Scheduled Fetch

For this exercise, we're going to be applying what we've learned from the previous lessons in a simple DAG that fetches data from wikipedia.

The API we use gets the number of view of a wikipedia page. For this exercise we will get the number of wikipedia views of a particular page.

Pageviews API

The documentation for wikimedia api can be found here:

https://wikimedia.org/api/rest_v1/

The form of the request of pageview per articles is like this:

GET: https://wikimedia.org/api/rest_v1/metrics/pageviews/per-article/en.wikipedia/all-access/all-agents/[ARTICLE]/daily/[START]/[END]

And the response is of the form:

```
{
   "items":[
      {
         "project":"en.wikipedia",
         "article":"Bitcoin",
         "granularity":"daily",
         "timestamp":"2015100100",
         "access":"all-access",
         "agent":"all-agents",
         "views":7942
      },
      {
         "project":"en.wikipedia",
         "article":"Bitcoin",
         "granularity":"daily",
         "timestamp":"2015100200",
         "access":"all-access",
         "agent":"all-agents",
         "views":7540
      }
   ]
}
```

The granularity for the pageview aggregation is "daily" pageviews.

We'll only be getting the pageviews per day, so the and date parameters must be the same day (`execution_date`).

Installing Libraries

To install any library for an airflow worker in `astro-cli`, you can put the library in the `requirements.txt` file in the root project directory.

`requirement.txt`

```
pandas
```

You can put other libraries you need here (ex. `scikit-learn`)

Then restart your `astro` project.

Dag Schedule

We'll schedule the DAG to start from 5 days ago and execute daily:

```
with DAG(
    dag_id="5-scheduled-fetch",
    start_date=airflow.utils.dates.days_ago(5),
    schedule_interval="@daily"
) as dag:
```

Fetch bitcoin views

We'll use `curl` on the `BashOperator` to make a GET request on the wikipedia pageviews endpoint.

```
fetch_bitcoin_views = BashOperator(
        task_id="fetch_bitcoin_views",
        bash_command="curl -o /tmp/{{ ds_nodash }}.json -L
'https://wikimedia.org/api/rest_v1/metrics/pageviews/per-article/en.wikipedia/all-access/all-agents/Bitcoin/daily/{{ ds_nodash }}00/{{ ds_nodash }}00'"
    )
```

Notice how we use the `ds_nodash` (YYYYMMDD) template variable to both:

a.) Save the response as a json file. b.) API parameters, since wikimedia api needs the start and end date in this format.

Convert to CSV

We'll use the `pandas` library to load the json array of views into a dataframe, then save only the timestamp and views to the csv file.

```
def _save_to_csv(ds_nodash):
    with open(f"/tmp/{ds_nodash}.json") as f:
        views_json = json.load(f)
        df = pd.DataFrame(views_json['items'])
        df[['timestamp', 'views']].to_csv(f"/tmp/{ds_nodash}.csv", index=False)

...

save_to_csv = PythonOperator(
        task_id="save_to_csv",
        python_callable=_save_to_csv,
        provide_context=True
    )
```

Challenge Exercise

Let's say that we want to correlate the views of the Bitcoin wikipedia page with it's price for that day.

You can use the `coindesk` api found here to get the closing price of bitcoin for a day:

https://www.coindesk.com/coindesk-api

Specifically, use this endpoint and modify the start and end date:

https://api.coindesk.com/v1/bpi/historical/close.json?start=2013-09-01&end=2013-09-05

Conver the bitcoin price for that day into CSV as well.

Finally, join both csv files according to their day:

End result: 1 CSV file with both the wikipedia views and the bitcoin price for that day.

Note:

You don't need to use `pandas` if you don't want to. As long as you have the final csv file of views and price.