FUTURE SALE PREDICTION PROJECT

Introduction:

Predicting future sales is a critical aspect of business planning and strategy. Whether you are a retailer, manufacturer, or service provider, having an accurate sales prediction model can help you make informed decisions, optimize inventory, and allocate resources more efficiently. In today's rapidly evolving business landscape, where data is abundant and technology is advancing at an unprecedented rate, the ability to forecast sales has never been more crucial.

Sales prediction involves analyzing historical sales data, market trends, consumer behavior, and various other factors to forecast future sales figures. With the help of advanced analytics, machine learning, and artificial intelligence, businesses can gain deeper insights into their sales patterns and make data-driven decisions. This not only aids in meeting customer demand but also in minimizing costs and maximizing profits.

In this introduction, we will explore the importance of sales prediction, the key factors involved, and the methods used to make accurate sales forecasts. We will also touch upon the potential benefits of having a reliable sales prediction model in place.

1.ARTIFICIAL NEURAL NETWORKS:

Neural Networks (ANNs) are machine learning algorithms that are designed to simulate the structure and function of the human brain. They can be used to make predictions about sales by considering complex relationships between inputs and outputs, providing a more nuanced and accurate sales forecast.

Required packages and Installation:

1. numpy
2. pandas
3. keras
4. tensorflow
5. csv
6. matplotlib.pyplot

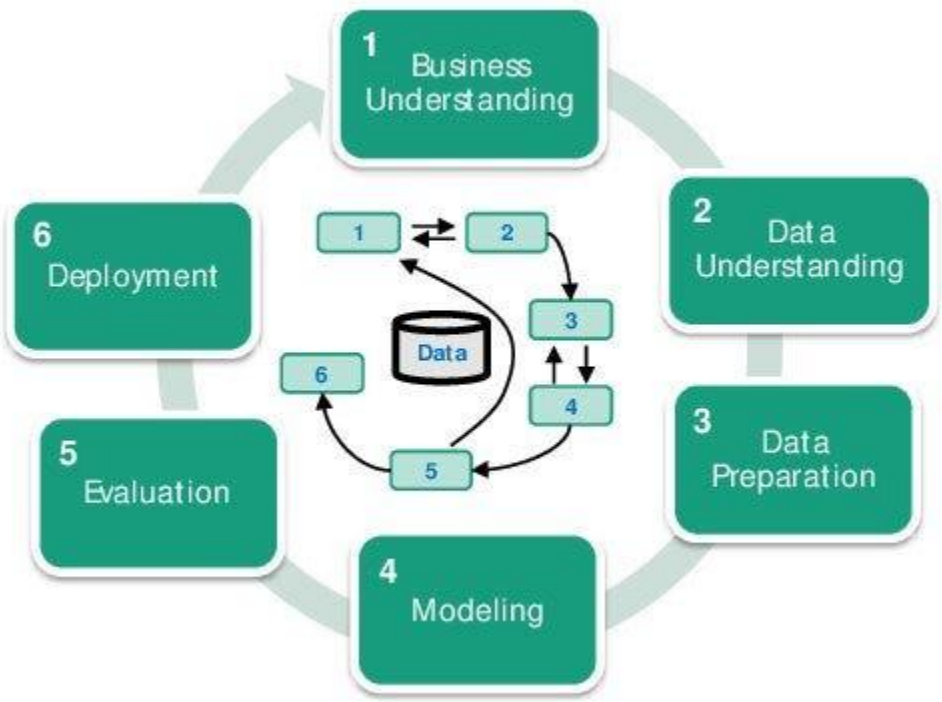
Sales prediction using Regression Analysis Problem Statement:

Regression is an important machine learning model for these kinds of problems. Predicting sales of a company needs time series data of that company and based on that data the model can predict the future sales of that company or product. So, in this research project we will analyze the time series sales data of a company and will predict the sales of the company for the coming quarter and for a specific product. For this kind of project of sales predict, we will apply the linear regression and logistic regression and evaluate the result based on the training, testing and validation set of the data.

Source: https://vishal2develop.github.io/flexible-jekyll/Bigmart-Sales-Prediction/

Importance of the problem and motivation for the implementation

Deployment is the final stage of the CRISP-DM (Cross-Industry Standard Process for Data Mining) model, where the developed model is integrated into a production environment to solve real-world problems. In this case, the webapp developed using Streamlit and Machine Learning models is aimed at predicting sales for a grocery shop.



Accurately predicting sales is an essential part of running a successful business, especially in the highly competitive retail industry. Retailers need to understand the patterns and trends of their customers' purchasing behavior to optimize inventory management, improve product placement, and develop effective marketing strategies. Predicting sales with reasonable accuracy allows retailers to make informed decisions that drive revenue growth and improve the bottom line.

Traditional methods of sales forecasting are often based on historical data, which might not capture the complex relationships between sales and other variables, such as promotions, weather conditions, and economic indicators. However, with the

advancement of Machine Learning and Artificial Intelligence techniques, it is now possible to leverage large amounts of data to generate accurate sales predictions.

The motivation for implementing a webapp to predict sales for a grocery shop is to provide retailers with a tool that leverages advanced Machine Learning algorithms to accurately predict sales. With this webapp, retailers can improve their decision-making process and take advantage of the insights generated by the model to make data-driven decisions.

By developing a webapp that makes it easy for retailers to input their data and generate predictions, we can help businesses of all sizes take advantage of the latest technology and drive growth. With this implementation, retailers can leverage the power of Machine Learning to gain insights into their customers' behavior and make informed decisions that drive revenue growth and improve the bottom line.

g applications. It provides built-in support for popular data science libraries such as pandas, NumPy, and Scikit-learn, and makes it easy to integrate machine learning models into web applications.

However, Streamlit also has some limitations. Since it is a relatively new framew Introduction to Streamlit

Streamlit is an open-source Python framework that allows developers to easily create web applications and data visualizations. It provides a simple and intuitive way to build interactive and responsive user interfaces without requiring any knowledge of web development languages such as HTML, CSS, or JavaScript.

Streamlit uses a reactive programming model, where changes in the user interface automatically trigger updates to the application's output. This makes it easy to build applications that respond dynamically to user input, such as filtering or sorting data, adjusting model parameters, or displaying real-time updates.

One of the key advantages of Streamlit is its ease of use. It has a simple and intuitive API that allows developers to quickly prototype and build interactive web applications with minimal code. It also provides a wide range of built-in widgets and visualizations, such as sliders, dropdowns, charts, and maps, that can be easily customized and extended.

Another advantage of Streamlit is its focus on data science and machine learninork, it may not have all the features and capabilities of more established web development frameworks. It also has some performance limitations, particularly when dealing with large datasets or computationally intensive machine learning models.

Despite these limitations, Streamlit is a powerful and flexible framework that can help developers quickly build and deploy interactive web applications and data visualizations. Its simplicity, ease of use, and focus on data science make it an ideal choice for many machine learning and data science projects.

II. Architecture and Design

Explanation of the overall architecture and design choices

The overall architecture of the grocery sales prediction webapp can be divided into several steps, each with its own set of design choices.

1. Importing packages and loading saved models: The first step is to import the necessary packages and load the trained models that will be used for prediction. In the provided code, we import the necessary packages such as streamlit, pandas, and joblib. We then load the pre-trained models for the prediction task using joblib.

2. Creating input fields: The second step is to create input fields where users can input data such as the store type, location, and date. The input fields are created using the streamlit library.

3. Converting input data to a pandas DataFrame: Once the user inputs are received, they are converted into a pandas DataFrame. This allows for easier manipulation and processing of the input data.

4. Selecting categorical and numerical columns separately: In the next step, the categorical and numerical columns are selected separately. This is necessary because different preprocessing steps are applied to each type of column.

5. Applying the imputers: Missing values are common in datasets, and imputing them with suitable values can improve the accuracy of the model. In the provided code, we apply the SimpleImputer from sklearn to handle missing values.

6. Encoding the categorical columns: Machine learning models require numerical data to work. Categorical columns need to be converted to numerical format through one-hot encoding. In the provided code, we use the OneHotEncoder from sklearn to perform this task.

7. Scaling the numerical columns: It is important to scale the numerical columns so that the range of values is consistent across all features. In the provided code, we use the StandardScaler from sklearn to scale the numerical columns.

8. Joining the categorical encoded and numerical scaled data: Once the categorical columns have been encoded and the numerical columns have been scaled, they are concatenated into a single DataFrame.

9. Making a prediction: With the preprocessed input data, the pre-trained model is used to make a prediction.

10. Displaying the prediction: Finally, the predicted sales value is displayed to the user on the webapp.

These design choices were made to ensure that the webapp is able to accurately predict grocery sales based on user input. By following these steps and using

appropriate preprocessing techniques, we can ensure that the model is able to make accurate predictions with minimal input from the user.

Advantages and disadvantages of the chosen architecture

Advantages:

- Separation of concerns: The use of functions and classes for specific tasks allows for a modular and organized approach to development. Each component can be developed, tested, and maintained independently.
- Scalability: The architecture is designed to be scalable by allowing the addition of more models and features as required. The use of pandas DataFrame also allows for the easy handling of large datasets.
- Flexibility: The architecture is flexible in terms of data input and output. It allows for the use of various data sources and the deployment of the model to different platforms.

Disadvantages:

- Complexity: The architecture may be too complex for small projects that do not require the use of multiple models or features.
- Time-consuming: The development of the architecture may be time-consuming, especially when compared to simpler approaches.
- Learning curve: The use of classes and functions may require some learning for developers who are not familiar with object-oriented programming.

III. Implementation Details

1. Import saves models and packages: In this section, we import all the necessary libraries and packages required to build our web app. These include pandas,

numpy, scikit-learn, and Streamlit. We also load the saved model (in this case, the trained model) that we will use for predictions.

```
import pandas as pd
import streamlit as st
import numpy as np
from matplotlib import pyplot as plt
import pickle
import sklearn
import joblib
from PIL import Image
import base64


num_imputer = joblib.load('numerical_imputer.joblib')
cat_imputer = joblib.load('categorical_imputer.joblib')
encoder = joblib.load('encoder.joblib')
scaler = joblib.load('scaler.joblib')
dt_model = joblib.load('Final_model.joblib')
```

This code section is used to load the trained models and transformers for the machine learning pipeline. Joblib is a popular library used for efficient caching of Python objects, especially large numerical data, and storing them to disk.

In this case, the trained machine learning models and transformers are saved as joblib files, and then loaded using the joblib.load() function. The models and transformers are then ready to be used for making predictions on new input data.

The advantage of using joblib for model and transformer storage is its speed and ability to handle large data objects efficiently. It is also designed to work well with numerical data, which is often the case in machine learning applications.

2. Create the input fields: The input fields are created using the Streamlit functions. We create input fields for the categorical and numerical features of our dataset that are necessary to make predictions.

```
input_data = {}
col1,col2 = st.columns(2)
with col1:
    input_data['store_nbr'] = st.slider("store_nbr",0,54)
```

```
    input_data['products'] = st.selectbox("products", ['AUTOMOTIVE', 'CLEANING', 'BEAUTY',
'FOODS', 'STATIONERY',
       'CELEBRATION', 'GROCERY', 'HARDWARE', 'HOME', 'LADIESWEAR',
       'LAWN AND GARDEN', 'CLOTHING', 'LIQUOR,WINE,BEER', 'PET SUPPLIES'])
    input_data['onpromotion'] =st.number_input("onpromotion",step=1)
    input_data['state'] = st.selectbox("state", ['Pichincha', 'Cotopaxi', 'Chimborazo',
'Imbabura',
       'Santo Domingo de los Tsachilas', 'Bolivar', 'Pastaza',
       'Tungurahua', 'Guayas', 'Santa Elena', 'Los Rios', 'Azuay', 'Loja',
       'El Oro', 'Esmeraldas', 'Manabi'])
    input_data['store_type'] = st.selectbox("store_type",['D', 'C', 'B', 'E', 'A'])
    input_data['cluster'] = st.number_input("cluster",step=1)

with col2:
    input_data['dcoilwtico'] = st.number_input("dcoilwtico",step=1)
    input_data['year'] = st.number_input("year",step=1)
    input_data['month'] = st.slider("month",1,12)
    input_data['day'] = st.slider("day",1,31)
    input_data['dayofweek'] = st.number_input("dayofweek,0=Sun and 6=Sat",step=1)
    input_data['end_month'] = st.selectbox("end_month",['True','False'])
```

This code block is creating input fields for the user to input their data. The user interface is created using Streamlit's columns function, which divides the interface into two columns. The left column has input fields for 'store_nbr', 'products', 'onpromotion', 'state', 'store_type', and 'cluster', while the right column has input fields for 'dcoilwtico', 'year', 'month', 'day', 'dayofweek', and 'end_month'.

The widgets used in the left column are:

- `st.slider()`: This creates a slider widget for the 'store_nbr' input. The slider ranges from 0 to 54, allowing the user to select a value in that range.
- `st.selectbox()`: This creates a drop-down menu widget for the 'products' input. The drop-down menu allows the user to select a product category from the given options.
- `st.number_input()`: This creates a number input widget for the 'onpromotion' input. The widget accepts integer values, and the step value is set to 1.
- `st.selectbox()`: This creates a drop-down menu widget for the 'state' input. The drop-down menu allows the user to select a state from the given options.

- `st.selectbox()`: This creates a drop-down menu widget for the 'store_type' input. The drop-down menu allows the user to select a store type from the given options.
- `st.number_input()`: This creates a number input widget for the 'cluster' input. The widget accepts integer values, and the step value is set to 1.

The widgets used in the right column are:

- `st.number_input()`: This creates a number input widget for the 'dcoilwtico' input. The widget accepts floating-point values, and the step value is set to 1.
- `st.number_input()`: This creates a number input widget for the 'year' input. The widget accepts integer values, and the step value is set to 1.
- `st.slider()`: This creates a slider widget for the 'month' input. The slider ranges from 1 to 12, allowing the user to select a value in that range.
- `st.slider()`: This creates a slider widget for the 'day' input. The slider ranges from 1 to 31, allowing the user to select a value in that range.
- `st.number_input()`: This creates a number input widget for the 'dayofweek' input. The widget accepts integer values, and the step value is set to 1. The user is instructed to input 0 for Sunday and 6 for Saturday.
- `st.selectbox()`: This creates a drop-down menu widget for the 'end_month' input. The drop-down menu allows the user to select either 'True' or 'False'.

3. Convert the input data to a pandas DataFrame: After creating the input fields, the user inputs are stored as variables. We then use these variables to create a pandas DataFrame that can be used as input to the model for predictions.

```
input_df = pd.DataFrame([input_data])
```

4. Selecting categorical and numerical columns separately: In this section, we separate the categorical and numerical columns from the input DataFrame to apply preprocessing steps separately. We store the categorical and numerical column names in separate variables.

```
# Selecting categorical and numerical columns separately
     cat_columns = [col for col in input_df.columns if input_df[col].dtype == 'object']
     num_columns = [col for col in input_df.columns if input_df[col].dtype != 'object']
```

5. Apply the imputers: We then apply the `imputer` model imported from our notebook to fill in missing values in the numerical columns. For the categorical columns, we impute missing values using the mode value of each column.

```
# Apply the imputers on the input data
     input_df_imputed_cat = cat_imputer.transform(input_df[cat_columns])
     input_df_imputed_num = num_imputer.transform(input_df[num_columns])
```

6. Encode the categorical columns: After imputing missing values in the categorical columns, we apply one-hot encoding using the `imported encoder`model from our notebook to convert the categorical columns into binary columns.

```
# Encode the categorical columns
     input_encoded_df = pd.DataFrame(encoder.transform(input_df_imputed_cat).toarray(),
                                     columns=encoder.get_feature_names_out(cat_columns))
```

7. Scale the numerical columns: Next, we apply feature scaling to the numerical columns using the `imported encoder model`function from our notebook. This ensures that all the features are on the same scale and avoids the issue of one feature dominating the others.

```
# Scale the numerical columns
input_df_scaled = scaler.transform(input_df_imputed_num)
input_scaled_df = pd.DataFrame(input_df_scaled , columns = num_columns)
```

8. Joining the cat encoded and num scaled: In this section, we join the categorical encoded and numerical scaled columns into a single DataFrame that can be used as input to the model.

```
#joining the cat encoded and num scaled
final_df = pd.concat([input_encoded_df, input_scaled_df], axis=1)
```

9. Make a prediction: Finally, we use the trained XGBoost model to make a prediction on the user input data.

```
# Make a prediction
        prediction = dt_model.predict(final_df)[0]
```

10. Display the prediction: We display the prediction to the user using the Streamlit `write` function. The user can then view the predicted sales value for the input data they provided.

```
        st.write(f"The predicted sales are: {prediction}.")
        input_df.to_csv("data.csv", index=False)
        st.table(input_df)
```

Methodology:

In this research, linear regression and logistic regression model will be trained and tested for our dataset. For this we will download the sample dataset from the given link in dataset section. The raw data is then undergoes for feature selection and feature extraction. After that we will apply machine learning regression models for the training dataset to train the model. This train model will be then tested on test dataset and validation dataset for checking the accuracy of the model.

Dataset We are using the superstore sales data for sales prediction. Sample data that appears in the December Tableau User Group presentation. Note: Geographic locations have been altered to include Canadian locations (provinces / regions). The link for sample dataset is

https://community.tableau.com/docs/DOC-1236

Evaluation Measures Measures such as Classification error, Computational cost, Accuracy can be used for calculating the accuracy of drug discovery using neural network.

Software and Hardware Requirements Python based Computer Vision and Deep Learning libraries will be exploited for the development and experiment

Sales Projection Vs. Sales Forecasting

The terms **sales projection** and **sales forecasting** are often used interchangeably. While they have a lot in common, they are two different things.

Sales Projection Vs. Sales Forecasting

Sales forecasting

- Estimates most likely sales
- Efficient for short-term predictions
- Works with historical data and real-time data

Sales projections

- Estimates desired sales results
- Efficient in long-term planning
- Works with historical, real-time, and predicted data

Sales forecasting uses real-life and historical data to calculate and predict realistic sales numbers that can be achieved in the foreseeable future.

It's a technique that businesses use to plan their budgeting, revenue, and finances for short periods of time, such as a quarter or a year, based on current information and factors that are more or less in their control.

While sales forecasts are, more often than not, accurate and efficient, as time passes, too many of the relevant factors they rely on become unstable and difficult to foresee. And this reduces their dependability.

SALES PROJECTION:

Sales projections include large-scale future planning and focus more on the sales the business *wants* to achieve, rather than those that the hard data show.

In other words, when they are conducting sales projections, companies set sales goals that they desire to achieve in the long run, and reverse-engineer how it can be done.

Sales projections are used in financial and strategic planning to measure the success of a business and its growth and development potential. They can also be used in risk assessment, investor hunting, talent acquisition, and resources management.

Here's how you can leverage them in your business:

- **Strategize your organization's development**. By estimating future sales and what is required to achieve them, you can plan and strategize growth. The data can help your decision-making process and, potentially, reduce the risks of failure.
- **Device new business models**. Based on your product's specifications and the market demand trends, you can construct business models that will facilitate your desired sales figures.
- **Plan resources over long periods of time**. With the data from the sales projections, you can calculate the resources needed for your business to grow and develop. This includes financing, budgets, manpower, supplies, organizational expenses, and everything else your business needs.
- **Attract new investors and financial support**. With a clear plan for the future and data-backed sales projections, you can foresee the financial support you need in terms of loans and investments. Furthermore, you can use the information to justify your needs to stakeholders and plan your finances.

How to Project Future Sales

1. List All Your Product and Services
2. Take into Account All Sales Channels

3. Analyze Current and Historical Data
4. Collect and Review PESTEL Information
5. Set Sales Goals
6. Create Sales Projections

1. List All Your Product and Services

To be able to create a sales projection, you need to know what products and services you will be selling. This includes not only the items that are currently in your inventory but also those that you are planning to launch in the designated period.

To that end, you need to have a clear idea of future launches, including when they are going to take place, how, what resources you'll be needing, etc.

If you don't have this information yet, you can create a projection with your current data. Then, update it once your new projects are in a more advanced stage of their development.

Furthermore, you can consider creating an overall sales projection for your whole inventory, and stand-alone projections for each separate product. This way you will have a better idea of how each of your solutions contributes to your overall growth and revenue plans. It also helps distribute resources accordingly.

2. Take into Account All Sales Channels

When building sales projections, make sure to take into account all sales channels, as well as potential opportunities that you are planning to explore.

For example, you may rely mostly on brick and mortar sales now, but if there's a digital transformation in the cards, this will completely change the numbers and will add new variables to the equation.

Digital sales open doors to new markets – local and international – and require different types of resources. Furthermore, they are supported by completely different marketing

channels. As a result, you may have to create new departments, hire new people, invest in new tools, etc.

How well you implement the transformation will define how successful your future sales will be.

Also, if you prosper in your digital efforts, you may want to reconsider your whole approach toward brick-and-mortar sales and change over to a whole new business model.

Speaking of business models, by analyzing the state of the market and the specifics of your product, you may come up with new ways to profit from your products and services. Planning for those, or at least considering them in your sales projections can be very beneficial.

3. Analyze Current and Historical Data

While projected future sales represent a fictional number, to serve their purpose properly and be reliable, they still need to be based on real data.

The relevant information that you need to take into account depends entirely on your business structure and organization. However, some of the areas to consider include:

- Number of employees
- Number of salespeople
- Sales reps quotas
- Sales reps results on a monthly and annual basis
- Sales channels performance
- Sales per channel
- Revenue per channel
- Market demand
- Demand creation success rates

- CRM data
- Sales and marketing budget
- Sales and marketing expenses
- Sales and marketing ROI
- Current revenue vs. desired revenue

By analyzing this information, you can obtain valuable insights into the unique sales ecosystem of your company and how changes affect the figures.

However, if you are a startup, you probably don't have much data to work with. In this case, you should consider looking into industry resources, market research, and whatever competitor intel you have access to.

Based on this information and your own product development plans, you can create a preliminary sales projection. Once you have enough data of your own, you can update the results and change course accordingly.

4. Collect and Review PESTEL Data

Aside from the efforts and planning within your organization, sales strongly depend on outside factors.

While you can't always predict those accurately, conducting or obtaining access to PESTEL analysis will provide you with valuable information on the current state of the market and economy, potential consumer trends that you need to consider, and other relevant factors.

The acronym PESTEL stands for Political, Economic, Sociological, Technological, Environmental, and Legal.

These elements define the success of any business endeavor, and, in order to see how your desired sales objectives may or may not develop in the future, you need to examine how they can affect your success.

5. Set Sales Goals

Your next step is to consider sales goals.

Whenever launching a new product or planning the further development of an existing one, you need to set realistic targets to pursue. This will help you estimate future revenues, plan resources, and set KPIs.

The best way to do this is to use the SMART goals framework, and ensure that your goals are: Specific, Measurable, Achievable, Realistic, and Time-Bound.

In other words, sales goals are not just random numbers and wishful thinking. They need to make sense and be something that affects your actions.

This way, you can use them to monitor performance, identify weak points in your strategy, and make improvements.

However, keep in mind that sales goals are not just the number of sales you want to make, they also include:

- **Sales team**: strategy, number of reps, efficiency, quotas, performance, learning and development, remuneration, overall development of the team, etc.
- **Clients**: types of clients, quality vs. quantity, customer acquisition costs, sales cycle length, retention strategies.
- **Demand**: marketing and advertising efforts, demand creation, demand generation, customer education.

6. Create Sales Projections

With all this information in place, you can build your sales projections.

As mentioned, you should consider creating an overall projection for all the sales your company wishes to achieve in a designated period. Also, it’s worth building individual

projections for current products and services, and any new solutions you are planning to build and market.

Generally, you can approach the sales projection in two ways – **ground-up** or **top-down**.

In the **ground-up** method, you analyze the information you have gathered and project an achievable number with the resources at your disposal, and the conditions that apply.

In the **top-down** approach, you choose a number and start to reverse engineer the resources and conditions required to achieve it. Once you do the calculations and see how realistic the number is, you can adjust it and/or the other factors necessary to create a realistic roadmap.

Regardless of which method you choose, you should consider visualizing the projections and making different versions, with different variables, to illustrate how each of them can affect and contribute to the final outcome.

These infographics will allow you to easily present the data to stakeholders, and will, in turn, help them to make more informed decisions

2.Predict Future Sales using Machine Learning

All industries aim to manufacture just the right number of products at the right time, but for retailers this issue is particularly critical as they also need to manage perishable inventory efficiently.

Too many items and too few items are both scenarios that are bad for business. (Estimates suggest that poor inventory management costs US retailers close to two billion dollars per year.)

Business Problem

Demand forecasting is the process of predicting what the demand for certain products will be in the future. This helps manufacturers to decide what they should produce and guides retailers toward what they should stock.

Demand forecasting is aimed at improving the following processes:

- Supplier relationship management
- Customer relationship management
- Order fulfillment and logistics
- Marketing campaigns
- Manufacturing flow management

Use of Machine Learning

Machine learning techniques allows for predicting the amount of products/services to be purchased during a defined future period. In this case, a software system can learn from data for improved analysis. Compared to traditional demand forecasting methods, a machine learning approach allows you to:

- Accelerate data processing speed
- Provide a more accurate forecast
- Automate forecast updates based on the recent data
- Analyze more data
- Identify hidden patterns in data
- Create a robust system
- Increase adaptability to changes

reference: If you want to learn above topics in more detail, here's a blog by ***Liudmyla Taranenko*** from where it is abstracted.

Data Gathering

Dataset is taken from kaggle competition and can be downloaded from here.

Data Description:

You are provided with daily historical sales data. The task is to forecast the total amount of products sold in every shop for the test set. Note that the list of shops and products slightly changes every month. Creating a robust model that can handle such situations is part of the challenge.

File descriptions

- sales_train.csv — the training set. Daily historical data from January 2013 to October 2015.
- test.csv — the test set. You need to forecast the sales for these shops and products for November 2015.

- sample_submission.csv — a sample submission file in the correct format.
- items.csv — supplemental information about the items/products.
- item_categories.csv — supplemental information about the items categories.
- shops.csv- supplemental information about the shops.

Data fields

- ID — an Id that represents a (Shop, Item) tuple within the test set
- shop_id — unique identifier of a shop
- item_id — unique identifier of a product
- item_category_id — unique identifier of item category
- item_cnt_day — number of products sold. You are predicting a monthly amount of this measure
- item_price — current price of an item
- date — date in format dd/mm/yyyy
- date_block_num — a consecutive month number, used for convenience. January 2013 is 0, February 2013 is 1,..., October 2015 is 33
- item_name — name of item
- shop_name — name of shop
- item_category_name — name of item category

Existing Approaches

This is a blog on analytics vidhya by **De-Yu Chao** where they had approach it as a machine learning problem. They used XGBoost regressor model to predict future sales and scored 0.8865 (root mean squared error).

Do check out above link to know more about their solution, they have explained it in very detailed with code.

My Improvements

I did experiment with various ML models, used some features from **De-Yu Chao's** solution , tried playing with some quarter based features (can also look it as a moving averages feature). Best model among all (LightGBM) scored 0.8607(root mean squared error) which is slight an improvement form **De-Yu Chao's** solution (0.8865 rmse).

Exploratory Data Analysis

Data processing and feature engineering

Here I' am removing outliers and negative item price and item counts

I' am considering all the 'item prices' >50000 and 'item count a day ' >1000 as an outliers.

Here I' am converting daily historical data into monthly sales data .Also extracting ('item_cnt_month', 'mean_item_cnt', 'mean_item_price', 'revenue_month') features using aggregation function.

In this code snippet I' am building a data set with all the possible combinations of ['date_block_num', 'shop_id', 'item_id'] so we won't have missing records.

Here I' am extracting various mean encoding features of shop, item , item category, date block .

This is code snippet is a function to compute lag features

Here I' am extracting quarter based features such as min, max, mean , std. This can also be consider as a moving average features.

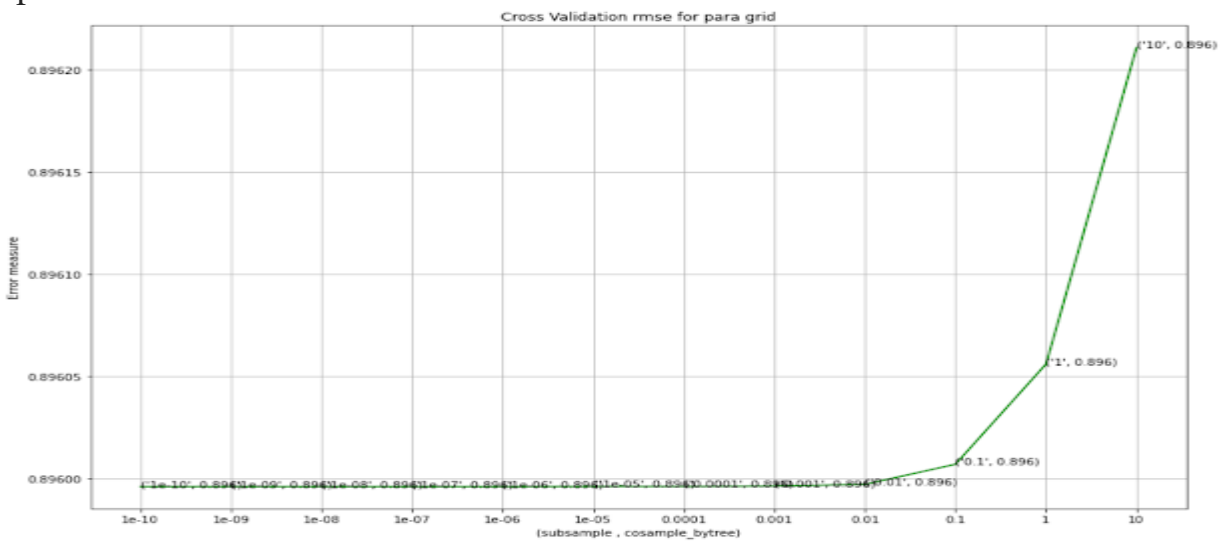
Machine Learning

We are done with feature extraction and data preprocessing. In this section I' am training different models , starting from very basic models like lasso regression to more complex models like XGBoost and LightGBM.

1. Lasso Regression

Here I' am tuning "alpha" parameter

Alpha vs score

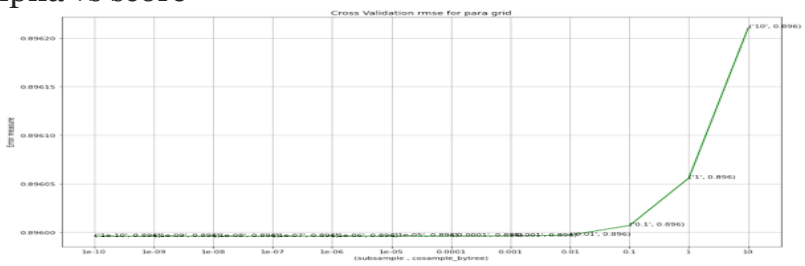


Lasso regression Best score train: 0.77688rmse test:0.896rmse.

2. Ridge Regression

Here I' am tuning "alpha " parameter

Alpha vs score

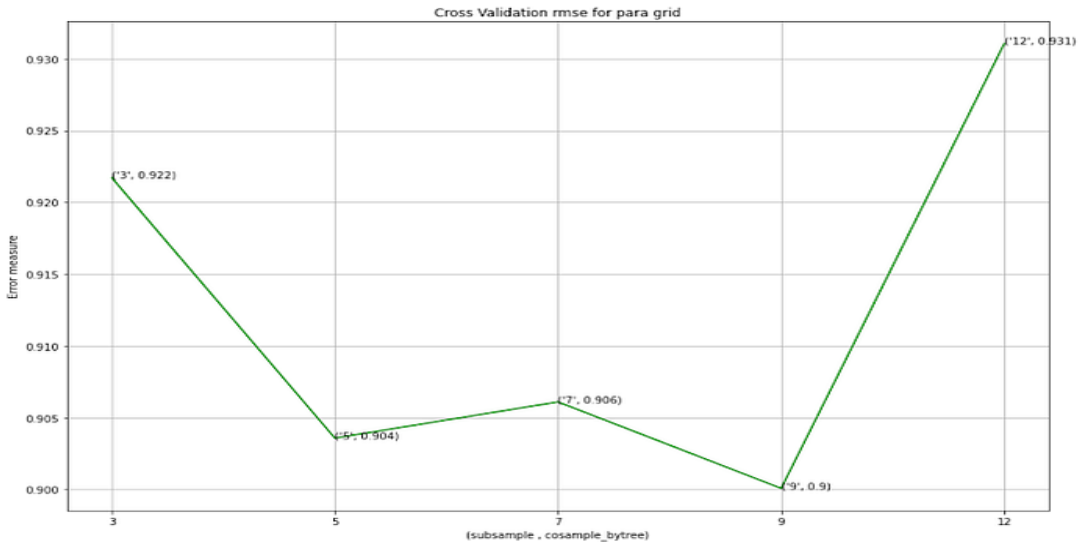


Ridge Regression best score train:0.77689 rmse test: 0.896rmse.

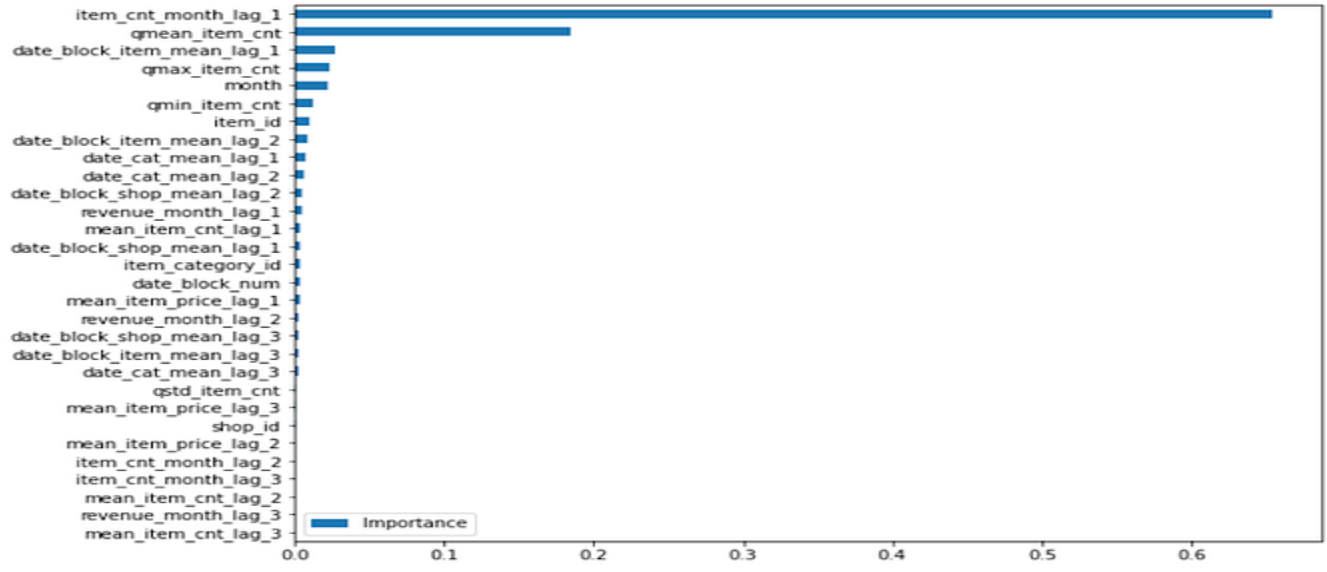
3. Dicision Tree

Here I' am tuning "max_depth " parameter

Here's a visualization max_depth vs score



Here's feature importance graph of decision tree

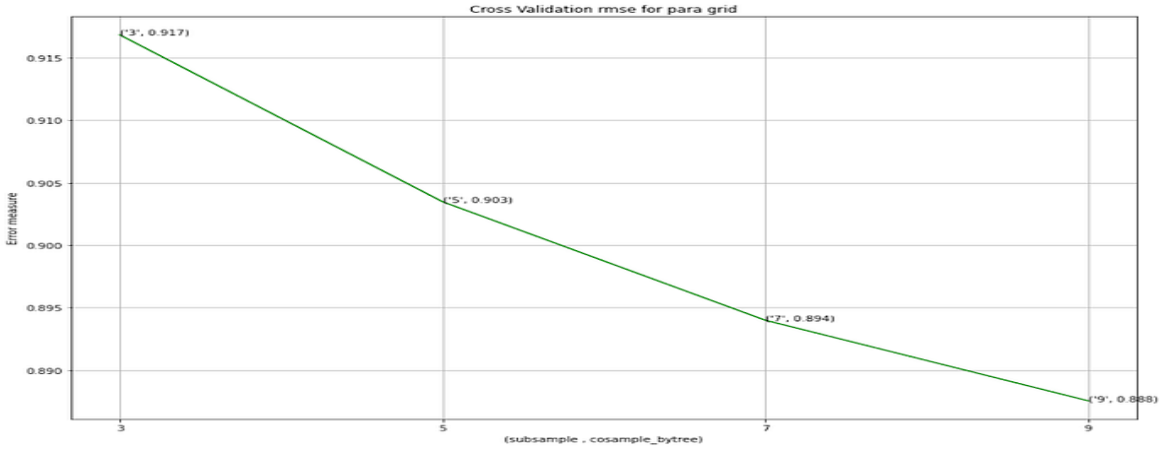


Decision tree Regressor best score train:0.74485rmse test:0.90008rmse.

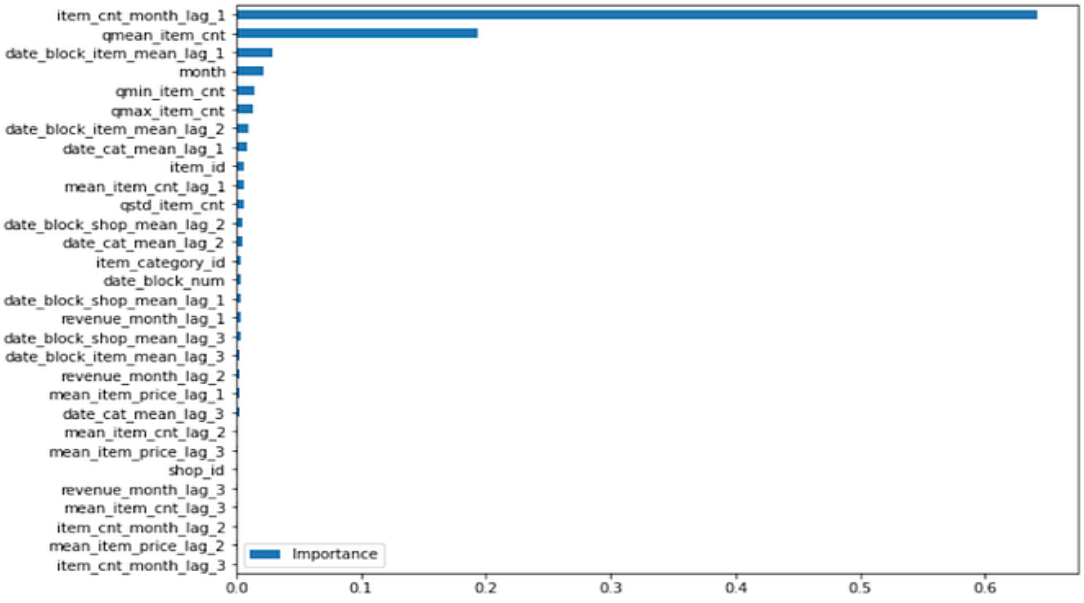
4. Random Forest

Here I' am tuning "max_depth " parameter

Here’s a graph showing of parameter vs score



Here’s a bar graph showing feature importance of best model



Random Forest Regressor best scoretrain:0.73291rmse test:0.88755rmse.

5. XGBoost Regressor

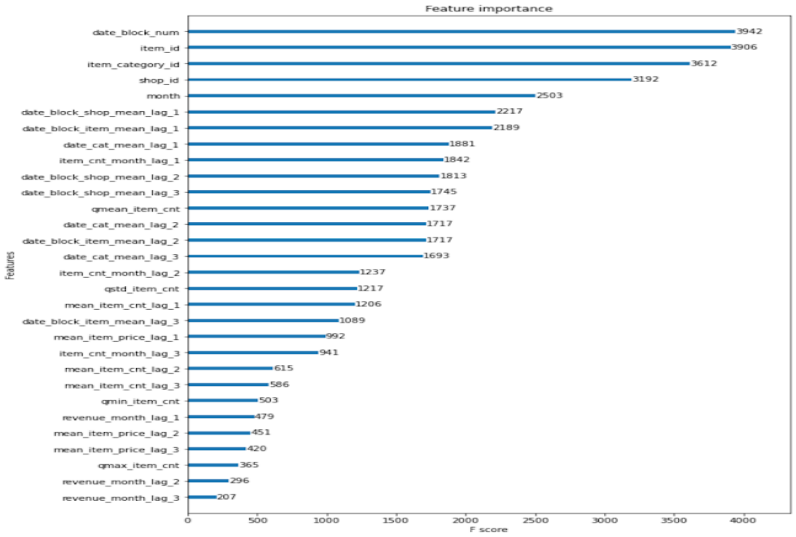
I did hyperparameter tuning of XGB in two parts
In first part I have tuned max_depth and min_child_weight parameter and in second part I have tuned subsample and cosample_bytree parameter

Part 1:

This is a custom grid search code

Part 2:

Here's bar graph showing feature importance of best scored model

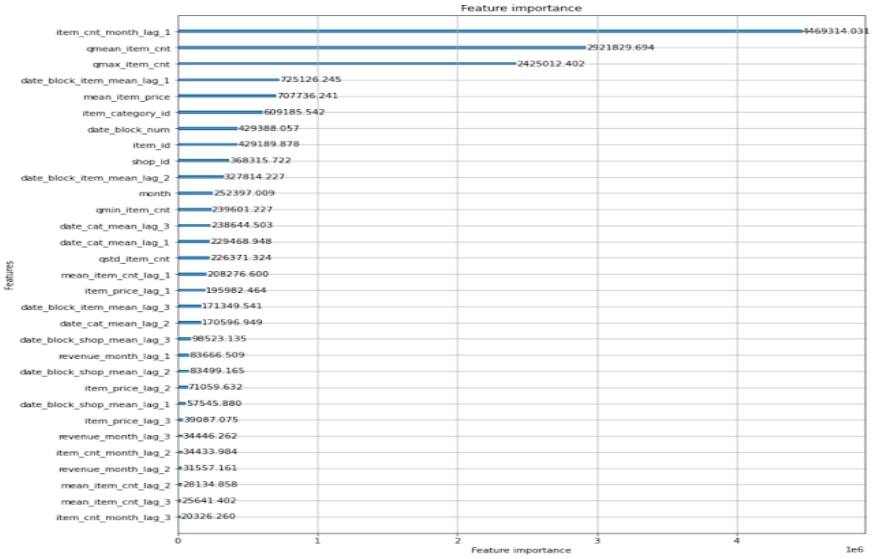


XGB Regressor part 2 best score train:0.70248 rmse test:0.87285 rmse.

6.LightGBM:

I have used LGB tuner to tune hyperparameters, you can go through this article to know more about this auto hyperparameter tuner.

Here’s bar graph showing feature importance of best scored model



LightGBM best score train:0.64502 rmse test:0.86073 rmse.

7.Ensemble Model:

At last I have trained ensemble model using sklearn’s Api. In this I have trained ensemble model using stacking method.

Here I am first loading all the saved pretrained models

Training ensemble Model using StackingRegressor Api

Ensemble Model score train:0.64698rmse test:0.87851rmse.

Comparison Of models

Model	train_rmse	val_rmse
Lasso Regression	0.77688	0.896
Ridge Regression	0.77689	0.896
Decision tree Regressor	0.74485	0.90008
Random Forest Regressor	0.73291	0.88755
XGB Regressor 1	0.68391	0.87192
XGB Regressor 2	0.70248	0.87285
LightGBM	0.64502	0.86073
Ensemble_stacking	0.64698	0.87851

Deployment

After training and experimenting with different models it is to deploy our project. For that I have used Flask API you can check out the code below.

.

3. Sales forecasting using python

It is determining present-day or future sales using data like past sales, seasonality, festivities, economic conditions, etc.

So, this model will predict sales on a certain day after being provided with a certain set of inputs.

In this model 8 parameters were used as input:

1. past seven day sales
2. day of the week
3. date – the date was transformed into 3 different inputs
4. season
5. Festival or not
6. sales on the same day in the previous year

How does it work?

First, all inputs are preprocessed to be understandable by the machine. This is a linear regression model based on supervised learning, so the output will be provided along with the input. Then inputs are then fed to the model along with desired output. The model will plot(learn) a relation(function) between the input and output. This function or relation is then used to predict the output for a specific set of inputs. In this case, input parameters like date and previous sales are labeled as input, and the amount of sales is marked as output. The model will predict a number between 0 and 1 as a sigmoid function is used in the last layer. This output can be multiplied by a specific number(in this case, maximum sales), this will be our corresponding sales amount for

a certain day. This output is then provided as input to calculate sales data for the next day. This cycle of steps will be continued until a certain date arrives.

Required packages and Installation

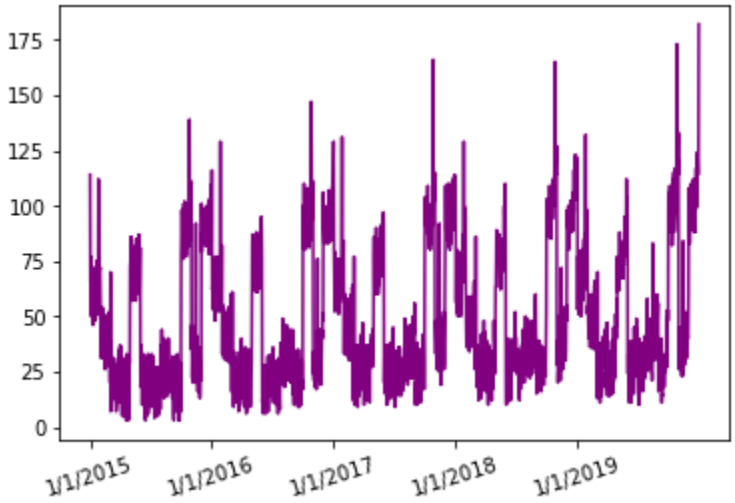
1. numpy
2. pandas
3. keras
4. tensorflow
5. csv
6. matplotlib.pyplot

- Python3

```
import pandas as pd                        # to extract data
from dataset(.csv file)

import csv                                 #used to read and
write to csv files

import numpy as np                         #used to convert
input into numpy arrays to be fed to the model

import matplotlib.pyplot as plt            #to plot/visualize
sales data and sales forecasting

import tensorflow as tf                    # acts as the
framework upon which this model is built

from tensorflow import keras              #defines layers and
functions in the model



#here the csv file has been copied into three lists to
allow better availability

list_row,date,traffic =
get_data('/home/abh/Documents/Python/Untitled
Folder/Sales_dataset')
```

The use of external libraries has been kept to a minimum to provide a simpler interface, you can replace the functions used in this tutorial with those already existing in established libraries.

Original data set for sales data for 5 years:



Sales data from Jan 2015 to Dec 2019

As you can see, the sales data seems to be following a similar kind of pattern for each year and the peak sales value seems to be increasing with time over the 5-year time frame.

In this 5-year time frame, the first 4 years will be used to train the model and the last year will be used as a test set.

Now, a few helper functions were used for processing the dataset and creating inputs of the required shape and size. They are as follows:

1. get_data – used to load the data set using a path to its location.
2. date_to_day – provides a day to each day. For example — 2/2/16 is Saturday and 9/5/15 is Monday.
3. date_to_enc – Encodes data into one-hot vectors, this provides a better learning opportunity for the model.

All the properties of these functions and a few other functions cannot be explained here as it would take too much time. Please visit this link if you want to look at the entire code.

Preprocessing:

Initially, the data set had only two columns: date and traffic(sales).

After the addition of different columns and processing/normalization of values, the data contained all these values.

1. Date
2. Traffic
3. Holiday or not
4. Day

All these parameters have to be converted into a form that the machine can understand, which will be done using this function below.

Instead of keeping date, month, and year as a single entity, it was broken into three different inputs. The reason is that the year parameter in these inputs will be the same most of the time, this will cause the model to become complacent i.e it will begin to overfit to the current dataset. To increase the variability between different various inputs dates, days and months were labeled separately. The following function conversion() will create six lists and append appropriate input to them. This is how years 2015 to 2019 will look as an encoding: is

{2015: array([1., 0., 0., 0., 0.], dtype=float32), 2016: array([0., 1., 0., 0., 0.], dtype=float32), 2017: array([0., 0., 1., 0., 0.], dtype=float32), 2018: array([0., 0., 0., 1., 0.], dtype=float32), 2019: array([0., 0., 0., 0., 1.], dtype=float32)

Each of them is a NumPy array of length 5 with 1s and 0s denoting its value

- Python3

```
def conversion(week,days,months,years,list_row):

  #lists have been defined to hold different inputs

  inp_day = []

  inp_mon = []

  inp_year = []

  inp_week=[]

  inp_hol=[]

  out = []

  #converts the days of a week(monday,sunday,etc.) into one
hot vectors and stores them as a dictionary

  week1 = number_to_one_hot(week)

  #list_row contains primary inputs

  for row in list_row:

        #Filter out date from list_row

        d = row[0]

        #the date was split into three values date, month
and year.

        d_split=d.split('/')

        if d_split[2]==str(year_all[0]):

          #prevents use of the first year data to ensure
each input contains previous year data as well.
```

```
          continue

        #encode the three parameters of date into one hot
vectors using date_to_enc function.

        d1,m1,y1 = date_to_enc(d,days,months,years) #days,
months and years and dictionaries containing the one hot
encoding of each date,month and year.

        inp_day.append(d1) #append date into date input

        inp_mon.append(m1) #append month into month input

        inp_year.append(y1) #append year into year input

        week2 = week1[row[3]] #the day column from list_is
converted into its one-hot representation and saved into
week2 variable

        inp_week.append(week2)# it is now appended into
week input.

        inp_hol.append([row[2]])#specifies whether the day
is a holiday or not

        t1 = row[1] #row[1] contains the traffic/sales value
for a specific date

        out.append(t1) #append t1(traffic value) into a
list out

  return inp_day,inp_mon,inp_year,inp_week,inp_hol,out #all
the processed inputs are returned



inp_day,inp_mon,inp_year,inp_week,inp_hol,out =
conversion(week,days,months,years,list_train)

#all of the inputs must be converted into numpy arrays to
be fed into the model

inp_day = np.array(inp_day)
```

```
inp_mon = np.array(inp_mon)

inp_year = np.array(inp_year)

inp_week = np.array(inp_week)

inp_hol = np.array(inp_hol)
```

We will now process some other inputs that were remaining, the reason behind using all these parameters is to increase the efficiency of the model, you can experiment with removing or adding some inputs.
Sales data of the past seven days were passed as an input to create a trend in sales data, this will the predicted value will not be completely random similarly, sales data of the same day in the previous year was also provided.

The following function(other_inputs) processes three inputs:

- sales data of past seven days
- sales data on the same date in the previous year
- seasonality – seasonality was added to mark trends like summer sales, etc.

- Python3

```
def other_inputs(season,list_row):

  #lists to hold all the inputs

  inp7=[]

  inp_prev=[]

  inp_sess=[]

  count=0 #count variable will be used to keep track of the
index of current row in order to access the traffic values
of past seven days.

  for row in list_row:

    ind = count

    count=count+1

    d = row[0] #date was copied to variable d
```

```
    d_split=d.split('/')

    if d_split[2]==str(year_all[0]):

      #preventing use of the first year in the data

      continue

    sess = cur_season(season,d) #assigning a season to the
current date

    inp_sess.append(sess) #appending sess variable to an
input list

    t7=[] #temporary list to hold seven sales value

    t_prev=[] #temporary list to hold the previous year
sales value

    t_prev.append(list_row[ind-365][1]) #accessing the
sales value from one year back and appending them

    for j in range(0,7):

        t7.append(list_row[ind-j-1][1]) #appending the last
seven days sales value

    inp7.append(t7)

    inp_prev.append(t_prev)

  return inp7,inp_prev,inp_sess



inp7,inp_prev,inp_sess = other_inputs(season,list_train)

inp7 = np.array(inp7)

inp7= inp7.reshape(inp7.shape[0],inp7.shape[1],1)

inp_prev = np.array(inp_prev)
```

```
inp_sess = np.array(inp_sess)
```

The reason behind so many inputs is that if all of these were combined into a single array, it would have different rows or columns of different lengths. Such an array cannot be fed as an input.

Linearly arranging all the values in a single array lead to the model having a high loss.

A linear arrangement will cause the model to generalize, as the difference between successive inputs would not be too different, which will lead to limited learning, decreasing the accuracy of the model.

Defining the Model

Eight separate inputs are processed and concatenated into a single layer and passed to the model.

The finalized inputs are as follows:

1. Date
2. Month
3. Year
4. Day
5. Previous seven days sales
6. sales in the previous year
7. Season
8. Holiday or not

Here in most layers, I have used 5 units as the output shape, you can further experiment with it to increase the efficiency of the model.

- Python

```
from tensorflow.keras.models import Model

from tensorflow.keras.layers import Input, Dense,LSTM,Flatten

from tensorflow.keras.layers import concatenate

#an Input variable is made from every input array

input_day = Input(shape=(inp_day.shape[1],),name = 'input_day')

input_mon = Input(shape=(inp_mon.shape[1],),name = 'input_mon')

input_year = Input(shape=(inp_year.shape[1],),name = 'input_year')

input_week = Input(shape=(inp_week.shape[1],),name = 'input_week')

input_hol = Input(shape=(inp_hol.shape[1],),name = 'input_hol')

input_day7 = Input(shape=(inp7.shape[1],inp7.shape[2]),name = 'input_day7')
```

```
input_day_prev = Input(shape=(inp_prev.shape[1],),name = 'input_day_prev')

input_day_sess = Input(shape=(inp_sess.shape[1],),name = 'input_day_sess')

# The model is quite straight-forward, all inputs were inserted into a dense
layer with 5 units and 'relu' as activation function

x1 = Dense(5, activation='relu')(input_day)

x2 = Dense(5, activation='relu')(input_mon)

x3 = Dense(5, activation='relu')(input_year)

x4 = Dense(5, activation='relu')(input_week)

x5 = Dense(5, activation='relu')(input_hol)

x_6 = Dense(5, activation='relu')(input_day7)

x__6 = LSTM(5,return_sequences=True)(x_6) # LSTM is used to remember the
importance of each day from the seven days data

x6 = Flatten()(x__10) # done to make the shape compatible to other inputs as
LSTM outputs a three dimensional tensor

x7 = Dense(5, activation='relu')(input_day_prev)

x8 = Dense(5, activation='relu')(input_day_sess)

c = concatenate([x1,x2,x3,x4,x5,x6,x7,x8]) # all inputs are concatenated into
one

layer1 = Dense(64,activation='relu')(c)

outputs = Dense(1, activation='sigmoid')(layer1) # a single output is produced
with value ranging between 0-1.

# now the model is initialized and created as well

model =
Model(inputs=[input_day,input_mon,input_year,input_week,input_hol,input_day7,
input_day_prev,input_day_sess], outputs=outputs)
```

```
model.summary() # used to draw a summary(diagram) of the model
```

Model Summary:

```
Model: "model_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_day7 (InputLayer)         [(None, 7, 1)]       0
__________________________________________________________________________________________________
dense_15 (Dense)                (None, 7, 5)         10          input_day7[0][0]
__________________________________________________________________________________________________
input_day (InputLayer)          [(None, 31)]         0
__________________________________________________________________________________________________
input_mon (InputLayer)          [(None, 12)]         0
__________________________________________________________________________________________________
input_year (InputLayer)         [(None, 5)]          0
__________________________________________________________________________________________________
input_week (InputLayer)         [(None, 7)]          0
__________________________________________________________________________________________________
input_hol (InputLayer)          [(None, 1)]          0
__________________________________________________________________________________________________
lstm_1 (LSTM)                   (None, 7, 5)         220         dense_15[0][0]
__________________________________________________________________________________________________
input_day_prev (InputLayer)     [(None, 1)]          0
__________________________________________________________________________________________________
input_day_sess (InputLayer)     [(None, 5)]          0
__________________________________________________________________________________________________
dense_10 (Dense)                (None, 5)            160         input_day[0][0]
__________________________________________________________________________________________________
dense_11 (Dense)                (None, 5)            65          input_mon[0][0]
__________________________________________________________________________________________________
dense_12 (Dense)                (None, 5)            30          input_year[0][0]
__________________________________________________________________________________________________
dense_13 (Dense)                (None, 5)            40          input_week[0][0]
__________________________________________________________________________________________________
dense_14 (Dense)                (None, 5)            10          input_hol[0][0]
__________________________________________________________________________________________________
flatten_1 (Flatten)             (None, 35)           0           lstm_1[0][0]
__________________________________________________________________________________________________
dense_16 (Dense)                (None, 5)            10          input_day_prev[0][0]
__________________________________________________________________________________________________
dense_17 (Dense)                (None, 5)            30          input_day_sess[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 70)           0           dense_10[0][0]
                                                                 dense_11[0][0]
                                                                 dense_12[0][0]
                                                                 dense_13[0][0]
                                                                 dense_14[0][0]
                                                                 flatten_1[0][0]
                                                                 dense_16[0][0]
                                                                 dense_17[0][0]
__________________________________________________________________________________________________
dense_18 (Dense)                (None, 64)           4544        concatenate_1[0][0]
__________________________________________________________________________________________________
dense_19 (Dense)                (None, 1)            65          dense_18[0][0]
==================================================================================================
Total params: 5,184
Trainable params: 5,184
Non-trainable params: 0
```

Compiling the model using RMSprop:

RMSprop is great at dealing with random distributions, hence its use here.

- Python3

```
from tensorflow.keras.optimizers import RMSprop



model.compile(loss=['mean_squared_error'],

              optimizer = 'adam',

              metrics = ['acc'] #while accuracy is used as a
metrics here it will remain zero as this is no
classification model

              )                  # linear regression models
are best gauged by their loss value
```

Fitting the model on the dataset:
The model will now be fed with the input and output data, this is the final step and now our model will be able to predict sales data.

- Python3

```
history = model.fit(

            x =
[inp_day,inp_mon,inp_year,inp_week,inp_hol,inp7,inp_prev,inp_sess],

            y = out,

            batch_size=16,

            steps_per_epoch=50,

            epochs = 15,

            verbose=1,

            shuffle =False

            )

#all the inputs were fed into the model and the training was
```

```
completed
```

Output:

```
Epoch 1/15
50/50 [==============================] - 6s 15ms/step - loss: 0.0612 - acc: 0.0000e+00
Epoch 2/15
50/50 [==============================] - 1s 18ms/step - loss: 0.0288 - acc: 0.0000e+00
Epoch 3/15
50/50 [==============================] - 1s 20ms/step - loss: 0.0172 - acc: 0.0000e+00
Epoch 4/15
50/50 [==============================] - 1s 15ms/step - loss: 0.0099 - acc: 0.0000e+00
Epoch 5/15
50/50 [==============================] - 1s 17ms/step - loss: 0.0084 - acc: 0.0000e+00
Epoch 6/15
50/50 [==============================] - 1s 18ms/step - loss: 0.0065 - acc: 0.0000e+00
Epoch 7/15
50/50 [==============================] - 1s 16ms/step - loss: 0.0053 - acc: 0.0000e+00
Epoch 8/15
50/50 [==============================] - 1s 18ms/step - loss: 0.0053 - acc: 0.0000e+00
Epoch 9/15
50/50 [==============================] - 1s 17ms/step - loss: 0.0038 - acc: 0.0000e+00
Epoch 10/15
50/50 [==============================] - 1s 15ms/step - loss: 0.0039 - acc: 0.0000e+00
Epoch 11/15
50/50 [==============================] - 1s 17ms/step - loss: 0.0037 - acc: 0.0000e+00
Epoch 12/15
50/50 [==============================] - 1s 17ms/step - loss: 0.0036 - acc: 0.0000e+00
Epoch 13/15
50/50 [==============================] - 1s 17ms/step - loss: 0.0035 - acc: 0.0000e+00
Epoch 14/15
50/50 [==============================] - 1s 17ms/step - loss: 0.0032 - acc: 0.0000e+00
Epoch 15/15
50/50 [==============================] - 1s 18ms/step - loss: 0.0029 - acc: 0.0000e+00
```

Now, to test the model, input() takes input and transform it into the appropriate form:

- Python3

```
def input(date):

    d1,d2,d3 = date_to_enc(date,days,months,years)
#separate date into three parameters

    print('date=',date)

    d1 = np.array([d1])

    d2 = np.array([d2])

    d3 = np.array([d3])

    week1 = number_to_one_hot(week)         #defining one hot
vector to encode days of a week
```

```
    week2 = week1[day[date]]

    week2=np.array([week2])

    //appeding a column for holiday(0-not holiday, 1-
holiday)

    if date in holiday:

        h=1

        #print('holiday')

    else:

        h=0

        #print("no holiday")

    h = np.array([h])

    sess = cur_season(season,date)        #getting
seasonality data from cur_season function

    sess = np.array([sess])

    return d1,d2,d3,week2,h,sess
```

Predicting sales data is not what we are here for right, so let's get on with the forecasting job.

Sales Forecasting

Defining *forecast_testing* function to forecast the sales data from one year back from provided date:
This function works as follows:

- A date is required as input to forecast the sales data from one year back till the mentioned date
- Then, we access the previous year's sales data on the same day and sales data of 7 days before it.
- Then, using these as input a new value is predicted, then in the seven days value the first day is removed and the predicted output is added as input for the next prediction

For eg: we require forecasting of one year till 31/12/2019

- First, the date of 31/12/2018 (one year back) is recorded, and also seven-day sales from (25/12/2018 – 31/12/2018)
- Then the sales data of one year back i.e 31/12/2017 is collected
- Using these as inputs with other ones, the first sales data(i.e 1/1/2019) is predicted
- Then 24/12/2018 sales data is removed and 1/1/2019 predicted sales are added. This cycle is repeated until the sales data for 31/12/2019 is predicted.

So, previous outputs are used as inputs.

- Python3

```
def forecast_testing(date):

    maxj = max(traffic) # determines the maximum sales value
in order to normalize or return the data to its original
form

    out=[]

    count=-1

    ind=0

    for i in list_row:

        count =count+1

        if i[0]==date: #identify the index of the data in
list

            ind = count

    t7=[]

    t_prev=[]

    t_prev.append(list_row[ind-365][1]) #previous year data

    # for the first input, sales data of last seven days
will be taken from training data

    for j in range(0,7):

          t7.append(list_row[ind-j-365][1])

    result=[] # list to store the output and values

    count=0

    for i in list_date[ind-364:ind+2]:

        d1,d2,d3,week2,h,sess = input(i) # using input
```

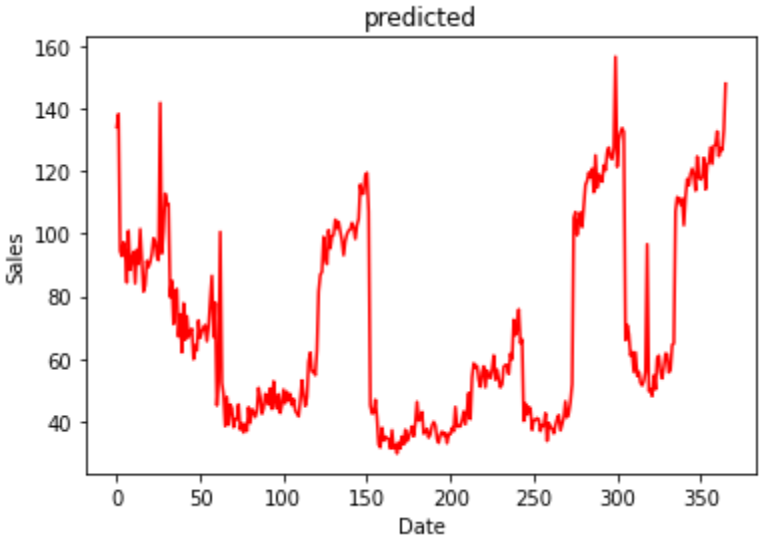
```
function to process input values into numpy arrays

        t_7 = np.array([t7]) # converting the data into a
numpy array

        t_7 = t_7.reshape(1,7,1)

        # extracting and processing the previous year sales
value

        t_prev=[]

        t_prev.append(list_row[ind-730+count][1])

        t_prev = np.array([t_prev])

        #predicting value for output

        y_out =
model.predict([d1,d2,d3,week2,h,t_7,t_prev,sess])

        #output and multiply the max value to the output
value to increase its range from 0-1

        print(y_out[0][0]*maxj)

        t7.pop(0) #delete the first value from the last
seven days value

        t7.append(y_out[0][0]) # append the output as input
for the seven days data

        result.append(y_out[0][0]*maxj) # append the output
value to the result list

        count=count+1

    return result
```

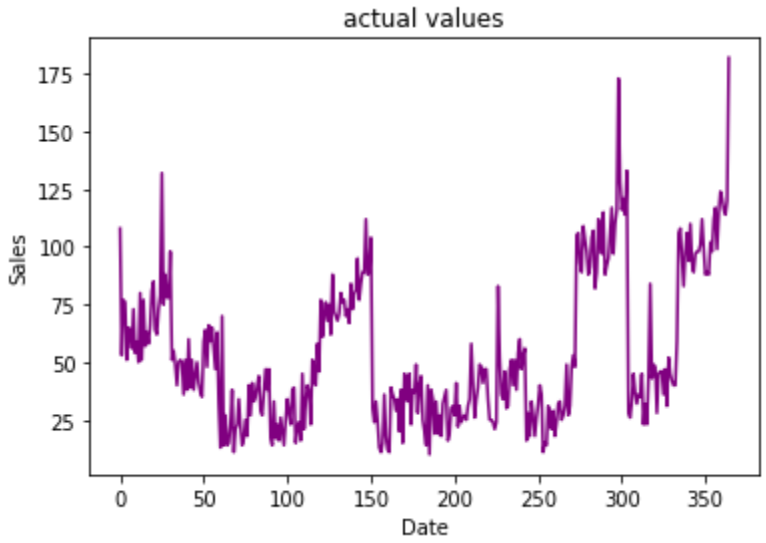
Run the forecast test function and a list containing all the sales data for that one year are returned

Result = forecast_testing('31/12/2019′, date)
Graphs for both the forecast and actual values to test the performance of the model

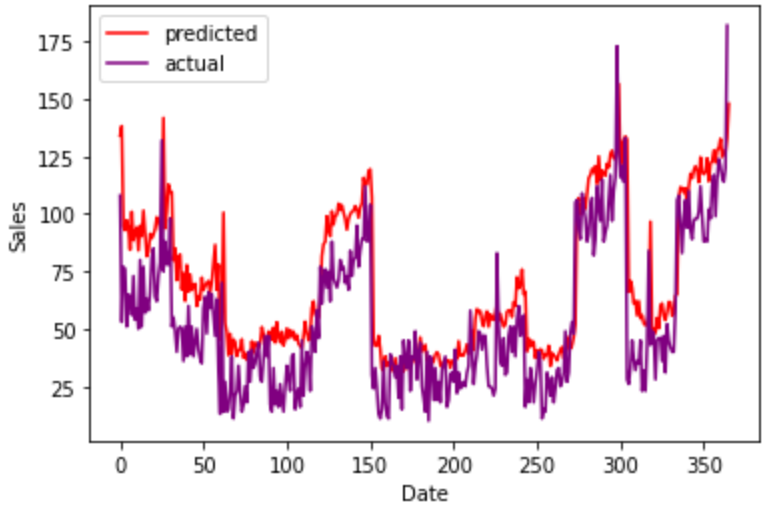
- Python3

```
plt.plot(result,color='red',label='predicted')

plt.plot(test_sales,color='purple',label="actual")

plt.xlabel("Date")

plt.ylabel("Sales")

leg =plt.legend()

plt.show()
```





Actual Values from 1/1/2019 to 31/12/2019



Comparison between prediction and actual values

As you can see, the predicted and actual values are quite close to each other, this proves the efficiency of our model. If there are any errors or possibilities of improvements in the above article, please feel free to mention them in the comment section.

Advantages:

1. **Data-Driven Decision Making:** Sales predictions are based on historical data and advanced analytics, allowing for more informed and data-driven decision-making. This can lead to better resource allocation and strategic planning.

2. **Optimized Inventory Management:** Accurate sales predictions help in managing inventory efficiently, reducing the risk of overstocking or understocking, which can save costs and improve customer satisfaction.
3. **Cost Reduction:** By better understanding future sales trends, businesses can optimize their operations, reduce waste, and lower overall costs.
4. **Improved Customer Satisfaction:** Anticipating customer demand and having products or services readily available can enhance customer satisfaction and loyalty.
5. **Market Trend Analysis:** Sales prediction models can identify emerging market trends and consumer preferences, enabling businesses to adapt and stay competitive.
6. **Risk Mitigation:** By identifying potential risks and challenges in advance, businesses can develop contingency plans and be better prepared for economic downturns or unexpected disruptions.

Disadvantages:

1. **Data Quality:** Sales prediction models heavily rely on the quality and quantity of historical data. Inaccurate or incomplete data can lead to unreliable forecasts.
2. **Complexity:** Developing and maintaining a sales prediction model can be complex and resource-intensive. It may require skilled data scientists and significant computational resources.
3. **Model Uncertainty:** Even the most advanced models may not account for all variables and uncertainties in the market, making predictions inherently imperfect.
4. **Changing Market Conditions:** Markets can change rapidly due to unforeseen events (e.g., pandemics, economic crises), making it challenging for models to adapt quickly.
5. **Costs:** Implementing and maintaining a sales prediction system can be expensive, including the costs associated with software, hardware, data collection, and staff training.
6. **Human Expertise:** While AI and machine learning can aid in prediction, they cannot replace human expertise and intuition. There's a need for a balance between automated predictions and human judgment.

7. **Privacy and Ethical Concerns:** Using customer data for sales prediction may raise privacy and ethical concerns. Businesses need to handle customer data responsibly and in compliance with regulations.

BENEFITS FOR FUTURE SALES PREDICTION

Improved Resource Allocation: Sales predictions help businesses allocate resources more efficiently. By anticipating sales trends, companies can better plan their production, staffing, and inventory, reducing waste and saving costs.

Optimized Inventory Management: Accurate sales predictions minimize the risk of overstocking or understocking products. This leads to reduced carrying costs, improved cash flow, and enhanced customer satisfaction due to product availability.

Enhanced Customer Satisfaction: Meeting customer demand on time is crucial for customer satisfaction. Sales prediction ensures products or services are available when needed, which can lead to higher customer retention and loyalty.

Better Marketing Strategies: Sales predictions can identify market trends and changing customer preferences. This information allows businesses to tailor their marketing strategies and product offerings more effectively.

Data-Driven Decision-Making: Sales prediction is based on historical data and advanced analytics, enabling data-driven decision-making. It reduces guesswork and subjectivity in strategic planning.

Risk Mitigation: By identifying potential challenges and risks, businesses can develop contingency plans to mitigate the impact of economic downturns, supply chain disruptions, or other unexpected events.

Cost Reduction: Accurate predictions lead to cost reduction through efficient operations, reduced waste, and optimized resource allocation.

Competitive Advantage: Businesses that can forecast sales trends have a competitive advantage. They can react to market changes more quickly and stay ahead of competitors.

Market Trend Analysis: Sales prediction models can help identify emerging market trends and consumer behaviors, allowing businesses to adapt and innovate accordingly.

Improved Financial Planning: Predictable sales figures make financial planning and budgeting more precise. This leads to better financial management and stability.

Strategic Planning: Sales predictions provide valuable insights into the future, helping businesses formulate long-term strategies and goals.

Customer Segmentation: Businesses can use sales predictions to segment their customer base and personalize their marketing and sales strategies for different customer groups.

Product Development: Predicting future sales can influence product development and help businesses invest in products or services with a higher likelihood of success.

Supply Chain Efficiency: Accurate sales predictions enable better supply chain management, reducing lead times, and ensuring timely deliveries.

Sustainable Growth: With insights from sales predictions, businesses can achieve sustainable and controlled growth, avoiding overexpansion or underutilization of resources.

In summary, future sales prediction provides a multitude of advantages that enable businesses to operate more efficiently, serve customers better, and make informed decisions. It enhances profitability, reduces risks, and positions organizations for long-term success in a competitive marketplace.

Conclusion:

In an ever-changing business environment, the ability to predict future sales is an invaluable asset for organizations of all sizes and industries. The dynamic interplay of market trends, consumer preferences, and external factors necessitates a data-driven approach to sales prediction. Leveraging advanced analytics, machine learning, and AI technologies can help businesses stay ahead of the competition and make informed decisions.

A robust sales prediction model can enable businesses to optimize their inventory management, allocate resources efficiently, and adapt to market changes proactively. It provides a strategic advantage by ensuring that products and services are available when and where they are needed, ultimately leading to increased customer satisfaction and enhanced profitability.

The journey to effective sales prediction involves data collection, analysis, model development, and continuous refinement. It is an ongoing process that demands a commitment to learning from past performance and adapting to future challenges.

In conclusion, sales prediction is not a luxury but a necessity for businesses looking to thrive in a competitive landscape. As technology continues to evolve and data becomes more abundant, those organizations that harness the power of accurate sales forecasting will be better equipped to navigate the uncertain road ahead and achieve sustained success.