Zadania dzień 1 - Pandas:

1. Porównywanie pd.Series

Utwórz dwa obiekty typu pd.Series, zawierające 100 wartości 0 lub 1. Wykorzystaj w tym celu generator liczb losowych z biblioteki NumPy np.random.randint. Następnie wyświetl wszystkie indeksy, na których wartości w obu Series się zgadzają.

Odp:

```
import numpy as np
import pandas as pd


a = pd.Series(np.random.randint(2, size=100))
b = pd.Series(np.random.randint(2, size=100))
a[a == b].index
```

---

2. Zbiór chipotle - kto zapłacił najwięcej?

Wczytaj zbiór chipotle z https://github.com/matzim95/ML-datasets i dodaj kolumnę price_dollars, zawierającą cenę w dolarach. Następnie sprawdź, które z zamówień opiewało na najwyższą kwotę (wykorzystaj w tym celu grupowanie i wybraną agregację). Podaj jego indeks (order_id) oraz wartość zamówienia.

Odp:

```
import pandas as pd


dolarizer = lambda x: float(x[1:])


chipotle = pd.read_csv('ML-datasets/chipotle.tsv', sep='\t')
chipotle['price_dollars'] = chipotle.item_price.apply(dolarizer)
price_summed = chipotle.groupby('order_id').price_dollars.sum()
print(f'Order {price_summed.idxmax()}: {price_summed.max()}')
```

---

3. Łączenie DataFrames

Wczytaj zbiory danych rating.csv oraz parking.csv (https://github.com/matzim95/ML-datasets). Spróbuj połączyć je wykorzystując metodę merge. Pamiętaj o odpowiedniej nazwie kolumn

---

Odp:

```
import pandas as pd


rating = pd.read_csv('ML-datasets/rating.csv')

parking = pd.read_csv('ML-datasets/parking.csv')


merged = rating.merge(parking, how='left', on='placeID')

merged.sample(5)
```

Zadania - dzień 2:

Część zadań pojawia się wcześniej na zajęciach z trenerem. Jednak ich wykonanie jest zarezerwowane dla osób, które skończyły wcześniej aniżeli reszta grupy. Prezentują się następująco:

Do bazy danych car_rental:

1. *Dodaj autoinkrementację dla kluczy.
2. *Zaktualizuj tabelę bookings o dwa klucze obce, które posiada.
3. *Zastąp dane wybranego klienta swoimi danymi.
4. *W związku z RODO - usuń swoje dane.
5. *Dodaj 2 nowych klientów.
6. *Wypisz wszystkie rezerwacje, których koszt całkowity znajduje się w przedziale 1000-2555.
7. *Wypisz id wszystkich klientów, których nazwisko zaczyna się na literę 'N' oraz imię kończy się na litery 'ej'.
8. *Wypisz liczbę wypożyczonych samochodów, dla których dzienny koszt wypożyczenia jest większy lub równy 300, grupując samochody po mocy silnika, sortując od najmniejszej.
9. *Wypisz sumę kosztów wszystkich rezerwacji, które zostały zrealizowane w okresie 14-18.07.2020.
10. *Wypisz:
    a. średnią liczbę pieniędzy wydanych przez każdego klienta - nazewnictwo kolumny: Srednia_wartosc_rezerwacji
    b. liczbę wypożyczonych samochodów dla każdego klienta, uwzględniając tylko tych klientów, którzy wypożyczyli co najmniej dwa samochody - nazewnictwo kolumny: Liczba_wypozyczonych_samochodow
    c. imię i nazwisko klienta - nazewnictwo kolumn: Imie, Nazwisko
    d. sortując po największej liczbie wypożyczonych samochodów. Wszystko jednym zapytaniem.

Dodatkowo zadania, których nie było na zajęciach:

11. Za pomocą odpowiedniej komendy otrzymaj informacje:
    a. Który samochód jest najczęściej wypożyczany.
    b. Który samochód przynosi najwięcej pieniędzy.
    c. Średnia moc samochodów w wypożyczalni.
    d. Które samochody przynoszą więcej pieniędzy, te wyprodukowane przed 2015 rokiem, czy te po.
    e. Ile samochodów zostało wypożyczonych po `'2020-07-15'`, a ile przed i za jaką łączną kwotę.

Odpowiedzi:

```
1)
ALTER TABLE clients MODIFY COLUMN client_id INTEGER AUTO_INCREMENT;
ALTER TABLE cars MODIFY COLUMN car_id INTEGER AUTO_INCREMENT;
ALTER TABLE bookings MODIFY COLUMN booking_id INTEGER
AUTO_INCREMENT;


2)
ALTER TABLE bookings
ADD CONSTRAINT client_id_fk FOREIGN KEY (client_id) REFERENCES
clients (client_id),
ADD CONSTRAINT car_id_fk FOREIGN KEY (car_id) REFERENCES cars
(car_id);

6)
SELECT *
FROM bookings
WHERE total_amount
BETWEEN 1000 AND 2555;
7)
SELECT client_id
FROM clients
WHERE surname
```

```
LIKE "N%" AND name LIKE "%ej";

8)
SELECT COUNT(b.car_id)
FROM bookings b
JOIN cars r ON b.car_id = r.car_id
WHERE r.price_per_day >= 300
GROUP BY r.horse_power
ORDER BY r.horse_power;
9)
SELECT SUM(total_amount)
FROM bookings
WHERE start_date >= '2020-07-14'
AND end_date <= '2020-07-18';
10)
SELECT AVG(b.total_amount) AS Srednia_wartosc_rezerwacji,
COUNT(b.car_id) AS Liczba_wypozyczonych_samochodow,
c.name AS Imie, c.surname AS Nazwisko
FROM bookings b
JOIN clients c ON c.client_id = b.client_id
GROUP BY b.client_id
HAVING Liczba_wypozyczonych_samochodow >= 2
ORDER BY Liczba_wypozyczonych_samochodow DESC;
```

Zadania - dzień 3:

1. Połącz się z serwerem mysql, ustawiając bazę cinematic jako bazę główną/domyślną. Stwórz definicję tabeli, korzystając z SQLAlchemy:
    a. Directors(klasa) - tabela directors: director_id(PK), name, surname, rating
    b. Movies(klasa) - tabela movies: movie_id(PK), title, year, category, director_id, rating
2. Stwórz sesję, którą będziemy wykorzystywać do wykonywania zapytań.
3. Uzupełnij danymi tabele directors oraz movies, korzystając ze stworzonej sesji. Dane:
    a. directors = [{'name': 'Frank', 'surname': 'Darabont', 'rating': 7}, {'name': 'Francis Ford', 'surname': 'Coppola', 'rating': 8}, {'name': 'Quentin', 'surname': 'Tarantino', 'rating': 10}, {'name': 'Christopher', 'surname':

'Nolan', 'rating': 9}, {'name': 'David', 'surname': 'Fincher', 'rating': 7}]

b. movies = [{'title': 'The Shawshank Redemption', 'year': 1994, 'category': 'Drama', 'director_id': 1, 'rating': 8}, {'title': 'The Green Mile', 'year': 1999, 'category': 'Drama', 'director_id': 1, 'rating': 6}, {'title': 'The Godfather', 'year': 1972, 'category': 'Crime', 'director_id': 2, 'rating': 7}, {'title': 'The Godfather III', 'year': 1990, 'category': 'Crime', 'director_id': 2, 'rating': 6}, {'title': 'Pulp Fiction', 'year': 1994, 'category': 'Crime', 'director_id': 3, 'rating': 9}, {'title': 'Inglourious Basterds', 'year': 2009, 'category': 'War', 'director_id': 3, 'rating': 8}, {'title': 'The Dark Knight', 'year': 2008, 'category': 'Action', 'director_id': 4, 'rating': 9}, {'title': 'Interstellar', 'year': 2014, 'category': 'Sci-fi', 'director_id': 4, 'rating': 8}, {'title': 'The Prestige', 'year': 2006, 'category': 'Drama', 'director_id': 4, 'rating': 10}, {'title': 'Fight Club', 'year': 1999, 'category': 'Drama', 'director_id': 5, 'rating': 7}, {'title': 'Zodiac', 'year': 2007, 'category': 'Crime', 'director_id': 5, 'rating': 5}, {'title': 'Seven', 'year': 1995, 'category': 'Drama', 'director_id': 5, 'rating': 8}, {'title': 'Alien 3', 'year': 1992, 'category': 'Horror', 'director_id': 5, 'rating': 5}]

4. Wypisz wszystkie filmy z kategorii Drama.
5. Wypisz tytuły filmów z kategorii Crime, które zostały wyprodukowane po roku 1994.
6. Wypisz kategorie wszystkich filmów oraz ich ranking dla filmów, które zostały wyprodukowane w latach 2000-2010 oraz których ranking jest większy niż 7, sortując po rankingu.
7. Wypisz nazwiska wszystkich reżyserów, których ranking jest większy bądź równy 6, a ich imię zaczyna się na literę 'D' lub kończy się na literę 'n'.
8. Zbadaj tabele za pomocą pandas profiling.

Odpowiedzi

1)

```
from sqlalchemy import create_engine, Column, String, Integer
from sqlalchemy.ext.declarative import declarative_base

eng = create_engine('mysql://user:password@localhost:3306/cinematic')
base = declarative_base()

class Directors(base):
    __tablename__ = 'directors'

    director_id = Column(Integer, primary_key=True, autoincrement=True)
    name = Column(String(30), nullable=False)
    surname = Column(String(30), nullable=False)
    rating = Column(Integer, nullable=False)

class Movies(base):
    __tablename__ = 'movies'

    movie_id = Column(Integer, primary_key=True, autoincrement=True)
    title = Column(String(30), nullable=False)
    year = Column(Integer, nullable=False)
    category = Column(String(30), nullable=False)
    director_id = Column(Integer, nullable=False)
    rating = Column(Integer, nullable=False)

2)
eng = create_engine('mysql://user:password@localhost:3306/cinematic')

from sqlalchemy.orm import sessionmaker

Session = sessionmaker(bind=eng)
session = Session()

3)

directors = [
    {'name': 'Frank', 'surname': 'Darabont', 'rating': 7},
    {'name': 'Francis Ford', 'surname': 'Coppola', 'rating': 8},
    # ... rest of the copied list
    ]

movies = [
    {'title': 'The Shawshank Redemption', 'year': 1994, 'category': 'Drama',
'director_id': 1, 'rating': 8},
    {'title': 'The Green Mile', 'year': 1999, 'category': 'Drama',
'director_id': 1, 'rating': 6},
    # ... rest of the copied list
    ]
```

```
session.add_all((Directors(**director) for director in directors))
session.commit()

from sqlalchemy import insert
conn = eng.connect()

insert_movies = insert(Movies)
conn.execute(insert_movies, movies)

conn.close()

4)
from sqlalchemy import select
conn = eng.connect()

select = select([Movies]).where(Movies.category =='Drama')
result = conn.execute(select)
print(result.fetchall())
conn.close()

result = session.query(Movies).filter(Movies.category == 'Drama').all()
print(result)

5)

from sqlalchemy import select, and_
conn = eng.connect()

select = select([Movies.title]).where(and_(
            Movies.category == 'Crime', Movies.year > 1994 ))
result = conn.execute(select)
print(result.fetchall())
conn.close()

result = session.query(Movies.title).filter(and_(
            Movies.category == 'Crime', Movies.year > 1994)).all()
print(result)

6)
from sqlalchemy import select, and_, between

conn = eng.connect()

select = select([Movies.category, Movies.rating]).where(
            and_(Movies.rating > 7, between(Movies.year, 2000, 2010))).order_by(
                Movies.rating.desc())
result = conn.execute(select)
print(result.fetchall())
conn.close()
```

```
result = session.query(Movies.category, Movies.rating).filter(
            and_(Movies.rating > 7, between(Movies.year, 2000, 2010))).order_by(
                Movies.rating.desc()).all()
print(result)

7)
from sqlalchemy import select, and_, or_

conn = eng.connect()

select = select([Directors.surname]).where(and_(
            Directors.rating >= 6, or_(
            Directors.name.like('D%'), Directors.name.like('%n'))))
result = conn.execute(select)
print(result.fetchall())
conn.close()

result = session.query(Directors.surname).filter(and_(
            Directors.rating >= 6, or_(
            Directors.name.like('D%'), Directors.name.like('%n')))).all()
print(result)
```