderstorm.

- Requirement-

  It's difficult for replenishment managers to correctly predict the level of inventory needed to avoid being out-of-stock or overstock during and after that storm or any other adverse natural events. Walmart has been relying on a variety of vendor tools to predict sales around specific weather events, but it's an ad-hoc and time-consuming process that lacks a systematic measure of effectiveness.

  It's required to create systematic approach which will help Walmart better predict sales of weather-sensitive products.

- Why it’s important –

  Creation of systematic approach using newest technologies backed up scientific & data driven reasoning is very much important for a client like Walmart, this will help them better predict sales of weather-sensitive products. Correct prediction will help replenishment managers to keep the stocks up to date on specific time of a year. If the forecast is too high, it may lead to over-investing and therefore losing money. If the forecast is too low, it may lead to under-investing and therefore losing opportunity.

- Business/Real-world impact of solving this problem –

  As of now it is requested to focus on only 45 retail locations and on 111 products. If the new approach gets successfully into production and

correctly predicts then inclusion of other stores, products across the countries will results huge profit to Walmart. As correct prediction will help them plan better in logistics & will help them keep customer base strong.

Dataset:

- Source

  As per the briefing, it's requested to use the data provided by Walmart. A zipped file namely "*Walmart-recruiting-sales-in-stormy-weather*" is received to deal with this specific requirement. The Zipped file contains below mentioned details.

  Data size is well within the scope.

  And being it's structured it can be dealt with standard libraries/tools like Pandas, Matplotlib, Scikit-learn, SQL

a. "weather.csv"

- This file contains details of weather on a specific day/date, starting from 1st Jan 2012 to 31st Oct 2014.
- Consisting of 216435 rows & 20 columns/features.

Feature No.	Feature Name	Description
F1	station_nbr	Weather station number, representing one of 20 weather stations
F2	date	DATE - MM/DD/YYYY
F3	tmax	Maximum Temparature Fahrenheit
F4	tmin	Minimum Temparature Fahrenheit
F5	tavg	Average Temparature Fahrenheit
F6	depart	Departure from normal, that is above/below 30 years' normal
F7	dewpoint	Average dew point
F8	wetbulb	Average wet bulb
F9	heat	Heating (season begins with july)
F10	cool	Cooling (season begins with january)
F11	sunrise	Sunrise (calculated, not observed)
F12	sunset	Sunset (calculated, not observed)
F13	codesum	Weather Phenomena - with 448
F14	snowfall	Height in INCHES
F15	preciptotal	Inches (24-hr period ending at indicated local standard time)
F16	stnpressure	Average Station pressure
F17	sealevel	Average Sea level pressure
F18	resultspeed	Speed in miles per hour

F19	resultdir	Direction to tens of degrees
F20	avgspeed	Average wind speed

b. "train.csv"

- This file contains Date, Store number (an id representing one of the 45 stores), Item number (an id representing one of the 111 products), Unit information (the quantity sold of an item on a given day)
- Consisting of 1048575 rows & 4 columns/features.

c. "key.csv"

- This file contains the mapping of Store number & station number
- Consisting of 1035 rows & 2 columns/features.

d. "sampleSubmission.csv"

- This file contains the input format "**StoreNbr_ProdNbr_Date**" which need to be passed to production unit for prediction
- Consisting of 526917 rows & 2 columns – ID, Units

e. "test.csv"

- Don't have password to open this file
- This file should have similar structure like b. "train.csv", & should contains Date, Store number, Item number, unit information
- Consisting of 4 columns/features.

NOTE:

- Data type, description & other details of these files/data will be further explored at the time of Exploratory data analysis stage.

- The Data received should be adequate to do the analysis & prediction

- If required and/or if we don't have the password to open "test.csv" then we can generate synthetic Data accordingly

Key Performance Indicator

Having a quantifiable activity used to measure how a key aspect of business is operating or how correctly future sales are being predicted in this case, it's important & needed.

Like other usual business cases, here in this scenario as well, we need to think of 4 aspects of KPI and implementing the same, that is implementation of KPI following 4 steps –

Timeframe – Here we will need to analyze the data first and need plan a window of time, like in this case we can consider 3 months window

Measurement/Business Metric – Choosing a business matric, here in this case we can plan for Root Mean Squared Logarithmic Error (RMSLE).

Monitor according to timeframe & improve by the time.

- Business Metric

  As per the requirement stated above it is requested to predict the sales of a specific product in near future using the past data, so it's a Regression problem.

  There are few Metrics which are used to measure the performance of a regression model as mentioned below –

Id	Metric	Aplication
MAE	Mean absolute error	Measure of difference between two continuous variables. MAE is the average
MSE	Mean squared error	Measure the average of the squares of the errors—that is, the average squared difference between the estimated values and the actual value
RMSE	Root mean squared error	Frequently used measure of the differences between values (sample or population values) predicted by a model or an estimator and the values observed. The RMSD represents the square root of the second sample moment of the differences between predicted values and observed values or the quadratic mean of these differences
MAPE	Mean absolute percentage error	Measure of prediction accuracy of a forecasting method in statistics, for example in trend estimation, also used as a loss function for regression problems in machine learning. It usually expresses accuracy as a percentage
RMSLE	Root Mean Squared Logarithmic Error	RMSLE is usually used when you don't want to penalize huge differences in the predicted and the actual values when both predicted and true values are huge numbers.
R-Squared	Coefficient of determination R^2	Is the proportion of the variance in the dependent variable that is predictable from
Adjusted R-Squared	R-Squared with adjust to features quantity	Adding new features to the model, the R-Squared value either increases or remains the same. R-Squared does not penalize for adding features that add no value to the model. So an improved version over the R-Squared is the adjusted R-Squared.

Ref: Image 1- copied from https://towardsdatascience.com/metrics-and-python-850b60710e0c

The problem we are dealing with here - given some data in time, we want to predict the dynamics of that same data in the future. Data, which is shared, contains series of data points indexed (or listed or graphed) in time order. Which is a sequence taken at successive equally spaced points in time. Thus, it is a sequence of discrete-time data.

Hence on its core, this is a time series problem.

And in Timeseries problems we usually use - Root Mean Squared Error (RMSE) and Root Mean Squared Logarithmic Error (RMSLE).

In this case, Root Mean Squared Logarithmic Error (RMSLE) will be used as business metric. As it is better compared to RMSE (details shared below)-

- Mathematical Definitions:

$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2},$	$\text{RMSLE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\log y_i - \log \hat{y}_i)^2}$

Ref: Image 2- copied from Datascience stackexchange

Where y^i is the target value for example i and y^i is the model's prediction. Both the metrics quantify the prediction error, so in general a high RMSE implies a high RMSLE as well.

RMSLE has the meaning of a relative error, while RMSE is an absolute error.

RMSE will have a drastic effect of outliers on its values. But in case of RMSLE we can reduce the effect of outliers by many magnitudes & their effects is much less.

RMSLE value will only consider the relative error between Predicted and the actual value neglecting the scale of data. But RMSE value will increase in magnitude if the scale of error increases. For e.g.

```
Actual value = 100
Predicted Value = 90
RMLSE: 0.1053
RMSE: 10

Actual value = 1000
Predicted Value = 900
RMSLE: 0.1053
RMSE : 100
```

Also, in case of under-estimation results from RMSLE are affected greatly. So, one can easily understand that it is better than RMSE in certain scenarios, but RMSE works better for generalize cases. At last, RMSLE is bit better than RMSE but based upon certain cases only.

Both the metrics that is Root Mean Squared Error (RMSE) and Root Mean Squared Logarithmic Error (RMSLE) can be used in other Timeseries problems where regression models are used like –

Weather forecasting, Stock price predicting, Unemployment for a state each quarter, Average price of gasoline etc.

- Code: Implementation of RMSLE metric using Python (used “math”, “numpy” library)

```
#Importing standard library math
import math
import numpy as np
#creating custom function for RMSLE calculation
def My_RMSLE(pred, trgt):
  #initializing tot variable, which will hold Total value
  tot = 0

  #Looping through the passback variable pred
  for k in range(len(pred)):
    #for each predicted value and corresponding target value is picked below
    #logarithm applied
    LPred= np.log1p(pred[k]+1)
    LTarg = np.log1p(trgt[k] + 1)

    #checking if none of the Pred & Target value is null then
    #squaring the differences
    if not (math.isnan(LPred)) and  not (math.isnan(LTarg)):
       tot = tot + ((LPred-LTarg) **2)

  #finding our average value and then doing Sqrroot & picking final value
  tot = tot / len(pred)
  return np.sqrt(tot)

y_pred = [2,5,6,1,7,9]
y      = [2.5,6,5,1,7,8.5]
print ('My custom RMSLE: ' + str(My_RMSLE(y_pred,y)))
```

Real world challenges and constraints

- Challenges, constraints foreseeing & the requirement

  Being this problem is a Timeseries & forecasting related we will need to check whether the data is stationary, seasonality, autocorrelated or not.
  Using Dickey-Fuller test, which is a statistical test that we will need to run to determine if the data received is stationary or not.
  If it's not stationary, then we will need to transform to make them stationary.
  As ideally, we want to have a stationary time series for modelling.

  And after that we will need to pick a right Model like- Moving Average, Exponential Smoothing, ARIMA, SARIMA to meet the final requirement to predict the further sales based on available past data.

Similar problems & references which can be referred

- It's a Regression problem as we need to predict sales (continuous value) of a product in near future using available past data

- The problem we are dealing with here - given some data in time, we want to predict the dynamics of that same data in the future. Data, which is shared, contains series of data points indexed (or listed or graphed) in time order. Which is a sequence taken at successive equally spaced points in time. Thus, it is a sequence of discrete-time data.
  Hence on its core, this is a time series problem.

- We can also refer other similar kind of problems like - Closing price of a stock each day, Product sales in units sold each day for a store, Unemployment for a state each quarter, The average price of gasoline each day, Forecasting Weather, Traffic, Churn, Text Generation etc.

- Other than using standard models like – AR, MA, ARMA, ARIMA, SARIMA, SARIMAX, VAR, VARMA, VARMAX, SES, HWES, we can try using Deep learning for predicting.

- Recurrent Neural Networks are the most popular Deep Learning technique for Time Series Forecasting since they allow to make reliable predictions on time series in many different problems. The main problem with RNNs is that they suffer from the vanishing gradient problem when applied to long sequences.

Phase 2: EDA and Feature Extraction

Data-set level and output-variable analysis:

We will deal with 3 set of Data.

- "train.csv":
  This holds for a specific date at a specific store for item, how much unit(s) is/are sold
  Below tables shows the basic structure of the tabular data-

	date	store_nbr	item_nbr	units
0	2012-01-01	1	1	0
1	2012-01-01	1	2	0
2	2012-01-01	1	3	0
3	2012-01-01	1	4	0
4	2012-01-01	1	5	0

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4617600 entries, 0 to 4617599
Data columns (total 4 columns):
 #   Column     Dtype
---  ------     -----
 0   date       object
 1   store_nbr  int64
 2   item_nbr   int64
 3   units      int64
dtypes: int64(3), object(1)
```

While reviewing this data set we found that –

1. There are many occasions where we have 0 units sold. These rows with 0 unit sale won't help us. So we have removed them and kept the required rows only

2. Apart from Units, we have 3 other columns namely date, store_nbr, item_nbr. These variables are fixed in nature. That is, these are constants which cannot be changed as 3 of them are identifiers.

3. So in this data set we focused on the feature "units". Univariate analysis is done at this variable. We can refer the notebook "Phase2_EDA_train.ipynb" for this.

4. We found outliers in this data set, those are treated through IQR

- "key.csv":
  This file is the link between train.csv & weather.csv. This file holds the information like at what weather station which store is located. This file will help us merging 2 other files.

	store_nbr	station_nbr
0	1	1
1	2	14
2	3	7
3	4	9
4	5	12

- “weather.csv”:

  This holds the information like on a specific date at a specific weather station what were the weather phenomenon. These details are captured under 20 columns, and we have data 20517 entries.

  Below tables shows the basic structure of the tabular data-

	station_nbr	date	tmax	tmin	tavg	depart	dewpoint	wetbulb	heat	cool	sunrise	sunset	codesum	snowfall	preciptotal	stnpressure	sealevel	resultspeed	resultdir	avgspeed
0	1	2012-01-01	52	31	42	M	36	40	23	0	-	-	RA FZFG BR	M	0.05	29.78	29.92	3.6	20	4.6
1	2	2012-01-01	48	33	41	16	37	39	24	0	0716	1626	RA	0.0	0.07	28.82	29.91	9.1	23	11.3
2	3	2012-01-01	55	34	45	9	24	36	20	0	0735	1720		0.0	0.00	29.77	30.47	9.9	31	10.0
3	4	2012-01-01	63	47	55	4	28	43	10	0	0728	1742		0.0	0.00	29.79	30.48	8.0	35	8.2
4	6	2012-01-01	63	34	49	0	31	43	16	0	0727	1742		0.0	0.00	29.95	30.47	14.0	36	13.8

	tmax	tmin	tavg	depart	dewpoint	wetbulb	heat	cool	snowfall	preciptotal	stnpressure	sealevel	resultspeed	resultdir	avgspeed
count	20517.000000	20517.000000	20517.000000	20517.000000	20517.000000	20517.000000	20517.000000	20517.000000	20517.000000	20517.000000	20517.000000	20517.000000	20517.000000	20517.000000	20517.000000
mean	71.570649	50.679168	61.555905	1.422576	47.234099	54.109032	9.258761	5.814666	0.028415	0.096420	29.169794	30.008790	6.517361	18.818663	7.983869
std	19.415409	18.719377	18.664973	5.068755	19.138339	16.561561	13.464017	7.708634	0.402737	0.337069	1.219092	0.183071	4.155430	9.558794	3.865889
min	-11.000000	-21.000000	-16.000000	-35.000000	-24.000000	-15.000000	0.000000	0.000000	0.000000	0.000000	23.720000	29.160000	0.000000	1.000000	0.000000
25%	60.000000	37.000000	50.000000	1.000000	33.000000	43.000000	0.000000	0.000000	0.000000	0.000000	29.130000	29.900000	3.300000	13.000000	5.200000
50%	75.000000	52.000000	64.000000	1.500000	51.000000	58.000000	1.000000	0.000000	0.000000	0.000000	29.380000	29.990000	5.800000	18.000000	7.500000
75%	86.000000	66.000000	77.000000	1.500000	64.000000	68.000000	15.000000	12.000000	0.000000	0.010000	29.750000	30.110000	9.200000	26.000000	9.900000
max	114.000000	88.000000	100.000000	33.000000	77.000000	80.000000	81.000000	35.000000	16.200000	7.360000	30.610000	30.800000	28.400000	36.000000	28.700000

Initial findings at this data set are-

1. If we review the data frame we can see that - apart from 'date' & 'codesum' column rest all should be treated as int64, but they are being treated as string

2. It is because the missing values are updated as 'M',"-",”T”, " "

3. We have replaced 'M',"-", " " with NaN, in the data frame. Converted the columns to required data format like - int64 for further proceedings.

4. We have a multi option categorical feature namely “codesum”, that is encoded

5. We had many NaN values that were filled

Feature Analysis

- Univariate Feature Analysis:

It is the simplest form of analyzing data. “Uni” means “one”, so in other words if dataset has only one variable. It doesn’t deal with causes or relationships (unlike regression) and its major purpose is to describe; it takes data, summarizes that data and finds patterns in the data.

Univariate Descriptive Statistics (**options we have**):

Some ways you can describe patterns found in univariate data include central tendency (mean, mode and median) and dispersion: range, variance, maximum, minimum, quartiles (including the interquartile range), and standard deviation.

- UNIVARIATE SCATTER PLOT: This plots different observations/values of the same variable corresponding to the index/observation number.

- LINE PLOT (with markers): This plot visualizes data by connecting the data points via line segments. It is similar to a scatter plot except that the measurement points are ordered (typically by their x-axis value) and joined with straight line segments.

- Uni-variate summary plots: These plots give a more concise description of the location, dispersion, and distribution of a variable than an enumerative plot. It is not feasible to retrieve every individual data value in a summary plot, but it helps in efficiently representing the whole data from which better conclusions can be made on the entire data set.

    a) HISTOGRAMS: Histograms are similar to bar charts which display the counts or relative frequencies of values falling in different class intervals or ranges. A histogram displays the shape and spread of continuous sample data. It also helps us understand the skewness and kurtosis of the distribution of the data.

    b) DENSITY PLOTS: A density plot is like a smoother version of a histogram. Generally, the kernel density estimate is used in density plots to show the probability

density function of the variable. A continuous curve, which is the kernel is drawn to generate a smooth density estimation for the whole data.

c) BOX PLOTS: A box-plot is a very useful and standardized way of displaying the distribution of data based on a five-number summary (minimum, first quartile, second quartile (median), third quartile, maximum). It helps in understanding these parameters of the distribution of data and is extremely helpful in detecting outliers.

d) Distplot (): The distplot () function of seaborn library was earlier mentioned under rug plot section. This function combines the matplotlib hist () function with the seaborn kdeplot () and rugplot () functions.

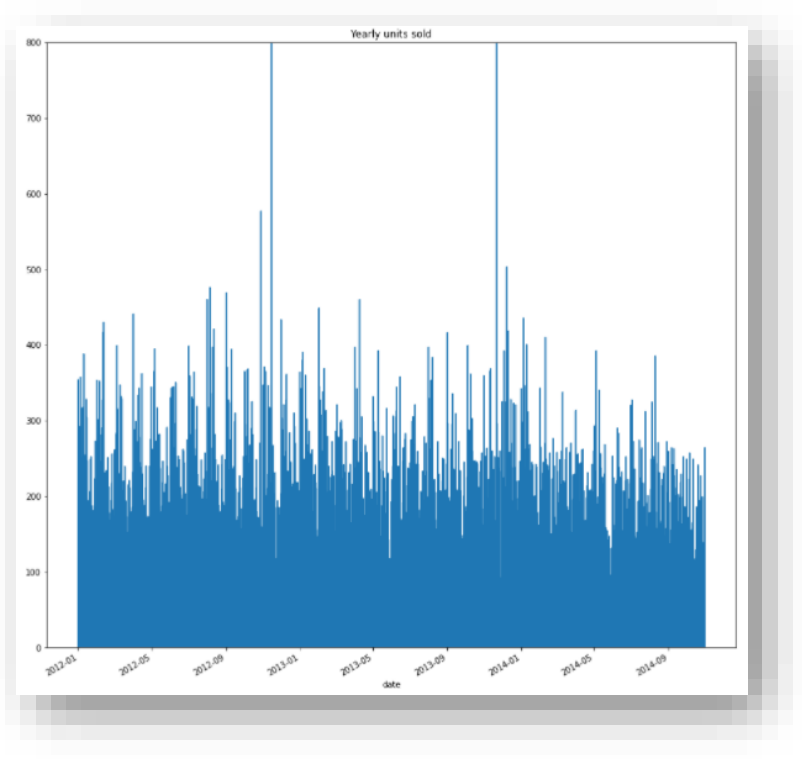
e) VIOLIN PLOTS: The Violin plot is very much similar to a box plot, with the addition of a rotated kernel density plot on each side. It shows the distribution of quantitative data across several levels of one (or more) categorical variables such that those distributions can be compared.

As mentioned earlier, in “train.csv” data set apart from “Units”, we have 3 other columns namely date, store_nbr, item_nbr. These variables are fixed in nature. That is, these are constants which cannot be changed as 3 of them are identifiers.

So we focused on the feature “units”. Univariate analysis is done on this variable. We can refer the notebook “Phase2_EDA_train.ipynb” for this.

We have used below plots here for Univariate analysis –

- ✓ Bar plot:

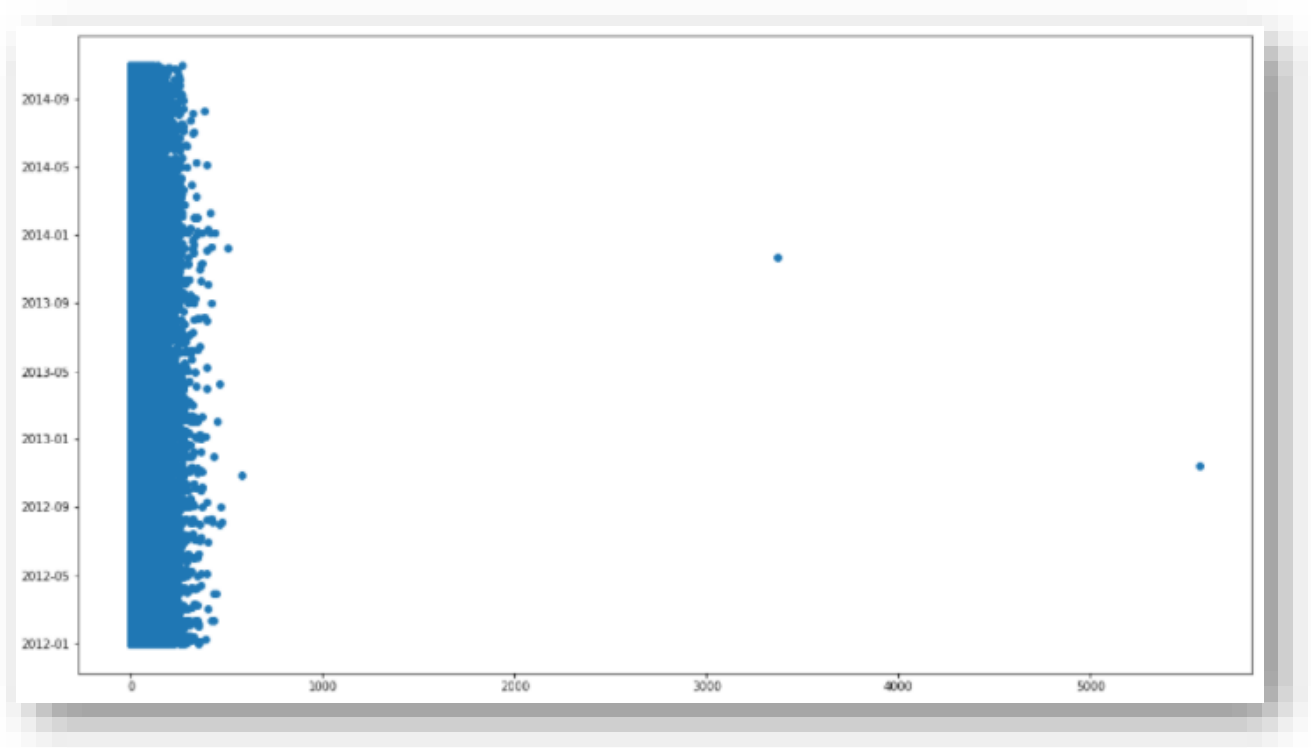


Uses: we have used to highlight separate values, especially the differences between these values. It is extremely useful for comparing values in different categories and can be used to describe the relationship of several variables at once. Using this plot we can clearly see the outliers. Quarter wise units sold, which quarter has pick value. Average unit sold etc.

Advantages: summarize a large dataset in visual form; easily compare two or three data sets; better clarify trends than do tables; estimate key values at a glance.

Disadvantages: require additional written or verbal explanation; can be easily manipulated to give false impressions.

- ✓ Scatter Plots



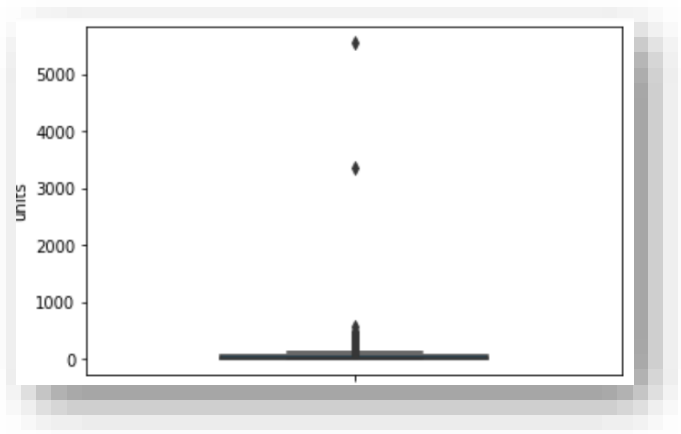
Uses: This plot illustrate paired data, that is, information regarding two related variables, in this case it's date (quarterly) Y axis & Units sold X axis. The resulting pattern (after all the points have been plotted) indicates the strength of the correlation between two variables. It's used here primarily to establish relationships, to see how data is arranged.

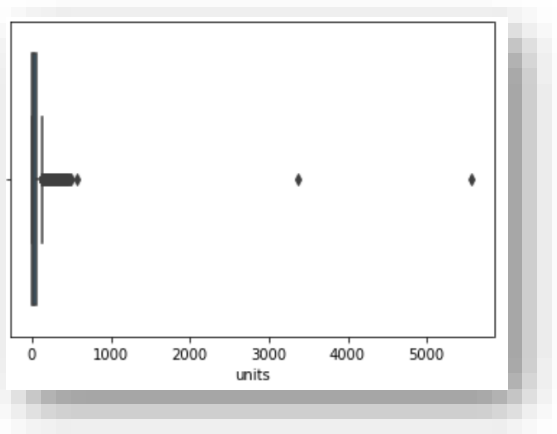
Advantages: Clearly indicates data correlation (illustrates positive, negative, strong, weak relationships); method of illustration non-linear patterns; shows spread of data, outliers; clearly demonstrate atypical relationships; used for data extrapolation and interpolation

Disadvantages: Impossible to label data points, hard to find out exact values; error bars and too many data points can quickly make graph unreadable; cannot show relationship between more than two variables at once

- ✓ BOX PLOTS:

Uses: It helped to check the distribution of data based on a five-number summary (minimum, first quartile, second quartile (median), third quartile, maximum). It helps in understanding these parameters of the distribution of data and is extremely helpful in detecting outliers.

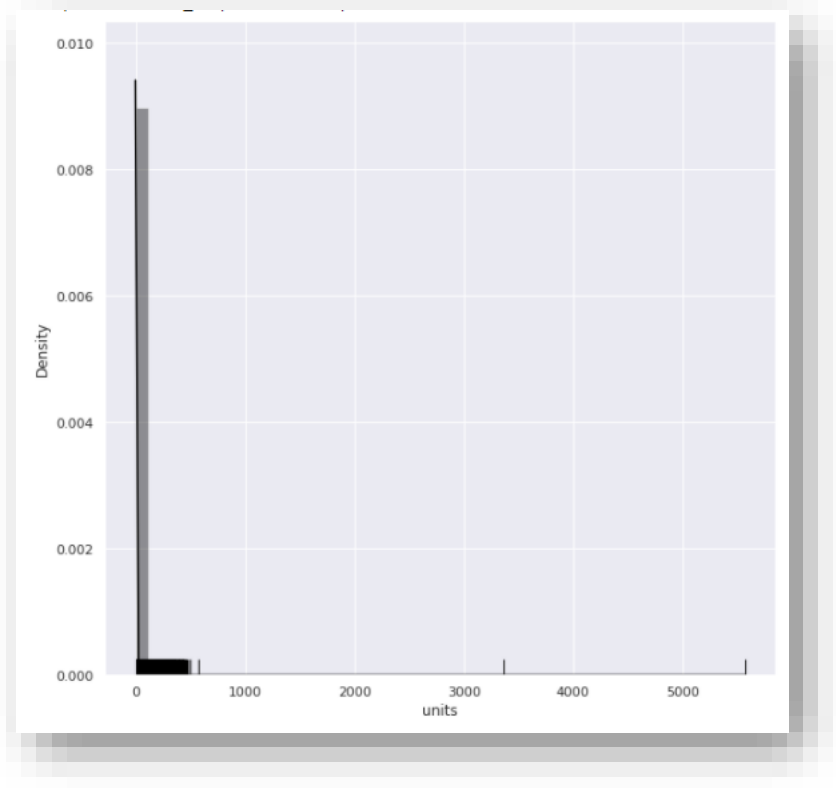




Advantages: It handles large data easily, shows a clear summary, Displays Outliers.

Disadvantages: Exact values and details of the distribution results are not retained, which is an issue with handling such large amounts of data in this graph type.

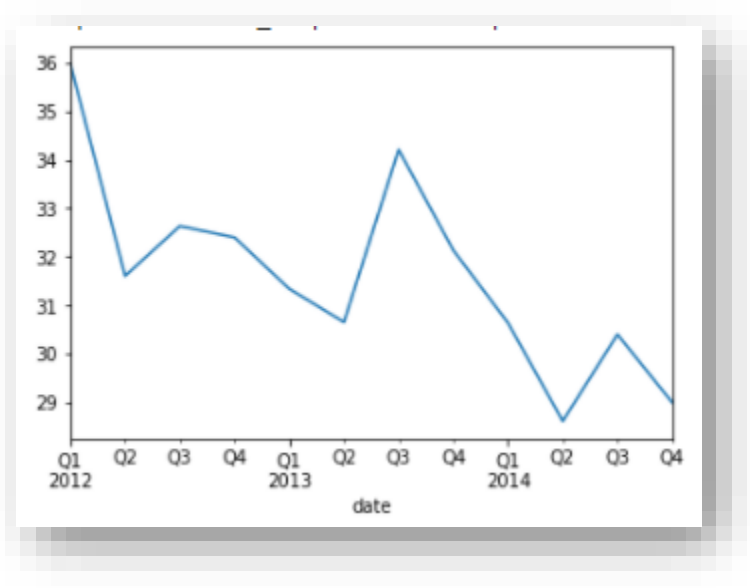
- ✓ Distplot ():



Advantage: An evident advantage is that it can easily display a general distribution of a set of numeric values over a range.

Disadvantage: One major limitation of a distribution plot is that it provides an overview of the values and their distribution. It does not give a detailed account of the ranging and distribution of the values.

- ✓ Line plot:



Uses: It is helpful to see how a value changes over time.

We have used this plot to see the trend of Units sold (irrespective of product, store) against quarterly date. As per our findings, in this case general trend across all the products sold (Units) is downwards. And for each year, Q3 sees more sells compared to other quarters

Advantage: It's better for seeing the rate clearly. Shows trends and relationships between data better than other graphs; compare trends in different groups of a variable; clearly show error values in the data.

Usually simple to read and understand

Disadvantage: It's harder to compare. For multiple lines on the graph, especially unrelated can be confusing; difficult to make out exact values for data.

- ✓ Descriptive statistics:
  Other than the above descriptive statistics for the units variable was referred, to check the central tendency, variance, min, and max & to gauge outliers

```
df_train["units"].describe()

count    4.617600e+06
mean     9.868756e-01
std      9.875798e+00
min      0.000000e+00
25%      0.000000e+00
50%      0.000000e+00
75%      0.000000e+00
max      5.568000e+03
Name: units, dtype: float64
```

Before outlier treatment

```
df_train["units"].describe()

count    113709.000000
mean         31.874240
std          31.659757
min           1.000000
25%           4.000000
50%          22.000000
75%          51.000000
max         133.000000
Name: units, dtype: float64
```

After outlier treatment

✓ IQR (Interquartile range):
We have used IQR as well, in descriptive statistics, the interquartile range (IQR) is a measure of statistical dispersion, which is the spread of the data. The IQR may also be called the midspread, middle 50%, or H-spread. It is defined as the difference between the 75th and 25th percentiles of the data.
Advantages: It is not affected by extreme values as in the case of range. It is useful in estimating dispersion in grouped data with open ended class.

Disadvantages: IQR as a measure of dispersion is most reliable only with symmetrical data series. Unfortunately, in social sciences most of data distributions are generally asymmetrical in nature. So, its use in social sciences is usually limited to data which are moderately skewed.

```
# Detecting the outliers using IQR and removing them
#As to remove outliers through IQR we will use standard index, so changing the Date time index to standard one
df_train.reset_index(inplace = True)
# Calculating Quantile 1
Q1 = np.percentile(df_train['units'], 25, interpolation = 'midpoint')
# Calculating Quantile 3
Q3 = np.percentile(df_train['units'], 75, interpolation = 'midpoint')

# IQR
IQR = Q3 - Q1

print("Old Shape: ", df_train.shape)

# Upper set
upper = np.where(df_train['units'] >= (Q3+1.5*IQR))
# Lower set
lower = np.where(df_train['units'] <= (Q1-1.5*IQR))

''' Removing the Outliers '''
df_train.drop(upper[0], inplace = True)
df_train.drop(lower[0], inplace = True) # Don't have any affect though

print("New Shape: ", df_train.shape)


Old Shape:  (118696, 4)
New Shape:  (113709, 4)
```

- Multivariate Feature analysis:

Multivariate means involving multiple dependent variables resulting in one outcome. This explains that the majority of the problems in the real world are Multivariate. For example, in this case we can predict sales based on multiple weather related features like temperature, Snowfall, dew etc.

It is a Statistical procedure for analysis of data involving more than one type of measurement or observation. It may also mean solving problems where more than one dependent variable is analyzed simultaneously with other variables.

Apart from reviewing min, max, std etc. Multivariate feature analysis also includes feature selections.

Different options we have:

- Pairwise plots: Pairwise plots are a great way to look at multi-dimensional data, and at the same time maintain the simplicity of a two-dimensional plot. It allows the analysts to view all combinations of the variables, each in a two-dimensional plot. In this way, they can visualize all the relations and interactions among the variables on one single screen.
  Disadvantage of pairwise plot is – if we have more features then generating output takes time, size of the plot becomes huge, difficult to review

- Correlation Analysis: Often, data sets contain variables that are either related to each other or derived from each other. It is important to understand these relations that exist in the data. In statistical terms, correlation can be defined as the degree to which a pair of variables are linearly related. In some cases, it is easy for the analyst to understand that the variables are related, but in most cases, it isn't. Thus, performing a correlation analysis is very critical while examining any data. Furthermore, feeding data which has variables correlated to one another is not a good statistical practice, since we are providing multiple weightage to the same type of data. To prevent such issues, correlation analysis is a must

- Principal Component Analysis and Factor Analysis: Although machine learning is a game of predicting the result given multiple predictors, there can be times when the number of these predictors is too large. Not only is such a data set difficult to analyze, but the models formed using this are susceptible to overfitting. Therefore, it makes sense to have the number of these variables reduced. Principal component analysis (PCA) and Factor analysis are two of the common techniques used to perform such a dimension reduction.

  PCA reduces the existing number of variables, such that the new set of reduced variables capture most of the total variance present in the existing set of

variables. Many times, we usually have only two or three features in this new set of features. What this allows us is to visualize all of the initial information in 2-D or 3-D plots, and thus aid in exploratory data analysis. Furthermore, these new features are a combination of the initial features which helps us understand the variables which are important.
Therefore, PCA is such a powerful tool for analysts since they now have a much smaller feature set to deal with, and at the same time having preserved most of the information which was initially present. While PCA extracts factors based on the total variance, the Factor Analysis Method extracts factors based on the variance shared by the factors. By providing the factors based on the variance they share, Factor Analysis enables data scientists to examine the underlying trends in the data.

- Other than the above we have Spider Plots, Cluster Analysis, MANOVA (Multivariate Analysis of Variance), Discriminant Analysis, Conjoint Analysis, Multiple Regression Analysis

We have done Multivariate Feature analysis/Selection on "Phase2_EDA_Merged.ipynb", which is actually merger of train_cleaned.csv, weather.csv using key.csv file.

After merging we had 113709 entries under 47 columns, 1 dependent target column. So dimensionality here too much, we will need to reduce the same using careful approach.

After train and test spilt we have 90967 train and 22742 test data.

By checking descriptive statistics of the below columns we can find the below:

```
['tmax', 'tmin', 'tavg', 'depart', 'dewpoint', 'wetbulb', 'heat', 'cool'
, 'snowfall', 'preciptotal', 'stnpressure', 'sealevel', 'resultspeed',
'resultdir', 'avgspeed']
```

	tmax	tmin	tavg	depart	dewpoint	wetbulb	heat	cool	snowfall	preciptotal	stnpressure	sealevel	resultspeed	resultdir	avgspeed
count	90967.000000	90967.000000	90967.000000	90967.000000	90967.000000	90967.000000	90967.000000	90967.000000	90967.000000	90967.000000	90967.000000	90967.000000	90967.000000	90967.000000	90967.000000
mean	71.876549	49.826157	61.173788	1.569481	45.933393	53.237537	9.538118	5.711907	0.011119	0.077323	28.752342	30.014228	6.374150	18.366050	8.008847
std	19.146519	19.025444	18.626153	4.294729	19.387231	16.659199	13.385621	7.667846	0.248512	0.296811	1.814396	0.185497	4.194766	9.621782	3.898091
min	-11.000000	-21.000000	-16.000000	-32.000000	-24.000000	-15.000000	0.000000	0.000000	0.000000	0.000000	23.720000	29.160000	0.000000	1.000000	0.000000
25%	60.000000	36.000000	49.000000	1.500000	31.000000	41.000000	0.000000	0.000000	0.000000	0.000000	28.780000	29.900000	3.200000	13.000000	5.200000
50%	75.000000	52.000000	64.000000	1.500000	49.000000	57.000000	1.000000	0.000000	0.000000	0.000000	29.330000	30.000000	5.500000	18.000000	7.400000
75%	87.000000	66.000000	77.000000	1.500000	63.000000	67.000000	16.000000	12.000000	0.000000	0.000000	29.810000	30.120000	8.800000	25.000000	9.900000
max	114.000000	88.000000	100.000000	33.000000	77.000000	80.000000	81.000000	35.000000	16.200000	7.360000	30.610000	30.800000	28.400000	36.000000	28.700000

We found that we will need to standardize the data as the data are in different scales.

And as we do not have properly distributed data we have not done Normalization.

	tmax	tmin	tavg	depart	dewpoint	wetbulb	heat	cool	snowfall	preciptotal	stnpressure	sealevel	resultspeed	resultdir	avgspeed
count	9.096700e+04	9.096700e+04	9.096700e+04	9.096700e+04	9.096700e+04	9.096700e+04	9.096700e+04	9.096700e+04	9.096700e+04	9.096700e+04	9.096700e+04	9.096700e+04	9.096700e+04	9.096700e+04	9.096700e+04
mean	1.957240e-16	1.495317e-17	3.683372e-17	4.394218e-17	1.891725e-17	2.273366e-16	-8.019157e-16	1.173846e-15	-1.688171e-15	-7.866953e-16	4.578499e-15	1.789922e-14	-1.166956e-16	-8.666787e-17	-1.257875e-16
std	1.000005e+00	1.000005e+00	1.000005e+00	1.000005e+00	1.000005e+00	1.000005e+00	1.000005e+00	1.000005e+00	1.000005e+00	1.000005e+00	1.000005e+00	1.000005e+00	1.000005e+00	1.000005e+00	1.000005e+00
min	-4.328568e+00	-3.722728e+00	-4.143325e+00	-7.816480e+00	-3.607208e+00	-4.096110e+00	-7.125684e-01	-7.449208e-01	-4.474429e-02	-2.605151e-01	-2.773579e+00	-4.605112e+00	-1.519557e+00	-1.804878e+00	-2.054567e+00
25%	-6.203015e-01	-7.267232e-01	-6.535893e-01	-1.617835e-02	-7.702738e-01	-7.345854e-01	-7.125684e-01	-7.449208e-01	-4.474429e-02	-2.605151e-01	1.524382e-02	-6.157996e-01	-7.566971e-01	-5.577012e-01	-7.205738e-01
50%	1.631351e-01	1.142604e-01	1.517343e-01	-1.617835e-02	1.581775e-01	2.258502e-01	-6.378610e-01	-7.449208e-01	-4.474429e-02	-2.605151e-01	3.183767e-01	-7.670330e-02	-2.083918e-01	-3.804413e-02	-1.561920e-01
75%	7.898843e-01	8.501211e-01	8.496815e-01	-1.617835e-02	8.803063e-01	8.261225e-01	4.827507e-01	8.200645e-01	-4.474429e-02	-2.605151e-01	5.829291e-01	5.702122e-01	5.783071e-01	6.894758e-01	4.851511e-01
max	2.200070e+00	2.006474e+00	2.084511e+00	7.318434e+00	1.602435e+00	1.606476e+00	5.338734e+00	3.819620e+00	6.514370e+01	2.453651e+01	1.023850e+00	4.236067e+00	5.250822e+00	1.832721e+00	5.308051e+00

- ✓ To reduce the dimensionality we have used Variance Threshold Feature selector And Pearson Correlation

- ✓ Variance Threshold Feature selector:
  This removes all low-variance features. This feature selection algorithm looks only at the features (X), not the desired outputs (y), and can thus be used for unsupervised learning. By using this we can remove 10 constant columns, as shown below (Threshold set as 0.001)

```
'''
The above features all have different medians, quartiles, and ranges – completely different distributions. We cannot compare these features to each other.
One method we can use is normalizing all features by dividing them by their mean:
'''
normalized_df = df_train_key_weather2 / df_train_key_weather2.mean()

from sklearn.feature_selection import VarianceThreshold
var_thres=VarianceThreshold(threshold=0.003)
var_thres.fit(normalized_df)
```

```
VarianceThreshold(threshold=0.003)
```

```
var_thres.get_support()
```

```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True, False,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True])
```

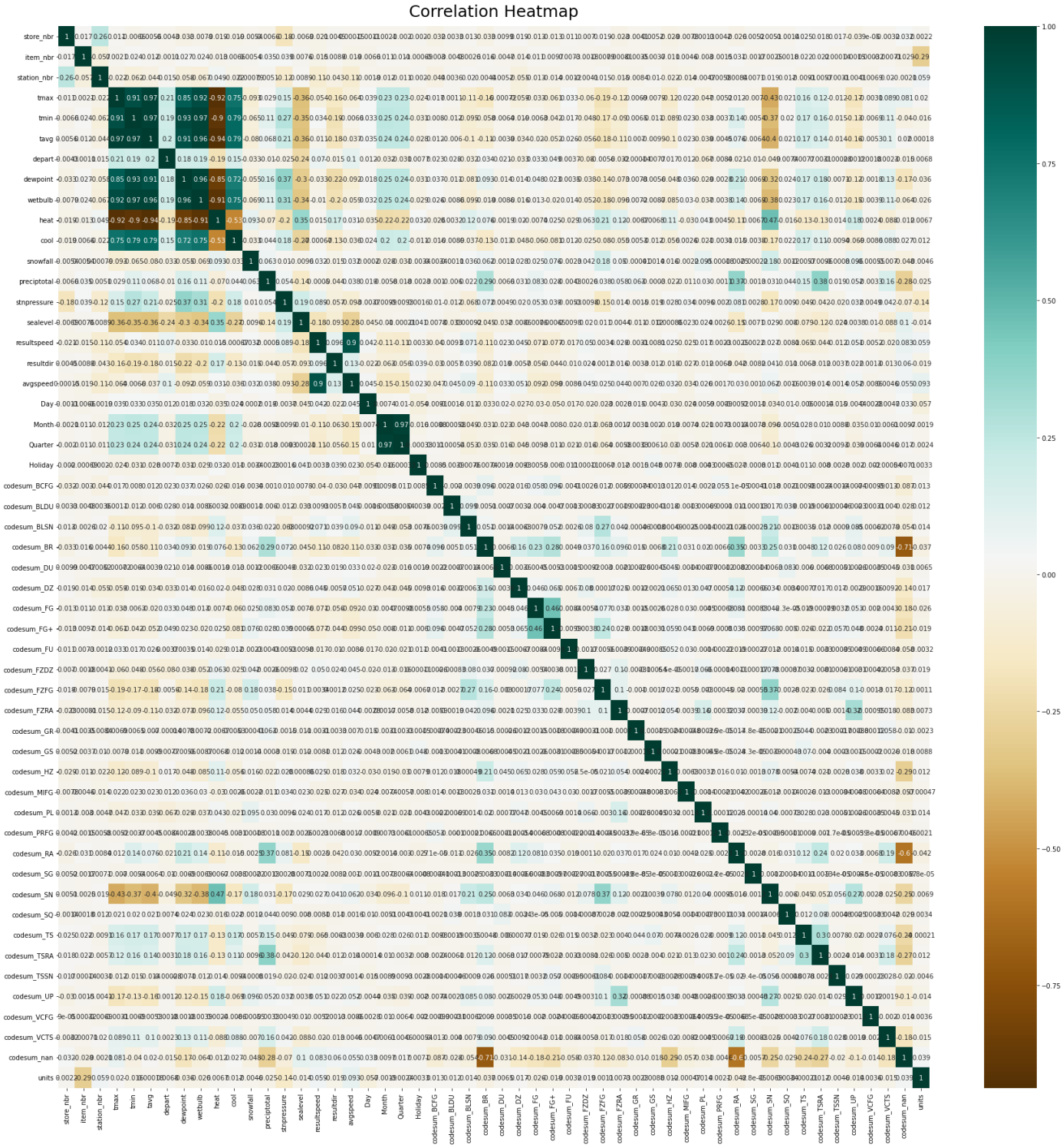
```
df_train_key_weather2.columns[var_thres.get_support()]
constant_columns = [column for column in df_train_key_weather2.columns
                    if column not in df_train_key_weather2.columns[var_thres.get_support()]]

print(len(constant_columns))
```

```
1
```

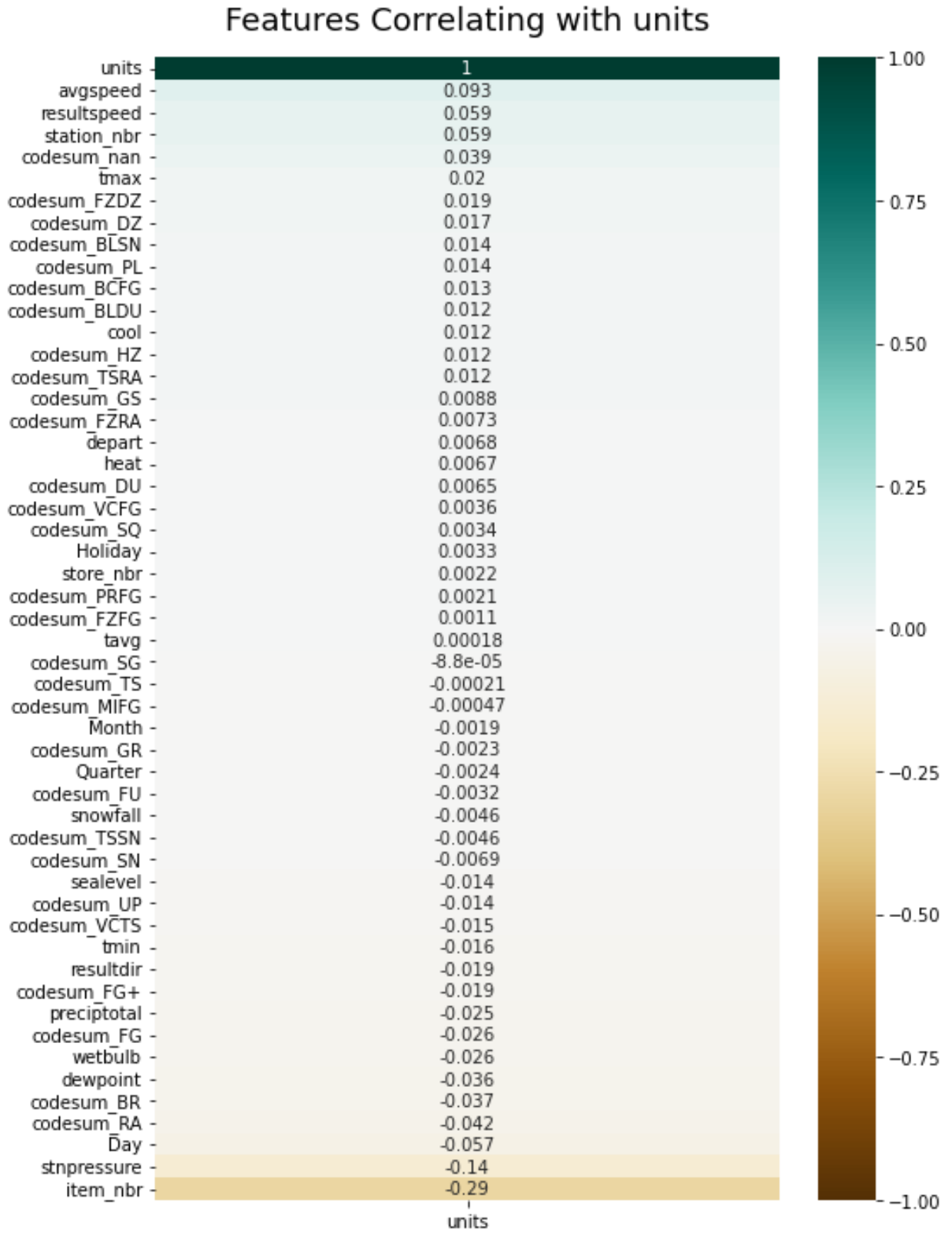
```
constant_columns
```

```
['sealevel']
```

✓ Pearson Correlation:
Using this feature we are going to keep only 1 feature out of multiple correlated features, again this will help us reducing the dimensionality.



We have used below function to read from the above and pick one feature among all other corelated features. After removing constants and corelated features we have 33 features with us now.


Features Correlating with units
units 1
avgspeed 0.093
resultspeed 0.059
station_nbr 0.059
codesum_nan 0.039
tmax 0.02
codesum_FZDZ 0.019
codesum_DZ 0.017
codesum_BLSN 0.014
codesum_PL 0.014
codesum_BCFG 0.013
codesum_BLDU 0.012
cool 0.012
codesum_HZ 0.012
codesum_TSRA 0.012
codesum_GS 0.0088
codesum_FZRA 0.0073
depart 0.0068
heat 0.0067
codesum_DU 0.0065
codesum_VCFG 0.0036
codesum_SQ 0.0034
Holiday 0.0033
store_nbr 0.0022
codesum_PRFG 0.0021
codesum_FZFG 0.0011
tavg 0.00018
codesum_SG -8.8e-05
codesum_TS -0.00021
codesum_MIFG -0.00047
Month -0.0019
codesum_GR -0.0023
Quarter -0.0024
codesum_FU -0.0032
snowfall -0.0046
codesum_TSSN -0.0046
codesum_SN -0.0069
sealevel -0.014
codesum_UP -0.014
codesum_VCTS -0.015
tmin -0.016
resultdir -0.019
codesum_FG+ -0.019
preciptotal -0.025
codesum_FG -0.026
wetbulb -0.026
dewpoint -0.036
codesum_BR -0.037
codesum_RA -0.042
Day -0.057
stnpressure -0.14
item_nbr -0.29
units
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

```
[15] # with the following function we can select highly correlated features
     # it will remove the first feature that is correlated with anything other feature

     def correlation(dataset, threshold):
         col_corr = set()  # Set of all the names of correlated columns
         corr_matrix = dataset.corr()
         for i in range(len(corr_matrix.columns)):
             for j in range(i):
                 if abs(corr_matrix.iloc[i, j]) > threshold: # we are interested in absolute coeff value
                     colname = corr_matrix.columns[i]  # getting the name of column
                     col_corr.add(colname)
         return col_corr
```

```
[16] corr_features = correlation(X_train_stand, 0.92)
     len(set(corr_features))#6
     corr_features

     {'dewpoint', 'heat', 'tavg', 'wetbulb'}
```

Feature encoding

The performance of a machine learning model not only depends on the model and the hyper parameters but also on how we process and feed different types of variables to the model. Since most machine learning models only accept numerical variables, preprocessing the categorical variables becomes a necessary step. We need to convert these categorical variables to numbers such that the model is able to understand and extract valuable information.

Categorical variables are usually represented as ‘strings’ or ‘categories’ and are finite in number. Further, we can see there are two kinds of categorical data-

Ordinal Data: The categories have an inherent order. In Ordinal data, while encoding, we should retain the information regarding the order in which the category is provided.

Nominal Data: The categories do not have an inherent order. While encoding Nominal data, we have to consider the presence or absence of a feature. In such a case, no notion of order is present.

- Options we have to do feature encoding:

- Label Encoding or Ordinal Encoding:
  We use this categorical data encoding technique when the categorical feature is ordinal. In this case, retaining the order is important. Hence encoding should reflect the sequence.
  In Label encoding, each label is converted into an integer value.

- One Hot Encoding:

We use this categorical data encoding technique when the features are nominal (do not have any order). In one hot encoding, for each level of a categorical feature, we create a new variable. Each category is mapped with a binary variable containing either 0 or 1. Here, 0 represents the absence, and 1 represents the presence of that category.
These newly created binary features are known as Dummy variables. The number of dummy variables depends on the levels present in the categorical variable.

- Dummy Encoding:
  Dummy coding scheme is similar to one-hot encoding. This categorical data encoding method transforms the categorical variable into a set of binary variables (also known as dummy variables). In the case of one-hot encoding, for N categories in a variable, it uses N binary variables. The dummy encoding is a small improvement over one-hot-encoding. Dummy encoding uses N-1 features to represent N labels/categories.

- Other than the above techniques we have - Effect Encoding, Hash Encoder, Binary Encoding, Base N Encoding, Target Encoding

- Here in this project we have a feature namely “codesum”, which is multi-option nominal data. It has 29 unique values.
  We have used one hot encoding for this.

  Advantage: One-Hot-Encoding has the advantage that the result is binary rather than ordinal and that everything sits in an orthogonal vector space.

  Disadvantage: The disadvantage is that for high cardinality, the feature space can really blow up quickly and you start fighting with the curse of dimensionality.

  Hence to reduce the dimensionality we have used Variance Threshold Feature selector And Pearson Correlation feature.
  We have not used PCA as we were not sure what number we should keep as parameter (i.e. how many features we should have).

Advanced Feature encoding mechanisms

There are other advanced feature encoding techniques are also available, based on the requirement below can be used.

- Matrix Factorization

We can do sparse encoding for more-interpretable feature-selecting representations in probabilistic matrix factorization.

Dimensionality reduction methods for count data are critical to a wide range of applications in medical informatics and other fields where model interpretability is paramount. For such data, hierarchical Poisson matrix factorization (HPF) and other sparse probabilistic non-negative matrix factorization (NMF) methods are considered to be interpretable generative models. They consist of sparse transformations for decoding their learned representations into predictions. However, sparsity in representation decoding does not necessarily imply sparsity in the encoding of representations from the original data features. HPF is often incorrectly interpreted in the literature as if it possesses encoder sparsity. The distinction between decoder sparsity and encoder sparsity is subtle but important. Due to the lack of encoder sparsity, HPF does not possess the column-clustering property of classical NMF -- the factor loading matrix does not sufficiently define how each factor is formed from the original features. We address this deficiency by self-consistently enforcing encoder sparsity, using a generalized additive model (GAM), thereby allowing one to relate each representation coordinate to a subset of the original data features. In doing so, the method also gains the ability to perform feature selection.

- Encoding using Deep-Learning:

Many machine learning algorithms and almost all deep learning architectures are incapable of processing plain texts in their raw form. This means that their input to the algorithms must be numerical in order to solve classification or regression problems. Hence, it is necessary to encode these categorical variables into numerical values using encoding techniques. Categorical features are common and often of high cardinality. One-hot encoding in such circumstances leads to very high dimensional vector representations, raising memory and computability concerns for machine learning models. This paper proposes a deep-learned embedding technique for categorical features encoding on categorical datasets. Our technique is a distributed representation for categorical features where each category is mapped to a distinct vector, and the properties of the vector are learned while training a neural network. First, we create a data vocabulary that includes only categorical data, and then we use word tokenization to make each categorical data a single word. After that, feature learning is introduced to map all of the categorical data from the vocabulary to word vectors. Three different datasets provided by the University of California Irvine (UCI) are used for training. The

experimental results show that the proposed deep-learned embedding technique for categorical data provides a higher F1 score of 89% than 71% of one-hot encoding, in the case of the long short-term memory (LSTM) model. Moreover, the deep-learned embedding technique uses less memory and generates fewer features than one-hot encoding.

High-dimensional data visualization:

- t-SNE:
- For high dimensional data visualization we can use t-SNE, which maps the multi-dimensional data to a lower-dimensional space, the input features are not useful and convey no information. Thus, no inferences can be drawn simply from the output of t-SNE alone.
  It is a Dimensionality Reduction technique and not a Clustering technique. But its output can be used as an input feature for other classification or clustering algorithms.

Phase 3: Modelling and Error Analysis

At this stage we will use the dataset we have created at our EDA stage and use the same for ML modelling.
The initial main data set that is “train.csv” which contains the details of Units sold, has imbalance dataset. That is maximum of the rows have 0 units sold and very few (comparitiveliy) rows have at least one unit sold.
The main data set "train.csv" has 4617600 rows of data. Among the total rows we have 4498904 rows where 0 units only sold. This shows how imbalanced data set is.
To overcome the issue of imbalanced dataset we can refer the below techniques & others –

- Choose Proper Evaluation Metric
- Resampling (Oversampling and Under sampling)
- Synthetic Minority Oversampling Technique or SMOTE/ Synthetic Minority Over-Sampling Technique for Regression with Gaussian Noise SMOGN
- BalancedBaggingClassifier etc.

But here in this case we have used a classifier algorithm first, just to check whether query point's output is 0 or not. If 0 then it will share the same information as an output against the query point raised, if the initial classification model predict ‘1’ then it will pass back all the query point(s) to Regression model(s).

We are going to use **Logistic Regression as initial classification model**, as its accuracy & F1-Score is better compare to other classifiers as per the below shared details (https://analyticsindiamag.com/7-types-classification-algorithms/)

Classification Algorithms	Accuracy	F1-Score
Logistic Regression	84.60%	0.6337
Naïve Bayes	80.11%	0.6005
Stochastic Gradient Descent	82.20%	0.5780
K-Nearest Neighbours	83.56%	0.5924
Decision Tree	84.23%	0.6308
Random Forest	84.33%	0.6275
Support Vector Machine	84.09%	0.6145

Details of matrixes used for the above mentioned comparison –

- Accuracy: (True Positive + True Negative) / Total Population
- Accuracy is a ratio of correctly predicted observation to the total observations. Accuracy is the most intuitive performance measure.
- True Positive: The number of correct predictions that the occurrence is positive
- True Negative: The number of correct predictions that the occurrence is negative

- F1-Score: (2 x Precision x Recall) / (Precision + Recall)
- F1-Score is the weighted average of Precision and Recall used in all types of classification algorithms. Therefore, this score takes both false positives and false negatives into account. F1-Score is usually more useful than accuracy, especially if you have an uneven class distribution.
- Precision: When a positive value is predicted, how often is the prediction correct?
- Recall: When the actual value is positive, how often is the prediction correct?

Below are some of the advantages and disadvantages for the classification models we considered and based on the below reasons we have picked Logistic Regression over other models.

- **Logistic Regression**

Definition: Logistic regression is a machine learning algorithm for classification. In this algorithm, the probabilities describing the possible outcomes of a single trial are modelled using a logistic function.

Advantages: Logistic regression is designed for this purpose (classification), and is most useful for understanding the influence of several independent variables on a single outcome variable.

Disadvantages: Works only when the predicted variable is binary, assumes all predictors are independent of each other and assumes data is free of missing values.

- **Naïve Bayes**

Definition: Naive Bayes algorithm based on Bayes' theorem with the assumption of independence between every pair of features. Naive Bayes classifiers work well in many real-world situations such as document classification and spam filtering.

Advantages: This algorithm requires a small amount of training data to estimate the necessary parameters. Naive Bayes classifiers are extremely fast compared to more sophisticated methods.

Disadvantages: Naive Bayes is known to be a bad estimator.

- **Stochastic Gradient Descent**

Definition: Stochastic gradient descent is a simple and very efficient approach to fit linear models. It is particularly useful when the number of samples is very large. It supports different loss functions and penalties for classification.

Advantages: Efficiency and ease of implementation.

Disadvantages: Requires a number of hyper-parameters and it is sensitive to feature scaling.

- **K-Nearest Neighbors**

Definition: Neighbors based classification is a type of lazy learning as it does not attempt to construct a general internal model, but simply stores instances of the training data. Classification is computed from a simple majority vote of the k nearest neighbors of each point.

Advantages: This algorithm is simple to implement, robust to noisy training data, and effective if training data is large.

Disadvantages: Need to determine the value of K and the computation cost is high as it needs to compute the distance of each instance to all the training samples.

- **Decision Tree**

Definition: Given a data of attributes together with its classes, a decision tree produces a sequence of rules that can be used to classify the data.

Advantages: Decision Tree is simple to understand and visualise, requires little data preparation, and can handle both numerical and categorical data.

Disadvantages: Decision tree can create complex trees that do not generalise well, and decision trees can be unstable because small variations in the data might result in a completely different tree being generated.

- **Random Forest**

Definition: Random forest classifier is a meta-estimator that fits a number of decision trees on various sub-samples of datasets and uses average to improve the predictive accuracy of the model and controls over-fitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement.

Advantages: Reduction in over-fitting and random forest classifier is more accurate than decision trees in most cases.

Disadvantages: Slow real time prediction, difficult to implement, and complex algorithm.

- **Support Vector Machine**

Definition: Support vector machine is a representation of the training data as points in space separated into categories by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

Advantages: Effective in high dimensional spaces and uses a subset of training points in the decision function so it is also memory efficient.

Disadvantages: The algorithm does not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation.

Metrics:

For the classification model we have used F1-Score as main metric, along with others. The reason of selecting F1-Score as main metric is –

The accuracy of a classifier is the total number of correct predictions by the classifier divided by the total number of predictions. This may be good enough for a well-balanced class but not ideal for the imbalanced class problem. The other metrics such as **precision** is the measure of how accurate the classifier's prediction of a specific class and **recal**l is the measure of the classifier's ability to identify a class.

For an imbalanced class dataset F1 score is a more appropriate metric. It is the harmonic mean of precision and recall and the expression is –

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

So, if the classifier predicts the minority class but the prediction is erroneous and false-positive increases, the precision metric will be low and so as F1 score. Also, if the classifier identifies the minority class poorly, i.e. more of this class wrongfully predicted as the majority class then false negatives will increase, so recall and F1 score will low. F1 score only increases if both the number and quality of prediction improves.

F1 score keeps the balance between precision and recall and improves the score only if the classifier identifies more of a certain class correctly.

- **Setting up Classification Model**

- Creating a new dataset by merging 3 different data sets

Merging Data sets for the Classification Model

```
[ ]  df_train_key = df_train.merge(df_key, how='left', on='store_nbr')
     df_train_key_weather = df_train_key.merge(df_weather_cleaned, how='left', on=['station_nbr','date'])
     #df_train_key_weather.head(2)
```

- As per EDA removing not required columns, outliers, setting up new feature “flag” which is going to hold ‘0’ if units = 0 and going to hold ‘1’ is units > 0.

```
df_Classifier_cleand.loc[df_Classifier_cleand['units'] == 0, 'flag'] = 0
df_Classifier_cleand.loc[df_Classifier_cleand['units']  > 0, 'flag'] = 1
df_Classifier_cleand.flag = df_Classifier_cleand.flag.astype('int64')
bkupdf_Classifier_cleand = df_Classifier_cleand.copy()
df_Classifier_cleand.head(2)
```

- Creating time based train-test split, fitting the model & predicting

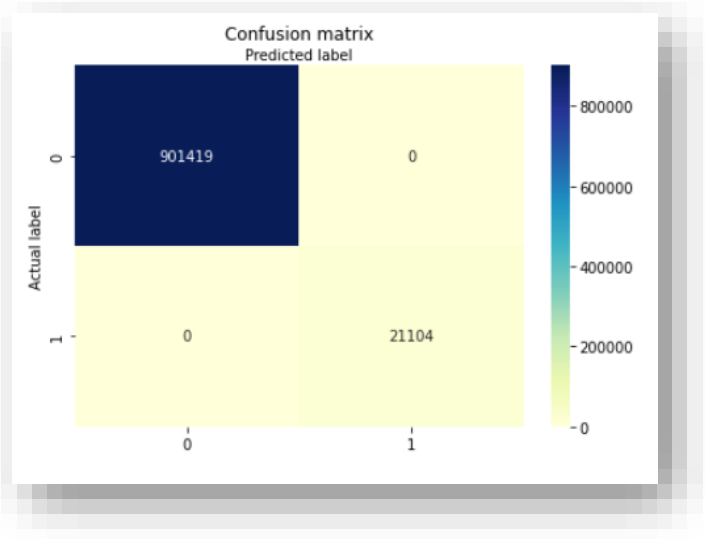
```
# create training and testing vars
X_train, X_test, y_train, y_test = train_test_split(df_Classifier_LgRgresn, y, test_size=0.2,shuffle=False,stratify=None)
```

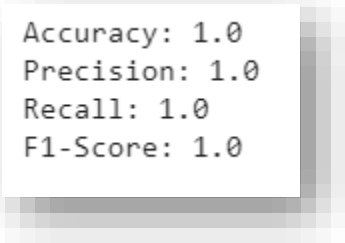
```
model = LogisticRegression(solver='liblinear', random_state=6)
model.fit(X_train, y_train)
```

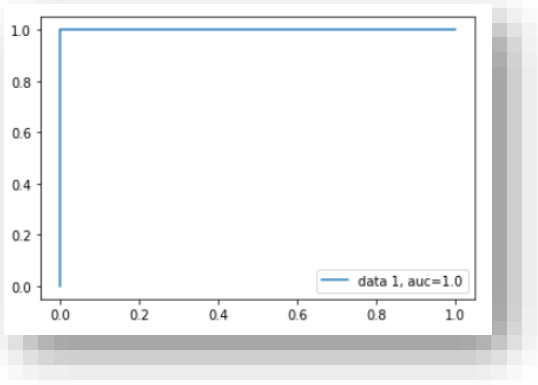
```
LogisticRegression(random_state=6, solver='liblinear')
```

```
y_pred=model.predict(X_test)
```

- Checking the metrics







After using the classification model (Logistic regression), which we used to segregate between sales or no sales to deal with imbalanced data, we will now use the regression models.
Starting with easier one to complex one.

Below are some of the Regression models & corresponding advantages and dis advantages.

Regression Model	Advantages	Disadvantages
Linear Regression	1. Works well irrespective of the dataset size. 2. Gives information about the relevance of features.	1. The assumptions of Linear Regression.
Polynomial Regression	1. Works on any size of the dataset. 2. Works very well on non-linear problems.	1. We need to choose the right polynomial degree for good bias/ variance tradeoff.
Support Vector Regression	1. Easily adaptable. 2. Works very well on non-linear problems. 3. Not biased by outliers (object that deviates significantly from the rest).	1. Compulsory to apply feature scaling. 2. Not well known. 3. Difficult to understand.
Decision Tree Regression	1. Interpretability. 2. Works well on both linear and non-linear problems. 3. No need to apply feature scaling.	1. Poor results on small datasets. 2. Overfitting can easily occur.
Random Forest Regression	1. Powerful. 2. Accurate. 3. Good performance on many problems including non-linear.	1. No interpretability. 2. Overfitting can easily occur. 3. We need to choose the number of trees.

Ref: https://www.geeksforgeeks.org/advantages-and-disadvantages-of-different-regression-models/

a) We have first used the simple Linear Regression Model as starter:
Accuracy of this model is not good

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
# Model initialization
model_lnReg = LinearRegression()
# Fit the data(train the model)
model_lnReg.fit(X_train, y_train)
# Predict
y_predlnReg = model_lnReg.predict(X_test)
# printing values
#print('Slope:' ,model_lnReg.coef_)
#print('\nIntercept:', model_lnReg.intercept_)
Evaluation_Metrics (model_lnReg,y_test,y_predlnReg,"Linear Regression")
```

```
'Mean Absolute Error(MAE)' for the model -  Linear Regression is:  20.51493283305906
'Mean Squared Error(MSE)' for the model -  Linear Regression is:  672.9435992193156
'Root Mean Squared Error(RMSE)' for the model -  Linear Regression is:  25.94115647420746
NO 'Root Mean Squared Log Error(RMSLE)' for the model -  Linear Regression  as, prediction have negative values.
'R Squared (R2)' for the model -  Linear Regression is:  0.10794408408210399

----------------------------------------------------------------------------------------

'Model Train Score' for the model -  Linear Regression is:  0.12265193249704864

----------------------------------------------------------------------------------------

'Model Test Score' for the model -  Linear Regression is:  0.10794408408210399
```

```
[ ] cross_validate(LinearRegression())

    Mean RMSE: 30.547 (0.091)

    ----------------------------------------------------------------------------------------
    Mean RMSLE: nan (nan)

    ----------------------------------------------------------------------------------------
    Mean MAE: 24.365 (0.096)

    ----------------------------------------------------------------------------------------
```

b) Then we used Lasso Regression – that Linear regression with L1 regularization
Again accuracy was not that good.

A brief details of regularization and why Lasso is used –
In order to create less complex (parsimonious) model when you have a large number of features in your dataset, some of the Regularization techniques used to address over-fitting and feature selection are:
1. L1 Regularization
2. L2 Regularization

A regression model that uses L1 regularization technique is called Lasso Regression and model which uses L2 is called Ridge Regression.
The key difference between these two is the penalty term.

Ridge regression adds “squared magnitude” of coefficient as penalty term to the loss function. Here the highlighted part represents L2 regularization element.

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

Cost function

Here, if lambda is zero then you can imagine we get back OLS. However, if lambda is very large then it will add too much weight and it will lead to under-fitting. Having said that it’s important how lambda is chosen. This technique works very well to avoid over-fitting issue.

Lasso Regression (Least Absolute Shrinkage and Selection Operator) adds “absolute value of magnitude” of coefficient as penalty term to the loss function.

$$\sum_{i=1}^{n}(Y_i - \sum_{j=1}^{p} X_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

Cost function

Again, if lambda is zero then we will get back OLS whereas very large value will make coefficients zero hence it will under-fit.
The key difference between these techniques is that **Lasso shrinks the less important feature’s coefficient to zero thus, removing some feature altogether. So, this works well for feature selection in case we have a huge number of features**.
Traditional methods like cross-validation, stepwise regression to handle overfitting and perform feature selection work well with a small set of features but these techniques are a great alternative when we are dealing with a large set of features.

```
#X_train, X_test, y_train, y_test
from sklearn import linear_model
model_lassoReg = linear_model.Lasso(alpha=1,max_iter=5,tol=0.1,fit_intercept=True)
model_lassoReg.fit(X_train,y_train)
y_pred_LassoReg = model_lassoReg.predict(X_test)
Evaluation_Metrics (model_lassoReg,y_test,y_pred_LassoReg,"Linear Lasso Regression - L1 Regularization")
```

```
'Mean Absolute Error(MAE)' for the model -  Linear Lasso Regression - L1 Regularization is:  20.836003300331274
'Mean Squared Error(MSE)' for the model -  Linear Lasso Regression - L1 Regularization is:  679.1819263910813
'Root Mean Squared Error(RMSE)' for the model -  Linear Lasso Regression - L1 Regularization is:  26.0611190548503
'Root Mean Squared Log Error(RMSLE)' for the model -  Linear Lasso Regression - L1 Regularization is:  1.23204633912395
'R Squared (R2)' for the model -  Linear Lasso Regression - L1 Regularization is:  0.09967454014787103

----------------------------------------------------------------------------------------

'Model Train Score' for the model -  Linear Lasso Regression - L1 Regularization is:  0.11131671313921088

----------------------------------------------------------------------------------------

'Model Test Score' for the model -  Linear Lasso Regression - L1 Regularization is:  0.09967454014787103
```

We tried hyper parameter tuning for this model, but the result is not satisfactory –

Hyperparameter Tuning for Lasso reg

+ Code + Text

```
model_params = {
    'Lasso': {
        'model': linear_model.Lasso(),
        'params' : {
          'alpha': [1,10,20,30,40,50,60,70,80],
          'max_iter': [5,10,20,30,40,50],
           'fit_intercept': [True,False]
        }
    }
}
hyperparametertuning(model_params)
```

```
+----+---------+--------------+-------------------------------------------------------+
|    | model   |   best_score | best_params                                           |
|----+---------+--------------+-------------------------------------------------------|
|  0 | Lasso   |     0.106515 | {'alpha': 1, 'fit_intercept': True, 'max_iter': 5} |
+----+---------+--------------+-------------------------------------------------------+
```

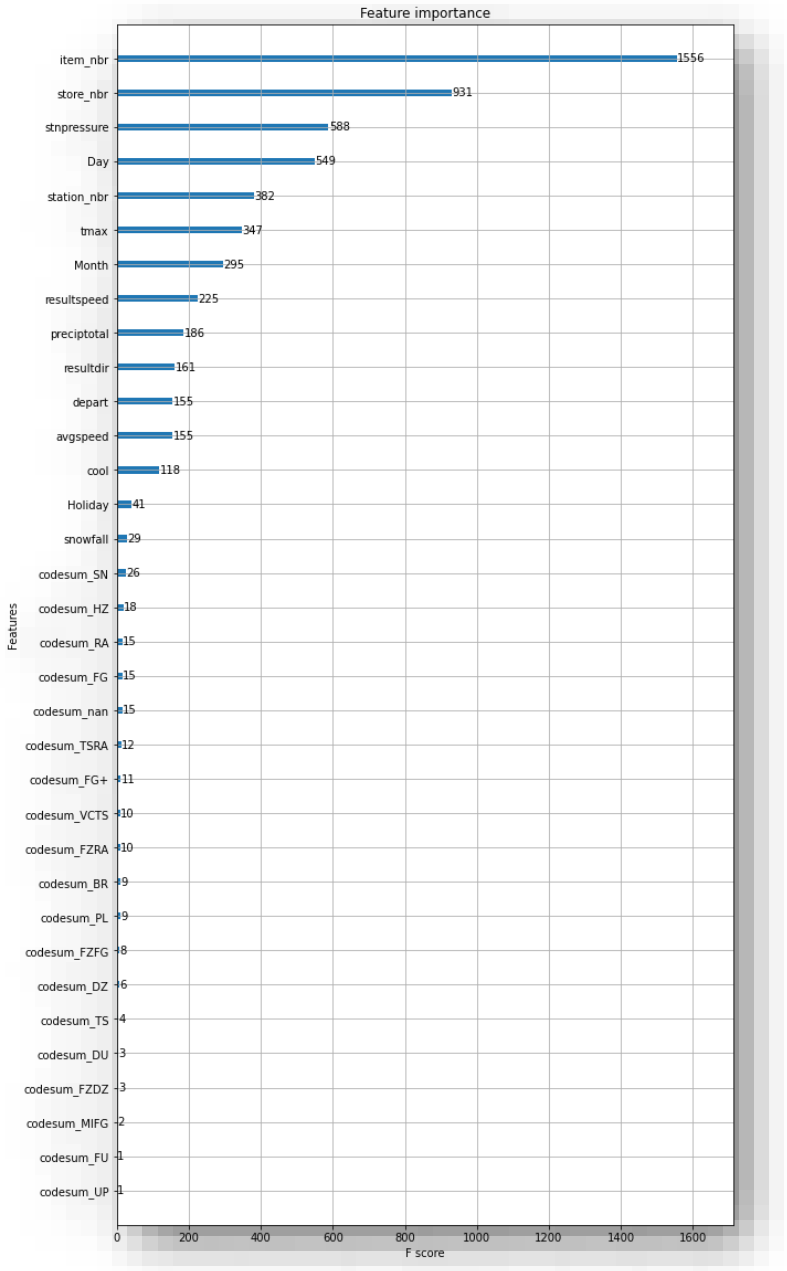
```
[ ] cross_validate(model_lassoReg)
```

```
Mean RMSE: 30.729 (0.080)

----------------------------------------------------------------------------------------
Mean RMSLE: 1.384 (0.010)

----------------------------------------------------------------------------------------
Mean MAE: 24.780 (0.091)

----------------------------------------------------------------------------------------
```

c) Next Model tested was XGBRegressor:

```
model = XGBRegressor(eta=.1,gamma=1,max_depth=6)
model.fit(X_train, y_train)
# plot feature importance
```

We checked- which features are the least important, below is our findings.



XGBRegressor performed well –

```
y_predXGBRegressor = model.predict(X_test)
Evaluation_Metrics (model,y_test,y_predXGBRegressor,"XGBRegressor")
```

```
'Mean Absolute Error(MAE)' for the model -  XGBRegressor is:  16.433493532214836
'Mean Squared Error(MSE)' for the model -  XGBRegressor is:  627.0443722182928
'Root Mean Squared Error(RMSE)' for the model -  XGBRegressor is:  25.040854063276132
NO 'Root Mean Squared Log Error(RMSLE)' for the model -  XGBRegressor  as, prediction have negative values.
'R Squared (R2)' for the model -  XGBRegressor is:  0.16878822767722979

----------------------------------------------------------------------------------------------

'Model Train Score' for the model -  XGBRegressor is:  0.7771205501968009

----------------------------------------------------------------------------------------------

'Model Test Score' for the model -  XGBRegressor is:  0.16878822767722979
```

```
cross_validate(model)
```

```
Mean RMSE: 15.745 (0.141)

----------------------------------------------------------------------------------------------
Mean RMSLE: nan (nan)

----------------------------------------------------------------------------------------------
Mean MAE: 10.406 (0.101)

----------------------------------------------------------------------------------------------
```

We tried hyper parameter tuning for this model, below is the result –

```
model_params = {
    'XGBRegressor': {
        'model': XGBRegressor(),
        'params' : {
          'eta': [.1,.2],
          'gamma': [0,1],
           'max_depth': [4,6,8]
           #'min_child_weight':[1,2]
        }
    }
}
hyperparametertuning(model_params)
```

```
+----+--------------+--------------+-----------------------------------------+
|    | model        |   best_score | best_params                             |
|----+--------------+--------------+-----------------------------------------|
|  0 | XGBRegressor |     0.730481 | {'eta': 0.1, 'gamma': 1, 'max_depth': 6} |
+----+--------------+--------------+-----------------------------------------+
```

d) Next we tried with SVR (Support vector regressor):

```
[ ] #X_train, X_test, y_train, y_test
    from sklearn.svm import SVR
    regressorSVR = SVR(kernel = 'rbf') #kernel = 'rbf'
    regressorSVR.fit(X_train, y_train)

    SVR()

[ ] y_predSVR = regressorSVR.predict(X_test)
    Evaluation_Metrics (regressorSVR,y_test,y_predSVR,"SVR (Support vector regressor)")

    'Mean Absolute Error(MAE)' for the model -  SVR (Support vector regressor) is:  18.818406797103272
    'Mean Squared Error(MSE)' for the model -  SVR (Support vector regressor) is:  673.7867578789894
    'Root Mean Squared Error(RMSE)' for the model -  SVR (Support vector regressor) is:  25.95740275680503
    NO 'Root Mean Squared Log Error(RMSLE)' for the model -  SVR (Support vector regressor)  as, prediction have negative values.
    'R Squared (R2)' for the model -  SVR (Support vector regressor) is:  0.10682639060631793

    ----------------------------------------------------------------------------------------

    'Model Train Score' for the model -  SVR (Support vector regressor) is:  0.09036113892631403

    ----------------------------------------------------------------------------------------

    'Model Test Score' for the model -  SVR (Support vector regressor) is:  0.10682639060631793
```

We tried hyper parameter tuning for this model, below is the result –

```
model_params = {
    'SVR': {
        'model': SVR(),
        'params' : {
            'kernel': [ 'rbf','poly'],#'linear','poly', , 'sigmoid', 'precomputed'
            'gamma': ['scale','auto'],
            'C': [1,2,3]
        }
    }
}
hyperparametertuning(model_params)
```

	model	best_score	best_params
0	SVR	0.0911346	{'C': 3, 'gamma': 'scale', 'kernel': 'rbf'}

e) Then the next model tried is Random Forest:

```
# Fitting Random Forest Regression to the dataset
# import the regressor
from sklearn.ensemble import RandomForestRegressor

 # create regressor object
RFregressor = RandomForestRegressor(n_estimators = 100, random_state = 0)

# fit the regressor with x and y data
RFregressor.fit(X_train, y_train)
```

```
RandomForestRegressor(random_state=0)
```

```
y_predRFregressor = RFregressor.predict(X_test)
Evaluation_Metrics (RFregressor,y_test,y_predRFregressor,"Random Forest")
```

```
'Mean Absolute Error(MAE)' for the model -  Random Forest is:  18.529043620083392
'Mean Squared Error(MSE)' for the model -  Random Forest is:  745.7857656063524
'Root Mean Squared Error(RMSE)' for the model -  Random Forest is:  27.309078446669567
'Root Mean Squared Log Error(RMSLE)' for the model -  Random Forest is:  1.190416229245406
'R Squared (R2)' for the model -  Random Forest is:  0.011384304734749162

----------------------------------------------------------------------------------------------

'Model Train Score' for the model -  Random Forest is:  0.9684715215735444

----------------------------------------------------------------------------------------------

'Model Test Score' for the model -  Random Forest is:  0.011384304734749162
```

After trying with few models, and using cross validation, Hyper tuning we found XGBRegressor is the most suitable one –

	model	best_score	RMSE	Cross Val - RMSE	Best_Params
0	Lasso Regression	0.106515	26.0611	30.729	{'alpha': 1, 'fit_intercept': True, 'max_iter': 5}
1	XGBRegressor	0.730481	25.0408	15.745	{'eta': 0.1, 'gamma': 1, 'max_depth': 6}
2	SVR (Support vector regressor)	0.091134	25.9574	31.229	{'C': 3, 'gamma': 'scale', 'kernel': 'rbf'}
3	Random Forest	0.300001	27.3091	15.488	best_params

Metrics preferred:

- **Mean Absolute Error (MAE)**

Mean absolute error, also known as L1 loss is one of the simplest loss functions and an easy-to-understand evaluation metric. It is calculated by taking the absolute difference between the predicted values and the actual values and averaging it across the dataset. Mathematically speaking, it is the arithmetic average of absolute errors. MAE measures only the magnitude of the errors and doesn't concern itself with their direction. The lower the MAE, the higher the accuracy of a model.

Mathematically, MAE can be expressed as follows,

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} | y_i - \hat{y}_i |$$

Where y_i = actual value, y_hat_i = predicted value, n = sample size

Pros of the Evaluation Metric:

- It is an easy to calculate evaluation metric.
- All the errors are weighted on the same scale since absolute values are taken.
- It is useful if the training data has outliers as MAE does not penalize high errors caused by outliers.
- It provides an even measure of how well the model is performing.

Cons of the evaluation metric:

- Sometimes the large errors coming from the outliers end up being treated as the same as low errors.
- MAE follows a scale-dependent accuracy measure where it uses the same scale as the data being measured. Hence it cannot be used to compare series' using different measures.
- One of the main disadvantages of MAE is that it is not differentiable at zero. Many optimization algorithms tend to use differentiation to find the optimum value for parameters in the evaluation metric.
- It can be challenging to compute gradients in MAE.

- **Root Mean Squared Error (RMSE)**
  RMSE is computed by taking the square root of MSE. RMSE is also called the Root Mean Square Deviation. It measures the average magnitude of the errors and is concerned with the deviations from the actual value. RMSE value with zero indicates that the model has a perfect fit. The lower the RMSE, the better

the model and its predictions. A higher RMSE indicates that there is a large deviation from the residual to the ground truth. RMSE can be used with different features as it helps in figuring out if the feature is improving the model's prediction or not.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Pros of the Evaluation Metric:

- RMSE is easy to understand.
- It serves as a heuristic for training models.
- It is computationally simple and easily differentiable which many optimization algorithms desire.
- RMSE does not penalize the errors as much as MSE does due to the square root.

Cons of the evaluation metric:

- Like MSE, RMSE is dependent on the scale of the data. It increases in magnitude if the scale of the error increases.
- One major drawback of RMSE is its sensitivity to outliers and the outliers have to be removed for it to function properly.
- RMSE increases with an increase in the size of the test sample. This is an issue when we calculate the results on different test samples.

As the same code we have used to evaluate the models, we have created a function namely "Evaluation_Metrics"

Cross Validated:

Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model.

We have also used K Fold cross validation to measure the model's performance.

And to do so a function is defined namely "cross_validate(model)".

Hyper parameter Tuning:

We have also tuned the hyper parameters of the model to get the optimal result.

A Machine Learning model is defined as a mathematical model with a number of parameters that need to be learned from the data. By training a model with existing data, we are able to fit the model parameters.

However, there is another kind of parameters, known as hyper parameters that cannot be directly learned from the regular training process. They are usually fixed before the actual training process begins. These parameters express important properties of the model such as its complexity or how fast it should learn.

Some examples of model hyper parameters include:

The penalty in Logistic Regression Classifier i.e. L1 or L2 regularization

The learning rate for training a neural network.

The C and sigma hyper parameters for support vector machines.

The k in k-nearest neighbors.

Vanilla linear regression does not have any hyperparameters. Variants of linear regression (ridge and lasso) have regularization as a hyperparameter. The decision tree has max depth and min number of observations in leaf as hyperparameters.

Phase 4: Advance Modelling & Feature Engineering

- At this stage we will try with **Advanced Modeling and Feature Engineering**

Advanced modelling consists ideas like combining various models to obtain a better result while respecting real-world constraints and needs.

There are different ways of combining the models as stated below –
There are two approaches i.e. the **Ensemble** and **Hybrid** techniques that we can use to combine several algorithms together.

By ensemble we use multiple learning algorithms to obtain better predictive results than we could have used one algorithm. Ensemble methods falls under three categories **Bagging** (Bootstrap Aggregation) which combines the outputs of several algorithms in a way that decreases the variance of prediction by generating additional data for training from our original data set. Boosting basically increases the amount of training data.

Another ensemble approach is **Boosting** where we use the subset of our original data to produce several models then we boost their performance by combining them together using techniques such as voting.

The final ensemble method is **Stacking**, this is simply applying several algorithms to our original data in a stacked way. We train the base classifiers also referred to as weak learners then take the algorithm with the highest performance.

The second option of combining several algorithms together is to develop a **Hybrid** model. Unlike ensemble a hybrid model gives we flexibility of going beyond the ensemble techniques to create our innovative model. An example of hybrid model is in tasks of text classification where we can use Convolution Neural Network with Recurrent Neural Network to classify text. Convolution Network is used for feature extraction while Recurrent Network is for classification purpose.

Before combining algorithms we need to know the strengths and weaknesses of each algorithm we are considering to combine together.

Further explanation addition to above…

Multiple-Model Techniques
Ensemble learning is concerned with approaches that combine predictions from two or more models.

We can characterize a model as an ensemble learning technique if it has two properties, such as:

- Comprising two or more models.
- Predictions are combined.

We might also suggest that the goal of an ensemble model is to improve predictions over any contributing member. Although a lesser goal might be to improve the stability of the model, e.g. reduce the variance in the predictions or prediction errors.

Nevertheless, there are models and model architectures that contain elements of ensemble learning methods, but it is not clear as to whether they may be considered ensemble learning or not.

For example, we might define an ensemble learning technique as being composed of two or more models. The problem is that there may be techniques that have more than two models, yet do not combine their predictions. Alternatively, they may combine their predictions in unexpected ways.

For a lack of a better name, we will refer to these as “multiple-model techniques” to help differentiate them from ensemble learning methods. Yet, as we will see, the line that separates these two types of machine learning methods is not that clear.

Multiple-Model Techniques: Machine learning algorithms that are composed of multiple models and combine the techniques but might not be considered ensemble learning.
As such, it is important to review and explore multiple-model techniques that sit on the border of ensemble learning to both better understand ensemble learning and to draw upon related ideas that may improve the ensemble learning models we create.

There are predictive modeling problems where the structure of the problem itself may suggest the use of multiple models.

Typically, these are problems that can be divided naturally into sub-problems. This does not mean that dividing the problems into sub problems is the best solution for a given example; it only means that the problem naturally lends itself to decomposition.

Two examples are multi-class classification and multiple-output regression.

Multiple Models for Multi-Class Classification
Classification problems involve assigning a class label to input examples.

Binary classification tasks are those that have two classes. One decision is made for each example, either assigning it to one class or another. If modeled using probability, a

single probability of the example being to one class is predicted, where the inverse is the probability for the second class, called a binomial probability distribution.
More than two classes can introduce a challenge. The techniques designed for two classes can be extended to multiple classes, and sometimes, this is straightforward.

Multi-Class Classification: Assign one among more than class labels to a given input example.
Alternatively, the problem can be naturally partitioned into multiple binary classification tasks.

There are many ways this can be achieved.

For example, the classes can be grouped into multiple one-vs-rest prediction problems. A model can then be fit for each subproblem and typically the same algorithm type is used for each model. When a prediction is required for a new example, then the model that responds more strongly than the other models can assign a prediction. This is called a one-vs-rest (OvR) or one-vs-all (OvA) approach.

- OvR: A technique that splits a multi-class classification into one binary classification problem per class.

The multi-class classification problem can be divided into multiple pairs of classes, and a model fit on each. Again, a prediction for a new example can be chosen from the model that responds more strongly. This is referred to as one-vs-one (OvO).

- OvR: A technique that splits a multi-class classification into one binary classification problem per each pair of classes.

For more on one-vs-rest and one-vs-one classification, see the tutorial:

This approach of partitioning a multi-class classification problem into multiple binary classification problems can be generalized. Each class can be mapped to a unique binary string for the class with arbitrarily length. One classifier can then be fit to predict each bit in the bit string, allowing an arbitrary number of classifiers to be used.

The bit string can then be mapped to the class label with the closest match. The additional bits act like error-correcting codes, improving the performance of the approach in some cases over simpler OvR and OvO methods. This approach is referred to as Error-Correcting Output Codes, ECOC.

- ECOC: A technique that splits a multi-class classification into an arbitrary number of binary classification problems.

In each of these cases, multiple models are used, just like an ensemble. Predictions are also combined, like an ensemble method, although in a winner-take-all method rather than a vote or weighted sum. Technically, this is a combination method, but unlike most typical ensemble learning methods.

Unlike ensemble learning, these techniques are designed to explore the natural decomposition of the prediction problem and leverage binary classification problems that may not easily scale to multiple classes.

Whereas ensemble learning is not concerned with opening up new capabilities and is typically only focused on improved predictive performance over contributing models. With techniques like OvR, OvR, and ECOC, the contributing models cannot be used to address the prediction problem in isolation, by definition.

Multiple Models for Multi-Output Regression

Regression problems involve predicting a numerical value given an input example.

Typically, a single output value is predicted. Nevertheless, there are regression problems where multiple numeric values must be predicted for each input example. These problems are referred to as multiple-output regression problems.

Multiple-Output Regression: Predict two or more numeric outputs given an input.

Models can be developed to predict all target values at once, although a multi-output regression problem is another example of a problem that can be naturally divided into subproblems.

Like binary classification in the previous section, most techniques for regression predictive modeling were designed to predict a single value. Predicting multiple values can pose a problem and requires the modification of the technique. Some techniques cannot be reasonably modified for multiple values.

One approach is to develop a separate regression model to predict each target value in a multi-output regression problem. Typically, the same algorithm type is used for each model. For example, a multi-output regression with three target values would involve fitting three models, one for each target.

When a prediction is required, the same input pattern is provided to each model and the specific target for each model is predicted and together represent the vector output of the method.

Multi-Output Regression: A technique where one regression model is used for each target in a multi-output regression problem.
Another related approach is to create a sequential chain of regression models. The difference is that the output of the first model predicts the first output target value, but this value is used as part of the input to the second model in the chain in order to predict the second output target value, and so on.

As such, the chain introduces a linear dependence between the regression models, allowing the outputs of models later in the chain to be conditional on the outputs of prior models in the chain.

Regression Chain: A technique where a sequential chain of regression models is used to predict each target in a multi-output regression problem, one model later in the chain uses values predicted by models earlier in the chain.

In each case, multiple regression models are used, just like an ensemble.

A possible difference from ensembles is that the predictions made by each model are not combined directly. We could stretch the definition of “combining predictions” to cover this approach, however. For example, the predictions are concatenated in the case of multi-output regression models and indirectly via the conditional approach in chained regression.

The key difference from ensemble learning methods is that no contributing ensemble member can solve the prediction problem alone. A solution can only be achieved by combining the predictions from all members.

Multiple Expert Models
So far, we have looked at dividing problems into subtasks based on the structure of what is being predicted.

There are also problems that can be naturally divided into sub problems based on the input data. This might be as simple as partitions of the input feature space, or something more elaborate, such as dividing an image into the foreground and background and developing a model for each.

A more general approach for this from the field of neural networks is referred to as a mixture of experts (MoE).

The approach involves first dividing the learning task into subtasks, developing an expert model for each subtask, using a gating model to decide or teach which expert to use for each example and the pool the outputs of the experts, and gating model together to make a final prediction.

MoE: A technique that develops an expert model for each subtask and learns how much to trust each expert when making a prediction for specific examples.

wo aspects of MoE make the method unique. The first is the explicit partitioning of the input feature space, and the second is the use of a gating network or gating model that learns which expert to trust in each situation, e.g., each input case.

Unlike the previous examples for multi-class classification and multi-output regression that divided the target into sub problems, the contributing members in a mixture of experts model can address the whole problem, at least partially or to some degree. Although an expert may not be tailored to a specific input, it can still be used to make a prediction on something outside of its area of expertise.

Also, unlike those previously reviewed methods, a mixture of experts also combines the predictions of all contributing members using a weighted sum, albeit measured by the gating network.

As such, it more closely resembles more familiar ensemble learning techniques, such as stacked generalization, known as stacking.

Hybrids Constructed From Multiple Models

Another type of machine learning that involves the use of multiple models and is loosely related to ensemble learning is hybrid models.

Hybrid models are those models that combine two or more models explicitly. As such, the definition of what does and does not constitute a hybrid model can be vague.

Hybrid Model: A technique that combines two or more different machine learning models in some way.

For example, an autoencoder neural network that learns how to compress input patterns to a bottleneck layer, the output of which is then fed to another model, such as a support vector machine, would be considered a hybrid machine learning model.

This example has two machine learning models, a neural network and a support vector machine. It just so happens that the models are linearly stacked one on top of another into a pipeline and the last model in the pipeline makes a prediction.

Consider an ensemble learning method that has multiple contributing ensemble members of different types (e.g. a logistic regression and a support vector machine) and uses voting to average their predictions. This ensemble too might be considered a hybrid machine learning model, under the broader definition.

Perhaps the key difference between ensemble learning and hybrid machine learning is the need in hybrid models to use models of differing types. Whereas in ensemble learning, contributing members to the ensemble may be of any type.

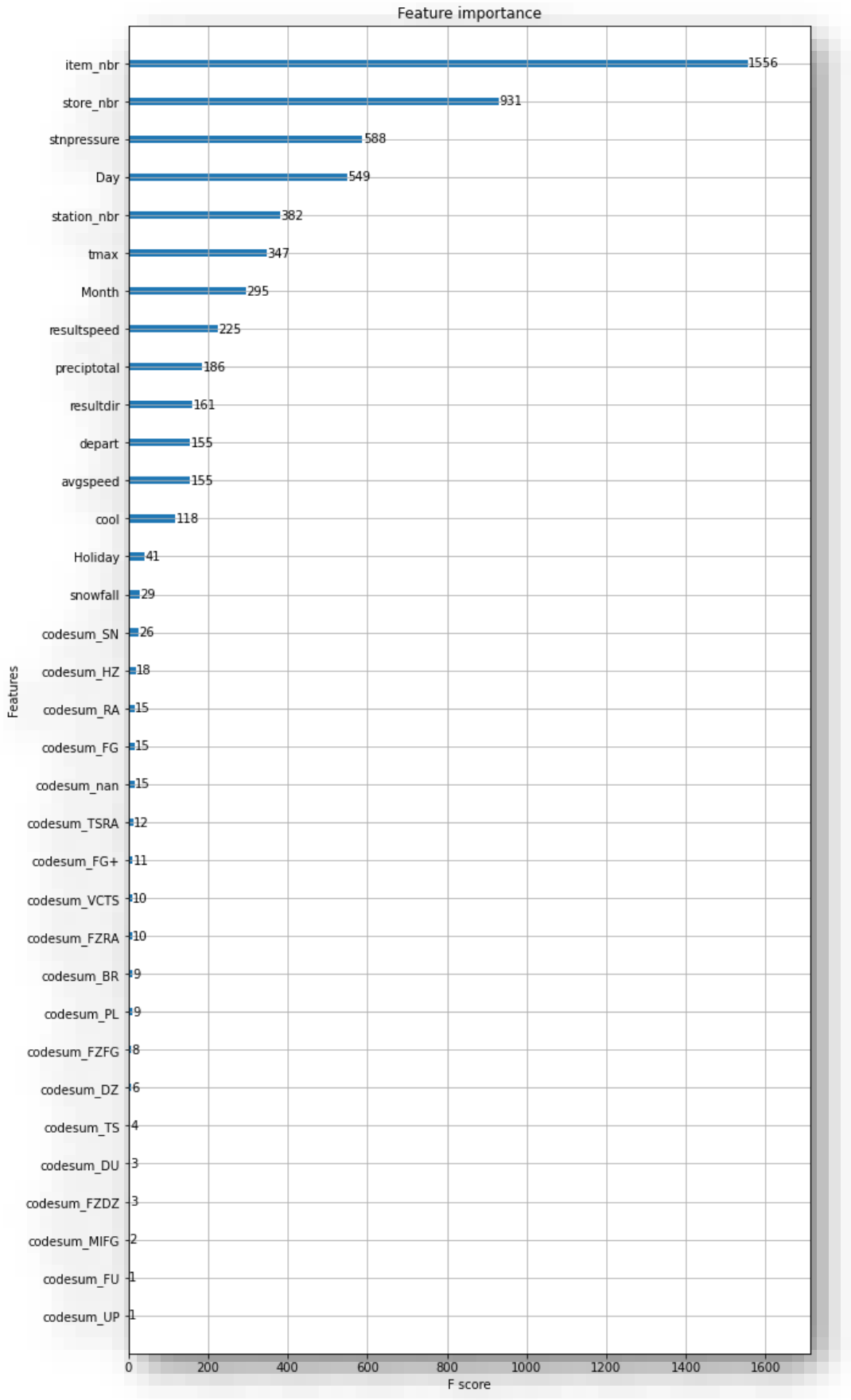
Further, it is more likely that a hybrid machine learning will graft one or more models onto another base model, which is quite different from fitting separate models and combining their predictions as we do in ensemble learning.

- Given the above detailed explanation, examples, pros, cons of combining different models through the above explained techniques, we found that the issue we are dealing with does not meet any requirement to use the combination of models.

Whereas we have used two different types of model to deal with this, one the classification problem to understand whether there is any sell or not and based on the prediction we have sent the query point to a regression model to predict the future sells.

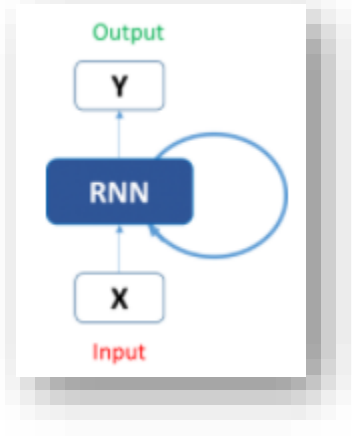
- We have also implemented the **advanced feature encoding and engineering methods**

Like we have encoded "**codesum**" variable which is categorical in nature to 29 features. Created new features like "Day" "Month" "Holiday". And below is the measure of impact for the newly engineered features and the existing one.
Below feature importance shows the newly engineered features like "Day" "Month" "Holiday" has great values whereas 29 codesum features created through one hot encoding do not have much importance.
So we removed those newly added codesum features while implementing advance modelling.

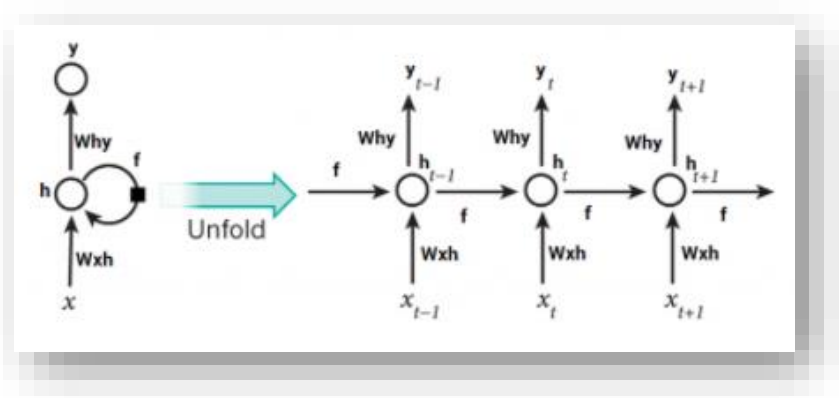

Feature importance
item_nbr
store_nbr
stnpressure
Day
station_nbr
tmax
Month
resultspeed
preciptotal
resultdir
depart
avgspeed
cool
Holiday
snowfall
codesum_SN
codesum_HZ
codesum_RA
codesum_FG
codesum_nan
codesum_TSRA
codesum_FG+
codesum_VCTS
codesum_FZRA
codesum_BR
codesum_PL
codesum_FZFG
codesum_DZ
codesum_TS
codesum_DU
codesum_FZDZ
codesum_MIFG
codesum_FU
codesum_UP
1556
931
588
549
382
347
295
225
186
161
155
155
118
41
29
26
18
15
15
15
12
11
10
10
9
9
8
6
4
3
3
2
1
1
Features
0
200
400
600
800
1000
1200
1400
1600
F score

- **Next we have tried with Advance modelling technique like neural networks – LSTM**

A brief details on Long Short-Term Memory Units (LSTMs) which we used for multivariate single step time series forecasting. We used LSTMs as it has an edge over conventional feed-forward neural networks and RNN in many ways. This is because of their property of selectively remembering patterns for long durations of time. The problem what we are solving is basically predicting future stocks based on everyday sales of past few years.
As for a specific store location the weather features, demographics are kind of same throughout the years. So we can use the past data to predict future **using Recurrent Neural Networks (RNN).**



- **Typical RNN looks like**



- **Expanded**

Recurrent Neural Networks work just fine when we are dealing with short-term dependencies. However, vanilla RNNs fail to understand the context behind an input for something that happened long before, cannot be recalled when making predictions in the present. The reason behind this is the problem of **Vanishing Gradient**.
As we know that for a conventional feed-forward neural network, the weight updating that is applied on a particular layer is a multiple of the learning rate, the error term from the previous layer and the input to that layer. Thus, the error term for a particular layer is somewhere a product of all previous layers' errors. When dealing with activation functions like the sigmoid function, the small values of its derivatives (occurring in the error function) gets multiplied multiple times as we move towards the starting layers. As a result of this, the gradient almost vanishes as we move towards the starting layers, and it becomes difficult to train these layers.

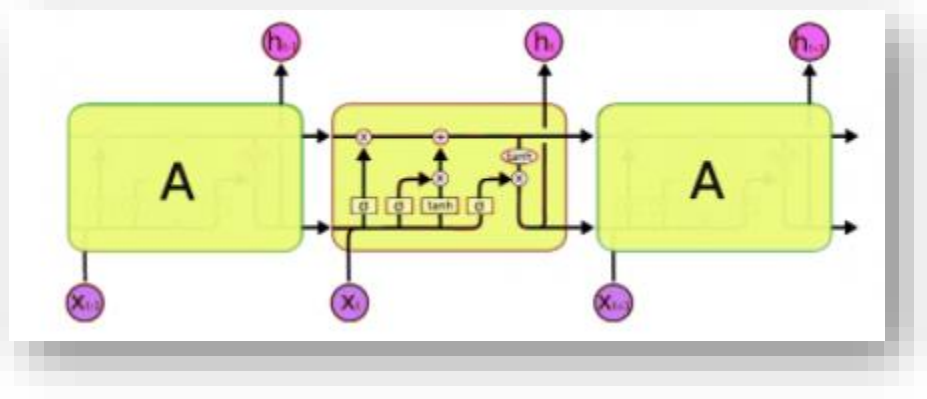
LSTMs on the other hand, make small modifications to the information by multiplications and additions. With LSTMs, the information flows through a mechanism known as cell states. This way, LSTMs can selectively remember or forget things. The information at a particular cell state has three different dependencies.

We'll visualize this with an example. Let's take the example of predicting stock prices for a particular stock. The stock price of today will depend upon:

The trend that the stock has been following in the previous days, maybe a downtrend or an uptrend.
The price of the stock on the previous day, because many traders compare the stock's previous day price before buying it.
The factors that can affect the price of the stock for today. This can be a new company policy that is being criticized widely, or a drop in the company's profit, or maybe an unexpected change in the senior leadership of the company.
These dependencies can be generalized to any problem as:

1. The previous cell state (i.e. the information that was present in the memory after the previous time step)
2. The previous hidden state (i.e. this is the same as the output of the previous cell)
3. The input at the current time step (i.e. the new information that is being fed in at that moment)

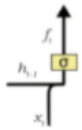
Architecture of LSTMs



A typical LSTM network is comprised of different memory blocks called cells
(The rectangles that we see in the image). There are two states that are being transferred to the next cell; the cell state and the hidden state. The memory blocks are responsible for remembering things and manipulations to this memory is done through three major mechanisms, called gates.

Forget Gate

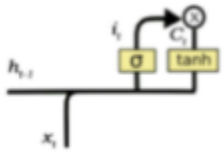
A forget gate is responsible for removing information from the cell state. The information that is no longer required for the LSTM to understand things or the information that is of less importance is removed via multiplication of a filter. This is required for optimizing the performance of the LSTM network.



This gate takes in two inputs; h_t-1 and x_t.

h_t-1 is the hidden state from the previous cell or the output of the previous cell and x_t is the input at that particular time step. The given inputs are multiplied by the weight matrices and a bias is added. Following this, the sigmoid function is applied to this value. The sigmoid function outputs a vector, with values ranging from 0 to 1, corresponding to each number in the cell state. Basically, the sigmoid function is responsible for deciding which values to keep and which to discard. If a ‘0’ is output for a particular value in the cell state, it means that the forget gate wants the cell state to forget that piece of information completely. Similarly, a ‘1’ means that the forget gate wants to remember that entire piece of information. This vector output from the sigmoid function is multiplied to the cell state.

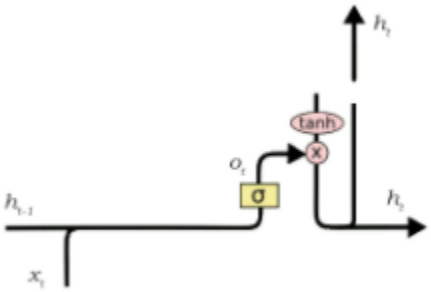
Input Gate



The input gate is responsible for the addition of information to the cell state. This addition of information is basically three-step process as seen from the diagram above.

1. Regulating what values need to be added to the cell state by involving a sigmoid function. This is basically very similar to the forget gate and acts as a filter for all the information from h_t-1 and x_t.
2. Creating a vector containing all possible values that can be added (as perceived from h_t-1 and x_t) to the cell state. This is done using the tanh function, which outputs values from -1 to +1.
3. Multiplying the value of the regulatory filter (the sigmoid gate) to the created vector (the tanh function) and then adding this useful information to the cell state via addition operation.

Output Gate



The functioning of an output gate can again be broken down to three steps:

1. Creating a vector after applying tanh function to the cell state, thereby scaling the values to the range -1 to +1.
2. Making a filter using the values of h_t-1 and x_t, such that it can regulate the values that need to be output from the vector created above. This filter again employs a sigmoid function.
3. Multiplying the value of this regulatory filter to the vector created in step 1, and sending it out as an output and also to the hidden state of the next cell.

We have tried using LSTMS with 3 different models –
a)

```
model = Sequential()
model.add(InputLayer((30, 14)))
model.add(LSTM(64))
model.add(Dense(8, 'relu'))
model.add(Dense(1, 'linear'))
```

Below is what it interpret:

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_1 (LSTM)               (None, 64)                20224

 dense_2 (Dense)             (None, 8)                 520

 dense_3 (Dense)             (None, 1)                 9

=================================================================
Total params: 20,753
Trainable params: 20,753
Non-trainable params: 0
_________________________________________________________________
```

Below is the Error analysis of Model 1:

Epoch 30/30
2666/2666 [==============================] - 11s 4ms/step - loss: 867.7957 - **root_mean_squared_error: 29.4584** - val_loss: 717.9977 - **val_root_mean_squared_error: 26.7955**

2. Model 2:

```
model2 = tf.keras.Sequential()
model2.add(tf.keras.layers.LSTM(128,input_shape=(30,14),return_sequences=True))
model2.add(tf.keras.layers.LeakyReLU(alpha=0.5))
model2.add(tf.keras.layers.LSTM(128,return_sequences=True))
model2.add(tf.keras.layers.LeakyReLU(alpha=0.5))
model2.add(tf.keras.layers.Dropout(0.3))
model2.add(tf.keras.layers.LSTM(64,return_sequences=True))
model2.add(tf.keras.layers.Dropout(0.3))
model2.add(tf.keras.layers.Dense(1))
```

Below is what it interpret:

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_5 (LSTM)               (None, 30, 128)           73216

 leaky_re_lu_2 (LeakyReLU)   (None, 30, 128)           0

 lstm_6 (LSTM)               (None, 30, 128)           131584

 leaky_re_lu_3 (LeakyReLU)   (None, 30, 128)           0

 dropout_2 (Dropout)         (None, 30, 128)           0

 lstm_7 (LSTM)               (None, 30, 64)            49408

 dropout_3 (Dropout)         (None, 30, 64)            0

 dense_5 (Dense)             (None, 30, 1)             65

=================================================================
Total params: 254,273
Trainable params: 254,273
Non-trainable params: 0
_________________________________________________________________
```

Below is the Error analysis of Model 2:

Epoch 30/30
2666/2666 [==============================] - 23s 9ms/step - loss: 1083.9133 - **root_mean_squared_error: 32.9230** - val_loss: 803.0638 - **val_root_mean_squared_error: 28.3410**

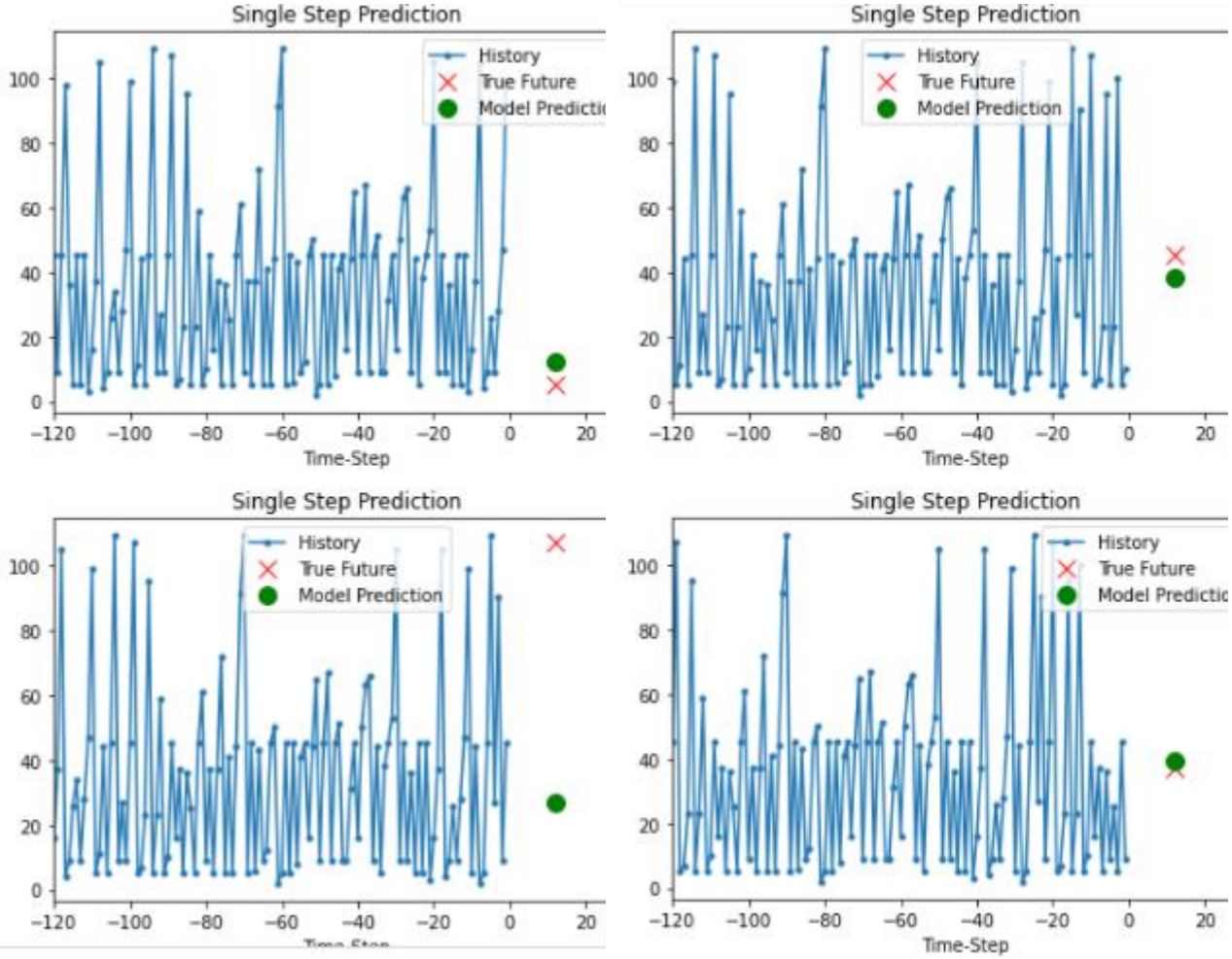
4. Model 3 (Best predicting)

```
# Model
single_step_model = tf.keras.models.Sequential()
single_step_model.add(tf.keras.layers.LSTM(32,
                                           input_shape=x_train_single.shape[-2:]))
single_step_model.add(tf.keras.layers.Dense(1))

single_step_model.compile(optimizer=tf.keras.optimizers.RMSprop(), loss='mae')

single_step_history = single_step_model.fit(train_data_single, epochs=EPOCHS,
                                            steps_per_epoch=STEPS_PER_EPOCH,
                                            validation_data=val_data_single,
                                            validation_steps=50)
```

Below is the Error analysis of Model 2:

```
Epoch 30/30
200/200 [==============================] - 1s 6ms/step - loss: 20.3042 -
val_loss: 21.2426
```

Visualizing the prediction:



By this we are done with Phase 4.

Phase 5: Deployment and Productionization

- At this stage we will work on **Deployment and Productionization of the best model**

There are different approaches of deploying a model – these are: Locally using scikit learn/ AWS/Heroku/GCP/Azure etc.

For this particular project we tried

1. **Deploying locally**: This is the easiest way of testing like how an end user will see the model at his/her end.
   This way is good to test the model from end user prospective, but in live scenario we will need to send a link to platform which is always available and user can check the query at any time. In this case there is no such option available

2. **Deploying at AWS:** This approach is something actually being used in live environment. This is one of the ways used in live environment other ways are Heroku, GCP, Azure.
   As we have used the free tier resources available, so the resources available is very limited. And for a model which is heavy in nature might get timeout

   Deploying model in AWS is a bit easy compared to others (though this statement varies based on users)

To deploy the model we have used –

Flask: To create the user interface & run the model
Joblib: Used this library to zip the trained model so that we can use the same in app

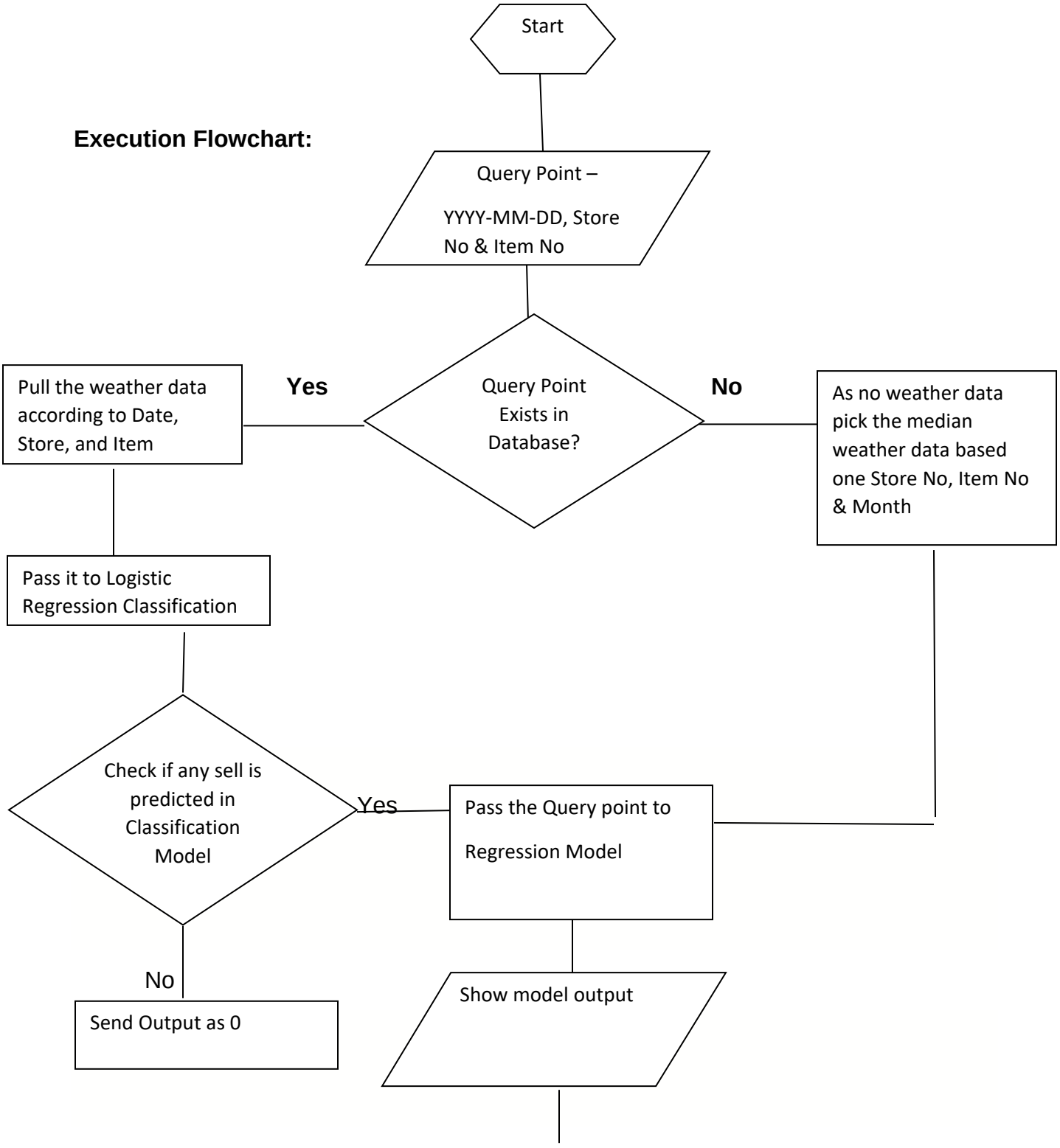
Saving the Logistic Regression model as Pickle file for Production purpose

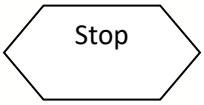
```
[ ]  from joblib import dump, load
     dump(model_cls, '/content/drive/MyDrive/Project_TS_Walmart/Phase 2/Data/cls_LogisticRegrsn.pkl')

     ['/content/drive/MyDrive/Project_TS_Walmart/Phase 2/Data/cls_LogisticRegrsn.pkl']
```

Loading the .pkl file

```
# Our best model is XGBRegressor, then Random Forest.
#But for both these Model- while loading pkl file we faced error in terms of Parameter so used Linear Regression
reg_RF = joblib.load('LinearReg.pkl')
predictionReg = reg_RF.predict(QueryPoint)
FinalPrediction = predictionReg
```

Execution Flowchart:

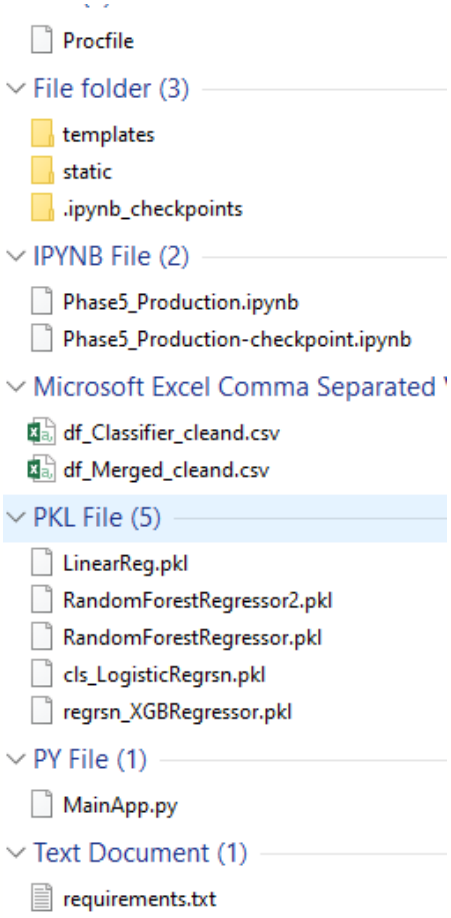




Below is the folder structure, which contains -

1. “**MainApp.py**” : has all the codes to read through the query point from user interface (created by Flask API)
2. “**df_Classifier_cleand.csv**”, “**df_Merged_cleand.csv**”: Cleaned merged data created after EDA
3. “**.pkl**” files: For all the models we created at Phase 4 modelling stage. Linear Regression classification model (LinearReg.pkl) will be used for sure, and any one of the regression models will be used.
4. “**static**” **folder**: Holds the images, static files if any
5. “**templates**” **folder**: Holds the .html files which are used for user interfaces

    a) “**querypage.html**”: Is the landing page where user will input the Date, Store No, Item no as Query point
    b) “**outputpage.html**”: Once the above is submitted “**outputpage.html**” will pop up with the output
    c) “**close.html**”: And lastly “close.html” will be called at the end



Deploying and checking the Model Locally:

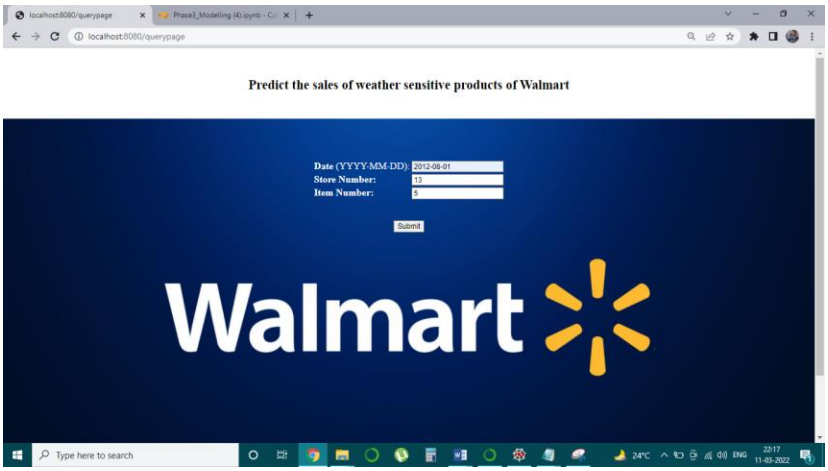
1. First we will need to run the “app.py” (a code is there to load the corresponding Model’s .pkl file)
2. Then our system will act as server, based on the port no given a link will be generated. Something like below –

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8080)
```

```
 * Running on http://0.0.0.0:8080/ (Press CTRL+C
to quit)
```

3. Use the link: http://localhost:8080/querypage (change the yellow part as per the port no given)

4. On first screen we will see below –



3. Need to provide the query point details date, store no, item no
4. Once page is submitted we will see below page –

Deploying the Model at AWS:

Steps:

1. Build a model on our local system and store the model and other key model related variables in .pkl files
2. Launch a micro instance on AWS.
3. Connect to the AWS box [ssh]
4. Move the files to an AWS EC2 instance/box [scp]
5. Install all packages needed on the AWS box.
6. Run app.py on the AWS box.
7. Check the output in the browser.

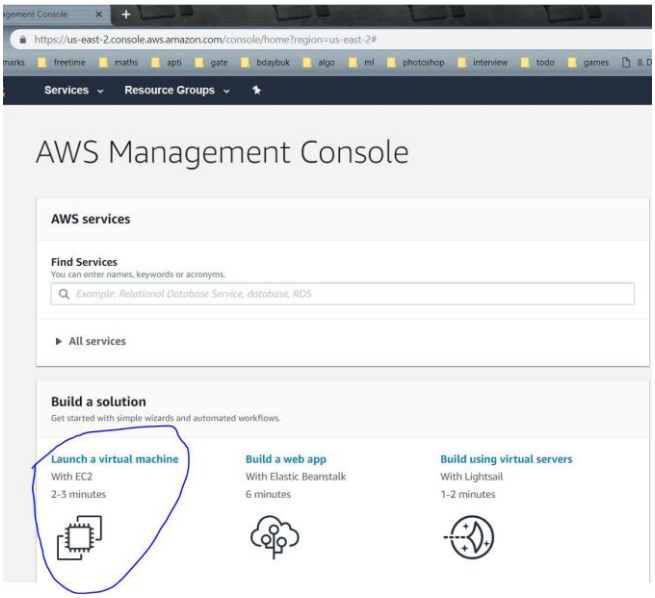
Packages needed:

1. pip3
2. pandas
3. numpy
4. sklearn
5. beautifulsoup4
6. lxml
7. flask
8. re

Launch a micro instance on AWS.

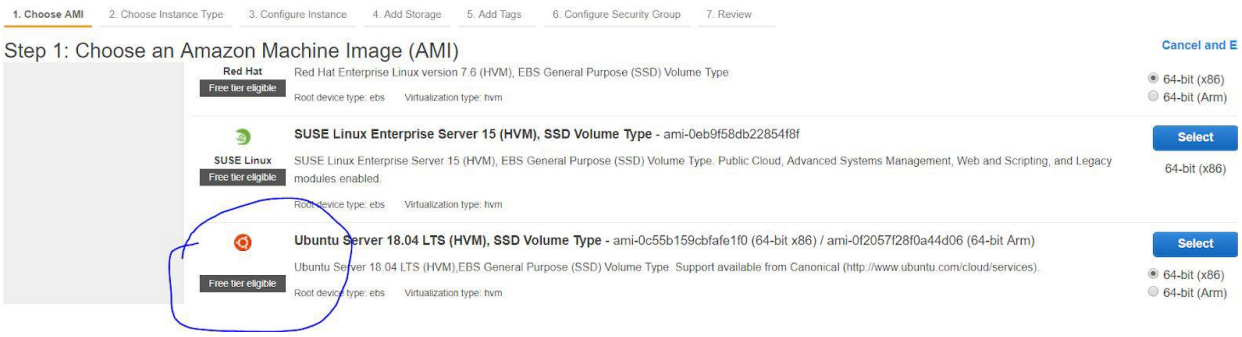
Creating an instance:

1. Create an AWS account https://aws.amazon.com , https://portal.aws.amazon.com/billing/signup#/start
2. Login: https://console.aws.amazon.com
   After login:

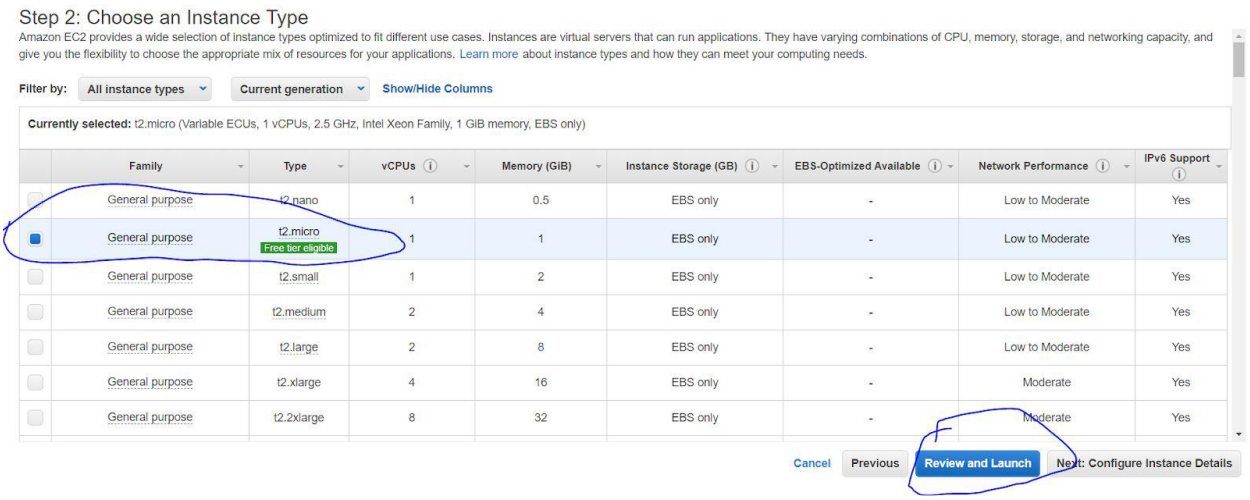


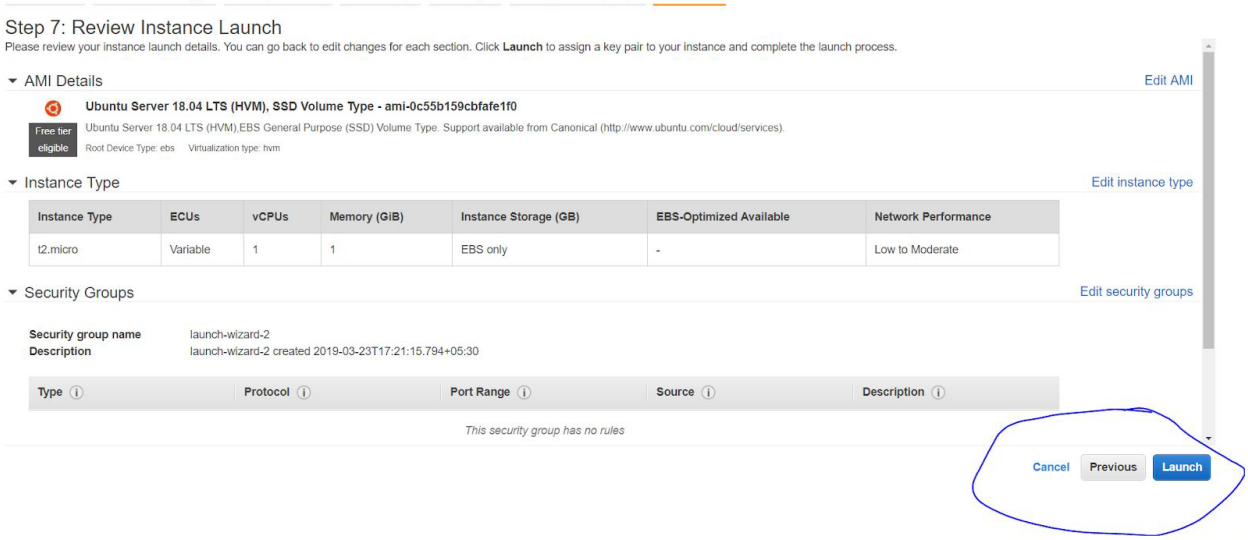
Launch the EC2 instance

3. Choose the ubuntu free tire

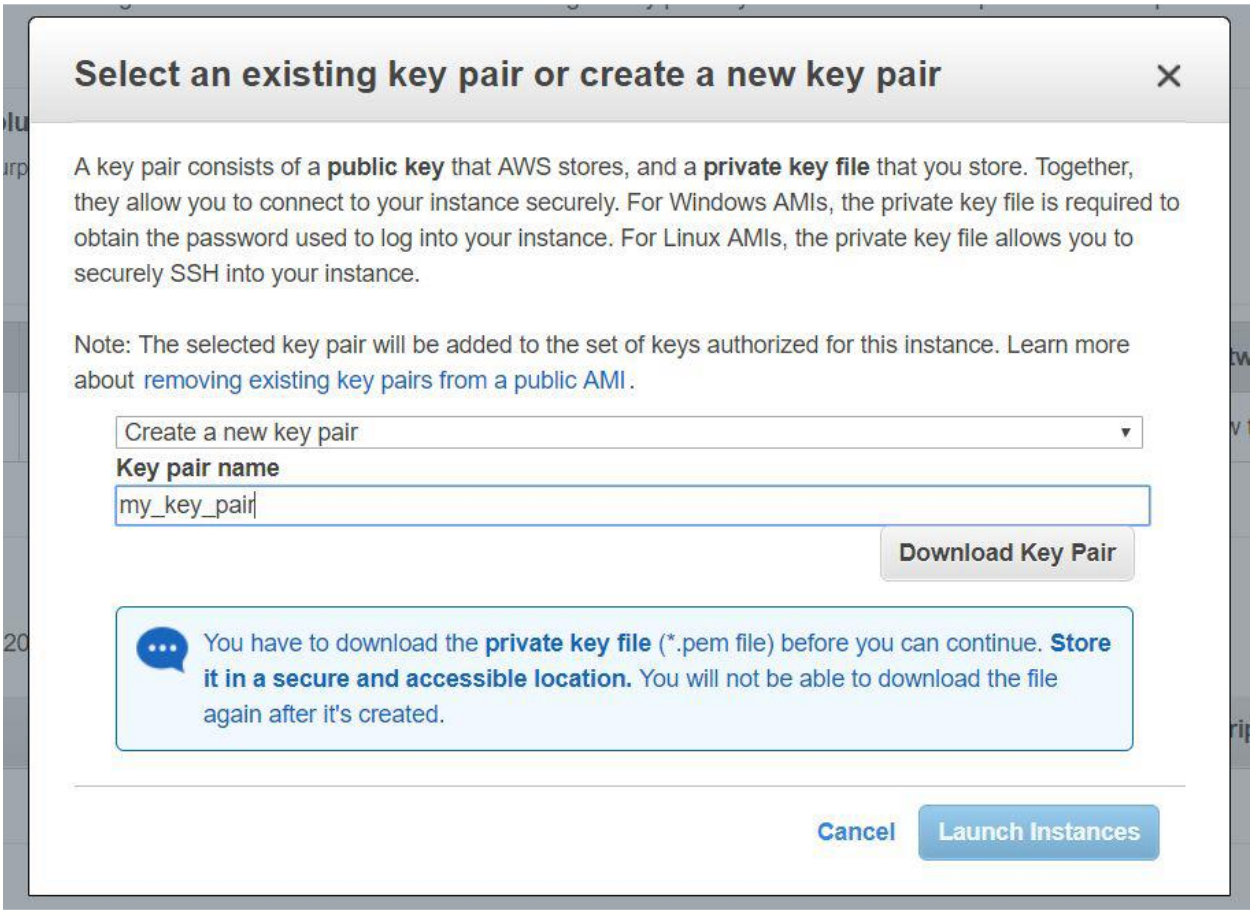


4. Choose t2.micro free tier eligible & then Click on review and launch

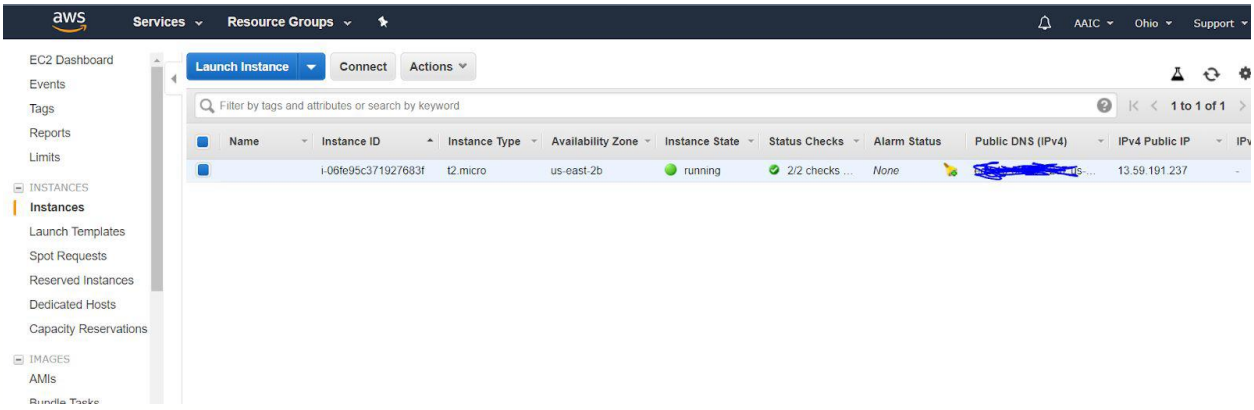




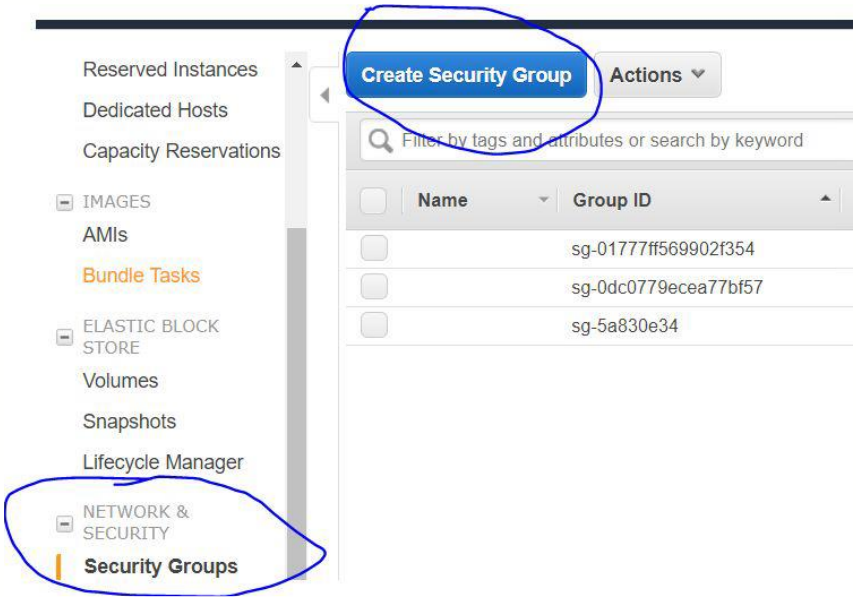
5. Click on “Download Key Pair” and save the .pem file then click on “Launch Instance”

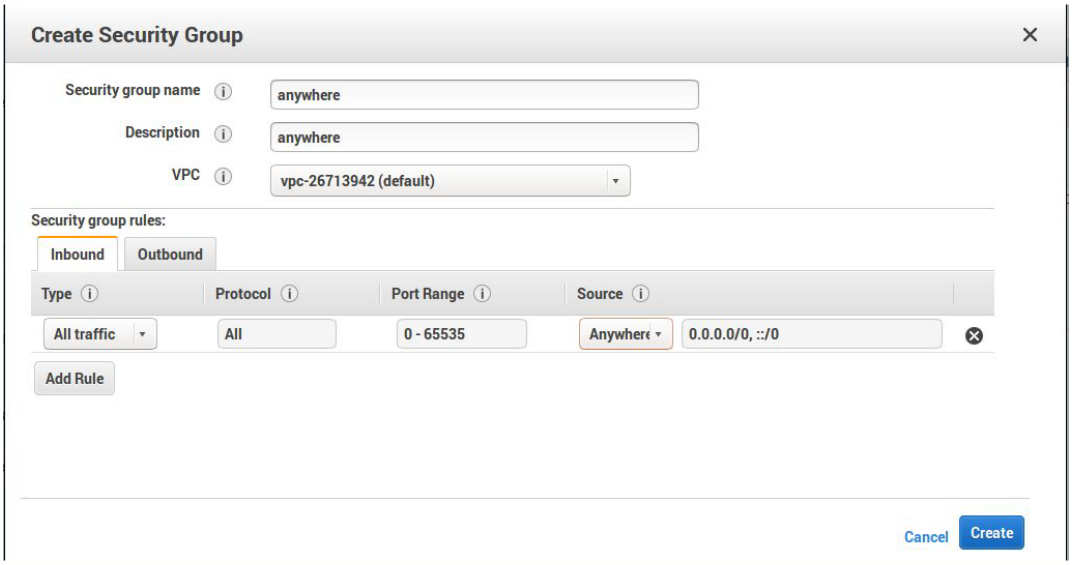


6. You will see this screen, you have successfully launched the an EC2 instance, now we need to
Launch an flask api in it



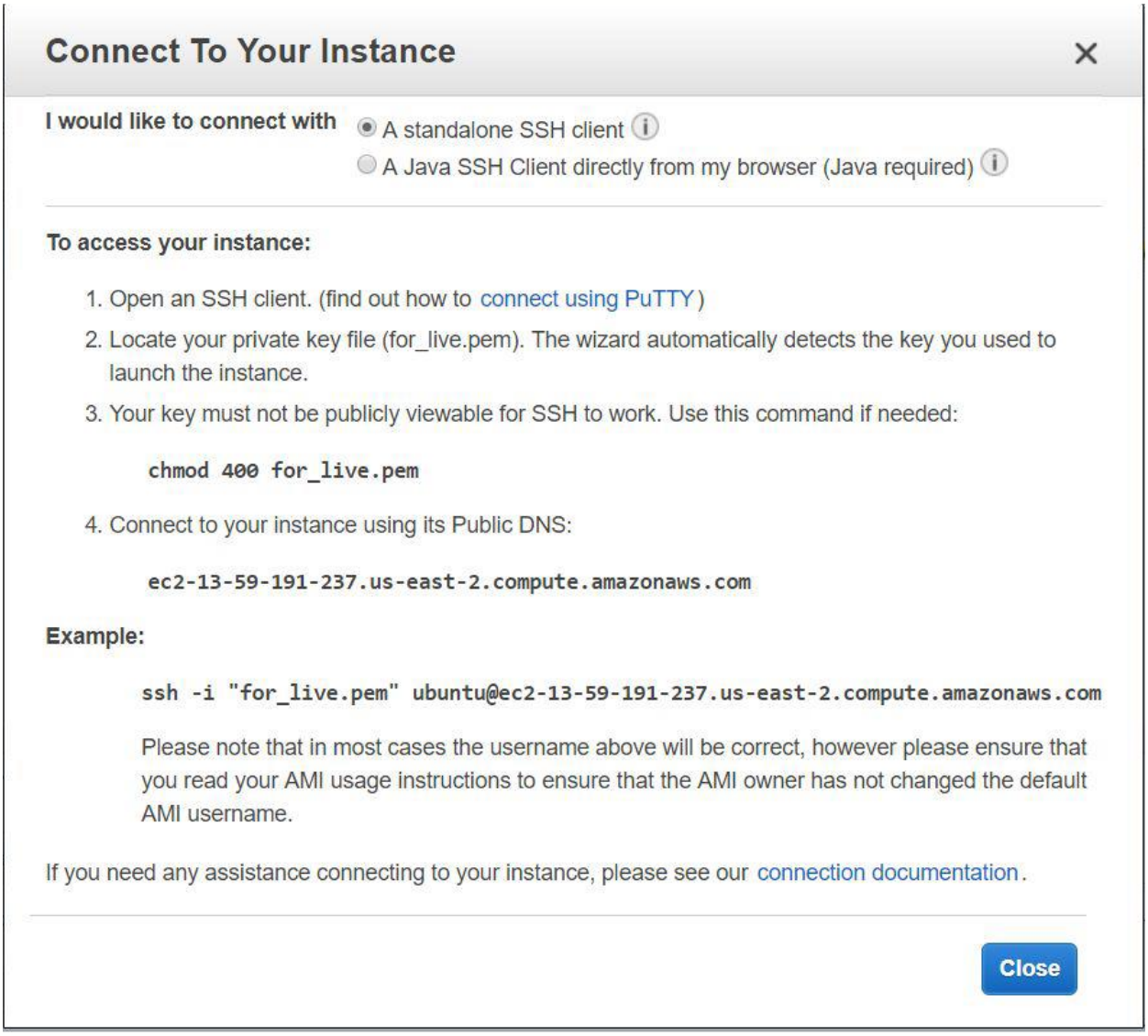
7. Select the “Network & security” -> Security groups and then click “Create Security Group”





Then add the specific security group to **network interface**

Connect to the AWS box



Move the files to an AWS EC2 instance/box

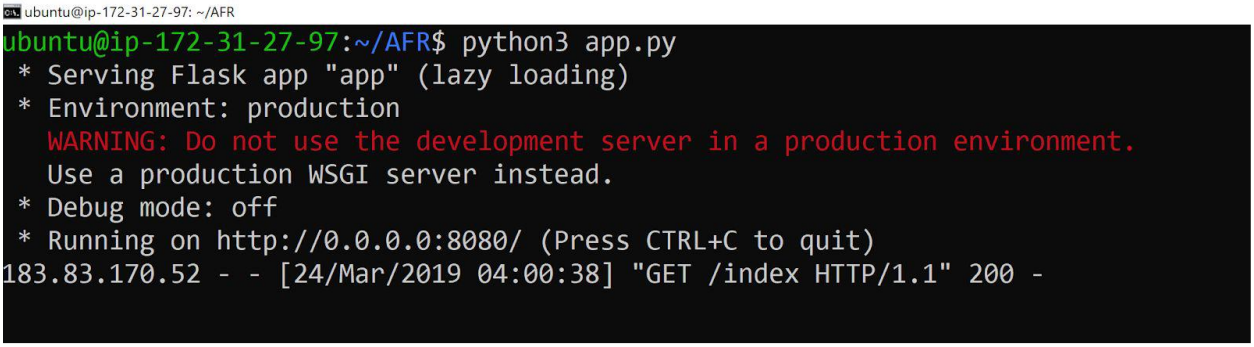
Command line to copy files

C:\Users\Asus\OneDrive\Desktop> **scp -r -i "for_live.pem" ./AFR**
ubuntu@ec2-13-59-191-237.us-east-2.compute.amazonaws.com :~/

Install all packages needed on the AWS box

sudo apt-get install python3-pip
pip3 install <each of the following packages>
Packages needed:
pip3
pandas
numpy
sklearn
beautifulsoup4
lxml
flask
re

Run app.py on the AWS box.



And then check in browser

For our case test link will be like –

ec2-18-224-96-3.us-east-2.compute.amazonaws.com:8080/querypage

- **Scalability, limitations, latency of our system**

In terms of scalability the model could have been designed better, at this stage predicting sales of future dates does not provide much accurate results, but in case of regression it is always an issue.

Whereas we have used free tier to deploy the project, so the latency is not good. But deploying at proper infrastructure it would have been results better.

Though we have not measure the latency.

- **Given more time**
  LSTM model could have been reengineered and used in production. Also standard Time series forecasting models like AR, MA, ARMA, ARIMA, SARIMA, SARIMAX, VAR, VARMA, VARMAX etc.

References

1. https://www.appliedroots.com/
2. https://towardsdatascience.com/metrics-and-python-850b60710e0c
3. https://www.quora.com/What-is-the-difference-between-an-RMSE-and-RMSLE-logarithmic-error-and-does-a-high-RMSE-imply-low-RMSLE
4. https://onstrategyhq.com/resources/kpis-vs-metrics-tips-tricks-to-performance-measures/
5. https://datascience.stackexchange.com/questions/63514/what-is-the-difference-between-an-rmse-and-rmsle-logarithmic-error
6. https://towardsdatascience.com/the-complete-guide-to-time-series-analysis-and-forecasting-70d476bfe775
7. https://towardsdatascience.com/sales-forecasting-from-time-series-to-deep-learning-5d115514bfac
8. https://www.tableau.com/learn/articles/time-series-forecasting
9. https://machinelearningmastery.com/time-series-forecasting-methods-in-python-cheat-sheet/
10. https://towardsdatascience.com/time-series-forecasting-with-deep-learning-and-attention-mechanism-2d001fc871fc
11. https://www.analyticsvidhya.com/blog/2020/07/univariate-analysis-visualization-with-illustrations-in-python/
12. https://www.mygreatlearning.com/blog/introduction-to-multivariate-analysis/
13. https://scikit-learn.org/stable/auto_examples/feature_selection/plot_feature_selection.html
14. https://blog.datadive.net/selecting-good-features-part-i-univariate-selection/
15. https://topepo.github.io/caret/feature-selection-using-univariate-filters.html
16. https://www.statisticshowto.com/univariate/
17. http://www.kmrom.com/Site-En/Articles/ViewArticle.aspx?ArticleID=416
18. https://data-flair.training/blogs/qlik-sense-distribution-plot/
19. https://online.stat.psu.edu/stat555/node/15/
20. https://datatron.com/multivariate-analysis-techniques-for-exploring-data/
21. https://www.analyticsvidhya.com/blog/2020/08/types-of-categorical-data-encoding/
22. https://datascience.stackexchange.com/questions/9443/when-to-use-one-hot-encoding-vs-labelencoder-vs-dictvectorizor
23. https://www.kaggle.com/prashanththangavel/advanced-feature-engineering-feature-encoding
24. https://towardsdatascience.com/the-art-of-effective-visualization-of-multi-dimensional-data-6c7202990c57
25. https://ieeexplore.ieee.org/abstract/document/9512057
26. https://blog.clairvoyantsoft.com/mlmuse-visualisation-of-high-dimensional-data-using-t-sne-ac6264316d7f
27. https://distill.pub/2016/misread-tsne/

28. https://towardsdatascience.com/strategies-and-tactics-for-regression-on-imbalanced-data-61eeb0921fca
29. https://github.com/nickkunz/smogn
30. https://analyticsindiamag.com/7-types-classification-algorithms/
31. https://towardsdatascience.com/building-a-logistic-regression-in-python-step-by-step-becd4d56c9c8
32. https://stats.stackexchange.com/questions/48360/is-standardization-needed-before-fitting-logistic-regression
33. https://www.listendata.com/2017/04/how-to-standardize-variable-in-regression.html
34. https://realpython.com/logistic-regression-python/
35. https://www.codegrepper.com/code-examples/python/standardization+classification+sklearn
36. https://www.aionlinecourse.com/tutorial/machine-learning/support-vector-regression
37. https://www.geeksforgeeks.org/linear-regression-python-implementation/
38. https://towardsdatascience.com/5-reasons-why-you-should-use-cross-validation-in-your-data-science-project-8163311a1e79
39. https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter
40. https://scikit-learn.org/stable/modules/cross_validation.html
41. https://www.analyticsvidhya.com/blog/2021/05/know-the-best-evaluation-metrics-for-your-regression-model/
42. https://machinelearningmastery.com/regression-metrics-for-machine-learning/
43. https://machinelearningmastery.com/hyperparameter-optimization-with-random-search-and-grid-search/
44. https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/
45. https://www.analyticsvidhya.com/blog/2021/06/5-techniques-to-handle-imbalanced-data-for-a-classification-problem/
46. https://towardsdatascience.com/choosing-the-best-classification-algorithm-f254f68cca39
47. https://analyticsindiamag.com/7-types-classification-algorithms/
48. https://www.geeksforgeeks.org/advantages-and-disadvantages-of-different-regression-models/
49. https://www.analyticsvidhya.com/blog/2021/10/evaluation-metric-for-regression-models/
50. https://machinelearningmastery.com/multiple-model-machine-learning/#:~:text=Hybrid%20Model%3A%20A%20technique%20that,learning%20models%20in%20some%20way.
51. https://www.quora.com/How-do-I-combine-machine-learning-models
52. https://analyticsindiamag.com/a-complete-guide-to-lstm-architecture-and-its-use-in-text-classification/

53. https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/
54. https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/