קורס בריית מידע - 20595

ממ"ן 21

תאריך הגשה: 5.5.2023

קבוצה: 82

למרצה: רועי רחמני

מגיש: עידן קוגן

ת.ז: 316375591

שאלה 1

א. מטרת בריית המידע

לחזות מחלה כלייתית כרונית (אי-ספיקת כליות כרונית) אצל נבדק נתון ע"פ נתונים שונים כמו גיל ובדיקת דם.

ב. הגדרת הבעיה והכנת הנתונים

אציין שעבדתי בסמוך למאמר והתבססתי על הרעיונות משם. ממליץ לפתוח את מחברת הפייתון בIDE.

שם התכונה	תיאור התכונה	סוג	יחידות מידה	תחום ערכים	ממוצע	סטיית תקן	ערכים לא חוקיים
age	גיל הנבדק	נומרי - רציף	years	0-120	51.48338	17.16971	
bp	לחץ דם	נומרי - רציף	mmHg (millimeters of mercury)	50-140	76.20155	12.64463	יש תצפית אחת עם bp של 180. כל השאר מתחת או שווים ל140.
sg		נומרי - רציף	unitless	1.005 - 1.025	1.0174	0.00572	
al	אלבומין (חלבון)	נומרי - רציף	g/dL (grams per deciliter)	0 - 5	1.01695	1.35268	
su	סוכר (כולל צום)	נומרי - רציף	mg/dL (milligrams per deciliter)	0 - 5	0.45014	1.09919	
bgr	סוכר (ללא צום)	נומרי - רציף	mg/dL (milligrams per deciliter)	70 - 490	148.39155	79.10971	יש תצפית אחת עם ערך 22. לא הגיוני.
bu	ריכוז החנקן בדם בצורת שתנן	נומרי - רציף	mg/dL (milligrams per deciliter)	1.5 - 241.0	55.17778	43.78448	יש 3 תצפיות עם ערכים גבוהים מ241.
sc	קריאטינין	נומרי - רציף	mg/dL (milligrams per deciliter)	0.4 - 18.1	2.62968	3.21041	יש 4 תצפיות עם ערכים גבוהים או שווים ל20.
sod	נתרן	נומרי - רציף	mEq/L (milliequivalents per liter)	104- 150	137.87460	7.05282	יש תצפית אחת עם ערך גבוה מ150. ויש תצפית אחת עם ערך נמוך מ-104
pot	אשלגן	נומרי - רציף	mEq/L (milliequivalents per liter)	2.5 - 47.0	4.62724	3.19390	2 ערכים גבוהים מאוד וכל השאר נמוכים.
hemo	המוגלובין	נומרי - רציף	g/dL (grams per deciliter)	3.1 - 17.8	12.52644	2.91259	

pcv	פרופורציית תאי דם אדומים מתוך הדם.	נומרי - רציף	% (percentage)	9 - 54	38.88450	8.99010	
wc	תאי דם לבנים	נומרי - רציף	cells/cumm (cells per cubic millimeter)	2200 - 16700	8225.51724	2501.55050	יש 4 תצפיות עם ערכים גבוהים מ16,700.
rc	ספירת תאי דם אדומים	נומרי - רציף	millions/cumm (millions per cubic millimeter)	2.1 - 8.0	4.70743	1.02532	
rbc	תקינות כמות תאי דם אדומים	קטגורי		normal abnormal			
pc	תאי מוגלה	קטגורי		normal abnormal			
pcc	גושי תאי מוגלה	קטגורי		present notpresent			
ba	בקטריה	קטגורי		present notpresent			
htn	לחץ דם גבוה	קטגורי		yes no			
dm	סוכרת	קטגורי		yes no			
cad	מחלת לב איסכמית	קטגורי		yes no			
appet	רעב	קטגורי		good poor			
pe	הצטברות נוזלים ברגליים	קטגורי		yes no			
ane	אנמיה	קטגורי		yes no			
class	אי-ספיקת כליות כרונית	קטגורי		ckd nockd			

* בנוסף מחקתי תצפיות שיש להן יותר מ9 ערכים חסרים. זה יוצא 11 שורות.

* הגרפים המראים וויזואלית את הערכים חריגים מופיעים בסעיף ה'. לא מחקתי את התצפיות הללו אלא רק הפכתי את ערכי החריגים לnan.

סה"כ רשומות לפני הניקוי: 400

סה"כ רשומות לאחר ניקוי: 389

ג. <u>שלבי הKDD:</u>

1. איסוף הנתונים.
2. ניקוי הנתונים:
    a. מחיקת עמודות לא רלוונטיות
    b. מחיקת תצפיות עם יותר מ9 נתונים חסרים – סה"כ 11 שורות.
    c. מחיקת נתונים חריגים.
    d. החלפת ערכים חסרים מ-"?" ל-nan.
    e. החלפת ערכי עמודות בוליאניות מno/yes ל0/1.
    f. השלמת נתונים חסרים
3. המרת נתונים וסטנדרטיזציה.
4. בחירת עמודות רלוונטיות לכריית במידע.
5. ביצוע דיסקרטיזציה לכל התכונות המספריות.
6. שימוש בטכניקות כריית מידע שנלמדו.
7. ניתוח התוצאות
8. הסקת מסקנות

ד. <u>סקירה השוואתית ל-4 חלופות אפשריות לביצוע כריית מידע (יתרונות / חסרונות)</u>

1. <u>K השכנים הקרובים ביותר</u> - שיטה זו משמשת לסיווג ורגרסיה. השיטה מוצאת את k השכנים הקרוסים ביותר לתצפית חדשה ושימוש במחלקת הרוב או בערך הממוצע של אותם k-שכנים כדי לבצע חיזוי.
<u>היתרונות</u> - האימון מהיר מאוד, רק שומרים את תצפיות האימון, ללא עיבוד שלהם. בנוסף ניתן ללמוד פונקציות מורכבות מאוד - כי אין לנו מודל שאנחנו צריכים לבנות אותו מתוך אילוצים קבועים מראש כמו עץ החלטה. מלבד זאת, אין מידע שהולך לעיבוד מכיוון ששומרים את תצפיות האימון עצמן ולא מודל שמייצג אותן. אין פעולה של דחיסת מידע - פשוט שומרים את המידע הגולמי - תצפיות האימון - עד שיש צורך לסווג תצפיות חדשות.

<u>חסרונות</u> - איטיות בזמן הסיווג - בכל פעם שמגיעה תצפית חדשה - עלינו קודם כל פעם למצוא את K השכנים הקרובים ביותר של אותה תצפית ורק אז לסווג אותה. ככל שיש יותר ממדים ויותר תצפיות בקבוצת האימון, פעולת הסיווג תיקח יותר זמן. חוסר מודל - מכיוון שתצפיות האימון מהוות את המודל אז קשה לנו להסביר מדוע האלגוריתם נותן סיווג כזה או אחר לתצפית מסויימת. זאת בניגוד לאלגוריתם של עץ החלטה שכל סיווג מוסבר על ידי איזשהו סט של חוקים שאנחנו יכולים להבין אותם ולבקר אותם ואולי אפילו לתקן אותם. בנוסף קיימת "קללת הממדיות" - ככל שיש יותר ממדים \ תכונות אז יש יותר סיכוי שתהיה השפעת יתר על המרחק בין השכנים של משתנים שהם לא רלוונטיים. יש מעט מאוד משתנים שמשפיעים על סיווג של תצפית מסויימת, אך מכיוון שלא יודעים מראש מי הם אז ניקח בחשבון את כל המשתנים, ואז חישוב המרחק יושפע יותר מדי ממשתנים שהם לא רלוונטיים.

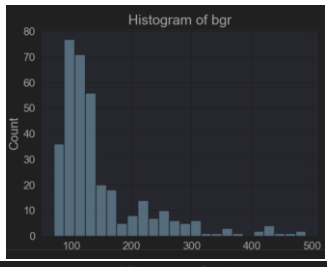
2. <u>רגרסיה לוגיסטית</u> - מודל סטטיסטי המתאר קשר בין משתנה מוסבר קטגורי לבין משתנים מסבירים כמותיים. בעזרת המודל ניתן לאמוד את <u>מידת ההשפעה</u> של שינוי בערכו של המשתנה המסביר על ערכו של המשתנה המוסבר.
משתמשים במודל רגרסיה לוגיסטית בבעיות מסוג סיווג וחיזוי וכן לזיהוי קשר בין משתנים. היתרונות של רגרסיה לוגיסטית היא שהיא קלה למימוש ויעילה בשלב האימון. ברוב בסיסי הנתונים רגרסיה לוגיסטית מביאה תוצאות טובות כאשר התכונות בלתי תלויות.

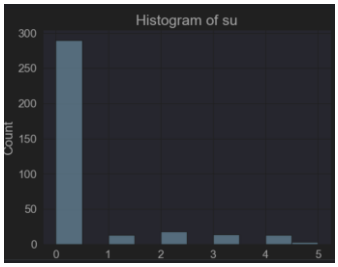
3. <u>נאייב בייס</u> - שיטת סיווג הסתברותית, המבוססת על משפט בייס.
<u>יתרונות</u> - מימוש קל מאוד. חישוב כמות מצומצמת יחסית של הסתברויות בגלל הנחת אי התלות. והרבה בסיסי נתונים מביא לתוצאות טובות.
<u>חסרונות</u> - במידה וכן מתקיימת תלות מסוימת בין המשתנים אז הסיווג יהיה פחות מדויק ובאופן מעשי אפשר למצוא הרבה תלויות בין משתנים, ובמצב זה השיטה לא תעבוד טוב.

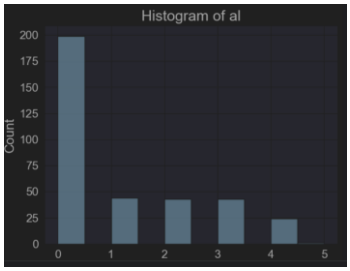
4. <u>רגרסיה לינארית מרובה</u> - טכניקה סטטיסטית המשמשת כדי למדל את הקשר בין משתנה תלוי לשניים או יותר משתנים בלתי תלויים. במילים פשוטות, זה עוזר לחזות את הערך של משתנה תלוי בהתבסס על הערכים של שניים או יותר משתנים בלתי תלויים. החסרונות שלה זה שהיא מניחה קשר לינארי בין המשתנים מה שעלול להוביל לדיוק נמוך כאשר אין קשר לינארי בין התכונות. בנוסף עלולה להכנס למצב של התאמת יתר overfitting. רגרסיה לינארית קלה למימוש אך לרוב בסיסי הנתונים פשוטה מדי כדי למדל את היחסים בין המשתנים.

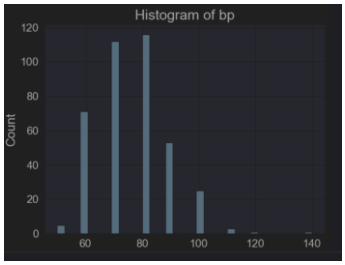
ה. תיאור שלבי הכנת הנתונים

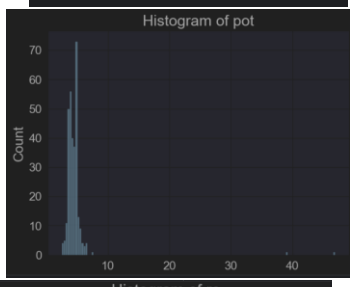
רשאית אציג היסטוגרמות של הנתונים הנומריים:

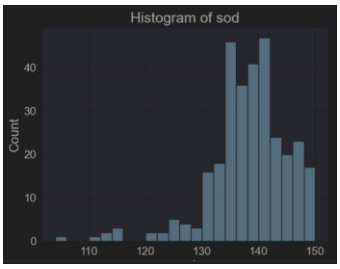


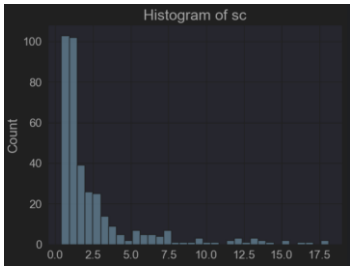


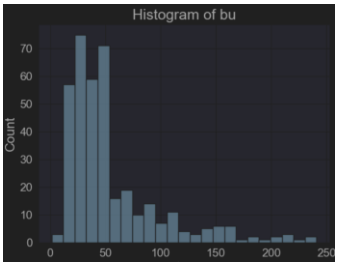


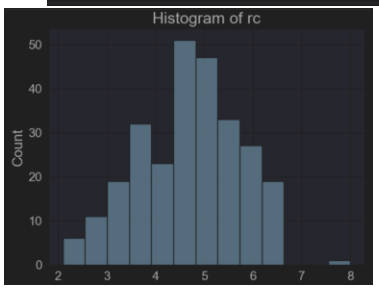


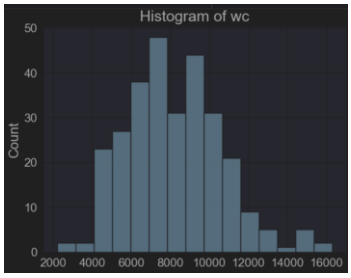


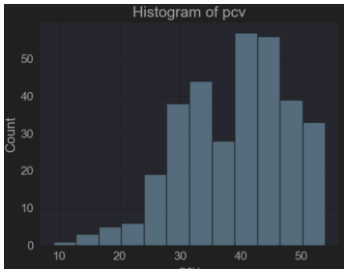


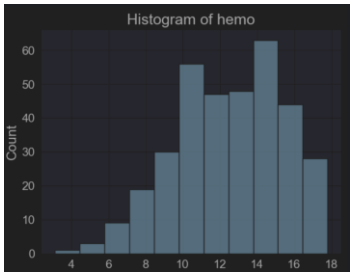


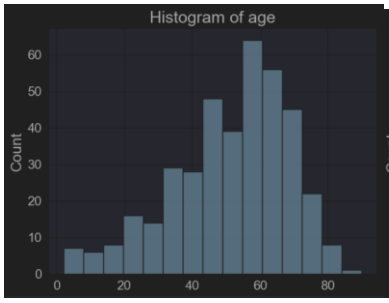


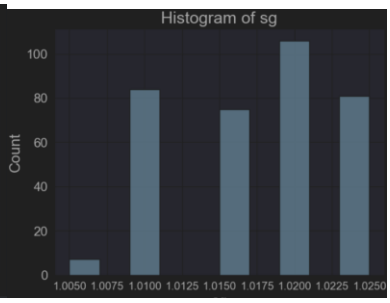




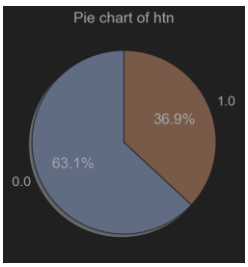


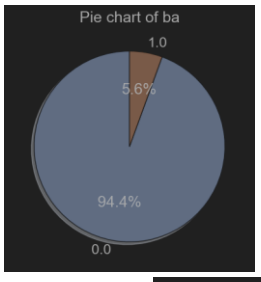


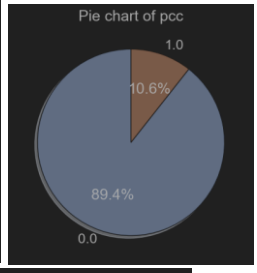


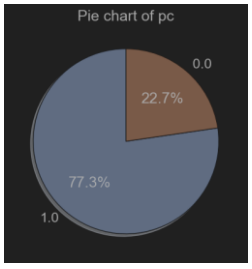


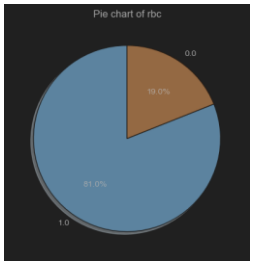
אציג דיאגרמת עוגה לתכונות הקטגוריות:

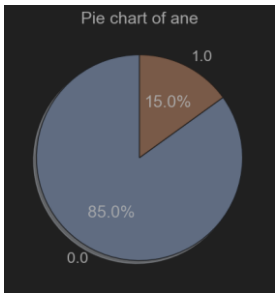


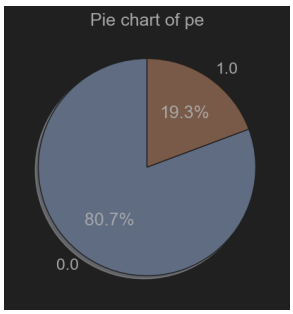


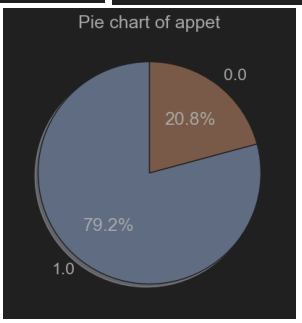


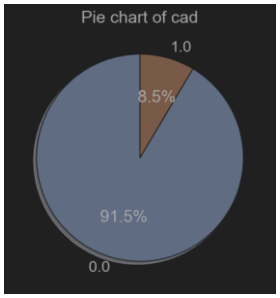


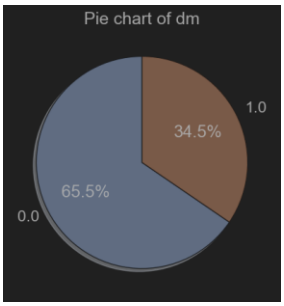




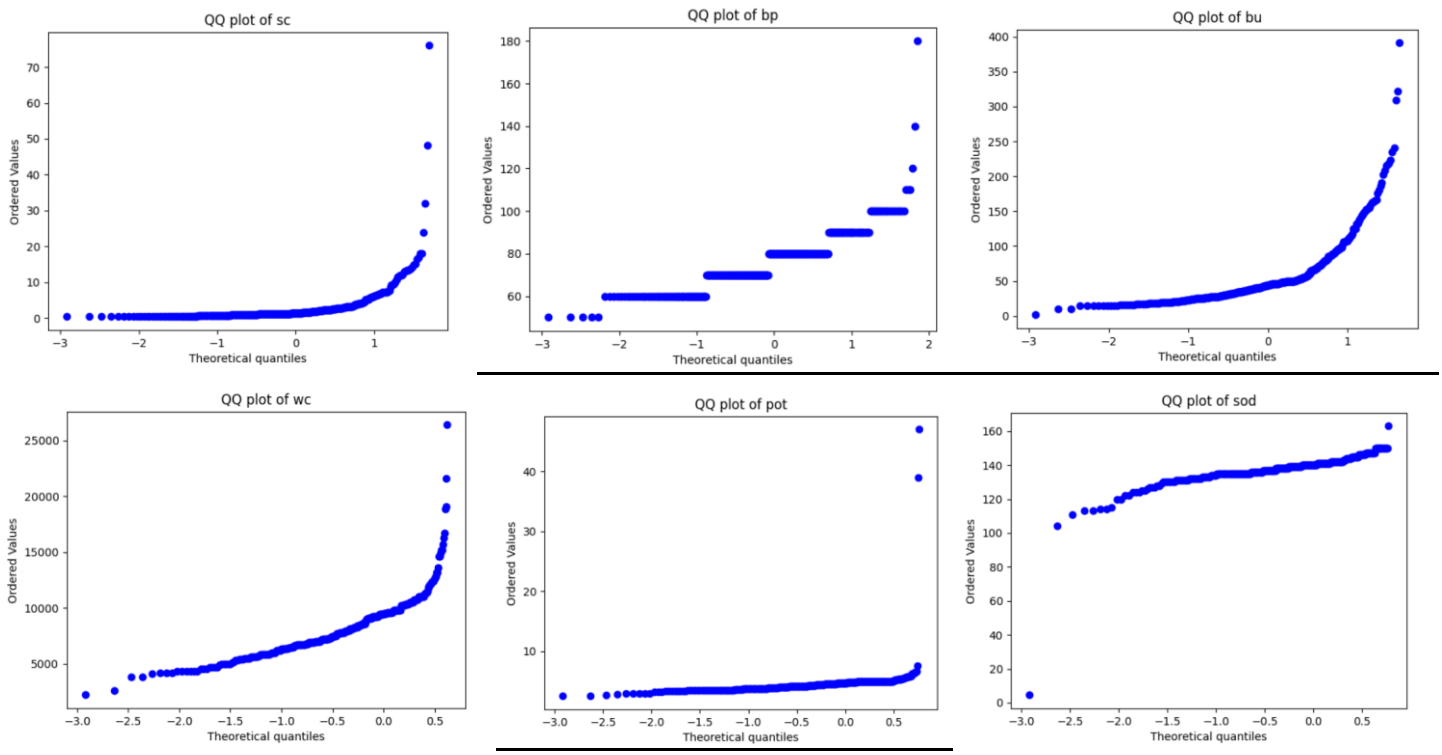








כעת אציג גרפים מסוג qq plot של התכונות שיש להן ערכים חריגים שיש למחוק (כדי לחסוך מקום לא אציג את של התכונות עם הערכים התקינים). qq plot הוא ייצוג גרפי של התפלגות קבוצת נתונים ומיועד כדי לבדוק את נוכחותם של חריגים או של "אי-נורמליות". הנתונים נחשבים להתפלגות נורמלית אם הנקודות בתרשים Q-Q יוצרים סוג של קו ישר. אם הנתונים אינם מחולקים בצורה נורמלית, הנקודות לא יפלו על קו ישר מה שמראה על סטייה מהקו, אשר מהווה מדד למידת אי תקינות הנתונים. כדי לחסוך במקום, אציג רק את התכונות בהן יש ערכים חריגים שניתן לזהות בנקל בהתבוננות בגרפים. אלו גם החריגים אותם הסרתי.



ניתן לראות שיש ערכים אינם תקינים / נקודות שאינן מתאימות. אהפוך אותם לערכי nan:

```
# After view the Q-Q plot, we can see that there are some outliers in the data
# Remove outliers rows of each feature of numeric columns.
# After it, Use KnnImputer to reassign them again
X.loc[X['bp'] >= 180, 'bp'] = np.nan
X.loc[X['bu'] > 241, 'bu'] = np.nan
X.loc[X['sc'] >= 20, 'sc'] = np.nan
X.loc[X['sc'] >= 100, 'sod'] = np.nan
X.loc[X['sod'] > 150, 'sod'] = np.nan
X.loc[X['sod'] <= 104, 'sod'] = np.nan
X.loc[X['wc'] >= 16_700, 'wc'] = np.nan
X.loc[X['bgr'] < 70, 'bgr'] = np.nan
```

אבצע פיצול לנתוני אימון ונתוני מבחן:

```
# Split the data to train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

אבצע נרמול של הנתונים הנומריים, שכן הם ביחידות מידה שונות, כך שכל ערך יהיה בין 0 ל1.

```
# Use StandardScaler on numeric columns to scale them to be in the range of 0-1
scaler = StandardScaler()
X_train[numeric_columns] = scaler.fit_transform(X_train[numeric_columns])
X_test[numeric_columns] = scaler.transform(X_test[numeric_columns])
```

אבצע סטנדרטיזציה מינימום-מקסימום על הערכים הנומריים כך ש

```
# Use MinMaxScaler on numeric columns for mean and standard deviation normalization to mean=0 and std=1
scaler = MinMaxScaler()
X_train[numeric_columns] = scaler.fit_transform(X_train[numeric_columns])
X_test[numeric_columns] = scaler.transform(X_test[numeric_columns])
```

אבצע השלמת התכונות הקטגוריות על ידי שימוש בערך השכיח ביותר בכל תכונה.

```
# Use SimpleImputer to impute missing categorical values
simple_imputer = SimpleImputer(strategy='most_frequent')
X_train[categorical_columns] = simple_imputer.fit_transform(X_train[categorical_columns])
X_test[categorical_columns] = simple_imputer.transform(X_test[categorical_columns])
```

אבצע השלמה של הנתונים החסרים בתכונות הנומריות על ידי שימוש בK השכנים הקרובים ביותר. שיטה הפועלת בגישה של אלגוריתם KNN ולא על בגישה הנאיבית של מילוי כל הערכים בממוצע או חציון. בשיטה זו, האלגוריתם מוצא את k השכנים הקרובים ביותר ("התצפיות הדומות ביותר") לתצפית עם הנתון החסר ואז משלים את הערך החסר בהתבסס על הערכים הקיימים של k השכנים הקרובים ביותר.

```
# Impute missing values of numeric columns with KNNImputer
knn_imputer = KNNImputer(n_neighbors=5)
X_train[numeric_columns] = knn_imputer.fit_transform(X_train[numeric_columns])
X_test[numeric_columns] = knn_imputer.transform(X_test[numeric_columns])
```

אמצא תכונות נומריות מרובות ערכים שיש לבצע עליהן דיסקרטיזציה.
אפשר לראות שעבור תכונות מעל 10 ערכים ייחודיים יש לבצע דיסקרטיזציה:

col_name	num_of_unique_values
bgr	186
wc	178
hemo	165
rc	155
bu	137
pot	115
sc	99
pcv	93
sod	92
age	83
sg	28
al	25
bp	19
su	15
ba	2
pcc	2
pc	2
rbc	2
htn	2
dm	2
cad	2
appet	2
pe	2
ane	2

הקוד שיוצר את הטבלה:

```
# Find features with a lot of values by printing the number of unique values in each column, using concat
col_df = pd.DataFrame(columns=['col_name', 'num_of_unique_values'])
for col in X_new.columns:
    col_df = pd.concat([col_df, pd.DataFrame({'col_name': col, 'num_of_unique_values': X_new[col].nunique()}, index=[0])], ignore_index=True)
# sort it
col_df = col_df.sort_values(by='num_of_unique_values', ascending=False)
```

לכן אבצע דיסקרטיזציה עבור תכונות נומריות מעל 10 ערכים ייחודים:

```
# KBinsDiscretizer Explanation:
# Bin continuous data into intervals and transform them with ordinal encoding. (it's a Combination of binning and encoding)
# Parameters:
#            n_bins: The number of bins to produce.
#            encode='ordinal': Encode the transformed result with values between 0 and n_bins-1.
#            strategy='quantile': Equal depth binning. The quantile strategy produces bins that have the same number of records in each bin.
#            strategy='uniform': Equal width binning. The width of each bin is (max - min) / n_bins.
# Binning age column to 5 equal-depth bins
k_bins_discretizer = KBinsDiscretizer(n_bins=5, encode='ordinal', strategy='quantile')
X_train['age'] = k_bins_discretizer.fit_transform(X_train[['age']])
X_test['age'] = k_bins_discretizer.transform(X_test[['age']])
# Binning numeric columns 3 bins with equal width
binning_columns_with_a_lot_of_values = ['bp', 'sg', 'al', 'su', 'bgr', 'bu', 'sc', 'sod', 'pot', 'hemo', 'pcv', 'wc', 'rc']
k_bins_discretizer = KBinsDiscretizer(n_bins=3, encode='ordinal', strategy='uniform')
for col in binning_columns_with_a_lot_of_values:
    X_train[col] = k_bins_discretizer.fit_transform(X_train[[col]])
    X_test[col] = k_bins_discretizer.transform(X_test[[col]])
```

כפי שמצוין בהערות בקוד, המחלקה KBinsDiscretizer מבצעת חלוקה לבינים/תיבות וגם ממירה אותם לערכים מספריים לפי סדר (אורדינלי).
עבור התכונה age אבצע חלוקה ל5 בינים לפי עומק שווה (הפרמטר quantile). כלומר כמות שווה של דגימות בכל "בין"/תיבה.

עבור שאר התכונות אבצע חלוקה ל3 בינים לפי רוחב שווה (הפרמטר uniform). כלומר טווח כל "בין"\תיבה הוא שווה אורך.

כעת אבחר את התכונות הטובות ביותר כדי לחזות את משתנה המטרה. לאחר בדיקה, מצאתי כי הפרמטר f_classif נותן את הביצועים הטובים ביותר ולכן אבחר בו. f_classif הוא מדד למובהקות של קשר בין שני משתנים או יותר בעזרת-ANOVA (ניתוח שונות). בחרתי ב15 תכונות לאחר שהתבוננתי בטבלה מתחת.

```
# Select K best features (score_func options are: f_classif, mutual_info_classif, chi2, f_regression, mutual_info_regression)
k_best = SelectKBest(k=15, score_func=f_classif)
X_train = k_best.fit_transform(X_train, y_train)
X_test = k_best.transform(X_test)
```

תוצאות הקשר בין כל תכונה למשתנה המטרה:

	Features	Scores
14	hemo	435.26208
2	sg	341.79317
15	pcv	331.85513
18	htn	157.89070
19	dm	151.54688
17	rc	121.16603
3	al	102.97837
12	sod	67.94396
21	appet	63.41654
6	pc	51.46052
22	pe	50.22696
1	bp	49.32078
9	bgr	46.80483
23	ane	36.69030
0	age	36.08245
10	bu	33.49102
4	su	26.55753
5	rbc	25.13460
7	pcc	22.50516
16	wc	21.60680
20	cad	19.14276
11	sc	18.88400
8	ba	12.12375
13	pot	1.72506

הקוד ליצירת הטבלה:

```
# print all the features with their scores, including the ones that were not selected
features_scores = pd.DataFrame({'Features': X.columns, 'Scores': k_best.scores_}).sort_values(by='Scores', ascending=False)
```

עד כאן הכנת הנתונים.

שאלה 2

א) אבחר בשיטת Random Forest ובשיטת AdaBoost Classifier.

a. **בשיטת Random Forest** יש שימוש בכמה עצים. הפקת עץ החלטה נעשית מתת קבוצה של התכונות של הנתונים, אשר שיבחרו באופן **אקראי**. ובנוסף כל אחד מעצי ההחלטה יופק מתת קבוצה **אחרת** של תצפיות האימון. כל קודקוד של כל עץ נבחר מתוך תת הקבוצה של התכונות שנבחרה. RandomForest מביאה בדר"כ לתוצאות טובות יותר מבחינת רמת הדיוק מAdaBoost בגלל שאלגוריתם AdaBoost המקורי לא משתמש בכל התכונות בכל עץ אלא רק באחד מהם עבור כל עץ. שיטת עץ אקראי עמידה יותר לנתונים בעלי הרבה רעש. האלגוריתם יבנה את העץ הגדול ביותר (ללא גיזום) - בדומה לאלגוריתם CART.

b. **שיטת AdaBoost** היא שיטת אנסמבל. בכל איטרציה מתווספים עצים "חלשים" - כולם בעלי רמה אחת (באלגוריתם הבסיסי) ובנוסף בכל איטרציה כל עץ מקבל משקל שונה ל"כוח" של בסיווג הסופי, ולכן נקרא Adaptive. נתונים אשר סווגו בצורה שגויה מקבלים יותר משקל, וכאשר מוסיפים עץ נוסף באיטרציה הבאה, הנתונים יקבלו יותר משקל בקבלת ההחלטה היכן לסווגם ובכך יתבצע תיקון ושיפור הסיווג. משקל נתון שגוי מקבל יותר משקל, ולכן זה נקרא Boosting.

ב) אציג את תיאור שלבי השיטות.

a. אתחיל מ-Random Forest.

<u>האלגוריתם הבסיסי (2001):</u>

עבור כל אחד מK עצי ההחלטה נבצע:

1. נדגום בשיטת bootstrap תצפיות (כגודל מאגר הנתונים המקורי)
2. נבחר אקראית תת קבוצה של תכונות.
3. כל קודקוד יבחר מתוך תת הקבוצה של משתנים שנבחרה.

<u>מאפייני האלגוריתם</u>

1. בעל דיוק רב בסיווג.
2. האלגוריתם יעיל ומהיר על בסיסי נתונים גדולים בזכות השימוש בחלק מהמשתנים וגם יש הזדמנות למקבול - בניית העצים במקביל כי הם לא תלויים אחד בשני.
3. מתמודד טוב גם עם מאגרי נתונים עם אלפי משתנים.
4. ניתן להעריך את חשיבות כל משתנה.
5. לזהות קשרי גומלין בין המשתנים השונים.
6. לקבל הערכה בלתי מוטת לשגיאה האמיתית של המודל.

עבור תצפית חדשה, כל מודל יחזה את התצפית, ולבסוף יתבצע דירוג עבור הסיווגים. אפשר להשתמש ביער האקראי כדי להשלים נתונים חסרים (בעלים). וגם אם חסרים אז האלגוריתם הזה יספק רמת דיוק יחסית גבוהה.

b. אפרט על AdaBoost:

<u>האלגוריתם</u>

1. באיטרציה הראשונה כל תצפית מקבל משקל זהה.
2. נריץ את האלגוריתם k פעמים:
   a. דוגמים מאגר נתוני אימון בגודל המאגר המקור (Bootstrap) . אם תצפית נבחרה באיטרציה הקודמת אז יש יותר סיכוי שהיא תבחר גם באיטרציה הנוכחית.
   b. נייצר מודל סיווג עבור מאגר הנתונים הנוכחי ונחשב את שגיאת המודל. במידה ותצפית לא מסווגת נכון, נגדיל את המשקל שלה. ואילו תצפיות שסווגו בצורה נכונה יקבלו משקל נמוך יותר.

ראשית אבחר את הפרמטרים למודל Random Forest בעזרת GridSearchCV אשר מוצא את הפרמטרים הטובים ביותר בהינתן:

1) מודל – שיטת הסיווג. במקרה הזה Random Forest.
2) מדד הצלחה - מדד להערכת הצלחת המודל
3) מספר הfolds – כמה חלוקות של הנתונים לקבוצות יש לבצע כדי לבצע עליהם את חישוב ההצלחה של המודל.
4) מקסימום תכונות – מספר התכונות שלוקחים בחשבון כשמחפשים את הפיצול הטוב ביותר.

```
# Use grid_search to find the best parameters for random forest classifier
import multiprocessing
n_jobs = multiprocessing.cpu_count()
print(f'Number of CPUs: {n_jobs}')
rfc = RandomForestClassifier(random_state=1)
param_grid = {'n_estimators': [80, 100, 200],
              'min_samples_split': [2, 3, 4],
              'max_depth': [3, 4, 5, 6, 7, 8],
              'criterion': ['gini'],
              'max_features': ['sqrt', 'log2']}

grid_search = GridSearchCV(rfc, param_grid, cv=10, scoring='accuracy', verbose=n_jobs-1)
grid_search.fit(X_train, y_train)
print(param_grid)
print(f'Best parameters: {grid_search.best_params_}')
print(f'Best score: {grid_search.best_score_}')
```

נמצא שהפרמטרים הטובים ביותר הם:

```
# Define the best parameters for Random Forest Classifier
random_forest_best_params = {'criterion': 'gini', 'max_depth': 5, 'max_features': 'sqrt', 'min_samples_split': 2, 'n_estimators': 80}
```

אעבור לשיטת סיווג AdaBoost

ראשית אבחר את הפרמטרים למודל AdaBoost בעזרת GridSearchCV אשר מוצא את הפרמטרים הטובים ביותר בהינתן:

1) מספר העצים – מספר עצי ההחלטה שמשתתפים באנסמבל.
2) יחס הלמידה – מה מידת ההשפעה של כל תת מודל בתוך האנסמבל.
3) עץ עם עומק שנמצא מראש – אפשר לבחור להפוך כל עץ לפחות "חלש" (יותר "חזק") על ידי העמקתו (לא אשתמש בפרמטר זה בשל סיבוכיות המימוש שלו ברגע וכי הוא לא חלק מהפרמטרים שחבילת sklearn מספקת).

```
# Use grid_search to find the best parameters for AdaBoost classifier
model = AdaBoostClassifier(random_state=1)
grid = {
    'n_estimators': [500, 1000, 2000, 5000],
    'learning_rate': [0.0001, 0.001, 0.01, 0.1, 1.0]
}
grid_search = GridSearchCV(estimator=model,
                           param_grid=grid,
                           cv=10,
                           scoring='accuracy',
                           verbose=5)
print('Fit the grid search model...')
grid_result = grid_search.fit(X_train, y_train)
print(f"Best: {grid_result.best_score_} using {grid_result.best_params_}")
```

ג) <u>תוצאות המודלים</u>

Random Forest - אצור את המודל ואבצע 10-cross-validation לפי המצוין במאמר המצורף.

```
# Initialize the Random Forest Classifier
rfc_model = RandomForestClassifier(**random_forest_best_params, random_state=42)
```

```
# Perform cross validation with 10 folds.
# Using a loop and not automatic function because I want to append the scores to few lists and display details of each fold
rfc_k_fold = KFold(n_splits=10, shuffle=True, random_state=42)
rfc_k_fold.get_n_splits(X_train)
folds_y_tests, predictions_of_all_folds = [], []
train_scores, test_scores = [], []
confusion_matrix_list = []
for train_index, test_index in rfc_k_fold.split(X_train):
    # Define the train and test sets for each fold
    X_train_k_fold, X_test_k_fold = X_train[train_index], X_train[test_index]
    y_train_k_fold, y_test_k_fold = y_train.iloc[train_index], y_train.iloc[test_index]
    # Fit the model and predict of current fold
    rfc_model.fit(X_train_k_fold, y_train_k_fold)
    # Calculate the scores on test set
    y_pred = rfc_model.predict(X_test_k_fold)
    # Append the y_test and y_pred to the lists for the classification report
    folds_y_tests.extend(y_test_k_fold)
    predictions_of_all_folds.extend(y_pred)
    # Append the scores to the lists
    train_scores.append(rfc_model.score(X_train_k_fold, y_train_k_fold))
    test_scores.append(rfc_model.score(X_test_k_fold, y_test_k_fold))
    confusion_matrix_list.append(confusion_matrix(y_test_k_fold, y_pred))
```

במקובל בcross-validation אבצע ממוצע של התוצאות כדי לקבל את דיוק המודל ואקבל:

```
Random Forest Classifier 10-Cross Validation Results:
```

Fold	Train scores	Test scores
1	0.989583	1
2	0.996528	1
3	0.989583	1
4	0.989583	1
5	0.989583	1
6	0.996528	0.9375
7	0.989583	1
8	0.993056	0.96875
9	0.989583	1
10	0.993056	1

```
Mean train score: 0.9916666666666666
Mean test score: 0.990625
```

כמתואר במאמר, קיבלתי הצלחה של 99% בנדרש.

הtest score מאוד קטן כאשר יש 10 folds ולכן קיבלתי 100% הצלחה במרבית הfolds.

AdaBoost – כמו כן אצור את המודל ואקבל את תוצאות הcross-validation לפי המצוין במאמר המצורף.

```
AdaBoost 10-Cross Validation Results:
+--------+----------------+---------------+
|   Fold |   Train scores |   Test scores |
|--------+----------------+---------------|
|      1 |       0.982639 |       1       |
|      2 |       0.982639 |       1       |
|      3 |       0.986111 |       0.96875 |
|      4 |       0.982639 |       1       |
|      5 |       0.982639 |       1       |
|      6 |       0.989583 |       0.9375  |
|      7 |       0.913194 |       0.875   |
|      8 |       0.986111 |       0.96875 |
|      9 |       0.982639 |       1       |
|     10 |       0.986111 |       0.96875 |
+--------+----------------+---------------+
```

```
Mean train score: 0.9774305555555556
Mean test score: 0.971875
```

במאמר קיבלו 96% אך אני קיבלתי 97% הצלחה. היו לי כמה הרצות שקיבלתי 96 אחוזי הצלחה. לבסוף הגעתי ל97%. בנוסף ניתן לראות שהיו כמה folds בcross-validation שקיבלו הצלחה של 96%, 93% ואפילו 87%. אך היו גם כאלה עם 100%.

ד) <u>הערכת מידת הדיוק של כל שיטה</u>

<u>הערכת Random Forest</u>

אשתמש בconfusion matrix אשר סוכם את כל התוצאות של כל הfolds. ובנוסף אשתמש במדדים המקובלים להערכת המודל.

```
Classification Report of all folds:
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       122
           1       0.99      0.99      0.99       198

    accuracy                           0.99       320
   macro avg       0.99      0.99      0.99       320
weighted avg       0.99      0.99      0.99       320
```

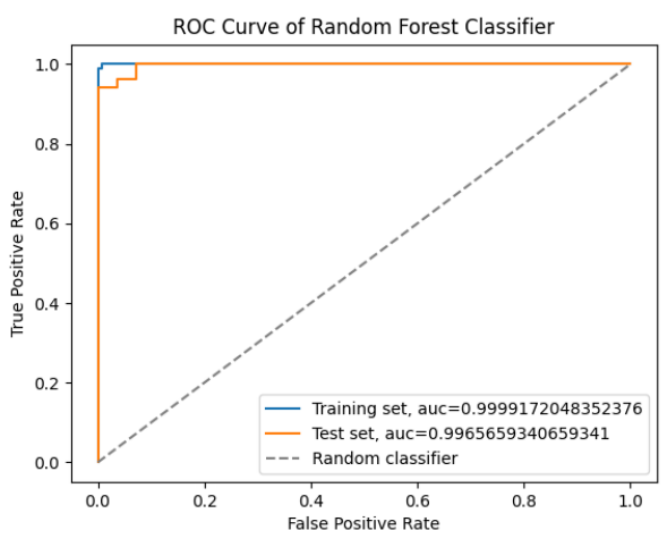
```
Confusion Matrix:
+----------------+---------------------+---------------------+
|                |   Predicted Positive |   Predicted Negative |
|----------------+---------------------+---------------------|
| Actual Positive |                 121 |                   2 |
| Actual Negative |                   1 |                 196 |
+----------------+---------------------+---------------------+
```

הקוד:

```
# Sum all the confusion matrices from the cross validation
confusion_matrix_sum = np.sum(confusion_matrix_list, axis=0)

# Print the confusion matrix with tabulate
print('Confusion Matrix:')
print(tabulate.tabulate({'': ['Actual Positive', 'Actual Negative'],
                         'Predicted Positive': [confusion_matrix_sum[0][0], confusion_matrix_sum[0][1]],
                         'Predicted Negative': [confusion_matrix_sum[1][0], confusion_matrix_sum[1][1]]},
                         headers='keys',
                         tablefmt='psql'))
```

```
# Print the classification report for the sum of the folds predictions
print('Classification Report of all folds:')
print(classification_report(folds_y_tests, predictions_of_all_folds))
```

לבסוף, אבצע הערכת דיוק (accuracy) על הtest set שפוצל בהתחלה ולא השתמשתי בו בכלל בcross-validation ואציג אותו עם ROC curve.




```
Mean Squared Error: 0.025
Test score: 0.975
```

הערכת AdaBoost

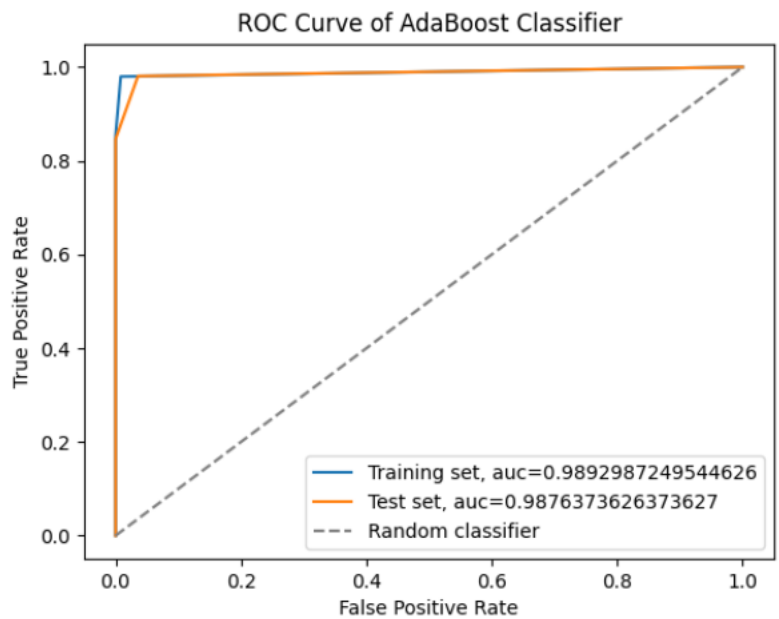
```
Classification Report of all folds:
              precision    recall  f1-score   support

           0       0.94      0.99      0.96       122
           1       0.99      0.96      0.98       198

    accuracy                           0.97       320
   macro avg       0.97      0.98      0.97       320
weighted avg       0.97      0.97      0.97       320
```

```
Confusion Matrix:
+----------------+---------------------+---------------------+
|                |   Predicted Positive |   Predicted Negative |
|----------------+---------------------+---------------------|
| Actual Positive |                 121 |                   8 |
| Actual Negative |                   1 |                 190 |
+----------------+---------------------+---------------------+
```

הערכת דיוק (accuracy) על הtest set:

```
Mean Squared Error: 0.025
Test score: 0.975
```

ROC curve:



ה) מהנתונים בסעיף הקודם ניתן לראות כי תוצאות Random Forest Classifier (RFC) טובות יותר מ(AdaBoost (AB.

תוצאות הcross validation-10: RFC קיבל 99% ואילו AB קיבל 97%. מה שמראה ש RFC מודל טוב יותר.

מדד הAUC (השטח שמתחת לעקומה): RFC קיבל 99% ואילו AB קיבל 98%. מה שמראה ש RFC מודל טוב יותר.

confusion matrix - לRFC יש 2 False Negative ואילו לAB יש 8 False Negative. עם זאת הFalse Positive של שניהם הוא 1. מה שמראה ש RFC מודל טוב יותר.

Precision	Precision = True Positives / (True Positives + False Positives) כמה מתוך אלו שסווגו כחיובים הם באמת חיוביים.
Recall	Recall = True Positives / (True Positives + False Negatives) זהה לSensitivity.

מדד הF1 - הסבר: זהו שילוב של מדדי הPrecision והRecall ומשמעותו הוא ממוצע הרמוני שלהם ונותן לכל אחד מהם משקל שווה. החישוב הוא:

F1 Score = 2 * Precision * Recall / (Precision + Recall)

RFC קיבל ציוני 99% בכל המדדים ואילו AB קיבל 96,97,98 במדדים השונים.

הצעות לשיפור

בכללי, אפשר למצוא עוד בסיסי נתונים עם תכונות זהות, להוסיף אותם ולאמן על יותר נתונים.

ניתן לבצע SMOTE – שיטת דגימת יתר UPSAMPLING - דוגמים אקראית תצפית ששייכת למחלקת המיעוט, בוחרים את K השכנים הקרובים ביותר של אות תצפית (שהם גם ממחלקת המיעוט) ויצירת תצפית חדשה סינטטית בעזרת אינטרפולציה רנדומלית. זאת עד אשר יושג היחס הרצוי בין תצפיות המיעוט לתצפיות הרוב.

Random Forest

אפשר להמשיך ולבדוק עוד פרמטרים כדי לשפר את דיוק המודל. אפשר לנסות לבחור תכונות אחרות לבצע עליהן את אימון המודל. אפשר לנסות לבצע Smote – שיטת דגימת יתר. אפשר להשתמש בXGBoost עם עצים, אשר ידוע שמשפר מודלים מאוד.

AdaBoost

אפשר לשפר את האנסמבל על ידי שימוש במודלים שונים (לא בעץ החלטה) כמודלי האנסמבל. המודלים האלה חייבים לסווג לפי ציונים הסתברותיים או ממושקלים. לדוגמא אפשר להשתמש בLogisticRegression .

הייתה עבודה מהנה ומלמדת! הוספתי במחברת פייתון גם חיזוי של Naïve Bayes.

תודה 😊