In [1]:
```
#inporting libraries
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')
plt.style.use("fivethirtyeight")
%matplotlib inline

# For reading stock data from yahoo
from pandas_datareader.data import DataReader

# For time stamps
from datetime import datetime
```

```
In C:\Users\vasan\anaconda3\lib\site-packages\matplotlib\mpl-data\stylelib\_classic_tes
t.mplstyle:
The text.latex.preview rcparam was deprecated in Matplotlib 3.3 and will be removed two
minor releases later.
In C:\Users\vasan\anaconda3\lib\site-packages\matplotlib\mpl-data\stylelib\_classic_tes
t.mplstyle:
The mathtext.fallback_to_cm rcparam was deprecated in Matplotlib 3.3 and will be removed
two minor releases later.
In C:\Users\vasan\anaconda3\lib\site-packages\matplotlib\mpl-data\stylelib\_classic_tes
t.mplstyle: Support for setting the 'mathtext.fallback_to_cm' rcParam is deprecated sinc
e 3.3 and will be removed two minor releases later; use 'mathtext.fallback : 'cm' instea
d.
In C:\Users\vasan\anaconda3\lib\site-packages\matplotlib\mpl-data\stylelib\_classic_tes
t.mplstyle:
The validate_bool_maybe_none function was deprecated in Matplotlib 3.3 and will be remov
ed two minor releases later.
In C:\Users\vasan\anaconda3\lib\site-packages\matplotlib\mpl-data\stylelib\_classic_tes
t.mplstyle:
The savefig.jpeg_quality rcparam was deprecated in Matplotlib 3.3 and will be removed tw
o minor releases later.
In C:\Users\vasan\anaconda3\lib\site-packages\matplotlib\mpl-data\stylelib\_classic_tes
t.mplstyle:
The keymap.all_axes rcparam was deprecated in Matplotlib 3.3 and will be removed two min
or releases later.
In C:\Users\vasan\anaconda3\lib\site-packages\matplotlib\mpl-data\stylelib\_classic_tes
t.mplstyle:
The animation.avconv_path rcparam was deprecated in Matplotlib 3.3 and will be removed t
wo minor releases later.
In C:\Users\vasan\anaconda3\lib\site-packages\matplotlib\mpl-data\stylelib\_classic_tes
t.mplstyle:
The animation.avconv_args rcparam was deprecated in Matplotlib 3.3 and will be removed t
wo minor releases later.
```

In [2]:
```
# The nse stocks we'll use for this analysis
tech_list = ['TCS', 'INFY', 'RELI']
```

In [3]:
```
# Set up End and Start times for data grab
end = datetime.now()
start = datetime(end.year - 1, end.month, end.day)
```

In [4]:
```
#For loop for grabin data and setting as a dataframe
```

```
for stock in tech_list:
    # Set DataFrame as the Stock Ticker
    globals()[stock] = DataReader(stock, 'stooq', start, end)
```

In [5]:
```
company_list = [TCS, INFY, RELI]
company_name = ['TCS', 'INFY', 'RELIANCE']

for company, com_name in zip(company_list, company_name):
    company["company_name"] = com_name
```

In [6]:
```
df = pd.concat(company_list, axis=0)
df.tail(10)
```

Out[6]:

Date	Open	High	Low	Close	Volume	company_name
2021-01-22	5.74	6.19	5.34	5.91	3730	RELIANCE
2021-01-21	5.83	5.91	5.14	5.74	6164	RELIANCE
2021-01-20	6.00	6.00	5.74	5.83	2886	RELIANCE
2021-01-19	6.43	6.43	5.74	5.87	1608	RELIANCE
2021-01-15	6.60	6.60	5.73	6.00	1783	RELIANCE
2021-01-14	5.94	6.51	5.70	5.70	2436	RELIANCE
2021-01-13	6.06	6.26	5.75	5.96	1385	RELIANCE
2021-01-12	6.26	6.26	5.91	6.26	198	RELIANCE
2021-01-11	6.03	6.06	5.83	5.91	1384	RELIANCE
2021-01-08	6.38	6.38	5.84	6.26	1008	RELIANCE

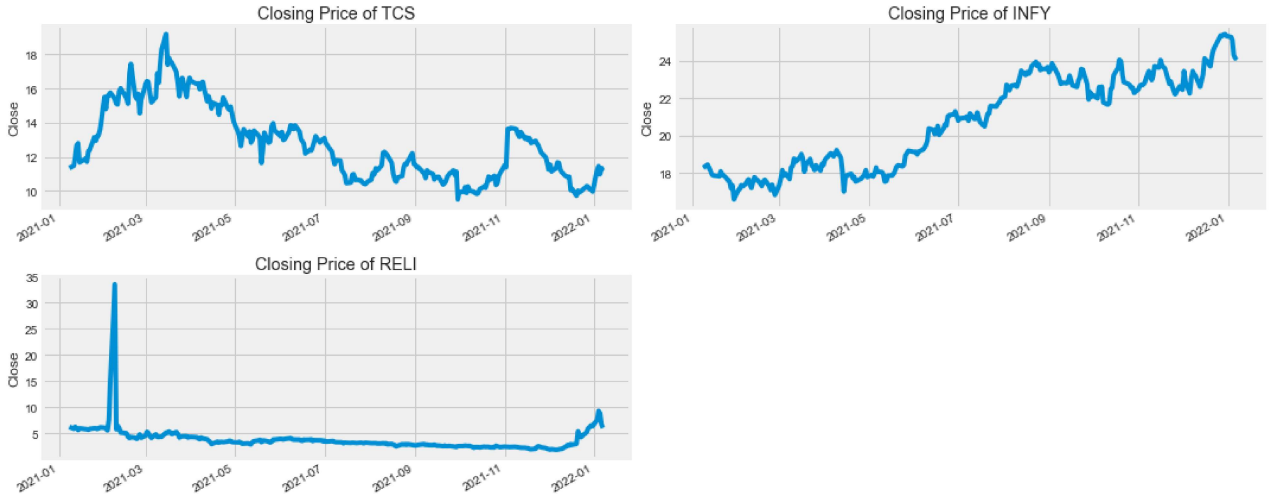
In [7]:
```
TCS.describe()
```

Out[7]:

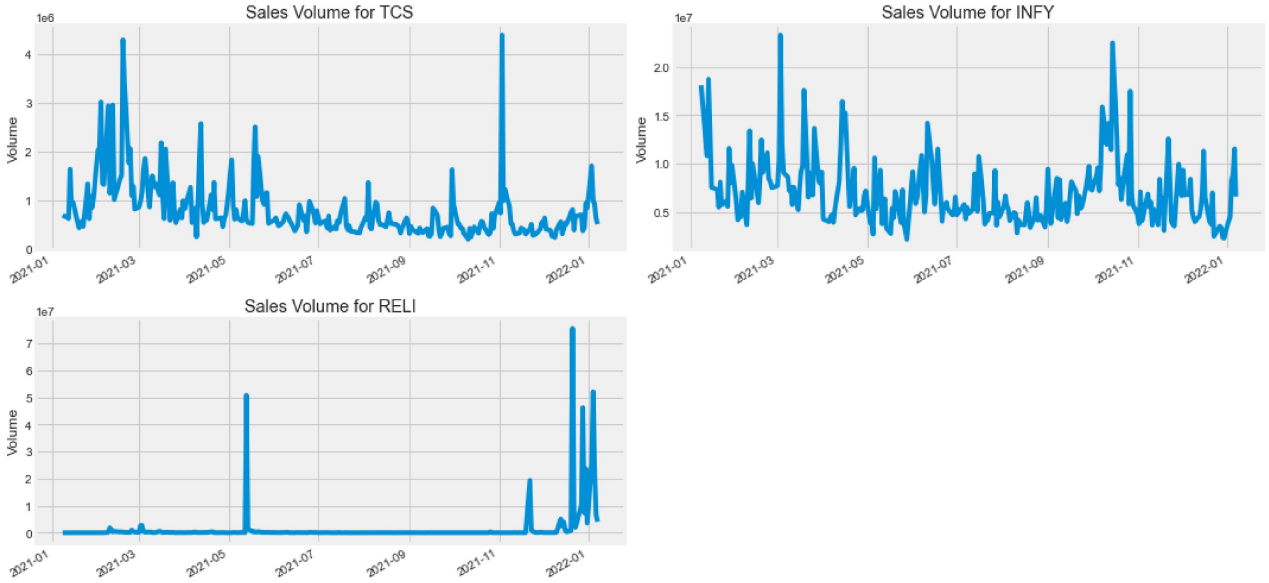
	Open	High	Low	Close	Volume
count	253.000000	253.000000	253.000000	253.000000	2.530000e+02
mean	12.798103	13.142382	12.430288	12.775929	8.016678e+05
std	2.146293	2.277328	2.012168	2.142961	6.097321e+05
min	9.570000	9.940000	9.250000	9.520000	2.050580e+05
25%	11.000000	11.210000	10.730000	10.990000	4.413970e+05
50%	12.480000	12.770000	12.095000	12.410000	6.153570e+05
75%	14.200000	14.557300	13.600000	13.970000	9.406930e+05
max	19.150000	19.310000	17.780000	19.190000	4.395301e+06

Data Visuliaztion

In [8]:
```
plt.figure(figsize=(15, 6))
plt.subplots_adjust(top=1.25, bottom=1.2)

for i, company in enumerate(company_list, 1):
    plt.subplot(2, 2, i)
    company['Close'].plot()
    plt.ylabel('Close')
    plt.xlabel(None)
    plt.title(f"Closing Price of {tech_list[i - 1]}")

plt.tight_layout()
```



In [9]:
```
# Now let's plot the total volume of stock being traded each day
plt.figure(figsize=(15, 7))
plt.subplots_adjust(top=1.25, bottom=1.2)

for i, company in enumerate(company_list, 1):
    plt.subplot(2, 2, i)
    company['Volume'].plot()
    plt.ylabel('Volume')
    plt.xlabel(None)
    plt.title(f"Sales Volume for {tech_list[i - 1]}")

plt.tight_layout()
```

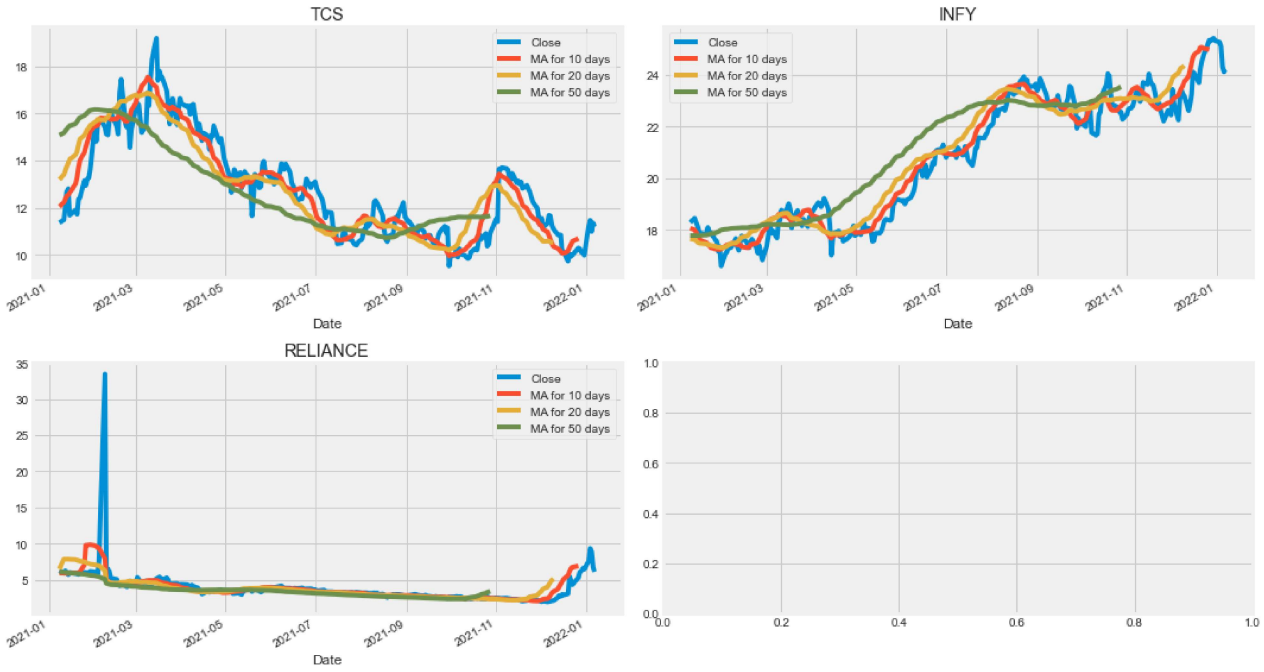


In [10]:

```
ma_day = [10, 20, 50]

for ma in ma_day:
    for company in company_list:
        column_name = f"MA for {ma} days"
        company[column_name] = company['Close'].rolling(ma).mean()
```

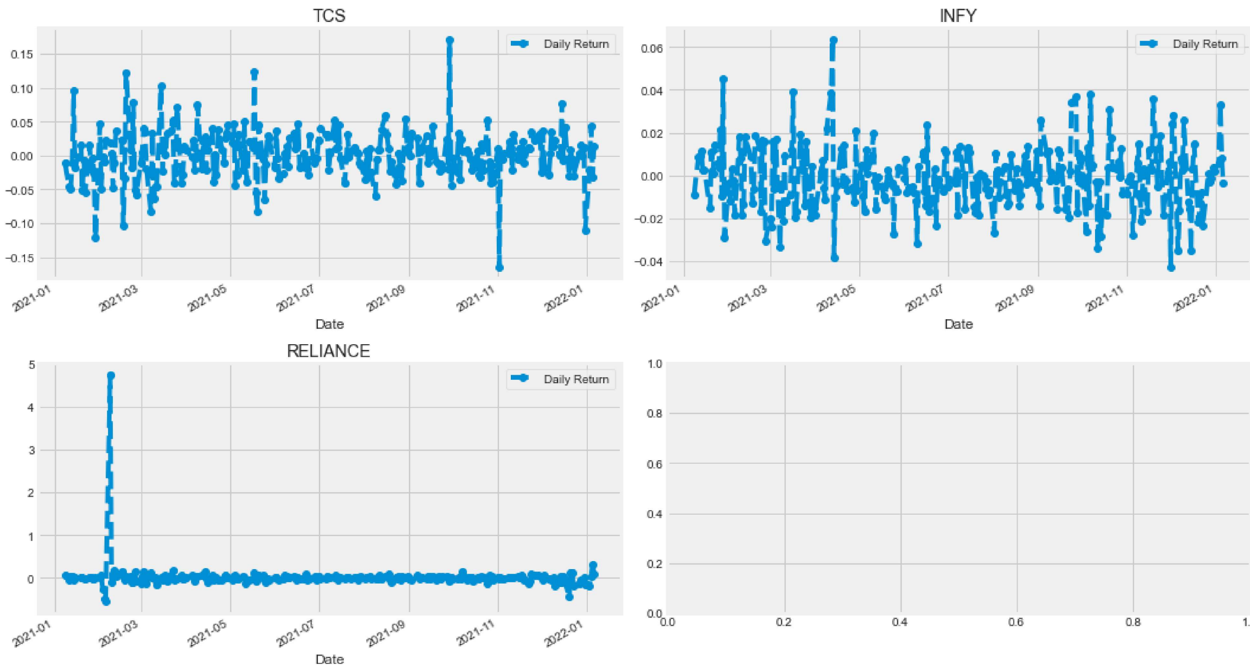
In [11]:

```
fig, axes = plt.subplots(nrows=2, ncols=2)
fig.set_figheight(8)
fig.set_figwidth(15)

TCS[['Close', 'MA for 10 days', 'MA for 20 days', 'MA for 50 days']].plot(ax=axes[0,0])
axes[0,0].set_title('TCS')

INFY[['Close', 'MA for 10 days', 'MA for 20 days', 'MA for 50 days']].plot(ax=axes[0,1]
axes[0,1].set_title('INFY')

RELI[['Close', 'MA for 10 days', 'MA for 20 days', 'MA for 50 days']].plot(ax=axes[1,0]
axes[1,0].set_title('RELIANCE')


fig.tight_layout()
```



In [12]:

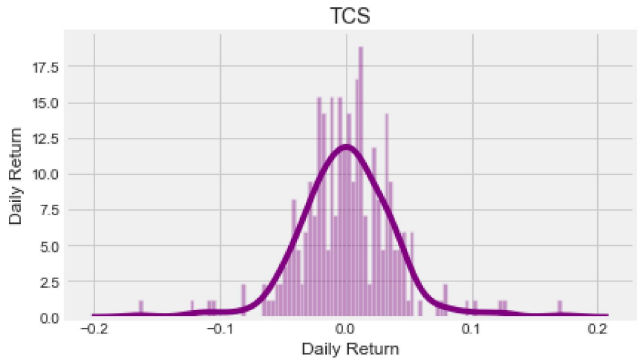
```
# We'll use pct_change to find the percent change for each day
for company in company_list:
    company['Daily Return'] = company['Close'].pct_change()

# Then we'll plot the daily return percentage
fig, axes = plt.subplots(nrows=2, ncols=2)
fig.set_figheight(8)
fig.set_figwidth(15)

TCS['Daily Return'].plot(ax=axes[0,0], legend=True, linestyle='--', marker='o')
axes[0,0].set_title('TCS')

INFY['Daily Return'].plot(ax=axes[0,1], legend=True, linestyle='--', marker='o')
axes[0,1].set_title('INFY')

RELI['Daily Return'].plot(ax=axes[1,0], legend=True, linestyle='--', marker='o')
axes[1,0].set_title('RELIANCE')

fig.tight_layout()
```

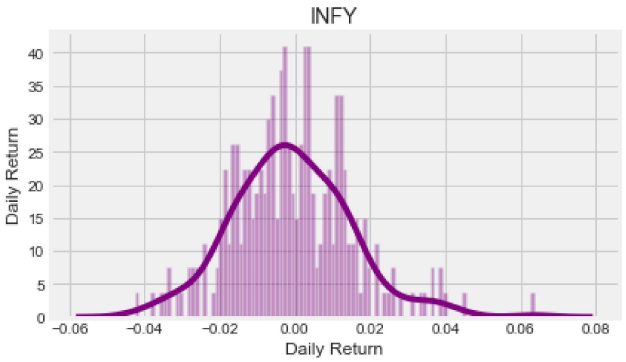


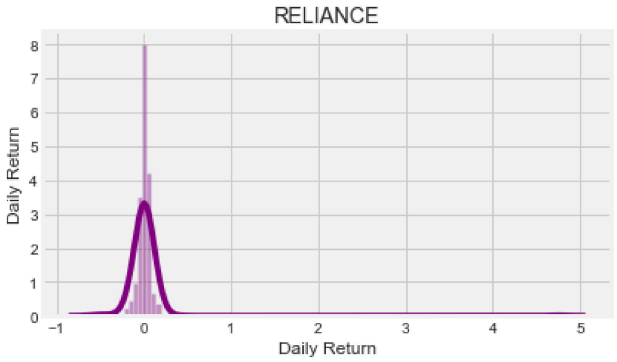
In [13]:

```
# Note the use of dropna() here, otherwise the NaN values can't be read by seaborn
plt.figure(figsize=(12, 7))

for i, company in enumerate(company_list, 1):
    plt.subplot(2, 2, i)
    sns.distplot(company['Daily Return'].dropna(), bins=100, color='purple')
    plt.ylabel('Daily Return')
    plt.title(f'{company_name[i - 1]}')

plt.tight_layout()
```

```
C:\Users\vasan\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\vasan\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\vasan\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```







In [14]:
```
# Grab all the closing prices for the tech stock list into one DataFrame
closing_df = DataReader(tech_list, 'stooq', start, end)['Close']

# Let's take a quick look
closing_df.head()
```

Out[14]:

Symbols	TCS	INFY	RELI
Date			
2022-01-07	11.20	24.20	6.10
2022-01-06	11.35	24.11	6.75
2022-01-05	10.99	24.30	8.90
2022-01-04	11.46	25.11	9.32
2022-01-03	11.21	25.26	7.58

In [15]:
```
# Make a new tech returns DataFrame
tech_rets = closing_df.pct_change()
tech_rets.head()
```

Out[15]:

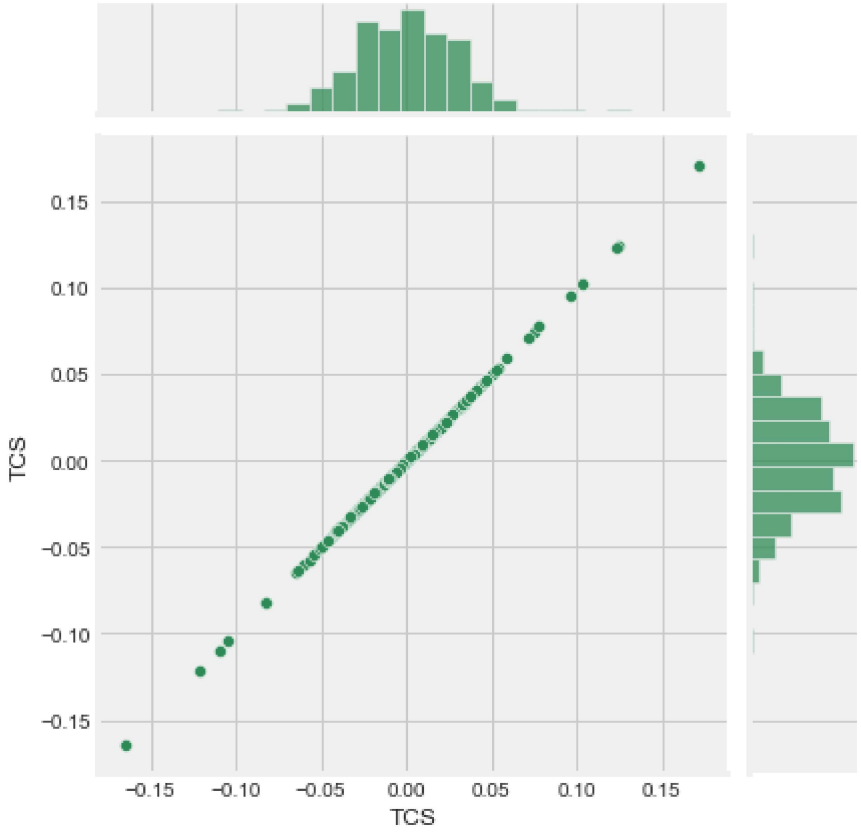
Symbols	TCS	INFY	RELI
Date			
2022-01-07	NaN	NaN	NaN
2022-01-06	0.013393	-0.003719	0.106557
2022-01-05	-0.031718	0.007881	0.318519
2022-01-04	0.042766	0.033333	0.047191

Symbols	TCS	INFY	RELI
Date			
2022-01-03	-0.021815	0.005974	-0.186695

In [16]:
```
# Comparing TCS to itself should show a perfectly linear relationship
sns.jointplot('TCS', 'TCS', tech_rets, kind='scatter', color='seagreen')
```

```
C:\Users\vasan\anaconda3\lib\site-packages\seaborn\_decorators.py:43: FutureWarning: Pas
s the following variables as keyword args: x, y, data. From version 0.12, the only valid
positional argument will be `data`, and passing other arguments without an explicit keyw
ord will result in an error or misinterpretation.
  FutureWarning
```

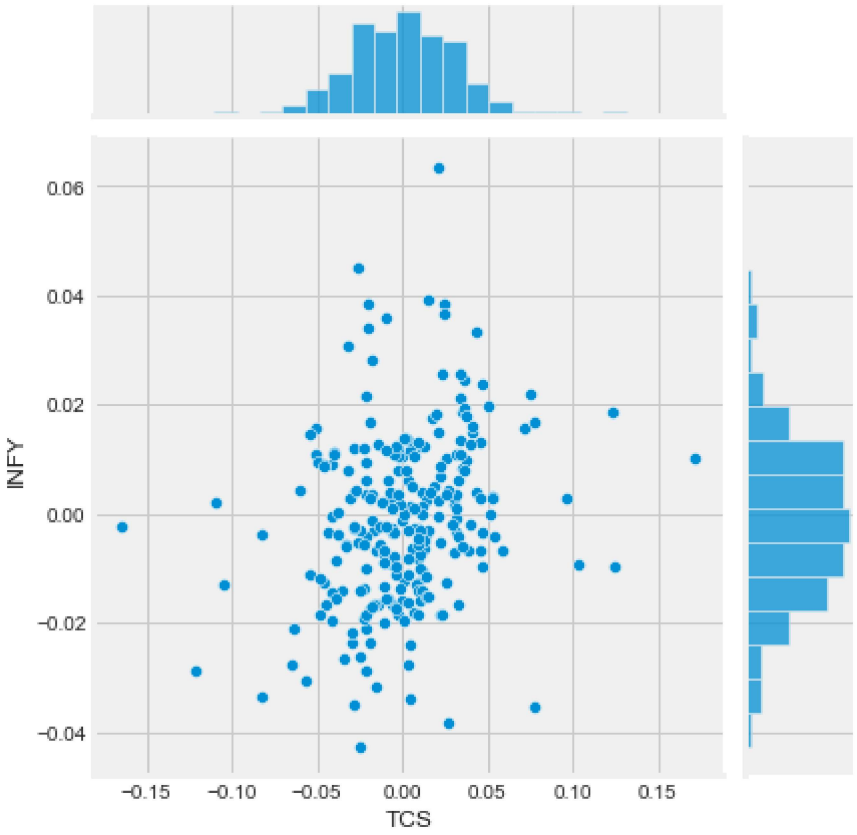
Out[16]: <seaborn.axisgrid.JointGrid at 0x22ac21bd978>



In [17]:
```
# We'll use joinplot to compare the daily returns of TCS and Infy
sns.jointplot('TCS', 'INFY', tech_rets, kind='scatter')
```

```
C:\Users\vasan\anaconda3\lib\site-packages\seaborn\_decorators.py:43: FutureWarning: Pas
s the following variables as keyword args: x, y, data. From version 0.12, the only valid
positional argument will be `data`, and passing other arguments without an explicit keyw
ord will result in an error or misinterpretation.
  FutureWarning
```

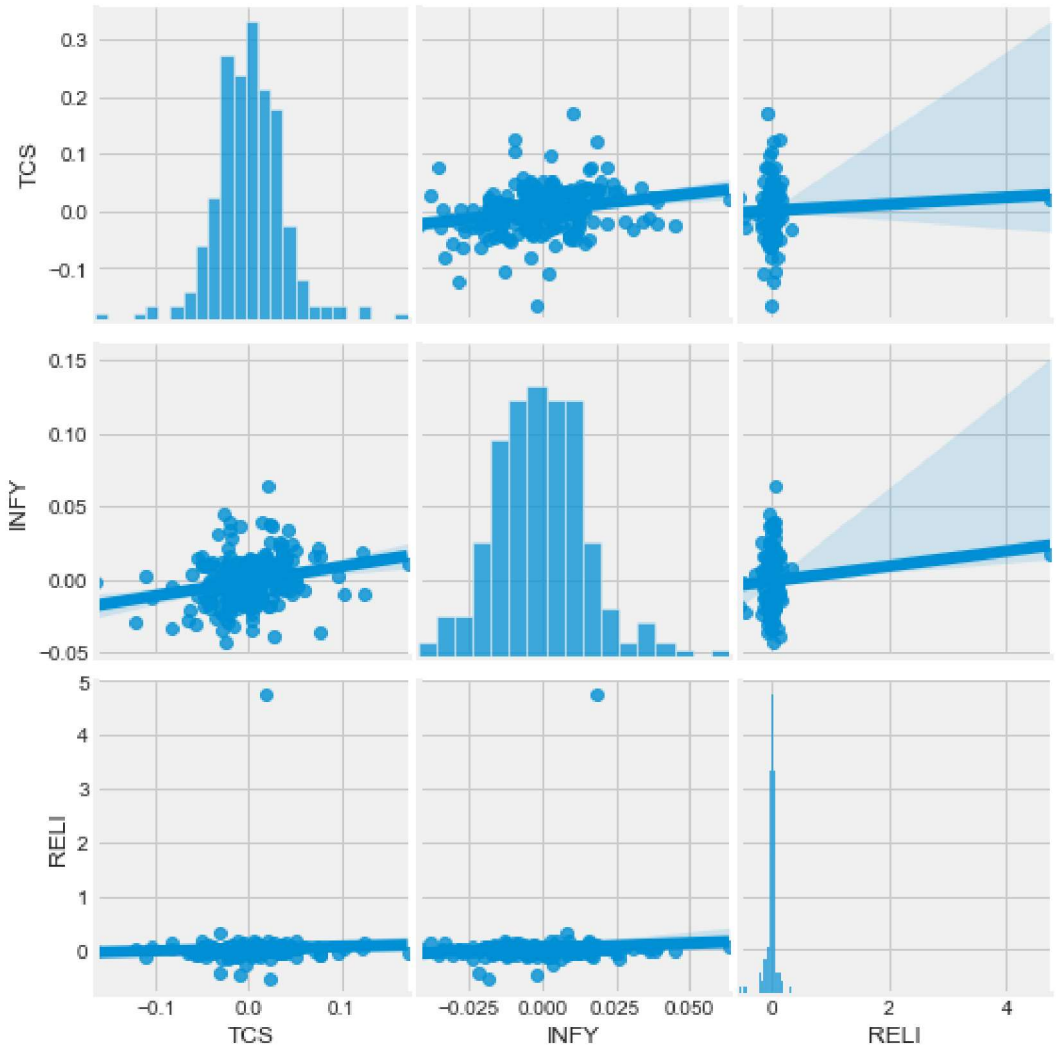
Out[17]: <seaborn.axisgrid.JointGrid at 0x22ac2225358>



In [18]:

```
# We can simply call pairplot on our DataFrame for an automatic visual analysis
# of all the comparisons

sns.pairplot(tech_rets, kind='reg')
```

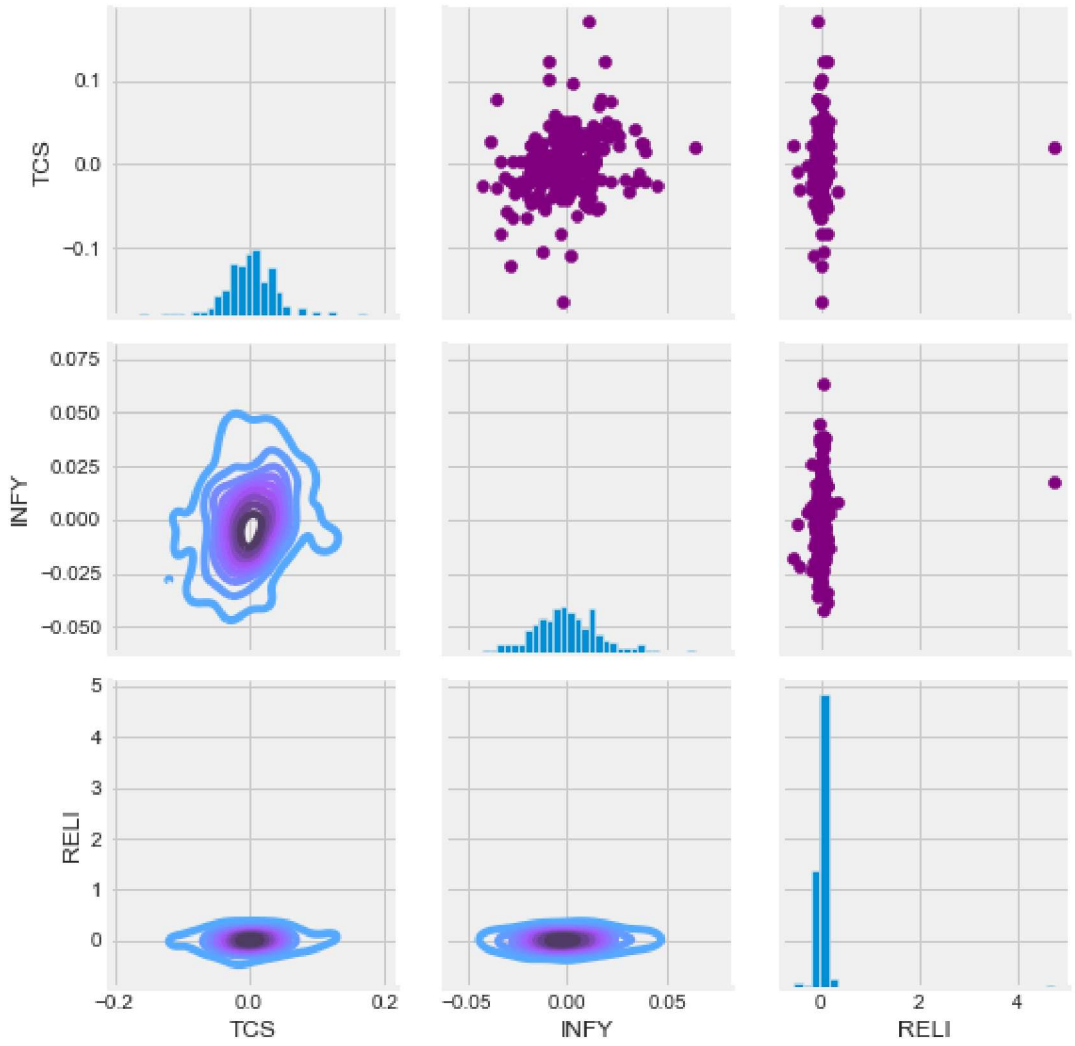
Out[18]: <seaborn.axisgrid.PairGrid at 0x22abfe2e438>



In [19]:

```
# Set up our figure by naming it returns_fig, call PairPLot on the DataFrame
return_fig = sns.PairGrid(tech_rets.dropna())

# Using map_upper we can specify what the upper triangle will look like.
return_fig.map_upper(plt.scatter, color='purple')

# We can also define the lower triangle in the figure, inclufing the plot type (kde)
# or the color map (BluePurple)
return_fig.map_lower(sns.kdeplot, cmap='cool_d')

# Finally we'll define the diagonal as a series of histogram plots of the daily return
return_fig.map_diag(plt.hist, bins=30)
```

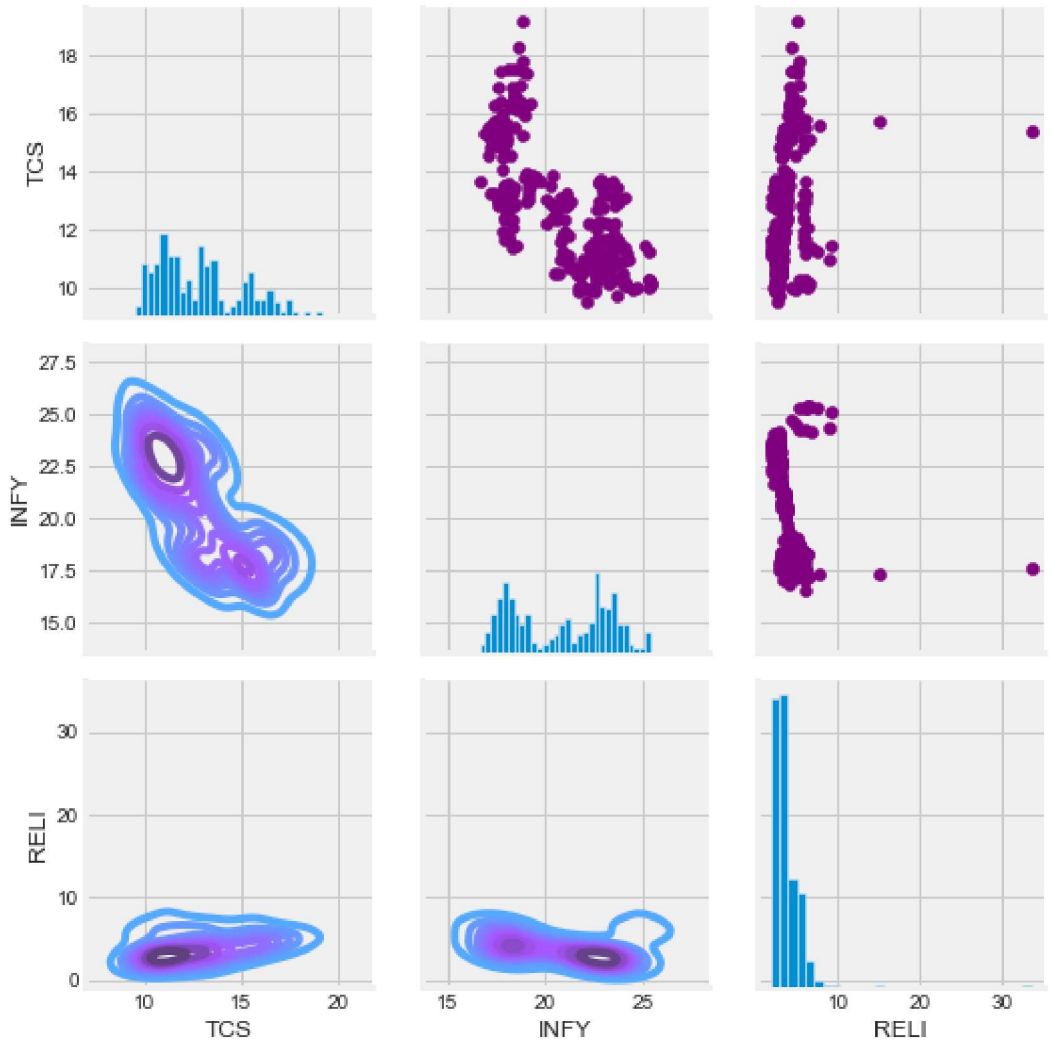
Out[19]: <seaborn.axisgrid.PairGrid at 0x22abf3387f0>



In [20]:

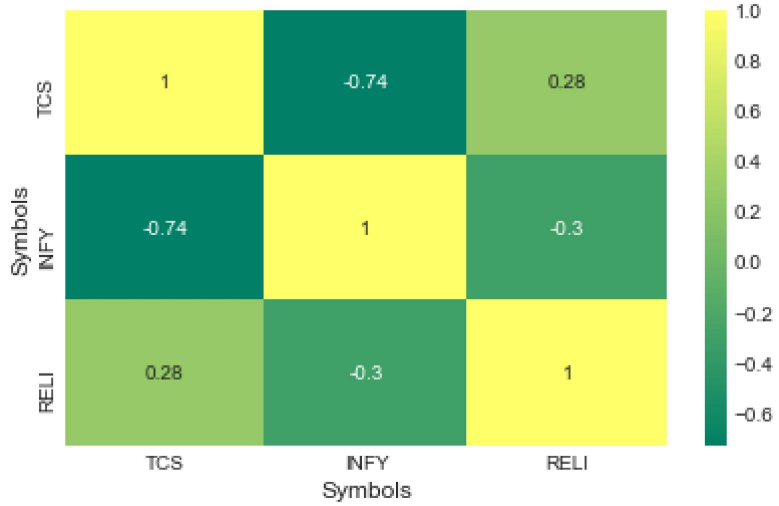
```
# Set up our figure by naming it returns_fig, call PairPLot on the DataFrame
returns_fig = sns.PairGrid(closing_df)

# Using map_upper we can specify what the upper triangle will look like.
returns_fig.map_upper(plt.scatter,color='purple')

# We can also define the lower triangle in the figure, inclufing the plot type (kde) or
returns_fig.map_lower(sns.kdeplot,cmap='cool_d')

# Finally we'll define the diagonal as a series of histogram plots of the daily return
returns_fig.map_diag(plt.hist,bins=30)
```

Out[20]: <seaborn.axisgrid.PairGrid at 0x22ac38bdeb8>



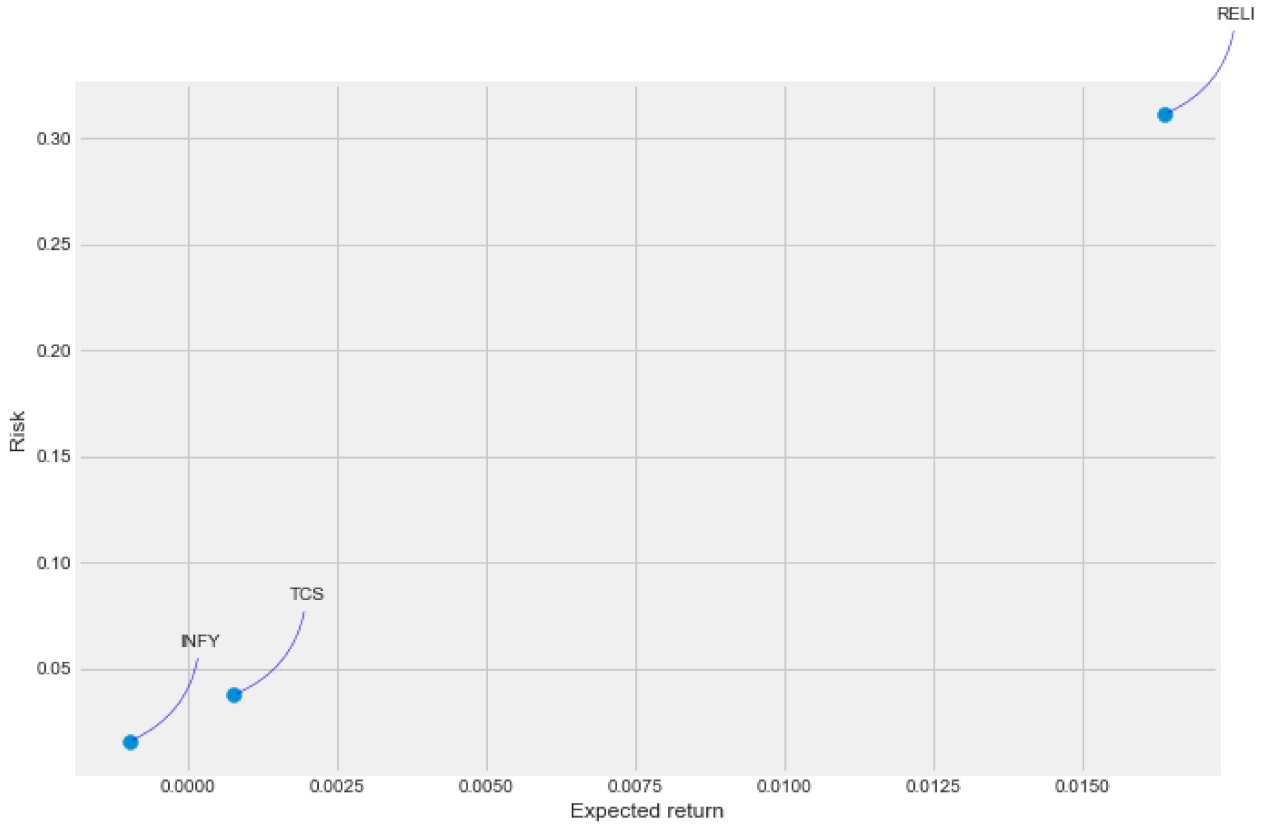
In [21]:
```
sns.heatmap(closing_df.corr(), annot=True, cmap='summer')
```

Out[21]: <AxesSubplot:xlabel='Symbols', ylabel='Symbols'>



In [22]:
```
#Expected Return vs Risk

rets = tech_rets.dropna()
```

```
area = np.pi * 20

plt.figure(figsize=(10, 7))
plt.scatter(rets.mean(), rets.std(), s=area)
plt.xlabel('Expected return')
plt.ylabel('Risk')

for label, x, y in zip(rets.columns, rets.mean(), rets.std()):
    plt.annotate(label, xy=(x, y), xytext=(50, 50), textcoords='offset points', ha='rig
                 arrowprops=dict(arrowstyle='-', color='blue', connectionstyle='arc3,ra
```



Predicting close price of TCS

In [23]:
```
# Get the stock quote
df = DataReader('TCS', data_source='stooq', start='2012-01-01', end=datetime.now())
# Show teh data
df
```

Out[23]:

Date	Open	High	Low	Close	Volume
2022-01-07	11.30	11.55	11.09	11.20	515729
2022-01-06	11.26	11.58	11.02	11.35	633517
2022-01-05	11.49	11.49	10.92	10.99	944114
2022-01-04	11.21	11.53	11.15	11.46	987435
2022-01-03	10.18	11.31	10.14	11.21	1713301
...	...	...	...	...	...

Date	Open	High	Low	Close	Volume
2013-11-07	36.80	36.80	34.12	35.31	883739
2013-11-06	35.61	36.49	35.61	35.90	549572
2013-11-05	35.11	35.50	34.82	35.35	337740
2013-11-04	36.80	36.80	34.69	35.35	1586558
2013-11-01	35.00	37.00	32.10	36.20	14669627

2061 rows × 5 columns

In [24]:
```
plt.figure(figsize=(16,6))
plt.title('Close Price History')
plt.plot(df['Close'])
plt.xlabel('Date', fontsize=18)
plt.ylabel('Close Price ', fontsize=18)
plt.show()
```

In [25]:
```
# Create a new dataframe with only the 'Close column
data = df.filter(['Close'])
# Convert the dataframe to a numpy array
dataset = data.values
# Get the number of rows to train the model on
training_data_len = int(np.ceil( len(dataset) * .95 ))

training_data_len
```

Out[25]: 1958

In [26]:
```
# Scale the data
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(dataset)

scaled_data
```

Out[26]:
```
array([[0.20640968],
       [0.2097714 ],
       [0.20170327],
       ...,
       [0.7476468 ],
       [0.7476468 ],
       [0.76669655]])
```

In [27]:
```
# Create the training data set
# Create the scaled training data set
train_data = scaled_data[0:int(training_data_len), :]
# Split the data into x_train and y_train data sets
x_train = []
y_train = []

for i in range(60, len(train_data)):
    x_train.append(train_data[i-60:i, 0])
    y_train.append(train_data[i, 0])
    if i<= 61:
        print(x_train)
        print(y_train)
        print()

# Convert the x_train and y_train to numpy arrays
x_train, y_train = np.array(x_train), np.array(y_train)

# Reshape the data
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
# x_train.shape
```

```
[array([0.20640968, 0.2097714 , 0.20170327, 0.21223667, 0.2066338 ,
       0.17906768, 0.18175706, 0.18175706, 0.18511878, 0.18601524,
       0.1792918 , 0.17817122, 0.18018826, 0.17346481, 0.1824294 ,
       0.1808606 , 0.19834155, 0.19834155, 0.19856567, 0.20416853,
       0.20753026, 0.21627073, 0.21716719, 0.2097714 , 0.20528911,
       0.2142537 , 0.20775437, 0.21604662, 0.22389063, 0.22971762,
       0.23935455, 0.24204393, 0.24562976, 0.24271627, 0.24876737,
       0.24675034, 0.24988794, 0.24809502, 0.25705961, 0.2507844 ,
       0.25414612, 0.26042134, 0.26154191, 0.26243837, 0.26064545,
       0.26109368, 0.21089198, 0.21335724, 0.20327208, 0.19968624,
       0.18960108, 0.18758404, 0.19968624, 0.1945316 , 0.19856567,
       0.19072165, 0.18399821, 0.18556701, 0.18220529, 0.17727476])]
[0.1759300761990139]

[array([0.20640968, 0.2097714 , 0.20170327, 0.21223667, 0.2066338 ,
       0.17906768, 0.18175706, 0.18175706, 0.18511878, 0.18601524,
       0.1792918 , 0.17817122, 0.18018826, 0.17346481, 0.1824294 ,
       0.1808606 , 0.19834155, 0.19834155, 0.19856567, 0.20416853,
       0.20753026, 0.21627073, 0.21716719, 0.2097714 , 0.20528911,
       0.2142537 , 0.20775437, 0.21604662, 0.22389063, 0.22971762,
       0.23935455, 0.24204393, 0.24562976, 0.24271627, 0.24876737,
       0.24675034, 0.24988794, 0.24809502, 0.25705961, 0.2507844 ,
       0.25414612, 0.26042134, 0.26154191, 0.26243837, 0.26064545,
       0.26109368, 0.21089198, 0.21335724, 0.20327208, 0.19968624,
       0.18960108, 0.18758404, 0.19968624, 0.1945316 , 0.19856567,
       0.19072165, 0.18399821, 0.18556701, 0.18220529, 0.17727476]), array([0.2097714 ,
0.20170327, 0.21223667, 0.2066338 , 0.17906768,
       0.18175706, 0.18175706, 0.18511878, 0.18601524, 0.1792918 ,
       0.17817122, 0.18018826, 0.17346481, 0.1824294 , 0.1808606 ,
       0.19834155, 0.19834155, 0.19856567, 0.20416853, 0.20753026,
       0.21627073, 0.21716719, 0.2097714 , 0.20528911, 0.2142537 ,
       0.20775437, 0.21604662, 0.22389063, 0.22971762, 0.23935455,
```

```
       0.24204393, 0.24562976, 0.24271627, 0.24876737, 0.24675034,
       0.24988794, 0.24809502, 0.25705961, 0.2507844 , 0.25414612,
       0.26042134, 0.26154191, 0.26243837, 0.26064545, 0.26109368,
       0.21089198, 0.21335724, 0.20327208, 0.19968624, 0.18960108,
       0.18758404, 0.19968624, 0.1945316 , 0.19856567, 0.19072165,
       0.18399821, 0.18556701, 0.18220529, 0.17727476, 0.17593008])]
[0.1759300761990139, 0.1768265351860153]
```

In [29]:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM

# Build the LSTM model
model = Sequential()
model.add(LSTM(128, return_sequences=True, input_shape= (x_train.shape[1], 1)))
model.add(LSTM(64, return_sequences=False))
model.add(Dense(25))
model.add(Dense(1))

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model.fit(x_train, y_train, batch_size=1, epochs=1)
```

```
WARNING:tensorflow:From C:\Users\vasan\AppData\Roaming\Python\Python36\site-packages\ten
sorflow\python\ops\init_ops.py:1251: calling VarianceScaling.__init__ (from tensorflow.p
ython.ops.init_ops) with dtype is deprecated and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the construct
or
WARNING:tensorflow:From C:\Users\vasan\AppData\Roaming\Python\Python36\site-packages\ten
sorflow\python\ops\math_grad.py:1250: add_dispatch_support.<locals>.wrapper (from tensor
flow.python.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
1898/1898 [==============================] - 123s 65ms/sample - loss: 8.8464e-04
```

Out[29]: <tensorflow.python.keras.callbacks.History at 0x22acc004400>

In [30]:

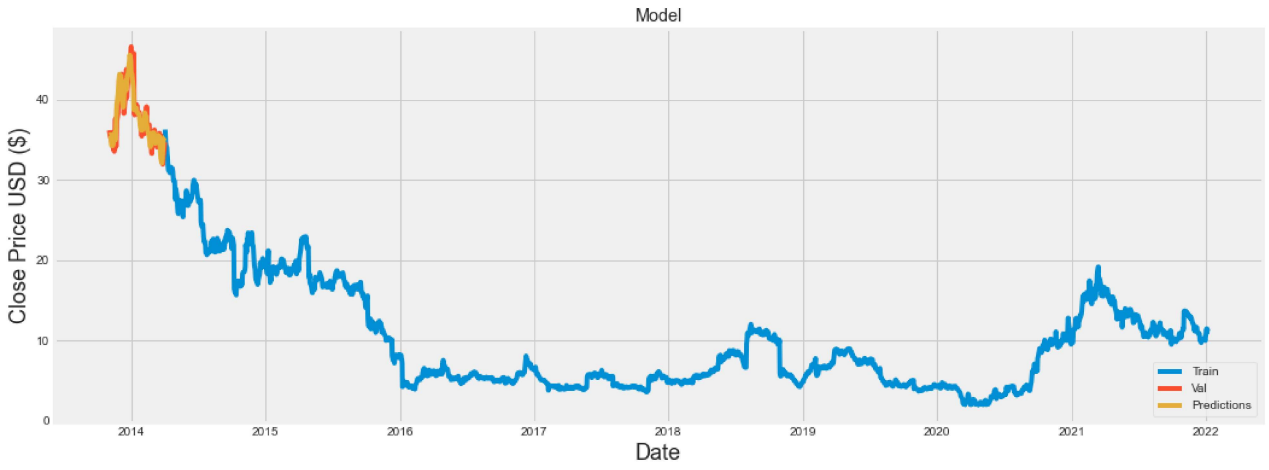
```
# Create the testing data set
# Create a new array containing scaled values from index 1543 to 2002
test_data = scaled_data[training_data_len - 60: , :]
# Create the data sets x_test and y_test
x_test = []
y_test = dataset[training_data_len:, :]
for i in range(60, len(test_data)):
    x_test.append(test_data[i-60:i, 0])

# Convert the data to a numpy array
x_test = np.array(x_test)

# Reshape the data
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1 ))

# Get the models predicted price values
predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)

# Get the root mean squared error (RMSE)
```

```
rmse = np.sqrt(np.mean(((predictions - y_test) ** 2)))
rmse
```

Out[30]: 1.5280928514390737

In [31]:
```
# Plot the data
train = data[:training_data_len]
valid = data[training_data_len:]
valid['Predictions'] = predictions
# Visualize the data
plt.figure(figsize=(16,6))
plt.title('Model')
plt.xlabel('Date', fontsize=18)
plt.ylabel('Close Price USD ($)', fontsize=18)
plt.plot(train['Close'])
plt.plot(valid[['Close', 'Predictions']])
plt.legend(['Train', 'Val', 'Predictions'], loc='lower right')
plt.show()
```

```
C:\Users\vasan\anaconda3\lib\site-packages\ipykernel_launcher.py:4: SettingWithCopyWarni
ng:
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
  after removing the cwd from sys.path.
```



In [32]:
```
# Show the valid and predicted prices
valid
```

Out[32]:

Date	Close	Predictions
2014-04-01	34.72	35.366158
2014-03-31	33.95	35.332256
2014-03-28	32.68	34.774765
2014-03-27	31.97	33.787048
2014-03-26	32.08	32.758297

Date	Close	Predictions
...	...	...
2013-11-07	35.31	34.576752
2013-11-06	35.90	34.987907
2013-11-05	35.35	35.489735
2013-11-04	35.35	35.636177
2013-11-01	36.20	35.605766

103 rows × 2 columns

In [ ]: