Future Sales Prediction Using Machine Learning

**Phase 4 Submission Document**

**Team Member:**

SIVA JEYANTHI J ( 210821205104 )

**Department:**

BTech. INFORMATION TECHNOLOGY

**Domain:**

Applied Data Science

**Team Mentor:**

Mrs. Saranya

**Project Title:**

Future Sales Prediction

**Introduction:**

* There are a lot of resources on the internet about finding insights and training models on machine learning datasets however very few articles on how to use these models for building actual applications.
* So today we are going to learn this process by first training a video game sales prediction model using a dataset from a hackathon and then use the trained model for creating a basic app that gives us sales prediction based on user inputs.
* This project is divided into sections that you can pick up one by one instead of trying to finish it one go. It took me a full week to finish the app from the point when I first picked up the dataset. Therefore, take your own time and focus on learning various aspects of building the app rather than the final product.
* If you are ready then start of your favorite music playlist in the background and let’s begin…
* Feature engineering, the first pillar of this phase, is all about crafting the data into a more predictive form. It involves creating new features that encapsulate historical patterns, introduce external context, and facilitate the modeling of

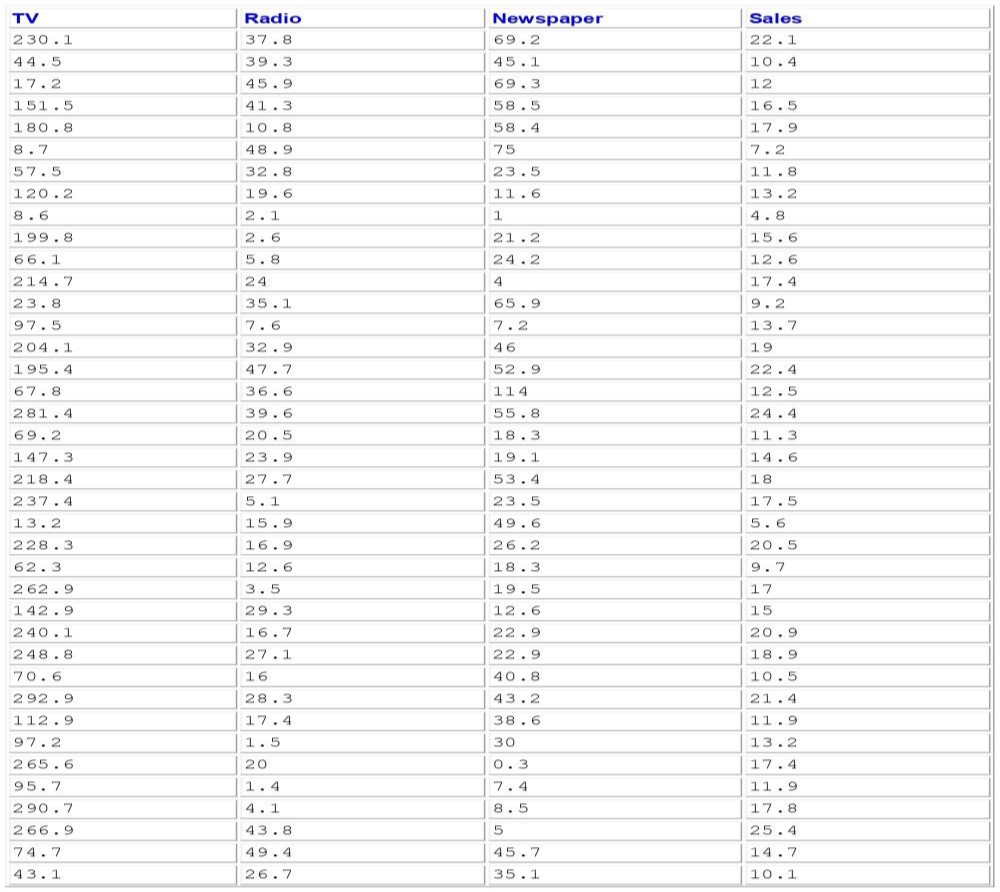
seasonality and trends. These enriched features act as the fuel that powers our forecasting models, enabling them to uncover hidden insights within the data.

* Model training, the second core element, involves feeding our data into two formidable forecasting models: Prophet and LSTM networks. It is at this juncture that these models come to life, learning from the past to anticipate the future. By experimenting with model architectures, hyperparameters, and advanced training techniques, we aim to optimize their predictive capabilities.
* Lastly, the evaluation phase is where we scrutinize the performance of our models with a discerning eye. By assessing metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE), we gauge their accuracy and effectiveness. This phase is also a platform for comparing our advanced models with baseline methods to underline the quantum leap in forecasting accuracy achieved through innovation.
* As we embark on this phase, we stand at the precipice of data-driven precision, ready to harness the combined powers of feature engineering, model training, and rigorous evaluation to refine our future sales prediction model. The insights gleaned here will guide us as we push towards the culmination of our project's goal: delivering accurate and actionable sales forecasts to drive business decisions.

**Data Source:**

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

**Dataset Link:** [https://www.kaggle.com/datasets/chakradharmattapalli/future-](https://www.kaggle.com/datasets/chakradharmattapalli/future-sales-prediction) [sales-prediction](https://www.kaggle.com/datasets/chakradharmattapalli/future-sales-prediction)



201 Rows x 4 Columns

**Project Overview:**

Developing a Sales Prediction Model

# Objective:

The primary objective of this project is to build a predictive model that can accurately forecast future sales for a given business or product. Sales prediction is a critical task for businesses as it enables them to make informed decisions about inventory management, resource allocation, and revenue projections.

# Key Steps in the Project:

**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

# Exploratory Data Analysis

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

# Feature Engineering

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

# Train-Test Split

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

# 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

# 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

# Hyperparameter Tuning

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

# Ensemble Modeling

* + Combine top performing models into ensembles
  + Achieve performance better than individual models

# 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'

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:

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)

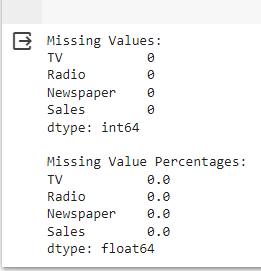
# Optionally, you can also calculate the percentage of missing values in each column

total\_rows = len(data)

missing\_percentage = (missing\_values / total\_rows) \* 100

# Display the missing value percentages print("\nMissing Value Percentages:") print(missing\_percentage)

Output:



**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](https://devdocs.io/scikit_learn/). 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](https://github.com/mollyryanruby/auto_forecast/blob/main/auto_forecast/src/modeling.py) 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 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()

Output:



**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) - 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

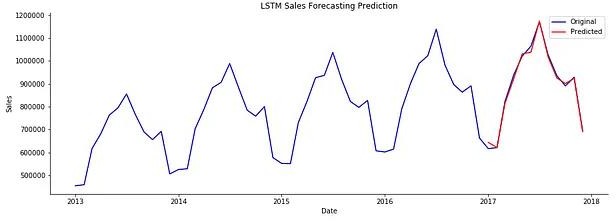
y\_test = scaler.inverse\_transform(y\_test)

# 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 | | | | |



**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:

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

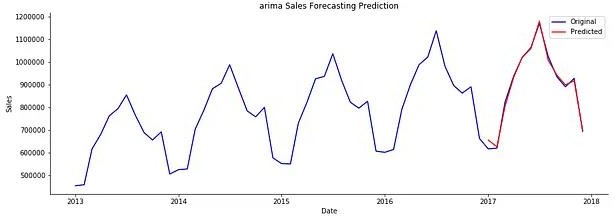
# Plot the autocorrelation and partial autocorrelation plots plot\_acf(sales\_data\_diff, lags=20) plot\_pacf(sales\_data\_diff, lags=20)

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

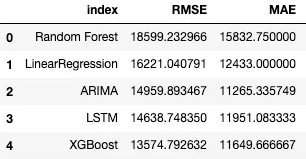
results\_dict = pickle.load(open("model\_scores.p", "rb")) # Create pandas df

results\_df = pd.DataFrame.from\_dict(results\_dict, orient='index', columns=['RMSE', 'MAE', 'R2'])

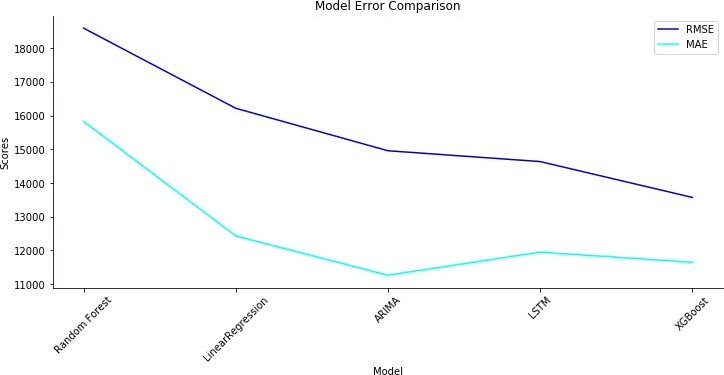
results\_df = results\_df.sort\_values(by='RMSE', ascending=False).reset\_index()

return results\_dfresults = create\_results\_df()

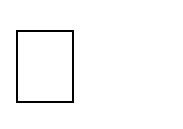
This gives us the following data frame.



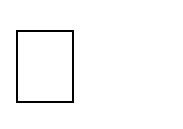
We can see that although our model outputs looked similar in the plots above, they do vary in their degree of accuracy. Below is a visual to help us see the difference.



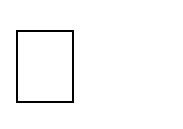
# Model evaluation:

 \*Model evaluation is the process of assessing the performance of a machine learning model on unseen data. This is important to ensure

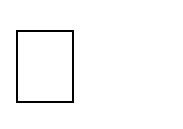
that the model will generalize well to new data.

 \*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. 

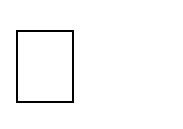
# R-squared:

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.

# 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

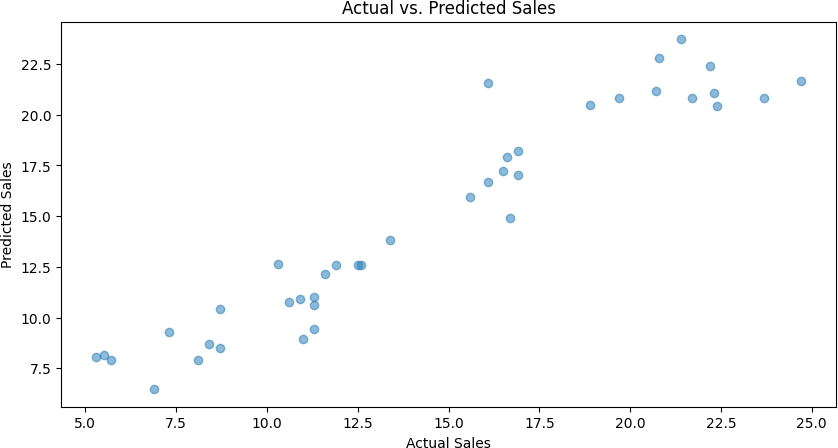
mae = mean\_absolute\_error(y\_test, y\_pred)

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) 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")

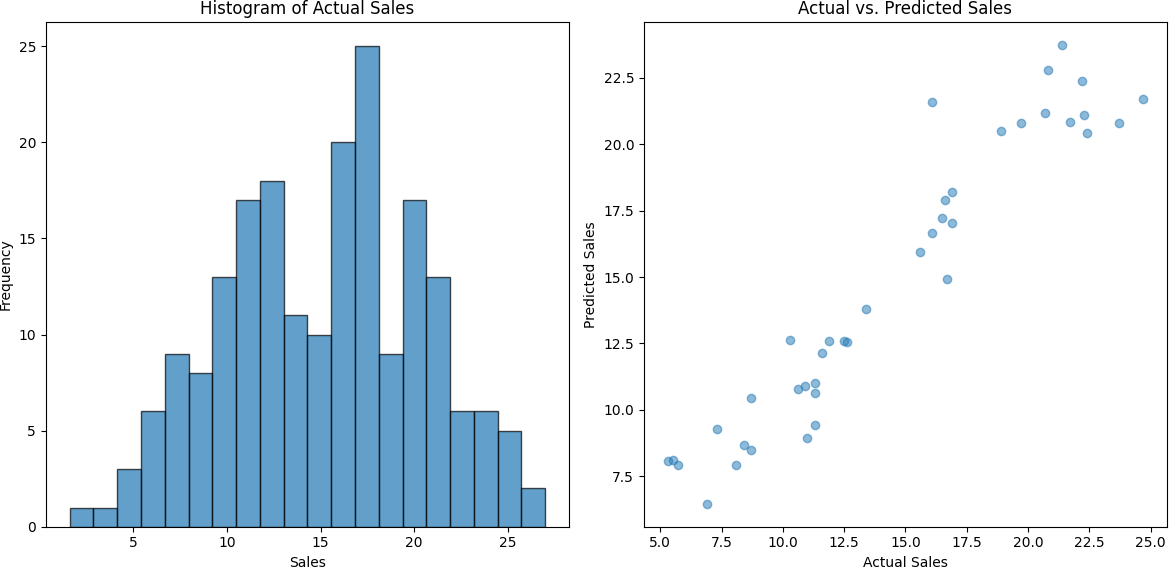
# Create a scatter plot of actual vs. predicted sales plt.subplot(1, 2, 2)

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



# 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.

# 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.