SALES PREDICTION ON THE ROSSMANN STORES

JUNE 03 2023

BY,

kothapally Jayanth (Team Lead)

Yashwanth Goduguchintha

Avesh

Laxman Kumar Busetty

Prathamesh

PROTOTYPE SELECTION

The demand for a product or service keeps changing from time to time. No business can improve its financial performance without estimating customer demand and future sales of products/services accurately. Sales forecasting refers to the process of estimating demand for or sales of a particular product over a specific period. will show you how **machine learning** can be used to **predict sales** on a real-world business problem taken. In this project we solve everything right from scratch. So, you will get to see every phase of the project.

Problem Statement

Rossman operates over 3,000 drug stores in 7 European countries. Currently, Rossman store managers are tasked with *predicting their daily sales for up to six weeks in advance.* Store sales are influenced by many factors, including promotions, competition, school and state holidays, seasonality, and locality. With thousands of individual managers predicting sales based on their unique circumstances, the accuracy of results can be quite varied.

Business model

A potential business model for predicting drug sales for the Rossman store could involve a combination of data analytics, machine learning, and strategic decision-making. Here is a step-by-step outline of the business model:

1.Data Collection:

Gather historical sales data from Rossman stores, including drug sales, customer demographics, store attributes, promotions, holidays, and any other relevant data points. Additionally, collect external data such as weather information, economic indicators, and local events that might impact sales.

2. Data Preprocessing:

Clean and preprocess the collected data to remove any inconsistencies, missing values, or outliers. Normalize numerical data and encode categorical variables to make it suitable for analysis.

3. Feature Engineering:

Create additional features that could potentially impact drug sales, such as seasonality, day of the week, proximity to competitors, and special events. Feature engineering can enhance the predictive power of the model.

4. Machine Learning Model Development:

Utilize machine learning algorithms, such as regression, time series analysis, or ensemble methods, to develop a predictive model. Train the model using the historical sales data, considering drug-specific sales patterns, store-specific factors, and other relevant variables.

5.Model Evaluation and Refinement:

Evaluate the performance of the trained model using appropriate evaluation metrics (e.g., mean absolute error, root mean square error) and validate it on a holdout dataset. Fine-tune the model by experimenting with different algorithms, hyperparameters, and feature combinations to improve its accuracy and robustness.

6.Integration and Deployment:

Integrate the trained model into a software application or an API that can accept inputs such as store attributes, promotion plans, and historical data. Develop a user-friendly interface that allows users to input relevant information and obtain sales predictions for specific drugs, stores, or time periods.

7.Decision Support and Insights:

Provide actionable insights and recommendations based on the predictions generated by the model. Enable users to simulate different scenarios, such as the impact of different promotional strategies, pricing changes, or store layout modifications, to make informed business decisions.

8.Continuous Monitoring and Model Maintenance:

Regularly monitor the model's performance and update it as new data becomes available. Incorporate feedback from users and stakeholders to refine the model and address any shortcomings or evolving business requirements.

9.Business Strategy Alignment:

Use the predictions and insights generated by the model to align business strategies, such as inventory management, pricing optimization, and resource allocation, with the expected sales demand. This alignment can help optimize operations, improve profitability, and enhance customer satisfaction.

10.Ongoing Improvement:

Continuously analyze the impact of the model's predictions on actual sales and iterate on the model to improve its accuracy and relevance over time. Stay up-to-date with industry trends, market dynamics, and technological advancements to ensure the business model remains competitive and effective.

By following this business model, Rossman store can leverage data-driven insights to make informed decisions, optimize operations, and maximize their drug sales.

Data can be downloaded from

https://www.kaggle.com/c/rossmann-store-sales/data

Files provided are

1.train.csv

2.test.csv

3.store.csv

Data fields:

- **Id -** an Id that represents a (Store, Date) duple within the test set.
- **Store -** a unique Id for each store.

● **Sales -** the turnover for any given day (this is what you are predicting).

● **Customers -** the number of customers on a given day.

● **Open -** an indicator for whether the store was open: 0 = closed, 1 = open.

● **State Holiday -** indicates a state holiday. Normally all stores, with few exceptions, are closed on state holidays. Note that all schools are closed on public holidays and weekends. a = public holiday, b = Easter holiday, c = Christmas, 0 = None.

● **School Holiday -** indicates if the (Store, Date) was affected by the closure of public schools.

● **Store Type -** differentiates between 4 different store models: a, b, c, d.

● **Assortment -** describes an assortment level: a = basic, b = extra, c = extended.

● **Competition Distance -** distance in meters to the nearest competitor store.

● **Competition Open Since [Month/Year] -** gives the approximate year and month of the time the nearest competitor was opened.

● **Promo -** indicates whether a store is running a promo on that day.

● **Promo2 -** Promo2 is a continuing and consecutive promotion for some stores: 0 = store is not participating, 1 = store is participating.

● **Promo2 Since [Year/Week] -** describes the year and calendar week when the store started participating in Promo2.

● **Promo Interval -** describes the consecutive intervals Promo2 is started, naming the months the promotion is started anew. E.g., "Feb, May, Aug, Nov" means each round starts in February, May, August, November of any given year for that store.

Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is a crucial step in the data analysis process that helps understand the data before applying any modelling techniques. It involves visually exploring and summarizing the main characteristics of the dataset. EDA includes tasks such as data visualization, data cleaning, identifying missing values or outliers, examining distributions, and assessing correlations between variables. EDA provides insights into the structure and patterns within the data, helps identify potential issues or anomalies, and guides subsequent analysis and modelling decisions. It plays a vital role in uncovering relationships, trends, and outliers, enabling data analysts to make informed decisions and generate meaningful hypotheses.

Data Processing

```
!pip install pmdarima
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting pmdarima
  Downloading pmdarima-2.0.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.manylinux_2_28_x86_64.whl (1.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.8/1.8 MB 31.8 MB/s eta 0:00:00
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (1.2.0)
Requirement already satisfied: Cython!=0.29.18,!=0.29.31,>=0.29 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (0.29.34)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (1.22.4)
Requirement already satisfied: pandas>=0.19 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (1.5.3)
Requirement already satisfied: scikit-learn>=0.22 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (1.2.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (1.10.1)
Requirement already satisfied: statsmodels>=0.13.2 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (0.13.5)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (1.26.15)
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (67.7.2)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.19->pmdarima) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.19->pmdarima) (2022.7.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.22->pmdarima) (3.1.0)
Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.10/dist-packages (from statsmodels>=0.13.2->pmdarima) (0.5.3)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from statsmodels>=0.13.2->pmdarima) (23.1)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.2->statsmodels>=0.13.2->pmdarima) (1.16.0)
Installing collected packages: pmdarima
Successfully installed pmdarima-2.0.3
```

Importing the required libraries.

```
import numpy as np
import pandas as pd
from pandas import datetime

# Data visualization
import matplotlib.pyplot as plt
import seaborn as sns # advanced vizs
import matplotlib.gridspec as gridspec
from IPython.display import display
%matplotlib inline

# Data Modeling
from statsmodels.tsa.ar_model import AR
from statsmodels.tsa.arima_model import ARMA,ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
from pmdarima import auto_arima # for determining ARIMA orders


from sklearn.linear_model import LinearRegression, Lasso
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor, GradientBoostingRegressor
from xgboost import XGBRegressor
import xgboost as xgb
from lightgbm import LGBMRegressor
import lightgbm

# Data Evaluation
from sklearn.metrics import mean_squared_error

# Statistics
from statsmodels.distributions.empirical_distribution import ECDF

# Time series analysis
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

# Warning ignore
import warnings
warnings.filterwarnings("ignore")
```

```
<ipython-input-3-45780a1d0c0b>:3: FutureWarning: The pandas.datetime class is deprecated and will be removed from pandas in a future version. Import from datetime module instead.
  from pandas import datetime
```

Data Exploration:

Data exploration is the process of investigating and understanding the data at hand before conducting any formal analysis. It involves examining the dataset's structure, contents, and relationships between variables. Through data exploration, analysts gain insights into the data's quality, patterns, and potential biases. It includes tasks such as data profiling, summarizing key statistics, visualizing distributions, identifying missing or inconsistent values, and detecting outliers. Data exploration aids in selecting appropriate analysis techniques, developing hypotheses, and identifying areas for further investigation, ultimately leading to more meaningful and accurate analysis results.

DATA EXPLORATION

```
train = pd.read_csv("/content/train.csv")
store = pd.read_csv("/content/store.csv")
test = pd.read_csv("/content/test.csv", parse_dates = True, index_col = 'Date')
```

```
train.head()
```

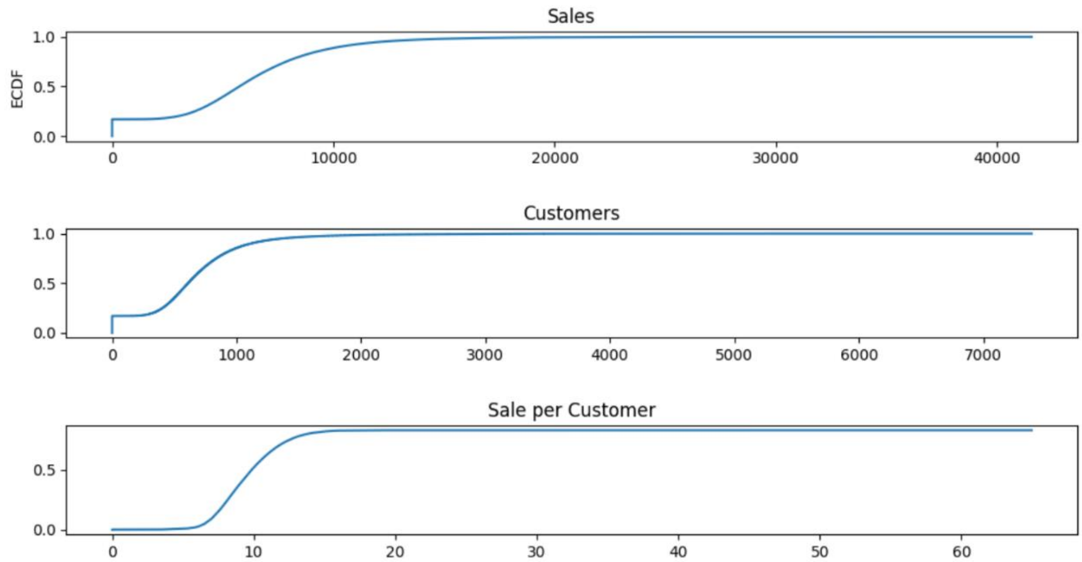
	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday
0	1	5	2015-07-31	5263	555	1	1	0	1
1	2	5	2015-07-31	6064	625	1	1	0	1
2	3	5	2015-07-31	8314	821	1	1	0	1
3	4	5	2015-07-31	13995	1498	1	1	0	1
4	5	5	2015-07-31	4822	559	1	1	0	1

```
# Change datatype of InvoiceDate as datetime type
train['Date'] = pd.to_datetime(train['Date'])
```

```
# Change datatype of InvoiceDate as datetime type
train['Date'] = pd.to_datetime(train['Date'])
```

```
# data extraction
train['Year'] = train['Date'].dt.year
train['Month'] = train['Date'].dt.month
train['Day'] = train['Date'].dt.day
train['WeekOfYear'] = train['Date'].dt.weekofyear

test['Year'] = test.index.year
test['Month'] = test.index.month
test['Day'] = test.index.day
test['WeekOfYear'] = test.index.weekofyear
```

```
plt.figure(figsize = (12, 6))

plt.subplot(311)
cdf = ECDF(train['Sales'])
plt.plot(cdf.x, cdf.y, label = "statmodels");
plt.title('Sales'); plt.ylabel('ECDF');

# plot second ECDF
plt.subplot(312)
cdf = ECDF(train['Customers'])
plt.plot(cdf.x, cdf.y, label = "statmodels");
plt.title('Customers');

# plot second ECDF
plt.subplot(313)
cdf = ECDF(train['SalePerCustomer'])
plt.plot(cdf.x, cdf.y, label = "statmodels");
plt.title('Sale per Customer');
plt.subplots_adjust(hspace = 0.8)
```



Filling missing values:

Filling missing values in a dataset is of great significance as it helps prevent biases, maximize data utility, maintain statistical power, enhance model performance, and preserve data integrity. When missing values are not addressed, it can introduce biases in the analysis, leading to skewed results and misleading conclusions. Filling missing values ensures that the data is more representative of the underlying population, allowing for more accurate and unbiased analysis. It also maximizes the utility of the data by retaining as much information as possible, enabling more comprehensive analysis and interpretation. Additionally, filling missing values helps maintain an adequate sample size, ensuring that statistical tests and models have sufficient power to detect meaningful patterns or relationships. Machine learning algorithms often cannot handle missing values, so filling them in allows for effective training and testing of models, improving their predictive performance. Lastly, filling missing values preserves the integrity and consistency of the dataset, minimizing errors and inconsistencies that may arise from incomplete or missing information.

Filling Missing Values

```
[ ]  # missing values?
     store.isnull().sum()
```

```
Store                         0
StoreType                     0
Assortment                    0
CompetitionDistance           3
CompetitionOpenSinceMonth   354
CompetitionOpenSinceYear    354
Promo2                        0
Promo2SinceWeek             544
Promo2SinceYear             544
PromoInterval               544
dtype: int64
```

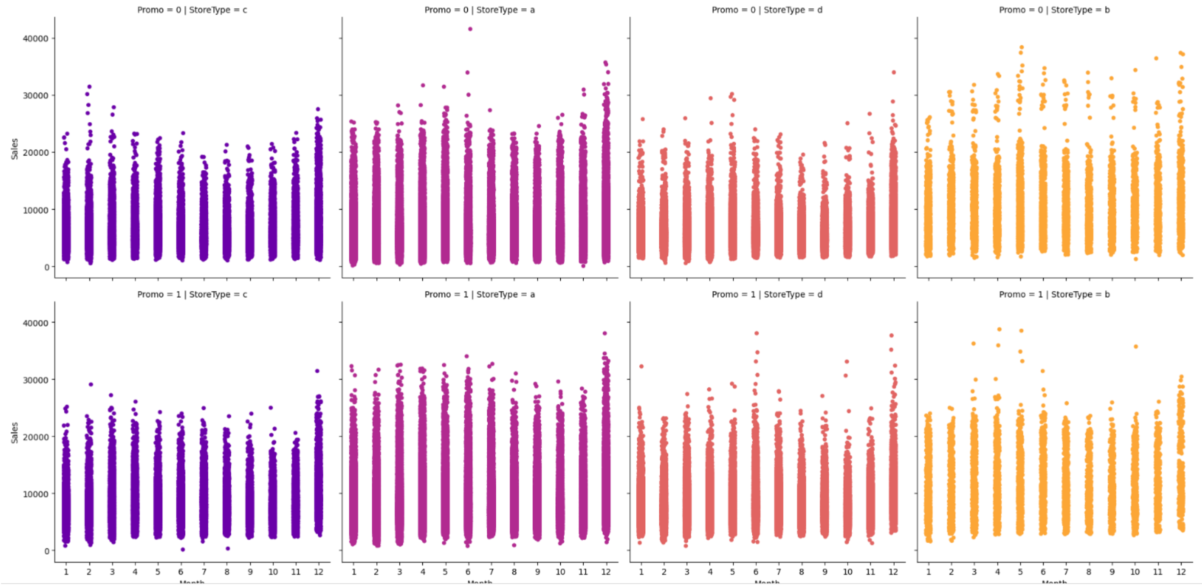
```
# missing values in CompetitionDistance
store[pd.isnull(store.CompetitionDistance)]
```

	Store	StoreType	Assortment	CompetitionDistance	CompetitionOpenSinceMonth	CompetitionOpenSinceYear	Promo2	Promo2SinceWeek	Promo2SinceYear	PromoInterval
290	291	d	a	NaN	NaN	NaN	0	NaN	NaN	NaN
621	622	a	c	NaN	NaN	NaN	0	NaN	NaN	NaN
878	879	d	a	NaN	NaN	NaN	1	5.0	2013.0	Feb,May,Aug,Nov

Observing sales trends depending on the promo availability:

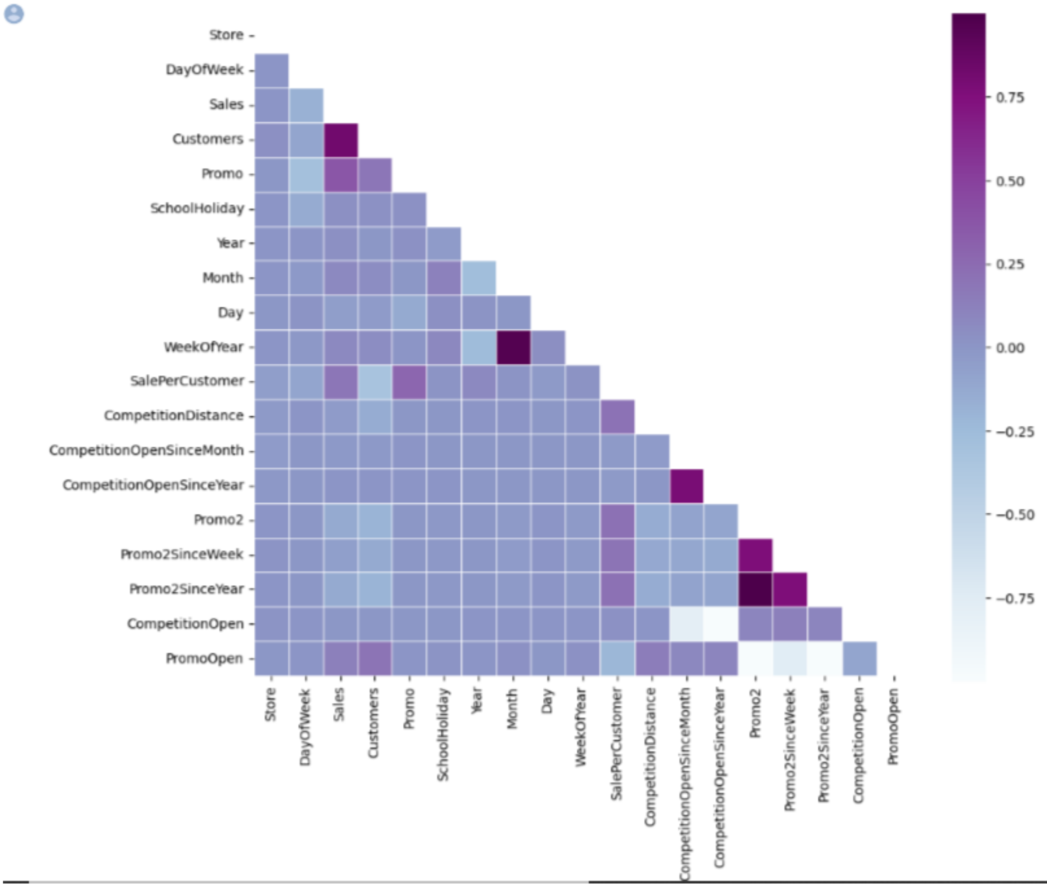
We can clearly see that when there is a promo available then the store sales are a bit higher than the day which has no promo available, thus we can confirm that store sales are affected by the availability of the promo. In the months of November and December we can see a spike in the sales due to the festival season and availability of the promo.

```
# sales trends
sns.catplot(data = train_store, x = 'Month', y = "Sales",
               col = 'StoreType', # per store type in cols
               palette = 'plasma',
               hue = 'StoreType',
               row = 'Promo')
```



Observation: There is high relationship between Customers and Sales and Promo

```
# Compute the correlation matrix
# exclude 'Open' variable
corr_all = train_store.drop('Open', axis = 1).corr()

# Generate a mask for the upper triangle
mask = np.zeros_like(corr_all, dtype = np.bool)
mask[np.triu_indices_from(mask)] = True

# Set up the matplotlib figure
f, ax = plt.subplots(figsize = (11, 9))

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr_all, mask = mask,
            square = True, linewidths = .5, ax = ax, cmap = "BuPu")
plt.show()
```



DATA MANIPULATION:

Data manipulation refers to the process of transforming and reorganizing data to extract valuable insights and facilitate analysis. It involves tasks such as filtering, sorting, merging, aggregating, and transforming data to meet specific requirements. Data manipulation allows for data cleaning, formatting, and restructuring, ensuring data quality and consistency. It enables the creation of new variables or derived metrics that provide deeper insights into the data. Through data manipulation techniques, analysts can tailor the data to suit their analytical needs, perform calculations, and create meaningful visualizations, ultimately enabling effective decision-making and generating actionable insights from the data.

DATA MANIPULATION

```
[ ] # to numerical
    mappings = {'0':0, 'a':1, 'b':2, 'c':3, 'd':4}

    train['StateHoliday'] = train['StateHoliday'].replace(mappings).astype('int64')

    test['StateHoliday'] = test['StateHoliday'].replace(mappings).astype('int64')

    store['StoreType'] = store['StoreType'].replace(mappings).astype('int64')
    store['Assortment'] = store['Assortment'].replace(mappings).astype('int64')
    store.drop('PromoInterval', axis = 1, inplace = True)

    train_store = pd.merge(train, store, how = 'inner', on = 'Store')
    test_store = pd.merge(test, store, how = 'inner', on = 'Store')
```

TIME SERIES ANALYSIS:

Time series analysis is a statistical technique used to analyse and interpret data that is collected over time. It involves studying the patterns, trends, and relationships within the time-dependent data to make predictions or understand underlying mechanisms. Time series analysis encompasses various methods such as decomposition, smoothing, autocorrelation, and forecasting. It is widely employed in fields such as finance, economics, weather forecasting, and sales forecasting. By analysing past data points and their temporal dependencies, time series analysis provides valuable insights into the behaviour of the data, allowing for informed decision-making, forecasting future values, and identifying patterns or anomalies that may impact future outcomes.

Seasonal_decompose

```
[ ] # Choose 1 store with type a, namely store 2
    sales_2 = train[train['Store'] == 2][['Sales', 'Date','StateHoliday','SchoolHoliday']]
```

```
[ ] sales_2['Date'].sort_index(ascending = False, inplace=True)
```

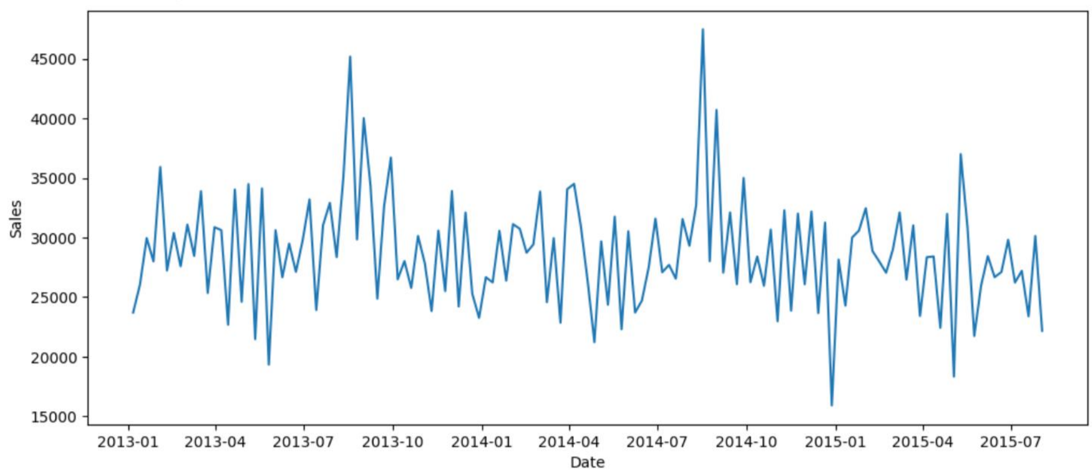
```
a = sales_2.set_index('Date').resample('W').sum()
a
```

Date	Sales	StateHoliday	SchoolHoliday
2013-01-06	23704	0	4
2013-01-13	26030	0	5
2013-01-20	29960	0	5
2013-01-27	28006	0	1
2013-02-03	35928	0	0
...	...	...	...
2015-07-05	26228	0	0
2015-07-12	27211	0	0
2015-07-19	23397	0	0
2015-07-26	30134	0	0
2015-08-02	22182	0	3

135 rows × 3 columns

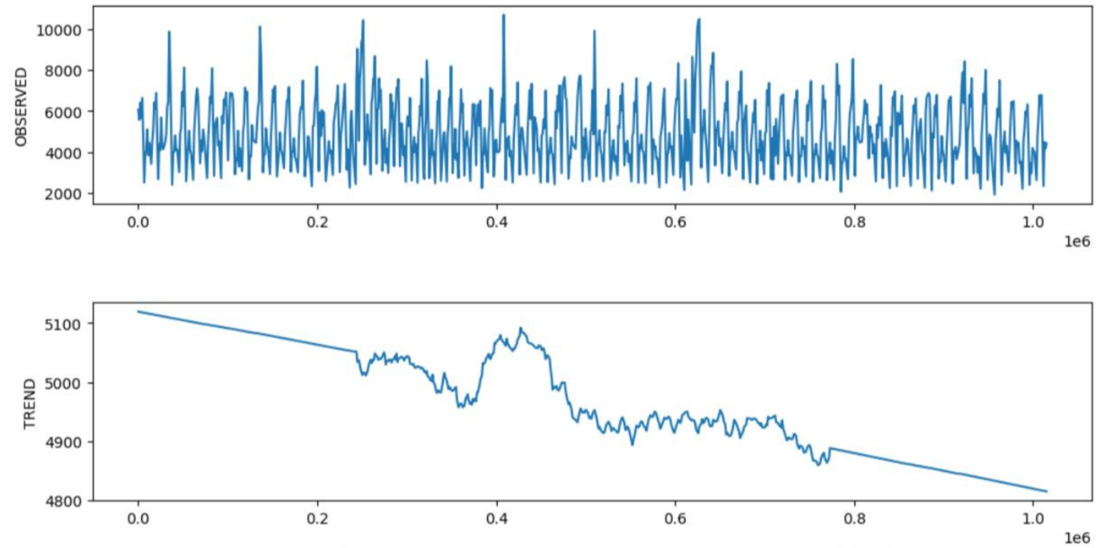
```
[ ] plt.figure(figsize=(12, 5))
    sns.lineplot(x=a.index, y=a['Sales'])
```

<Axes: xlabel='Date', ylabel='Sales'>



Observation: At the first glance, store 1 show stationary pattern. Unfortunately, the trend experience downward.

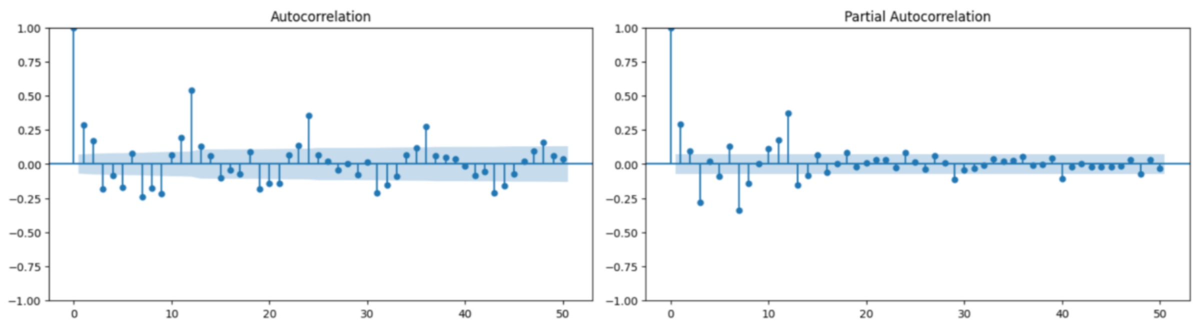
```
f, (ax1, ax2) = plt.subplots(2, figsize = (12, 6))

# monthly
decomposition_a = seasonal_decompose(sales_2['Sales'], model = 'additive', extrapolate_trend='freq', period=365)
decomposition_a.observed.plot(ax = ax1)
ax1.set_ylabel('OBSERVED')
decomposition_a.trend.plot(ax = ax2)
ax2.set_ylabel('TREND')
f.subplots_adjust(hspace = 0.5)
```



Autocorrelation:

Observation: ACF is a measure of the correlation between the timeseries with a lagged version of itself. I choose lags of 50. ACF help us choose MA model. PACF, on the other hand, measures the correlation between the timeseries with a lagged version of itself, after removing the influence of any variance in the middle. PACF helps us choose AR model. Looking at the Autocorrelation graph shows store 1 has high seasonality at 12 lags, it experienced positive spike at 12 lags, 24 lags, This store show high correlation between the current time unit with the previous time unit.

```
fig = plt.figure(constrained_layout=True, figsize=(15, 4))
grid = gridspec.GridSpec(nrows=1, ncols=2,  figure=fig)

# acf and pacf for A
ax1 = fig.add_subplot(grid[0, 0])
plot_acf(sales_2['Sales'], lags = 50, ax=ax1);

# acf and pacf for A
ax1 = fig.add_subplot(grid[0, 1])
plot_pacf(sales_2['Sales'], lags = 50, ax=ax1);

plt.show();
```



TIME SERIES MODELS:

Time series models are statistical models specifically designed to analyze and forecast data collected over time. These models include Autoregressive Integrated Moving Average (ARIMA) for capturing linear dependencies, Seasonal ARIMA (SARIMA) for handling seasonal patterns, Exponential Smoothing (ES) for weighted averaging, Vector Autoregression (VAR) for interdependent variables, Bayesian Structural Time Series (BSTS) for incorporating uncertainty, Prophet for decomposable time series modeling, and Long Short-Term Memory (LSTM) Networks for capturing long-term dependencies. Each model has its own strengths and is selected based on the characteristics of the time series data and the specific analysis requirements. These models enable accurate forecasting, trend identification, and pattern recognition in time series data, aiding in decision-making and planning.

TIME SERIES MODELS

ARIMA FAMILY FORECASTING

```python
# adfuller helps us to determine the right model for analysis.
# For example, the returned value from adf_test show 'Fail to reject the null hypothesis', it means we should make differencing.

from statsmodels.tsa.stattools import adfuller

def adf_test(series,title=''):
    """
    Pass in a time series and an optional title, returns an ADF report
    """
    print(f'Augmented Dickey-Fuller Test: {title}')
    result = adfuller(series.dropna(),autolag='AIC') # .dropna() handles differenced data

    labels = ['ADF test statistic','p-value','# lags used','# observations']
    out = pd.Series(result[0:4],index=labels)

    for key,val in result[4].items():
        out[f'critical value ({key})']=val

    print(out.to_string())          # .to_string() removes the line "dtype: float64"

    if result[1] <= 0.05:
        print("Strong evidence against the null hypothesis")
        print("Reject the null hypothesis")
        print("Data has no unit root and is stationary")
    else:
        print("Weak evidence against the null hypothesis")
        print("Fail to reject the null hypothesis")
        print("Data has a unit root and is non-stationary")
```

```
[ ] adf_test(sales_2['Sales'])

    Augmented Dickey-Fuller Test:
    ADF test statistic       -5.292708
    p-value                   0.000006
    # lags used              17.000000
    # observations          766.000000
    critical value (1%)      -3.438916
    critical value (5%)      -2.865321
    critical value (10%)     -2.568783
    Strong evidence against the null hypothesis
    Reject the null hypothesis
    Data has no unit root and is stationary
```

```
[ ] x = sales_2.set_index('Date').loc[:'2015-06-30']
    y = sales_2.set_index('Date').loc['2015-07-01':]
```

```
[ ] !pip install statsmodels --upgrade

    Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
    Requirement already satisfied: statsmodels in /usr/local/lib/python3.10/dist-packages (0.13.5)
    Collecting statsmodels
      Downloading statsmodels-0.14.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (10.1 MB)
         ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10.1/10.1 MB 54.4 MB/s eta 0:00:00
    Requirement already satisfied: numpy>=1.18 in /usr/local/lib/python3.10/dist-packages (from statsmodels) (1.22.4)
    Requirement already satisfied: scipy!=1.9.2,>=1.4 in /usr/local/lib/python3.10/dist-packages (from statsmodels) (1.10.1)
    Requirement already satisfied: pandas>=1.0 in /usr/local/lib/python3.10/dist-packages (from statsmodels) (1.5.3)
    Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.10/dist-packages (from statsmodels) (0.5.3)
    Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from statsmodels) (23.1)
    Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0->statsmodels) (2.8.2)
    Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0->statsmodels) (2022.7.1)
    Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.2->statsmodels) (1.16.0)
    Installing collected packages: statsmodels
      Attempting uninstall: statsmodels
        Found existing installation: statsmodels 0.13.5
        Uninstalling statsmodels-0.13.5:
          Successfully uninstalled statsmodels-0.13.5
    Successfully installed statsmodels-0.14.0
```

AR MODEL:

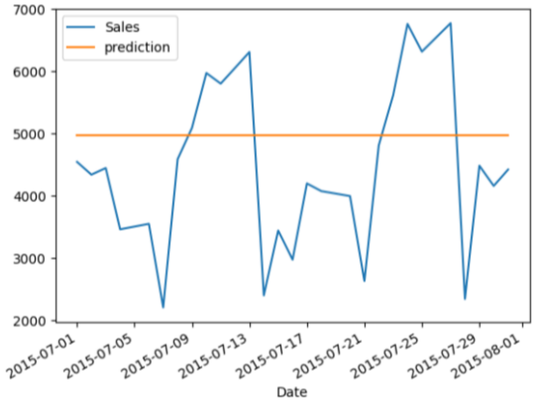
The Autoregressive (AR) model is a widely used time series model that captures the linear dependencies within a time series by using its own past values as predictors. In an AR model, the current value of the time series is assumed to be a linear combination of its past values, weighted by coefficients. The order of the AR model, denoted as AR(p), determines the number of lagged terms used as predictors, where 'p' represents the number of past observations considered. The AR model is suitable for stationary time series data, where the mean and variance remain constant over time. It is used to analyze the temporal patterns, detect trends, and forecast future values based on the historical behavior of the series. The coefficients estimated in the AR model provide insights into the strength and significance of the lagged terms, helping to understand the relationship between past and current observations in the time series.

AR Model

```
[ ] import statsmodels.tsa.ar_model as ar_model
```

```
[ ] from statsmodels.tsa.arima_model import ARMA
    import statsmodels.api as sm
```

```
model = sm.tsa.arima.ARIMA(x['Sales'])
AR1fit = model.fit()
#print(f'Lag: {AR1fit.k_ar}')
#print(f'Coefficients:\n{AR1fit.params}')

# This is the general format for obtaining predictions
start=len(x)
end=len(x)+len(y)-1
predictions1 = AR1fit.predict(start=start, end=end, dynamic=False)

predictions1.index = y.index

y['Sales'].plot()
predictions1.plot(label='prediction');
plt.legend()
plt.show()

print('Root Mean Squared Error: ', np.sqrt(mean_squared_error(y['Sales'], predictions1)))
```




```
Root Mean Squared Error:  1415.7818265115893
```

ARMA MODEL:

The Autoregressive Moving Average (ARMA) model is a popular time series model that combines the autoregressive (AR) and moving average (MA) components to analyze and forecast time series data. In an ARMA model, the current value of the time series is expressed as a linear combination of its own past values (AR component) and the past error terms (MA component). The AR component captures the linear dependence on past observations, while the MA component accounts for the residual error and captures the short-term fluctuations. The order of the ARMA model, denoted as ARMA (p, q), represents the number of past observations used in the AR component (p) and the number of past error terms considered in the MA component (q). The ARMA model is widely used for stationary time series data and provides valuable insights into the temporal patterns, noise, and forecasting of future values based on the historical behavior of the series.

ARMA Mode

```
[ ] # auto_arima help us choose the optimal model, sometime manually tweaking model hyperparameters yeild better result.
    auto_arima(x['Sales'],seasonal=False).summary()
```

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	757
Model:	SARIMAX(5, 0, 0)	**Log Likelihood**	-6597.707
Date:	Fri, 02 Jun 2023	**AIC**	13209.414
Time:	13:38:04	**BIC**	13241.819
Sample:	0 - 757	**HQIC**	13221.895
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	4397.3565	286.592	15.344	0.000	3835.647	4959.066
ar.L1	0.2923	0.037	7.830	0.000	0.219	0.365
ar.L2	0.1348	0.042	3.232	0.001	0.053	0.217
ar.L3	-0.2687	0.043	-6.314	0.000	-0.352	-0.185
ar.L4	0.0486	0.042	1.147	0.252	-0.034	0.132
ar.L5	-0.0918	0.037	-2.507	0.012	-0.163	-0.020
sigma2	2.184e+06	1.09e+05	20.104	0.000	1.97e+06	2.4e+06

Ljung-Box (L1) (Q):	0.12	**Jarque-Bera (JB):**	2.08
Prob(Q):	0.73	**Prob(JB):**	0.35
Heteroskedasticity (H):	0.79	**Skew:**	-0.04
Prob(H) (two-sided):	0.07	**Kurtosis:**	3.25

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

```
[ ] # Adding exorgenous variable may help accuracy improvement. Let's see
    # It doesnt improve as the store usually closed on StateHoliday or SchoolHoliday and sales may not escalated even if it oper
    auto_arima(x['Sales'], exorgenous=x[['StateHoliday','SchoolHoliday']],seasonal=False).summary()
```

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	757
Model:	SARIMAX(5, 0, 0)	**Log Likelihood**	-6597.707
Date:	Fri, 02 Jun 2023	**AIC**	13209.414
Time:	13:40:06	**BIC**	13241.819
Sample:	0 - 757	**HQIC**	13221.895
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	4397.3565	286.592	15.344	0.000	3835.647	4959.066
ar.L1	0.2923	0.037	7.830	0.000	0.219	0.365
ar.L2	0.1348	0.042	3.232	0.001	0.053	0.217
ar.L3	-0.2687	0.043	-6.314	0.000	-0.352	-0.185
ar.L4	0.0486	0.042	1.147	0.252	-0.034	0.132
ar.L5	-0.0918	0.037	-2.507	0.012	-0.163	-0.020
sigma2	2.184e+06	1.09e+05	20.104	0.000	1.97e+06	2.4e+06

Ljung-Box (L1) (Q):	0.12	**Jarque-Bera (JB):**	2.08
Prob(Q):	0.73	**Prob(JB):**	0.35
Heteroskedasticity (H):	0.79	**Skew:**	-0.04
Prob(H) (two-sided):	0.07	**Kurtosis:**	3.25

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

Warnings: [1] Covariance matrix calculated using the outer product of gradients (complex-step). [2] Covariance matrix is singular or near-singular, with condition number 1.2e+23. Standard errors may be unstable.

SARIMA MODEL:

The Seasonal Autoregressive Integrated Moving Average (SARIMA) model is an extension of the Autoregressive Integrated Moving Average (ARIMA) model that incorporates seasonal patterns in time series data. The SARIMA model captures the dependencies and trends present in a time series, considering both the non-seasonal and seasonal components. It combines the autoregressive (AR), differencing (I), and moving average (MA) components with additional seasonal terms. The order of the SARIMA model, denoted as SARIMA (p, d, q) (P, D, Q, s), represents the non-seasonal and seasonal components, including the number of autoregressive terms (p), differencing levels (d), moving average terms (q), seasonal autoregressive terms (P), seasonal differencing levels (D), seasonal moving average terms (Q), and the length of the seasonal period (s). The SARIMA model is widely used for time series data with prominent seasonal patterns, allowing for accurate analysis, forecasting, and identification of seasonal trends and fluctuations.

SARIMA Model

```
[ ] # As we talk above, we may be interested in the fact that event or seasonality can influence sale of store.
    # However, in this case, adding seasonality worsen model. Thus, there is no clear seasonal component in this case.

    # https://alkaline-ml.com/pmdarima/tips_and_tricks.html#setting-m
    # m = 7(daily), 12(monthly), 52(weekly)
    auto_arima(x['Sales'],seasonal=True, m=7).summary()
```

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	757
Model:	SARIMAX(4, 0, 4)x(0, 0, [1, 2], 7)	**Log Likelihood**	-6527.569
Date:	Fri, 02 Jun 2023	**AIC**	13079.138
Time:	13:45:07	**BIC**	13134.690
Sample:	0 - 757	**HQIC**	13100.534
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	6096.8860	2961.808	2.059	0.040	291.849	1.19e+04
ar.L1	0.3178	0.287	1.106	0.269	-0.245	0.881
ar.L2	-0.0415	0.043	-0.973	0.330	-0.125	0.042
ar.L3	-0.8786	0.035	-24.994	0.000	-0.948	-0.810
ar.L4	0.3762	0.261	1.440	0.150	-0.136	0.888
ma.L1	-0.0425	0.303	-0.140	0.889	-0.637	0.552
ma.L2	0.1796	0.096	1.862	0.063	-0.009	0.369
ma.L3	0.7582	0.083	9.094	0.000	0.595	0.922
ma.L4	-0.2270	0.259	-0.876	0.381	-0.735	0.281
ma.S.L7	-0.3058	0.049	-6.277	0.000	-0.401	-0.210
ma.S.L14	0.0902	0.046	1.942	0.052	-0.001	0.181
sigma2	1.99e+06	9.84e+04	20.226	0.000	1.8e+06	2.18e+06

Ljung-Box (L1) (Q):	0.02	**Jarque-Bera (JB):**	63.85
Prob(Q):	0.88	**Prob(JB):**	0.00
Heteroskedasticity (H):	0.73	**Skew:**	0.47
Prob(H) (two-sided):	0.01	**Kurtosis:**	4.07

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

```
model = SARIMAX(x['Sales'],order=(5, 0, 3),seasonal_order=(0, 0, 1, 7))
results = model.fit()

start=len(x)
end=len(x)+len(y)-1
predictions1 = results.predict(start=start, end=end, dynamic=False)

predictions1.index = y.index

y['Sales'].plot()
predictions1.plot(label='prediction');
plt.legend()

print('Root Mean Squared Error: ', np.sqrt(mean_squared_error(y['Sales'], predictions1)))
```

```
Root Mean Squared Error:  1178.0944803307423
```

REGRESSION MODELS:

Regression models are statistical models used to analyze the relationship between a dependent variable and one or more independent variables. These models estimate the impact of the independent variables on the dependent variable and allow for prediction or inference. In regression analysis, the dependent variable is modeled as a function of the independent variables, with the goal of finding the best-fitting line or curve that minimizes the difference between the predicted and observed values. Regression models come in various forms, such as linear regression, polynomial regression, logistic regression, and multiple regression, each suited for different types of data and analysis objectives. They provide valuable insights into the direction, strength, and significance of the relationships between variables, enabling decision-making, prediction, and understanding of the underlying factors influencing the dependent variable.

REGRESSION MODELS

```
[ ] train_store['CompetitionOpen'] = 12 * (train_store['Year'] - train_store['CompetitionOpenSinceYear']) + (train_store['Month'] - train_store['CompetitionOpenSinceMonth'])
    train_store['PromoOpen'] = 12 * (train_store['Year'] - train_store['Promo2SinceYear']) + (train_store['WeekOfYear'] - train_store['Promo2SinceWeek']) / 4.0

    test_store['CompetitionOpen'] = 12 * (test_store['Year'] - test_store['CompetitionOpenSinceYear']) + (test_store['Month'] - test_store['CompetitionOpenSinceMonth'])
    test_store['PromoOpen'] = 12 * (test_store['Year'] - test_store['Promo2SinceYear']) + (test_store['WeekOfYear'] - test_store['Promo2SinceWeek']) / 4.0
```

```
[ ] # Sorting dataframe according to datatime, the oldest is on top, the most recent is at the bottom.
    train_store['Date'].sort_index(ascending = False, inplace=True)
```

```
[ ] def rmsle(y_pred, y):
        return np.sqrt(mean_squared_error(y_pred, y))

    def model_check (estimators):
        model_table = pd.DataFrame()
        row_index = 0

        for est in estimators:
            MLA_name = est.__class__.__name__
            model_table.loc[row_index, 'Model Name'] = MLA_name

            est.fit(x_train, y_train)
            y_pred = est.predict(x_test)
            model_table.loc[row_index, 'Test Error'] = rmsle(y_pred, y_test)

            row_index += 1

            model_table.sort_values(by=['Test Error'],
                                    ascending=True,
                                    inplace=True)
        return model_table
```

```
[ ] # MODELS
    lr = LinearRegression()
    ls = Lasso()
    GBoost = GradientBoostingRegressor(random_state = 0)
    XGBoost = XGBRegressor(random_state = 0, n_job=-1)
    LGBM = LGBMRegressor(random_state = 0, n_job=-1)
```

```
[ ] # Training dataset is separated into train_a and test_a.
    # traing_a train data from 2013 till 2015-06-30, while test_a contain  data from 2015-07-01 till 2015-07-31.

    train_a = train_store.set_index('Date').loc[:'2015-06-30']
    test_a = train_store.set_index('Date').loc['2015-07-01':]

    x_train = train_a.drop(['Sales', 'Customers'], axis=1)
    y_train = train_a['Sales']
    x_test = test_a.drop(['Sales', 'Customers'], axis=1)
    y_test = test_a['Sales']
```

```
[ ] estimators = [lr, ls, GBoost, XGBoost, LGBM]
    model_check(estimators)
```

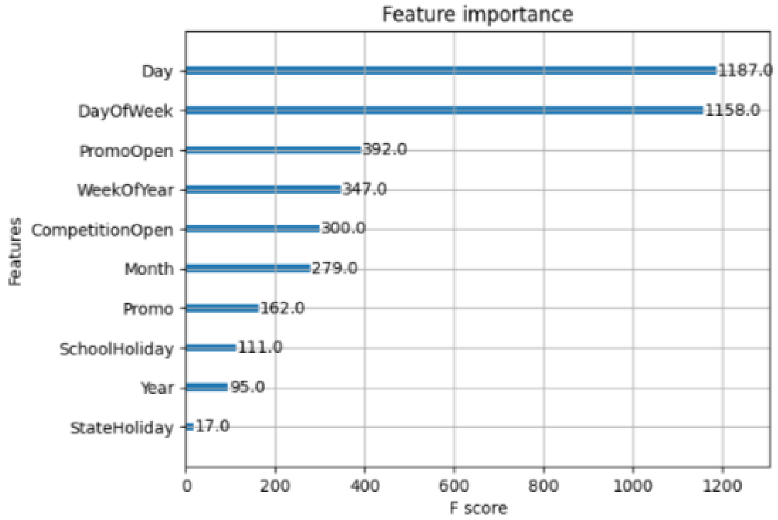
```
[14:02:44] WARNING: ../src/learner.cc:767:
Parameters: { "n_job" } are not used.

[LightGBM] [Warning] Unknown parameter: n_job
```

	Model Name	Test Error
3	XGBRegressor	1079.952905
4	LGBMRegressor	1616.072804
2	GradientBoostingRegressor	2272.147321
0	LinearRegression	2672.310857
1	Lasso	2672.421082

Observation: Seeing both feature importance of XGBoost and LightGBM show similar patterns. Day, DayOfWeek, WeekOfYear, PromoOpen, Promo primarily account for sale amount

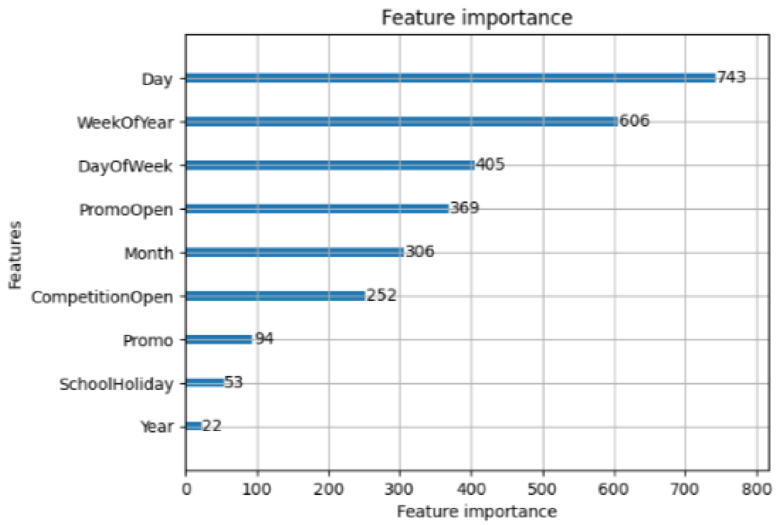
```
[ ] xgb.plot_importance(XGBoost)
```

```
<Axes: title={'center': 'Feature importance'}, xlabel='F score', ylabel='Features'>
```




```
[ ] lightgbm.plot_importance(LGBM)
```

```
<Axes: title={'center': 'Feature importance'}, xlabel='Feature importance', ylabel='Features'>
```



FINAL PRODUCT PROTOTYPE/PRODUCT DETAILS:

converting the Rossman store prediction project into a business model:

1. Define the Value Proposition: Identify the core value that the prediction project brings to the Rossman store. This could include improved sales forecasting, optimized inventory management, enhanced decision-making, or increased profitability.

2. Identify Target Customers: Determine the target customers who would benefit the most from the prediction project. This could include retail store owners, managers, or executives seeking data-driven insights to optimize their operations and improve business outcomes.

3. Develop Revenue Streams: Determine how the prediction project can generate revenue. This could involve offering the prediction model as a subscription-based service, charging consulting fees for implementation and customization, or bundling it with other related products or services.

4. Outline Key Activities and Resources: Identify the key activities required to deliver the prediction project as a business model. This includes data collection, data preprocessing, model training, implementation, ongoing maintenance, and customer support. Additionally, define the necessary resources such as data infrastructure, skilled personnel, and technological tools.

5. Consider Partnerships: Evaluate potential partnerships or collaborations that can enhance the value proposition. This could involve partnering with software developers, data providers, or other retail industry stakeholders to improve the accuracy and effectiveness of the prediction model.

6. Determine the Cost Structure: Assess the costs associated with developing, implementing, and maintaining the prediction project as a business model. Consider expenses such as data acquisition, infrastructure, staffing, software licenses, marketing, and ongoing research and development.

7. Create a Go-to-Market Strategy: Develop a comprehensive go-to-market strategy to reach and acquire customers. This could involve targeted marketing campaigns, industry partnerships, attending trade shows or conferences, or leveraging digital marketing channels to raise awareness and generate leads.

8. Establish Key Metrics: Define key performance indicators (KPIs) to measure the success of the business model. This could include metrics such as customer acquisition rate, revenue growth, customer satisfaction, and retention.

9. Continuously Improve and Adapt: Regularly evaluate customer feedback, market trends, and emerging technologies to enhance the prediction project and adapt the business model accordingly. Seek opportunities to expand the offering, integrate new features, or target additional customer segments.

By converting the Rossman store prediction project into a business model, we can provide a valuable solution to retail businesses, generate revenue, and establish a sustainable and scalable venture.

FEASIBILITY:

This project can be developed and deployed within a few years as SaaS (Software as a Service) for anyone to use. This will be very easily operated with our mobiles and Laptops. We can also analyse the different sectors data and we can give insights to them and make a handsome money. For building a data analytics team and data analytics tolls require a medium amount of time.

VIABILITY:

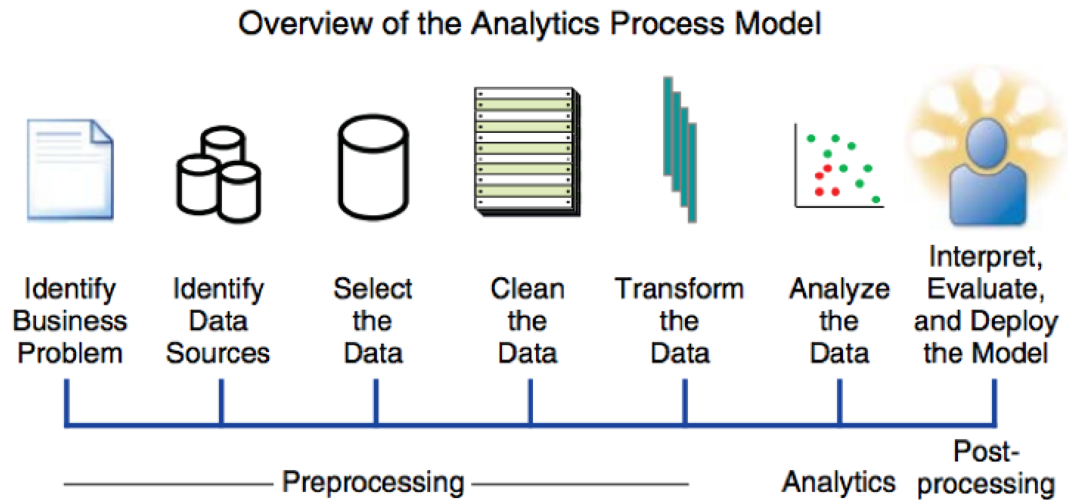
Data analytics is highly viable in today's business landscape, offering significant benefits and competitive advantages. By leveraging data, organizations can gain valuable insights into customer behaviour, market trends, and operational performance, enabling informed decision-making and improved business outcomes. Data analytics enhances efficiency, reduces costs, and facilitates personalized customer experiences. It enables businesses to identify growth opportunities, optimize operations, and drive innovation. With its ability to provide evidence-based insights and support continuous improvement, data analytics has become an essential capability for organizations seeking to stay competitive and thrive in the data-driven era.

MONITIZATION:

The monetization of drug sales forecasts can be achieved through various means. One approach is to offer the forecasts as a subscription-based service, providing pharmaceutical companies, healthcare providers, and other relevant stakeholders with regular access to accurate and reliable sales predictions. Additionally, the forecasts can be integrated into existing software platforms or analytics solutions, creating value-added offerings for clients. Another monetization strategy is to provide consulting services, where data analytics experts collaborate with organizations to interpret and apply the forecasted insights to optimize their sales strategies, resource allocation, and inventory management. Furthermore, partnerships with market research firms, pharmaceutical manufacturers, or retail chains can be explored to license or sell the drug sales forecast models and methodologies. By effectively monetizing drug sales forecasts, the prediction project can generate revenue streams while delivering valuable insights and aiding decision-making in the pharmaceutical industry.

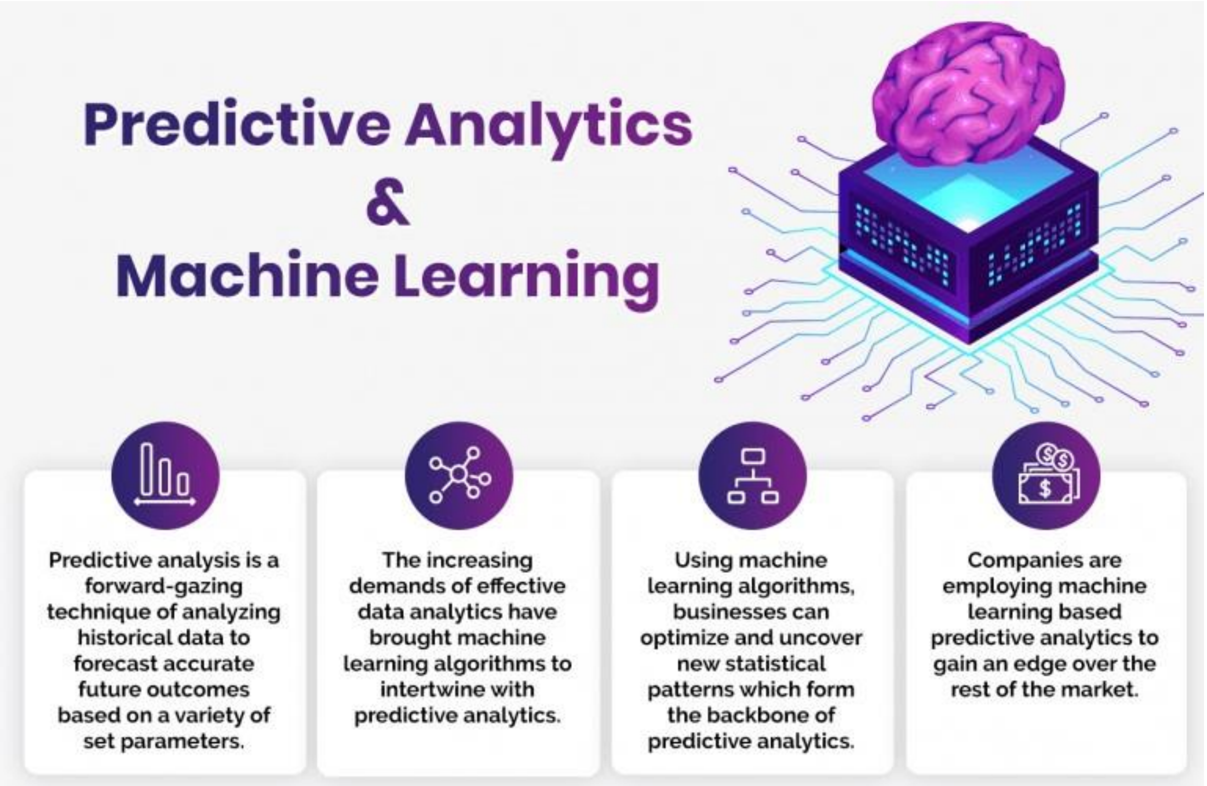
PROTOTYPE DEVELOPMENT:

https://github.com/yashwanthreddyGoduguchintha/feynn_Repo/blob/16f813ba94c0b48fff471210fd3cb8799a543e2f/SALE_PREDICTION_Using_Time_Series_Forecating_%26_Regressor_Problem_.ipynb



BUSINESS MODEL:

Data analytics as a business model involves leveraging data-driven insights and expertise to provide valuable services to clients. By collecting, analyzing, and interpreting data, businesses can offer customized analytics solutions, consulting services, or subscription-based access to their analytics platforms. They help clients make informed decisions, optimize operations, and gain a competitive advantage through data-driven strategies. Data analytics companies generate revenue through consulting fees, subscription charges, licensing data or software, and offering add-on services such as data visualization, predictive modeling, and data integration. As businesses increasingly recognize the importance of data-driven decision-making, data analytics as a business model presents a viable opportunity to meet the growing demand for actionable insights and drive success in the digital era.



CONCLUSION:

In conclusion, data analytics as a business model offers immense potential for organizations to capitalize on the power of data. By leveraging advanced analytics techniques and expertise, businesses can provide valuable insights and solutions to clients, enabling them to make data-driven decisions and optimize their operations. The monetization of data analytics can be achieved through various revenue streams, including consulting services, subscription-based access to analytics platforms, and licensing of data or software. With the increasing importance of data-driven decision-making in today's competitive landscape, data analytics as a business model presents a compelling opportunity for organizations to thrive and drive success in the digital age.