Python II

Python rettet mod databehandling:

Indhold:

Faglige mål:

Øvelsen dækker følgende målpinde:

1. **Eleven kan anvende Python til opbygning af en datapipeline baseret på ETL-programmeringsmønsteret.**
2. **Eleven kan oprette og bruge egne Python-moduler i komplekse programmeringsprojekter, så funktionalitet kan genbruges og vedligeholdes nemt.**
3. **Eleven kan anvende Python-moduler til at hente (extract) data fra internet- og netværkskilder og integrere dem i en programmeringspipeline på en sikker og robust måde, der demonstrerer beskyttelse af datatransport mod angreb samt håndtering af ustabile netværksforbindelser.**
4. **Eleven kan implementere databehandling ved hjælp af Python-moduler til databehandling, såsom pandas, dask eller PySpark.**
5. *Eleven kan anvende kryptografiske metoder til beskyttelse af data i en programmeringspipeline.*
6. **Eleven kan programmere mod databaser og anvende SQL i forbindelse med Python-programmering og gøre rede for, hvordan det anvendes i dataanalysen.**
7. **Eleven kan anvende Python-moduler til grafisk visualisering af data og gøre rede for, hvordan det anvendes i dataanalysen.**

Øvelse (Case) :

Baggrund

Du er ansat i en softwareudvikling organisation. Du får et naturcenter som kunde som ønsker at implementer et system på en Linux setup til analysering af blomster data til overvågning og dokumentation, med henblik på forskning. Blomster data indsamles og registeres af forsker og kan hentes/download (**extract**) fra et internationale datakilde på skyen som csv filer: https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv

Når data er hentet og gemt i destination folder på lokalt filsystem (LFS), skal data renses og behandles (**transform**) efter brugerens krav som er, at **KUN** observationer med arten **Iris-setosa** ønskes filteres til en ny csv fil. (*Altså filter data så kun "species" == "Iris-setosa" er tilbage).* Og det transformeret data gemmes ligeledes på LFS.

Til dataanalyse, ønsker kunden at anvende SQL-queries og udtræk data til deverse grafisk visninger. Det transformeret data ønskes derfor gemmes (**load**) i en MySQL database, hvorfra kunden kan fortag SQL-queries i Python til visuel præsentation af det transformeret data som: **scatter-plot**, **histogram** og **boxplots** diagrammer.

Problemformulering

- Hvordan kan du, på en Linux setup, implementer en Python script i en ETL (Extract-Transform-Load) mønster til gennemførsel af opgaven?
    - Hvordan kan du implementer **extract** delen med teknologier som er cross-platform hvor kode implementation er:
        1. beskyttes mod command-line injections
        2. HTTPS beskyttet under data transport
        3. Robusthed (data hentes i en stream af små blokker og ikke det hele på en gang)
    - Hvordan kan du med anvendelse af Python dataprocessering moduler, rense og filtere (**transform**) det hentede data.
    - Hvordan kan du gemme det transformeret data som csv
    - Hvordan kan du gemme det transformeret data som en tabel i din database.
    - Hvordan kan du anvend Python visualisering moduler til implementering af grafisk visninger af din transformeret data læst fra databasen.

Baggrundsteori

- **Python extract værktøjer**

  "Extract libraries/tools" til **at hente data fra eksterne kilder**, direkte i Python: **requests**, **wget**.
  Eller ved brug af eksterne programmer gennem **subprocess**: f.ex. kørsel af **curl** og **wget** via **subprocess**.

Værktøj	Type / Kategori	Use-case i "Extract"
requests	Python HTTP-bibliotek (3rd-party)	Hente data fra web/API direkte i Python
wget (Python-modul)	Python download modul (3rd-party)	Hente filer via HTTP/HTTPS direkte i Python
curl	Eksternt shell command (3rd-party)	Hente filer eller API-data via shell, pipes, ikke cross-platform
subprocess	Python standardbibliotek (stdlib)	Kører eksterne kommandoer (curl, wget. (*også hadoop distribueret fil system hvis data som skal extract er big data*)

- **pySpark som transform værktøj**

Emne	Forklaring
Hvad er PySpark?	**PySpark er Python-API'et til Apache Spark, som gør det muligt at bruge Spark via Python-kode**
Formål	**At arbejde effektivt med store datamængder sammen med DFS (f.ex. Hadoop), men også for normal datamængde til dataanalyse, rensning og transformation**
Kerneidé	**Behandler data i tabelform (DataFrames) på en måde, der minder om Pandas – men kan skalere**
Typiske anvendelser	**Data rensning, filtrering, transformation, aggregering og feature engineering**
DataFrame-model	**PySpark bruger DataFrames, som er strukturerede, kolonnebaserede datasæt**
Ydelse	**Hurtigere end klassisk Python ved større datasæt pga. optimering og lazy execution**
Lokalt vs. distribueret	**Kan bruges både lokalt i et Python-script og i distribuerede Spark-miljøer**
Integration	**Kan læse/skrive data fra CSV, JSON, Parquet, databaser (fx MySQL) m.m.**
Typisk brugsscenarie	**ETL-pipelines: Extract → Transform → Load**

- **Pyplot som visualiseringsværktøj**

Kategori	Beskrivelse	Eksempel / Bemærkning
Modulnavn	matplotlib.pyplot	Typisk importeres som import matplotlib.pyplot as plt
Formål	Til at lave 2D-grafik, plots og visualiseringer i Python	Bruges til scatter plots, linjeplots, histogrammer, barplots osv.
Typiske funktioner	plot(), scatter(), bar(), hist(), imshow()	F.eks. plt.scatter(x, y) til et scatterplot
Figur og akser	figure(), subplots()	fig, ax = plt.subplots() giver mulighed for flere plots i én figur
Labeling	xlabel(), ylabel(), title(), legend()	Sætter akse-navne, titel og forklaring
Styling	Farver (color), linjetype (linestyle), marker (marker)	F.eks. plt.plot(x, y, color='red', linestyle='--')
Visning	show()	Viser figuren i en grafisk vindue eller notebook
Gemme	savefig()	Gemmer figuren til fil, f.eks. plt.savefig("plot.png")
Interaktivitet	Zoom, pan, klik i notebook eller GUI	Standard i matplotlib interaktive mode (%matplotlib notebook)
Data input	Lister, NumPy arrays, pandas Series/DataFrames	Kan plotte direkte fra pandas: df.plot(x='col1', y='col2', kind='scatter')
Typiske brugsscenarier	Dataanalyse, videnskabelige plots, rapporter, dashboards	Kombineres ofte med pandas og numpy

Krav

- En flora data fil (blandt mange flora data filer) er **iris.csv** som ligger på det omtalte "blomster data fra internationale datakilder" beskrevet i problemformulering. Denne fil kan hentes fra kilden med følgende link: https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv
    - *TIPS: "**Data science**" er tit en del af abejde med data behandling. Her er et godt eksempel på, at en data sciencetist er påkrævet i arbejdet, således at du som udvikler kan få en forklaring på, hvordan du kan fortolke data. Vores data sciencetist har givet forklaring på format af data i url givet foroven, og kan læses i afsnit "**Data science**" længer nede i dette dokument.*

    - Du skal implementer **en main Python script** som implementer følgende **3 "custom" moduler** som du skal implementer og importer i din main script:

        1. Extract modul
        2. Transform modul
        3. Load modul
        4. Visualisering modul

        - **Extract modul opbygning**: indeholder **3 metoder** til at hente (extract) flora data fra datakilden:
            1. Metode 1, brug Python-bibliotk: **requests**
            2. Metode 2, brug Python- bibliotk **wget**.
            3. Metode 3, brug Python- bibliotk **subprocess**, til eksekvering af eksternt program: **curl.**

            ### Overvejelser til din implementation:

            - I hver metode skal du med kode-kommentar dokumentere:
                - Hvordan den håndterer datasikkerhed og dataintegritet (command-line injection)
                - Hvordan robusthed under download (afbrydelser under download) håndteres.
            - **Sikkerhedsvurdering:**
                - I scriptets main-funktion skal du vælge kun én af de 3 metoder til download, og skriv en kort refleksion i kommentarform om, hvorfor netop denne metode er bedst set ud fra sikkerhed og robusthed, og hvilke risici der kunne opstå ved de andre metoder.
            - **Gem det hentet data** i en csv fil i folder Input_dir. CSV filen skal indeholde passende kolonnenavne når den vises manuelt. (*Se "Data science" sektion for hvad kolonnenavne kan være*).
                - Du må ikke "hardcode" navnet på det hentet csv fil når gemmes på LFS. Det hentet csv fil skal anvend samme navn som angivet i dens download url.

        - **Transform modul opbygning**: anvender Pythons dataprocessering bibliotk **pyspark** til transformering af det hentede data. Krav til transformation er:
            - Med pyspark, transformer det original data hentet fra datakilden så **KUN** observationer med arten **Iris-setosa** er trukket ud. (*Altså rækker hvor kolonne "species" == "Iris-setosa").*

        - **Load modul opbygning**:
            - **En metode** som modtager det transformeret data som dataframe og gemmer det som en ny csv fil i folder Output_dir.
                - Det nye transformeret csv fil skal indeholde passende kolonnenavne når den vises manuelt.
                - Du må ikke "hardcode" navnet på det transformeret csv fil når gemmes på LFS. Det transformeret csv fil skal anvend samme navn som det angivet i dens download url før

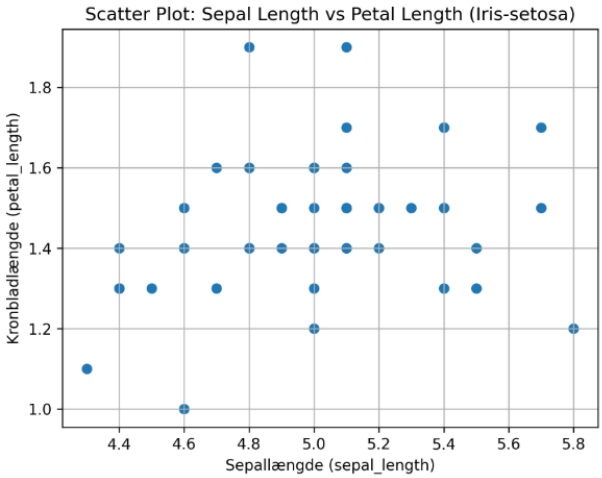
transformation, du skal dog ”append” teksten ”transform_” forand det nye navn, f.ex: **transformed_xxx.csv**.

- Hver gang app genstart -> extract, transform og Load, skal det gamle data i csv fil overskrives.

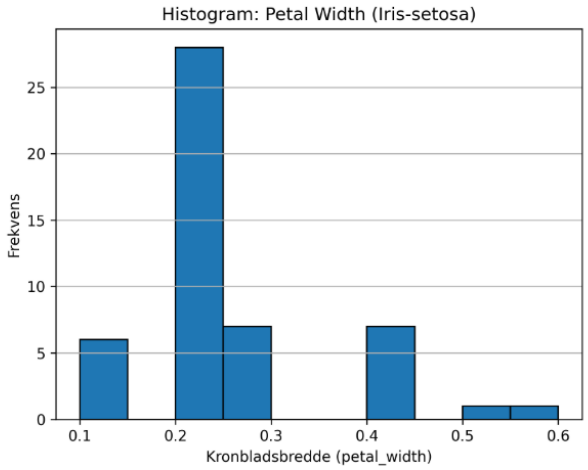
- **En metode** som modtager det transformeret data som dataframe og gemmer det som en tabel i en MySQL database.
  - Databasen hvor i tabellen oprettes, skal automatisk oprettes hvis den ikke eksister.
  - Hver gang app genstart -> extract, transform og Load, skal det gamle data i databasen overskrives.

- **Visualisering modul opbygning**:
  - Du skal ved visualisering sørg for, at navngive akser og give figuren en passende titel.
  - Der skal være 3 metoder i visualisering modulet: En til generering af en **scatter-plot**, en **histogram** og en **boxplot** diagram.
    - Hver metode skal modtag en “dataframe” objekt som parameter når kaldes, som bruges til generering af diagramet. Den som kalder metoderne i visualisering modulet sørge for at generere denne dataframe ud fra data gemt i databasen.

    - **Scatter-plot eksempel**: Visualisere sammenhængen mellem to målinger: sepal_length på x-aksen og petal_length på y-aksen. Eksempel på en scatter-plot:



- **Histogram eksempel:** over kolonnen petal_width med ca. 10 søjler ("bins"). Eksempel på en histogram:



- **Boxplots eksempel:** Et 2×2 layout med fire **boxplots**:
  - sepal_length
  - sepal_width
  - petal_length
  - petal_width

Hvordan bliver du bedømmet

- Fremvis din Python script hvor:
  - Alle 3 extract-metoder virker korrekt og downloader filen
    - Du skal vise hvordan du i din kode, løser punkterne under ” Overvejelser til din implementation” for kørslen.
    - Transform data skal virke korrekt efter det stillede krav.
      - I csv (Kravet om at genbrug csv filnavn ude fra det original filnavn fra download url samt generering af input/output folder til lagering af filen, samt kolonnenavne til filerne er alene for dig at demonstrer din programmeringsevne.)
    - Load data skal være korrekt efter det stillede krav.
      - I databasen (Kravet om at autogenerer dabasen hvis den ikke eksister, samt overskrivning af eksister tabel ved hver kørsel er alene for dig at demonstrer din programmeringsevne.)
      - Data fra databasen kan bruges til visualisering som **scatter**-, **histogram** og **boxplot** grafer.

Data science

In the field of flora following mesurment terms exist "sepal_length", "sepal_width", "petal_length" and "petal_width". (**På dansk: Sepal = bægerblad, og Petal = kronblad**)

Those terms are **botanical measurements** commonly used in plant (flora) datasets, especially the classic **Iris flower dataset**. They refer to the sizes of two main flower structures:

What they mean
Sepal – the outer parts of the flower (like protective leaf-like structures).
Petal – the inner, typically colorful parts of the flower.
So, the variables mean:

- **sepal_length** – the length of a flower's sepal
- **sepal_width** – the width of a flower's sepal
- **petal_length** – the length of a flower's petal
- **petal_width** – the width of a flower's petal

These measurements are usually recorded in **centimeters** and used to distinguish between species. When representing flora data as a record, these measurements are typically represented as 6.3,3.3,6.0,2.5, Iris-virginica ➔ **sepal_length, sepal_width, petal_length, petal_width**, **species.**

When to Use hdfs vs. subprocess

Task / Situation	Use hdfs module	Use subprocess (HDFS CLI)	Reason
Uploading/downloading files	Yes	Possible but unnecessary	WebHDFS supports this cleanly and safely
Listing files / directories	Yes	Yes	Both work, but Python API is cleaner
Reading/writing file contents	Yes	No	CLI can't stream content in the same way
Checking file existence	Yes	Yes	Avoid parsing CLI output when Python API handles it
Managing permissions (chmod, chown, ACLs)	No	Yes	WebHDFS doesn't support full permission features
Checking file sizes, usage, quotas	Limited	Yes	CLI offers richer filesystem metadata tools
Running MapReduce (hadoop jar, mapred job)	No	Yes	Only CLI can run jobs
Running Yarn commands	No	Yes	Only CLI supports Yarn administration
Running HDFS admin commands (safemode, balancer)	No	Yes	Not exposed via WebHDFS
Automating HDFS workflows in Python	Yes	Yes	Prefer Python unless CLI is required
High-performance / parallel file operations	Yes	No	CLI is slower because it spawns new processes
Clean, maintainable production code	Yes	Avoid	Python API is more stable and testable
Compatibility (no WebHDFS enabled)	No	Yes	CLI always works, WebHDFS may be off
Teaching / learning differences	Yes	Yes	Useful to compare both approaches