Project Big Data, Business Intelligence and NoSQL   
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
Teamleden:

Thomas Billiet

Ishan Ameel

Sven Depickere

Jonas Anseel

Inhoud

[1. Preprocessing van de dataset 2](#_Toc513734454)

[1.1. Data loading 2](#_Toc513734455)

[1.2. Cleaning functions 3](#_Toc513734456)

[1.3. Convert military time to datetime 5](#_Toc513734457)

[1.4. Add time zones 6](#_Toc513734458)

[1.5. Change local time to UTC 7](#_Toc513734459)

[1.6. Calculate duration 8](#_Toc513734460)

[1.7. Calculate distance 9](#_Toc513734461)

[1.8. Calculate speed 10](#_Toc513734462)

[1.9. Remove too long and too short flights 11](#_Toc513734463)

[1.10. Execute cleaning 12](#_Toc513734464)

[2. Tableau 13](#_Toc513734465)

[3. Machine learning met Scikit learn 22](#_Toc513734466)

[4. NetworkX 23](#_Toc513734467)

# Preprocessing van de dataset

We hebben onze code in verschillende stappen opgesplitst namelijk:

* Data loading
* Cleaning functions
* Convert military time to datetime
* Add time zones
* Change local time to UTC
* Calculate duration
* Calculate distance
* Calculate speed
* Remove too long too short flights
* Execute cleaning

## Data loading

1. IMPORT\_PATH = '../Initial\_Data/Unzipped'
2. IMPORT\_ALL\_PATH = IMPORT\_PATH + '/\*'
3. import pandas as pd
4. import numpy as np
5. import glob as glob
6. def readAllFiles():

files = glob.glob(IMPORT\_ALL\_PATH)

frames = []

for file in files:

df = pd.read\_csv(file, index\_col = 0)

frames.append(df)

return pd.concat(frames)

def readOneFile(url):

return pd.read\_csv(url, index\_col = 0)

df = readAllFiles()

Hier in lezen we de data in als een panda frame. De csv file bevinden zich in een bepaalde direct die luisterd naar de naam */Initial\_Data/Unzipped*.

Het aantal records dat we in het begin hebben zijn er **11.401.196**

## Cleaning functions

1. import pandas as pd
2. import numpy as np
3. import glob as glob from math
4. import \*
5. usaStates = ["AL", "AK", "AZ", "AR", "CA", "CO", "CT", "DE", "FL", "GA", "HI", "ID", "IL", "IN", "IA", "KS", "KY", "LA", "ME", "MD", "MA", "MI", "MN", "MS", "MO", "MT", "NE", "NV",  "NH", "NJ", "NM", "NY", "NC", "ND", "OH", "OK", "OR", "PA", "RI", "SC", "SD", "TN", "TX",  "UT", "VT", "VA", "WA", "WV", "WI", "WY"]
6. def deleteWrongStates(df):

print("Aantal records:", len(df))

for el in df.departure\_state.unique():

if (el not in usaStates):

df = df.drop(df[df['departure\_state'] == el].index)

print("Aantal records na verwijderen foute vertrek staat:", len(df))

for el in df.arrival\_state.unique():

if (el not in usaStates):

df = df.drop(df[df['arrival\_state'] == el].index)

print("Aantal records na verwijderen foute aankomst staat:", len(df))

return df

def convertColumnTypes(df):

df.departure\_schedule = df.departure\_schedule.astype(int)

df.departure\_delay = df.departure\_delay.astype(float)

df.arrival\_schedule = df.arrival\_schedule.astype(int)

df.arrival\_delay = df.arrival\_delay.astype(float)

df.arrival\_actual = df.arrival\_actual.astype(int)

df.departure\_actual = df.departure\_actual.astype(int)

return df

def dropMoreAdvancedDuplicates(df):

df = df.groupby(['date', 'airline', 'airline\_code', 'departure\_airport',

 'departure\_state', 'departure\_lat', 'departure\_lon', 'departure\_schedule',

 'arrival\_airport', 'arrival\_state', 'arrival\_lat', 'arrival\_lon',

'arrival\_schedule']).mean().reset\_index()

df = convertColumnTypes(df)

if 'index' in df.columns:

df = df.drop(['index'], axis = 1) #remove old index

return df

The United States Of America telt 50 staten. In de data dat we gekregen hebben zijn er 3 staten die geen staat zijn van The United States Of America. Deze zullen we namelijk verwijderen.

Die staten zijn:

* TT (Trust Territory of the Pacific Islands):
  + bestaat niet meer sinds 1986
    - Bestaat nu uit:
      * Republic of the Marshall Islands
      * Federated States of Micronesia
      * Republic of Palau
      * Commenwealth of the Northern Mariana Islands
* PR (Puerto Rico):
  + is een territorium
* VI (U.S Virgin Islands) :
  + is een territorium

Er zijn **56 413** vluchten die vertrekken van uit staten die gen officiële staten zijn.

Er zijn **53 184** vluchten die landen in staten die geen officiële staten zijn.

We veranderen ook de kolommen naar het gepaste type voor dat verder bewerking gemakkelijk gaan. Daarna gaan we nog eens controleren op verborgen dubbels die we dan opnieuw verwijderen.

## Converteer militaire tijd

1. **import** math
2. **from** datetime **import** datetime
4. **def** getTotalMinutes(time):
5. time\_str = str(time).rjust(4, "0")
6. hours = time\_str[:-2]
7. hours = int(hours)
8. **if** hours == 24:
9. hours = 0
10. minutes = int(time\_str[-2:])
11. **return** datetime(2010, 1, 1, hours, minutes, 0)
13. **def** convertMilitaryTime(df):
14. df["departure\_schedule"] = df["departure\_schedule"].map(getTotalMinutes)
15. df["arrival\_schedule"] = df["arrival\_schedule"].map(getTotalMinutes)
16. **return** df

De dataset bevat vertrek en aankomsttijden in militair formaat (bv. 615 = 06:15). Om met deze tijden te kunnen werken in Python moeten we ze omzetten naar een “datetime” object.

## Voeg tijdzones toe

1. # UTC offset by state
2. timezones = {
3. "AL" :-6,
4. "AK" :-9,
5. "AZ" :-7,
6. "AR" :-6,
7. "CA" :-8,
8. "CO" :-7,
9. "CT" :-5,
10. "DE" :-5,
11. "FL" :-5,
12. "GA" :-5,
13. "HI" :-10,
14. "ID" :-7,
15. "IL" :-6,
16. "IN" :-5,
17. "IA" : -6,
18. "KS" : -6,
19. "KY" : -6,
20. "LA" : -6,
21. "ME" : -5,
22. "MD" : -5,
23. "MA" : -5,
24. "MI" : -5,
25. "MN" : -6,
26. "MS" : -6,
27. "MO" : -6,
28. "MT" : -7,
29. "NE" : -6,
30. "NV" : -8,
31. "NH" : -5,
32. "NJ" : -5,
33. "NM" : -7,
34. "NY" : -5,
35. "NC" : -5,
36. "ND" : -6,
37. "OH" : -5,
38. "OK" : -6,
39. "OR" : -8,
40. "PA" : -5,
41. "RI" : -5,
42. "SC" : -5,
43. "SD" : -6,
44. "TN" : -6,
45. "UT": -7,
46. "TX": -6,
47. "VT": -5,
48. "VA": -5,
49. "WA": -8,
50. "DC": -5,
51. "WV": -5,
52. "WI": -6,
53. "WY": -5
54. }
56. **def** addTimezones(df):
57. df['arrival\_tz'] = df['arrival\_state'].map(**lambda** state: timezones[state])
58. df['departure\_tz'] = df['departure\_state'].map(**lambda** state: timezones[state])
59. **return** df

Alle tijden in de dataset zijn lokaal. Dus om ze bruikbaar te maken moeten we ze omzetten naar UTC-tijden. Hier voegen we de afwijking van UTC (in uren) toe aan elke vlucht.

## Zet alle tijden op UTC

1. **from** datetime **import** timedelta
3. **def** timeToUTC(schedule, tz):
4. tdelta = timedelta(hours=abs(tz))
5. **if** tz > 0:
6. schedule -= tdelta
7. **else**:
8. schedule += tdelta
9. **return** schedule
11. **def** convertLocalToUTC(df):
12. arrival\_schedules = []
13. departure\_schedules = []
14. **for** i, row **in** df.iterrows():
15. arrival\_schedule = row["arrival\_schedule"]
16. arrival\_tz = row["arrival\_tz"]
17. departure\_schedule = row["departure\_schedule"]
18. departure\_tz = row["departure\_tz"]
20. arrival\_schedules.append(timeToUTC(arrival\_schedule, arrival\_tz))
21. departure\_schedules.append(timeToUTC(departure\_schedule, departure\_tz))
23. df["arrival\_schedule"] = arrival\_schedules
24. df["departure\_schedule"] = departure\_schedules
25. **return** df

Nu we de tijdzones hebben kunnen we de tijden omzetten naar UTC.

## Voeg duur toe

1. **def** addDurations(df):
2. durations = []
3. **for** i, row **in** df.iterrows():
4. arrival = row["arrival\_schedule"]
5. departure = row["departure\_schedule"]
6. **if** arrival <= departure:
7. arrival += timedelta(days=1)
8. durations.append(arrival - departure)
10. df["duration"] = durations
11. **return** df

Met de vertrek en aankomsttijden in dezelfde tijdzone kunnen we nu de duur afleiden van de vluchten.

## Calculate distance

1. **def** addDistances(df):
2. distances = []
3. **for** index, row **in** df.iterrows():
4. distances.append(calcTheDistance(row))
5. df["distance"] = distances
6. **return** df
8. **def** calcTheDistance(el):
9. slat = radians(float(el["arrival\_lat"]))
10. slon = radians(float(el["arrival\_lon"]))
11. elat = radians(float(el["departure\_lat"]))
12. elon = radians(float(el["departure\_lon"]))
13. **return** 6371.01 \* acos(sin(slat)\*sin(elat) + cos(slat)\*cos(elat)\*cos(slon - elon))

We kunnen de afstand berekenen aan de hand van coördinaten van de luchthavens.

We zijn er bewust van dat die afstand niet 100% perfect is vanwege de gebruikte methode de *Great-circle distance* bereking. Die is namelijk niet 100% perfect want dat komt omdat de aarde namelijk geen perfecte bol is. En vliegtuigen vliegen namelijk niet altijd dezelfde routes vanwege de straalstroom, weeromstandigheden… Maar het is een goede benadering.

## Voeg snelheid toe

1. **def** addSpeed(df):
2. speeds = []
3. **for** i, row **in** df.iterrows():
4. duration = row["duration"]
5. speed = row["distance"] / ((duration.total\_seconds() / 60) / 60)
6. speeds.append(speed)
8. df["speed"] = speeds
9. **return** df

We berekenen de gemiddelde snelheid van het vliegtuig tijdens de geplande vlucht. Dit houd geen rekening met vertragingen.

## Remove too long and too short flights

1. **def** deleteImpossibleFlights(df):
2. shortest\_distance\_in\_us = 100 #KM
3. longest\_distance\_in\_us = 9000 #KM
4. shortest\_duration\_in\_us = timedelta(minutes=16)
5. longest\_duration\_in\_us = timedelta(hours=11, minutes=40)
7. df = df[df.distance.between(shortest\_distance\_in\_us, longest\_distance\_in\_us)]
8. df = df[df.duration.between(shortest\_duration\_in\_us,longest\_duration\_in\_us)]
9. **return** df

## Execute cleaning

1. **print**("Aantal records om te beginnen:", len(df))
2. cleaning = df
4. cleaning = cleaning.drop\_duplicates()
5. **print**("Aantal records na verwijderen van dubbels:", len(cleaning))
7. cleaning = cleaning.reset\_index()
8. **print**("Aantal records na nieuwe index:", len(cleaning))
10. cleaning = cleaning.dropna()
11. **print**("Aantal records na verwijderen lege waarden:", len(cleaning))
13. cleaning = deleteWrongStates(cleaning)
14. cleaning = convertColumnTypes(cleaning)
16. cleaning = dropMoreAdvancedDuplicates(cleaning)
17. **print**("Aantal records na het verwijderen van de verborgen dubbels:", len(cleaning))
19. cleaning = convertMilitaryTime(cleaning)
20. cleaning = addTimezones(cleaning)
21. cleaning = convertLocalToUTC(cleaning)
22. cleaning = addDurations(cleaning)
23. cleaning = addDistances(cleaning)
24. cleaning = addSpeed(cleaning)
26. cleaning = deleteImpossibleFlights(cleaning)
27. **print**("Aantal records na het verwijderen van de onmogelijk lange of korte vluchten:", len(cleaning))
29. cleanData = cleaning

Als je dit uitvoert print python dit:

Aantal records om te beginnen: 11401196

Aantal records na verwijderen van dubbels: 10751921

Aantal records na nieuwe index: 10751921

Aantal records na verwijderen lege waarden: 10751919

Aantal records: 10751919

Aantal records na verwijderen foute vertrek staat: 10695506

Aantal records na verwijderen foute aankomst staat: 10642322

Aantal records na het verwijderen van de verborgen dubbels: 10642032

Aantal records na het verwijderen van de onmogelijk lange of korte vluchten: 10552133

Dus we waren begonnen met **11.401.196** records en hebben er nog **10.552.133** over. Dat wilt zeggen dat er **849.063** verkeerde de data tussen zit, dat is ongeveer **7.447%** van de data.

# Tableau

Inhoud:

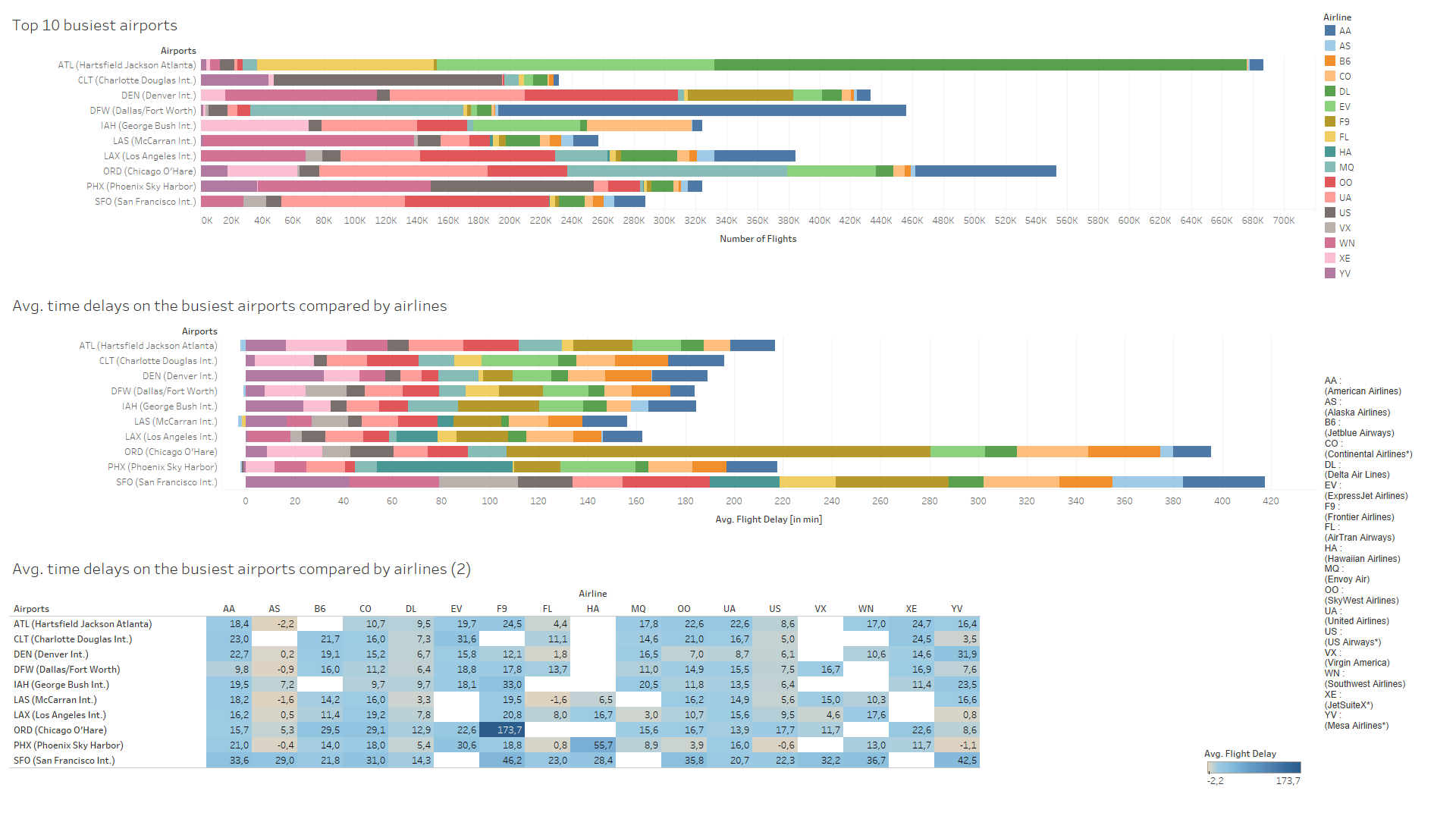
Grafiek 1

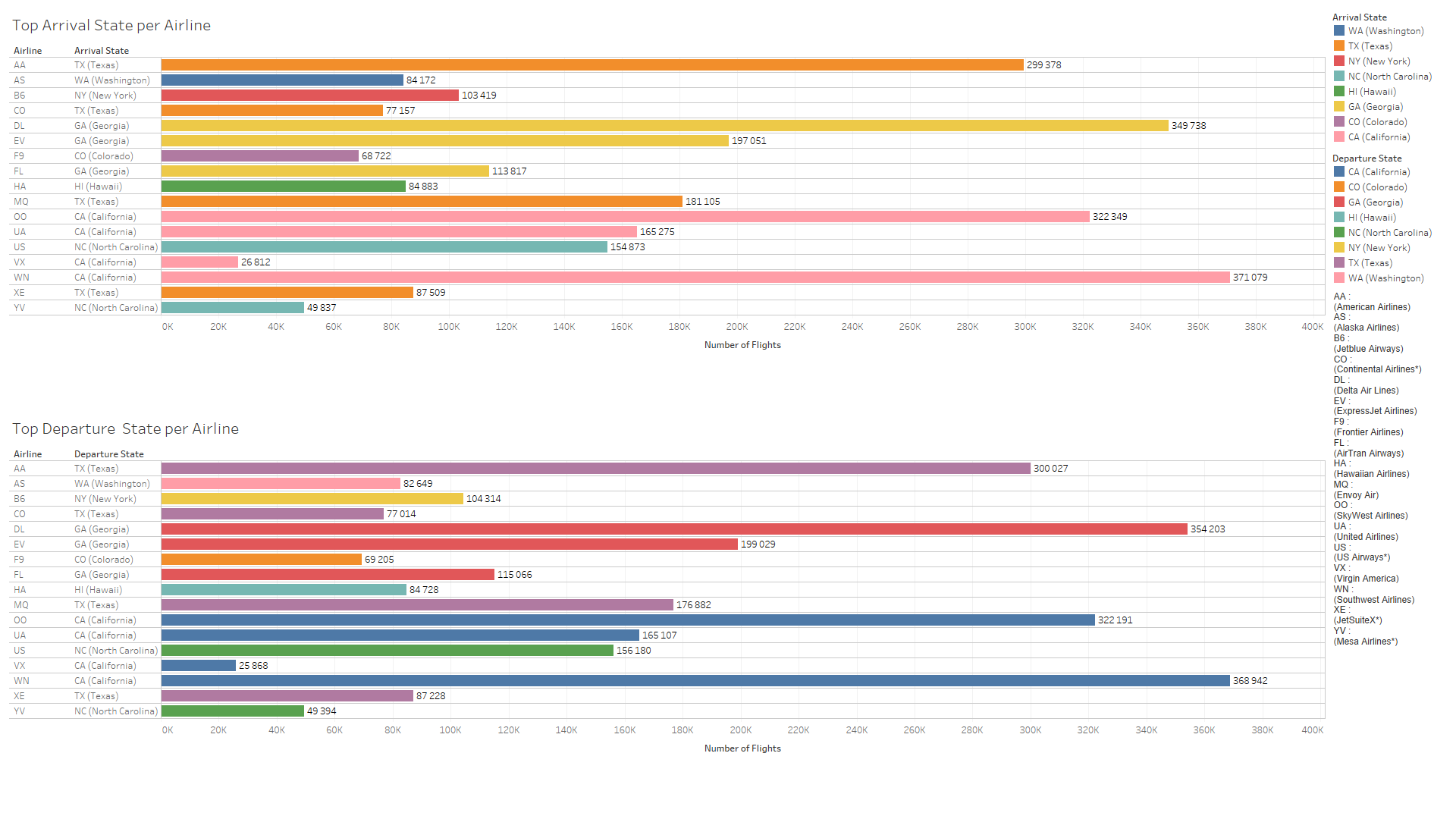
Grafiek 1

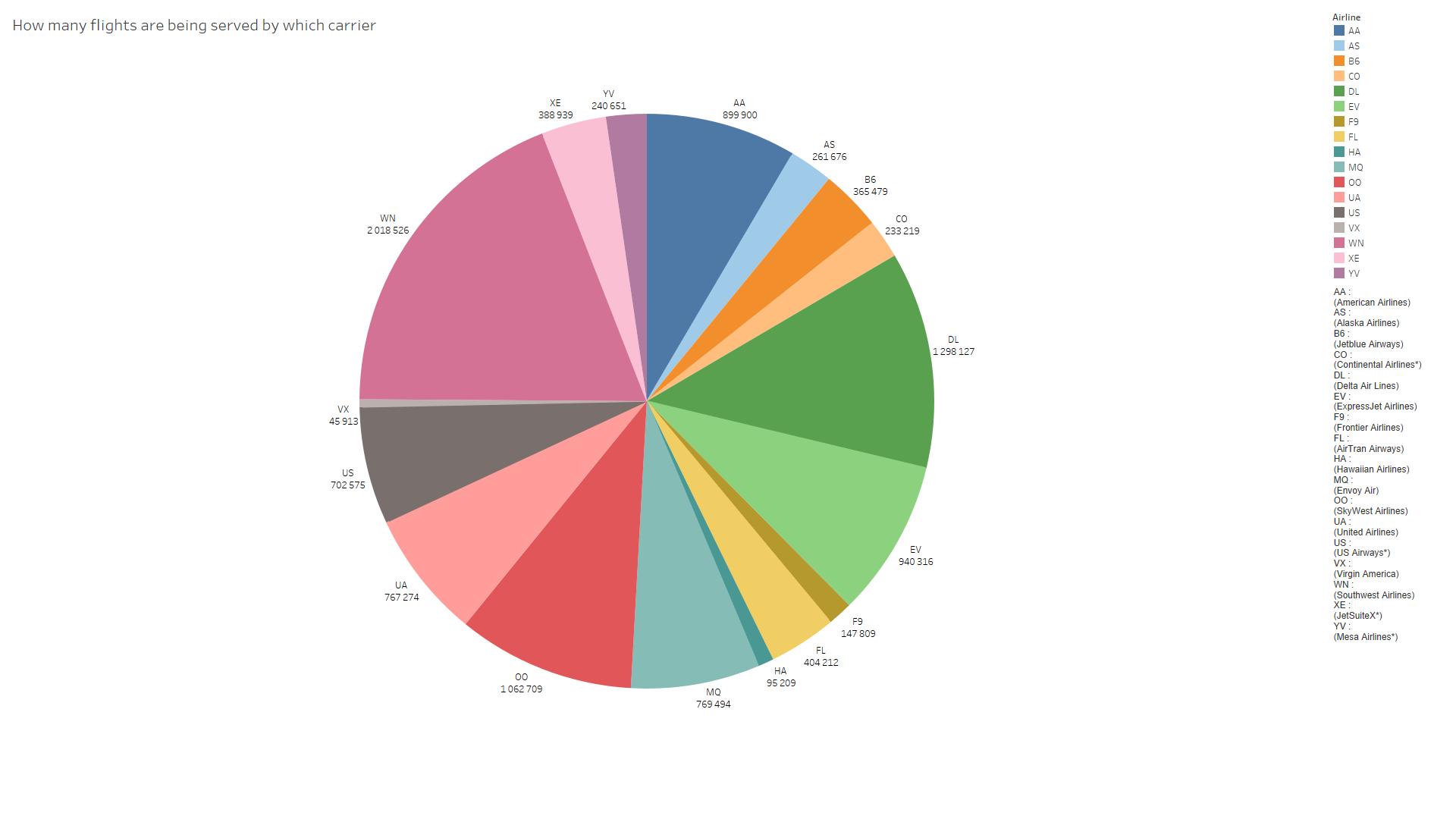
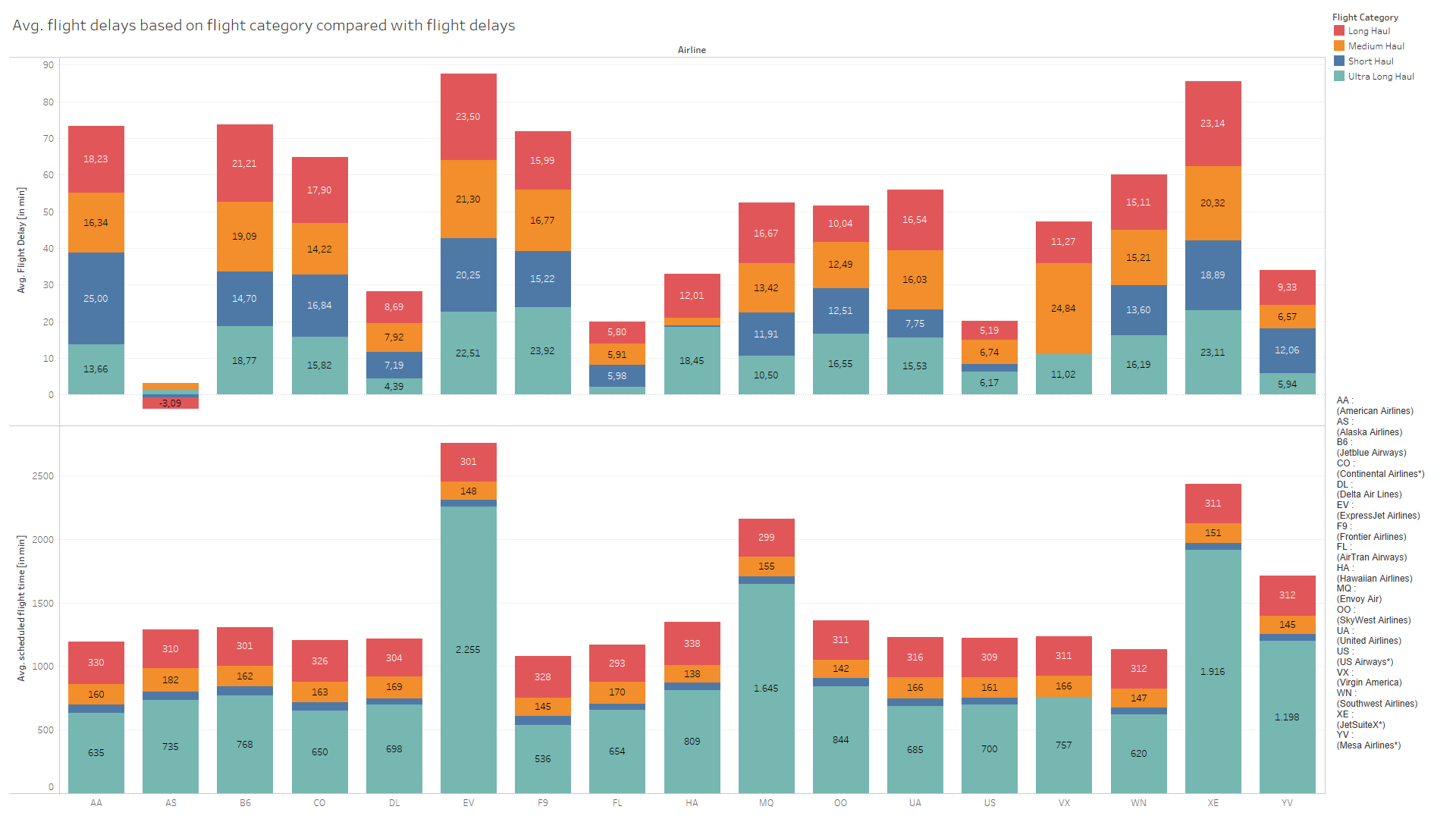
Grafiek 1

Grafiek 1

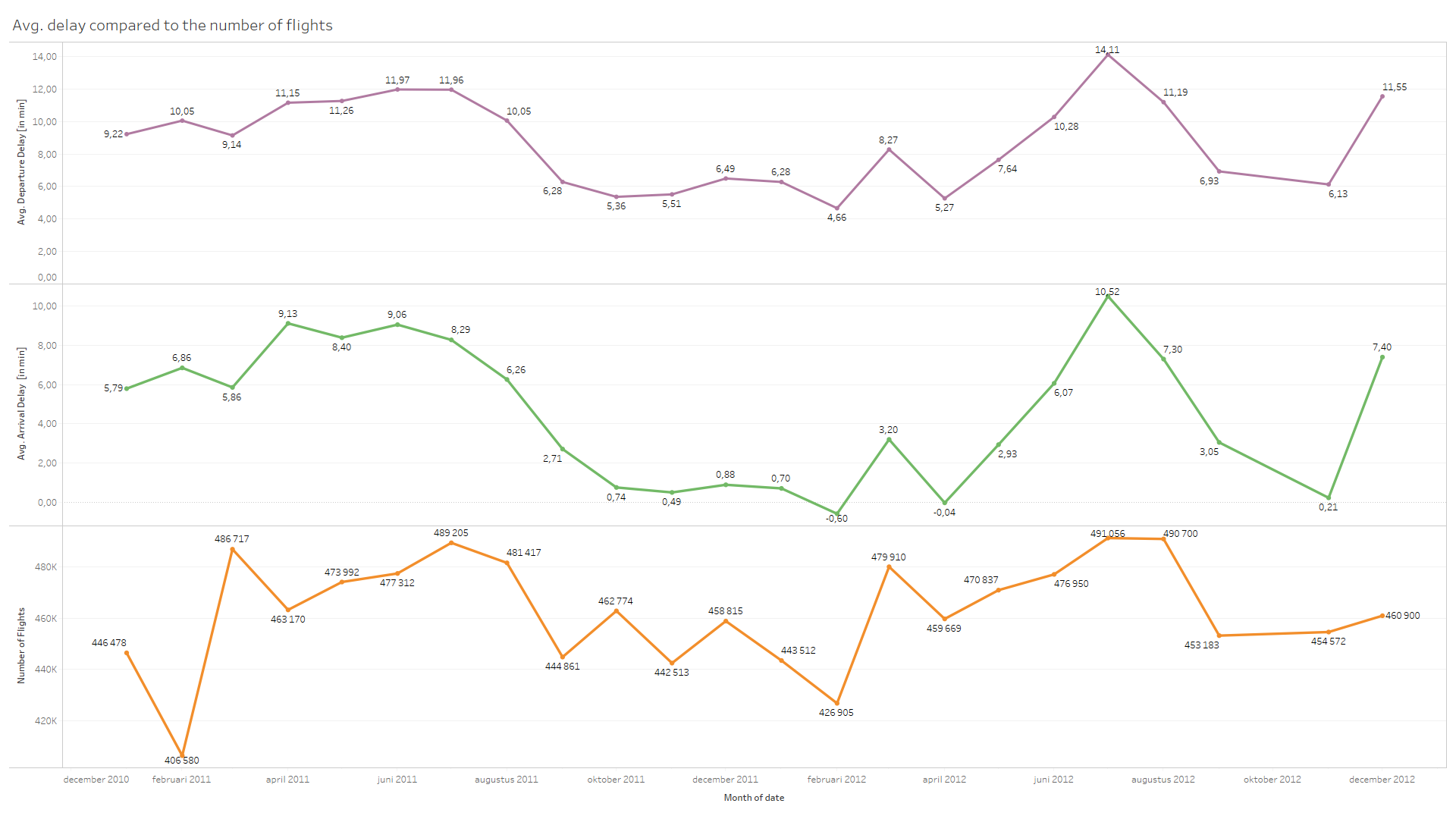
Grafiek 1

Grafiek 1,2 en 3:

Grafiek 4 en 5:

Grafiek 6: Grafiek 7: 

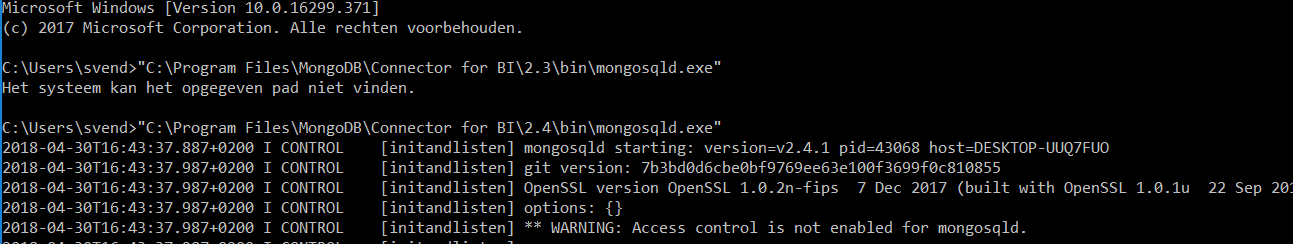
Grafiek 8:



"C:\Program Files\MongoDB\Connector for BI\2.4\bin\mongosqld.exe"

cd C:\Program Files\MongoDB\Server\3.6\bin

https://www.transtats.bts.gov/DL\_SelectFields.asp



# Machine learning met Scikit learn

# NetworkX

## Laad de dataset

1. **import** math
2. **import** json
3. **import** numpy as np
4. **import** pandas as pd
5. **import** networkx as nx
6. **import** cartopy.crs as ccrs
7. **import** matplotlib.pyplot as plt
8. **from** IPython.display **import** Image
9. %matplotlib inline
11. **import** glob
13. BASIC\_PATH = './Data/'
14. ALL\_FILES = BASIC\_PATH + '\*.csv'
16. **def** readOneFile(url):
17. **return** pd.read\_csv(url, index\_col = 0)
19. data = readOneFile(BASIC\_PATH + "flights\_2010\_9.csv")
20. flights = pd.DataFrame(data=data)
21. flights

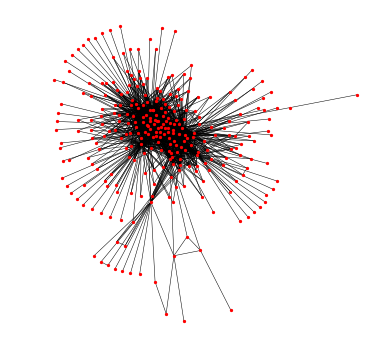
## Routes tussen luchthavens

1. edges = flights[['departure\_airport', 'arrival\_airport']].values
2. edges

Uit de vluchten leiden we de routes tussen luchthavens af.

## Grafiek van alle luchthavens en hun routes

1. g = nx.from\_edgelist(edges)
2. fig, ax = plt.subplots(1, 1, figsize=(6, 6))
3. nx.draw\_networkx(g, ax=ax, with\_labels=False,
4. node\_size=5, width=.5)
5. ax.set\_axis\_off()



## Vind de coördinaten van elke luchthaven

1. pos = {}
2. airports = flights[["departure\_airport", "departure\_lat", "departure\_lon"]].drop\_duplicates()
3. **for** i, row **in** airports.iterrows():
4. airport = row["departure\_airport"]
5. pos[airport] = (row["departure\_lon"], row["departure\_lat"])

Elke vlucht heeft coördinaten bij het vertrek punt en de aankomst. We kunnen dus de locatie van elke luchthaven hieruit afleiden.

## Plaats de routes op kaart

1. deg = nx.degree(g)
2. sizes = [5 \* deg[iata] **for** iata **in** g.nodes]
4. labels = {iata: iata **if** deg[iata] >= 60 **else** ''
5. **for** iata **in** g.nodes}
7. crs = ccrs.PlateCarree()
8. fig, ax = plt.subplots(
9. 1, 1, figsize=(12, 8),
10. subplot\_kw=dict(projection=crs))
12. ax.coastlines()
14. ax.set\_extent([-163, -67, 18, 64])
15. nx.draw\_networkx(g, ax=ax,
16. font\_size=14,
17. alpha=.5,
18. width=.07,
19. node\_size=sizes,
20. labels=labels,
21. pos=pos,
22. node\_color=sizes,
23. cmap=plt.cm.cool)

De volgende kaart bevat elke route en luchthaven in de dataset. De grote en kleur van een luchthaven geeft aan hoeveel connecties het bevat.

## C:\Users\jodus\AppData\Local\Packages\Microsoft.Office.Desktop_8wekyb3d8bbwe\AC\INetCache\Content.MSO\E873C252.tmp