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, '%')
```

2. 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()
```

3.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}")
```

4. 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, '%')
```

5. 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)
```

```python
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()
```

6. 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)
```