Predicting the Quarterly Net Income for Amazon

In [8]:

```
# 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 [9]:

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

Out[9]:

	Date	Quarterly Net Income
0	2009-03-31	177
1	2009-06-30	142
2	2009-09-30	199
3	2009-12-31	384
4	2010-03-31	299

In [10]:

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

In [11]:

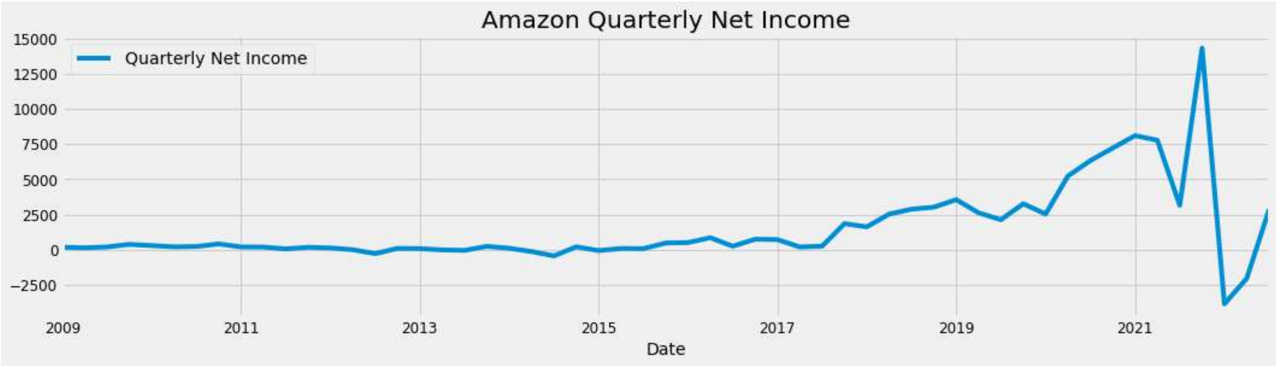
```
# Setting Date as Index
df = df.set_index('Date')
df.head()
```

Out[11]:

	Quarterly Net Income
Date	
2009-03-31	177
2009-06-30	142
2009-09-30	199
2009-12-31	384
2010-03-31	299

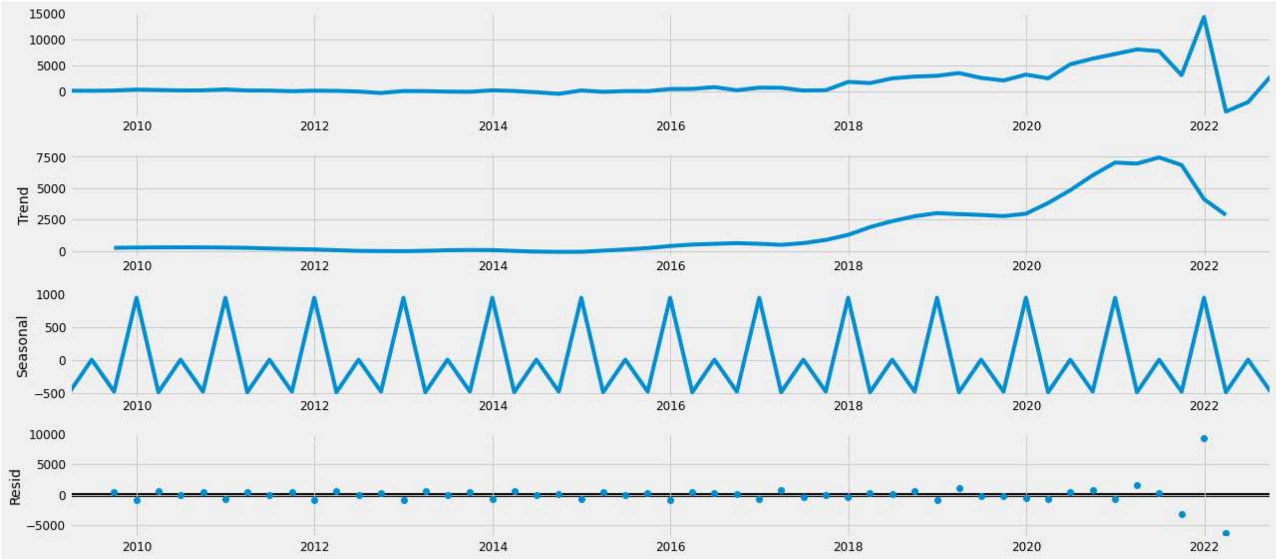
In [12]:

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



In [13]:

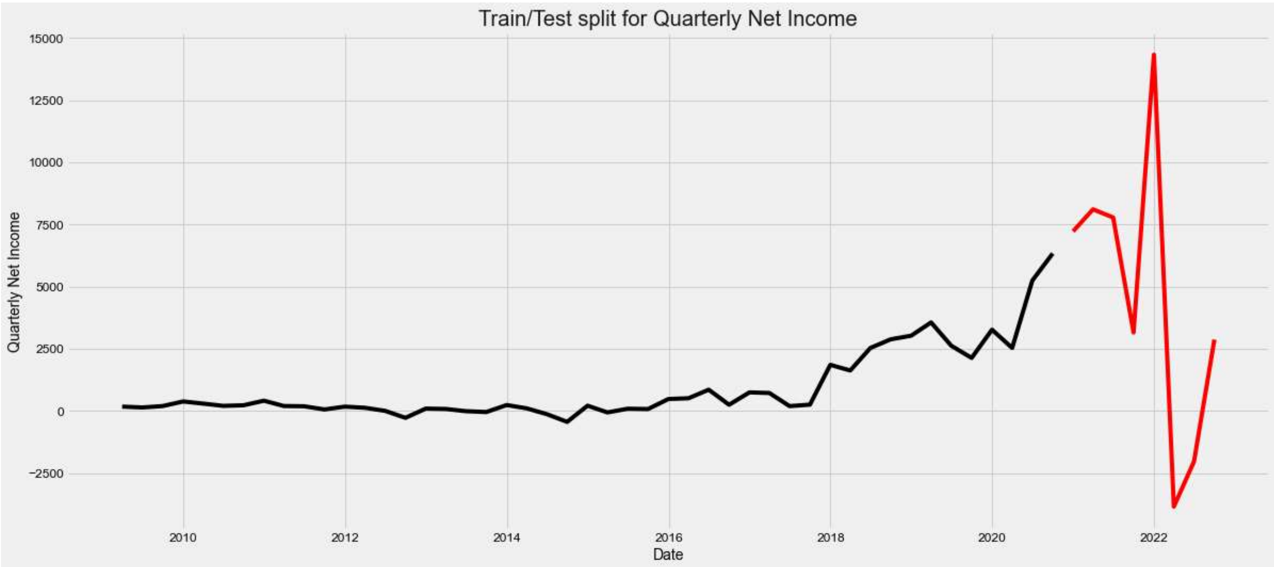
```
# Decomposition the data
from pylab import rcParams
rcParams['figure.figsize'] = 18, 8

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



In [14]:

```
# 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 Net Income']
del train['Date']
del train['Quarterly Net Income']
test = df[df['Date'] >= pd.to_datetime("2020-12", format='%Y-%m')]
del test['Date']
test['test'] = test['Quarterly Net Income']
del test['Quarterly Net Income']
plt.plot(train, color = "black")
plt.plot(test, color = "red")
plt.title("Train/Test split for Quarterly Net Income")
plt.ylabel("Quarterly Net Income")
plt.xlabel('Date')
sns.set()
plt.show()
```

```
C:\Users\ravit\AppData\Local\Temp\ipykernel_9332\2151717043.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-vers
us-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  train['train'] = train['Quarterly Net Income']
C:\Users\ravit\AppData\Local\Temp\ipykernel_9332\2151717043.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-vers
us-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  test['test'] = test['Quarterly Net Income']
```



Arima Model

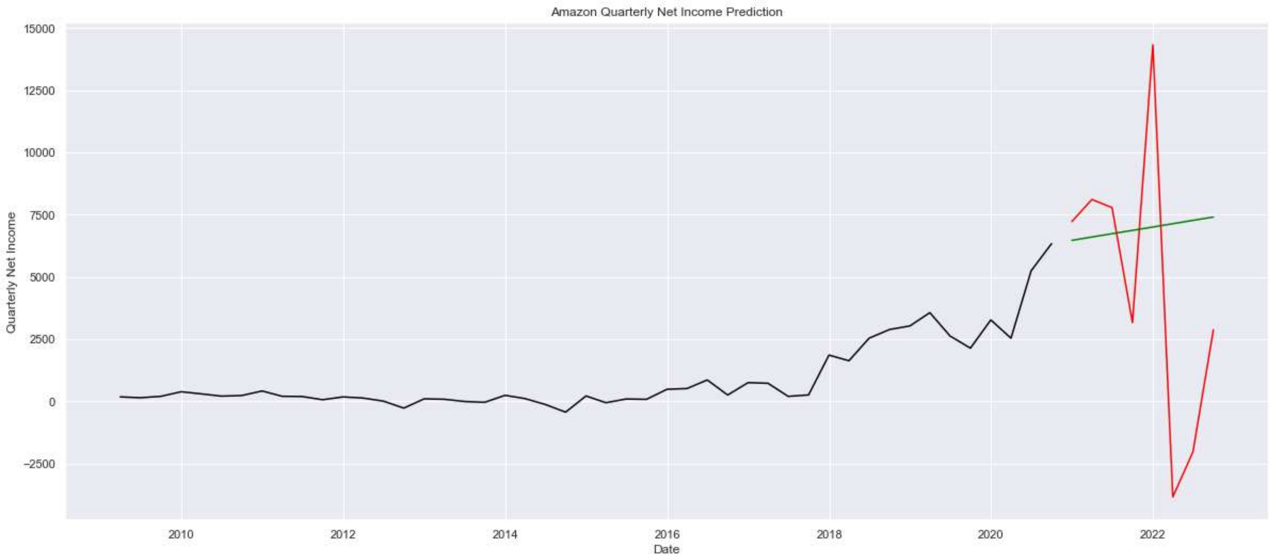
In [15]:

```
# 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=inf, Time=0.55 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=723.843, Time=0.04 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=725.025, Time=0.03 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=725.129, Time=0.10 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=724.038, Time=0.01 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=726.967, Time=0.11 sec

Best model:  ARIMA(0,1,0)(0,0,0)[0] intercept
Total fit time: 0.864 seconds
```

In [16]:

```
# Plotting the prediction
plt.plot(train, color = "black")
plt.plot(test, color = "red")
plt.plot(forecast, color = "green")
plt.title("Amazon Quarterly Net Income Prediction")
plt.ylabel("Quarterly Net Income")
plt.xlabel('Date')
sns.set()
plt.show()
```



In [17]:

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

```
RMSE:  6111.103976680451
```

SARIMA Model

In [18]:

```
df=pd.read_excel('Amazon Quarterly Net Income.xlsx')
df = df.set_index('Date')
```

In [19]:

```
# 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 [20]:
```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
```

```
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)

SARIMA(0, 0, 0)x(0, 0, 0, 12)12 - AIC:1045.8693541124176
```

In [21]:
```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: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         -0.2281      0.292     -0.781      0.435      -0.800       0.344
ma.L1         -0.5631      0.466     -1.208      0.227      -1.477       0.351
ar.S.L12      -0.0901      1.084     -0.083      0.934      -2.215       2.035
sigma2      9.071e+06    1.1e+06      8.245      0.000    6.91e+06    1.12e+07
==============================================================================
```

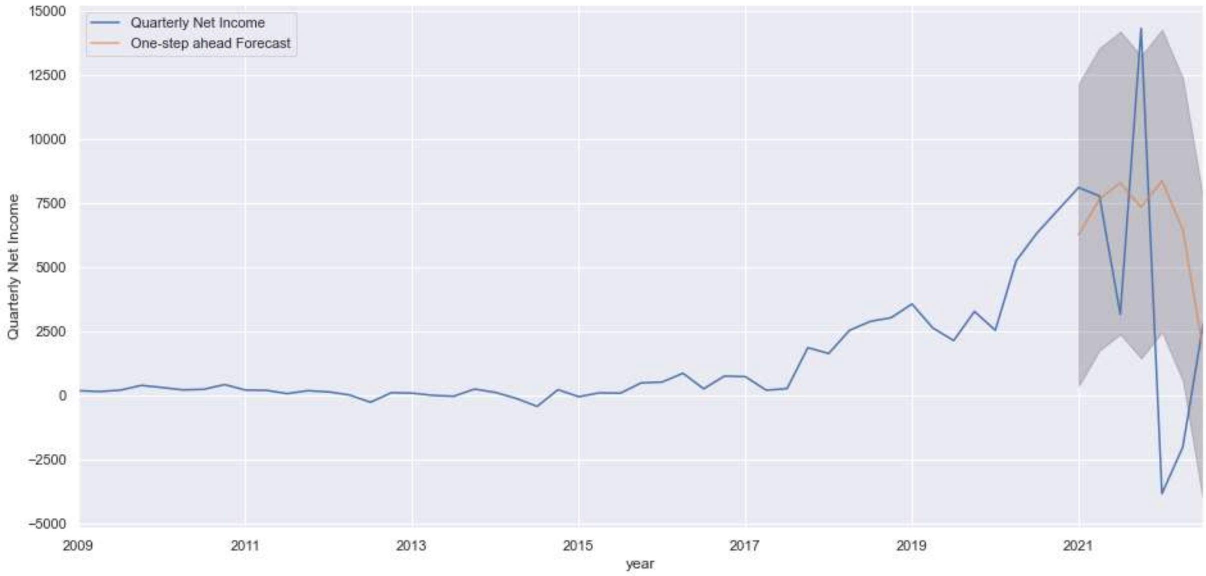
In [22]:
```python
prediction = result.get_prediction(start = pd.to_datetime('2021-03-31'), dynamic = False)
prediction_ci = prediction.conf_int()
prediction_ci
```

Out[22]:

	lower Quarterly Net Income	upper Quarterly Net Income
2021-03-31	367.936860	12174.322671
2021-06-30	1752.300168	13558.685979
2021-09-30	2388.203534	14194.589345
2021-12-31	1438.612959	13244.998769
2022-03-31	2467.783946	14274.169757
2022-06-30	566.712621	12373.098431
2022-09-30	-4248.722171	7557.663640

In [23]:

```
#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 Net Income')
plt.legend()
plt.show()
```



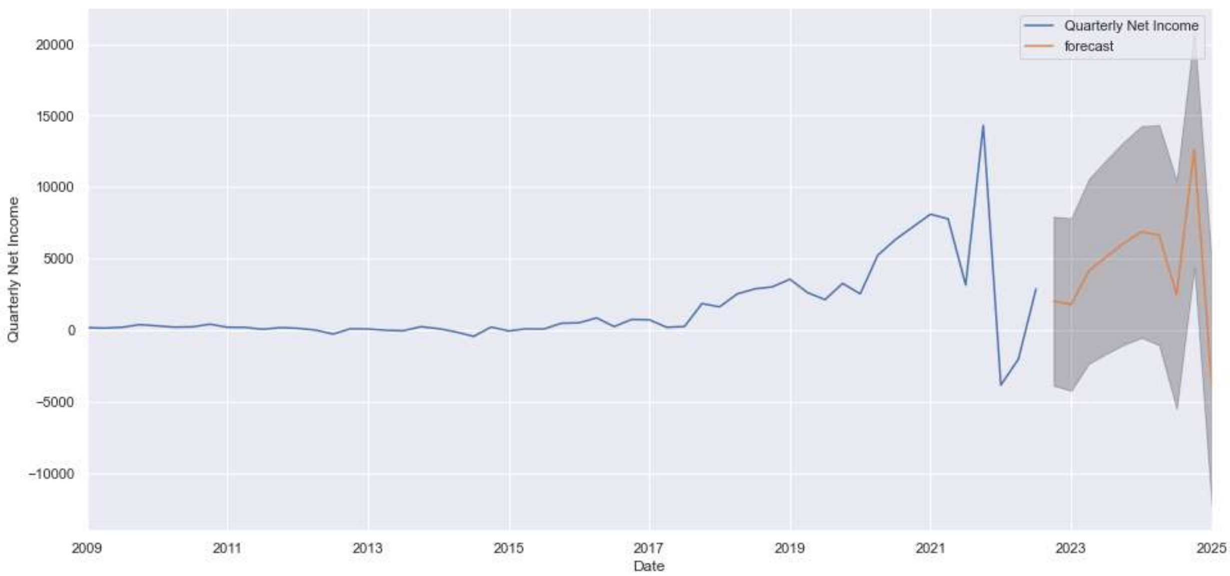
In [25]:

```
# 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 43056747.27616465
The Root Mean Squared Error of our forecasts is 6561.764036916037
```

In [26]:

```
# 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 Net Income')

plt.legend()
plt.show()

```



DNN MODEL

In [27]:
```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 [28]:
```python
df=pd.read_excel('Amazon Quarterly Net Income.xlsx')

df = df.set_index('Date')

```

In [29]:
```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 [30]:
```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 [31]:
```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 40ms/step - loss: 3621193.2500 - mse: 3621193.2500 - mae: 1064.9882 - val_loss: 14
530798.0000 - val_mse: 14530798.0000 - val_mae: 3766.7988
Epoch 2/500
12/12 [==============================] - 0s 5ms/step - loss: 3066935.7500 - mse: 3066935.7500 - mae: 969.4407 - val_loss: 1625
3394.0000 - val_mse: 16253394.0000 - val_mae: 3868.6843
Epoch 3/500
12/12 [==============================] - 0s 4ms/step - loss: 2573163.2500 - mse: 2573163.2500 - mae: 881.3557 - val_loss: 1857
1168.0000 - val_mse: 18571168.0000 - val_mae: 3979.4333
Epoch 4/500
12/12 [==============================] - 0s 4ms/step - loss: 2135605.5000 - mse: 2135605.5000 - mae: 797.1470 - val_loss: 2146
7206.0000 - val_mse: 21467206.0000 - val_mae: 4111.5200
Epoch 5/500
12/12 [==============================] - 0s 4ms/step - loss: 1764426.2500 - mse: 1764426.2500 - mae: 717.7985 - val_loss: 2477
3664.0000 - val_mse: 24773664.0000 - val_mae: 4254.0977
Epoch 6/500
12/12 [==============================] - 0s 4ms/step - loss: 1446902.6250 - mse: 1446902.6250 - mae: 653.2609 - val_loss: 2848
5308.0000 - val_mse: 28485308.0000 - val_mae: 4410.8984
Epoch 7/500
```

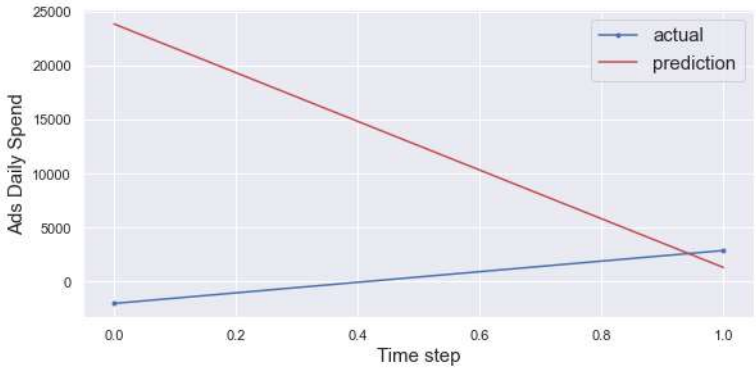
In [32]:
```python
def model_loss(history):
    plt.figure(figsize=(8,4))
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Test Loss')
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epochs')
    plt.legend(loc='upper right')
    plt.show();
```

In [33]:
```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): 325.12; Train Mean Absolute Error(MAE) : 164.25
[105702.1796875, 105702.1796875, 164.24574279785156]
[336099744.0, 336099744.0, 13721.6064453125]
Test Root Mean Squared Error(RMSE): 18333.02; Test Mean Absolute Error(MAE) : 13721.61
```

In [34]:
```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 [35]:
```python
test_predict = model.predict(testX)
prediction_plot(testY, test_predict)
```



GRU and BiLSTM Models

In [36]:
```python
df=pd.read_excel('Amazon Quarterly Net Income.xlsx')
df.head()
```

Out[36]:

	Date	Quarterly Net Income
0	2009-03-31	177
1	2009-06-30	142
2	2009-09-30	199
3	2009-12-31	384
4	2010-03-31	299

In [37]:
```python
df = df.set_index('Date')
df.head()
```

Out[37]:

	Quarterly Net Income
Date	
2009-03-31	177
2009-06-30	142
2009-09-30	199
2009-12-31	384
2010-03-31	299

In [38]:
```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 [39]:
```python
scaler = MinMaxScaler().fit(train_data)
train_scaled = scaler.transform(train_data)
test_scaled = scaler.transform(test_data)
```

In [40]:

```
# 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 [41]:

```
# 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 [42]:

```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 844ms/step - loss: 0.0311 - val_loss: 0.4045
Epoch 2/100
2/2 [==============================] - 0s 31ms/step - loss: 0.0138 - val_loss: 0.2520
Epoch 3/100
2/2 [==============================] - 0s 32ms/step - loss: 0.0084 - val_loss: 0.1504
Epoch 4/100
2/2 [==============================] - 0s 29ms/step - loss: 0.0095 - val_loss: 0.1066
Epoch 5/100
2/2 [==============================] - 0s 30ms/step - loss: 0.0147 - val_loss: 0.1066
Epoch 6/100
2/2 [==============================] - 0s 39ms/step - loss: 0.0124 - val_loss: 0.1293
Epoch 7/100
2/2 [==============================] - 0s 38ms/step - loss: 0.0100 - val_loss: 0.1597
Epoch 8/100
2/2 [==============================] - 0s 41ms/step - loss: 0.0083 - val_loss: 0.1866
Epoch 9/100
2/2 [==============================] - 0s 38ms/step - loss: 0.0092 - val_loss: 0.2030
Epoch 10/100
2/2 [==============================] - 0s 49ms/step - loss: 0.0095 - val_loss: 0.2062
Epoch 11/100
2/2 [==============================] - 0s 44ms/step - loss: 0.0087 - val_loss: 0.1977
Epoch 12/100
2/2 [==============================] - 0s 40ms/step - loss: 0.0095 - val_loss: 0.1822
Epoch 13/100
2/2 [==============================] - 0s 38ms/step - loss: 0.0087 - val_loss: 0.1630
Epoch 14/100
2/2 [==============================] - 0s 44ms/step - loss: 0.0081 - val_loss: 0.1434
Epoch 15/100
2/2 [==============================] - 0s 37ms/step - loss: 0.0089 - val_loss: 0.1272
```

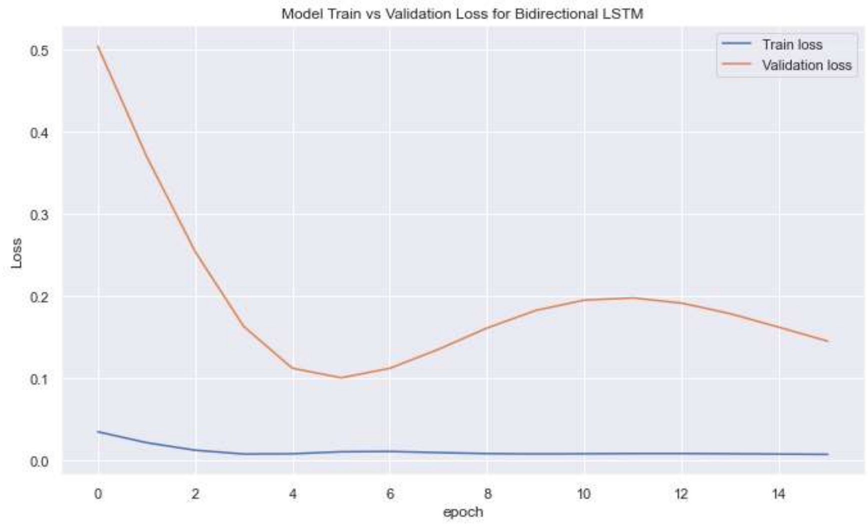
In [43]:

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

```
Epoch 1/100
2/2 [==============================] - 6s 2s/step - loss: 0.0349 - val_loss: 0.5041
Epoch 2/100
2/2 [==============================] - 0s 39ms/step - loss: 0.0217 - val_loss: 0.3704
Epoch 3/100
2/2 [==============================] - 0s 32ms/step - loss: 0.0125 - val_loss: 0.2542
Epoch 4/100
2/2 [==============================] - 0s 30ms/step - loss: 0.0078 - val_loss: 0.1630
Epoch 5/100
2/2 [==============================] - 0s 31ms/step - loss: 0.0081 - val_loss: 0.1121
Epoch 6/100
2/2 [==============================] - 0s 29ms/step - loss: 0.0106 - val_loss: 0.1006
Epoch 7/100
2/2 [==============================] - 0s 32ms/step - loss: 0.0110 - val_loss: 0.1120
Epoch 8/100
2/2 [==============================] - 0s 33ms/step - loss: 0.0096 - val_loss: 0.1352
Epoch 9/100
2/2 [==============================] - 0s 28ms/step - loss: 0.0083 - val_loss: 0.1611
Epoch 10/100
2/2 [==============================] - 0s 37ms/step - loss: 0.0079 - val_loss: 0.1825
Epoch 11/100
2/2 [==============================] - 0s 35ms/step - loss: 0.0081 - val_loss: 0.1951
Epoch 12/100
2/2 [==============================] - 0s 34ms/step - loss: 0.0084 - val_loss: 0.1977
Epoch 13/100
2/2 [==============================] - 0s 32ms/step - loss: 0.0084 - val_loss: 0.1914
Epoch 14/100
2/2 [==============================] - 0s 34ms/step - loss: 0.0081 - val_loss: 0.1785
Epoch 15/100
2/2 [==============================] - 0s 34ms/step - loss: 0.0077 - val_loss: 0.1620
Epoch 16/100
2/2 [==============================] - 0s 33ms/step - loss: 0.0074 - val_loss: 0.1451
```

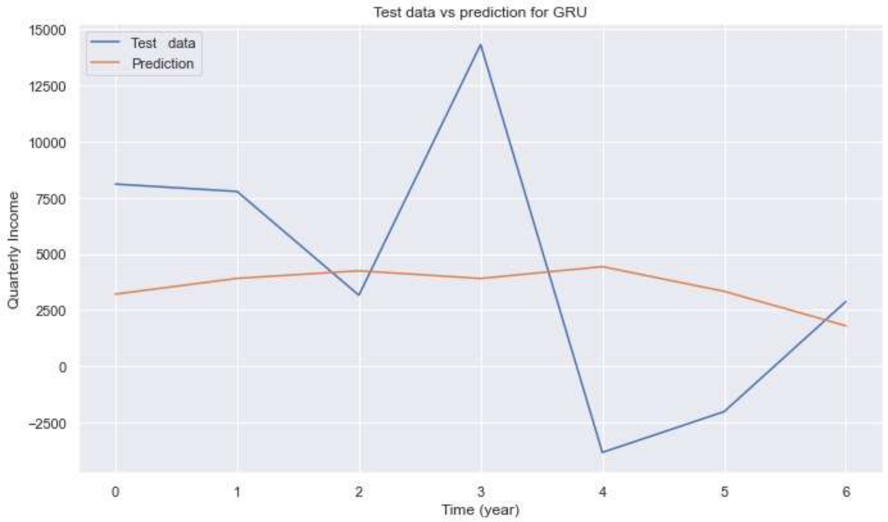
In [44]:

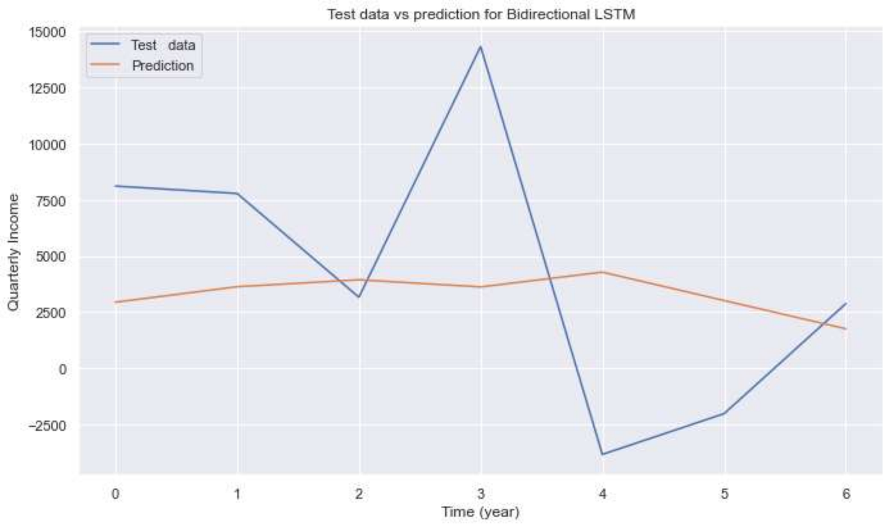
```
def plot_loss (history, model_name):
    plt.figure(figsize = (10, 6))
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model Train vs Validation Loss for ' + model_name)
    plt.ylabel('Loss')
    plt.xlabel('epoch')
    plt.legend(['Train loss', 'Validation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

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



In [45]:

```
# 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 [46]:

```
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: 4998.6921
Root Mean Square Error: 5941.3557

Bidirectional LSTM:
Mean Absolute Error: 5012.3960
Root Mean Square Error: 5997.4695
```

In [47]:

```
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: 1.2503
Root Mean Square Error: 1.4861

Bidirectional LSTM:
Mean Absolute Error: 1.2537
Root Mean Square Error: 1.5001


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 [ ]:

```

```