Project Report

Project name: "COVID-19 Data Integration, Analysis, and Visualization Platform"
Student: Sergejs Vlasovs
Bootcamp: Big Data Engineering

Task 1. Use Snowflake Marketplace and get COVID-19 free dataset. Setup Snowflake Resource monitors.

An account was created on Snowflake. Resource monitoring was configured using the SQL code below:

```
CREATE OR REPLACE RESOURCE MONITOR limiter
  WITH CREDIT_QUOTA = 400
  TRIGGERS ON 40 PERCENT DO NOTIFY
           ON 80 PERCENT DO SUSPEND
           ON 100 PERCENT DO SUSPEND_IMMEDIATE;
```

Task 2. Data Exploration and Enhancement.

After studying the provided COVID-19 dataset, the idea emerged to compare the ratio of deaths to cases with the number of vaccinations during the peak periods of the pandemic. The goal was to examine the impact of vaccination during peak periods on mortality rates.

A decision was made to search for additional data to complement the table COVID19_EPIDEMIOLOGICAL_DATA/PUBLIC/ECDC_GLOBAL with information beyond December 14, 2020. Several datasets from various sources were reviewed. The dataset 'Coronavirus (COVID-19) In-depth Dataset' was found on the kaggle.com platform.
The dataset was downloaded and uploaded to the storage of the AWS S3 service:
s3://sv-web-01/dataset/owid-covid-data (1).csv

At AWS, in the IAM (Identity and Access Management) service, a role and policy were configured to enable the uploading of files to Snowflake:

Policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:GetObjectVersion",
                "s3:DeleteObject",
                "s3:DeleteObjectVersion"
            ],
            "Resource": "arn:aws:s3:::sv-web-01/dataset/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:GetBucketLocation"
```

```
            ],
            "Resource": "arn:aws:s3:::sv-web-01",
            "Condition": {
                "StringLike": {
                    "s3:prefix": [
                        "dataset/*"
                    ]
                }
            }
        }
    ]
}
```

Role:

```
{

    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::926672153406:user/nqwh0000-s"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "sts:ExternalId":
"XE75838_SFCRole=2_yLMxFkb4djczbzQXZvf9lDz8mfw="
                }
            }
        }
    ]
}
```

The next step involved creating integration with the storage, setting up the database, stage, and performing other necessary actions for the dataset upload:

-- Creating integration with the S3 service

```
CREATE STORAGE INTEGRATION S3_int
  TYPE = EXTERNAL_STAGE
  STORAGE_PROVIDER = 'S3'
  ENABLED = TRUE
  STORAGE_AWS_ROLE_ARN = 'arn:aws:iam::533266996171:role/snoflake_role'
  STORAGE_ALLOWED_LOCATIONS = ('s3://sv-web-01/dataset/')

DESC INTEGRATION S3_int;
```

-- Creating of DB to operate with datasets

```
CREATE DATABASE kaggle_dataset;
```

-- Creating of CSV format file

```
CREATE FILE FORMAT my_csv_format
  TYPE = 'CSV'
  FIELD_OPTIONALLY_ENCLOSED_BY = '"'
  COMPRESSION = 'AUTO';
```

-- Creating of Stage

```
USE SCHEMA kaggle_dataset.public;
CREATE STAGE S3_stage_01
  STORAGE_INTEGRATION = S3_int
  URL = 's3://sv-web-01/dataset/'
  FILE_FORMAT = my_csv_format;
```

-- Creating the table with certain columns

```
CREATE TABLE kaggle_covid19 (
    iso_code      VARCHAR,
    continent     VARCHAR,
    location      VARCHAR,
    date          DATE,
    total_cases   FLOAT,
    new_cases     FLOAT,
    total_deaths  FLOAT,
    new_deaths    FLOAT
);
```

-- Copying and transforming of Kaggle's dataset

```
COPY INTO KAGGLE_COVID19 (iso_code, continent, location, date, total_cases,
new_cases, total_deaths, new_deaths)
FROM @S3_stage_02
ON_ERROR = 'CONTINUE'
FILE_FORMAT = (FORMAT_NAME = 'my_csv_format');
```

After the upload, both datasets were merged, resulting in a new table:

-- Changing of length of ISO3166_1 from 2 to 16777216

```
ALTER TABLE COVID19_EPIDEMIOLOGICAL_DATA.PUBLIC.ECDC_GLOBAL
MODIFY COLUMN ISO3166_1 VARCHAR(16777216);
```

-- Create a new table as a result of merging

```
CREATE TABLE IF NOT EXISTS covid19_MERGED_TABLE AS
SELECT *
FROM KAGGLE_DATASET.PUBLIC.ECDC_GLOBAL;
```

-- Merge operation to update and insert records into the new table

```
MERGE INTO covid19_MERGED_TABLE AS target
USING KAGGLE_DATASET.PUBLIC.KAGGLE_COVID19 AS source
ON target.DATE = source.dae AND target.ISO3166_1 = source.iso_code
WHEN MATCHED AND source.date >= '2020-12-15'
THEN UPDATE SET
    target.CASES = source.total_cases,
    target.CASES_SINCE_PREV_DAY = source.new_cases,
    target.CONTINENTEXP = source.continent,
    target.COUNTRY_REGION = source.location,
    target.DATE = source.date,
    target.DEATHS = source.total_deaths,
    target.DEATHS_SINCE_PREV_DAY = source.new_deaths
WHEN NOT MATCHED AND source.date >= '2020-12-15'
THEN INSERT (CASES, CASES_SINCE_PREV_DAY, CONTINENTEXP, COUNTRY_REGION,
DATE, DEATHS, DEATHS_SINCE_PREV_DAY, ISO3166_1)
VALUES (source.total_cases, source.new_cases, source.continent,
source.location, source.date, source.total_deaths, source.new_deaths,
LEFT(source.iso_code, 3));
```

After obtaining the new table and conducting a more detailed examination, it became apparent that, even in its expanded form, it provides too little data for analysis, as the latest information on cases and deaths is dated May 12, 2021. Following additional scrutiny of the original dataset, the decision was made to utilize the existing table of weekly cases and deaths since its latest entries are dated the end of 2023. However, since the vaccination table contains daily statistics rather than weekly, it became necessary to convert this table from a daily format to a weekly one:

-- Creating, formating and transforming vaccination table

```
CREATE OR REPLACE TABLE KAGGLE_DATASET.PUBLIC.OWID_VACCINATIONS AS
SELECT
    COUNTRY_REGION,
    ISO3166_1,
    SUM(PEOPLE_VACCINATED) AS PEOPLE_VACCINATED_WEEKLY,
    SUM(PEOPLE_FULLY_VACCINATED) AS PEOPLE_FULLY_VACCINATED_WEEKLY,
    SUM(DAILY_VACCINATIONS) AS WEEKLY_VACCINATIONS,
    MIN(DATE_TRUNC('week', DATE)) AS DATE
FROM COVID19_EPIDEMIOLOGICAL_DATA.PUBLIC.OWID_VACCINATIONS
WHERE DATE >= '2020-12-07'
GROUP BY COUNTRY_REGION, ISO3166_1, DATE_TRUNC('week', DATE);
```

Afterward, the table of weekly cases was prepared (transformed):

-- Creating and transforming weekly cases table

```
CREATE OR REPLACE TABLE KAGGLE_DATASET.PUBLIC.vacc_MERGED_TABLE AS
SELECT
    COUNTRY_REGION,
    CONTINENTEXP,
    ISO3166_1,
    CASES_WEEKLY,
    DEATHS_WEEKLY,
    DATE
FROM COVID19_EPIDEMIOLOGICAL_DATA.PUBLIC.ECDC_GLOBAL_WEEKLY;
```

The final step involved merging both tables and making some changes and improvements as described below:

-- Merging of two previous tables

```
CREATE OR REPLACE TABLE KAGGLE_DATASET.PUBLIC.MERGED_RESULT AS
SELECT
    VM.*,
    OV.PEOPLE_FULLY_VACCINATED_WEEKLY,
    OV.PEOPLE_VACCINATED_WEEKLY,
    OV.WEEKLY_VACCINATIONS
FROM KAGGLE_DATASET.PUBLIC.VACC_MERGED_TABLE VM
LEFT JOIN KAGGLE_DATASET.PUBLIC.OWID_VACCINATIONS OV
ON VM.DATE = OV.DATE AND VM.ISO3166_1 = OV.ISO3166_1
WHERE VM.DATE <= '2023-11-20';
```

-- Changing of values type

```
ALTER TABLE KAGGLE_DATASET.PUBLIC.MERGED_RESULT
ADD COLUMN DEATHS_PER_1000CASES_RATIO NUMBER(10,1);

ALTER TABLE KAGGLE_DATASET.PUBLIC.MERGED_RESULT
RENAME COLUMN CASES_DEATH_RATIO TO DEATHS_PER_1000CASES_RATIO;
```

-- Getting death/cases per 1000 ppl ratio

```
SET DEATHS_PER_1000CASES_RATIO = CASE
    WHEN CASES_WEEKLY = 0 THEN NULL -- to handle division by zero
    ELSE (DEATHS_WEEKLY / CASES_WEEKLY)*1000
END;
```

-- Add new columns

```
ALTER TABLE KAGGLE_DATASET.PUBLIC.MERGED_RESULT
ADD COLUMN WEEKLY_VAC_per1000 NUMBER(20,1);
ALTER TABLE KAGGLE_DATASET.PUBLIC.MERGED_RESULT
ADD COLUMN PEOPLE_FULLY_VAC_WEEKLY_per1000 NUMBER(20,1);
ALTER TABLE KAGGLE_DATASET.PUBLIC.MERGED_RESULT
ADD COLUMN PEOPLE_VAC_WEEKLY_per1000 NUMBER(20,1);
```

-- Update values in new columns

```
UPDATE KAGGLE_DATASET.PUBLIC.MERGED_RESULT
SET
    WEEKLY_VAC_per1000 = WEEKLY_VACCINATIONS / 1000,
    PEOPLE_FULLY_VAC_WEEKLY_per1000 = PEOPLE_FULLY_VACCINATED_WEEKLY /
1000,
    PEOPLE_VAC_WEEKLY_per1000 = PEOPLE_VACCINATED_WEEKLY / 1000;
```

Next was proceeded with visualizing the obtained data, as described below.

Task 3. Data Modeling in NoSQL.

An Ubuntu server was deployed on the AWS EC2 service. MongoDB was installed on it.

The database stores information about all the datasets studied during the selection process. The schema is provided below:

```
c19_project> db.resources.findOne({_id:4})
{
  _id: 4,
  resource: 'redivis',
  title: 'Coronavirus COVID-19 Global Cases',
  link: 'https://redivis.com/datasets/rxta-4v35cgyzf',
  last_update: '12/07/2020',
  comment: 'This data comes from the data repository for t
rsity Center for Systems Science and Engineering (JHU CSSE
gency to track reported cases in real-time.'
}
```

Data insertion:

```
# Create a document
resource_data = {
    "_id": 4,
    "resource": "redivis",
    "title": "Coronavirus COVID-19 Global Cases",
    "link": "https://redivis.com/datasets/rxta-4v35cgyzf",
    "last_update": "12/07/2020",
    "comment": "This data comes from the data repository for the 2019 Novel
Coronavirus Visual Dashboard operated by the Johns Hopkins University Center
for Systems Science and Engineering (JHU CSSE). This database was created in
response to the Coronavirus public health emergency to track reported cases in
real-time."
}
# Insert the document into the collection
result = coll.insert_one(resource_data)

print(f"Inserted document ID: {result.inserted_id}")
```

Task 4. API Development with Python:

The following APIs were deployed for working with MongoDB and Snowflake:

MongoDB:

```
import pymongo
from pymongo import MongoClient

# Connect to MongoDB
client = pymongo.MongoClient('mongodb:// 3.120.174.0:27017')
# Create or get the database
db = client.c19_project
coll = db.resources

client.close()
```

Snowflake:

```
# connector to Snowflake
conn = snowflake.connector.connect(
    user = 'SERGEI',
    password = 'changemeP1s',
    account = 'tgdbsvg-lh06896',
    warehouse = 'COMPUTE_WH',
    database = 'KAGGLE_DATASET',
    schema = 'PUBLIC',
    role = 'ACCOUNTADMIN'
)

cur = conn.cursor()
cur.execute('SELECT * FROM MERGED_RESULT')

# Storing the data into variable
data = cur.fetchall()
vac_death = pd.DataFrame(data, columns = [x[0] for x in cur.description])
vac_death.to_csv('vac_death_data.csv', index=False)
# print(vac_death)

cur.close()
conn.close()
```

Task 5. Interactive Visualization with Python:

For data visualization, the Dash framework is being used:

```
import dash
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output
import plotly.express as px
import pandas as pd
import snowflake.connector

vac_death['DATE'] = pd.to_datetime(vac_death['DATE'])

# Create Dash app
app = dash.Dash(__name__)

# Define layout
app.layout = html.Div([
    dcc.Dropdown(
        id='country-dropdown',
        options=[{'label': country, 'value': country} for country in
vac_death['COUNTRY_REGION'].unique()],
        value='Austria',
        multi=False
    ),
    dcc.Graph(id='ratio-vac-chart'),
])
# Define callback to update the line plot based on selected country
@app.callback(
    Output('ratio-vac-chart', 'figure'),
    [Input('country-dropdown', 'value')]
)

def update_scatter_plot(selected_country):
    # Filter data based on user selection or other criteria
    filtered_df = vac_death[vac_death['COUNTRY_REGION'] == selected_country]
    # Create scatter plot
    fig = px.scatter(
        filtered_df, x='DATE', y=['DEATHS_PER_1000CASES_RATIO',
'WEEKLY_VAC_PER1000'],
        title='Vaccination and Death cases', labels={'value': 'Cases per
thousand people', 'DATE': 'Time'}
    )
    return fig

# Run the app
if __name__ == '__main__':
    app.run_server(debug=True)
```

Post-Bootcamp actions

The following actions were taken to ensure an opportunity for further work with the dataset.

1. Data set was saved into CSV file:

```
vac_death.to_csv('modified_data.csv', index=False)
```

2. Original Python code was modified to work with the local CSV file:

```
df_pandas = pd.read_csv('modified_data.csv')
```

3. The Docker file was created:

```
FROM python:alpine
WORKDIR /project_app
COPY . .
RUN pip install --no-cache-dir dash plotly pandas
EXPOSE 8050
CMD ["python", "project.py", "--host", "0.0.0.0", "--port", "8050"]
```

4. Project folder with CSV, Docker file, and Python file was pushed to GitHub:

https://github.com/vl-sergei/Final_project/tree/main/project_app

To create the Docker image, execute the following command in the terminal from the project folder:

docker build . -t final_project

To deploy the container, use the following command:

docker run -it -p 8050:8050 final_project

Plots can be accessed at http://127.0.0.1:8050