Aktuelle Trends der Informations- und Kommunikationstechnik- SoSe24

Projektdokumentation von Nhi Nguyen

Motivation und Relevanz
Meine Leidenschaft für Brettspiele wollte ich im Bereich der Informatik integrieren. Die Aufgabe lautete, ein Machine Learning Modell zu einem Thema meiner Wahl zu bauen. Ich wollte eine App bauen, die den Nutzern ermöglicht individuelle Brettspielempfehlungen anzuzeigen. Die Relevanz dieser Aufgabe zeigt sich darin, dass Empfehlungsalgorithmen mittlerweile ein fester Bestandteil zahlreicher Anwendungen und Plattformen sind – sei es im Bereich von Online-Shopping, Musikempfehlungen. Da die Vielfalt an Brettspielen kontinuierlich wächst, ist es für viele Menschen schwierig, Spiele zu finden, die genau ihren Vorlieben entsprechen. Häufig verbringt man viele Stunden im Internet oder in Brettspielgeschäftigen damit, herauszufinden welches Brettspiel zu einem passt. Die Schwierigkeit kann zu hoch sein, man ist mit dem Preis nicht einverstanden oder das Thema des Spiels spricht einen nicht an und so setzt man die Suche weiter fort. Mit diesem Projekt wollte ich herausfinden, ob es eine Möglichkeit gibt, der Problematik zu umgehen und schneller und effektiver ein passendes Brettspiel für sich und seine Mitmenschen zu finden.

Beschreibung der verwendeten Daten
Die Datenbasis des Projekts besteht aus einer CSV-Datei, die eine umfassende Sammlung von Brettspielen und deren Eigenschaften enthält. Die Daten stammen aus der Website boardgamegeek.com aus dem Jahr 2017. Die Website bietet eine Plattform für alle Brettspielliebhaber, um Spiele zu bewerten, Bewertungen zu lesen oder über Spiele zu diskutieren. Der Datensatz umfasst 4998 Einträge und 20 Spalten: rank, bgg_url, game_id, names, mind_players, max_players, avg_time, min_time, max_time, year, avg_rating, geek_rating, num_votes, image_url, age, mechanic, owned, category, designer und weight. Von den 20 Spalten wurden im Modell folgende Spalten verwendet, die für die Empfehlungslogik entscheiden sind: Die bgg_url, um dem User einen Link mit der Beschreibung zu der Website mit den jeweiligen Brettspielen zu geben, damit sie sich über das Spiel informieren können. Die Spalte names, um den jeweiligen Namen des Spiels anzuzeigen. Die Spalten min_players und max_players wurden verwendet, um den User zu fragen, zu wievielt es ein Spiel spielen möchte und die avg_time ist dafür da, um die gewünschte Dauer anzugeben. Zusätzlich sind die Spalten category und weight verwendet worden, um ein Spielthema und Schwierigkeit eines Spiels abzufragen. Nachträglich wurde die Spalte theme entfernt, da die Begriffe für den User verwirrend waren, weshalb die Spalte durch category ersetzt wurde.
Zuletzt wurde die Spalte age verwendet, um sich ein Spiel passend zur gegebenen Altersklasse ausgeben zu lassen, damit Spieler*innen unter 18 Jahren in Erwägung gezogen werden können.
Ein wichtiger Teil der Datenvorbereitung war die Bereinigung der Daten, um irrelevante Kategorien wie Kriegsspiele zu entfernen und eine Kategorisierung in wichtige Merkmale wie Spieldauer, Mechaniken und Themen vorzunehmen. Diese Datenvorbereitung war notwendig, um eine solide Grundlage für den Empfehlungsalgorithmus zu schaffen und sicherzustellen, dass die Empfehlungen auf relevanten Informationen basieren.

Beschreibung der Vorgehensweise
Es gibt insgesamt 4 Sprints. Im Folgenden werden die Sprints in Detail mit Codeausschnitten erläutert. Es folgen die Sprintziele, Epics, die dazugehörigen Tasks, die nächsten Schritte sowie ein Sprintreview. Die User Stories wurden nachträglich entfernt, da sie sich in den Tasks wiederholt haben und zuletzt um die Seitenanzahl von ca. 10 Seiten nicht stark zu übersteigen.

Sprint 1 vom 15.04- 15.06.24

Sprintziel
Das Ziel des ersten Sprints ist es, eine grundlegende Anwendung in Streamlit zu entwickeln, die eine benutzerfreundliche Oberfläche bietet und die Verarbeitung von Brettspieldaten ermöglicht. Darüber hinaus soll die Anwendung mithilfe eines k-Nearest Neighbors (kNN) Modells 15 personalisierte Brettspielempfehlungen basierend auf dem Lieblingsspiel der Benutzer generieren.

Epic 1: Grundlegende Streamlit Anwendung entwickeln

Tasks

1.1 Streamlit Anwendung aufsetzen

- Terminalbefehle: cd boardgameproject, pip install streamlit, pip install scikit-learn
- Erstellung von Datei boardgameapp.py

1.2 Willkommensseite erstellen

- Erstellung text_input für die Frage "Wie heißt dein Lieblingsbrettspiel?"
- Auswahl von Spielnamen aus dem Datensatz
- Styling der Frage als custom-input-label im markdown
- Auswahl von einem Hintergrundbild

```
...
page_bg_img = '''
   <style>
   [data-testid="stAppViewContainer"] {
      background-image: url("https://www.washingtonpost.com/wp-apps/imrs.php?src=https://arc-   anglerfish-washpost-prod-
washpost.s3.amazonaws.com/public/K6QMID53AFASXKQ6GJOQ6STHIQ.jpg&w=1440&impolicy=high_res");
      background-size: cover;


   }
   </style>
   '''
   st.markdown(page_bg_img, unsafe_allow_html=True)

   st.markdown("""
               <style>
      .custom-input-label {
         background: rgb(255,192,118);
         border-radius: 20px;
         background: radial-gradient(circle, rgba(255,192,118,1) 0%, rgba(255,70,0,1) 100%);
         text-shadow: 2px 2px 4px #ff0000;
         border-color: black;
         border-width: 6px;
         border-style: dotted;
         color: rgba(255,249,243,1);
         padding: 10px;
         text-align: center;
         display: block;
         margin-bottom: 10px;
      }
      </style>
      """, unsafe_allow_html=True)

   st.markdown('<div class="custom-input-label">"Wie heißt dein Lieblingsbrettspiel?"</div>',
         unsafe_allow_html=True)

   favorite_game = st.text_input('')
   if favorite_game:
      filtered_games = df[df['names'].str.contains(favorite_game, case=False, na=False)]
      if filtered_games.empty:
```

```
        st.write("Kein Spiel gefunden. Bitte wähle ein Spiel aus der Liste unten, das deinem Lieblingsspiel am ähnlichsten ist:")
        all_games = df['names'].dropna().unique()
        selected_game = st.selectbox("Ähnlichstes Spiel:", all_games)
    else:
        st.write("Gefundene Spiele:")
        game_options = filtered_games['names'].dropna().unique()
        selected_game = st.selectbox("Wähle ein Spiel:", game_options)
st.write(f"Ausgewähltes Spiel: {selected_game}")
...
```

Epic 2: Datenverarbeitung und -bereinigung

Tasks

2.1 Laden der Brettspieldaten

- CSV-Datei mit Brettspieldaten laden

```
df = pd.read_csv('bgg_db_1806.csv')
```

2.2 Entfernen irrelevanter Begriffe

- Kategorien wie Kriegsspiele aus den Daten entfernen, damit sie nicht in der Auswahl vorkommen

```
...
war_keywords = ['War', 'Wargame', 'Conflict', 'Post-Napoleonic', 'Prehistoric', 'Napoleonic', 'Miscellaneous Game Accessory', 'Traditional Card Games']
filtered_rows = [row for index, row in df.iterrows() if not any(keyword in row['category'] for keyword in war_keywords)]
df = pd.DataFrame(filtered_rows)
...
```

Epic 3: Empfehlungslogik k-Nearest Neighbors implementieren

Tasks

3.1 Input Vector erstellen

- User soll Fragen beantworten zu folgenden Feldern:
- Spieleranzahl, Spieldauer, Mechaniken, Thema, Schwierigkeitsgrad und Mindestalter
- numpy Array für die Erstellung eines neuen Datenpunkts, der die Eigenschaften des Spiels zusammenfasst, für das wir eine Empfehlung mit ähnlichen Eigenschaften suchen
- numpy Array wird in ein DataFrame umgewandelt, damit es mit anderen Daten konsistent ist
- get_dummies() angewendet, um sicherzustellen, dass es die gleiche Struktur wie die Trainingsdaten hat und in numerische Werte umgewandelt wird
- reindex() um die Spalten des neues Datenpunkts anzupassen, dass sie der Reihenfolge der Spalten in X entsprechen und fehlende Spalten mit 0 auffüllen

3.2 kNN-Modell zur Generierung von Empfehlungen

- Auswahl von einem kNN-Modell, da Spiele, die viele gemeinsame Merkmale haben, als ähnlich angesehen und daher als mögliche Empfehlungen vorgeschlagen werden
- kNN kann sowohl für numerische als auch für kategoriale Daten verwendet werden, was vorteilhaft ist für die numerischen Features Alter, Anzahl der Spieler, Spieldauer, Schwierigkeitsgrad und kategorische Features wie Name, Spielmechanismus, Spielthema
- Erstellung der Funktion def get_knn_recommendations() mit übergebenen Parametern, die die Informationen des Users aufnehmen
- Zielvariable y ist der Name des Spiels, da wir dies vohersagen bzw. ermitteln wollen
- Features X sind Anzahl der Spieler, Spieldauer, Spielmechanismus, Spielthema, Schwierigkeitsgrad und Alter
- Kategorische Spaltenwerte wie main_mechanic und category in numerische Werte umwandeln mit pd.get_dummes() Methode
- Erstellen eines kNN Klassifikator mit KNeighborsClassifier namens knn mit 15 Nachbarn (n_neighbors=15)

- fit(X,y), um den kNN-Klassifikator an den Trainingsdaten (X und y) anzupassen, Modell lernt Spiele mit ähnlichen Eigenschaften
- Vorhersage machen mit knn.kneighbors(input_vector, return_distance= False), für den neuen Datenpunkt 15 nächstgelegene Nachbarn finden und in Variable recommendations speichern
- iloc(), um aus dem ursprünglichen DataFrame empfohlene Spiele auszuwählen
- Generierte Empfehlung bzw. die Rückgabe als DataFrame gibt Informationen basierend auf den eingegebenen Präferenzen
- Modell gibt 15 Empfehlungen aus

```
...
def get_knn_recommendations(df, num_players, game_duration, game_mechanics, game_theme, game_difficulty, game_age):
    X = df[['min_players', 'max_players', 'duration_category', 'main_mechanic', 'category', 'weight', 'age']]
    y = df['names']
    X = pd.get_dummies(X, columns=['main_mechanic', 'category'])
    input_vector = np.array([[
        num_players, num_players, game_duration, game_mechanics, game_theme, game_difficulty, game_age
    ]])
    input_vector = pd.DataFrame(input_vector, columns=['min_players', 'max_players', 'duration_category', 'main_mechanic', 'category',
'weight', 'age'])
    input_vector = pd.get_dummies(input_vector, columns=['main_mechanic', 'category'])

    input_vector = input_vector.reindex(columns=X.columns, fill_value=0)

    knn = KNeighborsClassifier(n_neighbors=15)
    knn.fit(X, y)

    recommendations = knn.kneighbors(input_vector, return_distance=False)
    recommended_games = df.iloc[recommendations[0]]

    return recommended_games[['names', 'min_players', 'max_players', 'avg_time', 'main_mechanic', 'category', 'weight', 'age']]
...
```

Sprint Review
Erreichte Ziele:
- Grundlegende Streamlit-Anwendung erfolgreich erstellt
- Datenverarbeitung und erste Version des Empfehlungsalgorithmus implementiert

Herausforderungen:
- Anpassung des KNN-Modells an die gegebenen Daten:
- Herausfinden, dass man kategorischen Werte in numerische Werte umwandeln muss
- Verwendung von reindex() um sicherzustellen, dass der neue Datenpunkt die gleiche Struktur wie Trainingsdaten hat

Nächste Schritte:
- Erweiterung und Verbesserung der Benutzeroberfläche
- Experimentieren mit anderen Modelltypen, z.B. Train-Validation-Test Splits einbeziehen, oder das Evaluieren von Alternativen zum KNN-Modell, z.B. Random Forest

---

Sprint 2 vom 16.06- 30.06.24

Sprintziel
Das Ziel des zweiten Sprints ist es, die Funktionalität und Benutzerfreundlichkeit der Anwendung zu erweitern. Dies beinhaltet die Festlegung von weiteren Fragen auf der Streamlit App für die Nutzung des kNN-Modells, eine einheitliche Schrift sowie die Einführung einer Startseite und einer Erklärungsseite, die zur existierenden Willkommensseite führt. Zusätzlich kommt die Implementierung zusätzlicher Modelle zur Evaluierung wie Train Validation Test Split, Random Forest und die Modularisierung des Codes, um die Wartbarkeit zu verbessern.

Tasks

1.1 Implementierung Train Validation Test Split mit Skalierung

- Funktion preprocess_data() für die Implementierung einer ordinalen Skalierung der Variablen mit OrdinalEncoder, um kategorische Spaltenwerte wie main_mechanic, category in numerische Werte umzuwandeln und in eine Reihenfolge zu bringen
- Funktion split_data für die Implementierung eines Train Validation Test Splits
- Modularisierung für Datenvorverarbeitung (preprocess_date) und Modelltraining (split_data)
- Features X werden gespeichert außer Spalte target
- Zielvariable y ist die Spalte target
- X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.4, random_state=42): Daten werden in Trainings- und Testdaten mit 40 Prozent aufgeteilt und Aufteilung ist immer gleich, wenn der Code erneut ausgeführt wird
- X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42): Testdaten werden weiter in eine Validierungsmenge und Testmenge aufgeteilt
- Funktion gibt verschiedene Teilmengen der Daten zurück, die für das Training des Modells verwendet werden können

```
...
import OrdinalEncoder
def preprocess_data(df):
    ordinal_features = ['main_mechanic', 'category']
    encoder = OrdinalEncoder()
    df[ordinal_features] = encoder.fit_transform(df[ordinal_features])
    return df
from sklearn.model_selection import train_test_split
def split_data(df):
    X = df.drop(columns=['target'])
    y = df['target']
    X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.4, random_state=42)
    X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
    return X_train, X_val, X_test, y_train, y_val, y_test
...
```

1.2 Evaluierung von alternativen Modellen

- Implementierung Random Forest
- Erstellung eines Random Forest Klassifikators mit 11 Bäumen und Zufallsgenerator random_state
- fit() um Random Forest an Trainingsdaten anzupassen und Algorithmus lernt aus den Trainingsdaten, um Vorhersagen zu treffen
- Trainiertes Random Forest Modell wird zurückgegeben

```
from sklearn.ensemble import RandomForestClassifier
def train_random_forest(X_train, y_train):
    rf = RandomForestClassifier(n_estimators=11, random_state=10)
    rf.fit(X_train, y_train)
    return rf
```

1.3 Train Test Split Modell mit Random Forest

- Random Forest in Train Test Split integrieren
- Rückgabe der Funktion war nicht effektiv, es wurde nur ein Spiel empfohlen, das nicht in von den kNN-Empfehlungen vorgekommen ist

```
def get_recommendations(df, num_players, game_duration, game_mechanics, game_theme, game_difficulty, game_age):
    X = df[['min_players', 'max_players', 'duration_category', 'main_mechanic', 'category', 'weight', 'age']]
    y = df['names']
```

```
    X = pd.get_dummies(X, columns=['duration_category', 'main_mechanic', 'category', 'weight', 'age'])

    input_vector = np.array([[
        num_players, num_players, game_duration, game_mechanics, game_theme, game_difficulty, game_age
    ]])
    input_vector = pd.DataFrame(input_vector, columns=['min_players', 'max_players', 'duration_category', 'main_mechanic', 'category',
'weight', 'age'])
    input_vector = pd.get_dummies(input_vector, columns=['duration_category', 'main_mechanic', 'category', 'weight', 'age'])
    input_vector = input_vector.reindex(columns=X.columns, fill_value=0)

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=10)

    rf = RandomForestClassifier(n_estimators=11, random_state=10)
    rf.fit(X_train, y_train)

    recommendations = rf.predict(input_vector)

    recommended_games = df[df['names'].isin(recommendations)]

    return recommended_games[['names', 'min_players', 'max_players', 'avg_time', 'main_mechanic', 'category', 'weight', 'age']]
```

Epic 2: Erweiterung der Benutzeroberfläche

Tasks

2.1 Einheitliche Schriftart

- Erstellung einer Methode load_global_styles() für eine einheitliche Schriftart
- Import mittels URL aus Google Fonts

```
def load_global_styles():
    font_css = '''
    <style>
    @import url('https://fonts.googleapis.com/css2?family=Press+Start+2P&display=swap');
    body * {
        font-family: "Press Start 2P", system-ui !important;
    }
    </style>
    '''
    st.markdown(font_css, unsafe_allow_html=True)
```

2.2 Modularisierung der Datei boardgameapp.py: Styling der Anwendung in eine neue Datei styles

- Erstellung eines Ordners design, darin die Erstellung der Datei styles.py
- Hinzufügen von Design, die für die Gestaltung der Seite sorgt
- styles.py enthält Funktionen load_global_styles(), load_welcome_page_styles(), load_explanation_page_styles()

```
def load_welcome_page_styles():
    welcome_page_markdown = """
    <style>
            body {
        background-color: black;
        color: white;
        font-family: 'Press Start 2P', system-ui;
        display: flex;
```

```
        justify-content: center;
        align-items: center;
        flex-direction: column;
        height: 100vh;
        margin: 0;
        text-align: center;
    }
...
st.markdown(welcome_page_markdown, unsafe_allow_html=True)
def load_explanation_page_styles():
    explanation_page_markdown = """
    <style>
                body {
        background-color: black;
        color: white;
        font-family: 'Press Start 2P', system-ui;
        display: flex;
        justify-content: center;
        align-items: center;
        flex-direction: column;
        height: 100vh;
        margin: 0;
        text-align: center;
    }
...
st.markdown(explanation_page_markdown, unsafe_allow_html=True)
```

2.3 Willkommensseite erstellen

- Funktion show_welcome_page() erstellen in boardgameapp.py
- Design der Seite mit styles.load_welcome_page_styles() anwenden
- Button "Los geht's" implementieren, der zur Erklärungsseite führt, wenn man raufklickt

```
def show_welcome_page():
    styles.load_welcome_page_styles()
    if st.button("Los geht's"):
        st.session_state.page = 'explanation'
```

2.4 Erklärungsseite erstellen

- Funktion show_explanation_page() erstellen in boardgameapp.py
- Design der Seite mit styles.load_explanation_page_styles() anwenden
- Button "Bereit" implementieren, der zur Empfehlungsseite führt, wenn man raufklickt

```
...
def show_explanation_page():
    styles.load_explanation_page_styles()
    if st.button("Bereit!"):
        st.session_state.page = 'recommendations'
...
```

2.5 Empfehlungsausgabe-Seite erstellen

- Funktion show_recommendations() erstellen in baordgameapp.py
- Laden der Brettspieldaten aus einer CSV-Datei
- Benutzerinteraktionen (Eingabe des Lieblingsspiels, Auswahl von Präferenzen)
- Generierung von Empfehlungen basierend auf Benutzerpräferenzen

```
...
def show_recommendations():
    df = pd.read_csv('bgg_db_1806.csv')
st.write("Empfohlene Spiele basierend auf deinen Präferenzen:")
    st.write(recommended_games[['names', 'min_players', 'max_players', 'avg_time', 'main_mechanic', 'category', 'weight', 'age']])
...
```

2.6 Verbesserte Nutzerführung

- Verbesserung der Navigation zwischen den Seiten
- Nutzung von session_state.page, um mehrere Seiten zu erstellen
- User sieht zuerst show_welcome page(), dann show_explanation page() und zuletzt show_recommendations()

```
...
if 'page'not in st.session_state:
    st.session_state.page = 'welcome'if st.session_state.page == 'welcome': show_welcome_page()
elif st.session_state.page == 'explanation':
    show_explanation_page()
elif st.session_state.page == 'recommendations':
    show_recommendations()
```

2.7 Musikdatei hinzufügen

- Auswahl einer Musikdatei, um Usererfahrung zu verbessern

```
...
audio_file = open('hail-126903.mp3', 'rb')
audio_bytes = audio_file.read()
st.audio(audio_bytes, format='audio/mp3', start_time=0)
...
```

2.8 Kategorisieren der Daten

- Mechaniken, Themen, Schwierigkeitsgrad und Mindestalter kategorisieren
- In vorgegebenen Wertebereichen kategorisiert, damit die Auswahl für den User einfacher fällt
- Mehrere Werte einer Spalte werden in Kategorien zusammengefasst

```
...
        game_duration = st.selectbox(
            'Wie lange sollte ein Spiel dauern (Durchschnittszeit in Minuten)?',
            ['0-30', '30-60', '60-120', '120+'])


        game_difficulty = st.selectbox(
            'Welchen Schwierigkeitsgrad bevorzugst du? (1= sehr leicht, 2= leicht, 3= medium, 4= schwer, 5= sehr schwer)',
            sorted(df['weight'].dropna().unique()))
...
```

2.9 Implementierung von zusätzlichen Fragen und Buttons

- Zusätzliche User Fragen zu folgenden Feldern, um es dem kNN-Modell zu übergeben: num_players, game_duration, game_mechanics, game_theme, game_difficulty, game_age
- Nutzung von selectbox(), um Auswahlfelder zu erstellen, aus denen der User seine Präferenzen auswählen kann

```
...
game_age = st.selectbox(
            'Für welches Mindestalter soll das Spiel geeignet sein?',
            sorted(df['age'].dropna().unique())
        )
```

```
    recommended_games = get_recommendations(df, num_players, game_duration, game_mechanics, game_theme, game_difficulty, game_age)
    st.write("Empfohlene Spiele basierend auf deinen Präferenzen:")
    st.write(recommended_games)
...
```

Sprint Review
Erreichte Ziele:

- Einführung von alternativen Modellen zur Evaluierung
- Modularisierung des Codes in funktionale Module und Design der Anwendung separiert
- Erweiterte und verbesserte Benutzeroberfläche durch neue Seiten

Herausforderungen:

- Evaluation von neuen Modellen
- Random Forest sorgte für sehr lange Ladezeiten/ Unendlichkeitsschleifen der Streamlit Anwendung, die ich durch tmux lösen konnte
- Anwendung von Train Test Split mit Random Forest führte zu nur einer Vorhersage, die nicht in den kNN-Vorhersagen vorkamen
- Fazit: Train Test Split kann nicht gut performen und nicht gut geeignet, weil im Testdatensatz Spiele vorkommen, die nicht im Train Datensatz vorkommen. Daher kann es Spiele vom Testdatensatz nicht empfehlen, sondern nur das empfehlen, was es aus den Trainingsdaten kennt
- Primäres Modell für das Projekt bleibt bei kNN

Nächste Schritte:

- Implementierung zusätzlicher Interaktionen und Benutzerfunktionen
- Kontinuierliche Verbesserung der Benutzeroberfläche und des Layouts
- Modularisierung der Datenverarbeitung

Sprint 3 vom 01.07 – 15.07

Sprintziel
Das Ziel des dritten Sprints ist es, die Anwendung weiter zu optimieren, indem der Code modularisiert und die Datenverarbeitung effizienter gestaltet wird. Dazu gehört die separate Implementierung der Datenvorverarbeitung und die Erweiterung der Empfehlungslogik durch Anwendung des Cross Validation Splits Modell. Zuletzt steht eine Zwischenpräsentation mit den bisherigen Fortschritten und Herausforderungen bereit.

Epic 1: Verbesserung der Datenvorverarbeitung

Tasks
1.1 Auslagerung der Datenvorverarbeitung

- Datenverarbeitungslogik in einem separaten Ordner model auslagern
- Datei preprocess.py erstellen

```
def preprocess_data(filepath):
    df = pd.read_csv(filepath)
    war_keywords = ['War', 'Wargame', 'Conflict', ...]
    filtered_rows = [row for index, row in df.iterrows() if not any(keyword in row['category'] for keyword in war_keywords)]
    df = pd.DataFrame(filtered_rows)

    df['duration_category'] = df['avg_time'].apply(categorize_duration)
    df['main_mechanic'] = df['mechanic'].apply(lambda x: x.split(',')[0])
    df['main_theme'] = df['category'].apply(lambda x: x.split(',')[0])
    df['weight'] = df['weight'].apply(categorize_difficulty)
    df['age'] = df['age'].apply(categorize_age)

    return df, unique_mechanics, unique_themes
```

1.2 Verbesserung der Kategorisierung

- Kategorierung-Funktion anpassen
- Schwierigkeitsgrad und Alter in klar definierte Kategorien einteilen

```
def categorize_difficulty(weight):
    if 1 <= weight < 2:
        return '1'
    elif 2 <= weight < 3:
        return '2'
    ...

def categorize_age(age):
    if age < 6:
        return '0'
    elif 6 <= age < 12:
        return '6'
            ...
```

Epic 2: Modularisierung der Anwendung

Tasks

2.1 Auslagerung der UI

- Erstellung einer neuen Datei app_ui.py im Ordner design
- Separierung der Benutzeroberflächenlogik in app_ui.py

```
from model.preprocess import preprocess_data
from design.styles import load_global_styles, load_welcome_page_styles

def show_welcome_page():
    load_welcome_page_styles()
    if st.button("Los geht's"):
        st.session_state.page = 'explanation'
        st.rerun()

def show_explanation_page():
    load_explanation_page_styles()
    if st.button("Bereit!"):
        st.session_state.page = 'recommendations'
        st.rerun()
def show_recommendations():
    df, unique_mechanics, unique_themes = preprocess_data('bgg_db_1806.csv')
    ...
```

2.2 Übertragung der neuen Datei boardgameapp.py

- Importe hinzufügen:

```
from design.app_ui import show_welcome_page, show_explanation_page, show_recommendations
```

Epic 3: Implementierung eines Cross Validation Split Modell als alternatives Empfehlungsmodell

Tasks

3.1. Empfehlungslogik Cross Validation Split

- Erstellung einer neuen Datei cross_validation_split im Ordner model
- Erstellung von Funktion cross_validation_split()

- Variable X enthält alle Spalten des DataFrames df außer target_column
- Zielariable y ist target_column
- KFold(), um KFold- Objekt zu erstellen mit n_splits folds
- Daten werden vor jeder Aufteilung gemischt (schuffle=True)
- Aufteilung in Trainings- und Validierungsdaten und gibt die Liste der aufgeteilten Daten zurück

3.2 Empfehlungslogik cross validation Split mit Random Forest erweitern

- Erstellung von Funktion perform_cross_validation(), um Kreuzvalidierung durchzuführen und Genauigkeit berechnen mit Random Forest Classifier
- accuracy_score() berechnet die Genauigkeit der Vorhersagen
- avg_accuracy = sum(acc for _, acc in results) / len(results) berechnet die durchschnittliche Genauigkeit über alle folds
- Gibt Liste der Ergebnisse jedes folds und durchschnitte Genauigkeit zurück
- Datei laden und anwenden in boardgameapp.py
- Beispielausgabe der Folds:

Durchführung der Cross-Validation... Fold 1 Accuracy: 0.7854 Fold 2 Accuracy: 0.8210 Fold 3 Accuracy: 0.7532 ... Fold 10 Accuracy: 0.8472 Average Accuracy: 0.7998

```
def cross_validation_split(df, target_column, n_splits=10):
    X = df.drop(columns=[target_column])
    y = df[target_column]
    kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)
    splits = []
    for train_index, val_index in kf.split(X):
        X_train, X_val = X.iloc[train_index], X.iloc[val_index]
        y_train, y_val = y.iloc[train_index], y.iloc[val_index]
        splits.append((X_train, X_val, y_train, y_val))
    return splits


def perform_cross_validation(df, target_column):
    splits = cross_validation_split(df, target_column)
    results = []
    for i, (X_train, X_val, y_train, y_val) in enumerate(splits):
        model = RandomForestClassifier(n_estimators=10, random_state=11)
        model.fit(X_train, y_train)
        y_pred = model.predict(X_val)
        accuracy = accuracy_score(y_val, y_pred)
        results.append((i, accuracy))
    avg_accuracy = sum(acc for _, acc in results) / len(results)
    return results, avg_accuracy
```

```
#boardgameapp.py
...
    st.write("Durchführung der Cross-Validation...")
        results, avg_accuracy = perform_cross_validation(df, 'target')
        for i, accuracy in results:
            st.write(f"Fold {i+1} Accuracy: {accuracy:.4f}")
        st.write(f"Average Accuracy: {avg_accuracy:.4f}")
...
```

Epic 4: Zwischenpräsentation vorbereiten

Tasks

4.1 Präsentationsfolien vorbereiten
- Präsentation auf Canva gestalten
- Gliederung erstellen
- Präsentationsfolien erstellen

4.2 Übersicht über bisherigen Ergebnissen und Herausforderungen machen
- Ergebnisse zusammenfassen
- Herausforderungen zusammenfassen
- Ausblick notieren

Sprint Review
Erreichte Ziele:
- Modularisierung der Datenverarbeitung und Benutzeroberfläche erfolgreich umgesetzt
- Optimierte Datenverarbeitungslogik zur effizienten Kategorisierung und Filterung der Brettspieldaten implementiert
- Weiteres Empfehlungsmodell Cross Validation Split gebaut
- Zwischenpräsentation erfolgreich erstellt

Herausforderungen:
- Evaluation des Models Cross Validation Split
- Fazit: Ausgabe der Genauigkeit der Folds nicht effektiv für die Vorhersage von Brettspielen, für die die Namen benötigt werden
- Primäres Modell bleibt das kNN-Modell

Nächste Schritte:
- Weitere Verfeinerung der Benutzeroberfläche und Interaktionsmöglichkeiten
- Möglichkeit, dass der User das Fragenkatalog überspringen kann, wenn es schon ein Lieblingsspiel hat
- Möglichkeit, dass der User nur das Fragenkatalog macht, wenn es kein Lieblingsspiel hat
- User kann mehrere Lieblingsspiele angeben

Sprint 4 vom 16.07- 23.09 (inkl. vorlesungsfreie Zeit)

Sprintziel
Das Hauptziel von Sprint 4 war die Verbesserung der Benutzerinteraktion durch die Implementierung weiterer Fragen im Empfehlungsprozess, die Optimierung der Benutzeroberfläche (UI) sowie die Umstellung der App-Sprache.
Zudem sollte der Empfehlungsalgorithmus so angepasst werden, dass 3 Spiele berücksichtigt werden können. Die Dokumentation des Projekts wurde ebenfalls abgeschlossen.

Epic 1: Verbesserung des Empfehlungsprozesses

Tasks
1.1 Implementierung des Pfads, wenn man ein Lieblingsspiel hat
- Ein Radio-Button wird hinzugefügt, der die Benutzer fragt, ob sie ein Lieblingsspiel haben
- Button Auswahl: I remembered some from last time und No, this is why I'm here
- Pfade mit session_state.proceed_choice erstellt
- Wenn Auswahl "remembered", dann folgt Abfrage von Lieblingsspielen und optional die Möglichkeit mehrere Spiele anzugeben und das Fragenkatalog durchzugehen

```
col1, col2 = st.columns(2)
    with col1:
        if st.button("I remembered some from last time."):
            st.session_state.proceed_choice = 'remembered'
    with col2:
            if st.button("No, this is why I'm here!"):
             st.session_state.proceed_choice = 'no_favorite'


if st.session_state.proceed_choice == 'remembered':
```

```
    st.markdown('<div class="custom-input-label"> A wondrous inclination! The magic bubble awaits your chosen game. Speak its name by typing it here below. </div>', unsafe_allow_html=True)
    favorite_game = st.text_input(' '
    ...
```

1.2 Implementierung des Pfads, wenn man kein Lieblingsspiel hat

- Wenn Auswahl "no_favorite" dann folgt das Fragenkatalog

```
...
elif st.session_state.proceed_choice == 'no_favorite':
    st.markdown('<div class="custom-input-label">A delightful preference! With how many companions do you prefer to play an adventure?</div>', unsafe_allow_html=True)
    num_players = st.selectbox('', sorted(df['min_players'].dropna().unique()), key='num_players')
...
```

Epic 2: Sprachumstellung

Tasks

2.1 Sprache der Benutzeroberfläche übersetzen

- Auswahl der englischen Sprache, damit die App universeller zu bedienen ist
- Alle UI-Elemente, wie Buttons, Fragen und Anweisungen, werden ins Englische übersetzt

2.2 Formulierung anpassen

- Sprache wurde authentisch zum Thema angepasst

```
st.markdown('<div class="custom-input-label">The ritual begins. Answer my questions truthfully, and the magic shall unfold. Is there already a game that stirs your soul, favorite among your collection?</div>', unsafe_allow_html=True)
```

Epic 3: UI-Verbesserungen

Tasks

3.1 Optimierung der Farben für Textfelder

- Verbesserung der Lesbarkeit durch Anpassung der Design Einstellungen

```
st.markdown("""
 <style>
 .custom-input-label {
    background: rgb(255,192,118);
    border-radius: 20px;
    background: radial-gradient(circle, rgba(255,192,118,1) 0%, rgba(255,70,0,1) 100%);
    text-shadow: 2px 2px 4px #000000;
    border-color: black;
    border-width: 6px;
    border-style: solid;
    color: rgba(255,249,243,1);
    padding: 10px;
    text-align: center;
    margin-bottom: 10px;
 }
 </style>
""", unsafe_allow_html=True)
```

3.2 Überarbeitung des Tabellen-Layouts

- Optimierung der Darstellung der Spielempfehlungen in tabellarischer Form
- Bearbeitung in styles.py

- Laden und Anwendung in boardgameapp.py

```
st.markdown(
        """
        <style>
    .styled-table {


    }
    .styled-table th, .styled-table td {
        text-align: center;
        padding: 8px;
        font-size: 22px;
         width: 80/%;
        margin: auto;
        border-collapse: collapse;
        text-align: center;
        background: radial-gradient(circle, rgba(255,192,118,1) 0%, rgba(255,70,0,1) 100%);
        text-shadow: 2px 2px 4px #ff0000;
        border-color: black;
        border-width: 6px;
        border-style: dashed;
        color: white;
    }
    </style>
    """,
        unsafe_allow_html=True
    )
```

Epic 4: Erweiterung des Empfehlungsalgorithmus für mehrere Spiele

Tasks

4.1. Implementierung der Mehrfachauswahl

- Hinzufügen einer Funktion, die es Benutzern ermöglicht, mehrere Lieblingsspiele auszuwählen
- Funktion get_recommendations_by_favorite_game() nimmt eine Liste von Lieblingsspielen als Eingabe

4.2. Anpassung des Empfehlungsalgorithmus

- Erweiterung des Algorithmus, um die Mehrfachauswahl der Spiele zu berücksichtigen
- Input Vector wird aus den Merkmalen der Lieblingsspiele extrahiert
- KNN sucht die 5 nächsten Nachbarn zu den Vektoren der Lieblingsspiele

```
def get_recommendations_by_favorite_games(df, favorite_games):
    X = df[['names', 'min_players', 'max_players', 'duration_category', 'category', 'weight', 'age']]
    y = df['names']


    favorite_games_df = df[df['names'].isin(favorite_games)]


    if favorite_games_df.empty:
        return pd.DataFrame()


    X = pd.get_dummies(X, columns=['duration_category', 'category', 'weight', 'age'])
```

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X.drop(columns=['names']))

favorite_games_vectors = X[X['names'].isin(favorite_games)].drop(columns=['names'])

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_scaled, y)

recommendations = knn.kneighbors(favorite_games_vectors, return_distance=False)
recommended_games = df.iloc[recommendations[0]]

return recommended_games[['names', 'bgg_url']]
```

Epic 5: Dokumentation des Projekts

Tasks

5.1. Erstellung der Projektdokumentation

- Vollständige Dokumentation des Projekts in Word

5.1. Code in GitHub hochladen

- Code in GitHub hochgeladen mit einer technischen Beschreibung in der README Datei
- pip.txt Befehl ausgeführt und Datei hinzugefügt
- Projekt Dokumentation Datei hinzugefügt

Reflexion

Es folgt nun eine Reflexion des Projekts, in der ich zunächst auf die erreichten Zielen eingehe, anschließend die nicht geschafften Punkte darlege und zuletzt mögliche Schritten erläutere.
Eine Streamlit Anwendung wurde erstellt, die es den Usern ermöglicht, Brettspielempfehlungen basierend auf ihren Antworten der Userfragen zu erhalten. Die Daten wurden verarbeitet und kategorisiert und das kNN-Modell konnte implementiert und trainiert werden. Im zweiten sowie dritten Sprint wurden verschiedene Empfehlungsalgorithmus ausprobiert und alternative Modelle wurden implementiert und evaluiert. Neben dem kNN-Modell wurde auch ein Random-Forest-Modell evaluiert. Dabei kamen sowohl ein einem Train Validation Test Split als auch ein Cross Validation Split zum Einsatz. Schlussfolgernd wurde das skalierte k-Nearest Neighbors (kNN) Modell als primäres Modell eingesetzt. Im nächsten Abschnitt gehe ich auf die Punkte ein, die noch nicht umgesetzt werden konnten.
Bisher wurde das kNN-Modell verwendet, ein unüberwachtes Verfahren zur Berechnung von Ähnlichkeiten. Der nächste Schritt wäre herauszufinden, das Modell zu einem überwachten Verfahren weiterzuentwickeln, das durch User Feedback kontinuierlich verbessert wird.
Außerdem hat man aktuell die Möglichkeit ein Lieblingsspiel auszuwählen oder durch den Fragenkatalog passende Features herauszufiltern. Aus Usability Perspektive wäre es sinnvoll gewesen, den Usern die Option zu geben, nur die Features auszuwählen, die für sie auch relevant sind. Dafür müsste man im Hintergrund verschiedene kNN-Modelle vorberechnen, sodass man bei keiner Angabe einer Kategorie trotzdem was empfohlen bekommt.
Bisher ist ein Datensatz aus dem Jahr 2017 verwendet worden. Ein nächster logischer Schritt wäre es gewesen, einen aktuelleren Datensatz, wie etwa den auf Kaggle https://www.kaggle.com/datasets/andrewmvd/board-games aus dem Jahr 2021 zu nutzen, damit Spiele, die nach 2017 veröffentlicht wurden, in die Empfehlungen einfließen.