PHASE1 – PROBLEM SOLVING AND DESIGN THINKING

Developing a predictive sales forecasting model for a retail company is a valuable project that can help optimize inventory management and improve business decision-making. Here's a step-by-step guide on how to approach this project:

1. **Understand the Problem:**
   * Meet with stakeholders to understand their specific needs and goals for sales forecasting.
   * Determine the time horizon for forecasting (e.g., daily, weekly, monthly).
   * Define the key performance indicators (KPIs) for model evaluation (e.g., Mean Absolute Error, Root Mean Square Error).
2. **Data Collection:**
   * Gather historical sales data, which may include information such as date, product/category, location, price, promotions, and external factors (e.g., holidays, weather).
   * Ensure the data is clean, consistent, and relevant to the forecasting task.
3. **Data Preprocessing:**
   * Handle missing data, outliers, and duplicates.
   * Convert categorical variables into numerical format using techniques like one-hot encoding or label encoding.
   * Normalize or scale numerical features to bring them to a common scale.
   * Split the data into training and testing sets (e.g., 70-30 or 80-20 split).
4. **Feature Engineering:**
   * Create relevant features that can help the model capture seasonality, trends, and other patterns in the data.
   * Consider lag features, rolling statistics, and external data integration (e.g., economic indicators, social media trends).
   * Engineer features like day-of-week, month, and holiday indicators.
5. **Model Selection:**
   * Experiment with different forecasting models such as Time Series models (e.g., ARIMA, Exponential Smoothing), Machine Learning models (e.g., Random Forest, Gradient Boosting), and Deep Learning models (e.g., LSTM, GRU).
   * Select the model that performs best based on your defined evaluation metrics.
   * Consider using ensemble methods or model stacking to combine the strengths of multiple models.
6. **Model Training:**
   * Train the selected model(s) on the training dataset.
   * Tune hyperparameters using techniques like grid search or random search to optimize model performance.
   * Validate the model using cross-validation if applicable.
7. **Model Evaluation:**
   * Evaluate the model(s) using the testing dataset.
   * Calculate relevant metrics (e.g., MAE, RMSE) to assess model accuracy.
   * Visualize model predictions against actual sales data to understand the model's performance.
8. **Deployment:**
   * Deploy the trained model in a production environment for real-time or batch forecasting.
   * Implement a data pipeline to feed new sales data into the model.
   * Set up automated retraining if the data distribution changes over time.
9. **Monitoring and Maintenance:**
   * Continuously monitor the model's performance in production.
   * Retrain the model periodically with new data to ensure its accuracy and relevance.
   * Make necessary updates to feature engineering or model selection as the business evolves.
10. **Reporting and Visualization:**
    * Provide stakeholders with easy-to-understand reports and dashboards displaying sales forecasts and performance metrics.
    * Use visualizations to communicate insights and trends from the data.
11. **Feedback Loop:**
    * Establish a feedback loop with stakeholders to gather insights and incorporate their feedback for model improvement.
12. **Documentation:**
    * Document the entire process, including data sources, preprocessing steps, model details, and deployment instructions.

Remember that sales forecasting is an ongoing process, and the model's accuracy can improve over time with more data and feedback. Collaboration with domain experts and stakeholders is crucial throughout the project to ensure the model aligns with business objectives.

**Top of Form**

**DESIGN THINKING**

Great, you have historical sales data containing essential features like date, product ID, store ID, and sales quantity. Here's how you can proceed with data preprocessing and feature engineering specifically for this dataset:

**Data Preprocessing:**

1. **Data Cleaning:**
   * Check for and handle missing data, if any, in the dataset.
   * Detect and remove or address outliers that might skew the forecasting model.
2. **Data Transformation:**
   * Convert the date column into a datetime format if it's not already.
   * Ensure that the data is sorted by date, as time series data depends on chronological order.
   * You may need to aggregate or resample data if the forecasting time horizon is different from the data's original granularity (e.g., daily data for weekly forecasts).
3. **Encoding Categorical Features:**
   * Encode categorical features like product ID and store ID using techniques such as one-hot encoding or label encoding.
4. **Feature Engineering:**
   * **Date Features:** Extract relevant date-related features like day of the week, month, quarter, and year. These features can help the model capture seasonality and trends.
   * **Lag Features:** Create lag features, such as sales from the previous day(s) or week(s). These can be crucial for capturing autocorrelation in the data.
   * **Rolling Statistics:** Calculate rolling statistics like moving averages or moving sums to capture trends over a specific window of time.
   * **Holiday Indicators:** If holidays impact sales, create binary indicators for holidays or special events.
   * **Price and Promotion Features:** If available, include features related to product prices and promotional activities, as they can significantly influence sales.
   * **Store-Specific Features:** Consider features specific to each store, such as store size, location, or historical performance.
5. **Splitting Data:**
   * Split the dataset into training and testing sets. Typically, you would use the most recent data for testing to simulate real-world forecasting scenarios.
6. **Scaling and Normalization:**
   * Depending on the chosen modeling approach, you may need to scale or normalize the target variable (sales quantity) and input features.

Now that you have preprocessed and engineered the features, you can proceed with model selection, training, and evaluation as described in the previous response. Different modeling approaches, such as Time Series models (e.g., ARIMA), Machine Learning models (e.g., Random Forest, XGBoost), or Deep Learning models (e.g., LSTM), can be trained on this prepared dataset to forecast future sales accurately.

Don't forget to iterate and refine your feature engineering and modeling steps based on the performance of your initial models. Continuous evaluation and improvement are key to developing an effective sales forecasting tool.

Top of Form

2. Categorical features like product ID and store ID need to be converted into numerical representations for machine learning models to work effectively. Here are some common methods:

* **Label Encoding:** Assign a unique integer to each category. This method is suitable when there's an ordinal relationship among categories (i.e., one category is "higher" or "lower" than another).
* **One-Hot Encoding:** Create binary columns for each category, where each column represents the presence or absence of a category. This is suitable when there is no inherent order among categories.
* **Embedding:** For deep learning models, you can use embedding layers to convert categorical variables into dense vectors. This can capture relationships between categories.
* **Target Encoding:** Replace categorical values with the mean or median of the target variable for each category. This can be useful when there's a strong relationship between the categorical feature and the target variable.

3. **1. Date-Based Features:**

* **Day of the Week:** Create a feature that represents the day of the week (e.g., Monday, Tuesday). This can help capture weekly seasonality.
* **Month:** Create a feature that represents the month (e.g., January, February). This can help capture monthly trends and seasonality.
* **Quarter:** Create a feature that represents the quarter of the year (Q1, Q2, Q3, Q4).
* **Year:** Extract the year from the date. This can help capture annual trends and changes over time.
* **Day of the Month:** Create a feature that represents the day of the month (e.g., 1st, 2nd, 3rd). This can capture monthly variations.

**2. Lag Features:**

* **Previous Sales:** Create lag features that represent sales from the previous days or weeks. For example, you can include sales from the previous day (lag-1), the previous week (lag-7), or the previous month (lag-30). These features can capture autocorrelation in the data.

**3. Rolling Statistics:**

* **Moving Averages:** Calculate rolling moving averages for sales over a specified window of time (e.g., 7-day moving average). This can help smooth out noise and reveal trends.
* **Moving Sum:** Similar to moving averages, calculate rolling moving sums to capture the cumulative effect of sales over time.

**4. Holiday Indicators:**

* **Binary Holiday Indicator:** Create binary indicators for holidays or special events. These can be important in capturing sales spikes or drops associated with holidays.

**5. Promotion Features:**

* **Promotion Indicator:** Create binary features to indicate whether a product or store is running a promotion or not. This can capture the impact of promotions on sales.
* **Promotion Duration:** If you have information about promotion start and end dates, create a feature that represents the duration of a promotion.

**6. Seasonal Features:**

* **Season Indicator:** If applicable, create features to indicate different seasons (e.g., spring, summer, fall, winter). This can capture seasonal variations in sales.

**7. Store-Specific Features:**

* **Store Size:** If you have data on store sizes, include this information as a feature.
* **Location:** If store locations differ significantly in terms of demographics or foot traffic, consider including location-based features.

**8. Price Features:**

* **Average Price:** Calculate the average price of products over a specific time window.
* **Price Changes:** Create features to represent price changes or fluctuation

4 Selecting the most suitable time series forecasting algorithm for predicting future sales depends on various factors, including the characteristics of your sales data and your specific forecasting goals. Here are some commonly used time series forecasting algorithms, along with considerations for choosing the right one:

1. **ARIMA (AutoRegressive Integrated Moving Average):**
   * Suitable for: Data with a clear trend and/or seasonality.
   * Consider when: Your sales data exhibits a strong temporal pattern and can be differenced to make it stationary. ARIMA models can handle both trend and seasonality.
2. **Exponential Smoothing (e.g., Holt-Winters):**
   * Suitable for: Data with seasonality and trend.
   * Consider when: Your sales data shows both short-term fluctuations and long-term trends. Holt-Winters is a good choice when you want to capture seasonality, trend, and level in your forecasts.
3. **Prophet:**
   * Suitable for: Data with seasonality, holidays, and special events.
   * Consider when: Your sales data has strong seasonal patterns, holiday effects, and other known events that impact sales. Prophet is designed to handle such data and can provide good results with minimal data preprocessing.
4. **SARIMA (Seasonal ARIMA):**
   * Suitable for: Data with seasonality and autocorrelation.
   * Consider when: Your sales data exhibits seasonality, but there are also autocorrelations present. SARIMA extends ARIMA to account for seasonality and autocorrelations simultaneously.
5. **TBATS (Trigonometric seasonality, Box-Cox transformation, ARMA errors, Trend, and Seasonal components):**
   * Suitable for: Data with multiple seasonal patterns and complex structures.
   * Consider when: Your sales data shows multiple seasonal patterns or complex relationships that are difficult to capture with traditional methods. TBATS can handle various seasonality types.
6. **Neural Networks (e.g., LSTM, GRU):**
   * Suitable for: Complex data with non-linear patterns.
   * Consider when: Your sales data has intricate patterns, dependencies, and long-term dependencies that may not be well-captured by traditional statistical models. Neural networks, especially LSTM and GRU, are suitable for such situations but require more data and computational resources.
7. **XGBoost or LightGBM:**
   * Suitable for: Data with multiple predictors and non-linear relationships.
   * Consider when: You have additional features or predictors (e.g., marketing spend, economic indicators) that influence sales. Gradient boosting algorithms like XGBoost and LightGBM can incorporate these features into the forecasting process.
8. **Hybrid Approaches:**
   * Consider combining multiple forecasting methods or ensembling them to take advantage of their strengths and mitigate weaknesses. For example, combining ARIMA and LSTM or using a weighted average of different models.

5. Training a time series forecasting model, such as ARIMA, Exponential Smoothing, Prophet, or any other chosen algorithm, involves the following steps:

1. **Data Preprocessing:**
   * Ensure that your sales data is properly preprocessed. This may include handling missing values, removing outliers, and making the data stationary if necessary (e.g., differencing).
2. **Splitting the Data:**
   * Split your preprocessed data into two sets: a training set and a testing/validation set. The training set is used to train the model, while the testing/validation set is used to evaluate its performance.
3. **Model Selection:**
   * As you've already selected the forecasting algorithm, make sure you have imported the necessary libraries or packages for that specific model. For instance, if you're using ARIMA, you should import the **statsmodels** library in Python.
4. **Training the Model:**
   * Train the selected model using the training data. The exact steps may vary depending on the chosen algorithm. Here's an example for ARIMA using Python and the **statsmodels** library:

pythonCopy code

from statsmodels.tsa.arima.model import ARIMA # Define the ARIMA model with the appropriate order (p, d, q) model = ARIMA(train\_data, order=(p, d, q)) # Fit the model to the training data model\_fit = model.fit()

Replace **(p, d, q)** with the appropriate values determined during model selection.

1. **Model Validation:**
   * After training the model, use it to make predictions on the testing/validation set.

pythonCopy code

# Make predictions on the testing/validation set predictions = model\_fit.forecast(steps=len(test\_data))

1. **Model Evaluation:**
   * Evaluate the model's performance using appropriate evaluation metrics (e.g., MAE, RMSE, MAPE) by comparing the predicted values to the actual values in the testing/validation set.

pythonCopy code

from sklearn.metrics import mean\_absolute\_error # Calculate MAE mae = mean\_absolute\_error(test\_data, predictions)

1. **Hyperparameter Tuning (if necessary):**
   * Depending on the algorithm, you may need to fine-tune hyperparameters (e.g., seasonality, lag orders) to improve model performance. This can involve using grid search or other optimization techniques.
2. **Model Deployment (if applicable):**
   * If you plan to use the model for ongoing sales forecasting, deploy it in your production environment, ensuring that it's integrated with your data pipeline.
3. **Monitoring and Maintenance:**
   * Continuously monitor the model's performance and retrain it periodically as new data becomes available. Models may need to be updated to adapt to changing sales patterns.

6. Evaluating the performance of your time series forecasting model is crucial to understand how well it's making predictions. You can use various metrics to assess its accuracy and reliability. Here are some commonly used time series forecasting metrics:

1. **Mean Absolute Error (MAE):**
   * MAE measures the average absolute difference between the predicted values and the actual values. It's less sensitive to outliers compared to RMSE.

pythonCopy code

from sklearn.metrics import mean\_absolute\_error mae = mean\_absolute\_error(actual\_values, predicted\_values)

1. **Root Mean Squared Error (RMSE):**
   * RMSE is similar to MAE but gives higher weight to large errors. It's sensitive to outliers and penalizes them more than MAE.

pythonCopy code

from sklearn.metrics import mean\_squared\_error import math rmse = math.sqrt(mean\_squared\_error(actual\_values, predicted\_values))

1. **Mean Absolute Percentage Error (MAPE):**
   * MAPE calculates the average percentage difference between predicted and actual values. It's useful for understanding the relative size of errors.

pythonCopy code

def mean\_absolute\_percentage\_error(actual, predicted): return np.mean(np.abs((actual - predicted) / actual)) \* 100 mape = mean\_absolute\_percentage\_error(actual\_values, predicted\_values)

1. **Symmetric Mean Absolute Percentage Error (sMAPE):**
   * sMAPE is similar to MAPE but symmetric, meaning it gives equal weight to over-predictions and under-predictions.

pythonCopy code

def symmetric\_mean\_absolute\_percentage\_error(actual, predicted): return 100 \* np.mean(2 \* np.abs(predicted - actual) / (np.abs(actual) + np.abs(predicted))) smape = symmetric\_mean\_absolute\_percentage\_error(actual\_values, predicted\_values)

1. **Forecast Bias:**
   * This metric measures the average over- or under-forecasting of the model. A bias close to zero indicates a well-calibrated model.

pythonCopy code

bias = np.mean(predicted\_values - actual\_values)

1. **Forecast Accuracy (Forecast vs. Actual Plot):**
   * Visualize the model's predictions alongside the actual sales data to get a qualitative sense of its accuracy. Plotting the forecasts over time can reveal trends, seasonality, and any systematic errors.
2. **AIC and BIC (for ARIMA models):**
   * For ARIMA models, you can use the Akaike Information Criterion (AIC) or Bayesian Information Criterion (BIC) to compare different model orders (p, d, q) and select the one with the lowest value. Lower AIC/BIC values indicate a better fit.

\*phase2- design into innovation

regression analysis :

* One of the most common methods used to predict sales is **regression analysis.** This method involves using historical sales data to train a model that can predict future sales. The model can take into account factors such as **past sales, marketing campaigns, and economic indicators** to make its predictions.

time series analysis :

* Another popular method for predicting sales is **time series analysis**. This method involves using historical sales data to identify patterns and trends in sales over time. The model can then use these patterns to make predictions about future sales. This method is particularly useful for predicting sales in seasonal industries, such as retail and tourism.

decision tree-based algorithms :

* Another approach is using **decision tree-based algorithms** like **Random Forest, Gradient Boosting** etc. These algorithms are particularly useful when there are many factors that can influence sales, such as product features, customer demographics, and market conditions. The algorithm can help identify the most important factors and use them to make predictions.

neural networks :

* In addition to these methods, machine learning can also be used to predict sales through the use of **neural networks.** Neural networks are a type of machine learning algorithm that can learn to recognize patterns in data. They can be trained on large amounts of sales data and can make predictions about future sales.

1.DATA PREPROCESSING IN SALES DATASET

Data preprocessing is an important step in the data mining process that involves cleaning and transforming raw data to make it suitable for analysis. Some common steps in data preprocessing include:

Data Cleaning:

This involves identifying and correcting errors or inconsistencies in the data, such as missing values, outliers, and duplicates. Various techniques can be used for data cleaning, such as imputation, removal, and transformation.

Data Integration:

This involves combining data from multiple sources to create a unified dataset. Data integration can be challenging as it requires handling data with different formats, structures, and semantics. Techniques such as record linkage and data fusion can be used for data integration.

Data Discretization:

This Involves Dividing Continuous Data Into Discrete Categories Or Intervals. Discretization Is Often Used In Data Mining And Machine Learning Algorithms That Require Categorical Data. Discretization Can Be Achieved Through Techniques Such As Equal Width Binning, Equal Frequency Binning, And Clustering.

And there are many more data preprocessing techniques that helps in obtaining of proper data set

2.CHOOSING THE RIGHT MODEL

The two types of sales forecasting process are generally split into two groups: quantitative sales forecasting and qualitative sales forecasting

QUANTITATIVE SALES FORECASTING :

The so called quantitative methods of sales forecasting are those used with the availability of historical sales data that can be extrapolated to predict future revenue. These methods rely more on sound, mathematical equation than opinionated judgement from expert peers.

some of the most popular techniques include:

•Trend analysis : The idea here is that through the study of past sales data you can pick up on certain trends that with reason, could be used to predict similar fluctuations in the future. This could either be from seasonality, random factor analysis and economic demand.

•Exponential Smoothing : Probably considered the most accurate and widely used for an accurate sales forecasting process it makes an exponentially considered average of past sales to try and predict future revenue.

•Simple Moving Average : This Technique Requires The Sales Manager To Extrapolate Sales Data From A “Dynamic” Set Period Of Time; A Rolling Window Of Maybe 2,3 Or Maybe Even 6 Months.

So if you **have sales data available** research the pros and cons for each of these techniques to find one that best suits your business model.

3. **SALES FORECASTING TOOLS**

**\*using spreadsheets for sales forecasting :**

Organizations typically use spreadsheets for [sales forecasting](https://www.clari.com/blog/sales-forecasting-guide/) when they are smaller or don’t have enough resources to purchase a more sophisticated platform. Using spreadsheets is easy to share with the team; however, the spreadsheets become obsolete the moment data gets manually added in. Without an [automation tool to automatically add this data](https://www.clari.com/products/product-overview/) in real time, the forecasting approach is never truly accurate.

\* **using crm for sales forecasting :**

Using your crm for sales forecasting seems like a pretty logical plan. I mean it’s serving as your system of record to track and manage every opportunity, so it should be easy to develop a forecast, right? Organizations typically use crm reports for sales forecasting when manual spreadsheets become too laborious to upkeep and teams need a tool to aggregate those numbers.

\* **Using a Revenue Operations Platform for Sales Forecasting :**

With the emergence of AI and automation, new tools are available that make [sales forecasting](https://www.clari.com/blog/sales-forecasting-guide/) more connected, efficient and predictable. [Revenue operations platforms](https://www.clari.com/products/product-overview/) bring a new way to generate revenue with the same level of transparency and rigor that companies expect from any other mission-critical business process (ERP, supply chain, etc…).

4. **Techniques for Sales Forecasting**

**\* Top-Down Sales Forecasting Method :**

The top down sales forecasting technique starts by identifying your total addressable market or TAM for each business segment. It takes a higher-level approach to viewing your business. According to the [Corporate Finance Institute](https://corporatefinanceinstitute.com/resources/knowledge/modeling/top-down-forecasting/), your TAM is developed by researching market valuations from reputable sources, such as Gartner. You then estimate how much market share you’ll be able to capture and the revenue you’ll be able to acquire. Calculating revenue is done by multiplying the TAM by the percentage of market share you think you’ll be able to achieve

**\*Bottom-up Sales Forecasting Method :**

On the opposite spectrum is bottom-up sales forecasting, which starts with the reps instead of the managers and actual opportunities in play instead of models. In this sales forecast method, every rep calls their number based on the opportunities they have in pipeline. Managers work with reps to [inspect pipeline](https://www.clari.com/blog/5-data-driven-methods-for-inspecting-sales-pipeline/), understand projections and look through sales activity data in order to justify the forecast.

The benefits of a bottoms up sales forecast is that it’s based on real-world opportunities happening in real-time. Reps and managers who are in tune with each of their deals can provide a more accurate sales forecast based on the opportunities in play.

**\*Qualitative Sales Forecasting Method :**

Qualitative [sales forecasting](https://www.clari.com/blog/sales-forecasting-guide/) involves the estimation of sales performance based on long-time expertise or well-versed industry knowledge. According to [AccountingTools](https://www.accountingtools.com/articles/what-is-qualitative-forecasting.html), qualitative sales forecasting “...relies upon the knowledge of highly experienced employees and consultants to provide insights into future outcomes.”

**\*Quantitative Sales Forecasting Method :**

Quantitative forecasting model uses historic sales data to [calculate accurate forecasts](https://www.clari.com/blog/sales-forecasting-accuracy/). It’s based on past performance and can be done in two ways ([Chegg](https://www.chegg.com/homework-help/definitions/quantitative-forecasting-methods-31)). The first method is a [time-series model](https://www.clari.com/blog/time-series-sales-forecasting-predictable-revenue/) which looks for patterns in the data to build the forecast and predict where you’ll land based on current [sales pipeline coverage](https://www.clari.com/blog/pipeline-coverage-best-practices/). The second method is called the associative model and it uses assumptions to build a linear regression-based forecast. The linear regression shows where you’ll end up based on those assumptions.

Organizations choose to use quantitative forecasting because it’s the first step in being data-driven. However, a major drawback from using this method is that it is highly dependent on an accurate data set, analysis of historic performance and complete visibility into pipeline. With these insights in hand, it’s possible for organizations to land within 2% of their forecast, just weeks into the quarter.

5.VALIDATING CORRECT MODEL

You might find that you try out several different sales forecast methods before settling on one. Or you may find that you start out with one, but soon outgrow it. The important thing to remember here is that you have to look at where your business is today and where you want it to go.

And, unless you’re using a revenue operations platform like [Clari](https://www.clari.com/), the reality is that all of these methods typically require manual data collection and consistent, cooperative effort from the sales reps, as well as very accurate data that's inputted manually — which rarely happens in the real world.

Let Clari take the guessing game out of sales forecasting and set you up for success using real data-driven techniques.

Start developing forecasting as a real-time function that guides your business, instead of scrambling to consolidate spreadsheets, trying to make sense of disjointed reports and wasting time with manual data input.

Clari adds a whole new layer to [sales management](https://www.clari.com/solutions/teams/sales/) beyond what your CRM can provide. By leveraging the automated activity capture, as well as the AI-powered insights that let customers improve team productivity, you’ll be able to drive and convert more pipeline, forecast the business and reduce churn.

Validation is very vital in choosing whether the method or program we are using is right interms of proper execution.

**6.ACCURACY AND PRECISION**

Since accuracy, precision, and recall are numerical measurements, you can conveniently use them to **track the model quality over time**. The only major limitation is the need for **true labels** in production.

To quickly calculate and visualize accuracy, precision, and recall for your machine learning models, you can use Evidently, an [open-source Python library](https://github.com/evidentlyai/evidently) that helps evaluate, test, and monitor ML models in production.

You will need to prepare your dataset that includes predicted values for each class and true labels and pass it to the tool. You will instantly get an interactive report that includes a [confusion matrix](http://www.evidentlyai.com/classification-metrics/confusion-matrix), accuracy, precision, recall metrics, [ROC curve](https://www.evidentlyai.com/classification-metrics/explain-roc-curve) and other visualizations. You can also integrate these model quality checks into your production pipelines.

To improve the accuracy of a prediction model, we can:

1. Add more data
2. Treat missing and outlier values
3. Perform feature engineering
4. Perform feature selection
5. Try multiple algorithms
6. Tune hyperparameters
7. Use ensemble methods
8. Use cross validation

Likewise the from obtaining data to measure accuracy of the FUTURE SALES PREDICTION model is implemented in pyhton programming language with help of machine learning and

APPLIED DATA SCIENCE operations and technique

# \*phase3 - loading and preprocessing the

DATASET.

# importing required packages

import numpy as np  
import pandas as pd  
import seaborn as sns  
import matplotlib.pyplot as plt  
import sklearn as sk  
import seaborn as sns  
import warnings  
import scipy  
import numpy as np  
from sklearn.preprocessing import MinMaxScaler  
import seaborn as sns  
import matplotlib.pyplot as plt  
warnings.filterwarnings('ignore')

# loading dataset

sales=pd.read\_csv('Sales.csv')  
print(sales)

TV Radio Newspaper Sales  
0 230.1 37.8 69.2 22.1  
1 44.5 39.3 45.1 10.4  
2 17.2 45.9 69.3 12.0  
3 151.5 41.3 58.5 16.5  
4 180.8 10.8 58.4 17.9  
.. ... ... ... ...  
195 38.2 3.7 13.8 7.6  
196 94.2 4.9 8.1 14.0  
197 177.0 9.3 6.4 14.8  
198 283.6 42.0 66.2 25.5  
199 232.1 8.6 8.7 18.4  
  
[200 rows x 4 columns]

# reading dataset

sales.head(10)

TV Radio Newspaper Sales  
0 230.1 37.8 69.2 22.1  
1 44.5 39.3 45.1 10.4  
2 17.2 45.9 69.3 12.0  
3 151.5 41.3 58.5 16.5  
4 180.8 10.8 58.4 17.9  
5 8.7 48.9 75.0 7.2  
6 57.5 32.8 23.5 11.8  
7 120.2 19.6 11.6 13.2  
8 8.6 2.1 1.0 4.8  
9 199.8 2.6 21.2 15.6

sales.tail(10)

TV Radio Newspaper Sales  
190 39.5 41.1 5.8 10.8  
191 75.5 10.8 6.0 11.9  
192 17.2 4.1 31.6 5.9  
193 166.8 42.0 3.6 19.6  
194 149.7 35.6 6.0 17.3  
195 38.2 3.7 13.8 7.6  
196 94.2 4.9 8.1 14.0  
197 177.0 9.3 6.4 14.8  
198 283.6 42.0 66.2 25.5  
199 232.1 8.6 8.7 18.4

# statistical analysis

sales.describe()

TV Radio Newspaper Sales  
count 200.000000 200.000000 200.000000 200.000000  
mean 147.042500 23.264000 30.554000 15.130500  
std 85.854236 14.846809 21.778621 5.283892  
min 0.700000 0.000000 0.300000 1.600000  
25% 74.375000 9.975000 12.750000 11.000000  
50% 149.750000 22.900000 25.750000 16.000000  
75% 218.825000 36.525000 45.100000 19.050000  
max 296.400000 49.600000 114.000000 27.000000

sales.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 200 entries, 0 to 199  
Data columns (total 4 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 TV 200 non-null float64  
 1 Radio 200 non-null float64  
 2 Newspaper 200 non-null float64  
 3 Sales 200 non-null float64  
dtypes: float64(4)  
memory usage: 6.4 KB

sales.shape

(200, 4)

# DATASET PREPROCESSING

# finding null values

sales.isnull()

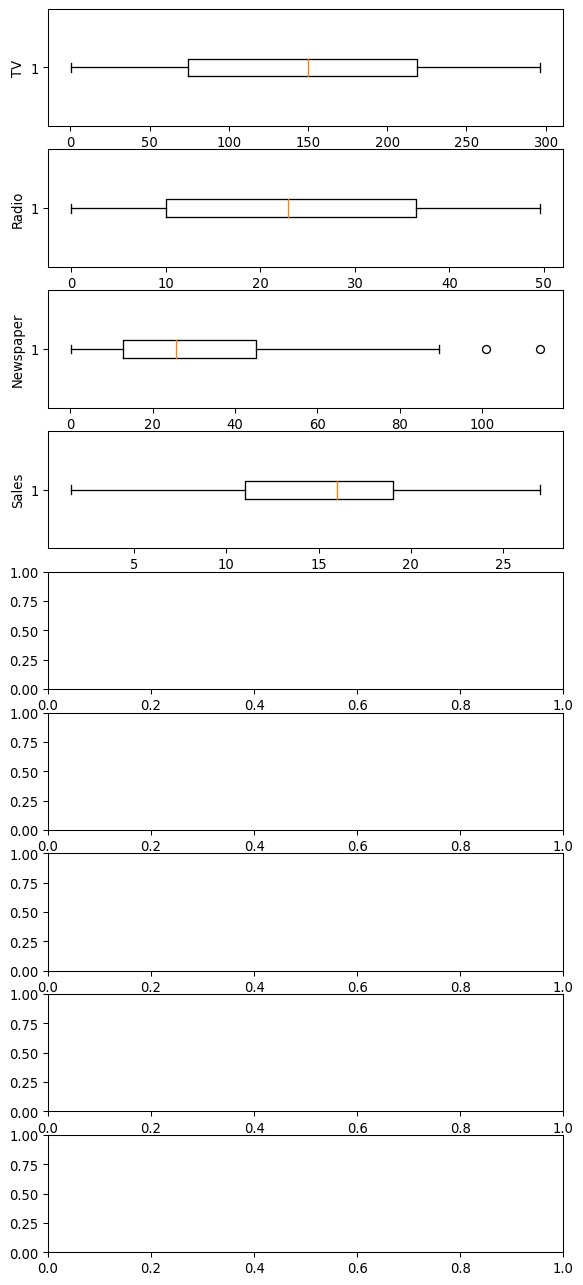
TV Radio Newspaper Sales  
0 False False False False  
1 False False False False  
2 False False False False  
3 False False False False  
4 False False False False  
.. ... ... ... ...  
195 False False False False  
196 False False False False  
197 False False False False  
198 False False False False  
199 False False False False  
  
[200 rows x 4 columns]

sales.isna().sum()

TV 0  
Radio 0  
Newspaper 0  
Sales 0  
dtype: int64

# checking for outliers

fig, axs = plt.subplots(9,1,dpi=95, figsize=(7,17))  
i = 0  
for col in sales.columns:  
 axs[i].boxplot(sales[col], vert=False)  
 axs[i].set\_ylabel(col)  
 i+=1  
plt.show()

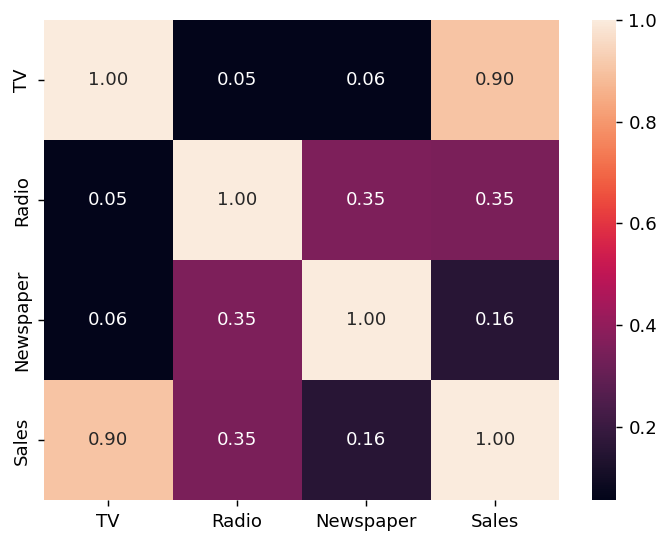


# drop the outliers

q1, q3 = np.percentile(sales['TV'], [25, 75])  
iqr = q3 - q1  
lower\_bound = q1 - (1.5 \* iqr)  
upper\_bound = q3 + (1.5 \* iqr)  
  
# Drop the outliers  
clean\_data = sales[(sales['TV'] >= lower\_bound)   
 & (sales['TV'] <= upper\_bound)]  
  
q1, q3 = np.percentile(clean\_data['Radio'], [25, 75])  
iqr = q3 - q1  
lower\_bound = q1 - (1.5 \* iqr)  
upper\_bound = q3 + (1.5 \* iqr)  
  
# Drop the outliers  
clean\_data = clean\_data[(clean\_data['Radio'] >= lower\_bound)   
 & (clean\_data['Radio'] <= upper\_bound)]  
q1, q3 = np.percentile(clean\_data['Newspaper'], [25, 75])  
iqr = q3 - q1  
lower\_bound = q1 - (1.5 \* iqr)  
upper\_bound = q3 + (1.5 \* iqr)  
  
# Drop the outliers  
clean\_data = clean\_data[(clean\_data['Newspaper'] >= lower\_bound)   
 & (clean\_data['Newspaper'] <= upper\_bound)]  
  
q1, q3 = np.percentile(clean\_data['Sales'], [25, 75])  
iqr = q3 - q1  
lower\_bound = q1 - (1.5 \* iqr)  
upper\_bound = q3 + (1.5 \* iqr)  
  
# Drop the outliers  
clean\_data = clean\_data[(clean\_data['Sales'] >= lower\_bound)   
 & (clean\_data['Sales'] <= upper\_bound)]

# finding correlation

corr = sales.corr()  
   
plt.figure(dpi=130)  
sns.heatmap(sales.corr(), annot=True, fmt= '.2f')  
plt.show()



\*PHASE4 – FEATUREENGINEERING , MODEL TRAINING

& EVALUATING THE DATAMODEL

# \*importing required packages

import numpy as np  
import pandas as pd  
import seaborn as sns  
import matplotlib.pyplot as plt  
import sklearn as sk  
import seaborn as sns  
import warnings  
import scipy  
import numpy as np  
from sklearn.preprocessing import MinMaxScaler  
import seaborn as sns  
import matplotlib.pyplot as plt  
warnings.filterwarnings('ignore')

\* reading dataset

sales=pd.read\_csv('Sales.csv')  
print(sales)

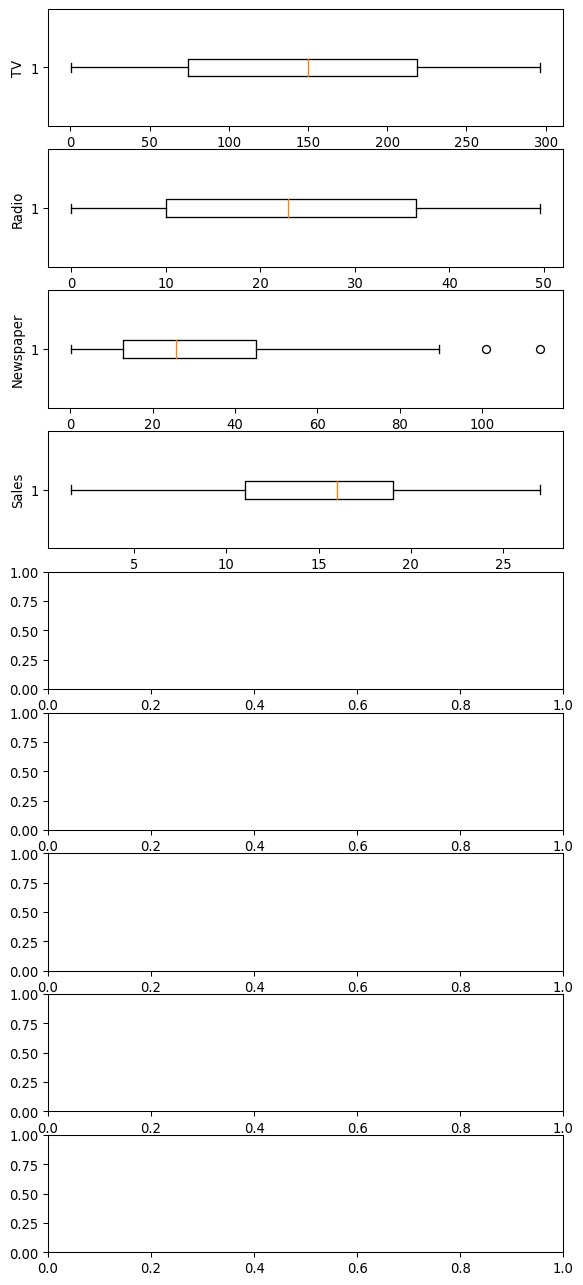
TV Radio Newspaper Sales  
0 230.1 37.8 69.2 22.1  
1 44.5 39.3 45.1 10.4  
2 17.2 45.9 69.3 12.0  
3 151.5 41.3 58.5 16.5  
4 180.8 10.8 58.4 17.9  
.. ... ... ... ...  
195 38.2 3.7 13.8 7.6  
196 94.2 4.9 8.1 14.0  
197 177.0 9.3 6.4 14.8  
198 283.6 42.0 66.2 25.5  
199 232.1 8.6 8.7 18.4  
  
[200 rows x 4 columns]

# 

# 1.FEATURE ENGINEERING

# checking for outliers

fig, axs = plt.subplots(9,1,dpi=95, figsize=(7,17))  
i = 0  
for col in sales.columns:  
 axs[i].boxplot(sales[col], vert=False)  
 axs[i].set\_ylabel(col)  
 i+=1  
plt.show()



# drop the outliers

q1, q3 = np.percentile(sales['TV'], [25, 75])  
iqr = q3 - q1  
lower\_bound = q1 - (1.5 \* iqr)  
upper\_bound = q3 + (1.5 \* iqr)  
  
# Drop the outliers  
clean\_data = sales[(sales['TV'] >= lower\_bound)   
 & (sales['TV'] <= upper\_bound)]  
  
q1, q3 = np.percentile(clean\_data['Radio'], [25, 75])  
iqr = q3 - q1  
lower\_bound = q1 - (1.5 \* iqr)  
upper\_bound = q3 + (1.5 \* iqr)  
  
# Drop the outliers  
clean\_data = clean\_data[(clean\_data['Radio'] >= lower\_bound)   
 & (clean\_data['Radio'] <= upper\_bound)]  
q1, q3 = np.percentile(clean\_data['Newspaper'], [25, 75])  
iqr = q3 - q1  
lower\_bound = q1 - (1.5 \* iqr)  
upper\_bound = q3 + (1.5 \* iqr)  
  
# Drop the outliers  
clean\_data = clean\_data[(clean\_data['Newspaper'] >= lower\_bound)   
 & (clean\_data['Newspaper'] <= upper\_bound)]  
  
q1, q3 = np.percentile(clean\_data['Sales'], [25, 75])  
iqr = q3 - q1  
lower\_bound = q1 - (1.5 \* iqr)  
upper\_bound = q3 + (1.5 \* iqr)  
  
# Drop the outliers  
clean\_data = clean\_data[(clean\_data['Sales'] >= lower\_bound)   
 & (clean\_data['Sales'] <= upper\_bound)]

# 

# 2.TRAINING THE DATASET

x = sales['TV']  
y = sales['Sales']

from sklearn.model\_selection import train\_test\_split  
# Split the data into training and test sets  
x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3,  
 random\_state=42)

from sklearn.model\_selection import train\_test\_split  
x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, train\_size = 0.7,  
 test\_size = 0.3, random\_state = 100)

x\_train.head()

74 213.4  
3 151.5  
185 205.0  
26 142.9  
90 134.3  
Name: TV, dtype: float64

y\_train.head()

74 17.0  
3 16.5  
185 22.6  
26 15.0  
90 14.0  
Name: Sales, dtype: float64

import statsmodels.api as sm  
x\_train\_sm = sm.add\_constant(x\_train)  
lr = sm.OLS(y\_train, x\_train\_sm).fit()  
lr.params

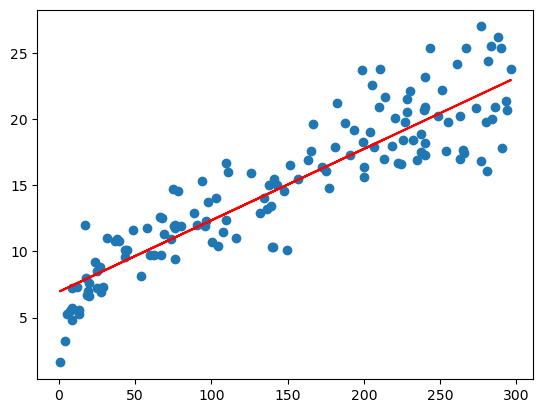
const 6.948683  
TV 0.054546  
dtype: float64

print(lr.summary())

OLS Regression Results   
==============================================================================  
Dep. Variable: Sales R-squared: 0.816  
Model: OLS Adj. R-squared: 0.814  
Method: Least Squares F-statistic: 611.2  
Date: Wed, 25 Oct 2023 Prob (F-statistic): 1.52e-52  
Time: 21:32:11 Log-Likelihood: -321.12  
No. Observations: 140 AIC: 646.2  
Df Residuals: 138 BIC: 652.1  
Df Model: 1   
Covariance Type: nonrobust   
==============================================================================  
 coef std err t P>|t| [0.025 0.975]  
------------------------------------------------------------------------------  
const 6.9487 0.385 18.068 0.000 6.188 7.709  
TV 0.0545 0.002 24.722 0.000 0.050 0.059  
==============================================================================  
Omnibus: 0.027 Durbin-Watson: 2.196  
Prob(Omnibus): 0.987 Jarque-Bera (JB): 0.150  
Skew: -0.006 Prob(JB): 0.928  
Kurtosis: 2.840 Cond. No. 328.  
==============================================================================

# \*plot for the dataset training

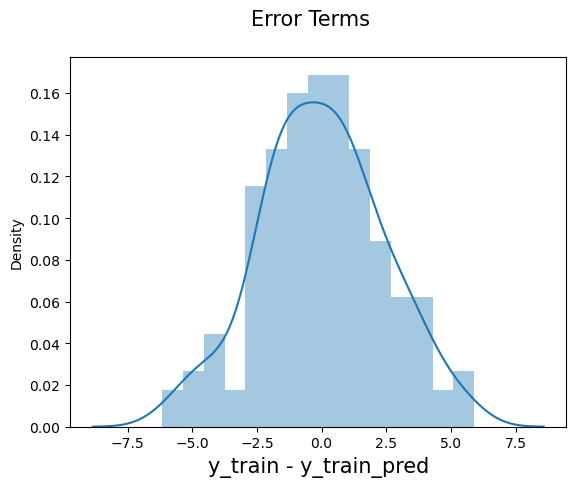
plt.scatter(x\_train, y\_train)  
plt.plot(x\_train, 6.948 + 0.054\*x\_train, 'r')  
plt.show()

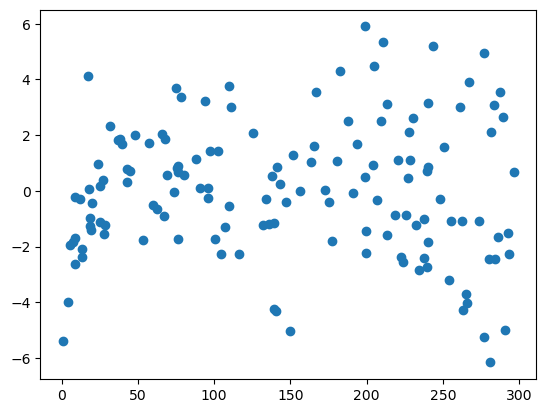


# 4.EVALUATION OF DATAMODEL

y\_train\_pred = lr.predict(x\_train\_sm)  
res = (y\_train - y\_train\_pred)

fig = plt.figure()  
sns.distplot(res, bins = 15)  
fig.suptitle('Error Terms', fontsize = 15)  
plt.xlabel('y\_train - y\_train\_pred', fontsize = 15)  
plt.show()  
plt.scatter(x\_train,res)  
plt.show()





x\_test\_sm = sm.add\_constant(x\_test)  
y\_pred = lr.predict(x\_test\_sm)  
y\_pred.head()

126 7.374140  
104 19.941482  
99 14.323269  
92 18.823294  
111 20.132392  
dtype: float64

# \*mean squared error

from sklearn.metrics import mean\_squared\_error  
from sklearn.metrics import r2\_score  
np.sqrt(mean\_squared\_error(y\_test, y\_pred))

2.019296008966232

# \* R squared evaluation

r\_squared = r2\_score(y\_test, y\_pred)  
r\_squared

0.7921031601245659

plt.scatter(x\_test, y\_test)  
plt.plot(x\_test, 6.948 + 0.054 \* x\_test, 'r')  
plt.show()

