Intro til Databehandling i Python

Opgavebeskrivelse

I denne uge skal I arbejde med 4 delopgaver, der giver jer en introduktion til grundlæggende databehandling i Python – herunder biblioteket *Pandas*. Evnen til at arbejde med data er i dag en eftertragtet kompetence – både i studier og på arbejdsmarkedet. Når man kan bruge data aktivt, bliver det nemmere at tage velinformerede beslutninger og identificere mønstre eller sammenhænge, som ikke nødvendigvis er synlige ved første øjekast.

Hvis du allerede har erfaring med nogle af værktøjerne, kan du bruge opgaverne som en god lejlighed til at genopfriske din viden og få et mere sikkert greb om de grundlæggende principper.

Før du starter anbefaler vi at læse følgende teoridokumenter: ***Python tips***, ***almene programmeringsprincipper***, ***filhåndtering***

Afleveringsformat

Afleveringen skal bestå af:

- Link til et Github-repository, der indeholder følgende:
    - Mapper tilknyttet hver delopgave ("Delopgave_1", "Delopgave_2" …)
    - En README.md fil, der kort beskriver
        - Hvordan man kører jeres kode
        - Hvilke packages I bruger
        - Hvilke delopgaver der bliver løst i de forskellige filer/mapper
        - En kort beskrivelse af hvilke elementer i opgaverne du har haft udfordringer med.

Delopgaver og opmærksomhedspunkter

OBS: Husk at lave udførlige kommentarer – vi ser hellere, at I arbejder i bund med de enkelte opgaver, end at I skynder jer gennem dem alle.

Delopgave 1: Intro til Python

Filen "Navneliste.txt" indeholder en lang række navne, som du skal sortere og derefter udskrive. Først skal navnene sorteres alfabetisk, og derefter efter længde. Til sidst skal du oprette et Python-dictionary, der tæller forekomsten af bogstaver i alle navnene

Tips til fremgangsmåde:

- Brug open() og .readlines() eller .splitlines() for at læse filen ind som en liste af navne
- Fjern evt. linjeskift eller mellemrum med .strip()
- sorted() returnerer en ny sorteret liste – du kan bruge key=-parameteren til at styre sorteringen (f.eks. key=len)
- Brug .lower() hvis store og små bogstaver skal behandles ens
- Til bogstavoptælling kan du bruge to for-løkker:
    - én over navnene
    - én over hvert bogstav i navnet
- Brug enten et almindeligt dictionary med dict.get(bogstav, 0) eller collections.defaultdict(int) for at forenkle optællingen

Ressourcer

- Real Python - Python Data Structures Tutorial: https://realpython.com/python-data-structures
- Pluralsight: Core Python: Getting Started

Avanceret udvidelse til delopgave 1:

Hvis du hurtigt bliver færdig med delopgave 1, kan du udfordre dig selv yderligere med denne udvidelse af opgaven – eller fortsætte direkte til delopgave 2.

Avanceret dataanalyse og visualisering gør det muligt at forstå komplekse datasæt på en mere intuitiv og dybdegående måde. Det kan afsløre skjulte mønstre, tendenser og sammenhænge, som en mere simpel analyse ikke viser. Visualiseringer gør det desuden lettere at formidle indsigter – også til beslutningstagere uden teknisk baggrund.

Byg videre på delopgave 1 ved at udføre en mere detaljeret analyse af navnedataene. Når du har talt forekomsten af hvert bogstav i navnene, kan du fortsætte med følgende::

- **Frekvensanalyse**: Beregn og visualisér hyppigheden af hvert bogstav i navnelisten. Brug f.eks. biblioteket *matplotlib* eller *seaborn* til at lave et histogram eller søjlediagram, der viser de mest almindelige bogstaver
- **Ordsky**: Generér en *wordcloud* baseret på, hvor ofte hvert bogstav forekommer. Jo oftere et bogstav optræder, desto større skal det vises. Du kan bruge *wordcloud*-biblioteket i Python til dette.
- **Navnelængde Analyse**: Undersøg fordelingen af navnelængder i datasættet. Beregn gennemsnit og median, og visualisér resultaterne med passende diagrammer.
- **Filtrering af duplikater:** Prøv at fjerne dubletter fra navnelisten, og gentag analysen. Sammenlign resultaterne før og efter filtreringen.

Ressourcer til avanceret udvidelse

- **Matplotlib-dokumentation** – til grundlæggende og avancerede plots: https://matplotlib.org
- **Seaborn-dokumentation** – til statistisk datavisualisering: https://seaborn.pydata.org
- **WordCloud-dokumentation** – til generering af ordskyer: https://github.com/amueller/word_cloud

Delopgave 2: Logfil-analyse

I denne opgave skal du arbejde med automatiseret loganalyse i Python. Det gør det hurtigt lettere at identificere f.eks. kritiske hændelser, uden at skulle gennemgå hele filen manuelt. Den samme metode kan også bruges i mange andre sammenhænge, hvor man skal udtrække relevant information fra store tekstbaserede datakilder.

Udvikl et Python-script, der automatisk analyserer indholdet af logfilen “app_log.txt” ved at filtrere bestemte typer af logmeddelelser – som for eksempel ERROR og WARNING – og gemmer dem i en ny fil, så kritiske hændelser bliver lettere at få overblik over.

Tips til fremgangsmåde:

- Importér os.path.join for at håndtere filstier på tværs af styresystemer. Dette bibliotek hjælper med at sammensætte stier på tværs af styresystemer, så du undgår problemer med forskellige filsti-delimitere (f.eks. / på Mac/Linux og \ på Windows).
- Åbn en ny fil for hver type logbesked, du vil gemme separat – f.eks. én til ERROR og én til WARNING. Filerne behøver ikke at eksistere på forhånd – Python opretter dem automatisk, når du åbner dem i skrivetilstand.
- Åbn herefter den originale logfil, som indeholder alle logbeskeder. Du kan evt. printe indholdet i terminalen for at sikre, at filen er åbnet korrekt.
- Gennemgå filen linje for linje. Undersøg hvilken type besked hver linje indeholder, og skriv linjen til den tilsvarende fil.
- Husk at lukke alle de filer, du har åbnet, når du er færdig.
- Tjek om outputfilerne indeholder det forventede indhold.

Ekstra udfordring til delopgave 2:

Implementér en løsning, der håndterer filåbning og -lukning korrekt og filtrerer logbeskeder effektivt baseret på deres type. Din løsning skal være enkel, men robust, og gerne kunne genbruges med forskellige logfiler. Test gerne dit script med andre logfiler for at sikre, at det fungerer fleksibelt og fejltolerant.

Delopgave 3: Fejlhåndtering ved datamigrering

I denne opgave skal du arbejde med fejlhåndtering i Python ved at lave et script, der kan håndtere uventede situationer som manglende filer, skrivefejl eller forkert dataformat – uden at programmet går ned. Fejlhåndtering er en vigtig del af professionel softwareudvikling, fordi det gør dine scripts mere robuste og brugervenlige. Samtidig opbygger du erfaring med avanceret filhåndtering og tydelig kommunikation med brugeren, hvis noget går galt.

Udvikl et Python-script, der læser data fra kildefilen (source_data.csv), og gemmer det i en destinationsfil. Undervejs skal scriptet kunne håndtere og reagere på forskellige typer fejl, fx:

- **Manglende fil:** Hvad sker der, hvis source_data.csv ikke findes?
- **Formateringsfejl:** Hvad hvis dataene ikke har det forventede format?

- **Skrivefejl:** Hvad hvis destinationsfilen er skrivebeskyttet, eller der mangler tilladelser?

Dit script skal ikke blot "gå ned", men i stedet “fange” fejlene og give brugeren en tydelig besked om, hvad der er sket – og gerne, hvordan det kan løses.

Tips til fremgangsmåde:

- Brug try-except blokke til at fange fejl og sikre robusthed.
- Start med at læse filen med open(). Hvis filen ikke findes, skal du give brugeren en klar besked.
- Prøv at læse og evt. parse indholdet af filen (f.eks. splitte linjer eller felter). Brug try-except til at fange fejl i denne proces.
- Når du skriver til destinationsfilen, skal du være opmærksom på mulige adgangsfejl.
- Tilføj print-statements eller fejlmeddelelser, der hjælper brugeren med at forstå, hvad der gik galt.

Ekstra udfordring til delopgave 3:

Sørg for, at dit script ikke bare fanger kendte fejl, men også har en robust undtagelseshåndtering der behandler uventede situationer på en kontrolleret måde. Det skal give tydelige og hjælpsomme beskeder til brugeren, uden at programmet går ned. Overvej også at udvide scriptet med mulighed for at angive egne filnavne via input eller som argumenter.

Delopgave 4: Introduktion til Pandas

Denne opgave introducerer brugen af Pandas – et bibliotek til struktureret databehandling i Python. Data organiseres i såkaldte DataFrames, som gør det nemt at analysere, gruppere og opsummere information. I kombination med visualisering kan man hurtigt få overblik over store datasæt, og giver jer mulighed for at eksperimentere og gå på opdagelse i dataen.

- Start med at læse datafilen (DKHousingPricesSample100k.csv) ind i et Pandas DataFrame og print de første 10 linjer
- Brug *Groupby*-funktionen til at sortere kolonner efter grupper:
    - Lav *groupby* på “region”

    - Lav et gennemsnit over “purchases”
    - Lav *groupby* på “house_type”
    - Plot det ved hjælp af Matplotlib-modulet og print resultatet
- Eksperimentér med *groupby* og *average* over forskellige kolonner, og plot resultaterne i Matplotlib

Ressourcer

- **Real Python –** Functional Programming in Python: https://realpython.com/courses/functional-programming-python
- **Faker** – Dokumentation: https://faker.readthedocs.io/en/master
- **Pluralsight** – Functional Programming in Python 3: https://www.pluralsight.com/courses/python-3-functional-programming