Tennis Competitor Data Analytics — Project Report

1. Overview

This project is a Streamlit-based interactive web app developed for analyzing professional tennis competitor data. It visualizes player performance, rankings, movements, and country-level stats, allowing users to explore the dataset efficiently using filters, charts, and tables.

1. Workflow

A. Data Source & Storage

* Data is stored in a MySQL database named gamedata\_db.
* Two main tables:  
  + Competitors: Contains player information.
  + Competitor\_Rankings: Stores performance metrics like rank, points, movement, and competitions played.

B. Backend (Python + Streamlit)

* Built using Streamlit to serve an interactive user interface.
* Backend connects to MySQL using mysql.connector.
* Data is fetched using custom queries and visualized using:  
  + pandas for data manipulation
  + matplotlib and seaborn for charts
  + streamlit-option-menu for sidebar navigation

C. Frontend Layout

* Sidebar for navigation: Overview, Filter Competitors, Competitor Details, Country Analysis.
* Each page dynamically queries and displays data using interactive components (charts, sliders, tables).

1. Database Schema Design

Table: Competitors

Columns:

* competitor\_id: INT, Primary Key, Unique ID
* name: VARCHAR, Player’s full name
* country: VARCHAR, Country represented

Table: Competitor\_Rankings

Columns:

* ranking\_id: INT, Primary Key, Unique ID
* competitor\_id: INT, Foreign Key to Competitors
* rank: INT, World rank
* points: INT, Ranking points
* movement: INT, Rank movement (positive or negative)
* competitions\_played: INT, Number of competitions played

1. Challenges Faced
2. Data Integration

Mapping competitor\_id between tables required careful joins to ensure accurate results. Needed normalization to avoid redundancy.

1. Frontend

Box plots and bar charts for country data became cluttered with too many countries. Used rotation and selective display to make visuals readable.

1. Handling Null or Missing Data

Had to manage scenarios where rankings or points were missing using default placeholders.

1. Deployment Issues

Needed to resolve dependencies like matplotlib and seaborn using pip install. Streamlit’s limited native support for some complex charts required using pyplot directly.

1. Insights Gained

A. Performance Patterns

Certain countries had consistently high-performing players. Competitors with higher movement often showed potential to break into top rankings.

B. Rank vs Competitions Played

Some players had high points despite fewer competitions, indicating better per-match performance.

C. Data Filtering Features

Filters by name, rank range, points, and country gave clear personalized analysis options. Helped identify hidden performers or emerging competitors.

D. Visual Trends

Boxplots showed outliers and spread of ranking points by country. Bar charts helped compare top 10 players easily

File name: tennisradar.ipynb

import requests

url = "https://api.sportradar.com/tennis/trial/v3/en/competitions.json?api\_key=I67UfClNfonMzr5oIX9kzC8DJnj9dimnezkjfaiE"

headers = {"accept": "application/json"}

response = requests.get(url, headers=headers)

print(response.text)

import json

data = json.loads(response.text)

print(json.dumps(data, indent=4))

competitions = data['competitions']

print(json.dumps(competitions, indent = 3))

categories\_table = []

competitions\_table = []

for c in competitions:

competition\_id = c['id']

competition\_name = c['name']

type = c['type']

gender = c['gender']

parent\_id = c.get('parent\_id')

category = c['category']

category\_id = category['id']

category\_name = category['name']

categories\_table.append((category\_id, category\_name))

competitions\_table.append((competition\_id, competition\_name, parent\_id, type, gender, category\_id))

print(categories\_table)

import mysql.connector

print("MySQL connector is installed and working!")

conn = mysql.connector.connect(

host="localhost",

user="root",

password="MySql@123"

)

cursor = conn.cursor();

cursor.execute("create database gamedata\_db")

cursor.execute("USE gamedata\_db")

cursor.execute("""

CREATE TABLE IF NOT EXISTS categories\_t (

category\_id VARCHAR(50) PRIMARY KEY,

category\_name VARCHAR(100) NOT NULL

)

""")

cursor.execute("""

CREATE TABLE competitions\_t (

competition\_id VARCHAR(50) PRIMARY KEY,

competition\_name VARCHAR(100),

parent\_id VARCHAR(50),

type VARCHAR(20) NOT NULL,

gender VARCHAR(10) NOT NULL,

category\_id VARCHAR(50),

FOREIGN KEY (category\_id) REFERENCES categories(category\_id)

)

""")

print(competitions\_table)

cursor.executemany("""

INSERT IGNORE INTO categories\_t (category\_id, category\_name)

VALUES (%s, %s)

""", categories\_table)

cursor.executemany("""

INSERT IGNORE INTO competitions\_t (competition\_id, competition\_name, parent\_id, type, gender, category\_id)

VALUES (%s, %s, %s, %s, %s, %s)

""", competitions\_table)

conn.commit()

1. List all competitions along with their category name

cursor.execute("""

SELECT c.competition\_name, cat.category\_name

FROM competitions\_t c

JOIN categories\_t cat ON c.category\_id = cat.category\_id

""")

print(cursor.fetchall())

2.​ Count the number of competitions in each category

cursor.execute("""

SELECT cat.category\_name, COUNT(\*) as total\_competitions

FROM competitions\_t c

JOIN categories\_t cat ON c.category\_id = cat.category\_id

GROUP BY cat.category\_name

""")

print(cursor.fetchall())

# 3) Find all competitions of type 'doubles'

cursor.execute("""

SELECT competition\_name

FROM competitions\_t

WHERE type = 'doubles'

""")

print(cursor.fetchall())

# 4) Get competitions that belong to a specific category (e.g., 'ITF Men')

cursor.execute("""

SELECT competition\_name

FROM competitions\_t c

JOIN categories\_t cat ON c.category\_id = cat.category\_id

WHERE cat.category\_name = 'ITF Men'

""")

print(cursor.fetchall())

# 5) Identify parent competitions and their sub-competitions

cursor.execute("""

SELECT parent.competition\_name AS parent\_competition, child.competition\_name AS sub\_competition

FROM competitions\_t child

JOIN competitions\_t parent ON child.parent\_id = parent.competition\_id

""")

print(cursor.fetchall())

# 6) Analyze the distribution of competition types by category

cursor.execute("""

SELECT cat.category\_name, c.type, COUNT(\*) as total

FROM competitions\_t c

JOIN categories\_t cat ON c.category\_id = cat.category\_id

GROUP BY cat.category\_name, c.type

""")

print(cursor.fetchall())

# 7) List all competitions with no parent (top-level competitions)

cursor.execute("""

SELECT competition\_name

FROM competitions\_t

WHERE parent\_id IS NULL

""")

print(cursor.fetchall())

# collecting the Complexes data and Analysis from here

complexes\_url = "https://api.sportradar.com/tennis/trial/v3/en/complexes.json"

complexes\_headers = {

"accept": "application/json",

"x-api-key": "I67UfClNfonMzr5oIX9kzC8DJnj9dimnezkjfaiE"

}

com\_response = requests.get(complexes\_url, headers=complexes\_headers)

print(com\_response.text)

com\_data = json.loads(com\_response.text)

print(json.dumps(com\_data, indent=4))

complexes\_table = []

venues\_table = []

for complex\_item in com\_data.get('complexes', []):

complex\_id = complex\_item['id']

complex\_name = complex\_item['name']

complexes\_table.append((complex\_id, complex\_name))

for venue in complex\_item.get('venues', []):

venue\_id = venue['id']

venue\_name = venue['name']

city\_name = venue['city\_name']

country\_name = venue['country\_name']

country\_code = venue['country\_code']

timezone = venue['timezone']

venues\_table.append(

(venue\_id, venue\_name, city\_name, country\_name, country\_code, timezone, complex\_id)

)

cursor.execute("""CREATE TABLE complexes (

complex\_id VARCHAR(50) PRIMARY KEY,

complex\_name VARCHAR(100) NOT NULL

)""")

cursor.execute("""CREATE TABLE venues (

venue\_id VARCHAR(50) PRIMARY KEY,

venue\_name VARCHAR(100) NOT NULL,

city\_name VARCHAR(100) NOT NULL,

country\_name VARCHAR(100) NOT NULL,

country\_code CHAR(3) NOT NULL,

timezone VARCHAR(100) NOT NULL,

complex\_id VARCHAR(50),

FOREIGN KEY (complex\_id) REFERENCES complexes(complex\_id)

)""")

cursor.executemany("""

INSERT INTO complexes (complex\_id, complex\_name)

VALUES (%s, %s)

""", complexes\_table)

cursor.executemany("""

INSERT INTO venues (venue\_id, venue\_name, city\_name, country\_name, country\_code, timezone, complex\_id)

VALUES (%s, %s, %s, %s, %s, %s, %s)

""", venues\_table)

conn.commit()

# 1) List all venues along with their associated complex name

cursor.execute("""

SELECT v.venue\_name, c.complex\_name

FROM venues v

JOIN complexes c ON v.complex\_id = c.complex\_id

""")

print(cursor.fetchall())

# 2) Count the number of venues in each complex

cursor.execute("""

SELECT c.complex\_name, COUNT(v.venue\_id) AS total\_venues

FROM venues v

JOIN complexes c ON v.complex\_id = c.complex\_id

GROUP BY c.complex\_name

""")

print(cursor.fetchall())

# 3) Get details of venues in a specific country

cursor.execute("""

SELECT venue\_name, city\_name, country\_name

FROM venues

WHERE country\_name = 'Chile'

""")

print(cursor.fetchall())

# 4) Identify all venues and their timezones

cursor.execute("""

SELECT venue\_name, timezone

FROM venues

""")

print(cursor.fetchall())

# 5) Find complexes that have more than one venue

cursor.execute("""

SELECT c.complex\_name, COUNT(v.venue\_id) AS venue\_count

FROM venues v

JOIN complexes c ON v.complex\_id = c.complex\_id

GROUP BY c.complex\_name

HAVING COUNT(v.venue\_id) > 1

""")

print(cursor.fetchall())

# 6) List venues grouped by country

cursor.execute("""

SELECT country\_name, GROUP\_CONCAT(venue\_name, ', ') AS venues

FROM venues

GROUP BY country\_name

""")

print(cursor.fetchall())

# 7) Find all venues for a specific complex

cursor.execute("""

SELECT v.venue\_name

FROM venues v

JOIN complexes c ON v.complex\_id = c.complex\_id

WHERE c.complex\_name = 'Nacional'

""")

print(cursor.fetchall())

# Collecting the doubles ranking competition data and analysis from here

doubles\_url = "https://api.sportradar.com/tennis/trial/v3/en/double\_competitors\_rankings.json"

doubles\_headers = {

"accept": "application/json",

"x-api-key": "I67UfClNfonMzr5oIX9kzC8DJnj9dimnezkjfaiE"

}

doubles\_response = requests.get(doubles\_url, headers=doubles\_headers)

print(doubles\_response.text)

doubles\_data = json.loads(doubles\_response.text)

print(json.dumps(doubles\_data, indent=4))

rankings\_data = doubles\_data['rankings'][0] # First ranking entry

rankings\_table = []

for rank\_info in rankings\_data['competitor\_rankings']:

rank = rank\_info['rank']

movement = rank\_info['movement']

points = rank\_info['points']

competitions\_played = rank\_info['competitions\_played']

competitor = rank\_info['competitor']

competitor\_id = competitor['id']

rankings\_table.append(

(rank, movement, points, competitions\_played, competitor\_id)

)

competitors\_table = []

for competitor\_info in rankings\_data['competitor\_rankings']:

comp = competitor\_info['competitor']

competitor\_id = comp['id']

name = comp['name']

country = comp['country']

country\_code = comp.get('country\_code', 'N/A')

abbreviation = comp['abbreviation']

competitors\_table.append((competitor\_id, name, country, country\_code, abbreviation))

cursor.execute("""

CREATE TABLE IF NOT EXISTS Competitors (

competitor\_id VARCHAR(50) PRIMARY KEY,

name VARCHAR(100) NOT NULL,

country VARCHAR(100) NOT NULL,

country\_code CHAR(3) NOT NULL,

abbreviation VARCHAR(10) NOT NULL

)

""")

cursor.execute("""

CREATE TABLE IF NOT EXISTS Competitor\_Rankings (

rank\_id INT NOT NULL AUTO\_INCREMENT PRIMARY KEY,

`rank` INT NOT NULL,

movement INT NOT NULL,

points INT NOT NULL,

competitions\_played INT NOT NULL,

competitor\_id VARCHAR(50),

FOREIGN KEY (competitor\_id) REFERENCES Competitors(competitor\_id)

)

""")

cursor.executemany("""

INSERT INTO Competitors (competitor\_id, name, country, country\_code, abbreviation)

VALUES (%s, %s, %s, %s, %s)

""", competitors\_table)

cursor.executemany("""

INSERT INTO Competitor\_Rankings (`rank`, movement, points, competitions\_played, competitor\_id)

VALUES (%s, %s, %s, %s, %s)

""", rankings\_table)

# 1) Get all competitors with their rank and points

cursor.execute("""

SELECT c.competitor\_id, c.name, cr.rank, cr.points

FROM Competitors c

JOIN Competitor\_Rankings cr ON c.competitor\_id = cr.competitor\_id

""")

print(cursor.fetchall())

# 2) Find competitors ranked in the top 5

cursor.execute("""

SELECT c.name, cr.`rank`, cr.points

FROM Competitors c

JOIN Competitor\_Rankings cr ON c.competitor\_id = cr.competitor\_id

WHERE cr.`rank` <= 5

ORDER BY cr.`rank`

""")

print(cursor.fetchall())

# 3) List competitors with no rank movement (stable rank)

cursor.execute("""

SELECT c.name, cr.`rank`, cr.movement

FROM Competitors c

JOIN Competitor\_Rankings cr ON c.competitor\_id = cr.competitor\_id

WHERE cr.movement = 0

""")

print(cursor.fetchall())

# 4) Get the total points of competitors from a specific country (e.g., Croatia)

cursor.execute("""

SELECT c.country, SUM(cr.points) AS total\_points

FROM Competitors c

JOIN Competitor\_Rankings cr ON c.competitor\_id = cr.competitor\_id

WHERE c.country = 'Croatia'

GROUP BY c.country

""")

print(cursor.fetchall())

# 5) Count the number of competitors per country

cursor.execute("""

SELECT c.country, COUNT(\*) AS num\_competitors

FROM Competitors c

GROUP BY c.country

ORDER BY num\_competitors DESC

""")

print(cursor.fetchall())

# 6) Find competitors with the highest points in the current week

cursor.execute("""

SELECT c.name, cr.points

FROM Competitors c

JOIN Competitor\_Rankings cr ON c.competitor\_id = cr.competitor\_id

WHERE cr.points = (

SELECT MAX(points)

FROM Competitor\_Rankings

)

""")

print(cursor.fetchall())

File name 2 : onapp.py

import streamlit as st

import pandas as pd

import mysql.connector

import seaborn as sns

import matplotlib.pyplot as plt

from streamlit\_option\_menu import option\_menu

# ---------- SETUP ----------

st.set\_page\_config(page\_title="Tennis Competitor Analytics", layout="wide")

def get\_connection():

return mysql.connector.connect(

host="localhost",

user="root",

password="MySql@123",

database="gamedata\_db"

)

def fetch\_query(query, params=None):

conn = get\_connection()

cursor = conn.cursor(dictionary=True)

cursor.execute(query, params or ())

results = cursor.fetchall()

cursor.close()

conn.close()

return pd.DataFrame(results)

# ---------- SIDEBAR MENU ----------

with st.sidebar:

selected = option\_menu(

menu\_title=" Navigation Menu",

options=["Overview", "Filter Competitors", "Competitor Details", "Country Analysis"],

icons=["bar-chart", "funnel-fill", "person-lines-fill", "globe2"],

menu\_icon="list",

default\_index=0,

)

# ---------- OVERVIEW PAGE ----------

if selected == "Overview":

st.title("🎾 Tennis Competitor Analytics")

# Metrics

col1, col2, col3 = st.columns(3)

with col1:

total = fetch\_query("SELECT COUNT(\*) AS total FROM Competitors")

st.metric("Total Competitors", total['total'][0] if not total.empty else 0)

with col2:

countries = fetch\_query("SELECT COUNT(DISTINCT country) AS countries FROM Competitors")

st.metric("Countries Represented", countries['countries'][0] if not countries.empty else 0)

with col3:

highest = fetch\_query("""

SELECT name, points FROM Competitors c

JOIN Competitor\_Rankings cr ON c.competitor\_id = cr.competitor\_id

ORDER BY points DESC LIMIT 1

""")

if not highest.empty:

st.metric("Highest Points", highest['points'][0], highest['name'][0])

else:

st.metric("Highest Points", 0)

# Rankings Table

st.subheader(" Full Competitor Rankings")

rankings = fetch\_query("""

SELECT c.name, c.country, cr.rank, cr.points, cr.competitions\_played, cr.movement

FROM Competitors c

JOIN Competitor\_Rankings cr ON c.competitor\_id = cr.competitor\_id

ORDER BY cr.rank ASC

""")

st.dataframe(rankings, use\_container\_width=True)

# Leaderboard Chart

st.subheader("🏅 Leaderboard (Top 10 by Points)")

top\_points = fetch\_query("""

SELECT c.name, cr.points FROM Competitors c

JOIN Competitor\_Rankings cr ON c.competitor\_id = cr.competitor\_id

ORDER BY cr.points DESC LIMIT 10

""")

if not top\_points.empty:

st.bar\_chart(top\_points.set\_index("name"))

st.subheader(" Top 5 Rank Movers")

movers = fetch\_query("""

SELECT c.name, cr.movement

FROM Competitors c

JOIN Competitor\_Rankings cr ON c.competitor\_id = cr.competitor\_id

WHERE cr.movement > 0

ORDER BY cr.movement DESC LIMIT 5

""")

st.table(movers if not movers.empty else pd.DataFrame({"No Movers": []}))

# ---------- FILTER COMPETITORS ----------

elif selected == "Filter Competitors":

st.title(" Filter Competitors")

name\_input = st.text\_input("Search by name")

min\_rank = st.slider("Rank Range", 1, 100, (1, 50))

min\_points = st.number\_input("Minimum Points", 0, 10000, 0)

countries\_list = fetch\_query("SELECT DISTINCT country FROM Competitors")['country']

country = st.selectbox("Filter by Country", options=["All"] + list(countries\_list))

query = """

SELECT c.name, c.country, cr.rank, cr.points

FROM Competitors c

JOIN Competitor\_Rankings cr ON c.competitor\_id = cr.competitor\_id

WHERE c.name LIKE %s AND cr.rank BETWEEN %s AND %s AND cr.points >= %s

"""

params = (f"%{name\_input}%", min\_rank[0], min\_rank[1], min\_points)

if country != "All":

query += " AND c.country = %s"

params += (country,)

filtered\_df = fetch\_query(query, params)

st.dataframe(filtered\_df if not filtered\_df.empty else pd.DataFrame({"No Data": []}), use\_container\_width=True)

# ---------- COMPETITOR DETAILS ----------

elif selected == "Competitor Details":

st.title(" Competitor Details")

competitors = fetch\_query("SELECT DISTINCT name FROM Competitors")

if not competitors.empty:

selected\_name = st.selectbox("Select a Competitor", competitors['name'])

details = fetch\_query("""

SELECT c.name, c.country, cr.rank, cr.points, cr.movement, cr.competitions\_played

FROM Competitors c

JOIN Competitor\_Rankings cr ON c.competitor\_id = cr.competitor\_id

WHERE c.name = %s

""", (selected\_name,))

st.write(details.T)

else:

st.warning("No competitors found.")

# ---------- COUNTRY ANALYSIS ----------

elif selected == "Country Analysis":

st.title(" Country-Wise Analysis")

country\_stats = fetch\_query("""

SELECT c.country, COUNT(\*) AS total\_competitors, AVG(cr.points) AS avg\_points

FROM Competitors c

JOIN Competitor\_Rankings cr ON c.competitor\_id = cr.competitor\_id

GROUP BY c.country

ORDER BY avg\_points DESC

""")

if not country\_stats.empty:

st.dataframe(country\_stats, use\_container\_width=True)

st.bar\_chart(country\_stats.set\_index("country")["avg\_points"])

# Box Plot Here

df = fetch\_query("""

SELECT c.country, cr.points

FROM Competitor\_Rankings cr

JOIN Competitors c ON c.competitor\_id = cr.competitor\_id

""")

if not df.empty:

st.subheader(" Points Distribution by Country (Box Plot)")

fig, ax = plt.subplots(figsize=(12, 6))

sns.boxplot(x="country", y="points", data=df, ax=ax)

ax.set\_xticklabels(ax.get\_xticklabels(), rotation=45)

st.pyplot(fig)

else:

st.warning("No country data available.")