CV CS4375 HW9

Ramesh Kanakala

This is a Python notebook with the purpose of running classification algorithms on a day trading stock data set to compare it's performance with R classification.

Initial Data Exploration and Cleaning

Check out the Kaggle link for more information on the data set and its variables:
https://www.kaggle.com/dawerty/cleaned-daytrading-training-data

In [1]:
```
import pandas as pd
#load data
df = pd.read_csv('stock.csv', header=0)
```

In [2]:
```
#exploration function 1
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 258729 entries, 0 to 258728
Data columns (total 23 columns):
 #   Column                    Non-Null Count   Dtype  
---  ------                    --------------   -----  
 0   is_profit                 258729 non-null  bool   
 1   sym                       258729 non-null  object 
 2   datetime                  258729 non-null  object 
 3   rsi14                     258729 non-null  float64
 4   sma9_var                  258729 non-null  float64
 5   sma180_var                258729 non-null  float64
 6   vwap_var                  258729 non-null  float64
 7   spread14_e                258729 non-null  float64
 8   volume14_34_var           258729 non-null  float64
 9   prev_close_var            258729 non-null  float64
 10  prev_floor_var            258729 non-null  float64
 11  prev_ceil_var             258729 non-null  float64
 12  prev1_candle_score        258729 non-null  float64
 13  prev2_candle_score        258729 non-null  float64
 14  prev3_candle_score        258729 non-null  float64
 15  mins_from_start           258729 non-null  float64
 16  valley_interval_mins      258729 non-null  float64
 17  valley_close_score        258729 non-null  float64
 18  valley_rsi_score          258729 non-null  float64
 19  day_open_var              258729 non-null  float64
 20  open_from_prev_close_var  258729 non-null  float64
 21  ceil_var                  258729 non-null  float64
 22  floor_var                 258729 non-null  float64
dtypes: bool(1), float64(20), object(2)
memory usage: 43.7+ MB
```

In [3]:
```
#exploration function 2
df.describe()
```

Out[3]:

	rsi14	sma9_var	sma180_var	vwap_var	spread14_e	volume14_34_var	p
count	258729.000000	258729.000000	258729.000000	258729.000000	258729.000000	258729.000000	2
mean	34.566266	-0.002659	-0.011785	-0.009576	0.000885	-0.043627	
std	5.463582	0.002344	0.014630	0.010056	0.000765	0.310340	

	rsi14	sma9_var	sma180_var	vwap_var	spread14_e	volume14_34_var	p
min	6.140843	-0.019984	-0.199053	-0.126805	0.000002	-1.000000	
25%	31.112562	-0.003322	-0.016985	-0.013535	0.000406	-0.230678	
50%	34.838873	-0.001950	-0.008588	-0.007803	0.000653	-0.047021	
75%	38.210769	-0.001157	-0.003302	-0.003722	0.001089	0.134563	
max	71.815499	0.002438	0.183595	0.071907	0.012104	1.428571	

In [4]:

```
#cleaning: changing is_profit data types
df.is_profit = df.is_profit.astype('category').cat.codes
del df["datetime"] #drop datetime column
del df["sym"] #drop sym column
```

In [5]:

```
#exploration function 3
df.head
```

Out[5]:

```
<bound method NDFrame.head of          is_profit      rsi14  sma9_var  sma180_var  vwap_v
ar  spread14_e  \
0               1  30.509761 -0.006223   -0.022679 -0.017526    0.000620
1               1  46.452741 -0.001062   -0.004721 -0.007713    0.000695
2               1  34.336224 -0.004443   -0.016648 -0.016589    0.000518
3               0  36.584676 -0.001006    0.005697 -0.004279    0.000327
4               1  29.113480 -0.000950    0.002626 -0.001767    0.000286
...           ...        ...       ...         ...       ...         ...
258724          0  32.602899 -0.002293   -0.033872 -0.011582    0.001114
258725          0  37.355860 -0.002370   -0.021928 -0.015194    0.000470
258726          1  41.550637 -0.001991   -0.014774 -0.003803    0.000993
258727          0  35.433061 -0.005427   -0.012600 -0.015767    0.001918
258728          1  37.648564 -0.004007   -0.016134 -0.016528    0.002374

        volume14_34_var  prev_close_var  prev_floor_var  prev_ceil_var  ...  \
0             -0.006472       -0.037037       -0.012658      -0.047328  ...
1              0.280249       -0.031893       -0.007384      -0.042239  ...
2              0.284800        0.011396        0.023360      -0.014706  ...
3             -0.514448        0.017371        0.023517       0.001266  ...
4             -0.033291        0.019482        0.035230      -0.001628  ...
...                 ...             ...             ...            ...  ...
258724         0.462271       -0.029817       -0.028145      -0.060000  ...
258725        -0.238875       -0.038417       -0.036760      -0.068333  ...
258726        -0.062510        0.057895        0.105147       0.050330  ...
258727         0.046143        0.004747        0.015770      -0.095270  ...
258728        -0.000039        0.002035        0.013027      -0.097713  ...

        prev2_candle_score  prev3_candle_score  mins_from_start  \
0                 0.000000            0.000000            103.0
1                 0.001062            0.000504            265.0
2                -0.001020            0.000000            278.0
3                -0.000210            0.000000            110.0
4                -0.000012            0.000000            229.0
...                    ...                 ...              ...
258724            0.000000           -0.000590             85.0
258725            0.001488            0.000209            247.0
258726            0.000000            0.003775            285.0
258727           -0.001341            0.002687            146.0
258728           -0.001509           -0.000506            216.0

        valley_interval_mins  valley_close_score  valley_rsi_score  \
0                       50.0            0.425532          0.758046
```

```
1                        67.0            0.633584        10.958588
2                        13.0            0.306356         2.964667
3                         8.0            0.042142         2.599359
4                        29.0            0.224383         0.091923
...                       ...                 ...              ...
258724                   19.0            0.118066         7.540690
258725                   46.0            0.563293        10.917935
258726                   20.0            0.077993         1.535276
258727                   12.0            0.235696         4.331218
258728                    9.0            0.002033        10.013797

        day_open_var  open_from_prev_close_var  ceil_var  floor_var
0          -0.032058                 -0.005144 -0.034554   0.000802
1          -0.026887                 -0.005144 -0.029397   0.006148
2          -0.003935                  0.015391 -0.030638   0.000000
3           0.003596                  0.013725 -0.014746   0.003596
4           0.008738                  0.010651 -0.016754   0.011470
...              ...                       ...       ...        ...
258724     -0.021400                 -0.008601 -0.030373   0.000000
258725     -0.030075                 -0.008601 -0.038968   0.000000
258726      0.070815                 -0.012065 -0.037562   0.070815
258727      0.003386                  0.001356 -0.033909   0.003386
258728      0.000677                  0.001356 -0.036518   0.000677

[258729 rows x 21 columns]>
```

In [6]:

```
#exploration function 3 (cont.)
df.tail
```

Out[6]:

```
<bound method NDFrame.tail of         is_profit      rsi14  sma9_var  sma180_var  vwap_v
ar  spread14_e  \
0               1  30.509761 -0.006223   -0.022679 -0.017526    0.000620
1               1  46.452741 -0.001062   -0.004721 -0.007713    0.000695
2               1  34.336224 -0.004443   -0.016648 -0.016589    0.000518
3               0  36.584676 -0.001006    0.005697 -0.004279    0.000327
4               1  29.113480 -0.000950    0.002626 -0.001767    0.000286
...           ...        ...       ...         ...       ...         ...
258724          0  32.602899 -0.002293   -0.033872 -0.011582    0.001114
258725          0  37.355860 -0.002370   -0.021928 -0.015194    0.000470
258726          1  41.550637 -0.001991   -0.014774 -0.003803    0.000993
258727          0  35.433061 -0.005427   -0.012600 -0.015767    0.001918
258728          1  37.648564 -0.004007   -0.016134 -0.016528    0.002374

        volume14_34_var  prev_close_var  prev_floor_var  prev_ceil_var  ...  \
0             -0.006472       -0.037037       -0.012658      -0.047328  ...
1              0.280249       -0.031893       -0.007384      -0.042239  ...
2              0.284800        0.011396        0.023360      -0.014706  ...
3             -0.514448        0.017371        0.023517       0.001266  ...
4             -0.033291        0.019482        0.035230      -0.001628  ...
...                 ...             ...             ...            ...  ...
258724         0.462271       -0.029817       -0.028145      -0.060000  ...
258725        -0.238875       -0.038417       -0.036760      -0.068333  ...
258726        -0.062510        0.057895        0.105147       0.050330  ...
258727         0.046143        0.004747        0.015770      -0.095270  ...
258728        -0.000039        0.002035        0.013027      -0.097713  ...

        prev2_candle_score  prev3_candle_score  mins_from_start  \
0                 0.000000            0.000000            103.0
1                 0.001062            0.000504            265.0
2                -0.001020            0.000000            278.0
3                -0.000210            0.000000            110.0
4                -0.000012            0.000000            229.0
...                    ...                 ...              ...
258724            0.000000           -0.000590             85.0
258725            0.001488            0.000209            247.0
```

```
258726          0.000000          0.003775          285.0
258727         -0.001341          0.002687          146.0
258728         -0.001509         -0.000506          216.0

        valley_interval_mins  valley_close_score  valley_rsi_score  \
0                       50.0            0.425532          0.758046
1                       67.0            0.633584         10.958588
2                       13.0            0.306356          2.964667
3                        8.0            0.042142          2.599359
4                       29.0            0.224383          0.091923
...                      ...                 ...               ...
258724                  19.0            0.118066          7.540690
258725                  46.0            0.563293         10.917935
258726                  20.0            0.077993          1.535276
258727                  12.0            0.235696          4.331218
258728                   9.0            0.002033         10.013797

        day_open_var  open_from_prev_close_var  ceil_var  floor_var
0          -0.032058                 -0.005144 -0.034554   0.000802
1          -0.026887                 -0.005144 -0.029397   0.006148
2          -0.003935                  0.015391 -0.030638   0.000000
3           0.003596                  0.013725 -0.014746   0.003596
4           0.008738                  0.010651 -0.016754   0.011470
...              ...                       ...       ...        ...
258724     -0.021400                 -0.008601 -0.030373   0.000000
258725     -0.030075                 -0.008601 -0.038968   0.000000
258726      0.070815                 -0.012065 -0.037562   0.070815
258727      0.003386                  0.001356 -0.033909   0.003386
258728      0.000677                  0.001356 -0.036518   0.000677

[258729 rows x 21 columns]>
```

In [7]:

```python
#exploration function 4
test= df.groupby(['is_profit','valley_rsi_score'])
test.size()
```

Out[7]:

```
is_profit  valley_rsi_score
0          0.000027            1
           0.000084            1
           0.000151            1
           0.000152            1
           0.000156            1
                              ..
1          36.952446           1
           37.433231           1
           37.983256           1
           39.189261           1
           44.138675           1
Length: 258729, dtype: int64
```

In [8]:

```python
#exploration function 5
df.isnull()
```

Out[8]:

	is_profit	rsi14	sma9_var	sma180_var	vwap_var	spread14_e	volume14_34_var	prev_close_var
0	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False

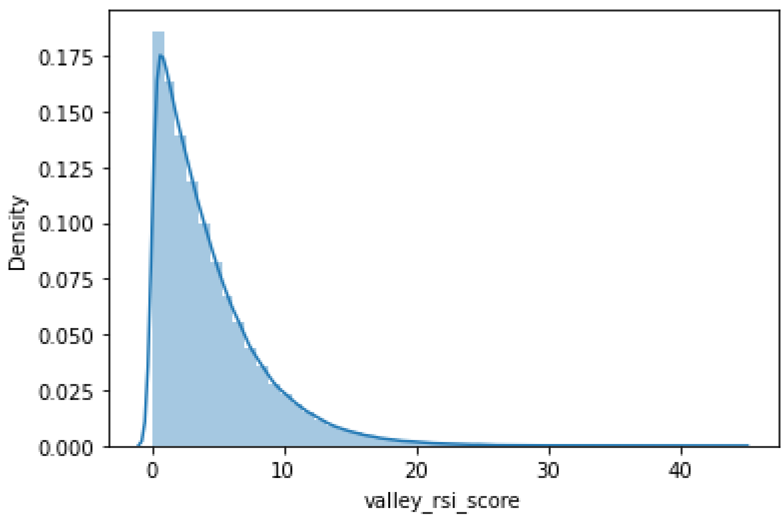
	is_profit	rsi14	sma9_var	sma180_var	vwap_var	spread14_e	volume14_34_var	prev_close_var
...	...	...	...	...	...	...	...	...
258724	False	False	False	False	False	False	False	False
258725	False	False	False	False	False	False	False	False
258726	False	False	False	False	False	False	False	False
258727	False	False	False	False	False	False	False	False
258728	False	False	False	False	False	False	False	False

258729 rows × 21 columns

In [9]:

```
#exploration graph 1
from plotnine import *
import seaborn as sns
sns.distplot(df['valley_rsi_score']);
```

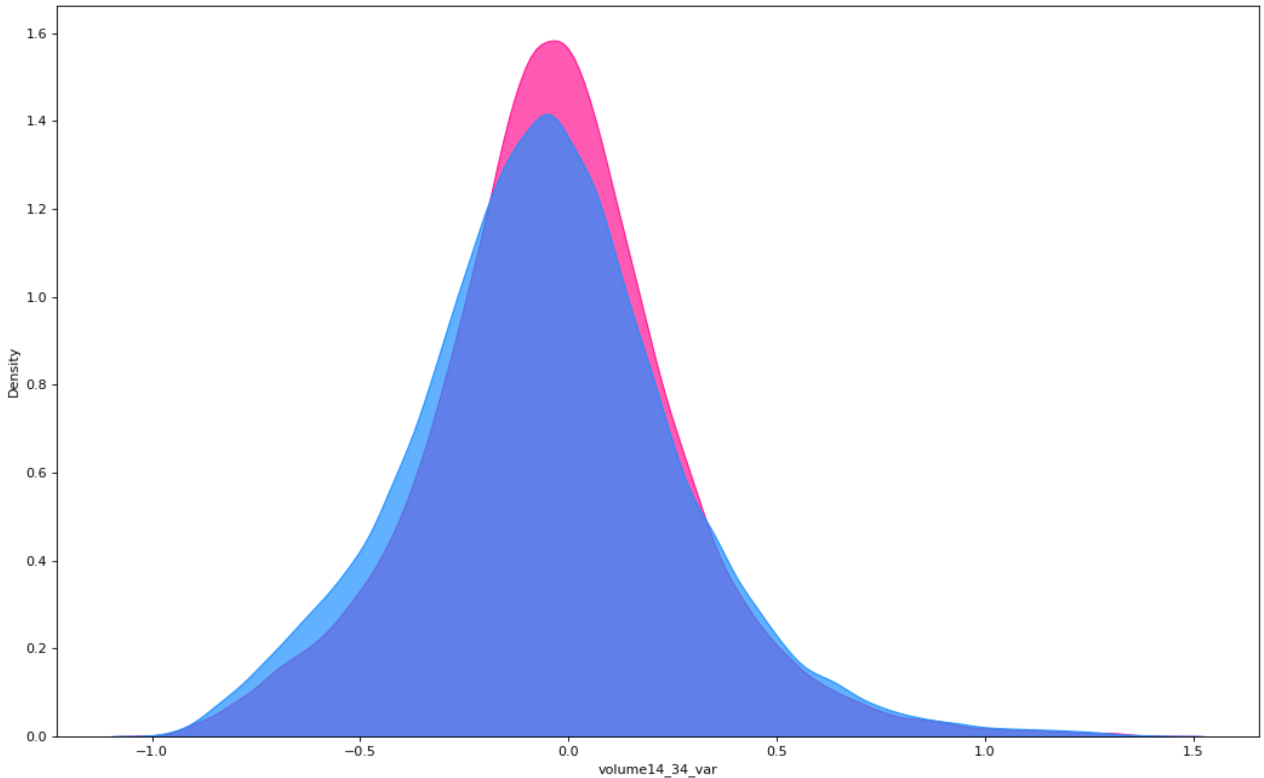
```
C:\Users\RaxyR\anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function for histograms).
```



In [37]:

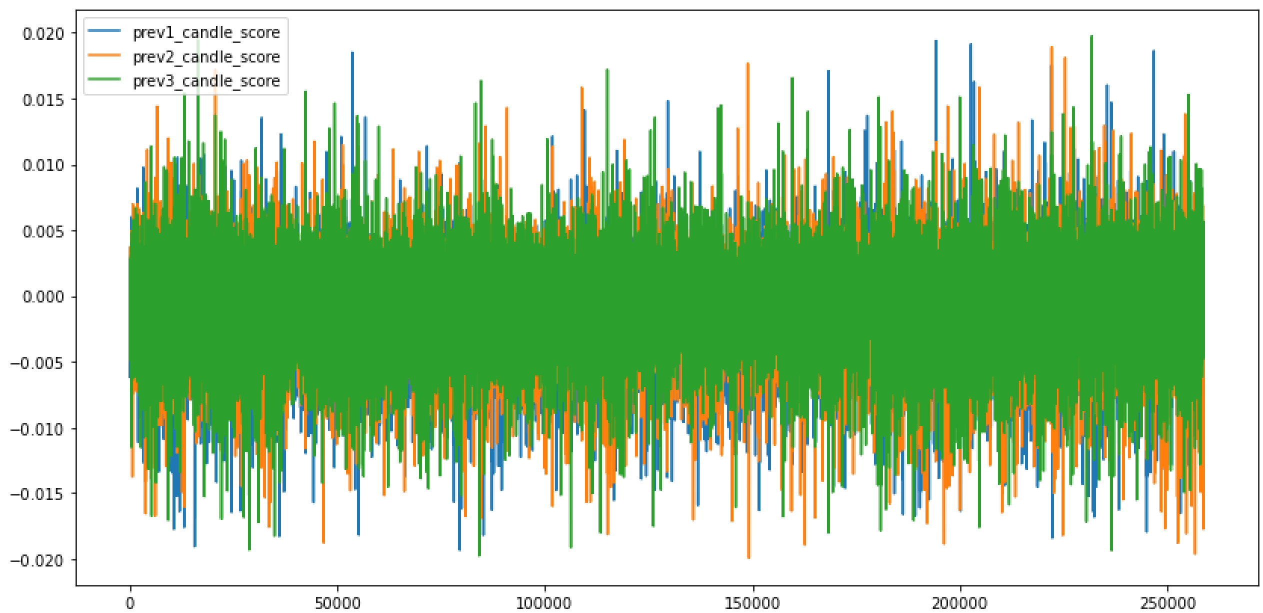
```
#exploration graph 2
import matplotlib.pyplot as plt
plt.figure(figsize=(16,10), dpi= 80)
sns.kdeplot(df.loc[df['is_profit'] == 0, "volume14_34_var"], shade=True, color="deeppin
sns.kdeplot(df.loc[df['is_profit'] == 1, "volume14_34_var"], shade=True, color="dodgerb
```

Out[37]: <AxesSubplot:xlabel='volume14_34_var', ylabel='Density'>



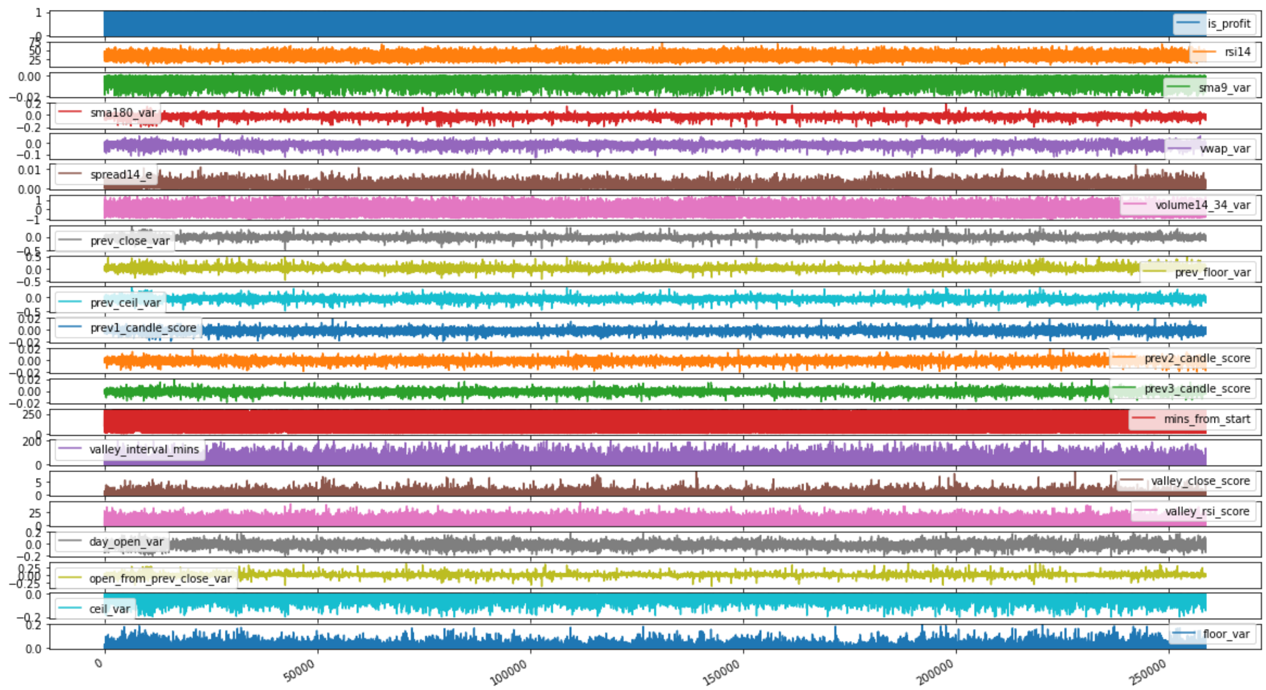
In [26]:
```
#exploration graph 4
df[['prev1_candle_score', 'prev2_candle_score', 'prev3_candle_score']].plot(figsize = (
```

Out[26]: <AxesSubplot:>



In [31]:
```
#exploration graph 4
df.plot(subplots=True, figsize=(20, 12))
```

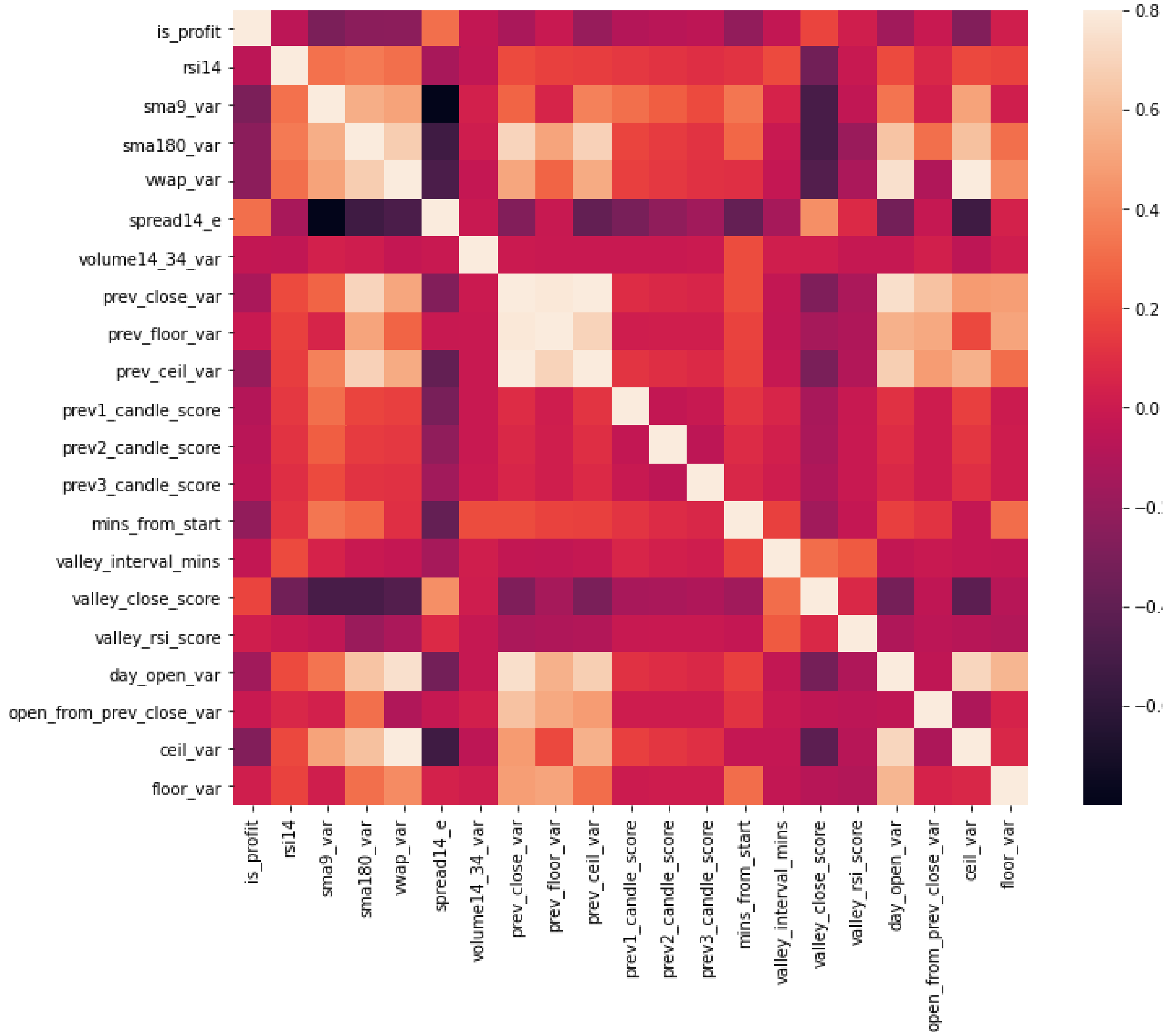
Out[31]:
```
array([<AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>,
       <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>,
       <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>,
       <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>,
       <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>,
       <AxesSubplot:>], dtype=object)
```



In [26]:

```
#exploration graph 5
corrmat = df.corr()
f, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corrmat, vmax=.8, square=True)
```

Out[26]: <AxesSubplot:>



Modeling

In [11]:

```
# train test split
from sklearn.model_selection import train_test_split
X = df.iloc[:, df.columns != 'is_profit']
y = df.iloc[:, df.columns=='is_profit']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=1234)
print('train size:', X_train.shape)
print('test size:', X_test.shape)
```

```
train size: (194046, 20)
test size: (64683, 20)
```

In [13]:

```
#logistic regression model
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(solver='lbfgs', max_iter=10000)
clf.fit(X_train, y_train)
clf.score(X_train, y_train)

# make predictions
pred1 = clf.predict(X_test)
from sklearn.metrics import classification_report
print(classification_report(y_test, pred1))
```

```
probas_pred = clf.predict_proba(X_test)[:,1]
```

```
C:\Users\RaxyR\anaconda3\lib\site-packages\sklearn\utils\validation.py:72: DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().

              precision    recall  f1-score   support

           0       0.70      0.88      0.78     40206
           1       0.66      0.38      0.48     24477

    accuracy                           0.69     64683
   macro avg       0.68      0.63      0.63     64683
weighted avg       0.68      0.69      0.67     64683
```

In [32]:

```
#naive bayes model
from sklearn.naive_bayes import GaussianNB #note: using GaussianNB due to negative numb
clf = GaussianNB()
clf.fit(X_train, y_train)
clf.score(X_train, y_train)

# make predictions
pred2 = clf.predict(X_test)
print(classification_report(y_test, pred2))
```

```
              precision    recall  f1-score   support

           0       0.68      0.88      0.77     40206
           1       0.62      0.34      0.44     24477

    accuracy                           0.67     64683
   macro avg       0.65      0.61      0.60     64683
weighted avg       0.66      0.67      0.64     64683

C:\Users\RaxyR\anaconda3\lib\site-packages\sklearn\utils\validation.py:72: DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
```

In [33]:

```
#decision tree model
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
clf.score(X_train, y_train)

# make predictions
pred3 = clf.predict(X_test)
print(classification_report(y_test, pred3))
```

```
              precision    recall  f1-score   support

           0       0.69      0.68      0.69     40206
           1       0.49      0.50      0.49     24477

    accuracy                           0.61     64683
   macro avg       0.59      0.59      0.59     64683
```

```
weighted avg       0.61      0.61      0.61     64683
```

Logistic regression actually performed just slightly worse with Python than with R (accuracy .69 < .70). Naive bayes also performed slightly worse with Python (.67 < .68). Finally, the decision tree performed slightly better with Python (.69 > .68). Ranking these three algorithms in Python, we have decision tree and logistic regression tied and then naive bayes. This is very similar with the R project with the contention between the decision tree and logistic regression being with the margins of 1-2% accuracy.

Some of the predictors may not have been independent so the naive assumption that they are may have limited the performance of the naives bayes model. This is most likely the reason it was outperformed by logistic regression and the decision tree. Logistic regression searches for a single linear decision boundary whereas the decision tree partitions the feature space into half spaces for a boundary but in this case the effect was more or less the same. However, because decision trees are so flexible, the model may have been prone to overfitting and logistic regression was less susceptible here. Maybe if any pruning was done, the accuracy could have increased. All in all, this was a battle of bias-variance tradeoff and logistic won, very slightly, and naive bayes struggled against the size of the data set.

Interestingly, naive bayes ran the fastest in Python and logistic regression the slowest, essentially the opposite of what occurred in R.

Personally, I lean towards machine learning in R rather than Python perhaps because I have had a longer history of experience with it but also because of ease of explanatory data analysis. There are a wide array of statistical functions and options with plotting and even though I was able to replicate ggplot2 in Python with the plotnine library, there are less nuances with R. I do enjoy the power of Python in that it can be used for almost anything being very flexible. R being very data analysis oriented makes it very useful for just focusing on that. So, to summarize, I believe data analysis and visualization is better done in R but more complicated modeling and deep learning should be done with Python. I think in the long run, I would use Python as it is more powerful and versatile but R will be a personal favorite with it's ease of use.