Marijuana stocks Project

By
Stacey Baroulette
Federico Cestau
Javy Armstrong

Why We Chose the Cannabis Industry

1. Emerging market: to determine if there is any value in the marijuana stock industry
2. If there is any value we wanted to see which stock has been consistent throughout the years
3. How well can a machine learning model project the outcome of this industry

Top 36 Stocks

1. ACAN
2. ACB
3. AMRS
4. APHA
5. CARA
6. CRBP
7. GTBIF
8. IIPR
9. MJ
10. MRMD
11. NBEV
12. TGOD
13. TGODF
14. TLRY
15. TRST
16. TRTC
17. ACRGF
18. ALEAF
19. CURLF
20. YCBD
21. CRON
22. GWPH
23. CGC
24. SMG
25. TAP
26. WEED.TO
27. MMNFF
28. MO
29. BUD
30. HRVSF
31. TLLTF
32. AYRSF
33. GRWG
34. PLNHF
35. CXXIF
36. KSHB

Industry Challenges

1. Regulation
2. New Industry
3. Legalization

Legal Visualization

Challenges We Faced

1. Fairly new industry
2. Volatility
3. Financial Differences (outliers)
4. False Data

Solutions

1. Chose top ten stocks based on market cap
2. We found higher market cap stocks has less price swings
3. Taking the top ten stocks eliminated some outliers in our data

Financial Data for Cannabis Stocks

```python
base_url = "https://financialmodelingprep.com/api/v3/key-metrics/"
query = f"?period=quarter&apikey=12ca4a34c2a88857222c5812d1fcfc50"
```

```python
ticker_list = ["AMRS", "CGC","TLRY","CARA","CRBP","IIPR","NBEV","TRST","YCBD","CRON","GWPH",
               "SMG","TAP","MO","BUD",]
```

```python
ROE = []
Price_Sales_Ratio = []
Payout_Ratio = []
Debt_Equity_Ratio = []
Dividend_Yield = []
for ticker in ticker_list:
    print("Try: " + ticker)


    url2 = base_url2 + ticker + query2
    response = requests.get(url2).json()

    for d in response:
        Price_Sales_Ratio.append(d['priceToSalesRatio'])
        Payout_Ratio.append(d['payoutRatio'])
        ROE.append(d['returnOnEquity'])
        Debt_Equity_Ratio.append(d['debtEquityRatio'])
        Dividend_Yield.append(d['dividendYield'])
```

Financial Data for Cannabis Stocks

```
Financial_DF = pd.DataFrame({
    "F_Date": Date,
    "EPS": Net_Income_Per_Share,
    "CFPS":Free_Cash_Flow_Per_Share,
    "PE_Ratio": PE_Ratio,
    "Market_Cap":Market_Cap,
    "ROE":ROE,
    "Price_Sales_Ratio":Price_Sales_Ratio,
    "Payout_Ratio":Payout_Ratio,
    "Debt_Equity_Ratio":Debt_Equity_Ratio,
    "Dividend_Yield":Dividend_Yield,
    "Ticker":Ticker})
Financial_DF
```

F_Date	EPS	CFPS	PE_Ratio	Market_Cap	ROE	Price_Sales_Ratio	Payout_Ratio	Debt_Equity_Ratio
2020-03-31	-0.565322	-0.299041	-4.776040	4.186772e+08	0.482354	14.372716	0.0	-1.889731
2019-12-31	-0.776181	-0.428783	-3.414150	2.686322e+08	0.307620	6.627003	0.0	-1.629322

```
Financial_DF['Quarter'] = pd.to_datetime(Financial_DF['F_Date']).dt.quarter
```

```
Financial_DF['Quarter-Year'] = pd.to_datetime(Financial_DF['F_Date']).dt.year
```

After we got the arrays with Quarterly financial data for each stock using the API we created a data frame with this information using this code:

Top 10 stocks by Market Capitalization

	Ticker	Quarter-Year	Market_Cap
0	AMRS	2010	3.798402e+09
1	AMRS	2011	1.323073e+10
2	AMRS	2012	2.498001e+09
3	AMRS	2013	3.504114e+09
4	AMRS	2014	3.532717e+09

```
Top_Stocks_3 = Top_Stocks_2.groupby(["Ticker"],as_index=False)["Market_Cap"].mean()
```

```
Top_Stocks.groupby(["Ticker","Quarter-Year"],as_index=False)["Market_Cap"].mean()
```

We chose the best stocks using the Market Capitalization criteria. Since we have quarterly data. First we got the average per year and then the average for all years group by Ticker using these codes:

Top Ten Stocks

```
Top_Stocks_4 = Top_Stocks_3.sort_values(by='Market_Cap',ascending=False).reset_index()
```

```
Top_Stocks_5 = Top_Stocks_4.head(10)
Top_Stocks_5
```

1. Corbus Pharmaceuticals Holdings inc (CRBP)
2. Altria Group inc (MO)
3. GW Pharmaceuticals (GWPH)
4. cbdMD (YCBD)
5. Molson Coors Beverage (TAP)
6. Scotts Miracle-Gro (SMG)
7. Canopy Growth Company (CGC)
8. Amyris (AMRS)
9. New Age Beverage corp (NBEV)
10. Tilray (TLRY)

Cleaning the Table

```python
def EPS_Rating(x):
    rating=0
    if x > 0:
        rating=1
#     elif (x>5):
#         rating=1
    else:
        rating=0
    return rating
```

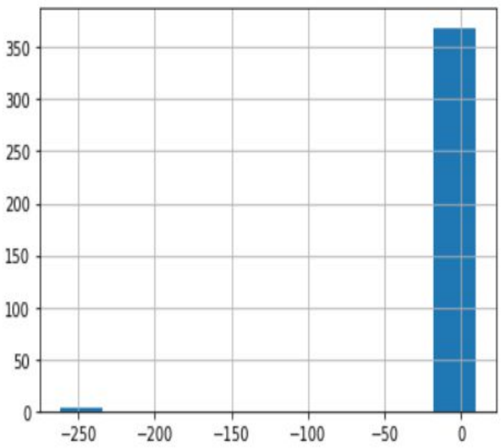
```python
Final_Table_1['Final_Rating']=Final_Table_1['EPS'].apply(EPS_Rating)
```

-We took the Earning per Share which is calculated as a company's profit divided by the outstanding shares of its common stock. We used the API to created a new column with the Annual EPS per Ticker and another column to define whether the company is making money or not using the following crtieria: EPS>0=1 if EPS<0=0.

Cleaning the Table

```
Final_Table['EPS'].hist()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x20a67c98948>
```




```
Final_Table_1=Final_Table[~(Final_Table['EPS']< -200)] # droppping companies with -250 EPS
```

-We removed the false outliers from each variable in order to make the data more realistic and make our models more accurate.

Cleaning the Table

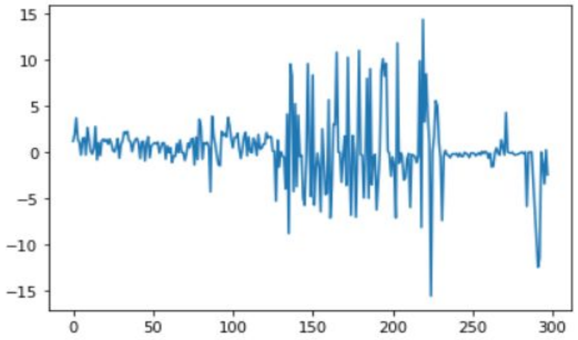
```
Top_Stocks.loc[Top_Stocks['CFPS']<-100]
```

	EPS-Annual Basis	Ticker	Quarter-Year	CFPS	PE_Ratio	Market_Cap	ROE	Price_Sales_Ratio	Payout_Ratio	Del
269	-0.024272	GWPH	2019	-191.086006	-1.250694	62614732	-0.078258	1.595402	0.000000	
288	-1.162883	NBEV	2019	-130.377408	-0.025301	295542	-0.074024	0.004454	0.000000	
290	1.880116	MO	2010	-858313.253000	110.000000	45982	0.230480	0.073300	0.699616	

```
Top_Stocks = Top_Stocks.loc[(Top_Stocks['CFPS']>-100) & (Top_Stocks['CFPS']<200)]
```

```
Top_Stocks['CFPS'].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b813769748>
```



ML Models

This is a binary classification case using the Final Rating as our dependent variable and the one we want to predict and the other variables will be our independent variables. At the end the models will try to predict whether the companies will be making money or not in the future.

TIME SERIES CONSIDERATIONS

1. Since this is a time series case we lagged our data.
2. A "lag" is a fixed amount of passing time, in this case quarter data.
3. Use data from one period to predict the Final Rating two time periods later.
4. Use historic data for the training data and more recent data for the testing data in order to make predictions.
5. We need to know whether a model can use previous data to predict new data. That's why we train the model on historic data and test whether it works on more recent data.

ML MODELS

```
#A lag is taking previous data and testing it with present data.
def make_lags(df,n,col):
    df[f'{col}_lagged_{n}']=df.groupby('Ticker')[col].shift(n)
    #df = df.dropna()
    return df
```

```
# call the function to create 2nd new lagged columns to predict data
for col in ['EPS-Annual Basis', 'CFPS', 'PE_Ratio', 'Market_Cap',
        'ROE', 'Price_Sales_Ratio', 'Debt_Equity_Ratio']:
    df = make_lags(Financial_Data, 2, col)
df = df.dropna()
```

```
Train = df[(df['Quarter-Year']<2018)]
Test = df[(df['Quarter-Year']>=2018)]
print(Train.shape)
print(Test.shape)
```

```
(208, 18)
(53, 18)
```

ML MODELS

```python
Y_Train = Train['Final_Rating']
Y_Test = Test['Final_Rating']
```

```python
#These are the variables we will use to fit into our model
X_Train = Train[['EPS-Annual Basis_lagged_2',
       'PE_Ratio_lagged_2', 'Market_Cap_lagged_2', 'ROE_lagged_2',
       'Price_Sales_Ratio_lagged_2',
       'Debt_Equity_Ratio_lagged_2']]
#These are the same variables we will use to test our model
X_Test = Test[['EPS-Annual Basis_lagged_2',
       'PE_Ratio_lagged_2', 'Market_Cap_lagged_2', 'ROE_lagged_2',
       'Price_Sales_Ratio_lagged_2',
       'Debt_Equity_Ratio_lagged_2']]
```

Dependent variable (Y) and the one we try to predict. Xs are my independent variables.

ML MODELS

```
X_train_scaled = X_scaler.transform(X_Train)
X_test_scaled = X_scaler.transform(X_Test)
```

```
#We split the data (208+53=261).208/261=79,69%
#79.69% it is my train data all the quarters before 2018
#20.31% it is my test data which is the second quarter 2018, what we are trying to predict
print(X_Train.shape)
print(X_Test.shape)
print(Y_Train.shape)
print(Y_Test.shape)
```

```
(208, 6)
(53, 6)
(208,)
(53,)
```

ML MODELS

```python
#create the model and fit the data
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
classifier
```

```python
#Fitting the train data.
classifier.fit(X_train_scaled, Y_Train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```python
print(f"Training Data Score: {classifier.score(X_train_scaled, Y_Train)}")
print(f"Testing Data Score: {classifier.score(X_test_scaled, Y_Test)}")
```

```
Training Data Score: 0.8942307692307693
Testing Data Score: 0.7924528301886793
```

```python
#Train and Test data have similar scores, therefore the model is accurate
```

MODEL CODES

```
#create the model and fit the data
from sklearn.neighbors import KNeighborsClassifier
```

```
#Fitting the train data.
knn = KNeighborsClassifier(n_neighbors=26)
knn.fit(X_train_scaled, Y_Train)
print('k=20 Test Acc: %.3f' % knn.score(X_test_scaled,Y_Test))
```

```
k=20 Test Acc: 0.642
```

```
#Data that you put aside=testing data.
print(f"Training Data Score: {knn.score(X_train_scaled, Y_Train)}")
print(f"Testing Data Score: {knn.score(X_test_scaled,Y_Test)}")
```

```
Training Data Score: 0.8798076923076923
Testing Data Score: 0.6415094339622641
```

MODEL CODES

```
# Create the GridSearch estimator along with a paramete
from sklearn.model_selection import GridSearchCV
param_grid = {'n_neighbors': [10,20,50,80]}
grid = GridSearchCV(knn, param_grid, verbose=3)
```

```
# Fit the model using the grid search estimator.
# This will take the KNN model and try each combination of parameters
grid.fit(X_train_scaled, Y_Train)
```

```
# List the best parameters for this dataset
print(grid.best_params_)
```

```
{'n_neighbors': 20}
```

```
# List the best score
print(grid.best_score_)
```

```
0.8699186991869918
```

```
#KNN has the best score in my testing data with 0.86 compared to the other models.
```

MODELS SCORES TRAIN DATA

1- Logistic Regression: 0.7924
2- Random Forest: 0.7819
3- KNN: 0.8699. Best score
4- XGB Regressor: 0.071. XGBoost works well with large datasets, it is not recommended for small dataset.
5- Support Vector Machine: 0.7924. Same score as Linear Regression.
6 Deep Learning: 0.7735

We Got the Data

1. Python Pandas
2. Python Matplotlib
3. API to Financial Websites
4. Tableau
5. Amazon Web Services

Sentiment Analysis

1. Locate Sentiment Data
2. Chart the Data
3. Compare with Stock Trends

Sentiment Visualization

Conclusion

1. Best Model - K-Nearest Neighbor (with 20 Neighbors)
2. Cannabis Stocks Industry is quite volatile
3. False public stock data
4. Difficult to predict stocks with ML
5. Industry to Watch