I. INTRODUCTION-

Overview of Demand Forecasting

Demand Forecasting is the method in which historical sales data are used to build an estimation of expected consumer demand forecast. For companies, Demand Forecasting provides an estimate of the amount of goods and services that will be purchased by its customers in the foreseeable future. Complex business assertions such as turnover, profit margins, cash flow, capital expenditure, risk assessment and mitigation plans, capacity planning, etc. rely on Demand Forecasts. Passive Demand Forecasting is carried out with very conservative growth plans for stable companies. Simple extrapolations of historical data are made with minimal hypotheses. This is a rare form of forecast limited to both local and small businesses.

For the scaling and diversification of companies with ambitious growth strategies in terms of marketing initiatives, product portfolio expansion and recognition of competitor initiatives and the global economic climate, Active Demand forecasting is carried out.

Some real-world realistic examples of Market Forecasting are – A leading car manufacturer, referring to the last 12 months of existing car sales at model, engine size, and color level; and based on projected production, predicts short-term demand for buying, processing, and inventory planning purposes over the next 12 months.

A leading food producer refers to the last 24 months of its extremely seasonal items, such as soups and mashed potatoes, being actually sold. An analysis is performed at the level of flavor and packaging size. Demand Forecasting of the food supply chain aims to address the issues faced by any food delivery company or service. The companies can forecast the daily or weekly demand of various raw materials with a short-lived shelf life that comprise their inventory in their warehouse.

Importance of demand forecasting

- Market Forecasting is the central market cycle in which a company's strategic and operating plans are conceived.
- A business' strategic and long-range plans such as budgeting, financial planning, sales and marketing plans, capacity planning, risk assessment, and mitigation plans are formulated on the basis of the Demand forecast.
- Execution is based on short to medium term tactical plans such as pre-building, make-to - stock, make-to-order, contract manufacturing, supply planning, network balancing, etc.
- Objectives of Demand Forecasting include Financial planning, Pricing policy, Manufacturing policy, Sales, and Marketing planning, Capacity planning and expansion, Manpower planning and Capital expenditure.
- Demand forecasts form the basis of all managerial decisions in logistics and supply chain management. Regardless of a push or pull type of a supply chain system, demand forecasting is the starting point for all planning activities and execution processes.
- Consider push processes that are performed in anticipation of customer needs - sourcing, production, transportation, operating activities and actions – all require demand forecasts as data input; the same is true for the pull processes – to plan necessary levels of activity and inventory the customer demand data should be a starting point.

Purpose of Demand Forecasting

- Excess supply or stocking can prove to be counterproductive in case of demand fluctuations in the market as the perishable inventory needs to be disposed in case of an unanticipated drop in demand.
- On the contrary, when the supply is scanty, an upsurge in demand can quickly overwhelm the inventory, resulting in a stock out. Both scenarios have detrimental financial repercussions.
- A plausible solution would be to devise an algorithm that can strike a balance between under stocking and over stocking by converging at a point of optimality.

- The demand can be represented as a probability distribution with a temporal component and the nature of this probability distribution can be estimated by a time series forecasting algorithm.

Role of Demand Forecasting in Inventory Management

- As businesses scale, they need to investigate the accuracy of their inventory management and forecasting processes. Demand forecasting goes beyond simple estimates of product demand, looking into complex patterns over time to produce more accurate and timely predictions. Through better demand, an organization will be able to better manage inventory, increase revenue, and improve customer support.
- Inventory management is extraordinarily complex. Every day, product demand can change. Trends, seasons, and customer demographics may all increase or reduce demand for specific items. These changes can lead to under buying, over buying, or buying the wrong items—all causing avoidable challenges for the company.
- Poor inventory management can cost an organization both money and its reputation. When items can't be sold at full price, the company loses revenue. When customers aren't able to get what they want, the company's reputation may suffer. Over time, poor inventory management cuts significantly into a company's profits and makes it less likely that customers will come to the company when they have specific needs.
- At its core, the success of most companies relies on inventory control. Retailers *need* to be able to identify and stock the products that customers need and increase their revenue per square footage. It isn't just retail companies: automotive service shops need to be able to house the right parts, HVAC repair technicians need to have the right tools, and all companies need to have the supplies to ensure all operations can run comfortably. Inventory management can be a downfall or a tool to increase profitability.

Advantages of Demand Forecasting in a Supply Chain

- "Smart" forecasting technology uses artificial intelligence and machine learning to help companies plan. Rather than having to

manually adjust your inventory based on customer needs, you can use past samples of inventory data to determine patterns that indicate product demand. Even patterns such as seasonal purchasing can be accounted for, helping modify your projected demand based on past years and current market trends.

- It can be difficult to perform such forecasting manually, as there are large amounts of data that need to be accounted for. A certain product may presently be in decline, but may see a boost every holiday season. A manual or traditional model of inventory management may be limited to the past few months, and therefore recommend that you cut back on supply. An inventory management system that digs deeper into analysis will realize that the product's demand will likely boost during the holiday season even though it's currently in decline.
- While a business owner will be able to recognize these types of trends over their highest profit or most notable items, it's unlikely that they will be able to notice those trends over hundreds or thousands of inventory items—and that could result in lost revenue. Advanced forecasting makes it possible to capture these insights, even over the largest amounts of inventory and particularly complex inventory chains.

II. LITERATURE SURVEY

Existing Problem

- Demand forecasting is a key component to every growing online business. Without proper demand forecasting processes in place, it can be nearly impossible to have the right amount of stock on hand at any given time.

- A food delivery service has to deal with a lot of perishable raw materials which makes it all the more important for such a company to accurately forecast daily and weekly demand.

- Too much inventory in the warehouse means more risk of wastage, and not enough could lead to out-of-stocks - and push customers to seek solutions from competitors.

- The client is a meal delivery company which operates in multiple cities. They have various fulfillment centers in these cities for dispatching meal orders to their customers. The client wants to help these centers with demand forecasting for upcoming weeks so that these centers will plan the stock of raw materials accordingly.

- The replenishment of majority of raw materials is done on weekly basis and since the raw material is perishable, the procurement planning is of utmost importance. Secondly, staffing of the centers is also one area wherein accurate demand forecasts are really helpful.

- A similar problem has been described for a school food delivery system. First, huge numbers of students have to be handled within a relatively small timespan. Secondly, the cost of food production must be closely regulated in order to stay within budget. Improved forecasting methods will help managers of the school food service face both challenges. Simple mathematical forecasting models were tested and found to be successful in food service environments in hospitals and college and universities. In the school food service environment even some of the same models can be successful. The aim of this article is to define and explain two basic mathematical forecasting models which can be applied to the operations of school food service.

Proposed Solution using Time Series Forecasting

Time series analysis and dynamic modeling is a research interesting area with a great number of applications in business, economics, finance and computer science. The aim of time series analysis is to study the path observations of time series and build a model to describe the structure of data and predict the future values of time series. Due to the importance of time series forecasting in many branches of applied sciences, it is essential to build an effective model with the aim of improving the forecasting

accuracy. A variety of the time series forecasting models have been evolved in the literature.

- Time series are a common data type occurring in many areas and applications such as finance, supply and demand prediction, and health monitoring. Given a vector of historical time series values T = [t1, t2, . . ., tN] ∈ R^N, a prevalent task in time series analysis is to forecast (or "extrapolate") future values tˆN+1, tˆN+2, . . . based on the historical data.
- Time series forecasting methods can be roughly grouped into two main categories: traditional statistical methods and methods based on machine learning models.
- Predictions are made for new data when the actual outcome may not be known until some future date. The future is being predicted, but all prior observations are almost always treated equally. Perhaps with some very minor temporal dynamics to overcome the idea of "*concept drift*" such as only using the last year of observations rather than all data available.

The decomposition of a time series into 4 constituent parts:

- ***Level***-The baseline value for the series if it were a straight line.
- ***Trend***- The optional and often linear increasing or decreasing behavior of the series over time.
- ***Seasonality***-The optional repeating patterns or cycles of behavior over time.
- ***Noise***- The optional variability in the observations that cannot be explained by the model.
- All time series have a level, most have noise, and the trend and seasonality are optional.
- The main features of many time series are trends and seasonal variations … another important feature of most time series is that observations close together in time tend to be correlated (serially dependent).
- Assumptions can be made about these components both in behavior and in how they are combined, which allows them to be modeled using traditional statistical methods.

- These components may also be the most effective way to make predictions about future values, but not always.
- In cases where these classical methods do not result in effective performance, these components may still be useful concepts, and even input to alternate methods.

Time series data often requires cleaning, scaling, and even transformation-

- **Frequency**. Perhaps data is provided at a frequency that is too high to model or is unevenly spaced through time requiring resampling for use in some models.
- **Outliers**. Perhaps there are corrupt or extreme outlier values that need to be identified and handled.
- **Missing**. Perhaps there are gaps or missing data that need to be interpolated or imputed.
- Often time series problems are real-time, continually providing new opportunities for prediction. This adds an honesty to time series forecasting that quickly flushes out bad assumptions, errors in modeling and all the other ways that we may be able to fool ourselves.

Employing Long Short-Term Memory Neural Networks For Time Series Forecasting

- The described problem is addressed by training an LSTM neural network on the historical time series data of market demand to analyze patterns and trends that the data follows.

- After successful training, the LSTM network will be able to provide accurate estimates of market demand in the future. Consequently, the inventory levels can be modelled as a function of the LSTM time series forecast.

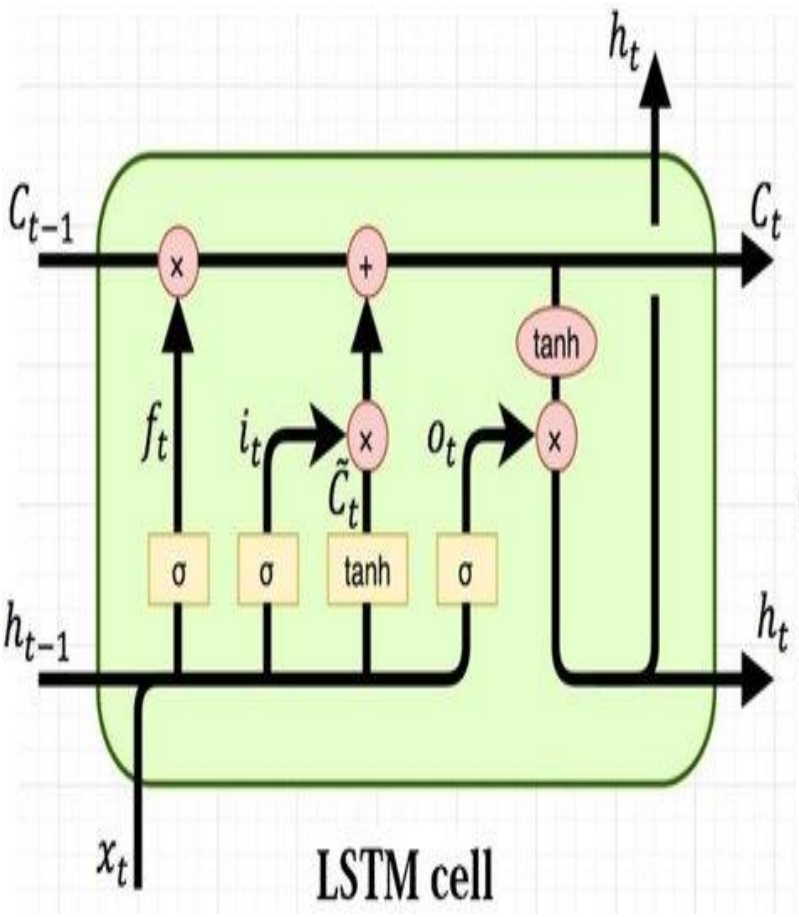
- LSTM networks are well suited for classifying, processing, and making predictions based on time series data, as there may be lags of unknown duration in a time series between important events. LSTMs

have been developed to deal with the problem of vanishing gradients that can be encountered when training traditional RNNs.

- Relative gap-length insensitivity is an advantage of LSTM over RNNs, hidden Markov models and other sequence-learning approaches in many applications.

III. THEORETICAL ANALYSIS

Architecture and Block Diagram



$$i_t = \sigma\left(x_t U^i + h_{t-1} W^i\right)$$

$$f_t = \sigma\left(x_t U^f + h_{t-1} W^f\right)$$

$$o_t = \sigma\left(x_t U^o + h_{t-1} W^o\right)$$

$$\tilde{C}_t = \tanh\left(x_t U^g + h_{t-1} W^g\right)$$

$$C_t = \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right)$$

$$h_t = \tanh(C_t) * o_t$$

There are several LSTM Device architectures. A common architecture consists of a cell (the memory part of the LSTM unit) and three "regulators" of the flow of information inside the LSTM unit, usually called gates: an input gate, an output gate, and a forget gate. Several LSTM unit combinations don't have one or more of these gates, or possibly have other

gates. Gated recurrent units (GRUs) for example, do not have an output gate. Intuitively the cell is responsible for keeping track of the dependencies in the input sequence between the elements. There are links to and from the LSTM portals, some of which are recurrent.

Software Design-

```
# Rolling LSTM
Inputs: series
Outputs: RMSE of the forecasted data
# Split data into 70% training and 30% testing data
1. size ← length(series) * 0.70
2. train ← series[0…size]
3. test ← series[size…length(size)]

# Set the random seed to a fixed value for replication purpose
4. set random.seed(7)

# Fit an LSTM model to training data
Procedure fit_lstm(train, epoch, neurons)
5.    X ← train
6.    y ← train - X
7.    model = Sequential()
8.    model.add(LSTM(neurons), stateful=True))
9.    model.compile(loss='mean_squared_error', optimizer='adam')
10.   for each i in range(epoch) do
11.      model.fit(X, y, epochs=1, shuffle=False)
12.      model.reset_states()
13. end for
return model

# Make a one-step forecast
Procedure forecast_lstm(model, X)
14. yhat ← model.predict(X)
return yhat

15. epoch ← 1
16. neurons ← 4
17. predictions ← empty

# Fit the lstm model
18. lstm_model = fit_lstm(train, epoch, neurons)

# Forecast the training dataset
19. lstm_model.predict(train)

# Walk-forward validation on the test data
20. for each i in range(length(test)) do
21.    # make one-step forecast
22.    X ← test[i]
23.    yhat ← forecast_lstm(lstm_model, X)
24.    # record forecast
25.    predictions.append(yhat)
26.    expected ← test[i]
27. end for

28. MSE ← mean_squared_error(expected, predictions))
29. RMSE ← sqrt(MSE)
Return RMSE
```

LSTM algorithm

a) Backend – Long Short-Term Memory

Time series forecasting using LSTM-

Information on the time series have a normal temporal order. This distinguishes time series analysis from cross-sectional studies in which there is no natural ordering of the observations (e.g., explaining the wages of individuals by reference to their respective levels of education, where data of individuals can be entered in any order). Study of time series is often distinguished from spatial data analysis where results usually refer to geographic locations (e.g. accounting by position for house prices as well as the inherent features of the houses). Analysis of time series can be extended to real valued, continuous data, discrete numerical data or discrete symbolic data.

Keras-

It is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, R, Theano, or PlaidML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible.

Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code. The code is hosted on GitHub, and community support forums include the GitHub issues page, and a Slack channel.

In addition to standard neural networks, Keras has support for convolutional and recurrent neural networks. It supports other common utility layers like dropout, batch normalization, and pooling.

Keras allows users to productize deep models on smartphones (iOS and Android), on the web, or on the Java Virtual Machine. It also allows use of distributed training of deep-learning models on clusters of Graphics processing units (GPU) and tensor processing units (TPU) principally in conjunction with CUDA.

Adam optimizer-

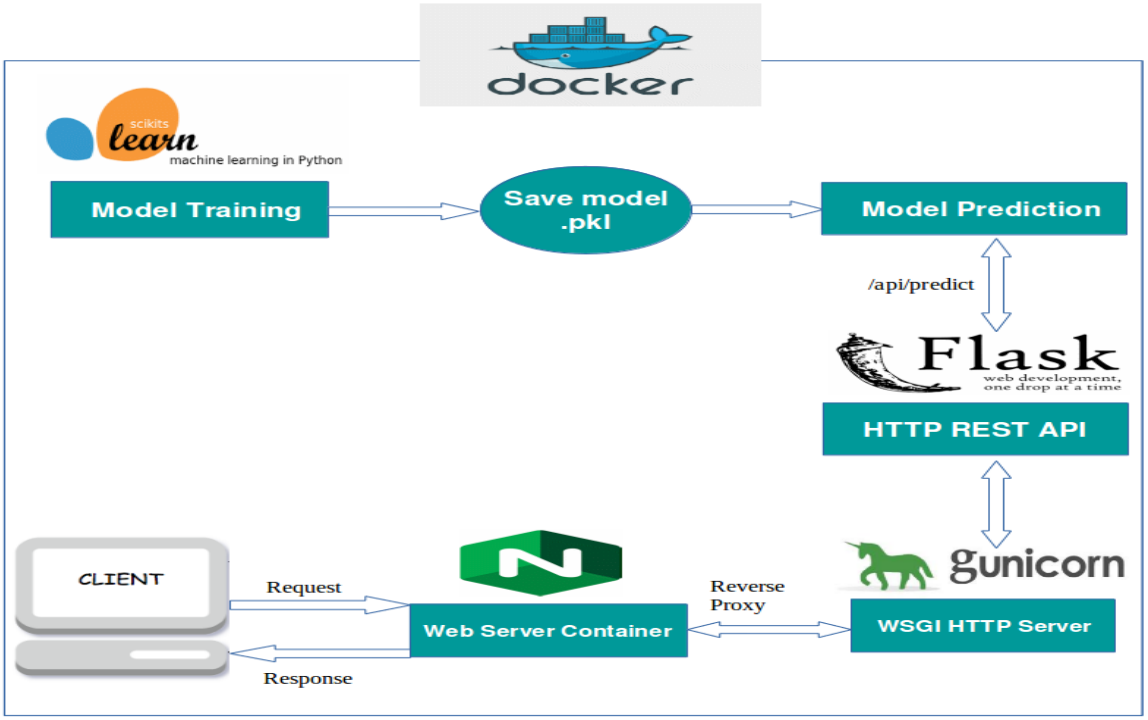
Adam is a method for efficient stochastic optimization that only requires first-order gradients with little memory requirement. The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients; the name Adam is derived from adaptive moment estimation. The method is designed to combine the advantages of two recently popular methods: AdaGrad, which works well with sparse gradients, and RMSProp, which works well in on-line and non-stationary settings. Some of Adam's advantages are that the magnitudes of parameter updates are invariant to rescaling of the gradient, its step sizes are approximately bounded by the step size hyperparameter, it does not require a stationary objective, it works with sparse gradients, and it naturally performs a form of step size annealing.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

Adam optimization algorithm

a) Frontend –

Web application deployment using Flask-Flask is a Python-written micro Application framework. This is known as a microframework because it does not need special resources or libraries. It does not provide a database abstraction layer, type validation or any other components where pre-existing third-party libraries have basic functionality. Flask does, however, allow plugins that can incorporate program functionality as if they were introduced in Flask itself. Extensions exist for object-relational mappers, type validation, handling of uploads, various open authentication frameworks and other methods relevant to common system. Extensions are much more regularly revised than the main Flask system.

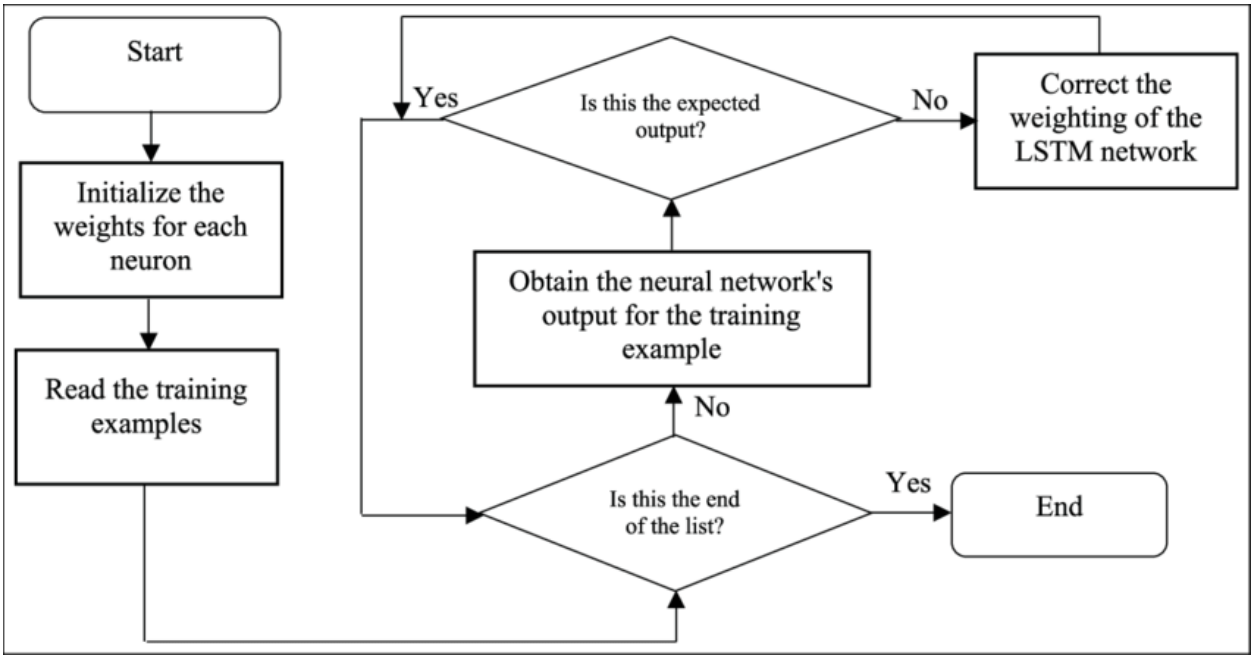


IV. EXPERIMENTAL INVESTIGATIONS

The LSTM Time series model was trained for 10 epochs, 18 steps per epoch and the progress of the training and validation loss were analyzed. The values of training and validation loss at the end of 10 epochs were 0.129 and

0.359 respectively. The results were obtained in the form of a csv file. The model was trained on Keras platform using TensorFlow backend.

V. FLOWCHART



VI. ADVANTAGES & DISADVANTAGES

Advantages-

- Neural networks provide the ability to know potentially chaotic and nonlinear relationships with randomly specified but constant numbers of multivariate and multi-step forecasting inputs and outputs.
- Recurrent neural networks add explicit handling of ordered observations and promise to learn temporary context-dependence.

Disadvantages-

- LSTMs need more time to train.
- LSTM's need more resources to practice
- LSTMs are easy to overfit
- Dropout is much more difficult to incorporate in LSTMs
- LSTMs are prone to different initializations of random weights

VII. APPLICATIONS

- The provision of reliable inputs to the modeling process is predicated on successful transport infrastructure or supply chain architecture. The demands imposed on the system are among the most significant inputs. In the face of uncertainty, modeling methods are used to anticipate what the demands will be on the network in the future so that appropriate prototypes and operational plans can be formulated.
- Demands are not the only unpredictable factors which may involve forecasting when designing decision-making support models for supply chain and transport systems. Equally relevant to predict could be the operating conditions on the supply side.
- Suppose, one would like to create a model for truck routing between various customer locations. The model's goal is to define minimal travel time tours. It is clear that parameters of travel time are required between customer locations and depots. It is possible to use the predictive techniques.

VIII. RESULTS AND CONCLUSION

An LSTM neural network was successfully trained for food demand forecasting from week 145 to week 155. The results were generated in the form of a csv file. The LSTM model was deployed for real time training using Flask library in Python.
The user can upload a custom dataset conforming to the specified format, train the LSTM model and generate results for the custom dataset. The training is real time unlike other static models.

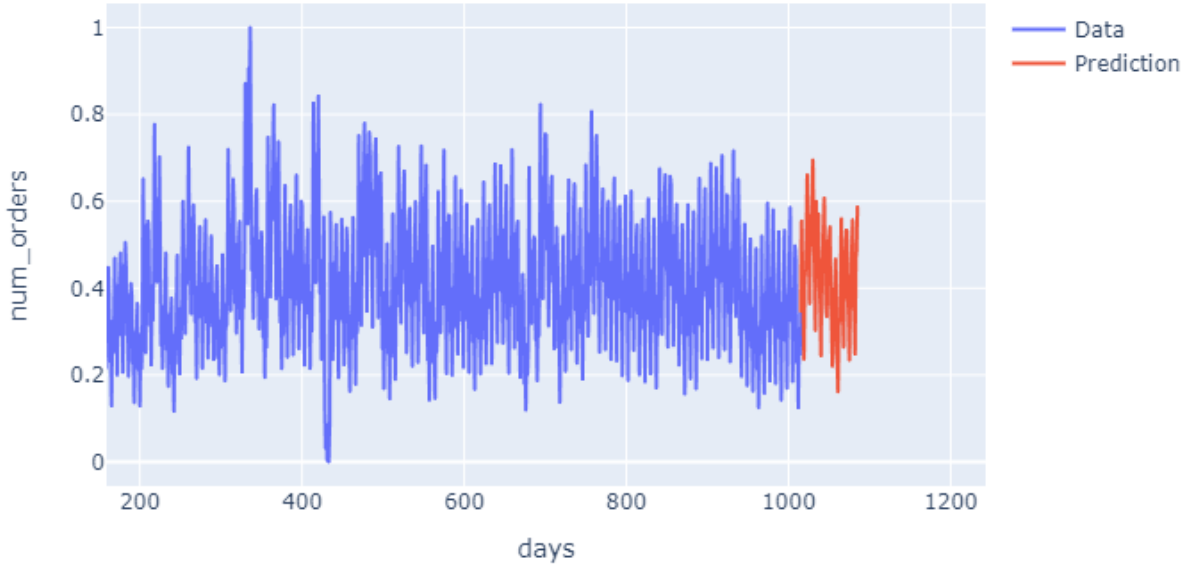
IX. FUTURE SCOPE

Training a custom neural network for to perform regression and predict demand for week corresponding to center-meal id pair. A primitive model with high training time has been implemented. Another future endeavor can be to optimize hyperparameters and reduce training time of the network.

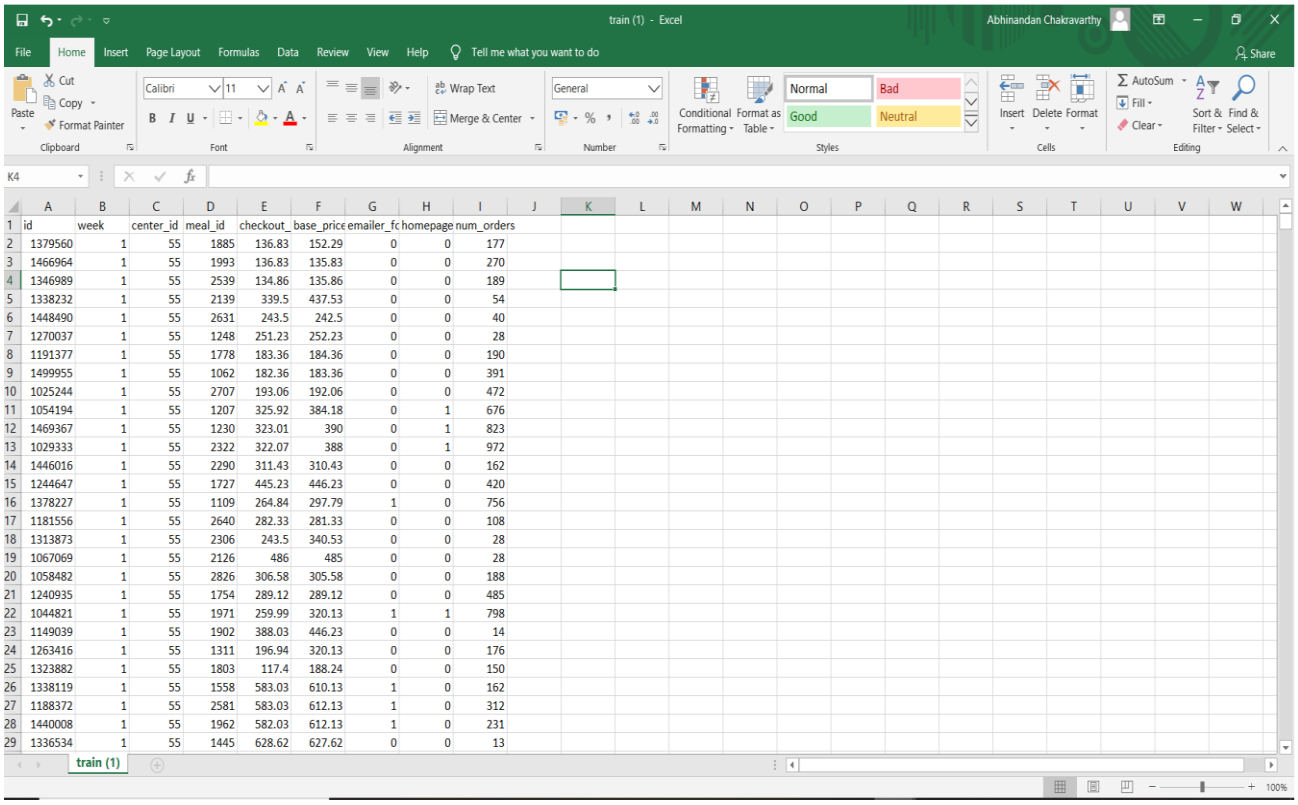
X. BIBLIOGRAPHY

- S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997
- F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with LSTM," *Neural Computation*, vol. 12, no. 10, pp. 2451–2471, 2000.
- F. A. Gers and J. Schmidhuber, "LSTM recurrent networks learn simple context free and context sensitive languages," *IEEE Transactions on Neural Networks*, 2001. accepted.
- Weigend and N. Gershenfeld, *Time Series Prediction: Forecasting the Future and Understanding the Past*. Addison-Wesley, 1993".
- S. P. Day and M. R. Davenport, "Continuous-time temporal back-progagation with adaptive time delays," *IEEE Transactions on Neural Networks*, vol. 4, pp. 348–354, 1993."

APPENDIX

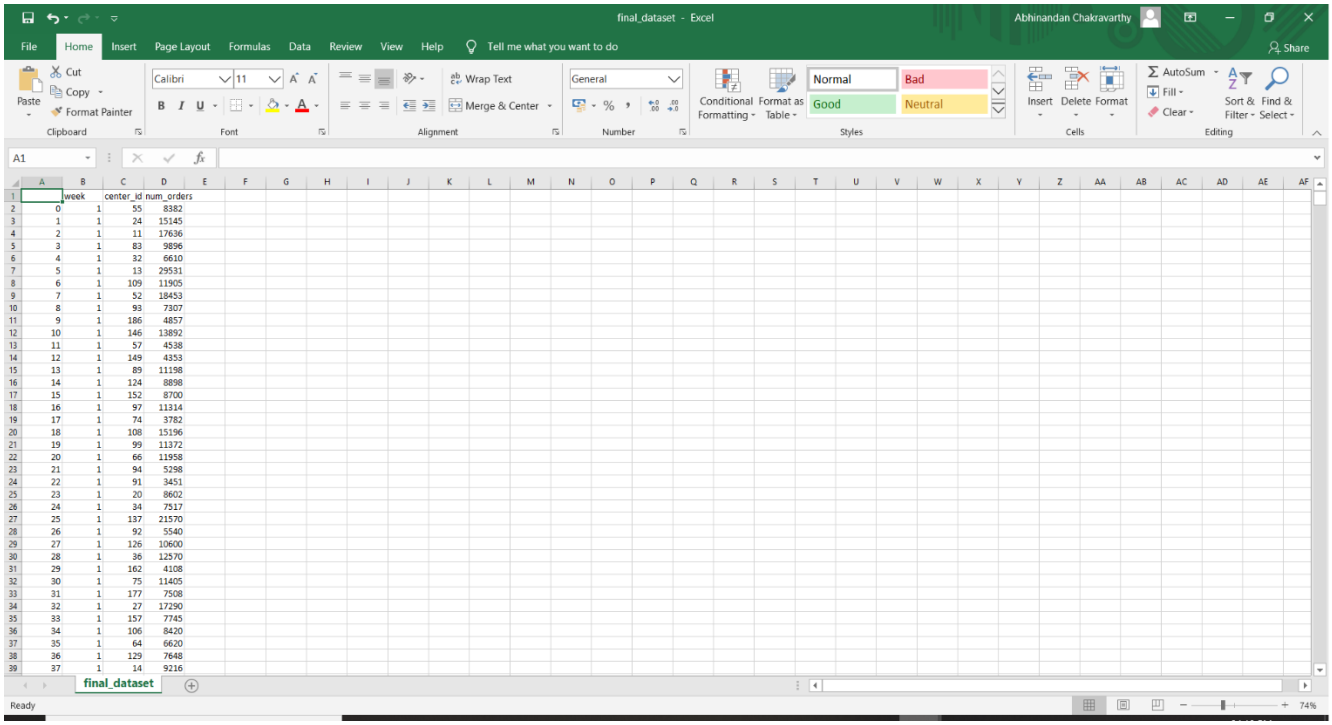


Graphical Representation of output



id	week	center_id	meal_id	checkout_	base_price	emailer_fc	homepage	num_orders
1379560	1	55	1885	136.83	152.29	0	0	177
1466964	1	55	1993	136.83	135.83	0	0	270
1346989	1	55	2539	134.86	135.86	0	0	189
1338232	1	55	2139	339.5	437.53	0	0	54
1448490	1	55	2631	243.5	242.5	0	0	40
1270037	1	55	1248	251.23	252.23	0	0	28
1191377	1	55	1778	183.36	184.36	0	0	190
1499955	1	55	1062	182.36	183.36	0	0	391
1025244	1	55	2707	193.06	192.06	0	0	472
1054194	1	55	1207	325.92	384.18	0	1	676
1469367	1	55	1230	323.01	390	0	1	823
1029333	1	55	2322	322.07	388	0	1	972
1446016	1	55	2290	311.43	310.43	0	0	162
1244647	1	55	1727	445.23	446.23	0	0	420
1378227	1	55	1109	264.84	297.79	1	0	756
1181556	1	55	2640	282.33	281.33	0	0	108
1313873	1	55	2306	243.5	340.53	0	0	28
1067069	1	55	2126	486	485	0	0	28
1058482	1	55	2826	306.58	305.58	0	0	188
1240935	1	55	1754	289.12	289.12	0	0	485
1044821	1	55	1971	259.99	320.13	1	1	798
1149039	1	55	1902	388.03	446.23	0	0	14
1263416	1	55	1311	196.94	320.13	0	0	176
1323882	1	55	1803	117.4	188.24	0	0	150
1338119	1	55	1558	583.03	610.13	1	0	162
1188372	1	55	2581	583.03	612.13	1	0	312
1440008	1	55	1962	582.03	612.13	1	0	231
1336534	1	55	1445	628.62	627.62	0	0	13

Training Dataset



	week	center_id	num_orders
0	1	55	8382
1	1	24	15145
2	1	11	17636
3	1	83	9896
4	1	32	6610
5	1	13	29531
6	1	109	11905
7	1	52	18453
8	1	93	7307
9	1	186	4857
10	1	146	13892
11	1	57	4538
12	1	149	4353
13	1	89	11198
14	1	124	8898
15	1	152	8700
16	1	97	11314
17	1	74	3782
18	1	108	15196
19	1	99	11372
20	1	66	11958
21	1	94	5298
22	1	91	3451
23	1	20	8602
24	1	34	7517
25	1	137	21570
26	1	92	5540
27	1	126	10600
28	1	36	12570
29	1	162	4108
30	1	75	11405
31	1	177	7508
32	1	27	17290
33	1	157	7745
34	1	106	8420
35	1	64	6620
36	1	129	7648
37	1	14	9216

Result dataset

- The data was assimilated from kaggle. The data consisted of excel files comprising of fulfilment center information, meal id and types and historical data going back about 2.8 years.
- The data was cleaned by removing/replacing alpha, alpha-numeric values to only integer/float values.
- Redundant columns that do not contribute to the prediction result were identified and eliminated.
- LSTM requires 3 dimensional data input for training and prediction, the columns are 1 dimensional array, hence they are reshaped into 3 dimensional format using numpy library. Mainly the number of orders is reshaped to confer with LSTM input format.
- The contributing columns like week, center ID and number of orders comprise the final dataset which is split into training, testing and validation by applying appropriate split ratio.
- The deep learning model is built using keras, tensorflow backend by using appropriate optimizer and activation function injected in every layer of the network.
- The input shape of the LSTM is a 3d array (1,x,1).The model employs the use of Keras Time Series Generator for predicting in the future and the evaluation metric used is RMSE (Root mean squared error).
- The model has 3 LSTM layers and a fully connected dense layer with a dropout of 20% between each LSTM layer.
- The dummy dataset consists of data of week 1, center-id 55 and its corresponding number of orders, the dummy dataset is fed into the model to obtain predictions.
- Once model trains on dummy dataset successfully, without any errors, the final dataset is fed into the model for training.
- The LSTM layers are altered by adding more/ removing layers, changing optimizers and activation functions.
- Extensive debugging is done by reshaping the data, altering batch size and number of epochs.

Source code-

```
from flask import Flask, make_response, request
import pandas as pd, numpy as np, io, operator, csv
from sklearn.preprocessing import MinMaxScaler
import keras
import tensorflow as tf
from keras.preprocessing.sequence import TimeseriesGenerator
from keras.models import Sequential
from keras.layers import LSTM, Dense ,Dropout
from keras.optimizers import RMSprop
app = Flask(__name__)
def transform():
    cols=['week','center_id','num_orders']
    df1=pd.read_csv('train.csv',usecols=cols)
    uniques=df1.center_id.unique()
    orders_per_center=[]
    for i in range(1,146):
        for j in uniques:
            orders_per_center.append(df1[df1['week']==i][df1['center_id']==j]['num_orders'].sum())
    weeks=[]
    centerids=[]
    for i in range(1,146):
        for j in range(0,77):
            centers=uniques[j]
            weeks.append(i)
            centerids.append(centers)
    finals=pd.DataFrame({'week':weeks,'center_id':centerids,'num_orders':orders_per_center})
    df_final=finals
    df=df_final.dropna()
    df.set_axis(df['week'], inplace=True)
    last_dates = df['week'].values[-1]
    uniques=df.center_id.unique()
    close_data = df['num_orders'].values
    close_data = close_data.reshape((-1,1))
    sc= MinMaxScaler(feature_range=(0,1))
    close_data = sc.fit_transform(close_data)
    split_percent = 0.98
    split = int(split_percent*len(close_data))
    close_train = close_data[:split]
    close_test = close_data[split:]
    date_train = df[['week','center_id']][:split]
    date_test = df[['week','center_id']][split:]
    print(len(close_train))
    print(len(close_test))
    look_back = 200
    train_generator = TimeseriesGenerator(close_train, close_train, length=look_back, batch_size=64)
    test_generator = TimeseriesGenerator(close_test, close_test, length=look_back, batch_size=1)
```

```
    model = Sequential()
    model.add(LSTM(32,return_sequences=True,input_shape=(look_back,1)))
    model.add(Dropout(0.2))
    model.add(LSTM(32))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mse')
    num_epochs = 1
    model.fit_generator(train_generator, epochs=num_epochs, verbose=1)
    prediction = model.predict_generator(test_generator)
    close_train = close_train.reshape((-1))
    close_test = close_test.reshape((-1))
    prediction = prediction.reshape((-1))
    close_data = close_data.reshape((-1))
    def predict(num_prediction, model):
        prediction_list = close_data[-look_back:]
        for _ in range(num_prediction*77):
            x = prediction_list[-look_back:]
            x = x.reshape((1, look_back, 1))
            out = model.predict(x)[0][0]
            prediction_list = np.append(prediction_list, out)
        prediction_list = prediction_list[look_back-1:]
        return prediction_list
    def predict_ids(num_prediction):
        centerid=[]
        for i in range(1,num_prediction+1):
            for j in range(0,77):
                centerid.append(uniques[j])
        return centerid
    num_prediction = 10
    forecast = predict(num_prediction, model)
    forecast_ids = predict_ids(num_prediction)
    forecast_vals=sc.inverse_transform(forecast.reshape(-1,1))
    forecast=forecast_vals.reshape((-1))
    forecast=forecast.tolist()
    forecast.pop(0)
    forecastss=np.array(forecast)
    weeeks=[]
    for i in range(1,11):
        for j in range(0,77):
            weeeks.append(last_dates+i)
    finals=pd.DataFrame({'weeks':weeeks,'center_id':forecast_ids,'num_orders':forecastss})
    return finals
@app.route('/')
def form():
  return """
    <html>
        <body style="background-color:#33FFEC;">
```

```
        <h1 style="text-align: center; color:#FF4933;" ><strong>OPTIMIZED WAREHOUSE
MANAGEMENT</strong></h1>
<hr style="border: 2px dashed black;width:80%" />
<br>
<h3><em><span style="text-decoration: underline;margin-left:70px">Please Upload The CSV
File</span></em></h3>
<form action="/transform" method="post" enctype="multipart/form-data">
<input type="file" name="data_file" style="background-color: #555;color: #fff;border-radius: 30px;text-
align: center;border: 3px outset buttonface;fontstyle:bold;font-size:16px;margin-top:15px;margin-
left:70px"/>


<input type="submit" value="SUBMIT FILE" style="margin-left:110px;background-color:
#FFDE7A;border-radius: 30px;text-align: center;border: 2px outset buttonface;fontstyle:bold;font-
size:16px;margin-top:15px"/>

<hr style="border: 1.25px dashed black;margin-top:25px;width:90%" />
<br>


<p><span style="color: #ff0000;text-decoration: underline;margin-
left:70px"><em>INSTRUCTIONS</em></span></p>
<ol>
<li><span style="color: #050200;"><em>Please Upload only the Historical Data
file.</em></span></li><br>
<li><span style="color: #050200;"><em>The Historical Data file must contain two columns: week and
corresponding demand</em></span></li><br>
<li><span style="color: #050200;"><em>Click on the Choose file button to upload the
file.</em></span></li><br>
<li><span style="color: #050200;"><em>Click on the SUBMIT FILE button to submit the
file.</em></span></li><br>
<li><span style="color: #050200;"><em>Please wait for some time while training takes
place.</em></span></li><br>
<li><span style="color: #050200;"><em>The prediction file automatically gets
downloaded.</em></span></li><br>
<li><span style="color: #050200;"><em>The predictions are in the result.csv file. </em></span></li>
</ol>
        </form>
      </body>
   </html>
"""

@app.route('/transform', methods=["POST"])
def transform_view():
  request_file = request.files['data_file']
  request_file.save("train.csv")
  if not request_file:
    return "No file"
```

```
  result = transform()
  print(result)
  response = make_response(result.to_csv())
  response.headers["Content-Disposition"] = "attachment; filename=result.csv"
  return response
if __name__ == '__main__':
    app.run(debug=True)#app.run(host='0.0.0.0', port=8005, debug=True)
```