Beginners_Guide_to PySpark

November 18, 2021

Table of Contents

1 Beginners Guide to PySpark

https://towardsdatascience.com/beginners-guide-to-pyspark-bbe3b553b79f

https://github.com/syamkakarla98/Beginners_Guide_to_PySpark

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

Create a spark session

```
[3]: from pyspark.sql import SparkSession

     spark = SparkSession.builder\
             .master("local[*]")\
             .appName('PySpark_Tutorial')\
             .getOrCreate()
```

1.1 Reading Data

1.1.1 Download Kaggle Movie Dataset

Use the Kaggle API Token(kaggle.json) to download the Movie Dataset

```
[54]: from google.colab import files

      ## Upload your kaggle json file (API Token)
```

```
files.upload()

!mkdir ~/.kaggle

!cp kaggle.json ~/.kaggle/

!chmod 600 ~/.kaggle/kaggle.json
```

```
<IPython.core.display.HTML object>
```

```
Saving kaggle.json to kaggle (1).json
mkdir: cannot create directory '/root/.kaggle': File exists
```

[ ]:

```
!kaggle datasets download -d dinnymathew/usstockprices
```

[ ]:

```
!ls
```

[ ]:

```
!mkdir data

!unzip usstockprices -d data
```

[ ]:

```
!ls -l data/
```

1.2 Import Modules

[9]:

```
from pyspark.sql import functions as f

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline
```

1.3 Read Data

[ ]:

```
# Before changing schema
b_data = spark.read.csv(
    'data/stocks_price_final.csv',
    sep = ',',
    header = True,
    )

b_data.printSchema()
```

[11]:

```
from pyspark.sql.types import *
```

```
data_schema = [
               StructField('_c0', IntegerType(), True),
               StructField('symbol', StringType(), True),
               StructField('data', DateType(), True),
               StructField('open', DoubleType(), True),
               StructField('high', DoubleType(), True),
               StructField('low', DoubleType(), True),
               StructField('close', DoubleType(), True),
               StructField('volume', IntegerType(), True),
               StructField('adjusted', DoubleType(), True),
               StructField('market.cap', StringType(), True),
               StructField('sector', StringType(), True),
               StructField('industry', StringType(), True),
               StructField('exchange', StringType(), True),
            ]

final_struc = StructType(fields=data_schema)
```

[12]:

```
data = spark.read.csv(
    'data/stocks_price_final.csv',
    sep = ',',
    header = True,
    schema = final_struc
    )
```

[ ]:

```
data.printSchema()
```

[ ]:

```
data.show(5)
```

[15]:

```
data = data.withColumnRenamed('market.cap', 'market_cap')
```

1.4 Inspect the data

[ ]:

```
# prints Schema of thte data
data.schema
```

[ ]:

```
data.dtypes
```

[ ]:

```
data.head(3)
```

[ ]:

```
data.show(5)
```

[ ]:

```
data.first()
```

[ ]:

```
data.describe().show()
```

[ ]:

```
data.columns
```

```
[ ]: data.count()
```

```
[ ]: data.distinct().count()
```

```
[ ]: data.printSchema()
```

1.5 Column Operations/Manipulations

```
[ ]: data = data.withColumn('date', data.data)

     data.show(5)
```

```
[ ]: data = data.withColumnRenamed('date', 'data_changed')

     data.show(5)
```

```
[ ]: data = data.drop('data_changed')

     data.show(5)
```

```
[ ]: data.select(['open', 'high', 'low', 'close', 'volume', 'adjusted']).describe().
      ↪show()
```

```
[ ]: data.groupBy('sector').count().show()
```

```
[32]: sec_x =  data.select(['sector', 'open', 'close', 'adjusted']).groupBy('sector').
       ↪mean().collect()
```

Convert the data into **list**

```
[ ]: for row in sec_x:
       print(list(row), end='\n')
```

Convert the data into **dictionary**

```
[ ]: for row in sec_x:
       print(row.asDict(), end='\n')
```

convert data into pandas **datafame**

```
[35]: sec_df =  data.select(['sector', 'open', 'close', 'adjusted']).
       ↪groupBy('sector').mean().toPandas()
```

```
[ ]: sec_df
```

```
[ ]: sec_df.plot(kind = 'bar', x='sector', y = sec_df.columns.tolist()[1:],␣
      ↪figsize=(12, 6))
```

Remove **basic industries** from the plot and view it again...

[ ]:

```
ind = list(range(12))
ind.pop(6)
sec_df.iloc[ind ,:].plot(kind = 'bar', x='sector', y = sec_df.columns.
 ↪tolist()[1:], figsize=(12, 6), ylabel = 'Stock Price', xlabel = 'Sector')
plt.show()
```

[ ]:

```
industries_x = data.select(['industry', 'open', 'close', 'adjusted']).
 ↪groupBy('industry').mean().toPandas()

industries_x.head()
```

[ ]:

```
industries_x.plot(kind = 'barh', x='industry', y = industries_x.columns.
 ↪tolist()[1:], figsize=(10, 50))
```

Remove **major chemicals** and **building products** to view the rest data clearly

[ ]:

```
q  = industries_x[(industries_x.industry != 'Major Chemicals') & (industries_x.
 ↪industry != 'Building Products')]

q.plot(kind = 'barh', x='industry', y = q.columns.tolist()[1:], figsize=(10,␣
 ↪50), xlabel='Stock Price', ylabel = 'Industry')

plt.show()
```

[ ]:

```
import pyspark.sql.functions as f

health = data.filter(f.col('sector') == 'Health Care')

health.show()
```

1.5.1 How to use Aggregation

[ ]:

```
from pyspark.sql.functions import col, min, max, avg, lit

data.groupBy("sector") \
    .agg(min("data").alias("From"),
         max("data").alias("To"),

         min("open").alias("Minimum Opening"),
         max("open").alias("Maximum Opening"),
         avg("open").alias("Average Opening"),

         min("close").alias("Minimum Closing"),
         max("close").alias("Maximum Closing"),
         avg("close").alias("Average Closing"),

         min("adjusted").alias("Minimum Adjusted Closing"),
        max("adjusted").alias("Maximum Adjusted Closing"),
        avg("adjusted").alias("Average Adjusted Closing"),

    ).show(truncate=False)
```

Get the min, max, avg data w.r.t sectors from **Jan 2019** to **Jan 2020**

[ ]:

```
data.filter( (col('data') >= lit('2019-01-02')) & (col('data') <=␣
 ↪lit('2020-01-31')) )\
    .groupBy("sector") \
    .agg(min("data").alias("From"),
        max("data").alias("To"),

        min("open").alias("Minimum Opening"),
        max("open").alias("Maximum Opening"),
        avg("open").alias("Average Opening"),

        min("close").alias("Minimum Closing"),
        max("close").alias("Maximum Closing"),
        avg("close").alias("Average Closing"),

        min("adjusted").alias("Minimum Adjusted Closing"),
        max("adjusted").alias("Maximum Adjusted Closing"),
        avg("adjusted").alias("Average Adjusted Closing"),

    ).show(truncate=False)
```

Plot the timeseries data od **technology** sector stock trade

[ ]:

```
tech = data.where(col('sector') == 'Technology').select('data', 'open',␣
 ↪'close', 'adjusted')

tech.show()
```

[ ]:

```
fig, axes = plt.subplots(nrows=3, ncols=1, figsize =(60, 30))

tech.toPandas().plot(kind = 'line', x = 'data', y='open', xlabel = 'Date␣
 ↪Range', ylabel = 'Stock Opening Price', ax = axes[0], color =␣
 ↪'mediumspringgreen')

tech.toPandas().plot(kind = 'line', x = 'data', y='close', xlabel = 'Date␣
 ↪Range', ylabel = 'Stock Closing Price', ax = axes[1], color = 'tomato')

tech.toPandas().plot(kind = 'line', x = 'data', y='adjusted', xlabel = 'Date␣
 ↪Range', ylabel = 'Stock Adjusted Price', ax = axes[2], color = 'orange')

plt.show()
```

[ ]:
```
data.select('sector').show(5)
```

[ ]:
```
data.select(['open', 'close', 'adjusted']).show(5)
```

[ ]:
```
data.filter(data.adjusted.between(100.0, 500.0)).show(5)
```

[ ]:
```
from pyspark.sql.functions import col, lit

data.filter( (col('data') >= lit('2020-01-01')) & (col('data') <=␣
 ↪lit('2020-01-31')) ).show(5)
```

[ ]:
```
data.select('open', 'close', f.when(data.adjusted >= 200.0, 1).otherwise(0)).
 ↪show(5)
```

[ ]:
```
data.select('sector',
            data.sector.rlike('^[B,C]').alias('Sector Starting with B or C')
            ).distinct().show()
```

[ ]:
```
data.select(['industry', 'open', 'close', 'adjusted']).groupBy('industry').
 ↪mean().show()
```