# Predicting the Quarterly Gross Profit for Amazon

In [10]:

```
# 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 [11]:

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

Out[11]:

| | Date | Quarterly Profit |
|---|---|---|
| **0** | 2009-03-31 | 1148 |
| **1** | 2009-06-30 | 1133 |
| **2** | 2009-09-30 | 1273 |
| **3** | 2009-12-31 | 1977 |
| **4** | 2010-03-31 | 1630 |

In [12]:

```
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 Profit  55 non-null     int64
dtypes: datetime64[ns](1), int64(1)
memory usage: 1008.0 bytes
```

In [13]:

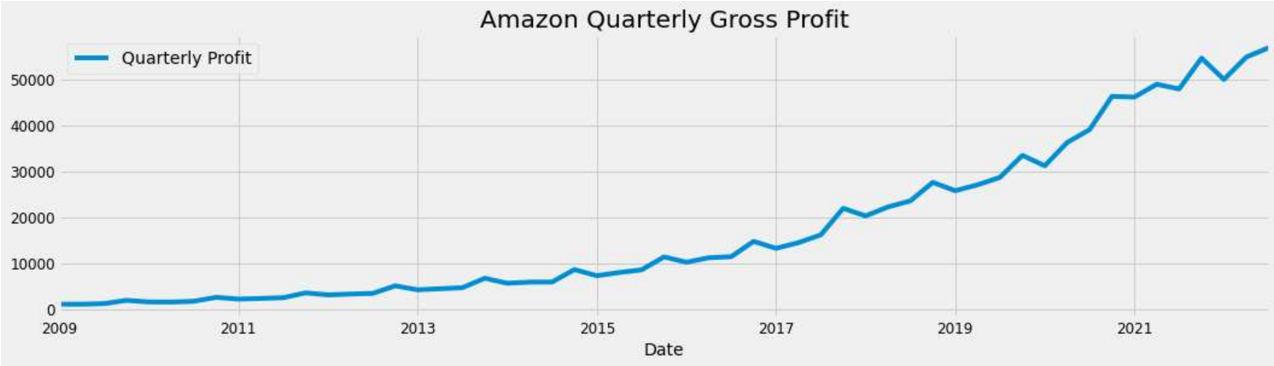
```
# Setting Date as Index
df = df.set_index('Date')
df.head()
```

Out[13]:

| Date | Quarterly Profit |
|---|---|
| **2009-03-31** | 1148 |
| **2009-06-30** | 1133 |
| **2009-09-30** | 1273 |
| **2009-12-31** | 1977 |
| **2010-03-31** | 1630 |

In [14]:

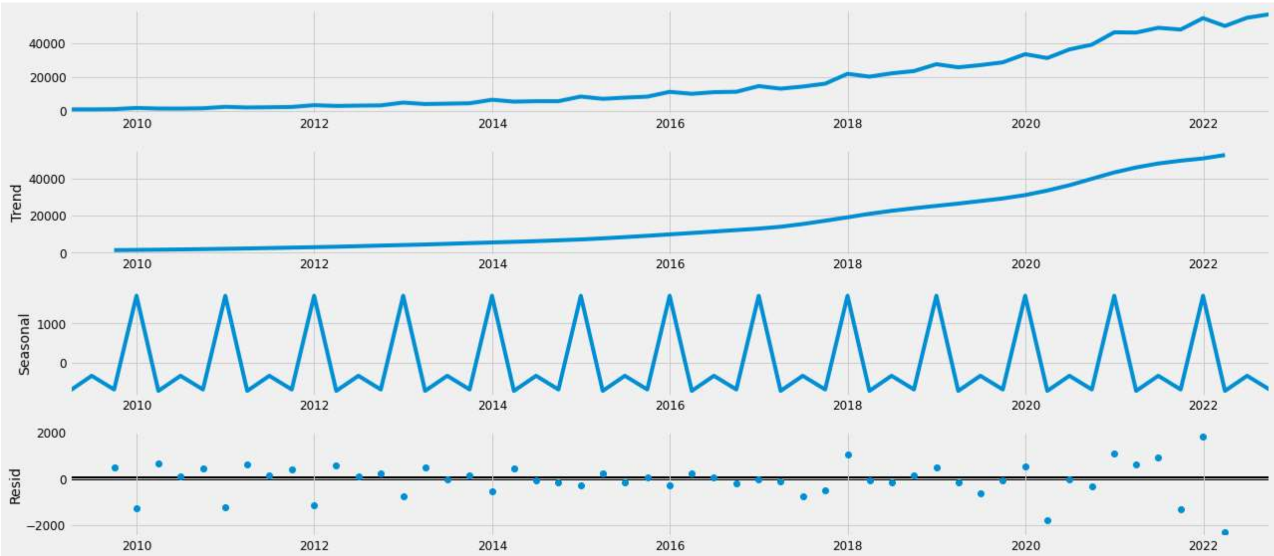
```python
# Plotting the data
df.plot(figsize=(16,4),legend=True)
plt.title('Amazon Quarterly Gross Profit')
plt.show()
```




In [15]:

```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()
```




## SARIMA Model

In [17]:

```python
df=pd.read_excel('Amazon Quarterly Gross Profit.xlsx')
df = df.set_index('Date')
```

In [18]:

```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 [19]:

```
# 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
```

```
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)
R:\Anaconda\envs\general\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: No frequency information was p
```

In [20]:

```
#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)

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.4749      0.492     -0.965      0.334      -1.439       0.489
ma.L1          0.1957      0.587      0.333      0.739      -0.955       1.346
ar.S.L12       0.8051      0.203      3.973      0.000       0.408       1.202
sigma2      1.255e+06   2.32e+05      5.421      0.000    8.02e+05    1.71e+06
==============================================================================
```

In [21]:

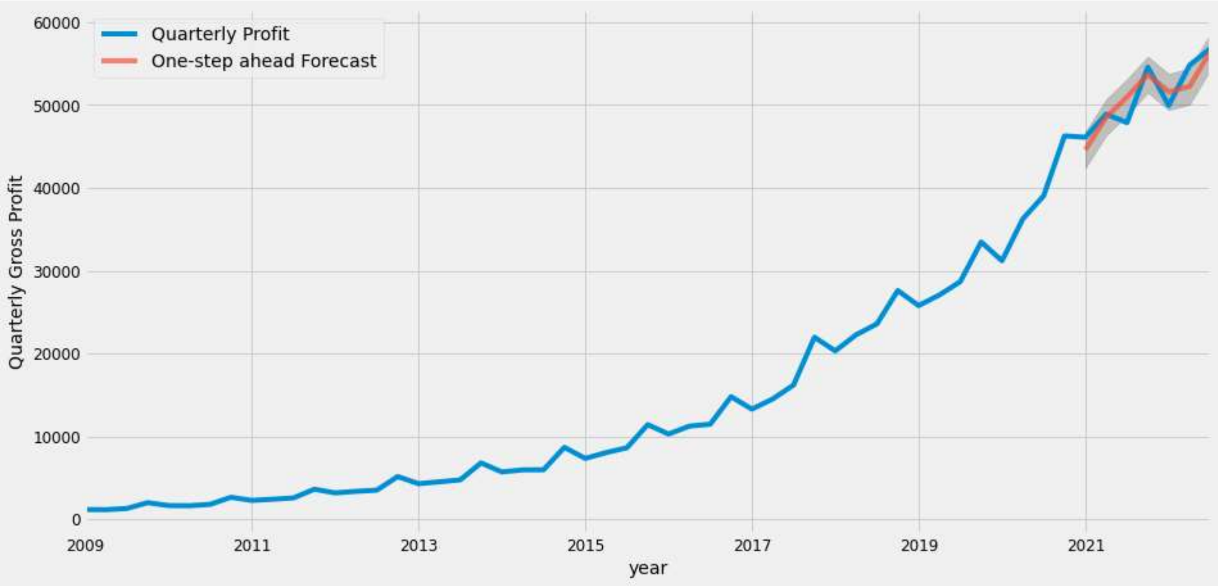
```
prediction = result.get_prediction(start = pd.to_datetime('2021-03-31'), dynamic = False)
prediction_ci = prediction.conf_int()
prediction_ci
```

Out[21]:

| | lower Quarterly Profit | upper Quarterly Profit |
|---|---|---|
| **2021-03-31** | 42398.000144 | 46790.208160 |
| **2021-06-30** | 46333.307402 | 50725.515418 |
| **2021-09-30** | 48772.541951 | 53164.749967 |
| **2021-12-31** | 51484.935925 | 55877.143941 |
| **2022-03-31** | 49395.266486 | 53787.474502 |
| **2022-06-30** | 50054.645715 | 54446.853731 |
| **2022-09-30** | 54264.242912 | 58656.450928 |

In [22]:

```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 Gross Profit')
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 3168962.372852964
The Root Mean Squared Error of our forecasts is 1780.1579628934517
```

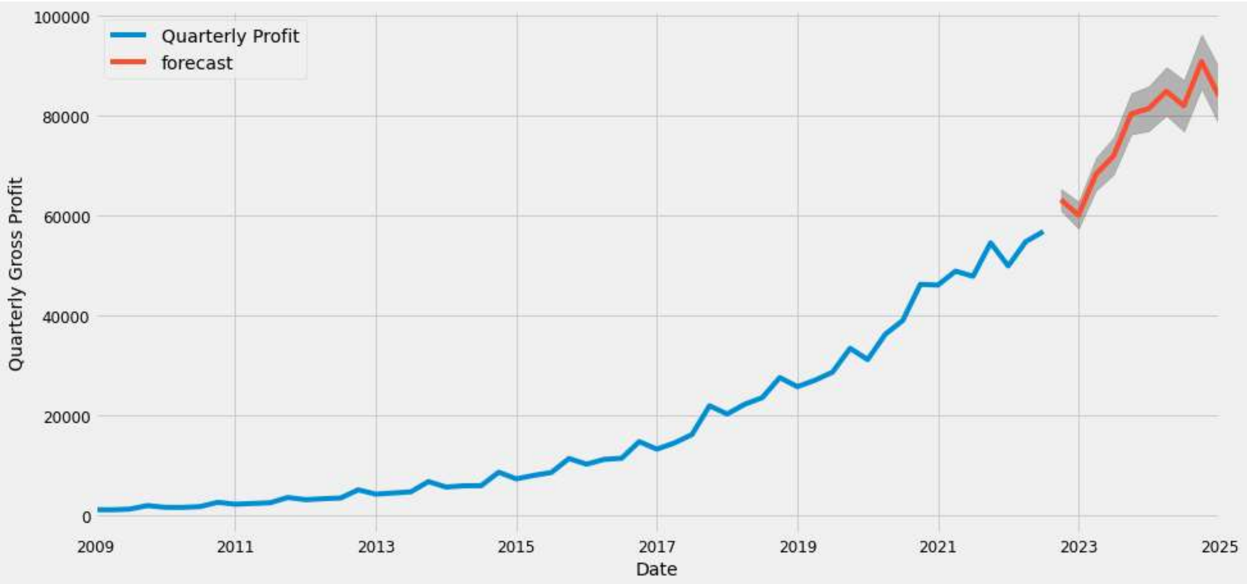
In [24]:

```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 Gross Profit')

plt.legend()
plt.show()

```



# DNN MODEL

In [25]:

```
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]:

```
df=pd.read_excel('Amazon Quarterly Gross Profit.xlsx')

df = df.set_index('Date')

```

In [27]:

```
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]:

```
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]:

```
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: 2350840.2500 - mse: 2350840.2500 - mae: 1033.5248 - val_loss: 54
679544.0000 - val_mse: 54679544.0000 - val_mae: 7304.3223
Epoch 2/500
12/12 [==============================] - 0s 4ms/step - loss: 2116283.5000 - mse: 2116283.5000 - mae: 986.0873 - val_loss: 5292
0608.0000 - val_mse: 52920608.0000 - val_mae: 7196.0234
Epoch 3/500
12/12 [==============================] - 0s 4ms/step - loss: 1979505.3750 - mse: 1979505.3750 - mae: 952.7255 - val_loss: 4929
4968.0000 - val_mse: 49294968.0000 - val_mae: 6949.4453
Epoch 4/500
12/12 [==============================] - 0s 4ms/step - loss: 2070286.6250 - mse: 2070286.6250 - mae: 966.9977 - val_loss: 5102
0464.0000 - val_mse: 51020464.0000 - val_mae: 7077.1367
Epoch 5/500
12/12 [==============================] - 0s 3ms/step - loss: 1921211.3750 - mse: 1921211.3750 - mae: 931.6425 - val_loss: 4565
1016.0000 - val_mse: 45651016.0000 - val_mae: 6691.1348
Epoch 6/500
12/12 [==============================] - 0s 3ms/step - loss: 2239483.5000 - mse: 2239483.5000 - mae: 992.1248 - val_loss: 5280
2632.0000 - val_mse: 52802632.0000 - val_mae: 7206.4922
Epoch 7/500
```

In [30]:

```
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();
```

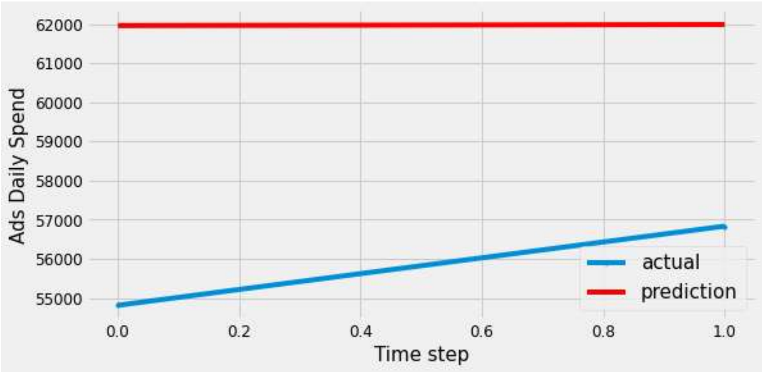
In [31]:

```
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): 1158.29; Train Mean Absolute Error(MAE) : 734.11
[1341638.25, 1341638.25, 734.1107788085938]
[38831080.0, 38831080.0, 6151.0078125]
Test Root Mean Squared Error(RMSE): 6231.46; Test Mean Absolute Error(MAE) : 6151.01
```

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




## GRU and BiLSTM Models

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

Out[34]:

| | Date | Quarterly Profit |
|---|---|---|
| **0** | 2009-03-31 | 1148 |
| **1** | 2009-06-30 | 1133 |
| **2** | 2009-09-30 | 1273 |
| **3** | 2009-12-31 | 1977 |
| **4** | 2010-03-31 | 1630 |

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

Out[35]:

| | Quarterly Profit |
|---|---|
| **Date** | |
| **2009-03-31** | 1148 |
| **2009-06-30** | 1133 |
| **2009-09-30** | 1273 |
| **2009-12-31** | 1977 |
| **2010-03-31** | 1630 |

In [36]:
```
# 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]:
```
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 [==============================] - 3s 715ms/step - loss: 0.0467 - val_loss: 0.4321
Epoch 2/100
2/2 [==============================] - 0s 35ms/step - loss: 0.0279 - val_loss: 0.3178
Epoch 3/100
2/2 [==============================] - 0s 32ms/step - loss: 0.0155 - val_loss: 0.2220
Epoch 4/100
2/2 [==============================] - 0s 31ms/step - loss: 0.0117 - val_loss: 0.1464
Epoch 5/100
2/2 [==============================] - 0s 35ms/step - loss: 0.0097 - val_loss: 0.0932
Epoch 6/100
2/2 [==============================] - 0s 32ms/step - loss: 0.0103 - val_loss: 0.0614
Epoch 7/100
2/2 [==============================] - 0s 32ms/step - loss: 0.0134 - val_loss: 0.0457
Epoch 8/100
2/2 [==============================] - 0s 37ms/step - loss: 0.0125 - val_loss: 0.0394
Epoch 9/100
2/2 [==============================] - 0s 29ms/step - loss: 0.0114 - val_loss: 0.0380
Epoch 10/100
2/2 [==============================] - 0s 28ms/step - loss: 0.0089 - val_loss: 0.0386
Epoch 11/100
2/2 [==============================] - 0s 35ms/step - loss: 0.0077 - val_loss: 0.0388
Epoch 12/100
2/2 [==============================] - 0s 37ms/step - loss: 0.0054 - val_loss: 0.0370
Epoch 13/100
2/2 [==============================] - 0s 34ms/step - loss: 0.0048 - val_loss: 0.0325
Epoch 14/100
2/2 [==============================] - 0s 35ms/step - loss: 0.0041 - val_loss: 0.0252
Epoch 15/100
2/2 [==============================] - 0s 33ms/step - loss: 0.0038 - val_loss: 0.0161
Epoch 16/100
2/2 [==============================] - 0s 37ms/step - loss: 0.0030 - val_loss: 0.0078
Epoch 17/100
2/2 [==============================] - 0s 37ms/step - loss: 0.0028 - val_loss: 0.0030
Epoch 18/100
2/2 [==============================] - 0s 33ms/step - loss: 0.0035 - val_loss: 0.0024
Epoch 19/100
2/2 [==============================] - 0s 38ms/step - loss: 0.0013 - val_loss: 0.0049
Epoch 20/100
2/2 [==============================] - 0s 35ms/step - loss: 0.0026 - val_loss: 0.0079
Epoch 21/100
2/2 [==============================] - 0s 35ms/step - loss: 0.0017 - val_loss: 0.0098
Epoch 22/100
2/2 [==============================] - 0s 35ms/step - loss: 0.0013 - val_loss: 0.0120
Epoch 23/100
2/2 [==============================] - 0s 39ms/step - loss: 0.0021 - val_loss: 0.0138
Epoch 24/100
2/2 [==============================] - 0s 55ms/step - loss: 0.0021 - val_loss: 0.0144
Epoch 25/100
2/2 [==============================] - 0s 42ms/step - loss: 0.0025 - val_loss: 0.0129
Epoch 26/100
2/2 [==============================] - 0s 42ms/step - loss: 0.0028 - val_loss: 0.0110
Epoch 27/100
2/2 [==============================] - 0s 41ms/step - loss: 0.0025 - val_loss: 0.0110
Epoch 28/100
2/2 [==============================] - 0s 44ms/step - loss: 0.0033 - val_loss: 0.0124
```

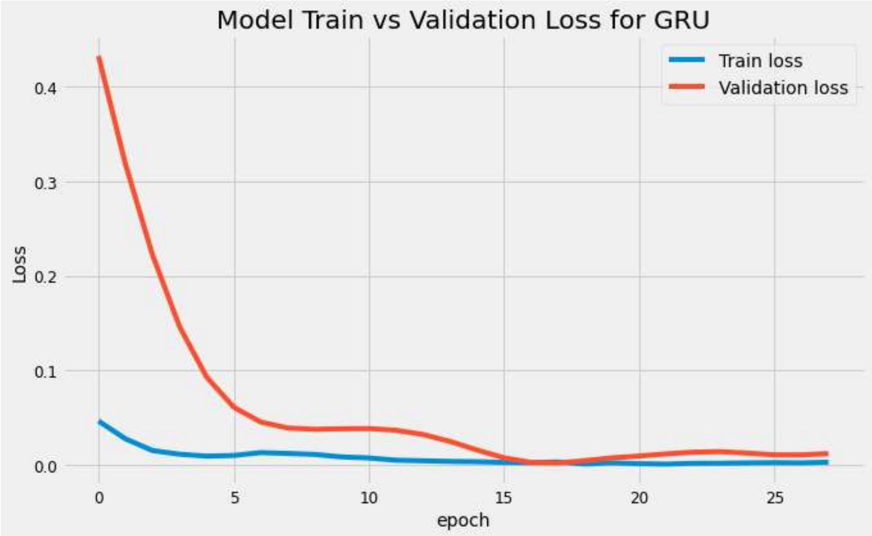
In [41]:
```python
history_bilstm = fit_model(model_bilstm)
```

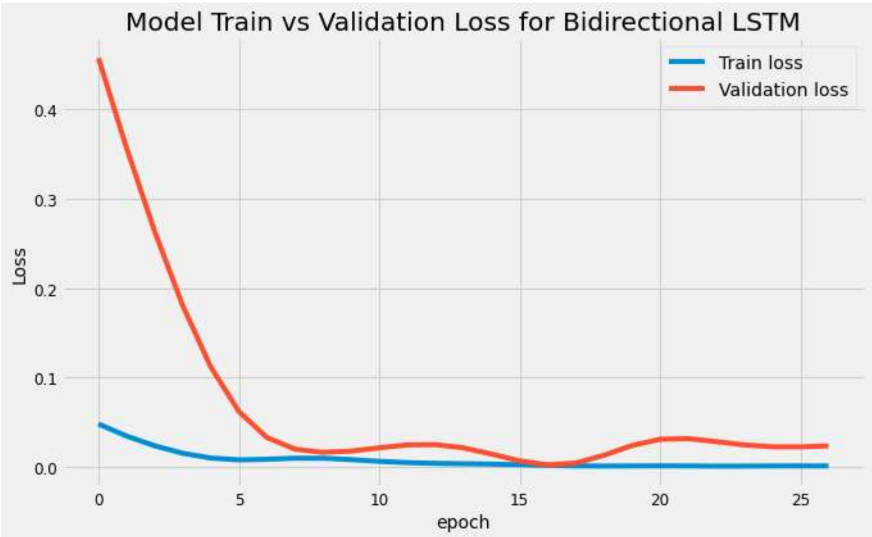
```
Epoch 1/100
2/2 [==============================] - 6s 1s/step - loss: 0.0481 - val_loss: 0.4576
Epoch 2/100
2/2 [==============================] - 0s 38ms/step - loss: 0.0347 - val_loss: 0.3571
Epoch 3/100
2/2 [==============================] - 0s 40ms/step - loss: 0.0238 - val_loss: 0.2638
Epoch 4/100
2/2 [==============================] - 0s 35ms/step - loss: 0.0154 - val_loss: 0.1807
Epoch 5/100
2/2 [==============================] - 0s 33ms/step - loss: 0.0100 - val_loss: 0.1119
Epoch 6/100
2/2 [==============================] - 0s 33ms/step - loss: 0.0080 - val_loss: 0.0622
Epoch 7/100
2/2 [==============================] - 0s 33ms/step - loss: 0.0086 - val_loss: 0.0330
Epoch 8/100
2/2 [==============================] - 0s 34ms/step - loss: 0.0099 - val_loss: 0.0200
Epoch 9/100
2/2 [==============================] - 0s 34ms/step - loss: 0.0098 - val_loss: 0.0164
Epoch 10/100
2/2 [==============================] - 0s 35ms/step - loss: 0.0083 - val_loss: 0.0178
Epoch 11/100
2/2 [==============================] - 0s 35ms/step - loss: 0.0064 - val_loss: 0.0215
Epoch 12/100
2/2 [==============================] - 0s 37ms/step - loss: 0.0049 - val_loss: 0.0247
Epoch 13/100
2/2 [==============================] - 0s 35ms/step - loss: 0.0041 - val_loss: 0.0251
Epoch 14/100
2/2 [==============================] - 0s 33ms/step - loss: 0.0037 - val_loss: 0.0214
Epoch 15/100
2/2 [==============================] - 0s 35ms/step - loss: 0.0033 - val_loss: 0.0145
Epoch 16/100
2/2 [==============================] - 0s 33ms/step - loss: 0.0026 - val_loss: 0.0069
Epoch 17/100
2/2 [==============================] - 0s 34ms/step - loss: 0.0019 - val_loss: 0.0025
Epoch 18/100
2/2 [==============================] - 0s 34ms/step - loss: 0.0013 - val_loss: 0.0044
Epoch 19/100
2/2 [==============================] - 0s 33ms/step - loss: 0.0010 - val_loss: 0.0131
Epoch 20/100
2/2 [==============================] - 0s 34ms/step - loss: 0.0012 - val_loss: 0.0240
Epoch 21/100
2/2 [==============================] - 0s 35ms/step - loss: 0.0013 - val_loss: 0.0310
Epoch 22/100
2/2 [==============================] - 0s 35ms/step - loss: 0.0012 - val_loss: 0.0318
Epoch 23/100
2/2 [==============================] - 0s 37ms/step - loss: 0.0010 - val_loss: 0.0285
Epoch 24/100
2/2 [==============================] - 0s 33ms/step - loss: 0.0010 - val_loss: 0.0248
Epoch 25/100
2/2 [==============================] - 0s 32ms/step - loss: 0.0012 - val_loss: 0.0227
Epoch 26/100
2/2 [==============================] - 0s 34ms/step - loss: 0.0012 - val_loss: 0.0225
Epoch 27/100
2/2 [==============================] - 0s 37ms/step - loss: 0.0012 - val_loss: 0.0237
```

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')
```

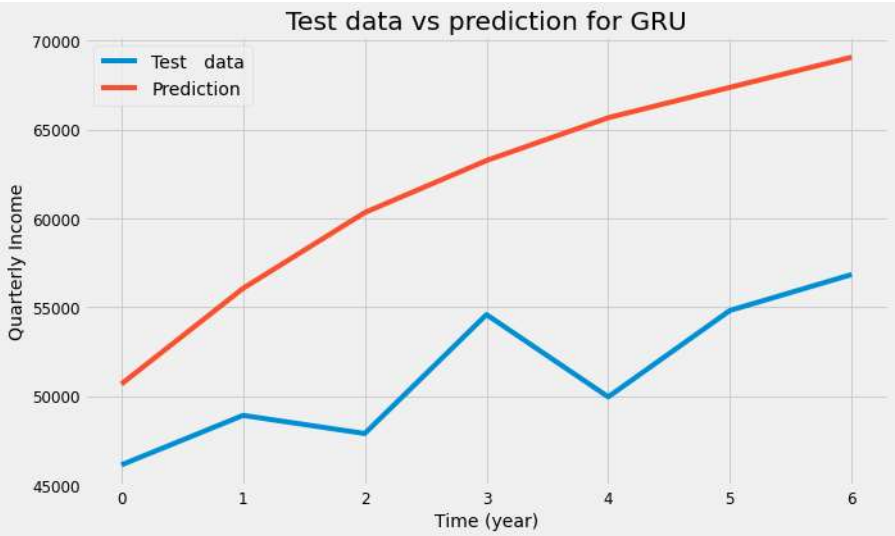


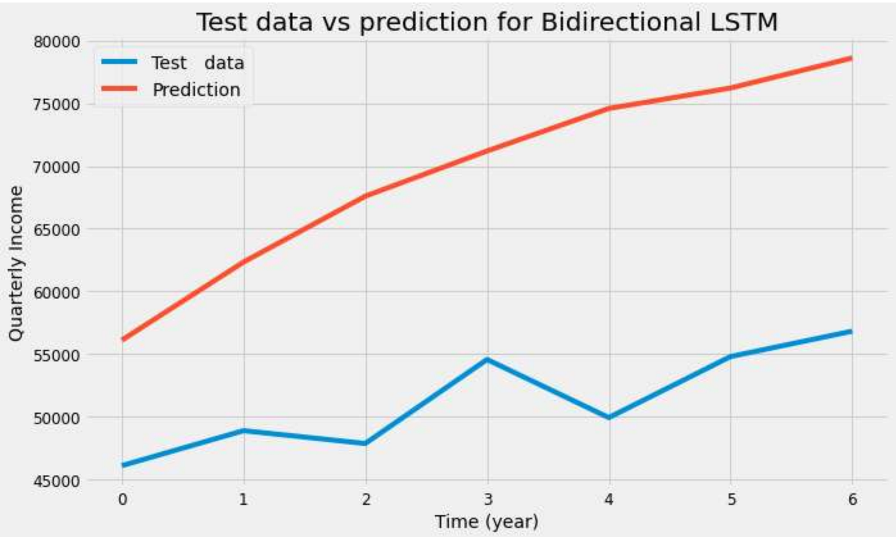



In [43]:

```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: 10472.3912
Root Mean Square Error: 11057.1284

Bidirectional LSTM:
Mean Absolute Error: 18223.3047
Root Mean Square Error: 18836.8948
```

In [45]:

```
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: 0.3240
Root Mean Square Error: 0.3420

Bidirectional LSTM:
Mean Absolute Error: 0.5637
Root Mean Square Error: 0.5827


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(
```