

Szoftverfejlesztés MI támogatással – Házi feladat

Árvai Péter

O38B1K

Budapest, 2025.12.02

BEVEZETÉS

Jelen projekt keretében egy gépi tanuláson alapuló csapadék-előrejelző rendszert valósítunk meg, amely a GeeksforGeeks „Rainfall Prediction using Machine Learning – Python” feladatára vonatkozó módszertanát tekinti referenciajának. A fejlesztés különlegessége a kísérleti megközelítés: a teljes implementációt kizárolag általános célú társalgási mesterséges intelligencia modellek (ChatGPT-5, Microsoft Copilot GPT-5) instrukciói alapján végezzük, szándékosan mellőzve a dedikált kódoló asszisztensek (coding assistants) használatát.

A rendszer architektúrája a Python backend mellett magában foglalja a Flask és HTML alapú frontend felület teljes körű tervezését és implementációját is. Az így létrejött alkalmazás meteorológiai adatok alapján, több algoritmus (pl. Logisztikus Rögzítési, Random Forest, Döntési Fák, SVM) összehasonlító elemzésével becsüli meg a várható csapadékmennyiséget, kiválasztva a legjobb modellt. A fejlesztés lefedi a teljes szoftver-életciklust az adattisztítástól és modellértékeléstől kezdve a felhasználói felület designjáig és a szerveroldali logikáig.

A kutatás elsődleges célja annak vizsgálata, hogy a széles körben elérhető, „trendi” AI modellek mennyire képesek komplex gépi tanulási és full-stack webfejlesztési feladatok önálló, vagy kis mértékben segített megoldására. A projekt során elemezzük, hogy a generált megoldások miben térnek el a referencia-dokumentációtól, mennyire képesek az algoritmusok optimalizálására, illetve milyen hatékonysággal működnek együtt fejlesztőpartnerként egy szakmai alapanyag biztosítása esetén.

A dokumentum a BME AUT Tanszék által kiadott Házi feladat követelmények alapján készült. A feladatkiírás és felhasznált előadásanyag a szerző és a BME AUT Tanszék tulajdonát képezik.

A dokumentumban található saját (nem AI generált) kódok és a teljes elemzések készítőjük, azaz Árvai Péter szellemi tulajdonát képezik.

A dokumentum egészének vagy részleteinek engedély nélküli másolása és felhasználása **szigorúan tilos**.

MACHINE LEARNING

A gépi tanulási modell iteratív fejlesztésének bemutatása mesterséges intelligenciával

A kutatásom során iteratív fejlesztési módszertant alkalmaztunk a prediktív teljesítmény maximalizálása érdekében. A fejlesztés három iterációban történt, melyben az első saját kódolás, a másik kettő GPT modellek módosításait alkalmazza. A modell komplexitását, az adatfeldolgozás módját és az alkalmazott algoritmusokat folyamatosan finomította a GPT modell a kapott kiértékelési eredmények alapján. A cél minél nagyobb pontosság elérése volt, ennek egy általam megadott **85 %**-ot állítottam be, az adathalmaz jellege miatt.

1. iteráció: A referencia modell (Baseline) létrehozása

Implementáció: train_model.py

A fejlesztés első lépéseként egy alapértelmezett (baseline) modellt hoztam létre a teljesítmény összehasonlíthatósága érdekében. Ez a legegyszerűbb eset volt.

- **Algoritmus:** Logisztikus Rögzítés.
- **Adatafelkészítés:** A modell a nyers meteorológiai mérési adatokat (hőmérséklet, légnyomás, szélsebesség stb.) használta fel, minimális technikai tisztítást követően. Nem történt sem változószel válogatás, sem transzformáció.
- **Kiértékelés:** Bár a modell működőképesnek bizonyult, a logisztikus regresszió lineáris természete korlátozta a prediktív pontosságot. Az algoritmus nem volt képes hatékonyan feltárnivalni a meteorológiai adatokban rejlő komplex, nem-lineáris összefüggéseket.
- **Azonosított hiányosságok:** A domén-specifikus összefüggések figyelmen kívül hagyása, valamint az adatok skálázásának hiánya, amely a változók eltérő nagyságrendje (pl. légnyomás vs. csapadék) miatt torzította az eredményeket. A **pontosság 72,6 %** volt.

2. iteráció: Ensemble módszerek és hiperparaméter-optimalizáció

Implementáció: train_modelV2.py

A második fázisban a modell robosztusságának növelése és a nem-lineáris kapcsolatok modellezése volt a cél.

Légyél machine learning expert! Itt Van egy train model aminek a pontossága 72.60% Az adatsorunk adott a modell egy model.pkl fájlba ment. sklearn-nel dolgozunk. Javítani szeretném a pontosságát 85% felé. Tegyél javaslatokat



Alaposan átnéztem a kódodat és az adatfeldolgozást. A jelenlegi modell LogisticRegression, amely egy lineáris osztályozó. Az adatok nagy valószínűséggel nem lineárisak, ezért a 72.6% körüli pontosság teljesen tipikus.

Az alábbiakban konkrét, rövid távon megvalósítható lépések következnek, amelyekkel **biztosan 85% fölé viheted a pontosságot**.

- **Algoritmus:** Random Forest Classifier. Ez az ensemble módszer döntési fák sokaságát alkalmazza, ami elméletileg jobb általánosító képességet biztosít.

1. Skálázd az adatokat (StandardScaler)

A logisztikus regresszió nagyon érzékeny a különböző skálázú feature-ökre. Ez önmagában +5–10% javulást hozhat.

2. Használj erősebb modelleket

A legnagyobb ugrást ez adja. Javaslatok:

Random Forest

Nagyon jól működik tabuláris adatokon.

Gradient Boosting (XGBoost / LightGBM / CatBoost)

Ezek szinte biztosan 85–95% körüli pontosságot hoznak.

3. Hyperparameter tuning (GridSearchCV)

Ez finomhangolja a modellt.

- **Technikai fejlesztések az AI által:**

- **Algoritmus-váltás:** A döntési fák alkalmazása lehetővé tette a nem-lineáris összefüggések tanulását.
- **Hiperparaméter-hangolás:** A GridSearchCV eljárás bevezetésével automatizáltuk az optimális paraméterek (pl. fák száma, maximális mélység) keresését.
- **Adattisztítás:** Szigorúbb validációs szabályokat vezettünk be (pl. string-ek tisztítása, kis- és nagybetűk egységesítése).

- **Kiértékelés:** A modell működése stabilabbá vált, azonban a rendelkezésre álló adathalmaz alacsony elemszáma miatt fellépett az overfitting jelensége. A modell kiválóan

teljesített a tanító adatokon, de gyengébben generalizált az ismeretlen tesztadatokra. Érdekes továbbá, hogy a pontosság csak 79.73 % volt, pedig az MI szerint ez már 85 és 95 % közötti érték lesz. Ezen az adathalmazon a Feature Engineering egy jó optimalizáció lehet, ebben az iterációban ezt azonban csak megemlíttette, nem implementálta.

4. Feature engineering ötletek (nagy hatás!)

A datasetben:

- pressure
- maxtemp
- mintemp
- humidity
- cloud
- windspeed
- dewpoint
- sunshine

Javaslatok:

1. Hőingás ($\text{maxtemp} - \text{mintemp}$)
2. Vapour pressure deficit ($\text{maxtemp} - \text{dewpoint}$)
3. Relative humidity nonlinear interactions (humidity^2)
4. Wind chill index
5. Temp anomalies ($\text{actual} - \text{seasonal mean}$), ha van időadat

Ezekkel ellenőrizhetően 5–15% pontosságnövekedés jön.

3. iteráció: Végeleges optimalizáció (SVC és Feature Engineering)

Implementáció: train_model_V3.py

A végső iterációban a hangsúlyt az adatminőség javítására (Feature Engineering) és a kis adathalmazokon is hatékony algoritmus kiválasztására helyezte. Ezt már korábban felvetette, így egy konkrét promthoz folyamodtam, ami ezt elő tudja segíteni:

A tesztek alapján a jelenlegi Random Forest modell valószínűleg nem a legjobb választás ehhez a konkrét adathalmazhoz, mert kis adaton hajlamos a túltanulásra (vagy épp nem találja meg az egyszerű összefüggéseket).

85%-os pontosságot szeretnék.

SVM használata? Jobb lehet mint a random forest. Lásd, az SVC jobban meg tudja húzni a határvonalat az "Eső" és "Nem eső" között, ha előtte skálázzuk az adatokat.

Feature engineeringgel pl.

A meglévő oszlopokból érdemes újakat számolni, amik jobban korrelálnak az esővel:

Hőingadozás (temp_range): maxtemp - mintemp.

Harmatpont-különbség (dew_depression): temperature - dewpoint. Ez fizikailag is szorosan kötődik a csapadékképződéshez (ha 0 közeli, telített a levegő -> eső várható).

Adatok skálázásával?

A nyomás 1000 körüli érték, a napsütés meg 0-10 közötti. Ez megzavarhat bizonyos algoritmusokat!!

Mi lehet a megoldás?

Az 85% eléréséhez teljesen jó irány az, amit írász:

→ **SVM (SVC)** + scaling + feature engineering

Ez a három együtt *nagyon erős*, kis és közepes adathalmazokon tipikusan jobb, mint a Random Forest.

Az adataid ráadásul **folytonos fizikai mérések**, ahol az SVM valóban jól tud döntési határvonalat húzni – főleg RBF kernellel.



Algoritmus: Support Vector Machine (SVC - Support Vector Classifier).

- **Feature Engineering (Jellemzők tervezése):** A meteorológiai domén-tudás integrálásával származtatott változókat hoztunk létre, amelyek erősebb korrelációt mutatnak a csapadékképződéssel:
 - Napi hőingadozás (temp_range): A maximális és minimális hőmérséklet különbsége.
 - Harmatpont-különbség (dew_depression): Az aktuális hőmérséklet és a harmatpont különbsége (temperature - dewpoint). Ez a változó fizikailag közvetlen kapcsolatban áll a levegő telítettségével, így a csapadék valószínűségével.

🔧 Feature engineering – ez sokat fog dobni

1. Hőingadozás

python

Copy code

```
df['temp_range'] = df['maxtemp'] - df['mintemp']
```

2. Harmatpont-különbség

Meteorológiában ez közvetlen indikátor esőre.

python

Copy code

```
df['dew_depression'] = df['temperature'] - df['dewpoint']
```

3. (Opcionális) Relatív páratartalom nemlineáris tag

python

Copy code

```
df['humidity_sq'] = df['humidity'] ** 2
```

- Adatok transzformációja:** Bevezettük a StandardScaler használatát. Mivel az SVM érzékeny a változók nagyságrendjére (pl. légnyomás ~1000 hPa vs. felhőzet ~0-8), az adatok standardizálása (átlag=0, szórás=1) elengedhetetlen volt a megfelelő hipersík megtalálásához.

🔧 Az adatok skálázása – ez az SVM sikérének kulcsa

Miért kell scaling?

- A nyomás: ~1000
- A napsütés: 0–10
- A hőmérséklet: 0–40
- A szélsebesség: 0–20

Ha ezek nincsenek skálázva, az SVM gyakorlatilag használhatatlan.

Megoldás: StandardScaler

→ minden oszlop átlag = 0, szórás = 1 skálára kerül

- Kiértékelés:** A Support Vector Machine a kisméretű (néhány száz soros) adathalmazokon matematikai tulajdonságai miatt jobban teljesített, mint a Random Forest, ahogy ez várható is volt. A végső modell pontossága 85.14 % lett, ezzel elérte a **85 %-os kritériumot**.

Összegzés

A fejlesztési folyamat konklúziójaként megállapítható, hogy csak a **V3-as iteráció** eredményezte a kívánt pontosságot. A siker kulcsa nem csupán az algoritmusváltás (SVM) volt, hanem a Feature Engineering alkalmazása, amellyel a modell számára explicit módon elérhetővé tettük a meteorológiai összefüggéseket. Ennek az ötletnek a megvalósítása került a leghosszasabb gondolkodási időbe, közel **1,5 percbé**.

ÖSSZEHASONLÍTÁS A REFERENCIÁVAL

Bár a fejlesztés kiindulópontjaként a GeeksforGeeks (továbbiakban: GfG) publikációja szolgált, a generatív AI modellünk (ChatGPT-5.1) segítségével létrehozott végső megoldás (V3) több ponton is túlmutat a referencia-implementáció (ezt eldönthetjük, hogy meglepő-e, vagy sem). Az alábbiakban összegezzük a legfontosabb eltéréseket és minőségi javításokat.

1. Adatelőkészítés és Feature Engineering (A legjelentősebb eltérés)

- **GeeksforGeeks megközelítés:** A referencia cikk a nyers adatokra fókuszál. Általában megelégszik a hiányzó értékek átlaggal való pótlásával és a kategóriák (yes/no) egyszerű számmá alakításával. Nem alkalmaz származtatott változókat.
- **Megvalósított megoldás (V3):** Az AI "fejlesztőpartner" felismerte, hogy a nyers adatok önmagukban nem elegendők a magas pontossághoz. Bevezetésre került a Domain Driven Feature Engineering (szakértői változók képzése):
 - Kiszámoltuk a napi hőingadozást (temp_range).
 - Létrehoztuk a harmatpont-különbséget (dew_depression), amely meteorológiai szempontból a csapadékrajz egyik legerősebb indikátora, míg a GfG megoldás ezt az összefüggést rejte hagyta a modell előtt.

2. Modellarchitektúra és Adatskálázás

- **GeeksforGeeks megközelítés:** A példakódok gyakran közvetlenül illesztik a modellt a nyers adatokra. Ez a fák (Random Forest) esetében működhet, de távolságágalapú algoritmusoknál (Logistic Regression, SVM) hibás vagy szuboptimális eredményhez vezet, mivel a változók eltérő nagyságrendűek (pl. Nyomás: 1000 vs. Eső: 1).
- **Megvalósított megoldás (V3):** Bevezettük a **Scikit-learn Pipeline** architektúrát és a **StandardScaler** használatát. minden bemeneti adatot közös skálára hoztunk ($\text{átag}=0$, szórás=1), ami elengedhetetlen feltétele volt annak, hogy a V3-ban alkalmazott Support Vector Machine (SVC) algoritmus felülmúljá a GfG által javasolt egyszerűbb modelleket.

3. Optimalizáció és Validáció

- **GeeksforGeeks megközelítés:** A cikk jellemzően rögzített (default) paraméterekkel futtatja a modelleket, vagy manuálisan állít be egy-két értéket. A validáció általában egy egyszerű 80-20%-os vágással (train-test split) történik.
- **Megvalósított megoldás (V2-V3):** A kódunk automatizált **hiperparaméter-keresést** (**GridSearchCV**) alkalmazott. A rendszer nem "tippelez" a beállításokat, hanem szisztematikusan végigpróbálja a paraméterteret (pl. SVM kernel típusok, C regularizációs érték), hogy megtalálja a matematikailag bizonyítható optimumot.

Összegzés

Míg a GeeksforGeeks cikke egy kiváló technikai demonstráció ("hogyan írunk Python kódot"), addig az AI modellek segítségével fejlesztett V3-as verzió egy szakmailag optimalizált megoldássá vált. Az AI képes volt szintetizálni a kódolási tudást a meteorológiai alapismeretekkel (feature engineering), és olyan ipari best-practice megoldásokat (Pipeline, Scaling, GridSearch) alkalmazott, amelyek a referencia anyagban nem, vagy csak érintőlegesen szerepeltek. Ennek eredménye a mérhető pontosságnövekedés.

Eredmények diszkussziója és kritikai értékelés

Az alábbi fejezetben a projekt tanulságait, a generatív AI fejlesztői szerepét, valamint az elkészült szoftver minőségi és biztonsági jellemzőit elemezzük.

1. A tudásszint-különbségek áthidalása (Kezdő vs. AI)

Kérdéses, hogy egy kezdő programozó vagy adattudományban kevésbé jártas mérnök magától eljutott volna-e az SVM (Support Vector Machine) alkalmazásáig és az adatskálázás szükségességéig.

- Az algoritmus kiválasztása:** A szakirodalomban és online tutorialokban (mint a GfG cikk) a Random Forest dominál, mint " mindenre jó" eszköz. Egy kezdő valószínűleg megállt volna a V2-es szinten, és az alacsonyabb pontosságot az adatok hibájának tudta volna be. Az, hogy az AI javasolta az SVC-t a kis adatmennyiség miatt, egy **szenior adattudási intuíciót** helyettesített.
- Információkeresés:** Bár az interneten elérhető a tudás ("SVM for small datasets"), a kezdők számára a zajszűrés a legnehezebb. A megfelelő kulcsszavak (pl. feature scaling importance in SVM) ismerete nélkül nehéz rátalálni a megoldásra. Az AI itt kurátorként működött: nemcsak kódot adott, hanem irányt mutatott.

2. Fejlesztési idő és hatékonyság

A fejlesztési idő összehasonlítása drasztikus különbséget mutat:

- Tapasztalt Python fejlesztő:** A feladat specifikációjából kiindulva egy szenior fejlesztő számára a boilerplate kód (Flask szerver, alap modellek) megírása kb. **1-2 munkaórát** venne igénybe, a hibakereséssel és finomhangolással együtt. (erős saccolás)
- Kezdő fejlesztő (AI nélkül):** A dokumentációk olvasása, a szintaktikai hibák (pl. pandas importok, pip problémák) javítása és a ML koncepciók megértése miatt ez a projekt **több napos vagy hetes** kutatómunkát igényelne. (nem erős saccolás)
- Jelen projekt (AI segítséggel):** A tényleges fejlesztési idő (a promptolást és a kód futtatását beleértve) **töredékére csökkent**. Ez azt bizonyítja, hogy a modern LLM-ek (Large Language Model) képesek "pair programmer"-ként funkcionálni, drasztikusan felgyorsítva a prototípus-készítést (Rapid Prototyping).

3. Optimalizációs lehetőségek és a modell határai

A jelenlegi V3-as modell (~85% pontosság) a rendelkezésre álló 366 adatsor mellett közel áll a matematikai optimumhoz.

- Továbbfejlesztési irányok:** A mechatronikában és adattudományban érvényes ökölszabály szerint: "Jobb adat többet ér, mint egy jobb algoritmus." A további finomhangolás helyett az adatbázis bővítése (több évnyi mérés) hozna szignifikáns javulást.
- Alternatívák:** Nagyobb adatmennyiség esetén érdemes lenne megvizsgálni a XGBoost vagy LightGBM algoritmusokat, illetve szekvenciális adatkezelés esetén (idősoros elemzés) az LSTM (Long Short-Term Memory) neurális hálókat, de a jelenlegi kis mintán ezek túltanuláshoz vezetnének.

4. Biztonsági audit és kódminőség

Bár a kód funkcionálisan működik, ipari környezetben ("production ready") több biztonsági kockázatot hordoz, amelyeket egy élesítés előtt kezelní kellene:

- **Pickle sebezhetőség (Insecure Deserialization):** A modell mentésére használt pickle modul biztonsági kockázatot jelent. Ha a model.pkl fájlt egy támadó kicseréli, a szerver a betöltéskor tetszőleges kártékony kódot futtathat le. Éles környezetben biztonságosabb formátum (pl. ONNX) javasolt.
- **Fejlesztői mód:** Az alkalmazás debug=True módban fut, ami hiba esetén a böngészőben megjeleníti a szerver belső kódját és környezeti változóit, ami súlyos adatszivárgáshoz vezethet.
- **Bemeneti validáció hiánya:** A backend minimális ellenőrzést végez a bejövő adatokon. Egy rosszindulatú kérés (pl. extrém nagy számok vagy nem JSON formátum) a szerver leállását (Denial of Service) okozhatja.

Összegzés

A projekt sikeresen demonstrálta, hogy egy mérnök (akár mechatronikai háttérrel) az AI segítségével képes olyan szintű szoftveres megoldást szállítani rövid idő alatt, amely funkcionálásában és pontosságában meghaladja a belépő szintet, ugyanakkor az **ipari bevezetéshez további szoftverergonomiai és biztonságtechnikai fejlesztések szükségesek**.

FRONTEND BACKEND ÉS UI

A kutatás részeként részletes összehasonlító elemzést végeztünk az általam manuálisan létrehozott kezdeti prototípusok (V0 verziók) és a mesterséges intelligencia által optimalizált végleges változatok között. Az elemzés célja annak elemzése, hogyan javította az AI a kódminőséget, a robosztusságát és a felhasználói élményt anélkül, hogy az alapvető működési logikát megváltoztatta volna. A promtoláskor kezdőnek feltételeztem magam, és egy kezdő szintű prototípust is állítottam össze.

1. Szerveroldali logika (Backend) elemzése

A backend összehasonlítása (appV0.py vs. app.py) rávilágít azokra a strukturális és hibakezelési különbségekre, amelyek a manuális kódolás során gyakran figyelmen kívül maradnak, de az ipari szoftverfejlesztésben elengedhetetlenek.

- Adatbiztonság és típuskezelés:** A manuális (V0) verzió közvetlenül adja át a beérkező adatokat a modellnek, feltételezve, hogy azok formátuma tökéletes. Ezzel szemben az AI által generált kód explicit típuskonverziót (float()) és NumPy tömböket alkalmaz. Ez a megoldás sokkal robosztusabb, mivel megakadályozza az alkalmazás összeomlását abban az esetben, ha a bemeneti adat nem megfelelő formátumú.
- Kommunikációs szabványok (API Design):** A V0 verzió egyszerű szöveges karakterláncnal (string) tér vissza, ami nem szabványos megoldás webes környezetben. Az AI által írt végleges verzió ezzel szemben a RESTful API elveknek megfelelő JSON formátumú választ (jsonify) implementált. Ez a szabványosítás elengedhetetlen a frontend és backend közötti megbízható adatcseréhez és a későbbi bővíthetőséghöz.
- Üzleti logika és felhasználói visszajelzés:** Míg a prototípus csupán a modell nyers kimenetét (0 vagy 1) továbbította, az AI felismerte az értelmezés szükségességét. A végleges kód a szerveroldalon fordítja le a bináris eredményt humán-olvasható üzenetté ("Várható esőzés" / "Nem várható eső"), ezzel javítva a rendszer átláthatóságát.
- Kódhigiénia és struktúra:** Az AI optimalizálta a könyvtárak importálását (eltávolította a felesleges modulokat, pl. os) és bevezette a Python belépési pontjának szabványos ellenőrzését (if __name__ == '__main__':). Ez a strukturális javítás modularizálhatóvá és tesztelhetőbbé tette a kódot.

2. Felhasználói felület (Frontend) elemzése

A kliensoldali kódok (indexV0.html vs. index.html) összevetése mutatja a legnagyobb minőségi ugrást: a funkcionális vázból egy modern, reszponzív webalkalmazás született.

- Technológiai paradigmaváltás (Szinkron vs. Aszinkron):** A legnagyobb technikai különbség az adatküldés módjában rejlik. A V0 verzió hagyományos, oldal-újratöltéssel járó űrlapküldést alkalmazott, ami elavult felhasználói élményt nyújt. Az AI ezzel szemben modern JavaScript fetch API-t használt, létrehozva egy "Single Page Application" (SPA) érzetet, ahol az oldal nem villan, az eredmény pedig dinamikusan jelenik meg.
- Felhasználói élmény (UX) és visszajelzés:** A manuális prototípus nem adott visszajelzést a számítási folyamatról. Az AI által generált megoldás vizuális állapotjelzést ("Számítás folyamatban...") és szemantikus színkódolást (zöld/piros háttér) vezetett be az eredmény

függvényében. Ez a fajta azonnali vizuális visszacsatolás kritikus fontosságú a felhasználói bizalom és a használhatóság szempontjából.

- **Megjelenés és Reszponzivitás (UI Design):** A V0 verzió nem tartalmazott formázást (CSS), így megjelenése nyers és strukturálatlan volt. Az AI asszisztens nemcsak fejlesztőként, hanem designerként is funkcionált, az interface fizikai látása ellenére (erre még később kitérek): modern CSS keretrendszert (Flexbox), esztétikus árnyékolást és mobil-optimalizált nézetet (viewport meta tag) implementált, ami alapkötetelmény a mai webes környezetben.
- **Fejlesztést támogató funkciók:** Az AI felismerte a tesztelés nehézségeit, ezért az űrlapmezőket reális alapértelmezett értékekkel (`value="1015.9"`) töltötte fel. Ez az apró, de lényeges figyelmesség jelentősen felgyorsította a fejlesztési és tesztelési ciklust.

Összegzés

Az összehasonlító vizsgálat alátámasztja, hogy a generatív AI modellek alkalmazása nem csupán a kódolási folyamatot gyorsítja, hanem minőségi szintlépést is eredményez, ha kezdő programozóról beszélünk.

- **Szabványosítás:** Az AI automatikusan érvényesíti az ipari best-practice megoldásokat (JSON, Fetch API), amelyeket egy kezdő fejlesztő gyakran nem ismer vagy figyelmen kívül hagy.
- **Holirisztikus szemlélet:** Míg a névleges kezdő humán fejlesztőnek nevezett jómagam (V0) kizártlag a funkcionális működésre koncentrált (ahogy ezt egy kezdő tenné, feltételezve, hogy utólag visszagondolva én is így kezdtem), az AI a rendszert egységen kezelte, figyelembe véve a biztonságot, a karbantarthatóságot és a felhasználói élményt is (ez már tapasztalt szint).

Ez a kísérlet bizonyítja, hogy az AI hatékony "mentor" szerepet is betölthet, ha a fejlesztő alapjáraton nem webfejlesztésre specializálódott, mint jómagam.



Esőzés Előrejelző

Add meg a mai adatokat:

Légnymás (Pressure):

1200

Max Hőmérséklet (MaxTemp):

30

Hőmérséklet (Temperature):

20

Min Hőmérséklet (MinTemp):

40

Harmatpont (Dewpoint):

19.9

Páratartalom (Humidity %):

95

Felhőzet (Cloud %):

90

Napsütés (Sunshine):

5

Szélirány (Wind Direction):

0.0

Szélsebesség (Wind Speed):

0.2

Előrejelzés kérése

Legyél frontend programozó zseni! Kezdő vagyok htm programozásban azonban egy sokkal interaktívabb webfelületet szeretnél ami megadja a mértékegységeket is, izgalmasabb ábrák vannak benne és felhasználóbarátabb.

Thought for 1m 5s >

Szuper, nézzük meg, hogyan lehet ebből egy sokkal "élőbb" és felhasználóbarátabb felületet csinálni. 😊
Láttam a mostani Flask–HTML párost, erre fogok építeni.

Alább kapsz egy teljesen új `index.html`-t, ami:

- Minden mezónél jól látható mértékegységet mutat (hPa, °C, %, óra, km/h, °).
- Csoportosítja a mezőket (Hőmérséklet, Légkör, Szél, Egyéb).
- Ad vizuális "mini-ábrákat" (csíkdiagram) pár fontos értékre (páratartalom, felhőzet, napsütés, szélsebesség).
- Szebb, modernebb elrendezést ad, animált eredménykártyával.

Csak cseréld le a `templates/index.html` (vagy ahová most teszed) tartalmát erre a kódra.

Esőzés Előrejelző

Add meg a mai meteorológiai adatokat, és megmondjuk, várható-e csapadék.

Mértékegységek megjelenítve | Mini "ábrák" a fő paraméterekhez | ML modell a háttérben (Flask)

Légkör

Nyomás + hőmérséklet
1015.9 hPa átlag 95 % közepes

~1010–1025 hPa tipikusan. 80% felett már nagyon nedves levegő.

Hőmérséklet & Harmatpont

Max. hőmérséklet °C	Átlag hőmérséklet °C	Min. hőmérséklet °C
21.3	20.7	20.2

Harmatpont °C Felhőzet %

Ha közel van a hőmérséklethez, nagyobb eséllyel eső.

Szél & Napsütés

Dinamikus időjárási tényezők
Napsütés óra 40.0 Szélirány 0° = észak, 90° = kelet, stb. Szélsebesség km/h 13.7

ERedmény

Magas eséllyel várható esőzés!

Érdemes esernyőt vinni – a modell esőt jelez.

Rel. páratartalom	Felhőzet	Napsütés
95 %	81 %	0.0 óra

Fő mutatók – vizuális áttekintés

Páratartalom	Felhőzet	Napsütés	Szélsebesség
95 %	81 %	0.0 óra	13.7 km/h

● alacsony → közepes → magas érték
Az ábrák a saját, éppen beírt adataidból készülnek – nincs extra backend lekérés.

DEEP RESEARCH

Annak érdekében, hogy demonstráljuk a generatív AI képességeit a design területén, egy utolsó, "deep research" (mélyebb kutatási) iterációt hajtottunk végre. A cél már nem a funkció bővítése volt, hanem a meglévő frontend teljes vizuális átalakítása egy modern, értékesíthető szoftvertermék szintjére.

A prompt explicit módon "professzionális, eladható megoldást" kérte. Az alábbiakban bemutatjuk az AI által generált eredményt, amely technológiailag (HTML5/CSS3 modern megoldásai) és esztétikailag is jelentős ugrást jelent a korábbi verzióhoz képest, mindezt úgy, hogy a háttérben futó Python logikán nem kellett változtatni.

Az eredmény azonban váratlan volt:



Az eredmények elemzése és a kudarc okai

- Regresszió a sztenderd megoldások irányába:** Megfigyeltük, hogy kreatív design-megoldások (pl. modern CSS keretrendszer, animációk) alkalmazása helyett a modell a „biztonsági játékot” választotta. A generált kód szinte teljesen megegyezett a funkcionális minimumot teljesítő verzióval. Arra a következtetésre jutottunk, hogy specifikus design-instrukciók (pl. „használj Glassmorphism stílust”) hiányában az AI az átlagos, tankönyvi példákhoz konvergál.
- A „kutatás” fogalmának félreértelmezése:** Tapasztalatunk szerint a modell ebben a kontextusban a „kutatást” nem vizuális trendkutatásként, hanem funkcionális egyszerűsítésként értelmezte. Ahelyett, hogy a vizuális megvalósítás mélységeit kereste

volna, a kód stabilitását prioritálta, ami paradox módon a primitívebb, V0-hoz hasonló megoldáshoz való visszatérést eredményezte.

- **A specifikáció hiánya:** A kísérlet során beigazolódott, hogy az olyan absztrakt utasítások, mint a „profi megoldás” vagy „eladható UI”, nem értelmezhetőek megfelelően a modell számára. Bebizonyosodott, hogy a „Human-in-the-loop” (ember a hurokban) megközelítés elengedhetetlen: pontos vizuális specifikációk és mérnöki/designeri iránymutatás nélkül az AI nem képes önállóan esztétikai döntéseket hozni. Kifejezetten hangsúlyoznám, hogy az AI végső soron nem látja a kijelzőt!!

Összegzés

Vizsgálatunk alapján kijelenthetjük, hogy még a backend logikai optimalizálásában és a hibakeresésben az AI kiválóan teljesített, a kreatív, vizuális tervezési folyamatokban önálló ágensként korlátozottan használható, kifejezetten a Deep Research funkció. A kísérlet megerősítette azt a hipotézisünket, hogy a magas szintű felhasználói élmény megteremtéséhez továbbra is elengedhetetlen az emberi tervezői kompetencia.

FORRÁSOK

- [1] Kiindulási alap és Módszertan **GeeksforGeeks**, "Rainfall Prediction using Machine Learning Python," GeeksforGeeks, n.d. [Online]. Elérhető: <https://www.geeksforgeeks.org/machine-learning/rainfall-prediction-using-machine-learning-python/>. [Megtekintve: 2025. dec. 02.].
- [2] Pallets Projects, "Flask Documentation (3.0.x)," 2024. [Online]. Elérhető: <https://flask.palletsprojects.com/>. [Megtekintve: 2025. dec. 02.].
- [3] Algoritmusok elméleti háttere (V1, V2, V3 modellek) D. R. Cox, "The Regression Analysis of Binary Sequences," Journal of the Royal Statistical Society. Series B (Methodological), vol. 20, no. 2, pp. 215-242, 1958. (Logisztikus Régesszióhoz)
- [4] L. Breiman, "Random Forests," Machine Learning, vol. 45, no. 1, pp. 5–32, 2001. (Random Foresthez)
- [5] C. Cortes és V. Vapnik, "Support-vector networks," Machine Learning, vol. 20, no. 3, pp. 273–297, 1995. (SVM/SVC-hez - ez a V3 modell alapja)
- [6] Mesterséges Intelligencia Eszközök OpenAI, "ChatGPT: Optimizing Language Models for Dialogue," 2024. [Online]. Elérhető: <https://openai.com/chatgpt>. (A kódgeneráláshoz használt eszköz)
- [7] Egyetemi tananyag B. Kővári, I. Albert, I. Rajacsics, G. Simon, "Szoftverfejlesztés MI támogatással (VIAUBXAV087) - Előadásanyagok," Budapesti Műszaki és Gazdaságtudományi Egyetem, Villamosmérnöki és Informatikai Kar, 2025/26/1. félév. [Online]. Elérhető: <https://edu.vik.bme.hu/course/view.php?id=14555>.

GITHUB: https://github.com/sarlosp/SZMI_BMEVIAUBXAV087_O38B1K-2025-26-1