פרויקט לימוד מכונה

חלק ב

מרצה הקורס: פרופ' בועז לרנר

בודק תרגילים ואחראי מעבדות: רועי וולף

בסיס נתונים: Paris Housing Data

תאריך הגשה: 14/01/2023

לימוד מכונה 364-1-1811

eitan halley

קבוצה 23

Table Of Contents

Contents:

Model Training

הצגת הנתונים לאחר סינון ובחירת השדות הרלוונטיים מחלק א':

לאחר שהתחלתי לעבוד על סט האימון שבניתי בחלק א' הבנתי שכל הנראה "הרסתי" את הנתונים עם סוג של התאמת יתר לעומת סט האימון המקורי ולכן החלטתי לתקן את הסט נתונים מחדש. במקום לעבוד עם הנתונים השגויים הסרתי את כולם (998 שדות), כלומר בכל מקום שהיה na ובכל מקום שהיה נתון לא הגיוני (לפי התיאור של חלק א') הסרתי את השדה, אני מבין שזה עלול להקשות על המודל להתמודד עם רעש אבל העדפתי לאמן מודל על נתונים איכותיים לעומת אימון מודל על נתונים שגויים שעברו טרנספורמציות שגויות. לאחר מכן השתמשתי בכל הנתונים שנשארו ונרמלתי אותם בעזרת StandardScaler, לבסוף הרצתי רגרסיה לוגיסטית וקיבלתי דיוק של 80%. לכן בחלק ב' אשתמש בסט הנתונים שהוסר ממנו הנתונים הבעייתיים ואאמן את הסטים בעזרתו.(נספח1)

הכנת הנתונים לאימון ובחירת מערכת לומדת:

בשלב האימון בחרתי לחלק את סט הנתונים כך ש-20% מכלל הנתונים יהוו סט הבחינה ו-80% מהנתונים יהוו סט האימון וסט הוולידציה, זאת מפני שחלוקה זאת מתקבלת עקב גודל המדגם וכמות הנתונים מספקת עבור סט אימון וסט הבחינה. בנוסף על מנת לבצע אימון למודל וכוונון פרמטרים, אני אשתמש בשיטת K-Fold cross-validation.

Decision Tree

כדי להשתמש במודל של עץ החלטה, המרתי את כל המשתנים בעזרת השיטה StandardScaler. אני אתחיל את האימון על ידי בניית עץ החלטה עם ערכים דיפולטיביים, לאחר מכן כוונתי את הפרמטרים בעזרת GridSearch לבסוף בחרתי את הערכים שהניבו את המודל עם הדיוק הגבוה ביותר. בחרתי את ה-K להיות 10 מפני ששוחחתי עם דאטה סיינטיסט והוא ציין שעבור סט נתונים בגודל כזה 10 הוא הגודל הטוב ביותר מפני שנוצר 9 סטי אימון וסט ולידציה יחיד.

```
Train accuracy    1.0
dtype: float64
Test accuracy    0.895789
dtype: float64
```

1)בניית עץ החלטה:

לאחר הרצת האלגוריתם המתאים קיבלתי דיוק עבור סט האימון של כ-100% ועבור סט הבדיקה כ-89.57%. מהתוצאות ניתן להבין שהמודל שהרצתי עבור עץ מלא במצב של התאמת יתר, דרך המימוש נוצר התאמה מושלמת של הנתונים לסט האימון אשר עלול לפגוע בגמישות ולהניב תוצאות שגויות עבור סט שלא זהה לסט זה. בהמשך נממש את העץ בעזרת קריטריונים נוספים אשר עשויים לשפר את אחוז הדיוק של סט הבחינה, למשל מיקסום ערך הדיוק של סט הבחינה יצור מודל גמיש שעשוי לתת ניבוי איכותי יותר לסט הבחינה.

1.2) כוונון פרמטרים:

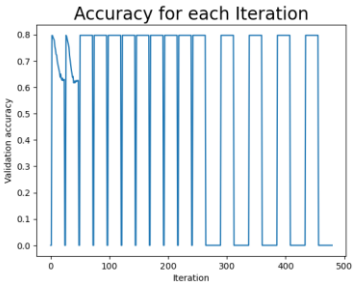
לאחר הרצאת המודל הפשוט ביצעתי כוונון פרמטרים בעזרת GridSearchCV כך שישלב בין K-Fold ו-cross-validation, ובכך אני עשוי למצוא את המאפיינים שאצטרך למקסם/למזער. זאת מפני ששיטה זו עוברת על כל הקומבינציות האפשריות בטווח הערכים המוזן. בחרתי לבחון את המאפיינים: max_depth, criterion: [entropy, gini], Ccp alpha.

2.1.1) ערכי ההיפר-פרמטרים שנבחנו כפונקציה של אחוז הדיוק על סט האימון וסט האימות:

הרצתי GridSearchCV וקיבלתי שהמאפיינים הטובים ביותר לחיזוי איכותי הינם:(נספח 2)

```
{'ccp_alpha': 0.0, 'criterion': 'gini', 'max_depth': 3}
```

20	entropy	21	0.0	0.989035	0.634438
21	entropy	22	0.0	0.993308	0.625450
22	entropy	23	0.0	0.995146	0.626032
23	entropy	24	0.0	0.996528	0.630078
24	gini	1	0.0	0.000000	0.000000
25	gini	2	0.0	0.000000	0.000000
26	gini	3	0.0	0.797658	0.797193
27	gini	4	0.0	0.800507	0.793655
28	gini	5	0.0	0.804015	0.782751
29	gini	6	0.0	0.813181	0.773184
30	gini	7	0.0	0.827999	0.762935
31	gini	8	0.0	0.846901	0.751677
32	gini	9	0.0	0.866723	0.720994



בטבלה המצורף ניתן לראות שעבור המאפיינים שנבחרו אנו נקבל מדד f1 מקסימלי לסט הוולידציה, כל שינוי עלול להוריד את המדד אך לא למקסם אותה. בנוסף ניתן לראות שקריטריון **gini** נותנת רמת דיוק טובה יותר מאשר **entropy**. **מאפייני מקס עומק**, ככל ש**מקס עומק** גדל כך דיוק סט הוולידציה יורד ודיוק סט האימון עולה כלומר אם נריץ את **עומק מקס** עד מספר גדולמאוד נקבל התאמת יתר. עבור **Ccp alpha** קיבלתי בזמן הרצת הבדיקות עבור **alpha>0.1** דיוק של 0 עבור סט האימון וגם סט הוולידציה לכן בדקתי שוב עד 0.1 וקיבלתי **ccp_alpha = 0.0** .

1.2.2) מוטיבציה לבחירת הפרמטרים:

Gini – הינו מדד אשר מתייחס למדידה של הסבירות לסיווג שגוי של מופע חדש של משתנה אקראי בתנאי שהוא הניב סיווג שגוי. בזמן בניית עץ החלטה האלגוריתם יעדיף תכונות בעלות ערך מדד נמוך.

Entropy – הינו מדד של חוסר סדר או אקראיות בנתונים, כאשר מדד זה יהיה גבוהה נקבל עץ עם הרבה רעש לכן נרצה למזער אותה ובכך נוכל לדעת איזו תכונה מספקת את מירב המידע לשם סיווג בזמן בניית העץ. בחרתי למדוד בין השנים כדי למצוא בעזרת אחד מהם ניבוי אופטימלי.

Max Depth – מדד זה עוזר לנו להכריע את עומק העץ על מנת לקבל ניבוי אופטימלי, בעזרתו אני אדאג לא להגיע למצב של overfitting(עומק גדול מידי שמתאים במדויק לסט שאיתו בניתי את העץ) וגם לא למצב של underfitting(עומק קטן מידי שלא אתן לי מספיק מידע).

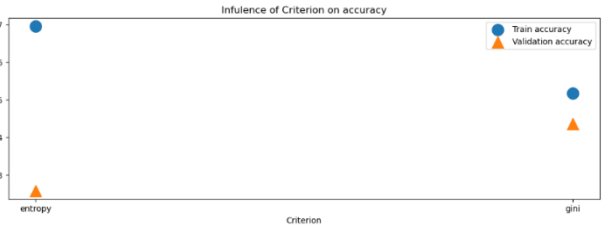
Ccp alpha - הינו היפר-פרמטר אשר מגדיר את הסף עבור ביצוע קטימה בעץ. האלפא עבור כל צומת נקבעת על פי מובהקות הפיצול ולכן ככל שהאלפא נמוכה יותר הפיצול פחות מובהק ולפי כך ניתן להבין אם צריך לקטום את העץ באותו שלב או לא. המוטיבציה לכוונון פרמטר זה

הייתה בכך שאני משתמש באלפא על מנת למנוע Overfitting ובכך אוכל לשמור על עץ אינפורמטיבי ומצד שני עץ בעל יכולת הכללה ולא מותאם לסט נתונים ספציפי.

1.2.3) משמעות על העץ הנלמד כתוצאה מהגדלת/הקטנת ערכי ההיפר-פרמטרים:

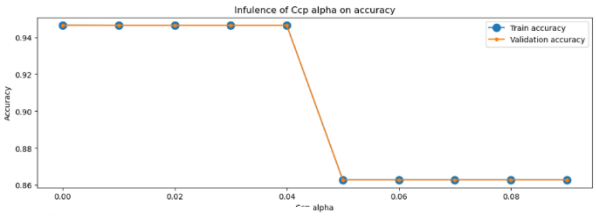
בשלב הנוכחי אני אבחן את ההשפעה של כל אחד מההיפר-פרמטרים על העץ ההחלטה כאשר שאר ההיפר-פרמטרים יקבלו את הערך האופטימלי כפי שהתקבל ב-2.1, המוטיבציה לבדיקה בשיטה זו הינה מפני שההיפר-פרמטרים משפיעים זה על זה.

השפעת Criterion על מדד Classification Accuracy:



ניתן לראות כי הקריטריון **entropy** נותן דיוק טוב יותר עבור סט האימון אך סט הווליידציה מקבל דיוק נמוך מאוד, לעומת זאת קריטריון **gini** נותן רמת דיוק טוב גם לסט הווליידציה וגם לסט האימון. לכן קריטריון **gini** תניב תוצאות טובות יותר עבור סטים מגוונים השונים מסט האימון.

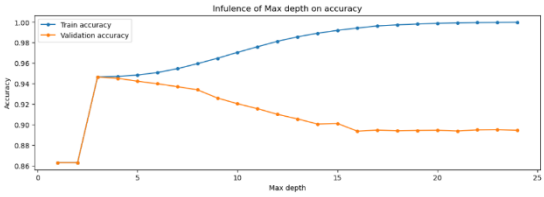
השפעת Ccp alpha על מדד Classification Accuracy:



$$\sum_{m=1}^{|T|} \sum_{i:\, x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

ניתן לראות שקיים התאמה מושלמת בין סט הווליידציה לבין סט האימון, בנוסף עבור alpha בין 0 לכ-0.05 אני אקבל דיוק מקסימלי. מכיוון ש-**alpha** שהמודל בחר הינו ערך 0 אז ניתן ללמוד שה-Bias שווה לאפס ניתן לראות זאת בנוסחת CCP.

השפעת Max depth על מדד Classification Accuracy:



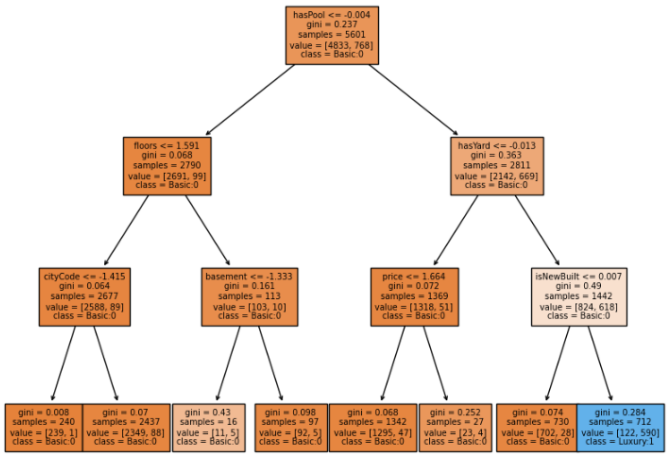
כפי שציינת בסעיף 2.1 ניתן לראות שככל ש-**Max_depth** עולה ככה סט האימון שואף להתאמת יתר מפני שהעץ מתאים את עצמו יותר ויותר לסט האימון בכל שכבה, לעומת זאת ככל ש-**Max_depth** יעלה ככה דיוק של סט הווליידציה ירד. דיוק הטוב ביותר עבור סט הווליידציה וגם האימון בו זמנית מתקבל עבור **Max_depth** של 3.

1.3) אימון עץ החלטה:

3.1)אחוזי דיוק ומסקנות: ניתן לראות שקיבלתי אחוזי דיוק גבוהים מאוד. בנוסף ניתן לראות כי לאחר כיוונון ההיפר פרמטרים לפי הסעיפים הקודמים, אקבל ערך דיוק מקס' עבור סט הוולידציה אך עם זאת ערך דיוק עבור סט האימון ירד ואינו מקס'. מתוך הממצאים אסיק כי דיוק מקס עבור סט האימון עלול להוביל להתאמת יתר שעלול ליצור עץ החלטה לא גמיש אשר יוביל לסיווג שגוי עבור סט חדש ששונה מסט האימון.

```
Train accuracy    0.946438
dtype: float64
Test accuracy     0.942184
dtype: float64
```

1.3.2) עץ החלטה:
(נספח 3)



1.3.3) רשומה לדוגמה:

	squareMeters	numberOfRooms	hasYard	hasPool	floors	cityCode	cityPartRange	numPrevOwners	made	isNewBuilt	hasStormProtector	attic	garage	hasStorageRoom	hasGuestRoom	price
0	-0.138116	-0.657135	-1.013119	0.996258	0.091475	-0.493228	-0.511699	0.529738	0.912182	-0.992706	-1.003040	1.381522	-1.693167	-1.001251	0.362173	-0.137990
1	-0.695868	-0.292877	0.987050	0.996258	-0.563707	0.466708	-0.165936	-1.584559	0.384445	1.007347	-1.003040	-0.381366	0.552872	-1.001251	1.629806	-0.694620

```
0    0
1    1
2    0
3    1
4    0
Name: category
```

תצפית 1 (לפי אינדקס) עם המשתנים המנורמלים מפני שהשתמשתי ב-StandardScaler כדי לבנות את סט האימון ובחינה, איתם אימנתי את העץ, תצפית 1 תסווג ימינה (F) מפני ש- hasPool = 0.996258, אחר כך תסווג תסווג ימינה (F) מפני ש- hasYard = 0.987050, לאחר מכן תסווג ימינה (F) מפני ש- isNewBuilt = 1.007347 , לבסוף תקבל ערך כצפוי Luxury:1. ניתן לראות שהתצפית קיבלה את הסיווג המתאים.

1.3.4) תובנות ממבנה העץ: ניתן ללמוד לפי מבנה העץ מה הם המאפיינים המשפיעים ביותר, מעבר לכך את דרך הסיווג עבור כל תצפית לפי הערכים שלהם, ניתן לראות כי שלושת המאפיינים המשפיעים ביותר על סיווג Luxury הינם: isNewBuilt, hasYard, and hasPool.

```
                a
isNewBuilt  0.574418
hasYard     0.274396
hasPool     0.146522
basement    0.002386
price       0.000864
floors      0.000844
cityCode    0.000569
```

1.3.5/6) **Feature importance:** מתוך הפונקציה ניתן לראות את החשיבות של כל השדות בקבלת החלטות של עץ ההחלטה שנוצר. מעבר לכך ניתן לראות כי isNewBuilt הוא המשתנה עם המשקל הגדול ביותר, נתון זה מתיישב עם עץ ההחלטה מפני שלא ניתן לקבל סיווג 1 (יוקרה) ללא ערך זה. בנוסף hasYard ו-hasPool יש השפעה רבה על התהליך ניתן לחזות זאת לפי מיקומם בעץ, לעומת זאת squareMeters, numberOfRooms, cityPartRange,

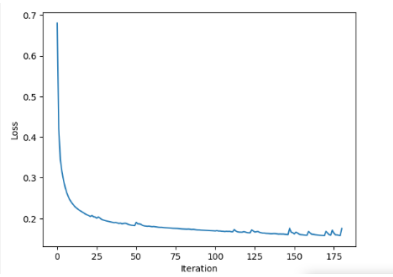
numPrevOwners, made,hasStormProtacter, hasStorageRoom, hasGuestRoom הם שדות חסרי חשיבות לפי עץ ההחלטה נתון זה לא מפתיע מפני שבחרתי לעבוד עם כל השדות וידעתי שחלקם אינם רלוונטיים לסיווג אך בכל זאת רציתי לבדוק איך המודל יתמודד איתם .

Neural Networks

2.1) אימון ובחינת רשת נוירונלי בעזרת ערכי ברירת מחדל:

```
Train accuracy    0.940903
dtype: float64
Test accuracy    0.92434
dtype: float64
```

ראשית נסביר את משמעות הקונפיגורציה, הערכים הדיפולטיים של קלאספייר MLP : 17 נוירונים בשכבת כניסה כלומר כמספר השדות, שכבה אחת חבויה, 100 נוירונים בשכבה החבויה, 2 נוירונים בשכבת יציאה כלומר כמספר הפלטים האפשריים(0,1) ו-175 איטרציות.



דיוק סט האימון המתקבל הינו כ-94% וסט בחינה הינו כ-92%, לפי הדיוקים האלו לא נראה שאני בהתאמת יתר מפני שדיוק סט ואימון ובחינה יחסית קרובים אך בכל זאת ננסה לצמצם את הפער. בגרף ניתן לראות את פונקציית המטרה Loss כפונקציה של מספר Iteration, את ה-Loss נרצה למזער כך שנוכל להריץ מספר גדול יותר של Iteration. פונקצית ה-Loss בסיווג הינו ממוצע של Cross-Entropy.

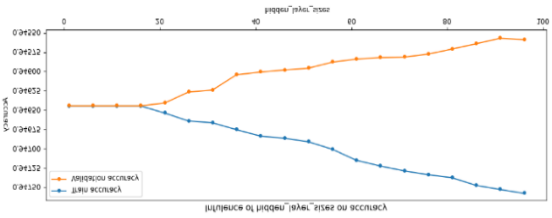
2.2) כוונון פרמטרים:

החלטתי לבצע GridSearch ולהתמקד בהיפר-פרמטרים activation ו-sizes_layer_hidde , כמו סעיף 2 השתמשתי ב-Kross validation עבור 10 פולדים. בחרתי בהיפר-פרמטרים האלו מפני שסט הבדיקה והאימון קיבלו דיוק יחסית קרוב וקראתי באינטרנט שבמצב כזה מומלץ לחקור את פונקציית האקטיבציה, מספר נוירונים בשכבה ומספר שכבות כדי למקסם את סט הבחינה. מעבר לכך בחרתי בטרנספורמציה של סטנדרטיזציה מפני שבזמן מחקר הנושא קראתי שבנורמליזציה הטרנספורמציה מוציאה ערכים חריגים וסטנדרטיזציה לא. התוצאות שהתקבלו:

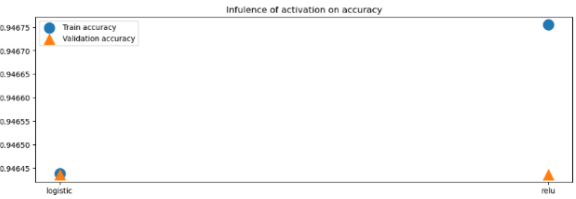
(נספח 4)

```
The best parameters are: {'activation': 'relu', 'hidden_layer_sizes': (7,), 'max_iter': 500}
```

Number	Parameters	Validation score
36	{'activation': 'relu', 'hidden_layer_sizes': (9,), 'max_iter': 500}	0.946615
32	{'activation': 'relu', 'hidden_layer_sizes': (7,), 'max_iter': 500}	0.946615
0	{'activation': 'logistic', 'hidden_layer_sizes': (1,), 'max_iter': 500}	0.946437



Hidden layer sizes – היפר-פרמטר זה מורכב ממספר הנוירונים בשכבה ומספר שכבות, ככל שאגדיל את מספר השכבות המודל יוכל להתמודד עם בעיות מסובכות יותר. כמו שנלמד בכיתה complex-2, convex-1,linear-0, אני מנסה לפתור בעייתי סיווג שאינו לינארי ולכן אני הנחתי שאצטרך שכבה אחת(גם לפי הממצעים). בנוסף קיים אפשרות שאקבל התאמת יתר עקב מספר גדול של נוירונים בשכבה, קיבלתי תוצאה של 7 נוירונים בשכבה החבויה שקטן ממספר השדות (17) ולכן אני לא חושב שקיים פה התאמת יתר. ניתן לראות ככל שמספר הנוירונים גדל כך סט האימון שואף להתאמת יתר.



Activation- פונקציית האקטיבציה הינה הפונקציה אשר עשויה להוריד את ערך ה-LOSS, מפני שפונקציית המטרה מורכבת מסכום ממושכל של הערכים מהקודקודים בשכבה הקודם כפול פונקציית האקטיבציה. קיבלת פה ממצע מעניין, לפי ה-GreadSearch פונקציית האקטיבציה relu מניע דיוק טוב יותר לסט הוולידציה אך לפי הגרף ניתן לראות שקיים הפרש קטן יותר בין הדיוק של סט האימון ולוולידציה עבור פונקציית האקטיבציה של logistic. לכן אבדוק את שניהם על סט הבחינה ורק אז אחליט באיזה פונקציה אשתמש.

2.3) רשת באמצעות הקונפיגורציה מסעיף הקודם:

```
Train accuracy relu     0.946438
dtype: float64
Test accuracy relu     0.942184
dtype: float64

Train accuracy logistic     0.946438
dtype: float64
Test accuracy logistic     0.942184
dtype: float64
```

עבור היפר-פרמטר Activation התקבל כי הפונקציה המתאימה הינה relu אך עם זאת לפי ממצאי הגרף קיים אפשרות שעדיף להשתמש בפונקציה logistic, נבדוק זאת לפי הרצת המודל על סט הבחינה ואימון. עבור Hidden_layer_size התקבל שכבה אחת חבויה עם 7 נוירונים כלומר אנו נמצאים בבעיה מסוג convex אשר מסתדר עם הנחתי מפני שהסט לא נותן לנו בעיה לינארי אך להנחתי הוא לא Complex. שכבת ה-input הינה כגודל מספר השדות 17 ושכבת ה-output הינה כגודל מספר המשתנים שניתן לקבל במשתנה המטרה כלומר 2.

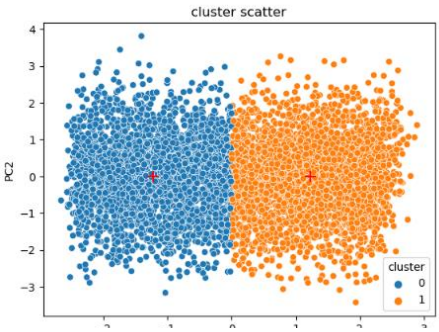
חיזויים סופיים – קיבלתי חיזויים מאוד מעניינים, זאת מפני שהחיזויים של פונקציית האקטיבציה של relu ו-logistic החזירו אחוזי דיוק זהים לחלוטין. אני מניח שזהות בין החיזויים בהתבסס על פונקיית האקטיבציה נובע מכך שיש מספר נקודות על logistic שמתנהגים זהה ל-relu, כלומר קיים קירוב לינארי למשל כמו שהקירוב של y = x לפונקציה של y = sin(x) סביב האפס שניתן

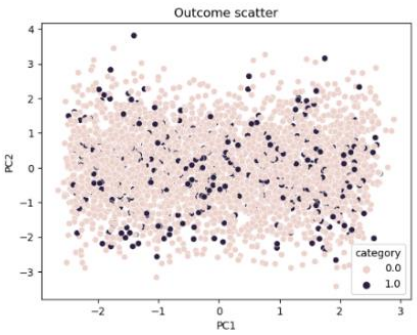
להוכיח בעזרת תור טיילור. בנוסף קיבלתי אחוזי דיוק זהים לזה של עץ ההחלטה ואני מסיק שמצב זה נוצר עקב הוצאת אחוז גבוה של שדות בזמן ניקוי הנתונים, כלומר הוצאתי הרבה מהרעש ויצרתי בסיס נתונים נקי ובכך הקלתי על המודל בזמן הלמידה.

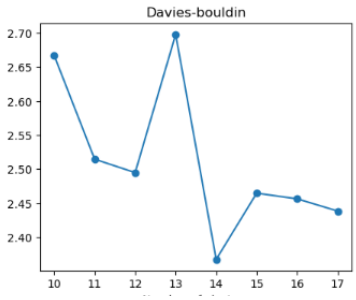
K-Means

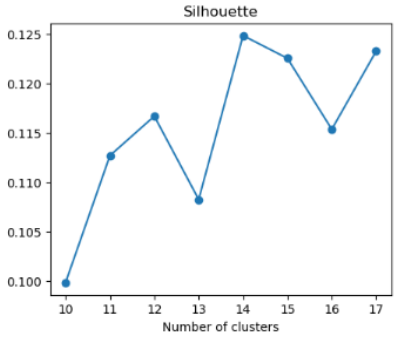
3.1/2) ערכי ברירת מחדל ב-K-Means:

```
[0.11787942 0.06396827]
0.18184768837451737
```









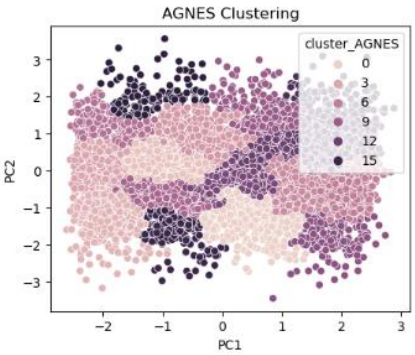
בחלק זה ביצעתי טרנספורמציה של סטנדרטיזציה עבור K-means, כדי לאפשר לכל הנתונים להיכנס בגרף עם ערכים סבירים השתמשתי ב-PCA בכדי לצמצם את כמות השדות ל-2. ראשית השתמשתי sns.scatterplot כדי לקבל תמונת מצב של החלוקה, לאחר מכן הזנתי את הנתונים למודל של K-means והתקבלו 2 גרפים. ניתן לראות שקיים הבדל משמעותי בין הגרפים, מעבר לכך התקבלו 5601 תצפיות כאשר 2578 מהם סווגו בצורה שגוי כלומר כ-46% עבור cluster scatter. ניתן לראות בעיתיות נוספת מפני שאחוז השונות המוסברת נמוך יחסית עם ערך כ-18%. שיוך התצפיות נעשה על ידי הפרדה ב-PC1 = 0.

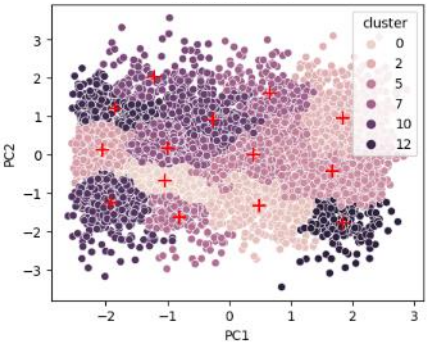
3.3) מודלים עם K שונה:

על מנת לבחון את המודלים השונים בחרתי להשתמש במדדים Davies-Bouldin ו-Sihouette. הראשון מודד את הציון עבור ממוצע כל cluster ביחס ל-cluster הדומה לו ביותר, ככל שהציון נמוך יותר ככה נדע כי ה-cluster מופרד בצורה מובהקת, לכן נרצה לקחת את ה-K עם הציון הנמוך ביותר. לעומת זאת המדד השני משמש למדידת המרחק של כל נקודה בין שני cluster ומחזיר את המרחק הקטן ביותר כלומר, נרצה למקסם את מדד זה. בחרתי לבחון K בין 10-18 בקפיצות של 1 מפני שרציתי לבחון את ההבדלים בין K קטן ל-K גדול. (נספח 5)

ניתן לראות לפי הגרפים שהערך הטוב ביותר הינו 14 בשני המדדים. נתון זה לא מפתיע מפני שלפי הגרפים מהסעיף הקודם קשה מאוד להבדיל בין התצפיות, נקבל הפרדה טובה יותר עם K גדול. ערך הנבחר הינו 14 ואין לו

קשר לסיפור מפני שקיים רק 2 אופציות לסיווג, במצב זה צריך הפרדה מוחלטת כדי ש-Kmeans יחזה בצורה איכותית. בחרתי ב-K זה מפני ששני המדדים מורים על כך.





3.4) שיטת אשכול נוספת:

בחרתי להשתמש באלגוריתם בשם Agnes Hirerarchical clustering, האלגוריתם מתחיל מכך שכל תצפית הינו cluster בפני עצמו. בשלב השני כל שני clustres אשר נמצאים המרחק מיני' מבין כולם מתאחדים, תהליך זה חוזר על עצמו עד שנהיה K clusters. לעומת זאת K-means מתחיל בכך שיש K clusters אקראיים, בכל שלב מצורף לכל cluster תצפיות אשר חורגים מהטווח של ה-cluster המקורי שלהם, האלגוריתם ממשיך עד שהוא מסיים את מספר האיטרציות שהוגדר לו. ניתן לראות שקיים הבדל מזערי בין ה-clusters של שני האלגוריתמים. היתרון הוא שלא צריך לדעת מראש כמה clusters יש ובנוסף ניתן לראות את המרחק בין כל cluster בעזרת גרף Dendorgram. במודל K-means צריך להריץ פונקציה כדי לדעת את מספר ה-cluster האופטימלי. קיים יתרון ל-kmeans מפני שזמן הריצה שלו קטנה יותר מאשר זו של Agnes.(נספח 6)

Comparison between models

4)השווה בין המודלים:

המשימה שנתנה לנו היה לסווג תצפיות בעזרת NN עצי החלטה ו-Kmeans, שני האלגוריתמים הראשונים עשויים להשתפר לאחר אימון וכיוונון היפר-פרמטרים ובך ללמוד את הקשרים ונהיים מדויקים יותר, לעומת זאת K-means עבור משימה זאת, אשר ידוע מספרים clusters וגם ערך השונות המוסברת לא משתפר בהרבה. על מנת למדוד את דיוק של K-means בחרתי לעשות הערכה גסה, סיווגתי כל cluster על ידי חישוב ממוצע. אם הממוצע היה גדול מ-0.5 הוא קיבל סיווג של 1 אחרת 0. לבסוף חישבתי את אחוזי הדיוק ומשם חילצי את ה-accuracy. לכן אני אשווה בין המודלים בעזרת מדד זה.

הדיוק הכי שמאלי הינו של עץ ההחלטה, העצמאי של מערכת נוירונים והימני של K-means עבור K = 14. בדיוק של K-means ניתן לראות שהדיוק של סט הבחינה גדול מזה של הסט אימון, זה ככל הנראה בגלל שסט הבחינה קטן יותר ולכן יותר קל לסווג וגם קיים underfitting עקב K קטן מידי, כלומר קיים אינדיקציה ישירה שצריך כמות cluster גדול יותר . מעבר לכך משהו משונה ביותר קרה, דיוק של עץ ההחלטה קיבל דיוק זהה למערכת נוירונים וגם ה-confusion matrix זהה, זה מקרה חריג ואף נשמע לא הגיוני, לאחר הרבה מחקר והתייעצות עם מספר אנשים הגעתי למסקנה שזה קרה בגלל סט אימון קטן יחסית ללא רעש ולא מאוזן. אני הייתי ממליץ על עצי החלטה או NN למשימה הזאת עקב הנאמר לעיל.

```
Train accuracy    0.946438
dtype: float64
Test accuracy     0.942184
dtype: float64
```

```
Train accuracy relu    0.946438
dtype: float64
Test accuracy relu    0.942184
dtype: float64
```

	Train accuracy
0	0.611429

	Test accuracy
0	0.860913

The select model

5)המודל הנבחר:

המודל הנבחר הינו DT. תחילה התחלתי לעבוד על הדאטה מתוך חלק א' אך לאחר הרצת מספר אלגוריתמים התחלתי לחשוש שההתאמות שעשיתי בחלק א' היו שגויות כלומר יצרתי סט אימון אחר שלא תואם את בעיית הסיווג שניתנה. החלטתי לנכות את סט האימון לחלוטין מנתונים לא הגיוניים ו-NA אך לאחר הרצת האלגוריתמים שמתי לב שאני מלמד מודל לא גמיש שלא יודע להתמודד עם רעש. לבסוף בעזרת R החלטתי לשנות ערכי NA עם ממוצעת השדה עבור משתנים רציפים ובשדות בינארים הגרלתי 0,1 בעזרת התפלגות יוניפורמי, אם ההסתברות לקבל 1 בעמודה קטן שווה למספר שהוגרל אחליף באחד אחרת אפס. קיים משתנים בעייתיים למשל שטח מוסך, עלית גג ומרתף שגדולים משטח הבית או מספר חדרי אירוח גדול ממספר החדרים בבית אבל, לאחר הרבה התחשבות במבנה סט הנתונים והשפעת השדות על סיווג משתנה המטרה הגעתי למסקנה שהשפעתם על משתנה המטרה הוא זניח ולכן לא אשנה אותם.

מעבר לכך בחרתי לא להוריד שדות כדי שהמודל כן ידע להתמודד עם שדות לא רלוונטיים לסיווג משתנה המטרה.(נספח 7)

סט האימון שיצרתי בחלק א':

hasYard	hasPool	cityCode	made	isNewBui	hasGuestF	category
0	0	2	1992	1	1	0
0	0	5	2012	1	1	0
0	1	1	2020	0	1	0
0	0	4	2005	0	1	0
1	1	3	2020	1	1	1

לאחר החלטות אלו ביצתי נרמול על סט האימון בעזרת פונקציית StandardScaler, זאת מפני שאלגוריתם של מערכת נוירונים ועצי החלטות מגיב בצורה מיטבית כאשר הנתונים בסט האימון דומים בסדר גודל, נניח כל הנתונים בטווח של סדר גודל $10^{-1} - 10^{2}$. לאחר מכן שמתי לב שסט הנתונים אינו מאוזן כאשר קיים 12% של בתי יוקרה ו-88% של בתים בסיסים, לכן השתמשתי בחבילה של SMOTE כדי לאזן את סט האימון וסט הוולידציה אשר משתמש בשיטה של K Nearest Neighbors שמטרתו למצוא את ה-K שכנים הכי קרובים וליצר תצפית נוספת במיקום הממוצע של המפגש שלהם וכך יצרתי סט מאוזן. לאחר מכן הרצתי GreadSearch כדי למצוא את ההיפר-פרמטרים של עץ ההחלטה שימקסמו את הדיוק של סט האימון, אך נוצר לי מצב של התאמת יתר על סט האימון ולכן שיחקתי ידני עם ההיפר-פרמטרים. למשל בעץ החלטה קיבלתי מקס עומק של 16, ראשית שינית את זה למקס עומק של 3 וקיבלתי underfitting מפני שדיוק סט הוולידציה היה גבוה מסט האימון, לבסוף שיניתי את עומק המקס ל-6 וקיבלתי פאר מזערי בין דיוק סט הוולידציה לסט אימון. מעבר לכך עבדתי גם על מערכת הנוירונים וגם שם קיבלתי התאמת יתר לאחר שהרצתי GreadSearch זיהיתי זאת מפני שהיה פאר גדול בין דיוק סט האימון (היה גבוהה מאוד כלומר שואף ל-1) לוולידציה, הנחתי שזה נוצר עקב השכבות החבויות מפני שהתקבל שני שכבות כך שיש 16 נוירונים בכל שכבה. לכן החלטתי לשחק עם הערכים, ראשית הכנסתי שיכבה אחד חבויה עם 7 נוירונים כמו שהתקבל בפתרון בסעיף 2, נוצר פאר קטן בין דיוק סט הוולידציה לסט האימון אבל הנחתי שניתן לשפר את התוצאה ולצמצם את הפאר לכן הכנסתי שכבה נוספת עם נוירון יחד. לבסוף הצלחתי לצמצם את הפאר וקיבלתי שני שכבות כך שקיים 7 נוירונים בשכבה הראשונה ונוירון יחיד בשכבה השנייה. לא התעסקתי עם K-mens מפני שהנחתי שאלגוריתם זה לא מיטבי לבעיה שלי עקב פיזור גבוה אצל משתנה המטרה אשר מקבל ערך של בית יוקרה. אני מודע לכך שהדיוק ירד יחסית לסעיפים 1/2 אך עם

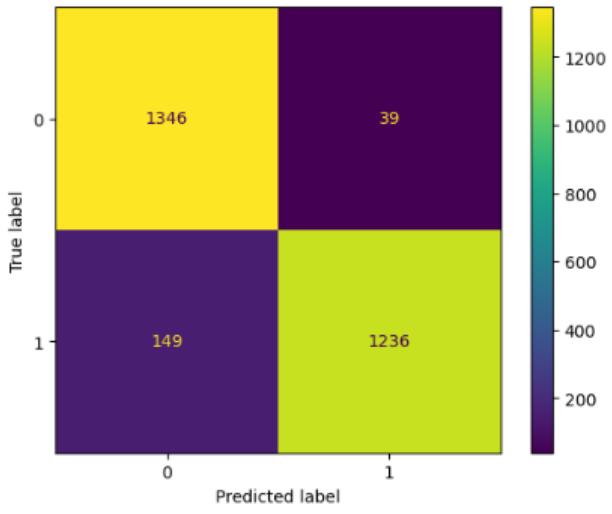
זאת אני מאמין שהמודלים שיצרתי גמישים יותר ויתמודדו בצורה טובה יותר עם סט בחינה אשר שונה מסט האימון ולכן אני אבחר במודל אשר יצר את הדיוק המירבי עבור סט הבחינה, כלומר המודל עץ ההחלטה.(נספח NN) (נספח DT)

<u>מדד הדיוק עבור המודל הנבחר וגם מטריצת המבוכה:</u>

```
best_tree = DecisionTreeClassifier(max_depth=6, criterion='gini', ccp_alpha = 0 ,  random_state=42)
```

על סמך המטריצה המבוכה ניתן לראות כי קיים סיכוי פי 4 שהמודל יחזה 0 ובפועל התצפית הייתה אמורה לקבל סיווג של 1 לעומת ההפך, כלומר קיים סיכוי פי 4 לקבל False Negative לעומת הסיכוי לקבל False Positive, ולכן קיים הטיה. מתוך הנאמר לעיל ניתן להסיק שאחד החוזקות של המודל הוא לסווג בתים בסיסים (דיוק של 97%), אך לא הייתי אומר שחולשה של המודל הוא סיווג של בתי יוקרה מכיוון שמודל חוזה בתי יוקרה בדיוק של 89%. נניח שהמודל הוא של חברת נדלן שמוכר בתים בפריז, ניתן לומר שהמודל הזה לטובות הלקוח מפני שקיים סיכוי של 11% שהלקוח יקנה בית יוקרה בטווח מחיר של בית בסיסי, לעומת זאת קיים סיכוי של 3% שהלקוח יקנה בית בסיסי במחיר של בית יוקרה. המודל שיצרתי גמיש ומוכן לחזות כל בסיס נתונים ללא תלות במודל האימון, אך עם זאת עדיין קיים אפשרות כי שכפול הנתונים יצרו מין סוג של התאמת יתר מפני שהאלגוריתם שבחרתי להרחיב איתו את סט הנתונים עלול ליצור מצב כזה עקב תהליך האלגוריתם וגם האיזון התבצע לאחר ההפרד של סט האימון לסט אימון ובחינה.

```
0    0.93213
dtype: float64
Confusion matrix <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x00000127C5E57BE0>
Classification report              precision    recall  f1-score   support

           0       0.90      0.97      0.93      1385
           1       0.97      0.89      0.93      1385

    accuracy                           0.93      2770
   macro avg       0.93      0.93      0.93      2770
weighted avg       0.93      0.93      0.93      2770
```



נספחים

6) נספחים:

1) **תיקון סט הנתונים:**

ניקוי גס של הנתונים:

```
In [6]: #drop unwanted col
        del df['Unnamed: 0.1']
        del df['Unnamed: 0']
        df.columns

Out[6]: Index(['squareMeters', 'numberOfRooms', 'hasYard', 'hasPool', 'floors',
               'cityCode', 'cityPartRange', 'numPrevOwners', 'made', 'isNewBuilt',
               'hasStormProtector', 'basement', 'attic', 'garage', 'hasStorageRoom',
               'hasGuestRoom', 'price', 'category'],
              dtype='object')

In [8]: #I will drop null values because they are less then 5%
        df=df.dropna()
        df.isnull().sum()
        df.shape

Out[8]: (7932, 18)
```

remove/replace imposible values

```
[11]: # I droped more then 5% because i will resample and add more data later
      df = df[df['cityCode'] >= 0]
      df = df[df['made'] < 2023]
      df = df[df['price'] > 0]
      df = df.replace(df['hasGuestRoom'] < df['numberOfRooms'],df['numberOfRooms'].mean())
      df = df.replace(df['basement'] < df['squareMeters'],df['basement'].mean())
      df = df.replace(df['attic'] < df['squareMeters'],df['attic'].mean())
      df = df.replace(df['garage'] < df['squareMeters'],df['garage'].mean())
      df = df[df['hasGuestRoom'] < df['numberOfRooms']]
      df = df[df['basement'] < df['squareMeters']]
      df = df[df['attic'] < df['squareMeters']]
      df = df[df['garage'] < df['squareMeters']]
      df.shape

[11]: (7002, 18)
```

איזון נתונים:

Balance data with imblearn_samplig (random data based X_train_scales and y_train)

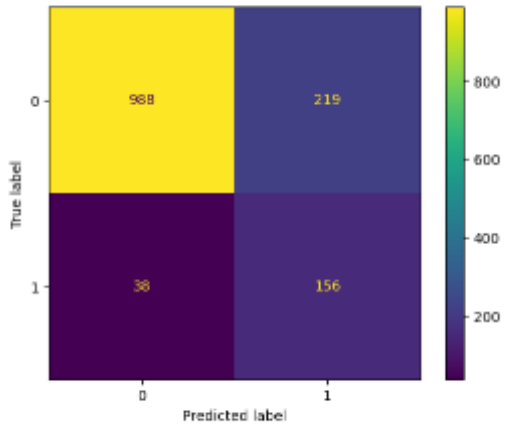
```
In [16]: ####_____from imblearn.over_sampling import RandomOverSampler
         ####_____from collections import Counter
         from imblearn.over_sampling import RandomOverSampler
         from collections import Counter
         ros= RandomOverSampler(random_state=0)
         X_resampled, y_resampled = ros.fit_resample(X_train_scaled, y_train)
         print("Original dataset shape: ", Counter(y_train))
         print("Original dataset shape: ", Counter(y_resampled))
         print(sorted(Counter(y_resampled).items()))

         Original dataset shape:  Counter({0: 4833, 1: 768})
         Original dataset shape:  Counter({0: 4833, 1: 4833})
         [(0, 4833), (1, 4833)]
```

רגרסיה לוגיסטית:

check if new dataFrame is efficient

In [19]:
```
#_________________from sklearn.linear_model import LogisticRegression
#_________________from sklearn.metrics import plot_confusion_matrix, classification_report
model = LogisticRegression()
model.fit(X_resampled, y_resampled)
print("Training accuracy: ", model.score(X_resampled, y_resampled))

y_pred = model.predict(X_test_scaled)
cm = confusion_matrix(y_test, y_pred, labels=model.classes_)
con =  ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=model.classes_)
print("Confusion matrix",con)
print("Classification report", classification_report(y_test,y_pred))
con.plot()
plt.show()
```

```
Training accuracy:  0.8055038278501966
Confusion matrix <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x0000021275432FD0>
Classification report              precision    recall  f1-score   support

           0       0.96      0.82      0.88      1207
           1       0.42      0.80      0.55       194

    accuracy                           0.82      1401
   macro avg       0.69      0.81      0.72      1401
weighted avg       0.89      0.82      0.84      1401
```



2)עצי החלטה:

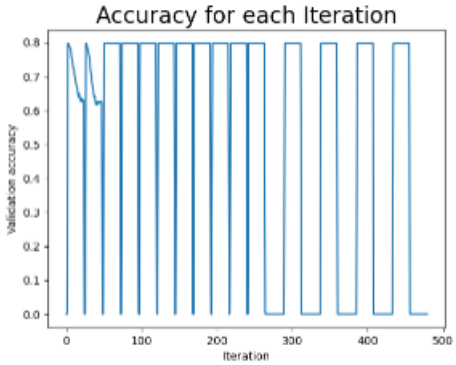
:greadSearch

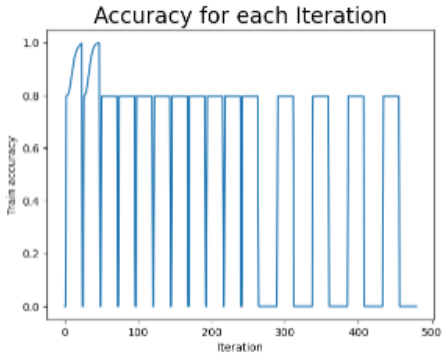
GridSearchCV

```
param_grid = {'max_depth': np.arange(1, 25, 1),
              'criterion': ['entropy', 'gini'],
              'ccp_alpha': np.arange(0, 0.1, 0.01)
             }

# use f1 because target variable is 0 or 1
grid_search = GridSearchCV(estimator=DecisionTreeClassifier(random_state=42),
                           param_grid=param_grid,
                           refit=True,
                           cv=KF, verbose=3 ,scoring='f1', return_train_score=True )

grid_search.fit(X_train_scaled, y_train)
best_model = grid_search.best_estimator_
print(grid_search.best_params_, '\n')
print(best_model.get_params(), '\n')
Results = pd.DataFrame(grid_search.cv_results_)
preds = best_model.predict(X_test_scaled)
print("Test accuracy: ", round(accuracy_score(y_test, preds), 3))
```

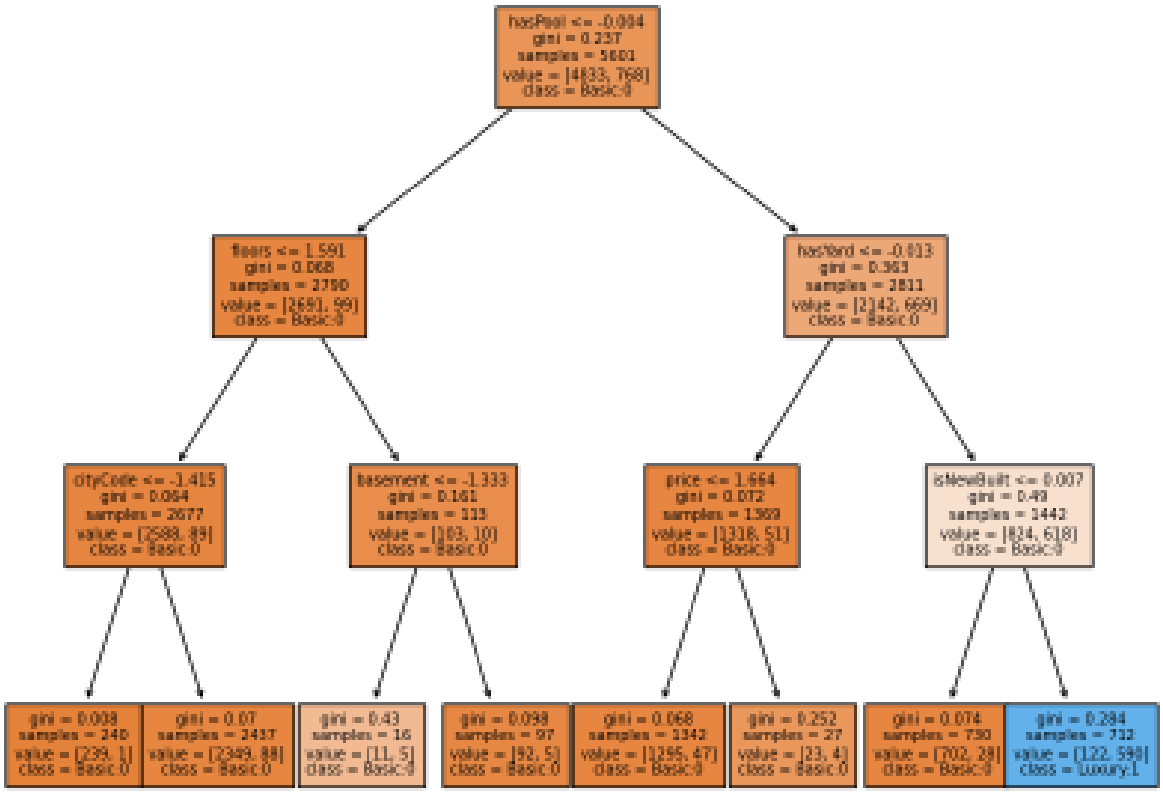
```
Fitting 10 folds for each of 480 candidates, totalling 4800 fits
[CV 1/10] END ccp_alpha=0.0, criterion=entropy, max_depth=1;, score=(train=0.000, test=0.000) total time=   0.0s
[CV 2/10] END ccp_alpha=0.0, criterion=entropy, max_depth=1;, score=(train=0.000, test=0.000) total time=   0.0s
[CV 3/10] END ccp_alpha=0.0, criterion=entropy, max_depth=1;, score=(train=0.000, test=0.000) total time=   0.0s
[CV 4/10] END ccp_alpha=0.0, criterion=entropy, max_depth=1;, score=(train=0.000, test=0.000) total time=   0.0s
[CV 5/10] END ccp_alpha=0.0, criterion=entropy, max_depth=1;, score=(train=0.000, test=0.000) total time=   0.0s
[CV 6/10] END ccp_alpha=0.0, criterion=entropy, max_depth=1;, score=(train=0.000, test=0.000) total time=   0.0s
[CV 7/10] END ccp_alpha=0.0, criterion=entropy, max_depth=1;, score=(train=0.000, test=0.000) total time=   0.0s
[CV 8/10] END ccp_alpha=0.0, criterion=entropy, max_depth=1;, score=(train=0.000, test=0.000) total time=   0.0s
[CV 9/10] END ccp_alpha=0.0, criterion=entropy, max_depth=1;, score=(train=0.000, test=0.000) total time=   0.0s
[CV 10/10] END ccp_alpha=0.0, criterion=entropy, max_depth=1;, score=(train=0.000, test=0.000) total time=   0.0s
[CV 1/10] END ccp_alpha=0.0, criterion=entropy, max_depth=2;, score=(train=0.000, test=0.000) total time=   0.0s
[CV 2/10] END ccp_alpha=0.0, criterion=entropy, max_depth=2;, score=(train=0.000, test=0.000) total time=   0.0s
[CV 3/10] END ccp_alpha=0.0, criterion=entropy, max_depth=2;, score=(train=0.000, test=0.000) total time=   0.0s
[CV 4/10] END ccp_alpha=0.0, criterion=entropy, max_depth=2;, score=(train=0.000, test=0.000) total time=   0.0s
[CV 5/10] END ccp_alpha=0.0, criterion=entropy, max_depth=2;, score=(train=0.000, test=0.000) total time=   0.0s
[CV 6/10] END ccp_alpha=0.0, criterion=entropy, max_depth=2;, score=(train=0.000, test=0.000) total time=   0.0s
[CV 7/10] END ccp_alpha=0.0, criterion=entropy, max_depth=2;, score=(train=0.000, test=0.000) total time=   0.0s
[CV 8/10] END ccp_alpha=0.0, criterion=entropy, max_depth=2;, score=(train=0.000, test=0.000) total time=   0.0s
[CV 9/10] END ccp_alpha=0.0, criterion=entropy, max_depth=2;, score=(train=0.000, test=0.000) total time=   0.0s
[CV 10/10] END ccp_alpha=0.0, criterion=entropy, max_depth=2;, score=(train=0.000, test=0.000) total time=   0.0s
[CV 1/10] END ccp_alpha=0.0, criterion=entropy, max_depth=3;, score=(train=0.794, test=0.843) total time=   0.0s
[CV 2/10] END ccp_alpha=0.0, criterion=entropy, max_depth=3;, score=(train=0.798, test=0.812) total time=   0.0s
[CV 3/10] END ccp_alpha=0.0, criterion=entropy, max_depth=3;, score=(train=0.801, test=0.770) total time=   0.0s
[CV 4/10] END ccp_alpha=0.0, criterion=entropy, max_depth=3;, score=(train=0.798, test=0.812) total time=   0.0s
[CV 5/10] END ccp_alpha=0.0, criterion=entropy, max_depth=3;, score=(train=0.798, test=0.787) total time=   0.0s
[CV 6/10] END ccp_alpha=0.0, criterion=entropy, max_depth=3;, score=(train=0.798, test=0.793) total time=   0.0s
[CV 7/10] END ccp_alpha=0.0, criterion=entropy, max_depth=3;, score=(train=0.798, test=0.802) total time=   0.0s
[CV 8/10] END ccp_alpha=0.0, criterion=entropy, max_depth=3;, score=(train=0.796, test=0.809) total time=   0.0s
[CV 9/10] END ccp_alpha=0.0, criterion=entropy, max_depth=3;, score=(train=0.805, test=0.731) total time=   0.0s
[CV 10/10] END ccp_alpha=0.0, criterion=entropy, max_depth=3;, score=(train=0.798, test=0.809) total time=   0.0s
[CV 1/10] END ccp_alpha=0.0, criterion=entropy, max_depth=4;, score=(train=0.795, test=0.838) total time=   0.0s
[CV 2/10] END ccp_alpha=0.0, criterion=entropy, max_depth=4;, score=(train=0.798, test=0.812) total time=   0.0s
[CV 3/10] END ccp_alpha=0.0, criterion=entropy, max_depth=4;, score=(train=0.802, test=0.770) total time=   0.0s
[CV 4/10] END ccp_alpha=0.0, criterion=entropy, max_depth=4;, score=(train=0.799, test=0.812) total time=   0.0s
[CV 5/10] END ccp_alpha=0.0, criterion=entropy, max_depth=4;, score=(train=0.799, test=0.787) total time=   0.0s
[CV 6/10] END ccp_alpha=0.0, criterion=entropy, max_depth=4;, score=(train=0.799, test=0.793) total time=   0.0s
[CV 7/10] END ccp_alpha=0.0, criterion=entropy, max_depth=4;, score=(train=0.799, test=0.802) total time=   0.0s
```

דיוק לכל איטרציה:

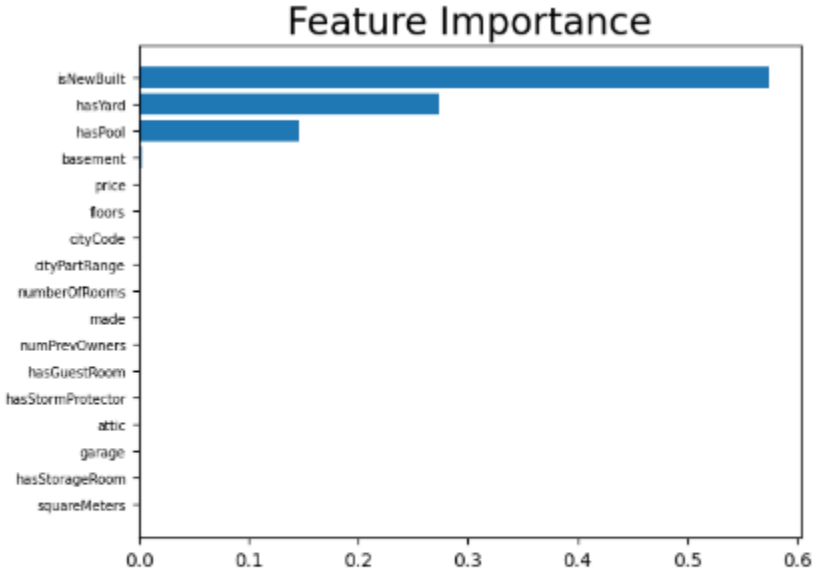




עץ ההחלטה שנבחר:



חשיבות כל שדה:

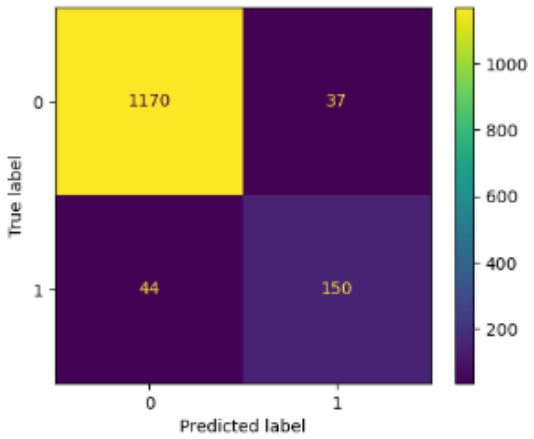


	a
isNewBuilt	0.574418
hasYard	0.274396
hasPool	0.146522
basement	0.002386
price	0.000864
floors	0.000844
cityCode	0.000569

:Confusion matrix

```
In [194]: model_best_tree = DecisionTreeClassifier(max_depth=3, criterion='gini', ccp_alpha = 0 ,  random_state=42)
          model_best_tree.fit(X_train_scaled, y_train)
          acc_test = accuracy_score(y_test, model_best_tree.predict(X_test_scaled))

          cm = confusion_matrix(y_test, model_best_tree.predict(X_test_scaled), labels=model_best_tree.classes_)
          con =  ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=model_best_tree.classes_)
          print("Confusion matrix",con)
          print("Classification report", classification_report(y_test,model_best_tree.predict(X_test_scaled)))
          con.plot()
          plt.show()
```

```
Confusion matrix <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x0000029A019AAA00>
Classification report               precision    recall  f1-score   support

           0       0.96      0.97      0.97      1207
           1       0.80      0.77      0.79       194

    accuracy                           0.94      1401
   macro avg       0.88      0.87      0.88      1401
weighted avg       0.94      0.94      0.94      1401
```



3)**מערכת נוירונים:**

:GreadSearch

```
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
The best parameters are: {'activation': 'relu', 'hidden_layer_sizes': (7,), 'max_iter': 500}
```

Number	Parameters	Validation score
36	{'activation': 'relu', 'hidden_layer_sizes': (9,), 'max_iter': 500}	0.946615
32	{'activation': 'relu', 'hidden_layer_sizes': (7,), 'max_iter': 500}	0.946615
0	{'activation': 'logistic', 'hidden_layer_sizes': (1,), 'max_iter': 500}	0.946437
40	{'activation': 'relu', 'hidden_layer_sizes': (11,), 'max_iter': 500}	0.946437
34	{'activation': 'relu', 'hidden_layer_sizes': (8,), 'max_iter': 500}	0.946437
31	{'activation': 'relu', 'hidden_layer_sizes': (4, 3), 'max_iter': 500}	0.946437

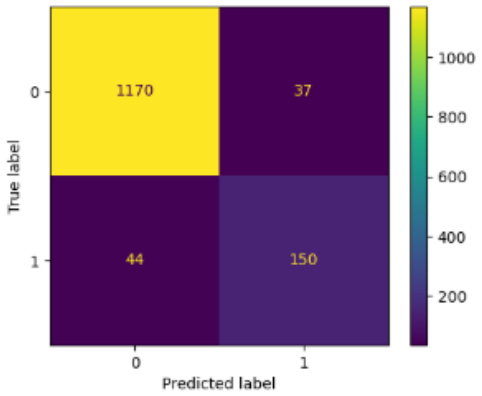
GreadSearch עם פרמטרים נוספים:

```
In [13]: TwoLayers = [(1,2),(2,),(3,),(4,),(5,),(6,),(7,),(8,),(9,),(10,),(11,)]
for x in np.arange(1,21,5):
    for y in np.arange(1,21,5):
        TwoLayers.insert(-1, (x,y))

param_grid = {
              'hidden_layer_sizes': TwoLayers,
              'activation': ['relu','logistic'],
               'learning_rate':['constant', 'adaptive']

              }

grid_search = GridSearchCV(estimator = MLPClassifier(random_state=42, verbose=True, max_iter=500), param_grid=(param_grid),
                           refit=True, cv=10, return_train_score=True)
grid_search.fit(X_train_N, y_train)
Results = pd.DataFrame(grid_search.cv_results_)
print('The best parameters are:', grid_search.best_params_)
y_val = Results['mean_test_score']
y_train = Results['mean_train_score']
results_grid_search1 = pd.DataFrame(Results).sort_values('rank_test_score')[['params', 'mean_test_score']]
results_grid_search2 = pd.DataFrame(Results).sort_values('mean_train_score', ascending=False)[
    ['params', 'mean_train_score']]
headers_val = ["Number", "Parameters", "Validation score"]
headers_train = ["Number", "Parameters", "Train score"]
print(tabulate(results_grid_search1, headers=headers_val, tablefmt="grid"))
print(tabulate(results_grid_search2, headers=headers_train, tablefmt="grid"))
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
The best parameters are: {'activation': 'relu', 'hidden_layer_sizes': (7,), 'learning_rate': 'constant'}
+----------+-------------------------------------------------------------------------------------------+---------------------+
|   Number | Parameters                                                                                |   Validation score |
+==========+===========================================================================================+=====================+
|       13 | {'activation': 'relu', 'hidden_layer_sizes': (7,), 'learning_rate': 'adaptive'}          |           0.946793 |
+----------+-------------------------------------------------------------------------------------------+---------------------+
|       12 | {'activation': 'relu', 'hidden_layer_sizes': (7,), 'learning_rate': 'constant'}          |           0.946793 |
+----------+-------------------------------------------------------------------------------------------+---------------------+
|       53 | {'activation': 'relu', 'hidden_layer_sizes': (11,), 'learning_rate': 'adaptive'}         |           0.946614 |
+----------+-------------------------------------------------------------------------------------------+---------------------+
|       25 | {'activation': 'relu', 'hidden_layer_sizes': (1, 11), 'learning_rate': 'adaptive'}       |           0.946614 |
+----------+-------------------------------------------------------------------------------------------+---------------------+
|       17 | {'activation': 'relu', 'hidden_layer_sizes': (9,), 'learning_rate': 'adaptive'}          |           0.946614 |
+----------+-------------------------------------------------------------------------------------------+---------------------+
|       16 | {'activation': 'relu', 'hidden_layer_sizes': (9,), 'learning_rate': 'constant'}          |           0.946614 |
+----------+-------------------------------------------------------------------------------------------+---------------------+
```

Confusion matrix:

```
model_best_MLP = MLPClassifier(max_iter=500, activation='relu', hidden_layer_sizes=(7, ),  random_state=42)
model_best_MLP.fit(X_train_N, y_train)
acc_test = accuracy_score(y_test, model_best_MLP.predict(X_test_N))

cm = confusion_matrix(y_test, model_best_MLP.predict(X_test_N), labels=model_best_MLP.classes_)
con =  ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=model_best_MLP.classes_)
print("Confusion matrix",con)
print("Classification report", classification_report(y_test,model_best_tree.predict(X_test_N)))
con.plot()
plt.show()
```

```
Confusion matrix <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x00000299AAB5B460>
Classification report              precision    recall  f1-score   support

           0       0.96      0.97      0.97      1207
           1       0.80      0.77      0.79       194

    accuracy                           0.94      1401
   macro avg       0.88      0.87      0.88      1401
weighted avg       0.94      0.94      0.94      1401
```



4) **K-means:**

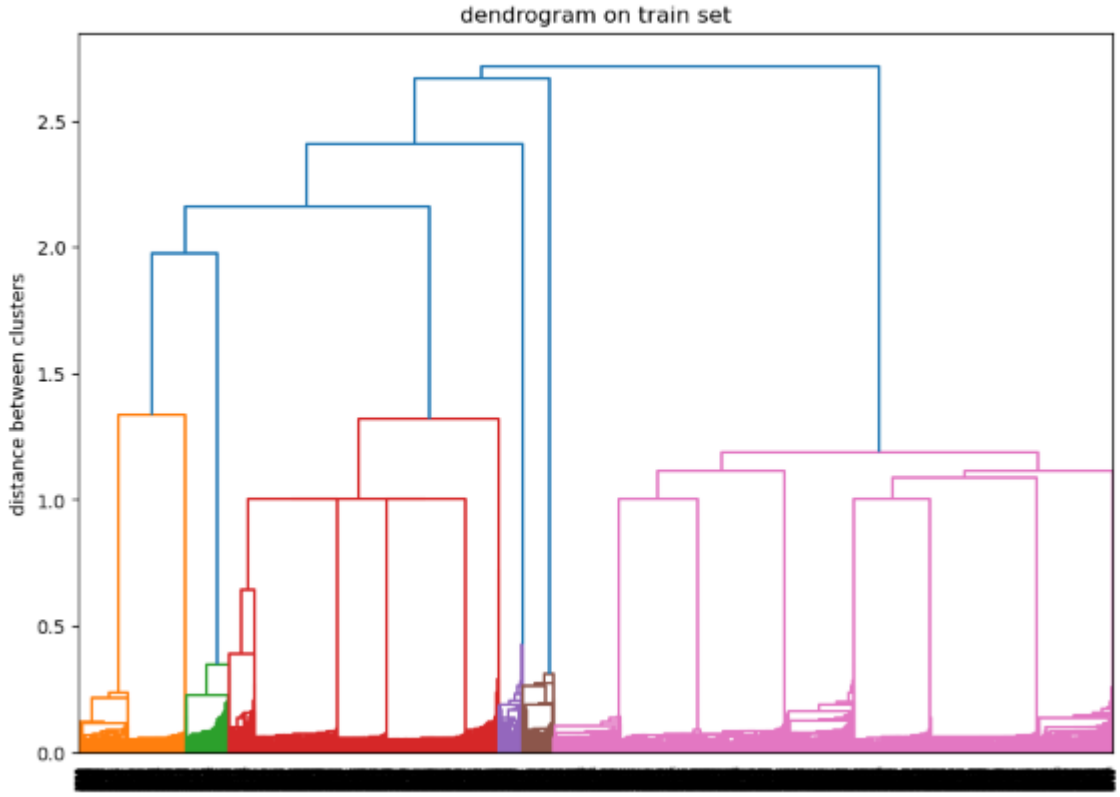
ציוני Davies-bouldin:

```
For k= 10 , Davies-Bouldin score is: 2.667001663211578
For k= 11 , Davies-Bouldin score is: 2.5145418365362633
For k= 12 , Davies-Bouldin score is: 2.494869954926493
For k= 13 , Davies-Bouldin score is: 2.697757176187323
For k= 14 , Davies-Bouldin score is: 2.3674261446892233
For k= 15 , Davies-Bouldin score is: 2.464679414924432
For k= 16 , Davies-Bouldin score is: 2.456366859369127
For k= 17 , Davies-Bouldin score is: 2.4384051917563063
```

ציוני Silhoute:

```
For k= 10 , Silhouette score is: 0.09986512799085101
For k= 11 , Silhouette score is: 0.11269587670340232
For k= 12 , Silhouette score is: 0.11667615791958644
For k= 13 , Silhouette score is: 0.10825964078973992
For k= 14 , Silhouette score is: 0.12481948914923816
For k= 15 , Silhouette score is: 0.12252435436677309
For k= 16 , Silhouette score is: 0.11534743944860662
For k= 17 , Silhouette score is: 0.12324781797277538
```

dendrogram on train set:



5)**המודל הנבחר:**

ניקוי תקין של הנתונים בעזרת R:

```r
#--------------------switch null values, part of the values need to be full numbers ----------------
clean_DataTable$numberOfRooms[is.na(clean_DataTable$numberOfRooms)]<-as.integer(mean(clean_DataTable$numberOfRooms,na.rm = TRUE))

prob_hasYard <- runif(1,0,1)

clean_DataTable$hasYard[is.na(clean_DataTable$hasYard)]<-
  if(mean(clean_DataTable$hasYard,na.rm = TRUE) <= prob_hasYard){
    1
  }else{
  0
  }

prob_hasPool <- runif(1,0,1)

clean_DataTable$hasPool[is.na(clean_DataTable$hasPool)]<-
  if(mean(clean_DataTable$hasPool,na.rm = TRUE) <= prob_hasPool){
  1
}else{
  0}

clean_DataTable$floors[is.na(clean_DataTable$floors)]<-as.integer(mean(clean_DataTable$floors,na.rm = TRUE))

clean_DataTable$cityCode[is.na(clean_DataTable$cityCode)]<-as.integer(mean(clean_DataTable$cityCode,na.rm = TRUE))

clean_DataTable$cityPartRange[is.na(clean_DataTable$cityPartRange)]<-as.integer(mean(clean_DataTable$cityPartRange,na.rm = TRUE))

clean_DataTable$numPrevOwners[is.na(clean_DataTable$numPrevOwners)]<-as.integer(mean(clean_DataTable$numPrevOwners,na.rm = TRUE))

clean_DataTable$made[is.na(clean_DataTable$made)]<-as.integer(mean(clean_DataTable$made,na.rm = TRUE))


prob_isNewBuilt <- runif(1,0,1)

clean_DataTable$isNewBuilt[is.na(clean_DataTable$isNewBuilt)]<-
  if(mean(clean_DataTable$isNewBuilt,na.rm = TRUE) <= prob_isNewBuilt){
    1
  }else{
    0}

prob_hasStormProtector <- runif(1,0,1)

clean_DataTable$hasStormProtector[is.na(clean_DataTable$hasStormProtector)]<-
  if(mean(clean_DataTable$hasStormProtector,na.rm = TRUE) <= prob_hasStormProtector){
    1
  }else{
    0}


clean_DataTable$basement[is.na(clean_DataTable$basement)]<-mean(clean_DataTable$basement,na.rm = TRUE)

clean_DataTable$attic[is.na(clean_DataTable$attic)]<-mean(clean_DataTable$attic,na.rm = TRUE)

clean_DataTable$garage[is.na(clean_DataTable$garage)]<-mean(clean_DataTable$garage,na.rm = TRUE)

prob_hasStorageRoom <- runif(1,0,1)

clean_DataTable$hasStorageRoom[is.na(clean_DataTable$hasStorageRoom)]<-
  if(mean(clean_DataTable$hasStorageRoom,na.rm = TRUE) <= prob_hasStorageRoom){
    1
  }else{
    0}

clean_DataTable$price[is.na(clean_DataTable$price)]<-mean(clean_DataTable$price,na.rm = TRUE)

clean_DataTable$squareMeters[is.na(clean_DataTable$squareMeters)]<-mean(clean_DataTable$squareMeters,na.rm = TRUE)
```

המשך ניקוי בערת פיתון:

Finale Cleanup and Modle

```
df = pd.read_csv("Xy_train_new.csv")

df.sample(10)

del df['Unnamed..0']
df.columns
#change data type to int64 and switch yes:1 no:0
df['squareMeters'] = df['squareMeters'].astype(np.int64)
df['numberOfRooms'] = df['numberOfRooms'].astype(np.int64)
df['hasYard'] = df['hasYard'].astype(np.int64)
df['hasPool'] = df['hasPool'].astype(np.int64)
df['floors'] = df['floors'].astype(np.int64)
df['cityCode'] = df['cityCode'].astype(np.int64)
df['cityPartRange'] = df['cityPartRange'].astype(np.int64)
df['numPrevOwners'] = df['numPrevOwners'].astype(np.int64)
df['made'] = df['made'].astype(np.int64)
df['isNewBuilt'] = df['isNewBuilt'].replace({'no':0, 'yes':1})
df['isNewBuilt'] = df['isNewBuilt'].astype(np.int64)
df['hasStormProtector'] = df['hasStormProtector'].replace({'no':0, 'yes':1})
df['hasStormProtector'] = df['hasStormProtector'].astype(np.int64)
df['basement'] = df['basement'].astype(np.int64)
df['attic'] = df['attic'].astype(np.int64)
df['garage'] = df['garage'].astype(np.int64)
df['hasStorageRoom'] = df['hasStorageRoom'].astype(np.int64)
df['hasGuestRoom'] = df['hasGuestRoom'].astype(np.int64)
df['price'] = df['price'].astype(np.int64)


# I droped more then 5% because i will resample and add more data later
df = df.replace([df['cityCode'] < 0],df['cityCode'].mean())
df = df.replace([df['made'] < 2023],df['made'].mean())
df = df.replace([df['price'] < 0],0)
df = df.replace(df['hasGuestRoom'] < df['numberOfRooms'],df['numberOfRooms'].mean())
df = df.replace(df['basement'] < df['squareMeters'],df['basement'].mean())
df = df.replace(df['attic'] < df['squareMeters'],df['attic'].mean())
df = df.replace(df['garage'] < df['squareMeters'],df['garage'].mean())
```

פיצול נרמול ואיזון נתונים:

```
X = df.drop(['category'],axis=1)
y = df['category']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=True, random_state=123)
print("X_train shape: ", X_train.shape)
print("y_train shape: ", y_train.shape)
print("X_test shape: ", X_test.shape)
print("y_test shape: ", y_test.shape)
scaler = StandardScaler()
X_train_scaled = pd.DataFrame(scaler.fit_transform(X_train),columns=X_train.columns)
X_test_scaled = pd.DataFrame(scaler.transform(X_test),columns=X_train.columns)
X_train_scaled
```

```
from imblearn.over_sampling import SMOTE
from collections import Counter
smote= SMOTE()
X_resampled, y_resampled = smote.fit_resample(X_train_scaled, y_train)
print("Original dataset shape: ", Counter(y_train))
print("Original dataset shape: ", Counter(y_resampled))
print(sorted(Counter(y_resampled).items()))
```

:Nerual Networks Remake

:GreadSearch

GreadSearchCV Remake

In [126]:
```
TwoLayers = [(1,2),(2,),(3,),(4,),(5,),(6,),(7,),(8,),(9,),(10,),(11,)]
for x in np.arange(1,21,5):
    for y in np.arange(1,21,5):
        TwoLayers.insert(-1, (x,y))

param_grid = {
               'hidden_layer_sizes': TwoLayers,
               'activation': ['relu','logistic'],
                'learning_rate':['constant', 'adaptive']

               }

grid_search = GridSearchCV(estimator = MLPClassifier(random_state=42, verbose=False, max_iter=500), param_grid=(param_grid),
                           refit=True, cv=10,verbose=3 ,scoring='f1', return_train_score=True)
grid_search.fit(X_resampled, y_resampled)


best_model = grid_search.best_estimator_
print(grid_search.best_params_, '\n')
print(best_model.get_params(), '\n')
Results = pd.DataFrame(grid_search.cv_results_)
preds = best_model.predict(X_resampled)
print("Test accuracy: ", round(accuracy_score( y_resampled, preds), 3))

Results = pd.DataFrame(grid_search.cv_results_)
print('The best parameters are:', grid_search.best_params_)
y_val = Results['mean_test_score']
y_train = Results['mean_train_score']
results_grid_search1 = pd.DataFrame(Results).sort_values('rank_test_score')[['params', 'mean_test_score']]
results_grid_search2 = pd.DataFrame(Results).sort_values('mean_train_score', ascending=False)[
    ['params', 'mean_train_score']]
headers_val = ["Number", "Parameters", "Validation score"]
headers_train = ["Number", "Parameters", "Train score"]
print(tabulate(results_grid_search1, headers=headers_val, tablefmt="grid"))
print(tabulate(results_grid_search2, headers=headers_train, tablefmt="grid"))
```

```
[CV 10/10] END activation=logistic, hidden_layer_sizes=(11,), learning_rate=adaptive;, score=(train=0.899, test=0.906) total
time=  11.1s
{'activation': 'relu', 'hidden_layer_sizes': (16, 16), 'learning_rate': 'constant'}

{'activation': 'relu', 'alpha': 0.0001, 'batch_size': 'auto', 'beta_1': 0.9, 'beta_2': 0.999, 'early_stopping': False, 'epsi
lon': 1e-08, 'hidden_layer_sizes': (16, 16), 'learning_rate': 'constant', 'learning_rate_init': 0.001, 'max_fun': 15000, 'ma
x_iter': 500, 'momentum': 0.9, 'n_iter_no_change': 10, 'nesterovs_momentum': True, 'power_t': 0.5, 'random_state': 42, 'shuf
fle': True, 'solver': 'adam', 'tol': 0.0001, 'validation_fraction': 0.1, 'verbose': False, 'warm_start': False}

Test accuracy:  0.954
The best parameters are: {'activation': 'relu', 'hidden_layer_sizes': (16, 16), 'learning_rate': 'constant'}
+----------+-------------------------------------------------------------------------------------------+--------------------+
|   Number | Parameters                                                                                |   Validation score |
+==========+===========================================================================================+====================+
|       51 | {'activation': 'relu', 'hidden_layer_sizes': (16, 16), 'learning_rate': 'adaptive'}       |           0.927655 |
+----------+-------------------------------------------------------------------------------------------+--------------------+
```

:Nerual Network prediction hidden layer size = (16,16)

Nerual Networks Remake Final Predictions

```
#--------------------summary and cunclusions multilayer perceptron
model_best_MLP = MLPClassifier(max_iter=500, activation='relu', hidden_layer_sizes=(16,16), learning_rate = 'constant', random_st
model_best_MLP.fit(X_resampled, y_resampled)

# ----------------for the train set
acc_train = accuracy_score(y_resampled, model_best_MLP.predict(X_resampled)) #accuracy of the train
best_train_result_MLP = {'Train accuracy': acc_train}

print(pd.Series(best_train_result_MLP))

# ---------------for the test set
acc_test = accuracy_score(y_test_resampled, model_best_MLP.predict(X_test_resampled))
best_test_result_MLP = {'Test accuracy': acc_test}

print(pd.Series(best_test_result_MLP))
```

```
Train accuracy    0.953674
dtype: float64
Test accuracy    0.856318
dtype: float64
```

:Nerual Network prediction hidden layer size = (7,)

Nerual Networks Remake Final Predictions

In [106]:
```
#--------------------summary and cunclusions multilayer perceptron
model_best_MLP = MLPClassifier(max_iter=500, activation='relu', hidden_layer_sizes=(7,), learning_rate = 'constant', random_state
model_best_MLP.fit(X_resampled, y_resampled)

# -----------------for the train set
acc_train = accuracy_score(y_resampled, model_best_MLP.predict(X_resampled)) #accuracy of the train
best_train_result_MLP = {'Train accuracy': acc_train}

print(pd.Series(best_train_result_MLP))

# ----------------for the test set
acc_test = accuracy_score(y_test_resampled, model_best_MLP.predict(X_test_resampled))
best_test_result_MLP = {'Test accuracy': acc_test}

print(pd.Series(best_test_result_MLP))
```

```
Train accuracy    0.909428
dtype: float64
Test accuracy    0.884838
dtype: float64
```

:Nerual Network prediction hidden layer size = (7,1)

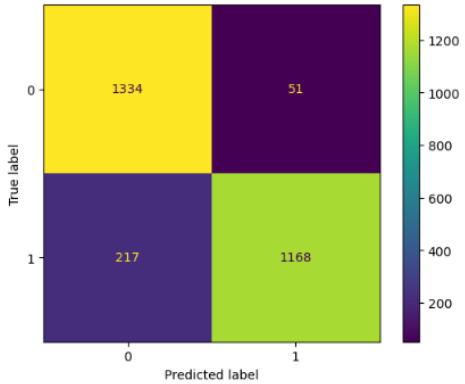
Nerual Networks Remake Final Predictions

In [118]:
```
#--------------------summary and cunclusions multilayer perceptron
model_best_MLP = MLPClassifier(max_iter=500, activation='relu', hidden_layer_sizes=(7,1), learning_rate = 'constant', random_stat
model_best_MLP.fit(X_resampled, y_resampled)

# -----------------for the train set
acc_train = accuracy_score(y_resampled, model_best_MLP.predict(X_resampled)) #accuracy of the train
best_train_result_MLP = {'Train accuracy': acc_train}

print(pd.Series(best_train_result_MLP))

# ----------------for the test set
acc_test = accuracy_score(y_test_resampled, model_best_MLP.predict(X_test_resampled))
best_test_result_MLP = {'Test accuracy': acc_test}

print(pd.Series(best_test_result_MLP))
```

```
Train accuracy    0.919653
dtype: float64
Test accuracy    0.903249
dtype: float64
```

:Confusion matrix

```
Test accuracy    0.903249
dtype: float64
Confusion matrix <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x00000127D9260400>
Classification report              precision    recall  f1-score   support

           0       0.86      0.96      0.91      1385
           1       0.96      0.84      0.90      1385

    accuracy                           0.90      2770
   macro avg       0.91      0.90      0.90      2770
weighted avg       0.91      0.90      0.90      2770
```



:Decision Tree Remake

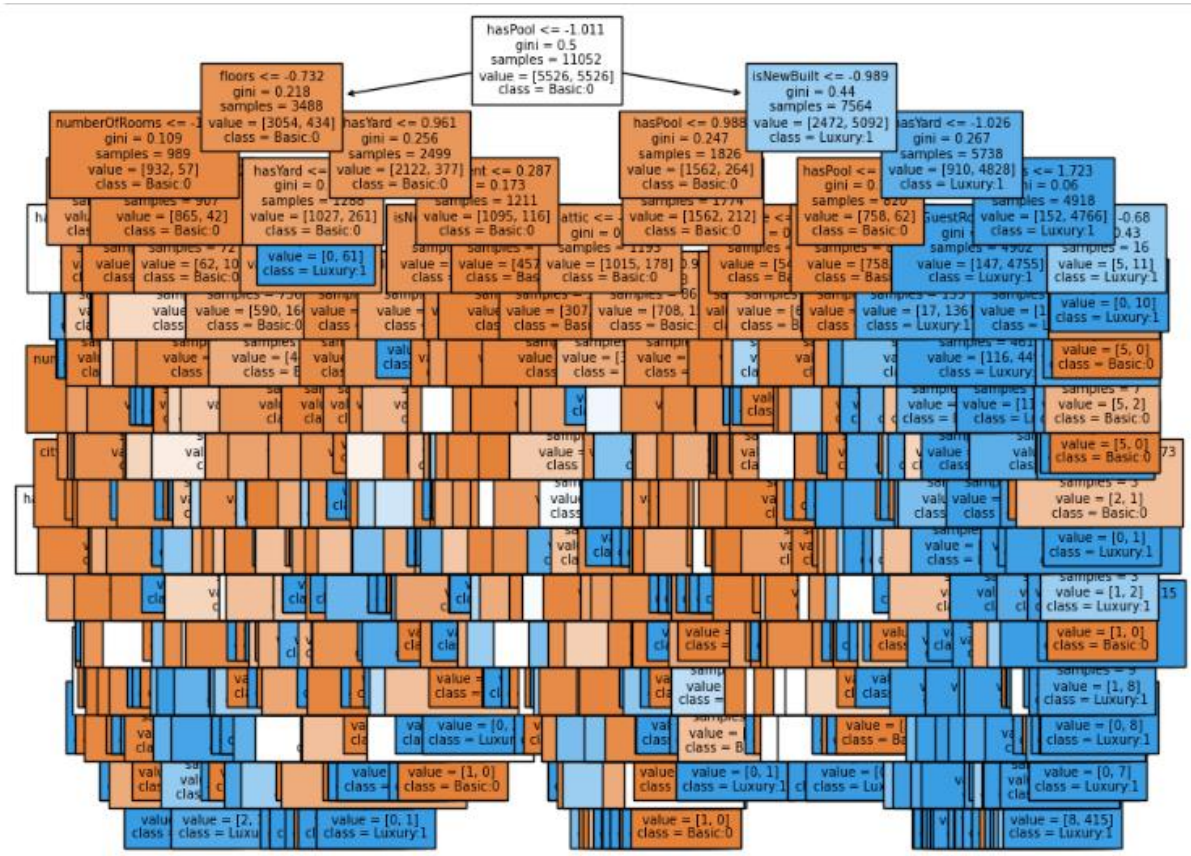
:GreadSearch

GreadSearchCV

In [125]:
```
param_grid = {'max_depth': np.arange(1, 25, 1),
              'criterion': ['entropy', 'gini'],
              'ccp_alpha': np.arange(0, 0.1, 0.01)
             }

# use f1 because target variable is 0 or 1
grid_CV = GridSearchCV(estimator=DecisionTreeClassifier(random_state=42),
                       param_grid=param_grid,
                       refit=True,
                       cv=10, verbose=3 ,scoring='f1', return_train_score=True )

grid_CV.fit(X_resampled, y_resampled)
best_model = grid_CV.best_estimator_
print(grid_CV.best_params_, '\n')
print(best_model.get_params(), '\n')
Results = pd.DataFrame(grid_CV.cv_results_)
preds = best_model.predict(X_test_resampled)
print("Test accuracy: ", round(accuracy_score(y_test_resampled, preds), 3))
```

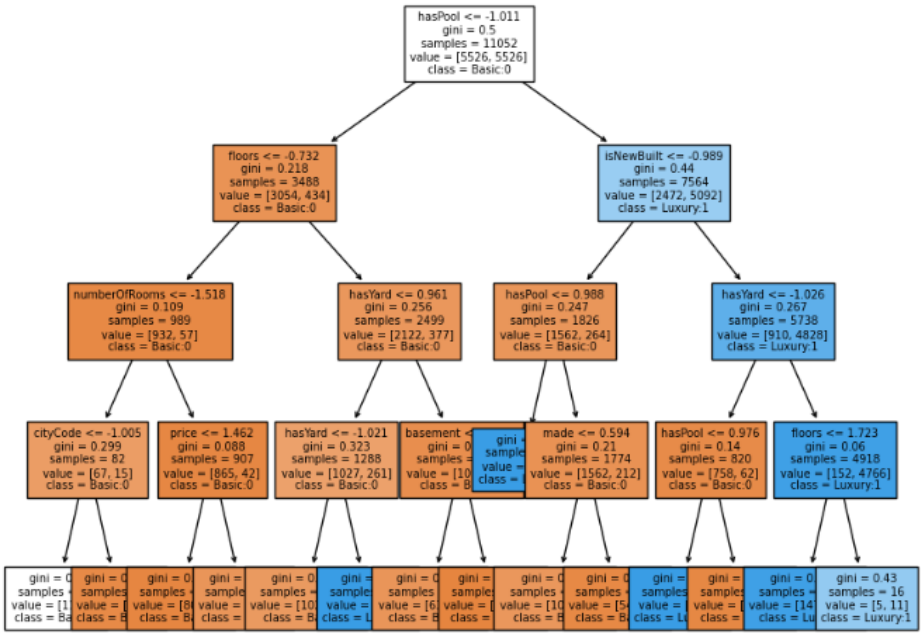
```
[CV 10/10] END ccp_alpha=0.09, criterion=gini, max_depth=23;, score=(train=0.911, test=0.932) total time=   0.1s

[CV 1/10] END ccp_alpha=0.09, criterion=gini, max_depth=24;, score=(train=0.918, test=0.864) total time=   0.1s
[CV 2/10] END ccp_alpha=0.09, criterion=gini, max_depth=24;, score=(train=0.915, test=0.892) total time=   0.2s
[CV 3/10] END ccp_alpha=0.09, criterion=gini, max_depth=24;, score=(train=0.912, test=0.919) total time=   0.1s
[CV 4/10] END ccp_alpha=0.09, criterion=gini, max_depth=24;, score=(train=0.911, test=0.923) total time=   0.1s
[CV 5/10] END ccp_alpha=0.09, criterion=gini, max_depth=24;, score=(train=0.911, test=0.923) total time=   0.1s
[CV 6/10] END ccp_alpha=0.09, criterion=gini, max_depth=24;, score=(train=0.912, test=0.920) total time=   0.1s
[CV 7/10] END ccp_alpha=0.09, criterion=gini, max_depth=24;, score=(train=0.914, test=0.899) total time=   0.1s
[CV 8/10] END ccp_alpha=0.09, criterion=gini, max_depth=24;, score=(train=0.911, test=0.925) total time=   0.1s
[CV 9/10] END ccp_alpha=0.09, criterion=gini, max_depth=24;, score=(train=0.911, test=0.926) total time=   0.1s
[CV 10/10] END ccp_alpha=0.09, criterion=gini, max_depth=24;, score=(train=0.911, test=0.932) total time=   0.1s
{'ccp_alpha': 0.0, 'criterion': 'gini', 'max_depth': 15}

{'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': 15, 'max_features': None, 'max_leaf_nodes': None,
'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'random_stat
e': 42, 'splitter': 'best'}

Test accuracy:  0.897
```

:Decision Tree max depth = 15



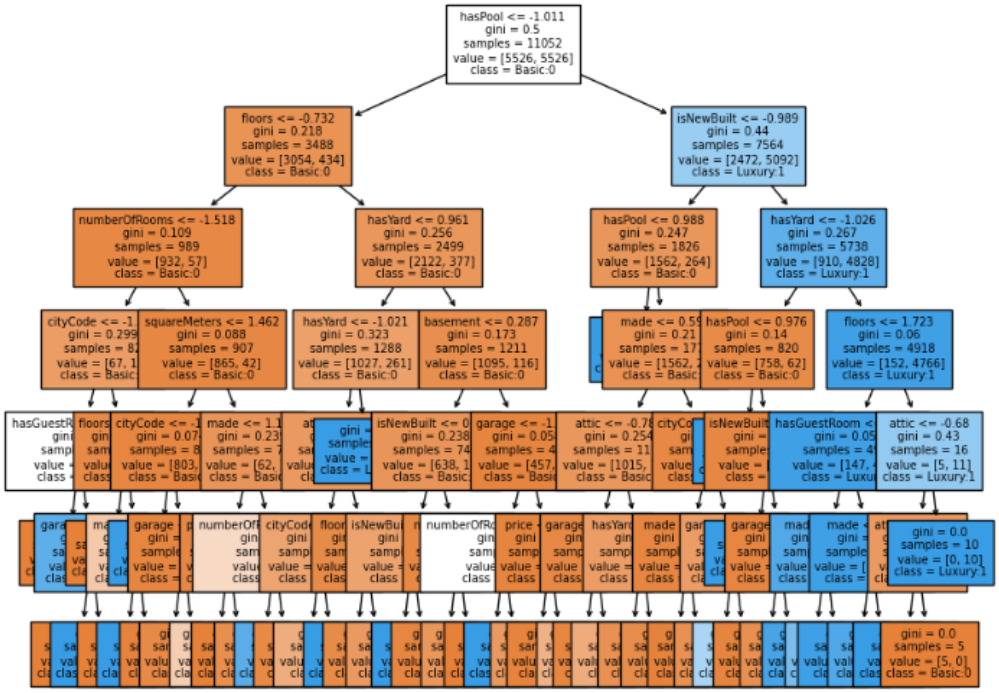

```
Train accuracy    0.992128
dtype: float64
Test accuracy    0.895307
dtype: float64
```

:Decision Tree max depth = 4




```
Train accuracy    0.929515
dtype: float64
Test accuracy    0.927437
dtype: float64
```

:Decision Tree max depth = 6




```
Train accuracy    0.935215
dtype: float64
Test accuracy    0.93213
dtype: float64
```

:Confusion matrix

```
0    0.93213
dtype: float64
Confusion matrix <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x00000127C5E57BE0>
Classification report              precision    recall  f1-score   support

           0       0.90      0.97      0.93      1385
           1       0.97      0.89      0.93      1385

    accuracy                           0.93      2770
   macro avg       0.93      0.93      0.93      2770
weighted avg       0.93      0.93      0.93      2770
```