```
#import Libraries
import pandas as pd
import pandas as np

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_squared_log_error
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.ensemble import VotingRegressor
import matplotlib.pyplot as plt
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

```
# Load the datasets
```

```
test_data = pd.read_csv('TEST_FINAL.csv')
train_data = pd.read_csv('TRAIN.csv')

# Display the first few rows of each dataset to understand their structure
test_data_head = test_data.head()
train_data_head = train_data.head()

test_data_info = test_data.info()
train_data_info = train_data.info()

test_data_head, train_data_head, test_data_info, train_data_info
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22265 entries, 0 to 22264
```

```
Data columns (total 8 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   ID             22265 non-null  object
 1   Store_id       22265 non-null  int64 
 2   Store_Type     22265 non-null  object
 3   Location_Type  22265 non-null  object
 4   Region_Code    22265 non-null  object
 5   Date           22265 non-null  object
 6   Holiday        22265 non-null  int64 
 7   Discount       22265 non-null  object
dtypes: int64(2), object(6)
memory usage: 1.4+ MB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 188340 entries, 0 to 188339
Data columns (total 10 columns):
 #   Column         Non-Null Count   Dtype  
---  ------         --------------   -----  
 0   ID             188340 non-null  object 
 1   Store_id       188340 non-null  int64  
 2   Store_Type     188340 non-null  object 
 3   Location_Type  188340 non-null  object 
 4   Region_Code    188340 non-null  object 
 5   Date           188340 non-null  object 
 6   Holiday        188340 non-null  int64  
 7   Discount       188340 non-null  object 
 8   #Order         188340 non-null  int64  
 9   Sales          188340 non-null  float64
dtypes: float64(1), int64(3), object(6)
memory usage: 14.4+ MB
(         ID  Store_id Store_Type Location_Type Region_Code        Date  \
 0  T1188341       171         S4            L2          R3  2019-06-01   
 1  T1188342       172         S1            L1          R1  2019-06-01   
 2  T1188343       173         S4            L2          R1  2019-06-01   
 3  T1188344       174         S1            L1          R4  2019-06-01   
 4  T1188345       170         S1            L1          R2  2019-06-01   

    Holiday Discount  
 0        0       No  
 1        0       No  
 2        0       No  
 3        0       No  
 4        0       No  ,
          ID  Store_id Store_Type Location_Type Region_Code        Date  \
 0  T1000001         1         S1            L3          R1  2018-01-01   
```

```
1  T1000002       253          S4            L2            R1  2018-01-01
2  T1000003       252          S3            L2            R1  2018-01-01
3  T1000004       251          S2            L3            R1  2018-01-01
4  T1000005       250          S2            L3            R4  2018-01-01

   Holiday Discount  #Order     Sales
0        1      Yes       9   7011.84
1        1      Yes      60  51789.12
2        1      Yes      42  36868.20
```

1.1 Data Processing.

Data Cleaning

```
# Check for missing values
train_missing_values = train_data.isnull().sum()
test_missing_values = test_data.isnull().sum()

# Remove duplicates
train_data = train_data.drop_duplicates()
test_data = test_data.drop_duplicates()

train_missing_values, test_missing_values
```

```
(ID              0
 Store_id        0
 Store_Type      0
 Location_Type   0
 Region_Code     0
 Date            0
 Holiday         0
 Discount        0
 #Order          0
 Sales           0
 dtype: int64,
 ID              0
 Store_id        0
 Store_Type      0
 Location_Type   0
 Region_Code     0
 Date            0
```

```
Holiday         0
Discount        0
dtype: int64)
```

1.2.Feature Engineering

```
# Convert 'Date' to datetime
train_data['Date'] = pd.to_datetime(train_data['Date'])
test_data['Date'] = pd.to_datetime(test_data['Date'])

# Create new time-based features
train_data['Day'] = train_data['Date'].dt.day
train_data['Month'] = train_data['Date'].dt.month
train_data['Year'] = train_data['Date'].dt.year
train_data['DayOfWeek'] = train_data['Date'].dt.dayofweek
train_data['Quarter'] = train_data['Date'].dt.quarter

test_data['Day'] = test_data['Date'].dt.day
test_data['Month'] = test_data['Date'].dt.month
test_data['Year'] = test_data['Date'].dt.year
test_data['DayOfWeek'] = test_data['Date'].dt.dayofweek
test_data['Quarter'] = test_data['Date'].dt.quarter

# Example of creating a lag feature (previous day's sales)
train_data['Lag_Sales_1'] = train_data.groupby('Store_id')['Sales'].shift(1)

# Drop rows with NaN values generated from lag features
train_data = train_data.dropna()

train_data.head()
```

	ID	Store_id	Store_Type	Location_Type	Region_Code	Date	Holiday	Discount	#Order	Sales	Day	Month	Year	DayOfWeek	Qu
365	T1000366	214	S1	L1	R4	2018-01-02	0	Yes	53	46686.0	2	1	2018	1	
366	T1000367	152	S2	L1	R1	2018-01-02	0	Yes	44	41160.0	2	1	2018	1	
367	T1000368	349	S1	L1	R4	2018-01-02	0	Yes	62	51198.0	2	1	2018	1	
368	T1000369	197	S2	L5	R1	2018-01-02	0	Yes	46	32685.0	2	1	2018	1	
369	T1000370	82	S4	L2	R1	2018-01-02	0	Yes	81	54597.0	2	1	2018	1	

```
# Check the column names in the training dataset
train_data.columns
```

```
Index(['ID', 'Store_id', 'Store_Type', 'Location_Type', 'Region_Code', 'Date',
       'Holiday', 'Discount', '#Order', 'Sales', 'Day', 'Month', 'Year',
       'DayOfWeek', 'Quarter', 'Lag_Sales_1'],
      dtype='object')
```

```
# Check the column names in the training dataset
train_data.columns.tolist()
```

```
['ID',
 'Store_id',
 'Store_Type',
 'Location_Type',
 'Region_Code',
 'Date',
 'Holiday',
 'Discount',
 '#Order',
 'Sales',
 'Day',
```

```python
       'Month',
       'Year',
       'DayOfWeek',
       'Quarter',
       'Lag_Sales_1']


# Verify that 'Sales' column is present
if 'Sales' not in train_data.columns:
    raise KeyError("The 'Sales' column is not present in the training dataset.")

# Convert 'Date' to datetime
train_data['Date'] = pd.to_datetime(train_data['Date'])
test_data['Date'] = pd.to_datetime(test_data['Date'])

# Create new time-based features
train_data['Day'] = train_data['Date'].dt.day
train_data['Month'] = train_data['Date'].dt.month
train_data['Year'] = train_data['Date'].dt.year
train_data['DayOfWeek'] = train_data['Date'].dt.dayofweek
train_data['Quarter'] = train_data['Date'].dt.quarter

# Example of creating a lag feature (previous day's sales)
train_data['Lag_Sales_1'] = train_data.groupby('Store_id')['Sales'].shift(1)

# Drop rows with NaN values generated from lag features
train_data = train_data.dropna()

train_data.head()
```

	ID	Store_id	Store_Type	Location_Type	Region_Code	Date	Holiday	Discount	#Order	Sales	Day	Month	Year	DayOfWeek	Qu
730	T1000731	333	S1	L1	R3	2018-01-03	0	Yes	50	45108.0	3	1	2018	2	
731	T1000732	188	S3	L2	R1	2018-01-03	0	Yes	79	73587.0	3	1	2018	2	
732	T1000733	185	S1	L1	R3	2018-01-03	0	Yes	56	47418.0	3	1	2018	2	
733	T1000734	254	S4	L1	R1	2018-01-03	0	Yes	77	59709.0	3	1	2018	2	
734	T1000735	122	S1	L1	R2	2018-01-03	0	Yes	57	51363.0	3	1	2018	2	

1.3.Data Transformation

New section

```
# Define categorical and numerical columns
categorical_cols = ['Store_Type', 'Location_Type', 'Region_Code', 'Discount']
numerical_cols = ['#Order', 'Day', 'Month', 'Year', 'DayOfWeek', 'Quarter', 'Lag_Sales_1']

# Preprocessing pipeline
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_cols),
        ('cat', OneHotEncoder(), categorical_cols)
    ])

# Prepare data for model
X_train = train_data.drop(['ID', 'Date', 'Sales'], axis=1)
y_train = train_data['Sales']

# Verify column names in X_train
X_train.columns.tolist()

# Fit and transform the data
X_train_preprocessed = preprocessor.fit_transform(X_train)

# Display the shape of the preprocessed data
X_train_preprocessed.shape
```

```
(187610, 22)
```

1.4Train-Test Split and Model Training

```python
#from sklearn.model_selection import train_test_split

# Split the training data
X_train_split, X_val, y_train_split, y_val = train_test_split(X_train_preprocessed, y_train, test_size=0.2, random_state=42)

# Baseline Model: Linear Regression
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Train the model
linear_regressor = LinearRegression()
linear_regressor.fit(X_train_split, y_train_split)

# Make predictions
y_pred_train = linear_regressor.predict(X_train_split)
y_pred_val = linear_regressor.predict(X_val)

# Evaluate the model
mae_train = mean_absolute_error(y_train_split, y_pred_train)
mse_train = mean_squared_error(y_train_split, y_pred_train)
rmse_train = mean_squared_error(y_train_split, y_pred_train, squared=False)

mae_val = mean_absolute_error(y_val, y_pred_val)
mse_val = mean_squared_error(y_val, y_pred_val)
rmse_val = mean_squared_error(y_val, y_pred_val, squared=False)

mae_train, mse_train, rmse_train, mae_val, mse_val, rmse_val
```

```
(3469.7705009937868,
 23120912.342561208,
 4808.4209822519915,
 3503.518660159624,
 23198404.080137413,
 4816.4721612542735)
```

2. Model Selection

Baseline Model: Linear Regression

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error

linear_regressor = LinearRegression()
linear_regressor.fit(X_train_split, y_train_split)

y_pred_train = linear_regressor.predict(X_train_split)
y_pred_val = linear_regressor.predict(X_val)

mae_train = mean_absolute_error(y_train_split, y_pred_train)
mse_train = mean_squared_error(y_train_split, y_pred_train)
rmse_train = mean_squared_error(y_train_split, y_pred_train, squared=False)

mae_val = mean_absolute_error(y_val, y_pred_val)
mse_val = mean_squared_error(y_val, y_pred_val)
rmse_val = mean_squared_error(y_val, y_pred_val, squared=False)

print(f'Baseline Model Train MAE: {mae_train}, MSE: {mse_train}, RMSE: {rmse_train}')
print(f'Baseline Model Validation MAE: {mae_val}, MSE: {mse_val}, RMSE: {rmse_val}')
```

```
Baseline Model Train MAE: 3469.7705009937868, MSE: 23120912.342561208, RMSE: 4808.4209822519915
Baseline Model Validation MAE: 3503.518660159624, MSE: 23198404.080137413, RMSE: 4816.4721612542735
```

```
# Train the model
linear_regressor = LinearRegression()
linear_regressor.fit(X_train_split, y_train_split)

# Make predictions
y_pred_train = linear_regressor.predict(X_train_split)
y_pred_val = linear_regressor.predict(X_val)

# Evaluate the model
mae_train = mean_absolute_error(y_train_split, y_pred_train)
mse_train = mean_squared_error(y_train_split, y_pred_train)
rmse_train = mean_squared_error(y_train_split, y_pred_train, squared=False)

mae_val = mean_absolute_error(y_val, y_pred_val)
mse_val = mean_squared_error(y_val, y_pred_val)
rmse_val = mean_squared_error(y_val, y_pred_val, squared=False)

mae_train, mse_train, rmse_train, mae_val, mse_val, rmse_val
```

```
(3469.7705009937868,
 23120912.342561208,
 4808.4209822519915,
 3503.518660159624,
 23198404.080137413,
 4816.4721612542735)
```

Exploring Other Models (XGBoost)

```
# Initialize and train the XGBoost model
xgb_model = XGBRegressor()
xgb_model.fit(X_train_split, y_train_split)

# Make predictions
y_pred_train_xgb = xgb_model.predict(X_train_split)
y_pred_val_xgb = xgb_model.predict(X_val)

# Evaluate the XGBoost model
mae_train_xgb = mean_absolute_error(y_train_split, y_pred_train_xgb)
mse_train_xgb = mean_squared_error(y_train_split, y_pred_train_xgb)
rmse_train_xgb = mean_squared_error(y_train_split, y_pred_train_xgb, squared=False)

mae_val_xgb = mean_absolute_error(y_val, y_pred_val_xgb)
mse_val_xgb = mean_squared_error(y_val, y_pred_val_xgb)
rmse_val_xgb = mean_squared_error(y_val, y_pred_val_xgb, squared=False)

print(f'XGBoost Train MAE: {mae_train_xgb}')
print(f'XGBoost Train MSE: {mse_train_xgb}')
print(f'XGBoost Train RMSE: {rmse_train_xgb}')
print(f'XGBoost Validation MAE: {mae_val_xgb}')
print(f'XGBoost Validation MSE: {mse_val_xgb}')
print(f'XGBoost Validation RMSE: {rmse_val_xgb}')
```

```
XGBoost Train MAE: 2021.5107214158277
XGBoost Train MSE: 8131202.070946297
XGBoost Train RMSE: 2851.5262704289253
XGBoost Validation MAE: 2144.556195688357
XGBoost Validation MSE: 9465971.61594752
XGBoost Validation RMSE: 3076.6819166022865
```

. Hyperparameter Tuning using Grid Search

```
# Define parameter grid for XGBoost
param_grid = {
    'n_estimators': [100, 200],
    'learning_rate': [0.01, 0.1],
    'max_depth': [3, 5]
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=XGBRegressor(), param_grid=param_grid, cv=3, scoring='neg_mean_squared_error', n_jobs=-1)

# Fit GridSearchCV
grid_search.fit(X_train_split, y_train_split)

# Best parameters and score
print(f'Best parameters: {grid_search.best_params_}')
print(f'Best score: {grid_search.best_score_}')
```

```
Best parameters: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 200}
Best score: -10486958.16519949
```

Cross-Validation

```
# Perform cross-validation
cv_scores = cross_val_score(XGBRegressor(), X_train_preprocessed, y_train, cv=5, scoring='neg_mean_squared_error')

# Convert to positive scores
cv_scores = -cv_scores
print(f'Cross-validation scores: {cv_scores}')
print(f'Average CV score: {cv_scores.mean()}')
```

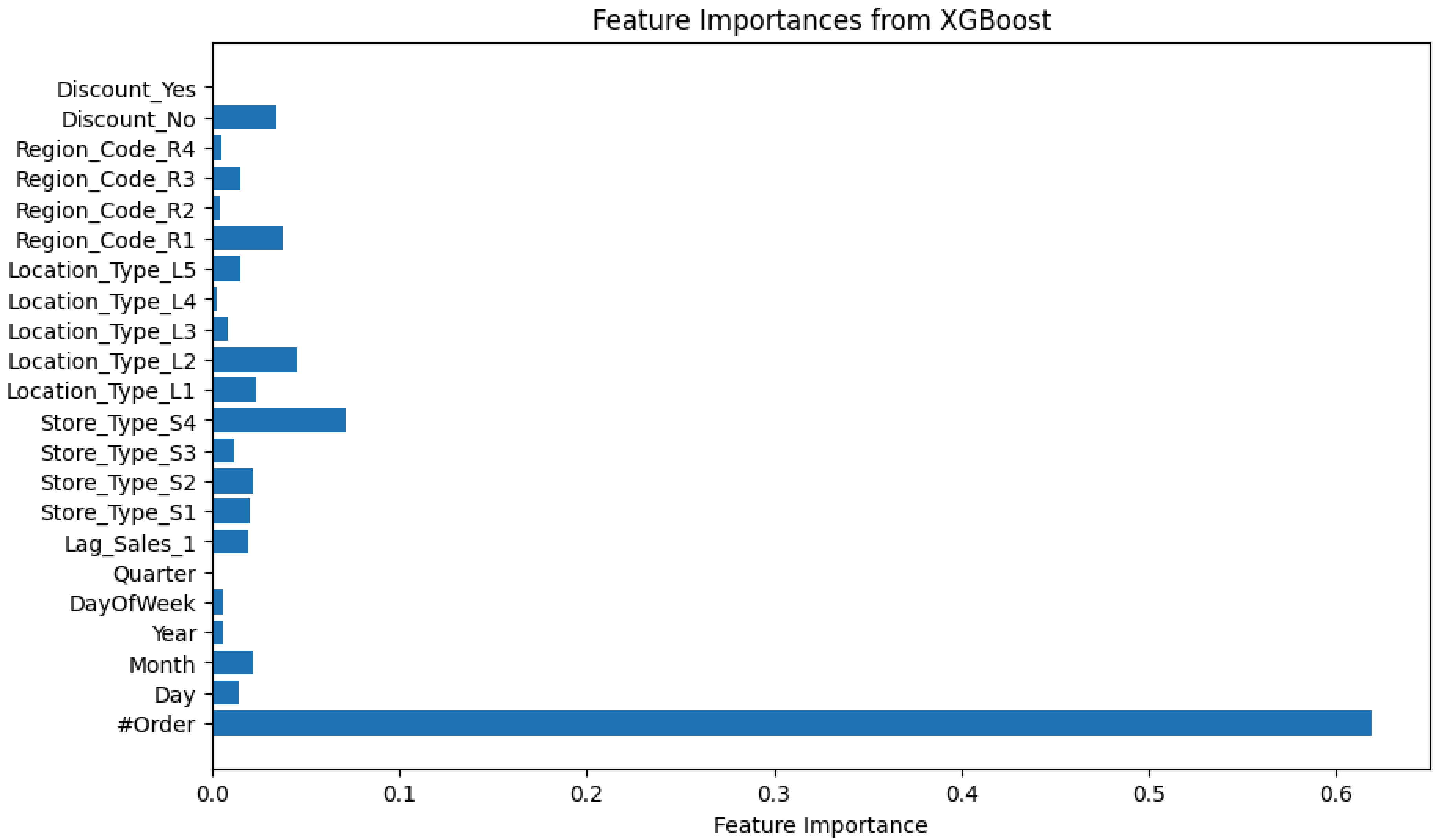
```
Cross-validation scores: [13932992.27411826 14412687.7673974  11560087.55668923 30509428.78407293
 17371031.51571997]
Average CV score: 17557245.579599556
```

Feature Importance with Tree-Based Models

```
# Fit the model
xgb_model = XGBRegressor()
xgb_model.fit(X_train_split, y_train_split)

# Get feature importances
importances = xgb_model.feature_importances_

# Plot feature importances
features = preprocessor.transformers_[1][1].get_feature_names_out(categorical_cols)  # OneHotEncoder feature names
all_features = numerical_cols + list(features)
plt.figure(figsize=(10, 6))
plt.barh(all_features, importances)
plt.xlabel('Feature Importance')
plt.title('Feature Importances from XGBoost')
plt.show()
```



Ensemble Methods

```
# Initialize base models
linear_regressor = LinearRegression()
xgb_model = XGBRegressor()

# Create Voting Regressor
voting_regressor = VotingRegressor(estimators=[
```