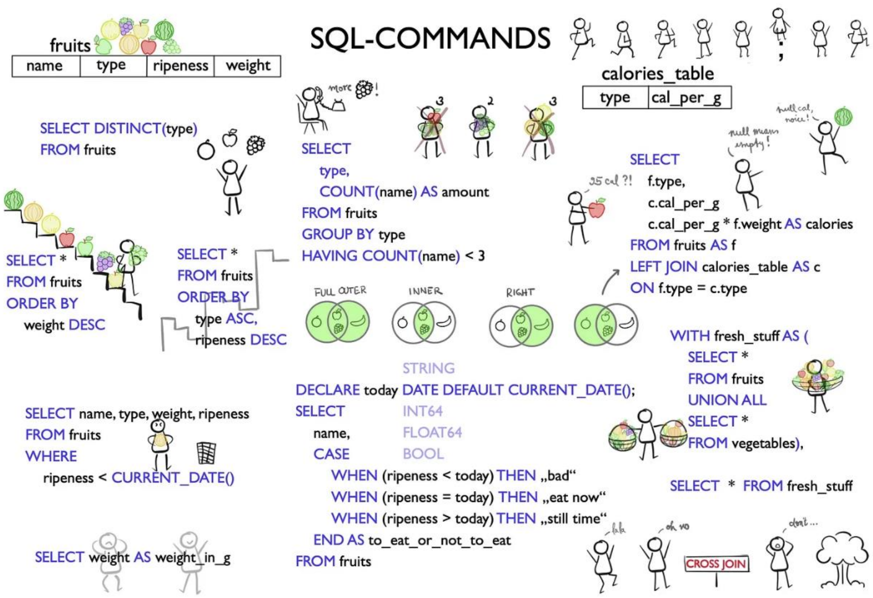
Capstone

1. Create a Google Cloud Account
2. Download R Studio
3. Upload dataset to Google Cloud Storage (create bucket)
4. Access dataset in Big Query (upload from GCS due to file size)
5. Some Helper basic SQL commands:



Source: https://medium.com/data-school/the-best-bigquery-sql-cheat-sheet-for-beginners-81c762f72845

6. Data explained -> note for next trimester to keep note which units were being used

- Timestamp – date + time (inc seconds) + time zone
- Timestamp_AEST – date + time (inc seconds) in current AEST time (Melbourne time is AEST in summer and AEDT in winter)
- Date_AEST – date only in format dd/mm/yyyy
- Distance – expressed in kilometers
- Enhanced_altitude – we ride faster at altitude than at sea level
- Ascent- usually expressed in %
- Grade – Gradient, Slope. It is steepness of an ascent of descent. Usually expressed in %
- Calories burnt
- Enhanced_speed – km/ min?
- Heart_rate – in beats per minute
- Temperature
- Cadence – RPM – number of revolutions per minute (pedalling rate). An RPM = 60 means 1 pedal revolution (whole 360 degrees) a second
- Power- measures how much work a cyclist is doing on the bike, and is expressed in watts

200-300 is average for recreational cycling

- GPS_accuracy
- Session_ID – add a unique one for each session
- User_ID – unique for each user
- Age
- Gender
- Weight – in kg
- FTP (functional threshold power)

Add:

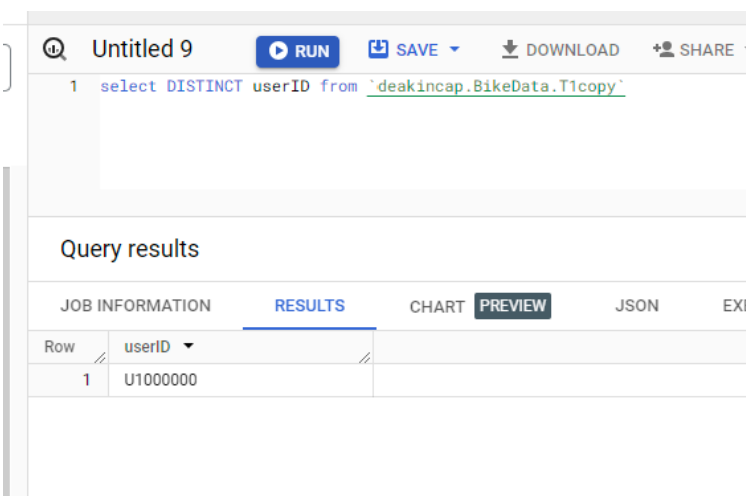
- Duration = distance / speed ; check units

Plan:

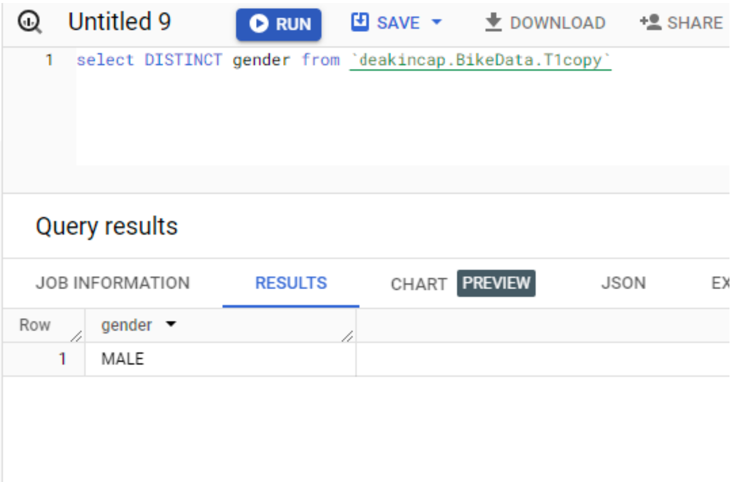
- Copy dataset , work on copy
- Check data types and units
- Make note that there are null values throughout the dataset
- Insert uniquely generated sessionID's
- Insert 'duration' column & create a formula to calculate it – need to check the data units

- DATA WRANGLING

Consistent userID check:



a) Consistent gender check (making sure MALE appears 1 way, not male, MALE, Male, MaLe etc)



b) Unify degrees of precision:

enhanced_speed
25.5384
32.8788
29.9952
28.2384
26.0892
9.2808
23.1912
26.7804
23.49
30.096
32.7888
29.9268
25.5672
24.6708
30.1968
28.6128
30.9996
30.3696
32.9292
30.87

enhanced_speed
11.0
29.8
20.0
17.8
29.8
31.6
30.1
21.9
25.2
30.0
22.2
26.6
30.0
29.4
29.9
22.1
29.0
7.8
31.2
25.4
32.8

NEXT:

- Finish data Wrangling:
- Delete rows with missing values, nulls

- BigQuery does not have a built-in K-means clustering function, so I will have to perform clustering in Rstudio
- Example patterns:

Time of day patterns, intensity levels, activity profile (casual, pro, intensive), effort consistency,

week 3 progress report:

- Have connected with the P1 SmartBike project
- Have agreed with Victor (Web Dev) on creating an additional page featuring visualizations and taking advantage of findings from clustering, such as fitness levels, engagement patterns, training intensity, goal achievement (+ projected results) and more
- Started Data Wrangling process in BigQuery so that the structure and quality of data can yield reliable results of data clustering – will be finished with Wrangling tonight; Will also try to use SQL queries in BigQuery to perform trend analysis.
- BigQuery does not have a built-in K-means clustering function, so I will have to perform clustering in Rstudio – start - tomorrow
- Will list all the possible inferences that can be drawn from the analysis in R, and contact Web Dev BIke team regarding which they would be most interested in featuring on the additional page
- By tues/ wed myself and Victor will start working on the visualization and I will additionally import the clustered data to Tableau to showcase different visualizations there (dashboards)

- Ask Ella about data Warehouse report (BikeData T1)

- NEXT trimester: application that fetches clustering results from your R script via an API.

1. Stream Data into BigQuery (use API)
2. Cluster in Rstudio
3. Visualization with React (?)
- Integrate app with R script
- Integrate React with API
4. Schedule Real time updates

Saturday 25 Nov 2023

So the data was collected as data points, which means that every second a data update was recorded and inserted in the CSV file.

1. Adding session ID to those workouts that were conducted consecutively. Any state of ‘idle’ for over 5 minutes (line 17) would be categorized as a new session. Later development can feature a ‘Are you still here?’ countdown of 5 minutes, when the bike is not being used (if the rider stopped interacting).

```
UPDATE `deakincap.BikeData.T1copy3`
SET session_ID = Subquery.session_ID
FROM (
  WITH WorkoutSessions AS (
    SELECT
      tiemstamp_AEST,
      heart_rate,
      LAG(tiemstamp_AEST) OVER (ORDER BY tiemstamp_AEST) AS prev_timestamp,
      TIMESTAMP_DIFF(tiemstamp_AEST, LAG(tiemstamp_AEST) OVER (ORDER BY tiemstamp_AEST), MINUTE) AS time_diff
    FROM
      `deakincap.BikeData.T1copy3`
  )

  SELECT
    tiemstamp_AEST,
    IFNULL(SUM(IF(time_diff > 5 OR prev_timestamp IS NULL, 1, 0))
      OVER (ORDER BY tiemstamp_AEST), 0) + 1 AS session_ID
  FROM
    WorkoutSessions
) AS Subquery
WHERE `deakincap.BikeData.T1copy3`.tiemstamp_AEST = Subquery.tiemstamp_AEST;
```

Result: (last column)

userID	age	gender	weight	FTP	session_ID
U1000000	33	MALE	80	301	512
U1000000	33	MALE	80	301	512
U1000000	33	MALE	80	301	512
U1000000	33	MALE	80	301	260
U1000000	33	MALE	80	301	260
U1000000	33	MALE	80	301	516
U1000000	33	MALE	80	301	519
U1000000	33	MALE	80	301	519
U1000000	33	MALE	80	301	519
U1000000	33	MALE	80	301	519
U1000000	33	MALE	80	301	522
U1000000	33	MALE	80	301	522
U1000000	33	MALE	80	301	523

2. In order to prepare data so that each category would reflect within a particular session and / or time window, I decided to group the records by day, hour and userID.

BEFORE:

```sql
select * from `deakincap.BikeData.T1copy`
order by tiemstamp_AEST
LIMIT 20
```

Processing location: US

Query results

SAVE RESULTS

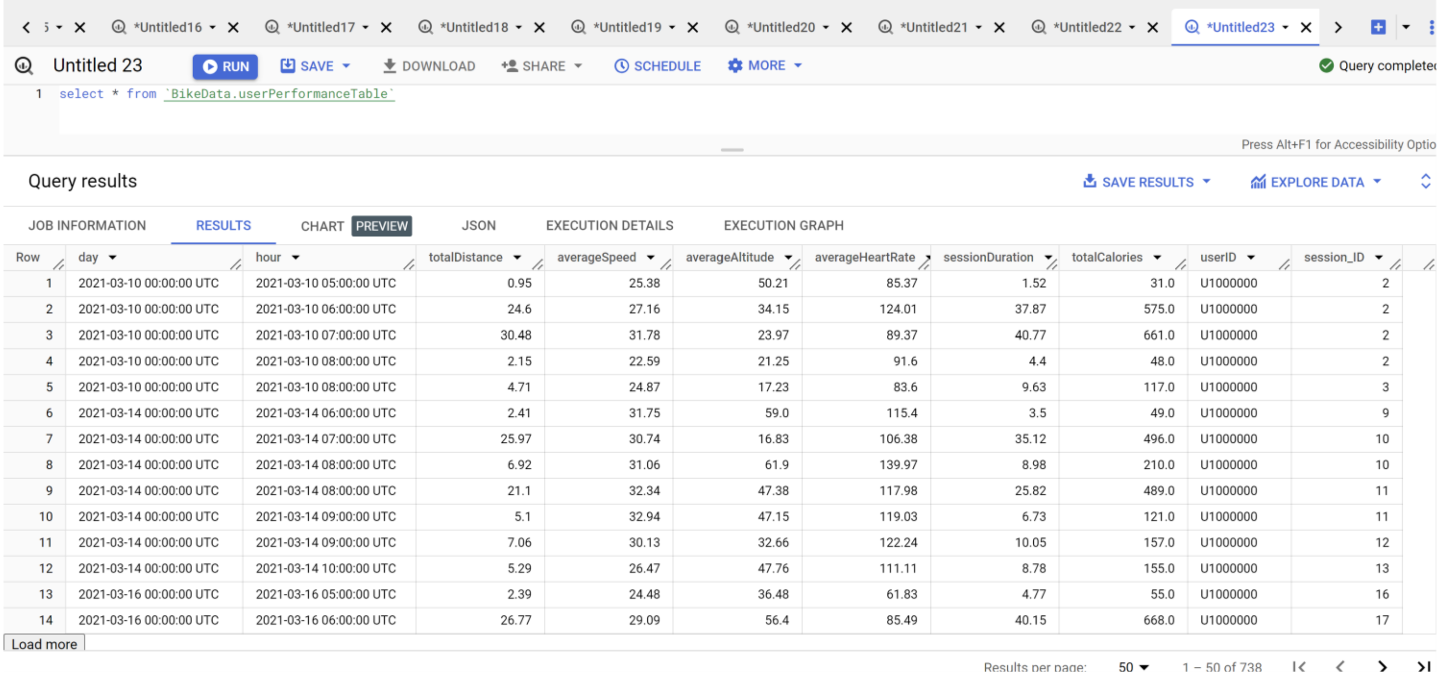
JOB INFORMATION | RESULTS | CHART PREVIEW | JSON | EXECUTION DETAILS | EXECUTION GRAPH

Row	tiemstamp_AEST	date_AEST	distance	enhanced_altitude	ascent	grade	calories	enhanced_speed	heart_rate	tempera
1	2021-03-10 05:57:32 UTC	2021-03-10	0.0	46.6	0	-3.83	0	25.8	80	
2	2021-03-10 05:57:33 UTC	2021-03-10	0.01	46.2	0	-3.91	0	25.2	80	
3	2021-03-10 05:57:34 UTC	2021-03-10	0.01	46.0	0	-4.01	0	24.6	81	
4	2021-03-10 05:57:35 UTC	2021-03-10	0.02	45.8	0	-4.07	0	24.5	81	
5	2021-03-10 05:57:36 UTC	2021-03-10	0.03	45.6	0	-3.99	0	23.9	81	
6	2021-03-10 05:57:37 UTC	2021-03-10	0.03	45.6	0	-3.8	0	23.8	80	
7	2021-03-10 05:57:38 UTC	2021-03-10	0.04	45.4	0	-3.51	0	23.3	80	
8	2021-03-10 05:57:39 UTC	2021-03-10	0.05	45.4	0	-2.99	0	23.0	80	
9	2021-03-10 05:57:40 UTC	2021-03-10	0.05	45.4	0	-2.37	0	24.1	80	
10	2021-03-10 05:57:41 UTC	2021-03-10	0.06	45.4	0	-1.79	0	25.1	81	
11	2021-03-10 05:57:45 UTC	2021-03-10	0.09	45.4	0	0.0	2	28.5	82	
12	2021-03-10 05:57:46 UTC	2021-03-10	0.1	45.4	0	0.0	2	29.2	82	
13	2021-03-10 05:57:48 UTC	2021-03-10	0.12	45.4	0	0.0	2	29.2	82	
14	2021-03-10 05:57:50 UTC	2021-03-10	0.13	45.4	0	0.0	3	29.7	82	
15	2021-03-10 05:57:52 UTC	2021-03-10	0.15	45.4	0	0.0	3	30.6	83	
16	2021-03-10 05:57:56 UTC	2021-03-10	0.18	45.4	0	0.0	5	31.4	84	
17	2021-03-10 05:57:57 UTC	2021-03-10	0.19	45.4	0	0.0	5	31.8	85	
18	2021-03-10 05:57:58 UTC	2021-03-10	0.2	45.4	0	0.0	5	31.9	85	
19	2021-03-10 05:58:02 UTC	2021-03-10	0.24	45.4	0	0.0	6	31.7	86	
20	2021-03-10 05:58:03 UTC	2021-03-10	0.25	45.4	0	0.0	6	32.0	86	

CREATE SORTED DATA TABLE 'userPerformanceTable', making sure no null values appear

```sql
create table deakincap.BikeData.userPerformanceTable AS
SELECT
  DATE_TRUNC(tiemstamp_AEST, DAY) AS day,
  DATE_TRUNC(tiemstamp_AEST, HOUR) AS hour,
  round(MAX(distance) - MIN(distance), 2) AS totalDistance,
  round(AVG(enhanced_speed),2) AS averageSpeed,
  round(AVG(enhanced_altitude),2) AS averageAltitude,
  round(AVG(heart_rate),2) AS averageHeartRate,
  round(COUNT(DISTINCT TIMESTAMP_TRUNC(tiemstamp_AEST, second)) / 60, 2) AS sessionDuration,
  ROUND(MAX(calories) - MIN(calories), 2) AS totalCalories,
  userID,
  session_ID
FROM
  `deakincap.BikeData.T1copy3`
GROUP BY
  day, hour, userID, session_ID
HAVING
  averageHeartRate IS NOT NULL
  AND totalDistance IS NOT NULL
  AND averageSpeed IS NOT NULL
  AND averageAltitude IS NOT NULL
  AND totalCalories IS NOT NULL
  AND sessionDuration IS NOT NULL
ORDER BY
  day, hour;
```

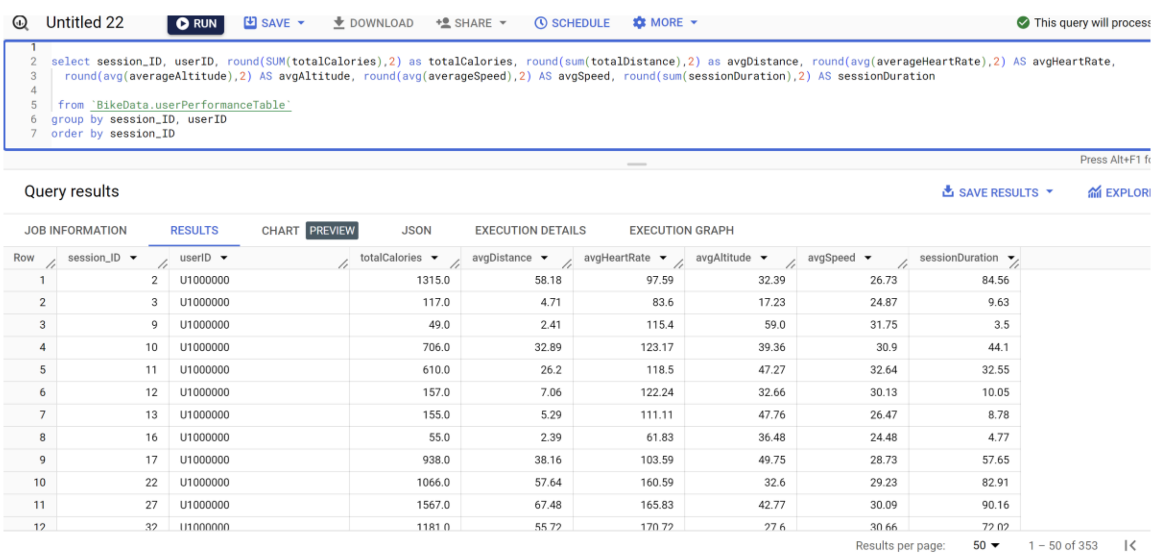
AFTER:




```
select * from `BikeData.userPerformanceTable`
```

Row	day	hour	totalDistance	averageSpeed	averageAltitude	averageHeartRate	sessionDuration	totalCalories	userID	session_ID
1	2021-03-10 00:00:00 UTC	2021-03-10 05:00:00 UTC	0.95	25.38	50.21	85.37	1.52	31.0	U1000000	2
2	2021-03-10 00:00:00 UTC	2021-03-10 06:00:00 UTC	24.6	27.16	34.15	124.01	37.87	575.0	U1000000	2
3	2021-03-10 00:00:00 UTC	2021-03-10 07:00:00 UTC	30.48	31.78	23.97	89.37	40.77	661.0	U1000000	2
4	2021-03-10 00:00:00 UTC	2021-03-10 08:00:00 UTC	2.15	22.59	21.25	91.6	4.4	48.0	U1000000	2
5	2021-03-10 00:00:00 UTC	2021-03-10 08:00:00 UTC	4.71	24.87	17.23	83.6	9.63	117.0	U1000000	3
6	2021-03-14 00:00:00 UTC	2021-03-14 06:00:00 UTC	2.41	31.75	59.0	115.4	3.5	49.0	U1000000	9
7	2021-03-14 00:00:00 UTC	2021-03-14 07:00:00 UTC	25.97	30.74	16.83	106.38	35.12	496.0	U1000000	10
8	2021-03-14 00:00:00 UTC	2021-03-14 08:00:00 UTC	6.92	31.06	61.9	139.97	8.98	210.0	U1000000	10
9	2021-03-14 00:00:00 UTC	2021-03-14 08:00:00 UTC	21.1	32.34	47.38	117.98	25.82	489.0	U1000000	11
10	2021-03-14 00:00:00 UTC	2021-03-14 09:00:00 UTC	5.1	32.94	47.15	119.03	6.73	121.0	U1000000	11
11	2021-03-14 00:00:00 UTC	2021-03-14 09:00:00 UTC	7.06	30.13	32.66	122.24	10.05	157.0	U1000000	12
12	2021-03-14 00:00:00 UTC	2021-03-14 10:00:00 UTC	5.29	26.47	47.76	111.11	8.78	155.0	U1000000	13
13	2021-03-16 00:00:00 UTC	2021-03-16 05:00:00 UTC	2.39	24.48	36.48	61.83	4.77	55.0	U1000000	16
14	2021-03-16 00:00:00 UTC	2021-03-16 06:00:00 UTC	26.77	29.09	56.4	85.49	40.15	668.0	U1000000	17

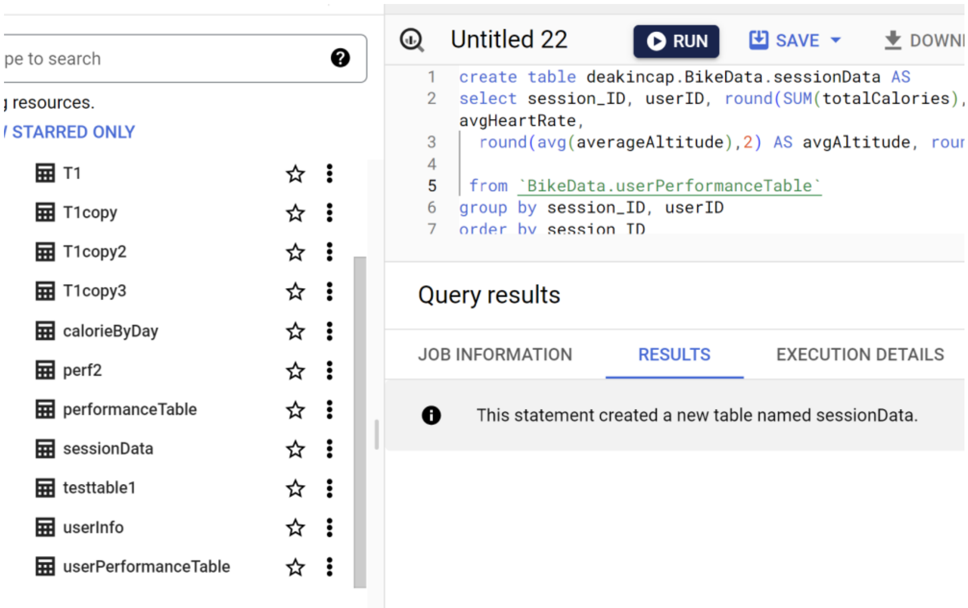
From here we can filter this table by each session's data:



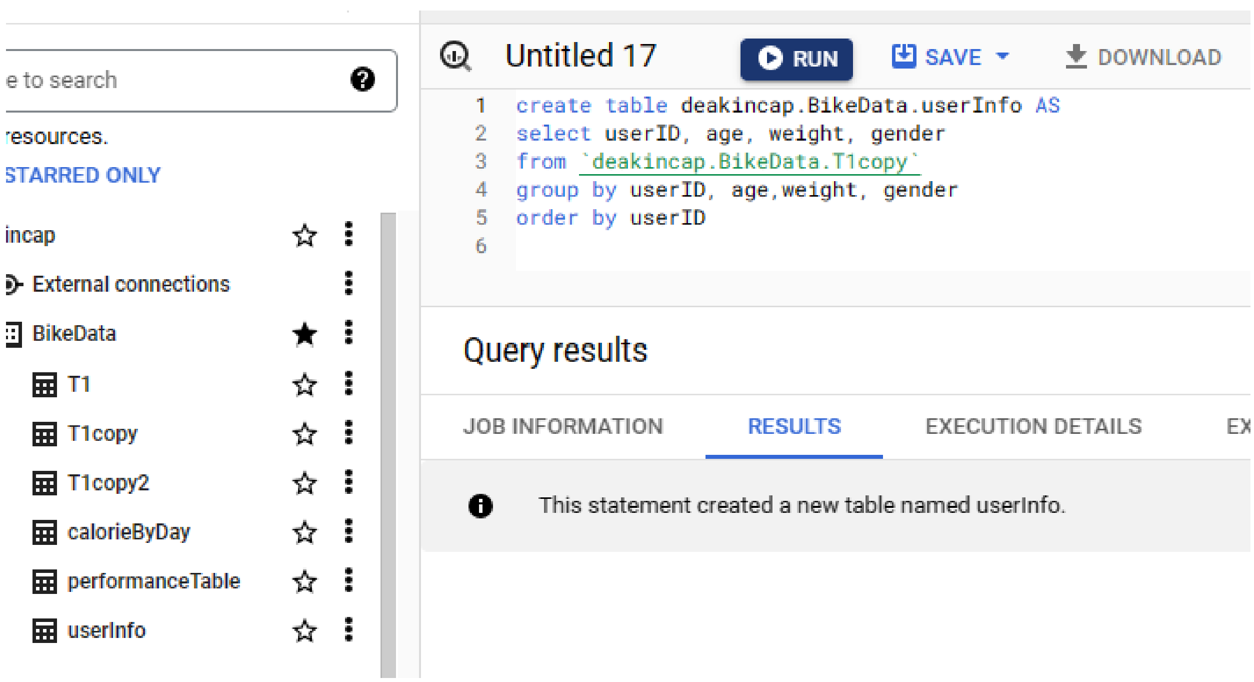

```
select session_ID, userID, round(SUM(totalCalories),2) as totalCalories, round(sum(totalDistance),2) as avgDistance, round(avg(averageHeartRate),2) AS avgHeartRate,
 round(avg(averageAltitude),2) AS avgAltitude, round(avg(averageSpeed),2) AS avgSpeed, round(sum(sessionDuration),2) AS sessionDuration

 from `BikeData.userPerformanceTable`
group by session_ID, userID
order by session_ID
```

Row	session_ID	userID	totalCalories	avgDistance	avgHeartRate	avgAltitude	avgSpeed	sessionDuration
1	2	U1000000	1315.0	58.18	97.59	32.39	26.73	84.56
2	3	U1000000	117.0	4.71	83.6	17.23	24.87	9.63
3	9	U1000000	49.0	2.41	115.4	59.0	31.75	3.5
4	10	U1000000	706.0	32.89	123.17	39.36	30.9	44.1
5	11	U1000000	610.0	26.2	118.5	47.27	32.64	32.55
6	12	U1000000	157.0	7.06	122.24	32.66	30.13	10.05
7	13	U1000000	155.0	5.29	111.11	47.76	26.47	8.78
8	16	U1000000	55.0	2.39	61.83	36.48	24.48	4.77
9	17	U1000000	938.0	38.16	103.59	49.75	28.73	57.65
10	22	U1000000	1066.0	57.64	160.59	32.6	29.23	82.91
11	27	U1000000	1567.0	67.48	165.83	42.77	30.09	90.16
12	32	U1000000	1181.0	55.72	170.72	27.6	30.66	72.02

So I will also save that Query result to a new Table called 'sessionData':



3. Create a user data table featuring only user – related data



Untitled 17

```sql
select userID, age, weight, gender
from `deakincap.BikeData.T1copy`
group by userID, age,weight, gender
order by userID
```

Query results

Row	userID	age	weight	gender
1	U1000000	33	80	MALE

Here we only have 1 user. In the future this table would show all users by userID and their personal data

4. Create a table 'dailyCalories' that would show total calories by day and respective userID it belongs to:

```sql
create table deakincap.BikeData.dailyCalories AS
select day,
round(sum(totalCalories),2) AS totalCalories,
userID
from `BikeData.userPerformanceTable`
group by day, userID
having totalCalories IS NOT NULL
order by day
```

RESULT:

```
select * from `BikeData.dailyCalories`
order by day
```

Query results

JOB INFORMATION | RESULTS | CHART PREVIEW | JSON | EXECUTION DETAILS

Row	day	totalCalories	userID
1	2021-03-10 00:00:00 UTC	1432.0	U1000000
2	2021-03-14 00:00:00 UTC	1677.0	U1000000
3	2021-03-16 00:00:00 UTC	993.0	U1000000
4	2021-03-19 00:00:00 UTC	1066.0	U1000000
5	2021-03-26 00:00:00 UTC	1567.0	U1000000
6	2021-03-31 00:00:00 UTC	1181.0	U1000000
7	2021-04-01 00:00:00 UTC	1342.0	U1000000
8	2021-04-03 00:00:00 UTC	1347.0	U1000000
9	2021-04-04 00:00:00 UTC	574.0	U1000000
10	2021-04-06 00:00:00 UTC	931.0	U1000000
11	2021-04-12 00:00:00 UTC	630.0	U1000000
12	2021-04-16 00:00:00 UTC	571.0	U1000000
13	2021-04-17 00:00:00 UTC	1934.0	U1000000
14	2021-04-24 00:00:00 UTC	1892.0	U1000000

Results per pa

5. Now, having those 3 tables : userPerformanceTable, userInfo and dailyCalories

- deakincap
  - External connections
  - BikeData
    - T1
    - T1copy
    - T1copy2
    - T1copy3
    - calorieByDay
    - dailyCalories
    - perf2
    - performanceTable
    - sessionData
    - testtable1
    - userInfo
    - userPerformanceTable

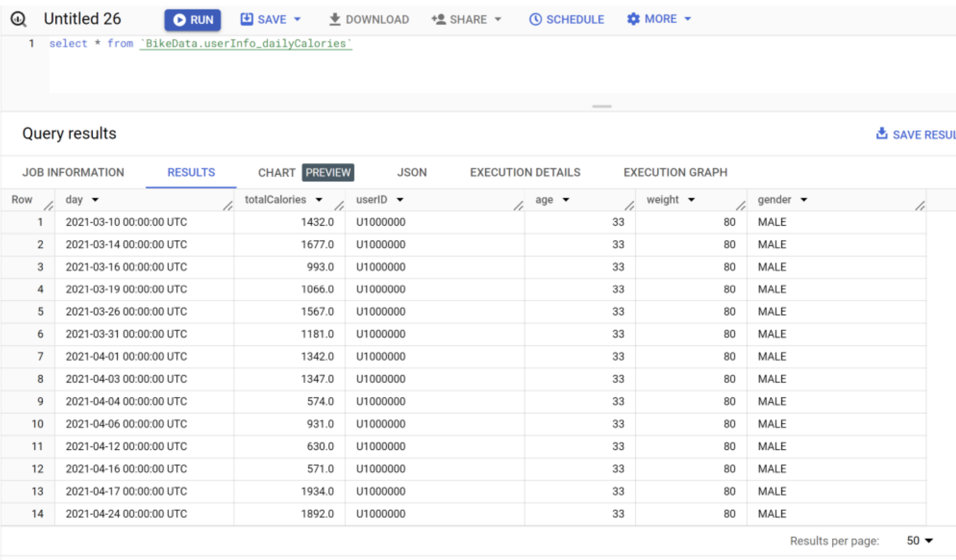
We can perform joins so that we retrieve the data needed for each particular purpose, such as:

a) Calories by day for a relevant user, featuring personal data

dailyCalories + userInfo = userInfo_dailyCalories

```sql
create table deakincap.BikeData.userInfo_dailyCalories as
select t1.day, t1.totalCalories, t2.userID, t2.age, t2.weight, t2.gender
from `BikeData.dailyCalories` as t1
left outer join
`BikeData.userInfo` as t2
on t1.userID = t2.userID
order by day
```

RESULT:



Row	day	totalCalories	userID	age	weight	gender
1	2021-03-10 00:00:00 UTC	1432.0	U1000000	33	80	MALE
2	2021-03-14 00:00:00 UTC	1677.0	U1000000	33	80	MALE
3	2021-03-16 00:00:00 UTC	993.0	U1000000	33	80	MALE
4	2021-03-19 00:00:00 UTC	1066.0	U1000000	33	80	MALE
5	2021-03-26 00:00:00 UTC	1567.0	U1000000	33	80	MALE
6	2021-03-31 00:00:00 UTC	1181.0	U1000000	33	80	MALE
7	2021-04-01 00:00:00 UTC	1342.0	U1000000	33	80	MALE
8	2021-04-03 00:00:00 UTC	1347.0	U1000000	33	80	MALE
9	2021-04-04 00:00:00 UTC	574.0	U1000000	33	80	MALE
10	2021-04-06 00:00:00 UTC	931.0	U1000000	33	80	MALE
11	2021-04-12 00:00:00 UTC	630.0	U1000000	33	80	MALE
12	2021-04-16 00:00:00 UTC	571.0	U1000000	33	80	MALE
13	2021-04-17 00:00:00 UTC	1934.0	U1000000	33	80	MALE
14	2021-04-24 00:00:00 UTC	1892.0	U1000000	33	80	MALE

b) Showing performance by day, by the hour for each user, featuring their personal info

userPerformance + userInfo = userPerformance_userInfo

```sql
create table deakincap.BikeData.userPerformance_userInfo AS
select t1.* , t2.age, t2.weight, t2.gender
from `BikeData.userPerformanceTable` as t1
left outer join
`BikeData.userInfo` as t2
on t1.userID = t2.userID
order by 1
```

RESULT:

Untitled 27 RUN SAVE DOWNLOAD SHARE SCHEDULE MORE

```
select * from `BikeData.userPerformance_userInfo`
```

Press Alt+F1 for Ac

Query results

SAVE RESULTS EXPLORE DA

JOB INFORMATION RESULTS CHART PREVIEW JSON EXECUTION DETAILS EXECUTION GRAPH

Row	day	hour	totalDistance	averageSpee	averageAltitude	averageHeartRate	sessionDuratio	totalCalories	userID	session_ID	age	weight	gender
1	2021-03-10 00:00:00 U...	2021-03-10 05:00:00 UTC	0.95	25.38	50.21	85.37	1.52	31.0	U1000000	2	33	80	MALE
2	2021-03-10 00:00:00 U...	2021-03-10 06:00:00 UTC	24.6	27.16	34.15	124.01	37.87	575.0	U1000000	2	33	80	MALE
3	2021-03-10 00:00:00 U...	2021-03-10 07:00:00 UTC	30.48	31.78	23.97	89.37	40.77	661.0	U1000000	2	33	80	MALE
4	2021-03-10 00:00:00 U...	2021-03-10 08:00:00 UTC	2.15	22.59	21.25	91.6	4.4	48.0	U1000000	2	33	80	MALE
5	2021-03-10 00:00:00 U...	2021-03-10 08:00:00 UTC	4.71	24.87	17.23	83.6	9.63	117.0	U1000000	3	33	80	MALE
6	2021-03-14 00:00:00 U...	2021-03-14 06:00:00 UTC	2.41	31.75	59.0	115.4	3.5	49.0	U1000000	9	33	80	MALE
7	2021-03-14 00:00:00 U...	2021-03-14 07:00:00 UTC	25.97	30.74	16.83	106.38	35.12	496.0	U1000000	10	33	80	MALE
8	2021-03-14 00:00:00 U...	2021-03-14 08:00:00 UTC	6.92	31.06	61.9	139.97	8.98	210.0	U1000000	10	33	80	MALE
9	2021-03-14 00:00:00 U...	2021-03-14 08:00:00 UTC	21.1	32.34	47.38	117.98	25.82	489.0	U1000000	11	33	80	MALE
10	2021-03-14 00:00:00 U...	2021-03-14 09:00:00 UTC	5.1	32.94	47.15	119.03	6.73	121.0	U1000000	11	33	80	MALE
11	2021-03-14 00:00:00 U...	2021-03-14 09:00:00 UTC	7.06	30.13	32.66	122.24	10.05	157.0	U1000000	12	33	80	MALE
12	2021-03-14 00:00:00 U...	2021-03-14 10:00:00 UTC	5.29	26.47	47.76	111.11	8.78	155.0	U1000000	13	33	80	MALE

Results per page: 50 1 – 50 of 738

Additional: add a separate date and time columns as R need to work with those values only, not as a whole timestamp; also add a ‘string’ column called ‘timeOfDay’ with 4 acceptable values : ‘morning’, ‘afternoon’, ‘evening’ and ‘night’.

```
ALTER TABLE deakincap.BikeData.userPerformanceTable
ADD COLUMN IF NOT EXISTS dayDateCol DATE;
```

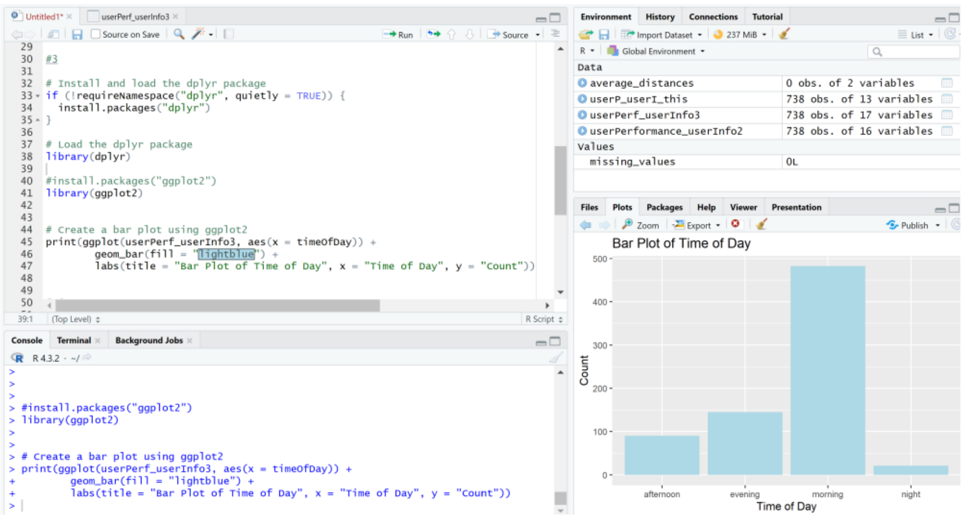
```
UPDATE deakincap.BikeData.userPerformanceTable
SET dayDateCol = DATE(FORMAT_TIMESTAMP('%Y-%m-%d', day))
where 1=1;
```

```
#1
ALTER TABLE `deakincap.BikeData.userPerformanceTable`
ADD COLUMN IF NOT EXISTS timeCol TIME;


#2
UPDATE `deakincap.BikeData.userPerformanceTable`
SET timeCol = TIME(hour)
where 1=1
```

```
create table `deakincap.BikeData.userPerformance_userinfo2` AS
select t1.*, t2.age, t2,weight, t2.gender
from `BikeData.userPerformanceTable` as t1
left outer join
`BikeData.userInfo` as t2
on t1.userID = t2.userID
order by 1,2
```

```
#1
ALTER TABLE deakincap.BikeData.userPerformance_userinfo2
ADD COLUMN IF NOT EXISTS timeOfDay STRING;

#2
UPDATE deakincap.BikeData.userPerformance_userinfo2
SET timeOfDay =
  CASE
    WHEN EXTRACT(HOUR FROM hour) BETWEEN 6 AND 11 THEN 'morning'
    WHEN EXTRACT(HOUR FROM hour) BETWEEN 12 AND 15 THEN 'afternoon'
    WHEN EXTRACT(HOUR FROM hour) BETWEEN 16 AND 21 THEN 'evening'
    ELSE 'night'
  END
  where 1=1;
```

Result in R:



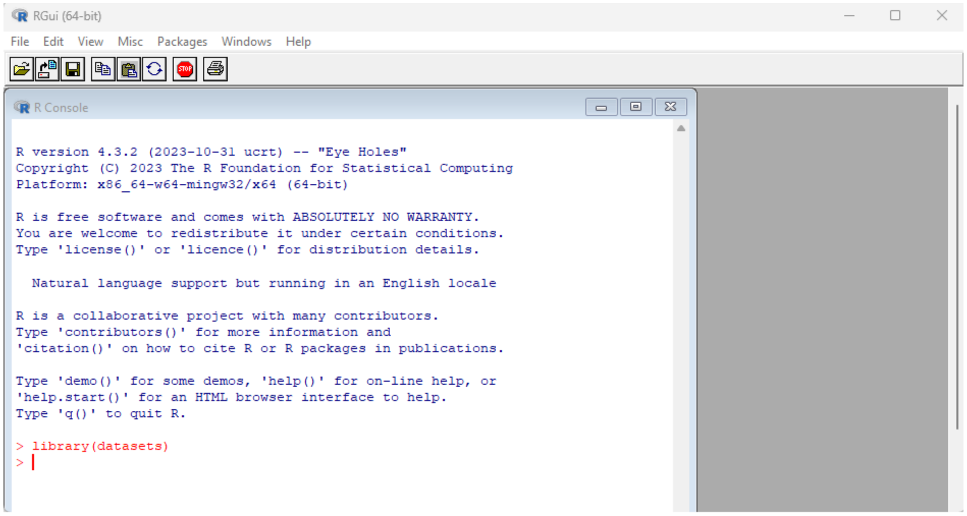
Then export the 'userPerformance_userInfo2' table

6. At this point we can export both joined tables (I exported to Google Sheets, downloaded onto my local computer, changes to .xls format) from point 4 , import them to RStudio and start working on Data Clustering.

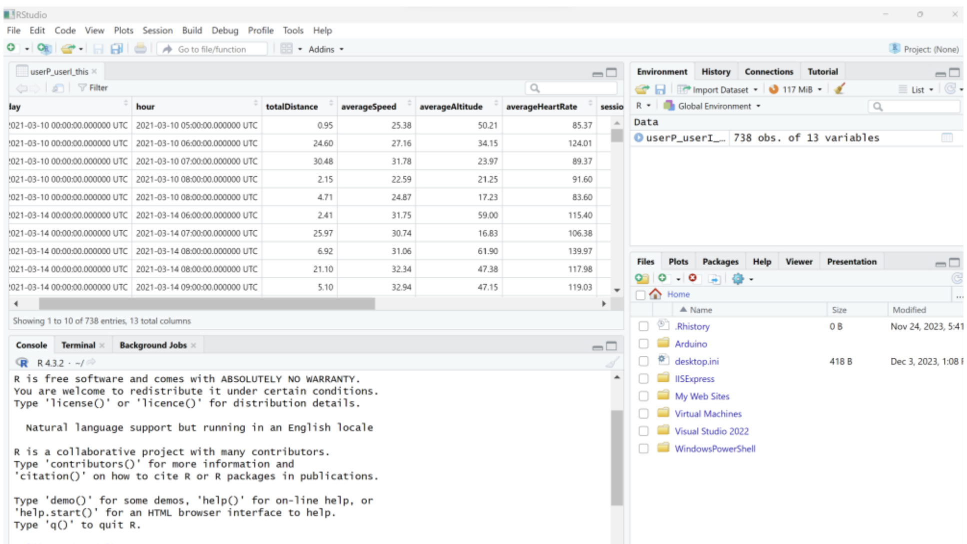
Tuesday (28.11) - onwards

RStudio (K-means cluster analysis)

1. In order to import dataset to RStudio it needs to be in .xls format
2. Download R:



And R studio:



3. Correlation analysis

- There is a weak positive correlation between **totalDistance** and **sessionDuration**.
- There is a weak negative correlation between **totalDistance** and **averageHeartRate**.
- There is a very weak negative correlation between **sessionDuration** and **averageHeartRate**.

```
    # correlation analysis

str(userPerf_userInfo3[, c("totalDistance", "sessionDuration", "averageHeartRate")])

userPerf_userInfo3$totalDistance <- as.numeric(userPerf_userInfo3$totalDistance)
userPerf_userInfo3$sessionDuration <- as.numeric(userPerf_userInfo3$sessionDuration)
userPerf_userInfo3$averageHeartRate <- as.numeric(userPerf_userInfo3$averageHeartRate)

cor(userPerf_userInfo3[, c("totalDistance", "sessionDuration", "averageHeartRate")])
```

222:1 (Top Level)

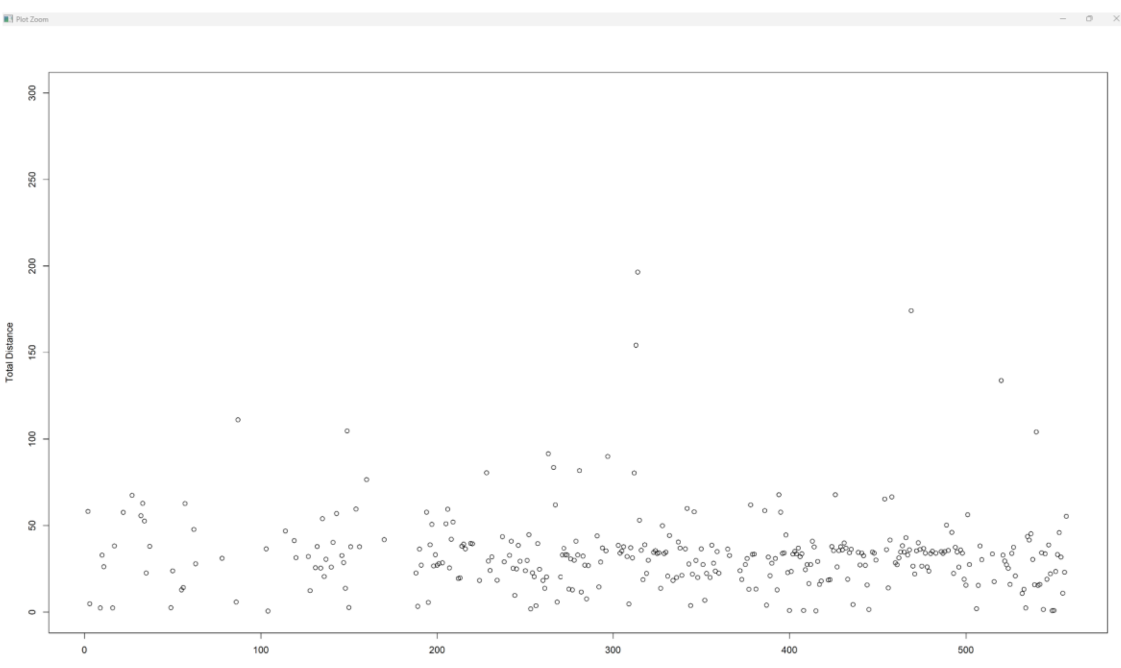
Console Terminal Background Jobs

R 4.3.2 · ~/

```
 $ averageHeartRate: chr [1:738] "85.37" "160.35" "115.0" "103.27" ...
> userPerf_userInfo3$totalDistance <- as.numeric(userPerf_userInfo3$totalDistance)
> userPerf_userInfo3$sessionDuration <- as.numeric(userPerf_userInfo3$sessionDuration)
> userPerf_userInfo3$averageHeartRate <- as.numeric(userPerf_userInfo3$averageHeartRate)
> cor(userPerf_userInfo3[, c("totalDistance", "sessionDuration", "averageHeartRate")])
                 totalDistance sessionDuration averageHeartRate
totalDistance        1.0000000      0.18779799      -0.11144762
sessionDuration      0.1877980      1.00000000      -0.02299837
averageHeartRate    -0.1114476     -0.02299837       1.00000000
>
```

4. Time series analysis : session_ID against total distance for that session. The session_ID were assigned in an ascending order to reflect time accurately

```
  # time series analysis

session_ID_totDistance$f0_ <- as.numeric(session_ID_totDistance$f0_)

session_ID_totDistance$session_ID <- as.numeric(session_ID_totDistance$session_ID)

plot(session_ID_totDistance$session_ID, session_ID_totDistance$f0_, type = "p",
     xlab = "Session ID", ylab = "Total Distance", ylim = c(0, 300))
```



We can clearly see that most of the sessions were below 50 km. A session was created as a continuous performance with less than 5 min of idle time.

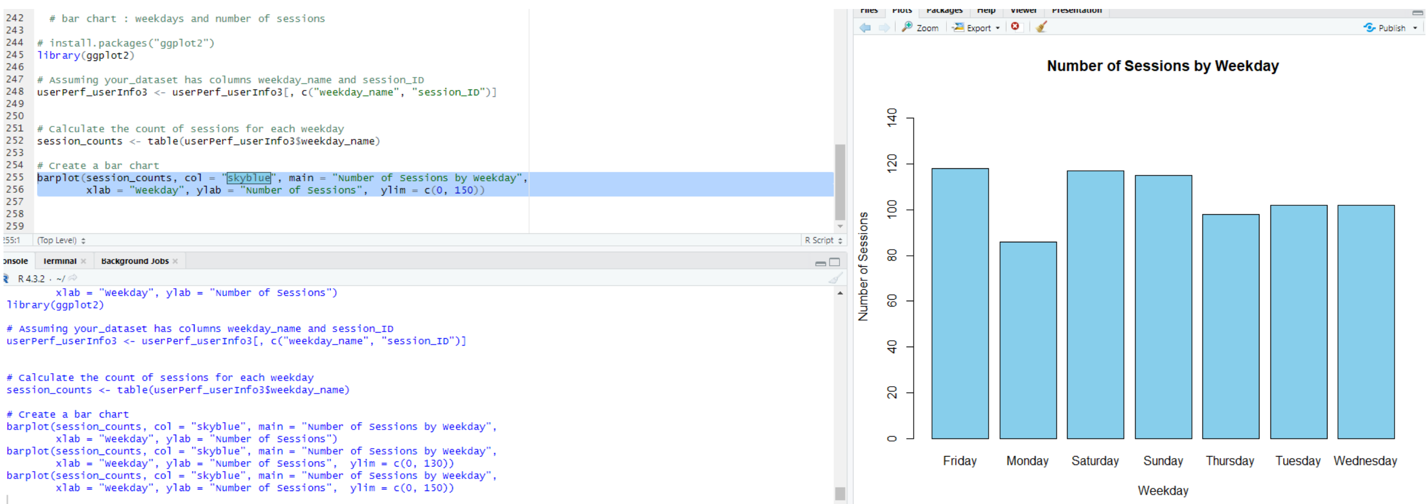
5. Analysis for particular weekdays

Add weekday in Big Query:

```sql

ALTER TABLE deakincap.BikeData.userPerformance_userinfo2
ADD COLUMN weekdayName STRING GENERATED ALWAYS AS (FORMAT_DATE('%A', DATE(day))) STORED;
```

Or weekday column in R:

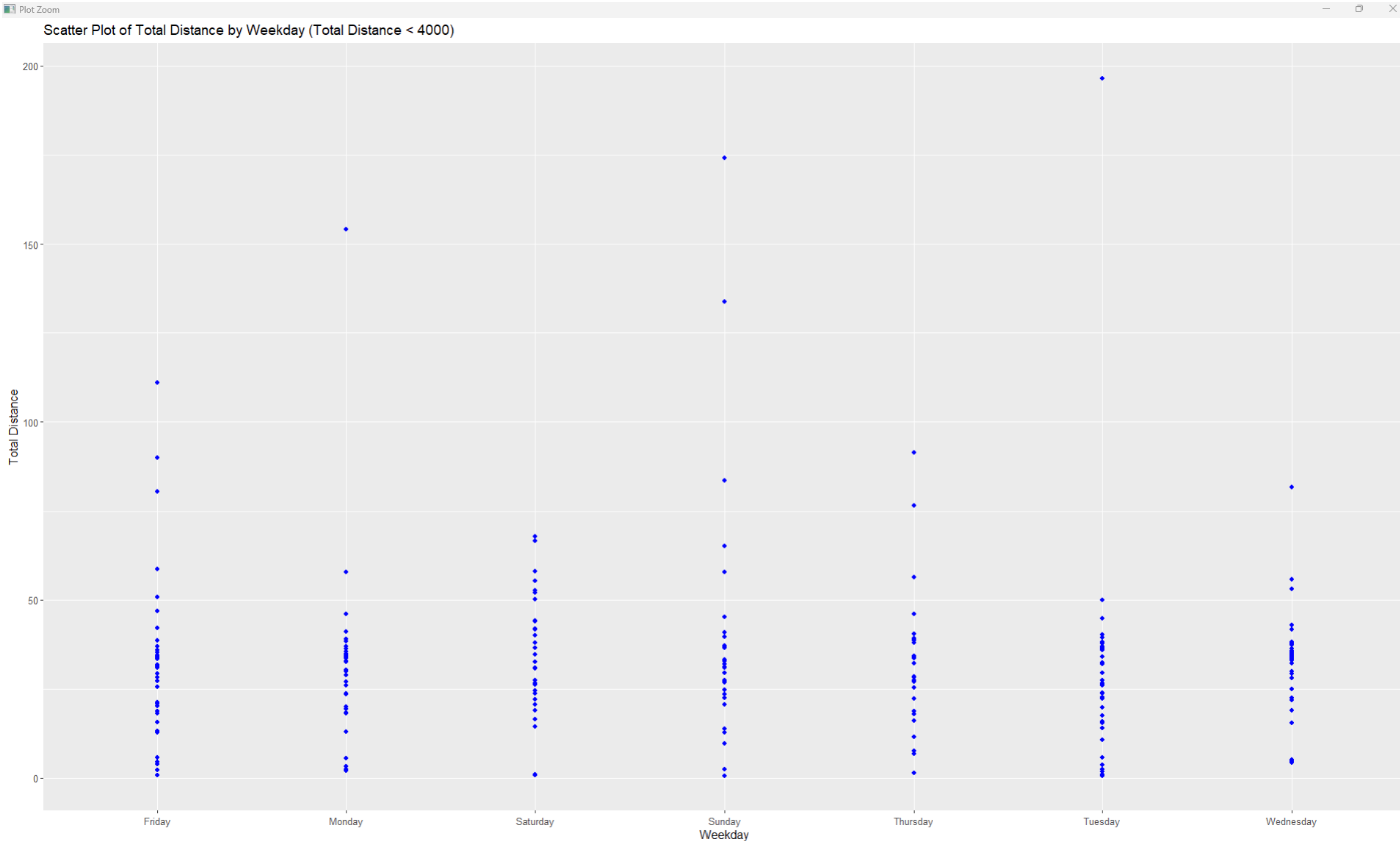
```r
 # add weekday name
install.packages("lubridate")
library(lubridate)

userPerf_userInfo3$weekday_name <- weekdays(userPerf_userInfo3$day)
```



6. Take totalDistance by every session (group by in SQL) and create a scatter plot against each weekday in R. Remove some outliers that slipped out my Data Wrangling in GCS by filtering out totalDistance < 44000:

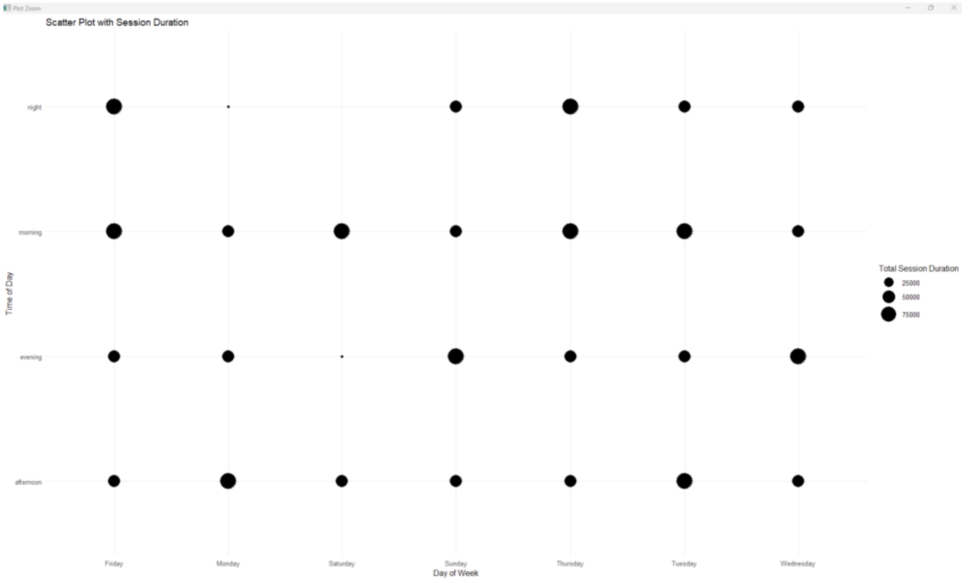
	day	hour	totalDistance
97	2022-06-22 00:00:00.000000 UTC	2022-06-22 12:00:00.000000 UTC	44971.00
98	2021-10-22 00:00:00.000000 UTC	2021-10-22 14:00:00.000000 UTC	44970.00
99	2022-01-16 00:00:00.000000 UTC	2022-01-16 17:00:00.000000 UTC	44965.00
100	2021-10-24 00:00:00.000000 UTC	2021-10-24 08:00:00.000000 UTC	44965.00
101	2022-06-22 00:00:00.000000 UTC	2022-06-22 12:00:00.000000 UTC	44963.00
102	2022-01-06 00:00:00.000000 UTC	2022-01-06 19:00:00.000000 UTC	44962.00
103	2021-06-27 00:00:00.000000 UTC	2021-06-27 14:00:00.000000 UTC	44962.00
104	2021-09-08 00:00:00.000000 UTC	2021-09-08 09:00:00.000000 UTC	44961.00
105	2021-04-06 00:00:00.000000 UTC	2021-04-06 05:00:00.000000 UTC	44959.00
106	2022-06-09 00:00:00.000000 UTC	2022-06-09 12:00:00.000000 UTC	44957.00
107	2021-10-24 00:00:00.000000 UTC	2021-10-24 05:00:00.000000 UTC	44955.00
108	2022-11-09 00:00:00.000000 UTC	2022-11-09 17:00:00.000000 UTC	44952.00
109	2022-10-29 00:00:00.000000 UTC	2022-10-29 08:00:00.000000 UTC	44942.00
110	2022-06-04 00:00:00.000000 UTC	2022-06-04 10:00:00.000000 UTC	44935.00
111	2021-05-27 00:00:00.000000 UTC	2021-05-27 18:00:00.000000 UTC	44933.00
112	2022-05-01 00:00:00.000000 UTC	2022-05-01 15:00:00.000000 UTC	44931.00
113	2021-07-10 00:00:00.000000 UTC	2021-07-10 09:00:00.000000 UTC	44930.00
114	2021-10-22 00:00:00.000000 UTC	2021-10-22 13:00:00.000000 UTC	44927.00
115	2021-12-27 00:00:00.000000 UTC	2021-12-27 10:00:00.000000 UTC	120.95
116	2022-07-24 00:00:00.000000 UTC	2022-07-24 09:00:00.000000 UTC	101.61
117	2021-12-28 00:00:00.000000 UTC	2021-12-28 10:00:00.000000 UTC	100.96
118	2021-11-24 00:00:00.000000 UTC	2021-11-24 17:00:00.000000 UTC	66.28
119	2021-05-07 00:00:00.000000 UTC	2021-05-07 16:00:00.000000 UTC	65.11
120	2021-10-28 00:00:00.000000 UTC	2021-10-28 16:00:00.000000 UTC	51.27
121	2021-04-03 00:00:00.000000 UTC	2021-04-03 13:00:00.000000 UTC	36.47
122	2022-11-19 00:00:00.000000 UTC	2022-11-19 07:00:00.000000 UTC	34.16

```r
# Load the dplyr package
library(dplyr)

# Convert totalDistance to numeric (if not done already)
userPerf_userInfo3$totalDistance <- as.numeric(userPerf_userInfo3$totalDistance)

# Aggregate data to calculate the sum of totalDistance for each session_ID and weekday_name
summed_data <- userPerf_userInfo3 %>%
  group_by(weekday_name, session_ID) %>%
  summarise(totalDistance = sum(totalDistance, na.rm = TRUE))

# Filter data where totalDistance is less than 4000
filtered_data <- summed_data[summed_data$totalDistance < 44000, ]

# Create a scatter plot
ggplot(filtered_data, aes(x = weekday_name, y = totalDistance)) +
  geom_point(color = "blue") +
  labs(title = "Scatter Plot of Total Distance by Weekday (Total Distance < 4000)",
       x = "Weekday",
       y = "Total Distance")
```



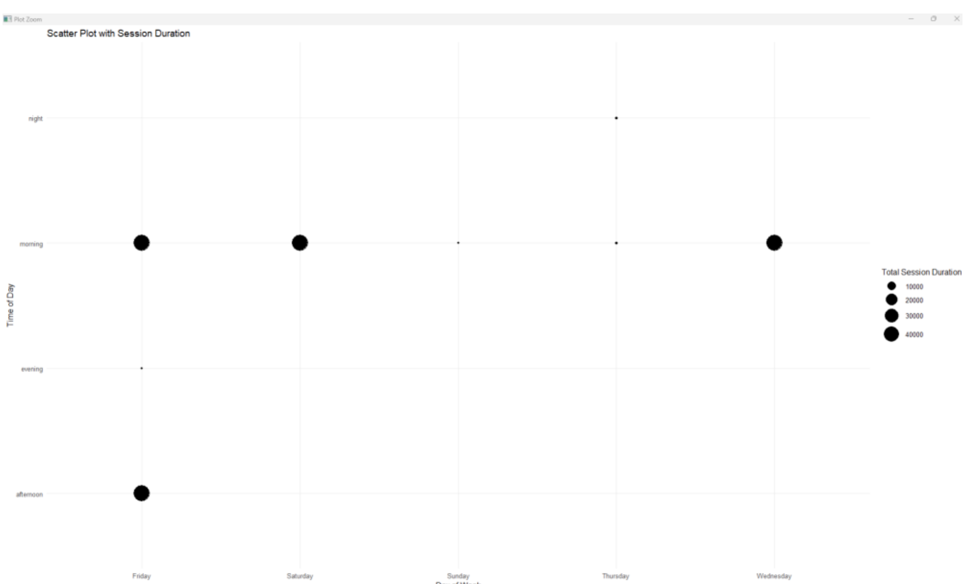
7. Categorize records by day of the week, and time of day. Show sessionDuration as dots and vary their size depending on the length of the session. This one is for the entire dataset. Later could be narrowed down to a particular week, and displayed as a 'weekly summary'

```
  # a rundown of session Durations for each day and time of day (for the whole dataset). It can be later narrowed down to
  # a single week, like when showing a summary for last week etc

library(ggplot2)
library(dplyr)

# Assuming 'newest_joined' is your data frame
df <- newest_joined %>%
  mutate(sessionDuration = as.numeric(sessionDuration),
         timeOfDay = timeOfDay) %>%
  group_by(session_ID) %>%
  summarise(total_sessDur = sum(sessionDuration, na.rm = TRUE)) %>%
  left_join(newest_joined, by = "session_ID")

# Now, you can use ggplot to create your scatter plot
ggplot(df, aes(x = weekdayName, y = timeOfDay, size = total_sessDur)) +
  geom_point() +
  scale_size_continuous(range = c(1, 10)) +  # Adjust the range for desired size scaling
  labs(title = "Scatter Plot with Session Duration",
       x = "Day of Week",
       y = "Time of Day",
       size = "Total Session Duration") +
  theme_minimal()
```



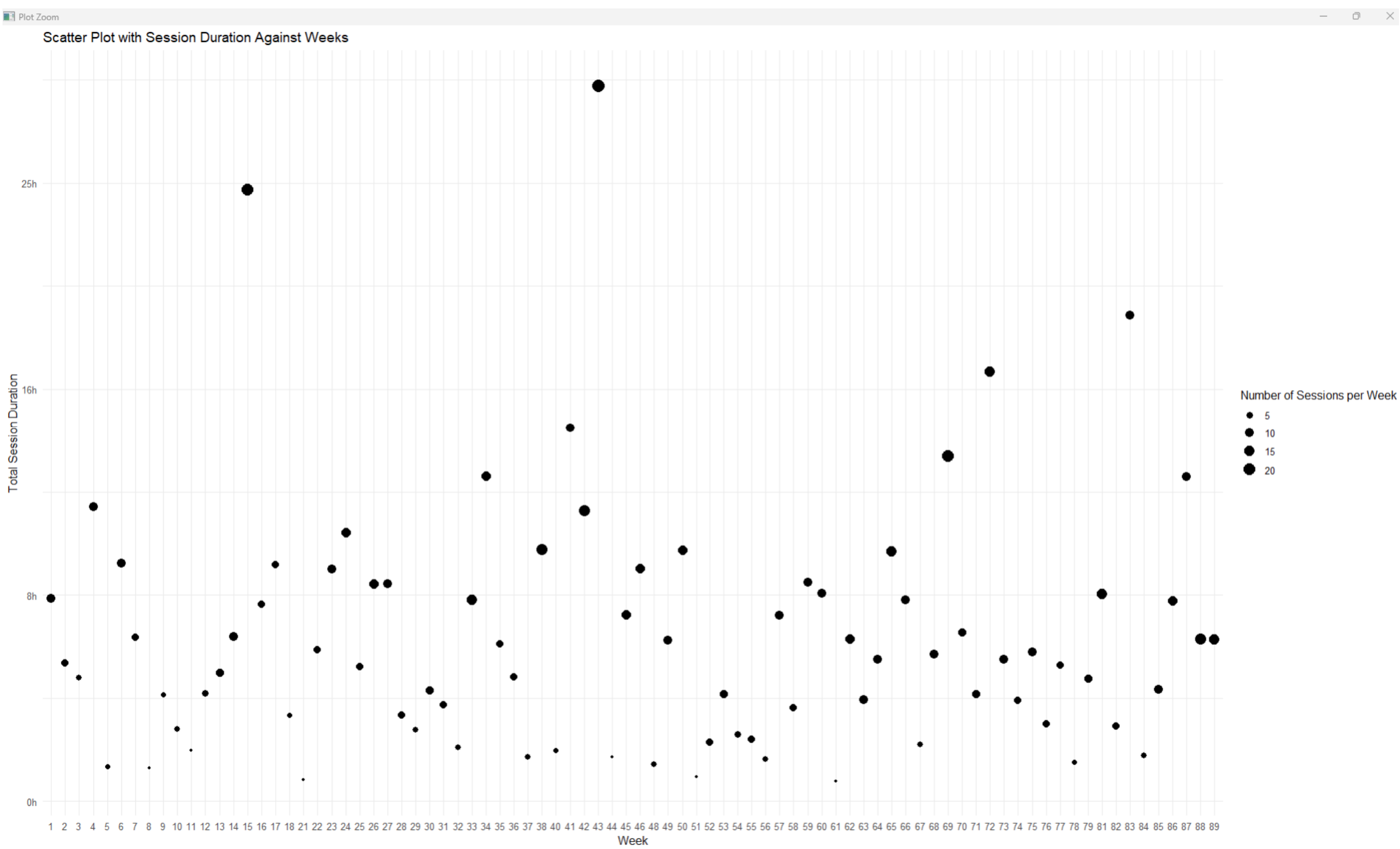
… then see the most recent week:

```
  # for most recent  week only

library(ggplot2)
library(dplyr)

# Assuming 'newest_joined' is your data frame
df <- newest_joined %>%
  mutate(sessionDuration = as.numeric(sessionDuration),
         timeOfDay = timeOfDay,
         dayDateCol = as.Date(dayDateCol)) %>%
  filter(dayDateCol >= max(dayDateCol) - 6) %>%  # Filter for the most recent week
  group_by(session_ID) %>%
  summarise(total_sessDur = sum(sessionDuration, na.rm = TRUE)) %>%
  left_join(newest_joined, by = "session_ID")

# Now, you can use ggplot to create your scatter plot
ggplot(df, aes(x = weekdayName, y = timeOfDay, size = total_sessDur)) +
  geom_point() +
  scale_size_continuous(range = c(1, 10)) +  # Adjust the range for desired size scaling
  labs(title = "Scatter Plot with Session Duration",
       x = "Day of Week",
       y = "Time of Day",
       size = "Total Session Duration") +
  theme_minimal()
```



8. All days grouped into weeks against the total session duration. The size of the dot represents the number of sessions that week

```r
# all days grouped into weeks against the total session duration. The size of the dot represents the number
# of sessions that week


library(ggplot2)
library(dplyr)

#  'newest_joined' is my dataset
df <- newest_joined %>%
  mutate(sessionDuration = as.numeric(sessionDuration),
         dayDateCol = as.Date(dayDateCol)) %>%
  filter(sessionDuration < 40000) %>%  # Filter sessionDuration as there were some outliers
  group_by(session_ID) %>%  # so that each session in that week contributes to the size of the dot on the graph
  summarise(total_sessDur = sum(sessionDuration, na.rm = TRUE),
            sessions_per_week = n_distinct(format(dayDateCol, "%V"))) %>%
  left_join(newest_joined, by = "session_ID")

# group by 7-day intervals and calculate total session duration and number of sessions per week
df_weekly <- df %>%
  mutate(week_interval = cut(dayDateCol, breaks = "1 week", labels = FALSE)) %>%
  group_by(week_interval) %>%
  summarise(total_sessDur_weekly = sum(total_sessDur, na.rm = TRUE),
            total_sessions_per_week = sum(sessions_per_week, na.rm = TRUE))

# use ggplot to create your scatter plot with session duration against weeks
ggplot(df_weekly, aes(x = as.factor(week_interval), y = total_sessDur_weekly, size = total_sessions_per_week)) +
  geom_point() +
  scale_size_continuous(range = c(1, 5)) +  # Adjust the range for desired size scaling
  labs(title = "Scatter Plot with Session Duration Against Weeks",
       x = "Week",
       y = "Total Session Duration",
       size = "Number of Sessions per Week") +
  theme_minimal() +
  scale_y_continuous(labels = function(x) paste0(floor(x / 60), "h"))
```



9. What is K-Means Clustering?

- One of the most popular unsupervised learning technique
- Used to group together observations

- Using a fixed number of clusters (centroid), group together observations based on similarities
- Uses Euclidean distance

a) The dataset for clustering Is the `` `BikeData.userPerformance_userInfo2_new` `` that will be referenced in github at the end of the document. We take this dataset and group it according to sessions, as we would like insight into each individual session stats.

So in Big Query we will perform the grouping:

```
select sum(totalDistance) AS totalDistance,
  avg(averageSpeed) AS averageSpeed,
  avg(averageAltitude) AS averageAltitude,
  avg(averageHeartRate) AS averageHeartRate,
  sum(sessionDuration) AS sessionDuration,
  sum(totalCalories) AS totalCalories,
  userID,
  session_ID,
  dayDateCol,
  timeCol,
  age,weight, gender, timeOfDay, weekdayName

 from `BikeData.userPerformance_userInfo2_new`
  group by session_ID, userID,
  dayDateCol,
  timeCol,
  age,weight, gender, timeOfDay, weekdayName
  order by    session_ID
```

Save the dataset as .xls and import to R Studio.

b) Apart from data, import the libraries needed. Do some more data checks (I had to filter out some outliers).

```r
# install : https://cran.rstudio.com/bin/windows/Rtools/rtools43/rtools.html
# then :
install.packages("stats")
install.packages("dplyr")
install.packages("ggplot2")
install.packages("ggfortify")

library(stats)
library(dplyr)
library(ggplot2)
library(ggfortify)

View(newest_joined)

# store all numeric (not char, not string in a separate data object called my_data)


                  # OPTION 1 ---------------------------------------------------------------------------------
                 # my_data grouped by session_id

                  # define wssplot function, where we grouped dataset by session_ID
                  wssplot <- function(data, nc = 15, seed = 1234) {
                    wss <- (nrow(data) - 1) * sum(apply(data[, -1], 2, var))
                    for (i in 2:nc) {
                      set.seed(seed)
                      wss[i] <- sum(kmeans(data[, -1], centers = i)$withinss)
                    }
                    plot(1:nc, wss, type = "b", xlab = "Number of clusters", ylab = "Within groups sum of squares")
                  }

                  # prepare  data
                  my_data <- newest_joined %>%
                    select(1, 2, 4, 5, 6, 8) %>%
                    filter(totalDistance < 40000 & sessionDuration < 40000) %>%
                    mutate_all(as.numeric)

                  # Group by session_ID and calculate summary statistics
                  grouped_data <- my_data %>%
                    group_by(session_ID) %>%
                    summarise(
                      totDistance = sum(totalDistance),
                      avgSpeed = mean(averageSpeed),
                      avgHeartRate = mean(averageHeartRate),
                      sessDur = sum(sessionDuration),
                      totCalories = sum(totalCalories)
                    )

                  #  wssplot to choose the optimum number of clusters
                  wssplot(grouped_data)

                  # exclude session_ID for kmeans clustering
                  KM <- kmeans(grouped_data[, -1], 6) # exclude session_ID here as we needed it for grouping only

                  # visualize the results
                  autoplot(KM, data = grouped_data, frame = TRUE)

                  # view the cluster centers
                  KM$centers
```

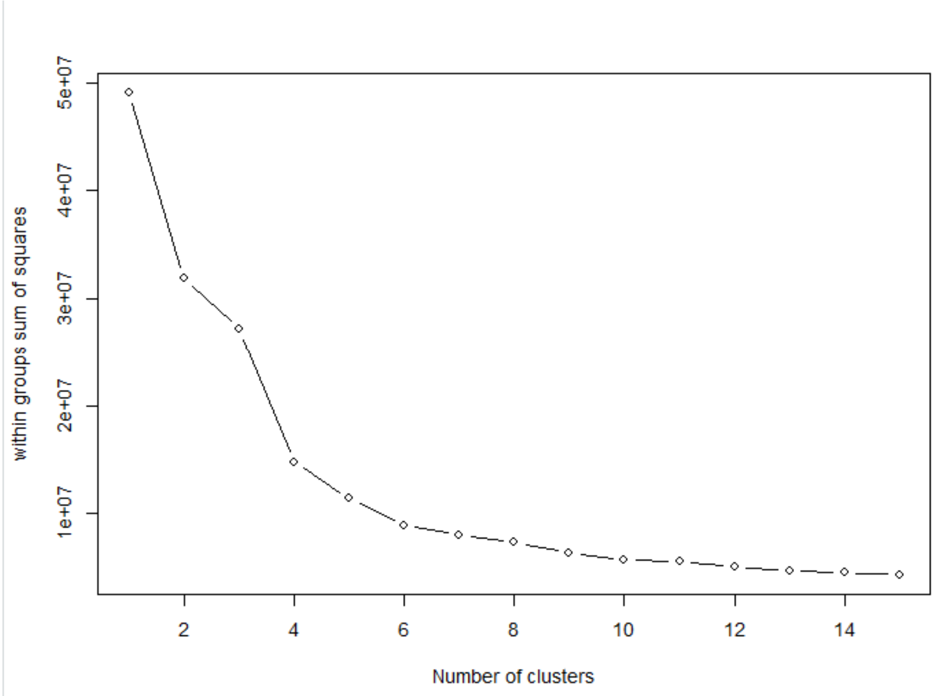
```
>                   # view the cluster centers
>                   KM$centers
  totDistance avgSpeed avgHeartRate     sessDur totCalories
1    73.01105 29.64618     149.3857   88.022105   1787.4211
2    18.70049 30.35226     155.7045   22.849146    438.3780
3   162.97000 29.14604     155.2922  120.077500   3901.0000
4    28.90036 30.01611     159.6645   36.186396    693.4955
5    41.91439 29.01629     155.0658   53.974091   1029.6212
6     5.88500 26.01833     140.4521    8.874583    138.5833
```

c) Analyse the plot and cluster centers:

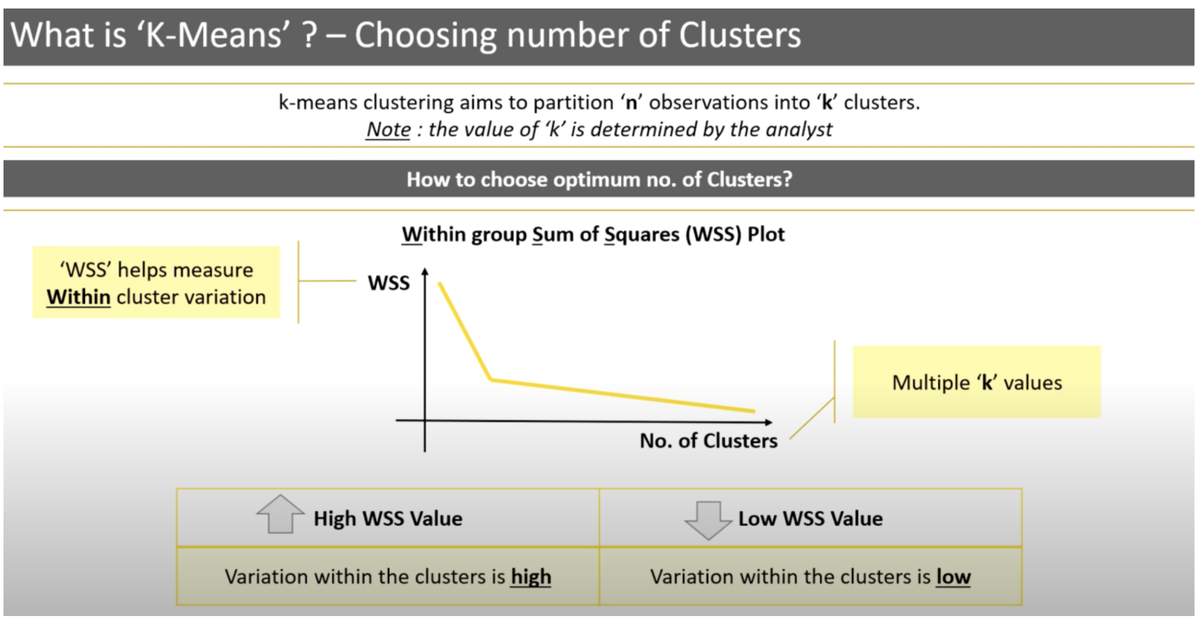
The centers of those clusters are not overlapping. The averageSpeed and averageHeartRate are indicative of a distinct person's physical ability, meaning that it is probably one and the same person here, which is true as we only have 1 distinct user in this dataset.

Future reference: we could include a bigger population of users and include their age, weight (numeric vars) to perform cluster analysis that would show performance of different age groups/ weight groups.

The optimum number of clusters (line 662) was chosen as 6

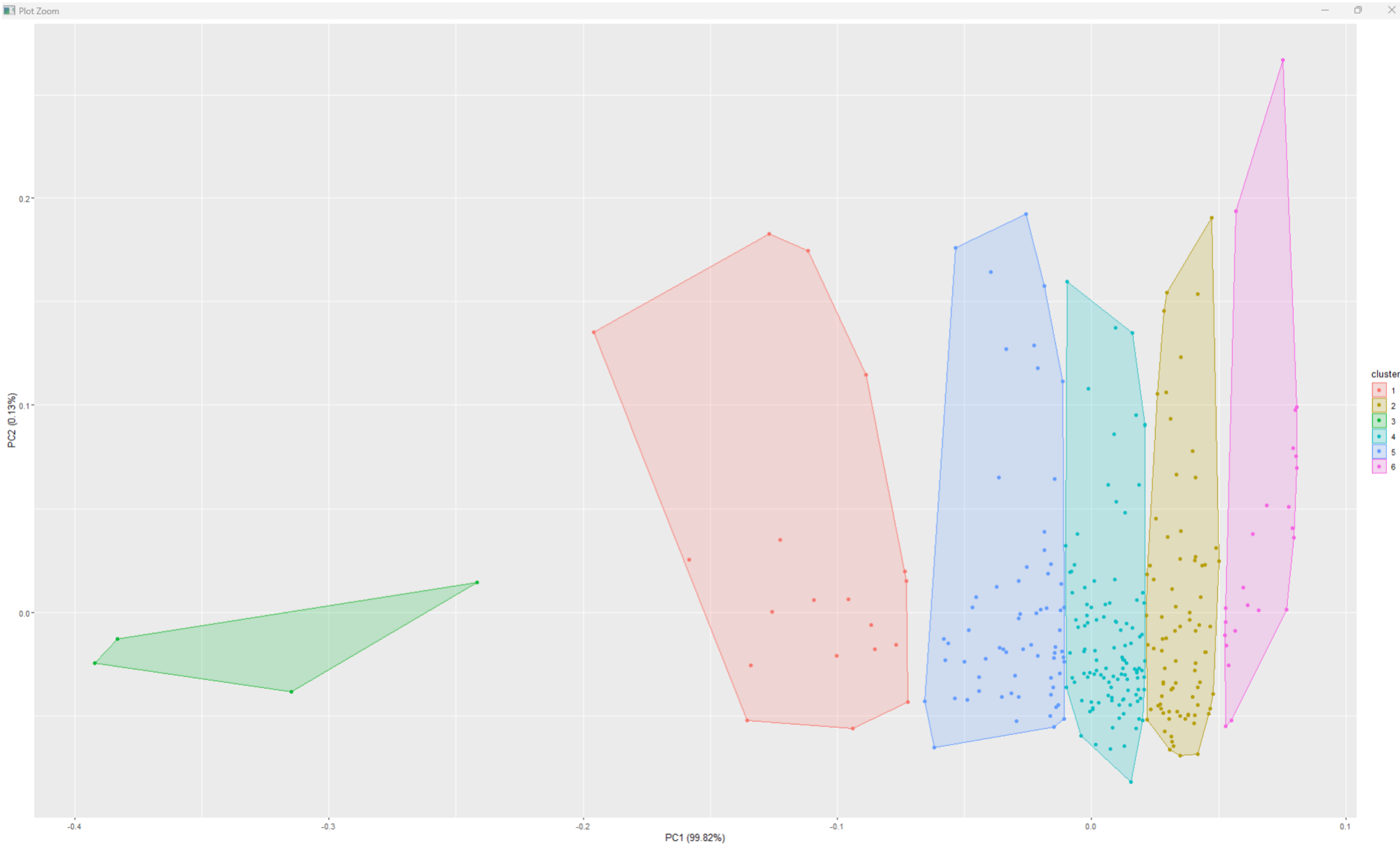


Rule:

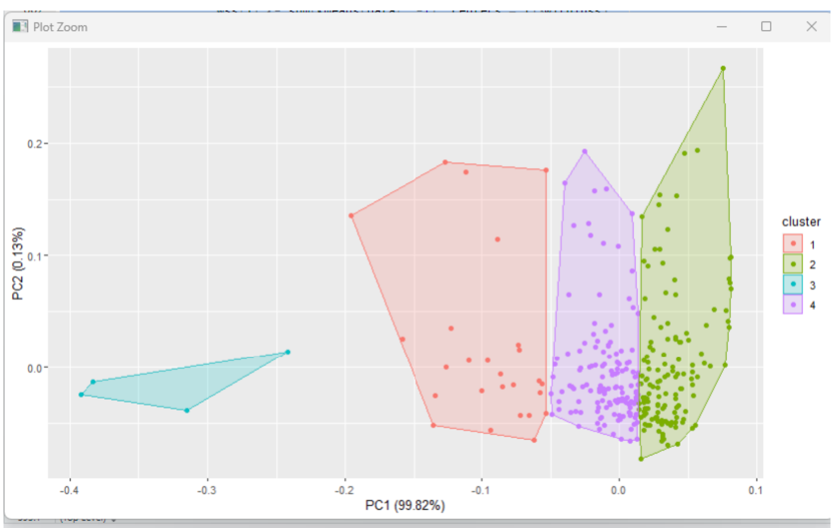


Source: https://www.youtube.com/watch?v=DWLoY6I6d34

Result:



d) When optimum is chosen as 4 we get more ‘tight’ groupings, where we can infer 4 types of performance at average same sppeeds: short ~ 20 min, medium ~ 40 min, long ~ 1h 20 min and very long ~ 2 h:




```
Console   Terminal   Background Jobs
R 4.3.2 · ~/
>
>                  # view the cluster centers
>                  KM$centers
  totDistance avgSpeed avgHeartRate   sessDur totCalories
1    67.99923 29.59619     151.9716  83.08577   1659.0000
2    17.90942 29.42620     153.9934  22.43328    420.8686
3   162.97000 29.14604     155.2922 120.07750   3901.0000
4    34.63748 29.65881     157.4510  43.91676    844.4173
>
```

7. TABLEAU

- Work was done on the following dataset:

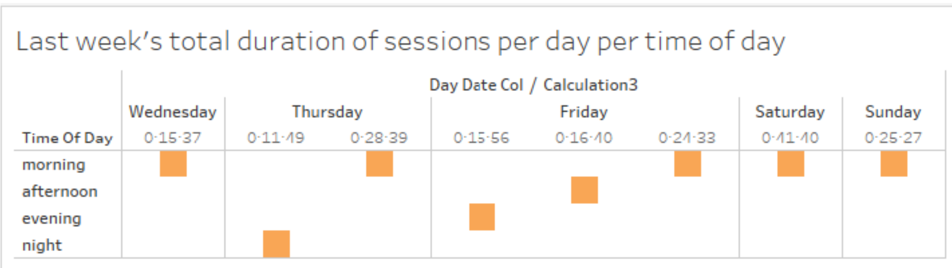
https://github.com/alexbaar/Historical-Dataset-Analysis/blob/main/Data/data_for_Tableau_dashboard.csv

- The Tableau file can be found here:

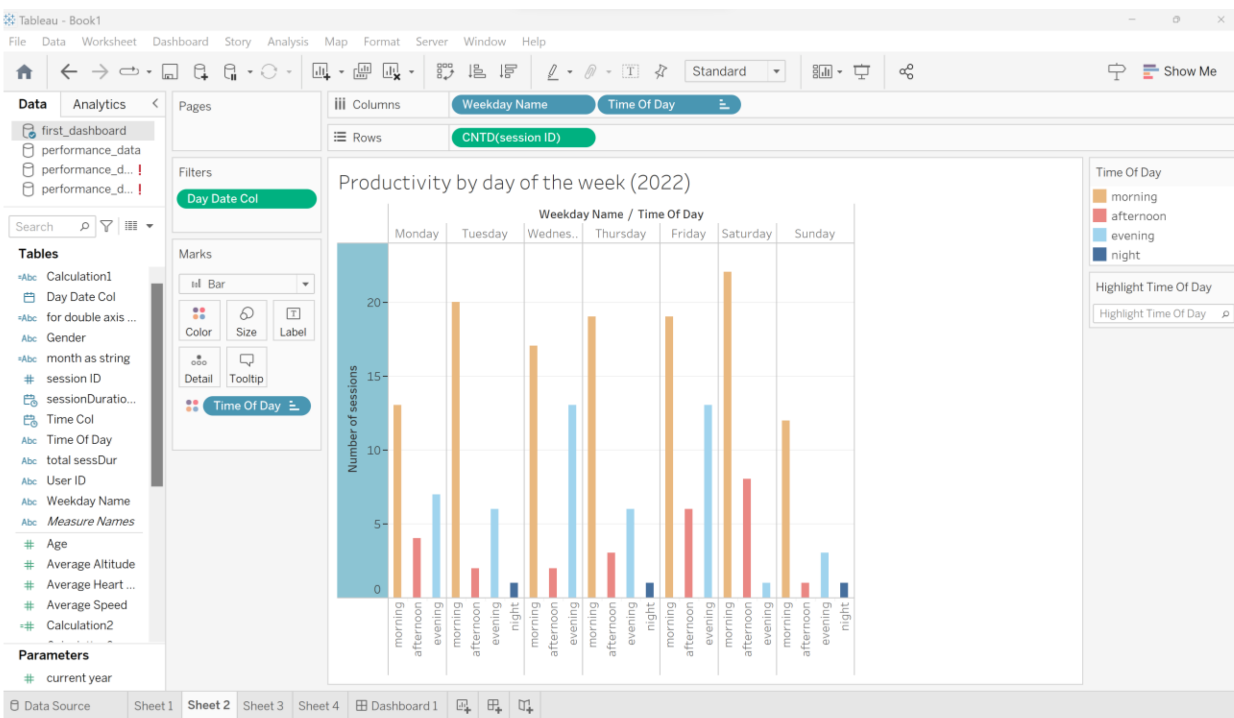
https://github.com/alexbaar/Historical-Dataset-Analysis/tree/main/Tableau

- Dashboard results:

a) Here, we can clearly see that morning sessions were the most frequent



b) We can conclude that mornings in general and Wednesday/Friday evenings featured the most activity

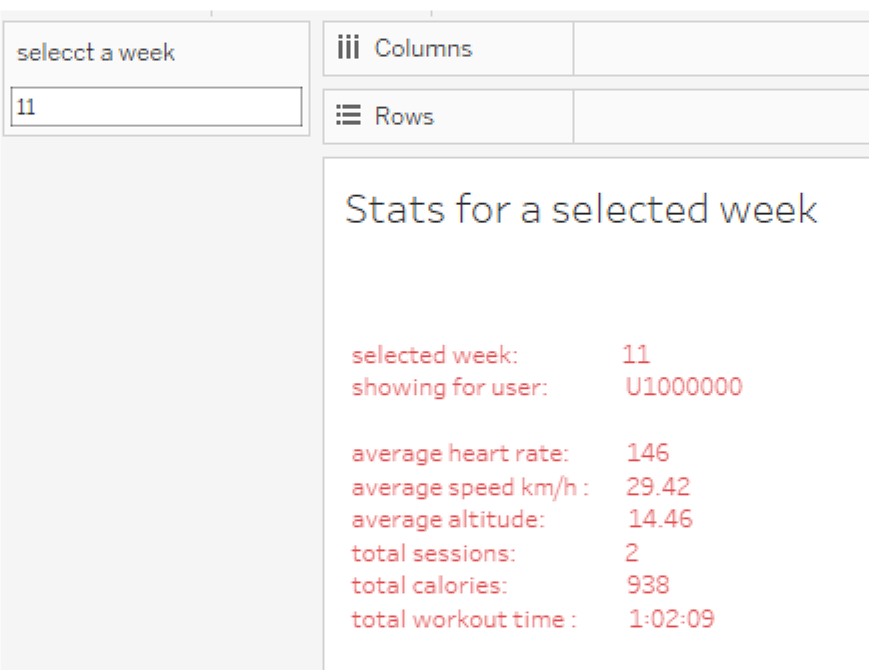


c) Current vs previous year comparison presenting monthly total time of sessions. The width of the lines depend on the number of individual sessions (thin = less, thick = more).

d) Select a week and show its stats

<u>Note for future work:</u>

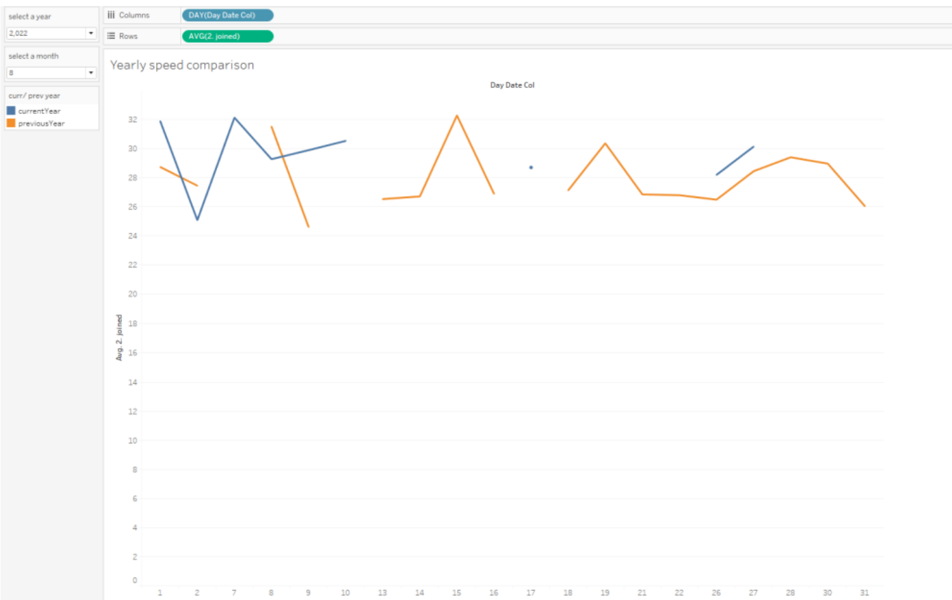
Introduce second parameter ‘year’. Adjust the ‘selecct a week’ parameter to accept values from 1 to 52 (1 year = 52 weeks). At the moment, a script was created to number all weeks from the dataset (across 2021 and 2022 which gave 84 weeks in total). So at the moment we can input any number from 1 to 84, but there is no distinction regarding the year/month that week is pulled from.



e) A comparison of daily average speed across a chosen month vs the month before the chosen one. Ideally we would get more datapoints so that line plot appears clearer.

<u>Note for future work:</u>

Find a way to display null/ zero values for the days that no activity was recorded – include all dates in the data collection. Also change the x axis scale so that all days appear, even those with no values.



8. FLUTTER DASHBOARD VISUALIZATION

- Set up FLUTTER:

Follow Victor Qin's 'Smart Bike Mobile App development environment setup' from User Manual.

If encountering the below error from step 'install Flutter-Firebase console on your machine':

```
firebase : The term 'firebase' is not recognized as the name of a cmdlet, function, script file, or operable program. Check the spelling of
the name, or if a path was included, verify that the path is correct and try again.
At line:1 char:1
+ firebase --version
+ ~~~~~~~~
    + CategoryInfo          : ObjectNotFound: (firebase:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException
```

Make sure the following conditions are met: (all should be executed in terminal)

a) Check if the latest version of node.js LTS is installed. If in doubt, uninstall the current version and download the latest one, which is a compatible one with firebase.

b) Reinstall firebase CLI:

npm uninstall -g firebase-tools

npm install -g firebase-tools

c) Set the execution policy and verify the installation. The desired output should be:

```
PS C:\Users\milly> Get-ExecutionPolicy
Restricted
PS C:\Users\milly> Set-ExecutionPolicy RemoteSigned
PS C:\Users\milly> firebase --version
13.0.2
```

d) Execute the below line. It will redirect you to firebase. Enter you login details.

```
firebase login
```

```
PS C:\Users\milly> firebase login
i  Firebase optionally collects CLI and Emulator Suite usage and error reporting information to help improve
privacy policy (https://policies.google.com/privacy) and is not used to identify you.

? Allow Firebase to collect CLI and Emulator Suite usage and error reporting information? Yes
i  To change your data collection preference at any time, run `firebase logout` and log in again.

Visit this URL on this device to log in:
https://accounts.google.com/o/oauth2/auth?client_id=563584335869-fgrhgmd47bqnekij5i8b5pr03ho849e6.apps.googl
gleapis.com%2Fauth%2Fcloudplatformprojects.readonly%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Ffirebase%20h
nse_type=code&state=670215369&redirect_uri=http%3A%2F%2Flocalhost%3A9005

Waiting for authentication...

+  Success! Logged in as o.bartosiak@gmail.com
```

e) Finally execute this command

```
dart pub global activate flutterfire_cli
```

- Access the Mobile App files:

https://github.com/redbackoperations/Projects/tree/main/SmartBikeMobileApp/smart_bike_mobile_app

Helpful links:

https://www.youtube.com/watch?v=aCUM4r1ONhg 8:54

https://www.youtube.com/watch?v=GRQIYu6JxSg 8:38

https://www.youtube.com/watch?v=_V8eKsto3Ug 2:10:38

https://www.youtube.com/watch?v=KmYUE7Of5rU 9:35

https://www.youtube.com/watch?v=DWLoY6I6d34 5:11

https://www.youtube.com/watch?v=QnNOh-Wza_Q 3:24

Tableau:

https://www.youtube.com/watch?v=Zb-2RR2VbJo 31:31

https://www.youtube.com/watch?v=2oO7lzWr0f0 12:50

Flutter:

https://www.youtube.com/playlist?list=PL4cUxeGkcC9jLYyp2Aoh6hcWuxFDX6PBJ 3:00:00