In [1]:

```
#Binary Classification - Given a stock and it’s data, you have to predict whether it wi
llclose lower than it opened (red) or higher than it opened (green)
```

In [2]:

```
#Import the libraries
import pandas as pd
import datetime as dt
import seaborn as sns
from matplotlib import pyplot as plt
plt.style.use('ggplot')
from datetime import datetime
import numpy as np
%matplotlib inline
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
```

In [3]:

```
cd C:\Users\harsha.teja\Desktop\myg\congitensor
```

```
C:\Users\harsha.teja\Desktop\myg\congitensor
```

In [4]:

```
df=pd.read_csv("cs-1.csv")
```

In [5]:

```
 df['date'] = df['date'].map(lambda t: datetime.strptime(str(t), '%Y-%m-%d'))
```

In [6]:

```
df.isnull().sum()
```

Out[6]:

```
date       0
open      11
high       8
low        8
close      0
volume     0
Name       0
dtype: int64
```

In [7]:

```
df = df.dropna()
```

In [8]:

```
#Binary Classification - Given a stock and it’s data, you have to predict whether it wi
llclose lower than it opened (red) or higher than it opened (green) [Continued on the n
extpage]""""""
```

In [9]:

```
#Manipulate the data set
#Create the target column
df['Price_Up_down'] = np.where(df['close'] > df['open'], 1, 0)
#Remove the date column
remove_list = ['date','Name']
df = df.drop(columns=remove_list)
#Show the data
df
```

Out[9]:

	open	high	low	close	volume	Price_Up_down
0	15.07	15.12	14.63	14.75	8407500	0
1	14.89	15.01	14.26	14.46	8882000	0
2	14.45	14.51	14.10	14.27	8126000	0
3	14.30	14.94	14.25	14.66	10259500	1
4	14.94	14.96	13.16	13.99	31879900	0
...	...	...	...	...	...	...
619035	76.84	78.27	76.69	77.82	2982259	1
619036	77.53	78.12	76.73	76.78	2595187	0
619037	76.64	76.92	73.18	73.83	2962031	0
619038	72.74	74.56	72.13	73.27	4924323	1
619039	72.70	75.00	72.69	73.86	4534912	1

619029 rows × 6 columns

In [10]:

```
correlation =df.corr()
```

In [11]:

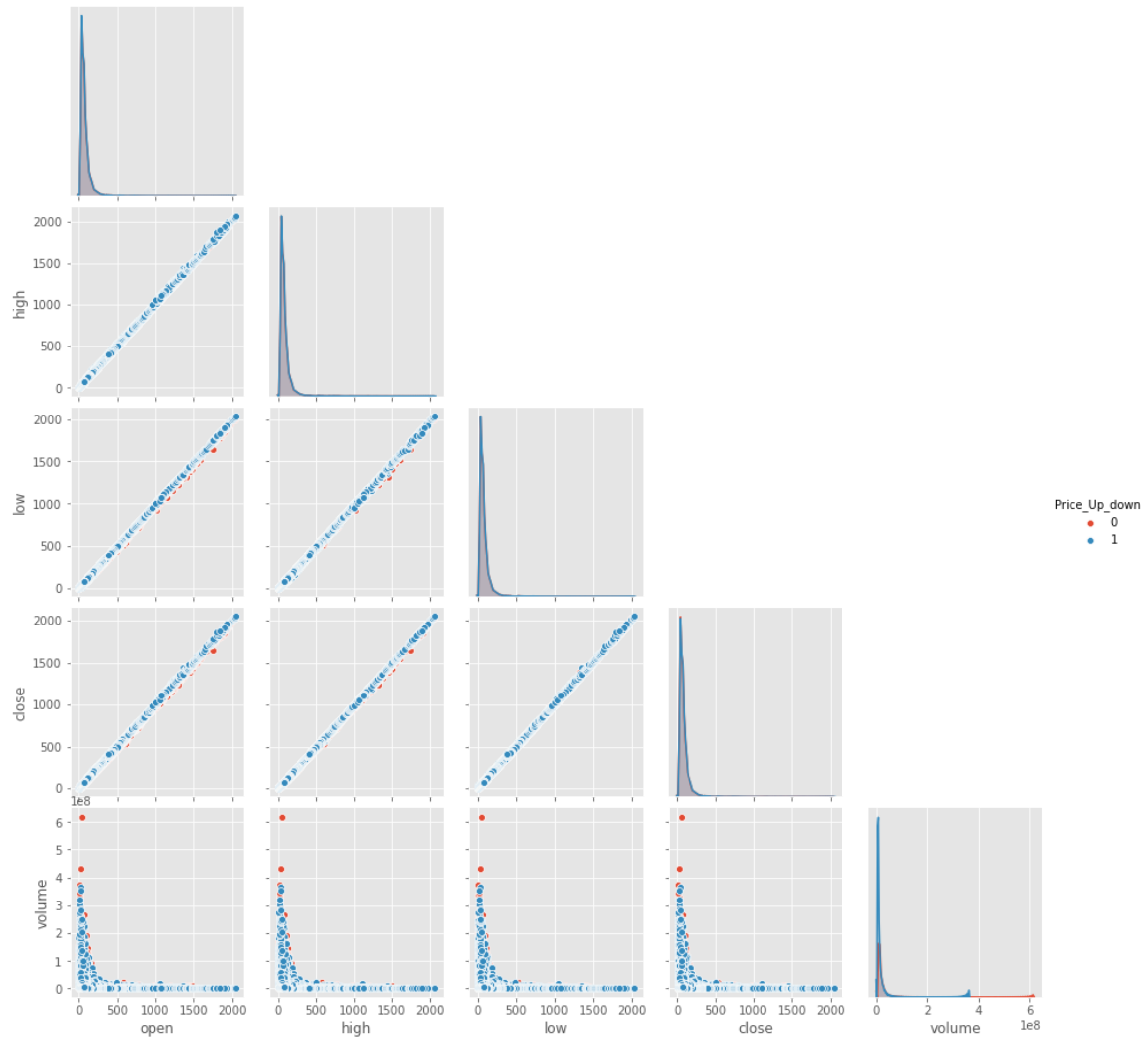
```
cols= ['#00876c','#85b96f','#f7e382','#f19452','#d43d51']
plt.figure(figsize=(15,9))
sns.heatmap(correlation,cmap=cols ,annot=True, linewidths=0.5)
plt.show()
```

In [12]:

```
#pair plot
sns.pairplot(df, hue='Price_Up_down',corner=True)
```

Out[12]:

```
<seaborn.axisgrid.PairGrid at 0x205ef9e0dc0>
```



In [12]:

```
#distrubution of data
def disbution_of_data(feture):
    plt.figure(figsize=(15,9))
    sns.distplot(feture,color='green',bins=100)
```

In [13]:

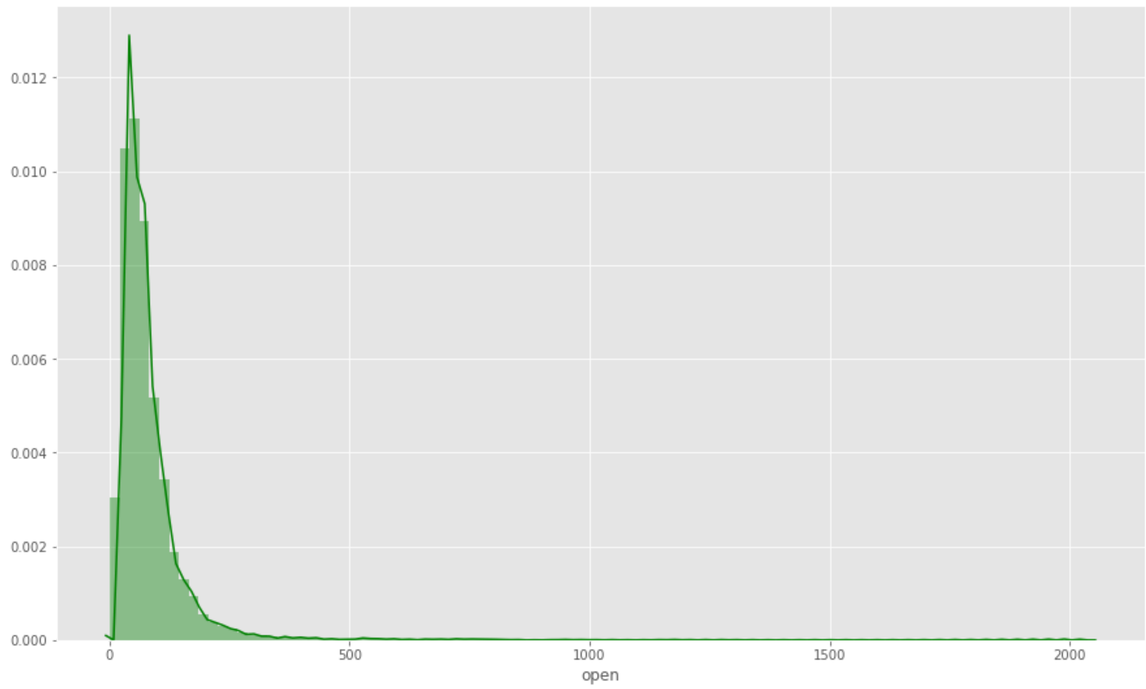
```
df.columns
```

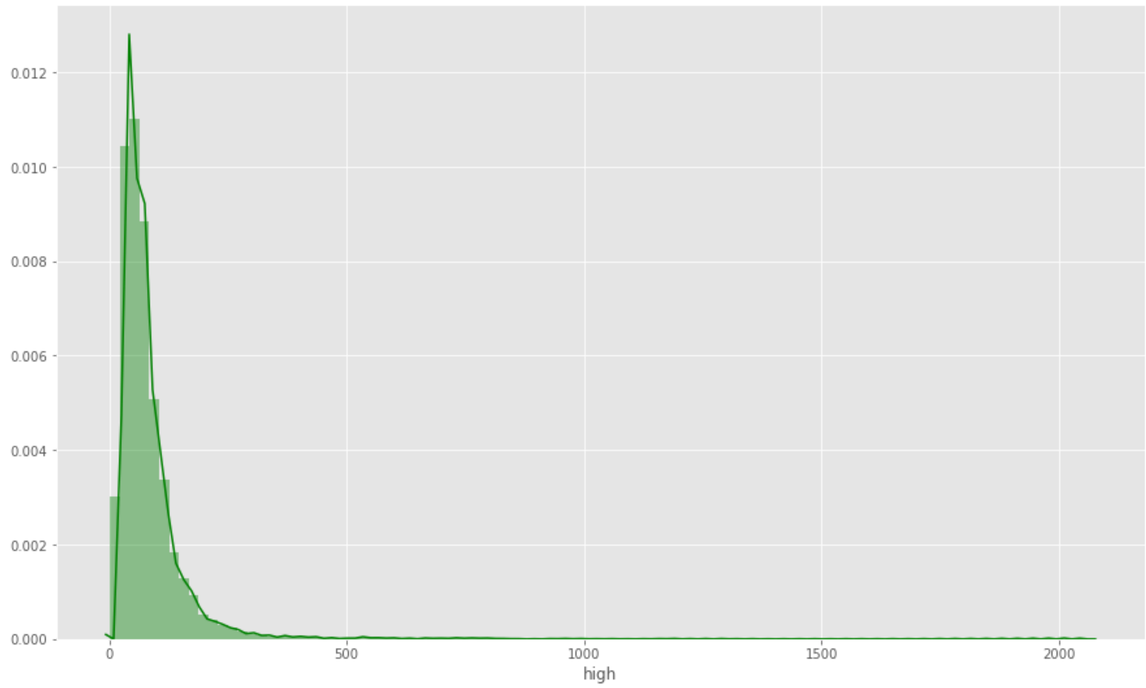
Out[13]:

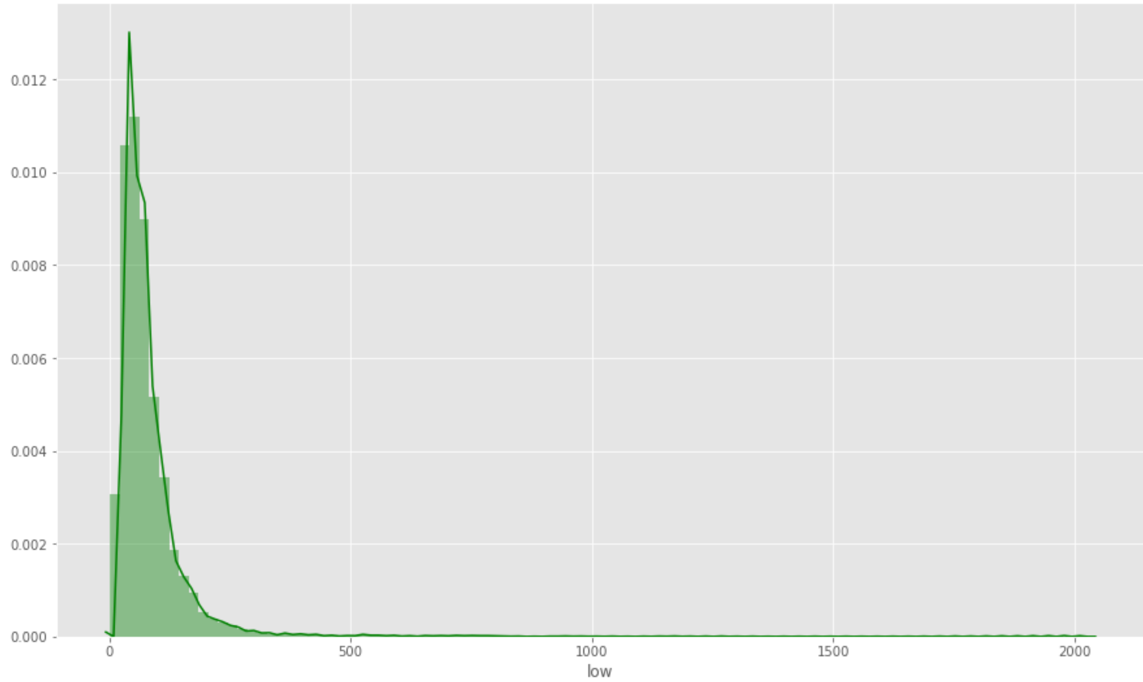
```
Index(['open', 'high', 'low', 'close', 'volume', 'Price_Up_down'], dtype
='object')
```

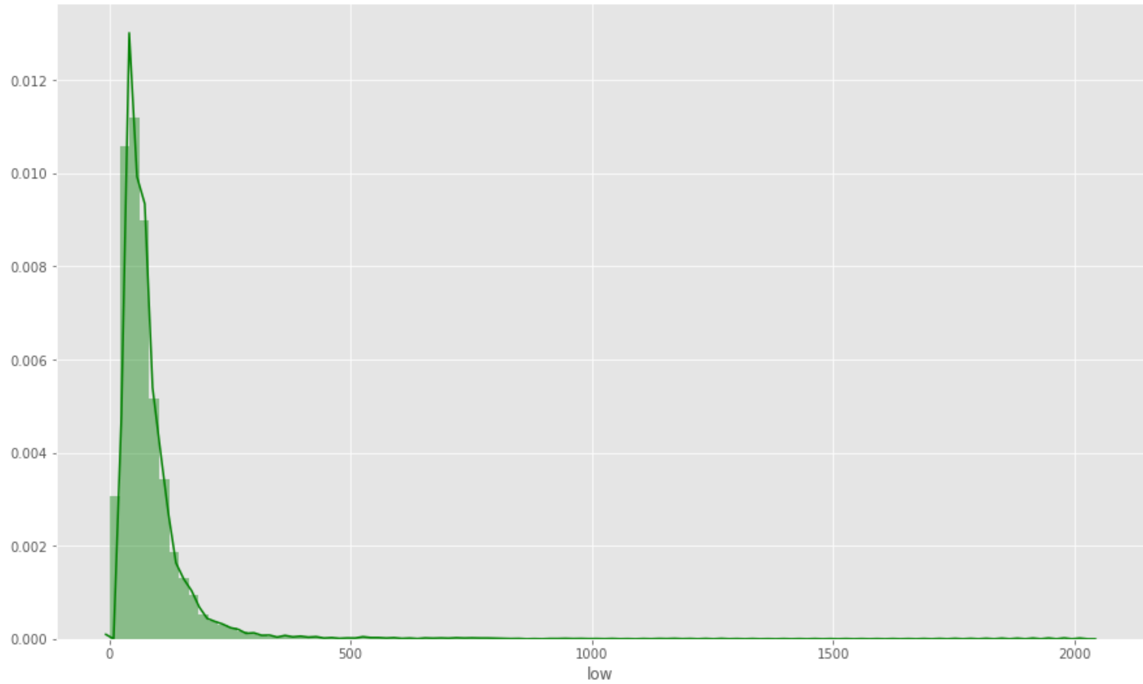
In [14]:

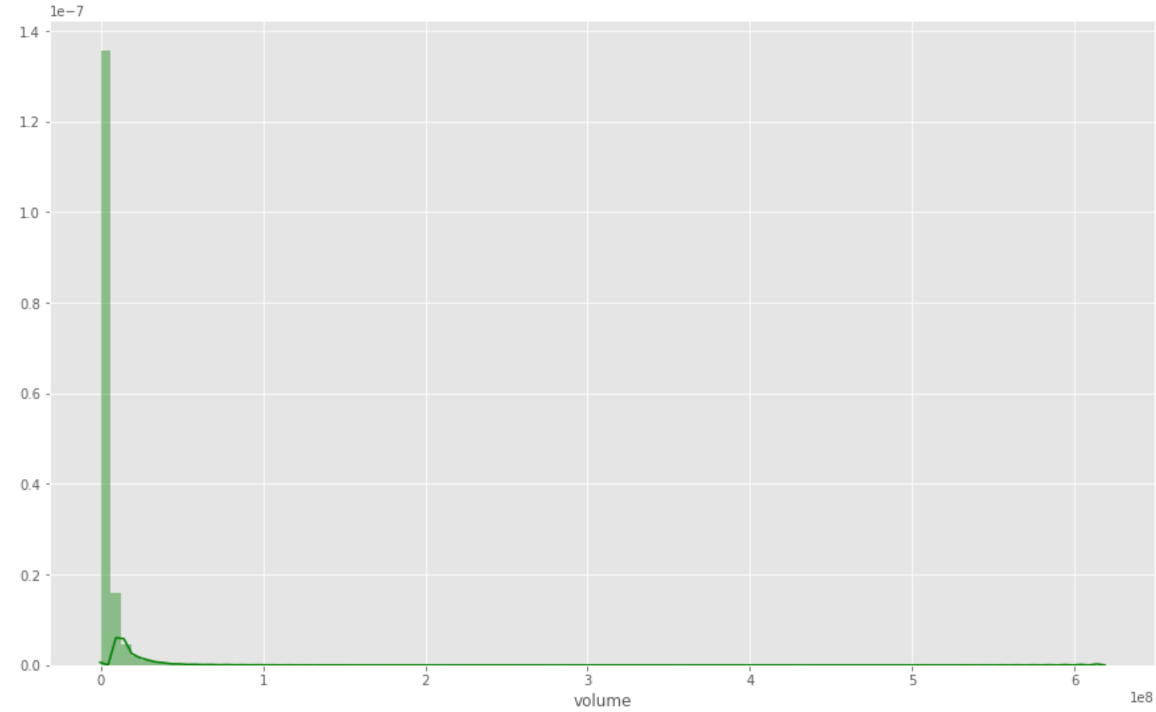
```
disbution_of_data(df['open'])
disbution_of_data(df['high'])
disbution_of_data(df['low'])
disbution_of_data(df['low'])
disbution_of_data(df['volume'])
disbution_of_data(df['Price_Up_down'])
```

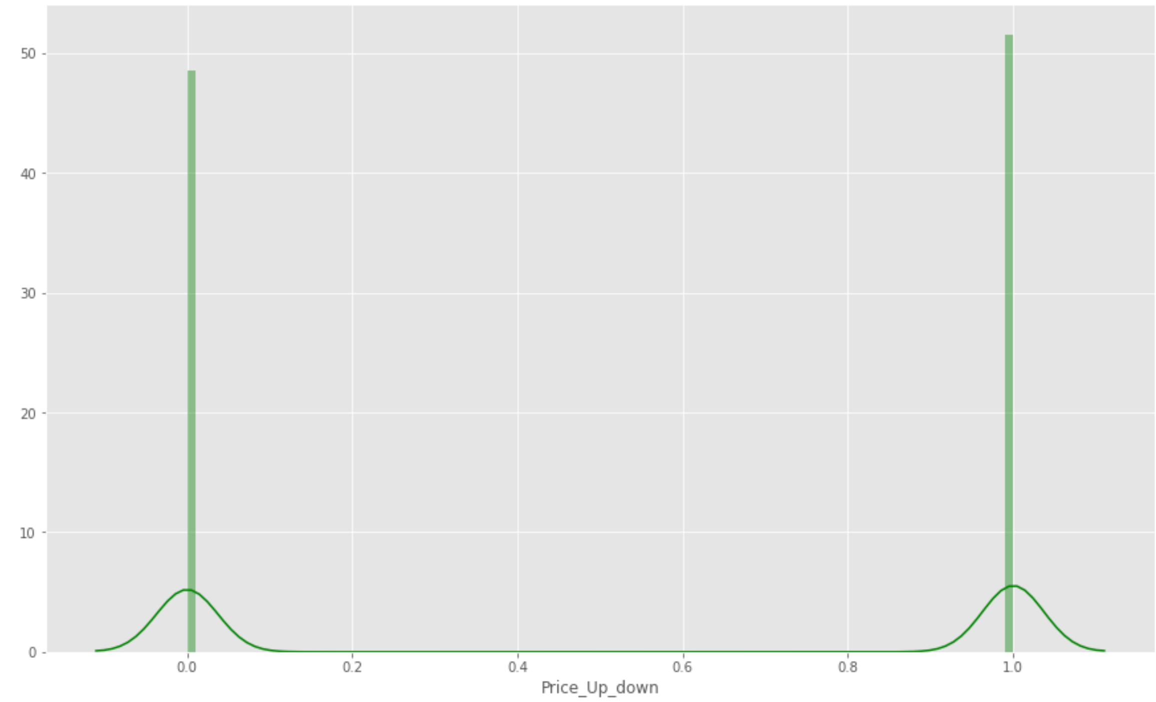

0.012
0.010
0.008
0.006
0.004
0.002
0.000
0
500
1000
1500
2000
open


0.012
0.010
0.008
0.006
0.004
0.002
0.000
0
500
1000
1500
2000
high


0.012
0.010
0.008
0.006
0.004
0.002
0.000
0
500
1000
1500
2000
low

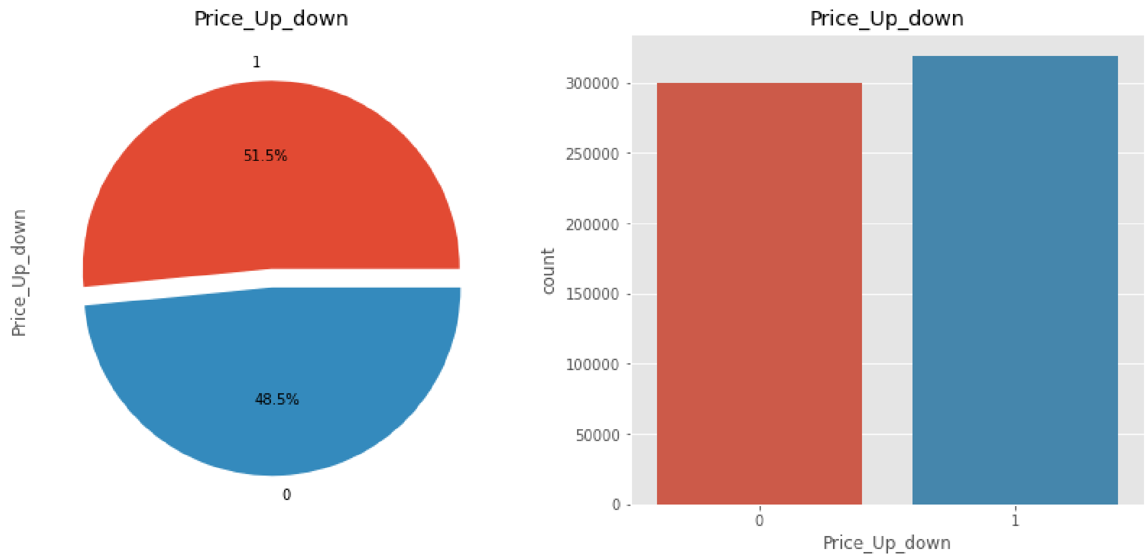

0.012
0.010
0.008
0.006
0.004
0.002
0.000
0
500
1000
1500
2000
low


1e−7
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
6
volume
1e8


50
40
30
20
10
0
0.0
0.2
0.4
0.6
0.8
1.0
Price_Up_down

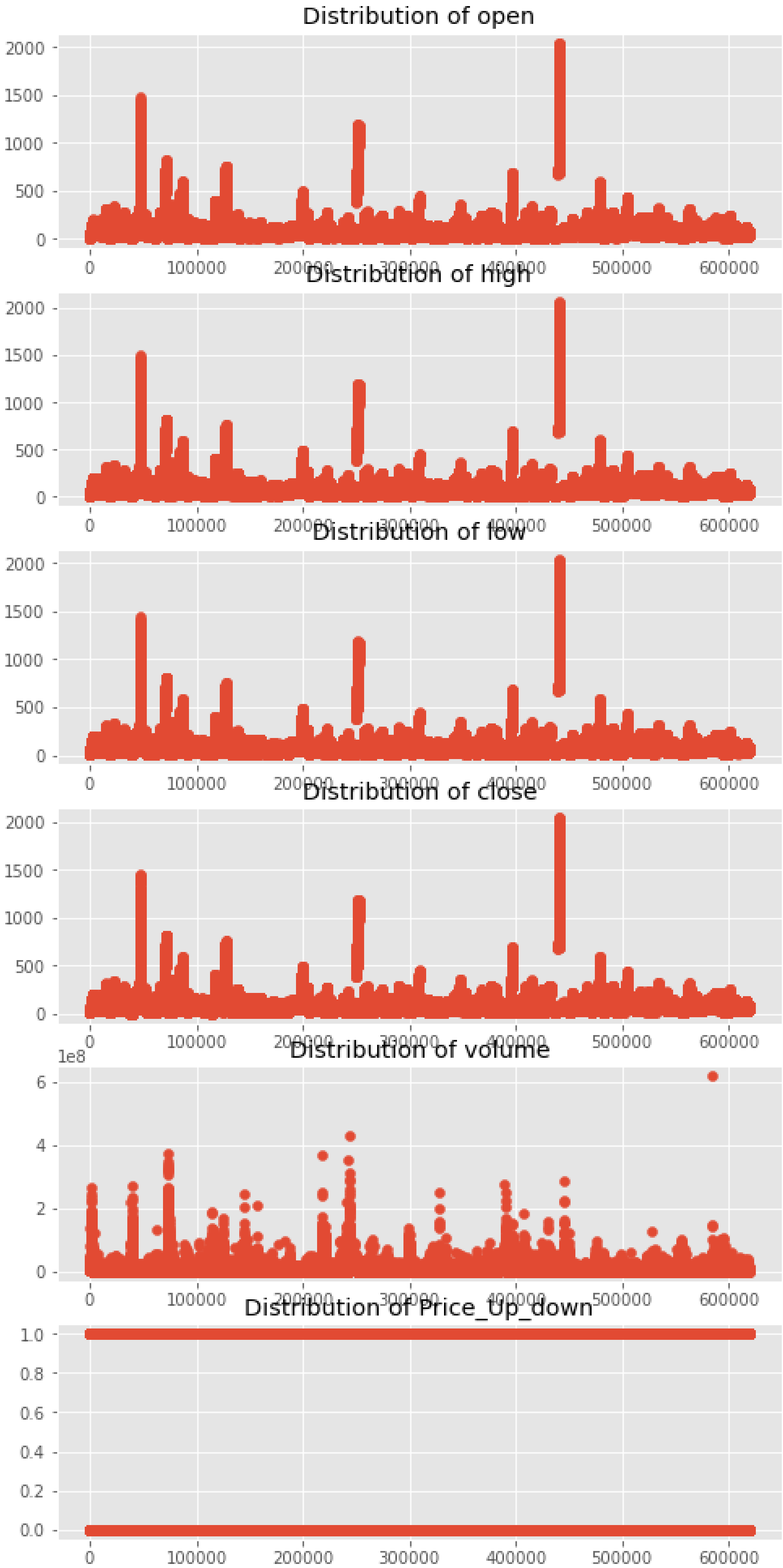
In [15]:

```
f,ax=plt.subplots(1,2,figsize=(14,6))
df['Price_Up_down'].value_counts().plot.pie(explode=[0,0.1],autopct='%1.1f%%',ax=ax[0],
shadow=False)
ax[0].set_title('Price_Up_down')
ax[0].set_ylabel('Price_Up_down')
sns.countplot('Price_Up_down',data=df,ax=ax[1])
ax[1].set_title('Price_Up_down')
plt.show()
```



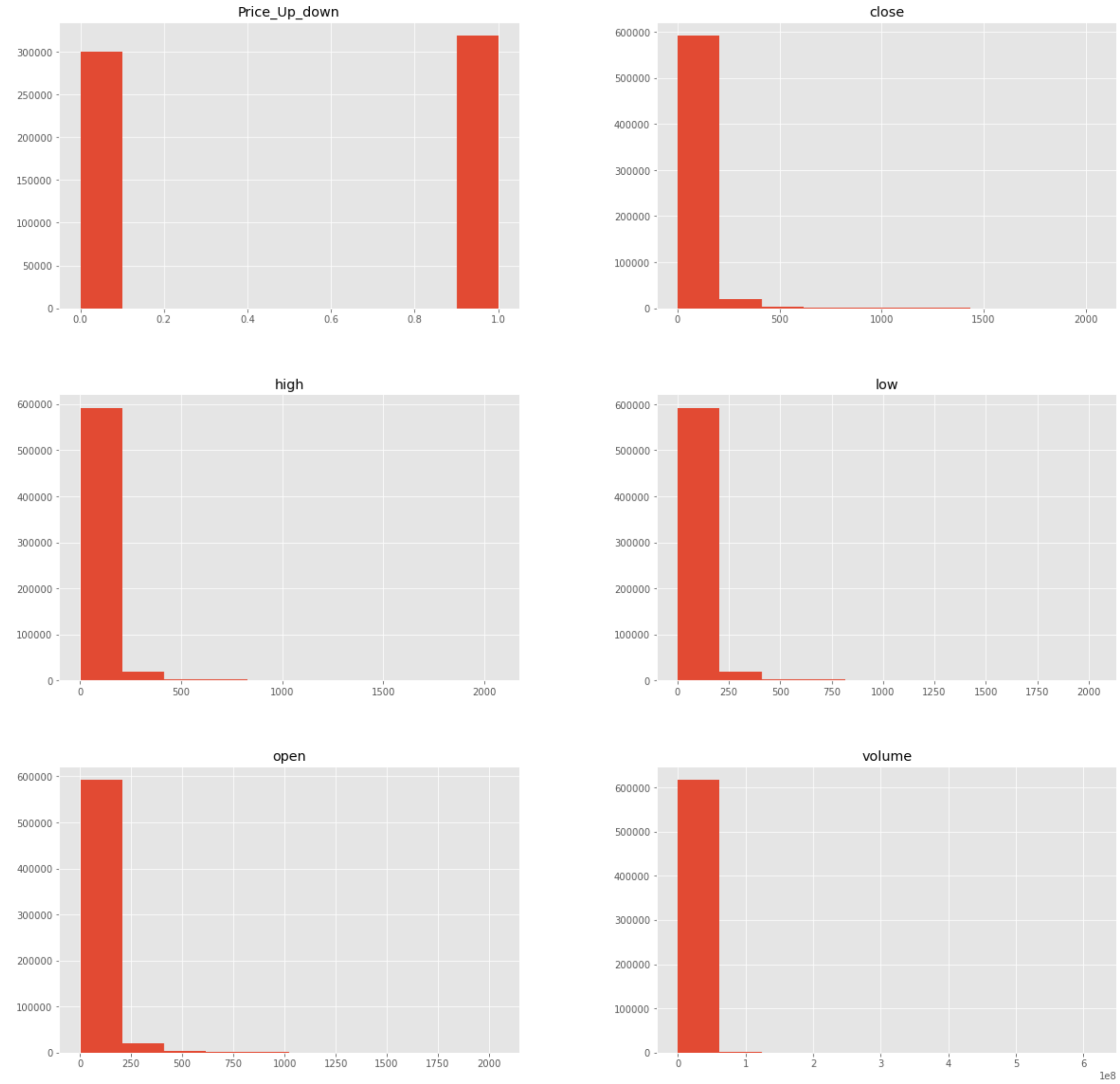
In [16]:

```
cols_to_use = ['open', 'high', 'low', 'close', 'volume', 'Price_Up_down']
fig = plt.figure(figsize=(8, 20))
plot_count = 0
for col in cols_to_use:
    plot_count += 1
    plt.subplot(7, 1, plot_count)
    plt.scatter(range(df.shape[0]), df[col].values)
    plt.title("Distribution of "+col)
plt.show()
```


Distribution of open
Distribution of high
Distribution of low
Distribution of close
Distribution of volume
1e8
Distribution of Price_Up_down
0
100000
200000
300000
400000
500000
600000
2000
1500
1000
500
6
4
2
1.0
0.8
0.6
0.4
0.2
0.0

In [17]:

```
#histogram of all fetures
p =df.hist(figsize = (20,20))
```



In [18]:

```
#fininding the feture importance using XGboost
import xgboost as xgb

train_y = df['Price_Up_down']
train_X = df.drop(['Price_Up_down'], axis=1)

xgb_params = {
    'eta': 0.05,
    'max_depth': 10,
    'subsample': 1.0,
    'colsample_bytree': 0.7,
    'objective': 'reg:linear',
    'eval_metric': 'rmse',
    'silent': 1
}
```

In [19]:

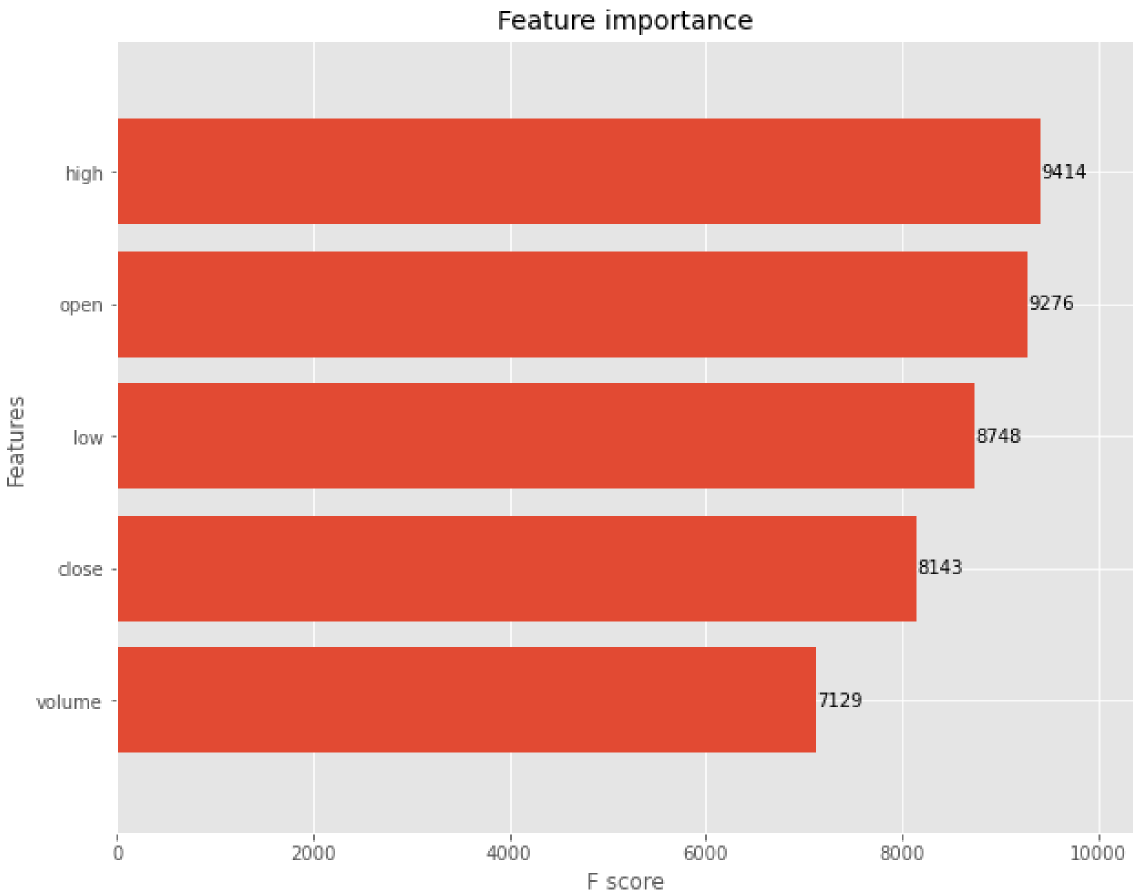
```
import warnings
warnings.filterwarnings('ignore')
dtrain = xgb.DMatrix(train_X, train_y, feature_names=train_X.columns.values)
model = xgb.train(dict(xgb_params, silent=0), dtrain, num_boost_round=100)
remain_num = 99

fig, ax = plt.subplots(figsize=(10,8))
xgb.plot_importance(model, max_num_features=remain_num, height=0.8, ax=ax)
plt.show()
```

```
[06:17:27] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release
_1.2.0/src/objective/regression_obj.cu:174: reg:linear is now deprecated i
n favor of reg:squarederror.
[06:17:27] WARNING: C:\Users\Administrator\workspace\xgboost-win64_release
_1.2.0\src\learner.cc:516:
Parameters: { silent } might not be used.

  This may not be accurate due to some parameters are only used in languag
e bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip t
hrough this
  verification. Please open an issue if you find above cases.


[06:23:49] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release
_1.2.0/src/objective/regression_obj.cu:174: reg:linear is now deprecated i
n favor of reg:squarederror.
```



In [26]:

```
#Split the data set into a feature or independent data set (X) and a target or dependen
t data set (Y)
x = df.iloc[:, 0:df.shape[1] -1].values #Get all the rows and columns except for the ta
rget column
y = df.iloc[:, df.shape[1]-1].values  #Get all the rows from the target column
```

In [28]:

```
#Split the data again but this time into 80% training and 20% testing data sets
x_train, x_test, y_train, y_test = train_test_split(x,y,
                                                    test_size=0.20, random_state = 0)
```

In [29]:

```
from sklearn.ensemble import RandomForestClassifier
ran_clf = RandomForestClassifier(max_depth=2, random_state=0)
```

In [30]:

```
rfmodel = ran_clf.fit(x_train,y_train)
y_pred2 = rfmodel.predict(x_test)
accuracy_score(y_test,y_pred2)
```

Out[30]:

```
0.5264526759607774
```

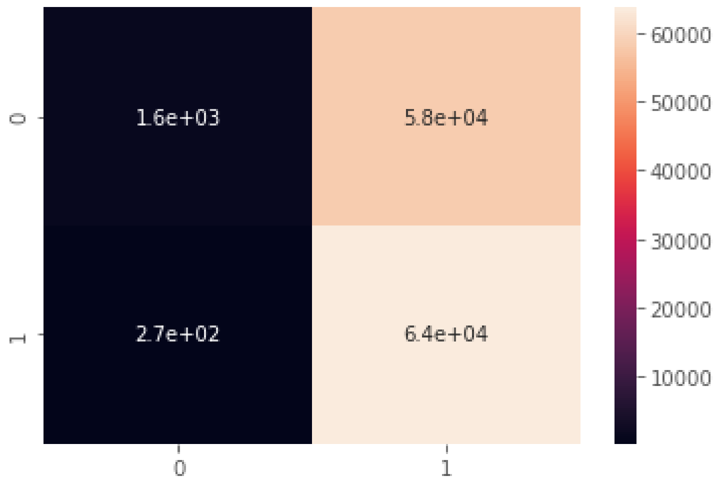
In [31]:

```
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
cn = confusion_matrix(y_test,y_pred2)
sns.heatmap(cn,annot=True)
print(confusion_matrix(y_test,y_pred2))
print(accuracy_score(y_test,y_pred2))
print(classification_report(y_test,y_pred2))
```

```
[[ 1573 58357]
 [  271 63605]]
0.5264526759607774
              precision    recall  f1-score   support

           0       0.85      0.03      0.05     59930
           1       0.52      1.00      0.68     63876

    accuracy                           0.53    123806
   macro avg       0.69      0.51      0.37    123806
weighted avg       0.68      0.53      0.38    123806
```

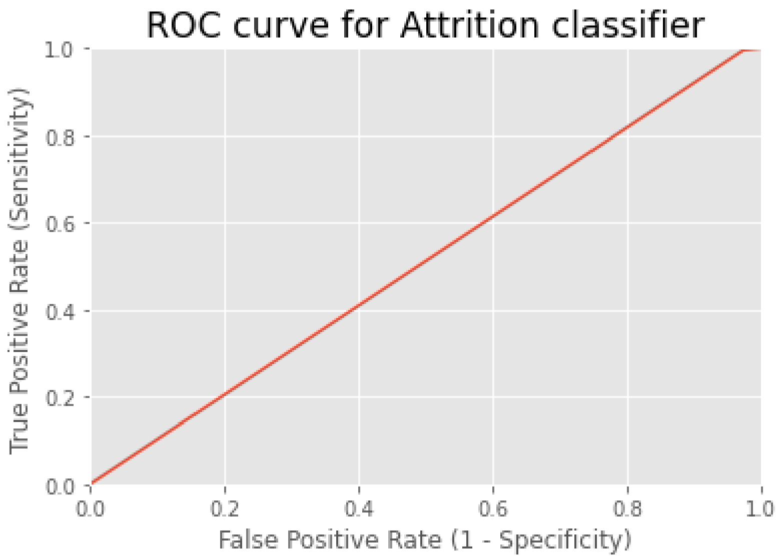


In [33]:

```
from sklearn import metrics
#IMPORTANT: first argument is true values, second argument is predicted probabilities

# we pass y_test and y_pred_prob
# we do not use y_pred_class, because it will give incorrect results without generating an error
# roc_curve returns 3 objects fpr, tpr, thresholds
# fpr: false positive rate
# tpr: true positive rate
fpr, tpr, thresholds = metrics.roc_curve(y_test,y_pred2)

plt.plot(fpr, tpr)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.rcParams['font.size'] = 12
plt.title('ROC curve for Attrition classifier')
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.grid(True)
```



In [34]:

```
#Create and train the decision tree Classifier model
tree = DecisionTreeClassifier().fit(X_train, Y_train)
```

In [35]:

```
#Check how well the model did on the training data set
print( tree.score(X_train, Y_train))
```

```
1.0
```

In [36]:

```
#Check how well the model did on the test data set
print( tree.score(X_test, Y_test))
```

```
0.9737573300163158
```

In [37]:

```
#Show the actual values from the test data set
print(Y_test)
```

```
[1 0 0 ... 1 1 1]
```

In [ ]:

In [ ]: