DataFrame Import

Xander Palermo

This file extracts an existing the f1db from the Postgres dump provided by the F1DB project. This file then saves a handful of pickled DataFrames to be used in other notebooks

```
import json
import pickle

import sqlalchemy
import pandas as pd
```

Give access to notebook for Postgres Database actions

```
# Replace these with your actual credentials
db_username = "postgres"
db_password = input("Database Password: ")
db_host = "localhost"
db_port = "5432"
db_name = "f1db"

# Connect to postgres database
engine =
sqlalchemy.create_engine(f"postgresql+psycopg2://{db_username}:
{db_password}@{db_host}:{db_port}/{db_name}")
```

Import Info

Functionality of these tools are saved to *DataFrameImport.py* so that they can be utilized in other Notebooks.

```
from DataFrameImport import *
```

Table Look Up Functionality

Create a function that can be used to look up specific tables, or list all table

```
schema_info_path = "resources/f1db/"
schema_file = "f1db.schema.json"

with open(schema_info_path + schema_file, 'r') as f:
    info_json = json.load(f)

schema_names = list(dict(info_json["properties"].items()).keys())
schema_descriptions = [x["description"] for x in
list(dict(info_json["properties"].items()).values())]
```

```
schema_info = dict(zip(schema_names, schema_descriptions))

with open("resources/info/schema_info_dump.pkl", "wb") as file:
    pickle.dump(schema_info, file)

# Prints alias of every table
def list_schemas():
    for s in schema_info.keys():
        print(s)

#Prints table alias and description, or provide specific table alias
to get single description
def get_schema_info(t: str = ""):
    if t in schema_info.keys():
        print(t)
        print(schema_info[t])
        print()
    else:
        for t in schema_info.keys():
            get_schema_info(t)

# ex:
# list_schemas()
# get_schema_info()
```

Column Look Up Functionality

Create an object that holds information of all schema descriptions and data types (accessed using dot notation)

```
# Placeholder class that is used to attach dynamically named
attributes
class Null:
    def __str__(self):
        return "\n".join(f"{k} = {v}" for k, v in vars(self).items())
+ "\n"

column_info = info_json['definitions']

with engine.connect() as c:
    query = c.execute(sqlalchemy.text(""
                                      "SELECT table_name "
                                      "FROM information_schema.tables
"
                                      "WHERE table_schema = 'public' "
                                      "    AND table_type = 'BASE
TABLE';"))
    table_names = query.all()
```

```
table_names = [list(i)[0] for i in table_names]

table_var_names = list(map(lambda x: x.replace('_', '
').title().replace(' ', ''), table_names))

schema = Null()

for table in table_var_names:
    try:
        #Create column object
        attr_names = list(column_info[table]['properties'].keys())
        exec(f"schema.{table} = Null()")


        for attr_name in attr_names:
            # Assign type and description name
            exec(f"schema.{table}.{attr_name} = Null()")

            try:
                attr_type = column_info[table]['properties']
[attr_name]["type"]
                exec(f"schema.{table}.{attr_name}.type = attr_type")
            except KeyError:
                pass

            try:
                attr_desc = column_info[table]['properties']
[attr_name]["description"]
                exec(f"schema.{table}.{attr_name}.description =
attr_desc")
            except KeyError:
                pass
    except KeyError:
        pass

with open("resources/info/schema_columns_info_dump.pkl", "wb") as
file:
    pickle.dump(schema, file)


# ex:
# schema.{table_name}.{column_name(optional)}
# print(schema.Continent)           #Gives info on all columns in
DataFame Continent
# print(schema.Continent.id)        #Gives info on specific column
(id) in DataFrame Continent
```

Pickle Tables

Saving tables as pickled DataFrames so that work can be independently from existence of a postgres database.

```
path = "resources/pickled_tables/"
extension = ".plk"

for table in table_names:
    with engine.connect() as conn:
        query = f"SELECT * FROM {table}"

        dataFrame = pd.read_sql_query(query, conn)
        dataFrame.to_pickle(f'{path}{table}{extension}')

# Read using the following code
# table = dataFrame you want to load
# with open(f"resources/pickled_tables/{table}.pkl","rb") as file:
#   dataFrame = pickle.load(file)
```