nft_collection_analysis

March 3, 2022

1 EDA, Feature Engineering, Hypothesis Testing, and Classification on NFT Collection Dataset

1.0.1 Introduction

What is Niftyprices? Niftyprice is a new and small team focused on providing the most up to date and comprehensive NFT data in the market today. As you know, this is a fast moving space, so we are working hard to continually push updates, increase our coverage, and enhance our data engine to provide you the best product possible while staying up to speed on all the rapid changes taking place in the NFT world. At the end of the day, we're here for you, the NFT investors and patrons, so please let us know any feedback you have or how we can help solve your burning NFT issues. -NP (source)

What are NFT's? A non-fungible token is a unit of data stored on a digital ledger, called a blockchain, that certifies a digital asset to be unique and therefore not interchangeable. NFTs can be used to represent items such as photos, videos, audio, and other types of digital files. (source)

1.0.2 Dataset Overview

Feature Information

- Collection Name: The name of the NFT collection.
- Floor Purchase Price: The lowest price of any NFT in the collection in Ethereum (ETH).
- 24%: The percentage of floor price's moving values per 24 hours.
- 7d%: The percentage of floor price's moving values per 7 days.
- Total Float: The toral amount of minted NFTs.
- Floor Cap: The lowest market capitalization—total value of the collection's items in circulation—in in Ethereum (ETH).
- Volume: The volume of sales from the NFT collection in Ethereum (ETH) per 24 hours.
- 24h Volume%: The percentage of volume's moving values per 24 hours.
- Sales: The number of sales from the NFT collection.
- 24h Owners%: The ownership percentage of all items in the collection per 24 hours.
- %Float: The percentage of listed NFTs.
- 24h supply%: The percentage of supply's moving values per 24 hours.
- image_url: The associated image of the NFT collection.

Source Dataset was scraped from niftyprices on February 20, 2022. You can scrape the latest data by yourself using 'scraper.py' python script on 'data' folder.

1.0.3 Section 1: Setup, Load, and Clean

[ ]:
```
import os
data_path = ['data']
```

[ ]:
```
## Import neccessary libraries to load data
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
```

[ ]:
```
## Load in the Dataset
filepath = os.sep.join(
    data_path + ['2022-03-01_niftyprice.csv'])
df = pd.read_csv(filepath)
df = df.sort_values(by=['Floor Cap'], ascending=False, ignore_index=True)
```

[ ]:
```
df.head()
```

[ ]:
```
          Collection Name  Floor Purchase Price  24h%   7d%  Total Float  \
0      boredapeyachtclub                 88.888  2.18 -2.85        10000
1            cryptopunks                 68.000  0.00  1.64         9999
2              wolf-game                 36.000  0.00  0.00        10443
3  mutant-ape-yacht-club                 17.800 -3.21 -2.73        16403
4                 clonex                 13.250  3.52 -1.85        16710

   Floor Cap  Volume  24h Volume%  Sales  24h Owners%  %Float  24h supply%  \
0   888880.0     619     0.004870      7         0.05    6.06         3.06
1   679932.0     826    -0.694752     11         0.00   13.54         0.30
2   375948.0       0          NaN      0         0.00    0.01         0.00
3   291973.4     384    -0.020408     19         0.03    6.21        -0.29
4   221407.5     298    -0.305361     20         0.08    8.08         2.90

                                           image_url
0  https://lh3.googleusercontent.com/Ju9CkWtV-1Ok…
1  https://lh3.googleusercontent.com/BdxvLseXcfl5…
2  https://lh3.googleusercontent.com/MMwQ4ukXLIqz…
3  https://lh3.googleusercontent.com/lHexKRMpw-ao…
4  https://lh3.googleusercontent.com/XN0XuD8Uh3jy…
```

[ ]:
```
## Examine the information from the data
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 609 entries, 0 to 608
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
```

```
 0   Collection Name       609 non-null    object
 1   Floor Purchase Price  609 non-null    float64
 2   24h%                  609 non-null    float64
 3   7d%                   609 non-null    float64
 4   Total Float           609 non-null    int64
 5   Floor Cap             609 non-null    float64
 6   Volume                609 non-null    int64
 7   24h Volume%           428 non-null    float64
 8   Sales                 609 non-null    int64
 9   24h Owners%           609 non-null    float64
 10  %Float                609 non-null    float64
 11  24h supply%           609 non-null    float64
 12  image_url             580 non-null    object
dtypes: float64(8), int64(3), object(2)
memory usage: 62.0+ KB
```

[ ]:
```
## If there's no volume change in past 7 days it might be newly listed
## Therfore we gonna remove rows with 0 value on '7d%' column
df = df[df['7d%'] != 0]
```

[ ]:
```
## Fill NaN value to 0
df['24h Volume%'].fillna(value=0, inplace=True)
```

[ ]:
```
## We plan to choose top 200 NFT's collections
df = df.drop(df.index[200:])
```

[ ]:
```
df= df.reset_index(drop=True)
```

[ ]:
```
## Drop all unnecessary columns
df.drop(columns=['image_url', '24h supply%', '7d%',
        'Floor Cap', 'Collection Name'], inplace=True)
```

[ ]:
```
## Take a quick look of the dataframe
df
```

[ ]:
```
     Floor Purchase Price   24h%  Total Float  Volume  24h Volume%  Sales  \
0                 88.8880   2.18        10000     619     0.004870      7
1                 68.0000   0.00         9999     826    -0.694752     11
2                 17.8000  -3.21        16403     384    -0.020408     19
3                 13.2500   3.52        16710     298    -0.305361     20
4                 24.8000  -0.40         7558       0     0.000000      0
..                    …      …           …       …          …      …
195                0.4500 -10.00         5073       3    -0.400000      4
196                0.4968 -14.34         4469      18    -0.672727     32
197                0.2200  -9.47        10010       3    -0.250000     12
198                0.2785  -2.28         7790       4    -0.200000     15
199                0.2400   0.00         8999       0     0.000000      1
```

```
     24h Owners%  %Float
0           0.05    6.06
1           0.00   13.54
2           0.03    6.21
3           0.08    8.08
4          -0.18    0.17
..           …       …
195         0.04    7.45
196        -0.15    7.03
197        -0.28    5.97
198        -0.86    4.63
199         0.00    0.62

[200 rows x 8 columns]
```

[ ]:
```
## Re-examine the information from the data
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 8 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Floor Purchase Price  200 non-null    float64
 1   24h%                  200 non-null    float64
 2   Total Float           200 non-null    int64
 3   Volume                200 non-null    int64
 4   24h Volume%           200 non-null    float64
 5   Sales                 200 non-null    int64
 6   24h Owners%           200 non-null    float64
 7   %Float                200 non-null    float64
dtypes: float64(5), int64(3)
memory usage: 12.6 KB
```

[ ]:
```
## Create range section in describe table
nft_df = df.copy()
stat_df = nft_df.describe()
stat_df.loc['range'] = stat_df.loc['max'] - stat_df.loc['min']
stat_df.T
```

[ ]:
```
                      count         mean          std    min          25%  \
Floor Purchase Price  200.0     6.815387    28.279739   0.10     0.474500
24h%                  200.0    -1.999250    14.490176 -34.02    -7.987500
Total Float           200.0  8221.940000  4737.843085  35.00  5036.250000
Volume                200.0    64.965000   164.997073   0.00     7.000000
24h Volume%           200.0     0.299050     1.818345  -1.00    -0.376488
Sales                 200.0    36.775000    67.912776   0.00     5.750000
24h Owners%           200.0    -0.099700     1.387376 -15.02    -0.150000
```

```
%Float                 200.0    5.555300    4.994214   0.17    2.770000

                          50%          75%      max     range
Floor Purchase Price   1.0695     2.602500   299.00    298.90
24h%                  -2.7350     0.000000   153.06    187.08
Total Float         9043.0000 10005.000000 28561.00  28526.00
Volume                18.0000    48.250000  1718.00   1718.00
24h Volume%            0.0000     0.430195    22.00     23.00
Sales                 14.5000    35.000000   494.00    494.00
24h Owners%            0.0000     0.042500     7.41     22.43
%Float                 4.4200     6.527500    34.76     34.59
```

[ ]:
```python
## Import neccessary libraries to visualize data
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set_theme(style="dark")
```

[ ]:
```python
plt.figure(figsize=[15, 10])
sns.heatmap(data=nft_df.corr(), annot=True)
```

[ ]: <AxesSubplot:>

[ ]:
```
## There are 3 potential column to be our target variable
## We gonna choose column with highest correlation with each others

corr_column_target = ['Volume', 'Sales', '%Float']
headers = ['Column Name', 'Sum of Correlations']
df_corr_result = pd.DataFrame(columns=headers)
i = 0

for k in corr_column_target:
    fields = list(nft_df.columns)
    fields.remove(k)
    y = (nft_df[k])
    correlations = nft_df[fields].corrwith(y)
    df_corr_result.loc[i] = [k, correlations.abs().sum()]
    i += 1

df_corr_result.sort_values(
    by=['Sum of Correlations'], ascending=False, ignore_index=True)
```

[ ]:

	Column Name	Sum of Correlations
0	%Float	1.599193
1	Sales	1.543980
2	Volume	1.341167

First Section Short Recap/Conclusion: * '%Float' have the highest correlation with each other. * We gonna compare each column as a target for Regression session.

1.0.4 Section 2: Simple Exploratory Data Analysis (EDA)

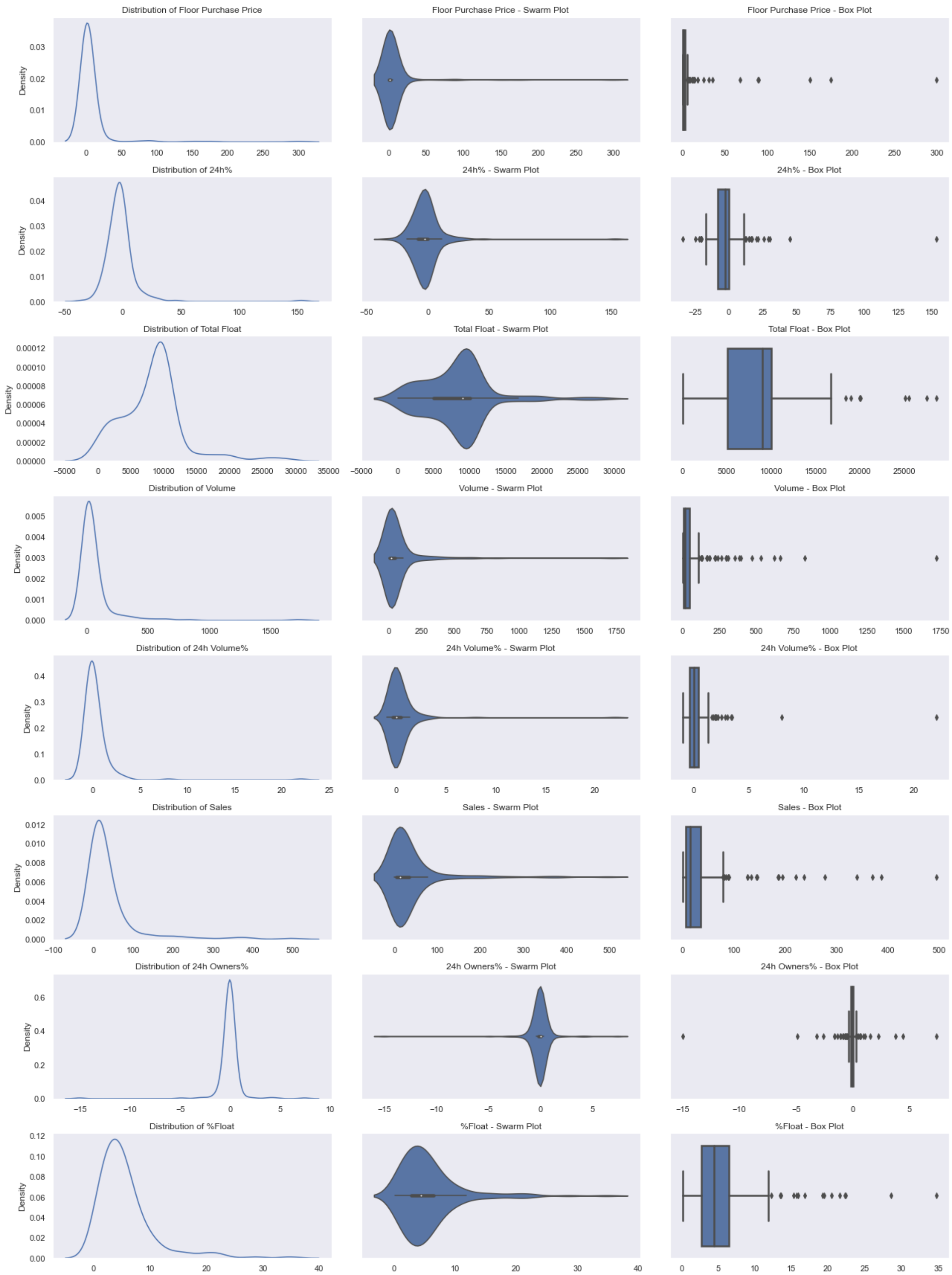
[ ]:
```
## Check for unique variables on each features
## Making sure that all of the columns were numerical

nft_df.nunique()
```

[ ]:
```
Floor Purchase Price    176
24h%                    171
Total Float             161
Volume                   82
24h Volume%             144
Sales                    73
24h Owners%              82
%Float                  185
dtype: int64
```

[ ]:
```
numerical_data_columns = list(nft_df.columns)
```

Numerical Data Columns EDA

[ ]:
```
## Visualize distribution on numerical features
rows = len(numerical_data_columns)
cols = 3

fig = plt.figure(1, (18, rows*3))

i = 0
for feature in numerical_data_columns:

    i += 1
    ax1 = plt.subplot(rows, cols, i)
    sns.kdeplot(data=nft_df, x=feature)
    ax1.set_xlabel(None)
    ax1.set_title(f'Distribution of {feature}')
    plt.tight_layout()

    i += 1
    ax2 = plt.subplot(rows, cols, i)
    sns.violinplot(data=nft_df, x=feature)
    ax2.set_xlabel(None)
    ax2.set_title(f'{feature} - Swarm Plot')
    plt.tight_layout()

    i += 1
    ax3 = plt.subplot(rows, cols, i)
    sns.boxplot(data=nft_df, x=feature, orient='h', linewidth=2.5)
    ax3.set_xlabel(None)
    ax3.set_title(f'{feature} - Box Plot')
    plt.tight_layout()

plt.show()
```



[ ]:
```python
## Find outliers using Tukey's method
def tukey_outliers(x):
```

```
    ## Tukey outliers are based on the boundaries defined by quantiles and IQR
    q1 = np.percentile(x, 25)
    q3 = np.percentile(x, 75)

    iqr = q3 - q1

    lower_boundary = q1 - (iqr * 1.5)
    upper_boundary = q3 + (iqr * 1.5)

    outliers = x[(x < lower_boundary) | (x > upper_boundary)]
    return outliers
```

[ ]:
```
## Calculate the tukey outliers
outlier_dict = {}
for num_feature in numerical_data_columns:
    outliers = tukey_outliers(nft_df[num_feature])
    if len(outliers):
        print(f"-> {num_feature} has {len(outliers)} tukey outliers")
        outlier_dict[num_feature] = outliers
    else:
        print(f"-> {num_feature} doesn't have any tukey outliers.")
        outlier_dict[num_feature] = None
```

```
-> Floor Purchase Price has 26 tukey outliers
-> 24h% has 21 tukey outliers
-> Total Float has 9 tukey outliers
-> Volume has 25 tukey outliers
-> 24h Volume% has 15 tukey outliers
-> Sales has 22 tukey outliers
-> 24h Owners% has 34 tukey outliers
-> %Float has 15 tukey outliers
```

[ ]:
```
## Show the percentage of outliers

for x in numerical_data_columns:
    outliers = nft_df.loc[outlier_dict[x].index]
    print("{} has {}% of outliers".format(
        x, round(len(outliers)/len(nft_df) * 100, 2)))
```

```
Floor Purchase Price has 13.0% of outliers
24h% has 10.5% of outliers
Total Float has 4.5% of outliers
Volume has 12.5% of outliers
24h Volume% has 7.5% of outliers
Sales has 11.0% of outliers
24h Owners% has 17.0% of outliers
%Float has 7.5% of outliers
```

```
[ ]: ## Perform test whether a sample differs from a normal distribution
from scipy.stats import normaltest

ALPHA = 0.05

for col in nft_df:
    stat, p = normaltest(nft_df[col].values)
    print('{}: stat={}, p={}'.format(col, stat, p))
    if p <= ALPHA:
        print('Probably not Gaussian\n')
    else:
        print('Probably Gaussian\n')
```

```
Floor Purchase Price: stat=316.07195614607167, p=2.321917478231033e-69
Probably not Gaussian

24h%: stat=291.31790311159415, p=5.509613060927256e-64
Probably not Gaussian

Total Float: stat=54.298023333540655, p=1.61932508648859e-12
Probably not Gaussian

Volume: stat=284.7421228745872, p=1.4758208276623873e-62
Probably not Gaussian

24h Volume%: stat=354.26211772159263, p=1.182928684697652e-77
Probably not Gaussian

Sales: stat=194.42056191809934, p=6.0549616277049535e-43
Probably not Gaussian

24h Owners%: stat=271.33532249822673, p=1.2030490840542885e-59
Probably not Gaussian

%Float: stat=129.43076004327602, p=7.842735389163618e-29
Probably not Gaussian
```

Numerical Data Columns EDA Short Recap/Conclusion: * '24h Owners%' has the highest percentage of outliers with '17%'. * All of the columns were probably not normally distributed. * We gonna analyze skewness on next section.

1.0.5 Section 3: Feature Engineering

```
[ ]: from sklearn.preprocessing import MinMaxScaler, StandardScaler
```

```
[ ]: ## Create list of scaler
```

```
scalers = [
    (MinMaxScaler(), "MinMaxScaler"),
    (StandardScaler(), "StandardScaler")
]
```

[ ]:
```
## Compare result of skewness after scaled from each scaler
for scaler, scaler_desc in scalers:
    nft_df_fe = nft_df.copy()
    skew_result = []
    for column in numerical_data_columns:
        nft_df_fe[[column]] = scaler.fit_transform(nft_df_fe[[column]])
        skew_result.append({column: nft_df_fe[column].skew()})
    print("Skew Result After " + scaler_desc)
    print(skew_result)
    print("----------")
```

```
Skew Result After MinMaxScaler
[{'Floor Purchase Price': 7.622971987823639}, {'24h%': 6.445260418708285},
{'Total Float': 1.151688867075795}, {'Volume': 6.38053339982749}, {'24h
Volume%': 9.110121232395436}, {'Sales': 3.961280903326318}, {'24h Owners%':
-5.483347390827999}, {'%Float': 2.550361678976997}]
----------
Skew Result After StandardScaler
[{'Floor Purchase Price': 7.622971987823639}, {'24h%': 6.445260418708285},
{'Total Float': 1.151688867075795}, {'Volume': 6.3805333998274865}, {'24h
Volume%': 9.110121232395436}, {'Sales': 3.961280903326317}, {'24h Owners%':
-5.483347390828004}, {'%Float': 2.5503616789769974}]
----------
```

[ ]:
```
## Both were pretty same, so use any of those wouldn't be much problem
nft_df_fe = nft_df.copy()
for column in [numerical_data_columns]:
    nft_df_fe[column] = StandardScaler().fit_transform(nft_df_fe[column])
```

[ ]:
```
## Display statistical value after scaling
nft_df_fe.describe().T
```

[ ]:

```
                      count          mean       std        min       25%  \
Floor Purchase Price  200.0  0.000000e+00  1.002509  -0.238059 -0.224783
24h%                  200.0  0.000000e+00  1.002509  -2.215370 -0.414300
Total Float           200.0 -1.065814e-16  1.002509  -1.732325 -0.674080
Volume                200.0 -1.776357e-17  1.002509  -0.394722 -0.352191
24h Volume%           200.0  1.776357e-17  1.002509  -0.716206 -0.372445
Sales                 200.0  1.776357e-17  1.002509  -0.542862 -0.457982
24h Owners%           200.0  4.440892e-18  1.002509 -10.781321 -0.036346
%Float                200.0  1.776357e-17  1.002509  -1.081014 -0.559105

                           50%       75%        max
```

```
Floor Purchase Price -0.203690 -0.149346  10.357869
24h%                 -0.050903  0.138319  10.727845
Total Float           0.173733  0.377289   4.303667
Volume               -0.285356 -0.101559  10.043712
24h Volume%          -0.164876  0.072304  11.964403
Sales                -0.328817 -0.026202   6.749428
24h Owners%           0.072043  0.102753   5.426465
%Float               -0.227893  0.195154   5.862381
```

[ ]:
```
## Take a quick look of the dataframe
nft_df_fe
```

[ ]:
```
     Floor Purchase Price       24h%  Total Float    Volume  24h Volume%  \
0                2.909453  0.289143     0.376231  3.366274    -0.162190
1                2.168979  0.138319     0.376019  4.623990    -0.547914
2                0.389402 -0.083766     1.731081  1.938432    -0.176127
3                0.228105  0.381852     1.796041  1.415903    -0.333231
4                0.637550  0.110645    -0.140487 -0.394722    -0.164876
..                    …         …           …         …           …
195             -0.225651 -0.553536    -0.666304 -0.376495    -0.385408
196             -0.223992 -0.853800    -0.794108 -0.285356    -0.535771
197             -0.233805 -0.516867     0.378347 -0.376495    -0.302708
198             -0.231731 -0.019424    -0.091397 -0.370419    -0.275142
199             -0.233096  0.138319     0.164423 -0.394722    -0.164876

        Sales  24h Owners%    %Float
0   -0.439530     0.108172  0.101311
1   -0.380483     0.072043  1.602802
2   -0.262390     0.093720  0.131421
3   -0.247628     0.129850  0.506794
4   -0.542862    -0.058024 -1.081014
..        …            …         …
195 -0.483815     0.100946  0.380331
196 -0.070487    -0.036346  0.296023
197 -0.365722    -0.130284  0.083244
198 -0.321436    -0.549388 -0.185739
199 -0.528101     0.072043 -0.990683

[200 rows x 8 columns]
```

[ ]:
```
skew_limit = 0.75
df_skew = nft_df_fe.copy()
skew_vals = df_skew.skew()
```

[ ]:
```
## Display skewness value for each columns
skew_cols = (skew_vals
             .sort_values(ascending=False)
```

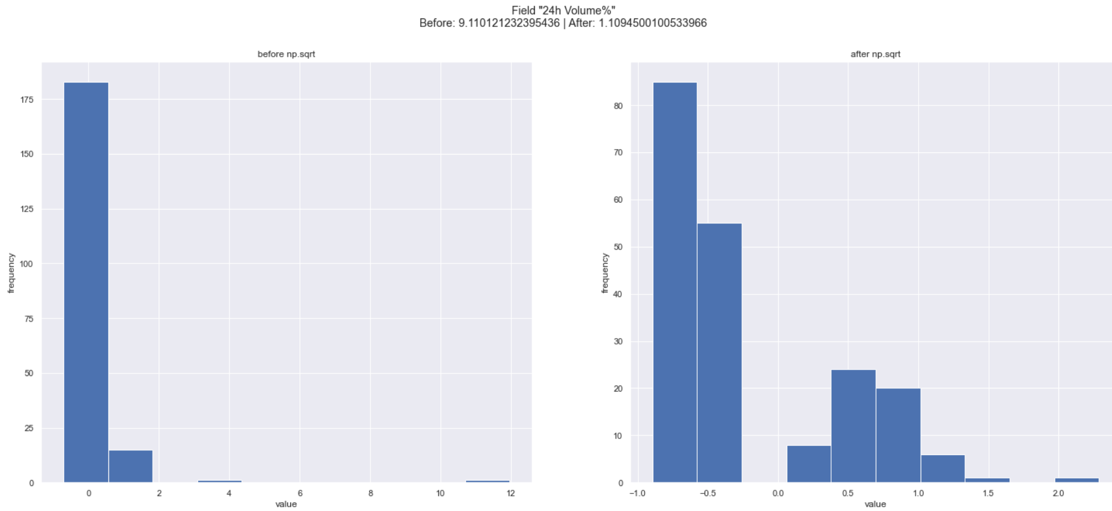
```
             .to_frame()
             .rename(columns={0: 'Skew'})
             .query('abs(Skew) > {}'.format(skew_limit))

skew_cols
```

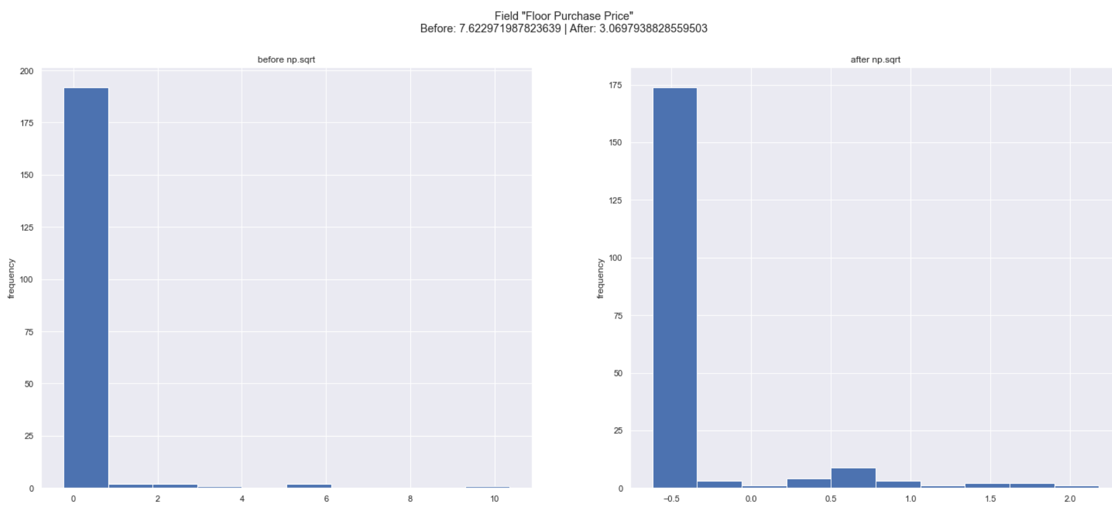
[ ]:

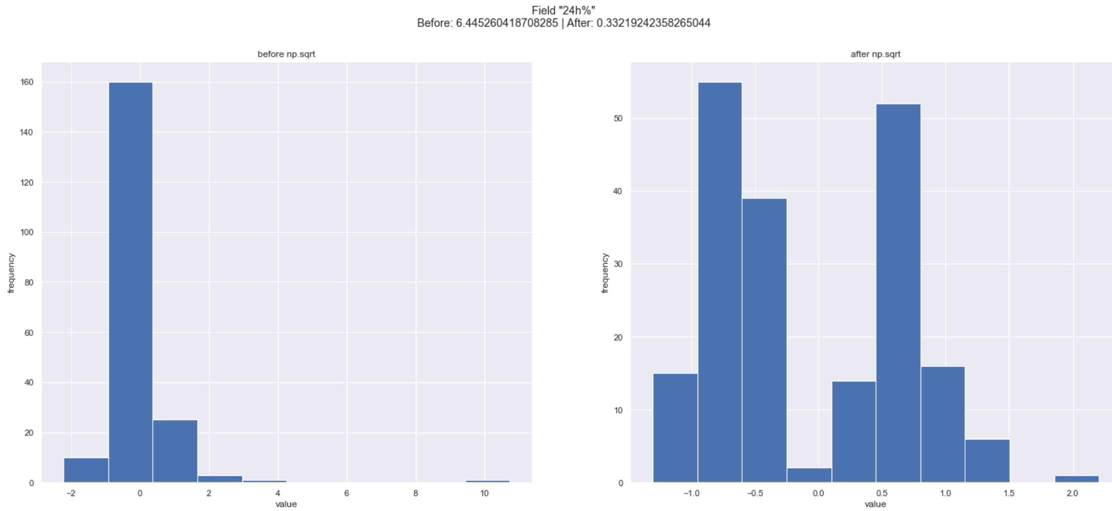
	Skew
24h Volume%	9.110121
Floor Purchase Price	7.622972
24h%	6.445260
Volume	6.380533
Sales	3.961281
%Float	2.550362
Total Float	1.151689
24h Owners%	-5.483347

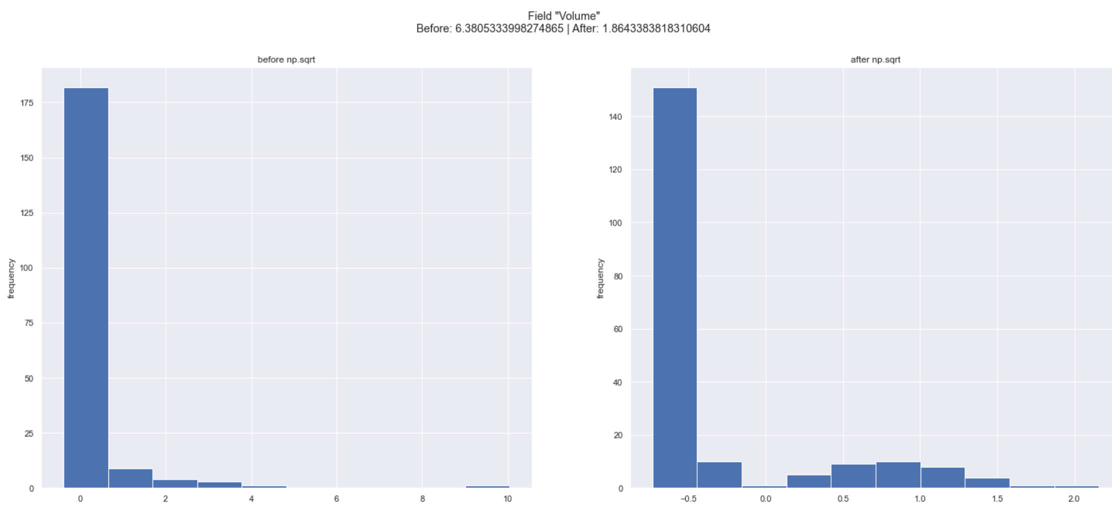
[ ]:

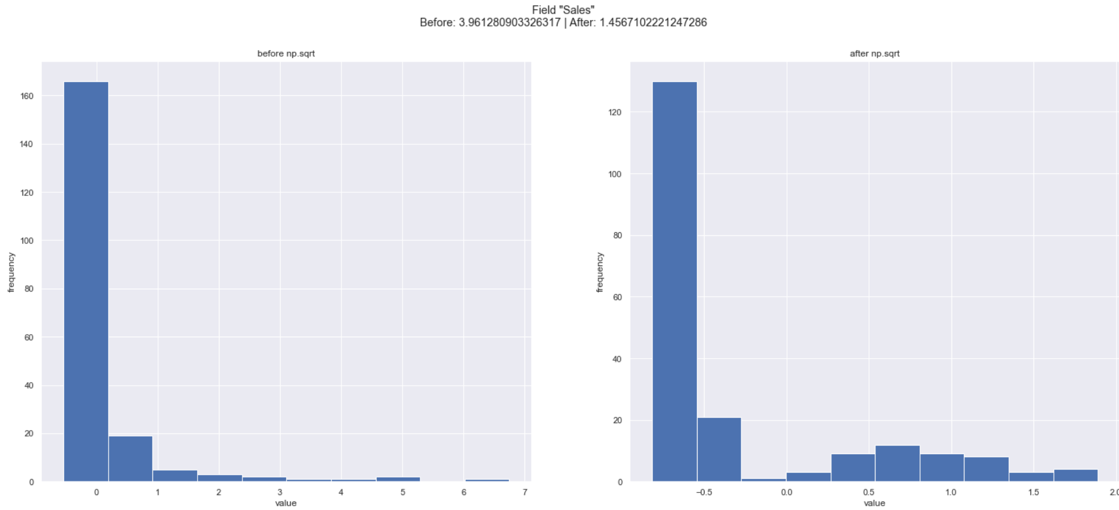
```
## Create before-after tansformation graph
skew_features = skew_cols.index.tolist()
for field in skew_features:
    # Create two "subplots" and a "figure" using matplotlib
    fig, (ax_before, ax_after) = plt.subplots(1, 2, figsize=(25, 10))

    # Create a histogram on the "ax_before" subplot
    df_skew[field].hist(ax=ax_before)

    # after_skew = np.sqrt(df_skew[field] + 0 - min(df_skew[field]))
    after_skew = np.cbrt(df_skew[field])

    # Apply a log transformation (numpy syntax) to this column
    after_skew.hist(ax=ax_after)

    # Formatting of titles etc. for each subplot
    ax_before.set(title='before np.sqrt', ylabel='frequency', xlabel='value')
    ax_after.set(title='after np.sqrt', ylabel='frequency', xlabel='value')
    fig.suptitle('Field "{}"\nBefore: {} | After: {}\n'.format(
        field, df_skew[field].skew(), after_skew.skew()))
```

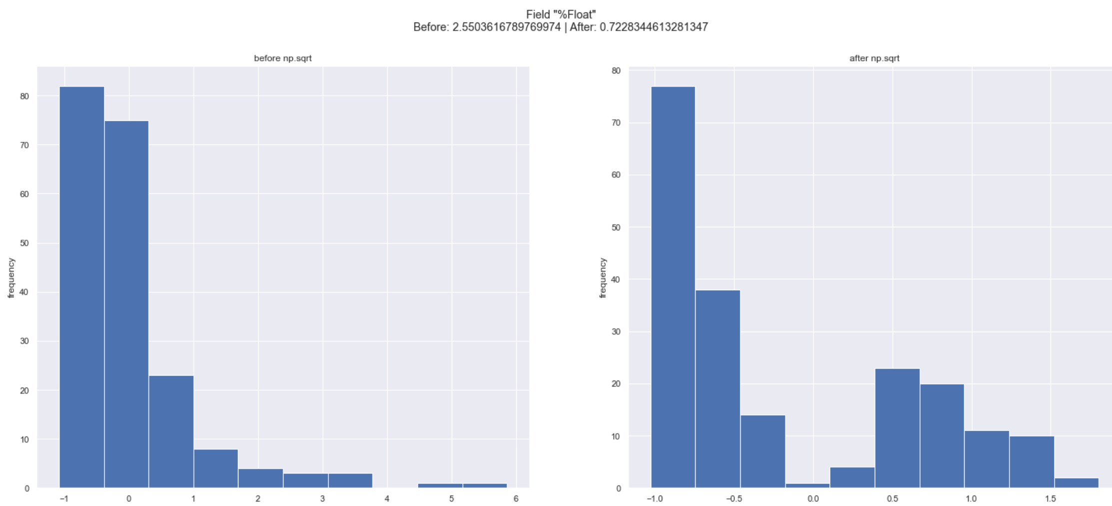

Field "24h Volume%"
Before: 9.110121232395436 | After: 1.1094500100533966
before np.sqrt
after np.sqrt
frequency
value

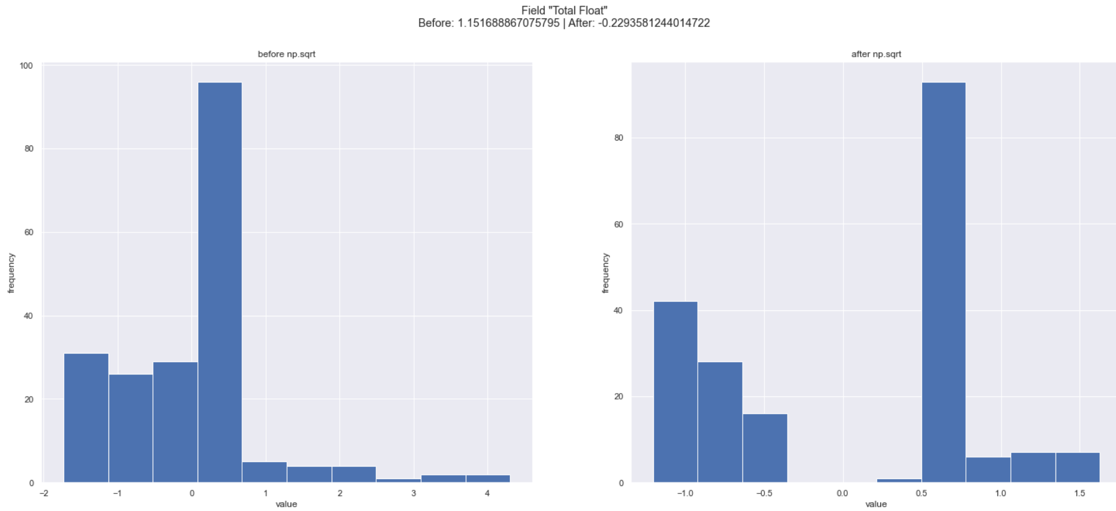

Field "Floor Purchase Price"
Before: 7.622971987823639 | After: 3.0697938828559503
before np.sqrt
after np.sqrt
frequency
value

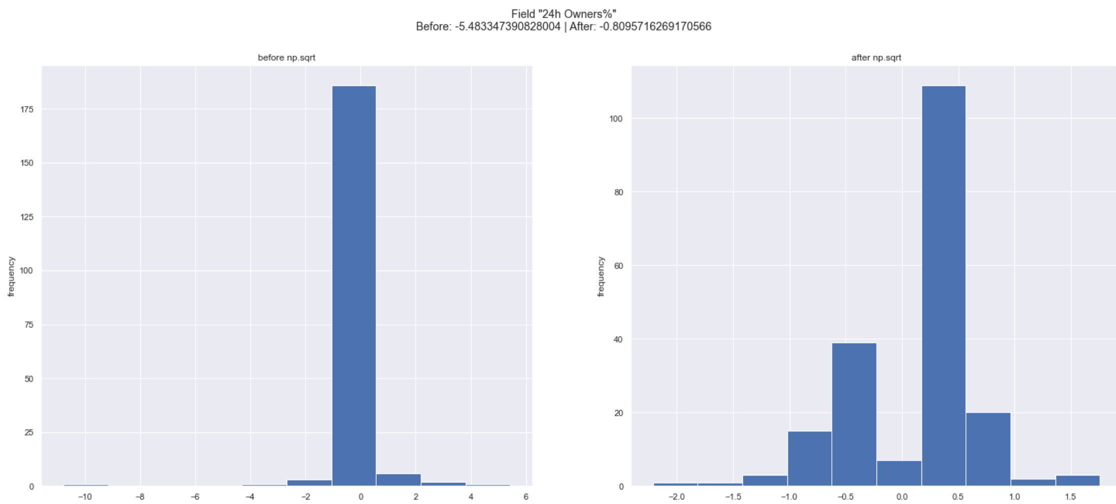

Field "24h%"
Before: 6.445260418708285 | After: 0.33219242358265044
before np.sqrt
after np.sqrt
frequency
value


Field "Volume"
Before: 6.3805333998274865 | After: 1.8643383818310604
before np.sqrt
after np.sqrt
frequency
value


Field "Sales"
Before: 3.961280903326317 | After: 1.4567102221247286
before np.sqrt
after np.sqrt
frequency
value


Field "%Float"
Before: 2.5503616789769974 | After: 0.7228344613281347
before np.sqrt
after np.sqrt
frequency
value





[ ]:
```
## Apply transformation to the feature
for column in numerical_data_columns:
    df_skew[column] = np.cbrt(df_skew[column])
```

[ ]:
```
df_skew
```

[ ]:

	Floor Purchase Price	24h%	Total Float	Volume	24h Volume%	\
0	1.427591	0.661258	0.721913	1.498706	-0.545350	
1	1.294449	0.517163	0.721777	1.665990	-0.818284	
2	0.730241	-0.437545	1.200713	1.246858	-0.560543	
3	0.611006	0.725490	1.215548	1.122909	-0.693290	
4	0.860673	0.480076	-0.519851	-0.733551	-0.548343	

```
..                  …         …         …         …         …
195                -0.608807 -0.821073    -0.873422 -0.722082    -0.727735
196                -0.607311 -0.948678    -0.926033 -0.658358    -0.812194
197                -0.616053 -0.802527     0.723264 -0.722082    -0.671441
198                -0.614226 -0.268810    -0.450447 -0.718176    -0.650407
199                -0.615429  0.517163     0.547840 -0.733551    -0.548343

        Sales  24h Owners%    %Float
0   -0.760320     0.476473  0.466178
1   -0.724623     0.416099  1.170289
2   -0.640200     0.454232  0.508418
3   -0.627962     0.506385  0.797279
4   -0.815762    -0.387142 -1.026306
..        …           …         …
195 -0.785043     0.465619  0.724526
196 -0.413082    -0.331249  0.666461
197 -0.715128    -0.506948  0.436635
198 -0.685012    -0.819017 -0.570560
199 -0.808299     0.416099 -0.996885

[200 rows x 8 columns]
```

Feature Transformation Short Recap/Conclusion: * Because there's negative value on skewed features ('Oldpeak'), we gonna use Cube Root as Feature Transformation approach. * After Feature Transformation with Cube Root method, all of the skewness seems getting close to 0.75.

1.0.6 Section 4: Regression

[ ]:
```
df_skew.head()
```

[ ]:
```
   Floor Purchase Price       24h%  Total Float    Volume  24h Volume%  \
0              1.427591  0.661258     0.721913  1.498706    -0.545350
1              1.294449  0.517163     0.721777  1.665990    -0.818284
2              0.730241 -0.437545     1.200713  1.246858    -0.560543
3              0.611006  0.725490     1.215548  1.122909    -0.693290
4              0.860673  0.480076    -0.519851 -0.733551    -0.548343

      Sales  24h Owners%    %Float
0 -0.760320     0.476473  0.466178
1 -0.724623     0.416099  1.170289
2 -0.640200     0.454232  0.508418
3 -0.627962     0.506385  0.797279
4 -0.815762    -0.387142 -1.026306
```

[ ]:
```
## Split the Training and Test set with KFold
## We gonna make 3 type of Training and Test set: %Float, Sales, and Volume

from sklearn.model_selection import KFold
```

```
X_float = df_skew.drop(columns=["%Float"])
y_float = df_skew["%Float"]

X_sales = df_skew.drop(columns=["Sales"])
y_sales = df_skew["Sales"]

X_volume = df_skew.drop(columns=["Volume"])
y_volume = df_skew["Volume"]

kf = KFold(shuffle=True, random_state=72018, n_splits=4)
```

[ ]:
```
## Import neccessary libraries for modelling

from sklearn.preprocessing import MinMaxScaler, StandardScaler,␣
 ↪PolynomialFeatures
from sklearn.pipeline import make_pipeline, Pipeline
from sklearn.model_selection import GridSearchCV

from sklearn.linear_model import Lasso, Ridge
from sklearn.metrics import r2_score, mean_squared_error

import datetime
```

[ ]:
```
## Create list of transformers

transformers = [
    (PolynomialFeatures(degree=1), "PolynomialFeatures (Degree 1)"),
    (PolynomialFeatures(degree=2), "PolynomialFeatures (Degree 2)"),
    (PolynomialFeatures(degree=3), "PolynomialFeatures (Degree 3)")
]
```

[ ]:
```
## Create list of models

models = [
    Lasso(max_iter=1000),
    Ridge(max_iter=1000)
]
```

[ ]:
```
search_space_dict = {}

search_space_dict['Lasso'] = {
    'lasso__alpha': np.geomspace(0.001, 0.1, 50)
}

search_space_dict['Ridge'] = {
    'ridge__alpha': np.geomspace(0.001, 0.1, 50)
```

```
}
```

```
[ ]: ## Create pipeline matrix

pipelines_matrix = {}

for transformer, transformer_desc in transformers:
    pipelines_matrix[transformer_desc] = {}
    print(transformer_desc)
    for model in models:
        print("            ", model.__class__.__name__)
        pipelines_matrix[transformer_desc][model.__class__.__name__] =␣
 ↪make_pipeline(
            transformer, model)
```

```
PolynomialFeatures (Degree 1)
             Lasso
             Ridge
PolynomialFeatures (Degree 2)
             Lasso
             Ridge
PolynomialFeatures (Degree 3)
             Lasso
             Ridge
```

```
[ ]: ## Create a function for performing cross validation of all algorithms
## Fuction will return a dataframe with the result from each pipeline

def cross_validator(X_train, y_train, pipelines_matrix):
    i = 0
    for transformer in pipelines_matrix:
        print("----------------------", transformer)
        for model in pipelines_matrix[transformer]:
            i += 1
            print("     +++++++", model)
            startT = datetime.datetime.now()

            pipeline = pipelines_matrix[transformer][model]

            search_space = search_space_dict[model]
            regressor = GridSearchCV(pipeline,
                                   search_space,
                                   scoring='neg_root_mean_squared_error',
                                   cv=kf)
            regressor.fit(X_train, y_train)

            print("          rmse: ", regressor.best_score_)
```

```
            headers = ['transformer', 'model',
                       'rmse', 'best_params']
            dfResultsTemp = pd.DataFrame(columns=headers)
            dfResultsTemp.loc[0] = [
                transformer, model, regressor.best_score_, regressor.
↪best_params_]

            print("             exec time:", datetime.datetime.now() -
                  startT, datetime.datetime.now())

            if i == 1:
                data_concat = dfResultsTemp.copy()
            else:
                data_concat = pd.concat([data_concat, dfResultsTemp])

    return data_concat
```

GridSearch with '%Float' as Target

[ ]:
```
grid_search_df = cross_validator(X_float, y_float, pipelines_matrix)
```

```
---------------------- PolynomialFeatures (Degree 1)
     +++++++ Lasso
          rmse:  -0.7023030330440436
             exec time: 0:00:00.948267 2022-03-03 17:41:10.539171
     +++++++ Ridge
          rmse:  -0.7039029939697747
             exec time: 0:00:00.937164 2022-03-03 17:41:11.478338
---------------------- PolynomialFeatures (Degree 2)
     +++++++ Lasso
          rmse:  -0.6909864058967283
             exec time: 0:00:00.947069 2022-03-03 17:41:12.426411
     +++++++ Ridge
          rmse:  -0.7992305748273081
             exec time: 0:00:01.025774 2022-03-03 17:41:13.452185
---------------------- PolynomialFeatures (Degree 3)
     +++++++ Lasso
          rmse:  -0.68868226478335
             exec time: 0:00:02.291095 2022-03-03 17:41:15.745319
     +++++++ Ridge
          rmse:  -1.3486868925269526
             exec time: 0:00:01.280983 2022-03-03 17:41:17.027304
```

[ ]:
```
grid_search_df.sort_values(by=['rmse'], ascending=False, ignore_index=True)
```

[ ]:
```
                     transformer  model      rmse  \
0  PolynomialFeatures (Degree 3)  Lasso -0.688682
```

```
1  PolynomialFeatures (Degree 2)  Lasso -0.690986
2  PolynomialFeatures (Degree 1)  Lasso -0.702303
3  PolynomialFeatures (Degree 1)  Ridge -0.703903
4  PolynomialFeatures (Degree 2)  Ridge -0.799231
5  PolynomialFeatures (Degree 3)  Ridge -1.348687

                              best_params
0   {'lasso__alpha': 0.01151395399326447}
1  {'lasso__alpha': 0.022229964825261943}
2  {'lasso__alpha': 0.013894954943731374}
3                   {'ridge__alpha': 0.1}
4                   {'ridge__alpha': 0.1}
5                   {'ridge__alpha': 0.1}
```

[ ]:
```
pipeline = Pipeline(steps=[
    ('transformer', PolynomialFeatures(degree=3)),
    ('model', Lasso(alpha=0.01151395399326447, max_iter=1000))])
```

[ ]:
```
pipeline.fit(X_float, y_float)
```

[ ]:
```
Pipeline(steps=[('transformer', PolynomialFeatures(degree=3)),
                ('model', Lasso(alpha=0.01151395399326447))])
```

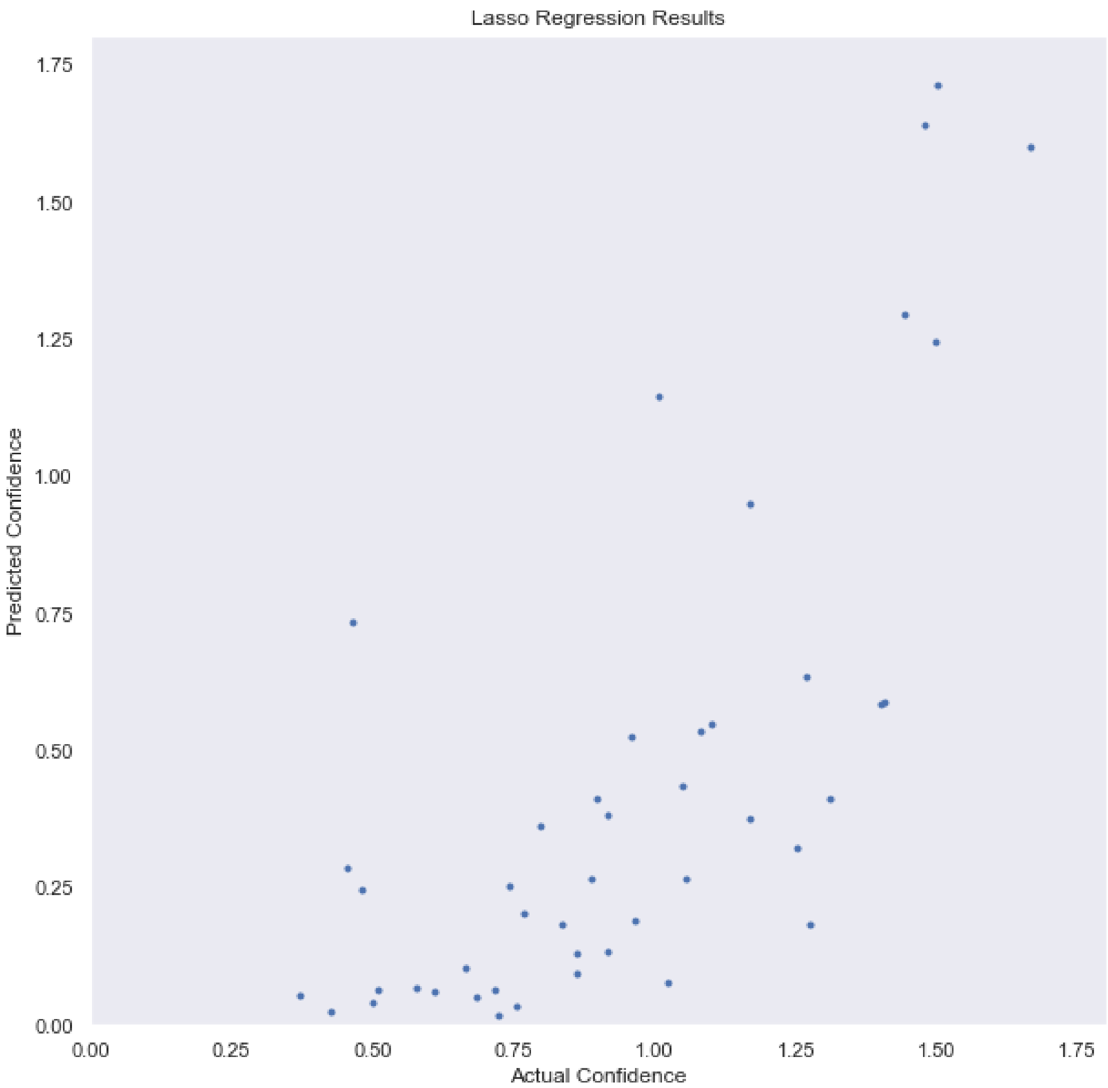
[ ]:
```
y_predict = pipeline.predict(X_float)
print(
    f"RMSE Score for Lasso Regression: {mean_squared_error(y_float, y_predict,␣
 ↪squared=False)}")
print(f"R2 Score for Lasso Regression: {r2_score(y_float, y_predict)}")
```

```
RMSE Score for Lasso Regression: 0.5593478150731449
R2 Score for Lasso Regression: 0.5153900992431408
```

[ ]:
```
f = plt.figure(figsize=(10, 10))
ax = plt.axes()

ax.plot(y_float, pipeline.predict(X_float),
        marker='o', ls='', ms=3.0)

lim = (0, y_float.max())

ax.set(xlabel='Actual Confidence',
       ylabel='Predicted Confidence',
       xlim=lim,
       ylim=lim,
       title='Lasso Regression Results')
```

[ ]:
```
[Text(0.5, 0, 'Actual Confidence'),
 Text(0, 0.5, 'Predicted Confidence'),
```

```
 (0.0, 1.8031202068813825),
 (0.0, 1.8031202068813825),
 Text(0.5, 1.0, 'Lasso Regression Results')]
```



GridSearch with 'Sales' as Target

```
[ ]: grid_search_df = cross_validator(X_sales, y_sales, pipelines_matrix)
```

```
---------------------- PolynomialFeatures (Degree 1)
     +++++++ Lasso
          rmse:  -0.5025950022194425
             exec time: 0:00:00.947502 2022-03-03 17:41:18.560089
     +++++++ Ridge
          rmse:  -0.5032805392463919
             exec time: 0:00:00.946067 2022-03-03 17:41:19.506156
```

```
---------------------- PolynomialFeatures (Degree 2)
     +++++++ Lasso
          rmse:  -0.4574792458012503
             exec time: 0:00:00.972455 2022-03-03 17:41:20.480611
     +++++++ Ridge
          rmse:  -0.5340761238077115
             exec time: 0:00:01.001784 2022-03-03 17:41:21.482395
---------------------- PolynomialFeatures (Degree 3)
     +++++++ Lasso
          rmse:  -0.455916357434325
             exec time: 0:00:01.951274 2022-03-03 17:41:23.433669
     +++++++ Ridge
          rmse:  -0.8766472571548042
             exec time: 0:00:01.261778 2022-03-03 17:41:24.696449
```

[ ]:
```
grid_search_df.sort_values(by=['rmse'], ascending=False, ignore_index=True)
```

[ ]:
```
                    transformer  model      rmse  \
0  PolynomialFeatures (Degree 3)  Lasso -0.455916
1  PolynomialFeatures (Degree 2)  Lasso -0.457479
2  PolynomialFeatures (Degree 1)  Lasso -0.502595
3  PolynomialFeatures (Degree 1)  Ridge -0.503281
4  PolynomialFeatures (Degree 2)  Ridge -0.534076
5  PolynomialFeatures (Degree 3)  Ridge -0.876647

                               best_params
0   {'lasso__alpha': 0.016768329368110076}
1   {'lasso__alpha': 0.020235896477251564}
2  {'lasso__alpha': 0.0071968567300115215}
3                   {'ridge__alpha': 0.1}
4                   {'ridge__alpha': 0.1}
5                   {'ridge__alpha': 0.1}
```

[ ]:
```
pipeline = Pipeline(steps=[
    ('transformer', PolynomialFeatures(degree=3)),
    ('model', Lasso(alpha=0.016768329368110076, max_iter=1000))])
```

[ ]:
```
pipeline.fit(X_sales, y_sales)
```

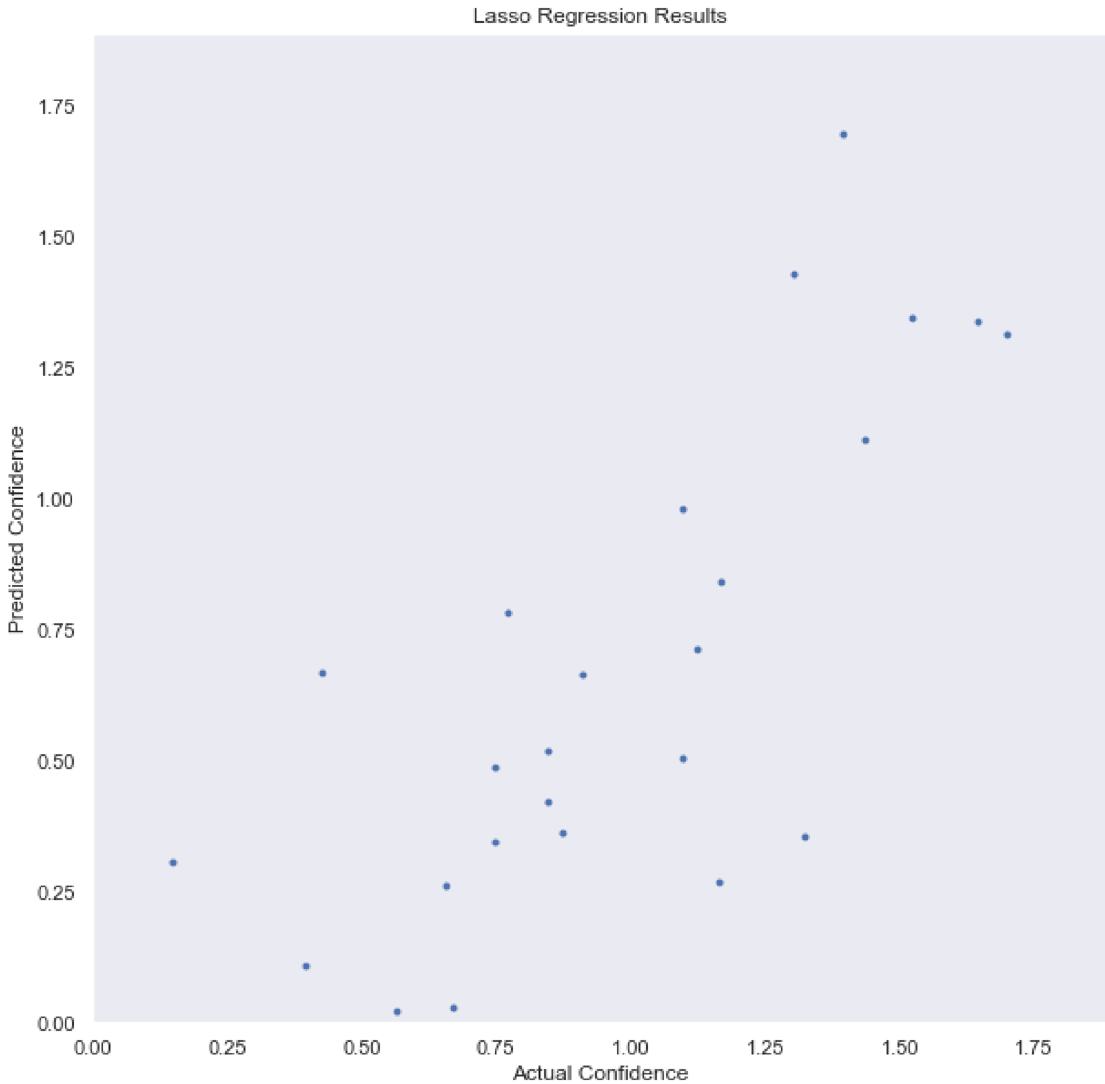
[ ]:
```
Pipeline(steps=[('transformer', PolynomialFeatures(degree=3)),
                ('model', Lasso(alpha=0.016768329368110076))])
```

[ ]:
```
y_predict = pipeline.predict(X_sales)
print(
    f"RMSE Score for Lasso Regression: {mean_squared_error(y_sales, y_predict,␣
 ↪squared=False)}")
print(f"R2 Score for Lasso Regression: {r2_score(y_sales, y_predict)}")
```

```
RMSE Score for Lasso Regression: 0.39144298699421104
R2 Score for Lasso Regression: 0.6916537936265998
```

[ ]:
```
f = plt.figure(figsize=(10, 10))
ax = plt.axes()

ax.plot(y_sales, pipeline.predict(X_sales),
        marker='o', ls='', ms=3.0)

lim = (0, y_sales.max())

ax.set(xlabel='Actual Confidence',
       ylabel='Predicted Confidence',
       xlim=lim,
       ylim=lim,
       title='Lasso Regression Results')
```

[ ]:
```
[Text(0.5, 0, 'Actual Confidence'),
 Text(0, 0.5, 'Predicted Confidence'),
 (0.0, 1.8898281586702523),
 (0.0, 1.8898281586702523),
 Text(0.5, 1.0, 'Lasso Regression Results')]
```



GridSearch with ‘Volume’ as Target

```
[ ]: grid_search_df = cross_validator(X_volume, y_volume, pipelines_matrix)
```

```
---------------------- PolynomialFeatures (Degree 1)
     +++++++ Lasso
          rmse:  -0.475885736625466624
             exec time: 0:00:00.915372 2022-03-03 17:41:26.119464
     +++++++ Ridge
          rmse:  -0.47654348250383327
             exec time: 0:00:00.886080 2022-03-03 17:41:27.007550
---------------------- PolynomialFeatures (Degree 2)
     +++++++ Lasso
          rmse:  -0.391776427371398856
             exec time: 0:00:00.980460 2022-03-03 17:41:27.989009
```

```
    +++++++ Ridge
        rmse:  -0.4875678236371811
            exec time: 0:00:01.004929 2022-03-03 17:41:28.995977
---------------------- PolynomialFeatures (Degree 3)
    +++++++ Lasso
        rmse:  -0.3407160459127194
            exec time: 0:00:01.797005 2022-03-03 17:41:30.792982
    +++++++ Ridge
        rmse:  -0.7271183882052082
            exec time: 0:00:01.260016 2022-03-03 17:41:32.053998
```

[ ]:
```
grid_search_df.sort_values(by=['rmse'], ascending=False, ignore_index=True)
```

[ ]:
```
                    transformer  model      rmse  \
0  PolynomialFeatures (Degree 3)  Lasso -0.340716
1  PolynomialFeatures (Degree 2)  Lasso -0.391776
2  PolynomialFeatures (Degree 1)  Lasso -0.475886
3  PolynomialFeatures (Degree 1)  Ridge -0.476543
4  PolynomialFeatures (Degree 2)  Ridge -0.487568
5  PolynomialFeatures (Degree 3)  Ridge -0.727118

                           best_params
0  {'lasso__alpha': 0.007196856730011521}
1   {'lasso__alpha': 0.013894954943731374}
2   {'lasso__alpha': 0.005428675439323859}
3                  {'ridge__alpha': 0.1}
4                  {'ridge__alpha': 0.1}
5                  {'ridge__alpha': 0.1}
```

[ ]:
```
pipeline = Pipeline(steps=[
    ('transformer', PolynomialFeatures(degree=3)),
    ('model', Lasso(alpha=0.007196856730011521, max_iter=1000))])
```

[ ]:
```
pipeline.fit(X_volume, y_volume)
```

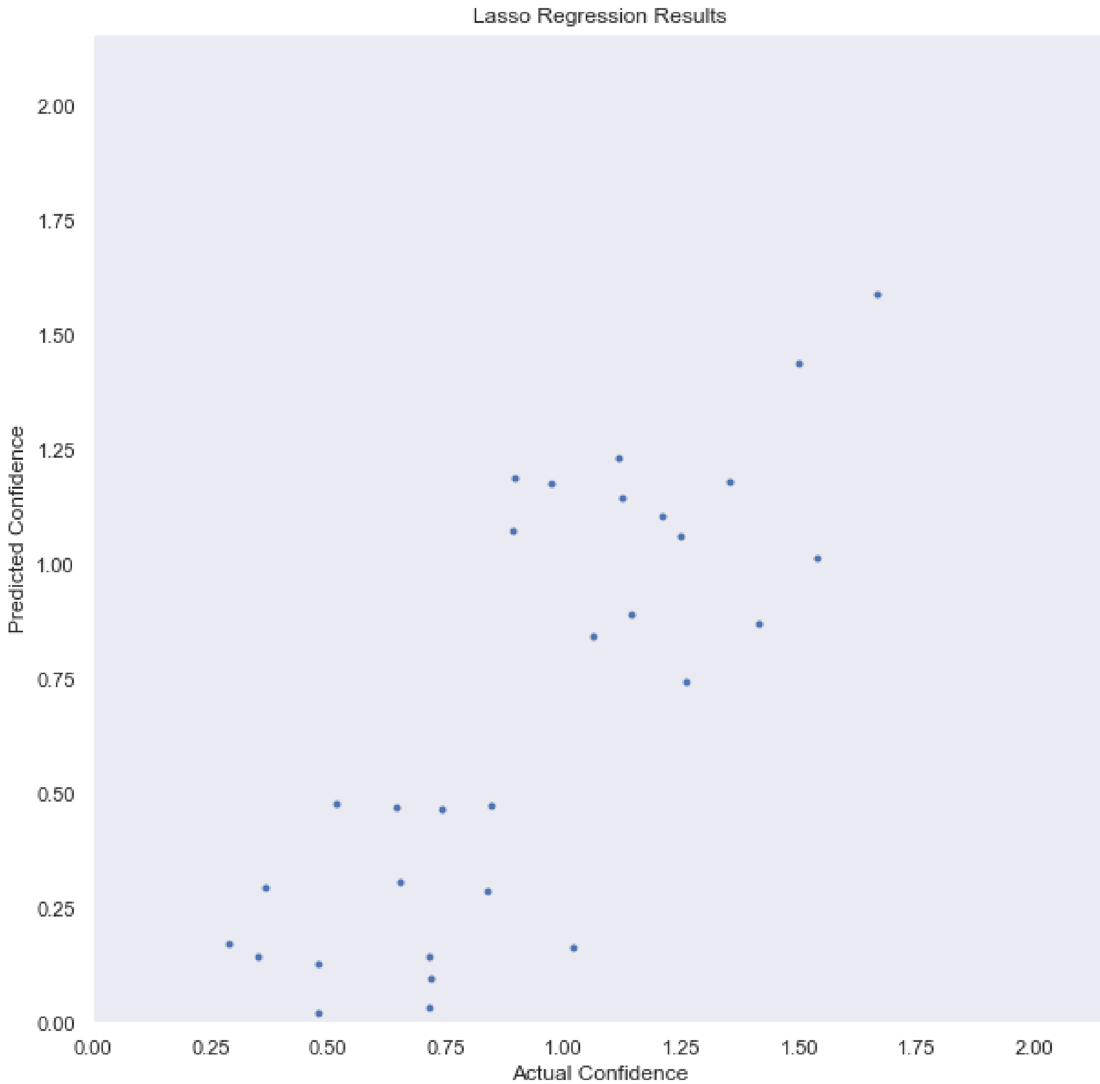
[ ]:
```
Pipeline(steps=[('transformer', PolynomialFeatures(degree=3)),
                ('model', Lasso(alpha=0.007196856730011521))])
```

[ ]:
```
y_predict = pipeline.predict(X_volume)
print(
    f"RMSE Score for Lasso Regression: {mean_squared_error(y_volume, y_predict,␣
 ↪squared=False)}")
print(f"R2 Score for Lasso Regression: {r2_score(y_volume, y_predict)}")
```

```
RMSE Score for Lasso Regression: 0.2602602177765632
R2 Score for Lasso Regression: 0.8328034222859398
```

[ ]:
```
f = plt.figure(figsize=(10, 10))
ax = plt.axes()

ax.plot(y_volume, pipeline.predict(X_volume),
        marker='o', ls='', ms=3.0)

lim = (0, y_volume.max())

ax.set(xlabel='Actual Confidence',
       ylabel='Predicted Confidence',
       xlim=lim,
       ylim=lim,
       title='Lasso Regression Results')
```

[ ]:
```
[Text(0.5, 0, 'Actual Confidence'),
 Text(0, 0.5, 'Predicted Confidence'),
 (0.0, 2.157569313449022),
 (0.0, 2.157569313449022),
 Text(0.5, 1.0, 'Lasso Regression Results')]
```



Final Model Evaluation Short Recap/Conclusion: * After building model for 3 different target, 'Volume' got the best score with the highest R2 Score

---

2022 | Dimas Adrian Mukti / @berodimas