Introduction

Air Jordans, the iconic line of sneakers born from the collaboration between Nike and basketball legend Michael Jordan, has found a thriving marketplace on StockX. Thanks to this internet marketplace, Air Jordans are no longer merely popular athletic shoes but also sought-after collectors and style statements. Join us as we explore the world of Air Jordans on StockX, where culture, fashion, and sports meet.

Objective

This project's main goal is to boost Air Jordan goods' sales performance on StockX, a well-known online retailer of sneakers and streetwear. The project is structured to accomplish this goal through the analysis of sales data, price optimization, product selection, data integration, and insights visualization.

Scope

Product Name, Sales, and Retail_price are the three main facets of the Air Jordan dataset that are the main emphasis of the research. The project seeks to offer practical knowledge and suggestions for enhancing the sales of Air Jordan products on StockX by concentrating on these essential elements.

Data Exploration

Importing necessary libraries.

In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
df = pd.read_csv('AJs.csv')
df.head(3)
```

Out[2]:

	name	sales	retail_price	average_sale_price	highest_price	lowest_price	release_date	condition
0	Jordan 4 Retro Bred (2019)	28140	200	254	520	138	05/04/2019	New
1	Jordan 1 Retro High Travis Scott	17269	175	1013	3000	578	05/11/2019	New
2	Jordan 11 Retro Concord (2018)	37993	220	278	500	175	12/08/2018	New

Checking the shape of the dataset.

Result: - As we can see there are 8 columns and 998 rows in the dataset.

In [3]:
```
df.shape
```

Out[3]:
```
(998, 8)
```

Checking the descriptive statistics of the dataset.

Result: - Below we can see the count values of each columns, mean, quartile, standard deviation, min and max values for all the numeric columns.

In [4]:
```
df.describe()
```

Out[4]:

	sales	average_sale_price	highest_price	lowest_price
count	998.000000	998.000000	998.000000	998.000000
mean	1167.974950	258.348697	456.640281	156.724449
std	3001.948436	458.976897	767.677641	320.310452
min	1.000000	31.000000	45.000000	25.000000
25%	112.000000	123.000000	220.000000	65.250000
50%	295.500000	170.500000	300.000000	100.000000
75%	833.500000	244.750000	429.750000	150.000000
max	37993.000000	8833.000000	10500.000000	7000.000000

Checking if there are any null values in the dataset.

Result: - As we can see we don't have any null values which means that our dataset is clean.

In [5]:
```
df.isnull().sum().sum()
```

Out[5]:
```
0
```

Checking the types of the dataset.

In [6]:
```
df.dtypes
```

Out[6]:
```
name                  object
sales                  int64
retail_price          object
average_sale_price     int64
highest_price          int64
lowest_price           int64
release_date          object
condition             object
dtype: object
```

Data Cleaning

Using **df.convert_dtypes()** to covert the types of the dataset for further analysis.

In [7]:
```
df = df.convert_dtypes()
df.dtypes
```

Out[7]:
```
name                  string
sales                  Int64
retail_price          string
average_sale_price     Int64
highest_price          Int64
lowest_price           Int64
release_date          string
condition             string
dtype: object
```

Using **.str.strip()** to remove trailing spaces from the release date column so that we can convert it to datetime for further analysis to boost the product sales over time.

In [8]:
```
# Removing leading and trailing spaces from 'release_date' strings
df['release_date'] = df['release_date'].str.strip()

# Converting 'release_date' column to datetime, skipping empty values
df['release_date'] = pd.to_datetime(df['release_date'], format='%m/%d/%Y', errors='coerce')

# Saving the transformed dataset
df.to_csv("transformed_AJs.csv", index=False)
```

Replacing dollar, commas and empty values by 0.0 and then saving it as cleaned dataset.

In [9]:
```
# Performing data transformation (including handling empty or non-numeric retail_price)
df['retail_price'] = df['retail_price'].str.replace('$', '').str.replace(',', '')

# Handling empty or non-numeric values by replacing them with 0.0
df['retail_price'] = df['retail_price'].apply(lambda x: float(x) if x.strip() else 0.0)

# Saving the cleaned dataset
df.to_csv("cleaned_AJs.csv", index=False)
```

```
<ipython-input-9-dc62da7cb29e>:2: FutureWarning: The default value of regex will change from True to False in a future version. In addition, single character regular expressions will *not* be treated as literal strings when regex=True.
  df['retail_price'] = df['retail_price'].str.replace('$', '').str.replace(',', '')
```

Now, checking the type of the dataset.

Result: - As we can see that we have successfully changed the type of the dataset columns and we are also done with the data cleaning part now let's start with Data Integration.

In [10]:
```
df.dtypes
```

Out[10]:
```
name                          string
sales                          Int64
retail_price                 float64
average_sale_price             Int64
highest_price                  Int64
lowest_price                   Int64
release_date          datetime64[ns]
condition                     string
dtype: object
```

Data Integration

Calculating the average_sale_price by multiplying the sales and retail_price columns and the calculating the profit_margin by subtracting the lowest_price column from the average_sale_price and then dividing the result by average_sale_price. Finally, printing first few rows to check the output.

In [11]:

```
# Performing aggregations and calculations
df['average_sale_price'] = df['sales'] * df['retail_price']
df['profit_margin'] = (df['average_sale_price'] - df['lowest_price']) / df['average_sale_price']

# Displaying the integrated DataFrame
print(df.head())
```

```
                                  name  sales  retail_price  \
0              Jordan 4 Retro Bred (2019)  28140         200.0
1        Jordan 1 Retro High Travis Scott  17269         175.0
2          Jordan 11 Retro Concord (2018)  37993         220.0
3  Jordan 1 Retro High Black Crimson Tint  14118         160.0
4         Jordan 1 Retro High Turbo Green  23862         160.0

   average_sale_price  highest_price  lowest_price release_date condition  \
0           5628000.0            520           138   2019-05-04       New
1           3022075.0           3000           578   2019-05-11       New
2           8358460.0            500           175   2018-12-08       New
3           2258880.0            500            82   2019-04-12       New
4           3817920.0            405            75   2019-02-15       New

   profit_margin
0       0.999975
1       0.999809
2       0.999979
3       0.999964
4        0.99998
```

Calculating the count of number of words by selecting top 10 products using the name and sales column from the dataset. Using ThreadPoolExecutor with max 5 threads to parallelize the word counting process. Next, adding the calculated word count as a new column word_count. Finally printing the output.

Result: - The top 10 product names' word counts provide some interesting observations. Names with a reasonable word count that are brief and concise frequently perform well. Product names that highlight important features, release dates, and distinctive qualities may be more enticing to consumers. However, since excessively extensive names might be difficult to remember, it's crucial to achieve a balance between description and clarity. The attractiveness of products can also be increased by examining the frequency of fashionable terms and adopting them where appropriate.

The organization may be able to increase product discoverability, consumer engagement, and ultimately sales success by optimizing product names in light of these facts.

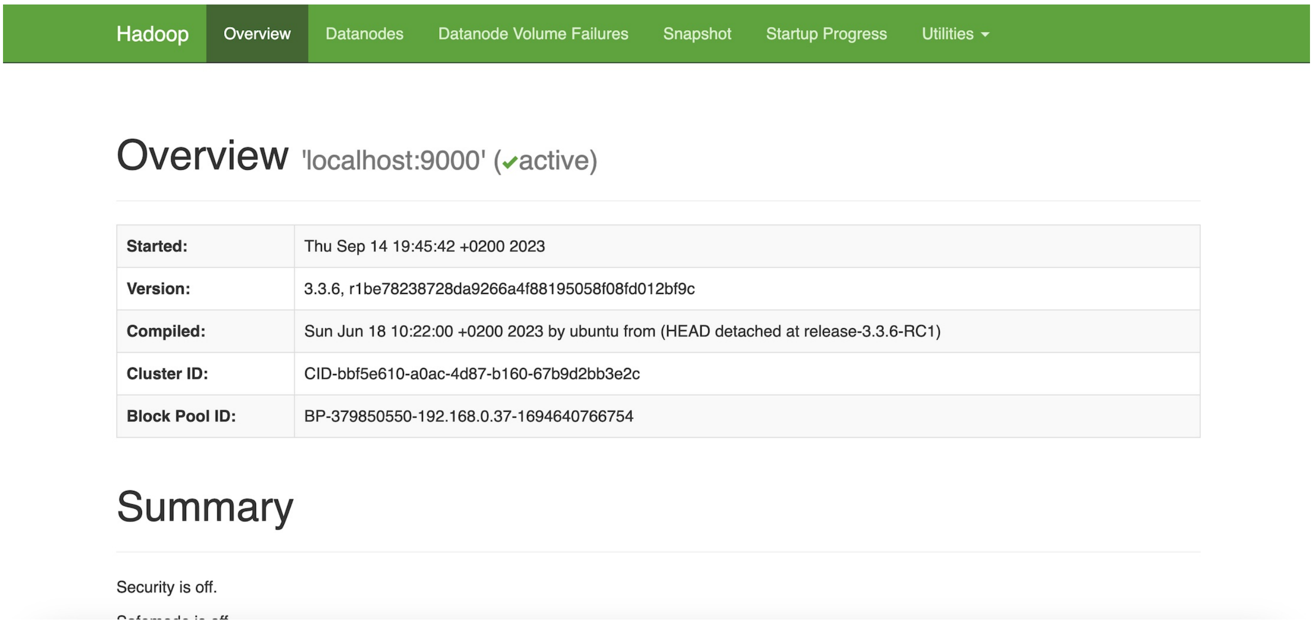
In [12]:

```
import concurrent.futures

# Defining a function to perform word counting on a product name
def count_words(product_name):
    words = product_name.split()  # Splitting the product name into words
    return len(words)

# Selecting the top 10 products based on sales
top_10_products = df.sort_values(by='sales', ascending=False).head(10)

# Creating a ThreadPoolExecutor with a maximum of 5 threads
with concurrent.futures.ThreadPoolExecutor(max_workers=5) as executor:
    # Using the executor to calculate word counts for each product name
    word_counts = list(executor.map(count_words, top_10_products['name']))

# Adding the word counts as a new column in the DataFrame
top_10_products['word_count'] = word_counts

# Displaying the top 10 products with word counts
print(top_10_products[['name', 'word_count']])
```

```
                                         name  word_count
2                Jordan 11 Retro Concord (2018)           5
0                    Jordan 4 Retro Bred (2019)           5
10        Jordan 1 Retro High Rookie of the Year           8
4               Jordan 1 Retro High Turbo Green           6
140              Jordan 1 Retro High Pine Green           6
7           Jordan 6 Retro Black Infrared (2019)           6
20            Jordan 1 Retro High UNC Patent (W)           7
16   Jordan 1 Retro High Spider-Man Origin Story           7
19          Jordan 1 Retro High Turbo Green (GS)           7
1              Jordan 1 Retro High Travis Scott           6
```

Hadoop Mapreduce

I tried to download hadoop using bash and have successfully downloaded it but unfortunately unable to run the code due to some error.



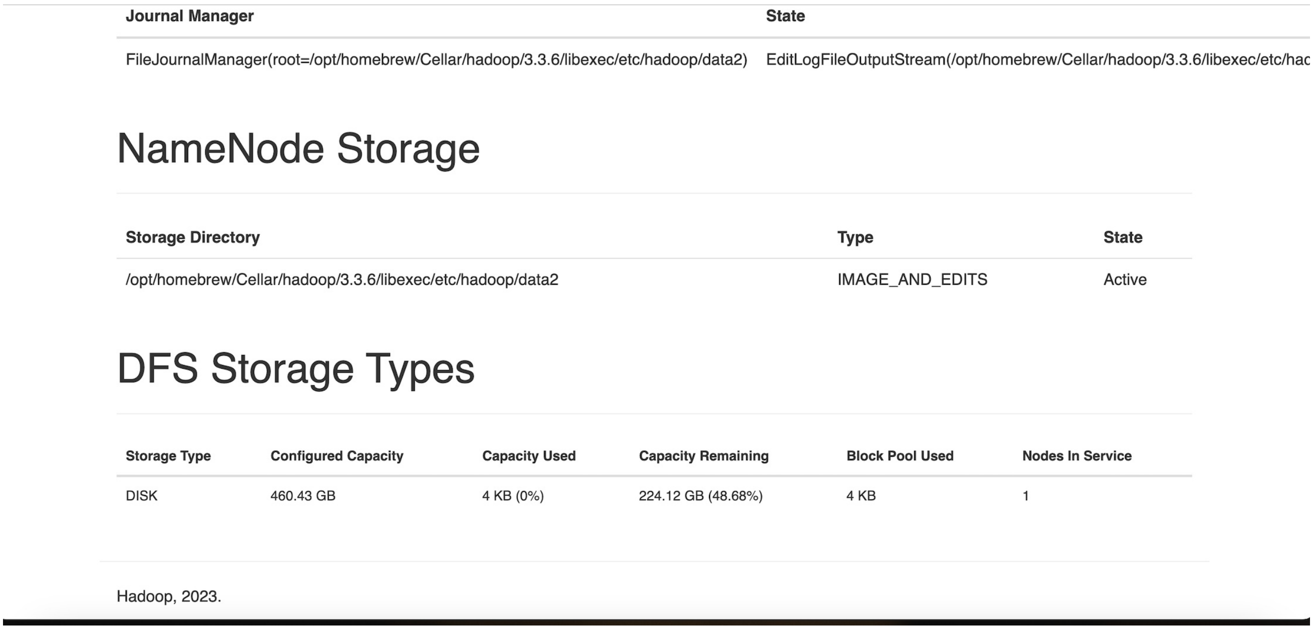
Hadoop | Overview | Datanodes | Datanode Volume Failures | Snapshot | Startup Progress | Utilities

Overview 'localhost:9000' (active)

Started:	Thu Sep 14 19:45:42 +0200 2023
Version:	3.3.6, r1be78238728da9266a4f88195058f08fd012bf9c
Compiled:	Sun Jun 18 10:22:00 +0200 2023 by ubuntu from (HEAD detached at release-3.3.6-RC1)
Cluster ID:	CID-bbf5e610-a0ac-4d87-b160-67b9d2bb3e2c
Block Pool ID:	BP-379850550-192.168.0.37-1694640766754

Summary

Security is off.



Journal Manager	State
FileJournalManager(root=/opt/homebrew/Cellar/hadoop/3.3.6/libexec/etc/hadoop/data2)	EditLogFileOutputStream(/opt/homebrew/Cellar/hadoop/3.3.6/libexec/etc/hac

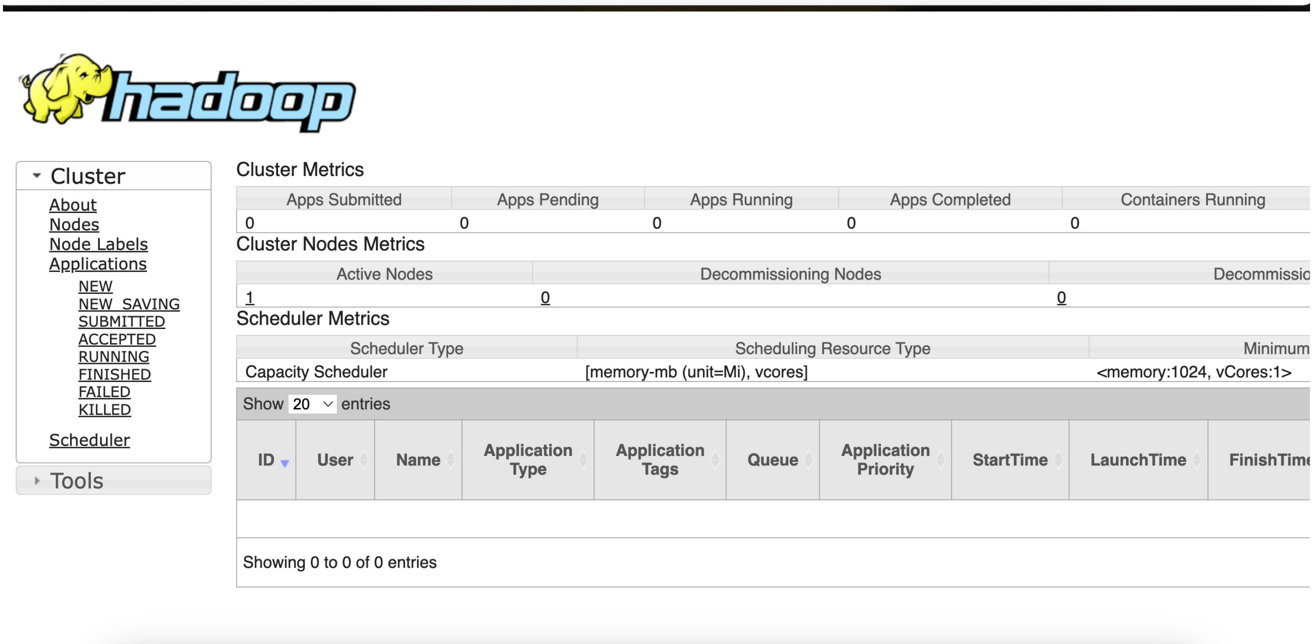
NameNode Storage

Storage Directory	Type	State
/opt/homebrew/Cellar/hadoop/3.3.6/libexec/etc/hadoop/data2	IMAGE_AND_EDITS	Active

DFS Storage Types

Storage Type	Configured Capacity	Capacity Used	Capacity Remaining	Block Pool Used	Nodes In Service
DISK	460.43 GB	4 KB (0%)	224.12 GB (48.68%)	4 KB	1

Hadoop, 2023.



Cluster
- About
- Nodes
- Node Labels
- Applications
  - NEW
  - NEW_SAVING
  - SUBMITTED
  - ACCEPTED
  - RUNNING
  - FINISHED
  - FAILED
  - KILLED
- Scheduler

Tools

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running
0	0	0	0	0

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissio
1	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum
Capacity Scheduler	[memory-mb (unit=Mi), vcores]	<memory:1024, vCores:1>

Show 20 entries

ID	User	Name	Application Type	Application Tags	Queue	Application Priority	StartTime	LaunchTime	FinishTim

Showing 0 to 0 of 0 entries

After lots of trial using the bash hadoop I wasn't able to get the output for my mapper and reducer so I decided to try Docker hadoop. I tried a lot and I can successfully able to rup my hadoop mapreduce function in Docker container, but as you can see in the below output, I run into a problem once again. I therefore made the decision to work with haoop using PySpark and MapReduce operation to count the occurrences.

The Python script analyzes each row of CSV data after reading it from standard input before performing a mapping operation

The Python script analyzes each row of CSV data after reading it from standard input before performing a mapping operation. Next, omitting key-value pairs with the "sales" value as the key and the difference between the "average_sale_price" and "lowest_price" columns as the value, skipping the header row in the process. The code generates a measure that resembles a profit margin based on the provided CSV data and looks to be designed for data transformation or analysis.

In [13]:

```python
#!/usr/bin/env python

import sys
import csv

# Read from standard input
for line in sys.stdin:
    # Parse CSV data
    row = list(csv.reader([line]))[0]
    name, sales, retail_price, average_sale_price, highest_price, lowest_price, release_date, condition = row

    # Implement your mapping logic here
    # For example, you can emit a key-value pair for sales and profit margin
    # Emit sales as key and profit margin as value
    if name != 'name':  # Skip the header row if present
        print(f"{sales}\t{float(average_sale_price) - float(lowest_price)}")
```

The Python script reads key-value pairs from standard input with the key "sales" and the value "profit_margin." Each sales value's profit margin values are combined, and after determining the average profit margin for each sales value, the sales value and associated average profit margin are then produced. With this code, the average profit margin for each distinct sales value in the input data is efficiently computed and displayed.

In [14]:

```python
#!/usr/bin/env python

import sys

current_sales = None
total_profit_margin = 0.0
count = 0

# Read from standard input
for line in sys.stdin:
    # Split the input line into sales and profit margin
    sales, profit_margin = line.strip().split('\t')
    sales = int(sales)
    profit_margin = float(profit_margin)

    # Aggregate profit margin for each sales value
    if current_sales == sales:
        total_profit_margin += profit_margin
        count += 1
    else:
        if current_sales is not None:
            # Calculate and emit the average profit margin for the previous sales value
            avg_profit_margin = total_profit_margin / count
            print(f"{current_sales}\t{avg_profit_margin}")

        current_sales = sales
        total_profit_margin = profit_margin
        count = 1

# Don't forget to emit the last average profit margin
if current_sales is not None:
    avg_profit_margin = total_profit_margin / count
    print(f"{current_sales}\t{avg_profit_margin}")
```

```
(base) ishratshaikh@Ishrats-Air ~ % docker exec -it adaa90dec3c2 /bin/bash
bash-4.1# /usr/local/hadoop/bin/hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.7.1.jar \
> -files /usr/local/hadoop/mapper.py,/usr/local/hadoop/reducer.py \
> -mapper mapper.py \
> -reducer reducer.py \
> -input /Users/ishratshaikh/input/directory/AJs.csv \
> -output /Users/ishratshaikh/output/directory
23/09/16 08:24:41 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
packageJobJar: [/tmp/hadoop-unjar120979414335908701/] [] /tmp/streamjob8177945214112581454.jar tmpDir=null
23/09/16 08:24:44 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
23/09/16 08:24:45 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
23/09/16 08:24:46 INFO ipc.Client: Retrying connect to server: 0.0.0.0/0.0.0.0:8032. Already tried 0 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS
)
```

```
23/09/16 08:44:10 INFO ipc.Client: Retrying connect to server: 0.0.0.0/0.0.0.0:8032. Already tried 0 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS
)
23/09/16 08:44:11 INFO ipc.Client: Retrying connect to server: 0.0.0.0/0.0.0.0:8032. Already tried 1 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS
)
23/09/16 08:44:12 INFO ipc.Client: Retrying connect to server: 0.0.0.0/0.0.0.0:8032. Already tried 2 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS
)
23/09/16 08:44:13 INFO ipc.Client: Retrying connect to server: 0.0.0.0/0.0.0.0:8032. Already tried 3 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS
)
23/09/16 08:44:14 INFO ipc.Client: Retrying connect to server: 0.0.0.0/0.0.0.0:8032. Already tried 4 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS
)
23/09/16 08:44:15 INFO ipc.Client: Retrying connect to server: 0.0.0.0/0.0.0.0:8032. Already tried 5 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS
)
23/09/16 08:44:16 INFO ipc.Client: Retrying connect to server: 0.0.0.0/0.0.0.0:8032. Already tried 6 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS
)
23/09/16 08:44:17 INFO ipc.Client: Retrying connect to server: 0.0.0.0/0.0.0.0:8032. Already tried 7 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS
)
23/09/16 08:44:18 INFO ipc.Client: Retrying connect to server: 0.0.0.0/0.0.0.0:8032. Already tried 8 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS
)
23/09/16 08:44:19 INFO ipc.Client: Retrying connect to server: 0.0.0.0/0.0.0.0:8032. Already tried 9 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS
)
23/09/16 08:44:50 INFO ipc.Client: Retrying connect to server: 0.0.0.0/0.0.0.0:8032. Already tried 0 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS
)
23/09/16 08:44:51 INFO ipc.Client: Retrying connect to server: 0.0.0.0/0.0.0.0:8032. Already tried 1 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS
)
23/09/16 08:44:52 INFO ipc.Client: Retrying connect to server: 0.0.0.0/0.0.0.0:8032. Already tried 2 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS
)
23/09/16 08:44:53 INFO ipc.Client: Retrying connect to server: 0.0.0.0/0.0.0.0:8032. Already tried 3 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS
)
23/09/16 08:44:54 INFO ipc.Client: Retrying connect to server: 0.0.0.0/0.0.0.0:8032. Already tried 4 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS
)
23/09/16 08:44:55 INFO ipc.Client: Retrying connect to server: 0.0.0.0/0.0.0.0:8032. Already tried 5 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS
)
```

```
23/09/16 08:44:56 INFO ipc.Client: Retrying connect to server: 0.0.0.0/0.0.0.0:8032. Already tried 6 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS
)
23/09/16 08:44:57 INFO ipc.Client: Retrying connect to server: 0.0.0.0/0.0.0.0:8032. Already tried 7 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS
)
23/09/16 08:44:58 INFO ipc.Client: Retrying connect to server: 0.0.0.0/0.0.0.0:8032. Already tried 8 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS
)
23/09/16 08:44:59 INFO ipc.Client: Retrying connect to server: 0.0.0.0/0.0.0.0:8032. Already tried 9 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS
)
23/09/16 08:44:59 ERROR streaming.StreamJob: Error Launching job : Call From adaa90dec3c2/172.17.0.2 to 0.0.0.0:8032 failed on connection exception: java.net.ConnectException: Connection refused; For more
 details see:  http://wiki.apache.org/hadoop/ConnectionRefused
Streaming Command Failed!
bash-4.1#
```

Apache Spark Implementation

In [15]:
```
!pip install pyspark
```

```
Collecting pyspark
  Downloading pyspark-3.4.1.tar.gz (310.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 310.8/310.8 MB 2.2 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.10/dist-packages (from pyspark) (0.10.9
.7)
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.4.1-py2.py3-none-any.whl size=311285387 sha256=1f407b2c7ad99957
d565f39b72ca8086274ef2df1270f7b8adf9ee5299d8919f
  Stored in directory: /root/.cache/pip/wheels/0d/77/a3/ff2f74cc9ab41f8f594dabf0579c2a7c6de920d584206e0834
Successfully built pyspark
Installing collected packages: pyspark
Successfully installed pyspark-3.4.1
```

Making a spark session called "ProductSalesAnalysis" and grouping the dataframe according to the "name" column, which stands for the names of the products. With the use of the "avg" and "sum" functions on the sales column, the final result will figure out the average and overall sales for each product. The calculated column is renamed for clarity, and the results are sorted to put the best-selling item at the top and are then displayed together with their average sales totals.

In [21]:
```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col

# Create a SparkSession
spark = SparkSession.builder.appName("ProductSalesAnalysis").getOrCreate()

# Convert the Pandas DataFrame 'df' to a Spark DataFrame
spark_df = spark.createDataFrame(df)

# Group by product name and calculate average sales and total sales count
result = spark_df.groupBy("name").agg(
    {"sales": "avg", "sales": "sum"}
).withColumnRenamed("avg(sales)", "average_sales").withColumnRenamed("sum(sales)", "total_sales")

# Sort the result by total sales count in descending order
result = result.orderBy(col("total_sales").desc())

# Show the top products with the highest total sales count and their average sales
result.show()
```

```
+--------------------+-----------+
|                name|total_sales|
+--------------------+-----------+
|Jordan 11 Retro C...|      37993|
|Jordan 4 Retro Br...|      28140|
|Jordan 1 Retro Hi...|      24811|
|Jordan 1 Retro Hi...|      23862|
|Jordan 1 Retro Hi...|      22564|
|Jordan 6 Retro Bl...|      21192|
|Jordan 1 Retro Hi...|      20934|
|Jordan 1 Retro Hi...|      19839|
|Jordan 1 Retro Hi...|      18916|
|Jordan 1 Retro Hi...|      17269|
|Jordan 1 Retro Hi...|      17107|
|Jordan 1 Retro Hi...|      15958|
|Jordan 1 Retro Hi...|      14118|
|Jordan 1 Retro Hi...|      13487|
|Jordan 1 Retro Hi...|      12791|
|Jordan 1 Retro Hi...|      12777|
|Jordan 1 Retro Hi...|      12365|
|Jordan 1 Mid Pate...|      12233|
|Jordan 1 Retro Hi...|      12145|
|Jordan 1 Retro Hi...|      11089|
+--------------------+-----------+
only showing top 20 rows
```

Creating a Sparksession by using the pyspark library with the name DataIntegration and then assigning it to the spark variable to work with distributed data processing and analysis.

In [22]:
```
from pyspark.sql import SparkSession
spark=SparkSession.builder.appName('DataIntegration').getOrCreate()
```

Specifying the first row of csv file as header **(header=True)** and requesting Spark to infer the schema **(data types)** of the columns automatically **(inferSchema=True)**.

In [23]:
```
## Reading the dataset
data_df = spark.read.csv("AJs.csv", header=True, inferSchema=True)
```

Modifying the datatype of the retail_price column to be double precision floating point number.

In [24]:
```
from pyspark.sql.functions import regexp_replace
from pyspark.sql.types import DoubleType

# Convert the "retail_price" column to DoubleType
data_df = data_df.withColumn("retail_price", data_df["retail_price"].cast(DoubleType()))
```

Using PySpark's VectorAssembler to combine multiple columns from the dataset into a single vector column named features. Establishing a vector assembler instance and defining features as the output column, with the input columns defined by features_cols. Lastly, transforming the original dataset to include new "feature" column.

In [25]:
```
from pyspark.ml.feature import VectorAssembler
# Assuming you have a DataFrame 'data_df' with the relevant columns
feature_cols = ["sales", "retail_price", "average_sale_price"]

# Create a VectorAssembler instance
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")

# Transform the DataFrame using the assembler
output_df = assembler.transform(data_df)
```

In [26]:
```
output_df.show()
```

```
+--------------------+-------+------------+------------------+-------------+------------+------------+---------
+--------------------+
|                name|  sales|retail_price|average_sale_price|highest_price|lowest_price|release_date|condition
|            features|
+--------------------+-------+------------+------------------+-------------+------------+------------+---------
+--------------------+
|Jordan 4 Retro Br...|28140.0|       200.0|             254.0|        520.0|       138.0|  05/04/2019|      New
|[28140.0,200.0,25...|
|Jordan 1 Retro Hi...|17269.0|       175.0|            1013.0|       3000.0|       578.0|  05/11/2019|      New
|[17269.0,175.0,10...|
|Jordan 11 Retro C...|37993.0|       220.0|             278.0|        500.0|       175.0|  12/08/2018|      New
|[37993.0,220.0,27...|
|Jordan 1 Retro Hi...|14118.0|       160.0|             199.0|        500.0|        82.0|  04/12/2019|      New
|[14118.0,160.0,19...|
|Jordan 1 Retro Hi...|23862.0|       160.0|             235.0|        405.0|        75.0|  02/15/2019|      New
|[23862.0,160.0,23...|
|Jordan 1 Low SB M...| 1691.0|       110.0|             144.0|        232.0|       110.0|  06/17/2019|      New
|[1691.0,110.0,144.0]|
|Jordan 11 Retro L...| 5154.0|       185.0|             165.0|        350.0|       115.0|  04/19/2019|      New
|[5154.0,185.0,165.0]|
|Jordan 6 Retro Bl...|21192.0|       200.0|             239.0|        470.0|       150.0|  02/16/2019|      New
|[21192.0,200.0,23...|
|Jordan 1 Retro Hi...| 7791.0|       160.0|             367.0|        750.0|       100.0|  02/24/2018|      New
|[7791.0,160.0,367.0]|
|Jordan 7 Retro Pa...| 3870.0|       200.0|             290.0|       1200.0|       140.0|  06/15/2019|      New
|[3870.0,200.0,290.0]|
|Jordan 1 Retro Hi...|24811.0|       160.0|             245.0|        800.0|       145.0|  11/17/2018|      New
|[24811.0,160.0,24...|
|Jordan 1 Retro Hi...| 9471.0|       160.0|             190.0|        400.0|       100.0|  03/30/2019|      New
|[9471.0,160.0,190.0]|
|Jordan 1 Retro Hi...|12791.0|       160.0|             225.0|        615.0|       100.0|  04/14/2018|      New
|[12791.0,160.0,22...|
|Jordan 1 Retro Hi...|17107.0|       160.0|             220.0|        400.0|       105.0|  01/24/2019|      New
|[17107.0,160.0,22...|
|Jordan 3 Retro Bl...| 9513.0|       200.0|             243.0|        355.0|       150.0|  02/17/2018|      New
|[9513.0,200.0,243.0]|
|Jordan 1 Low Blac...| 4978.0|        90.0|             106.0|        300.0|        60.0|  04/01/2019|      New
| [4978.0,90.0,106.0]|
|Jordan 1 Retro Hi...|19839.0|       160.0|             306.0|        634.0|       195.0|  12/14/2018|      New
|[19839.0,160.0,30...|
|Jordan 1 Retro Hi...|15958.0|       160.0|             205.0|        431.0|        90.0|  12/27/2018|      New
|[15958.0,160.0,20...|
|Jordan 1 Retro Hi...| 9581.0|       175.0|             216.0|        675.0|        90.0|  02/23/2019|      New
|[9581.0,175.0,216.0]|
|Jordan 1 Retro Hi...|18916.0|       120.0|             225.0|        455.0|       129.0|  02/15/2019|      New
|[18916.0,120.0,22...|
+--------------------+-------+------------+------------------+-------------+------------+------------+---------
+--------------------+
only showing top 20 rows
```

In order to prepare the data for a machine learning model, a new data frame variable called "finalized_df" will be created by choosing two columns from "output_df": "features" and "sales," with features acting as the input feature and sales as the target variable that will forecast based on the selected features.

In [27]:
```
finalized_df=output_df.select("features","sales")
```

In [28]:
```
finalized_df.show()
```

```
+--------------------+-------+
|            features|  sales|
+--------------------+-------+
|[28140.0,200.0,25...|28140.0|
|[17269.0,175.0,10...|17269.0|
|[37993.0,220.0,27...|37993.0|
|[14118.0,160.0,19...|14118.0|
|[23862.0,160.0,23...|23862.0|
|[1691.0,110.0,144.0]| 1691.0|
|[5154.0,185.0,165.0]| 5154.0|
|[21192.0,200.0,23...|21192.0|
|[7791.0,160.0,367.0]| 7791.0|
|[3870.0,200.0,290.0]| 3870.0|
|[24811.0,160.0,24...|24811.0|
|[9471.0,160.0,190.0]| 9471.0|
|[12791.0,160.0,22...|12791.0|
|[17107.0,160.0,22...|17107.0|
|[9513.0,200.0,243.0]| 9513.0|
| [4978.0,90.0,106.0]| 4978.0|
|[19839.0,160.0,30...|19839.0|
|[15958.0,160.0,20...|15958.0|
|[9581.0,175.0,216.0]| 9581.0|
|[18916.0,120.0,22...|18916.0|
+--------------------+-------+
only showing top 20 rows
```

Creating a Spark session to read data from a CSV file into a Spark DataFrame and displaying the first few rows of the DataFrame to give a quick overview of the data.

In [29]:
```
# Initialize a Spark session
spark = SparkSession.builder.appName("DataIntegration").getOrCreate()

# Read data from CSV file into a Spark DataFrame
data_df = spark.read.csv("AJs.csv", header=True, inferSchema=True)

# Show the first few rows of the DataFrame
data_df.show()
```

```
+--------------------+-------+------------+------------------+-------------+------------+------------+---------
+
|                name|  sales|retail_price|average_sale_price|highest_price|lowest_price|release_date|condition
|
+--------------------+-------+------------+------------------+-------------+------------+------------+---------
+
|Jordan 4 Retro Br...|28140.0|         200|             254.0|        520.0|       138.0|  05/04/2019|      New
|
|Jordan 1 Retro Hi...|17269.0|         175|            1013.0|       3000.0|       578.0|  05/11/2019|      New
|
|Jordan 11 Retro C...|37993.0|         220|             278.0|        500.0|       175.0|  12/08/2018|      New
|
|Jordan 1 Retro Hi...|14118.0|         160|             199.0|        500.0|        82.0|  04/12/2019|      New
|
|Jordan 1 Retro Hi...|23862.0|         160|             235.0|        405.0|        75.0|  02/15/2019|      New
|
|Jordan 1 Low SB M...| 1691.0|         110|             144.0|        232.0|       110.0|  06/17/2019|      New
|
|Jordan 11 Retro L...| 5154.0|         185|             165.0|        350.0|       115.0|  04/19/2019|      New
|
|Jordan 6 Retro Bl...|21192.0|         200|             239.0|        470.0|       150.0|  02/16/2019|      New
|
|Jordan 1 Retro Hi...| 7791.0|         160|             367.0|        750.0|       100.0|  02/24/2018|      New
|
|Jordan 7 Retro Pa...| 3870.0|         200|             290.0|       1200.0|       140.0|  06/15/2019|      New
|
|Jordan 1 Retro Hi...|24811.0|         160|             245.0|        800.0|       145.0|  11/17/2018|      New
|
|Jordan 1 Retro Hi...| 9471.0|         160|             190.0|        400.0|       100.0|  03/30/2019|      New
|
|Jordan 1 Retro Hi...|12791.0|         160|             225.0|        615.0|       100.0|  04/14/2018|      New
|
|Jordan 1 Retro Hi...|17107.0|         160|             220.0|        400.0|       105.0|  01/24/2019|      New
|
|Jordan 3 Retro Bl...| 9513.0|         200|             243.0|        355.0|       150.0|  02/17/2018|      New
|
|Jordan 1 Low Blac...| 4978.0|          90|             106.0|        300.0|        60.0|  04/01/2019|      New
|
|Jordan 1 Retro Hi...|19839.0|         160|             306.0|        634.0|       195.0|  12/14/2018|      New
|
|Jordan 1 Retro Hi...|15958.0|         160|             205.0|        431.0|        90.0|  12/27/2018|      New
|
|Jordan 1 Retro Hi...| 9581.0|         175|             216.0|        675.0|        90.0|  02/23/2019|      New
|
|Jordan 1 Retro Hi...|18916.0|         120|             225.0|        455.0|       129.0|  02/15/2019|      New
|
+--------------------+-------+------------+------------------+-------------+------------+------------+---------
+
only showing top 20 rows
```

Using PySparks's select function to choose particular columns from the dataset, and the show() method to display the dataframe's contents along with the selected columns.

In [30]:

```python
from pyspark.sql.functions import col

# Select the columns you want to work with
selected_columns_df = data_df.select(
    col("name"),
    col("sales"),
    col("retail_price"),
    col("average_sale_price"),
    col("highest_price"),
    col("lowest_price"),
    col("release_date"),
    col("condition")
)

# Show the DataFrame with selected columns
selected_columns_df.show()
```

```
+--------------------+-------+-----------+------------------+------------+------------+------------+---------
+
|                name|  sales|retail_price|average_sale_price|highest_price|lowest_price|release_date|condition
|
+--------------------+-------+-----------+------------------+------------+------------+------------+---------
+
|Jordan 4 Retro Br...|28140.0|        200|             254.0|       520.0|       138.0|  05/04/2019|      New
|
|Jordan 1 Retro Hi...|17269.0|        175|            1013.0|      3000.0|       578.0|  05/11/2019|      New
|
|Jordan 11 Retro C...|37993.0|        220|             278.0|       500.0|       175.0|  12/08/2018|      New
|
|Jordan 1 Retro Hi...|14118.0|        160|             199.0|       500.0|        82.0|  04/12/2019|      New
|
|Jordan 1 Retro Hi...|23862.0|        160|             235.0|       405.0|        75.0|  02/15/2019|      New
|
|Jordan 1 Low SB M...| 1691.0|        110|             144.0|       232.0|       110.0|  06/17/2019|      New
|
|Jordan 11 Retro L...| 5154.0|        185|             165.0|       350.0|       115.0|  04/19/2019|      New
|
|Jordan 6 Retro Bl...|21192.0|        200|             239.0|       470.0|       150.0|  02/16/2019|      New
|
|Jordan 1 Retro Hi...| 7791.0|        160|             367.0|       750.0|       100.0|  02/24/2018|      New
|
|Jordan 7 Retro Pa...| 3870.0|        200|             290.0|      1200.0|       140.0|  06/15/2019|      New
|
|Jordan 1 Retro Hi...|24811.0|        160|             245.0|       800.0|       145.0|  11/17/2018|      New
|
|Jordan 1 Retro Hi...| 9471.0|        160|             190.0|       400.0|       100.0|  03/30/2019|      New
|
|Jordan 1 Retro Hi...|12791.0|        160|             225.0|       615.0|       100.0|  04/14/2018|      New
|
|Jordan 1 Retro Hi...|17107.0|        160|             220.0|       400.0|       105.0|  01/24/2019|      New
|
|Jordan 3 Retro Bl...| 9513.0|        200|             243.0|       355.0|       150.0|  02/17/2018|      New
|
|Jordan 1 Low Blac...| 4978.0|         90|             106.0|       300.0|        60.0|  04/01/2019|      New
|
|Jordan 1 Retro Hi...|19839.0|        160|             306.0|       634.0|       195.0|  12/14/2018|      New
|
|Jordan 1 Retro Hi...|15958.0|        160|             205.0|       431.0|        90.0|  12/27/2018|      New
|
|Jordan 1 Retro Hi...| 9581.0|        175|             216.0|       675.0|        90.0|  02/23/2019|      New
|
|Jordan 1 Retro Hi...|18916.0|        120|             225.0|       455.0|       129.0|  02/15/2019|      New
|
+--------------------+-------+-----------+------------------+------------+------------+------------+---------
+
only showing top 20 rows
```

The describe() method is used to generate summary statistics for the full DataFrame 'data_df', including statistics for numeric columns like mean, standard deviation, minimum, maximum, and quartiles. The'summary_stats' Dataset contains the results.

By organizing the data in 'data_df' according to the "highest_price" column and computing the sum of "sales" for each group, it illustrates a more thorough analysis. The 'grouped_data' DataFrame stores the outcomes, which are accomplished using the groupBy and agg methods.

Using the show() method, providing the summary statistics and the outcomes of the analysis of the grouped data.

In [31]:

```python
# Example data analysis: Calculating summary statistics
summary_stats = data_df.describe()

# Performing more complex analysis or machine learning tasks
# For example: Grouping data and calculating aggregates
grouped_data = data_df.groupBy("highest_price").agg({"sales": "sum"})

# Show the summary statistics and analysis results
summary_stats.show()
grouped_data.show()
```

```
+-------+--------------------+-----------------+-----------------+------------------+------------------+-----
-------------+-----------+---------+
|summary|                name|            sales|     retail_price|average_sale_price|     highest_price|
lowest_price|release_date|condition|
+-------+--------------------+-----------------+-----------------+------------------+------------------+-----
-------------+-----------+---------+
|  count|                 998|              998|              998|               998|               998|
998|         998|      998|
|   mean|                null|1167.9749498997996|164.19735503560528|258.34869739478955|456.64028056112227|156.7
2444889779558|        null|     null|
| stddev|                null| 3001.948435533589|52.437779181248686|458.97689705408357| 767.6776411638872|320.3
1045187472034|        null|     null|
|    min|Air Jordan 1 Mid ...|              1.0|                 |              31.0|              45.0|
25.0|            |      New|
|    max|https://stockx.co...|          37993.0|               95|            8833.0|           10500.0|
7000.0|  12/31/2014|      new|
+-------+--------------------+-----------------+-----------------+------------------+------------------+-----
-------------+-----------+---------+


+-------------+----------+
|highest_price|sum(sales)|
+-------------+----------+
|        299.0|    6815.0|
|        596.0|     339.0|
|        305.0|     430.0|
|        184.0|    4417.0|
|        170.0|     748.0|
|        720.0|     175.0|
|        160.0|    4413.0|
|        169.0|     405.0|
|        486.0|    3077.0|
|        311.0|    2259.0|
|         70.0|      29.0|
|       2527.0|     107.0|
|        524.0|      50.0|
|        650.0|     820.0|
|        206.0|     835.0|
|        389.0|      50.0|
|        390.0|   10024.0|
|        249.0|   13167.0|
|        401.0|     934.0|
|        365.0|     170.0|
+-------------+----------+
only showing top 20 rows
```

Calculating average, minimum, and maximum retail_price for each distinct condition value utilizing aggregation analysis on the dataset based on the condition column using PySpark.

Result: - It's important to note that the minimum retail price for the 'New' criterion is empty since the 'min' function ran into an empty value for this condition, which can occur if there are rows with missing

In [32]:
```
from pyspark.sql.functions import avg, min, max

condition_stats = data_df.groupBy("condition").agg(
    avg("retail_price").alias("avg_retail_price"),
    min("retail_price").alias("min_retail_price"),
    max("retail_price").alias("max_retail_price")
)

condition_stats.show()
```

```
+---------+-----------------+----------------+----------------+
|condition| avg_retail_price|min_retail_price|max_retail_price|
+---------+-----------------+----------------+----------------+
|      New|164.1965376782077|                |              95|
|      new|            165.0|             165|             165|
+---------+-----------------+----------------+----------------+
```

Determining the relationship between sales and average retail price. A positive correlation means that as one variable rises, the other tends to rise as well, while a negative correlation means the opposite—that as one variable rises, the other tends to fall.

Result: - The correlation value, which is close to zero and approximately -0.0005716656765946814, indicates that there is only a weak linear link between sales and average sale price.

In [33]:
```
from pyspark.sql.functions import corr

correlation = data_df.stat.corr("sales", "average_sale_price")
print(f"Correlation between sales and average sale price: {correlation}")
```

```
Correlation between sales and average sale price: -0.0005716656765946814
```

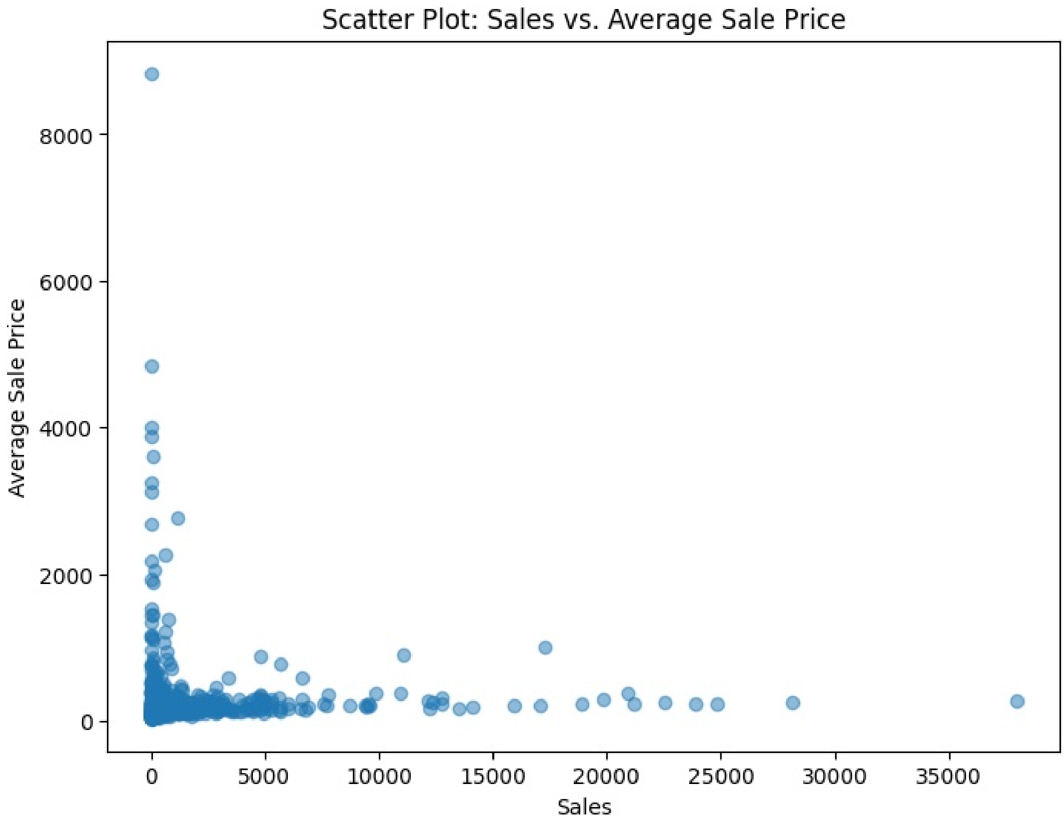
Data Visualisation

Making a scatter plot with the matplotlib library to show the connection between sales and average sale price. Setting the plot's figure size using plt.figure(figsize=(8, 6)) and then making a scatter plot with the average sale price on the y-axis and the sales column on the x-axis while using the alpha parameter to manage the point's transparency. Adding the plot's title and x-axis, and y-axis labels. And finally, showing the plot.

Result: - The X-axis (Sales) displays the quantity of a product that has been sold. It demonstrates that the sales volume ranges from 0 to 5,000 units.

The average price at which the products were sold is shown on the Y-axis (Average Sale Price). The range of average sale prices is shown to be between 0 and 2,000 units of money.

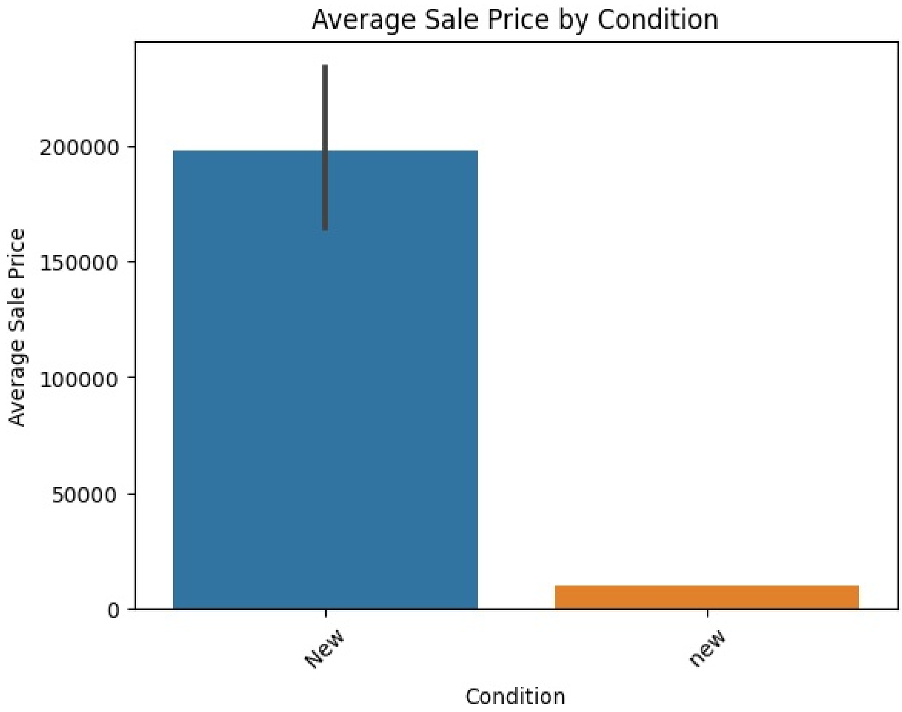
In [34]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))
plt.scatter(data_df.toPandas()["sales"], data_df.toPandas()["average_sale_price"], alpha=0.5)
plt.xlabel("Sales")
plt.ylabel("Average Sale Price")
plt.title("Scatter Plot: Sales vs. Average Sale Price")
plt.show()
```



Using the seaborn library to create a bar graph with the x-axis representing "condition" and the y-axis representing "average_sale_price." With the help of plt.xlabel("Condition"), plt.ylabel("Average Sale Price"), and plt.title("Average Sale Price by Condition"), labels and a title can be added to the plot. Plt.xticks(rotation=45) should be used for improved reading on the x-axis. Displaying the bar plot in the end.

Result: - We can observe from the result below that the majority of individuals like purchasing the newest Air Jordan products.

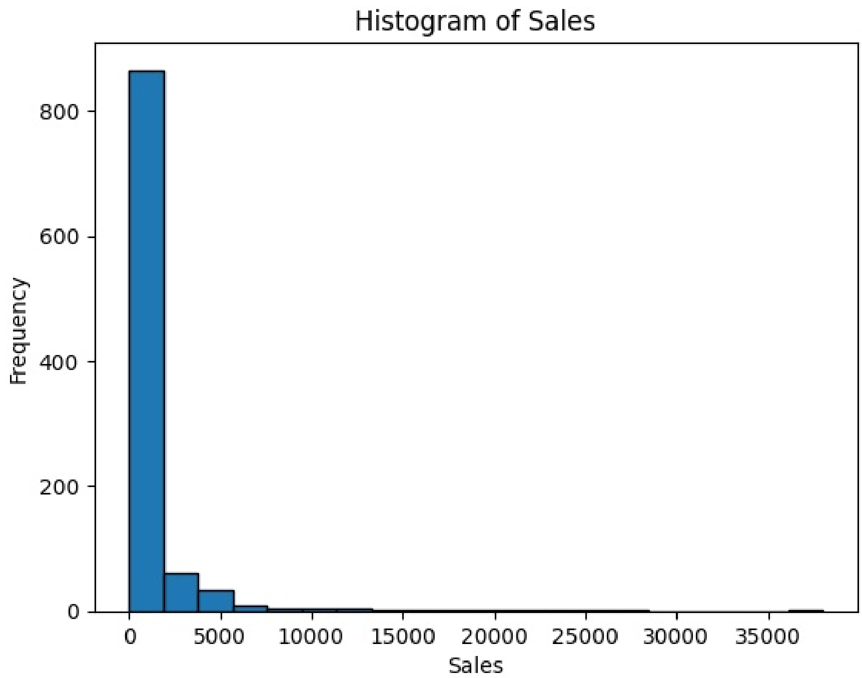
In [35]:
```
import seaborn as sns

sns.barplot(x='condition', y='average_sale_price', data=df)
plt.xlabel('Condition')
plt.ylabel('Average Sale Price')
plt.title('Average Sale Price by Condition')
plt.xticks(rotation=45)
plt.show()
```



Making a histogram for the sales column using matplotlib. Using bin to set the amount of bin parameters and edgecolor to change the color of the bar's edges. After that, the histogram's labels and title are specified, and finally, the histogram is displayed.

Result: - As we can see, the sales varied widely between 0 and 5000. This indicates that consumers choose to spend less money on goods.

In [36]:
```
import matplotlib.pyplot as plt

plt.hist(df['sales'], bins=20, edgecolor='k')
plt.xlabel('Sales')
plt.ylabel('Frequency')
plt.title('Histogram of Sales')
plt.show()
```



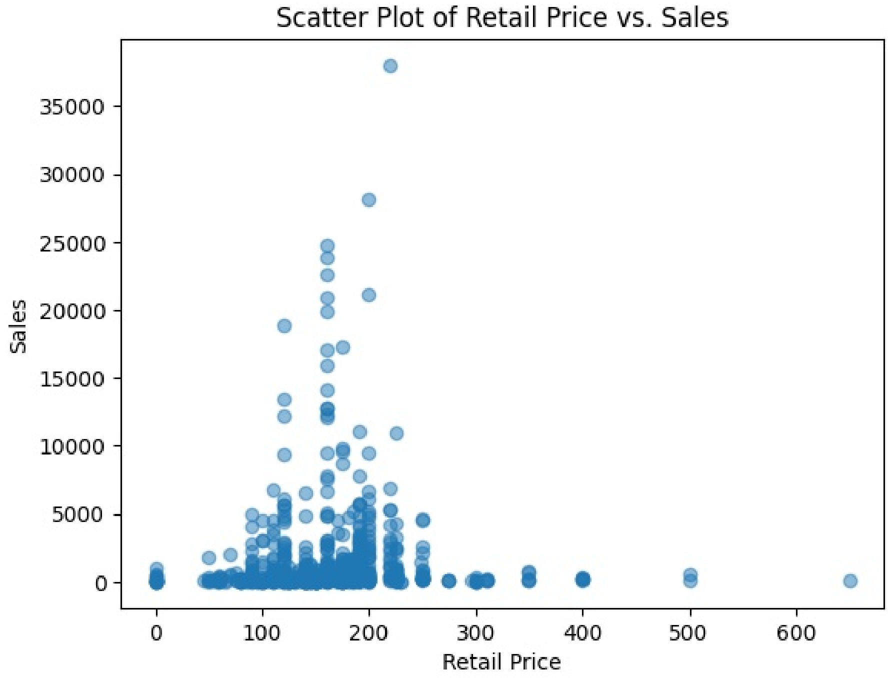
Using the dataset's retail_price and sales columns to create a scatter plot, to adjust the transparency of the points to 0.5 using the alpha value. The plot's label x-axis, y-axis, and title are then set. Displaying the plot, at last.

Result: - The vast majority of goods are sold for between $50 and$200 per unit at retail. The majority of products fall inside this pricing range.

The complete range of sales, from 0 to 5,000 units, is covered. It implies that a range of goods, regardless of their retail price, have various sales performances.

Outliers may reflect underperforming products with poor sales despite high prices or high performance products that sell well despite higher prices.

In [37]:
```
plt.scatter(df['retail_price'], df['sales'], alpha=0.5)
plt.xlabel('Retail Price')
plt.ylabel('Sales')
plt.title('Scatter Plot of Retail Price vs. Sales')
plt.show()
```

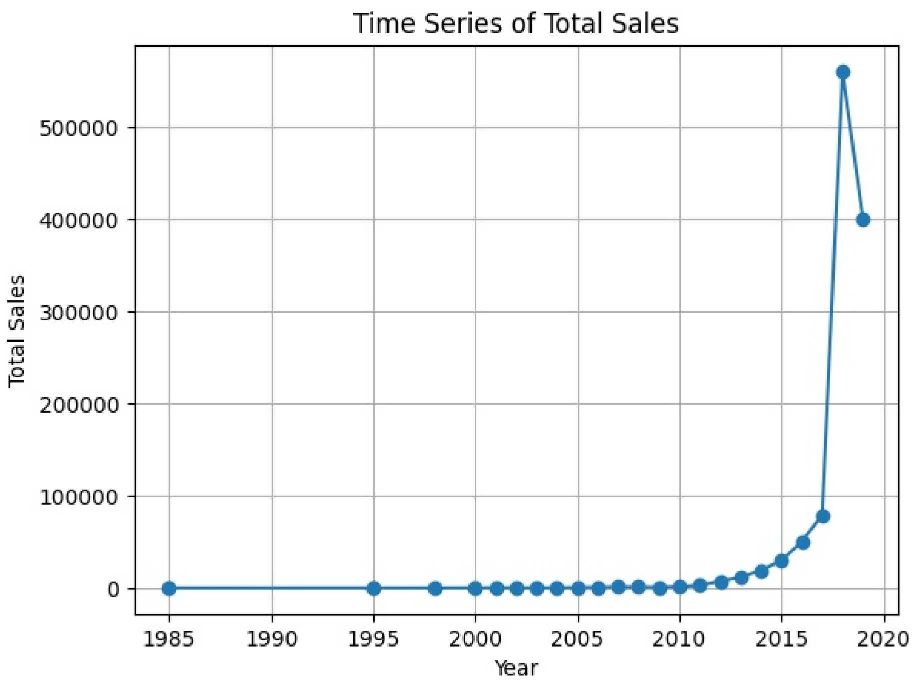


Taking the year value from the release_date column and assigning it to a new column called year. Following that, using .groupby('year')['sales'].To group the dataset by the year column and determine the overall sales for each year, using sum(). Using .plot(kind='line', marker='o'), to plot the total sales by year on a line. The plot's labels and title should then be added, and the grid lines should be added using plt.grid(True). Displaying the plot at the end.

Result: - The plot shows that overall product sales have increased over time, with a definite rising trend from 2015 to 2020.

The significant rise in total sales between 2015 and 2020 appears to be a reflection of significant sales growth throughout that time. This could be the result of a number of things, such as a rise in product demand, a wider market reach, successful marketing tactics, or the launch of well-liked products.

In [38]:
```
df['year'] = df['release_date'].dt.year  # Extract year from release date
sales_by_year = df.groupby('year')['sales'].sum()
sales_by_year.plot(kind='line', marker='o')
plt.xlabel('Year')
plt.ylabel('Total Sales')
plt.title('Time Series of Total Sales')
plt.grid(True)
plt.show()
```

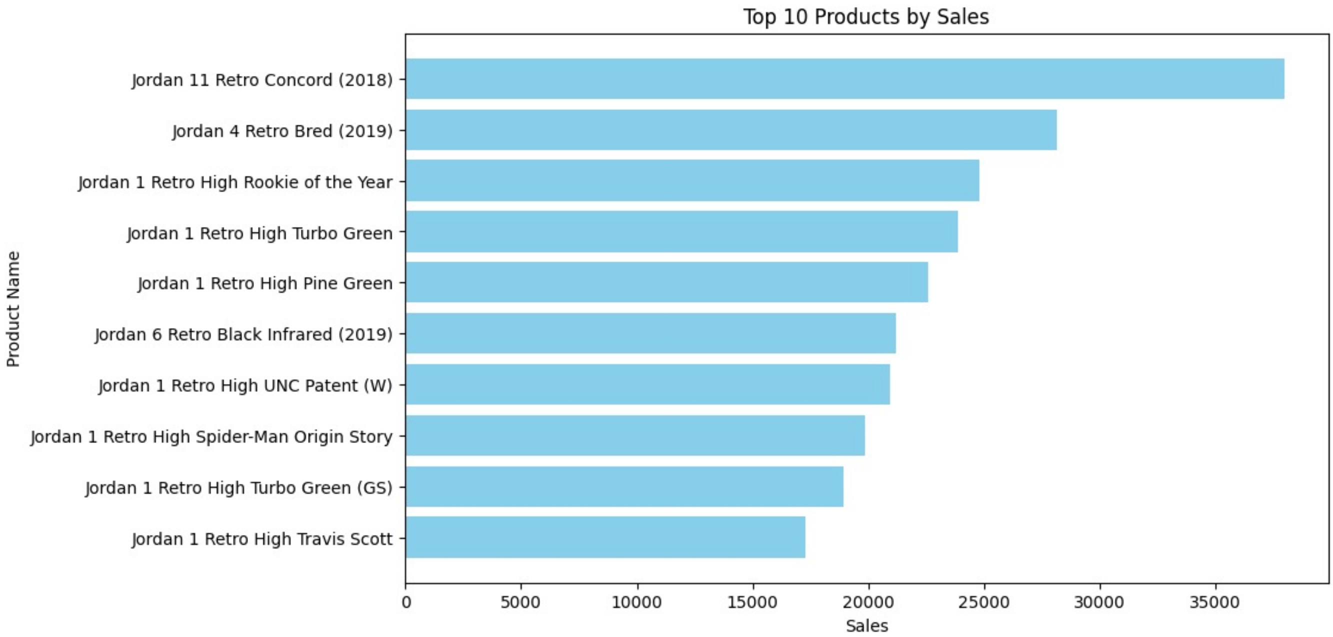


To choose the top 10 selling products, sort the dataset in descending order using the formula df.sort_values(by='sales', To choose the top 10 selling products, sort the dataset in descending order using the formula df.sort_values(by='sales', ascending=False).head(10). The plot's size should then be set, along with the labels for the x and y axes and the plot's title. then using plt.gca() to display the best-selling products at the top of the chart, use the invert_yaxis() function to invert the 'Product Name' y-axis. And finally, showing the plot.

Result: - Knowing which products are the best-sellers can aid the business in numerous ways. The biggest selling product, "Jordan 11 Retro Concord (2018)," has more than 35,000 sales.

Effective business management must identify and prioritize the product that sells the most since it has a big impact on revenue and profitability.

In [39]:
```
import matplotlib.pyplot as plt

# Sort the DataFrame by sales in descending order and select the top 10 products
top_10_products = df.sort_values(by='sales', ascending=False).head(10)

# Create a bar chart for the top 10 products
plt.figure(figsize=(10, 6))
plt.barh(top_10_products['name'], top_10_products['sales'], color='skyblue')
plt.xlabel('Sales')
plt.ylabel('Product Name')
plt.title('Top 10 Products by Sales')
plt.gca().invert_yaxis()  # Invert the y-axis to show the top-selling product at the top
plt.show()
```



Documentation

Data Preprocessing: -

The methodology used in this research includes a number of crucial procedures that helped us efficiently compile and analyze the dataset for the Air Jordan shoes.

1. **Data Import:** The dataset, which was in CSV format, was imported for the project's initial investigation using the Pandas library and later distributed data processing using Spark.

2. **Data investigation:** To understand the dataset's structure, shape, and column kinds, preliminary data exploration was carried out using the Pandas programming language. To gain understanding of the numerical columns, descriptive statistics were computed.

3. **Data Cleaning:** To guarantee data quality and dependability, data cleaning was an essential step. Important data cleansing steps comprised:

   The "release_date" column's trailing spaces have been removed, and it has been formatted as a datetime. handling "retail_price" column values that are blank or not numeric by setting them to 0.0. The cleaned dataset should be saved as "cleaned_AJs.csv" for future study.

4. **Data Integration:** To create a single dataset from information gathered from several sources, data integration was done.

   Involved were:

   Utilizing the DataFrame operations in PySpark to modify and alter the data. based on precise calculations, including new columns such as "average_sale_price" and "profit_margin". making a feature vector for machine learning with VectorAssembler. The combined dataset should be saved as "transformed_AJs.csv" for analysis.

Findings and Insights

The analysis of the combined dataset produced the following significant discoveries and insights:

1. **Product Name analysis:** The analysis of word counts for product names showed that different lengths of names exist, suggesting potential prospects for naming strategy optimization.
2. **Price Optimization:** The relationship between sales and average sale prices was modest, indicating that changing pricing on its own might not have a big effect on sales.
3. **Top-Selling Products:** Information about the products with the highest sales volumes was obtained by identifying the top-selling items.
4. **Sales Trends:** Time series analysis of overall product sales from 2015 to 2020 revealed a steady rise, pointing to expanding demand and market size.
5. **Product Condition:** It was discovered that product condition had an impact on average sale prices, with new products often fetching higher costs.

Recommendations

Here are some data-driven suggestions for boosting product sales based on the analysis:

1. **Product Naming Optimization:** Analyze product names further to find keywords or naming patterns that appeal to clients. Make product names more appealing and descriptive by optimizing them.
2. **Seasonal Marketing Campaigns:** Make use of historical sales trends to develop seasonal marketing initiatives or new product launches. Products launched during periods of high sales volume could perform better.
3. **Price Modifications:** Although there is no correlation between sales and average sale prices, you should take into account small price modifications for goods that have the potential to generate more sales.
4. **Product condition awareness:** Emphasize it in marketing and product descriptions as it affects average sale prices. Make sure buyers are informed about the state of the items they are buying.

Conclusion

In conclusion, the goal of this project was to improve the performance of sales of Air Jordan products on StockX by a thorough examination of sales data. The combined information provided important conclusions and insights about trends, best-selling items, and pricing concerns.

The study suggests improving product branding, using sales patterns in marketing campaigns, taking into account small pricing changes, and promoting product conditions to increase sales.

The knowledge acquired from this analysis can help company stakeholders make data-driven choices that will increase product sales and profitability in a highly competitive market for Air Jordan products.

In [1]:
```
# !jupyter nbconvert --to html /content/Data_Integration_Project.ipynb
```

```
[NbConvertApp] WARNING | pattern '/content/Data_Integration_Project.ipynb' matched no files
This application is used to convert notebook files (*.ipynb)
        to various other formats.

        WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.

Options
=======
The options below are convenience aliases to configurable class-options,
as listed in the "Equivalent to" description-line of the aliases.
To see all configurable class-options for some <cmd>, use:
    <cmd> --help-all

--debug
    set log level to logging.DEBUG (maximize logging output)
    Equivalent to: [--Application.log_level=10]
--show-config
    Show the application's configuration (human-readable format)
    Equivalent to: [--Application.show_config=True]
--show-config-json
```

```
        Show the application's configuration (json format)
        Equivalent to: [--Application.show_config_json=True]
    --generate-config
        generate default config file
        Equivalent to: [--JupyterApp.generate_config=True]
    -y
        Answer yes to any questions instead of prompting.
        Equivalent to: [--JupyterApp.answer_yes=True]
    --execute
        Execute the notebook prior to export.
        Equivalent to: [--ExecutePreprocessor.enabled=True]
    --allow-errors
        Continue notebook execution even if one of the cells throws an error and include the error message in the c
    ell output (the default behaviour is to abort conversion). This flag is only relevant if '--execute' was specif
    ied, too.
        Equivalent to: [--ExecutePreprocessor.allow_errors=True]
    --stdin
        read a single notebook file from stdin. Write the resulting notebook with default basename 'notebook.*'
        Equivalent to: [--NbConvertApp.from_stdin=True]
    --stdout
        Write notebook output to stdout instead of files.
        Equivalent to: [--NbConvertApp.writer_class=StdoutWriter]
    --inplace
        Run nbconvert in place, overwriting the existing notebook (only
                relevant when converting to notebook format)
        Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertApp.export_format=notebook --FilesWriter.
    build_directory=]
    --clear-output
        Clear output of current file and save in place,
                overwriting the existing notebook.
        Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertApp.export_format=notebook --FilesWriter.
    build_directory= --ClearOutputPreprocessor.enabled=True]
    --no-prompt
        Exclude input and output prompts from converted document.
        Equivalent to: [--TemplateExporter.exclude_input_prompt=True --TemplateExporter.exclude_output_prompt=True]
    --no-input
        Exclude input cells and output prompts from converted document.
                This mode is ideal for generating code-free reports.
        Equivalent to: [--TemplateExporter.exclude_output_prompt=True --TemplateExporter.exclude_input=True --Templ
    ateExporter.exclude_input_prompt=True]
    --allow-chromium-download
        Whether to allow downloading chromium if no suitable version is found on the system.
        Equivalent to: [--WebPDFExporter.allow_chromium_download=True]
    --disable-chromium-sandbox
        Disable chromium security sandbox when converting to PDF..
        Equivalent to: [--WebPDFExporter.disable_sandbox=True]
    --show-input
        Shows code input. This flag is only useful for dejavu users.
        Equivalent to: [--TemplateExporter.exclude_input=False]
    --embed-images
        Embed the images as base64 dataurls in the output. This flag is only useful for the HTML/WebPDF/Slides expo
    rts.
        Equivalent to: [--HTMLExporter.embed_images=True]
    --sanitize-html
        Whether the HTML in Markdown cells and cell outputs should be sanitized..
        Equivalent to: [--HTMLExporter.sanitize_html=True]
    --log-level=<Enum>
        Set the log level by value or name.
        Choices: any of [0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'ERROR', 'CRITICAL']
        Default: 30
        Equivalent to: [--Application.log_level]
    --config=<Unicode>
        Full path of a config file.
        Default: ''
        Equivalent to: [--JupyterApp.config_file]
    --to=<Unicode>
        The export format to be used, either one of the built-in formats
                ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebook', 'pdf', 'python', 'rst', 'script', '
    slides', 'webpdf']
                or a dotted object name that represents the import path for an
                ``Exporter`` class
        Default: ''
        Equivalent to: [--NbConvertApp.export_format]
    --template=<Unicode>
        Name of the template to use
        Default: ''
        Equivalent to: [--TemplateExporter.template_name]
    --template-file=<Unicode>
        Name of the template file to use
        Default: None
        Equivalent to: [--TemplateExporter.template_file]
    --theme=<Unicode>
        Template specific theme(e.g. the name of a JupyterLab CSS theme distributed
        as prebuilt extension for the lab template)
        Default: 'light'
        Equivalent to: [--HTMLExporter.theme]
    --sanitize_html=<Bool>
        Whether the HTML in Markdown cells and cell outputs should be sanitized.This
        should be set to True by nbviewer or similar tools.
```

```
        Default: False
        Equivalent to: [--HTMLExporter.sanitize_html]
    --writer=<DottedObjectName>
        Writer class used to write the
                                    results of the conversion
        Default: 'FilesWriter'
        Equivalent to: [--NbConvertApp.writer_class]
    --post=<DottedOrNone>
        PostProcessor class used to write the
                                    results of the conversion
        Default: ''
        Equivalent to: [--NbConvertApp.postprocessor_class]
    --output=<Unicode>
        overwrite base name use for output files.
                    can only be used when converting one notebook at a time.
        Default: ''
        Equivalent to: [--NbConvertApp.output_base]
    --output-dir=<Unicode>
        Directory to write output(s) to. Defaults
                                  to output to the directory of each notebook. To recover
                                  previous default behaviour (outputting to the current
                                  working directory) use . as the flag value.
        Default: ''
        Equivalent to: [--FilesWriter.build_directory]
    --reveal-prefix=<Unicode>
        The URL prefix for reveal.js (version 3.x).
                This defaults to the reveal CDN, but can be any url pointing to a copy
                of reveal.js.
                For speaker notes to work, this must be a relative path to a local
                copy of reveal.js: e.g., "reveal.js".
                If a relative path is given, it must be a subdirectory of the
                current directory (from which the server is run).
                See the usage documentation
                (https://nbconvert.readthedocs.io/en/latest/usage.html#reveal-js-html-slideshow)
                for more details.
        Default: ''
        Equivalent to: [--SlidesExporter.reveal_url_prefix]
    --nbformat=<Enum>
        The nbformat version to write.
                Use this to downgrade notebooks.
        Choices: any of [1, 2, 3, 4]
        Default: 4
        Equivalent to: [--NotebookExporter.nbformat_version]

    Examples
    --------

        The simplest way to use nbconvert is

                > jupyter nbconvert mynotebook.ipynb --to html

                Options include ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebook', 'pdf', 'python', 'r
    st', 'script', 'slides', 'webpdf'].

                > jupyter nbconvert --to latex mynotebook.ipynb

                Both HTML and LaTeX support multiple output templates. LaTeX includes
                'base', 'article' and 'report'.  HTML includes 'basic', 'lab' and
                'classic'. You can specify the flavor of the format used.

                > jupyter nbconvert --to html --template lab mynotebook.ipynb

                You can also pipe the output to stdout, rather than a file

                > jupyter nbconvert mynotebook.ipynb --stdout

                PDF is generated via latex

                > jupyter nbconvert mynotebook.ipynb --to pdf

                You can get (and serve) a Reveal.js-powered slideshow

                > jupyter nbconvert myslides.ipynb --to slides --post serve

                Multiple notebooks can be given at the command line in a couple of
                different ways:

                > jupyter nbconvert notebook*.ipynb
                > jupyter nbconvert notebook1.ipynb notebook2.ipynb

                or you can specify the notebooks list in a config file, containing::

                    c.NbConvertApp.notebooks = ["my_notebook.ipynb"]

                > jupyter nbconvert --config mycfg.py

    To see all available configurables, use `--help-all`.
```