

# File Explorer Insight

## Problem Description

Get file insights of all files inside a folder path (eg. largest and smallest file in the folder, most recently updated file, number of file per extension type etc.)

## GitHub Repository Link

[Github Link](#)

## Requirements

1. Python 3.10.9 is a high-level, interpreted, general-purpose programming language. It contains various built-in methods for file handling and supports various file management libraries, making it highly attractive for our use-case.
2. Java 19.0.2 is a high-level object-oriented programming language. Java is necessary to properly run Kafka on the system.
3. Kafka 2.13-3.4.0 is an open source data streaming technology used for asynchronous messages and events processing. It provides a high-level, low latency platform for real-time data streams handling.
4. Kafka-python 2.0.2 is the python client for the kafka application. We utilize kafka-python for integration with our programming language.
5. Visual Studio Code 1.75.1 is a light-weight code editor with an extensive extension library.

## Installations (Mac)

### Python 3.10.9

Step 1:

Download the installer package from Python's official website. [Download Python](#)

Step 2:

Wait for the download to complete. Once it's finished, double-click the package to start the installation process.

Step 3:

Once the installation is complete, the installer will automatically open Python's installation directory in a new Finder window.

## Java 19.0.2

Step 1:

Visit the official Java documentation page and download JDK 19. [Java Downloads | Oracle](#)

Step 2:

Open the .dmg file and follow the instructions to install Java in any preferred location in the system.

Step 3:

Install homebrew from the official website for quick installations of packages on mac. [Homebrew](#)

Step 4:

Once homebrew is installed, type the following commands in the terminal: `brew tap caskroom/cask` and `brew install --cask oracle-jdk-javadoc`

Step 5:

Type the command `java -version` in the terminal to check if Java is properly installed on the system.

## Kafka 3.4.0

Step 1:

Download the binary installer with Scala version 2.13 from Apache Kafka's official website. [Download Kafka](#)

Step 2:

Unzip the downloaded .tar file to any desired location.

Step 3:

Run the command `brew install kafka` on the terminal.

## Kafka-python 2.0.2

Step 1:

Run the command `pip3 install kafka-python` on the terminal.

## Visual Studio Code 1.75.1

### Step 1:

Open the official Visual Studio Code website and download the latest version for mac. [Download Visual Studio Code - Mac, Linux, Windows](#)

### Step 2:

Click on Apple Silicon under the Mac icon to download VS Code's package installer for Mac in a ZIP file.

### Step 3:

Open finder in your mac and drag Visual Studio Code.app to the Applications folder, making it available in the macOS Launchpad.

## **Kafka server and topic setup**

Locate the kafka folder which was unzipped during downloads. Open config/server.properties file and add the lines `advertised.listeners=PLAINTEXT://localhost:9092` and `delete.topic.enable=true` to the file.

Next, open a terminal at the location of the kafka folder and use the commands `KAFKA_CLUSTER_ID="$(bin/kafka-storage.sh random-uuid)"`,

`bin/kafka-storage.sh format -t $KAFKA_CLUSTER_ID -c config/kraft/server.properties` and `bin/kafka-server-start.sh config/kraft/server.properties` to start the server. Subsequent server loggings won't require the first two statements.

Upon starting the kafka server, use the command `bin/kafka-topics.sh --create --bootstrap-server 127.0.0.1:9092 --replication-factor 1 --partitions 1 --topic file-input` to create a new topic 'file-input'

Use the command `bin/kafka-server-stop.sh config/kraft/server.properties` to stop the kafka servers after running the project.

## Python libraries utilized

1. Os module: In the Python script we have used the “**os**” module to navigate to the provided directories, finding files within folders and extracting file information.
2. Datetime module: We have used the “**datetime**” module to convert the UNIX time given by the os module to a more readable datetime format.
3. Json module: The “**json**” module is used to serialize the data into a json format before sending via the producer. It is also used to load back the data within the consumer side.
4. Kafka module: We use the **KafkaConsumer** and **KafkaProducer** functions from the “**kafka-python**” library to create producer and consumer clients.
5. Pprint module: We used the “**pprint**” module in order to display the json data format in a more readable way.

## Function Descriptions

These are the functions available to the consumer to call for file processing.

### find\_details

Description-

This function is used to extract file insights and store them into a dictionary which can be accessed by other functions for further processing.

Parameters-

The function accepts 3 parameters:

- DIR - This path of the directory or folder provided by the user. String type.
- files - Dictionary for the function to store file insights.
- recursion - Boolean value indicating whether or not to check within the folders found in the folder path itself.

Output-

Returns the files dictionary containing the file insights.

### number\_of\_files

Description-

This function is used to find the number of files held within the directory path specified by the user.

Parameters-

The function accepts 1 parameter:

- files - Dictionary containing file insights

Output-

Returns the number of files within the user-specified path.

### **number\_of\_extension**

Description-

This function is used to find the number of files with the extension specified by the user.

Parameters-

This function accepts 2 parameters:

- files - Dictionary containing file insights
- extension - Extension specified by the user to check against files. String type.

Output-

Returns the number of files with the same extension as what the user specified.

### **size**

Description-

This function is used to find the largest and smallest files on the user-specified path

Parameters-

This function accepts 1 parameter:

- files - Dictionary containing file insights

Output-

Returns a tuple containing information (file name, location and size) about the largest and smallest files.

### **modify**

Description-

This function is used to find the most recently modified file on the user-specified path

Parameters-

This function accepts 1 parameter:

- files - Dictionary containing file insights

Output-

Returns a tuple containing the name and location of the most recently modified time, along with the modification time in datetime format.

## Running the application

Open a terminal to the folder location of the project and run the producer file (producer.py) on the terminal. Provide the details asked. Then, open another terminal to the folder location of the project and run the consumer file (consumer.py) on the terminal.

The file details within the folder mentioned in the path given to the producer should be visible in the current terminal.

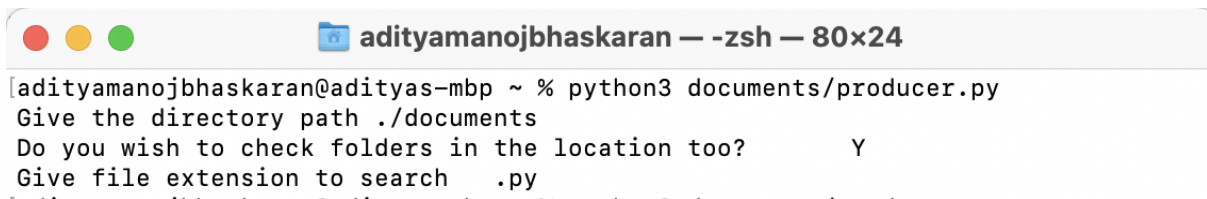
## Producer Details

```
4 producer = KafkaProducer(bootstrap_servers=['localhost:9092'],
5                           value_serializer=lambda x:
6                           dumps(x).encode('utf-8'))
```

## Consumer Details

```
6 consumer = KafkaConsumer(
7     'file-input',
8     bootstrap_servers=['localhost:9092'],
9     auto_offset_reset='earliest',
10    enable_auto_commit=True,
11    auto_commit_interval_ms=1000,
12    group_id='output-test',
13    value_deserializer=lambda x: loads(x.decode('utf-8')),
14    consumer_timeout_ms = 1000
15 )
```

## Producer's Terminal



```
adityamanojbhaskaran — -zsh — 80x24
[adityamanojbhaskaran@adityas-mbp ~ % python3 documents/producer.py
Give the directory path ./documents
Do you wish to check folders in the location too? Y
Give file extension to search .py
```

## Consumer's Terminal

```
adityamanojbhaskaran — -zsh — 80x24
[adityamanojbhaskaran@adityas-mbp ~ % python3 documents/consumer.py
ConsumerRecord(topic='file-input', partition=0, offset=5, timestamp=167739915853
3, timestamp_type=0, key=None, value=['./documents', 'Y', '.py'], headers=[], ch
ecksum=None, serialized_key_size=-1, serialized_value_size=27, serialized_header
_size=-1)
{'DS_Store': ['./documents/GitHub/.DS_Store',
'',
'DS_Store',
datetime.datetime(2023, 2, 6, 17, 25, 33, 729947),
datetime.datetime(2023, 2, 6, 17, 25, 33, 727417),
datetime.datetime(2023, 2, 4, 21, 42, 23, 703091),
6148],
'.env': ['./documents/GitHub/User-authentication-login/.env',
'',
'.env',
datetime.datetime(2023, 2, 13, 21, 49, 52, 528944),
datetime.datetime(2023, 2, 13, 21, 49, 52, 516160),
datetime.datetime(2023, 2, 13, 21, 49, 52, 515922),
170],
'.gitignore': ['./documents/GitHub/User-authentication-login/.gitignore',
'',
'.gitignore',
datetime.datetime(2023, 2, 13, 21, 49, 52, 529037),
datetime.datetime(2023, 2, 13, 21, 49, 52, 516487),
datetime.datetime(2023, 2, 6, 16, 55, 23, 43456),
3650],
'validate.html': ['./documents/GitHub/User-authentication-login/templates/valid
ate',
'.html',
'validate.html',
datetime.datetime(2023, 2, 7, 9, 59, 30, 588597),
datetime.datetime(2023, 2, 7, 9, 59, 30, 578332),
datetime.datetime(2023, 2, 7, 9, 59, 30, 578120),
2243],
'verify.html': ['./documents/GitHub/User-authentication-login/templates/verify'
,
'.html',
'verify.html',
datetime.datetime(2023, 2, 13, 21, 49, 52, 530180),
datetime.datetime(2023, 2, 13, 21, 49, 52, 519774),
datetime.datetime(2023, 2, 13, 21, 49, 52, 519590),
3436]}
Number of files in directory is 153
Number of files with the extension .py are 9
Largest file is Aruba VIA.zip in ./documents/Aruba VIA with size 7486661
Smallest file is .localized in ./documents/.localized with size 0
Most recently modified file is files.cpython-310.pyc in ./documents/__pycache__/
files.cpython-310 with time 2023-02-26 13:42:40.620924
```

## Troubleshooting

If the error 'Address is already in use' is encountered, it indicates that there already exists some process running on the port we are trying to use. To solve this issue, either utilize a different port when running the kafka servers or kill the processes currently running on the port.

To do the first, use the command `bin/kafka-topics.sh --create --bootstrap-server 127.0.0.1:<port-number> --replication-factor 1 --partitions 1 --topic file-input`

For the second, use the command `lsof -i tcp:<port-number>` to get a list of processes running on the port number. Then use the command `kill -9 <pid>` to kill the relevant processes. Run the command to start the server again to start the Kafka servers.

To list all the topics currently on the given server and port, use the command `bin/kafka-topics.sh --list --bootstrap-server 127.0.0.1:<port-number>`. You can remove any topics from the list using the command `bin/kafka-topics.sh --bootstrap-server 127.0.0.1:<port-number> --delete --topic <topic-name>`.

Use the command `bin/kafka-console-consumer.sh --bootstrap-server=127.0.0.1:9092 --from-beginning --topic file-input max-messages 10` to check the messages along with the entire message queue sent to the consumer by the producer.