Del 1: Teoretiska frågor:

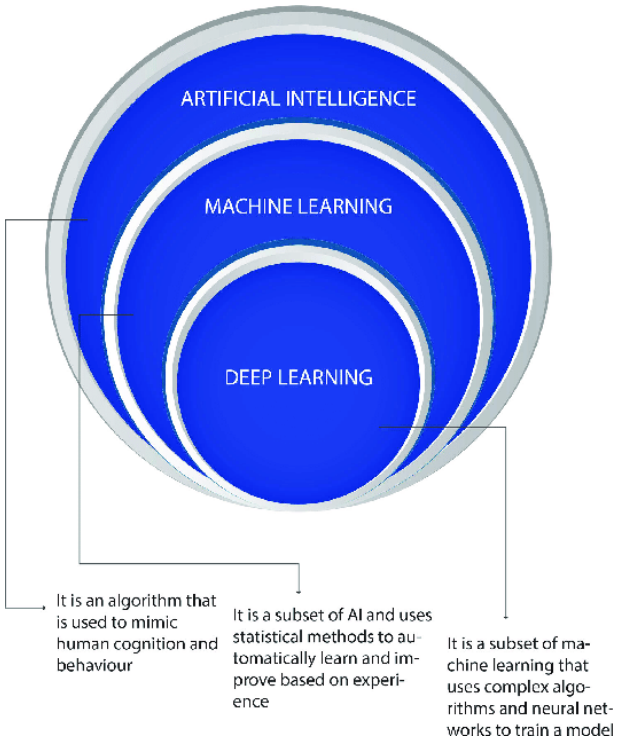
1. Hur är AI, Maskininlärning och Deep Learning relaterat?

AI är det breda området där datorer försöker efterlikna mänsklig intelligens, alltså lösa problem, lära sig och fatta beslut.

Maskininlärning är en del av AI där datorer lär sig saker själva genom att hitta mönster i data (utan att man skriver exakt hur de ska göra).

Deep Learning är en speciell typ av maskininlärning där man använder neurala nätverk med många lager. Deep learning används ofta till bildigenkänning, taligenkänning och mycket mer.

Enkelt som beskrivs i bilden =>



2. Hur är TensorFlow och Keras relaterat?

TensorFlow är som motorn i en bil stark, kraftfull och kan göra nästan vad som helst inom maskininlärning. Men den är också ganska teknisk och kräver ofta en hel del kod för att komma i gång.

Keras, däremot, är som bilens instrumentbräda eller ett enkelt användargränssnitt. Med Keras kan man snabbt och smidigt bygga och träna neurala nätverk utan att behöva tänka på allt det krångliga som händer under huven.

Relationen mellan dem är alltså att Keras är det användarvänliga lagret ovanpå TensorFlow. När man bygger en modell i Keras, översätts det automatiskt till TensorFlow-kod i bakgrunden. Idag är Keras dessutom helt integrerat i TensorFlow (man använder *tf.keras*), vilket gör det ännu smidigare. Man får det bästa av två världar: enkelheten från Keras och kraften från TensorFlow.

3. Vad är en parameter? Vad är en hyperparameter?

En parameter är något som modellen själv lär sig under träningen. Till exempel, i ett neuralt nätverk är vikterna och biasar (dvs. själva värdena i noderna och kopplingarna) exempel på parametrar. De justeras automatiskt så att modellen blir bättre och bättre på sin uppgift.

En hyperparameter är däremot något som du som utvecklare bestämmer **innan** träningen börjar. Det är alltså inte något modellen lär sig själv. Exempel på hyperparametrar är hur många lager ditt nätverk ska ha, hur många noder det ska vara i varje lager, vilket "lärande" (learning rate) modellen ska ha, hur stor batch-storleken är, och hur många varv (epochs) den ska träna. Hyperparametrarna bestämmer alltså mer "ramverket" för hur modellen ska tränas.

Kortfattat:

- **Parameter = lärs av modellen själv under träning**
- **Hyperparameter = bestäms av dig innan träning**

```
# hyperparametrar:
chunks = chunk_text(text_to_analyze, chunk_size=700)   # 'chunk_size' är en hyperparameter
embedded_chunks = []
for chunk in chunks:
    embedding = get_embedding(chunk, model="text-embedding-3-small")  # modellnamnet är en hyperparameter
    embedded_chunks.append({"text": chunk, "embedding": embedding})
```

Hyperparametrar styrs i koden (till exempel chunk-size, modellval, antal top-k chunks för retrieval).

Parametrarna är dolda och inbyggda i den underliggande modellen (GPT:s vikter, embedding weights etc.).

4. Tränings-, valideringss- och testdataset – vad är skillnaden?

Träningsdata
Används för att *lära* modellen. Det är med data, modellen ser samband och "hittar regler".

Valideringsdata
Används för att *justera* modellen. Efter varje träningsomgång testar man modellen på valideringsdata för att se hur bra den presterar på *osett* data. Här används resultaten till att t.ex. välja rätt hyperparametrar eller för att undvika överanpassning.

Testdata
Används för att *utvärdera* modellen slutgiltigt. Testdatan har modellen aldrig sett tidigare. Här mäter man den verkliga prestandan, som ett "facit".

Kortfattat:

- Träning = modellen lär sig

- Validering = vi justerar och utvärderar under utvecklingen
- Test = vi mäter hur bra modellen faktiskt blev, på riktigt nya data

5. Förklara vad nedanstående kod gör:

Bygger, kompilerar och tränar ett neuralt nätverk för binär klassificering.

Modellen lär sig att skilja mellan två klasser (t.ex. ja/nej, sant/falskt) utifrån träningsdata.

Syftet är: skapa och träna en maskininlärningsmodell (ett neuralt nätverk) som automatiskt kan förutsäga vilket av två möjliga utfall (klasser) ett visst indataexempel tillhör t.ex. om ett mejl är spam eller inte, eller om en kund kommer köpa eller inte.

Koden används alltså för att lösa binära klassificeringsproblem med hjälp av ett neuralt nätverk.

```
n_cols = x_train.shape[1]  # Antal kolumner (features) i träningsdatan

nn_model = Sequential()  # Skapar en sekventiell modell

# Första dolda lagret, 100 noder, ReLU-aktivering, input shape enligt antal kolumner
nn_model.add(Dense(100, activation='relu', input_shape=(n_cols, )))

# Dropout-lager, stänger av 20 % av noderna vid varje träning för att minska överanpassning
nn_model.add(Dropout(rate=0.2))

# Andra dolda lagret, 50 noder, ReLU-aktivering
nn_model.add(Dense(50, activation='relu'))

# Ut-lager, 1 nod, sigmoid-aktivering (för binär klassificering)
nn_model.add(Dense(1, activation='sigmoid'))

# Kompilerar modellen med Adam-optimerare och binär korsentropi som förlustfunktion
nn_model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)

# Stoppar träningen om valideringsresultatet inte förbättras på 5 epoker
early_stopping_monitor = EarlyStopping(patience=5)

# Tränar modellen på x_train och y_train, 20% valideringsdata, max 100 epoker
nn_model.fit(
    x_train,
    y_train,
    validation_split=0.2,
    epochs=100,
    callbacks=[early_stopping_monitor]
)
```

6. Vad är syftet med att regularisera en modell?

Syftet med att regularisera en modell är att minska risken för överanpassning (overfitting). Regularisering hjälper modellen att inte bli för bra på träningsdatan alltså att inte lära sig brus och oviktiga detaljer utan i stället prestera bättre på nya, osedda data.
Det gör att modellen kan ge mer tillförlitliga och korrekta förutsägelser på nya data.

7. ”Dropout” är en regulariseringsteknik, vad är det för något?

Dropout är en regulariseringsteknik som används i neurala nätverk för att motverka överanpassning.

Vid träning stängs (”släcks”) slumpmässigt en andel av noderna i nätverket av i varje träningsomgång.

Med andra ord Dropout innebär att man under varje träningsrunda slumpmässigt tar bort eller hoppar över ett antal neuroner i nätverket.
Det gör att modellen inte kan lita på samma vägar hela tiden, utan måste lära sig flera olika sätt att lösa uppgiften.
På så sätt minskar risken att modellen fastnar i att bara minnas träningsdatan, och i stället blir bättre på att hantera nya data.

8. ”Early stopping” är en regulariseringsteknik, vad är det för något?

Early stopping innebär att man slutar träna modellen så fort den presterar som bäst på valideringsdatan, i stället för att köra alla epoker.
Man övervakar modellens resultat och avbryter träningen när det inte längre blir någon förbättring.
Detta hjälper till att undvika att modellen börjar lära sig onödiga detaljer i träningsdatan och ger bättre resultat på nya data.

Det sker dock inte automatiskt, utan man måste själv aktivera *early stopping* i koden till exempel genom att använda EarlyStopping i Keras och ställa in hur länge modellen får fortsätta utan förbättring innan träningen stoppas.

9. Din kollega frågar dig vilken typ av neuralt nätverk som är populärt för bildanalys,

vad svarar du?

Konvolutionella neurala nätverk (CNN, Convolutional Neural Networks) är den populäraste typen av neurala nätverk för bildanalys.
De är särskilt bra på att hitta mönster och detaljer i bilder, som kanter, former och strukturer.
Därför används de ofta i till exempel bildigenkänning, ansiktsigenkänning och objektidentifiering.

Här är några verkliga, storskaliga projekt där CNN-modeller används:

Googles bildsök (Google Images):
Använder avancerade CNN-modeller för att automatiskt känna igen och kategorisera miljarder bilder.

Självkörande bilar (t.ex. Tesla, Waymo):
Fordonens kameror använder CNN-modeller för att känna igen vägskyltar, gångtrafikanter, andra bilar och faror.

Satellitbildsanalys:
CNN-modeller hjälper till att upptäcka förändringar i miljön, skogsskövling, urbanisering och till och med identifiera fordon eller fartyg från rymden.

10. Förklara översiktligt hur ett ”Convolutional Neural Network” fungerar.

Ett Convolutional Neural Network (CNN) fungerar ungefär som människans syn:
Det analyserar bilder steg för steg, från enkla detaljer till hela motiv.

1. Först hittar det enkla saker, som kanter och färgskiftningar, genom att låta små filter “söka igenom” bilden.

2. Sen kombinerar det ihop detaljerna till större mönster, till exempel cirklar, hörn eller texturer.
3. Till slut pusslar nätverket ihop allt för att förstå hela bilden och avgöra vad den föreställer.

CNN gör att datorn kan se och förstå bilder på egen hand, likt hur vi människor tolkar det vi ser.

11. Vad gör nedanstående kod?

```
model.save("model_file.keras")
my_model = load_model("model_file.keras")
```

model.save("*model_file.keras*") sparar hela den tränade modellen till en fil på datorn.

my_model = load_model("model_file.keras") läser in (laddar) modellen igen från filen, så att man kan använda den senare utan att behöva träna om den.

12. Deep Learning modeller kan ta lång tid att träna, då kan GPU via t.ex. Google Colab skynda på träningen avsevärt. Skriv mycket kortfattat vad CPU och GPU är.

CPU (Central Processing Unit): Datorns huvudsakliga "hjärna" som utför de flesta beräkningar, allround och bra på många olika typer av uppgifter.

GPU (Graphics Processing Unit): Specialiserad på att snabbt hantera många beräkningar parallellt, särskilt användbar för bildbehandling och träning av deep learning-modeller.

från början utvecklades GPU:er (grafikkort) för att hantera grafik och spel (gaming), eftersom de är väldigt bra på att bearbeta många bildpunkter (pixlar) samtidigt.

Men idag används GPU:er också mycket inom AI och deep learning, eftersom samma typ av parallella beräkningar som behövs för spelgrafik också är perfekta för att träna stora neurala nätverk snabbt.

Rapport – AI-baserad Rapportanalys med GPT och RAG

Min Streamlit-app analyserar och sammanfattar finansiella rapporter automatiskt med hjälp av AI, baserat på OpenAI:s GPT-modeller och RAG-teknik.
Appen kan hantera rapporter i flera format, extrahera text, dela upp innehållet och använda GPT för att besvara frågor eller generera sammanfattningar.
Ett inbyggt evalueringssystem, RAGAS, bedömer dessutom kvaliteten och tillförlitligheten i AIs svar direkt i appen.
Syftet är att spara tid för användare som snabbt vill få insikter ur omfattande dokument till exempel investerare eller studenter.

Appens struktur och moduler

core/

- chunking.py: Textdelning/chunkning
- embedding_utils.py: Hantering av embedding
- file_processing.py: Filhantering & läsning
- gpt_logic.py: GPT-relaterad logik
- __init__.py: Initieringsfil

data/

- outputs/: Sparade resultat
- uploads/: Uppladdade filer

services/

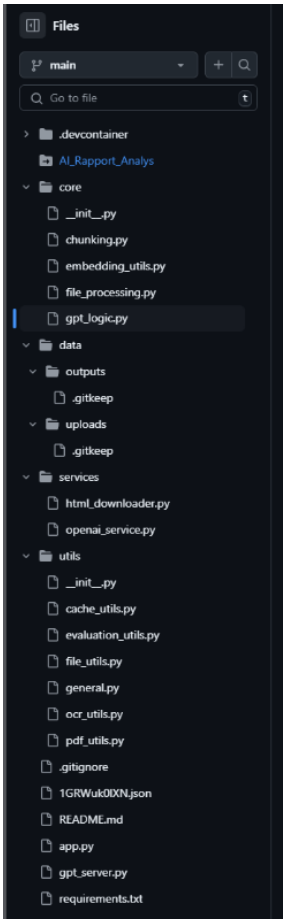
- html_downloader.py: Hämtar HTML-data
- openai_service.py: OpenAI-integration

utils/

- cache_utils.py: Caching-funktioner
- evaluation_utils.py: Utvärderingsverktyg
- file_utils.py: Extra filhantering
- general.py: Diverse hjälpfunktioner
- ocr_utils.py: OCR-funktioner
- pdf_utils.py: PDF-funktioner
- __init__.py: Initieringsfil

Övrigt

- README.md: Dokumentation
- app.py: Huvudapp/Streamlit
- requirements.txt: Krav för installation



Kritisk diskussion: Användning av AI Rapportanalys i verkligheten

Min AI Rapportanalys-modell, som bygger på GPT och RAG, har stor potential i praktisk användning särskilt för att snabbt analysera och sammanfatta finansiella rapporter. Tjänsten kan spara tid och öka effektiviteten för investerare, ekonomer och företag som annars lägger mycket resurser på manuell rapportanalys.

Det finns även affärsmöjligheter att erbjuda detta som en SaaS-tjänst (Software as a service) där användare enkelt kan ladda upp egna dokument och direkt få ut sammanfattningar, nyckeltal och analyser.

Möjligheter:

Sparar tid för användare genom att automatiskt hitta viktiga siffror och fakta i stora rapporter.

Kan vidareutvecklas för fler områden, exempelvis juridik eller utbildning, genom anpassning av kod och promptar.

Ger objektiv och snabb åtkomst till information, vilket kan minska risken för mänskliga missar.

Utmaningar:

Domänbegränsning: Modellen är anpassad för finansiella texter. För att tolka andra dokument krävs både ny utveckling och omfattande tester.

Risk för fel: AI kan ibland missa detaljer, misstolka siffror eller ge felaktiga svar, vilket är kritiskt i finansiella beslut.

Etik och ansvar: Om modellen ger fel svar som leder till ekonomisk skada vem bär ansvaret? Dessutom krävs dataskydd, särskilt vid hantering av känsliga eller konfidentiella rapporter.

Datakvalitet: Dålig kvalitet på uppladdade filer (t.ex. bilder, oläsliga tabeller) kan påverka analysen negativt.

Konkurrens och teknik: Det finns redan liknande tjänster på marknaden. Tjänsten kräver kontinuerligt underhåll, rätt API-nycklar och teknisk kapacitet för att hantera tillväxt.

AI Rapportanalys är ett kraftfullt stöd för finansiell rapportanalys, men kräver medvetenhet om både teknikens möjligheter och dess begränsningar. För att skapa verklig nytta måste man väga in affärsmässiga, etiska och tekniska aspekter och alltid vara uppmärksam på datakvalitet, ansvarsfrågor och behovet av fortlöpande utveckling.

Självutvärdering

1. Vad har varit roligast i kunskapskontrollen?
Det roligaste har varit att bygga appen och se hur AI:n kan analysera och svara på frågor om riktiga rapporter.

2. Vilket betyg anser du att du ska ha och varför?
Jag tycker att jag förtjänar VG eftersom jag har löst uppgiften, byggt en fungerande app och lagt ner tid på att strukturera koden så professionellt som möjligt.

3. Vad har varit mest utmanande i arbetet och hur har du hanterat det?
Det mest utmanande har varit att få tekniken att fungera med olika rapportformat och att tolka AI's svar. Jag har löst det genom att läsa dokumentation, testa mig fram och använda AI.