Machine Learning Project: Predicting SNAP binary return(Positive/Negative) by using XGBoost Model

x: return of the last 60 trading days(before y)

y: binary return

```python
#library
import math
from yahoofinancials import YahooFinancials as yf
import numpy as np
import pandas as pd
from xgboost import XGBClassifier
from sklearn.preprocessing import StandardScaler
import datetime
```

[1] Python

```python
#Stock Data
ticker = 'SNAP'
ticker_info = yf('SNAP')
hist = ticker_info.get_historical_price_data('2000-01-01','2024-03-20','daily')[ticker]['prices']
df = pd.DataFrame(hist)

#Stock Data Cleaning
def date_formatting(input_date):
    year = input_date[:4]
    month = input_date[5:7]
    day = input_date[-2:]
    return year + month + day

#Creating a column to indicate the positive return by "1" & negative return by "0"
def binary_output(input):
    if input >=0:
        return 1
    else:
        return 0

df['date']= df.apply(lambda x: date_formatting(x['formatted_date']),axis=1)
df= df[['date','close','adjclose','volume']]
df = df.set_index(['date'])
df['return'] = (df['adjclose']/df['adjclose'].shift(1))-1
df['binary_return'] = df.apply(lambda x: binary_output(x['return']),axis=1)
df = df.dropna()
df
```

[2] ✓ 9.4s Python

...

date	close	adjclose	volume	return	binary_return
20170303	27.090000	27.090000	148166400	0.106618	1
20170306	23.770000	23.770000	72903000	-0.122554	0
20170307	21.440001	21.440001	71857800	-0.098023	0
20170308	22.809999	22.809999	49819100	0.063899	1
20170309	22.709999	22.709999	25803200	-0.004384	0
...	...	...	...	...	...
20240313	11.900000	11.900000	29908900	0.011045	1
20240314	11.390000	11.390000	21971800	-0.042857	0
20240315	11.190000	11.190000	26630100	-0.017559	0
20240318	11.060000	11.060000	28446200	-0.011617	0
20240319	11.050000	11.050000	25976800	-0.000904	0

1773 rows × 5 columns

```python
#Scaling the Train & Test Data
x = df[['return']]
y = df[['binary_return']]
training_data_len = math.ceil(len(df)*0.7)
scaler = StandardScaler()

x_train = x.iloc[0:training_data_len,:]
x_test = x.iloc[training_data_len-60:,:] #Prevent fitted data from polluting the test data set
x_train_s = scaler.fit_transform(x_train)
x_test_s = scaler.transform(x_test)

y_train = y.iloc[60:training_data_len,:]

x_train_s2 = []

for i in range(60, len(x_train_s)):
    x_train_s2.append(x_train_s[i-60 : i, 0])

y_train_s2 = np.array(y_train)
```

[3] ✓ 0.0s Python

```python
x_train_df = pd.DataFrame(x_train_s2, columns=["return_" + str(i) for i in range(1,61)])
var_list = x_train_df.columns
```

[4] ✓ 0.0s Python

```python
#Building XGBoost Model
xgb = XGBClassifier(objective = 'binary:logistic', random_state = 42)
xgb.fit(x_train_df, y_train_s2)
```

[5] ✓ 1.6s Python

```python
#Creating Test Data Set
x_test_s2=[]
for i in range(60, len(x_test_s)):
    x_test_s2.append(x_test_s[i-60:i,0])
x_test_df = pd.DataFrame(x_test_s2, columns=["return_" + str(i) for i in range(1,61)])
```

[6] ✓ 0.0s Python

```python
#Applying XGBoost Model to Test Data
predictions = xgb.predict(x_test_df)
predictions_with_prob = xgb.predict_proba(x_test_df)
```

[7] ✓ 0.0s Python

```python
#Result
valid = df[training_data_len:]
valid['Predictions'] = predictions
valid['class_1_prob'] = predictions_with_prob[:,[1]]
valid
```

[8] ✓ 0.0s Python

	close	adjclose	volume	return	binary_return	Predictions	class_1_prob
date							
20220207	37.880001	37.880001	96743300	-0.026471	0	1	0.733314
20220208	37.560001	37.560001	85621100	-0.008448	0	1	0.523302
20220209	40.279999	40.279999	67166800	0.072417	1	1	0.674830
20220210	40.619999	40.619999	65264300	0.008441	1	1	0.837801
20220211	39.490002	39.490002	42826900	-0.027819	0	1	0.536061
...	...	...	...	...	...	...	...
20240313	11.900000	11.900000	29908900	0.011045	1	0	0.440638
20240314	11.390000	11.390000	21971800	-0.042857	0	0	0.302521
20240315	11.190000	11.190000	26630100	-0.017559	0	1	0.902237
20240318	11.060000	11.060000	28446200	-0.011617	0	0	0.499316
20240319	11.050000	11.050000	25976800	-0.000904	0	0	0.177999

531 rows × 7 columns

```python
#Get the success rate of the prediction
Total = len(valid)
success_case = len(valid[valid['binary_return']==valid['Predictions']])
success_rate = success_case/Total
success_rate
```

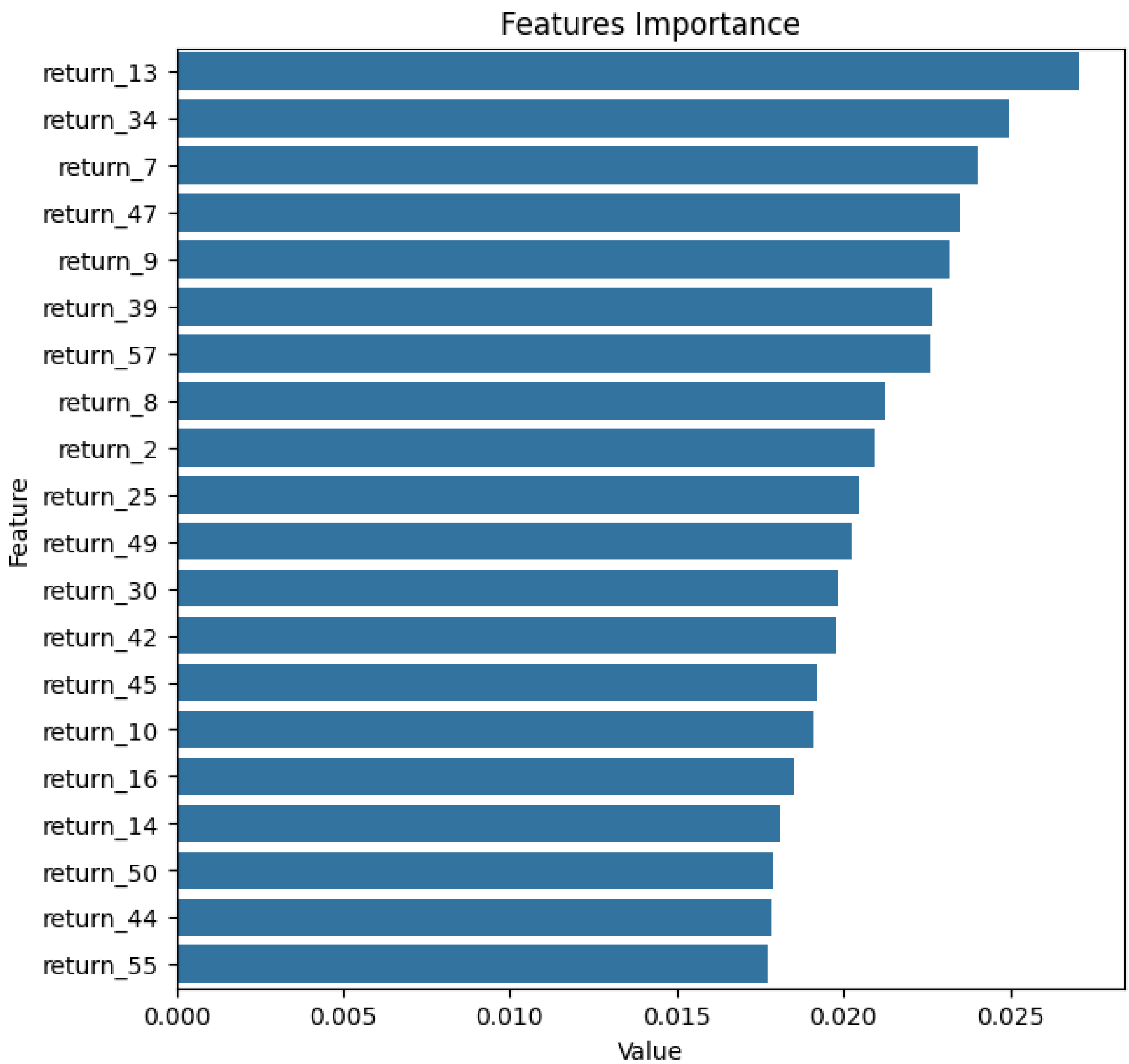
[9] ✓ 0.0s Python

```
0.4745762711864407
```

```python
#Feature Importance
feature_imp = pd.DataFrame(sorted(zip(xgb.feature_importances_,var_list)), columns=['Value','Feature'])

plt.figure(figsize=(7, 7))
sns.barplot(x="Value", y="Feature", data=feature_imp.nlargest(20,"Value").sort_values(by="Value", ascending=False))
plt.title('Features Importance')
plt.show()
```

[11] ✓ 0.8s Python



return_n: lag (n)th return before the prediction date