Assignment_5

February 3, 2023

Task 5D: Working with pandas Data Frames Part 2

Name: Prateek Singh

Student number: 221218743

Email: singhprate@deakin.edu.au

Unit: SIT731

This task allows us to explore how we can perform similar tasks in SQL and Pandas library, thus it also allows us learn tranferable concepts across both tech stacks. I have also used timeit to perform fime analysis to benchmark sql and pure pandas execution time. [It is interesting to note that pure pandas outperform sql way in case of joins performed in later tasks.]

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import numpy as np
import sqlite3
import tempfile, os.path
import timeit
```

[2]:
```
df_airlines = pd.read_csv("Resource_Data/airlines.csv", comment="#")
df_airports = pd.read_csv("Resource_Data/airports.csv", comment="#")
df_flights = pd.read_csv("Resource_Data/flights.csv", comment="#")
df_planes = pd.read_csv("Resource_Data/planes.csv", comment="#")
df_weather = pd.read_csv("Resource_Data/weather.csv", comment="#")
```

[3]:
```
dbfile = os.path.join(tempfile.mkdtemp(), "nycflights13.db")
conn = sqlite3.connect(dbfile)
```

[4]:
```
df_airlines.to_sql("airlines", conn, index=False)
df_airports.to_sql("airports", conn, index=False)
df_flights.to_sql("flights", conn, index=False)
df_planes.to_sql("planes", conn, index=False)
df_weather.to_sql("weather", conn, index=False)
```

[4]:
```
26130
```

1. SELECT DISTINCT engine FROM planes

[5]:
```
def sql_version():
    query = "SELECT DISTINCT engine FROM planes"
    task1_sql = pd.read_sql_query(query, conn)
    return task1_sql

def pandas_version():
    task1_my = df_planes['engine'].drop_duplicates().reset_index(drop=True)
    task1_my = task1_my.to_frame()
    return task1_my

task1_sql = sql_version()
task1_my = pandas_version()
pd.testing.assert_frame_equal(task1_sql, task1_my)

sql_time = timeit.timeit(sql_version, number=100)
#To compare time I'm doing each version 100 times to capture accurate difference
pandas_time = timeit.timeit(pandas_version, number=100)
print(f"SQL version took {sql_time:.6f} seconds")
print(f"Pandas version took {pandas_time:.6f} seconds")
```

```
SQL version took 0.095311 seconds
Pandas version took 0.027895 seconds
```

2. SELECT DISTINCT type, engine FROM planes

[6]:
```
def sql_version():
    query = "SELECT DISTINCT type, engine FROM planes"
    task1_sql = pd.read_sql_query(query, conn)
    return task1_sql

def pandas_version():
    task1_my = df_planes[['type', 'engine']].drop_duplicates().
 ↪reset_index(drop=True)
    return task1_my

task1_sql = sql_version()
task1_my = pandas_version()
pd.testing.assert_frame_equal(task1_sql, task1_my)

sql_time = timeit.timeit(sql_version, number=100)
#To compare time I'm doing each version 100 times to capture accurate difference
pandas_time = timeit.timeit(pandas_version, number=100)
print(f"SQL version took {sql_time:.6f} seconds")
print(f"Pandas version took {pandas_time:.6f} seconds")
```

```
SQL version took 0.130043 seconds
Pandas version took 0.138504 seconds
```

3. SELECT COUNT(*), engine FROM planes GROUP BY engine

[7]:
```
def sql_version():
    query = "SELECT COUNT(*), engine FROM planes GROUP BY engine"
    task1_sql = pd.read_sql_query(query, conn)
    return task1_sql

def pandas_version():
    task1_my = df_planes.groupby('engine').size().reset_index(name='COUNT(*)')
    task1_my = task1_my[['COUNT(*)', 'engine']]
    return task1_my

task1_sql = sql_version()
task1_my = pandas_version()
pd.testing.assert_frame_equal(task1_sql, task1_my)

#To compare time I'm doing each version 100 times to capture accurate difference
sql_time = timeit.timeit(sql_version, number=100)
pandas_time = timeit.timeit(pandas_version, number=100)
print(f"SQL version took {sql_time:.6f} seconds")
print(f"Pandas version took {pandas_time:.6f} seconds")
```

```
SQL version took 0.185039 seconds
Pandas version took 0.133389 seconds
```

4. SELECT COUNT(*), engine, type FROM planes GROUP BY engine, type

[8]:
```
def sql_version():
    query = "SELECT COUNT(*), engine, type FROM planes GROUP BY engine, type"
    task1_sql = pd.read_sql_query(query, conn)
    return task1_sql

def pandas_version():
    task1_my = df_planes.groupby(['engine', 'type']).size().
 ↪reset_index(name='COUNT(*)')
    task1_my = task1_my[['COUNT(*)', 'engine', 'type']]
    return task1_my

task1_sql = sql_version()
task1_my = pandas_version()
pd.testing.assert_frame_equal(task1_sql, task1_my)

sql_time = timeit.timeit(sql_version, number=100)
#To compare time I'm doing each version 100 times to capture accurate difference
pandas_time = timeit.timeit(pandas_version, number=100)
print(f"SQL version took {sql_time:.6f} seconds")
print(f"Pandas version took {pandas_time:.6f} seconds")
```

```
SQL version took 0.272021 seconds
Pandas version took 0.222611 seconds
```

5. SELECT MIN(year), AVG(year), MAX(year), engine, manufacturer FROM planes GROUP BY engine, manufacturer

[9]:
```
def sql_version():
    query = "SELECT MIN(year), AVG(year), MAX(year), engine, manufacturer FROM␣
 ↪planes GROUP BY engine, manufacturer"
    task1_sql = pd.read_sql_query(query, conn)
    return task1_sql

def pandas_version():
    grouped = df_planes.groupby(['engine', 'manufacturer'])
    task1_my = grouped['year'].agg(['min', 'mean', 'max']).reset_index()
    task1_my.rename(columns={'min': 'MIN(year)', 'mean': 'AVG(year)', 'max':␣
 ↪'MAX(year)'}, inplace=True)
    task1_my = task1_my[['MIN(year)', 'AVG(year)', 'MAX(year)', 'engine',␣
 ↪'manufacturer']]

    return task1_my

task1_sql = sql_version()
task1_my = pandas_version()
pd.testing.assert_frame_equal(task1_sql, task1_my)

sql_time = timeit.timeit(sql_version, number=100)
#To compare time I'm doing each version 100 times to capture accurate difference
pandas_time = timeit.timeit(pandas_version, number=100)
print(f"SQL version took {sql_time:.6f} seconds")
print(f"Pandas version took {pandas_time:.6f} seconds")
```

```
SQL version took 0.431289 seconds
Pandas version took 0.294459 seconds
```

6. SELECT * FROM planes WHERE speed IS NOT NULL

[10]:
```
def sql_version():
    query = "SELECT * FROM planes WHERE speed IS NOT NULL"
    task1_sql = pd.read_sql_query(query, conn)
    return task1_sql

def pandas_version():
    task1_my = df_planes[df_planes['speed'].notnull()].reset_index(drop=True)
    return task1_my

task1_sql = sql_version()
task1_my = pandas_version()
pd.testing.assert_frame_equal(task1_sql, task1_my)

sql_time = timeit.timeit(sql_version, number=100)
#To compare time I'm doing each version 100 times to capture accurate difference
```

```
pandas_time = timeit.timeit(pandas_version, number=100)
print(f"SQL version took {sql_time:.6f} seconds")
print(f"Pandas version took {pandas_time:.6f} seconds")
```

```
SQL version took 0.132846 seconds
Pandas version took 0.027316 seconds
```

7. SELECT tailnum FROM planes WHERE seats BETWEEN 150 AND 190 AND year >= 2012

[11]:
```
def sql_version():
    query = "SELECT tailnum FROM planes WHERE seats BETWEEN 150 AND 190 AND␣
 ↪year >= 2012"
    task1_sql = pd.read_sql_query(query, conn)
    return task1_sql

def pandas_version():
    task1_my = df_planes[(df_planes['seats'] >= 150) &
                         (df_planes['seats'] <= 190) &
                         (df_planes['year'] >= 2012)]['tailnum']
    task1_my = task1_my.reset_index(drop=True).to_frame()
    return task1_my

task1_sql = sql_version()
task1_my = pandas_version()
pd.testing.assert_frame_equal(task1_sql, task1_my)

sql_time = timeit.timeit(sql_version, number=100)
#To compare time I'm doing each version 100 times to capture accurate difference
pandas_time = timeit.timeit(pandas_version, number=100)
print(f"SQL version took {sql_time:.6f} seconds")
print(f"Pandas version took {pandas_time:.6f} seconds")
```

```
SQL version took 0.078752 seconds
Pandas version took 0.067646 seconds
```

8. SELECT tailnum, manufacturer, seats FROM planes WHERE manufacturer IN ("BOEING", "AIRBUS", "EMBRAER") AND seats>390

[12]:
```
def sql_version():
    query = ("SELECT tailnum, manufacturer, seats FROM planes "
             "WHERE manufacturer IN (\"BOEING\", \"AIRBUS\", \"EMBRAER\") AND␣
 ↪seats>390")
    task1_sql = pd.read_sql_query(query, conn)
    return task1_sql

def pandas_version():
```

```
    task1_my = df_planes[(df_planes['manufacturer'].isin(["BOEING", "AIRBUS",␣
 ↪"EMBRAER"])) & (df_planes['seats'] > 390)][['tailnum', 'manufacturer',␣
 ↪'seats']]
    return task1_my.reset_index(drop=True)

task1_sql = sql_version()
task1_my = pandas_version()
pd.testing.assert_frame_equal(task1_sql, task1_my)

sql_time = timeit.timeit(sql_version, number=100)
#To compare time I'm doing each version 100 times to capture accurate difference
pandas_time = timeit.timeit(pandas_version, number=100)
print(f"SQL version took {sql_time:.6f} seconds")
print(f"Pandas version took {pandas_time:.6f} seconds")
```

```
SQL version took 0.124904 seconds
Pandas version took 0.101751 seconds
```

9. SELECT DISTINCT year, seats FROM planes WHERE year >= 2012 ORDER BY year ASC, seats DESC

[13]:
```
def sql_version():
    query = ("SELECT DISTINCT year, seats FROM planes WHERE year >= 2012 ORDER␣
 ↪BY year ASC, seats DESC")
    task1_sql = pd.read_sql_query(query, conn)
    return task1_sql

def pandas_version():
    task1_my = df_planes[df_planes['year'] >= 2012][['year', 'seats']].
 ↪drop_duplicates().sort_values(
        by=['year', 'seats'], ascending=[True, False])
    return task1_my.reset_index(drop=True)

task1_sql = sql_version()
task1_my = pandas_version()
pd.testing.assert_frame_equal(task1_sql, task1_my)

sql_time = timeit.timeit(sql_version, number=100)
#To compare time I'm doing each version 100 times to capture accurate difference
pandas_time = timeit.timeit(pandas_version, number=100)
print(f"SQL version took {sql_time:.6f} seconds")
print(f"Pandas version took {pandas_time:.6f} seconds")
```

```
SQL version took 0.088858 seconds
Pandas version took 0.198935 seconds
```

10. SELECT DISTINCT year, seats FROM planes WHERE year >= 2012 ORDER BY seats DESC, year ASC

[14]:
```
def sql_version():
    query = ("SELECT DISTINCT year, seats FROM planes WHERE year >= 2012 ORDER␣
 ↪BY seats DESC, year ASC")
    task1_sql = pd.read_sql_query(query, conn)
    return task1_sql

def pandas_version():
    task1_my = df_planes[df_planes['year'] >= 2012][['year', 'seats']].
 ↪drop_duplicates().sort_values(
        by=['seats', 'year'],
        ascending=[False, True])
    return task1_my.reset_index(drop=True)

task1_sql = sql_version()
task1_my = pandas_version()
pd.testing.assert_frame_equal(task1_sql, task1_my)

sql_time = timeit.timeit(sql_version, number=100)
#To compare time I'm doing each version 100 times to capture accurate difference
pandas_time = timeit.timeit(pandas_version, number=100)
print(f"SQL version took {sql_time:.6f} seconds")
print(f"Pandas version took {pandas_time:.6f} seconds")
```

```
SQL version took 0.084111 seconds
Pandas version took 0.192135 seconds
```

11. SELECT manufacturer, COUNT(*) FROM planes WHERE seats > 200 GROUP BY manufacturer

[15]:
```
def sql_version():
    query = ("SELECT manufacturer, COUNT(*) FROM planes WHERE seats > 200 GROUP␣
 ↪BY manufacturer")
    task1_sql = pd.read_sql_query(query, conn)
    return task1_sql

def pandas_version():
    task1_my = df_planes[df_planes['seats'] > 200].groupby('manufacturer').
 ↪size().reset_index(name='COUNT(*)')
    return task1_my

task1_sql = sql_version()
task1_my = pandas_version()
pd.testing.assert_frame_equal(task1_sql, task1_my)

sql_time = timeit.timeit(sql_version, number=100)
#To compare time I'm doing each version 100 times to capture accurate difference
pandas_time = timeit.timeit(pandas_version, number=100)
print(f"SQL version took {sql_time:.6f} seconds")
```

```
print(f"Pandas version took {pandas_time:.6f} seconds")
```

```
SQL version took 0.088228 seconds
Pandas version took 0.096110 seconds
```

12. SELECT manufacturer, COUNT() *FROM planes GROUP BY manufacturer HAVING COUNT() > 10*

[16]:
```
def sql_version():
    query = ("SELECT manufacturer, COUNT() FROM planes GROUP BY manufacturer␣
 ↪HAVING COUNT() > 10")
    task1_sql = pd.read_sql_query(query, conn)
    return task1_sql

def pandas_version():
    task1_my = df_planes.groupby('manufacturer').filter(
        lambda x: x['manufacturer'].count() > 10).
 ↪groupby('manufacturer')['manufacturer'].count().reset_index(
        name='COUNT()')
    return task1_my


task1_sql = sql_version()
task1_my = pandas_version()
pd.testing.assert_frame_equal(task1_sql, task1_my)

sql_time = timeit.timeit(sql_version, number=100)
#To compare time I'm doing each version 100 times to capture accurate difference
pandas_time = timeit.timeit(pandas_version, number=100)
print(f"SQL version took {sql_time:.6f} seconds")
print(f"Pandas version took {pandas_time:.6f} seconds")
```

```
SQL version took 0.178891 seconds
Pandas version took 0.436890 seconds
```

13. SELECT manufacturer, COUNT() *FROM planes WHERE seats > 200 GROUP BY manufacturer HAVING COUNT() > 10*

[17]:
```
def sql_version():
    query = ("SELECT manufacturer, COUNT(*) FROM planes WHERE seats > 200 GROUP␣
 ↪BY manufacturer HAVING COUNT(*) > 10")
    task1_sql = pd.read_sql_query(query, conn)
    return task1_sql

def pandas_version():
    task1_my = df_planes[df_planes['seats'] > 200].groupby('manufacturer').
 ↪size().reset_index(name='COUNT(*)')
    task1_my = task1_my[task1_my['COUNT(*)'] > 10].reset_index(drop=True)
    return task1_my
```

```
task1_sql = sql_version()
task1_my = pandas_version()
pd.testing.assert_frame_equal(task1_sql, task1_my)

sql_time = timeit.timeit(sql_version, number=100)
#To compare time I'm doing each version 100 times to capture accurate difference
pandas_time = timeit.timeit(pandas_version, number=100)
print(f"SQL version took {sql_time:.6f} seconds")
print(f"Pandas version took {pandas_time:.6f} seconds")
```

```
SQL version took 0.095159 seconds
Pandas version took 0.128881 seconds
```

14. SELECT manufacturer, COUNT(*) AS howmany FROM planes GROUP BY manufacturer ORDER BY howmany DESC LIMIT 5

[18]:
```
def sql_version():
    query = ("SELECT manufacturer, COUNT(*) AS howmany FROM planes GROUP BY␣
 ↪manufacturer ORDER BY howmany DESC LIMIT 5")
    task1_sql = pd.read_sql_query(query, conn)
    return task1_sql

def pandas_version():
    task1_my = df_planes.groupby('manufacturer').size().
 ↪reset_index(name='howmany').sort_values(
        by='howmany',
        ascending=False).head(5)
    task1_my = task1_my.reset_index(drop=True)
    return task1_my

task1_sql = sql_version()
task1_my = pandas_version()
pd.testing.assert_frame_equal(task1_sql, task1_my)

sql_time = timeit.timeit(sql_version, number=100)
#To compare time I'm doing each version 100 times to capture accurate difference
pandas_time = timeit.timeit(pandas_version, number=100)
print(f"SQL version took {sql_time:.6f} seconds")
print(f"Pandas version took {pandas_time:.6f} seconds")
```

```
SQL version took 0.181498 seconds
Pandas version took 0.112822 seconds
```

15. SELECT flights.*, planes.year AS plane_year, planes.speed AS plane_speed, planes.seats AS plane_seats FROM flights LEFT JOIN planes ON flights.tailnum=planes.tailnum

[19]:
```
def sql_version():
    query = ("SELECT flights.*, planes.year AS plane_year, planes.speed AS "
             "plane_speed, planes.seats AS plane_seats FROM flights LEFT JOIN␣
 ↪planes ON flights.tailnum=planes.tailnum")
    task1_sql = pd.read_sql_query(query, conn)
    return task1_sql

def pandas_version():
    task1_my = pd.merge(df_flights, df_planes, how="left", on="tailnum")
    task1_my = task1_my.rename(columns={'year_x': 'year', 'speed':␣
 ↪'plane_speed',
                                        'seats': 'plane_seats', 'year_y':␣
 ↪'plane_year'})

    columns = ['year', 'month', 'day', 'dep_time', 'sched_dep_time',␣
 ↪'dep_delay',
               'arr_time', 'sched_arr_time', 'arr_delay', 'carrier', 'flight',
               'tailnum', 'origin', 'dest', 'air_time', 'distance', 'hour',␣
 ↪'minute',
               'time_hour', 'plane_year', 'plane_speed', 'plane_seats']

    task1_my = task1_my[columns]
    return task1_my


task1_sql = sql_version()
task1_my = pandas_version()
pd.testing.assert_frame_equal(task1_sql, task1_my)
print ("Done assert")

sql_time = timeit.timeit(sql_version, number=1)
#This takes too long for 100
pandas_time = timeit.timeit(pandas_version, number=1)
print(f"SQL version took {sql_time:.6f} seconds")
print(f"Pandas version took {pandas_time:.6f} seconds")
```

```
Done assert
SQL version took 3.525982 seconds
Pandas version took 0.305607 seconds
```

16. SELECT planes.*, *airlines.* FROM (SELECT DISTINCT carrier, tailnum FROM flights) AS cartail INNER JOIN planes ON cartail.tailnum=planes.tailnum INNER JOIN airlines ON cartail.carrier=airlines.carrier

[20]:
```
def sql_version():
    query = ("SELECT planes.*, airlines.* FROM (SELECT DISTINCT carrier,␣
 ↪tailnum FROM flights) "
```

```
            "AS cartail INNER JOIN planes ON cartail.tailnum=planes.tailnum␣
 ↪INNER JOIN airlines "
            "ON cartail.carrier=airlines.carrier")
    task1_sql = pd.read_sql_query(query, conn)
    return task1_sql

def pandas_version():
    df1 = df_flights[['carrier', 'tailnum']].drop_duplicates()
    df2 = df1.merge(df_planes, on='tailnum')
    task1_my = df2.merge(df_airlines, on='carrier')
    #task1_my = task1_my.rename(columns={'year_x': 'year'})

    columns = ['tailnum', 'year', 'type', 'manufacturer', 'model', 'engines',
               'seats', 'speed', 'engine', 'carrier', 'name']
    return task1_my[columns]

task1_sql = sql_version()
task1_my = pandas_version()

task1_sql.sort_values(by=['tailnum','year', 'carrier'], inplace=True)
task1_my.sort_values(by=['tailnum', 'year', 'carrier'], inplace=True)
task1_sql = task1_sql.reset_index(drop=True)
task1_my = task1_my.reset_index(drop=True)

pd.testing.assert_frame_equal(task1_sql, task1_my)

sql_time = timeit.timeit(sql_version, number=100)
#To compare time I'm doing each version 100 times to capture accurate difference
pandas_time = timeit.timeit(pandas_version, number=100)
print(f"SQL version took {sql_time:.6f} seconds")
print(f"Pandas version took {pandas_time:.6f} seconds")
```

```
SQL version took 31.029198 seconds
Pandas version took 5.942164 seconds
```

2 Additional Tasks for Postgraduate (SIT731) Students (*)

17. An additional SQL query to implement: SELECT flights2.*, *atemp, ahumid FROM (SELECT* FROM flights WHERE origin='EWR') AS flights2 LEFT JOIN (SELECT year, month, day, AVG(temp) AS atemp, AVG(humid) AS ahumid FROM weather WHERE origin='EWR' GROUP BY year, month, day) AS weather2 ON flights2.year=weather2.year AND flights2.month=weather2.month AND flights2.day=weather2.day

[21]:
```
def sql_version():
    query = ("SELECT flights2.*, atemp, ahumid FROM ( SELECT * FROM flights␣
 ↪WHERE origin='EWR' ) "
            "AS flights2 LEFT JOIN ( SELECT year, month, day, AVG(temp) AS␣
 ↪atemp, AVG(humid) "
```

```
                                    "AS ahumid FROM weather WHERE origin='EWR'␣
 ↪GROUP BY year, month, day ) "
             "AS weather2 ON flights2.year=weather2.year AND flights2.
 ↪month=weather2.month "
             "AND flights2.day=weather2.day")
    task1_sql = pd.read_sql_query(query, conn)
    return task1_sql

def pandas_version():
    flights2 = df_flights[df_flights['origin'] == 'EWR'].copy()
    weather2 = df_weather[df_weather['origin'] == 'EWR'].
 ↪groupby(['year','month','day']).agg({'temp':'mean',
                                                                        ␣
 ↪        'humid':'mean'}).reset_index()
    task1_my = pd.merge(flights2, weather2, how='left',␣
 ↪left_on=['year','month','day'], right_on=['year','month','day'])
    task1_my.rename(columns={'humid': 'ahumid', 'temp': 'atemp'}, inplace=True)
    return task1_my

task1_sql = sql_version()
task1_my = pandas_version()
pd.testing.assert_frame_equal(task1_sql, task1_my)

sql_time = timeit.timeit(sql_version, number=100)
#To compare time I'm doing each version 100 times to capture accurate difference
pandas_time = timeit.timeit(pandas_version, number=100)
print(f"SQL version took {sql_time:.6f} seconds")
print(f"Pandas version took {pandas_time:.6f} seconds")
```

```
SQL version took 123.495584 seconds
Pandas version took 10.999886 seconds
```

This experiment shows that pandas can be used to perfrom SQL like operations directly on dataframes. A major advantage of this apprach is that as the data is loaded in memory, a pure pandas solution might be faster than the SQL way of doing things. Although, a downside of this method is that because the data has to reside in RAM so this is not feasible for very high size of data, an example would be a dynamic datasource where the records are updated and it might be inefficent to hold all the records in memory (although this can be handled by caching useful records). In that case SQL might be preferred, or a mixure of SQL and in-memory apprach, even through there might be some tradeoff in exection time. This experiment shows that both of these methods have their advantages and careful evaluation must be done before finalising any method for use in a real world setting. In case of the given dataset, the Pandas method work well, as it is on a static dataset and can be effectively handled by in-memory operations.