Predicting the Quarterly Revenue for Walmart

In [24]:

```python
# Importing Packages
import itertools
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
import matplotlib
import itertools
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
import matplotlib
import sklearn.preprocessing
from sklearn.metrics import r2_score
import keras

from keras.layers import Dense,Dropout,SimpleRNN,GRU, Bidirectional,LSTM
from tensorflow.keras.optimizers import SGD
from keras.models import Sequential
from sklearn.preprocessing import MinMaxScaler, StandardScaler

plt.style.use('fivethirtyeight')
matplotlib.rcParams['axes.labelsize'] = 14
matplotlib.rcParams['xtick.labelsize'] = 12
matplotlib.rcParams['ytick.labelsize'] = 12
matplotlib.rcParams['text.color'] = 'k'
```

In [25]:

```python
# Reading the Data
df=pd.read_excel('Walmart Quarterly Revenue.xlsx')
df.head()
```

Out[25]:

	Date	Quarterly Revenue
0	2009-01-31	108627
1	2009-04-30	94242
2	2009-07-31	100876
3	2009-10-31	99373
4	2010-01-31	113594

In [26]:

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 56 entries, 0 to 55
Data columns (total 2 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Date               56 non-null     datetime64[ns]
 1   Quarterly Revenue  56 non-null     int64
dtypes: datetime64[ns](1), int64(1)
memory usage: 1.0 KB
```

In [27]:

```python
# Setting Date as Index
df = df.set_index('Date')
df.head()
```

Out[27]:

Date	Quarterly Revenue
2009-01-31	108627
2009-04-30	94242
2009-07-31	100876
2009-10-31	99373
2010-01-31	113594

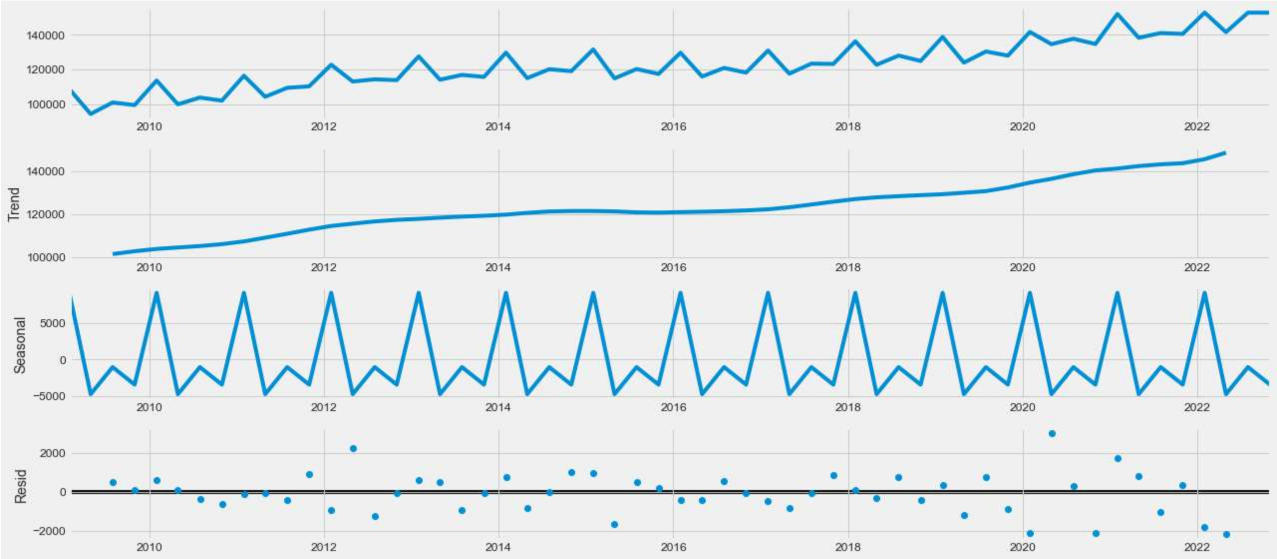
In [28]:

```python
# Plotting the data
df.plot(figsize=(16,4),legend=True)
plt.title('Walmart Quarterly Revenue')
plt.show()
```

In [29]:

```python
# Decomposition the data
from pylab import rcParams
rcParams['figure.figsize'] = 18, 8

decomposition = sm.tsa.seasonal_decompose(df, model = 'additive')
fig = decomposition.plot()
plt.show()
```



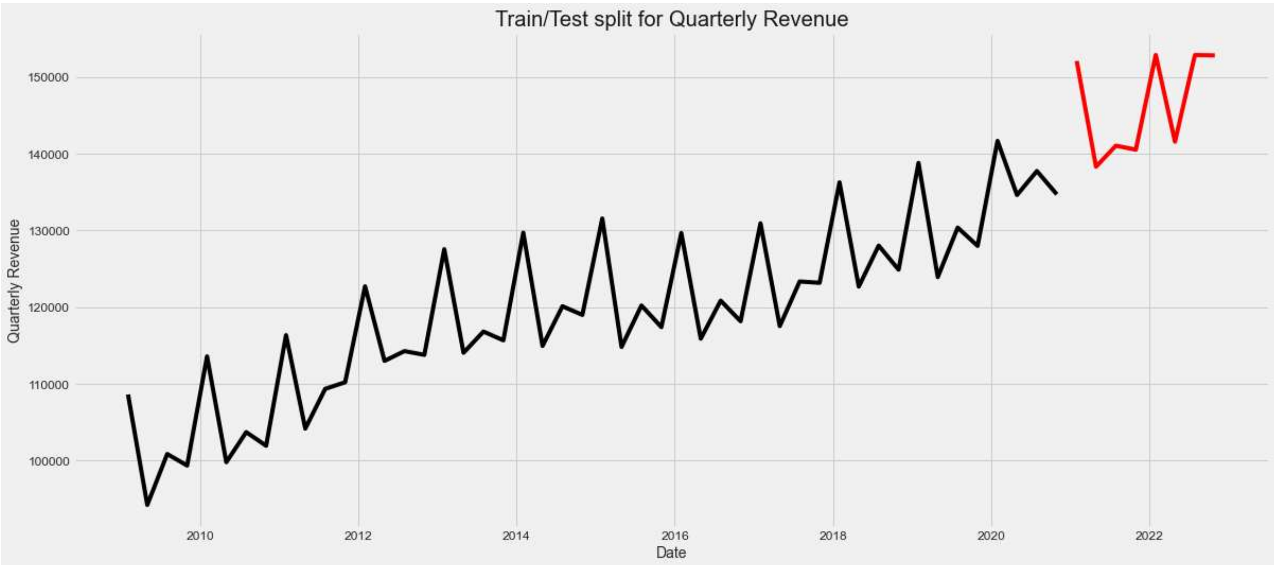
In [30]:

```
1  # Dividing the data into training and testing
2  # Ploting the data
3  import seaborn as sns
4  df['Date'] = df.index
5  train = df[df['Date'] < pd.to_datetime("2020-12", format='%Y-%m')]
6  train['train'] = train['Quarterly Revenue']
7  del train['Date']
8  del train['Quarterly Revenue']
9  test = df[df['Date'] >= pd.to_datetime("2020-12", format='%Y-%m')]
10 del test['Date']
11 test['test'] = test['Quarterly Revenue']
12 del test['Quarterly Revenue']
13 plt.plot(train, color = "black")
14 plt.plot(test, color = "red")
15 plt.title("Train/Test split for Quarterly Revenue")
16 plt.ylabel("Quarterly Revenue")
17 plt.xlabel('Date')
18 sns.set()
19 plt.show()
```

```
C:\Users\ravit\AppData\Local\Temp\ipykernel_14256\2575482981.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  train['train'] = train['Quarterly Revenue']
C:\Users\ravit\AppData\Local\Temp\ipykernel_14256\2575482981.py:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  test['test'] = test['Quarterly Revenue']
```



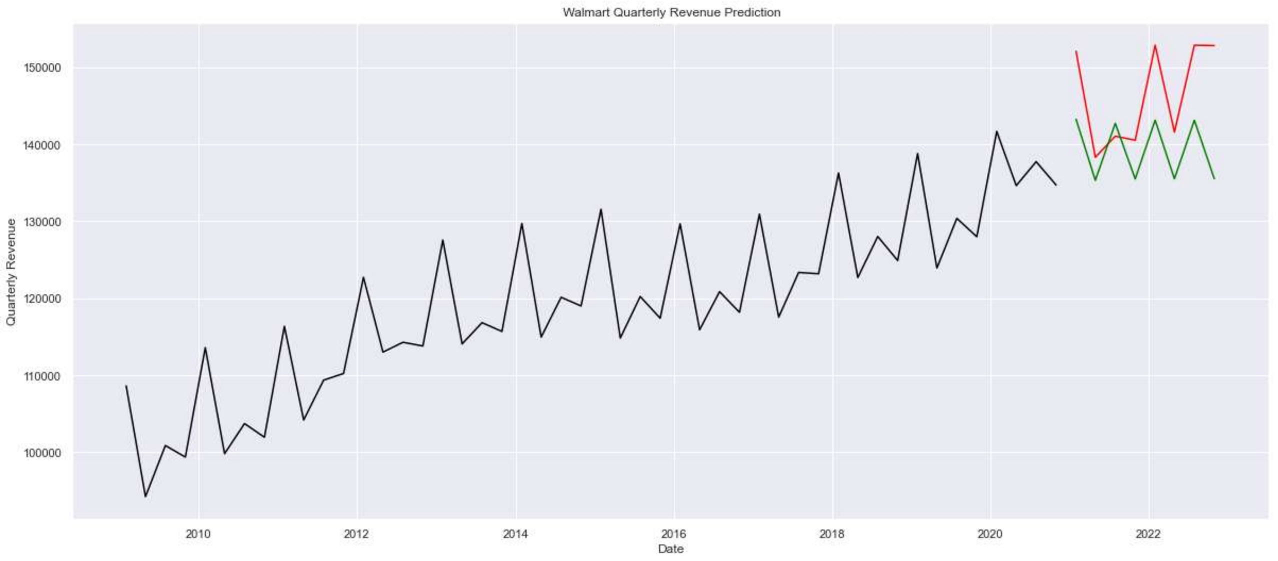
Arima Model

In [31]:
```python
# Applying ARIMA Model
from pmdarima.arima import auto_arima
model = auto_arima(train, trace=True, error_action='ignore', suppress_warnings=True)
model.fit(train)
forecast = model.predict(n_periods=len(test))
forecast = pd.DataFrame(forecast,index = test.index,columns=['Prediction'])
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=955.046, Time=0.20 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=1000.490, Time=0.01 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=987.989, Time=0.02 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=990.035, Time=0.04 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=998.653, Time=0.01 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.25 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=inf, Time=0.26 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.38 sec
 ARIMA(2,1,3)(0,0,0)[0] intercept   : AIC=953.871, Time=0.27 sec
 ARIMA(1,1,3)(0,0,0)[0] intercept   : AIC=952.720, Time=0.21 sec
 ARIMA(0,1,3)(0,0,0)[0] intercept   : AIC=991.409, Time=0.07 sec
 ARIMA(1,1,4)(0,0,0)[0] intercept   : AIC=952.424, Time=0.26 sec
 ARIMA(0,1,4)(0,0,0)[0] intercept   : AIC=984.050, Time=0.08 sec
 ARIMA(2,1,4)(0,0,0)[0] intercept   : AIC=952.498, Time=0.40 sec
 ARIMA(1,1,5)(0,0,0)[0] intercept   : AIC=950.471, Time=0.36 sec
 ARIMA(0,1,5)(0,0,0)[0] intercept   : AIC=980.089, Time=0.09 sec
 ARIMA(2,1,5)(0,0,0)[0] intercept   : AIC=951.493, Time=0.48 sec
 ARIMA(1,1,5)(0,0,0)[0]             : AIC=949.092, Time=0.30 sec
 ARIMA(0,1,5)(0,0,0)[0]             : AIC=978.866, Time=0.09 sec
 ARIMA(1,1,4)(0,0,0)[0]             : AIC=951.495, Time=0.26 sec
 ARIMA(2,1,5)(0,0,0)[0]             : AIC=953.488, Time=0.48 sec
 ARIMA(0,1,4)(0,0,0)[0]             : AIC=982.276, Time=0.08 sec
 ARIMA(2,1,4)(0,0,0)[0]             : AIC=951.561, Time=0.39 sec

Best model:  ARIMA(1,1,5)(0,0,0)[0]
Total fit time: 5.039 seconds
```

In [32]:
```python
# Plotting the prediction
plt.plot(train, color = "black")
plt.plot(test, color = "red")
plt.plot(forecast, color = "green")
plt.title(" Walmart Quarterly Revenue Prediction")
plt.ylabel("Quarterly Revenue")
plt.xlabel('Date')
sns.set()
plt.show()
```



In [33]:
```python
from math import sqrt
from sklearn.metrics import mean_squared_error
rms = sqrt(mean_squared_error(test,forecast))
print("RMSE: ", rms)
```

```
RMSE:  8948.314010936627
```

SARIMA Model

In [34]:
```python
df=pd.read_excel('Walmart Quarterly Revenue.xlsx')
df = df.set_index('Date')
```

In [35]:
```python
# set the typical ranges for p, d, q
p = d = q = range(0, 2)

#take all possible combination for p, d and q
pdq = list(itertools.product(p, d, q))
seasonal_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]

print('Examples of parameter combinations for Seasonal ARIMA...')
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[1]))
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[2]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[3]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[4]))
```

```
Examples of parameter combinations for Seasonal ARIMA...
SARIMAX: (0, 0, 1) x (0, 0, 1, 12)
SARIMAX: (0, 0, 1) x (0, 1, 0, 12)
SARIMAX: (0, 1, 0) x (0, 1, 1, 12)
SARIMAX: (0, 1, 0) x (1, 0, 0, 12)
```

In [36]:
```python
# Using Grid Search find the optimal set of parameters that yields the best performance
for param in pdq:
    for param_seasonal in seasonal_pdq:
        try:
            mod = sm.tsa.statespace.SARIMAX(df, order = param, seasonal_order = param_seasonal, enforce_stationary = False,enf
            result = mod.fit()
            print('SARIMA{}x{}12 - AIC:{}'.format(param, param_seasonal, result.aic))
        except:
            continue
```

```
SARIMA(0, 0, 0)x(0, 0, 0, 12)12 - AIC:1474.5164167195842

R:\Anaconda\envs\general\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: No frequency information was p
rovided, so inferred frequency Q-OCT will be used.
  self._init_dates(dates, freq)
R:\Anaconda\envs\general\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: No frequency information was p
rovided, so inferred frequency Q-OCT will be used.
  self._init_dates(dates, freq)
R:\Anaconda\envs\general\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: No frequency information was p
rovided, so inferred frequency Q-OCT will be used.
  self._init_dates(dates, freq)
R:\Anaconda\envs\general\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: No frequency information was p
rovided, so inferred frequency Q-OCT will be used.
  self._init_dates(dates, freq)
R:\Anaconda\envs\general\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: No frequency information was p
rovided, so inferred frequency Q-OCT will be used.
  self._init_dates(dates, freq)
R:\Anaconda\envs\general\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: No frequency information was p
rovided, so inferred frequency Q-OCT will be used.
  self._init_dates(dates, freq)
```

In [37]:
```python
#Fitting the SARIMA model using above optimal combination of p, d, q (optimal means combination at which we got lowest AIC sco

model = sm.tsa.statespace.SARIMAX(df, order = (1, 1, 1),
                                  seasonal_order = (1, 1, 0, 12)
                                 )
result = model.fit()
print(result.summary().tables[1])
```

```
R:\Anaconda\envs\general\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: No frequency information was prov
ided, so inferred frequency Q-OCT will be used.
  self._init_dates(dates, freq)
R:\Anaconda\envs\general\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: No frequency information was prov
ided, so inferred frequency Q-OCT will be used.
  self._init_dates(dates, freq)

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.5736      0.586     -0.978      0.328      -1.723       0.576
ma.L1          0.4940      0.632      0.781      0.435      -0.745       1.733
ar.S.L12       0.0019      0.164      0.011      0.991      -0.320       0.324
sigma2      6.155e+06   2.96e-08   2.08e+14      0.000    6.15e+06    6.15e+06
==============================================================================
```

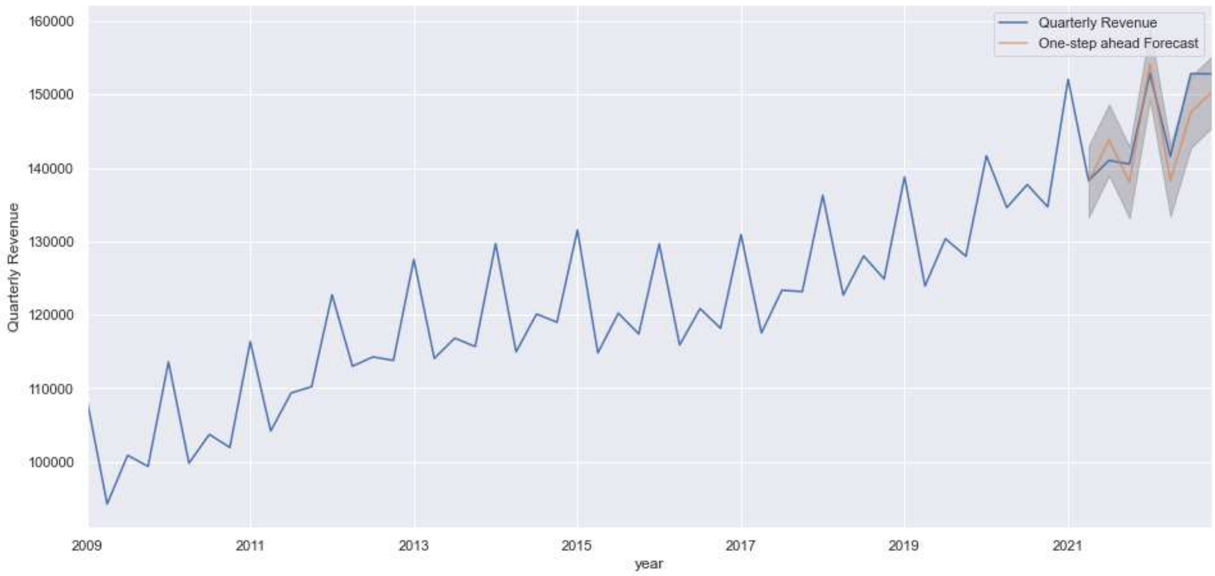
In [38]:
```python
prediction = result.get_prediction(start = pd.to_datetime('2021-04-30'), dynamic = False)
prediction_ci = prediction.conf_int()
prediction_ci
```

Out[38]:

	lower Quarterly Revenue	upper Quarterly Revenue
2021-04-30	133299.243433	143023.993306
2021-07-31	138972.339021	148697.088894
2021-10-31	133165.550097	142890.299970
2022-01-31	149300.397194	159025.147067
2022-04-30	133392.899885	143117.649758
2022-07-31	142751.969848	152476.719722
2022-10-31	145428.832133	155153.582006

In [39]:

```python
#Visualize the forecasting
ax = df['2009':].plot(label = 'observed')
prediction.predicted_mean.plot(ax = ax, label = 'One-step ahead Forecast', alpha = 0.7, figsize = (14, 7))
ax.fill_between(prediction_ci.index, prediction_ci.iloc[:, 0], prediction_ci.iloc[:, 1], color = 'k', alpha = 0.2)
ax.set_xlabel("year")
ax.set_ylabel('Quarterly Revenue')
plt.legend()
plt.show()
```



In [40]:

```python
# Evaluation metrics are Squared Mean Error(SME) and Root Mean Squared Error(RMSE)

from sklearn.metrics import mean_squared_error

y_hat = prediction.predicted_mean
y_truth = df['2021-04-30':]
mse = mean_squared_error(y_truth,y_hat)
rmse = np.sqrt(mse)

print('The Mean Squared Error of our forecasts is', mse)
print('The Root Mean Squared Error of our forecasts is', rmse)
```

```
The Mean Squared Error of our forecasts is 8648353.268182157
The Root Mean Squared Error of our forecasts is 2940.808267837629
```

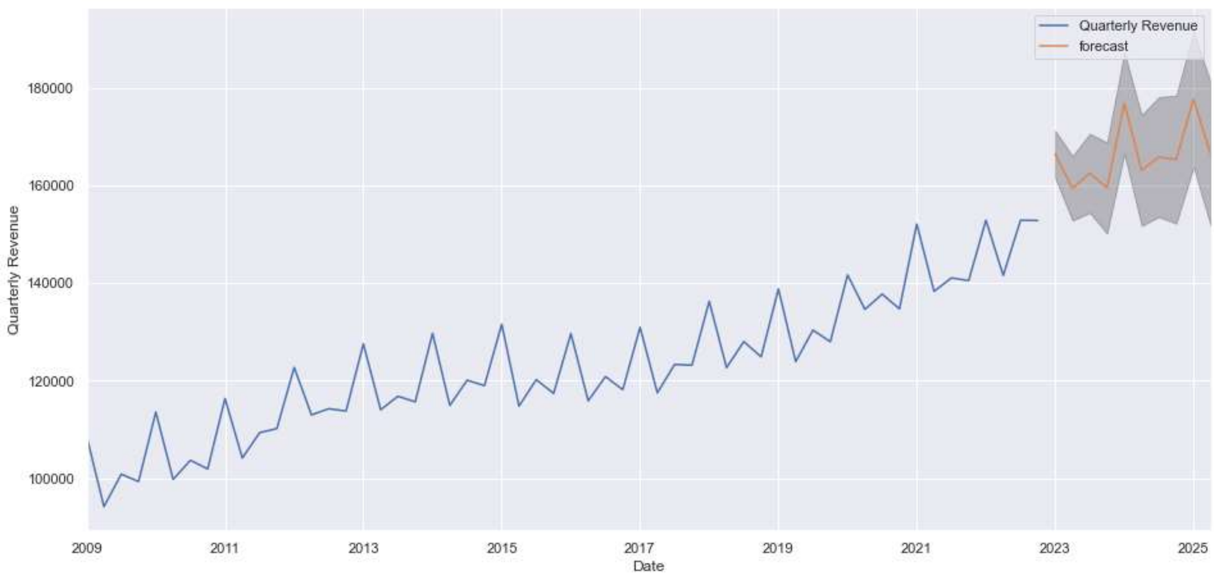
In [41]:

```python
# forcasting for out of sample data
pred_uc = result.get_forecast(steps = 10)
pred_ci = pred_uc.conf_int()

ax = df.plot(label = 'observed', figsize = (14, 7))
pred_uc.predicted_mean.plot(ax = ax, label = 'forecast')
ax.fill_between(pred_ci.index, pred_ci.iloc[:, 0], pred_ci.iloc[:, 1], color = 'k', alpha = 0.25)
ax.set_xlabel('Date')
ax.set_ylabel('Quarterly Revenue')

plt.legend()
plt.show()

```



DNN MODEL

In [42]:
```
def convert2matrix(data_arr, look_back):
    X, Y =[], []
    for i in range(len(data_arr)-look_back):
        d=i+look_back
        X.append(data_arr[i:d,0])
        Y.append(data_arr[d,0])
    return np.array(X).astype('int'), np.array(Y).astype('int')
```

In [43]:
```
df=pd.read_excel('Walmart Quarterly Revenue.xlsx')

df = df.set_index('Date')

```

In [44]:
```
df1 = df
#Split data set into testing dataset and train dataset
train_size = 49
train, test =df1.values[0:train_size,:],df1.values[train_size:len(df1.values),:]
# setup look_back window
look_back = 4
#convert dataset into right shape in order to input into the DNN
trainX, trainY = convert2matrix(train, look_back)
testX, testY = convert2matrix(test, look_back)
```

In [45]:
```
from keras.models import Sequential
from keras.layers import Dense
def model_dnn(look_back):
    model=Sequential()
    model.add(Dense(units=32, input_dim=look_back, activation='relu'))
    model.add(Dense(8, activation='relu'))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error',  optimizer='adam',metrics = ['mse', 'mae'])
    return model
```

In [46]:
```
model=model_dnn(look_back)
history=model.fit(trainX,trainY, epochs=500, batch_size=4, verbose=1, validation_data=(testX,testY),shuffle=False)
```

```
Epoch 1/500
12/12 [==============================] - 1s 16ms/step - loss: 17379080192.0000 - mse: 17379080192.0000 - mae: 131621.1875 - va
l_loss: 21999052800.0000 - val_mse: 21999052800.0000 - val_mae: 148240.0156
Epoch 2/500
12/12 [==============================] - 0s 3ms/step - loss: 12832726016.0000 - mse: 12832726016.0000 - mae: 113052.1875 - val
_loss: 15932937216.0000 - val_mse: 15932937216.0000 - val_mae: 126136.8750
Epoch 3/500
12/12 [==============================] - 0s 3ms/step - loss: 9045407744.0000 - mse: 9045407744.0000 - mae: 94868.5781 - val_lo
ss: 10838700032.0000 - val_mse: 10838700032.0000 - val_mae: 104005.6172
Epoch 4/500
12/12 [==============================] - 0s 3ms/step - loss: 5943562752.0000 - mse: 5943562752.0000 - mae: 76815.9219 - val_lo
ss: 6807878144.0000 - val_mse: 6807878144.0000 - val_mae: 82382.6172
Epoch 5/500
12/12 [==============================] - 0s 3ms/step - loss: 3581512448.0000 - mse: 3581512448.0000 - mae: 59482.8984 - val_lo
ss: 3852875776.0000 - val_mse: 3852875776.0000 - val_mae: 61911.6406
Epoch 6/500
12/12 [==============================] - 0s 3ms/step - loss: 1903954176.0000 - mse: 1903954176.0000 - mae: 43107.1484 - val_lo
ss: 1883590656.0000 - val_mse: 1883590656.0000 - val_mae: 43177.4336
Epoch 7/500
```

In [47]:
```
def model_loss(history):
    plt.figure(figsize=(8,4))
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Test Loss')
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epochs')
    plt.legend(loc='upper right')
    plt.show();
```

In [48]:
```
train_score = model.evaluate(trainX, trainY, verbose=0)
print('Train Root Mean Squared Error(RMSE): %.2f; Train Mean Absolute Error(MAE) : %.2f '
% (np.sqrt(train_score[1]), train_score[2]))
test_score = model.evaluate(testX, testY, verbose=0)
print(train_score)
print(test_score)
print('Test Root Mean Squared Error(RMSE): %.2f; Test Mean Absolute Error(MAE) : %.2f '
% (np.sqrt(test_score[1]), test_score[2]))
```

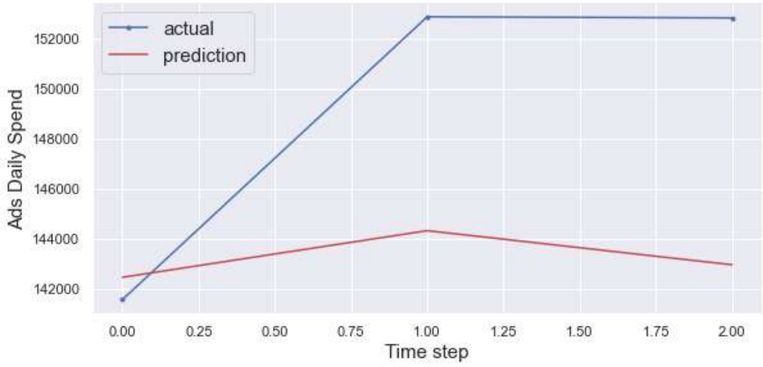
```
Train Root Mean Squared Error(RMSE): 2998.15; Train Mean Absolute Error(MAE) : 2389.67
[8988885.0, 8988885.0, 2389.6708984375]
[56950688.0, 56950688.0, 6428.11962890625]
Test Root Mean Squared Error(RMSE): 7546.57; Test Mean Absolute Error(MAE) : 6428.12
```

In [49]:

```
def prediction_plot(testY, test_predict):
     len_prediction=[x for x in range(len(testY))]
     plt.figure(figsize=(8,4))
     plt.plot(len_prediction, testY[:8], marker='.', label="actual")
     plt.plot(len_prediction, test_predict[:8], 'r', label="prediction")
     plt.tight_layout()
     sns.despine(top=True)
     plt.subplots_adjust(left=0.07)
     plt.ylabel('Ads Daily Spend', size=15)
     plt.xlabel('Time step', size=15)
     plt.legend(fontsize=15)
     plt.show();
```

In [50]:

```
test_predict = model.predict(testX)
prediction_plot(testY, test_predict)
```



GRU and BiLSTM Models

In [51]:

```
df=pd.read_excel('Walmart Quarterly Revenue.xlsx')
df.head()
```

Out[51]:

	Date	Quarterly Revenue
0	2009-01-31	108627
1	2009-04-30	94242
2	2009-07-31	100876
3	2009-10-31	99373
4	2010-01-31	113594

In [52]:

```
df = df.set_index('Date')
df.head()
```

Out[52]:

	Quarterly Revenue
Date	
2009-01-31	108627
2009-04-30	94242
2009-07-31	100876
2009-10-31	99373
2010-01-31	113594

In [53]:

```
# Split train data and test data
train_size = int(len(df)*0.8)

train_data = df.iloc[:train_size]
test_data = df.iloc[train_size:]
```

In [54]:

```
scaler = MinMaxScaler().fit(train_data)
train_scaled = scaler.transform(train_data)
test_scaled = scaler.transform(test_data)
```

In [55]:

```python
# Create input dataset
def create_dataset (X, look_back = 1):
    Xs, ys = [], []

    for i in range(len(X)-look_back):
        v = X[i:i+look_back]
        Xs.append(v)
        ys.append(X[i+look_back])

    return np.array(Xs), np.array(ys)
LOOK_BACK = 4
X_train, y_train = create_dataset(train_scaled,LOOK_BACK)
X_test, y_test = create_dataset(test_scaled,LOOK_BACK)
# Print data shape
print('X_train.shape: ', X_train.shape)
print('y_train.shape: ', y_train.shape)
print('X_test.shape: ', X_test.shape)
print('y_test.shape: ', y_test.shape)
```

```
X_train.shape:  (40, 4, 1)
y_train.shape:  (40, 1)
X_test.shape:  (8, 4, 1)
y_test.shape:  (8, 1)
```

In [56]:

```python
# Create BiLSTM model
def create_bilstm(units):
    model = Sequential()
    # Input layer
    model.add(Bidirectional(
              LSTM(units = units, return_sequences=True),
              input_shape=(X_train.shape[1], X_train.shape[2])))
    # Hidden layer
    model.add(Bidirectional(LSTM(units = units)))
    model.add(Dense(1))
    #Compile model
    model.compile(optimizer='adam',loss='mse')
    return model
model_bilstm = create_bilstm(64)
# Create GRU model
def create_gru(units):
    model = Sequential()
    # Input layer
    model.add(GRU (units = units, return_sequences = True,
    input_shape = [X_train.shape[1], X_train.shape[2]]))
    model.add(Dropout(0.2))
    # Hidden layer
    model.add(GRU(units = units))
    model.add(Dropout(0.2))
    model.add(Dense(units = 1))
    #Compile model
    model.compile(optimizer='adam',loss='mse')
    return model
model_gru = create_gru(64)
```

In [57]:

```
def fit_model(model):
    early_stop = keras.callbacks.EarlyStopping(monitor = 'val_loss',
                                               patience = 10)
    history = model.fit(X_train, y_train, epochs = 100,
                        validation_split = 0.2,
                        batch_size = 16, shuffle = False,
                        callbacks = [early_stop])
    return history
history_gru = fit_model(model_gru)
```

```
Epoch 1/100
2/2 [==============================] - 3s 604ms/step - loss: 0.2240 - val_loss: 0.3596
Epoch 2/100
2/2 [==============================] - 0s 28ms/step - loss: 0.1307 - val_loss: 0.2115
Epoch 3/100
2/2 [==============================] - 0s 27ms/step - loss: 0.0726 - val_loss: 0.1028
Epoch 4/100
2/2 [==============================] - 0s 27ms/step - loss: 0.0423 - val_loss: 0.0369
Epoch 5/100
2/2 [==============================] - 0s 26ms/step - loss: 0.0261 - val_loss: 0.0171
Epoch 6/100
2/2 [==============================] - 0s 26ms/step - loss: 0.0322 - val_loss: 0.0242
Epoch 7/100
2/2 [==============================] - 0s 30ms/step - loss: 0.0418 - val_loss: 0.0271
Epoch 8/100
2/2 [==============================] - 0s 25ms/step - loss: 0.0453 - val_loss: 0.0213
Epoch 9/100
2/2 [==============================] - 0s 24ms/step - loss: 0.0284 - val_loss: 0.0175
Epoch 10/100
2/2 [==============================] - 0s 26ms/step - loss: 0.0182 - val_loss: 0.0204
Epoch 11/100
2/2 [==============================] - 0s 29ms/step - loss: 0.0209 - val_loss: 0.0283
Epoch 12/100
2/2 [==============================] - 0s 26ms/step - loss: 0.0186 - val_loss: 0.0372
Epoch 13/100
2/2 [==============================] - 0s 27ms/step - loss: 0.0196 - val_loss: 0.0432
Epoch 14/100
2/2 [==============================] - 0s 26ms/step - loss: 0.0233 - val_loss: 0.0445
Epoch 15/100
2/2 [==============================] - 0s 28ms/step - loss: 0.0242 - val_loss: 0.0411
```

In [58]:

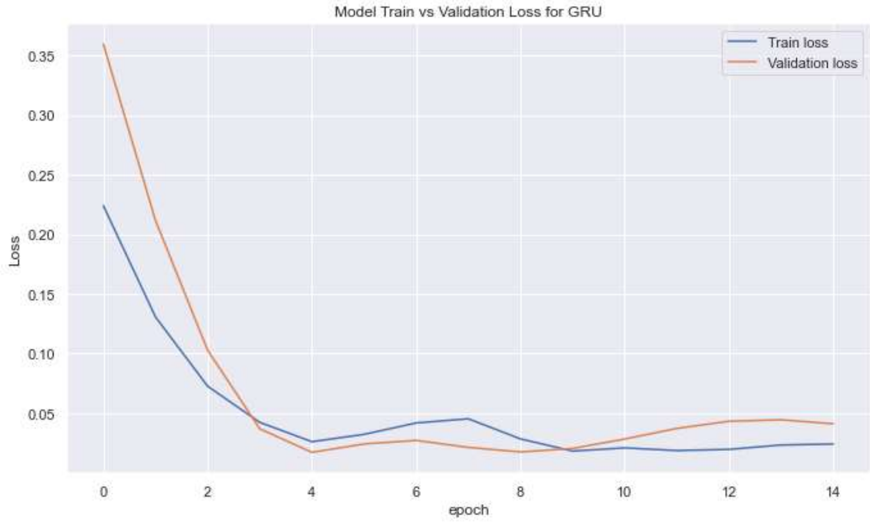
```
history_bilstm = fit_model(model_bilstm)
```

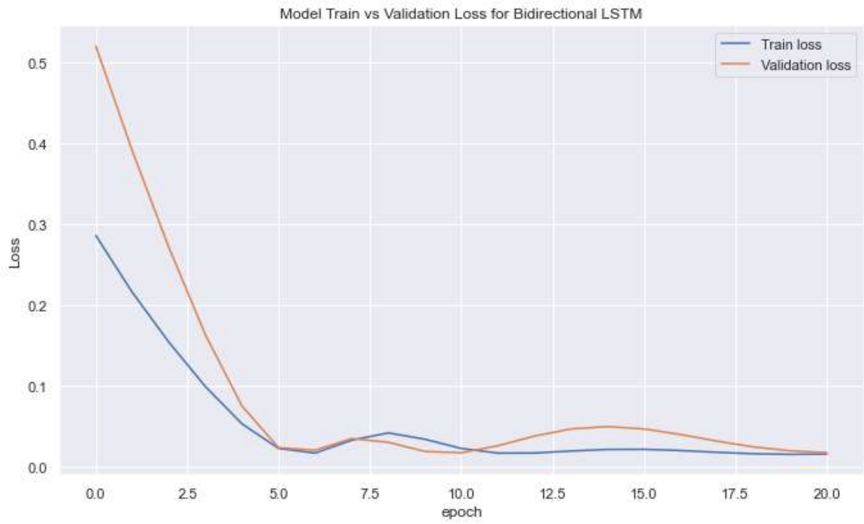
```
Epoch 1/100
2/2 [==============================] - 6s 1s/step - loss: 0.2862 - val_loss: 0.5206
Epoch 2/100
2/2 [==============================] - 0s 29ms/step - loss: 0.2158 - val_loss: 0.3906
Epoch 3/100
2/2 [==============================] - 0s 29ms/step - loss: 0.1541 - val_loss: 0.2712
Epoch 4/100
2/2 [==============================] - 0s 34ms/step - loss: 0.0992 - val_loss: 0.1632
Epoch 5/100
2/2 [==============================] - 0s 27ms/step - loss: 0.0531 - val_loss: 0.0751
Epoch 6/100
2/2 [==============================] - 0s 30ms/step - loss: 0.0227 - val_loss: 0.0236
Epoch 7/100
2/2 [==============================] - 0s 28ms/step - loss: 0.0170 - val_loss: 0.0206
Epoch 8/100
2/2 [==============================] - 0s 29ms/step - loss: 0.0329 - val_loss: 0.0349
Epoch 9/100
2/2 [==============================] - 0s 28ms/step - loss: 0.0420 - val_loss: 0.0304
Epoch 10/100
2/2 [==============================] - 0s 31ms/step - loss: 0.0341 - val_loss: 0.0190
Epoch 11/100
2/2 [==============================] - 0s 30ms/step - loss: 0.0227 - val_loss: 0.0173
Epoch 12/100
2/2 [==============================] - 0s 32ms/step - loss: 0.0169 - val_loss: 0.0260
Epoch 13/100
2/2 [==============================] - 0s 30ms/step - loss: 0.0170 - val_loss: 0.0380
Epoch 14/100
2/2 [==============================] - 0s 35ms/step - loss: 0.0195 - val_loss: 0.0468
Epoch 15/100
2/2 [==============================] - 0s 30ms/step - loss: 0.0214 - val_loss: 0.0497
Epoch 16/100
2/2 [==============================] - 0s 32ms/step - loss: 0.0216 - val_loss: 0.0468
Epoch 17/100
2/2 [==============================] - 0s 33ms/step - loss: 0.0201 - val_loss: 0.0399
Epoch 18/100
2/2 [==============================] - 0s 31ms/step - loss: 0.0179 - val_loss: 0.0317
Epoch 19/100
2/2 [==============================] - 0s 30ms/step - loss: 0.0161 - val_loss: 0.0245
Epoch 20/100
2/2 [==============================] - 0s 30ms/step - loss: 0.0154 - val_loss: 0.0198
Epoch 21/100
2/2 [==============================] - 0s 29ms/step - loss: 0.0158 - val_loss: 0.0175
```

In [59]:

```python
def plot_loss (history, model_name):
    plt.figure(figsize = (10, 6))
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model Train vs Validation Loss for ' + model_name)
    plt.ylabel('Loss')
    plt.xlabel('epoch')
    plt.legend(['Train loss', 'Validation loss'], loc='upper right')

plot_loss (history_gru, 'GRU')
plot_loss (history_bilstm, 'Bidirectional LSTM')
```

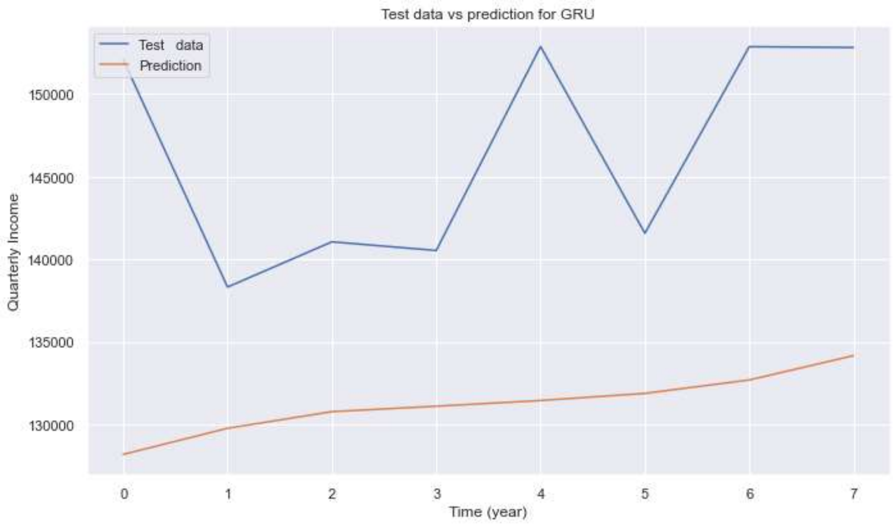


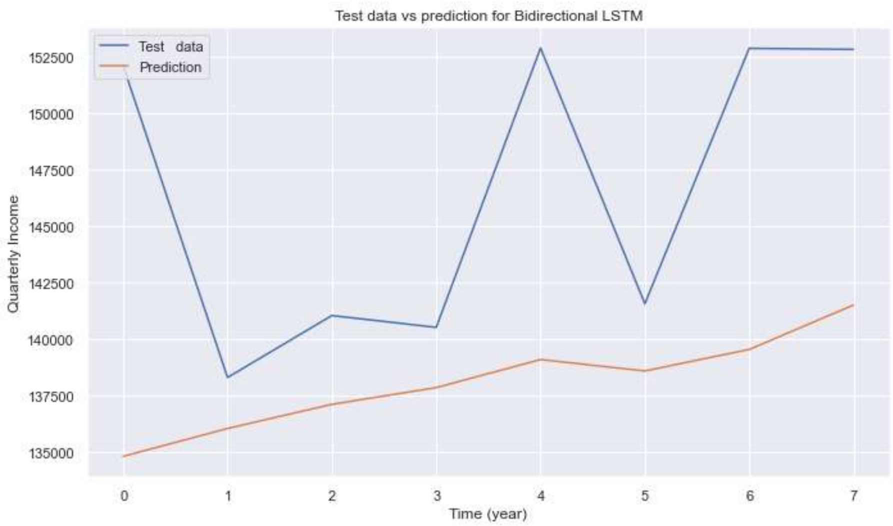


In [60]:

```python
# Make prediction
def prediction(model):
    prediction = model.predict(X_test)
    prediction = scaler.inverse_transform(prediction)
    return prediction
prediction_gru = prediction(model_gru)
prediction_bilstm = prediction(model_bilstm)
# Plot test data vs prediction
def plot_future(prediction, model_name, y_test):
    plt.figure(figsize=(10, 6))
    range_future = len(prediction)
    plt.plot(np.arange(range_future), np.array(scaler.inverse_transform(y_test)),
             label='Test   data')
    plt.plot(np.arange(range_future),
             np.array(prediction),label='Prediction')
    plt.title('Test data vs prediction for ' + model_name)
    plt.legend(loc='upper left')
    plt.xlabel('Time (year)')
    plt.ylabel('Quarterly Income')

plot_future(prediction_gru, 'GRU', y_test)
plot_future(prediction_bilstm, 'Bidirectional LSTM', y_test)
```





In [61]:

```python
def evaluate_prediction(predictions, actual, model_name):
    errors = predictions - actual
    mse = np.square(errors).mean()
    rmse = np.sqrt(mse)
    mae = np.abs(errors).mean()
    print(model_name + ':')
    print('Mean Absolute Error: {:.4f}'.format(mae))
    print('Root Mean Square Error: {:.4f}'.format(rmse))
    print('')
evaluate_prediction(prediction_gru, scaler.inverse_transform(y_test), 'GRU')
evaluate_prediction(prediction_bilstm, scaler.inverse_transform(y_test), 'Bidirectional LSTM')
```

```
GRU:
Mean Absolute Error: 15263.1260
Root Mean Square Error: 16381.6322

Bidirectional LSTM:
Mean Absolute Error: 8430.6992
Root Mean Square Error: 10176.7920
```

In [62]:

```
def evaluate_prediction(predictions, actual, model_name):
    errors = predictions - actual
    mse = np.square(errors).mean()
    rmse = np.sqrt(mse)
    mae = np.abs(errors).mean()
    print(model_name + ':')
    print('Mean Absolute Error: {:.4f}'.format(mae))
    print('Root Mean Square Error: {:.4f}'.format(rmse))
    print('')
evaluate_prediction(scaler.transform(prediction_gru), y_test, 'GRU')
evaluate_prediction(scaler.transform(prediction_bilstm), y_test, 'Bidirectional LSTM')
```

```
GRU:
Mean Absolute Error: 0.3426
Root Mean Square Error: 0.3677

Bidirectional LSTM:
Mean Absolute Error: 0.1892
Root Mean Square Error: 0.2284


R:\Anaconda\envs\general\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but MinMaxScale
r was fitted with feature names
  warnings.warn(
R:\Anaconda\envs\general\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but MinMaxScale
r was fitted with feature names
  warnings.warn(
```

In [ ]:

```

```