Programmeertechnieken 2025
Opdracht 1: Data Analysis Pipelines

Deadline: Maandag 3 maart 08u59.

1 Introductie

Het analyseren van datasets is een veelvoorkomende taak in de informatica. Er zijn vele verschillende typen datasets, zoals bijvoorbeeld een dataset met resultaten van een set experimenten, "access logs" van een webserver of voorraadbeheer van een bedrijf opgeslagen in een relationeel databasesysteem. Op een dataset kunnen analyses worden uitgevoerd met variërende complexiteit: eenvoudige analyses zoals het tellen hoe vaak een bepaalde karakteristiek voorkomt (hoe vaak is een bepaalde pagina van een website bezocht in de laatste maand?) tot ingewikkelde analyses waarbij er bijvoorbeeld wordt gezocht naar patronen in de data (als iemand product A koopt, wat voor ander type producten zitten er dan in dezelfde aankoop?). In deze opdracht beperken we ons tot de eenvoudigere analyses. Het zoeken naar patronen in data komt later in de studie uitgebreid aan de orde bij het vak Data Mining.

In deze opdracht gaan we voertuiggegevens analyseren. De RDW houdt gegevens bij over alle voertuigen met een kenteken. De opdracht bestaat uit het genereren van verschillende statistieken gegeven de open data van de RDW. Bij het uitvoeren van de analyses is het de bedoeling optimaal gebruik te maken van de kracht van UNIX-systemen: in plaats van zelf één groot programma te schrijven dat de analyse uitvoert, maken we een aaneenschakeling van al bestaande tools op een UNIX systeem. In een dergelijke aaneenschakeling wordt de uitvoer van het ene programma steeds doorgestuurd naar een ander programma. Het doorsturen van data wordt gedaan met behulp van een "pipe", vandaar dat voor dergelijke aaneenschakelingen vaak de term "pipeline" wordt gebruikt.

Het komt wel eens voor dat er nog geen geschikt programma bestaat om in de pipeline op te nemen. In dat geval zit er niets anders op dan het programma zelf te schrijven. Vaak wordt er gebruik gemaakt van een scripttaal, zoals Perl of Python, om in weinig tijd een geschikt programma te schrijven. We zullen dat in deze opdracht ook doen. Houd bij het ontwikkelen van deze extra programma's de UNIX filosofie in het achterhoofd! Maak de programma's zo generiek mogelijk, zodat je ze in de toekomst zou kunnen hergebruiken.

2 Dataset

Als dataset zullen we gebruik maken van de dataset "Gekentekende voertuigen" van de RDW[1]. Het betreft hier een publieke dataset. Je kan deze dataset downloaden door te kiezen voor "Exporteer" en vervolgens "CSV". Ongecomprimeerd omvat de database ongeveer 7 GB aan data (ca. 15 miljoen regels). Voor het verslag is het de bedoeling dat *alle* data wordt verwerkt.

Het gaat om een tekstbestanden geformatteerd met kommagescheiden velden (CSV, comma-separated values). Elke regel in het bestand is één voertuig. Een overzicht van de kolommen staat op de gelinkte website van de RDW en de eerste regel van het bestand geeft ook de veldnamen aan. Het `cut`-commando is handig om specifieke kolommen uit het bestand te selecteren.

Sommige regels bevatten dubbele aanhalingstekens om aan te geven dat er komma's in een veld staan. Deze regels worden door `cut` niet goed verwerkt. Je mag alle regels met dubbele aanhalingstekens negeren om dit probleem te voorkomen. Dat kan je doen met behulp van `grep -v`.

Op de Liacs scratch kan je het bestand "Gekentekende_voertuigen.csv.bz2" vinden. Dit is een gecomprimeerde versie (met bzip2) van het databestand die een stuk minder ruimte inneemt. De

[1] `https://opendata.rdw.nl/Voertuigen/Open-Data-RDW-Gekentekende_voertuigen/m9d7-ebf2`

regels met aanhalingstekens zijn hier ook al uitgehaald. Het is de bedoeling dat je voor de opdracht deze versie gebruikt. Je kan `bzcat` gebruiken als deel van je pipeline om dit bestand uit te pakken. Er staat ook een kleiner bestand "Gekentekende_voertuigen-sample.csv.bz2" dat een willekeurige sample van 10,000 regels van het origineel bevat. Dit bestand kan je gebruiken om je pipelines veel sneller te kunnen testen.

3 Te verrichten analyses

De opdracht is nu als volgt: bouw geschikte pipelines om de onderstaande uitvoer te genereren. Plaats de pipeline steeds in een shell script (`pipeline1.sh`, enz.). De shell scripts hebben één argument, namelijk de locatie van het gecomprimeerde databestand. Dit verwijst naar een bestand met de versie van de dataset.

1. Een lijst van alle mogelijke kentekenindelingen van personenauto's (zie `grep`), waarbij elke positie met een letter wordt aangegeven als "X" en elke positie met een cijfer als "9" (kijk bijvoorbeeld naar `tr`). Het kenteken "123XYZ" heeft bijvoorbeeld indeling "999XXX". Elke regel bevat enkel een kentekenindeling. Zorg ervoor dat geen kentekenindeling meer dan eens voorkomt (zie `sort` en `uniq`). N.b. de term 'auto' en 'personenauto' worden soms door elkaar gebruikt.

2. Overzicht top tien modellen van personenauto's, aflopend gesorteerd op aantal personenauto's met dezelfde merk/handelsbenaming combinatie. De uitvoer bevat alleen de naam van het merk gevolgd door een komma en de naam van het handelsbenaming. De regels zijn gesorteerd zodanig dat het populairste handelsbenaming eerst komt, maar het aantal staat niet in de uitvoer.

3. Hetzelfde, maar dan alleen voor modellen die als taxi geregistreerd staan. Maak gebruik van `awk`.

4. Bereken voor elk merk waarvan minstens 100,000 personenauto's geregistreerd zijn de gemiddelde leeftijd van personenauto's van dat merk, afgerond op gehele jaren. Er is geen tool die de leeftijd berekent, noch een tool die het gemiddelde berekent. Je zal daarvoor dus zelf (Python/awk/...) scripts moeten schrijven. Bereken de leeftijd op basis van het jaartal uit de "Datum eerste toelating", waarbij elk voertuig uit 2021 0 jaar oud is, uit 2020 1 jaar oud, enzovoorts. Negeer regels zonder datum (dus tel ze ook niet mee als nul). Elke regel bevat de gemiddelde leeftijd in gehele jaren, een spatie, en vervolgens het merk. De uitvoer is oplopend gesorteerd op leeftijd.

5. Een grafiek die voor elke dag van de week laat zien hoeveel personenauto's er op die dag tenaamgesteld zijn (op basis van de kolom "Datum tenaamstelling"). De grafiek heeft dus de 7 dagen van de week op de x-as en het totaal aantal personenauto's op de y-as. Gebruik een "bar plot", zie ook de appendix. De waarde afgebeeld voor dinsdag is dus bijvoorbeeld het totaal aantal personenauto's dat op een dindag tenaamgesteld is in de dataset.

   (Schrijf bijvoorbeeld een script dat een datum om kan zetten naar een dag van de week (kijk eens naar de `datetime` *module). Dan staat alle data klaar om de totalen te berekenen. Tenslotte zou je een Python script kunnen schrijven dat gegeven een lijstje van dagen van de week en totaal aantal tenaamgestelde personenauto's op die dag een PDF-bestand met een grafiek genereert (bijvoorbeeld* NumPy *en* matplotlib *of* pandas*, zie ook de appendix)).*

6. Bepaal van personenauto's per merk de gemiddelde cataloguswaarde en het gemiddelde vermogen per kg (het veld "Vermogen massarijklaar"). Negeer regels zonder cataloguswaarde of vermogen (dus tel ze ook niet mee als nul). Maak een scatterplot van de merken met minstens 10,000 geregistreerde personenauto's, met de gemiddelde cataloguswaarde op de x-as en het gemiddelde vermogen per kg op de y-as. Documentatie over scatterplots met matplotlib is hier te vinden: `https://matplotlib.org/stable/gallery/shapes_and_collections/scatter.html`.

4 git

Je moet voor deze opdracht gebruik maken van het source control systeem git via de LIACS git server `git.liacs.nl`. Het is de bedoeling dat je actief gebruik maakt van git tijdens het ontwikkelen en debuggen van je scripts, dus niet dat je alleen op het eind je definitieve versie commit. Je moet het git repository (de `.git` directory) ook inleveren.

Één groepslid maakt op `https://git.liacs.nl/` een nieuw project aan met de naam `pt1`. Zet de toegangsinstelling op "private" (dus *niet* op "internal") en geef toegang aan je groepsgenoot. Vervolgens kunnen beide groepsgenoten het repository klonen als volgt (vul het studentnummer van de student die het project heeft aangemaakt in):

```
git clone git@git.liacs.nl:s1234567/pt1.git
```

Commit *niet* het databestand in git, dat is veel te groot. Commit alleen je code.

5 Vereisten

Er wordt gewerkt in teams van twee personen. Het volgende moet worden ingeleverd:

- De source code van alle geschreven scripts.
- Shell scripts welke de geschreven pipelines uitvoeren.
- Een kort verslag in PDF formaat, gemaakt met behulp van LaTeX, waarin per analyse wordt uitgelegd hoe de pipeline is opgebouwd, de uitvoer van de pipeline wordt getoond (grafieken, tabellen of tekst) en de genereerde statistieken kort worden bediscussieerd. Voeg de gemaakte tabellen/grafieken dus in je verslag! De code hoef je *niet* in het verslag te zetten.
- Het git repository waar dit alles instaat (de `.git` directory).
- Let op: het hele databestand moeten worden meegenomen in de analyse.

Je levert je code in als gzipped tar bestand met in de bestandsnaam de studentnummers van de groepsleden. Ervan uitgaande dat je git repository in de directory `pt1` staat gebruik je het volgende commando:

```
tar -czvf pt1-sXXXXXXX-sYYYYYYY.tar.gz pt1/
```

Vul op de plek van `XXXXXXX` en `YYYYYYY` de bijbehorende studentnummers in. Lever dit bestand in via Brightspace. Het verslag hoeft *niet* op papier te worden ingeleverd.

Normering: Verslag (3/10), werking (4/10), kwaliteit code/commentaar/modulariteit (3/10).

Appendix: Tips & Tricks

Shellscripts

Een shellscript is simpelweg een tekstbestand dat shell-commando's bevat. De eerste regel bevat een hashbang. Dat is in dit geval de volgende tekst:

```
#!/bin/sh
```

Verder moet het bestand uitvoerbaar zijn. Dat doe je bijvoorbeeld als volgt (vul de naam van je eigen script in):

```
chmod +x myscript.sh
```

Als je een shellscript wilt uitvoeren moet je de directory opgeven, zelfs als het script in de huidige directory staat. Je kan het dan bijvoorbeeld als volgt uitvoeren (`.` verwijst naar de huidige directory):

```
./myscript.sh
```

In je shellscript kan je de opgegeven parameters krijgen als genummerde variabelen. Zo verwijs je bijvoorbeeld naar de eerste parameter als `"$1"`.

Handige commando's

De onderstaande commando's kunnen van pas komen bij de opdracht. Gebruik `man` om er meer informatie over te krijgen.

- `awk` (zie de slides);
- `bzcat` decomprimeert een `.bz2` bestand naar de standaard uitvoer;
- `cut` selecteert specifieke velden van de invoer op basis van een scheidingsteken (met `-f` en `-d`) of op basis van de positie op de regel van characters (met `-c`);
- `grep` selecteert regels van de invoer volgens een regular expression;
- `head -n`n selecteert de eerste n regels van de invoer;
- `python` (zie de slides);
- `rm` verwijdert een bestand;
- `sort` sorteert de regels van de invoer op alfabet;
- `sort -n` sorteert de regels van de invoer als numerieke waarden;
- `tail -n`n selecteert de laatste n regels van de invoer;
- `tr` vervangt opgegeven characters door andere characters;
- `uniq` verwijdert opeenvolgende dubbele regels in de invoer (meestal gebruikt in combinatie met `sort`).
- `uniq -c` telt opeenvolgende dubbele regels in de invoer (meestal gebruikt in combinatie met `sort`).

Eenvoudig plot voorbeeld

Het volgende voorbeeld leest een reeks regels van *stdin* welke bestaan uit (bijvoorbeeld) het uur van de dag en het aantal verstuurde bytes, gescheiden door een spatie. Gebaseerd op deze data wordt een plot gemaakt. Je kunt dit als startpunt gebruiken. Bekijk de documentatie van `matplotlib` om de plot verder op te maken.

```
import pandas
import matplotlib.pyplot as plt
import sys

D = pandas.read_csv(sys.stdin, sep=",", header=None, index_col=0)
D.plot(kind="bar", rot=30, legend=False)
plt.title("Mijn plot")
plt.show()
exit(0)
```