Predicting the Quarterly Revenue for Amazon

In [1]:

```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 [2]:

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

Out[2]:

	Date	Quarterly Revenue
0	2009-03-31	4889
1	2009-06-30	4651
2	2009-09-30	5449
3	2009-12-31	9520
4	2010-03-31	7131

In [3]:

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55 entries, 0 to 54
Data columns (total 2 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Date               55 non-null     datetime64[ns]
 1   Quarterly Revenue  55 non-null     int64
dtypes: datetime64[ns](1), int64(1)
memory usage: 1008.0 bytes
```

In [4]:

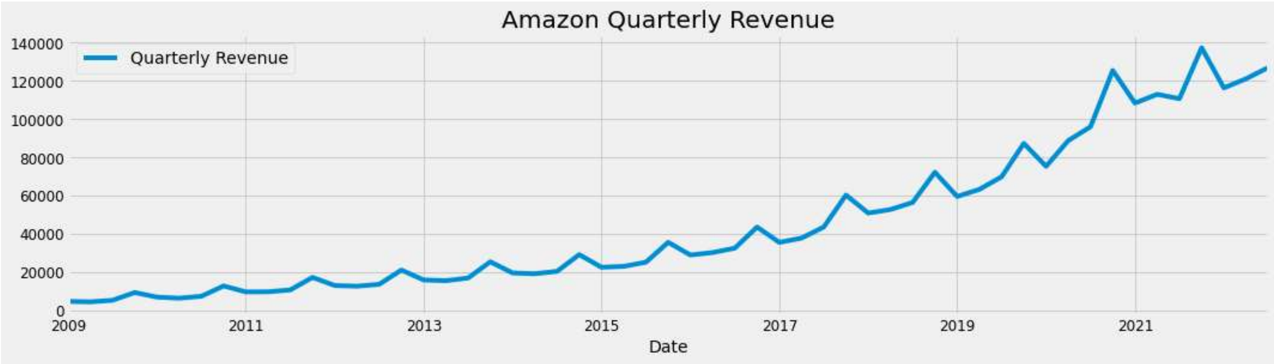
```python
# Setting Date as Index
df = df.set_index('Date')
df.head()
```

Out[4]:

Date	Quarterly Revenue
2009-03-31	4889
2009-06-30	4651
2009-09-30	5449
2009-12-31	9520
2010-03-31	7131

In [5]:

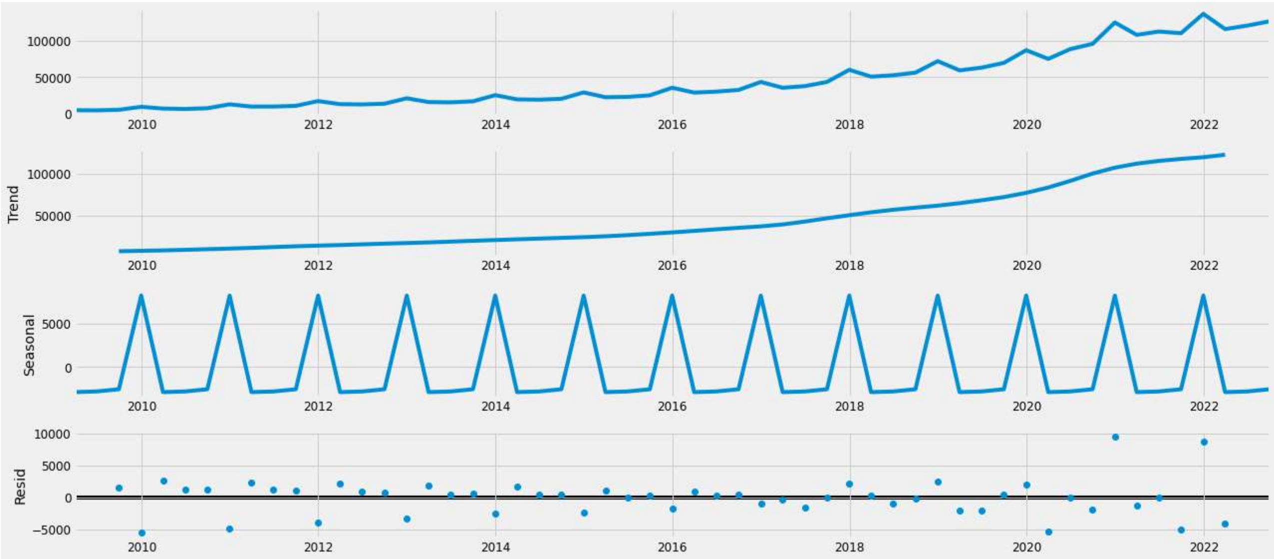
```python
# Plotting the data
df.plot(figsize=(16,4),legend=True)
plt.title('Amazon Quarterly Revenue')
plt.show()
```



In [6]:

```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()
```



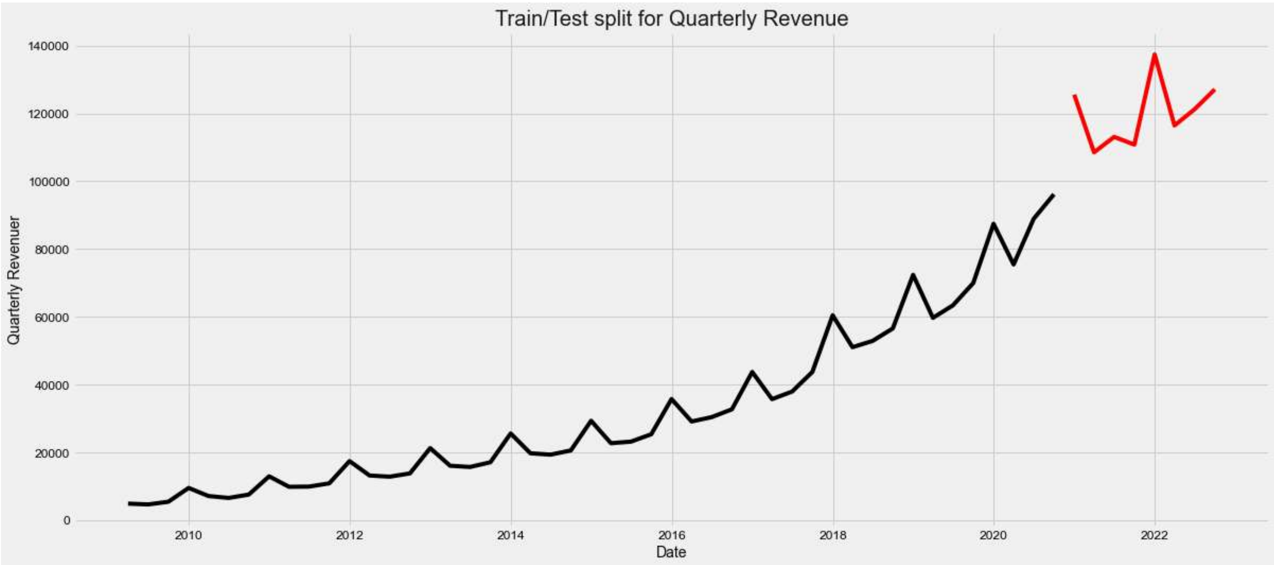
In [7]:

```python
# Dividing the data into training and testing
# Ploting the data
import seaborn as sns
df['Date'] = df.index
train = df[df['Date'] < pd.to_datetime("2020-12", format='%Y-%m')]
train['train'] = train['Quarterly Revenue']
del train['Date']
del train['Quarterly Revenue']
test = df[df['Date'] >= pd.to_datetime("2020-12", format='%Y-%m')]
del test['Date']
test['test'] = test['Quarterly Revenue']
del test['Quarterly Revenue']
plt.plot(train, color = "black")
plt.plot(test, color = "red")
plt.title("Train/Test split for Quarterly Revenue")
plt.ylabel("Quarterly Revenuer")
plt.xlabel('Date')
sns.set()
plt.show()
```

```
C:\Users\ravit\AppData\Local\Temp\ipykernel_23444\1521455262.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_23444\1521455262.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

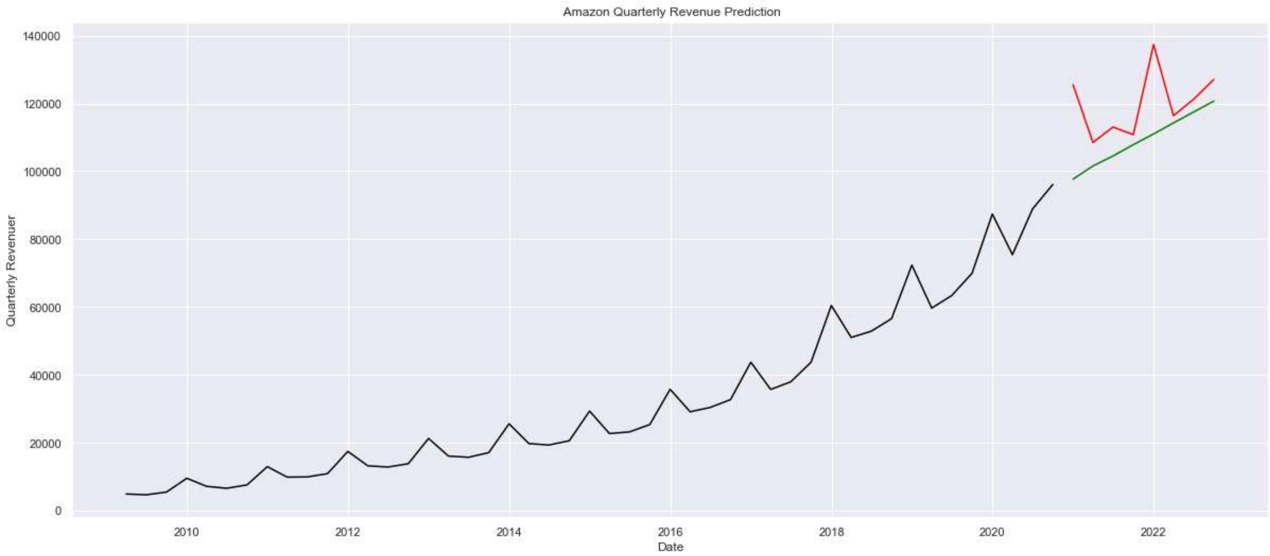
In [8]:

```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(2,2,2)(0,0,0)[0]             : AIC=inf, Time=0.23 sec
 ARIMA(0,2,0)(0,0,0)[0]             : AIC=971.654, Time=0.01 sec
 ARIMA(1,2,0)(0,0,0)[0]             : AIC=952.220, Time=0.02 sec
 ARIMA(0,2,1)(0,0,0)[0]             : AIC=inf, Time=0.05 sec
 ARIMA(2,2,0)(0,0,0)[0]             : AIC=949.387, Time=0.02 sec
 ARIMA(3,2,0)(0,0,0)[0]             : AIC=inf, Time=0.17 sec
 ARIMA(2,2,1)(0,0,0)[0]             : AIC=inf, Time=0.12 sec
 ARIMA(1,2,1)(0,0,0)[0]             : AIC=924.973, Time=0.04 sec
 ARIMA(1,2,2)(0,0,0)[0]             : AIC=inf, Time=0.14 sec
 ARIMA(0,2,2)(0,0,0)[0]             : AIC=inf, Time=0.08 sec
 ARIMA(1,2,1)(0,0,0)[0] intercept   : AIC=inf, Time=0.16 sec

Best model:  ARIMA(1,2,1)(0,0,0)[0]
Total fit time: 1.042 seconds
```

In [9]:
```
# Plotting the prediction
plt.plot(train, color = "black")
plt.plot(test, color = "red")
plt.plot(forecast, color = "green")
plt.title(" Amazon Quarterly Revenue Prediction")
plt.ylabel("Quarterly Revenuer")
plt.xlabel('Date')
sns.set()
plt.show()
```



In [10]:
```
from math import sqrt
from sklearn.metrics import mean_squared_error
rms = sqrt(mean_squared_error(test,forecast))
print("RMSE: ", rms)
```

```
RMSE:  14397.60532056588
```

SARIMA Model

In [11]:
```
df=pd.read_excel('Amazon Quarterly Revenue.xlsx')
df = df.set_index('Date')

```

In [12]:
```
# 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 [13]:

```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:1367.2792850832486

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-DEC 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-DEC 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-DEC 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-DEC 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-DEC 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-DEC will be used.
  self._init_dates(dates, freq)
```

In [14]:

```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-DEC 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-DEC will be used.
  self._init_dates(dates, freq)
R:\Anaconda\envs\general\lib\site-packages\statsmodels\tsa\statespace\sarimax.py:966: UserWarning: Non-stationary starting autore
gressive parameters found. Using zeros as starting parameters.
  warn('Non-stationary starting autoregressive parameters'
R:\Anaconda\envs\general\lib\site-packages\statsmodels\tsa\statespace\sarimax.py:997: UserWarning: Non-stationary starting season
al autoregressive Using zeros as starting parameters.
  warn('Non-stationary starting seasonal autoregressive'

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.9427      0.140     -6.720      0.000      -1.218      -0.668
ma.L1          0.7850      0.263      2.983      0.003       0.269       1.301
ar.S.L12       0.8606      0.142      6.053      0.000       0.582       1.139
sigma2       7.27e+06   9.43e-09   7.71e+14      0.000    7.27e+06    7.27e+06
==============================================================================
```

In [16]:

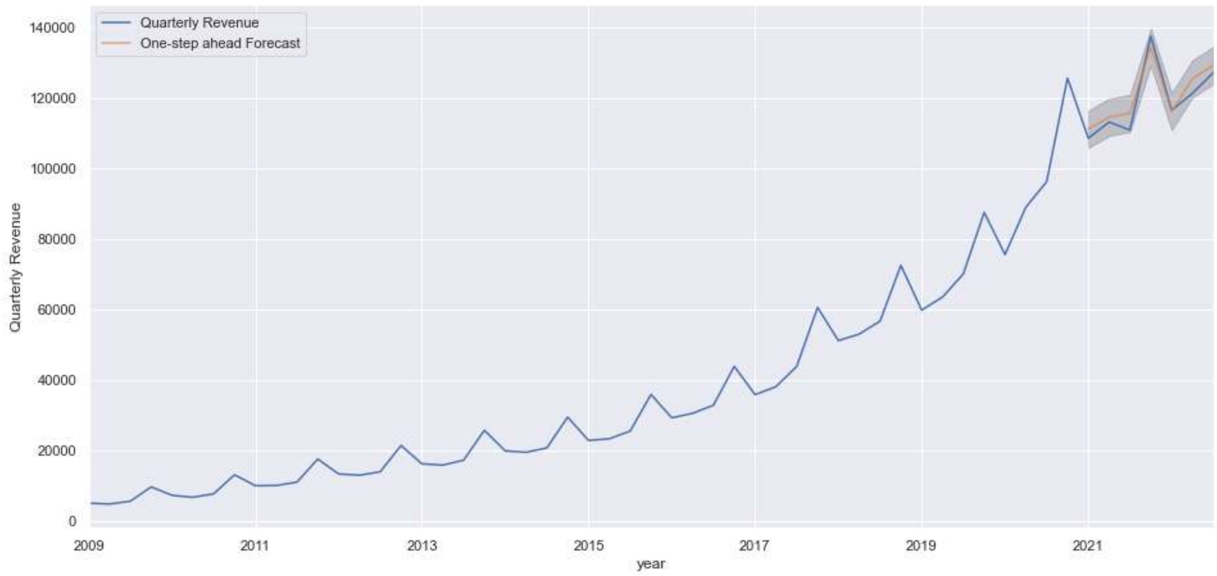
```python
prediction = result.get_prediction(start = pd.to_datetime('2021-03-31'), dynamic = False)
prediction_ci = prediction.conf_int()
prediction_ci
```

Out[16]:

	lower Quarterly Revenue	upper Quarterly Revenue
2021-03-31	105815.138599	116384.229973
2021-06-30	109152.276180	119721.367405
2021-09-30	110279.879532	120848.970664
2021-12-31	129114.345905	139683.436980
2022-03-31	110812.233037	121381.324076
2022-06-30	120115.384149	130684.475167
2022-09-30	123874.142821	134443.233826

In [17]:

```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 [23]:

```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['2021-03-31':]
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 8840005.78536543
The Root Mean Squared Error of our forecasts is 2973.214722378024
```

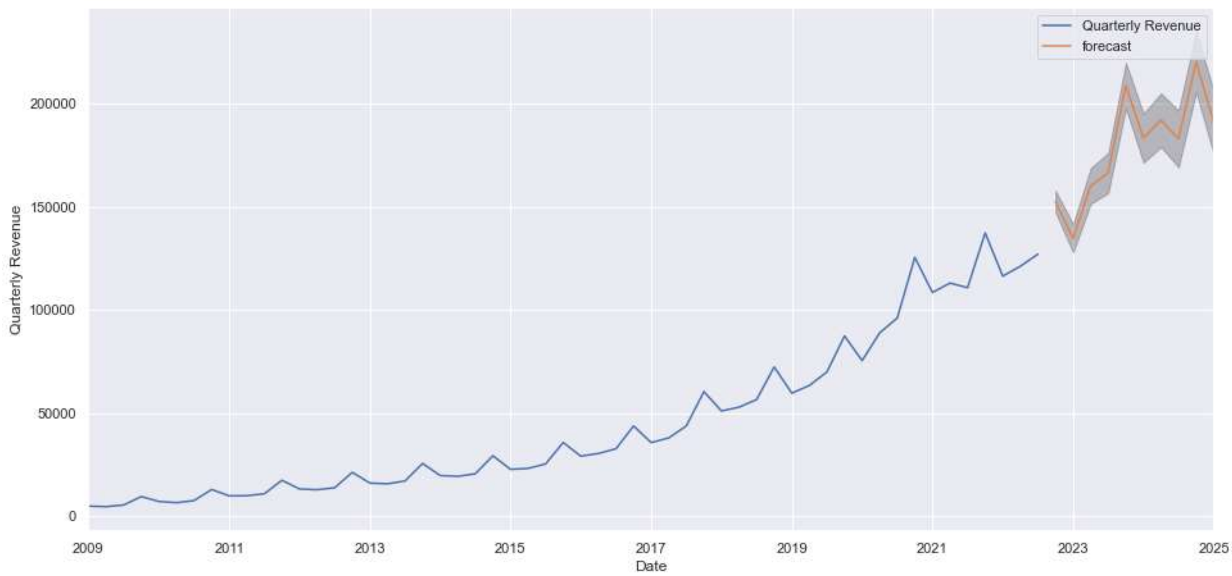
In [24]:

```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 [25]:
```python
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 [26]:
```python
df=pd.read_excel('Amazon Quarterly Revenue.xlsx')

df = df.set_index('Date')

```

In [27]:
```python
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 [28]:
```python
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 [29]:
```python
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 [==============================] - 2s 22ms/step - loss: 1201522048.0000 - mse: 1201522048.0000 - mae: 27822.0586 - val_l
oss: 5053336064.0000 - val_mse: 5053336064.0000 - val_mae: 70970.8594
Epoch 2/500
12/12 [==============================] - 0s 4ms/step - loss: 831125184.0000 - mse: 831125184.0000 - mae: 22757.4551 - val_los
s: 2913479680.0000 - val_mse: 2913479680.0000 - val_mae: 53744.7383
Epoch 3/500
12/12 [==============================] - 0s 3ms/step - loss: 542273984.0000 - mse: 542273984.0000 - mae: 18141.4531 - val_los
s: 1351266816.0000 - val_mse: 1351266816.0000 - val_mae: 36260.3984
Epoch 4/500
12/12 [==============================] - 0s 3ms/step - loss: 321139008.0000 - mse: 321139008.0000 - mae: 13588.3086 - val_los
s: 407396864.0000 - val_mse: 407396864.0000 - val_mae: 19005.3984
Epoch 5/500
12/12 [==============================] - 0s 3ms/step - loss: 165593744.0000 - mse: 165593744.0000 - mae: 9163.9678 - val_loss:
68919288.0000 - val_mse: 68919288.0000 - val_mae: 7771.8203
Epoch 6/500
12/12 [==============================] - 0s 3ms/step - loss: 75645000.0000 - mse: 75645000.0000 - mae: 5967.2075 - val_loss: 2
00441792.0000 - val_mse: 200441792.0000 - val_mae: 11544.0547
Epoch 7/500
```

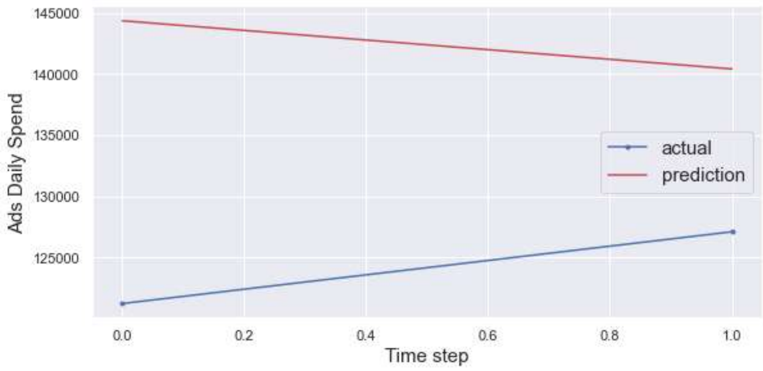
In [30]:
```python
def model_loss(history):
    plt.figure(figsize=(8,4))
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Test Loss')
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epochs')
    plt.legend(loc='upper right')
    plt.show();
```

In [31]:
```python
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): 2863.85; Train Mean Absolute Error(MAE) : 1954.37
[8201623.0, 8201623.0, 1954.36669921875]
[355905248.0, 355905248.0, 18217.0234375]
Test Root Mean Squared Error(RMSE): 18865.45; Test Mean Absolute Error(MAE) : 18217.02
```

In [32]:
```python
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 [33]:
```python
test_predict = model.predict(testX)
prediction_plot(testY, test_predict)
```



GRU and BiLSTM Models

In [34]:
```python
df=pd.read_excel('Amazon Quarterly Revenue.xlsx')
df.head()
```

Out[34]:

	Date	Quarterly Revenue
0	2009-03-31	4889
1	2009-06-30	4651
2	2009-09-30	5449
3	2009-12-31	9520
4	2010-03-31	7131

In [35]:
```python
df = df.set_index('Date')
df.head()
```

Out[35]:

Date	Quarterly Revenue
2009-03-31	4889
2009-06-30	4651
2009-09-30	5449
2009-12-31	9520
2010-03-31	7131

In [36]:
```python
# 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 [37]:
```python
scaler = MinMaxScaler().fit(train_data)
train_scaled = scaler.transform(train_data)
test_scaled = scaler.transform(test_data)
```

In [38]:

```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:  (7, 4, 1)
y_test.shape:  (7, 1)
```

In [39]:

```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 [40]:

```python
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 [==============================] - 4s 639ms/step - loss: 0.0568 - val_loss: 0.3605
Epoch 2/100
2/2 [==============================] - 0s 32ms/step - loss: 0.0325 - val_loss: 0.2366
Epoch 3/100
2/2 [==============================] - 0s 29ms/step - loss: 0.0159 - val_loss: 0.1404
Epoch 4/100
2/2 [==============================] - 0s 28ms/step - loss: 0.0121 - val_loss: 0.0744
Epoch 5/100
2/2 [==============================] - 0s 29ms/step - loss: 0.0106 - val_loss: 0.0389
Epoch 6/100
2/2 [==============================] - 0s 34ms/step - loss: 0.0139 - val_loss: 0.0255
Epoch 7/100
2/2 [==============================] - 0s 29ms/step - loss: 0.0141 - val_loss: 0.0244
Epoch 8/100
2/2 [==============================] - 0s 31ms/step - loss: 0.0132 - val_loss: 0.0302
Epoch 9/100
2/2 [==============================] - 0s 31ms/step - loss: 0.0105 - val_loss: 0.0395
Epoch 10/100
2/2 [==============================] - 0s 27ms/step - loss: 0.0111 - val_loss: 0.0490
Epoch 11/100
2/2 [==============================] - 0s 31ms/step - loss: 0.0063 - val_loss: 0.0544
Epoch 12/100
2/2 [==============================] - 0s 30ms/step - loss: 0.0063 - val_loss: 0.0540
Epoch 13/100
2/2 [==============================] - 0s 33ms/step - loss: 0.0078 - val_loss: 0.0476
Epoch 14/100
2/2 [==============================] - 0s 30ms/step - loss: 0.0074 - val_loss: 0.0377
Epoch 15/100
2/2 [==============================] - 0s 30ms/step - loss: 0.0043 - val_loss: 0.0268
Epoch 16/100
2/2 [==============================] - 0s 31ms/step - loss: 0.0040 - val_loss: 0.0169
Epoch 17/100
2/2 [==============================] - 0s 29ms/step - loss: 0.0035 - val_loss: 0.0100
Epoch 18/100
2/2 [==============================] - 0s 31ms/step - loss: 0.0047 - val_loss: 0.0069
Epoch 19/100
2/2 [==============================] - 0s 32ms/step - loss: 0.0035 - val_loss: 0.0065
Epoch 20/100
2/2 [==============================] - 0s 32ms/step - loss: 0.0044 - val_loss: 0.0072
Epoch 21/100
2/2 [==============================] - 0s 32ms/step - loss: 0.0037 - val_loss: 0.0075
Epoch 22/100
2/2 [==============================] - 0s 30ms/step - loss: 0.0022 - val_loss: 0.0073
Epoch 23/100
2/2 [==============================] - 0s 31ms/step - loss: 0.0024 - val_loss: 0.0069
Epoch 24/100
2/2 [==============================] - 0s 33ms/step - loss: 0.0028 - val_loss: 0.0066
Epoch 25/100
2/2 [==============================] - 0s 30ms/step - loss: 0.0038 - val_loss: 0.0066
Epoch 26/100
2/2 [==============================] - 0s 28ms/step - loss: 0.0028 - val_loss: 0.0068
Epoch 27/100
2/2 [==============================] - 0s 31ms/step - loss: 0.0030 - val_loss: 0.0074
Epoch 28/100
2/2 [==============================] - 0s 29ms/step - loss: 0.0045 - val_loss: 0.0088
Epoch 29/100
2/2 [==============================] - 0s 29ms/step - loss: 0.0031 - val_loss: 0.0102
```

In [41]:

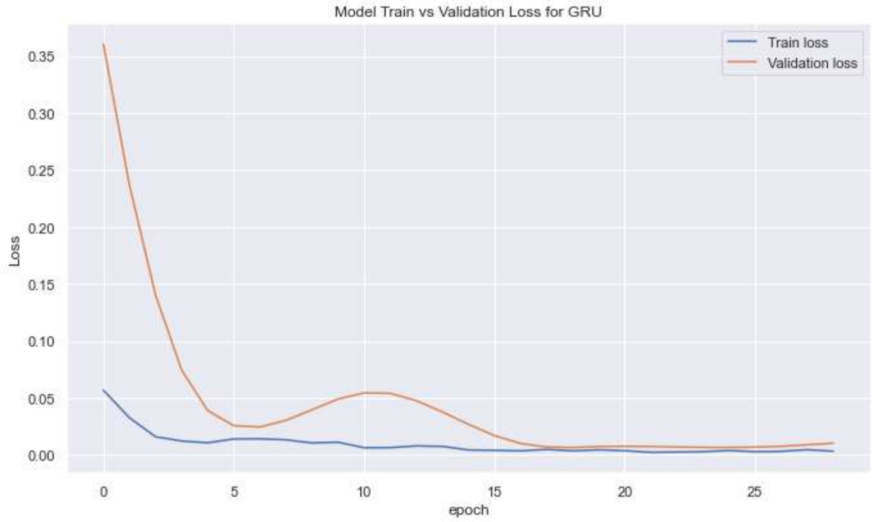
```
history_bilstm = fit_model(model_bilstm)
```

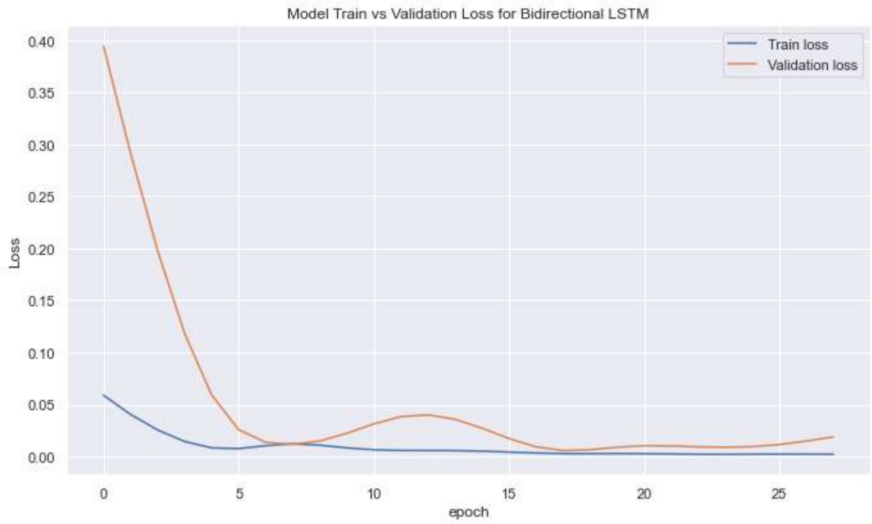
```
Epoch 1/100
2/2 [==============================] - 5s 1s/step - loss: 0.0588 - val_loss: 0.3944
Epoch 2/100
2/2 [==============================] - 0s 28ms/step - loss: 0.0404 - val_loss: 0.2915
Epoch 3/100
2/2 [==============================] - 0s 27ms/step - loss: 0.0255 - val_loss: 0.1979
Epoch 4/100
2/2 [==============================] - 0s 27ms/step - loss: 0.0144 - val_loss: 0.1182
Epoch 5/100
2/2 [==============================] - 0s 27ms/step - loss: 0.0083 - val_loss: 0.0592
Epoch 6/100
2/2 [==============================] - 0s 27ms/step - loss: 0.0076 - val_loss: 0.0257
Epoch 7/100
2/2 [==============================] - 0s 29ms/step - loss: 0.0104 - val_loss: 0.0135
Epoch 8/100
2/2 [==============================] - 0s 29ms/step - loss: 0.0121 - val_loss: 0.0117
Epoch 9/100
2/2 [==============================] - 0s 28ms/step - loss: 0.0109 - val_loss: 0.0150
Epoch 10/100
2/2 [==============================] - 0s 32ms/step - loss: 0.0083 - val_loss: 0.0223
Epoch 11/100
2/2 [==============================] - 0s 32ms/step - loss: 0.0065 - val_loss: 0.0314
Epoch 12/100
2/2 [==============================] - 0s 27ms/step - loss: 0.0058 - val_loss: 0.0383
Epoch 13/100
2/2 [==============================] - 0s 30ms/step - loss: 0.0058 - val_loss: 0.0400
Epoch 14/100
2/2 [==============================] - 0s 30ms/step - loss: 0.0057 - val_loss: 0.0358
Epoch 15/100
2/2 [==============================] - 0s 31ms/step - loss: 0.0051 - val_loss: 0.0272
Epoch 16/100
2/2 [==============================] - 0s 28ms/step - loss: 0.0043 - val_loss: 0.0172
Epoch 17/100
2/2 [==============================] - 0s 29ms/step - loss: 0.0034 - val_loss: 0.0093
Epoch 18/100
2/2 [==============================] - 0s 30ms/step - loss: 0.0028 - val_loss: 0.0057
Epoch 19/100
2/2 [==============================] - 0s 33ms/step - loss: 0.0027 - val_loss: 0.0065
Epoch 20/100
2/2 [==============================] - 0s 28ms/step - loss: 0.0028 - val_loss: 0.0089
Epoch 21/100
2/2 [==============================] - 0s 30ms/step - loss: 0.0027 - val_loss: 0.0104
Epoch 22/100
2/2 [==============================] - 0s 31ms/step - loss: 0.0024 - val_loss: 0.0101
Epoch 23/100
2/2 [==============================] - 0s 33ms/step - loss: 0.0021 - val_loss: 0.0093
Epoch 24/100
2/2 [==============================] - 0s 32ms/step - loss: 0.0021 - val_loss: 0.0089
Epoch 25/100
2/2 [==============================] - 0s 32ms/step - loss: 0.0022 - val_loss: 0.0095
Epoch 26/100
2/2 [==============================] - 0s 30ms/step - loss: 0.0022 - val_loss: 0.0115
Epoch 27/100
2/2 [==============================] - 0s 31ms/step - loss: 0.0022 - val_loss: 0.0149
Epoch 28/100
2/2 [==============================] - 0s 30ms/step - loss: 0.0022 - val_loss: 0.0188
```

In [42]:

```
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')
```

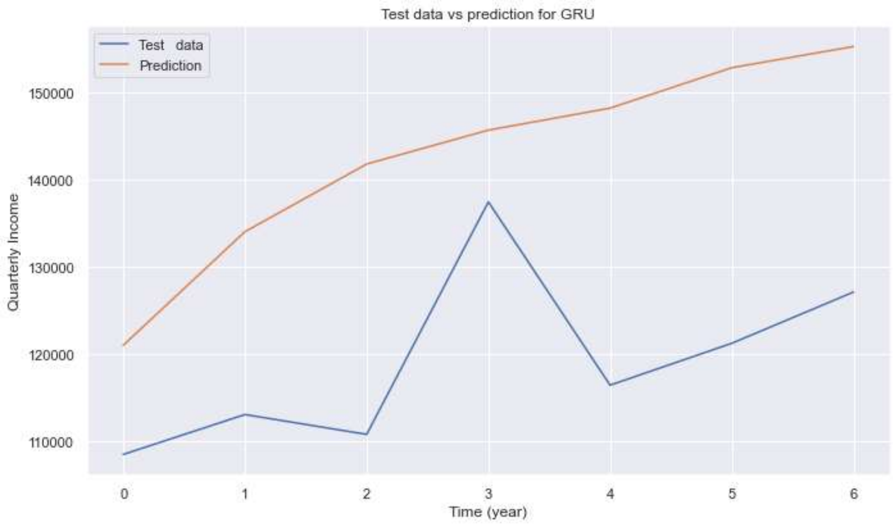


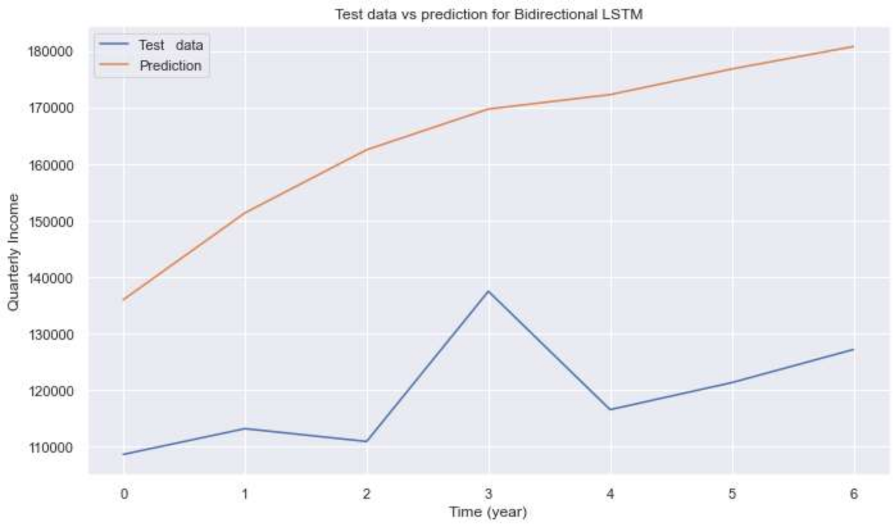


In [43]:

```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 [44]:

```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: 23415.6786
Root Mean Square Error: 25093.8335

Bidirectional LSTM:
Mean Absolute Error: 44927.9888
Root Mean Square Error: 46282.9922
```

In [45]:

```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(scaler.transform(prediction_gru), y_test, 'GRU')
evaluate_prediction(scaler.transform(prediction_bilstm), y_test, 'Bidirectional LSTM')
```

```
GRU:
Mean Absolute Error: 0.2828
Root Mean Square Error: 0.3031

Bidirectional LSTM:
Mean Absolute Error: 0.5427
Root Mean Square Error: 0.5591


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(
```