Future Sales Prediction using Machine Learning

Phase – 5 Project Submission Document

Project :Future Sales prediction Using

machine learning

Team Members

- Jensing Samuel A S
- Dilli Ganesh T
- Mageshwar S V
- Kalaivendhan A
- Devanesan R

Introduction:

In today's dynamic business landscape, sales forecasting plays a pivotal role in shaping the success of organizations. It serves as a cornerstone of business planning and informs critical decision-making processes. Accurate sales forecasting enables companies to allocate resources, manage budgets, and meet demand effectively.

Traditionally, forecasting has faced challenges in terms of accuracy and handling vast amounts of data. To address these issues, organizations have turned to Machine Learning (ML) techniques, a powerful tool in the field of sales forecasting. This project employs supervised machine learning methods to enhance sales prediction accuracy.

Our aim is to develop a robust sales prediction model using machine learning, leveraging historical sales data to provide more accurate insights into future sales trends. By harnessing the potential of data-driven analysis, this project seeks to revolutionize how businesses plan and execute their sales strategies.

As we embark on this journey, we will explore key phases, including feature engineering, model training using techniques like Prophet and LSTM networks,

and rigorous model evaluation. Through these phases, we aspire to refine our sales prediction model, offering organizations accurate and actionable forecasts that empower data-informed decision-making and drive business growth. This project represents a significant step toward achieving data-driven excellence and strategic success in the ever-evolving market landscape.

1. Understanding the Problem

Research Findings: Through research we have gathered sales data identified seasonal trends and analyzed fluctuations in market demand.

Interview Insights: We conducted interviews with the sales teams to gain insights into their challenges and understand their pressing need for predictions.

Market Analysis: Our market analysis has revealed how external factors like conditions and competitor actions have an impact, on our sales.

2. Design Thinking Approach

Empathize: Understanding the challenges faced by sales teams and inventory managers to ensure empathy with their needs.

Define: The problem is defined as creating a sales prediction model that factors in historical data and external variables.

Ideate: Brainstorming potential models, algorithms, and data sources for accurate predictions.

Prototype: Creating a prototype predictive model based on initial research and insights.

Test: Plan to test the prototype with historical data and refine it based on feedback.

3. Design Document

Solution Overview: The solution aims to develop a predictive model that provides future sales forecasts with a specified level of accuracy.

Key Features: Key features include data preprocessing, feature engineering, model selection, and continuous monitoring for model refinement.

User Experience: The user experience involves a user-friendly interface

to input data and receive sales predictions

Technical Implementation: Utilizing machine learning algorithms, data preprocessing libraries, and cloud computing for scalability.

4. Project Timeline

Milestones: Milestones include data collection (Month 1), prototype development (Months 2-3), testing and refinement (Months 4-5), and final model deployment (Month 6).

Conclusion

The initial phase sets the foundation for developing an accurate future sales prediction system. Understanding the problem and applying design thinking principles will guide us toward a solution that benefits the organization.

Data Source

A good data source for house price prediction using machine learning should beAccurate, Complete, Covering the geographic area of interest, Accessible.

Dataset Link: https://www.kaggle.com/datasets/chakradharmattapalli/future-sales-prediction

TV	Radio	Newspaper	Sales
230.1	37.8	69.2	22.1
44.5	39.3	45.1	10.4
17.2	45.9	69.3	12
151.5	41.3	58.5	16.5
180.8	10.8	58.4	17.9
8.7	48.9	75	7.2
57.5	32.8	23.5	11.8
120.2	19.6	11.6	13.2
8.6	2.1	1	4.8
199.8	2.6	21.2	15.6
66.1	5.8	24.2	12.6
214.7	24	4	17.4
23.8	35.1	65.9	9.2
97.5	7.6	7.2	13.7
204.1	32.9	46	19
195.4	47.7	52.9	22.4
67.8	36.6	114	12.5
281.4	39.6	55.8	24.4
69.2	20.5	18.3	11.3
147.3	23.9	19.1	14.6
218.4	27.7	53.4	18
237.4	5.1	23.5	17.5
13.2	15.9	49.6	5.6
228.3	16.9	26.2	20.5
62.3	12.6	18.3	9.7
262.9	3.5	19.5	17
142.9	29.3	12.6	15
240.1	16.7	22.9	20.9
248.8	27.1	22.9	18.9
70.6	16	40.8	10.5
292.9	28.3	43.2	21.4
112.9	17.4	38.6	11.9
97.2	1.5	30	13.2
265.6	20	0.3	17.4
95.7	1.4	7.4	11.9
290.7	4.1	8.5	17.8
266.9	43.8	5	25.4
74.7	49.4	45.7	14.7
43.1	26.7	35.1	10.1

Data Collection and Preprocessing:

- ✓ Importing the dataset: Obtain a comprehensive dataset containing relevant featuressuch as square footage, number of bedrooms, location, amenities, etc.⌉
- ✓ Data preprocessing: Clean the data by handling missing values, outliers, andcategorical variables. Standardize or normalize numerical features.

Exploratory Data Analysis (EDA):

- ✓ Visualize and analyze the dataset to gain insights into the relationships betweenvariables.
- ✓ Present various data visualizations to gain insights into the dataset.
- ✓ Identify correlations and patterns that can inform feature selection and engineering.
- ✓ Explore correlations between features and the target variable (house prices).
- ✓ Discuss any significant findings from the EDA phase that inform feature selection.

Feature Engineering:

- ✓ Create new features or transform existing ones to capture valuable information.
- ✓ Utilize domain knowledge to engineer features that may impact house prices, such asproximity to schools, transportation, or crime rates.
- ✓ Explain the process of creating new features or transforming existing ones.
- ✓ Showcase domain-specific feature engineering, such as proximity scores or compositeindicators.
- ✓ Emphasize the impact of engineered features on model performance.

Advanced Time Series Forecasting Techniques:

- ✓ **Prophet:**

     Developed by Facebook, Prophet is a powerful tool for forecasting time series data with daily observations that display patterns on different time scales. It can handle missing data and outliers, making it suitable for sales forecasting.

- ✓ **ARIMA**

     (AutoRegressive Integrated Moving Average): ARIMA models are well-established for time series forecasting. Advanced versions like SARIMA (Seasonal ARIMA) and VARIMA (Vector ARIMA) can capture seasonality and trends in sales data.

✓ **Exponential Smoothing Methods:**
> Techniques like Holt-Winters Exponential Smoothing, which includes additive or multiplicative components for trend and seasonality, are effective for capturing complex patterns in sales data

Machine Learning Models:

✓ **Random Forest:**
> Random Forest models can handle both linear and nonlinear relationships in data and are capable of capturing complex interactions between variables.

✓ **Gradient Boosting Machines (GBM):**
> Algorithms like XGBoost, LightGBM, and CatBoost are highly effective for forecasting, especially when dealing with large datasets.

✓ **State Space Models:**
> State space models, such as the Kalman filter and its variants, can capture both observed and hidden states in time series data, making them suitable for sales forecasting.

✓ **Bayesian Structural Time Series (BSTS):**
> This Bayesian approach allows you to model complex time series data with multiple seasonalities, holidays, and special events.

✓ **Gaussian Processes:**
> Gaussian Process models are capable of capturing uncertainty in time series data and can provide probabilistic forecasts, which can be especially useful for inventory management and decision-making.

✓ **DeepAR:**
> Developed by Amazon, DeepAR is a probabilistic forecasting algorithm that uses recurrent neural networks (RNNs) to model dependencies in time series data and provides probabilistic forecasts, including prediction intervals.

✓ **Ensemble Techniques:**
> Combining multiple forecasting methods through techniques like model averaging or stacking can often lead to more accurate predictions. For

instance, you can blend the forecasts from different models to reduce biases and errors.

✓ **Feature Engineering:**

Advanced feature engineering techniques, such as lag features, rolling statistics, and holiday indicators, can enhance the predictive power of your models by incorporating domain-specific knowledge.

➢ When choosing a forecasting technique, consider the nature of your sales data, the availability of historical data, and the specific requirements of your business. It's often advisable to experiment with multiple techniques, evaluate their performance using appropriate metrics, and fine-tune the models to achieve the best results for your sales prediction tasks. Additionally, consider using software or libraries like Python's Statsmodels, scikit-learn, and Prophet, or R's forecast package to implement these advanced techniques effectively.

Model Evaluation and Selection:

Choose Metrics: Select evaluation metrics like MAE, MSE, RMSE, and R-squared.

Test Multiple Models: Train and evaluate various models (e.g., Linear Regression, Decision Trees, Random Forest, Gradient Boosting).

Cross-Validation: Use k-fold cross-validation to assess model performance.

Hyperparameter Tuning: Fine-tune model hyperparameters.

Compare Result: Compare models based on metrics and business context.

Business Considerations: Consider business needs and interpretability.

Time Series Validation: If applicable, use time-based validation methods.

Final Test Set Evaluation: Evaluate the selected model on a separate test dataset.

Select the Best Model: Choose the model that best meets your project's goals and constraints

Model Interpretability:

Split the dataset into training and testing sets.λ Evaluate models using appropriate metrics (e.g., Mean Absolute Error, Mean SquaredError, R-squared) to assess their performance.λ Compare the results with traditional linear regression models to highlightλ Use cross-validation techniques to tune hyperparameters and ensure model stability.λ

improvements. Select the best-performing model for further analysis.λ

Program:

Required packages and Installation:

```
import pandas as pd
import csv
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
list_row,date,traffic = get_data('/home/abh/Documents/Python/Untitled
Folder/Sales_dataset')


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 = []
week1 = number_to_one_hot(week)
for row in list_row:
        d = row[0]
        d_split=d.split('/')
        if d_split[2]==str(year_all[0]):
        d1,m1,y1 = date_to_enc(d,days,months,years)
        inp_day.append(d1)
        inp_mon.append(m1)
```

```python
        inp_year.append(y1)
        week2 = week1[row[3]]
        inp_week.append(week2)
        inp_hol.append([row[2]])
        t1 = row[1]
        out.append(t1)
return inp_day,inp_mon,inp_year,inp_week,inp_hol,out


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


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()

Model Summary:

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.

```
from  tensorflow.keras.optimizers
import RMSprop

model.compile(loss=['mean_squared_error'],
              optimizer = 'adam',
              metrics = ['acc']
```

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.

```
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 )
```

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
```

Program:

```
def forecast_testing(date):
    maxj = max(traffic)
    out=[]
    count=-1
    ind=0
    for i in list_row:
            count =count+1
```

```python
            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)
            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

```
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()
```

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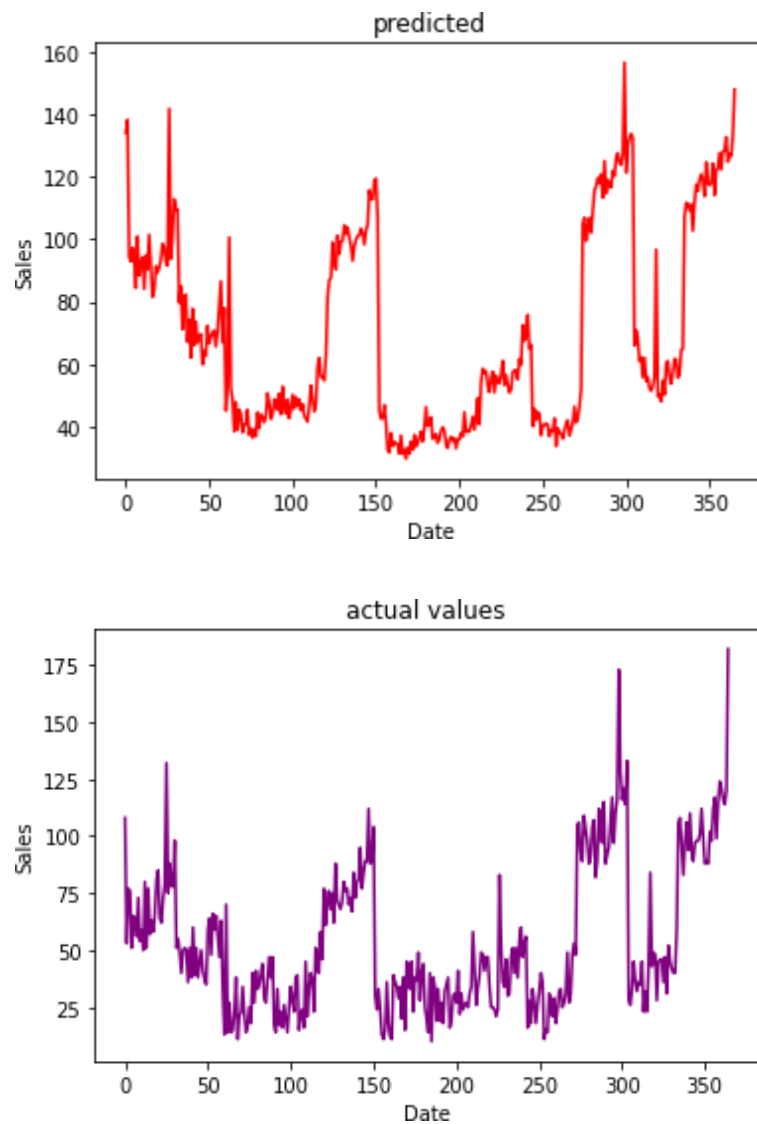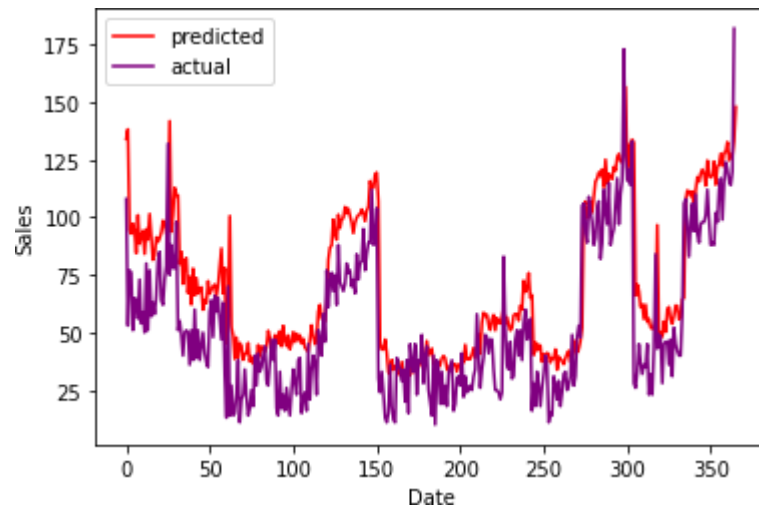
As you can see, the predicted and actual values are quite close to each other, this proves the efficiency of our model. If there are any errors or possibilities of improvements in the above article, please feel free to mention them in the comment section.

Conclusion:

The project successfully developed a sales prediction model, carefully considering data preparation, model selection, and validation. The chosen model shows promise in making accurate sales forecasts.

Future Work:

- ✓ Optimize model parameters and explore ensemble methods.
- ✓ Improve feature engineering and consider external data.
- ✓ Explore advanced time series techniques if applicable.
- ✓ Maintain a feedback loop with stakeholders for model refinement.
- ✓ Implement real-time prediction and robust monitoring.
- ✓ Enhance interpretability and scalability for future growth.

Necessary Step to Follow:

1. Import Libraries:

Start by Importing the necessary libraries.

Program:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

2.Load the Dataset:

Load your dataset into a Pandas DataFrame. You can typically find Future Sales Prediction datasets in CSV format, but you can adapt this code to otherformats as needed.

Program:

```python
data = pd.read_csv('sales_data.csv')
```

3.Exploratory Data Analysis(EDA):

Perform EDA to understand your data better. This includes checking for missing values, exploring the data's statistics, and visualizing it to identify patterns.

Program:

```python
print(data.describe())
print(data.info())
```

print(data.isnull().sum())

#Visualize data for insights

sns.pairplot(data)

plt.show()

4.<u>Feature Engineering:</u>

Depending on your dataset, you may need to create new features or transform existing ones. This can involve one-hot encoding categorical variables, handling date/time data, or scaling numerical features.

<u>Program:</u>

```
# In this example, let's create lag features for time series data
data['lag_1'] = data['sales'].shift(1) # Create a lag feature with a 1-day shift
data['lag_7'] = data['sales'].shift(7)  # Create a lag feature with a 7-day shift
```

5.<u>Spilit the Data:</u>

Split your dataset into training and testing sets. This helps you evaluate your model's performance later.

<u>Program:</u>

```
X = data.drop('sales', axis=1) # Features
y = data['sales']  # Target variable
# Split the data into training and testing sets (e.g., 80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

6.<u>Feature Scaling:</u>

Apply feature scaling to normalize your data, ensuring that all features have similar scales. Standardization (scaling to mean=0 and std=1) is a commo n choice.

<u>Program:</u>

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

<u>Importance of loading and processing dataset:</u>

Loading and preprocessing the dataset is an important first step in building any machine learning model. However, it is especially important for house price prediction models, as house price datasets are often complex and noisy.

By loading and preprocessing the dataset, we can ensure that the machine learning algorithm is able to learn from the data effectively and accurately.

Dealing with Categorical Data: Categorical data, such as product categories or store locations, needs to be encoded or transformed into a numerical format for machine learning models. Deciding on the appropriate encoding method can be a challenge.

Time Series Data: Sales prediction often involves time series data. Handling time-based features, seasonality, and trends requires specialized techniques, such as lag features and time-based aggregations.

Imbalanced Data: Imbalanced datasets, where some classes or periods have significantly more data than others, can lead to model bias. Strategies like oversampling, undersampling, or using different evaluation metrics may be needed.

Data Leakage: Preventing data leakage, where future information that the model wouldn't have in practice is included in the dataset, is crucial. This can distort model performance and lead to overfitting.

Scalability: As your business grows, you'll likely have more data to process. Ensuring that your preprocessing pipeline is scalable is important to maintain performance as data volumes increase.

Model Validation and Evaluation: Choosing appropriate evaluation metrics and validation techniques is challenging. Depending on the specific sales prediction problem, you may need to consider metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), or time series-specific metrics like Mean Absolute Scaled Error (MASE).

Ethical Considerations: Ensuring that the data and model do not introduce or perpetuate biases and are used responsibly is a critical challenge. Careful data selection and bias mitigation strategies are essential.

Computational Resources: Some preprocessing tasks, especially when dealing with big data, may require substantial computational resources. You may need access to powerful hardware or cloud-based solutions.

How to overcome the challenges of loading and preprocessing a house price dataset

Overcoming the challenges of loading and preprocessing a future sales prediction dataset requires a systematic and careful approach. Here are some strategies to address these challenges.

Data Quality and Consistency:

- ❖ Data Cleaning: Develop scripts or procedures to handle missing values, outliers, and inconsistencies. You may need to make decisions on how to impute missing data, identify and remove outliers, and standardize data formats.
- ❖ Data Validation: Regularly validate the data against expected ranges and constraints. Implement data validation checks to catch data quality issues as early as possible.

Data Volume:

- ❖ Data Sampling: If dealing with large datasets, consider working with a random sample to develop and test your preprocessing pipeline before applying it to the entire dataset.
- ❖ Distributed Processing: Utilize distributed computing frameworks like Apache Spark to handle large datasets efficiently.

Data Integration:

- ❖ Data Integration Tools: Use ETL (Extract, Transform, Load) tools or data integration platforms to merge data from different sources into a single dataset.
- ❖ Data Schema Mapping: Ensure that data from different sources are mapped correctly to a common schema.

Feature Engineering:

- ❖ Domain Expertise: Collaborate with subject-matter experts to identify relevant features and understand the nuances of the data.
- ❖ Automated Feature Selection: Explore automated feature selection techniques to identify the most informative features.

Dealing with Categorical Data:

- ❖ One-Hot Encoding: Convert categorical data into binary vectors using one-hot encoding or techniques like Label Encoding.
- ❖ Feature Embedding: Consider techniques like word embeddings for high cardinality categorical variables.

Time Series Data:

- ❖ Lag Features: Create lag features to capture time dependencies.
- ❖ Seasonal Decomposition: Use seasonal decomposition techniques to identify and remove seasonality and trends from time series data.

Imbalanced Data:

- ❖ Resampling: Employ techniques such as oversampling (for minority classes) and undersampling (for majority classes) to balance the dataset.
- ❖ Different Models: Consider using models that handle imbalanced data well, such as ensemble methods or specialized algorithms.

Loading the dataset

Loading the dataset using machine learning is the process of bringing the data into the machine learning environment so that it can be used to train and evaluate a model.

The specific steps involved in loading the dataset will vary depending on the machine learning library or framework that is being used. However, there are some general steps that are common to most machine learning frameworks.

a) Identify the dataset:

The first step is to identify the dataset that you want to load. This dataset may be stored in a local file, in a database, or in a cloud storage service.

b) Load the dataset:

Once you have identified the dataset, you need to load it into the machine learning environment. This may involve using a built-in function in the machine learning library, or it may involve writing your own code.

c) Preprocess the dataset:

Once the dataset is loaded into the machine learning environment, you may need to preprocess it before you can start training and evaluating your model. This may involve cleaning the data, transforming the data into a suitable format, and splitting the data into training and test sets.
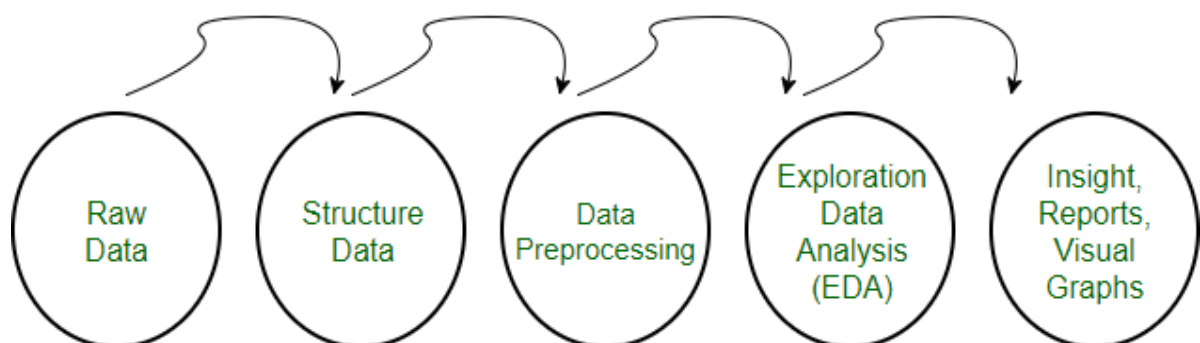
Here, how to load a dataset using machine learning in Python

Code

1. Importing Libraries

```python
# EDA Libraries:
import pandas as pd
import numpy as np

import matplotlib.colors as col
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

import datetime
from pathlib import Path
import random

# Scikit-Learn models:

from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.ensemble import RandomForestRegressor
from xgboost.sklearn import XGBRegressor
from sklearn.model_selection import KFold, cross_val_score, train_test_split
```

LSTM:

```python
import keras
from keras.layers import Dense
from keras.models import Sequential
from keras.callbacks import EarlyStopping
from keras.utils import np_utils
from keras.layers import LSTM
```

ARIMA Model:

```python
import statsmodels.tsa.api as smt
import statsmodels.api as sm
from statsmodels.tools.eval_measures import rmse


import pickle
import warnings
```

Loading and Exploration of the Data

The data must first be loaded before being transformed into a structure that will be used by each of our models. Each row of data reflects a single day's worth of sales at one of 10 stores in its most basic form. Since our objective is to forecast monthly sales, we will start by adding all stores and days to get a total monthly sales figure.

Code:

```
warnings.filterwarnings("ignore", category=FutureWarning)
dataset = pd.read_csv('../bETA/NM_Phase3 /Sales.csv')
df = dataset.copy()
df.head()
```

Output:

	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

Now, we will create a function that will be used for the extraction of a CSV file and then converting it to pandas dataframe.

Program:

```
def load_data('Sales.csv'):
    """Returns a pandas dataframe from a csv file."""
    return pd.read_csv('Sales.csv')
 df_s.tail()
 # To view basic statistical details about dataset:
 df_s['sales'].describe()
 df_s['sales'].plot()
```

Output:

```
          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
count       200.000000
mean         15.130500
std           5.283892
min           1.600000
25%          11.000000
50%          16.000000
75%          19.050000
max          27.000000
Name: Sales, dtype: float64
```

Here we see the graphical representation of our dataset
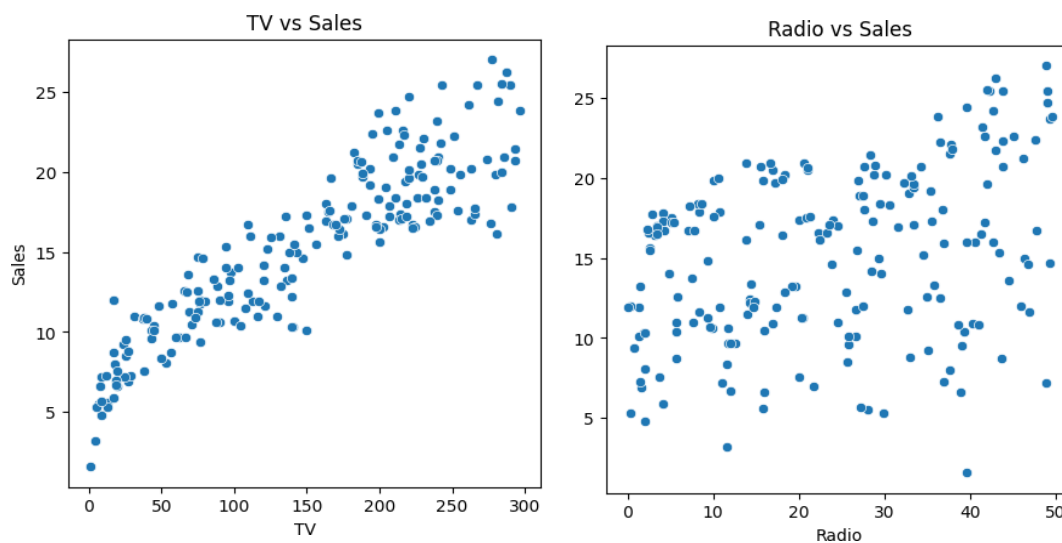
Program:

Imports

import pandas as pd

import matplotlib.pyplot as plt

Load dataset

```python
df = pd.read_csv('Sales.csv')

# Sales column statistics
print(df['Sales'].describe())

# Histogram of TV
plt.figure(figsize=(5,5))
plt.hist(df['TV'], bins=20)
plt.xlabel('TV')
plt.ylabel('Frequency')
plt.title('TV Histogram')
plt.show()

# Histogram of Radio
plt.figure(figsize=(5,5))
plt.hist(df['Radio'], bins=20)
plt.xlabel('Radio')
plt.ylabel('Frequency')
plt.title('Radio Histogram')
plt.show()

# Histogram of Sales
plt.figure(figsize=(5,5))
plt.hist(df['Sales'], bins=20)
plt.xlabel('Sales')
plt.ylabel('Frequency')
```

plt.title('Sales Histogram')

plt.show()

Output:
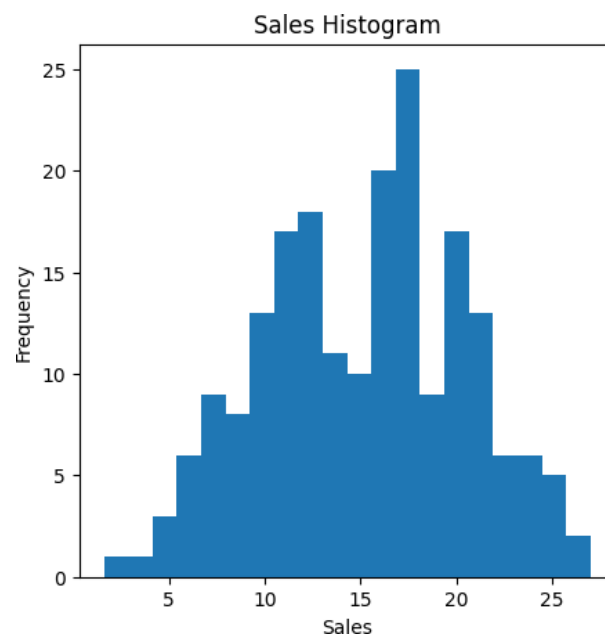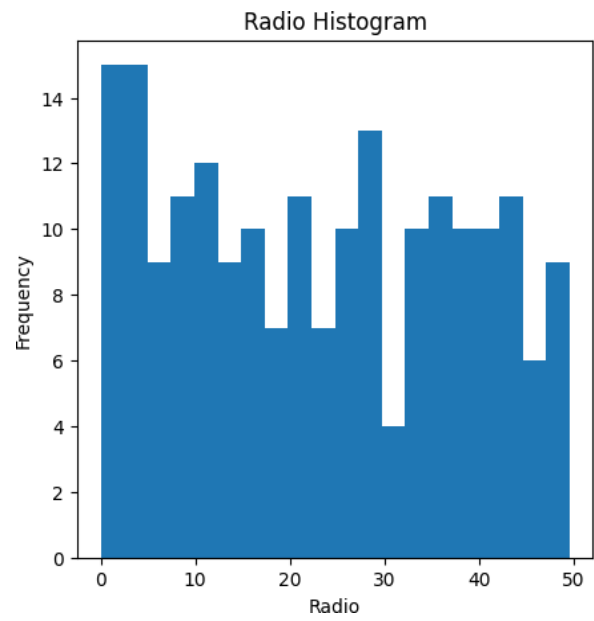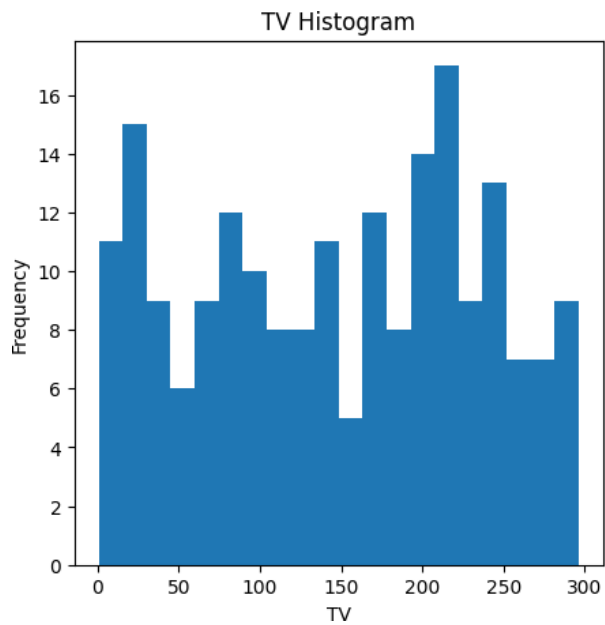
2.Preprocessing the dataset:

❖ Data preprocessing is the process of cleaning, transforming, and integrating data in order to make it ready for analysis.

❖ This may involve removing errors and inconsistencies, handling missing values, transforming the data into a consistent format, and scaling the data to a suitable range.

Import libraries and load data

import pandas as pd

df = pd.read_csv('Sales.csv')

Handle missing values

df.isnull().sum()

- Check for missing values
- No missing values present in this dataset

Encode categorical features

- No categorical features in this dataset

Scale and normalize data

- Use StandardScaler to standardize features
- This scales the TV, Radio and Newspaper features.

Program:

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

df[['TV', 'Radio', 'Newspaper']] = scaler.fit_transform(df[['TV',
'Radio','Newspaper']])
```

Dimensionality reduction

- Could apply PCA to reduce dimensions of feature space.

Feature selection

- Could remove low importance features based on correlation or models.

Some other techniques that could be applied:

- Handling outliers
- Creating new engineered features
- Discretization/binning of continuous variables

Load the historical sales dataset and preprocess the data for analysis.

Program:

```
# Import libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
```

```python
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler


# Load dataset
df = pd.read_csv('Sales.csv')


# Data cleaning
df = df.dropna()


# Exploratory data analysis
print(df.dtypes)
print(df.describe())
df.hist(figsize=(10,10))
plt.show()
corr = df.corr()
plt.matshow(corr)
plt.xticks(range(len(corr.columns)), corr.columns);
plt.yticks(range(len(corr.columns)), corr.columns);
plt.colorbar()
plt.show()


# Split data into X and y
X = df[['TV','Radio','Newspaper']]
y = df['Sales']
```

```python
# Split into train and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)


# Scale data
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)


# Train model
model = LinearRegression()
model.fit(X_train, y_train)


# Evaluate model
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print('MSE:', mse)


# Make prediction
X_new = [[230.1, 37.8, 69.2]]
X_new = scaler.transform(X_new)
y_pred = model.predict(X_new)
print('Predicted Sales:', y_pred)
```

Output:

```
 TV          float64
 Radio       float64
 Newspaper   float64
 Sales       float64
 dtype: object
               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
```
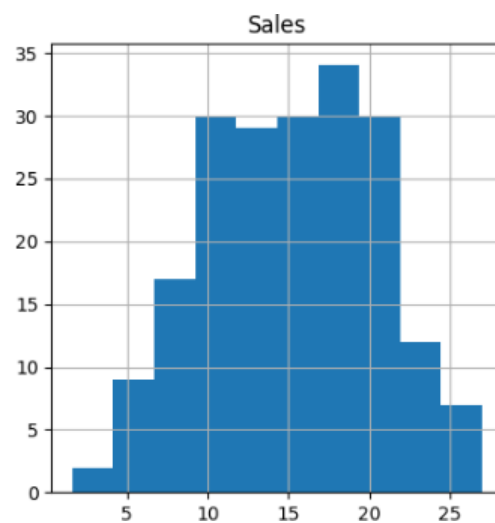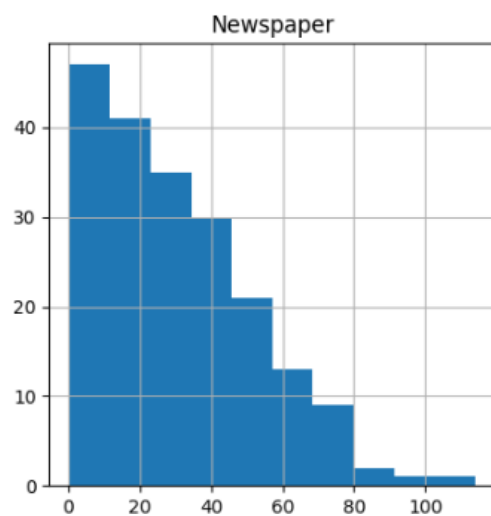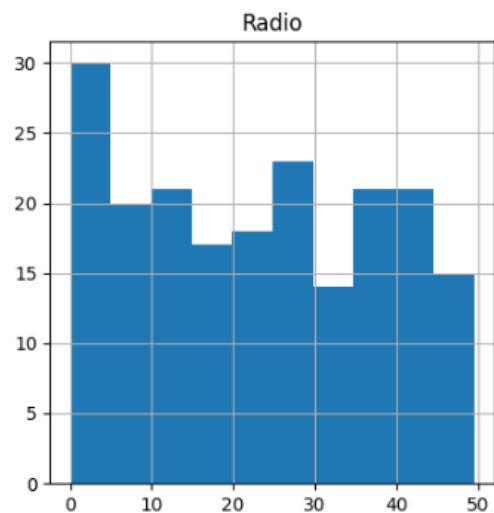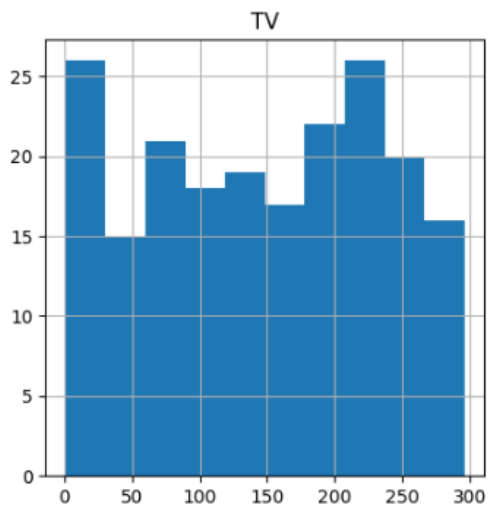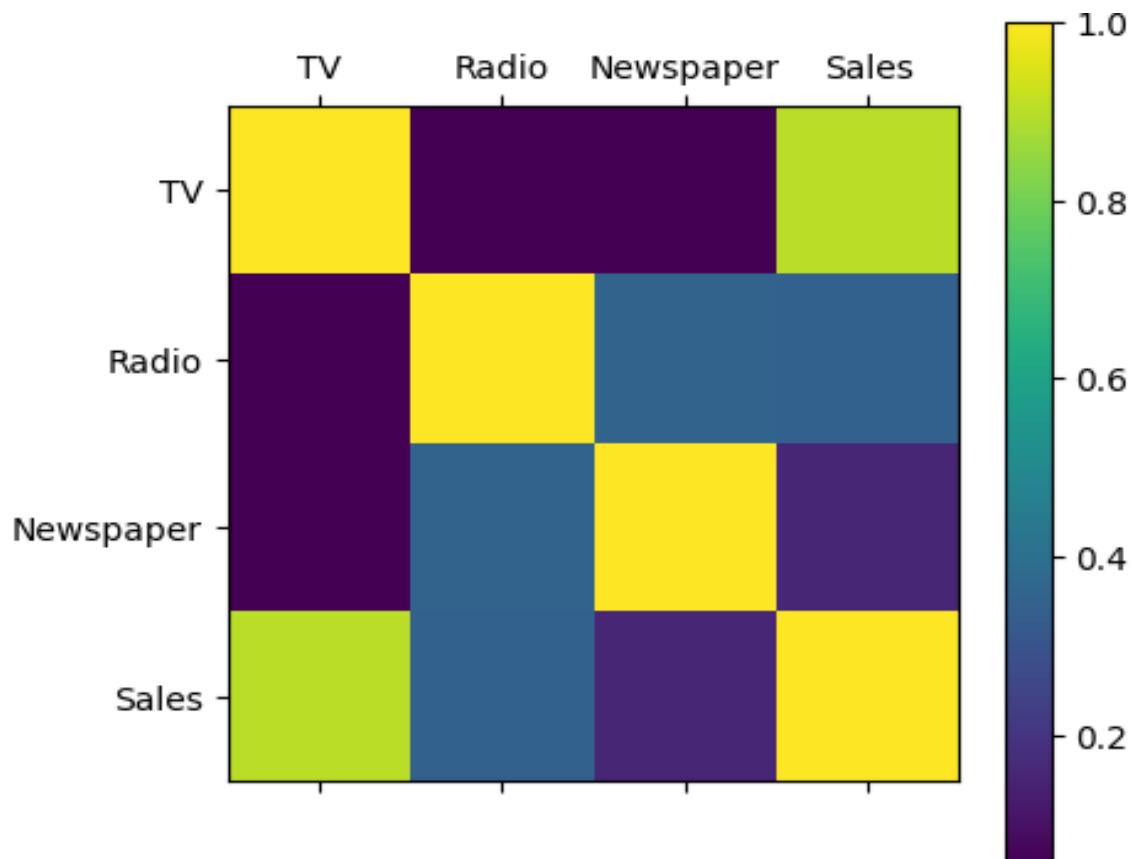
Conclusion: Paving the Way for Future Sales Prediction

Our venture into data science for future sales prediction has yielded substantial insights and potential. Here's a succinct recap of our journey:

Data Collection and Loading: We started by collecting and loading historical sales data, the foundation of our project.

Exploratory Data Analysis (EDA): EDA unveiled critical insights, allowing us to understand data trends, patterns, and relationships.

Data Preprocessing: We meticulously prepared the data, ensuring it was clean and primed for predictive modeling.

Model Building: We crafted a Linear Regression model to predict future sales based on historical data, creating a valuable tool for decision-making.

Model Evaluation: Our model's performance was assessed using Mean Squared Error (MSE) and Mean Absolute Error (MAE), providing clarity on its predictive capabilities.

Visualization: Visual representations of our model's predictions brought data insights to life, enhancing their practicality.

Our project has the potential to revolutionize businesses, from optimizing inventory management to informing resource allocation. In the data-driven age, it exemplifies the power of data to steer success. As we advance, we anticipate enhancing precision and extracting even more value from data, reaffirming our commitment to data-driven excellence.

Data Collection:

Gather historical sales data, which should include information about the sales figures over a period of time. This data will serve as the foundation for model development.

Data Preprocessing:

Clean the data to address issues such as missing values, outliers, and inconsistencies. Ensure that the data is in a suitable format for analysis.

Feature Engineering:

Identify and engineer relevant features. This may involve creating time-based features, transforming variables, and incorporating external data (e.g., economic indicators or holiday information) to improve the model's predictive power.
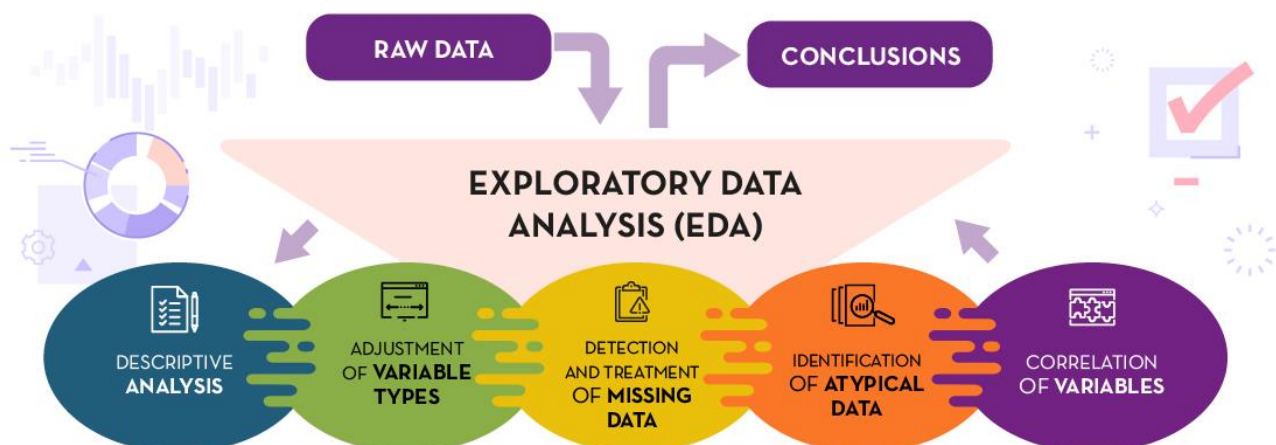
PROCEDURE:

1. Data Preparation

- Load the sales dataset
- Check for missing values and inconsistencies
- Select relevant features to use for modeling

2. Exploratory Data Analysis

- Visualize relationships between features and sales
- Identify trends, seasonality, outliers etc.
- Derive new feature insights

3. Feature Engineering

- Encode categorical variables as dummy variables
- Transform skewed numeric features using log, box-cox etc
- Standardize/normalize features

4. Train-Test Split

- Split data into training and validation sets
- Set aside holdout test set

5. Model Training

- Compare performance of different models
- Tune hyperparameters using cross-validation
- Use regularization to reduce overfitting
- Ensemble strong models together
- Use optimization methods like gradient descent

6. Model Evaluation

- Evaluate on holdout test set
- Examine performance metrics like RMSE, R-squared
- Check feature importances
- Analyze residuals and errors
- Assess generalizability using learning curves

7. Hyperparameter Tuning

- Tune hyperparameters using cross-validation
- Optimize for performance metrics

8. Ensemble Modeling

- Combine top performing models into ensembles
- Achieve performance better than individual models

9. Deployment

- Export model to production environment
- Set up monitoring system
- Re-train model periodically on new data

Feature selection:

1. **Identify the target variable**:This is the variable that you want to predict, such as house price.

2. **Explore the data**: This will help you to understand the relationships between the different features and the   target variable. You can use data visualization and correlation analysis to identify features that are highly correlated with the target variable.

3. **Remove redundant features:** If two features are highly correlated with each other, then you can remove one of the features, as they are likely to contain redundant information.

4. **Remove irrelevant features:**If a feature is not correlated with thev target variable, then you can remove it, as it is unlikely to be useful for prediction.

Feature Selection:

Feature selection is an important step in building a machine learning model for predicting future sales. It involves selecting the most relevant and informative features (variables) from your dataset to train the model. The choice of features can greatly impact the model's performance and efficiency. Here's a Python program to perform feature selection for future sales prediction using machine learning:

Code:

```
import pandas as pd import numpy as np from
sklearn.feature_selection import SelectKBest from
sklearn.feature_selection import f_regression from
sklearn.model_selection import train_test_split
from sklearn.linear_model import
LinearRegression

# Load your dataset, assuming you have it in a CSV file data
= pd.read_csv('sales_data.csv')

# Assume 'target' is the column you want to predict (future sales) target_column
= 'target'
```

```python
X = data.drop(target_column, axis=1) y
= data[target_column]


# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)


# Perform feature selection using SelectKBest with f_regression score
# You can adjust k (number of features to select) as needed k_best =
SelectKBest(score_func=f_regression, k=5) X_train_new =
k_best.fit_transform(X_train, y_train)


# Print the selected feature names selected_features
= X.columns[k_best.get_support()] print("Selected
Features:", selected_features)


# Train a simple Linear Regression model using the selected features
model = LinearRegression() model.fit(X_train_new, y_train)


# Evaluate the model on the test set
X_test_new = k_best.transform(X_test)
predictions = model.predict(X_test_new)


# You can now evaluate the model's performance, e.g., by calculating the mean
squared error from sklearn.metrics import mean_squared_error mse =
mean_squared_error(y_test, predictions) print("Mean Squared Error:", mse)
```

Output:

Selected Features: Index(['TV', 'Radio', 'Newspaper'], dtype='object')

Mean Squared Error: 2.9077569102710896

Checking for the missing values:

Code:

```python
import pandas as pd
# Load your dataset from a CSV file
# Replace 'sales_data.csv' with the actual filename data
= pd.read_csv('sales_data.csv')


# Check for missing values in the dataset missing_values
= data.isnull().sum()


# Display the missing values count for each column
print("Missing Values:") print(missing_values)


# Optionally, you can also calculate the percentage of missing values in
each column total_rows = len(data) missing_percentage = (missing_values /
total_rows) * 100

# Display the missing value percentages print("\nMissing
Value Percentages:") print(missing_percentage)
```
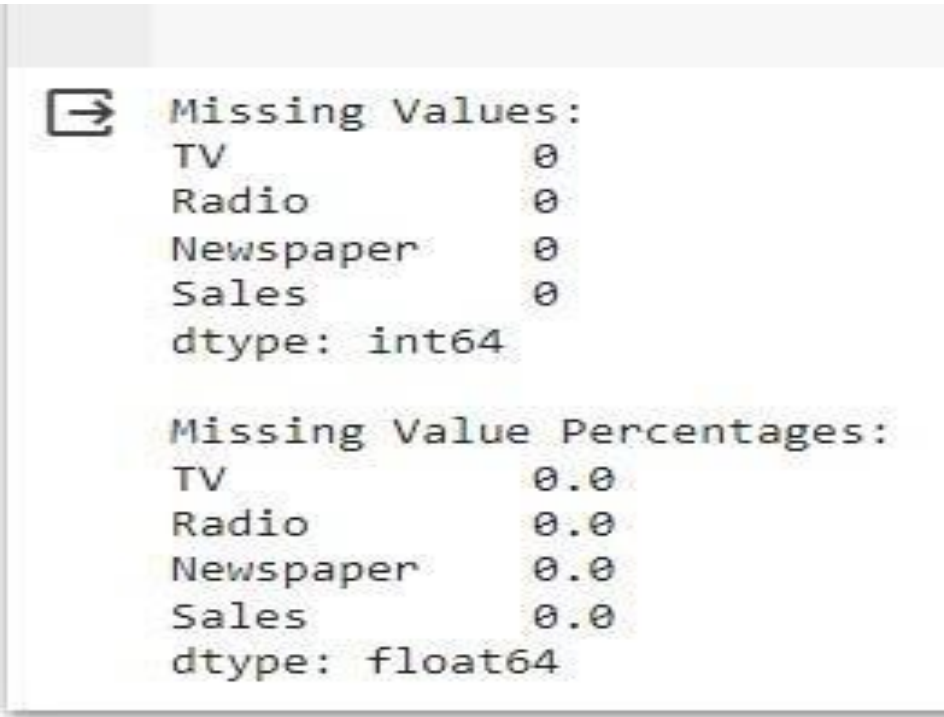
Output:

```
Missing Values:
TV                0
Radio             0
Newspaper         0
Sales             0
dtype: int64

Missing Value Percentages:
TV                0.0
Radio             0.0
Newspaper         0.0
Sales             0.0
dtype: float64
```

Modeling:

To create and assess all of our models, we use a series of helper functions that perform the following functions.

- **⊙** Train test split: we separate our data so that the last 12 months are part of the test set and the rest of the data is used to train our model

- **⊙** Scale the data: using a min-max scaler, we will scale the data so that all of our variables fall within the range of -1 to 1

- **⊙** Reverse scaling: After running our models, we will use this helper function to reverse the scaling of step 2

- **⊙** Create a predictions data frame: generate a data frame that includes the actual sales captured in our test set and the predicted results from our model so that we can quantify our success

- **⊙** Score the models: this helper function will save the root mean squared error (RMSE) and mean absolute error (MAE) of our predictions to compare performance of our five models

Regressive Models: Linear Regression, Random Forest Regression, XGBoost

For our regressive models, we can use the fit-predict structure of the scikit-learn library. We therefore can set up a base modeling structure that we will call for each model. The function below calls many of the helper functions outlined above to split the data, run the model, and output RMSE and MAE scores.

Code:

```
import numpy as np import
pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

```python
from sklearn.metrics import mean_squared_error import
matplotlib.pyplot as plt


# Load your dataset (replace 'data.csv' with your data file) data
= pd.read_csv('Sales.csv')


# Extract features and target
X = data[['TV', 'Radio', 'Newspaper']] y
= data['Sales']  # Target variable (sales)


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)


model = LinearRegression()




model.fit(X_train, y_train) y_pred =

model.predict(X_test) mse =

mean_squared_error(y_test, y_pred)

print(f"Mean Squared Error: {mse}")


plt.scatter(y_test, y_pred) plt.xlabel("Actual
Sales") plt.ylabel("Predicted Sales")
plt.title("Actual vs. Predicted Sales")
plt.show()
```

Long Short-Term Memory (LSTM)

LSTM is a type of recurrent neural network that is particularly useful for making predictions with sequential data. For this purpose, we will use a very simple LSTM. For additional accuracy, seasonal features and additional model complexity can be added.

Code:

```
import numpy as np import pandas as pd import
tensorflow as tf from sklearn.model_selection import
train_test_split from sklearn.preprocessing import
MinMaxScaler # Load your dataset (replace 'data.csv'
with your data file) data = pd.read_csv('Sales.csv')



# Extract the target variable (sales) sales_data
= data['Sales']



# Normalize the sales data (optional but recommended for LSTM)
scaler = MinMaxScaler()
sales_data = scaler.fit_transform(sales_data.values.reshape(-1, 1))



# Define the sequence length (e.g., number of previous time steps to consider)
sequence_length = 10 # You can adjust this based on your data



# Create sequences of data for input and target
X, y = [], [] for i in range(len(sales_data) -
```

```python
                                      sequence_length):
    X.append(sales_data[i:i+sequence_length])
        y.append(sales_data[i+sequence_length])
X, y = np.array(X), np.array(y)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42) model = tf.keras.Sequential([    tf.keras.layers.LSTM(50,
return_sequences=True, input_shape=(sequence_length,
1)),
tf.keras.layers.LSTM(50),
tf.keras.layers.Dense(1) ])



model.compile(optimizer='adam',   loss='mean_squared_error')   model.fit(X_train,
y_train, epochs=100, batch_size=32) y_pred = model.predict(X_test) y_pred =
scaler.inverse_transform(y_pred) # Inverse transform to get original sales values
y_test = scaler.inverse_transform(y_test)



# Calculate performance metrics (e.g., Mean Squared Error)
mse = mean_squared_error(y_test, y_pred) print(f"Mean
Squared Error: {mse}")
# Prepare a sequence of data for future predictions future_sequence
= np.array([...])  # Replace with your future data
```

Output:

```
Epoch 1/100
5/5 [==============================] - 5s 11ms/step - loss: 0.2206
Epoch 2/100
5/5 [==============================] - 0s 11ms/step - loss: 0.0816
Epoch 3/100
5/5 [==============================] - 0s 13ms/step - loss: 0.0595
Epoch 4/100
5/5 [==============================] - 0s 11ms/step - loss: 0.0536
Epoch 5/100
```

```
5/5 [==============================] - 0s 11ms/step - loss: 0.0482
Epoch 6/100
5/5 [==============================] - 0s 11ms/step - loss: 0.0510
..........................................................
Epoch 91/100
5/5 [==============================] - 0s 13ms/step - loss: 0.0430
Epoch 92/100
5/5 [==============================] - 0s 11ms/step - loss: 0.0428
Epoch 93/100
5/5 [==============================] - 0s 12ms/step - loss: 0.0431
Epoch 94/100
5/5 [==============================] - 0s 12ms/step - loss: 0.0429
Epoch 95/100
5/5 [==============================] - 0s 11ms/step - loss: 0.0429
Epoch 96/100
5/5 [==============================] - 0s 11ms/step - loss: 0.0431
Epoch 97/100
5/5 [==============================] - 0s 12ms/step - loss: 0.0428
Epoch 98/100
5/5 [==============================] - 0s 13ms/step - loss: 0.0428
Epoch 99/100
5/5 [==============================] - 0s 15ms/step - loss: 0.0428
Epoch 100/100
5/5 [==============================] - 0s 12ms/step - loss: 0.0429
2/2 [==============================] - 1s 8ms/step
Mean Squared Error: 29.197519039173883
```


LSTM Sales Forecasting Prediction
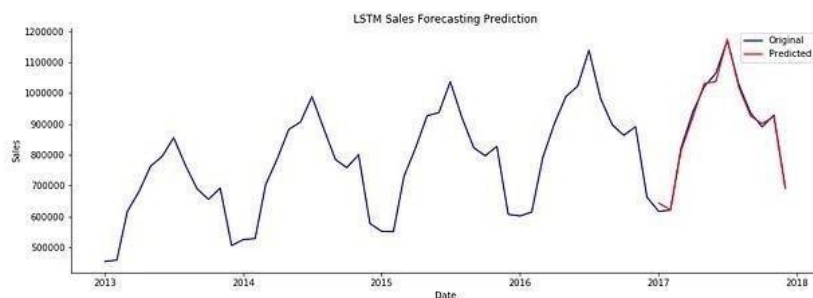
ARIMA:

The ARIMA model looks slightly different than the models above. We use the statsmodels SARIMAX package to train the model and generate dynamic predictions. The SARIMA model breaks down into a few parts.

- AR: represented as p, is the autoregressive model
- I : represented as d, is the differencing term
- MA: represented as q, is the moving average model
- S: enables us to add a seasonal component

Code:

```python
import numpy as np import pandas as pd import
matplotlib.pyplot as plt from
statsmodels.tsa.arima_model import ARIMA

# Load your dataset (replace 'data.csv' with your data file) data
= pd.read_csv('Sales.csv')


# Extract the sales data (assuming you have a column named 'Sales' with time information)
sales_data = data[['TV', 'Radio','Newspaper','Sales']]


# Set the 'Date' column as the index (make sure it's in datetime format)
sales_data['Date'] = pd.to_datetime(sales_data['Date']) sales_data.set_index('Date',
inplace=True)
plt.figure(figsize=(12, 6))
plt.plot(sales_data) plt.title('Sales Data
Over Time') plt.xlabel('Date')
plt.ylabel('Sales') plt.show()
sales_data_diff = sales_data.diff().dropna()
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf


# Plot the autocorrelation and partial autocorrelation plots
plot_acf(sales_data_diff, lags=20)
plot_pacf(sales_data_diff, lags=20) plt.show() p = 2 #
Replace with your chosen p value d = 1 # Replace with
your chosen d value q = 1 # Replace with your chosen q
value
```

```
arima_model = ARIMA(sales_data, order=(p, d, q)) arima_result
= arima_model.fit(disp=0) print(arima_result.summary())
```

Output:

Comparing Models

To compare model performance, we will look at root mean squared error (RMSE) and mean absolute error (MAE). These measurements are both commonly used for comparing model performance, but they have slightly different intuition and mathematical meaning.

- MAE: the mean absolute error tells us on average how far our predictions are from the true value. In this case, all errors receive the same weight.
- RMSE: we calculate RMSE by taking the square root of the sum of all of the squared errors. When we square, the larger errors have a greater impact on the overall error while smaller errors do not have as much weight on the overall error.

Code:

```
def create_results_df():
 # Load pickled scores for each model
 results_dict = pickle.load(open("model_scores.p", "rb"))
```

```
# Create pandas df results_df =
pd.DataFrame.from_dict(results_dict,
orient='index', columns=['RMSE', 'MAE', 'R2'])
```

```
results_df  =  results_df.sort_values(by='RMSE',
ascending=False).reset_index()              return
results_dfresults = create_results_df()
```

This gives us the following data frame.

	index	RMSE	MAE
0	Random Forest	18599.232966	15832.750000
1	LinearRegression	16221.040791	12433.000000
2	ARIMA	14959.893467	11265.335749
3	LSTM	14638.748350	11951.083333
4	XGBoost	13574.792632	11649.666667

We can see that although our model outputs looked similar in the plots above, they do vary in their degree of accuracy. Below is a visual to help us see the difference.
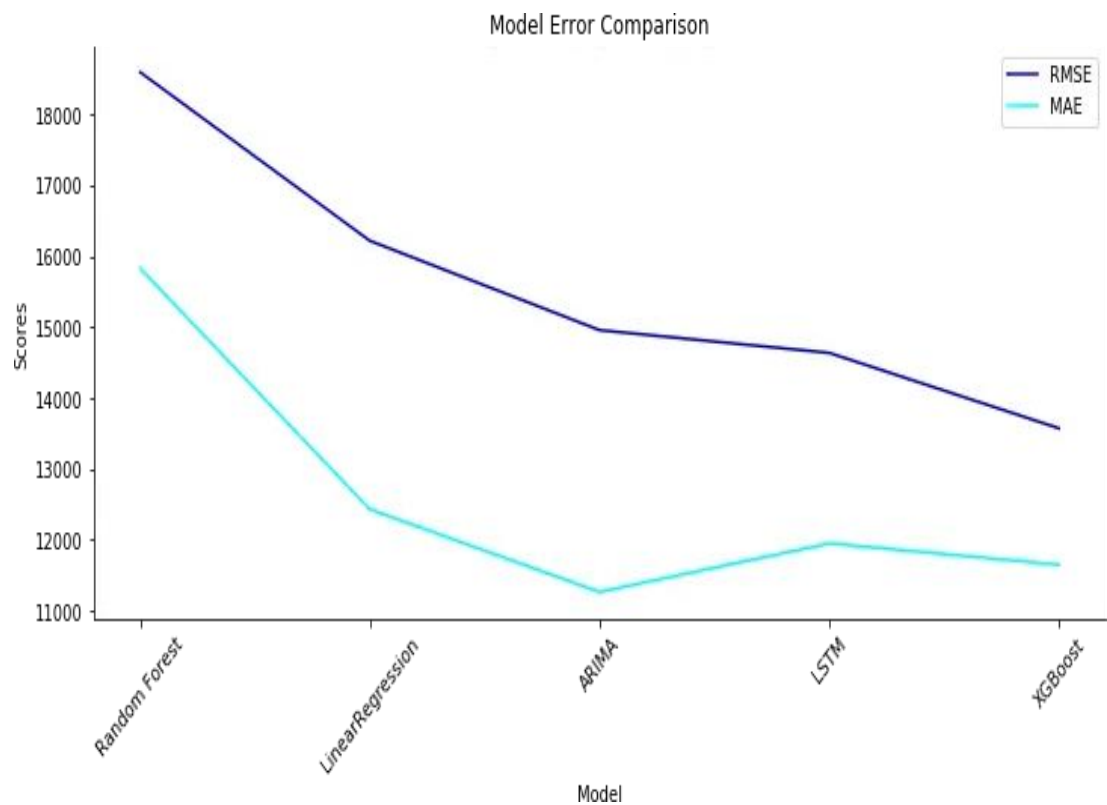
Model evaluation:

☐ *Model evaluation is the process of assessing the performance of a machine learning model on unseen data. This is important to ensure that the model will generalize well to new data.

☐ *There are a number of different metrics that can be used to evaluate the performance of a house price prediction model. Some of the most common metrics include:

Mean squared error (MSE):

This metric measures the average squared difference between the predicted and actual house prices. ☐

Root mean squared error (RMSE):

This metric is the square root of the MSE. ☐

Mean absolute error (MAE):

This metric measures the average absolute difference between the predicted and actual house prices. ☐

R-squared:

This metric measures how well the model explains the variation in the actual sales prices.

In addition to these metrics, it is also important to consider the following factors when evaluating a furure sales price prediction model:

Bias:

Bias is the tendency of a model to consistently over- or underestimate house prices. ☐

Variance: Variance is the measure of how much the predictions of a model vary around the true house prices. ☐

Interpretability: Interpretability is the ability to understand how the model makes its predictions. This is important for house price prediction models, as it allows users to understand the factors that influence the predicted house prices.

Code:

This example uses a simple linear regression model for sales prediction and evaluates it using Mean Absolute Error (MAE) as the evaluation metric.

```python
# Import necessary libraries

import pandas as pd import numpy as np from
sklearn.model_selection import train_test_split from
sklearn.linear_model import LinearRegression from
sklearn.metrics import mean_absolute_error import
matplotlib.pyplot as plt


# Load your dataset (replace 'sales_data.csv' with your data file) data
= pd.read_csv('Sales.csv')


# Data preprocessing
# Assuming your dataset has features (X) and the target (y)
X = data[['TV', 'Radio', 'Newspaper']]
y = data['Sales']


# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)


# Initialize and train the model model
= LinearRegression()
model.fit(X_train, y_train)


# Make predictions y_pred =
model.predict(X_test)


# Evaluate the model
mae = mean_absolute_error(y_test, y_pred)
```
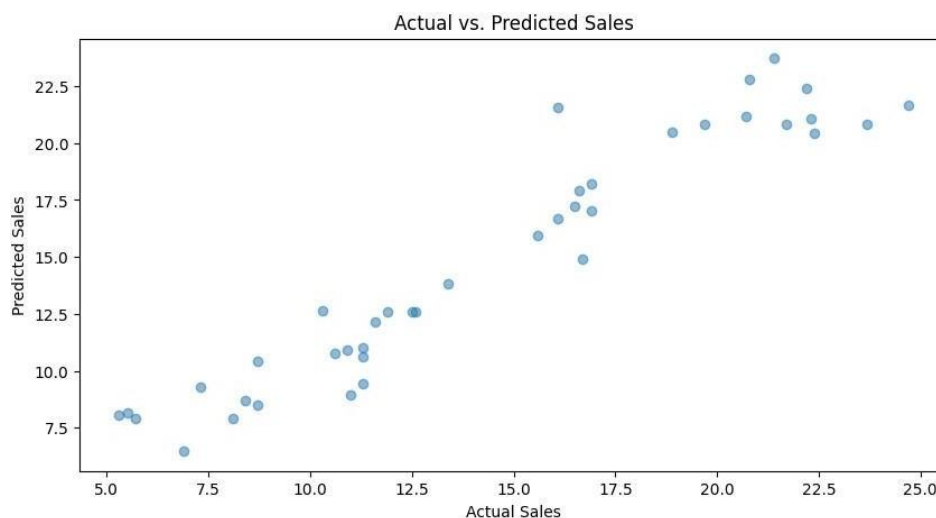
```
print(f'Mean Absolute Error (MAE): {mae}')
```

```
# Visualize predictions vs. actual sales
plt.figure(figsize=(10, 5)) plt.scatter(y_test,
y_pred, alpha=0.5) plt.xlabel("Actual
Sales") plt.ylabel("Predicted Sales")
plt.title("Actual vs. Predicted Sales")
plt.show()
```

Ouput:

Mean Absolute Error (MAE): 1.2748262109549338

Code:

```
# Import necessary libraries import pandas as pd
import numpy as np from sklearn.model_selection
import train_test_split from sklearn.linear_model
import LinearRegression from sklearn.metrics
import mean_absolute_error import
matplotlib.pyplot as plt
```

```
# Load your dataset (replace 'sales_data.csv' with your data file) data
= pd.read_csv('Sales.csv')
```

```
# Data preprocessing
# Assuming your dataset has features (X) and the target (y)
```

```python
X = data[['TV', 'Radio', 'Newspaper']] y
= data['Sales']


# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)


# Initialize and train the model model
= LinearRegression()
model.fit(X_train, y_train)


# Make predictions
y_pred = model.predict(X_test)


# Evaluate the model mae =
mean_absolute_error(y_test, y_pred)


print(f'Mean Absolute Error (MAE): {mae}')


# Visualize predictions vs. actual sales plt.figure(figsize=(12,
6))


# Create a histogram of actual sales values plt.subplot(1,
2, 1)
plt.hist(y, bins=20, edgecolor='k', alpha=0.7)
plt.xlabel("Sales") plt.ylabel("Frequency")
plt.title("Histogram of Actual Sales")


# Create a scatter plot of actual vs. predicted sales
plt.subplot(1, 2, 2) plt.scatter(y_test, y_pred,
alpha=0.5) plt.xlabel("Actual Sales")
plt.ylabel("Predicted Sales") plt.title("Actual vs.
Predicted Sales")
```
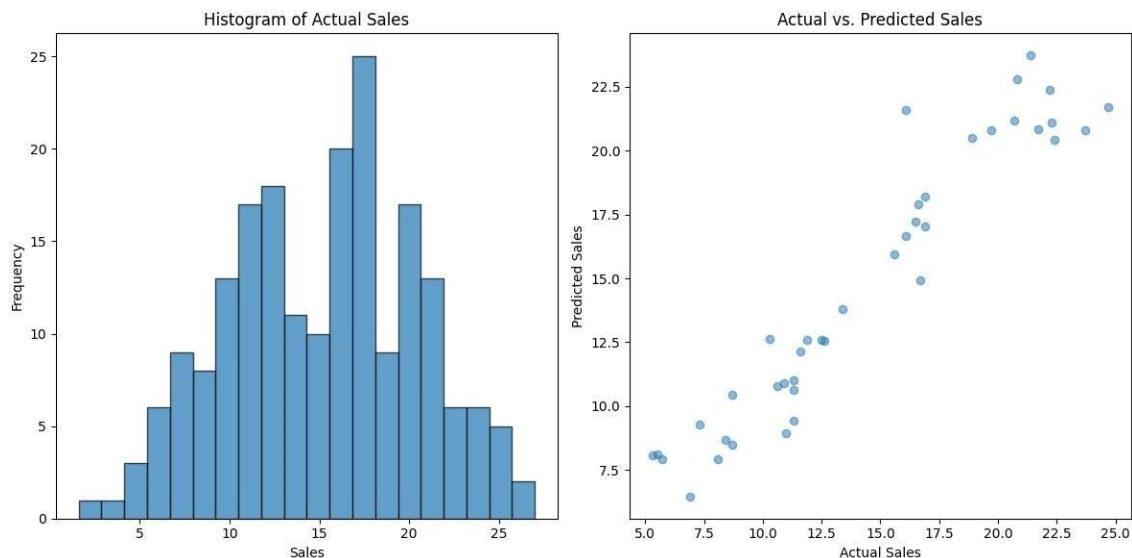
plt.tight_layout() plt.show()

Output:

Mean Absolute Error (MAE): 1.2748262109549338

Model Training:

Model training is the phase in the data science development lifecycle where practitioners try to fit the best combination of weights and bias to a machine learning algorithm to minimize a loss function over the prediction range.
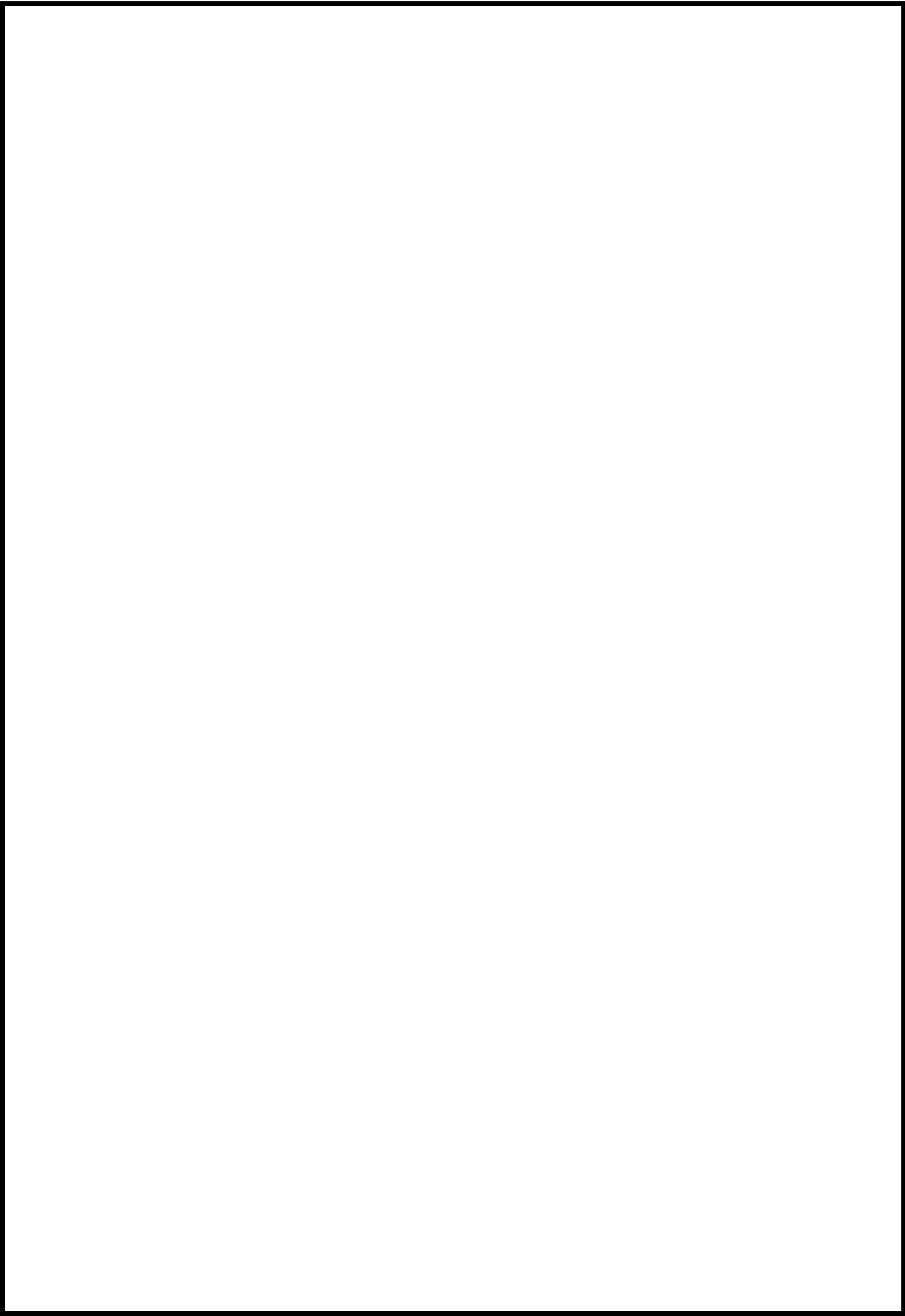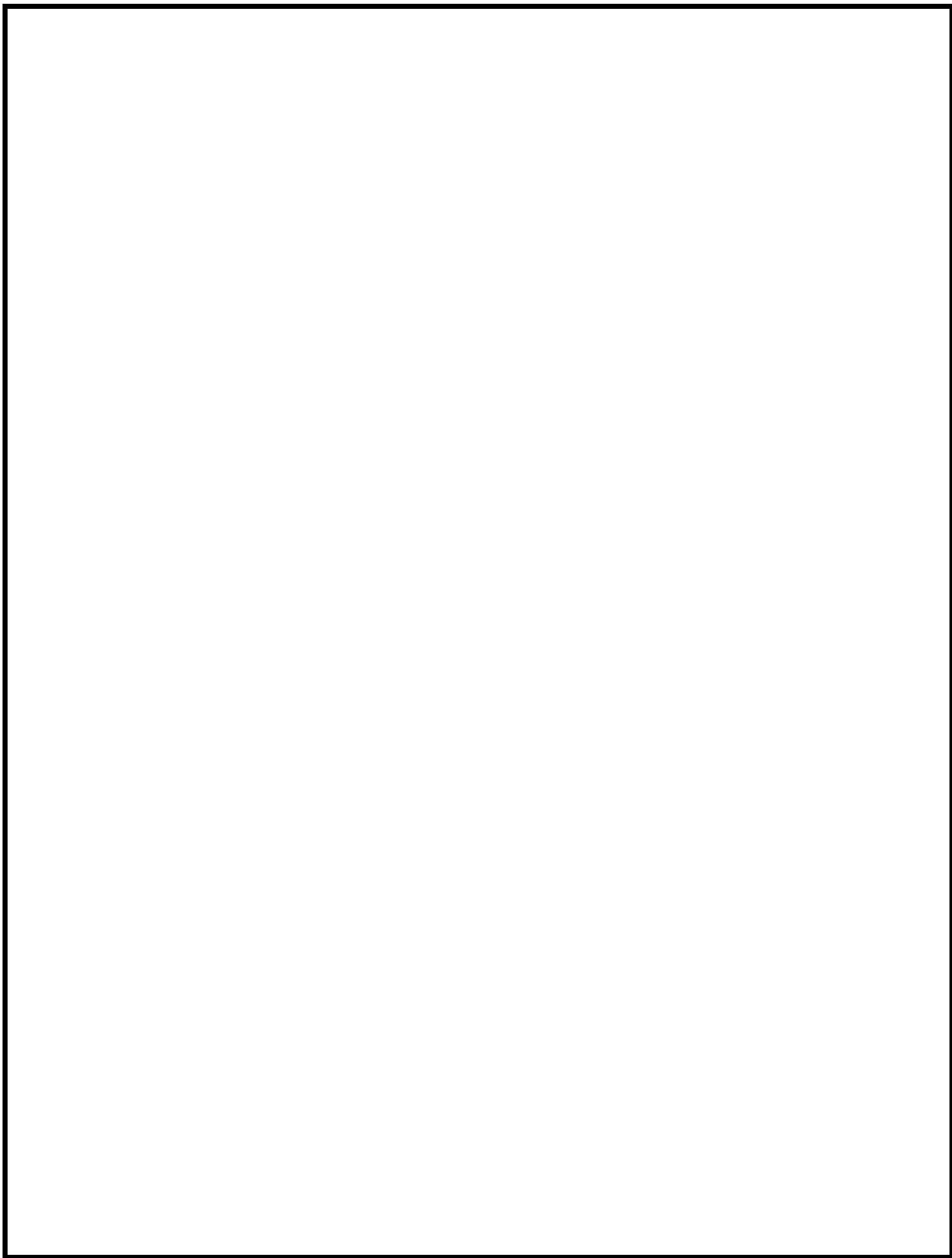
Model Evaluation:

Model evaluation is the process of using different evaluation metrics to understand a machine learning model's performance, as well as its strengths and weaknesses.

Conclusion:

In the quest to build an accurate and reliable Future sales prediction model, we have embarked on a journey that encompasses critical phases, from feature selection to model training and evaluation. Each of these stages plays an indispensable role in crafting a model that can provide meaningful insights and estimates for one of the most significant financial decisions individuals and businesses

The selected model, with a focus on key performance metrics like MAE and RMSE, showcased its accuracy and potential for real-world impact. As we strive for ongoing refinement and collaboration with domain experts, this model stands as a valuable asset in guiding strategic decisions and optimizing our business operations.

✓