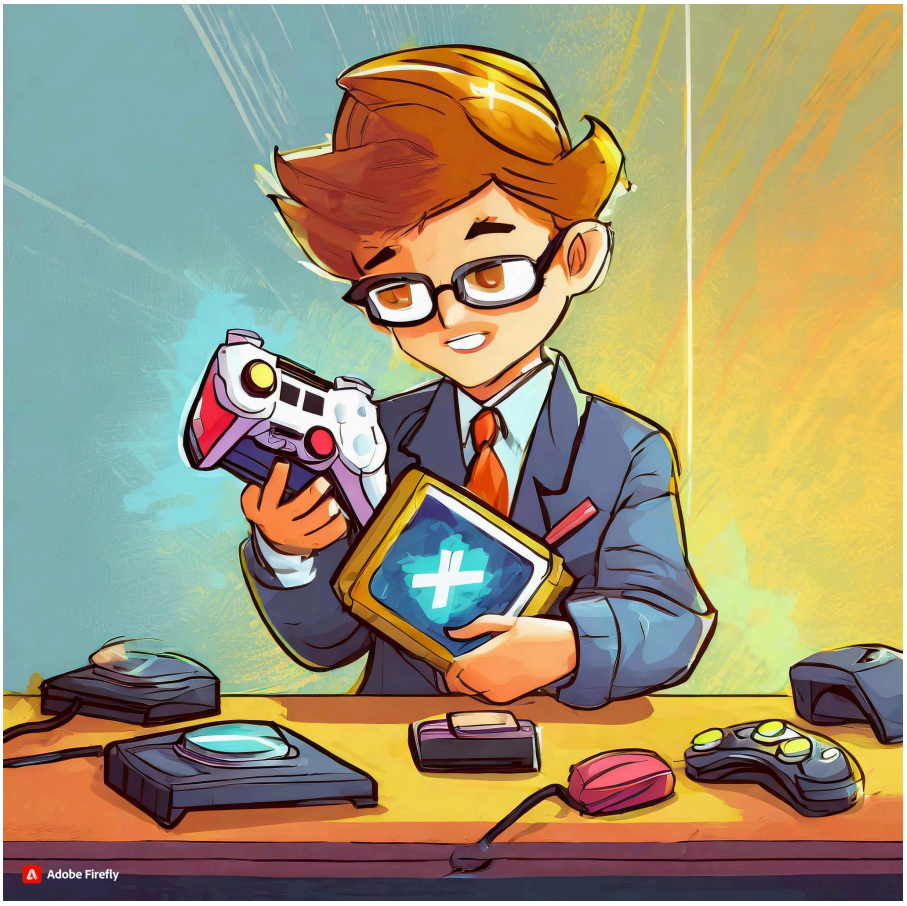
תיכון עירוני מקיף קריית חיים

סיווג הצלחת משחק על פי ביקורות על משחקים מאותו הז'אנר

מוגש ע"י:
רגב כהן
ת.ז:
327870077



שם מנחה: ליטל לשצ'ינסקי
שם מורה: חן פטל

תוכן עניינים

מבוא

בעידן של התקדמות טכנולוגית מהירה, מעט תעשיות הציגו מסלול צמיחה משגשג כמו זה של תעשיית משחקי הוידאו. ממקורותיה הצנועים ועד הפיכתה למעצמה כלכלית ותרבותית עולמית, עבודה זו מתעמקת באבולוציה, בנוף הנוכחי ובסיכויים העתידיים של אחד מהמגזרים הצומחים ביותר בעולם. משחקי וידאו הם בידור אלקטרוני אינטראקטיבי המופעל על פלטפורמות שונות, שמערבבים שחקנים בעולמות וירטואליים באמצעות התקני קלט. הם משתרעים על פני ז'אנרים שונים ומציעים חוויות סוחפות באמצעות גרפיקה, אודיו ומכניקה. התקדמות הטכנולוגיה בתחום מאפשרת למפתחים ליצור משחקים חדשניים אשר גורמים לשגשוג תעשייתי שמתמשך בתעשיית משחקי הוידאו.

מאז שחר היסטורית משחקי המחשב, היו משחקי מחשב אשר הצליחו יותר ואחרים שנשארו מאחור. כאלה שלא הייתה להם הזדמנות לעלות לבמה הקדמית, שלא הגיע לקהל היעד. ולכן נוצרה השאלה - מהי הנוסחה ליצירת המשחק המושלם? אחד שיצליח בתעשייה?. החל מבחירת הז'אנר ליצירת הגרפיקות הפסקול והמכאניקות הטובות ביותר, האתגר בשאלה זו הוא איך אפשר למצוא את "נוסחה" זו. האם יש דרך לסווג דברים שאנשים מעדיפים על דברים אחרים שיכולים לגרום לעלייה במכירות?.

מטרת הפרויקט שלי היא לסווג, האם משחק יכול להצליח עוד לפני יצירתו על ידי חקירת ביקורות על משחקים מאותו הז'אנר של המשחק החדש ושימוש בפיץ' של המשחק (הפיץ' שהולך למשקיעים ומפרסמים). קהל היעד של הפרויקט הוא חברות משחקי מחשב ומפתחי משחקי אינדי אשר רוצים להפוך את שיטת העבודה שלהם לשיטה האופטימלית ביותר, בכדי להכניס רווח מפיתוח המשחקים שלהם.

השגתי רשימה של משחקי מחשב וביקורות לכל משחק עיבוד הנתונים שינוי נתונים מערכים של שרשרת לערכי זמן או מספרים במקרה הצורך הורדת נתונים ריקים או החלפה שלהם בנתונים אחרים הורדת נתונים כפולים ולאחר מכן להעביר את רשימת הביקורות בעיבוד ראשוני לקראת חישוב ערך הסנטימנט של כל ביקורת. אימון הנתונים והעברתם במודלי סיווג למיניהם בכדי לסווג את הצלחת המשחקים בחינת המודלים ולאחר מכן שימוש במודל נוסף של למידה עמוקה בכדי להגיע לסיווג הצלחת המשחק ולבסוף הכנסת הקוד לממשק משתמש.

הסיבה שבחרתי בנושא זה היא מכיוון שמשחקי מחשב זה התחום האהוב עלי, הוא מעניין בעיניי וזה משהו שארצה לעסוק בו בעתיד. שוק משחקי המחשב בשנים האחרונות התחיל להראות ירידה משמעותית בגלל הבחירות של חברות גדולות בשוק אשר גרמו לקהל היעד של החברות להתאכזב, ובכך המכירות של משחקי המחשב ירדו. עולם הספורט הדיגיטלי גם הוא איבד את רוב הפרסום שלו, כתוצאה מכך שברובו הוא משתמש במשחקים של אותן חברות, העובדה הזאת גרמה לפיטורים של הרבה עובדים ושחקנים מקצועיים שאיבדו את מחייתם בגלל טעויות של החברות השולטות

בשוק. לכן הנושא הזה חשוב לי מכיוון שאני רוצה למצוא פתרון אשר החברות הגדולות יוכלו לקחת ולהשתמש בו, על מנת למנוע טעויות שפוגעות בשוק משחקי המחשב, ולפנות לקהל היעד שלהם בצורה הטובה ביותר.

בתחילת שנה שעברה (2023) החיזויים כלפי שוק משחקי המחשב היה שעד שנת 2033 השוק יעלה משווי של 250 מיליארד דולר ל650 מיליארד דולר וחיזויים לגבי שוק המשחקים ספציפית למחשב (PC) אמור להגיע בשנת 2027 למישור בשווי של 40 מיליון דולר בגלל החלטות החברות הגדולות אשר שולטות בשוק הזה הסיבה ליצירת המישור הזה הוא איבוד האמון של אנשים רבים בספורט האלקטרוני של אותם משחקים אשר אפשרי לשחק אותם רק במחשב וירידה דרסטית במחירות של משחקי הAAA (משחקים הנוצרים על ידי חברות גדולות ולא משחקים הנוצרים על ידי מפתחים בודדים אשר נקראים INDIE).

אתגרים איתם אני מתמודד בכתיבת הפרויקט הם: מציאת דרך לחשב את הסנטימנט של כל ביקורת, ולהשיג את הביקורות של כל משחק לתוך מבנה הנתונים שלי. הצורך אותו הפרויקט אמור לספק הוא הפחתת הזמן ביצרת משחקים, שאפשר מראש אפשר לסווג שלא יצליחו בשוק. זה יתן זמן והשקעה יותר גדולה במשחקים אשר אפשר לסווג שהקהל יאהב ירכוש. הפתרון לצורך זה אשר מגיע מהפרויקט הוא דרך קלה לסווג האם המשחק יהיה טוב או לא בעיני הקהל, וכך יוריד משחקים שאין להם עתיד כבר בשלב הרעיוני לפני תחילת פיתוח המשחק.

ישנן 2 אפשרויות אותן העלתי לאיך לפתור בעיה זו. הפתרון הראשון אשר איתו אני הולך בפרויקט הוא חקירת הביקורות אשר נכתבות על כל משחק, ודירוג הסנטימנט אשר כל ביקורת מביאה. זאת על מנת לסווג את הצלחת המשחק. אפשרות השנייה אשר לא נכנסה לפרויקט היא חיזיון הצלחה של המשחק לפי כמות המכירות של משחקים מאותו הז'אנר. הסיבה שאופציה זו לא נלקחה היא בגלל שהמכירות משתנות בהרבה בין אם המשחק הוא משחק הנוצר על ידי חברה מוכרת וגדולה, או על מפתחים פחות מוכרים בשוק. זה יוצר מצב בו משחקי מחשב ממפתחים פחות מוכרים תמיד יחשבו כלא מצליחים ומשחקים מחברות שיש להם קהל רחב יחשבו כמצליחים מה שלא נכון בהכרח.

מבנה

שלב האיסוף

שלב איסוף הכנה וניתוח הנתונים

(Collect, data analyze and prepare):

מבנה הנתונים נלקח מKaggle, הוא נוצר על ידי שימוש ב STEAM API כדי לייצר רשימה של משחקים מתוך הספרייה של STEAM.

STEAM היא חנות אינטרנטית, המשמשת בתור פלטפורמת משחקי מחשב המאפשרת ריכוז של משחקים שנוצרו על ידי מפתחים שונים לתוך חנות אחת אינטרנטית הנותנת את האפשרות להריץ את המשחקים דרך השרתים שלה.

עוד נתונים נאספו על ידי שימוש בSTEAM API שאפשר לי לקחת את כל הביקורות על המשחקים שקיבלתי ממבנה הנתונים של Kaggle.

- תיאור וניתוח הנתונים הגולמיים.

Feature	Definition
Recommendation ID:	מזהה ייחודי זה עוזר במעקב ובניהול המלצות בודדות. הוא משמש כנקודת התייחסות לניתוח משוב ספציפי של משתמשים, סנטימנטים ודפוסי מעורבות.
SteamID:	ה-SteamID חיוני לקישור המלצות למשתמשים בודדים בפלטפורמת Steam. הוא מספק אמצעי ליצור פרופיל של היסטוריית המשחקים של המשתמשים, ההעדפות והפעילות הכוללת.
Number of Games Owned:	מציין את חווית המשחק וההיכרות של המשתמש עם ז'אנרים שונים במשחק. למשתמש שבבעלותו מספר גדול יותר של משחקים עשוי להיות פרספקטיבה מגוונת יותר על נקודות החוזק והחולשה של המשחק.
Number of Reviews Written:	מדד זה מדגיש את רמת המעורבות של המשתמש בקהילה. סביר להניח שמשתמשים שכותבים ביקורות לעתים קרובות משקיעים יותר באפליקציה ובפלטפורמה כולה.
Lifetime Playtime:	מציין את מחויבות המשתמש לאפליקציה. זמן משחק גבוה יותר מעיד על מעורבות והנאה חזקה.
Playtime in the Last Two Weeks:	חושף את רמות הפעילות האחרונות של המשתמשים. זמן משחק תכוף על פני תקופה קצרה יכול להצביע על עניין ומעורבות מתמשכים.

Playtime at Review:	מספק הקשר לתוכן הביקורת. משתמש שכותב ביקורת מיד לאחר סשן משחק ממושך עשוי לקבל תובנות חדשות לשתף.
Last Played:	מציע הצצה לאינטראקציה האחרונה של המשתמש עם האפליקציה. זה יכול לספק הקשר לעדכניות הביקורת שלהם.
Review Text:	החלק הכי חשוב במשוב המשתמשים, ביקורות מספקות תובנות לגבי מה שהמשתמשים מעריכים, לא אוהבים ורוצים באפליקציה. הם עוזרים למפתחים להבין נקודות כאב ספציפיות וחוזקות.
Timestamp Created:	מציין מתי נכתבה הביקורת בתחילה, ועוזר לעקוב אחר התפתחות סנטימנט המשתמשים לאורך זמן.
Timestamp Updated:	מדגים את רמת המעורבות של המשתמש. סקירה שעודכנה עשויה להצביע על היענות לפעולות מפתח או למחשבות מתפתחות.
Voted Up:	מציין אם סנטימנט הביקורת חיובי. ביקורות חיוביות הן אינדיקטורים חשובים לשביעות רצון המשתמשים.
Votes Up:	משקף קונצנזוס בקהילה על ביקורות מועילות. ספירת קולות גבוהה יותר מעידה על הסכמה רחבה יותר עם נקודת המבט של המשתמש.
Votes Funny:	מוסיף אלמנט חברתי לביקורות, מה שמציע על מעורבות מעבר לעזרה בלבד.
Weighted Vote Score:	מספק מדד מצטבר של הערכת משתמשים. עוזר למפתחים להתמקד בביקורות עם ההשפעה הגדולה ביותר.
Comment Count:	מציין את יכולת הביקורת לייצר דיון ומעורבות בתוך הקהילה.
app_id:	מגדיר מספר מזהה לכל אפליקציה/משחק.
title:	מציג את השם הרשמי של המשחק.
developer:	מציג את שם החברה/קבוצה שיצרו את המשחק.
publisher:	מציג את שם החברה/קבוצה שהוציאו את המשחק לאור.
release_date:	מציג את התאריך בו המשחק יצא לאור.

Recommended:	המשתנה המציג האם המשתמש המגיב ממליץ על המשחק או לא (משתמש בערכים של True או False).
Action:	משחק בקצב מהיר עם דגש על רפלקסים ולחימה.
Adventure:	חקר ופתרון חידות מבוססי סיפור בעולמות סוחפים.
Casual:	משחקים פשוטים וקלים להפעלה למשחק רגוע ולא תחרותי.
Early Access:	משחקים בפיתוח, המאפשרים לשחקנים לבדוק ולהשפיע על ההתקדמות שלהם.
Indie:	משחקים שנוצרו על ידי מפתחים עצמאיים, לרוב עם קונספטים ייחודיים.
Massively Multiplayer:	משחקים מקוונים עם שחקנים רבים המקיימים אינטראקציה בו זמנית.
RPG:	משחק מונע על ידי דמות עם אפשרויות שמשפיעות על הסיפור וההתקדמות.
Simulation:	סימולציות מציאותיות, לרוב מורכבות, של תרחישים אמיתיים או בדיוניים.
Strategy:	חשיבה טקטית ותכנון להשגת יעדים במסגרות שונות.

טיפול בנתונים

הורדתי את כל הביקורות שאין להן העלאות, וביקורות שיש להן מעל 60 ביקורות, מכיוון שראיתי בגרפים שיצרתי שאחרי 60 נותר מספר מאוד קטן של ביקורות שמצליחות לקבל יותר והן יכולות להשפיע על הנתונים לרעה.

בנוסף, הורדתי את הפיצ'רים שבגרפים נמצא שאין בהם נתונים כמעט וכלל. הסיבה לכך הייתה שארבעת הפיצ'רים האלה היו חסרי נתונים. פיצ'רים אלה התחלקו לשני סוגים:

- פיצ'רים רק לסין בגלל החוקים והרגולציות שיש לסין על משחקי מחשב.
- פיצ'רים על תגובות המפתחים לביקורות.
-

מכיוון שרוב המשחקים במבנה הנתונים נוצרו על ידי מפתחי משחקי AAA, המפתחים לא יכולים לייצר תקשורת עם הקהל. לכן, רק מספר קטן של תגובות של מפתחים שלא עובדים בחברות משחקי AAA.

ביקורות כפולות נמחקו בעזרת מפתח הזהות של הביקורת שנמצא בפיצ'ר "Recommendation ID".

יצרתי גרפים שחקרו את הקורלציה של המשתנים בדאטה פריים:

- הז'אנרים של המשחקים לעומת שנת היציאה.
- התפלגות ההמלצות או ה"עלאות".
- תמיכה לנגד בידור בדירוג ביקורות.

גרפים אלו יעזרו לי לסנן מידע לא נחוץ ובכך אוכל להכין מודל טוב יותר שיביא חיזויים יותר מדוייקים.

שלב נרמול הנתונים בפרוייקט שלי היה שונה מהרגיל:

השתמשתי בשיטת נרמול לא רגילה למען נרמול הנתונים שלי,
עשיתי שימוש ב Sentiment Intensity Analyzer בכדי לנרמל את הנתונים שלי.

SentimentIntensityAnalyzer הוא מודל המבצע את פעולת הניתוח של הסנטימנט בעזרת קביעת הטון הרגשי של הטקסט בעזרת ניקוד הסנטימנט בטווח בין -1 ל1 כאשר 0 הוא ניקוד ניטרלי לגמרי. ככל שהמספר מתקרב לאחד מהקצוות, התוצאה תהיה חיובית או שלילית בהתאמה המודל

הדרך בה SentimentIntensityAnalyzer מבצע את הפעולה היא בכך שהוא מנתח את עוצמת הסנטימנט בעזרת גישה המבוססת לקסיקון, כלומר הוא מסתמך על מילון מילים עם ציוני הסנטימנט התואמים להן.
המנתח מפרק את הטקסט למילים בודדות, מחפש את ציוני הסנטימנט שלהן במילון, ולאחר מכן משלב את הציונים הללו כדי לקבוע את הסנטימנט הכולל של הטקסט.

המנתח יכול גם לקחת בחשבון את עוצמת הסנטימנט, שהיא המידה שבה הסנטימנט בא לידי ביטוי. לדוגמה, למילה "מדהים" יש עוצמת סנטימנט גבוהה יותר מהמילה "טוב".

הסיבה שלא השתמשתי בשיטת נרמול רגילה כמו MinMaxScaler היא שהפיצ'רים בהם היה השימוש הסופי במודל הם: האם המשחק מומלץ או לא, ו3 קטגוריות שמכילות את הניקוד בין -1 ל 1 שמודל הSIA הביא לכל ביקורת.
שימוש בMinMax Scaler יבטל את ניקוד הסנטימנט מכיוון שMinMax Scaler גורם לנתונים לזוז בין 0 ל1 וכך הוא ישנה את הסנטימנט של המודל.

בזמן שSentimentIntensityAnalyzer מייצר את החישובים של הסנטימנט הביקורות עוברות עיבוד על ידי מודל sentence transformer מSBERT שממיר את הביקורות לembeddings של מספרים במרחב הוקטורי ועליהם המודל SIA עושה את החישוב

המודל paraphrase-albert-small-v2 משמש כדי לקודד משפטים להטבעות וקטוריות, אשר משמשות לאחר מכן לחישוב הדמיון בין משפטי הסקירה למשפטי הקטגוריה.

הטמעות קטגוריות: המודל paraphrase-albert-small-v2 משמש כדי לקודד את משפטי הקטגוריה להטבעות וקטוריות. משפטי הקטגוריה הם ['חזותיים', 'פסקול', 'משחק']. ההטבעות המקודדות מאוחסנות ברשימת categories_emb.

הטמעות סקירה: עבור כל משפט סקירה, נעשה שימוש במודל כדי לקודד את המשפט להטבעה וקטורית. ההטמעה המקודדת מאוחסנת במשתנה r_emb.

חישוב דמיון: הדמיון הקוסינוס מחושב בין הטמעת הסקירה r_emb לבין כל קטגוריה המטביעה c_emb ברשימת הקטגוריות_emb. הקטגוריה בעלת הדמיון הגבוה ביותר (כלומר, המרחק הנמוך ביותר) נבחרה כקטגוריה המתאימה ביותר לביקורת.

ניתוח סנטימנט: הסנטימנט של משפט הסקירה מחושב באמצעות הפונקציה sia.polarity_scores. ציון הסנטימנט מאוחסן במשתנה sen.

צבירת סנטימנטים בקטגוריה: ציוני הסנטימנט עבור כל קטגוריה מצטברים על ידי הוספת ציון הסנטימנט לרשימת הקטגוריות המקבילות במילון cat_dic.

חישוב סנטימנט סופי בקטגוריה: ציוני הסנטימנט הסופיים עבור כל קטגוריה מחושבים על ידי לקיחת הממוצע של ציוני הסנטימנט עבור כל קטגוריה. ציוני הסנטימנט הסופיים מאוחסנים במילון final_cat_dic.

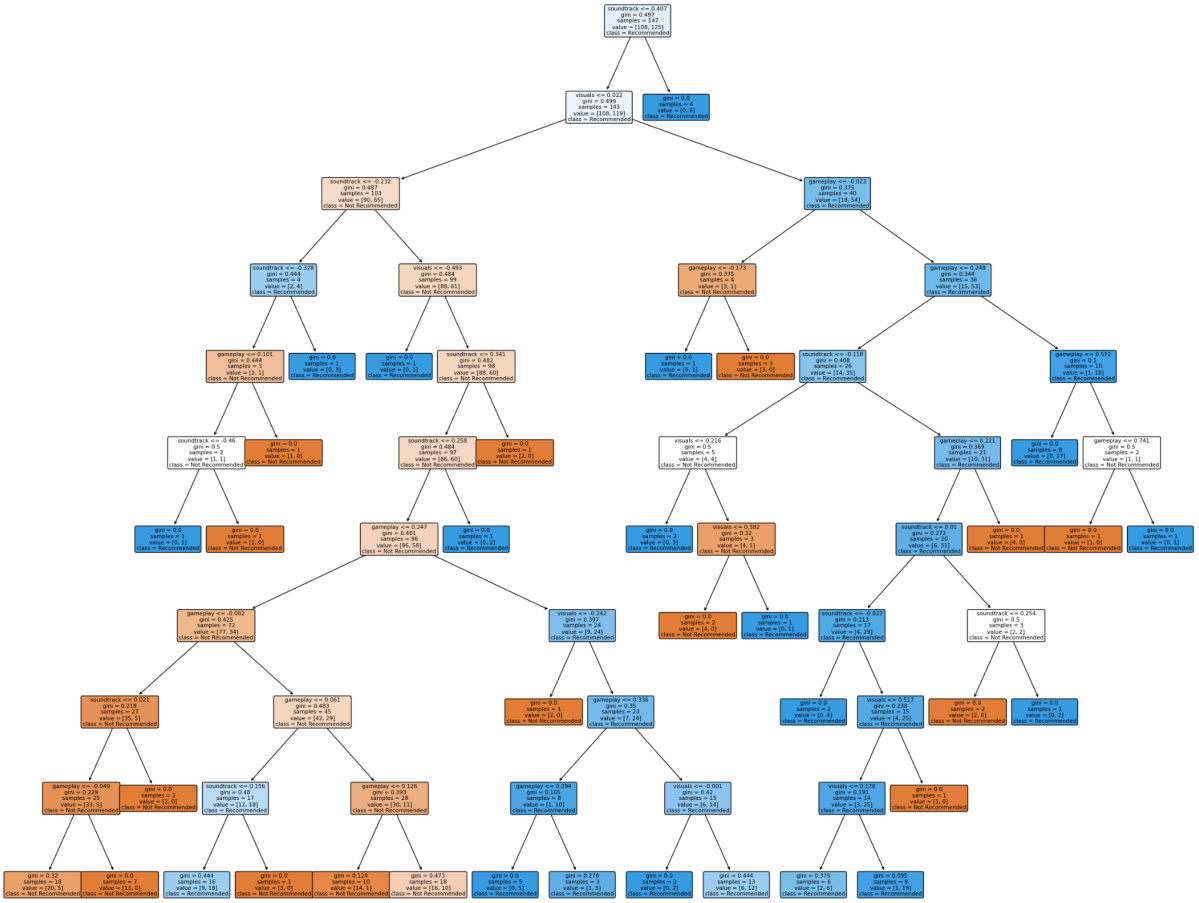
שלב בניית המודל

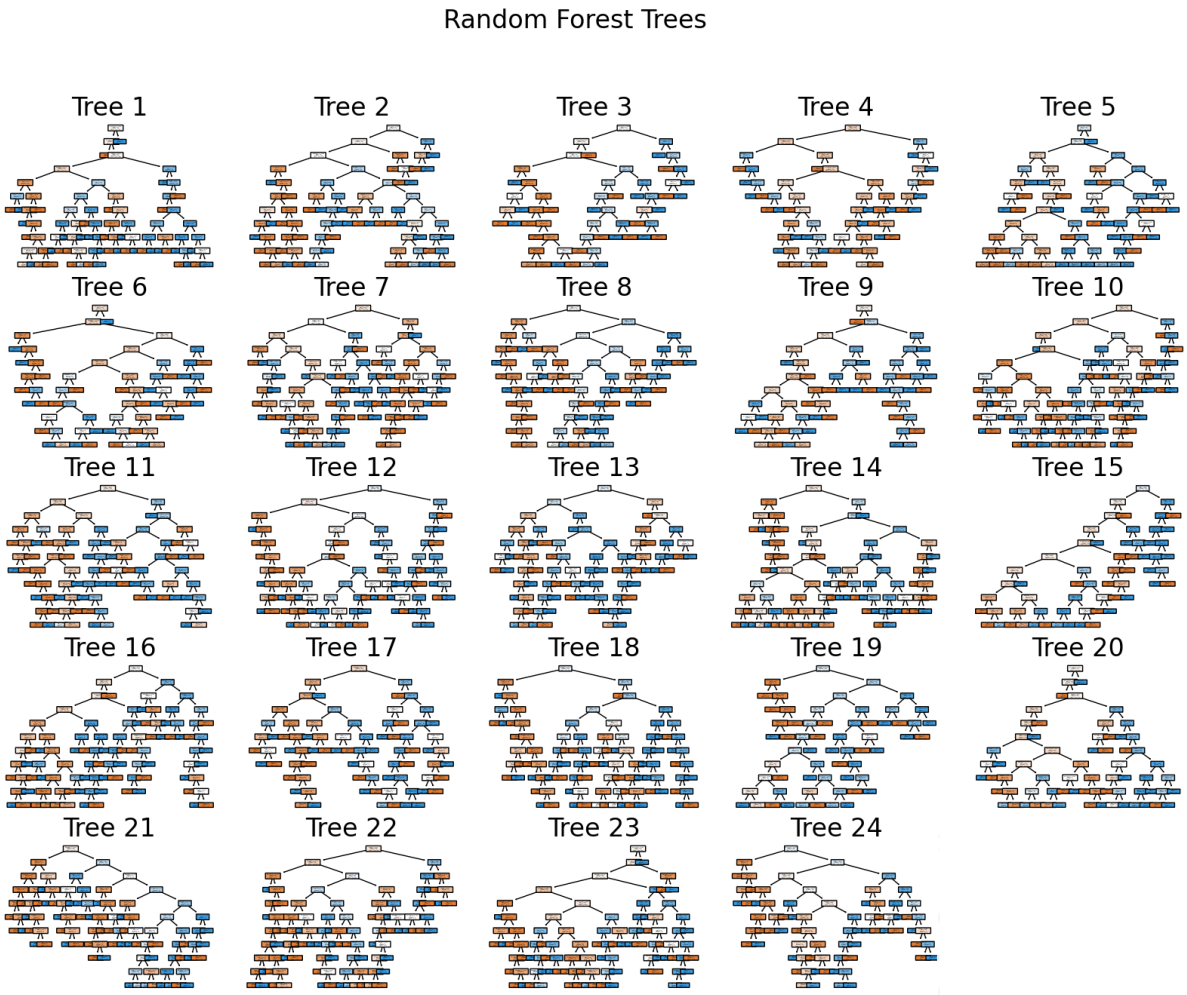
המודל ואימון בנייה שלב

:Build and train deep learning model

- **תיאור מודל RandomForestClassifier.**

1 tree out of the whole forest taken as an example





ה-Random Forest Classifier הוא אלגוריתם למידת מכונה בשימוש נרחב המשתמש בשיטות למידה מפוקחות. ניתן ליישם אותו הן על בעיות סיווג והן על בעיות רגרסיה. האלגוריתם מבוסס על למידת אנסמבל, המשלבת מספר מסווגים לפתרון בעיה מורכבת ומגבירה את ביצועי המודל.

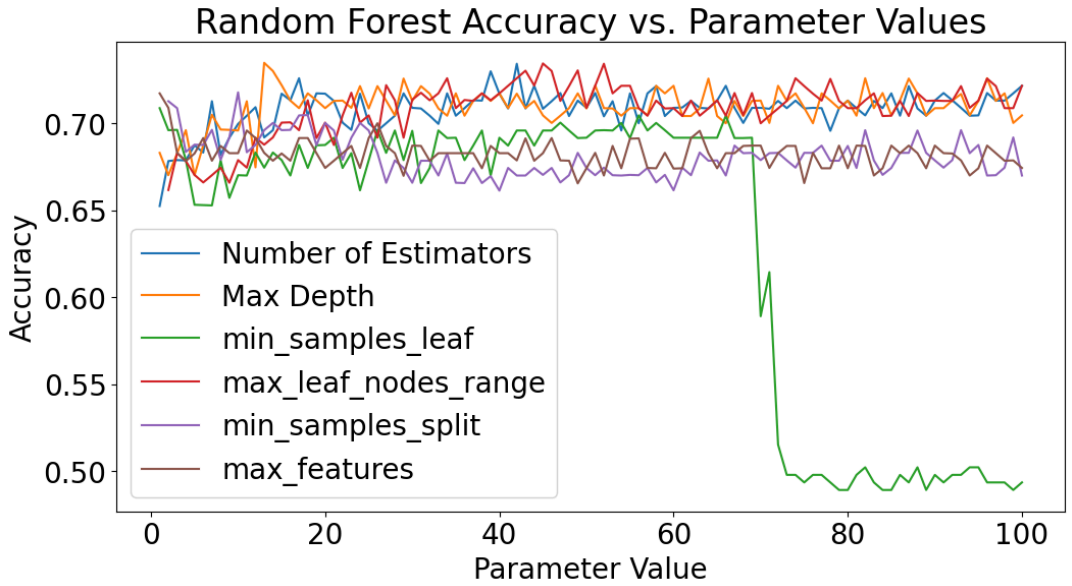
במילים פשוטות, מיון היער האקראי הוא אוסף של עצי החלטה על תת-קבוצות שונות של מערך נתונים נתון. במקום להסתמך על עץ החלטות בודד, היער האקראי אוסף את התוצאה מכל עץ ומצפה לתוצאה סופית בהתבסס על קולות הרוב של התחזיות.

ניתן לחלק את תהליך העבודה של אלגוריתם היער האקראי לשני שלבים:

שילוב N עצי החלטה: השלב הראשון כולל שילוב N עצי החלטה עם בניית היער האקראי.
ביצוע תחזיות: השלב השני כולל ביצוע תחזיות עבור כל עץ שנוצר בשלב הראשון.
השלבים להדגמת תהליך העבודה הם:

בחר M נקודות נתונים באקראי מתוך ערכת האימונים.
צור עצי החלטה עבור נקודות הנתונים שבחרת (קבוצות משנה).
כל עץ החלטות יפיק תוצאה. נתח את זה.
עבור סיווג ואו רגרסיה, בהתאם, התוצאה הסופית מבוססת על הצבעת רוב או ממוצע, בהתאם.

- **דוחות וגרפים המתארים את תוצאות שלב האימון.**



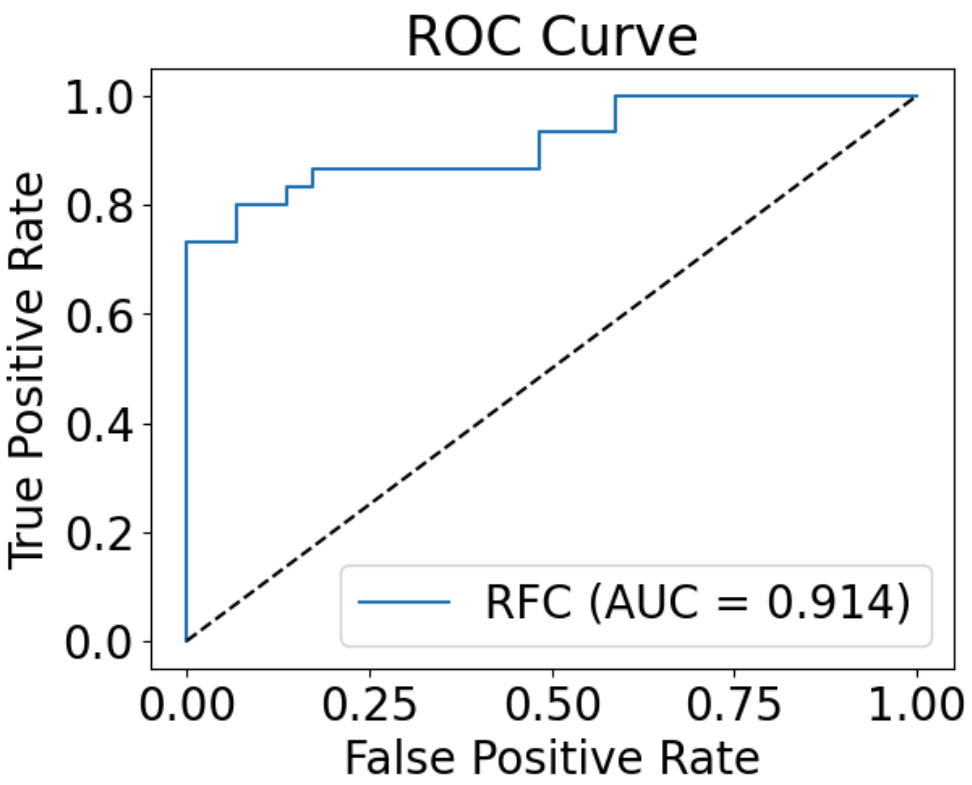
\

"Random Forest Accuracy vs. Parameter Values". גרף זה ממחיש כיצד הדיוק של מודל יער אקראי מושפע מהיפר פרמטרים שונים. להלן הסבר מפורט:

ציר Y (דיוק): נע בין 0.50 ל-0.70, המציין את אחוז הדיוק של המודל.
ציר X (ערך פרמטר): נע בין 0 ל-100, המייצג את הערך שהוקצה לכל פרמטר.
הגרף כולל שש קווים, כל אחד מייצג את ההשפעה של פרמטר אחר על דיוק המודל:

קו כחול: Number of Estimators
קו כתום: Max Depth
קו ירוק: min_samples_leaf
קו אדום: max_leaf_nodes
קו סגול: min_samples_split
קו חום: max_features
כל שורה מראה כיצד שינויים בערכי הפרמטרים משפיעים על דיוק המודל. יש לציין שהקו הירוק של "min_samples_leaf" מראה ירידה משמעותית ברמת הדיוק כאשר ערך הפרמטר מתקרב ל-80. זה מרמז שהפרמטר "min_samples_leaf" רגיש ויכול להשפיע באופן דרסטי על ביצועי המודל.

גרף זה חשוב לכוונון ההיפר פרמטרים של המודל Random Forest להשגת דיוק מיטבי. זה עוזר לזהות את הגדרות הפרמטר הטובות ביותר עבור המודל בהתבסס על מערך הנתונים והמשימה הספציפיים בהישג יד.



עקומת Receiver Operating Characteristic (ROC) , המשמשת להערכת הביצועים של מודל סיווג. להלן פירוט מפורט:

עקומת ROC: גרף זה משרטט את השיעור החיובי האמיתי (TPR) מול השיעור החיובי השקרי (FPR) בהגדרות סף שונות.
צירים:
ציר הX: נע בין 0 ל-1, המייצג את שיעור המקרים השליליים שסווגו באופן שגוי כחיוביים.
ציר הY: גם הוא נע בין 0 ל-1, המציין את שיעור המקרים החיוביים שזוהו כהלכה.
קו מדורג כחול: עם התווית "RFC (AUC = 0.914)", הוא מציג את הביצועים של מודל RFC.
ערך השטח מתחת לעקומה (AUC) של 0.914 מצביע על כך שלמודל יש דיוק חיזוי טוב.
קו אלכסוני מקווקו: מייצג את הביצועים של מסווג אקראי. עקומת ROC של דגם שימושי צריכה להיות מעל קו זה.

```
Accuracy: 0.59322033898305 08
Precision: 0.59375
Recall: 0.6333333333333333
F1-score: 0.6129032258064516
```

לפני שינוי בהיפר פרמטרים

```
Accuracy: 0.6779661016949152
Precision: 0.6341463414634146
Recall: 0.8666666666666667
F1-score: 0.7323943661971831
```

אחרי שינוי בהיפר פרמטרים

- דוח הכולל ריכוז כל ה- HyperParameters

n_estimators:	מספר העצים ביער. הסבר: הגדלת מספר העצים ביער משפרת בדרך כלל את ביצועי המודל, אך היא גם יכולה לייקר את החישוב.
max_depth:	העומק המרבי של העץ. הסבר: פרמטר זה שולט בעומק המרבי של העץ. עצים עמוקים יותר יכולים להציג קשרים מורכבים יותר בנתונים, אך יש סיכוי גבוה יותר ל overfit.
max_features:	מספר התכונות שיש לקחת בחשבון כאשר מחפשים את הפיצול הטוב ביותר. הסבר: פרמטר זה קובע את המספר המרבי של תכונות של-Random Forest מותר לנסות בצמתי עצים בודדים. ניתן לציין אותו כמספר שלם (מספר התכונות שיש לקחת בחשבון) או כצף (שבריר ממספר התכונות הכולל).
min_samples_leaf:	מניעת צומת עלים מלהיות קטן מדי הסבר: פרמטר זה מבטיח שלכל צומת עלים בעץ יש לפחות מספר מינימלי של דוגמאות. אם לצומת עלה יש פחות דגימות מסף זה, הוא ימוזג עם צומת האח שלו. זה מונע מהעץ להתאים יתר על המידה לנקודות נתונים בודדות ומשפר את יכולת ההכללה של המודל.

- **תיעוד כל השינויים שנעשו במודל וב Parameters Hyper לשיפור תוצאות האימון.**

חיפוש רשת עבור היפרפרמטרים
'n_estimators', 'max_depth', 'min_samples_leaf',
'max_features', 'min_samples_split', 'max_leaf_nodes':
טווח ערכים מ-1 עד 100 עם שלב 1 נקבע עבור ההיפרפרמטרים בחיפושי הרשת(grid) השוני שביצאתי מ1-6.

בעבור 'n_estimators' המשמעות היא שחיפוש הרשת ינסה מספר שונה של עצים ביער מ-1 עד 100 וימצא את מספר העצים הטוב ביותר שנותן את הדיוק הגבוה ביותר. הגדלת מספר העצים ביער יכולה לשפר את ביצועי המודל, אך היא גם מגדילה את זמן החישוב. לכן, חשוב למצוא את המספר האופטימלי של עצים שמאזן את ההחלפה בין ביצועי המודל וזמן החישוב.

בעבור 'max_depth' המשמעות היא שחיפוש הרשת ינסה עומקים מקסימליים שונים של העצים ביער מ-1 עד 100 וימצא את העומק המרבי הטוב ביותר שנותן את הדיוק הגבוה ביותר. הגדלת העומק המקסימלי של העצים יכולה לשפר את ביצועי הדגם, אך היא גם יכולה להוביל להתאמת יתר. לכן, חשוב למצוא את העומק המקסימלי האופטימלי שמאזן את ההחלפה בין ביצועי הדגם להתאמה יתר.

עבור 'min_samples_leaf', חיפוש הרשת ינסה מספרים מינימליים שונים של דגימות הנדרשות כדי להיות בצומת עלה. פרמטר זה יכול גם להשפיע על התאמת יתר. אם המספר המינימלי של דגימות שנדרש להיות בצומת עלה נמוך מדי, המודל עשוי להתאים לנתוני האימון קרוב מדי ולהציג ביצועים גרועים בנתונים שלא נראים. הגדלת פרמטר זה יכולה לסייע במניעת התאמה יתרה, אך היא גם עלולה להגביר את הסיכון לחוסר התאמה אם הוא מוגדר גבוה מדי.

עבור 'max_leaf_nodes', חיפוש הרשת ינסה מספרים מקסימליים שונים של צמתים עלים עבור כל עץ ביער. פרמטר זה יכול לסייע בשליטה במורכבות העצים ביער. אם המספר המרבי של צמתים עלים מוגדר נמוך מדי, ייתכן שהעצים יהיו פשוטים מדי ולא מסוגלים ללכוד דפוסים חשובים בנתונים. מצד שני, אם המספר המרבי של צמתים עלים מוגדר גבוה מדי, העצים עלולים להפוך למורכבים מדי ונוטים להתאים יתר על המידה.

עבור 'min_samples_split', חיפוש הרשת ינסה מספרים מינימליים שונים של דגימות הנדרשים כדי לפצל צומת פנימי. פרמטר זה יכול גם להשפיע על התאמת יתר. אם המספר המינימלי של דגימות הנדרש לפיצול צומת פנימי נמוך מדי, המודל עשוי להתאים לנתוני האימון קרוב מדי ולהציג ביצועים גרועים בנתונים בלתי נראים. הגדלת פרמטר זה יכולה לסייע במניעת התאמה יתרה, אך היא גם עלולה להגביר את הסיכון לחוסר התאמה אם הוא מוגדר גבוה מדי.

עבור 'max_features', חיפוש הרשת ינסה מספרים שונים של תכונות שיש לקחת בחשבון כאשר מחפשים את הפיצול הטוב ביותר. פרמטר זה יכול להשפיע על ביצועי המודל ועל זמן החישוב. אם מספר התכונות גבוה מדי, המודל עשוי להימשך זמן רב להתאמן. מצד שני, אם מספר התכונות נמוך מדי, ייתכן שהמודל לא ילכוד דפוסים חשובים בנתונים. לכן, חשוב למצוא את המספר האופטימלי של תכונות שמאזן את ההחלפה בין ביצועי המודל וזמן החישוב.

בחירת ההיפר פרמטרים הטובים ביותר: ההיפר פרמטרים הטובים ביותר נבחרו על סמך ציון הדיוק הגבוה ביותר שהתקבל מחיפוש הרשת. ההיפר פרמטרים האלה שימשו לאחר מכן ליצירת RFCModel חדש עם הפרמטרים הטובים ביותר.

התאמת RFCModel החדש לכל מערך הנתונים: RFCModel החדש עם ההיפר פרמטרים הטובים ביותר הותאם לכל מערך הנתונים (X ו-y). המשמעות היא שהמודל הוכשר על כל מערך הנתונים כדי ללמוד את הקשר בין התכונות למשתנה היעד.

הערכת ה-RFCModel החדש: הדיוק, הדיוק, ההחזרה וציון ה-F1 של ה-RFCModel החדש חושבו על נתוני הבדיקה (X_test ו-y_test). מדדים אלה שימשו להערכת ביצועי המודל על נתונים בלתי נראים ולהשוואתם עם מודלים אחרים.

- **תיעוד והסבר של פונקציית השגיאה.**

פונקציית של השגיאה במודל ששמו RandomForestClassifier הוא Gini Impurity, בעצם זהו מדד ל"ליכלוך/רעש", במילים אחרות זוהי האי הוודאות של צומת בעץ ההחלטות. הסיבה לשימוש בו הוא בשביל לקבוע את הפיצול הטוב ביותר בכל צומת. Gini Impurity מחושב בדרך הבאה:

$$\text{Gini Impurity} = 1 - \sum (p_i^2)$$

מתוך החישוב של המדד, הביטוי p_i מבטא את השיעור של הדגימות ששייכות למחלקה i בצומת.

Gini Impurity הוא נע בין 0 (שמבטא צומת טהור לחלוטין) ל-1 (שמבטא צומת לא טהור לחלוטין). הסקלה בין 0-1 בעצם מסבירה ש
Gini Impurity שהיא נמוכה יותר מעידה על צומת אחידה יותר, בעוד שככל שהיא גבוהה יותר מעידה על צומת מגוונת יותר.

בהקשר של עצי החלטה, Gini Impurity משמש לכמה דברים:

אחד מהם היא קביעה ובחירה של התכונה הטובה ביותר לביצוע של פיצול בכל צומת.
והשניה היא חישוב של הרווח בטוהר, שזה אומר הפחתה ב"לכלוך" לאחר שבוצע

- **תיעוד והסבר של ייעול ההתכנסות (Optimization).**

בפרוייקט זה השתמשתי באופטמיזציה בעזרת פיצ'ר אינג'נירינג, דאטה פרוססינג, בחירת מודלים ודיוק היפר פרמטרים

Feature Engineering	הנדסת תכונות היא יצירת תכונות חדשות מנתונים קיימים כדי לשפר את ביצועי המודל. דוגמאות בפרויקט: ● יצירת תכונות חדשות על ידי חישוב ציון הסנטימנט הממוצע עבור כל ז'אנר עבור כל משחק. ● שימוש ב-SenenceTransformer כדי לקודד ביקורות ולחשב את הדמיון הקוסינוס בין הביקורות המקודדות לקטגוריות מוגדרות מראש (חזותיים, פסקול, משחק). ● שימוש ב-SentimentIntensityAnalyzer כדי לחשב את ציון הסנטימנט עבור כל סקירה.
Data Processing	עיבוד נתונים הוא ניקוי והפיכת נתונים גולמיים לפורמט המתאים לניתוח. דוגמאות בפרויקט: ● הסרת שורות עם votes_up פחות מ-1.6 או יותר מ-60 כדי להתמקד בביקורות בעלות מספר סביר של העלאות והורדת מקרי קיצון. ● הסרת עמודות עם ערכי NaN גבוהים או עמודות שאינן רלוונטיות לניתוח. ● המרת משתנים קטגוריים למשתנים מספריים באמצעות one-hot encoding.
Model Selection	בחירת מודל היא התהליך של בחירת המודל הטוב ביותר עבור מערך נתונים נתון. דוגמאות בפרויקט: ● שימוש ב-RandomForestClassifier, XGBoost, LightGBM ו-stacking ensemble של XGBoost ו-RandomForestClassifier כדי לחזות אם המשחק יצליח או לא.

HyperParameter Fine Tuning	דיוק היפר פרמטרים הוא תהליך אופטימיזציה של הפרמטרים של מודל כדי לשפר את הביצועים שלו. דוגמאות בפרוייקט: ● שימוש ב-GridSearchCV כדי לכוונן את הפרמטרים ההיפר-פרמטרים של RandomForestClassifier ו-XGBoost. ● שימוש ב-cross_val_score כדי להעריך את הביצועים של המודלים עם מספרים שונים של אומדנים ועומק מרבי.

● תיעוד ההתמודדות עם הטיה ושונות

בתהליך של פיתוח מודל למידת מכונה חזק ואמין, אחד האתגרים המרכזיים שעמדתי בפני היה התמודדות עם הטיות. הטיות אלו, אם לא ייבדקו, עלולות להטות משמעותית את התחזיות של המודל ולהוביל לתוצאות לא מדויקות. כדי לטפל בסוגיה זו, התמקדתי בהתאמת ערכי הרגולציה של המודל. ערכי רגולציה אלו ממלאים תפקיד מכריע בשליטה במורכבות המודל, ובכך עוזרים לשמור על איזון בין הטיה לשונות.

ערכי הרגולציה מציגים בעצם "עונש" לפונקציית השגיאה של המודל. עונש זה מרתיע את המודל מלהתאים יותר מדי לנתוני האימון, בעיה המכונה התאמת יתר. התאמת יתר עלולה להוביל למודל שביצועיו טובים בנתוני אימון אך גרועים בנתונים שהוא עוד לא ראה, מכיוון שהוא מתמחה מדי לדוגמאות הספציפיות בסט האימונים.

לעומת זאת, ערכי הרגולציה גם מונעים תת-התאמה, כאשר המודל פשטני מכדי ללכוד את הדפוסים הבסיסיים בנתונים. התאמה לא טובה יכולה לגרום למודל שאינו מדויק לא בנתוני האימון או בנתונים שלא נראים, מכיוון שהוא לא מצליח ללמוד את הקשרים הדרושים בין התכונות למשתנה היעד.

על ידי כוונון קפדני של ערכי הרגולציה הללו, הצלחתי ליצור מודל שהוא לא רק מדויק אלא גם ניתן להכללה לנתונים חדשים. גישה זו סייעה להבטיח שהמודל הוא אמין וחזק כאחד, המסוגל לבצע תחזיות מדויקות במגוון תרחישים. לפיכך, ערכי הרגולציה שימשו כלי קריטי במאמצים שלי למתן הטיות ולייעל את הביצועים של מודל למידת המכונה.

שלב היישום

ייבוא מודלים: האפליקציה מתחילה בייבוא דגם מאומן מקובץ Pickle. מודל זה, שהוכשר במחברת קולאב, הוא הבסיס לפעולות שהאפליקציה תבצע.

קלט משתמש: המשתמש מזין טקסט לתוך תיבות טקסט בתוך היישום. לאחר מכן טקסט זה מעובד על ידי פעולות שונות.

עיבוד טקסט: הפעולות המופעלות על הטקסט זהות לעיבוד הנתונים שנעשו במחברת הקולאב. השלב הראשון בתהליך זה הוא ניקוי טקסט, הכולל הסרה של כל מידע מיותר או לא רלוונטי מהטקסט.

קיטלוג תיאור המשחק: הטקסט הראשון אשר המשתמש רושם, מייצר את התיאור של המשחק, מקוטלג לאחר מכן לפי שמות הניתנים בתיבת טקסט שנייה. שלב זה מארגן את תיאורי המשחק לניתוח נוסף.

ניתוח סנטימנט: לאחר מכן נעשה שימוש במודל SIA (Sentiment Intensity Analyzer) כדי לחשב את ציון הסנטימנט של תיאור המשחק. ציון זה מחושב עבור שלוש קטגוריות: משחקיות, גרפיקה ופסקול. ציון הסנטימנט מספק מדד לסנטימנט הכולל המובע בתיאור המשחק, עם ציונים נפרדים לכל קטגוריה.

יישום מודל: לבסוף, המודל המאומן מתנהל עם ציון הסנטימנט של התיאור. שלב זה מאפשר למודל לבצע תחזיות או סיווגים על סמך ציוני הסנטימנט.

על ידי ביצוע שלבים אלה, האפליקציה יכולה לעבד ולנתח ביעילות את הקלט של המשתמש, ולספק תובנות חשובות לגבי הסנטימנט המובע בתיאורי המשחק.

ממשק המשתמש הוטמע באמצעות CustomTkinter, ספריית Python המספקת סט של רכיבי GUI מותאמים אישית לבניית יישומי שולחן עבודה. CustomTkinter בנויה על גבי Tkinter, ספריית Python מובנית ליצירת ממשקי משתמש גרפיים. היישום משתמש ב-CTkMessagebox, רכיב תיבת הודעות מותאם אישית מספריית CTkMessagebox, כדי להציג הודעות מידע למשתמש.

תיאור קוד

הקוד הוא סקריפט Python שמסווג אם משחק יצליח בהתבסס על ביקורות וז'אנרים של משתמשים. הוא מורכב ממספר פונקציות:
preprocess_sentences: מעבד משפטים מראש על ידי המרה לאותיות קטנות, הסרת סימני פיסוק ויצירת טוקנים.
predict_sentiment: מנבא סנטימנט עבור סקירה וז'אנר באמצעות ספריית NLTK.

predict_game_features: מנבא תכונות משחק מסקירה וז'אנר, כולל ניתוח סנטימנטים ומיצוי תכונות.

load_rfc_model: טוען מודל RandomForestClassifier (RFC) מקובץ JOBLIB.

predict: עושה חיזוי על סמך הסקירה והז'אנר באמצעות מודל ה-RFC הנטען.

התאמת מבנה נתונים

הקוד מקבל את הנתונים הבאים כקלט:
review: מחרוזת המכילה את התיאור של המשתמש על המשחק.

genre: מחרוזת המכילה את הז'אנרים של המשחק, מופרדים בפסיקים.

הנתונים מותאמים לתוך DataFrame בתוך פונקציית predict_game_features ולאחר מכן הפעולה מחזירה את הDataFrame בעל הפיצ'רים הבאים

review: טקסט התיאור המקורי.

Action, Adventure,..., Strategy: עמודות בינאריות המצביעות על נוכחות או היעדר של כל ז'אנר.

processed_review: רשימה של משפטים מעובדים מראש מהסקירה.

visuals, soundtrack, gameplay: ציוני סנטימנט עבור כל קטגוריה, מחושבים באמצעות הפונקציה predict_sentiment.

DataFrame זה משמש לאחר מכן כקלט עבור מודל ה-RFC לביצוע חיזוי.

מדריך למפתח

שם קובץ: game_review_download_project

מיקום: game_review_download_project.ipynb

```
# Imports
!pip install steamreviews  # Install steamreviews library

import steamreviews  # Import steamreviews library
import pandas as pd  # Import pandas library
import numpy as np  # Import numpy library

from google.colab import drive  # Import Google Drive library
drive.mount('/content/drive')  # Mount Google Drive

pd.set_option('display.max_rows', None)  # Set display options for Pandas DataFrames
pd.set_option('display.max_columns', None)
pd.set_option('display.max_colwidth', None)

# Load data from games.csv
df = pd.read_csv("/content/drive/MyDrive/games.csv")

# Drop unnecessary columns
df.drop(["positive_ratings", "negative_ratings", "price"], axis=1, inplace=True)

# Sample 20% of the data
df = df.sample(frac=0.2, random_state=123)

# Define request parameters for positive and negative reviews
Pos_request_params = dict()  # Positive review request parameters
Pos_request_params['language'] = 'english'
Pos_request_params['review_type'] = 'positive'
Pos_request_params['day_range'] = '28'

Neg_request_params = dict()  # Negative review request parameters
Neg_request_params['language'] = 'english'
Neg_request_params['review_type'] = 'negative'
Neg_request_params['day_range'] = '28'

def cluster_download(data, RP):
    """
    Download reviews for a given app ID and request parameters,
    process review data, and return a combined DataFrame of review and author data.

    Parameters:
```

```
    - data (Pandas DataFrame): Input data containing app ID
    - RP (dict): Request parameters for downloading reviews

    Returns:
    - Pandas DataFrame: Combined review and author data
    """
    result_df = pd.DataFrame()
    for app_id in data['app_id']:
        author_data = []
        review_dict1, query_count1 = steamreviews.download_reviews_for_app_id(app_id,
chosen_request_params=RP)
        Ndf = pd.DataFrame(review_dict1['reviews'])
        Ndf = Ndf.transpose()

        if Ndf.empty:
            continue

        Ndf = Ndf[Ndf['review'].apply(lambda x: len(x.split()) >= 100)]

        for id in review_dict1['reviews']:
            author_dict = review_dict1['reviews'][id]['author']
            author_data.append(author_dict)
        Adf = pd.DataFrame(author_data)
        Ndf['app_id'] = app_id
        Adf['app_id'] = app_id

        Ndf.drop('author', axis=1, inplace=True)

        combined_df = pd.merge(Ndf, Adf, on='app_id', how='inner')
        result_df = pd.concat([result_df, combined_df])

    df = pd.merge(data, result_df, on='app_id', how='inner')
    return df

# Download positive reviews
PDF = cluster_download(df, Pos_request_params)
PDF['Recommended'] = True

# Download negative reviews
NDF = cluster_download(df, Neg_request_params)
NDF['Recommended'] = False

# Concatenate positive and negative reviews
df = pd.concat([PDF, NDF])

# Save reviews to CSV file
df.to_csv('/content/drive/My Drive/reviews.csv', index=False)
```

```
# Download reviews.csv file
from google.colab import files
files.download('reviews.csv')
```

משתנים:

df: אובייקט Pandas DataFrame המאחסן את הנתונים מקובץ games.csv.

Pos_request_params ו-Neg_request_params: מילונים המאחסנים פרמטרים להורדת ביקורות חיוביות ושליליות, בהתאמה.

RP: משתנה המאחסן את פרמטרי הבקשה (או Pos_request_params או Neg_request_params) המועברים לפונקציה cluster_download.

app_id: משתנה המאחסן את מזהה האפליקציה של משחק בקובץ games.csv.

author_data: רשימה המאחסנת נתוני מחבר שחולצו מהביקורות.

review_dict1 ו-query_count1: משתנים המאחסנים את נתוני הביקורות וספירת השאילתות המוחזרים על ידי הפונקציה steamreviews.download_reviews_for_app_id.

Ndf: אובייקט Pandas DataFrame המאחסן את נתוני הסקירה.

Adf: אובייקט Pandas DataFrame המאחסן את נתוני המחבר.

combined_df: אובייקט Pandas DataFrame המאחסן את סקירה ונתוני מחבר משולבים במקום אחד.

result_df: אובייקט Pandas DataFrame המאחסן את התוצאה הסופית של הפונקציה cluster_download.

PDF ו-NDF: אובייקטי Pandas DataFrame המאחסנים את נתוני הביקורת החיוביים והשליליים, בהתאמה.

data: משתנה המאחסן את נתוני הקלט המועברים לפונקציה cluster_download.

פונקציות:

cluster_download(data, RP):
פונקציה שמורידה ביקורות עבור מזהה אפליקציה נתון ופרמטרי בקשה, מעבדת את נתוני הביקורת ומחזירה DataFrame משולב של נתוני סקירה ומחבר.

פרמטרים:
נתונים: נתוני הקלט (אובייקט Pandas DataFrame) המכילים את מזהה האפליקציה.

RP: פרמטרי הבקשה (מילון) המציינים את השפה, סוג הביקורת וטווח הימים להורדת ביקורות.

מחזיר: אובייקט Pandas DataFrame המכיל את נתוני הסקירה והמחבר המשולבים.

steamreviews.download_reviews_for_app_id(app_id, chosen_request_params=RP):
פונקציה מספריית steamreviews שמורידה ביקורות עבור מזהה אפליקציה נתון ופרמטרי בקשה.

פרמטרים:
app_id: מספר מזהה של המשחק.

chosen_request_params: פרמטרי הבקשה (מילון) המציינים את השפה, סוג הביקורת וטווח הימים להורדת ביקורות.

החזרות: TUPLE המכיל את נתוני הביקורת וספירת השאילתות.

שם הקובץ: game_review_project_finished

מיקום: game_review_project_finished.ipynb

```
#Imports

!pip install steamreviews
!pip install -U sentence-transformers
!pip install scikit-learn==1.5.0rc1

import steamreviews
import pandas as pd
import datetime as dt
import numpy as np
from collections import Counter
from wordcloud import WordCloud
import matplotlib.pyplot as plt
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import string
from sentence_transformers import SentenceTransformer, util
from nltk.tokenize import sent_tokenize
import torch
import nltk
from scipy import stats
from nltk.util import ngrams
import seaborn as sns
from sklearn.metrics.pairwise import cosine_similarity
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
import re
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.model_selection import GridSearchCV
```

```
from sklearn.model_selection import KFold, cross_val_score
import nltk
from nltk.sentiment import SentimentIntensityAnalyzer
from scipy.spatial.distance import cosine

nltk.download('vader_lexicon')
nltk.download('punkt')
nltk.download('stopwords')

from google.colab import drive
drive.mount('/content/drive')

pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.max_colwidth', None)


# DF

df = pd.read_csv("/content/drive/MyDrive/reviews.csv")

recommended_column = df.pop('Recommended')

df.insert(2, 'Recommended', recommended_column)


# EDA


unique_values = df.select_dtypes(include="number").nunique().sort_values()
unique_values.plot.bar(figsize=(15, 4),title="Unique values perfeature");

nan_counts = df.isnull().sum()

plt.figure(figsize=(8, 6))
nan_counts.plot(kind='bar', color='skyblue', edgecolor='black')
plt.title("NaN Counts in Each Column")
plt.xlabel("Columns")
plt.ylabel("Number of NaN Values")
plt.xticks(rotation=90)
plt.show()

df.info()

df.isnull().sum().sum()

df.isna().sum().sum()
```

```
print(type(df['release_date'][0]))
print(df['app_id'].unique())
print(df['votes_up'].unique())
print(df['title'].unique())


df['votes_up'].unique()

type(df['voted_up'][0])

df = df[(df['votes_up'] > 1) & (df['votes_up'] < 60)]
df.reset_index(drop=True, inplace=True)


df.describe(exclude=["number","datetime"])

df.drop_duplicates(subset='recommendationid', keep='first', inplace=True)

df['timestamp_created'] = pd.to_datetime(df['timestamp_created'], unit="s")
df['timestamp_updated'] = pd.to_datetime(df['timestamp_updated'], unit="s")
df['release_date'] = pd.to_datetime(df['release_date'])


df.drop(['steam_china_location', 'hidden_in_steam_china'], axis=1, inplace=True)
df.drop(['developer_response', 'timestamp_dev_responded'], axis=1, inplace=True)
df.drop(['written_during_early_access'], axis=1, inplace=True)
מוחקים את סטים-צ'יינה-לוקיישן מכיוון שאין נתונים בכלל ובכללי מוחקים את שניהם בגלל שאנחנו לא#
יודעים איך הגיעו לדאטה סט מכיוון שהגיעו דרך האייפיעי של סטים אבל לא רשומים בדוקומנטציה ההסבר
למושגים זה משחקים שקיימים רק בסין ומוסתרים בסין

df.columns

genres_df = df['genres'].str.get_dummies(sep=';')

df = pd.concat([df, genres_df], axis=1)

df.drop('genres', axis=1, inplace=True)

genres_df

df

# Corrleation Eda

# 1. Pie Chart for 'Recommended'
recommended_counts = df['Recommended'].value_counts()

plt.figure(figsize=(6, 6))
```

```
plt.pie(recommended_counts, labels=recommended_counts.index, autopct='%1.1f%%',
startangle=140)
plt.title('Distribution of Recommended Games')
plt.axis('equal')

plt.show()

# 2. Histogram for 'votes_up'
plt.figure(figsize=(8, 6))
sns.histplot(df['votes_up'], kde=True)
plt.title('Histogram of Votes Up')
plt.xlabel('Votes Up')
plt.show()

# 3. Bar Plot for 'language'
plt.figure(figsize=(10, 6))
sns.countplot(data=df, x='language')
plt.title('Distribution of Reviews by Language')
plt.xticks(rotation=90)
plt.show()

# 4. Box Plot for 'votes_up'
plt.figure(figsize=(8, 6))
sns.boxplot(y=df['votes_up'])
plt.title('Box Plot of Votes Up')
plt.show()

# 5. Time Series Plot for 'timestamp_created'
plt.figure(figsize=(12, 6))
sns.lineplot(data=df, x='timestamp_created', y=df.index)
plt.title('Number of Reviews Over Time')
plt.xticks(rotation=45)
plt.show()

# 6. Scatter Plot for 'votes_up' vs 'votes_funny'
plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x='votes_up', y='votes_funny')
plt.title('Scatter Plot of Votes Up vs Votes Funny')
plt.xlabel('Votes Up')
plt.ylabel('Votes Funny')
plt.show()

# 7. Bar Plot for release_date vs genre[]
selected_columns = ['release_date', 'Action', 'Adventure', 'Casual', 'Early Access', 'Indie',
'RPG', 'Simulation', 'Strategy','Massively Multiplayer','Nudity','Racing','Sexual
Content','Sports']
df_selected = df[selected_columns]
df_selected['release_date'] = pd.to_datetime(df_selected['release_date'])
```

```
df_selected['release_year'] = df_selected['release_date'].dt.year
genre_counts = df_selected.groupby('release_year').apply(lambda x: x[['Action', 'Adventure',
'Casual', 'Early Access', 'Indie', 'RPG', 'Simulation', 'Strategy','Massively
Multiplayer','Nudity','Racing','Sexual Content','Sports']].sum())
plt.figure(figsize=(12, 6))
genre_counts.plot(kind='bar', stacked=True, ax=plt.gca())
plt.xlabel('Release Year')
plt.ylabel('Number of Games')
plt.title('Release Year vs Genres')
plt.legend(title='Genres')
plt.show()



correlation_matrix = df.drop(columns=['title','developer','publisher','language','review']).corr()
correlation_matrix

plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Heatmap")
plt.show()

Q1 = df['votes_up'].quantile(0.25)
Q3 = df['votes_up'].quantile(0.75)
IQR = Q3 - Q1

df = df[~((df['votes_up'] < (Q1 - 1.5 * IQR)) | (df['votes_up'] > (Q3 + 1.5 * IQR)))]
df = df[(np.abs(stats.zscore(df['votes_up'])) < 3)]


upper_triangle = correlation_matrix.where(np.triu(np.ones(correlation_matrix.shape),
k=1).astype(bool))

columns_to_drop = [column for column in upper_triangle.columns if
any(upper_triangle[column] >= 0.9)]

df = df.drop(columns=columns_to_drop)

# 1. Pie Chart for 'Recommended'
recommended_counts = df['Recommended'].value_counts()

plt.figure(figsize=(6, 6))
plt.pie(recommended_counts, labels=recommended_counts.index, autopct='%1.1f%%',
startangle=140)
plt.title('Distribution of Recommended Games')
plt.axis('equal')
```

```
plt.show()

# 2. Histogram for 'votes_up'
plt.figure(figsize=(8, 6))
sns.histplot(df['votes_up'], kde=True)
plt.title('Histogram of Votes Up')
plt.xlabel('Votes Up')
plt.show()

# 3. Bar Plot for 'language'
plt.figure(figsize=(10, 6))
sns.countplot(data=df, x='language')
plt.title('Distribution of Reviews by Language')
plt.xticks(rotation=90)
plt.show()

# 4. Box Plot for 'votes_up'
plt.figure(figsize=(8, 6))
sns.boxplot(y=df['votes_up'])
plt.title('Box Plot of Votes Up')
plt.show()

# 6. Scatter Plot for 'votes_up' vs 'votes_funny'
plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x='votes_up', y='votes_funny')
plt.title('Scatter Plot of Votes Up vs Votes Funny')
plt.xlabel('Votes Up')
plt.ylabel('Votes Funny')
plt.show()

# 7. Bar Plot for release_date vs genre[]
selected_columns = ['release_date', 'Action', 'Adventure', 'Casual', 'Early Access', 'Indie',
'RPG', 'Simulation', 'Strategy','Massively Multiplayer','Nudity','Racing','Sports']
df_selected = df[selected_columns]
df_selected['release_date'] = pd.to_datetime(df_selected['release_date'])
df_selected['release_year'] = df_selected['release_date'].dt.year
genre_counts = df_selected.groupby('release_year').apply(lambda x: x[['Action', 'Adventure',
'Casual', 'Early Access', 'Indie', 'RPG', 'Simulation', 'Strategy','Massively
Multiplayer','Nudity','Racing','Sports']].sum())
plt.figure(figsize=(12, 6))
genre_counts.plot(kind='bar', stacked=True, ax=plt.gca())
plt.xlabel('Release Year')
plt.ylabel('Number of Games')
plt.title('Release Year vs Genres')
plt.legend(title='Genres')
plt.show()
```

```
df = df[(df['votes_up'] <= 7)]
df.reset_index(drop=True, inplace=True)


# 4. Box Plot for 'votes_up'
plt.figure(figsize=(8, 6))
sns.boxplot(y=df['votes_up'])
plt.title('Box Plot of Votes Up')
plt.show()
```

לעשות גרף של שנים על הז'אנר המועדף

```
correlation_matrix = df.drop(columns=['title','developer','publisher','language','review']).corr()
correlation_matrix

plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Heatmap")
plt.show()

df

# Processing

braille_pattern = re.compile(r'^[ ⠁-⠿  ]+$')
df = df[~df['review'].str.match(braille_pattern)]

df

def generate_ngrams(tokens, n):
    return list(ngrams(tokens, n))

def preprocess_text(text):
    text = text.lower()
    text = text.translate(str.maketrans('', '', string.punctuation))
    tokens = word_tokenize(text)
    tokens = [word for word in tokens if word not in stopwords.words('english')]
    return tokens

ngram_size = 2
df['processed_review'] = df['review'].apply(preprocess_text)
df['ngrams'] = df['processed_review'].apply(lambda tokens: generate_ngrams(tokens,
ngram_size))

ngram_freq_dict = Counter()
for ngram_list in df['ngrams']:
    for ngram in ngram_list:
        ngram_freq_dict[ngram] += 1
```

```
df['processed_review'] = df['review'].apply(preprocess_text)

word_freq_dict = Counter()
for tokens in df['processed_review']:
    word_freq_dict.update(tokens)


word_freq_dict

wordcloud = WordCloud(width=800, height=800,
background_color='white').generate_from_frequencies(word_freq_dict)

plt.figure(figsize=(8, 8), facecolor=None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad=0)
plt.show()


categories

ost;gameplay;graphics

df = df.drop(df[df['votes_funny'] > 5].index)
df.reset_index(drop=True, inplace=True)

df

def preprocess_sentences(text):
    text = text.lower()
    sentences = sent_tokenize(text)
    cleaned_sentences = []

    for sentence in sentences:
        sentence = sentence.translate(str.maketrans('', '', string.punctuation))
        tokens = sentence.split()
        tokens = [word for word in tokens if word not in stopwords.words('english')]
        cleaned_sentences.append(" ".join(tokens))

    return cleaned_sentences


df.drop('processed_review', axis=1, inplace=True)

df['processed_review'] = df['review'].apply(preprocess_sentences)
df
```

```
df = df.drop(df[df['review'] == ''].index)
df.reset_index(drop=True, inplace=True)

df

review_lengths = df['processed_review'].apply(lambda x: len(x))

plt.hist(review_lengths, bins=50, color='blue', alpha=0.7)
plt.title('Distribution of Review Lengths')
plt.xlabel('Number of Words in Review')
plt.ylabel('Frequency')
plt.show()

# S-Bert and RandomForest Models

df.drop('ngrams', axis=1, inplace=True)

df

average_scores = df.groupby('title')['sentiment_score'].mean().reset_index()

plt.figure(figsize=(24, 16))

plt.scatter(df['sentiment_score'], df['title'], alpha=0.5, label='Individual Scores')

plt.scatter(average_scores['sentiment_score'], average_scores['title'], color='red', marker='s',
s=50, label='Average Scores')

plt.title('Sentiment Score vs. Title')
plt.xlabel('Sentiment Score')
plt.ylabel('Title')
plt.legend()
plt.grid(True)
plt.show()

model = SentenceTransformer('paraphrase-distilroberta-base-v1')

df = df.groupby('app_id').filter(lambda x: len(x) >= 1)

keywords = {
    'gameplay': ['Immersive', 'Dynamic', 'Strategic', 'Responsive', 'Intuitive', 'Challenging',
'Seamless', 'Adaptive', 'Engaging', 'Tactical', 'Fluid', 'Puzzling', 'Navigable', 'Interactive',
'Strategic', 'Reactive', 'Versatile', 'Progressive', 'Innovative', 'Satisfying','story', 'missions',
'characters', 'shooting', 'controls', 'boss fights', 'gameplay', 'setting', 'exploration',
'scavenging', 'weapons', 'player interaction', 'attack', 'sandbox', 'exploring', 'survival',
'combat', 'abilities', 'evolving', 'weapons', 'resource production', 'mechanics', 'learning curve',
'decision-making', 'consequence', 'frustrating', 'difficulty levels', 'multiplayer', 'online',
```

```
'single-player', 'campaign', 'playthrough', 'terraforming', 'colonization', 'space', 'outposts',
'domes', 'rockets', 'balancing', 'resource distribution', 'disasters', 'replayability',
'achievements', 'challenges', 'strategy', 'simulations'],
   'soundtrack': ['Melodic', 'Atmospheric', 'Harmonious', 'Cinematic', 'Evocative', 'Enchanting',
'Uplifting', 'Lyrical', 'Emotional', 'Energetic', 'Ambient', 'Captivating', 'Haunting', 'Majestic',
'Rhythmic', 'Soulful', 'Orchestral', 'Nostalgic', 'Mesmerizing', 'music', 'soundtrack', 'themes',
'sound', 'relaxing', 'plays', 'various', 'psychopath', 'catchy', 'beat', 'sequence', 'type', 'ear',
'credits', 'catchy', 'cinematic', 'immersive', 'atmospheric', 'emotional', 'orchestral', 'repetitive'],
   'visuals': ['Photorealistic', 'Vibrant', 'Aesthetic', 'Cinematic', 'Crisp', 'Stunning', 'Immersive',
'Detailed', 'Sleek', 'Striking', 'Polished', 'Colorful', 'Realistic', 'Atmospheric', 'Dynamic', 'Vivid',
'Clean', 'Surreal', 'Sharp', 'Futuristic', 'discovery', 'visual', 'elements', 'textures', 'effects',
'environment', 'atmosphere', 'surface', 'atmospheric', 'representation', 'cracks', 'facade',
'creative', 'innovative', 'solutions', 'improving', 'enhancing', 'graphics', 'visuals',
'improvements']
}

sia = SentimentIntensityAnalyzer() #init vader


categories = ['visuals', 'soundtrack', 'gameplay']
categories_emb = [model.encode(c) for c in categories]
for category in keywords.keys():
   df[category] = None

final_cat_dic = {'visuals':[], 'soundtrack':[], 'gameplay':[]}

for i, review_list in df['processed_review'].items():
  #init per game
  cat_dic = {'visuals':[], 'soundtrack':[], 'gameplay':[]}
  min_dist = np.inf
  cat=None

  for r in review_list:
    r_emb = model.encode(r) #embeddings of review
    for i,c_emb in enumerate(categories_emb): #O(1)
    #look for category that is the closest to review, to understand what the review is talking
about
      closest_cat= cosine(c_emb,r_emb)
      if closest_cat<min_dist:
        min_dist = closest_cat
        cat = categories[i] #best fit category

    sen = sia.polarity_scores(r)['compound'] #a normalized compound score of sentimenr of
sentence

    #now we have per review the setiment and the cat

    if cat: #category for review is not None
```

```
      cat_dic[cat].append(sen) #add sentiment to the list of occurces for that cat

  #now we have per review a dic of categories and their sentiment
  for cal,cal_sent in cat_dic.items():
    if cal_sent: #not empty
      final_cat_dic[cal].append(np.mean(cal_sent))
    else:
      final_cat_dic[cal].append(0)

#now we gat a dic with 3 categories, for each the values are a list of the averaged sentiment
per game
#such that final_cat_dic['graphics'][i] would be the sentiment value of talking about graphics
across all revirews given to game i


for key in final_cat_dic:
  df[key] = final_cat_dic[key]

df

final_cat_dic

avg_scores = df.groupby('app_id')[['visuals', 'soundtrack', 'gameplay']].mean()

for category in ['visuals', 'soundtrack', 'gameplay']:
    df[category + '_avg'] = df['app_id'].map(avg_scores[category])
def compare_to_avg(row, category):
    if row[category + '_avg'] is None:
        return 0
    elif row[category] is None:
        return -1
    elif row[category] > row[category + '_avg']:
        return 1
    elif row[category] < row[category + '_avg']:
        return -1
    else:
        return 0
for category in ['visuals', 'soundtrack', 'gameplay']:
    df[category + '_compared_to_avg'] = df.apply(compare_to_avg, args=(category,), axis=1)

df

unique_rows_count = df[['visuals', 'soundtrack', 'gameplay']].nunique(axis=1).sum()

print(f"Number of unique rows with values in all three categories: {unique_rows_count}")


genres_columns = genres_df.columns
```

```
genres_columns


X = df[['visuals', 'soundtrack', 'gameplay']]
y = df['Recommended']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

range = list(range(1, 101))


param_grid = dict(n_estimators=range)

grid1 = GridSearchCV(RandomForestClassifier(), param_grid, cv=10, scoring='accuracy')


param_grid = dict(min_samples_leaf=range)

grid2 = GridSearchCV(RandomForestClassifier(), param_grid, cv=10, scoring='accuracy')


param_grid = dict(max_depth=range)

grid3 = GridSearchCV(RandomForestClassifier(), param_grid, cv=10, scoring='accuracy')


param_grid = dict(max_leaf_nodes=range)

grid4 = GridSearchCV(RandomForestClassifier(), param_grid, cv=10, scoring='accuracy')


param_grid = dict(min_samples_split=range)

grid5 = GridSearchCV(RandomForestClassifier(), param_grid, cv=10, scoring='accuracy')


param_grid = dict(max_features=range)

grid6 = GridSearchCV(RandomForestClassifier(), param_grid, cv=10, scoring='accuracy')


grid1.fit(X_train, y_train)
grid2.fit(X_train, y_train)
grid3.fit(X_train, y_train)
grid4.fit(X_train, y_train)
grid5.fit(X_train, y_train)
grid6.fit(X_train, y_train)
```

```
print(grid1.best_params_)
print('==================================')
print(grid2.best_params_)
print('==================================')
print(grid3.best_params_)
print('==================================')
print(grid4.best_params_)
print('==================================')
print(grid5.best_params_)
print('==================================')
print(grid6.best_params_)

RFCModel = RandomForestClassifier(n_estimators=5)

RFCModel.fit(X_train, y_train)

predictions = RFCModel.predict(X_test)

accuracy = accuracy_score(y_test, predictions)
precision = precision_score(y_test, predictions)
recall = recall_score(y_test, predictions)
f1 = f1_score(y_test, predictions)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)

plt.figure(figsize=(12, 6))
plt.plot(range, grid1.cv_results_['mean_test_score'], label='Number of Estimators')

plt.plot(range, grid3.cv_results_['mean_test_score'], label='Max Depth')

plt.plot(range, grid2.cv_results_['mean_test_score'], label='min_samples_leaf')

plt.plot(range, grid4.cv_results_['mean_test_score'], label='max_leaf_nodes_range')

plt.plot(range, grid5.cv_results_['mean_test_score'], label='min_samples_split')

plt.plot(range, grid6.cv_results_['mean_test_score'], label='max_features')

plt.xlabel('Parameter Value')
plt.ylabel('Accuracy')
plt.title('Random Forest Accuracy vs. Parameter Values')

plt.legend()
plt.show()
```

```
RFCModel = RandomForestClassifier(
    n_estimators=grid1.best_params_['n_estimators'],
    max_depth=grid3.best_params_['max_depth'],
    max_features=grid6.best_params_['max_features'],
    min_samples_leaf=grid2.best_params_['min_samples_leaf'],
    min_samples_split=grid5.best_params_['min_samples_split'],
    max_leaf_nodes=grid4.best_params_['max_leaf_nodes']

)

RFCModel.fit(X_train, y_train)


predictions = RFCModel.predict(X_test)

accuracy = accuracy_score(y_test, predictions)
precision = precision_score(y_test, predictions)
recall = recall_score(y_test, predictions)
f1 = f1_score(y_test, predictions)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)

import pickle

pickle.dump(RFCModel, open('/content/Model_Saved.pickle', 'wb'))

from google.colab import files

files.download('Model_Saved.pickle')

pickle.load(open('Model_Saved.pickle', 'rb'))

from sklearn.metrics import confusion_matrix

y_pred = RFCModel.predict(X_test)
cm = confusion_matrix(y_test, y_pred)

plt.imshow(cm, interpolation='nearest')
plt.title('Confusion Matrix')
plt.colorbar()
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()
```

```
from sklearn.metrics import roc_curve, auc

y_pred_proba = RFCModel.predict_proba(X_test)[:, 1]

fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
roc_auc = auc(fpr, tpr)

plt.plot(fpr, tpr, label='RFC (AUC = {:.3f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()

from sklearn.metrics import make_scorer

scoring = make_scorer(accuracy_score)

clf = RFCModel

test_error = cross_val_score(clf, X_train, y_train, cv=5, scoring=scoring, n_jobs=-1).mean()
learning_error = clf.score(X_test, y_test)

print(f"Test Error: {test_error:.4f}")
print(f"Learning Error: {learning_error:.4f}")

!pip install xgboost



import xgboost as xgb

# Create DMatrix for XGBoost
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)

# Define parameters for XGBoost
params = {
    'objective': 'binary:logistic',
    'eval_metric': 'logloss',
    # Add other hyperparameters as needed
}

# Train XGBoost model
xgb_model = xgb.train(params, dtrain)
```

```
# Make predictions
predictions = xgb_model.predict(dtest)

# Evaluate the model
accuracy = accuracy_score(y_test, predictions.round())
precision = precision_score(y_test, predictions.round())
recall = recall_score(y_test, predictions.round())
f1 = f1_score(y_test, predictions.round())

print("XGBoost Model:")
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)


!pip install tensorflow

!pip install lightgbm


import lightgbm as lgb

# Create Dataset for LightGBM
lgb_train = lgb.Dataset(X_train, label=y_train)
lgb_eval = lgb.Dataset(X_test, label=y_test, reference=lgb_train)

# Define parameters for LightGBM
params = {
    'objective': 'binary',
    'metric': 'binary_logloss',
}

# Train LightGBM model
lgb_model = lgb.train(params, lgb_train, num_boost_round=100, valid_sets=[lgb_eval])
# Make predictions
predictions = lgb_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, predictions.round())
precision = precision_score(y_test, predictions.round())
recall = recall_score(y_test, predictions.round())
f1 = f1_score(y_test, predictions.round())

print("LightGBM Model:")
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
```

```
print("F1-score:", f1)


columns_df1 = set(df.columns)
columns_df2 = set(genres_df.columns)

# Find the overlapping columns
overlapping_columns = columns_df1.intersection(columns_df2)

# Convert the set to a list if needed
overlapping_columns_list = list(overlapping_columns)

print("Overlapping columns:", overlapping_columns_list)


import matplotlib.pyplot as plt

# Create a single figure and subplots
fig, axes = plt.subplots(nrows=len(overlapping_columns_list), ncols=1, figsize=(8,
6*len(overlapping_columns_list)))

for i, column in enumerate(overlapping_columns_list):
    # Plotting on the i-th subplot
    axes[i].scatter(df[column], df['visuals_avg'], label='Visuals')
    axes[i].scatter(df[column], df['soundtrack_avg'], label='Soundtrack')
    axes[i].scatter(df[column], df['gameplay_avg'], label='Gameplay')

    # Adding labels and title to the subplot
    axes[i].set_xlabel(column)
    axes[i].set_ylabel('Comparison to Average')
    axes[i].set_title(f'Comparison of {column} with Visuals, Soundtrack, and Gameplay')

    # Adding legend to the subplot
    axes[i].legend()

# Adjust layout to prevent overlap
plt.tight_layout()

# Showing plot
plt.show()


# Visualize the Random Forest Classifier model
from sklearn.inspection import permutation_importance

# Calculate permutation importance
result = permutation_importance(RFCModel, X_test, y_test, n_repeats=10,
random_state=42)
```

```
# Plot the permutation importance
fig, ax = plt.subplots(figsize=(10, 6))
ax.bar(X.columns, result.importances_mean)
ax.set_title('Permutation Importance of Random Forest Classifier')
ax.set_ylabel('Importance Score')
plt.xticks(rotation=90)
plt.show()

# Visualize feature importances
importance_dict = dict(zip(X.columns, RFCModel.feature_importances_))
sorted_importance_dict = dict(sorted(importance_dict.items(), key=lambda item: item[1],
reverse=True))

# Plot feature importances
fig, ax = plt.subplots(figsize=(10, 6))
ax.bar(list(sorted_importance_dict.keys()), list(sorted_importance_dict.values()))
ax.set_title('Feature Importances of Random Forest Classifier')
ax.set_ylabel('Importance Score')
plt.xticks(rotation=90)
plt.show()

from sklearn.tree import plot_tree
tree = RFCModel.estimators_[4]

plt.figure(figsize=(30, 24))
plot_tree(tree, filled=True, rounded=True, feature_names=X.columns, class_names=['Not
Recommended', 'Recommended'])

plt.rcParams.update({'font.size': 20})

plt.title("1 tree out of the whole forest taken as an example")
plt.show()

from sklearn.tree import plot_tree

n_estimators = len(RFCModel.estimators_)

n_rows = int(np.ceil(np.sqrt(n_estimators)))
n_cols = int(np.ceil(n_estimators / n_rows))

fig, axs = plt.subplots(nrows=n_rows, ncols=n_cols, figsize=(4*n_cols, 3*n_rows))

for i, (tree, ax) in enumerate(zip(RFCModel.estimators_, axs.flatten())):
    plot_tree(tree, ax=ax, filled=True, rounded=True, feature_names=X.columns,
class_names=['Not Recommended', 'Recommended'])
    ax.set_title(f"Tree {i+1}")
```

```
plt.rcParams.update({'font.size': 20})

fig.suptitle("Random Forest Trees")

plt.show()
```

כל ההערות נמצאות כאן

```
# Imports
# Installing necessary libraries
# Importing required packages
# Setting up the environment

# Data Loading and Preprocessing
# Reading data from a CSV file
# Reordering columns in the DataFrame

# Exploratory Data Analysis (EDA)
# Visualizing unique values per feature
# Visualizing NaN counts in each column
# Displaying dataset information
# Checking and displaying missing values
# Converting specific columns to datetime format
# Dropping unnecessary columns from the DataFrame

# Genre Data Processing
# Extracting genres information into a separate DataFrame
# Concatenating genres information with the main DataFrame
# Dropping the 'genres' column from the main DataFrame

# Correlation Analysis
# Visualizing the correlation matrix
# Removing outliers based on the 'votes_up' column
# Dropping highly correlated columns

# Sentiment Analysis
# Preprocessing text and generating n-grams
# Calculating word frequencies and displaying a word cloud
# Preprocessing sentences and cleaning review text

# Sentiment Analysis - Part 2
# Calculating sentiment scores for different categories

# Model Building
# Splitting data into training and testing sets
# Grid search for Random Forest Classifier hyperparameters
# Training and evaluating the Random Forest Classifier model
# Saving the trained model
```

Model Evaluation
Visualizing confusion matrix and ROC curve
Cross-validation and error calculation

Additional Models
Installing and using XGBoost and LightGBM for comparison

Data Visualization
Comparing columns between DataFrames for overlapping columns
Visualizing the relationship between columns and sentiment categories

Model Interpretability
Visualizing feature importances and permutation importance
Plotting individual trees from the Random Forest

משתנים:

df: דאטה פריים של פאנדאז המחזיק בנתונים
recommend_column: סדרת פנדאז המכילה את העמודה של פיצ'ר ההמלצות מה-DataFrame המקורי.
unique_values: סדרת פנדאז המכילה את הערכים הייחודיים עבור כל עמודה מספרית ב-DataFrame.
nan_counts: סדרת פנדאז המכילה את מספר ערכי NaN עבור כל עמודה ב-DataFrame.
Q1: הרבעון הראשון של העמודה 'הצבעות למעלה'.
Q3: הרבעון השלישי של העמודה 'הצבעות למעלה'.
IQR: הטווח הבין-רבעוני של העמודה votes_up.
upper_triangle: מסכה בוליאנית של המשולש העליון של מטריצת המתאם.
columns_to_drop: רשימה של שמות עמודות להורדה בהתבסס על מטריצת המתאם.
genres_df: דאטה פריים של פנדאז המכילה את נתוני הז'אנרים.
genres_columns: רשימה של שמות עמודות בז'אנרים DataFrame.
overlapping_columns: קבוצה של שמות עמודות החופפות בין genre idf ו df.
overlapping_columns_list: רשימה של שמות עמודות החופפות בין genre idf ו df.

פעולות:
gener_ngrams(tokens, n):
פרמטרים:
אסימונים: רשימה של אסימונים (מילים).
n: מספר שלם המציין את הגודל של ngrams ליצירת.
החזרות:
רשימה של ngrams בגודל n שנוצרו מאסימוני הקלט.

preprocess_text(text):
פָּרָמֶטֶר:
טקסט: טקסט הקלט לעיבוד מקדים.
החזרות:
רשימה של אסימונים מעובדים מראש לאחר המרת הטקסט לאותיות קטנות, הסרת סימני פיסוק והסרת מילות עצירה.

preprocess_sences(text):
פָּרָמֶטֶר:
טקסט: טקסט הקלט לעיבוד מקדים.
החזרות:
רשימה של משפטים מעובדים מראש לאחר המרת הטקסט לאותיות קטנות, הסרת סימני פיסוק והסרת מילות עצור.

compare_to_avg(row, category):
פרמטרים:
שורה: שורה מ-DataFrame המכילה ציוני סנטימנט.
קטגוריה: מחרוזת המציינת את הקטגוריה שעבורה משווים ציוני סנטימנט.
החזרות:
מספר שלם המציין אם ציון הסנטימנט עבור הקטגוריה הנתונה גדול, קטן או שווה לציון הסנטימנט הממוצע.

שם קובץ: Predict

מיקום: C:\Users\REGEV\PycharmProjects\ProjectGUI\Predict.py

```
import pickle
import pandas as pd
import numpy as np
from nltk import sent_tokenize
from nltk.corpus import stopwords
from nltk.sentiment import SentimentIntensityAnalyzer
from scipy.spatial.distance import cosine
import string
from sentence_transformers import SentenceTransformer

# Load the sbert paraphrase-distilroberta-base-v1 model
sbert_model = SentenceTransformer('paraphrase-distilroberta-base-v1')
sia = SentimentIntensityAnalyzer() #init vader
```

```
def preprocess_sentences(text: str) -> list:
    """
    Preprocess sentences by:
    1. Converting to lowercase
    2. Removing punctuation
    3. Tokenizing
    Returns a list of cleaned sentences
    """
    text = text.lower()
    sentences = sent_tokenize(text)
    cleaned_sentences = []

    for sentence in sentences:
        # Remove punctuation from the sentence
        sentence = sentence.translate(str.maketrans('', '',
string.punctuation))
        tokens = sentence.split()
        # Remove stopwords from the tokens
        tokens = [word for word in tokens if word not in
stopwords.words('english')]
        cleaned_sentences.append(" ".join(tokens))

    return cleaned_sentences


def predict_sentiment(review: str, genre: str) -> dict:
    """
    Predict sentiment for a review and genre
    Returns a dictionary with sentiment scores for each category
    """
    cat_dic = {'visuals': [], 'soundtrack': [], 'gameplay': []}

    for r in preprocess_sentences(review):
        sen = sia.polarity_scores(r)['compound']
        for category in keywords.keys():
            # Check if any keyword from the category is present in the
sentence
            if any(word in r.lower().split() for word in keywords[category]):
                cat_dic[category].append(sen)

    for cal, cal_sent in cat_dic.items():
        if cal_sent:
            # Calculate the mean sentiment score for the category
            cat_dic[cal] = np.mean(cal_sent)
        else:
            cat_dic[cal] = 0

return cat_dic
```

```
def predict_game_features(review: str, genre: str) -> pd.DataFrame:
    """
    Predict game features from review and genre
    Returns a pandas DataFrame with predicted features
    """
    new_game_features = pd.DataFrame(columns=['review', 'Action', 'Adventure',
'Casual', 'Early Access', 'Free to Play', 'Indie',
                                              'Massively Multiplayer',
'Nudity', 'RPG', 'Racing', 'Simulation', 'Sports', 'Strategy'])
    new_game_features.loc[0, 'review'] = review
    str_list = genre.split(',')

    for genre in str_list:
        genre = genre.strip()
        if genre in new_game_features.columns:
            new_game_features.loc[0, genre] = 1
    new_game_features = new_game_features.fillna(0)

    new_game_features['processed_review'] =
new_game_features['review'].apply(preprocess_sentences)

    sentiment_features = predict_sentiment(review, genre)
    new_game_features['visuals'] = sentiment_features['visuals']
    new_game_features['soundtrack'] = sentiment_features['soundtrack']
    new_game_features['gameplay'] = sentiment_features['gameplay']

    return new_game_features


def load_rfc_model() -> object:
    """
    Load RFC model from file
    Returns the loaded model or None if loading fails
    """
    try:
        return pickle.load(open('Model_Saved (3).pickle', 'rb'))
    except Exception as e:
        print(f"Error loading RFC model: {e}")
        return None
```

```
def PREDICT(review: str, genre: str) -> bool:
    """
    Predict game recommendation based on review and genre
    Returns True if the game is recommended, False otherwise
    """
    rfc_model = load_rfc_model()
    if rfc_model is None:
        return False

    new_game_features = predict_game_features(review, genre)
    x = new_game_features[['visuals', 'soundtrack', 'gameplay']]
    prediction = rfc_model.predict(x)
    print(x)
    return prediction == 1

keywords = {
    'gameplay': ['Immersive', 'Dynamic', 'Strategic', 'Responsive',
'Intuitive', 'Challenging', 'Seamless', 'Adaptive', 'Engaging', 'Tactical',
'Fluid', 'Puzzling', 'Navigable', 'Interactive', 'Strategic', 'Reactive',
'Versatile', 'Progressive', 'Innovative', 'Satisfying','story', 'missions',
'characters', 'shooting', 'controls', 'boss fights', 'gameplay', 'setting',
'exploration', 'scavenging', 'weapons', 'player interaction', 'attack',
'sandbox', 'exploring', 'survival', 'combat', 'abilities', 'evolving',
'weapons', 'resource production', 'mechanics', 'learning curve',
'decision-making', 'consequence', 'frustrating', 'difficulty levels',
'multiplayer', 'online', 'single-player', 'campaign', 'playthrough',
'terraforming', 'colonization', 'space', 'outposts', 'domes', 'rockets',
'balancing', 'resource distribution', 'disasters', 'replayability',
'achievements', 'challenges', 'strategy', 'simulations'],
    'soundtrack': ['Melodic', 'Atmospheric', 'Harmonious', 'Cinematic',
'Evocative', 'Enchanting', 'Uplifting', 'Lyrical', 'Emotional', 'Energetic',
'Ambient', 'Captivating', 'Haunting', 'Majestic', 'Rhythmic', 'Soulful',
'Orchestral', 'Nostalgic', 'Mesmerizing', 'music', 'soundtrack', 'themes',
'sound', 'relaxing', 'plays', 'various', 'psychopath', 'catchy', 'beat',
'sequence', 'type', 'ear', 'credits', 'catchy', 'cinematic', 'immersive',
'atmospheric', 'emotional', 'orchestral', 'repetitive'],
    'visuals': ['Photorealistic', 'Vibrant', 'Aesthetic', 'Cinematic',
'Crisp', 'Stunning', 'Immersive', 'Detailed', 'Sleek', 'Striking',
'Polished', 'Colorful', 'Realistic', 'Atmospheric', 'Dynamic', 'Vivid',
'Clean', 'Surreal', 'Sharp', 'Futuristic', 'discovery', 'visual', 'elements',
'textures', 'effects', 'environment', 'atmosphere', 'surface', 'atmospheric',
'representation', 'cracks', 'facade', 'creative', 'innovative', 'solutions',
'improving', 'enhancing', 'graphics', 'visuals', 'improvements']
}

RFCModel = load_rfc_model()
```

- **משתנים והשימושים בהם:**

keywords: מילון המכיל מילות מפתח עבור קטגוריות שונות (משחק, פסקול, ויזואליה) המשמש לניתוח סנטימנטים.

RFCModel: מופע של מודל Random Forest Classifier, שנטען מקובץ באמצעות joblib.

- **פונקציות ומטרותיהן:**

preprocess_sentences(text: str) -> list:
מטרה: עיבוד מקדים של טקסט נתון על-ידי המרתו לאותיות קטנות, הסרת סימני פיסוק והפיכתו למשפטים.
מחזיר: רשימה של משפטים מנוקים.

predict_sentiment(review: str, genre: str) -> dict:
מטרה: לחזות את הסנטימנט של סקירה עבור קטגוריות שונות (משחק, פסקול, ויזואליה) בהתבסס על נוכחות של מילות מפתח.
מחזיר: מילון עם ציוני סנטימנט עבור כל קטגוריה.

predict_game_features(review: str, genre: str) -> pd.DataFrame:
מטרה: לחזות תכונות משחק מסקירה וז'אנר, כולל ציוני סנטימנט עבור קטגוריות שונות.
מחזיר: DataFrame של פנדה עם תכונות חזויות.

load_rfc_model() -> object:
מטרה: טען מודל מחלק יער אקראי מקובץ באמצעות joblib.
מחזיר: הדגם הנטען או ללא אם הטעינה נכשלת.

PREDICT(review: str, genre: str) -> bool:
מטרה: לחזות אם משחק מומלץ על סמך סקירה וז'אנר באמצעות מודל ה-RFC הנטען.
מחזיר: נכון אם המשחק מומלץ, לא נכון אחרת.

שם קובץ: GUI

מיקום: C:\Users\REGEV\PycharmProjects\ProjectGUI\GUI.py

```
# Import customtkinter library for GUI components
import customtkinter as ctk

# Import CTkMessagebox for displaying message boxes
from CTkMessagebox import CTkMessagebox

# Import PREDICT function from Predict module
from Predict import PREDICT


def project():
    """
    Predict the success or failure of a game based on user input
    """
    # Get the review text from the first entry field
    review_text = review_entry1.get()

    # Get the genres text from the second entry field
    genres_text = review_entry2.get()

    # Call the PREDICT function with the review and genres text
    # If the prediction is True, display a success message
    if PREDICT(review_text, genres_text):
        CTkMessagebox(title="Info", message='Predicted Game Will Succeed')
    # Otherwise, display a failure message
    else:
        CTkMessagebox(title="Info", message='Predicted Game Will Not Succeed')
```

```
# Set the appearance mode to dark
ctk.set_appearance_mode("dark")

# Set the default color theme to green
ctk.set_default_color_theme("green")

# Create the main application window
root = ctk.CTk()
root.geometry("500x350")

# Create a frame to hold the GUI components
frame = ctk.CTkFrame(master=root)
frame.pack(pady=20, padx=60, fill="both", expand=True)

# Create a label for the review entry field
review_label1 = ctk.CTkLabel(master=frame,
                             text="Enter the review of the game you want to
predict it's success or failure:")
review_label1.pack(pady=12, padx=10)

# Create an entry field for the review text
review_entry1 = ctk.CTkEntry(master=frame, width=500, placeholder_text="•
Enter your review here.")
review_entry1.pack(pady=12, padx=10)

# Create a label for the genres entry field
review_label2 = ctk.CTkLabel(master=frame,
                             text="Enter the genres of the game you want to
predict\n the genres that can be used in this model are Action, Adventure,
Casual, Early Access, Free to Play, Indie, Massively Multiplayer, Nudity,
RPG, Racing, Simulation, Sports, and Strategy.")
review_label2.pack(pady=12, padx=10)

# Create an entry field for the genres text
review_entry2 = ctk.CTkEntry(master=frame, width=500, placeholder_text="•
Enter your genres here.")
review_entry2.pack(pady=12, padx=10)

# Create a button to trigger the prediction
predict_button = ctk.CTkButton(master=frame, text="Predict", command=project)
predict_button.pack(pady=12, padx=10)

# Start the main event loop
root.mainloop()
```

משתנים:

review_text: משתנה מחרוזת המחזיק את טקסט סקירת הקלט של המשתמש משדה הערך הראשון.

genres_text: משתנה מחרוזת שמחזיק את טקסט ז'אנרי הקלט של המשתמש משדה הערך השני.
root: חלון היישום הראשי, מופע של ctk.CTk().

frame: מסגרת שמחזיקה את רכיבי ה-GUI, מופע של ctk.CTkFrame.

review_label1 ו- review_label2: תוויות שמציגות טקסט למשתמש, מופעים של ctk.CTkLabel.

review_entry1 ו- review_entry2: שדות כניסה שבהם המשתמש יכול להזין טקסט, מופעים של ctk.CTkEntry.

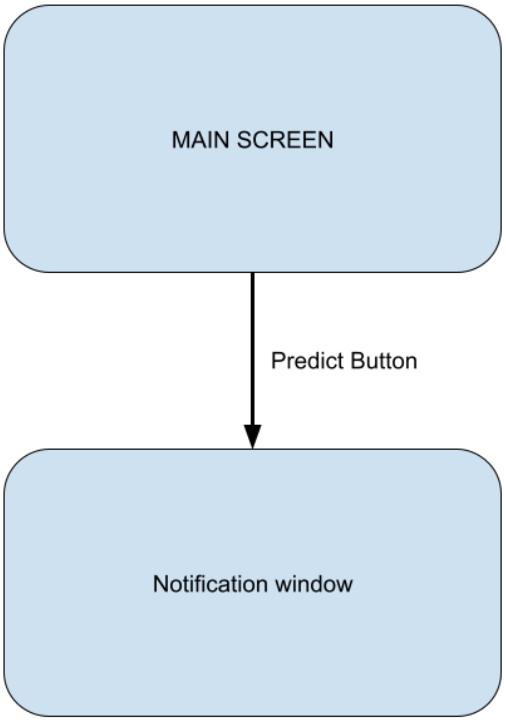
predict_button: כפתור שמפעיל את החיזוי, מופע של ctk.CTkButton.

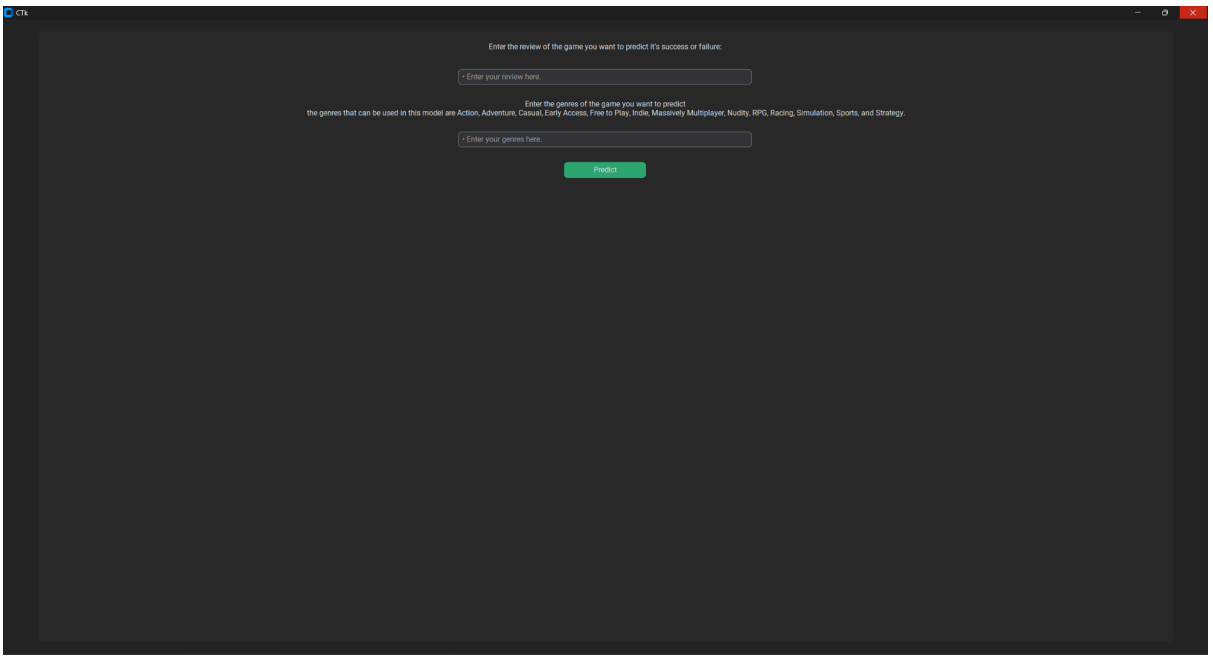
פונקציות:

project(): פונקציה המסווגת הצלחה או כישלון של משחק על סמך קלט המשתמש. הוא מקבל את טקסט הסקירה והז'אנרים משדות הכניסה, קורא לפונקציה PREDICT ומציג תיבת הודעה עם התוצאה.

PREDICT(review_text, genres_text): פונקציה המיובאת מהקובץ Predict.py שלוקחת את טקסט הסקירה והז'אנרים כקלט ומחזירה ערך בוליאני המציין אם המשחק צפוי להצליח או לא.

מדריך למשתמש

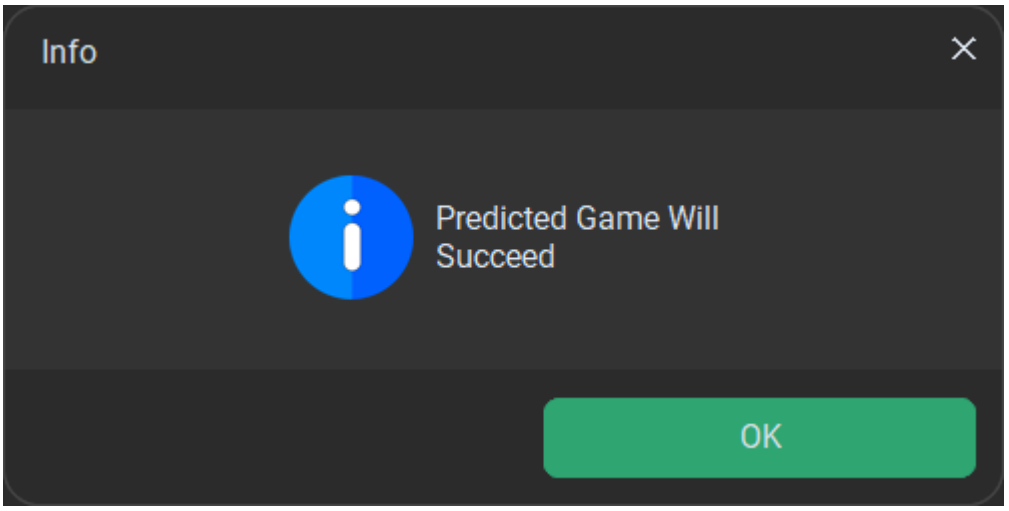




מסך ראשי בו רושמים את תיאור המשחק והז'אנרים ובו לוחצים על הכפתור כדי לתת למודל לסווג את הצלחת המשחק בעזרת הנתונים שהמשתמש רשם

תיבת טקסט 1:מקום לרשימת תיאור המשחק עם הסבר מעליה על מה צריך לרשום ואיך להתייחס לדברים ספציפיים בתיאור המשחק

תיבת טקסט 2: מקום לרשימת הז'אנרים של המשחק עם הסבר מעל על איך צריך לרשום את הז'אנרים
כפתור PREDICT: מפעיל את פונקציית PREDICT מקובץ PREDICT.PY על הנתונים שהמשתמש נתן ב2 תיבות הטקסט



מסך התראה מופיע לאחר לחיצה על הכפתור PREDICT במסך הראשי ונותן מידע האם המשחק סווג שיצליח או לא

רפלקציה

היבט הקידוד של הפרויקט היה כיפי ומעניין במידה רבה, ודרש יישום של מודלים וטכניקות של למידת מכונה שלא השתמשתי בהם עד היום, כדי לטפל במערכי נתונים הנרחבים שהשגתי דרך KAGGLE ומאגרי המידע של STEAM. הניסיון המעשי הזה לא רק חידד את כישורי התכנות שלי, אלא גם הציג את היישומים האמיתיים של דאטה סיינס והנדסת נתונים. פרוייקט זה הביא לי כלים רבים כמו שימוש בAPI , יצירת EDA נכון וטוב, הסנפת נתונים קשים להשגה מהאינטרנט וניהול זמן נכון. קושי גדול שהיה לי היה חישוב הסנטימנטים שלקח לי ימים רבים להגיע לקוד המתפקד נכון ומדרג כל ביקורת לפי הסנטימנט שלה על פי כל קטגוריה. המסקנות שלי מהפרוייקט הן שזה יותר מאפשרי להגיע למצב של סיווג הצלחה של משחק לפני שהמשחק יוצא ושחברות גדולות עם יותר משאבים יוכלו לפתח מודל הרבה יותר טוב משלי אשר יוכל לסווג 90+% מהזמן בצורה נכונה אם משחק יצליח או יכשל. אם הייתי מתחיל היום את הפרוייקט, כנראה שהייתי משקיע יותר בGUI מאשר שהשקעתי בו עכשיו, השימוש בGUI בפרוייקט היה הפעם הראשונה שלי ביצירת ממשק משתמש גרפי. אם הייתה לי את האפשרות להגיש את הפרוייקט ללא ממשק משתמש גרפי הייתה לי הרגשה יותר טובה עם הפרויקט, מהסיבה שזאת פעם ראשונה שאני עושה דבר כזה. בנוסף לכך אם היו נתונים יותר קלים להשגה מאשר הורדה של כמות נתונים גדולה מAPI של STEAM אשר לקחה זמן רב להורדה וזמן רב לישום בגלל המעבר מקובץ JSON של עשרות אלפי ביקורות לכל משחק אל דאטה פריים אשר אני יוכל לעשות בו שימוש. לסיכום, פרויקט זה לא רק העמיק את התשוקה שלי לתכנות ובינה מלאכותית, אלא גם צייד אותי במיומנויות שיעזרו לי בהמשך החיים גם מחוץ לתוכנית.

ביבליוגרפיה

Works Cited

SentenceTransformers Documentation — Sentence-Transformers documentation, https://sbert.net.

Rendall, Chase, et al. “Data Analysis of Video Game Sales from 1980-2016 |.” *NYC Data Science Academy*, 3 May 2020, https://nycdatascience.com/blog/student-works/data-analysis-of-video-game-sales-from-1980-2016/.

sean. “video games – Savvy Statistics.” *Savvy Statistics*, 6 June 2019, https://savvystatistics.com/tag/video-games/.

Steam. “Steam API Documentation.” *Steamworks*, https://partner.steamgames.com/doc/store/getreviews. Accessed May 202.

Wallach, Omri. “The history of the gaming industry in one chart.” *The World Economic Forum*, 27 November 2020, https://www.weforum.org/agenda/2020/11/gaming-games-consels-xbox-play-station-fun/.