In [1]:
```
import csv
import sys
from datetime import datetime
import pandas as pd
from statsmodels.tsa.stattools import grangercausalitytests
import pandas as pd
import numpy as np
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoin
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
```

```
2023-05-20 03:48:42.266214: I tensorflow/core/platform/cpu_feature_guard.cc:
193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Lib
rary (oneDNN) to use the following CPU instructions in performance-critical
operations:  SSE4.1 SSE4.2
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
```

In [2]:
```
company_name = 'Cisco'
close_price_as_predictive_column = False

start_date = datetime.strptime('03/02/2020', '%m/%d/%Y')
end_date = datetime.strptime('03/03/2023', '%m/%d/%Y')

data = []

with open('Historical Data/' + company_name + '.csv', 'r') as csv_file:
    csv_reader = csv.reader(csv_file)
    next(csv_reader)

    for row in csv_reader:
        date_str = row[0]
        date = datetime.strptime(date_str, '%m/%d/%Y')
        if start_date <= date <= end_date:
            data.append(row)

rows = list(data)

averages = []
for i in range(0, len(rows), 5):
    row_subset = rows[i:i+5]
    col_subset = [float(row[1][1:]) for row in row_subset]
    avg = sum(col_subset) / len(col_subset)
    averages.append(round(avg,2))

processed_data = []
processed_data.append(['Date', 'Close', 'Layoffs'])
for i in range(0, len(rows), 5):
    row_subset = rows[i:i+5]
    first_row = row_subset[0][0]
    mapped_avg = averages[int(i/5)]
    date = datetime.strptime(first_row, '%m/%d/%Y')
    processed_data.append([first_row, mapped_avg,0])
```

In [3]:
```
companies = ['Adobe', 'Airbnb', 'Amazon', 'Apple', 'Atlassian', 'Cisco', 'Cc

with open('layoffs.csv', 'r') as csv_file:
    csv_reader = csv.reader(csv_file)
    rows = list(csv_reader)

    """
    for row in csv_reader:
        if row[0] in companies:
            print(row)
    """


sorted_rows = sorted(rows, key=lambda x: x[0])

processed_rows = list(processed_data)

start = 1

for row in sorted_rows:
    if (row[0] == company_name):
        print(row)
        date = datetime.strptime(row[5], '%Y-%m-%d')
        for i in range(start,len(processed_rows)-1):
            second_date = datetime.strptime(processed_rows[i][0], '%m/%d/%Y'
            first_date = datetime.strptime(processed_rows[i+1][0], '%m/%d/%Y
            if second_date == date:
                processed_data[i][2] = int(row[3])
                start = i
            else:
                if first_date == date:
                    processed_data[i+1][2] = int(row[3])
                    start = i
                else:
                    if first_date < date < second_date and row[3].isdigit():
                        processed_data[i+1][2] = int(row[3])
                        start = i


with open('Processed Data/' + company_name + '.csv', 'w', newline='') as csv
    csv_writer = csv.writer(csv_file)
    for row in processed_data:
        csv_writer.writerow(row)
```

```
['Cisco', 'SF Bay Area', 'Infrastructure', '4100', '0.05', '2022-11-16', 'Po
st-IPO', 'United States', '2.0']
```

In [4]:
```
df = pd.read_csv('Processed Data/' + company_name + '.csv')

ts1 = df.iloc[:, 1]
ts2 = df.iloc[:, 2]

results = grangercausalitytests(df[['Layoffs', 'Close']], maxlag=4, verbose=

print(results)

for lag in range(1, 5):
    print(f'Lag {lag}:')
    print(f'F-test p-value: {results[lag][0]["params_ftest"][1]}')
    print(f'Chi-squared p-value: {results[lag][0]["ssr_chi2test"][1]}')
    print('\n')
```

```
{1: ({'ssr_ftest': (0.19506614726493515, 0.6593777627979982, 148.0, 1), 'ssr
_chi2test': (0.19902019079057573, 0.6555128848116205, 1), 'lrtest': (0.19888
915017054387, 0.655618989966779, 1), 'params_ftest': (0.19506614726478214, 0
.6593777627981194, 148.0, 1.0)}, [<statsmodels.regression.linear_model.Regre
ssionResultsWrapper object at 0x7fefa1370f10>, <statsmodels.regression.linea
r_model.RegressionResultsWrapper object at 0x7fefa1370370>, array([[0., 1.,
0.]])]), 2: ({'ssr_ftest': (1.8492376661899852, 0.16104890830951132, 145.0,
2), 'ssr_chi2test': (3.8260089645310043, 0.14763614938565114, 2), 'lrtest':
(3.7780286520187474, 0.15122078980032644, 2), 'params_ftest': (1.84923766618
99415, 0.16104890830951787, 145.0, 2.0)}, [<statsmodels.regression.linear_mo
del.RegressionResultsWrapper object at 0x7fefd243dca0>, <statsmodels.regress
ion.linear_model.RegressionResultsWrapper object at 0x7fefd243d850>, array([
[0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.]])]), 3: ({'ssr_ftest': (1.2553797315704955, 0.29
21240011913806, 142.0, 3), 'ssr_chi2test': (3.95179394374656, 0.266716783193
43484, 3), 'lrtest': (3.900297537190454, 0.27243340912215636, 3), 'params_ft
est': (1.2553797315704842, 0.2921240011913826, 142.0, 3.0)}, [<statsmodels.r
egression.linear_model.RegressionResultsWrapper object at 0x7fefd243d7c0>, <
statsmodels.regression.linear_model.RegressionResultsWrapper object at 0x7fe
fd243dc70>, array([[0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0.]])]), 4: ({'ssr_ftest': (1.12760453451615
94, 0.34605749923378787, 139.0, 4), 'ssr_chi2test': (4.802459600241485, 0.30
817338133477945, 4), 'lrtest': (4.726187555447723, 0.3165631724820432, 4), '
params_ftest': (1.1276045345161352, 0.34605749923379747, 139.0, 4.0)}, [<sta
tsmodels.regression.linear_model.RegressionResultsWrapper object at 0x7fefa1
3e40d0>, <statsmodels.regression.linear_model.RegressionResultsWrapper objec
t at 0x7fefa13e41f0>, array([[0., 0., 0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1., 0.]])])}
Lag 1:
F-test p-value: 0.6593777627981194
Chi-squared p-value: 0.6555128848116205


Lag 2:
F-test p-value: 0.16104890830951787
Chi-squared p-value: 0.14763614938565114


Lag 3:
F-test p-value: 0.2921240011913826
Chi-squared p-value: 0.26671678319343484


Lag 4:
F-test p-value: 0.34605749923379747
Chi-squared p-value: 0.30817338133477945
```

In [5]:
```
df = df.iloc[::-1]
df['Pct_Change'] = round(df['Close'].pct_change(periods=1)*100,2)
if df['Pct_Change'].isnull().values.any():
    df['Pct_Change'] = df['Pct_Change'].fillna(0)

def direction(row):
    if(row['Pct_Change'] < 0):
        return -1
    else:
        return 1

df['Direction_Change'] = df.apply(direction, axis=1)

#print(df)
df.to_csv('Processed Data/' + company_name + '.csv', index=False, mode='w')
```

In [6]:
```
X = df[['Layoffs','Close']]
y = df['Close']
#y = df['Pct_Change']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran

xgb_model = xgb.XGBRegressor(objective ='reg:squarederror', colsample_bytree
                max_depth = 5, alpha = 10, n_estimators = 100)


xgb_model.fit(X_train, y_train)

y_pred = xgb_model.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print("RMSE:", rmse)

r2 = r2_score(y_test, y_pred)
print('R2 score: {:.2f}'.format(r2))
```

```
RMSE: 0.21648598470263977
R2 score: 1.00
```

In [7]:
```
df = pd.read_csv('Processed Data/' + company_name + '.csv')
print(df)
```

```
           Date  Close  Layoffs  Pct_Change  Direction_Change
0    03/04/2020  40.87        0        0.00                 1
1    03/11/2020  38.87        0       -4.89                -1
2    03/18/2020  35.43        0       -8.85                -1
3    03/25/2020  36.84        0        3.98                 1
4    04/01/2020  39.47        0        7.14                 1
..          ...    ...      ...         ...               ...
147  02/02/2023  48.66        0        2.06                 1
148  02/09/2023  47.55        0       -2.28                -1
149  02/16/2023  48.45        0        1.89                 1
150  02/24/2023  49.49        0        2.15                 1
151  03/03/2023  48.66        0       -1.68                -1

[152 rows x 5 columns]
```

In [8]:
```
inputcols =['Layoffs','Close']
outputcols = ['Direction_Change']
data = df[inputcols]
scaler = StandardScaler()
scaled = scaler.fit_transform(data)
nlags = 4
X = []
y = []
for i in range(nlags, len(scaled)):
    X.append(scaled[i - nlags:i])
    if(close_price_as_predictive_column):
        y.append(scaled[i,1]) # for close price as a predictive column


if(not close_price_as_predictive_column):
    y = np.array(df.loc[nlags:,outputcols]) #for direction change as a predi
X = np.array(X)
y = np.array(y)

sc_predict = StandardScaler()
sc_predict.fit_transform(np.array(data.iloc[:, 1]).reshape(-1, 1))
```

Out[8]:
```
array([[-1.21066426],
       [-1.53480748],
       [-2.09233381],
       [-1.86381284],
       [-1.43756451],
       [-1.26576861],
       [-1.08586912],
       [-1.06155838],
       [-0.90434892],
       [-1.14907705],
       [-0.85410672],
       [-0.57372283],
       [-0.40840979],
       [-0.22364816],
       [-0.30306325],
       [-0.40678908],
```

```
[-0.4797213 ],
[-0.35006401],
[-0.35492616],
[-0.30954611],
[-0.22040673],
[-0.25930391],
[-0.12154304],
[-0.61910288],
[-1.00807475],
[-0.99835045],
[-1.15069776],
[-1.33383868],
[-1.3857016 ],
[-1.58667039],
[-1.55911822],
[-1.37759802],
[-1.40028804],
[-1.63529187],
[-2.00319443],
[-1.73253484],
[-1.22849214],
[-1.10207628],
[-0.84438242],
[-0.64179291],
[-0.62882718],
[-0.56723997],
[-0.59479214],
[-0.62720647],
[-0.47323844],
[-0.52672207],
[-0.49916989],
[-0.18961312],
[-0.14909522],
[-0.40516836],
[-0.45865199],
[-0.33547757],
[ 0.10697792],
[ 0.18801373],
[ 0.49594979],
[ 0.58995132],
[ 0.5478127 ],
[ 0.62560707],
[ 0.51377766],
[ 0.44408687],
[ 0.70502216],
[ 0.71474646],
[ 0.75202293],
[ 0.74716078],
[ 0.95461244],
[ 0.89626666],
[ 0.72447076],
[ 0.76660937],
[ 0.81361014],
```

```
[ 0.85250733],
[ 0.92706027],
[ 1.11020119],
[ 1.20096129],
[ 1.23499632],
[ 1.33710144],
[ 1.74228046],
[ 1.76497049],
[ 1.6353132 ],
[ 1.41975796],
[ 1.25444492],
[ 1.07292472],
[ 1.03564825],
[ 1.07940758],
[ 1.17340911],
[ 1.25120349],
[ 1.45055157],
[ 1.43758584],
[ 1.08751116],
[ 1.11506333],
[ 1.32089428],
[ 1.57210527],
[ 1.9578357 ],
[ 2.36625616],
[ 2.26901319],
[ 2.16042521],
[ 1.87517918],
[ 1.23013418],
[ 1.17665055],
[ 1.13775336],
[ 0.96109531],
[ 1.19934057],
[ 1.2123063 ],
[ 1.08913188],
[ 1.1717884 ],
[ 1.11506333],
[ 1.18637484],
[ 1.04861397],
[ 0.56564058],
[ 0.62398636],
[ 0.25932524],
[ 0.24960094],
[ 0.10859864],
[-0.47810058],
[-0.59155071],
[-0.47161772],
[-0.74227731],
[-0.7617259 ],
[-0.80872667],
[-0.92703894],
[-0.88003818],
[-0.75524304],
[-0.60937859],
```

```
       [-0.48782488],
       [-0.48134202],
       [-0.18475097],
       [-0.11506018],
       [-0.48134202],
       [-0.53482565],
       [-0.66934508],
       [-1.05021337],
       [-1.25442359],
       [-1.12800774],
       [-1.366253  ],
       [-1.0437305 ],
       [-0.62720647],
       [-0.57858498],
       [-0.58992999],
       [-0.43109982],
       [-0.02105865],
       [ 0.17180657],
       [ 0.05835644],
       [ 0.01945926],
       [-0.14099164],
       [-0.12964662],
       [ 0.00325209],
       [-0.07778371],
       [-0.1069566 ],
       [ 0.05187358],
       [-0.12802591],
       [ 0.01783854],
       [ 0.18639301],
       [ 0.05187358]])
```

In [9]:
```
print(X.shape)
print (y.shape)
y = y.reshape([y.shape[0],1])
print (y.shape)
```

```
(148, 4, 2)
(148, 1)
(148, 1)
```

In [10]:
```
X_train = X[:103,:,:]
X_test = X[103:,:,:]
y_train = y[:103,:]
y_test = y[103:,:]
print('X_train shape : ', X_train.shape)
print('X_test shape : ', X_test.shape)
print('y_train shape : ', y_train.shape)
print('y_test shape : ', y_test.shape)
```

```
X_train shape :  (103, 4, 2)
X_test shape :  (45, 4, 2)
y_train shape :  (103, 1)
y_test shape :  (45, 1)
```

In [11]:
```
# Initializing the Neural Network based on LSTM
model = Sequential()

# Adding 1st LSTM layer
model.add(LSTM(units=64, return_sequences=True, input_shape=(nlags, len(inpu

# Adding 2nd LSTM layer
model.add(LSTM(units=10, return_sequences=False))

# Adding Dropout
model.add(Dropout(0.2))

# Output layer
if(close_price_as_predictive_column):
    model.add(Dense(units=1, activation='linear'))
else:
    model.add(Dense(units=1, activation='tanh'))

# Compiling the Neural Network
if(close_price_as_predictive_column):
    model.compile(optimizer = 'adam', loss='mean_squared_error')
else:
    model.compile(loss='binary_crossentropy', optimizer='nadam',metrics=['ac
model.summary()
```

```
2023-05-20 03:48:44.799670: I tensorflow/core/platform/cpu_feature_guard.cc:
193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Lib
rary (oneDNN) to use the following CPU instructions in performance-critical
operations:  SSE4.1 SSE4.2
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 4, 64)             17152

 lstm_1 (LSTM)               (None, 10)                3000

 dropout (Dropout)           (None, 10)                0

 dense (Dense)               (None, 1)                 11

=================================================================
Total params: 20,163
Trainable params: 20,163
Non-trainable params: 0
_________________________________________________________________
```

In [12]:
```
es = EarlyStopping(monitor='val_loss', min_delta=1e-10, patience=10, verbose
rlr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=10, verbose
mcp = ModelCheckpoint(filepath='weights.h5', monitor='val_loss', verbose=1,

tb = TensorBoard('logs')

history = model.fit(X_train, y_train, shuffle=True, epochs=50, callbacks=[es
```

```
Epoch 1/50
1/6 [====>.....................] - ETA: 12s - loss: -0.0903 - accuracy:
0.0000e+00
Epoch 1: val_loss improved from inf to -2.20356, saving model to weights.h5
6/6 [==============================] - 3s 126ms/step - loss: 3.1775 - accura
cy: 0.0000e+00 - val_loss: -2.2036 - val_accuracy: 0.0000e+00 - lr: 0.0010
Epoch 2/50
1/6 [====>.....................] - ETA: 0s - loss: 3.2869 - accuracy: 0.
0000e+00
Epoch 2: val_loss did not improve from -2.20356
6/6 [==============================] - 0s 11ms/step - loss: 2.6256 - accurac
y: 0.0000e+00 - val_loss: -2.2036 - val_accuracy: 0.0000e+00 - lr: 0.0010
Epoch 3/50
1/6 [====>.....................] - ETA: 0s - loss: -1.1279 - accuracy: 0
.0000e+00
Epoch 3: val_loss did not improve from -2.20356
6/6 [==============================] - 0s 12ms/step - loss: 1.7905 - accurac
y: 0.0000e+00 - val_loss: -2.2036 - val_accuracy: 0.0000e+00 - lr: 0.0010
Epoch 4/50
1/6 [====>.....................] - ETA: 0s - loss: 0.3924 - accuracy: 0.
0000e+00
Epoch 4: val_loss improved from -2.20356 to -2.91778, saving model to weight
s.h5
6/6 [==============================] - 0s 13ms/step - loss: 1.8872 - accurac
y: 0.0000e+00 - val_loss: -2.9178 - val_accuracy: 0.0000e+00 - lr: 0.0010
Epoch 5/50
1/6 [====>.....................] - ETA: 0s - loss: 3.0530 - accuracy: 0.
0000e+00
Epoch 5: val_loss did not improve from -2.91778
6/6 [==============================] - 0s 9ms/step - loss: 1.3304 - accuracy
: 0.0000e+00 - val_loss: -0.7277 - val_accuracy: 0.0000e+00 - lr: 0.0010
Epoch 6/50
1/6 [====>.....................] - ETA: 0s - loss: 1.8208 - accuracy: 0.
0000e+00
Epoch 6: val_loss did not improve from -2.91778
6/6 [==============================] - 0s 13ms/step - loss: 0.9452 - accurac
y: 0.0000e+00 - val_loss: -2.3751 - val_accuracy: 0.0000e+00 - lr: 0.0010
Epoch 7/50
1/6 [====>.....................] - ETA: 0s - loss: 1.8940 - accuracy: 0.
0000e+00
Epoch 7: val_loss did not improve from -2.91778
6/6 [==============================] - 0s 9ms/step - loss: 1.0320 - accuracy
: 0.0000e+00 - val_loss: -0.2386 - val_accuracy: 0.0000e+00 - lr: 0.0010
Epoch 8/50
1/6 [====>.....................] - ETA: 0s - loss: 1.4647 - accuracy: 0.
```

```
0000e+00
Epoch 8: val_loss did not improve from -2.91778
6/6 [==============================] - 0s 9ms/step - loss: 0.7071 - accuracy
: 0.0000e+00 - val_loss: -0.1626 - val_accuracy: 0.0000e+00 - lr: 0.0010
Epoch 9/50
1/6 [====>.........................] - ETA: 0s - loss: 1.1860 - accuracy: 0.
0000e+00
Epoch 9: val_loss did not improve from -2.91778
6/6 [==============================] - 0s 9ms/step - loss: 0.6742 - accuracy
: 0.0000e+00 - val_loss: -0.1236 - val_accuracy: 0.0000e+00 - lr: 0.0010
Epoch 10/50
1/6 [====>.........................] - ETA: 0s - loss: 0.9995 - accuracy: 0.
0000e+00
Epoch 10: val_loss did not improve from -2.91778
6/6 [==============================] - 0s 9ms/step - loss: 0.6183 - accuracy
: 0.0000e+00 - val_loss: -0.1030 - val_accuracy: 0.0000e+00 - lr: 0.0010
Epoch 11/50
1/6 [====>.........................] - ETA: 0s - loss: 0.7319 - accuracy: 0.
0000e+00
Epoch 11: val_loss did not improve from -2.91778
6/6 [==============================] - 0s 9ms/step - loss: 0.5618 - accuracy
: 0.0000e+00 - val_loss: -0.0924 - val_accuracy: 0.0000e+00 - lr: 0.0010
Epoch 12/50
1/6 [====>.........................] - ETA: 0s - loss: 0.7586 - accuracy: 0.
0000e+00
Epoch 12: val_loss did not improve from -2.91778
6/6 [==============================] - 0s 9ms/step - loss: 0.6752 - accuracy
: 0.0000e+00 - val_loss: -0.0709 - val_accuracy: 0.0000e+00 - lr: 0.0010
Epoch 13/50
1/6 [====>.........................] - ETA: 0s - loss: 0.2679 - accuracy: 0.
0000e+00
Epoch 13: val_loss did not improve from -2.91778
6/6 [==============================] - 0s 9ms/step - loss: 0.6621 - accuracy
: 0.0000e+00 - val_loss: -0.0640 - val_accuracy: 0.0000e+00 - lr: 0.0010
Epoch 14/50
1/6 [====>.........................] - ETA: 0s - loss: 0.7015 - accuracy: 0.
0000e+00
Epoch 14: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.

Epoch 14: val_loss did not improve from -2.91778
6/6 [==============================] - 0s 9ms/step - loss: 0.6302 - accuracy
: 0.0000e+00 - val_loss: -0.0620 - val_accuracy: 0.0000e+00 - lr: 0.0010
Epoch 14: early stopping
```

In [13]:
```
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 2))
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 2))
model.evaluate(X_test, y_test)
predictions = model.predict(X_test)
predictions_train = model.predict(X_train)

if(close_price_as_predictive_column):
    y_pred = sc_predict.inverse_transform(predictions)
    y_pred_train = sc_predict.inverse_transform(predictions_train)
else:
    predictions[predictions >= 0] = 1
    predictions[predictions < 0] = -1
    predictions = predictions.astype('int32')

print(mean_squared_error(y_test,predictions))
```

```
2/2 [==============================] - 0s 2ms/step - loss: 0.2761 - accuracy
: 0.0000e+00
2/2 [==============================] - 0s 3ms/step
4/4 [==============================] - 0s 2ms/step
2.2222222222222223
```

In [14]:
```
from sklearn.metrics import classification_report
if(not close_price_as_predictive_column):
    print(classification_report(y_test, predictions))
else:
    plt.rcParams["figure.figsize"] = [10,5]
    plt.rcParams["figure.autolayout"] = True

    plt.plot(df.loc[:,'Date'], df.loc[:,'Close'], color='b', label='Actual S
    plt.plot(df.loc[:102,'Date'], y_pred_train, color='orange', label='Train
    plt.plot(df.loc[103+nlags:,'Date'], y_pred, color='r', label='Predicted

    plt.xticks(np.arange(0, len(df), 20))

    plt.legend(shadow=True)
    plt.title(company_name + ' (LO,Close price)',fontsize=12)
    plt.xlabel('Timeline', fontsize=10)
    plt.ylabel('Stock Price Value', fontsize=10)
    plt.show()
```

```
              precision    recall  f1-score   support

          -1       0.33      0.04      0.07        24
           1       0.45      0.90      0.60        21

    accuracy                           0.44        45
   macro avg       0.39      0.47      0.34        45
weighted avg       0.39      0.44      0.32        45
```

In [ ]: