1. **Arima**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

from datetime import datetime

import math

import pmdarima as pm

from pmdarima.arima import auto\_arima

# Load data

train = pd.read\_csv('trains.csv')

store = pd.read\_csv('stores.csv')

feature = pd.read\_csv('features.csv')

# Data Preprocessing and Merging

data = train.merge(feature, on=['Store', 'Date'], how='inner').merge(store, on=['Store'], how='inner')

# Fill NaNs

data['MarkDown1'] = data['MarkDown1'].replace(np.nan, 0)

data['MarkDown2'] = data['MarkDown2'].replace(np.nan, 0)

data['MarkDown3'] = data['MarkDown3'].replace(np.nan, 0)

data['MarkDown4'] = data['MarkDown4'].replace(np.nan, 0)

data['MarkDown5'] = data['MarkDown5'].replace(np.nan, 0)

# Filter data

data = data[data['Weekly\_Sales'] >= 0]

# Convert categorical variables to dummy/indicator variables

data = pd.get\_dummies(data, columns=['Type'])

data['Date'] = pd.to\_datetime(data['Date'])

# Extract features from date

data['month'] = data['Date'].dt.month

data['year'] = data['Date'].dt.year

data['dayofweek\_name'] = data['Date'].dt.day\_name()

data['is\_weekend'] = data['dayofweek\_name'].isin(['Sunday', 'Saturday']).astype(int)

# Drop unnecessary columns

data = data.drop(columns=['dayofweek\_name'])

# Prepare features and target variable for regression

X = data[["Store", "Dept", "Size", "IsHoliday\_x", "CPI", "Temperature", "Type\_B", "Type\_C", "month", "year", "is\_weekend"]]

y = data["Weekly\_Sales"]

# Train-test split for regression

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Regression model: XGBoost

import xgboost as xgb

xg\_reg = xgb.XGBRegressor(objective='reg:squarederror', nthread=4, n\_estimators=500, max\_depth=4, learning\_rate=0.5)

xg\_reg.fit(X\_train, y\_train)

pred = xg\_reg.predict(X\_train)

y\_pred = xg\_reg.predict(X\_test)

# Calculate regression metrics

print('Regression Model Accuracy (R²):', r2\_score(y\_test, y\_pred) \* 100, '%')

print('RMSE:', mean\_squared\_error(y\_test, y\_pred, squared=False))

print('MAE:', mean\_absolute\_error(y\_test, y\_pred))

# Time series analysis using ARIMA

data.set\_index('Date', inplace=True)

data = data.resample('MS').mean() # Resample the time series data with month starting first.

# Train-Test splitting of time series data

train\_data = data[:int(0.7 \* (len(data)))]

test\_data = data[int(0.7 \* (len(data))):]

train\_data = train\_data['Weekly\_Sales']

test\_data = test\_data['Weekly\_Sales']

# Plot of Weekly\_Sales with respect to years in train and test

train\_data.plot(figsize=(20, 8), title='Weekly\_Sales', fontsize=14)

test\_data.plot(figsize=(20, 8), title='Weekly\_Sales', fontsize=14)

plt.show()

# Fit ARIMA model

model\_auto\_arima = auto\_arima(train\_data, trace=True, error\_action='ignore',

suppress\_warnings=True, start\_p=0, start\_q=0, start\_P=0,

start\_Q=0, max\_p=10, max\_q=10, max\_P=10, max\_Q=10,

seasonal=True, stepwise=False, D=1, max\_D=10,

approximation=False)

model\_auto\_arima.fit(train\_data)

# Predicting the test values using predict function

forecast = model\_auto\_arima.predict(n\_periods=len(test\_data))

forecast = pd.DataFrame(forecast, index=test\_data.index, columns=['Prediction'])

# Plot predictions

plt.figure(figsize=(20, 6))

plt.title('Prediction of Weekly Sales using Auto ARIMA model', fontsize=20)

plt.plot(train\_data, label='Train')

plt.plot(test\_data, label='Test')

plt.plot(forecast, label='Prediction using ARIMA Model')

plt.legend(loc='best')

plt.xlabel('Date', fontsize=14)

plt.ylabel('Weekly Sales', fontsize=14)

plt.show()

# Performance metrics for ARIMA model

print('Mean Squared Error (MSE) of ARIMA:', mean\_squared\_error(test\_data, forecast))

print('Root Mean Squared Error (RMSE) of ARIMA:', math.sqrt(mean\_squared\_error(test\_data, forecast)))

print('Mean Absolute Error (MAE) of ARIMA:', mean\_absolute\_error(test\_data, forecast))

# Calculate accuracy for ARIMA model

def calculate\_accuracy(actual, predicted):

return 1 - (np.sum(np.abs(actual - predicted)) / np.sum(np.abs(actual)))

arima\_accuracy = calculate\_accuracy(test\_data, forecast['Prediction'])

print('ARIMA Model Accuracy:', arima\_accuracy \* 100, '%')

1. **Decision Tree**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.tree import DecisionTreeRegressor

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score, confusion\_matrix, classification\_report, accuracy\_score

# Load data

train = pd.read\_csv('train[2].csv')

store = pd.read\_csv('stores[1].csv')

feature = pd.read\_csv('features[1].csv')

# Data Preprocessing and Merging

data = train.merge(feature, on=['Store', 'Date'], how='inner').merge(store, on=['Store'], how='inner')

# Fill NaNs

data['MarkDown1'] = data['MarkDown1'].replace(np.nan, 0)

data['MarkDown2'] = data['MarkDown2'].replace(np.nan, 0)

data['MarkDown3'] = data['MarkDown3'].replace(np.nan, 0)

data['MarkDown4'] = data['MarkDown4'].replace(np.nan, 0)

data['MarkDown5'] = data['MarkDown5'].replace(np.nan, 0)

# Filter data

data = data[data['Weekly\_Sales'] >= 0]

# Convert categorical variables to dummy/indicator variables

data = pd.get\_dummies(data, columns=['Type'])

data['Date'] = pd.to\_datetime(data['Date'])

# Extract features from date

data['month'] = data['Date'].dt.month

data['year'] = data['Date'].dt.year

data['dayofweek\_name'] = data['Date'].dt.day\_name()

data['is\_weekend'] = data['dayofweek\_name'].isin(['Sunday', 'Saturday']).astype(int)

# Drop unnecessary columns

data = data.drop(columns=['dayofweek\_name', 'Date'])

# Prepare features and target variable

X = data[["Store", "Dept", "Size", "IsHoliday\_x", "CPI", "Temperature", "Type\_B", "Type\_C", "month", "year", "is\_weekend"]]

y = data["Weekly\_Sales"]

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Define the Decision Tree Regressor model

dt = DecisionTreeRegressor(random\_state=0)

# Define the parameter grid

param\_grid = {

'max\_depth': [3, 5, 7, 10, None],

'min\_samples\_split': [2, 5, 10],

'min\_samples\_leaf': [1, 2, 4]

}

# Perform Grid Search

grid\_search = GridSearchCV(estimator=dt, param\_grid=param\_grid, cv=5, scoring='neg\_mean\_squared\_error', verbose=1, n\_jobs=-1)

grid\_search.fit(X\_train, y\_train)

# Print the best parameters and best score

print("Best Parameters:", grid\_search.best\_params\_)

print("Best Score:", grid\_search.best\_score\_)

# Train the model with the best parameters

best\_dt = grid\_search.best\_estimator\_

best\_dt.fit(X\_train, y\_train)

# Predict on test data

y\_pred = best\_dt.predict(X\_test)

# Evaluation

mse = mean\_squared\_error(y\_test, y\_pred)

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

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

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

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

print(f"R-squared (R²): {r2}")

# Example input for prediction

new\_data = [[30, 5, 2000, 0, 211.0, 45.0, 1, 0, 7, 2023, 0]] # Adjust values accordingly

# Predicting

pred1 = best\_dt.predict(new\_data)

print(f"Prediction for new data: {pred1}")

# Define thresholds for categorizing sales

def categorize\_sales(sales):

if sales < 5000:

return 'Low'

elif sales < 20000:

return 'Medium'

else:

return 'High'

# Apply the categorization to actual and predicted sales

y\_test\_cat = y\_test.apply(categorize\_sales)

y\_pred\_cat = pd.Series(y\_pred).apply(categorize\_sales)

# Generate the confusion matrix

cm = confusion\_matrix(y\_test\_cat, y\_pred\_cat, labels=['Low', 'Medium', 'High'])

print("Confusion Matrix:")

print(cm)

# Classification report for more detailed metrics

report = classification\_report(y\_test\_cat, y\_pred\_cat, labels=['Low', 'Medium', 'High'])

print("Classification Report:")

print(report)

# Calculate accuracy

accuracy = accuracy\_score(y\_test\_cat, y\_pred\_cat)

print(f"Accuracy: {accuracy}")

# Visualize the confusion matrix using a heatmap

plt.figure(figsize=(10, 7))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Low', 'Medium', 'High'], yticklabels=['Low', 'Medium', 'High'])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

1. **linear regression**

linear import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

from datetime import datetime

# Load data

train = pd.read\_csv('train.csv')

store = pd.read\_csv('stores.csv')

feature = pd.read\_csv('features.csv')

# Data Preprocessing and Merging

data = train.merge(feature, on=['Store', 'Date'], how='inner').merge(store, on=['Store'], how='inner')

# Fill NaNs

data['MarkDown1'] = data['MarkDown1'].replace(np.nan, 0)

data['MarkDown2'] = data['MarkDown2'].replace(np.nan, 0)

data['MarkDown3'] = data['MarkDown3'].replace(np.nan, 0)

data['MarkDown4'] = data['MarkDown4'].replace(np.nan, 0)

data['MarkDown5'] = data['MarkDown5'].replace(np.nan, 0)

# Filter data

data = data[data['Weekly\_Sales'] >= 0]

# Convert categorical variables to dummy/indicator variables

data = pd.get\_dummies(data, columns=['Type'])

data['Date'] = pd.to\_datetime(data['Date'])

print(data.head())

# Extract features from date

data['month'] = data['Date'].dt.month

data['year'] = data['Date'].dt.year

data['dayofweek\_name'] = data['Date'].dt.day\_name()

data['is\_weekend'] = data['dayofweek\_name'].isin(['Sunday', 'Saturday']).astype(int)

# Drop unnecessary columns

data = data.drop(columns=['dayofweek\_name', 'Date'])

# Prepare features and target variable

X = data[["Store", "Dept", "Size", "IsHoliday\_x", "CPI", "Temperature", "Type\_B", "Type\_C", "month", "year", "is\_weekend"]]

y = data["Weekly\_Sales"]

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train the linear regression model

lr = LinearRegression()

lr.fit(X\_train, y\_train)

# Predict on test data

y\_pred = lr.predict(X\_test)

# Evaluation

mse = mean\_squared\_error(y\_test, y\_pred)

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

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

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

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

print(f"R-squared (R²): {r2}")

# Example input for prediction

new\_data = [[30, 5, 2000, 0, 211.0, 45.0, 1, 0, 7, 2023, 0]] # Adjust values accordingly

# Predicting

pred1 = lr.predict(new\_data)

print(f"Prediction for new data: {pred1}")

1. XgBoost

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error

import xgboost as xgb

import warnings

# Load data

train = pd.read\_csv('trains.csv')

store = pd.read\_csv('stores.csv')

feature = pd.read\_csv('features.csv')

# Data Preprocessing and Merging

data = train.merge(feature, on=['Store', 'Date'], how='inner').merge(store, on=['Store'], how='inner')

# Fill NaNs

data['MarkDown1'] = data['MarkDown1'].replace(np.nan, 0)

data['MarkDown2'] = data['MarkDown2'].replace(np.nan, 0)

data['MarkDown3'] = data['MarkDown3'].replace(np.nan, 0)

data['MarkDown4'] = data['MarkDown4'].replace(np.nan, 0)

data['MarkDown5'] = data['MarkDown5'].replace(np.nan, 0)

# Filter data

data = data[data['Weekly\_Sales'] >= 0]

# Convert categorical variables to dummy/indicator variables

data = pd.get\_dummies(data, columns=['Type'])

data['Date'] = pd.to\_datetime(data['Date'])

# Extract features from date

data['month'] = data['Date'].dt.month

data['year'] = data['Date'].dt.year

data['dayofweek\_name'] = data['Date'].dt.day\_name()

data['is\_weekend'] = data['dayofweek\_name'].isin(['Sunday', 'Saturday']).astype(int)

# Drop unnecessary columns

data = data.drop(columns=['dayofweek\_name', 'Date'])

# Prepare features and target variable

X = data[["Store", "Dept", "Size", "IsHoliday\_x", "CPI", "Temperature", "Type\_B", "Type\_C", "month", "year", "is\_weekend"]]

y = data["Weekly\_Sales"]

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Initialize and train XGBoost model

xg\_reg = xgb.XGBRegressor(objective='reg:squarederror', nthread=4, n\_estimators=500, max\_depth=4, learning\_rate=0.5)

xg\_reg.fit(X\_train, y\_train)

# Predict

pred = xg\_reg.predict(X\_train)

y\_pred = xg\_reg.predict(X\_test)

# Print metrics

print('Test Accuracy:', xg\_reg.score(X\_test, y\_test) \* 100, '%')

rms = mean\_squared\_error(y\_test, y\_pred, squared=False)

print('RMSE:', rms)

print('MAE:', mean\_absolute\_error(y\_test, y\_pred))

print('Training Accuracy:', xg\_reg.score(X\_train, y\_train) \* 100, '%')

1. **Random Forest**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

from datetime import datetime

# Load data

train = pd.read\_csv('train[2].csv')

store = pd.read\_csv('stores[1].csv')

feature = pd.read\_csv('features[1].csv')

# Data Preprocessing and Merging

data = train.merge(feature, on=['Store', 'Date'], how='inner').merge(store, on=['Store'], how='inner')

# Fill NaNs

data['MarkDown1'] = data['MarkDown1'].replace(np.nan, 0)

data['MarkDown2'] = data['MarkDown2'].replace(np.nan, 0)

data['MarkDown3'] = data['MarkDown3'].replace(np.nan, 0)

data['MarkDown4'] = data['MarkDown4'].replace(np.nan, 0)

data['MarkDown5'] = data['MarkDown5'].replace(np.nan, 0)

# Filter data

data = data[data['Weekly\_Sales'] >= 0]

# Convert categorical variables to dummy/indicator variables

data = pd.get\_dummies(data, columns=['Type'])

data['Date'] = pd.to\_datetime(data['Date'])

# Extract features from date

data['month'] = data['Date'].dt.month

data['year'] = data['Date'].dt.year

data['dayofweek\_name'] = data['Date'].dt.day\_name()

data['is\_weekend'] = data['dayofweek\_name'].isin(['Sunday', 'Saturday']).astype(int)

# Drop unnecessary columns

data = data.drop(columns=['dayofweek\_name', 'Date'])

# Prepare features and target variable

X = data[["Store", "Dept", "Size", "IsHoliday\_x", "CPI", "Temperature", "Type\_B", "Type\_C", "month", "year", "is\_weekend"]]

y = data["Weekly\_Sales"]

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train the Random Forest model

rf = RandomForestRegressor(n\_estimators=100, random\_state=42)

rf.fit(X\_train, y\_train)

# Predict on test data

y\_pred = rf.predict(X\_test)

# Evaluation

mse = mean\_squared\_error(y\_test, y\_pred)

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

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

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

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

print(f"R-squared (R²): {r2}")

# Example input for prediction

new\_data = [[30, 5, 2000, 0, 211.0, 45.0, 1, 0, 7, 2023, 0]] # Adjust values accordingly

# Predicting

pred1 = rf.predict(new\_data)

print(f"Prediction for new data: {pred1}")

from sklearn.metrics import r2\_score

acc=r2\_score(y\_pred,y\_test)

print(acc)

# Histogram for CPI

plt.figure(figsize=(10, 6))

sns.histplot(data['CPI'], bins=30, kde=True)

plt.title('Distribution of Consumer Price Index (CPI)')

plt.xlabel('CPI')

plt.ylabel('Frequency')

plt.show()

from sklearn.metrics import confusion\_matrix, classification\_report

# Define thresholds for categorizing sales

def categorize\_sales(sales):

if sales < 5000:

return 'Low'

elif sales < 20000:

return 'Medium'

else:

return 'High'

# Apply the categorization to actual and predicted sales

y\_test\_cat = y\_test.apply(categorize\_sales)

y\_pred\_cat = pd.Series(y\_pred).apply(categorize\_sales)

# Generate the confusion matrix

cm = confusion\_matrix(y\_test\_cat, y\_pred\_cat, labels=['Low', 'Medium', 'High'])

print("Confusion Matrix:")

print(cm)

# Classification report for more detailed metrics

report = classification\_report(y\_test\_cat, y\_pred\_cat, labels=['Low', 'Medium', 'High'])

print("Classification Report:")

print(report)

# Visualize the confusion matrix using a heatmap

plt.figure(figsize=(10, 7))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Low', 'Medium', 'High'], yticklabels=['Low', 'Medium', 'High'])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

1. **Comparing The Models**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

import pickle

from prettytable import PrettyTable

from datetime import datetime

# Load data

train = pd.read\_csv('trains.csv')

store = pd.read\_csv('stores.csv')

feature = pd.read\_csv('features.csv')

# Data Preprocessing and Merging

data = train.merge(feature, on=['Store', 'Date'], how='inner').merge(store, on=['Store'], how='inner')

# Fill NaNs

data['MarkDown1'] = data['MarkDown1'].replace(np.nan, 0)

data['MarkDown2'] = data['MarkDown2'].replace(np.nan, 0)

data['MarkDown3'] = data['MarkDown3'].replace(np.nan, 0)

data['MarkDown4'] = data['MarkDown4'].replace(np.nan, 0)

data['MarkDown5'] = data['MarkDown5'].replace(np.nan, 0)

# Filter data

data = data[data['Weekly\_Sales'] >= 0]

# Convert categorical variables to dummy/indicator variables

data = pd.get\_dummies(data, columns=['Type'])

data['Date'] = pd.to\_datetime(data['Date'])

# Extract features from date

data['month'] = data['Date'].dt.month

data['year'] = data['Date'].dt.year

data['dayofweek\_name'] = data['Date'].dt.day\_name()

data['is\_weekend'] = data['dayofweek\_name'].isin(['Sunday', 'Saturday']).astype(int)

# Drop unnecessary columns

data = data.drop(columns=['dayofweek\_name', 'Date'])

# Prepare features and target variable

X = data[["Store", "Dept", "Size", "IsHoliday\_x", "CPI", "Temperature", "Type\_B", "Type\_C", "month", "year", "is\_weekend"]]

y = data["Weekly\_Sales"]

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train the Random Forest model with specific hyperparameters

rf = RandomForestRegressor(n\_estimators=58, max\_depth=27, min\_samples\_split=3, min\_samples\_leaf=1)

rf.fit(X\_train, y\_train.ravel())

# Predict on test data

y\_pred = rf.predict(X\_test)

# Evaluation metrics

mse = mean\_squared\_error(y\_test, y\_pred)

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

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

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

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

print(f"R-squared (R²): {r2}")

# Cross-validation

cv = cross\_val\_score(rf, X, y.ravel(), cv=6)

cv\_mean = np.mean(cv)

print(f"Cross-Validation Score: {cv\_mean}")

# Save the model to disk

pickle.dump(rf, open('rf\_model.pkl', 'wb'))

# Display the evaluation results using PrettyTable

tb = PrettyTable()

tb.field\_names = ["Model", "Training Accuracy", "Testing Accuracy", "RMSE", "MAE/ MAD(Arima)"]

# Assuming you have the training accuracy and testing accuracy calculated elsewhere

training\_accuracy\_rf = 99.07 # Example value

testing\_accuracy\_rf = 96.72 # Example value

tb.add\_row(["Random Forest", training\_accuracy\_rf, testing\_accuracy\_rf, np.sqrt(mse), mae])

tb.add\_row(["Decision Tree", 100.00, 94.56, 5323.15, 2068.02])

tb.add\_row(["XgBoost", 94.12, 94.04, 5572.25, 3104.22])

tb.add\_row(["ARIMA", '-', '-', 685.54, 446.99])

print(tb)