MSA 2024 Phase 2 - Part 2 Training and Evaluation

In [ ]:
```
!/opt/anaconda3/bin/python -m pip install joblib
!/opt/anaconda3/bin/python -m pip install Cython
!/opt/anaconda3/bin/python -m pip install pystan==2.19.1.1
!/opt/anaconda3/bin/python -m pip install prophet
!/opt/anaconda3/bin/python -m pip install xgboost
!/opt/anaconda3/bin/python -m pip install keras
!/opt/anaconda3/bin/python -m pip install tensorflow
!/opt/anaconda3/bin/python -m pip install statsmodels
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from keras.models import Sequential
from keras.layers import Input,LSTM, Dense
from statsmodels.tsa.arima.model import ARIMA
from prophet import Prophet
import tensorflow as tf
import sklearn.metrics as skm
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
```

1.Data normalization and definition of model evaluation indicators and model dictionaries

In [ ]:
```
# 归一化
# 需要归一化的列
columns_to_scale = ['Weekly_Sales', 'Temperature', 'Fuel_Price', 'CPI','Unemployment','Size']

# 初始化一个字典来存储每个周的归一化器
scalers = {}

for week, df in data_frames.items():
    # 使用副本进行操作，避免改变原始数据
    df_scaled = df.copy()
```

```
    # 为每个周创建一个新的归一化器
    scaler = MinMaxScaler()
    df_scaled[columns_to_scale] = scaler.fit_transform(df_scaled[columns_to_scale])

    # 将归一化器存储在字典中
    scalers[week] = scaler

    # 更新归一化后的数据回字典
    data_frames[week] = df_scaled

    # 查看归一化数据
    print(f"Normalized data set for {week}:\n{df_scaled.head()}")
```

In [ ]:
```
# Define a function to calculate the metrics including MSE\MAE\RMSE\R-square
def calculate_metrics(y_true,y_pred):
    mse=skm.mean_squared_error(y_true,y_pred)
    mae=skm.mean_absolute_error(y_true, y_pred)
    rmse=np.sqrt(mse)
    r2=skm.r2_score(y_true, y_pred)
    non_zero_idx = y_true != 0
    # Calculate MAPE
    mape = np.mean(np.abs((y_true[non_zero_idx] - y_pred[non_zero_idx]) / y_true[non_zero_idx])) * 100

    # Calculate SMAPE
    smape = 100 * np.mean(2 * np.abs(y_pred - y_true) / (np.abs(y_true) + np.abs(y_pred)))

    # Calculate MPE
    mpe = np.mean((y_true[non_zero_idx] - y_pred[non_zero_idx]) / y_true[non_zero_idx]) * 100

    return mse, mae, rmse, r2, mpe, mape, smape
```

In [ ]:
```
# Create LSTM model
def create_lstm_model(input_shape):
    model = Sequential([
        Input(shape=input_shape),
        LSTM(50, activation='relu'),
        Dense(1)
    ])
    model.compile(optimizer='adam', loss='mean_squared_error')
```

```python
    return model
# Creat neural network model
def create_nn_model(input_dim):
    model = Sequential()
    model.add(Dense(128, activation='relu', input_shape=(input_dim,)))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mean_squared_error')
    return model
```

In [ ]:
```python
# Calculate the number of features for LSTM
# Add some other needed features
additional_features = ['Store', 'Dept', 'Type', 'IsHoliday']

num_features = len(columns_to_scale) + len(additional_features)
lstm_model = create_lstm_model((1, num_features))
```

In [ ]:
```python
# Define the models dictionary
models = {
    'LR': LinearRegression(),
    'Ridge': Ridge(alpha=10.0),
    'Lasso': Lasso(alpha=0.1),
    'DT': DecisionTreeRegressor(max_depth=5, min_samples_split=50),
    'RF': RandomForestRegressor(n_estimators=200, max_depth=10, min_samples_split=10),
    'XGB': XGBRegressor(learning_rate=0.01, max_depth=10, n_estimators=500, subsample=0.8,colsample_bytree=0.8),
    'LSTM': create_lstm_model,
    'NN': create_nn_model
}
```

2. Load and split preprocessed data

3. Choose an algorithm

4. Train and test a model

5. Evaluate the model

In [ ]:
```
from joblib import dump

# 迭代每个DataFrame
for week, df in data_frames.items():
    # 获取当前周数，假设 week 格式为 'Weekly_Sales_Xw'
    current_week = int(week.split('_')[2][:-1])

    # 选择float64和int64类型的列
    numerical_df = df.select_dtypes(include=['float64', 'int64'])

    # 准备训练和测试数据
    if f'Weekly_Sales_{current_week}w' in numerical_df.columns:
        y = numerical_df[f'Weekly_Sales_{current_week}w']
        # 删除所有不相关的销售周和预测列
        columns_to_drop = [col for col in numerical_df.columns if 'Weekly_Sales_' in col  or 'Mark'  in col]
        X  = numerical_df.drop(columns=columns_to_drop)

        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)

        print(f"Processing model for week {current_week}")

        # 训练模型并进行预测
        for model_name, model in models.items():
            # 为当前模型准备数据
            X_train_next, X_test_next = X_train.copy(), X_test.copy()
            if model_name == 'LSTM':
                # 重新初始化 LSTM 模型
                lstm_model = model((1, X_train_next.shape[1]))
                X_train_reshaped = X_train_next.values.reshape((-1, 1, X_train_next.shape[1]))
                X_test_reshaped = X_test_next.values.reshape((-1, 1, X_test_next.shape[1]))
                lstm_model.fit(X_train_reshaped, y_train, epochs=50, verbose=0)
                y_train_pred = lstm_model.predict(X_train_reshaped).flatten()
                y_test_pred = lstm_model.predict(X_test_reshaped).flatten()
                model_to_save = lstm_model
            elif model_name == 'NN':
                # 重新初始化 NN 模型
                nn_model = model(X_train_next.shape[1])
                nn_model.fit(X_train_next, y_train, epochs=50, verbose=0)
                y_train_pred = nn_model.predict(X_train_next).flatten()
                y_test_pred = nn_model.predict(X_test_next).flatten()
```

```
            model_to_save = nn_model
        else:
            model.fit(X_train_next, y_train)
            y_train_pred = model.predict(X_train_next)
            y_test_pred = model.predict(X_test_next)
            model_to_save = model
        # 组合模型名和周数，创建唯一的文件名
        filename = f'model_{model_name}_{current_week}w.joblib'
        # 保存模型到指定的文件
        dump(model_to_save, filename)

        # 计算并打印评估指标
        mse, mae, rmse, r2, mpe, smape, mape = calculate_metrics(y_train, y_train_pred)
        print(f"Week {current_week}, Model {model_name} - Training Metrics: MSE={mse:.2f}, MAE={mae:.2f}, RMSE=

        mse, mae, rmse, r2, mpe,smape, mape = calculate_metrics(y_test, y_test_pred)
        print(f"Week {current_week}, Model {model_name} - Testing Metrics: MSE={mse:.2f}, MAE={mae:.2f}, RMSE={
    # 更新 data_frames 字典
    data_frames[week] = df
```

6. Summary

Data Selection and Preparation: The dataset was centered on the "w" store, where tables containing features, stores, and sales were merged to create a comprehensive dataset. Columns with high missing values, such as MarkDown1-5, were excluded to maintain data integrity. Data normalization was applied to standardize the range of variables, ensuring all features contributed equally to model performance. The dataset was split into training and testing sets, with a 70/30 split to balance model training and validation.

Choice of Algorithms: A diverse set of models was selected, including Linear Regression, Ridge, Lasso, Decision Tree, Random Forest, XGBoost, LSTM, Neural Network. These models cover a range of techniques from linear and regularized regression to tree-based and time-series forecasting, as well as deep learning. The selection aimed to explore different data patterns and relationships within the dataset. XGBoost was chosen for its efficiency and effectiveness in handling diverse data types.

Evaluation Metrics: The models were evaluated using metrics such as MSE, MAE, RMSE, R2, MPE, SMAPE, and MAPE. These metrics provided insights into model accuracy, error magnitude, and percentage errors, which are crucial for assessing forecast performance. SMAPE and MAPE were particularly relevant for financial forecasting, offering insights into the error proportions relative to actual values.

Training and Parameter Tuning: During initial training, some models showed signs of overfitting, indicated by discrepancies between training and testing metrics. To mitigate this, rigorous parameter tuning was conducted, particularly for XGBoost and neural networks. Adjustments to parameters like learning rate, tree depth, and the number of estimators were made to balance bias and variance, leading to improved model performance.

Model Saving and Selection: After training, each model was saved using joblib for future use. XGBoost emerged as the best-performing model, offering a strong balance between accuracy and computational efficiency. The final selection of XGBoost was based on its superior performance across multiple evaluation metrics.

Conclusion: This structured approach ensured the application of appropriate machine learning algorithms, leading to robust predictive models for sales forecasting. The thorough evaluation and refinement process, guided by multiple metrics, resulted in the selection of XGBoost as the most effective model, capable of generating reliable sales forecasts.