Data from Yahoo Finance ,Yahoo Finance for GE was used for getting stock history prices over some period of time

In [69]:

```
import sklearn
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns

%matplotlib inline
```

In [70]:

```
data = pd.read_csv('GE.csv')
data.head(10)
```

Out[70]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	1962-01-02	0.751202	0.763722	0.743690	0.748698	0.001785	2156500
1	1962-01-03	0.744942	0.744942	0.738682	0.741186	0.001767	1477600
2	1962-01-04	0.741186	0.747446	0.726162	0.732422	0.001746	1837000
3	1962-01-05	0.732422	0.733674	0.701122	0.713642	0.001701	2725600
4	1962-01-08	0.713642	0.713642	0.691106	0.712390	0.001698	3095000
5	1962-01-09	0.712390	0.722406	0.707382	0.716146	0.001707	2326200
6	1962-01-10	0.716146	0.723658	0.713642	0.719902	0.001716	1956800
7	1962-01-11	0.719902	0.721154	0.707382	0.721154	0.001719	1627300
8	1962-01-12	0.721154	0.722406	0.708634	0.711138	0.001695	1687200
9	1962-01-15	0.712390	0.721154	0.712390	0.718650	0.001713	2116600

In [71]:

```
data.shape
```

Out[71]:

```
(14587, 7)
```

In [72]:

```
#Converting date column to datetime
data.loc[:,'Date'] = pd.to_datetime(data['Date'],format='%Y-%m-%d')
```

In [73]:

```
#change all columns headings to be lower case, and remove spacing
data.columns = [str(x).lower().replace(' ','_') for x in data.columns]
```

In [74]:

```
#Get month of each sample
#data['month'] = data['date'].dt.month
```

In [75]:

```
#Sort by datetime
```

```
#data.sort_values(by='date',inplace=True, ascending=True)
```

In [76]:

```
data.corr()
```

Out[76]:

	open	high	low	close	adj_close	volume
open	1.000000	0.999887	0.999869	0.999778	0.955874	0.281335
high	0.999887	1.000000	0.999812	0.999895	0.955286	0.283466
low	0.999869	0.999812	1.000000	0.999877	0.956642	0.278006
close	0.999778	0.999895	0.999877	1.000000	0.955971	0.280554
adj_close	0.955874	0.955286	0.956642	0.955971	1.000000	0.376047
volume	0.281335	0.283466	0.278006	0.280554	0.376047	1.000000

In [77]:

```
data.info() #data doesn't have any missing values
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14587 entries, 0 to 14586
Data columns (total 7 columns):
date          14587 non-null datetime64[ns]
open          14587 non-null float64
high          14587 non-null float64
low           14587 non-null float64
close         14587 non-null float64
adj_close     14587 non-null float64
volume        14587 non-null int64
dtypes: datetime64[ns](1), float64(5), int64(1)
memory usage: 797.9 KB
```

In [78]:

```
data.Date = pd.to_datetime(data['date'])
```

```
C:\Users\ravikiran\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: UserWarning: Pandas
doesn't allow columns to be created via a new attribute name - see
https://pandas.pydata.org/pandas-docs/stable/indexing.html#attribute-access
  """Entry point for launching an IPython kernel.
```

In [79]:

```
data = data.set_index('date')
```

In [80]:

```
data.head()
```

Out[80]:

	open	high	low	close	adj_close	volume
date						
1962-01-02	0.751202	0.763722	0.743690	0.748698	0.001785	2156500
1962-01-03	0.744942	0.744942	0.738682	0.741186	0.001767	1477600
1962-01-04	0.741186	0.747446	0.726162	0.732422	0.001746	1837000
1962-01-05	0.732422	0.733674	0.701122	0.713642	0.001701	2725600
1962-01-08	0.713642	0.713642	0.691106	0.712390	0.001698	3095000

In [15]:

```
from pylab import rcParams
from sklearn.metrics import mean_squared_error
from tqdm import tqdm_notebook
from xgboost import XGBRegressor
```

In [81]:

```
data.columns
```

Out[81]:

```
Index(['open', 'high', 'low', 'close', 'adj_close', 'volume'], dtype='object')
```

In [82]:

```
data.index
```

Out[82]:

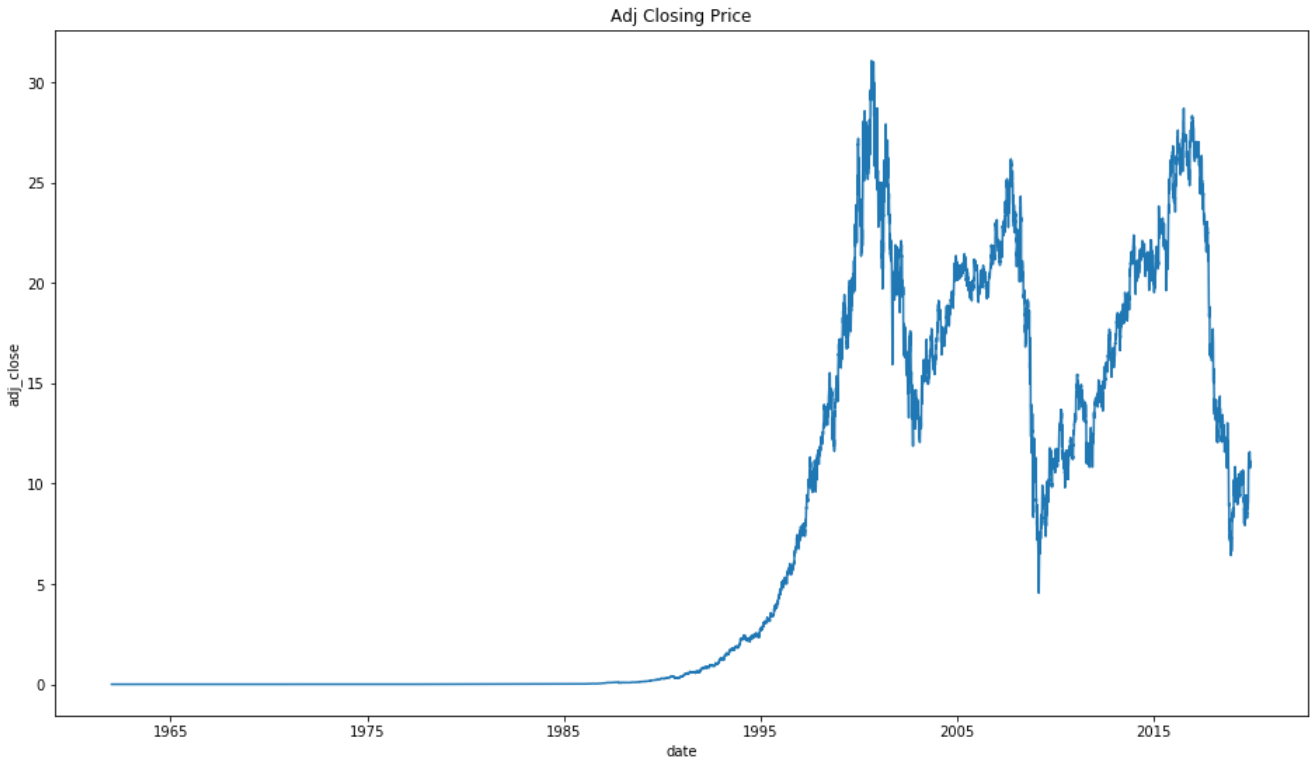
```
DatetimeIndex(['1962-01-02', '1962-01-03', '1962-01-04', '1962-01-05',
               '1962-01-08', '1962-01-09', '1962-01-10', '1962-01-11',
               '1962-01-12', '1962-01-15',
               ...
               '2019-11-27', '2019-11-29', '2019-12-02', '2019-12-03',
               '2019-12-04', '2019-12-05', '2019-12-06', '2019-12-09',
               '2019-12-10', '2019-12-11'],
              dtype='datetime64[ns]', name='date', length=14587, freq=None)
```

In [83]:

```
plt.figure(figsize=(16,9))
sns.lineplot(data.index,data.adj_close)
plt.title('Adj Closing Price')
```

Out[83]:

```
Text(0.5,1,'Adj Closing Price')
```



In [84]:

```
plt.figure(figsize=(16,9))
sns.lineplot(data.index.values[10000:],data.adj_close.values[10000:])
plt.title('Adj Closing Price')
```

Out[84]:

```
Text(0.5,1,'Adj Closing Price')
```

In [85]:

```
#Supervised problem with moving window method
#for one month data considering last 30 days as features and 31 day as target
window_size = 30

#get indices of access for the data
num_samples = len(data) - window_size
indices = np.arange(num_samples).astype(np.int)[:,None] + np.arange(window_size + 1).astype(np.int)
```

In [86]:

```
indices
```

Out[86]:

```
array([[    0,     1,     2, ...,    28,    29,    30],
       [    1,     2,     3, ...,    29,    30,    31],
       [    2,     3,     4, ...,    30,    31,    32],
       ...,
       [14554, 14555, 14556, ..., 14582, 14583, 14584],
       [14555, 14556, 14557, ..., 14583, 14584, 14585],
       [14556, 14557, 14558, ..., 14584, 14585, 14586]])
```

In [87]:

```
data_movingwindow = data['adj_close'].values[indices]
X = data_movingwindow[:,:-1]
y = data_movingwindow[:,-1]
```

In [88]:

```
X.shape, y.shape
```

Out[88]:

```
((14557, 30), (14557,))
```

Feature Engineering

In [25]:

```
#Get difference between high and low of each day
#data['range_hl'] = data['high'] - data['low']
#data.drop(['high','low'], axis=1, inplace=True)
```

In [26]:

```
#Get difference between open and close of each day
#data['range_oc'] = data['open'] - data['close']
#data.drop(['open','close'], axis=1, inplace=True)
```

In [89]:

```
data.head()
```

Out[89]:

	open	high	low	close	adj_close	volume
date						
1962-01-02	0.751202	0.763722	0.743690	0.748698	0.001785	2156500
1962-01-03	0.744942	0.744942	0.738682	0.741186	0.001767	1477600
1962-01-04	0.741186	0.747446	0.726162	0.732422	0.001746	1837000
1962-01-05	0.732422	0.733674	0.701122	0.713642	0.001701	2725600
1962-01-08	0.713642	0.713642	0.691106	0.712390	0.001698	3095000

In [28]:

```
#Add a column 'order_day' to indicate the order of the rows by date
#data['order_day'] = [x for x in list(range(len(data)))]
```

In [30]:

```
from sklearn.preprocessing import MinMaxScaler
#scaler = MinMaxScaler(feature_range=(0, 1))
#data_scaler = scaler.fit_transform(data)
```

In [31]:

```
#label=data['adj_close']
#data=data.drop('date', axis=1)
#data=data.drop('adj_close', axis=1)
```

In [90]:

```
#time series data so splitting data into 85% for train and 15% for test
from sklearn.model_selection import train_test_split

#splitting data into train and test where 80% data used for train and 20% for test model
X_train,X_test, y_train, y_test = train_test_split(X, y, test_size=0.15)
# split the train data into train and cross validation by maintaining same distribution of output
varaible 'label'
#x_train, x_cv, y_train, y_cv = train_test_split(X_train, Y_train, test_size=0.2)
```

In [91]:

```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[91]:

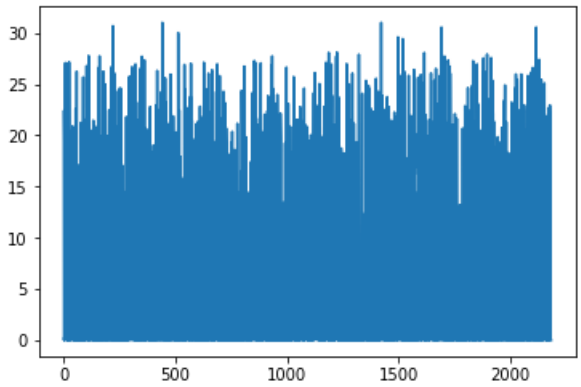
```
((12373, 30), (2184, 30), (12373,), (2184,))
```

In [92]:

```
plt.plot(y_test)
```

Out[92]:

```
[<matplotlib.lines.Line2D at 0xae025d1550>]
```



Model 1 : Lasso Regression

In [93]:

```
from sklearn.linear_model import Ridge, Lasso, LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error as mae
```

In [94]:

```
Lasso_model = Lasso(alpha=1.0)
Lasso_model.fit(X_train, y_train)
```

```
C:\Users\ravikiran\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_descent.py:491: Con
vergenceWarning: Objective did not converge. You might want to increase the number of iterations.
Fitting data with very small alpha may cause precision problems.
  ConvergenceWarning)
```

Out[94]:

```
Lasso(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=1000,
   normalize=False, positive=False, precompute=False, random_state=None,
   selection='cyclic', tol=0.0001, warm_start=False)
```

In [95]:

```
y_pred_train = Lasso_model.predict(X_train)
y_pred_test = Lasso_model.predict(X_test)
```

In [96]:

```
print('Mean Absolute Error for train data', mae(y_train, y_pred_train))
print('Mean Absolute Error for train data', mae(y_test, y_pred_test))
```

```
Mean Absolute Error for train data 0.16842095530685663
Mean Absolute Error for train data 0.16943909671924634
```

In [98]:

```
from sklearn.metrics import r2_score
```

```
r2_score_etr = r2_score(y_train, y_pred_train)
print("r2 score:",r2_score_etr)
```

```
r2 score: 0.9991565309201584
```

In [99]:

```
r2_score_etr = r2_score(y_test, y_pred_test)
print("r2 score:",r2_score_etr)
```

```
r2 score: 0.9991989710771192
```

In [26]:

```
import scipy
for i in scipy.stats.uniform.rvs(loc=0, scale=30,size=10, random_state=3):
    Lasso_model = Lasso(alpha=i)
    Lasso_model.fit(X_train,y_train)
    print('alpha', i, 'Train Error', mae(y_train, y_pred_train), 'Test Error', mae(y_test, y_pred_t
est))
```

```
C:\Users\ravikiran\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_descent.py:491: Con
vergenceWarning: Objective did not converge. You might want to increase the number of iterations.
Fitting data with very small alpha may cause precision problems.
  ConvergenceWarning)
```

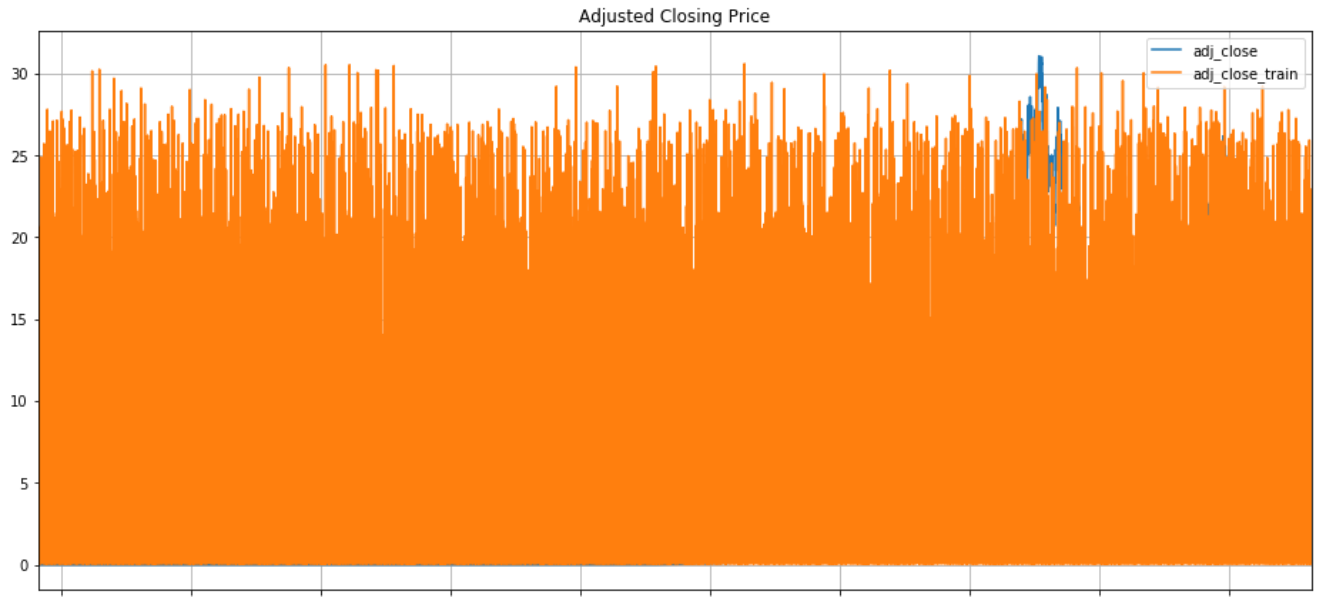
```
alpha 16.523937077237264 Train Error 0.16938533528159092 Test Error 0.16704483253088898
alpha 21.244434678543143 Train Error 0.16938533528159092 Test Error 0.16704483253088898
alpha 8.72714216738833 Train Error 0.16938533528159092 Test Error 0.16704483253088898
alpha 15.32482815592989 Train Error 0.16938533528159092 Test Error 0.16704483253088898
alpha 26.78840863042964 Train Error 0.16938533528159092 Test Error 0.16704483253088898
alpha 26.88879266800314 Train Error 0.16938533528159092 Test Error 0.16704483253088898
alpha 3.7675593139150876 Train Error 0.16938533528159092 Test Error 0.16704483253088898
alpha 6.217286344145602 Train Error 0.16938533528159092 Test Error 0.16704483253088898
alpha 1.5440160990248963 Train Error 0.16938533528159092 Test Error 0.16704483253088898
alpha 13.224295309519094 Train Error 0.16938533528159092 Test Error 0.16704483253088898
```

In [111]:

```
xgb_df = data.copy()
xgb_df.drop(['open','high','low','close','volume'], axis=1, inplace=True)
xgb_df = xgb_df.iloc[window_size:12373]
xgb_df['adj_close_train'] = y_pred_train[:-window_size]
xgb_df.plot(kind='line', figsize=(16,8),title='Adjusted Closing Price',grid=True)
```

Out[111]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xae02862400>
```

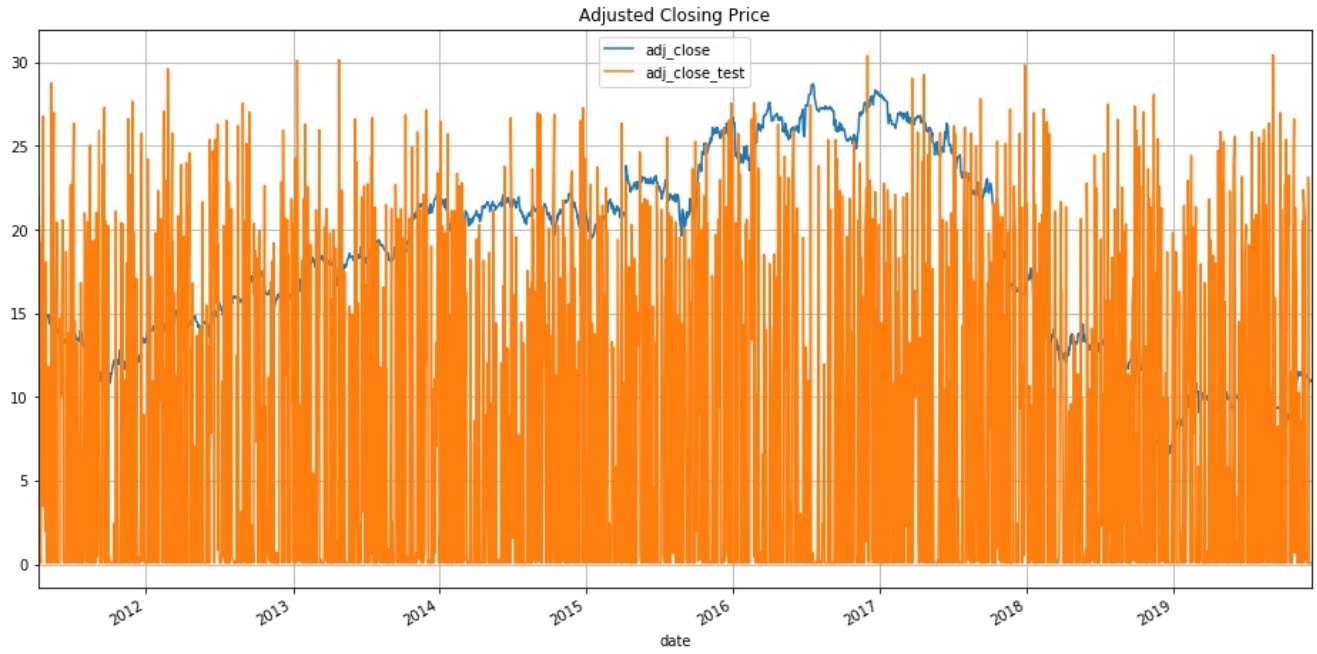


In [112]:

```
#same for test
lasso_df = data.copy()
lasso_df.drop(['open','high','low','close','volume'], axis=1, inplace=True)
lasso_df = lasso_df.iloc[12373+window_size:] #past 32 days we don't know yet
lasso_df['adj_close_test'] = y_pred_test
lasso_df.plot(kind='line', figsize=(16,8),title='Adjusted Closing Price',grid=True)
```

Out[112]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xae32b27198>
```



Model 2 : XGBoost Regression

In [114]:

```
import xgboost as xgb
```

In [116]:

```
xgb.XGBRegressor(max_depth=3,learning_rate=0.05,n_estimators=100,verbosity=1,silent=None,objective
='reg:squarederror',
                 booster='gbtree',n_jobs=-1,nthread=None,gamma=0,min_child_weight=10,max_delta_step=
0,subsample=1,

colsample_bytree=0.9,colsample_bylevel=1,colsample_bynode=1,reg_alpha=0.5,reg_lambda=1,scale_pos_we
ight=1,
                 base_score=0.5,random_state=0,seed=None,missing=None,importance_type='gain',eval_me
tric='mae')
model.fit(X_train, np.log(y_train))
```

```
[18:43:30] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
```

Out[116]:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=1, gamma=0,
       importance_type='gain', learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
       n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=100,
       silent=None, subsample=1, verbosity=1)
```

silent=None, subsample=1, verbosity=1)

In [117]:

```
# Do prediction on train set and test set
y_pred_train = np.exp(model.predict(X_train))
y_pred_test = np.exp(model.predict(X_test))
```

In [118]:

```
print('Mean Absolute Error for train data', mae(y_train, y_pred_train))
print('Mean Absolute Error for train data', mae(y_test, y_pred_test))
```

```
Mean Absolute Error for train data 0.10049369207852707
Mean Absolute Error for train data 0.11107803024274926
```

In [119]:

```
r2_score_etr = r2_score(y_train, y_pred_train)
print("r2 score:",r2_score_etr)
```

```
r2 score: 0.9994952423130017
```

In [120]:

```
r2_score_etr = r2_score(y_test, y_pred_test)
print("r2 score:",r2_score_etr)
```

```
r2 score: 0.9993765926601546
```

In [121]:

```
xgb_df = data.copy()
xgb_df.drop(['open','high','low','close','volume'], axis=1, inplace=True)
xgb_df = xgb_df.iloc[window_size:12373]
xgb_df['adj_close_train'] = y_pred_train[:-window_size]
xgb_df.plot(kind='line', figsize=(16,8),title='Adjusted Closing Price',grid=True)
```

Out[121]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xae32baf9e8>
```

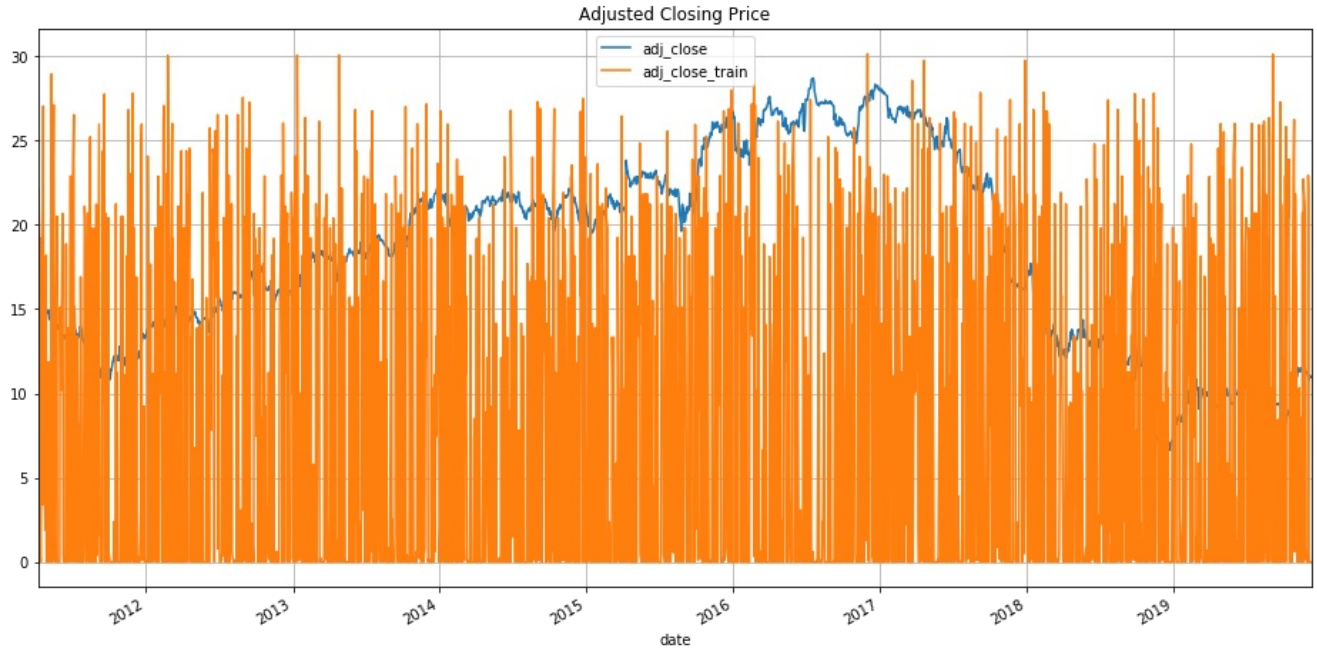
In [124]:

```
xgb_df = data.copy()
```

```
xgb_df.drop(['open','high','low','close','volume'], axis=1, inplace=True)
xgb_df = xgb_df.iloc[12373+window_size:]
xgb_df['adj_close_train'] = y_pred_test
xgb_df.plot(kind='line', figsize=(16,8),title='Adjusted Closing Price',grid=True)
```

Out[124]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xae32fb0710>
```



In [126]:

```
from prettytable import PrettyTable
pretty_table = PrettyTable()
pretty_table.field_names = ["Model", "MAE-Train","MAE-Test", "R² score-Train", "R² score-Test"]
pretty_table.add_row(["Lasso Regressor",0.168,0.169,0.999,0.999])
pretty_table.add_row(["\n","\n","\n","\n","\n"])

pretty_table.add_row(["XGBoost Regressor",0.1004,0.1110,0.9994,0.9993])
pretty_table.add_row(["\n","\n","\n","\n","\n"])

print(pretty_table)
```

```
+-------------------+-----------+----------+----------------+---------------+
|       Model       | MAE-Train | MAE-Test | R² score-Train | R² score-Test |
+-------------------+-----------+----------+----------------+---------------+
|  Lasso Regressor  |   0.168   |  0.169   |     0.999      |     0.999     |
|                   |           |          |                |               |
|                   |           |          |                |               |
| XGBoost Regressor |   0.1004  |  0.111   |     0.9994     |     0.9993    |
|                   |           |          |                |               |
|                   |           |          |                |               |
+-------------------+-----------+----------+----------------+---------------+
```

Conclusion:

As both models Lasso regression and Xgboost regression performed well on this dataset with less mean squared error and with high r2-score on test and train data.