POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Systems and Methods for Big and Unstructured Data Project

Author(s): **Gabriele Bandino 11098651**
Leonardo Bianconi 10660229
Emanuele Dossi 10722324

Group Number: **127**

Academic Year: 2024-2025

Contents

1 | Dataset 1

1.1. Introduction

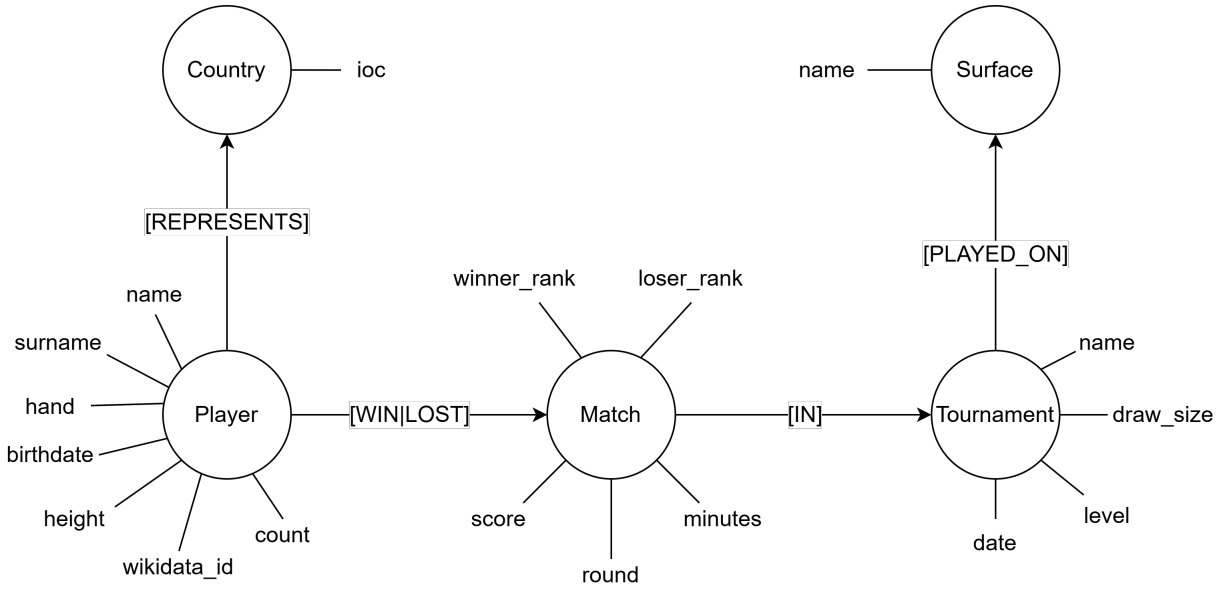
The first dataset we have chosen is *Huge Tennis Database*, a comprehensive archive of match results, player statistics, and tournament details regarding historical male professional tennis events. The aim of the analysis is to get summarized informations about various aspects of the game, as a company involved in the analysis of the game would do. Such company could then sell to players' teams and other stakeholders various aggregate results, pertaining subsets of matches (including, for example, only matches of certain years, tournament, surface of the court, played by certain players, etc.). We have chosen a graph database technology such as Neo4J because this domain can be naturally mapped to a graph, in which players are connected through matches in which they played against (and then matches to tournaments) and because our queries would need different joins, which are expensive operations to perform in relational databases and can be performed easily by pattern matching on a graph.

1.2. Data Wrangling and Generation

We decided, for performance reasons, not to include all data on past tournament matches, but to only cover recent games (from 2018 to the present). Note that this decision still preserves the big cardinality of the dataset, which is comprised of 64 thousands nodes and 233 thousands relations. Moreover, we decided to exclude players' rankings from our analysis. This choice was to reduce overall complexity and to the limited number of queries to perform. Lastly, the only real data wrangling operation we have performed is the creation of a unique match identifier, created by appending the id of the tournament and the number of match (match_id: row.tourney_id + '_' + row.match_num).

1.3. Database schema

The image below illustrates our database schema.



1.3.1. Nodes

We start by describing each type of node present in the database.

1. **:Player** this node type describes a male tennis player, its attributes are:
   - player_id: a unique identifier for the player.
   - name: name of the player.
   - surname: surname of the player.
   - hand: playing hand.
   - birthdate: the date of birth of the player, stored as a *apoc.date* type.
   - height: the height of the player, expressed in centimeters.
   - wikidata_id: external ID for searching more informations about the player on Wikidata.org.
   - count: debugging value only used for duplicate checking.
2. **:Country** this node type describes a country, of which a player may be citizen, it only has one attribute:
   - ioc: a string of 3 characters specifying the country's abbreviation.

3. **:Tournament** this node type describes a single edition of a tournament in the major professional male tennis circuits, its attributes are:
    - tourney_id: a unique identifier for the tournament.
    - name: the generic name of the tournament (e.g. "Wimbledon", "Roland Garros").
    - draw_size: the number of participants to that tournament
    - level: a 1-character string defining the level of the tournament.
    - date: a *apoc.date* object indicating the starting date of the tournament.
4. **:Match** describes a single match of a tournament:
    - score: final score of the match, as a string of per-set couples of values.
    - round: a string describing the round of the tournament that match is valid for (e.g. "QF" means quarter-finals).
    - minutes: the total game time, in minutes.
    - winner_rank: the ranking of the winning player at the moment of the match.
    - loser_rank: same, for the loser.
5. **:Surface** this is a node describing the surface type of the courts on which a tournament is held:
    - name: the type of surface.

1.3.2. Relations

In this section, we explain the different relationships that we decided to include in the graph.

1. **:REPRESENTS** is a directed relationship connecting a Player to the Country he's representing
2. **:IN** is another directed relationship indicating that a Match is part of a certain Tournament
3. **:LOST|WIN** is a directed relationship going from a Player to a Match, indicating that Player lost/win that match. It includes several attributes about the Player's performance for that match:

- aces: number of aces
- double_faults: number of double faults
- serve_points: number of serve points
- first_in: number of first serves made
- first_won: number of first-serve points won
- second_won: number of second-serve points won
- serve_games: number of own serve games played
- bp_saved: number of break points saved
- bp_faced: number of break points faced

4. **:PLAYED_ON**: is a directed relationship connecting a Tournament to the Surface where it takes place.

1.4. Constraints

We have defined constraints on the unique identifiers of the nodes:

```
CREATE CONSTRAINT FOR (p:Player) REQUIRE p.player_id IS UNIQUE;
CREATE CONSTRAINT FOR (m:Match) REQUIRE m.match_id IS UNIQUE;
CREATE CONSTRAINT FOR (t:Tourney) REQUIRE t.tourney_id IS UNIQUE;
```

1.5. Import

This is the *cypher* script we have used for importing the .csv dataset, which is composed of multiple files, into the Neo4J database:

```
//Player
LOAD CSV WITH HEADERS FROM "file:///atp_players.csv" AS row
WITH row WHERE row.player_id IS NOT NULL
WITH row WHERE row.dob IS NOT NULL
MERGE (p:Player player_id: row.player_id)
ON CREATE SET p.name = row.name_first,
p.surname = row.name_last,
p.hand = row.hand,
```

```
p.birthdate = date(datetime(epochMillis: apoc.date.parse(row.dob, 'ms', 'yyyyMMdd'))),
p.height = row.height,
p.wikidata_id = row.wikidata_id
ON MATCH SET p.count = coalesce(p.count, 0) + 1;

//Country
LOAD CSV WITH HEADERS FROM 'file:///atp_players.csv' AS row
WITH row WHERE row.ioc IS NOT NULL
MERGE (c:Country ioc: row.ioc);

//Tournament
LOAD CSV WITH HEADERS FROM 'file:///atp_matches_2018-2024.csv' AS row
WITH row WHERE row.tourney_id IS NOT NULL
MERGE (t:Tournament tourney_id: row.tourney_id)
ON CREATE SET t.name = row.tourney_name,
t.draw_size = toInteger(row.draw_size),
t.level = row.tourney_level,
t.date = date(datetime(epochMillis: apoc.date.parse(row.tourney_date, 'ms', 'yyyyM-
Mdd')));

//Match
LOAD CSV WITH HEADERS FROM 'file:///atp_matches_2018-2024.csv' AS row
MERGE (m:Match match_id: row.tourney_id + '_' + row.match_num)
ON CREATE SET m.score = row.score,
m.round = row.round,
m.minutes = toInteger(row.minutes),
m.winner_rank = toInteger(row.winner_rank),
m.loser_rank = toInteger(row.loser_rank);

//Surface
LOAD CSV WITH HEADERS FROM 'file:///atp_matches_2018-2024.csv' AS row
WITH row WHERE row.surface IS NOT NULL
MERGE (s:Surface name: row.surface);


//— Relations —

LOAD CSV WITH HEADERS FROM 'file:///atp_players.csv' AS row
MATCH (p:Player player_id: row.player_id)
MATCH (c:Country ioc: row.ioc)
```

```
MERGE (p)-[:REPRESENTS]->(c);

LOAD CSV WITH HEADERS FROM 'file:///atp_matches_2018-2024.csv' AS row
MATCH (m:Match match_id: row.tourney_id + '_' + row.match_num)
MATCH (t:Tournament tourney_id: row.tourney_id)
MERGE (m)-[:IN]->(t);

LOAD CSV WITH HEADERS FROM 'file:///atp_matches_2018-2024.csv' AS row
MATCH (m:Match match_id: row.tourney_id + '_' + row.match_num)
MATCH (p1:Player player_id: row.winner_id)
MERGE (p1)-[w:WON]->(m)
ON CREATE SET w.aces = toInteger(row.w_ace),
w.double_faults = toInteger(row.w_df),
w.serve_points = toInteger(row.w_svpt),
w.first_in = toInteger(row.w_1stIn),
w.first_won = toInteger(row.w_1stWon),
w.second_won = toInteger(row.w_2ndWon),
w.serve_games = toInteger(row.w_SvGms),
w.bp_saved = toInteger(row.w_bpSaved),
w.bp_faced = toInteger(row.w_bpFaced)
WITH m, row
MATCH (p2:Player player_id: row.loser_id)
MERGE (p2)-[l:LOST]->(m)
ON CREATE SET l.aces = toInteger(row.l_ace),
l.double_faults = toInteger(row.l_df),
l.serve_points = toInteger(row.l_svpt),
l.first_in = toInteger(row.l_1stIn),
l.first_won = toInteger(row.l_1stWon),
l.second_won = toInteger(row.l_2ndWon),
l.serve_games = toInteger(row.l_SvGms),
l.bp_saved = toInteger(row.l_bpSaved),
l.bp_faced = toInteger(row.l_bpFaced);

LOAD CSV WITH HEADERS FROM 'file:///atp_matches_2018-2024.csv' AS row
MATCH (t:Tournament tourney_id: row.tourney_id)
MATCH (s:Surface name: row.surface)
MERGE (t)-[:PLAYED_ON]->(s);
```

2 | Queries 1

2.1. Tournaments won by country

For each country return the number of tournaments and matches won.

```
MATCH (p:Player)-[:WON]->(m:Match), (p)-[:REPRESENTS]->(c:Country)
WHERE m.round = 'F'
WITH c, COUNT(m) AS TournamentsWon
MATCH (p:Player)-[:WON]->(m2:Match), (p)-[:REPRESENTS]->(c:Country)
RETURN c.ioc AS Country, TournamentsWon, COUNT(m2) AS MatchesWon
ORDER BY TournamentsWon DESC
```

Country	TournamentsWon	MatchesWon
"ESP"	49	1494
"RUS"	45	1012
"SRB"	37	853
"ITA"	35	1115
"USA"	32	1766
"FRA"	28	1367
"GER"	19	852
"AUS"	18	899
"ARG"	18	879
"GRE"	12	330

2.2. Total duration of each tournament

Compute the total duration (in minutes) of each tournament by summing the time of the matches.

```
MATCH (m:Match)-[:IN]->(t:Tournament)
WHERE m.minutes IS NOT NULL
RETURN t, SUM(m.minutes) AS TotalMatchDuration
ORDER BY TotalMatchDuration DESC
```

t	TotalMatchDuration
(:Tournament {tourney_id: "2024-580",date: "2024-01-15",draw_size: 128,level: "G",name: "Australian Open"})	22761
(:Tournament {tourney_id: "2023-520",date: "2023-05-29",draw_size: 128,level: "G",name: "Roland Garros"})	22003
(:Tournament {tourney_id: "2022-560",date: "2022-08-29",draw_size: 128,level: "G",name: "Us Open"})	21948
(:Tournament {tourney_id: "2023-580",date: "2023-01-16",draw_size: 128,level: "G",name: "Australian Open"})	21822
(:Tournament {tourney_id: "2021-560",date: "2021-08-30",draw_size: 128,level: "G",name: "Us Open"})	21606
(:Tournament {tourney_id: "2022-580",date: "2022-01-17",draw_size: 128,level: "G",name: "Australian Open"})	20734
(:Tournament {tourney_id: "2019-560",date: "2019-08-26",draw_size: 128,level: "G",name: "US Open"})	20341
(:Tournament {tourney_id: "2023-540",date: "2023-07-03",draw_size: 128,level: "G",name: "Wimbledon"})	20271
(:Tournament {tourney_id: "2023-560",date: "2023-08-28",draw_size: 128,level: "G",name: "Us Open"})	20170
(:Tournament {tourney_id: "2018-560",date: "2018-08-27",draw_size: 128,level: "G",name: "US Open"})	20138

2.3. Players' winning ratio

For each player, the won and lost matches with the ratio, excluding players who only played very few matches.

```
MATCH (p:Player)-[:WON]->(m1:Match)
WITH p, COUNT(m1) AS wins
MATCH (p:Player)-[:LOST]->(m2:Match)
WITH p, wins, COUNT(m2) AS losses
WHERE wins + losses > 20
RETURN p.name, p.surname, wins, losses, (1.0*wins/(wins+losses)) AS WinRatio
ORDER BY WinRatio DESC
```

p.name	p.surname	wins	losses	WinRatio
"Novak"	"Djokovic"	321	56	0.8514588859416445
"Rafael"	"Nadal"	204	41	0.8326530612244898
"Roger"	"Federer"	120	27	0.8163265306122449
"Carlos"	"Alcaraz"	173	48	0.7828054298642534
"Daniil"	"Medvedev"	328	111	0.7471526195899773
"Juan Martin"	"del Potro"	56	19	0.7466666666666667
"Jannik"	"Sinner"	222	81	0.7326732673267327
"Alexander"	"Zverev"	304	117	0.7220902612826603
"Stefanos"	"Tsitsipas"	322	142	0.6939655172413793
"Andrey"	"Rublev"	283	129	0.6868932038834952

2.4. Tournaments per surface

Number of tournaments played on different surfaces.

MATCH (t:Tournament)-[]->(s:Surface)
WITH s, COUNT(t) AS numTournaments
RETURN s.name AS Surface, numTournaments;

Surface	numTournaments
"Hard"	513
"Clay"	224
"Grass"	49

2.5. Players with most aces in a match

Top 20 matches with the highest number of aces from one player.

```
MATCH (p:Player)-[played]->(m:Match)<-[played2]-(p2:Player)
WHERE played.aces IS NOT NULL AND played2.aces IS NOT NULL AND p <> p2
RETURN p.name + ' ' + p.surname AS MostAcesPlayer, m.score AS Score, p2.name +
' ' + p2.surname AS Opponent, played.aces AS NumAces
ORDER BY NumAces DESC
LIMIT 20;
```

MostAcesPlayer	Score	Opponent	NumAces
"Reilly Opelka"	"6-7(15) 6-2 6-4 3-6 7-6(5)"	"Thomas Fabbiano"	67
"John Isner"	"6-1 6-4 6-7(6) 6-7(3) 7-5"	"Ruben Bemelmans"	64
"Ivo Karlovic"	"6-7(5) 3-6 7-6(4) 7-6(4) 13-11"	"Jan Lennard Struff"	61
"Ivo Karlovic"	"6-3 7-6(6) 5-7 5-7 7-6(7)"	"Kei Nishikori"	59
"John Isner"	"6-7(6) 7-6(3) 4-6 6-3 7-5"	"Enzo Couacaud"	54
"Ivo Karlovic"	"7-6(3) 6-7(3) 7-5 4-6 12-10"	"Yuichi Sugita"	53
"John Isner"	"7-6(6) 6-7(5) 6-7(9) 6-4 26-24"	"Kevin Anderson"	53
"Ivo Karlovic"	"6-3 7-6(4) 6-7(3) 6-7(5) 9-7"	"Andreas Seppi"	52
"John Isner"	"6-7(5) 6-3 6-7(5) 6-3 7-6(3)"	"Steve Johnson"	52
"Kevin Anderson"	"7-6(6) 6-7(5) 6-7(9) 6-4 26-24"	"John Isner"	49

2.6. Youngest champion

Find the youngest player to win a professional tournament.

```
MATCH (p:Player)-[:WON]->(m:Match), (m)-[:IN]->(t:Tournament)
WHERE m.round = 'F'
WITH p, t, MIN(duration.between(p.birthdate, t.date)) AS YearsChampion WITH p, t,
YearsChampion.years as Years, YearsChampion.months as Months, YearsChampion.days
as Days, YearsChampion
RETURN p.name + ' ' + p.surname AS Player, toString(Years) + ' years ' + toString(toInteger(Months
- Years*12.0)) + ' months ' + toString(Days) + ' days ' AS ChampionAt, t.name AS Tour-
nament, t.date AS DateTournament
ORDER BY YearsChampion ASC
```

LIMIT 1;

Player	ChampionAt	Tournament	DateTournament
"Carlos Alcaraz"	"18 years 2 months 14 days "	"Umag"	"2021-07-19"

2.7. Win ratio for each player, surface

Find, for each player, his win ratio on each type of surface.

MATCH (p:Player)-[played]->(m:Match)-[]->(t:Tournament)-[]->(s:Surface)
WHERE type(played) = "WON"
WITH p, s, count(m) AS wins
MATCH (p:Player)-[played]->(m:Match)-[]->(t:Tournament)-[]->(s:Surface)
WITH p, s,wins, count(m) AS totMatch
RETURN p.name, p.surname, s.name AS surface, wins, totMatch, 1.0*wins/totMatch AS ratio
ORDER BY p.player_id, ratio DESC;

p.name	p.surname	surface	wins	totMatch	ratio
"Alexander"	"Zverev"	"Clay"	101	133	0.7593984962406015
"Alexander"	"Zverev"	"Hard"	190	266	0.7142857142857143
"Alexander"	"Zverev"	"Grass"	13	22	0.5909090909090909
"Martin"	"Damm"	"Hard"	2	3	0.6666666666666666
"Ivo"	"Karlovic"	"Clay"	6	14	0.42857142857142855
"Ivo"	"Karlovic"	"Grass"	4	10	0.4
"Ivo"	"Karlovic"	"Hard"	15	40	0.375
"Aqeel"	"Khan"	"Grass"	6	10	0.6
"Aisam Ul Haq"	"Qureshi"	"Grass"	4	7	0.5714285714285714
"Stephane"	"Robert"	"Grass"	1	2	0.5

2.8. Most frequent match-ups

Find the players that have faced each other the most times, along with the number of victories of both players.

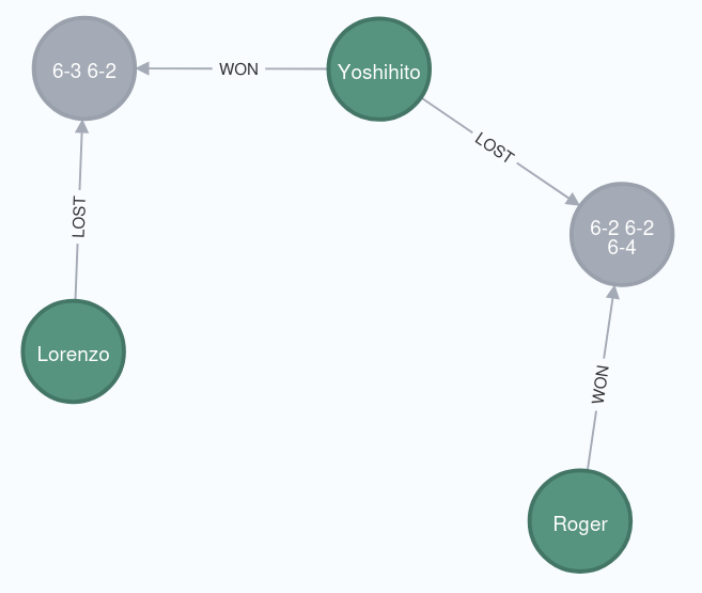
```
MATCH (p1:Player)-[:WON]->(m_won:Match)<-[:LOST]-(p2:Player)
WHERE p1.player_id < p2.player_id
WITH p1, p2, COUNT(m_won) AS NumVictoriesP1
MATCH (p1)-[:LOST]->(m_lost:Match)<-[:WON]-(p2)
WHERE p1.player_id < p2.player_id
WITH p1, p2, NumVictoriesP1, COUNT(m_lost) AS NumVictoriesP2
RETURN p1.name + ' ' + p1.surname AS Player1, p2.name + ' ' + p2.surname AS
Player2, NumVictoriesP1 + NumVictoriesP2 AS TotMatches, NumVictoriesP1, NumVic-
toriesP2
ORDER BY TotMatches DESC
LIMIT 10;
```

Player1	Player2	TotMatches	NumVictoriesP1	NumVictoriesP2
"Alexander Zverev"	"Daniil Medvedev"	17	5	12
"Alexander Zverev"	"Stefanos Tsitsipas"	15	5	10
"Cameron Norrie"	"Taylor Fritz"	14	6	8
"Daniil Medvedev"	"Stefanos Tsitsipas"	13	9	4
"Novak Djokovic"	"Daniil Medvedev"	13	8	5
"Novak Djokovic"	"Stefanos Tsitsipas"	13	11	2
"Stefanos Tsitsipas"	"Alex De Minaur"	12	11	1
"Alexander Zverev"	"Novak Djokovic"	11	3	8
"Dominic Thiem"	"Stefanos Tsitsipas"	11	5	6
"Andrey Rublev"	"Stefanos Tsitsipas"	11	5	6

2.9. Shortest path between two players

Find the shortest path in terms of matches between two given players.

```
MATCH (p1:Player name: 'Roger', surname: 'Federer'), (p2:Player name: 'Lorenzo',
surname: 'Musetti'), path = shortestPath((p1)-[:WON|LOST*]-(p2))
RETURN path;
```



2.10. Dominance cycles

Find "dominance cycles" of length 3 between players (i.e. a first player has beaten a second, that has beaten a third, that, in his turn, has beaten the first one), on different tournaments, on the same surface..

MATCH (p1:Player)-[w1:WON]->(m1:Match)-[:IN]->(t1:Tournament)-[:PLAYED_ON]->(s:Surface),
(p2:Player)-[l1:LOST]->(m1),
(p2)-[w2:WON]->(m2:Match)-[:IN]->(t2:Tournament)-[:PLAYED_ON]->(s),
(p3:Player)-[l2:LOST]->(m2),
(p3)-[w3:WON]->(m3:Match)-[:IN]->(t3:Tournament)-[:PLAYED_ON]->(s),
(p1)-[l3:LOST]->(m3)
WHERE t1.tourney_id <> t2.tourney_id AND t2.tourney_id <> t3.tourney_id AND t1.tourney_id <> t3.tourney_id
WITH DISTINCT p1, p2, p3, s.name AS Surface,
t1.name AS Tournament1, t1.date AS Date1,
t2.name AS Tournament2, t2.date AS Date2,
t3.name AS Tournament3, t3.date AS Date3
ORDER BY Date1, Date2, Date3
RETURN p1.name + ' ' + p1.surname AS Player1, p2.name + ' ' + p2.surname AS Player2, p3.name + ' ' + p3.surname AS Player3, Surface, Tournament1, Date1, Tournament2, Date2, Tournament3, Date3;

Player1	Player2	Player3	Surface	Tournament1	Date1	Tournament2	Date2	Tournament3	Date3
"Kyle Edmund"	"Hyeon Chung"	"John Isner"	"Hard"	"Brisbane"	"2018-01-01"	"Auckland"	"2018-01-08"	"Miami Masters"	"2019-03-18"
"Gael Monfils"	"Paolo Lorenzi"	"Albert Ramos"	"Hard"	"Doha"	"2018-01-01"	"Sydney"	"2018-01-08"	"Zhuhai"	"2019-09-23"
"Kyle Edmund"	"Hyeon Chung"	"John Isner"	"Hard"	"Brisbane"	"2018-01-01"	"Auckland"	"2018-01-08"	"Auckland"	"2020-01-13"
"Grigor Dimitrov"	"John Millman"	"Alexei Popyrin"	"Hard"	"Brisbane"	"2018-01-01"	"Sydney"	"2018-01-08"	"Us Open"	"2021-08-30"
"Gael Monfils"	"Paolo Lorenzi"	"Jordan Thompson"	"Hard"	"Doha"	"2018-01-01"	"Sydney"	"2018-01-08"	"Indian Wells Masters"	"2023-03-06"
"Jan Lennard Struff"	"Tomas Berdych"	"Alex De Minaur"	"Hard"	"Doha"	"2018-01-01"	"Australian Open"	"2018-01-15"	"Indian Wells Masters"	"2018-03-05"
"Grigor Dimitrov"	"Kyle Edmund"	"Kevin Anderson"	"Hard"	"Brisbane"	"2018-01-01"	"Australian Open"	"2018-01-15"	"Canada Masters"	"2018-08-06"
"Gilles Simon"	"Marin Cilic"	"Ryan Harrison"	"Hard"	"Pune"	"2018-01-01"	"Australian Open"	"2018-01-15"	"Winston-Salem"	"2018-08-20"
"Kyle Edmund"	"Hyeon Chung"	"Alexander Zverev"	"Hard"	"Brisbane"	"2018-01-01"	"Australian Open"	"2018-01-15"	"Shanghai Masters"	"2018-10-08"
"Andrey Rublev"	"Fernando Verdasco"	"Roberto Bautista Agut"	"Hard"	"Doha"	"2018-01-01"	"Australian Open"	"2018-01-15"	"Shanghai Masters"	"2018-10-08"

3 | Dataset 2

3.1. Introduction

The second dataset we have chosen is *Crime Data from 2020 to Present*, which reflects incidents of crime in the City of Los Angeles dating back to 2020. This dataset is a comprehensive repository of crime reports, including detailed information about dates, times, locations, types of crimes, victim details, and modus operandi. The dataset is maintained by the Los Angeles Police Department (LAPD), and while it aims to be accurate, it may include some discrepancies due to the manual transcription of original crime reports.

The primary objective of this analysis is to extract valuable insights that can assist public safety authorities, policymakers, and the community in understanding crime trends and patterns. As the dataset contains over 1 million data points, it provides a rich basis for large-scale analytics. For our analysis, we have utilized MongoDB, a NoSQL database, due to its capability to efficiently handle large datasets and support flexible querying of complex JSON data structures.

3.2. Data Wrangling and Transformation

The raw dataset was provided in CSV format and required preprocessing before being loaded into MongoDB. The transformation process was implemented in Python, as shown in the code snippet below:

Listing 3.1: Transformation of CSV to JSON

```
import csv
import json

def transform_csv_to_json(csv_file, json_file):
    with open(csv_file, mode='r') as file:
        reader = csv.DictReader(file)
```

```
transformed_data = []

for row in reader:
    document = {
        "DR_NO": row["DR_NO"],
        "DateReported": row["Date␣Rptd"],
        "DateOccurred": row["DATE␣OCC"],
        "TimeOccurred": row["TIME␣OCC"],
        "Area": {
            "Code": row["AREA"],
            "Name": row["AREA␣NAME"]
        },
        "ReportDistrict": row["Rpt␣Dist␣No"],
        "Part": row["Part␣1-2"],
        "CrimeCode": {
            "Primary": row["Crm␣Cd"],
            "Description": row["Crm␣Cd␣Desc"],
            "AdditionalCodes": [
                row["Crm␣Cd␣1"],
                row["Crm␣Cd␣2"],
                row["Crm␣Cd␣3"],
                row["Crm␣Cd␣4"]
            ]
        },
        "MOCodes": row["Mocodes"],
        "Victim": {
            "Age": row["Vict␣Age"],
            "Sex": row["Vict␣Sex"],
            "Descent": row["Vict␣Descent"]
        },
        "Premises": {
            "Code": row["Premis␣Cd"],
            "Description": row["Premis␣Desc"]
        },
        "Weapon": {
            "Code": row["Weapon␣Used␣Cd"],
            "Description": row["Weapon␣Desc"]
        },
        "Status": {
```

```
                    "Code": row["Status"],
                    "Description": row["Status␣Desc"]
                },
                "Location": {
                    "Address": row["LOCATION"],
                    "CrossStreet": row["Cross␣Street"],
                    "Coordinates": {
                        "Latitude": float(row["LAT"]) if row["LAT"
                            ] else None,
                        "Longitude": float(row["LON"]) if row["LON
                            "] else None
                    }
                }
            }

            # Remove empty additional crime codes
            document["CrimeCode"]["AdditionalCodes"] = [
                code for code in document["CrimeCode"]["
                    AdditionalCodes"] if code
            ]

            transformed_data.append(document)

    with open(json_file, mode='w') as file:
        json.dump(transformed_data, file, indent=4)
```

Key transformations performed include:

- Creation of nested structures for areas, crime codes, victim details, premises, weapons, and location to facilitate efficient querying.
- Conversion of latitude and longitude fields to numeric data types to enable geospatial operations.
- Removal of empty additional crime codes to reduce data redundancy.

This transformation resulted in a JSON file containing documents, which were subsequently imported into MongoDB for further analysis.

3.3. Database Schema

The MongoDB schema used for this dataset reflects the structure defined in the transformation script. The key collections and their attributes are outlined below:

3.3.1. Attributes

- **DR_NO**: Unique Division of Records number.
- **DateReported**: The date the crime was reported (MM/DD/YYYY).
- **DateOccurred**: The date the crime occurred (MM/DD/YYYY).
- **TimeOccurred**: The time the crime occurred in 24-hour format.
- **Area**: Nested structure containing:
  - **Code**: Numeric code for the LAPD Geographic Area.
  - **Name**: Name of the LAPD Geographic Area.
- **ReportDistrict**: Reporting district code for statistical purposes.
- **Part**: Indicates Part 1 or Part 2 crime classification.
- **CrimeCode**: Nested structure containing:
  - **Primary**: Primary crime code.
  - **Description**: Description of the primary crime code.
  - **AdditionalCodes**: List of additional crime codes, if any.
- **MOCodes**: Modus operandi codes describing the suspect's actions.
- **Victim**: Nested structure containing:
  - **Age**: Age of the victim.
  - **Sex**: Sex of the victim (F, M, or X).
  - **Descent**: Descent code for the victim.
- **Premises**: Nested structure containing:
  - **Code**: Code for the type of premises.
  - **Description**: Description of the premises type.
- **Weapon**: Nested structure containing:

  - **Code**: Code for the weapon used.
  - **Description**: Description of the weapon used.
- **Status**: Nested structure containing:
  - **Code**: Status code for the case.
  - **Description**: Description of the status code.
- **Location**: Nested structure containing:
  - **Address**: Approximate address of the incident.
  - **CrossStreet**: Cross street of the address.
  - **Coordinates**: Latitude and longitude of the location.

4 | Queries 2

4.1. Youngest Victim

Find the crime typologies with the lowest average age of the victim.

```
    db.crime\_reports.aggregate([
    { $match: { "Victim.Age": { $ne: "0" } } },
    { $group: { \_id: "$CrimeCode.Description", avgAge: { $avg: { $toInt:
    "$Victim.Age" } } } },
    { $sort: { avgAge: 1 } },
    { $limit: 10 }
])
```


```
[
  { "_id": "CHILD NEGLECT (SEE 300 W.I.C.)", "avgAge": 8.180310880829015 },
  {
    "_id": "CRM AGNST CHLD (13 OR UNDER) (14-15 & SUSP 10 YRS OLDER)",
    "avgAge": 10.65020103388857
  },
  { "_id": "CHILD ABANDONMENT", "avgAge": 11 },
  {
    "_id": "CHILD ABUSE (PHYSICAL) - SIMPLE ASSAULT",
    "avgAge": 11.526331018518519
  },
  {
    "_id": "CHILD ABUSE (PHYSICAL) - AGGRAVATED ASSAULT",
    "avgAge": 11.804263565891473
  },
  { "_id": "CHILD ANNOYING (17YRS & UNDER)", "avgAge": 13.342079689018465 },
  {
    "_id": "LEWD/LASCIVIOUS ACTS WITH CHILD",
```

```
    "avgAge": 13.590361445783133
  },
  { "_id": "CONTRIBUTING", "avgAge": 15.318181818181818 },
  {
    "_id": "SEX,UNLAWFUL(INC MUTUAL CONSENT, PENETRATION W/ FRGN OBJ",
    "avgAge": 16.008434864104967
  },
  {
    "_id": "HUMAN TRAFFICKING - COMMERCIAL SEX ACTS",
    "avgAge": 17.471238938053098
  }
]
```

4.2. Number of crimes per year

Find the total amount of crimes happened every year.

```
    db.crime_reports.aggregate([
    {
        $addFields: {
            year: { $year: { $dateFromString: { dateString: "$DateReported" } } },
            month: { $month: { $dateFromString: { dateString: "$DateReported" } } }
        }
    },
    {
        $group: {
            _id: { year: "$year"},
            totalCrimes: { $sum: 1 }
        }
    },
    { $sort: { "_id.year": 1, "_id.month": 1 } }
]);
```

```
[
  { _id: { year: 2020 }, totalCrimes: 192708 },
  { _id: { year: 2021 }, totalCrimes: 208284 },
  { _id: { year: 2022 }, totalCrimes: 235064 },
```

```
  { _id: { year: 2023 }, totalCrimes: 234651 },
  { _id: { year: 2024 }, totalCrimes: 130405 }
]
```

4.3. Most Dangerous Areas for Battery

Find the areas where battery (simple assault) has happened the most.

```
    db.crime_reports.aggregate([
    { $match: { "CrimeCode.Description": "BATTERY - SIMPLE ASSAULT" } },
    {
        $group: {
            _id: "$Area.Name",
            totalCrimes: { $sum: 1 }
        }
    },
    { $sort: { totalCrimes: -1 } }
]);
```

```
[
  { _id: 'Central', totalCrimes: 6806 },
  { _id: '77th Street', totalCrimes: 4707 },
  { _id: 'Southwest', totalCrimes: 4528 },
  { _id: 'Hollywood', totalCrimes: 4484 },
  { _id: 'Olympic', totalCrimes: 4380 },
  { _id: 'Newton', totalCrimes: 4156 },
  { _id: 'Rampart', totalCrimes: 4133 },
  { _id: 'Southeast', totalCrimes: 3939 },
  { _id: 'N Hollywood', totalCrimes: 3433 },
  { _id: 'Pacific', totalCrimes: 3332 },
  { _id: 'Wilshire', totalCrimes: 3164 },
  { _id: 'Harbor', totalCrimes: 3156 },
  { _id: 'Hollenbeck', totalCrimes: 3091 },
  { _id: 'Van Nuys', totalCrimes: 2850 },
  { _id: 'West LA', totalCrimes: 2831 },
  { _id: 'Topanga', totalCrimes: 2766 },
  { _id: 'West Valley', totalCrimes: 2749 },
```

```
  { _id: 'Devonshire', totalCrimes: 2687 },
  { _id: 'Mission', totalCrimes: 2606 },
  { _id: 'Northeast', totalCrimes: 2593 },
  { _id: 'Foothill', totalCrimes: 2419 }
]
```

4.4. Most Used Weapons

Find the most used weapons, excluding crimes for which it is not specified.

```
    db.crime_reports.aggregate([
    {
        $match: { "Weapon.Description": { $ne: "" } }
    },
    {
        $group: {
            _id: "$Weapon.Description",
            totalIncidents: { $sum: 1 }
        }
    },
    {
        $sort: { totalIncidents: -1 }
    },
    { $limit: 10 }
]);
```

```
[
  { _id: 'STRONG-ARM (HANDS, FIST, FEET OR BODILY FORCE)', totalIncidents: 174693 },
  { _id: 'UNKNOWN WEAPON/OTHER WEAPON', totalIncidents: 36289 },
  { _id: 'VERBAL THREAT', totalIncidents: 23835 },
  { _id: 'HAND GUN', totalIncidents: 20179 },
  { _id: 'SEMI-AUTOMATIC PISTOL', totalIncidents: 7266 },
  { _id: 'KNIFE WITH BLADE 6INCHES OR LESS', totalIncidents: 6836 },
  { _id: 'UNKNOWN FIREARM', totalIncidents: 6581 },
  { _id: 'OTHER KNIFE', totalIncidents: 5880 },
  { _id: 'MACE/PEPPER SPRAY', totalIncidents: 3729 },
  { _id: 'VEHICLE', totalIncidents: 3258 }
```

```
]
```

4.5. Crimes Percentage per Area

Find, for each area, the total amount of reported crimes and the percentage over the area's population, sorting the results on this decreasing values of this percentage (corresponds to sketchiest areas first).

```
db.crime_reports.aggregate([
{
    $group: {
        _id: "$Area.Name",
        totalCrimes: { $sum: 1 }
    }
},
{
    $group: {
        _id: null,
        areas: { $push: { area: "$_id", count: "$totalCrimes" } },
        totalCrimes: { $sum: "$totalCrimes" }
    }
},
{ $unwind: "$areas" },
{
    $project: {
        _id: "$areas.area",
        totalCrimes: "$areas.count",
        percentage: { $multiply: [{ $divide: ["$areas.count", "$totalCrimes"] }
    }
},
{
    $sort: { percentage: -1 }
}
]);
```

```
[
  {
```

```
    "_id": "Central",
    "totalCrimes": 69330,
    "percentage": 6.9252990674370105
  },
  {
    "_id": "77th Street",
    "totalCrimes": 61624,
    "percentage": 6.15555502281463
  },
  { "_id": "Pacific", "totalCrimes": 59184, "percentage": 5.911826049433031 },
  {
    "_id": "Southwest",
    "totalCrimes": 57198,
    "percentage": 5.713446647328171
  },
  {
    "_id": "Hollywood",
    "totalCrimes": 52239,
    "percentage": 5.2180974756071254
  },
  {
    "_id": "N Hollywood",
    "totalCrimes": 50911,
    "percentage": 5.085444985176483
  },
  {
    "_id": "Olympic",
    "totalCrimes": 49887,
    "percentage": 4.9831587274950255
  },
  {
    "_id": "Southeast",
    "totalCrimes": 49827,
    "percentage": 4.977165392084003
  },
  {
    "_id": "Newton",
```

```
    "totalCrimes": 49041,
    "percentage": 4.898652698199602
  },
  {
    "_id": "Wilshire",
    "totalCrimes": 48015,
    "percentage": 4.79616666267111
  },
  { "_id": "Rampart", "totalCrimes": 46678, "percentage": 4.662615171928816 },
  { "_id": "West LA", "totalCrimes": 45536, "percentage": 4.548542021272345 },
  {
    "_id": "Northeast",
    "totalCrimes": 42772,
    "percentage": 4.272449036671222
  },
  {
    "_id": "Van Nuys",
    "totalCrimes": 42730,
    "percentage": 4.268253701883506
  },
  {
    "_id": "West Valley",
    "totalCrimes": 41998,
    "percentage": 4.195135009869025
  },
  {
    "_id": "Devonshire",
    "totalCrimes": 41578,
    "percentage": 4.153181661991865
  },
  { "_id": "Topanga", "totalCrimes": 41216, "percentage": 4.117021871678693 },
  { "_id": "Harbor", "totalCrimes": 41190, "percentage": 4.11442475966725 },
  {
    "_id": "Mission",
    "totalCrimes": 40211,
    "percentage": 4.0166335035440595
  },
```

```
  {
    "_id": "Hollenbeck",
    "totalCrimes": 36913,
    "percentage": 3.6871998337848315
  },
  {
    "_id": "Foothill",
    "totalCrimes": 33034,
    "percentage": 3.2997306994621978
  }
]
```


4.6. Crimes statistics for hour of the day

Find, for each hour of the day, how many crimes were reported in total and the most frequent type of crime.

```
db.crime_reports.aggregate([
{
    $addFields: {
        hourOccurred: { $toInt: { $substr: ["$TimeOccurred", 0, 2] } }
    }
},
{
    $group: {
        _id: { hour: "$hourOccurred", crime: "$CrimeCode.Description" },
        totalCrimes: { $sum: 1 }
    }
},
{
    $sort: { "_id.hour": 1, totalCrimes: -1 }
},
{
    $group: {
        _id: "$_id.hour",
        totalCrimes: { $sum: "$totalCrimes" },
        mostCommonCrime: { $first: "$_id.crime" },
```

```
            mostCommonCrimeCount: { $first: "$totalCrimes" }
        }
    },
    {
        $sort: { _id: 1 }
    }
]);
```

```
[
  {
    _id: 0,
    totalCrimes: 40351,
    mostCommonCrime: 'THEFT OF IDENTITY',
    mostCommonCrimeCount: 6143
  },
  {
    _id: 1,
    totalCrimes: 29655,
    mostCommonCrime: 'VEHICLE - STOLEN',
    mostCommonCrimeCount: 3206
  },
  {
    _id: 2,
    totalCrimes: 25149,
    mostCommonCrime: 'VEHICLE - STOLEN',
    mostCommonCrimeCount: 2875
  },
  {
    _id: 3,
    totalCrimes: 22112,
    mostCommonCrime: 'BURGLARY',
    mostCommonCrimeCount: 2991
  },
  {
    _id: 4,
    totalCrimes: 18719,
```

```
    mostCommonCrime: 'BURGLARY',
    mostCommonCrimeCount: 2754
  },
  {
    _id: 5,
    totalCrimes: 17225,
    mostCommonCrime: 'VEHICLE - STOLEN',
    mostCommonCrimeCount: 2352
  },
  {
    _id: 6,
    totalCrimes: 23116,
    mostCommonCrime: 'THEFT OF IDENTITY',
    mostCommonCrimeCount: 3803
  },
  {
    _id: 7,
    totalCrimes: 26171,
    mostCommonCrime: 'VEHICLE - STOLEN',
    mostCommonCrimeCount: 2896
  },
  {
    _id: 8,
    totalCrimes: 37106,
    mostCommonCrime: 'THEFT OF IDENTITY',
    mostCommonCrimeCount: 4150
  },
  {
    _id: 9,
    totalCrimes: 36396,
    mostCommonCrime: 'THEFT OF IDENTITY',
    mostCommonCrimeCount: 3575
  },
  {
    _id: 10,
    totalCrimes: 42846,
    mostCommonCrime: 'BATTERY - SIMPLE ASSAULT',
```

```
    mostCommonCrimeCount: 3672
  },
  {
    _id: 11,
    totalCrimes: 43482,
    mostCommonCrime: 'BATTERY - SIMPLE ASSAULT',
    mostCommonCrimeCount: 3911
  },
  {
    _id: 12,
    totalCrimes: 67549,
    mostCommonCrime: 'THEFT OF IDENTITY',
    mostCommonCrimeCount: 10340
  },
  {
    _id: 13,
    totalCrimes: 45387,
    mostCommonCrime: 'BATTERY - SIMPLE ASSAULT',
    mostCommonCrimeCount: 4212
  },
  {
    _id: 14,
    totalCrimes: 49098,
    mostCommonCrime: 'BATTERY - SIMPLE ASSAULT',
    mostCommonCrimeCount: 4327
  },
  {
    _id: 15,
    totalCrimes: 52613,
    mostCommonCrime: 'VEHICLE - STOLEN',
    mostCommonCrimeCount: 5068
  },
  {
    _id: 16,
    totalCrimes: 52748,
    mostCommonCrime: 'VEHICLE - STOLEN',
    mostCommonCrimeCount: 5728
```

```
  },
  {
    _id: 17,
    totalCrimes: 58533,
    mostCommonCrime: 'VEHICLE - STOLEN',
    mostCommonCrimeCount: 7452
  },
  {
    _id: 18,
    totalCrimes: 59703,
    mostCommonCrime: 'VEHICLE - STOLEN',
    mostCommonCrimeCount: 8687
  },
  {
    _id: 19,
    totalCrimes: 55379,
    mostCommonCrime: 'VEHICLE - STOLEN',
    mostCommonCrimeCount: 8139
  },
  {
    _id: 20,
    totalCrimes: 56094,
    mostCommonCrime: 'VEHICLE - STOLEN',
    mostCommonCrimeCount: 8494
  },
  {
    _id: 21,
    totalCrimes: 50632,
    mostCommonCrime: 'VEHICLE - STOLEN',
    mostCommonCrimeCount: 7916
  },
  {
    _id: 22,
    totalCrimes: 48929,
    mostCommonCrime: 'VEHICLE - STOLEN',
    mostCommonCrimeCount: 8217
  },
```

```
  {
    _id: 23,
    totalCrimes: 42119,
    mostCommonCrime: 'VEHICLE - STOLEN',
    mostCommonCrimeCount: 6467
  }
]
```

4.7. Most Common Premises for Crime Type

Find where the most common crimes happen the most.

```
    db.crime_reports.aggregate([
    {
        $group: {
            _id: { crime: "$CrimeCode.Description",
            premises: "$Premises.Description" },
            totalCrimes: { $sum: 1 }
        }
    },
    {
        $sort: { "_id.crime": 1, totalCrimes: -1 }
    },
    {
        $group: {
            _id: "$_id.crime",
            mostCommonPremises: { $first: "$_id.premises" },
            totalCrimes: { $first: "$totalCrimes" }
        }
    },
    { $sort: { totalCrimes: -1 } },
    { $limit: 10}

]);
```

```
[
  {
```

```
    _id: 'VEHICLE - STOLEN',
    mostCommonPremises: 'STREET',
    totalCrimes: 89364
  },
  {
    _id: 'THEFT OF IDENTITY',
    mostCommonPremises: 'SINGLE FAMILY DWELLING',
    totalCrimes: 30827
  },
  {
    _id: 'THEFT FROM MOTOR VEHICLE - PETTY ($950 & UNDER)',
    mostCommonPremises: 'STREET',
    totalCrimes: 27214
  },
  {
    _id: 'BURGLARY FROM VEHICLE',
    mostCommonPremises: 'STREET',
    totalCrimes: 25499
  },
  {
    _id: 'ASSAULT WITH DEADLY WEAPON, AGGRAVATED ASSAULT',
    mostCommonPremises: 'STREET',
    totalCrimes: 18830
  },
  {
    _id: 'BURGLARY',
    mostCommonPremises: 'SINGLE FAMILY DWELLING',
    totalCrimes: 18329
  },
  {
    _id: 'THEFT FROM MOTOR VEHICLE - GRAND ($950.01 AND OVER)',
    mostCommonPremises: 'STREET',
    totalCrimes: 18167
  },
  {
    _id: 'VANDALISM - FELONY ($400 & OVER, ALL CHURCH VANDALISMS)',
    mostCommonPremises: 'VEHICLE, PASSENGER/TRUCK',
```

```
    totalCrimes: 17898
  },
  {
    _id: 'INTIMATE PARTNER - SIMPLE ASSAULT',
    mostCommonPremises: 'MULTI-UNIT DWELLING (APARTMENT, DUPLEX, ETC)',
    totalCrimes: 15973
  },
  {
    _id: 'BATTERY - SIMPLE ASSAULT',
    mostCommonPremises: 'SINGLE FAMILY DWELLING',
    totalCrimes: 12611
  }
]
```

4.8. Top 5 Areas with the Youngest Average Victims

Find the five areas in which the victim reporting the crime is youngest on average.

```
db.crime_reports.aggregate([
{ $match: { "Victim.Age": { $ne: "0" } } },
{
    $group: {
        _id: "$Area.Name", // Group by area name
        avgVictimAge: { $avg: { $toInt: "$Victim.Age" } },
        totalVictims: { $sum: 1 }
    }
},
{ $sort: { avgVictimAge: 1 } },
{ $limit: 5 }
]);
```

```
[
  {
    _id: 'Southwest',
    avgVictimAge: 35.707343208320204,
    totalVictims: 47595
  },
```

```
  {
    _id: 'Newton',
    avgVictimAge: 37.29770373705538,
    totalVictims: 33315
  },
  {
    _id: 'Rampart',
    avgVictimAge: 37.591953848489446,
    totalVictims: 32935
  },
  {
    _id: 'Hollywood',
    avgVictimAge: 37.866876149601474,
    totalVictims: 39144
  },
  {
    _id: 'Southeast',
    avgVictimAge: 37.919636981794255,
    totalVictims: 36472
  }
]
```

4.9. Top 5 Crime Types Involving Victims Over 65

Find which are the most common types of crime in which the victim is over 65y.o.

```
    db.crime_reports.aggregate([
    {
        $addFields: {
            victimAgeInt: { $toInt: "$Victim.Age" }
        }
    },
    { $match: { victimAgeInt: { $gte: 65 } } },
    {
        $group: {
            _id: "$CrimeCode.Description",
            totalCrimes: { $sum: 1 }
```

```
        }
    },
    { $sort: { totalCrimes: -1 } },
    { $limit: 5 }
]);
```

```
[
  { "_id": "BURGLARY", "totalCrimes": 6476 },
  { "_id": "BATTERY - SIMPLE ASSAULT", "totalCrimes": 6349 },
  { "_id": "THEFT OF IDENTITY", "totalCrimes": 5520 },
  { "_id": "THEFT PLAIN - PETTY ($950 & UNDER)", "totalCrimes": 3821 },
  {
    "_id": "VANDALISM - FELONY ($400 & OVER, ALL CHURCH VANDALISMS)",
    "totalCrimes": 3516
  }
]
```

4.10. Most Frequent Crimes with Female Victims

Find the 10 crime typologies in which a female is the victim.

```
    db.crime_reports.aggregate([
    {
        $match: { "Victim.Sex": "F" }
    },
    {
        $group: {
            _id: "$CrimeCode.Description",
            totalCrimes: { $sum: 1 }
        }
    },
    {
        $sort: { totalCrimes: -1 }
    },
    { $limit: 10 }
]);
```

```
[
  { "_id": "THEFT OF IDENTITY", "totalCrimes": 35633 },
  { "_id": "INTIMATE PARTNER - SIMPLE ASSAULT", "totalCrimes": 35506 },
  { "_id": "BATTERY - SIMPLE ASSAULT", "totalCrimes": 35136 },
  { "_id": "BURGLARY FROM VEHICLE", "totalCrimes": 26336 },
  { "_id": "THEFT PLAIN - PETTY ($950 & UNDER)", "totalCrimes": 23043 },
  {
    "_id": "VANDALISM - FELONY ($400 & OVER, ALL CHURCH VANDALISMS)",
    "totalCrimes": 21264
  },
  { "_id": "BURGLARY", "totalCrimes": 15858 },
  {
    "_id": "ASSAULT WITH DEADLY WEAPON, AGGRAVATED ASSAULT",
    "totalCrimes": 14325
  },
  {
    "_id": "THEFT FROM MOTOR VEHICLE - GRAND ($950.01 AND OVER)",
    "totalCrimes": 13839
  },
  {
    "_id": "THEFT-GRAND ($950.01 & OVER)EXCPT,GUNS,FOWL,LIVESTK,PROD",
    "totalCrimes": 12432
  }
]
```