Stock Market Data Cleaning, Profiling and Exploration

FINISHED

We're going to perform cleansing, profiling and exploratory data analysis on the real stock market data. We will be using the Huge Stock Market Data from NYSE, NASDAQ, and NYSE MKT (https://www.kaggle.com/datasets/borismarjanovic/price-volume-data-for-all-us-stocks-etfs?datasetId=4538&sortBy=voteCount).

Took 0 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:13:41 AM.

Source Data Folder Exploration

FINISHED

In this step, we will explore the data folder to understand the organization of the data physically. This will also give us the insights into how should be write data reader.

Took 0 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:13:41 AM.

```
val dataFolder = sc.wholeTextFiles("loudacre/Stocks")
```

FINISHED

```
dataFolder: org.apache.spark.rdd.RDD[(String, String)] = loudacre/Stocks MapPartitionsRDD[647] at wholeTextFiles at <console>:33
```

Took 0 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:13:41 AM.

```
dataFolder.count()
```

SPARK JOB (http://nyu-dataproc-sw-zwd3.c.hpc-dataproc-19b8.internal:42621/jobs/job?id=189) FINISHED

```
res62: Long = 7195
```

Took 10 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:13:51 AM.

```
dataFolder.take(5).foreach(println)
```

SPARK JOB (http://nyu-dataproc-sw-zwd3.c.hpc-dataproc-19b8.internal:42621/jobs/job?id=190) FINISHED

```
(hdfs://nyu-dataproc-m/user/hd2225_nyu_edu/loudacre/Stocks/aaap.us.txt,Date,Open,High,Low,Close,Volume,OpenInt
2015-11-11,18.5,25.9,18,24.5,1584600,0
2015-11-12,24.25,27.12,22.5,25,83000,0
2015-11-13,25.47,26.2,24.55,25.26,67300,0
2015-11-16,25.01,26.19,24.13,25.02,16900,0
2015-11-17,24.46,25.51,24.38,24.62,25900,0
2015-11-18,24.62,26.31,24.06,25,111100,0
2015-11-19,24.85,26,24.71,25.9,113100,0
2015-11-20,26,27.01,25.1,25.2,60300,0
2015-11-23,25.6,25.6,25,25.15,59700,0
2015-11-24,25.21,27,25.09,25.62,56000,0
2015-11-25,26.49,26.49,25.19,25.8,11800,0
2015-11-27,25.37,26.5,25.01,26.5,15400,0
2015-11-30,27.67,28.08,25.21,26.44,33100,0
2015-12-01,26.5,27.5,26.11,26.58,96226,0
2015-12-02,26.2,27.94,25.995,26.44,53579,0
2015-12-03,26.03,26.99,26.03,26.64,6266,0
2015-12-04,26.21,26.64,26.02,26.5,8000,0
```

Took 1 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:13:52 AM.

Reading Data

FINISHED

In this step, we will read all of the data from the source as DataFrame. Since the data is CSV, we will try to ensure that we infer the schema of the source data.

Took 0 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:13:52 AM.

Reading input data as DataFrame

SPARK JOB FINISHED

```
val input_path = "loudacre/Stocks"
val input_data = spark.read.format("csv")
  .option("header", "true")
  .option("multiLine", "true")
  .option("inferSchema", "true")
  .option("escape", "\"")
  .load(input_path)
```

```
input_path: String = loudacre/Stocks
input_data: org.apache.spark.sql.DataFrame = [Date: string, Open: double ... 5 more fields]
```

Took 2 min 34 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:16:26 AM.

```
z.show(input_data)
```

SPARK JOB (http://nyu-dataproc-sw-zwd3.c.hpc-dataproc-19b8.internal:42621/jobs/job?id=193) FINISHED

settings

Date	Open	High	Low	Close	Volume
1962-01-02	0.6277	0.6362	0.6201	0.6201	2575579
1962-01-03	0.6201	0.6201	0.6122	0.6201	1764749
1962-01-04	0.6201	0.6201	0.6037	0.6122	2194010
1962-01-05	0.6122	0.6122	0.5798	0.5957	3255244
1962-01-08	0.5957	0.5957	0.5716	0.5957	3696430
1962-01-09	0.5957	0.6037	0.5878	0.5957	2778285
1962-01-10	0.5957	0.6037	0.5957	0.5957	2337096
1962-01-11	0.5957	0.5957	0.5878	0.5957	1943605

Output is truncated to 1000 rows. Learn more about **zeppelin.spark.maxResult**

Took 0 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:16:26 AM.

Finding Number of Columns in Data

FINISHED

```
input_data.columns.length
```

```
res65: Int = 7
```

Took 0 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:16:26 AM.

Exploring Data Schema

FINISHED

```
input_data.printSchema
```

```
root
 |-- Date: string (nullable = true)
 |-- Open: double (nullable = true)
 |-- High: double (nullable = true)
 |-- Low: double (nullable = true)
 |-- Close: double (nullable = true)
 |-- Volume: long (nullable = true)
 |-- OpenInt: integer (nullable = true)
```

Took 0 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:16:27 AM.

Understanding Data Schema

FINISHED

Column	Definition
Date	Date is for the date of trading day.
Open	Open stands for open price on the trading date.
High	High stands for high price on the trading date.
Low	Low stands for low price on the trading date.
Close	Close stands for clsoe price on the trading date.
Volume	Volume stands for the trading volume of shares on the trading date.

Took 0 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:16:27 AM.

Data Summary And Statistics

FINISHED

In this step, we will try to get some insights into the statistics of data columns such as min, max, avg, etc. We will also try to understand Inter Quartile Range of the columns.

Took 0 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:16:27 AM.

Data Summary, Statistics & IQR

SPARK JOB FINISHED

```
val summary = input_data.describe()
z.show(summary)
```

settings

summary	Date	Open	High	Low	Close
count	14887665	14887665	14887665	14887665	14887665
mean	null	30385.382295750554	31212.467319553587	29361.760762006124	30245.21834669
stddev	null	4202500.953751057	4323485.549288387	4046981.0035713837	4180590.065664
min	1962-01-02	0.0	0.004	-1.0	0.0037
max	2017-11-10	1.423712891E9	1.44204863645E9	1.36211784398E9	1.43798624044E

```
summary: org.apache.spark.sql.DataFrame = [summary: string, Date: string ... 6 more fields]
```

Took 1 min 49 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:18:16 AM.

```
z.show(input_data.summary())
```

SPARK JOB FINISHED

settings

summary	Date	Open	High	Low	Close
count	14887665	14887665	14887665	14887665	14887665
mean	null	30385.38229575055	31212.46731955359	29361.76076200612	30245.21834669
stddev	null	4202500.953751057	4323485.549288387	4046981.0035713827	4180590.065664
min	1962-01-02	0.0	0.004	-1.0	0.0037
25%	null	7.75	7.8827	7.6093	7.75
50%	null	15.679	15.917	15.43	15.679
75%	null	28.897	29.273	28.497	28.893
max	2017-11-10	1.423712891E9	1.44204863645E9	1.36211784398E9	1.43798624044E

Took 3 min 50 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:22:06 AM.

Data Cleaning

FINISHED

In this step, we will perform data cleaning, we will try to find rows, columns and values that don't align with the data format. We will perform several steps to ensure that the data is standardized.

Took 0 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:22:06 AM.

Function to find null values distribution

FINISHED

```
import org.apache.spark.sql.DataFrame
import org.apache.spark.sql.functions._
import org.apache.spark.sql.types._
import scala.collection.mutable.ArrayBuffer

def getNullCount(df: DataFrame): DataFrame = {
    val df_columns = df.columns
    val count_buffer = ArrayBuffer[(String, Long)]()
    for(cur_column <- df_columns) {
        val nullCondition = col(cur_column).isNull
        val nullCount = df.select(col(cur_column)).filter(nullCondition).count()
        count_buffer.append((cur_column, nullCount))
    }
    val null_df = spark.createDataFrame(count_buffer).toDF("column", "null_count")
    null_df
}
```

```
import org.apache.spark.sql.DataFrame
import org.apache.spark.sql.functions._
import org.apache.spark.sql.types._
import scala.collection.mutable.ArrayBuffer
getNullCount: (df: org.apache.spark.sql.DataFrame)org.apache.spark.sql.DataFrame
```

Took 1 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:22:07 AM.

Function to find NaN values distribution

FINISHED

```
def getNanCount(df: DataFrame): DataFrame = {
    val df_columns = df.columns
    val count_buffer = ArrayBuffer[(String, Long)]()
    for(cur_column <- df_columns) {
        val nanCondition = col(cur_column).isNaN
        val nanCount = df.select(col(cur_column)).filter(nanCondition).count()
        count_buffer.append((cur_column, nanCount))
    }
    val nan_df = spark.createDataFrame(count_buffer).toDF("column", "nan_count")
    nan_df
}
```

```
getNanCount: (df: org.apache.spark.sql.DataFrame)org.apache.spark.sql.DataFrame
```

Took 0 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:22:07 AM.

Function to find blank values distribution

FINISHED

```
def getBlankCount(df: DataFrame): DataFrame = {
    val df_columns = df.columns
    val count_buffer = ArrayBuffer[(String, Long)]()
    for(cur_column <- df_columns) {
        val blankCount = df.select(col(cur_column)).filter(col(cur_column) === " ").count()
        count_buffer.append((cur_column, blankCount))
    }
    val blank_df = spark.createDataFrame(count_buffer).toDF("column", "blank_count")
    blank_df
}
```

```
getBlankCount: (df: org.apache.spark.sql.DataFrame)org.apache.spark.sql.DataFrame
```

Took 0 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:22:07 AM.

Function to find Zero values distribution

FINISHED

```
def getZeroCount(df: DataFrame): DataFrame = {
    val df_columns = df.columns
    val count_buffer = ArrayBuffer[(String, Long)]()
    for(cur_column <- df_columns) {
        val nullCondition = col(cur_column).isNull
        val zeroCount = df.select(col(cur_column)).filter(col(cur_column) === 0).count()
        count_buffer.append((cur_column, zeroCount))
    }
    val count_df = spark.createDataFrame(count_buffer).toDF("column", "zero_count")
    count_df
}
```

```
getZeroCount: (df: org.apache.spark.sql.DataFrame)org.apache.spark.sql.DataFrame
```

Took 0 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:22:07 AM.

SPARK JOB FINISHED

```
val null_count = getNullCount(input_data)
val nan_count = getNanCount(input_data)
val blank_count = getBlankCount(input_data)
val zero_count = getZeroCount(input_data)
```

```
null_count: org.apache.spark.sql.DataFrame = [column: string, null_count: bigint]
nan_count: org.apache.spark.sql.DataFrame = [column: string, nan_count: bigint]
blank_count: org.apache.spark.sql.DataFrame = [column: string, blank_count: bigint]
zero_count: org.apache.spark.sql.DataFrame = [column: string, zero_count: bigint]
```

Took 11 min 11 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:33:18 AM.

Distribution of Null | NaN | Blank | Zero

SPARK JOB FINISHED

```
var merged_df=null_count.join(nan_count, Seq("column"))
merged_df=merged_df.join(blank_count, Seq("column"))
merged_df=merged_df.join(zero_count, Seq("column"))
z.show(merged_df)
```

settings

column	null_count	nan_count	blank_count	zero_count
Date	0	0	0	0
Open	0	0	0	15
High	0	0	0	0
Low	0	0	0	27
Close	0	0	0	0

Volume	0	0	0	24638
OpenInt	0	0	0	14887665

```
merged_df: org.apache.spark.sql.DataFrame = [column: string, null_count: bigint ... 3 more fields]
merged_df: org.apache.spark.sql.DataFrame = [column: string, null_count: bigint ... 3 more fields]
merged_df: org.apache.spark.sql.DataFrame = [column: string, null_count: bigint ... 3 more fields]
```

Took 0 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:33:18 AM.

Basic Feature Engineering

FINISHED

In this step, we will perform some basic feature engineering to explore the potential features for our future model training.

Took 0 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:33:18 AM.

Added New Feature Price

SPARK JOB (http://nyu-dataproc-sw-zwd3.c.hpc-dataproc-19b8.internal:42621/jobs/job?id=252) FINISHED

```
val new_df = input_data.withColumn("Price",
            (col("High") + col("Low") + col("Open") + col("Close"))/4)
z.show(new_df)
```

settings

Date	Open	High	Low	Close	Volume
1962-01-02	0.6277	0.6362	0.6201	0.6201	2575579
1962-01-03	0.6201	0.6201	0.6122	0.6201	1764749
1962-01-04	0.6201	0.6201	0.6037	0.6122	2194010
1962-01-05	0.6122	0.6122	0.5798	0.5957	3255244
1962-01-08	0.5957	0.5957	0.5716	0.5957	3696430
1962-01-09	0.5957	0.6037	0.5878	0.5957	2778285
1962-01-10	0.5957	0.6037	0.5957	0.5957	2337096
1962-01-11	0.5957	0.5957	0.5878	0.5957	1943605

Output is truncated to 1000 rows. Learn more about **zeppelin.spark.maxResult**

```
new_df: org.apache.spark.sql.DataFrame = [Date: string, Open: double ... 6 more fields]
```

Took 1 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:33:19 AM.

Added New Feature PriceVolume

SPARK JOB (http://nyu-dataproc-sw-zwd3.c.hpc-dataproc-19b8.internal:42621/jobs/job?id=253) FINISHED

```
val new_df_vol = new_df.withColumn("PriceVolume",
            (col("Price") * col("Volume")))
z.show(new_df_vol)
```

settings

Date	Open	High	Low	Close	Volume
1962-01-02	0.6277	0.6362	0.6201	0.6201	2575579
1962-01-03	0.6201	0.6201	0.6122	0.6201	1764749
1962-01-04	0.6201	0.6201	0.6037	0.6122	2194010
1962-01-05	0.6122	0.6122	0.5798	0.5957	3255244
1962-01-08	0.5957	0.5957	0.5716	0.5957	3696430
1962-01-09	0.5957	0.6037	0.5878	0.5957	2778285
1962-01-10	0.5957	0.6037	0.5957	0.5957	2337096
1962-01-11	0.5957	0.5957	0.5878	0.5957	1943605

Output is truncated to 1000 rows. Learn more about **zeppelin.spark.maxResult**

```
new_df_vol: org.apache.spark.sql.DataFrame = [Date: string, Open: double ... 7 more fields]
```

Took 0 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:33:19 AM.

Adding Stock Name As New Label

SPARK JOB (http://nyu-dataproc-sw-zwd3.c.hpc-dataproc-19b8.internal:42621/jobs/job?id=254) FINISHED

```
val new_input_file = new_df_vol.withColumn("filename", input_file_name)
z.show(new_input_file)
```

settings

Date	Open	High	Low	Close	Volume
1962-01-02	0.6277	0.6362	0.6201	0.6201	2575579
1962-01-03	0.6201	0.6201	0.6122	0.6201	1764749
1962-01-04	0.6201	0.6201	0.6037	0.6122	2194010

Output is truncated to 102400 bytes. Learn more about **ZEPPELIN_INTERPRETER_OUTPUT_LIMIT**

Took 1 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:33:20 AM.

UDF For Filename Processing

FINISHED

```
val fileStripping = udf((input: String) => {
    val temp = input.split("/")(7)
    temp.split("\\.")(0)
})
```

```
fileStripping: org.apache.spark.sql.expressions.UserDefinedFunction = SparkUserDefinedFunction($Lambda$5246/1520374711@53b39a7,StringType,List(Some(class[value[0]: string])),Some(class[value[0]: string]),None,true,true)
```

Took 0 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:33:20 AM.

```
val final_input = new_input_file.withColumn("Label", fileStripping(col("filename"))).drop("filename")
z.show(final_input)
```

SPARK JOB (http://nyu-dataproc-sw-zwd3.c.hpc-dataproc-19b8.internal:42621/jobs/job?id=255) FINISHED

settings

Date	Open	High	Low	Close	Volume
1962-01-02	0.6277	0.6362	0.6201	0.6201	2575579
1962-01-03	0.6201	0.6201	0.6122	0.6201	1764749
1962-01-04	0.6201	0.6201	0.6037	0.6122	2194010
1962-01-05	0.6122	0.6122	0.5798	0.5957	3255244
1962-01-08	0.5957	0.5957	0.5716	0.5957	3696430
1962-01-09	0.5957	0.6037	0.5878	0.5957	2778285
1962-01-10	0.5957	0.6037	0.5957	0.5957	2337096
1962-01-11	0.5957	0.5957	0.5878	0.5957	1943605

Output is truncated to 1000 rows. Learn more about **zeppelin.spark.maxResult**

```
final_input: org.apache.spark.sql.DataFrame = [Date: string, Open: double ... 8 more fields]
```

Took 0 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:33:20 AM.

```
z.show(final_input.describe())
```

SPARK JOB FINISHED

settings

summary	Date	Open	High	Low	Close
count	14887665	14887665	14887665	14887665	14887665
mean	null	30385.382295750554	31212.46731955359	29361.76076200612	30245.21834669
stddev	null	4202500.953751058	4323485.549288387	4046981.0035713837	4180590.065664

min	1962-01-02	0.0	0.004	-1.0	0.0037
max	2017-11-10	1.423712891E9	1.44204863645E9	1.36211784398E9	1.43798624044E

Took 3 min 28 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:36:48 AM.

Exploratory Data Analysis

FINISHED

In this set, we will do exploratory data analysis on the features present and handengineered, we will perform univariate and some multivariate analysis to undertand features and their importance.

Took 0 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:36:49 AM.

Univariate Data Analysis

FINISHED

In this step, we will start with univariate feature analysis. We will mostly perform all exploration and analysis on one feature for now called PriceVolume.

Took 0 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:36:49 AM.

Analysis on Feature PriceVolume

SPARK JOB (http://nyu-dataproc-sw-zwd3.c.hpc-dataproc-19b8.internal:42621/jobs/job?id=258) FINISHED

```
val exp_data = final_input.select("Date", "PriceVolume", "Label")
z.show(exp_data)
```

settings

Date	PriceVolume	Label
1962-01-02	1612376.8434749998	ge
1962-01-03	1090835.475625	ge
1962-01-04	1347176.99025	ge
1962-01-05	1953065.0188999998	ge
1962-01-08	2179692.36025	ge
1962-01-09	1655093.8316249999	ge
1962-01-10	1396882.2792	ge
1962-01-11	1153966.878625	ge

Output is truncated to 1000 rows. Learn more about **zeppelin.spark.maxResult**

```
exp_data: org.apache.spark.sql.DataFrame = [Date: string, PriceVolume: double ... 1 more field]
```

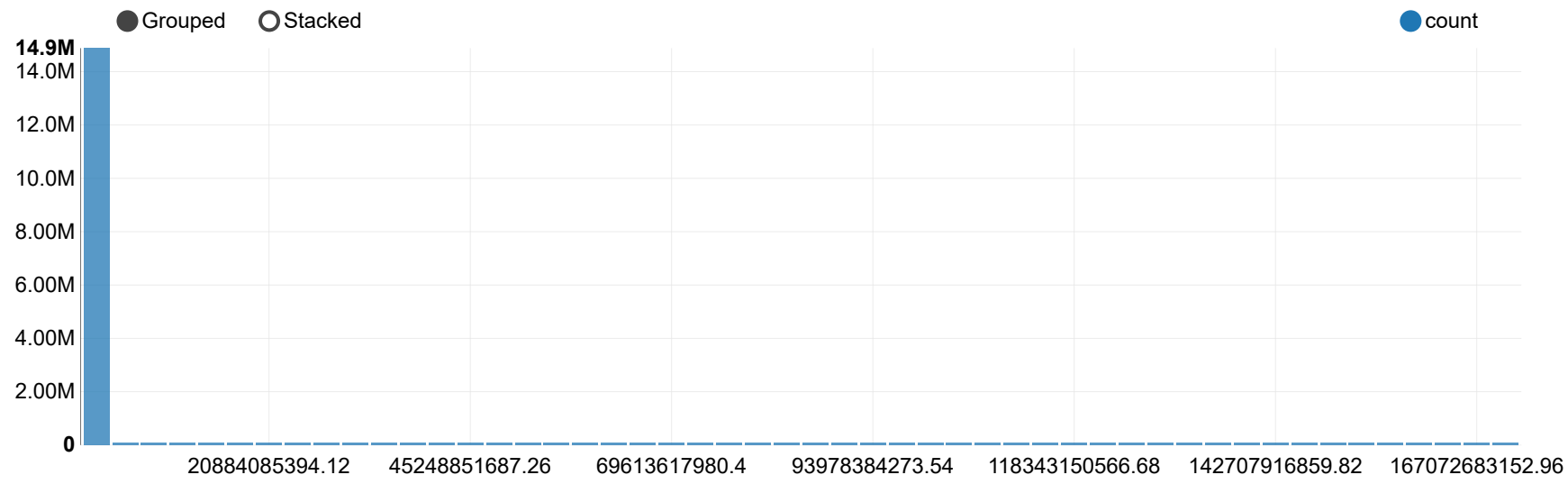
Took 0 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:36:49 AM.

Data Distribution of the Feature

SPARK JOB FINISHED

```
val (startValues,counts) = exp_data.select("PriceVolume").map(value => value.getDouble(0)).rdd.histogram(50)
val zippedValues = startValues.zip(counts)
case class HistRow(startPoint:Double,count:Long)
val rowRDD = zippedValues.map( value => HistRow(value._1,value._2))
val histDf = spark.createDataFrame(rowRDD)
z.show(histDf)
```

settings




```
startValues: Array[Double] = Array(0.0, 3.48068089902E9, 6.96136179804E9, 1.044204269706E10, 1.392272359608E10, 1.74034044951E10, 2.088408539412E10,
 2.436476629314E10, 2.784544719216E10, 3.132612809118E10, 3.48068089902E10, 3.828748988922E10, 4.176817078824E10, 4.524885168726E10, 4.872953258628E1
0, 5.22102134853E10, 5.569089438432E10, 5.917157528334E10, 6.265225618236E10, 6.613293708138E10, 6.96136179804E10, 7.309429887942E10, 7.657497977844E
10, 8.005566067746E10, 8.353634157648E10, 8.70170224755E10, 9.049770337452E10, 9.397838427354E10, 9.745906517256E10, 1.0093974607158E11, 1.0442042697
06E11, 1.0790110786962E11, 1.1138178876864E11, 1.1486246966766E11, 1.1834315056668E11, 1.218238314657E11, 1.2530451236472E11, 1.2878519326374E11, 1.3
226587416276E11, 1.35746555...
```

Took 1 min 19 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:38:08 AM.

Finding Skewness of the feature data distribution

SPARK JOB FINISHED

```
exp_data.select(skewness(col("PriceVolume"))).show()
```

```
+--------------------+
|skewness(PriceVolume)|
+--------------------+
|   117.59610566155635|
+--------------------+
```

Took 40 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:38:48 AM.

Finding Kurtosis of the feature data distribution

SPARK JOB FINISHED

```
exp_data.select(kurtosis(col("PriceVolume"))).show()
```

```
+--------------------+
|kurtosis(PriceVolume)|
+--------------------+
|    66291.06807414972|
+--------------------+
```

Took 41 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:39:29 AM.

Understanding Results

FINISHED

The above data distribution for feature, its skewness and kurtosis clearly indicate that the feature needs some engineering. It is very skewed and the data distribution is far from normal. In the steps following, we will be applying some functions to standardize and scale the given feature.

Took 0 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:39:30 AM.

Feature Scaling

SPARK JOB (http://nyu-dataproc-sw-zwd3.c.hpc-dataproc-19b8.internal:42621/jobs/job?id=265) FINISHED

```
val logged_data = final_input.withColumn("PriceVolumeLog",
                          log(col("PriceVolume"))
                          )
z.show(logged_data)
```

settings

Date	Open	High	Low	Close	Volume
1962-01-02	0.6277	0.6362	0.6201	0.6201	2575579
1962-01-03	0.6201	0.6201	0.6122	0.6201	1764749
1962-01-04	0.6201	0.6201	0.6037	0.6122	2194010
1962-01-05	0.6122	0.6122	0.5798	0.5957	3255244
1962-01-08	0.5957	0.5957	0.5716	0.5957	3696430
1962-01-09	0.5957	0.6037	0.5878	0.5957	2778285
1962-01-10	0.5957	0.6037	0.5957	0.5957	2337096
1962-01-11	0.5957	0.5957	0.5878	0.5957	1943605

Output is truncated to 1000 rows. Learn more about **zeppelin.spark.maxResult**

```
logged_data: org.apache.spark.sql.DataFrame = [Date: string, Open: double ... 9 more fields]
```

Took 0 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:39:30 AM.

```
val null_cnts = getNullCount(logged_data)
z.show(null_cnts)
```

SPARK JOB FINISHED

settings

column	null_count
Date	0
Open	0
High	0
Low	0
Close	0
Volume	0
OpenInt	0
Price	0

```
null_cnts: org.apache.spark.sql.DataFrame = [column: string, null_count: bigint]
```

Took 6 min 47 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:46:17 AM.

We can clearly see that after applying log, there are null values introduced, and so we need to filter those null values before further processing.

FINISHED

Took 0 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:46:18 AM.

```
val logged_fil_data = logged_data.filter($"PriceVolumeLog".isNotNull)
val (startValues,counts) = logged_fil_data.select("PriceVolumeLog").map(value => value.getDouble(0)).rdd.histogram(50)
```

SPARK JOB FINISHED

```
logged_fil_data: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [Date: string, Open: double ... 9 more fields]
startValues: Array[Double] = Array(-2.5257286443082556, -1.957563735868894, -1.3893988274295324, -0.8212339189901707, -0.2530690105508091, 0.3150958978885523, 0.8832608063279141, 1.4514257147672756, 2.0195906232066374, 2.587755531645999, 3.15592044008536, 3.7240853485247216, 4.292250256964084, 4.860415165403444, 5.428580073842807, 5.996744982282168, 6.56490989072153, 7.133074799160891, 7.701239707600253, 8.269404616039616, 8.837569524478976, 9.405734432918337, 9.973899341357699, 10.542064249797061, 11.110229158236423, 11.678394066675786, 12.246558975115144, 12.814723883554507, 13.382888791993869, 13.951053700433231, ...
```

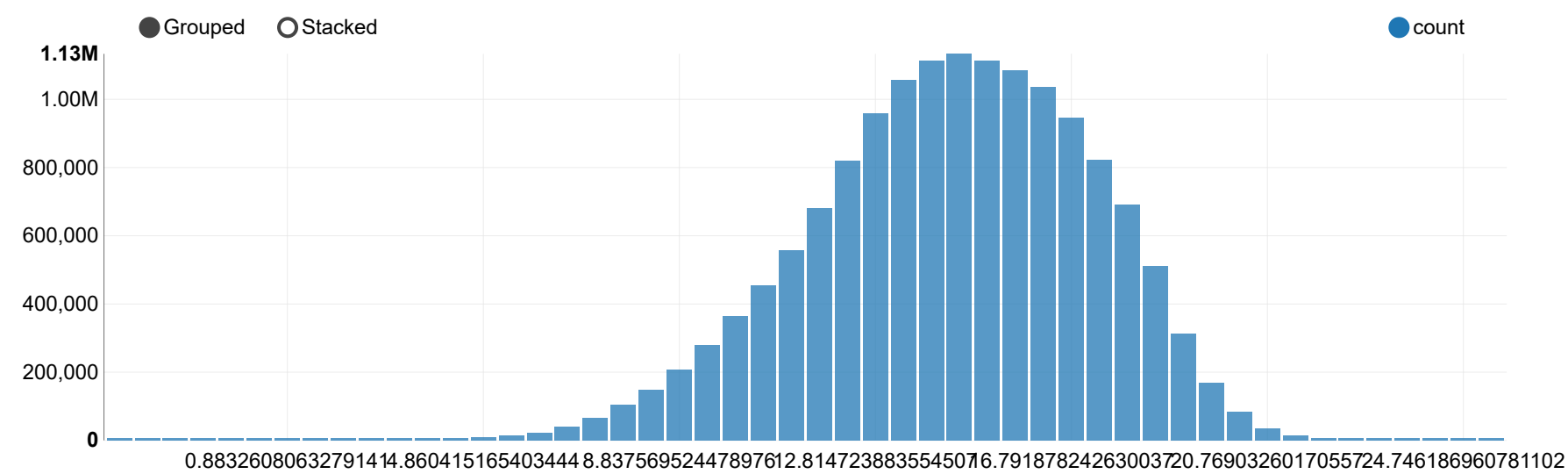
Took 1 min 28 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:47:46 AM.

Feature Data Distribution

FINISHED

```
val zippedValues = startValues.zip(counts)
case class HistRow(startPoint:Double,count:Long)
val rowRDD = zippedValues.map( value => HistRow(value._1,value._2))
val histDf = spark.createDataFrame(rowRDD)
z.show(histDf)
```

settings



-
```
zippedValues: Array[(Double, Long)] = Array((-2.5257286443082556,8), (-1.957563735868894,0), (-1.3893988274295324,5), (-0.8212339189901707,5), (-0.2530690105508091,27), (0.3150958978885523,34), (0.8832608063279141,128), (1.4514257147672756,219), (2.0195906232066374,457), (2.587755531645999,895), (3.15592044008536,1733), (3.7240853485247216,2839), (4.292250256964084,5147), (4.860415165403444,8187), (5.428580073842807,14260), (5.99674498228216
```

```
8,22838), (6.56490989072153,40260), (7.133074799160891,65039), (7.701239707600253,103026), (8.269404616039616,148353), (8.837569524478976,206831),
 (9.405734432918337,278222), (9.973899341357699,363349), (10.542064249797061,455161), (11.110229158236423,558462), (11.678394066675786,680623), (12.2
46558975115144,820879), (12.814...
```

Took 0 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:47:46 AM.

Analyzing the PriceVolume Feature for Single Stock Over Time

SPARK JOB (http://nyu-dataproc-sw-zwd3.c.hpc-dataproc-19b8.internal:42621/jobs/job?id=290) FINISHED

```
val fil = exp_data.where($"Label" === "ge")
z.show(fil)
```

settings

Date	PriceVolume	Label
1962-01-02	1612376.8434749998	ge
1962-01-03	1090835.475625	ge
1962-01-04	1347176.99025	ge
1962-01-05	1953065.0188999998	ge
1962-01-08	2179692.36025	ge
1962-01-09	1655093.8316249999	ge
1962-01-10	1396882.2792	ge
1962-01-11	1153966.878625	ge

Output is truncated to 1000 rows. Learn more about **zeppelin.spark.maxResult**

```
fil: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [Date: string, PriceVolume: double ... 1 more field]
```

Took 1 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:47:47 AM. (outdated)

```
val year_data = fil.withColumn("Year",
                date_format(col("Date"), "y")
)
z.show(year_data)
```

SPARK JOB (http://nyu-dataproc-sw-zwd3.c.hpc-dataproc-19b8.internal:42621/jobs/job?id=291) FINISHED

settings

Date	PriceVolume	Label	Year
1962-01-08	2179692.36025	ge	1962
1962-01-09	1655093.8316249999	ge	1962
1962-01-10	1396882.2792	ge	1962
1962-01-11	1153966.878625	ge	1962
1962-01-12	1196495.90625	ge	1962
1962-01-15	1505857.5203	ge	1962
1962-01-16	993963.1698	ge	1962
1962-01-17	1520192.55175	ge	1962

Output is truncated to 1000 rows. Learn more about **zeppelin.spark.maxResult**

```
year_data: org.apache.spark.sql.DataFrame = [Date: string, PriceVolume: double ... 2 more fields]
```

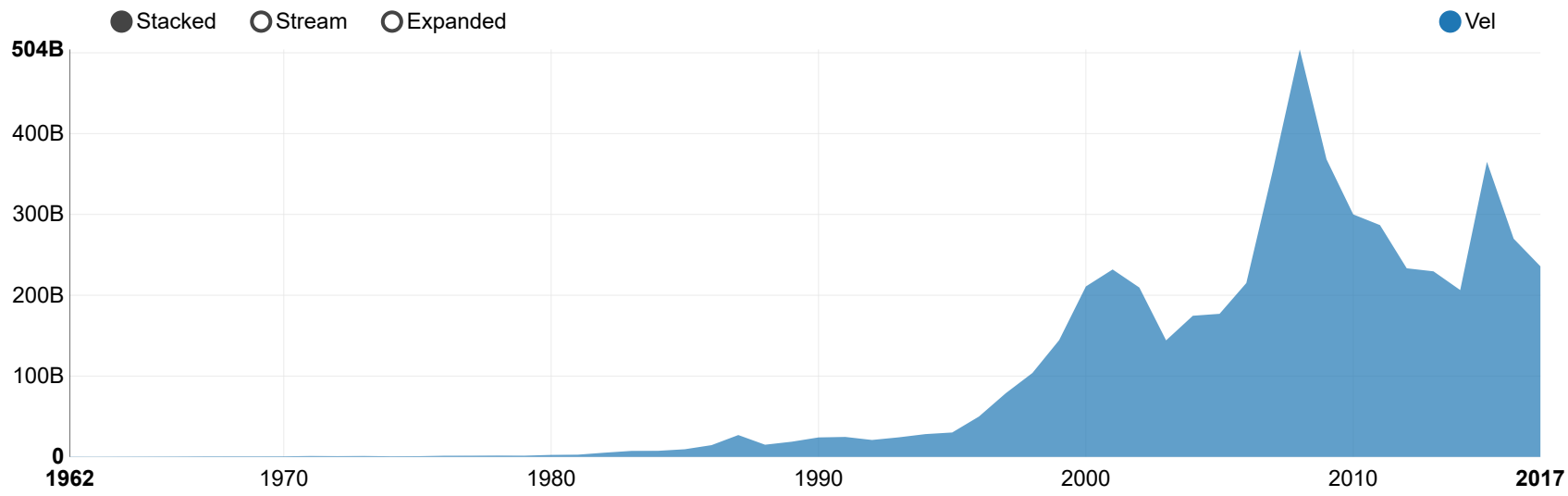
Took 0 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:47:47 AM.

Distribution of PriceVolume for Single Stock Over Time

SPARK JOB FINISHED

```
val group_data = year_data.groupBy("Year")
        .agg(
            sum("PriceVolume").as("Vel")
            )
z.show(group_data)
```

settings




```
group_data: org.apache.spark.sql.DataFrame = [Year: string, Vel: double]
```

Took 54 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:48:41 AM.

Mutlivariate Analysis

FINISHED

In this step we will perform multivariate data analysis, we will perform bivariate analysis between features and try rto understand correlation between them.

Took 0 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:48:41 AM.

Correlation between Price and PriceVolume

SPARK JOB FINISHED

```
import org.apache.spark.mllib.stat.{MultivariateStatisticalSummary, Statistics}
Statistics.corr(logged_fil_data.select("Price").rdd.map(x=> x.getDouble(0)), logged_fil_data.select("PriceVolumeLog").rdd.map(x=> x
    .getDouble(0)))
```

```
import org.apache.spark.mllib.stat.{MultivariateStatisticalSummary, Statistics}
res85: Double = 0.006360136880506116
```

Took 2 min 56 sec. Last updated by hd2225_nyu_edu at November 25 2022, 7:51:38 AM.