31.01.2021

למידת מכונה - מודל לחיזוי ערך נדל"ן

נושא המחקר:

בניית מודל מתמטי ויישום אלגוריתם של למידת מכונה לצורכי חיזוי ערך נדל"ן בארה"ב על בסיס נתוני מכירה של נכסים אחרים.

תחום הדעת: מדעי המחשב

שם בית הספר: הכפר הירוק ע"ש לוי אשכול

שם תלמיד: דן סמוילוביץ'

מנחה: יהודה אור

שם בית הספר: הכפר הירוק ע"ש לוי אשכול

סמל המוסד: 580019

כתובת: כפר הירוק 47800 רמת השרון

טלפון: 03-6455621

פרטי התלמיד
דן סמוילוביץ'
מספר ת.ז: 325973428
כתובת: דפנה 28, תל אביב-יפו
נייד: 054-5881348
כתובת מייל: dankot2004@gmail.com
בית הספר: הכפר הירוק ע"ש לוי אשכול
מספר סמל בית הספר: 580019

פרטי המנחה
שם המנחה האקדמי: יהודה אור
מספר ת.ז: 023098007
כתובת: הכרם 3, תל אביב-יפו
נייד: 050-7344457
כתובת מייל: yoooda@gmail.com
תואר אקדמי: מהנדס MA + מוסמך הטכניון בהנדסה, הסבה אקדמאית למחשבים מטע
מקום עבודה: הכפר הירוק ע"ש לוי אשכול

תוכן עניינים

מבוא

מטרת העבודה היא ניתוח מידע ועל פי הסקת מסקנות ממנו והשוואה בין שיטות ואלגוריתמים שונים של למידת מכונה, ליצור וליישם על ידי קוד, מודל אשר מאפשר לחזות עם אחוז שגיאה מינימלי את מחירו של נדל"ן, בהתאם למאגר מידע נתון של נכסים קודמים שנמכרו. באופן זה, המודל יילמד מנכסים קודמים שנמכרו, את מחירו של נדל"ן אחר.

למידת מכונה עוסקת בטיפול באופן תכנותי בנתונים מן העולם האמיתי לפתרון בעיות שונות ומגוונות. בעבודה זו, בחרתי בתחום הנדל"ן, תחום בו יש פוטנציאל רב ליישומי למידת מכונה.

בפרויקט זה, התחלתי מחקר הבעיות השונות הקיימות בלמידת מכונה, השיטות השונות של למידת מכונה, והמתמטיקה והאלגוריתמיקה העומדות מאחוריהן. המשכתי מלמידת תחום חקר הנתונים, למדתי לחקור ולנתח מידע לעומק, על מנת להסיק ממנו מסקנות ובכך לבנות מודל התואם לפתרון הבעיה באופן האופטימלי ביותר.

על ידי הבנת שיטות אלו לעומק, וניתוח מידע ברמה גבוהה - תכננתי ויישמתי מודל שיאפשר חיזוי ערך נדל"ן בניו-יורק. למדתי את השיטות השונות לפתירת בעיות התאמת יתר והתאמת חסר ויישמתי אותן בפרויקט.

בפרויקט תכננתי ויישמתי מודל מתמטי באופן תכנותי - אלגוריתם, אשר פותר את הבעיה. בעבודה, השקעתי מאמצים רבים תוך שימוש בניסיון והידע התיאורטי אותו צברתי במהלך המחקר על מנת לייעל ככל האפשר את ביצועי המודל - הן את דיוקו של המודל והן את מהירותו. כל זאת, על מנת לשפר את האלגוריתם ולצמצם את אחוזי השגיאה בחיזוי ערכם הכספי של נכסים.

- חלק עיוני -

פרק אחת - הרקע המתמטי - מבוא לאלגברה לינארית

על מנת להבין את שאר העבודה נצטרך להבין את מבני נתונים עיקריים בהם נשתמש - מטריצות ווקטורים וכיצד להשתמש בהם. לשם כך נצטרך להבין את המשמעות המתמטית שלהם ואת הפעולות השונות אשר ניתן לבצע איתם.

לכתיבת פרק זה, נעזרתי בספר "אלגברה לינארית ו" של האוניברסיטה הפתוחה וחומרי שיעור של הקורס "Machine Learning" של אוניברסיטת סטנפורד בהנחיית אנדרו נייג.

1.1 הייצוגים באלגברה לינארית - מטריצה ווקטור

1.1.1 המטריצה - Matrix

מטריצה ניתנת להקבלה למערך דו-מימדי. לדוגמה, נבטא מטריצה A:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1k} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2k} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3k} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & & \vdots & & \vdots \\ a_{j1} & a_{j2} & a_{j3} & \cdots & a_{jk} & \cdots & a_{jn} \\ \vdots & \vdots & \vdots & & \vdots & & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mk} & \cdots & a_{mn} \end{bmatrix}$$

המטריצה A היא מטריצה בעלת m שורות ו-n טורים. על כן היא מטריצה בגודל m * n.

כפי שניתן לראות בדוגמה להעיל, ניתן לבטא כל איבר במטריצה על ידי אות המטריצה ולאחריה מספר השורה ומספר הטור. למשל הסימון הבא:

$$a_{31}$$

מסמל את האיבר בשורה השלישית בטור הראשון.

1.1.2 הווקטור - Vector

לעומת המטריצה, וקטור ניתן להקבלה למערך חד מימדי. לדוגמה, נבטא וקטור V:

$$V = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_j \\ \vdots \\ v_m \end{bmatrix}$$

כפי שניתן לראות, הוקטור דומה מאוד למטריצה. זאת בגלל שהוקטור הוא מטריצה. וקטור הוא מטריצה בעלת מספר שורות אך רק בעלת טור אחד. למשל, הוקטור הנוכחי הוא מטריצה בגודל 1 * m.

בדומה למטריצה, ניתן לבטא כל איבר בוקטור על ידי אות הוקטור ולאחריה מספר השורה של האיבר. למשל הסימון הבא:

$$V_3$$

מסמל את האיבר בשורה השלישית של הוקטור, או אם נתייחס לזה כמטריצה אז את האיבר בשורה השלישית ובטור הראשון (והיחיד).

חשוב לציין, בעוד שבמתמטיקה אנו נתחיל את סימון השורות והטורים מהמספר 1, בשפות תכנות אנו נתחיל מ-0.

1.2 הפעולות

1.2.1 חיבור וחיסור מטריצות

תנאי הכרחי לחיבור או חיסור מטריצות הוא ששני הצדדים יהיו בעלי גודל שווה (בעלי מספר שורות שווה ומספר טורים שווה).

כאשר אנו מבצעים חיבור או חיסור מטריצות, התוצאה תהיה מטריצה באותו גודל של שני המטריצות עליהן מתבצעת הפעולה. במטריצת התוצאה, כל איבר יהיה שווה לתוצאת הפעולה בין כל שני איברים תואמים מבין המטריצות, באופן הבא:

$$Cij = Aij + Bij$$

$$\mathbf{A}+\mathbf{B} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{bmatrix}$$
$$= \begin{bmatrix} a_{11}+b_{11} & a_{12}+b_{12} & \cdots & a_{1n}+b_{1n} \\ a_{21}+b_{21} & a_{22}+b_{22} & \cdots & a_{2n}+b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}+b_{m1} & a_{m2}+b_{m2} & \cdots & a_{mn}+b_{mn} \end{bmatrix}$$

1.2.2 כפל וחילוק מטריצה בסקלר

כאשר אנו מכפילים או מחלקים מטריצה או וקטור במספר, התוצאה תהיה מטריצה באותו גודל בה חילקנו או הכפלנו כל איבר בסקלר, באופן הבא:

אם נתון לנו מטריצה A וסקלר k.

האיבר במטריצת תוצאת המכפלה A * k (נסמן מטריצה B) תראה כך עבור כל איבר:

$$Bij = Aij * k$$

האיבר במטריצת תוצאת החילוק A / k (נסמן מטריצה B) תראה כך עבור כל איבר:

$$Bij = Aij / k$$

1.2.3 כפל מטריצה בוקטור

בכפל מטריצה בוקטור, תנאי הכרחי למכפלה הוא שמספר הטורים במטריצה יהיה שווה למספר השורות בוקטור. כמו כן, תוצאת המכפלה תהיה תמיד וקטור בגודל מספר השורות של המטריצה.

מכפלה של וקטור בגודל m * n ווקטור בגודל n * 1 יהיה וקטור בגודל m * 1.

במכפלת וקטור ומטריצה. ניקח שורה שורה מתוך המטריצה, והערך שיהיה בוקטור התוצאה בשורה הנ"ל יהיה שווה לסכום המכפלות של כל איבר בשורה, באיבר המתאים מתוך הוקטור. לדוגמה במכפלה הבאה:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} ax + by + cz \\ dx + ey + fz \\ gx + hy + iz \end{bmatrix}$$

על מנת לחשב את הערך הראשון בוקטור התוצאה, ניקח את השורה הראשונה של המטריצה - [a b c]. לאחר מכן, נכפיל כל איבר בערך המתאים לו מתוך הערך וניקח את הסכום. כך, ערך האיבר הראשון בוקטור יהיה שווה ל-a*x + b*y + c*z.

1.2.4 כפל מטריצה במטריצה

בכפל בין שתי מטריצות, אנו מפרקים את התרגיל לכמה מכפלות של מטריצה בוקטור ומשרשרים את התוצאה.

בנוסף, תוצאת מכפלה של מטריצה בגודל n * m עם מטריצה בגודל o * n תהיה מטריצה בגודל m * o.

מפה ניתן להסיק, שכמו במכפלת וקטור ומטריצה, גם כאן קיים תנאי למכפלה שהוא שמספר הטורים במטריצה הראשונה יהיה שווה למספר הטורים במטריצה השנייה.

ניקח את הדוגמה הבאה:

$$c_{11}=a_{11}b_{11}+a_{12}b_{21}+a_{13}b_{31}+a_{14}b_{41}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \\ b_{41} & b_{42} & b_{43} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{bmatrix}$$

2 x 4 4 x 3 2 x 3

$$c_{22}=a_{21}b_{12}+a_{22}b_{22}+a_{23}b_{32}+a_{24}b_{42}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \\ b_{41} & b_{42} & b_{43} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{bmatrix}$$

במקרה זה, ניקח את המטריצה הראשונה ונכפיל אותה עם הוקטור שהוא טור אחד במטריצה השנייה, וכך עם כל טור, כפי שניתן לראות בתמונה.

ישנם כמה עקרונות חשובים בכפל מטריצות:

- A * B ≠ B * A - כפל מטריצות היא לא פעולה חילופית עבורם.
- A * (B * C) = (A * B) * C - כפל מטריצות היא פעולה קיבוצית.

1.2.5 ביצוע הפיכה - Inverse

נסמן את המטריצה ההופכית של המטריצה A ב-A^{-1}.

המשמעות של המטריצה ההופכית היא שמכפלת המטריצה במטריצה ההופכית שלה נותנת לנו את מטריצת הזהות.

מטריצת הזהות זו מטריצה ריבועית (בה מספר השורות והטורים שווה), כאשר כל הערכים הם אפס מלבד האלכסון העובר משמאל למעלה לימין למטה, אשר בו כל הערכים הם 1.

מטריצה הופכית - מטריצה אשר ניתן לבצע לה הפיכה, היא מטריצה בעלת מספר שורות וטורים שווה.

1.2.6 ביצוע טרנספוזיציה - Transpose

טרנספוזיציה של מטריצה, היא אותה המטריצה אשר סובבו אותה 90 מעלות בכיוון השעון. לדוגמה:

$$A\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix}\qquad A^{T}\begin{bmatrix}1&4&7\\2&5&8\\3&6&9\end{bmatrix}$$

1.3 למה נשתמש בידע זה?

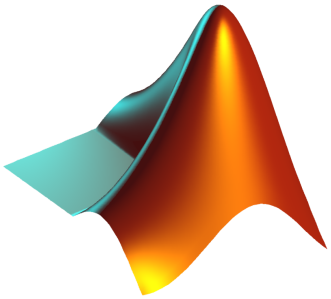
השאלה בה חשוב לסיים פרק זה, היא מדוע נזדקק בידע מפרק זה?
התשובה לכך, היא שבמהלך העבודה יעשה שימוש רחב בוקטורים ומטריצות לצורך שמירה ועבודה על מאגרי המידע איתם אעבוד. כמו כן, למידת מכונה משתמשת רבות בשיטת הוקטורים והמטריצות בשל החישובים הרבים הנעשים בתחום אשר השימוש במטריצות ווקטורים מקל עליהם. כתוצאה מכך, מרבית הנוסחאות בלמידת מכונה נכתבות בצורה של מטריצות.

מטריצות ווקטורים יאפשרו לנו לשמור בהמשך את המידע בו נשתמש באופן נוח ויעיל. כמו כן, שימוש במטריצות ווקטורים יקצר לנו מאוד את הקוד, ויהפוך חישובים רבים שלנו ליותר יעילים ומהירים. ביצוע פעולות על מטריצות במקום על הנתונים באופן השמור אחרת יתן תוצאה הרבה יותר מהירה.

פרק שתיים - מבוא ל-Matlab

לצורך כתיבת האב-טיפוס אשתמש בשפת Matlab. לכן, בפרק זה אציג את השפה, ופקודותיה - ה-Syntax.

2.1 מהי שפת Matlab?



MATLAB היא שפת תכנות בתחום המתמטי, המיוצרת על ידי חברת MathWorks. השם MATLAB הוא ראשי תיבות של המילים "מעבדת מטריצות" - "matrix laboratory". בהתאם לשם, התוכנה מאפשרת מאפשרת טיפול קל ונוח במטריצות. בנוסף לכך, השפה מאפשרת: שימוש בפונקציות ובנתונים, מימוש אלגוריתמים על נתונים, ועוד.

התוכנה נמצאת בשימוש נרחב באקדמיה, במיוחד בנושאי הלימוד של אלגברה ליניארית ואנליזה נומרית, והיא נפוצה גם בתחום חקר הנתונים (Data Science) - עיבוד תמונה, רשתות נוירונים מלאכותיות, סימולציות של תהליכים פיזיקליים, חישובים נומריים ותחומים נוספים.

2.2 מדוע אשתמש בשפה זו בעבודה?

אז מה הצורך לשימוש בשפה זו בעבודה?

כפי שכבר הסברתי, השפה נמצאת בשימוש רחב בתחום ה-Data Science ולמידת המכונה. חברות רבות העוסקות בתחום זה, משתמשות בשפה זו לצורך כתיבת אב-טיפוסים. לכן, ידע בשפה זו מאוד חיוני לעיסוק בתחומים אלו.

כמו כן, שימוש בשפה זו וכתיבה בה, יעזור לי ללמוד באופן עמוק יותר את היישום הלינארי של למידת מכונה - את השימוש הנחוץ במטריצות ווקטורים, וכדומה. על ידי מעבר ממתמטיקה לכתיבת קוד מתמטי, אבין טוב יותר את החישובים אותם אבצע לאחר מכן בקוד הפייתון ובקוד למידת המכונה בכלל.

בנוסף לכך, שפה זו שפה דגש על ההיבט המתמטי - ההיבט המעניין אותי בתחום למידת המכונה. חלק שחשוב לי מאוד בפרוייקט זה, ההבנה המתמטית של למידת מכונה.

2.3 הפקודות - Syntax ואופן העבודה ב-Matlab

כעת נעבור על הידע לו נזדקק לצורך כתיבת האב-טיפוס. לכתיבת תת פרק זה, נעזרתי ב-MATLAB Documentation של MathWorks.

2.3.1 הגדרת משתנים

בשונה משפות תכנון כגון Java או #C, אך בדומה לשפת פייתון עליה נדבר בהמשך, ב-Matlab אין צורך בהגדרת טיפוס המשתנה, שמות המשתנים מתפקדים רק כשמות אשר יכולים לאגור כל סוג מידע אותו נכניס אליהם.

לכן, אם נרצה למשל להגדיר משתנה A השווה ל-5, נכתוב פשוט:

```
A = 5;
```

ואם נרצה אחר כך להחליף אותו כך שיהיה וקטור או מחרוזת, נוכל פשוט להשוות אותו כך שוב.

2.3.2 הגדרת מטריצות ווקטורים

ישנם מספר דרכים ליצור וקטורים ומטריצות ב-MATLAB. אחת מהן היא הגדרה ישירה בסוגריים מרובעות, באופן הבא:

```
A = [1 2 3 4 5];
```

כעת יצרנו וקטור A שאיבריו הם מ-1 עד הספרה 5. יכולנו לעשות זאת גם כך:

```
A = 1:6;
```

כמו כן, ניתן היה לשים לב, שעל מנת להפריד איברים צריך להפריד אותם אך ורק ברווח.
אם אנו רוצים להכניס מספר שורות, ובכך ליצור מטריצה נעשה זאת על ידי הפרדת השורות בתוך הסוגריים המרובעות בתו **;** המציין גם סוף שורת קוד. למשל:

```
A = [1 2 3; 4 5 6];
```

כך יצרנו מטריצה בעלות שתי שורות ו-3 עמודות.

דרך נוספת ליצירת מטריצה היא **הפונקציה zeros**, היוצרת מטריצה שכל איבריה שווים ל-0 בגודל הניתן לה עבור כל מימד אחד אחר השני' אם ניתן מספר אחד תחזיר מטריצה דו מימדית בה הגדלים של שתי המימדים שווים למספר זה.

למשל ניצור מטריצה תלת מימדית בגודל 3 * 2 * 1:

```
A = zeros(1, 2, 3);
```

בדומה ל-zeros קיימת גם **הפונקציה ones**, המבצעת אותו דבר באותו אופן אך כל איבריה שווים ל-1.

על מנת ליצור מטריצת זהות נשתמש ב**פונקציה eye** היוצרת מטריצת זהות בגודל הניתן לה. למשל ניצור מטריצת זהות בגודל 5 * 5:

```
A = eye(5);
```

ניתן לגשת לכל איבר במטריצה או וקטור על ידי הוספת סוגריים והמיקום שאנחנו רוצים בהתאם למימדים. למשל בשביל לקבל את האיבר הראשון בוקטור A:

```
S = A(1);
```

2.3.3 לולאות

לולאות for יתחילו במילה for ויסתיימו בשורה בה כתוב end. דוגמה ללולאת for להגדרת ערכי מטריצה:

```
s = 10;
H = zeros(s);
for c = 1:s
   for r = 1:s
      H(r,c) = 1/(r+c-1);
   end
end
```

לולאות while יתחילו במילה while ויסתיימו בשורה בה כתוב end. דוגמה ללולאת while:

```
n = 10;
while n > 1
   n = n-1;
end
```

2.3.3 כתיבת פונקציות

כעת נלמד לכתוב פונקציות.

כל פונקציה תוגדר בקובץ משלה, המתחילה במילה function ומסתיימת ב-end.

ניקח לדוגמה את הפונקציה הבאה:

```
function A = func(num1, num2)
A = num1 + num2;
end
```

כפי שניתן לראות, בשורה הראשונה יוגדר לפונקציה מה שם הדבר אותו תחזיר, שם הפרמטרים אותם תקבל ומה שמה. במקרה זה:

- A - המשתנה המוחזר. לעומת מרבית השפות האחרות, ב-MATLAB הגדרת החזרה תופיע בהתחלה, שם גם יכתב שם המשתנה המוחזר.
- func - שם הפונקציה
- num1 - הפרמטר הראשון
- num2 - הפרמטר השני

חשוב להבין שב-MATLAB דרך נהוגה להחזיר מספר משתנים היא על ידי החזרת וקטור. למשל:

```
function [A, B] = func(num1, num2)
```

2.3.4 פקודות שימושיות לעבודה עם מטריצות

כעת נעבור על פונקציות שימושיות.

הפונקציה sum

כשמה, פונקציה sum מקבלת מטריצה או וקטור, ומחזירה את סכומה באופן הבא:

- אם A הוא וקטור, אז sum(A) מחזיר את סכום כל התאים (האיברים).
- אם A היא מטריצה, אז sum(A) מחזיר וקטור שורה המכיל את הסכום של כל עמודה.

הפונקציה length

כשמה, פונקציה length מקבלת מטריצה או וקטור, ומחזירה את האורך שלה באופן הבא:

- אם A הוא וקטור, אז length(A) מחזיר את האורך שלו.
- אם A היא מטריצה, אז length(A) מחזיר את אורך המימד ההכי ארוך שלו.

הפונקציה size

פונקציה זו, מקבלת מטריצה או וקטור ומחזירה וקטור של אורך כל הממדים שלה.
למשל, אם A היא מטריצה בגודל 4 * 3, הוקטור שיוחזר הוא [4 3].

כמו כן, ניתן להכניס לפונקציה זו, אילו מימדים ספציפיים מעניינים אותנו, בין אם מימד אחד, בין אם מספר מימדים. נעשה זאת למשל כך, size(A, 1) לשם קבלת אורך המימד הראשון של A.

הפונקציה mean

פונקציה mean מקבלת מטריצה או וקטור, ומחזירה את הממוצע שלה באופן הבא:

- אם A הוא וקטור, אז mean(A) מחזיר את הממוצע של האיברים שלו.
- אם A היא מטריצה, אז mean(A) מחזיר וקטור המכיל את הממוצע של כל עמודה.

הפונקציה std

פונקציה std מקבלת מטריצה או וקטור, ומחזירה את סטיית התקן שלה באופן הבא:

- אם A הוא וקטור של תצפיות, אז סטיית התקן היא סקלרית.
- אם A היא מטריצה שהעמודות שלה הן משתנים אקראיים ושורותיה הן תצפיות, אז S הוא וקטור שורה המכיל את סטיות התקן המתאימות לכל עמודה.
- אם A הוא מערך רב מימדי, אז std(A) פועל לאורך ממד המערך הראשון שגודלו אינו שווה ל-1, ומתייחס לאלמנטים כאל וקטורים. הגודל של ממד זה הופך ל-1 בעוד הגדלים של כל שאר הממדים נשארים זהים.

הפונקציות inv ו-pinv

פונקציות inv ו-pinv מקבלות מטריצה ומחזירה את המטריצה ההופכית שלה. כפי שהגדרנו לא כל מטריצה היא הופכית, לשם כך קיים pinv המחזיר את המטריצה ההופכית גם אם המקורית אינה מקיימת הופכיות.

ביצוע טרנספוזיציה

אם ברצוננו לבצע טרנספוזיציה לוקטור או מטריצה, כל שעלינו לעשות הוא להוסיף את התו **'**, באופן הבא:

```
X_transpose = X';
```

פרק שלוש - מבוא ל-Python

3.1 מדוע בחרתי ב-Python?

אז נשאלת השאלה, באיזה שפה יכתב הקוד לפרוייקט ולמה?

בחרתי בפייתון מכמה סיבות. הראשונה, פייתון היא אחת השפות הנפוצות ביותר לשימוש בתחום למידת המכונה. זאת מכיוון, שפייתון מאפשרת בשורות קוד קלות לבצע פעולות מורכבות כגון פעולות מתמטיות ברמה גבוה, בקלות.

סיבה נוספת לשימוש בפייתון היא שפייתון היא שפה בעלת מאגר ספריות גדול מאוד ומגוונות, בין היתר לצרכי למידת מכונה המקלים את העבודה. אך כאן חשוב לציין, שבעבודתי מטרתי לחקור לעומק את ההיבט המתמטי והאלגוריתמי בלמידת מכונה לכן לא אשתמש בספריות היוצרות ומאמנות עבורך מודלים ללמידה, אלא רק בספריות המספקות דברים בסיסיים הנחוצים לעבודה של למידת מכונה, על ספריות אלו אדבר בהמשך.

הסיבה השלישית לשימוש בפייתון, היא שמטרת העבודה שלי היא לחקור ולהרחיב אופקים. לכן, החלטתי להשתמש בין היתר בשפה שפחות יצא לי להשתמש בה. לדעתי, נסיון כתיבת פרוייקט מורכב בשפה זו ילמד אותי הרבה.

3.2 הספריות בהם אשתמש

לצורך הפרוייקט אשתמש במספר ספריות ומודולים לצרכים שונים, אשר אפרט עליהם כעת. בהסברי אתבסס על ה-Documantations של ספריות אלו.

3.2.1 Numpy

אז מה זה Numpy?

Numpy זו ספרייה לחישובים מדעיים בפייתון. ספרייה זו מספקת מערך בעל מספר מימדים, אובייקטים כגון מטריצות, ופעולות שונות לביצוע על מערכים מסוגים שונים - פעולות מתמטיות, לוגיות, שינוי צורה, ארגון, אלגברה לינארית בסיסית ועוד.

הבסיס לספרייה הוא האובייקט ndarray. אובייקט של מערך n-מימדי מסוגים שונים, בעל פעולות רבות בעלות ביצועים מהירים יותר מאלו של פייתון הרגילה. לאובייקט זה מספר מאפיינים:

- למערכי NumPy יש גודל קבוע בעת היצירה, בניגוד לרשימות Python (שיכולות לגדול באופן דינמי). שינוי הגודל של ndarray ייצור מערך חדש וימחק את המקור.

- האלמנטים במערך NumPy נדרשים כולם להיות מאותו סוג נתונים, ולכן יהיו באותו גודל בזיכרון. היוצא מן הכלל: ndarray יכול להיות מערך של מערכים של אובייקטים שונים (כולל מערכי NumPy ומערכים של Python), ובכך לאפשר מערכים של אלמנטים בגדלים שונים.

- מערכי NumPy מאפשרים ביצוע פעולות מתמטיות מתקדמות ואחרות על מספר רב של נתונים. בדרך כלל, בצורה יעילה יותר אשר גם דורשת פחות קוד ממה שניתן לבצע בעזרת מבני נתונים בסיסיים ב-Python ופעולות שלהם.

- שפע של ספריות מדעיות ומתמטיות הכתובות ל-Python משתמשות במערכי NumPy. על אף, שבדרך כלל הם מקבלים קלט רגיל של Python, הם ממירים קלט כזה למערכים של NumPy לפני העיבוד, ולעתים קרובות הם מחזירים מערכי NumPy. במילים אחרות, על מנת להשתמש ביעילות בהרבה (אולי אפילו ברוב) התוכנות המדעיות/מתמטיות מבוססות Python של ימינו, עצם הידיעה כיצד להשתמש בסוגי מבני הנתונים המובנים של Python אינה מספיקה - צריך גם לדעת איך להשתמש במערכי NumPy.

ההתחשבות בגודל ומהירות הרצף חשובות במיוחד במחשוב מדעי. לדוגמה, במקרה של הכפלת כל אלמנט במערך חד מימדי עם האלמנט המתאים ברצף אחר באותו אורך. אם הנתונים מאוחסנים בשתי רשימות a ו-b שונות בפייתון, נוכל לחזור על כל אלמנט באופן הבא:

```
result = []
for i in range(len(a)):
    result.append(a[i]*b[i])
```

אבל מה אם המעריכים מכילים מס' גדול מאוד של ערכים, אם למשל מדובר במליונים של ערכים? במקרה זה תפגע היעילות בשל השימוש הלא יעיל בלולאות.

במקרה של מערכים דו מימדיים היעילות תפגע אך יותר:

```
for i in range(rows):
    for j in range(columns):
        result[i][j] = a[i][j]*b[i][j];
```

Numpy פותר בעיה זו, על ידי כתיבה נקייה יותר אשר גם מאפשרת ביצועים גבוהים יותר.

```
c = a * b
```

Pandas 3.2.2

אז מה זה Pandas?

Pandas היא ספרייה של פייתון אשר אובייקטה העיקרי הוא DataFrame - אובייקט מהיר ויעיל למניפולציה של נתונים בעזרת מערכת ציון (indexing) יעילה ונוחה של הספרייה.

כמו כן, הספרייה מספקת כלים לקריאה וכתיבת מידע למבני נתונים שונים בזיכרון בפורמטים שונים. לדוגמה: קבצי CSV וטקסט, קבצי Excel, מאגרי SQL, ועוד.

בנוסף, הספרייה מבצעת ארגון נתונים חכם וטיפול בנתונים חסרים על ידי שימוש אוטומטי מבוסס תוויות, בהם ניתן להשתמש בחישובים ותמרון קל בנתונים מבולגנים לצורה מסודרת.

בין היתר, הספרייה מאפשרת שינוי צורה קל של מאגרי נתונים, תיוג מבוסס תוויות, שימוש במערכת אינדקס נוחה, שימוש בתת מערכים של מערכי נתונים גדולים, מחיקה והכנסת מידע פשוטה לתוך ומחוץ המאגר תוך הגדלת והקטנת גודל המאגר באופן דינמי ועוד.

באופן זה, הספרייה מספקת דרך פשוטה לביצוע מניפולציות על מאגרי מידע גדולים וקטנים.

פייתון תוך שימוש ב-pandas נמצא בשימוש במגוון רחב של תחומים אקדמיים ומסחריים, כולל פיננסים, מדעי המוח, כלכלה, סטטיסטיקה, פרסום, ניתוח אינטרנט ועוד.

Matplotlib 3.2.3

matplotlib היא ספריית הדמיית נתונים וציור גרפי עבור Python המשתמשת בספרייה NumPy.

סקריפט של matplotlib ב-Python בנוי כך שכמה שורות קוד הן כל מה שנדרש ברוב המקרים כדי ליצור גרף נתונים חזותי. בתוך matplotlib ישנו API בו נשתמש בעיקר שממו pyplot:

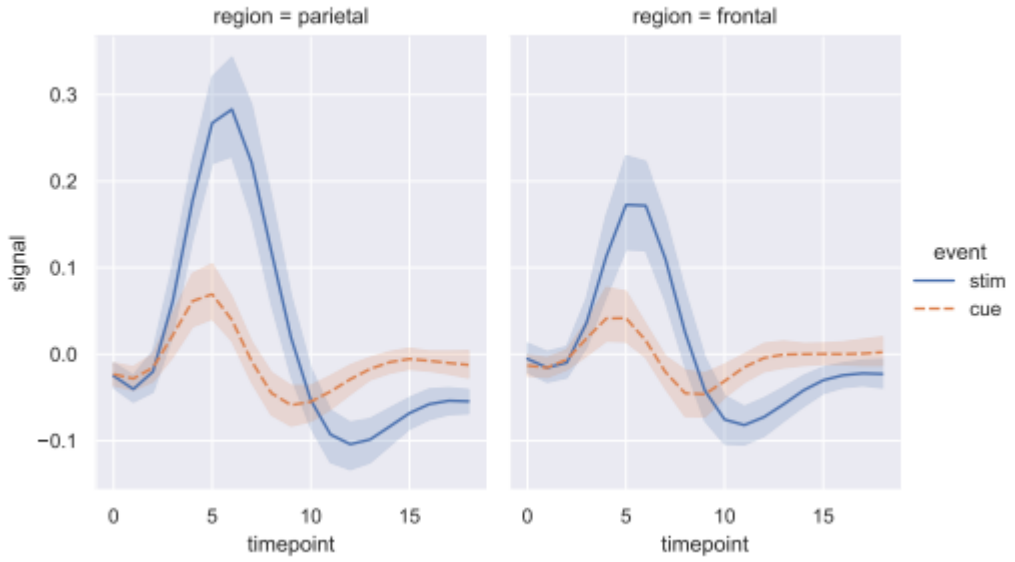
ה-API של pyplot, הענף של אובייקטי קוד של Python ובראשם matplotlib.pyplot. המספק את האובייקט של הגרף עצמו, ומערכת הצירים עליהן יצויר הגרף. זהו החלק בו נשתמש בעיקר בעבודה.

Pyplot בעל ממשק נוח בסגנון MATLAB. למעשה, matplotlib נכתב במקור כחלופה של קוד פתוח עבור MATLAB. בזמן ש-Matplotlib בעל API נוסף של אובייקטים מונחי עצמים, נעשה יותר שימוש בpyplot כתוצאה מכך שה-API האחר נחשב לקשה יותר לשימוש. כתוצאה מכך, ממשק ה-pyplot נמצא בשימוש נפוץ יותר, והוא מכונה כברירת מחדל.

Seaborn 3.2.4

Seaborn היא ספרייה ליצירת גרפיקה סטטיסטית ב-Python. הוא מספק ממשק ברמה גבוהה ל-matplotlib ומשתלב באופן הדוק עם מבני נתונים של pandas. ספריית seaborn מספקת API מונחה מערך נתונים, אשר מקל על תרגום שאלות על נתונים לגרפיקה שיכולה לענות עליהן.

כאשר נתון מערך נתונים ומפרט של הגרף שצריך לעשות, Seaborn ממפה אוטומטית את ערכי הנתונים לתכונות חזותיות כגון צבע, גודל או סגנון, ומוסיף לייצוג הגרפי תוויות צירים אינפורמטיביות. לדוגמה:



פונקציות רבות של Seaborn יכולות ליצור צורות בעלות פאנלים מרובים המעוררים השוואות בין קבוצות משנה מותנות של נתונים או בין זוגות שונים של משתנים במערך הנתונים.

Seaborn נועד להיות שימושי לאורך כל מחזור החיים של פרויקט מדעי. על ידי הפקת גרפיקה מלאה מקריאת פונקציה אחת עם ארגומנטים מינימליים, Seaborn מאפשר יצירת אב טיפוס מהיר וניתוח נתונים חקרניים. ועל ידי מתן אפשרויות נרחבות להתאמה אישית, יחד עם חשיפת אובייקטי ה-matplotlib הבסיסיים, ניתן להשתמש בו כדי ליצור צורות מלוטשות באיכות גבוהה.

PySimplyGUI 3.2.5

PySimpleGUI היא ספרייה ליצירת GUI ב-Python.

באנגלית - Graphical User Interface (או בקיצור GUI) ובעברית - ממשק משתמש גרפי, הוא ממשק משתמש לתוכנה או לאתר אינטרנט, על בסיס עיצוב גרפי של המסך המוצג למשתמש. ממשק מסוג זה מאפשר למשתמשים ליצור אינטראקציה עם התוכנה באמצעות אייקונים גרפיים כגון: תיבות טקסט, כפתורים, ועוד. זאת בניגוד לאינטרקציה הנעשת על ידי פקודות בלבד.

הספרייה מאפשרת לממש ממשק גרפי על ידי כך שמאפשרת ליצור חלונות ולהגדיר בהם widgets שונים (כפתורים, תיבות טקסט, ועוד), ובכך להנגיש את התוכנה למשתמש.
בשילוב עם Matplotlib זה יאפשר לי להנגיש את המודל, וליצור אפשרות נוחה להשתמש בו.

Itertools 3.2.6

Itertools היא מודול של Python המספק פונקציות שונות שעובדות על איטרטורים כדי לייצר איטרטורים מורכבים. מודול זה פועל ככלי מהיר וחסכוני בזיכרון המשמש לבד או בשילוב ליצירת אלגברה איטרטיבית.

הפונקציונליות העיקרית בה אשתמש מתוך מודול זה היא יצירת רשימה של פייתון בעלת כל הקומבינציות של רשימת ערכים. באופן הבא:

```
possible_col = range(num)
all_combinations = []
for r in range(len(possible_col) + 1):
      combinations_object = itertools.combinations(possible_col, r)
      combinations_list = list(combinations_object)
      all_combinations += combinations_list
```

Math 3.2.7

Math הו-א מודול של Python המספק פונקציות שונות המאפשרות להשתמש בפעולות מתמטיקה שונת, כגון: חזקה, שורש, ערך מוחלט, ועוד.

os.path 3.2.8

os.path היא מודול של Python המספק פונקציות שונות המאפשרות להשתמש במערכת הקבצים של מערכת ההפעלה, כגון: לקרוא קבצים, לפתוח אותם, לשמור אותם, לעבוד עם pathnames ועוד.

פרק ארבע - מבוא ללמידת מכונה

לכתיבת פרק זה, נעזרתי ב-"Introduction to Machine Learning" מאת א' סמולה ו-ס.ב.נ וישווניטן מאוניברסיטת קיימברידג'. כמו כן, נעזרתי ב-"Data Mining and Knowledge Discovery Handbook" של א' מיימון ו-ל' רוקח, מאוניברסיטת תל אביב.

4.1 מהי למידת מכונה?

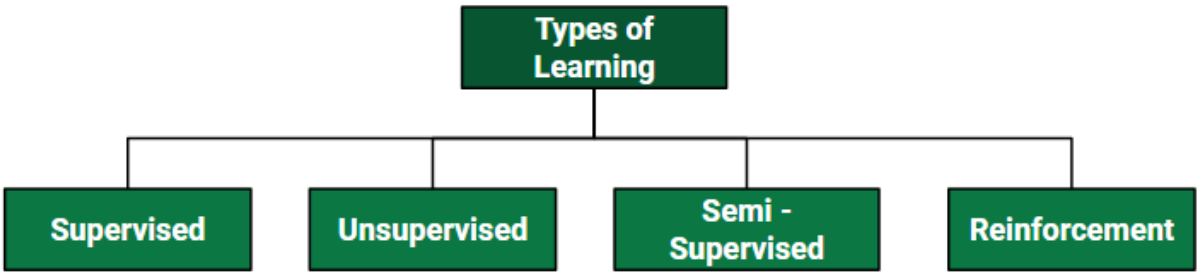
על פי IBM, למידת מכונה (Machine Learning) הוא ענף בתחום של בינה מלאכותית (AI) ומדעי המחשב אשר שם דגש על השימוש בנתונים ואלגוריתמים לצורך חיקוי הדרך בה אנשים לומדים, על מנת לבצע שיפור בהדרגה של הדיוק שלו.

הרעיון העיקרי של אלגוריתם למידת מכונה הוא השימוש בניסיונות חוזרים תוך שינוי המודל עצמו לצורך הסתגלות בצורה מרבית עבור המידע הקיים. בנוסף, ככל שמודל זה מקבל יותר נתונים, כך הוא מדייק את עצמו יותר. הסתגלות למספר רב יותר של נתונים מאפשרת למודל להיות מדויק יותר עבור מקרים שונים. כך בעצם נוצרת "למידת מכונה", למידת המכונה בכוחות עצמה על ידי ניסיונות חוזרים.

4.2 סוגי למידת מכונה

לכתיבת תת פרק זה, נעזרתי ב-"Introduction to Machine Learning" של אוניברסיטת קיימברידג' ו-.

למידת מכונה (Machine Learning) מחולקת **לארבעה סוגים עיקריים**: למידה מונחית (Supervised), למידה בלתי מונחית (Unsupervised), למידה חלקית מונחית (Semi-supervised) ולמידת חיזוק (Reinforcement).



4.2.1 למידה מונחית (Supervised)

למידה מונחית היא למידה בה גם הבעיה וגם התוצאה מוגדרות מראש. למשל, מודל חישובי אשר מטרתו לקטלג את המיילים לספאם ולא ספאם, הוא מודל הנמצא תחת למידה מונחית מכיוון שגם הבעיה - לקטלג את הדואר האלקטרוני, וגם התוצאות האפשריות - ספאם ולא ספאם, מוגדרות מראש.

האלגוריתם יתבסס על מאגר מידע שבו קיימות גם ה"תשובות" לבעייתו - מה שנקרא **מידע מסומן**. למשל, במקרה של מודל לצורך קטלוג הדואר האלקטרוני, האלגוריתם יתבסס על מאגרי מידע הכוללים מיילים קודמים אשר כבר קוטלגו לספאם ולא ספאם, ומכך הוא יסיק על מיילים חדשים האם הם ספאם או לא.

4.2.2 למידה בלתי מונחית (Unsupervised)

למידה בלתי מונחית היא למידה בה הבעיה מוגדרת על ידי המתכנת אמנם התוצאה לא מוגדרת. ניקח שוב לדוגמה את המודל לקטלוג הדואר האלקטרוני, אך כעת איננו נגדיר לו את התוצאה. המודל יהיה תחת למידה בלתי מונחית כיוון שהבעיה מוגדרת לו - קיטלוג הדואר האלקטרוני, אך התוצאות האפשריות לא מוגדרות לו. במקרה זה, מטרת המודל תהיה לארגן את המידע לקבוצות על פי נקודות דמיון ושוני שהוא מוצא. המתכנת אמנם הגדיר את הבעיה, אולם המודל עצמו הוא שיגדיר את הקבוצות אליהן יארגן את המידע על פי מאפיינים משותפים שימצא.

האלגוריתם תחת למידה בלתי מונחית יתבסס על מאגר מידע לא מסומן - מידע שלא קיימות גם ה"תשובות" לבעייתו. לדוגמה, במקרה של מודל לצורך קטלוג הדואר האלקטרוני, האלגוריתם יתבסס על מאגרי מידע הכוללים מיילים קודמים, ועל פי נקודות דמיון ושוני שהוא ימצא הוא יחלק את המייל למספר קבוצות כאשר כל מייל חדש שיקבל יוכנס לקבוצה על פי אותם נקודות דמיון ושוני. כך יכול להיות שהמודל יחלק לקבוצות שונות על פי שולח המייל, או יכול להיות שהמודל יחלק על פי מיילים אשר צורף אליהם קובץ וכאלו שלא, כיוון שהמודל עצמו שהוא שמגדיר את התוצאה, היא לא צפויה.

4.2.3 למידה חלקית מונחית (Semi-supervised)

למידה חלקית מונחית היא למידה בה גם הבעיה וגם התוצאה מוגדרות מראש, אך בניגוד ללמידה מונחית היא משתמשת במאגר מידע בו מעט מידע מסומן והרבה מידע לא מסומן. כמו כן, הוא שונה מלמידה בלתי מונחית בכך שהוא בעל תוצאות אפשריות מוגדרות מראש מה שממעיט את האפשרות לטעות או לחזות את המידע לא בהתאם לצורך.

ניעזר באלגוריתם המשתמש בלמידה חלקית מונחית כאשר אנחנו יודעים את התוצאות האפשריות מראש אך אין לנו מאגר מידע מספיק גדול שכולו מסומן או שיש קושי רב לייצרו.

4.2.4 למידת חיזוק (Reinforcement)

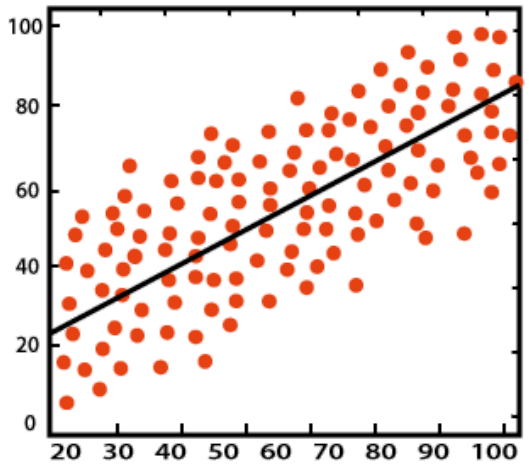
למידת חיזוק היא למידה בה המטרה לאלגוריתם מוגדרת והוא משתמש במידע שזמין לו על מנת לשפר את עבודתו. אופן פעולת האלגוריתם הוא שימוש במידע הזמין לו וניסיון שיפור יעילות פעולתו על ידי קבלת משוב בעת ביצועיו, לפיו הוא מגיע להחלטה של כישלון או הצלחה. למשל, ניקח אלגוריתם המלמד בוט בתוך משחק ללכת. לבוט נתון משטח המהווה רצפה ושתי רגליים - זהו המידע שזמין לו. אותו הבוט, ככל שהוא מצליח ללכת יותר רחוק הוא מקבל ניקוד גבוה יותר, על פי ניקוד זה הוא לומד להשתפר ומשפר את תוצאותיו.

אופן המשוב יכול להיות חיובי (**חיזוק חיובי**) בו האלגוריתם יקבל נקודות כאשר ישפר את אופן פעולתו, ומשוב שלילי (**חיזוק שלילי**) בו האלגוריתם יאבד נקודות כאשר יתרחק מהמטרה. אלגוריתם של למידת חיזוק ישתמש לרוב גם בחיזוק שלילי וגם בחיזוק חיובי. למשל, אלגוריתם המלמד בוט ללכת, ישתמש בחיזוק חיובי בכך שייתן ניקוד גבוה יותר ככל שהבוט הצליח לעבור מרחק רב יותר, ויוריד ניקוד כאשר ייפול או יתרחק (יילך לאחור).

4.3 סוגים שונים של בעיות

ישנם מספר סוגי בעיות עיקריות כאשר מדובר בלמידת מכונה, כגון: רגרסיה (Regression), קלסיפיקציה (Classification) וניתוח אשכולות (Clustering).

4.3.1 רגרסיה (Regression)

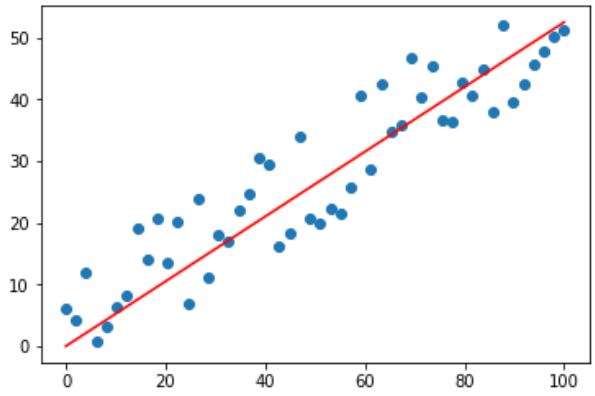


הסוג הראשון של בעיות הוא בעיות רגרסיה (regression), אשר נכלל תחת למידה מונחית (supervised). הרעיון העיקרי של מודל מסוג זה הוא לחזות מידע על ידי שימוש במידע קודם לו.

בבעיית רגרסיה, על האלגוריתם להעריך ולהבין את הקשר שבין המשתנים. שימוש ברגרסיה מתמקד במשתנה תלוי אחד (לצורך העניין נקרא לו כעת Y) ובסדרה של משתנים אחרים (המסמלים סוגי נתונים שונים) - מה שהופך אותו שימושי במיוחד למטרות חיזוי.

לרוב נשתמש במודל רגרסיה כאשר מדובר בבעיה אשר מספר התוצאות האפשריות עבורה הוא אמנם מוגדר אך גדול מאוד. למשל, כאשר עלינו לחזות ערך של משהו על בסיס שוני מאפייניו השונים בעזרת נתוני מכירות של סוג מוצר זה מחודשים עברו (אשר תוצאותיו בטווח מספרי גדול).

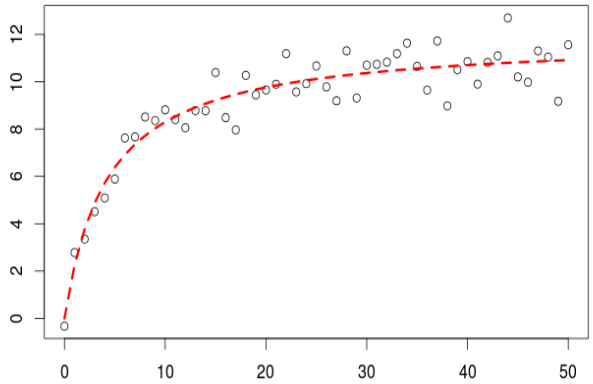
רגרסיה מתחלקת לשני סוגים עיקריים: רגרסיה לינארית ורגרסיה לא לינארית.



רגרסיה לינארית - באה מהמילה לינאר (linear) משמעו ישר. זאת בעקבות שרגרסיה מסוג זה ניתנת לתיאור על ידי פונקציה של קו ישר. דוגמה למודל רגרסיה לינארית להעיל.

ניתן לראות בתמונה נקודות כחולות המסמלות את הנתונים שנאספו, והקו האדום הוא פונקציה המתארת את החיזוי של ערך ה-Y בהתאם לערכי ה-X השונים.

רגרסיה לא לינארית - בשונה מרגרסיה לינארית מהווה פונקציה אשר מתוארת כפונקציה של קו לא ישר, דבר היכול לתרום לדיוק הפונקציה בהתאם לנתונים. דוגמה למודל רגרסיה לא לינארית להעיל.



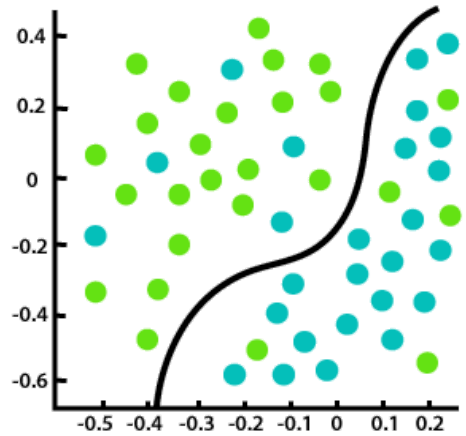
ניתן לראות בתמונה את הנקודות המסמלות את הנתונים שנאספו, והקו האדום הוא פונקציה המתארת את החיזוי של ערך ה-Y בהתאם לערכי ה-X השונים. כפי שניתן לראות הקו לא ישר על מנת להתאים בצורה מירבית לנתונים.

4.3.2 קלסיפיקציה (Classification)

סוג נוסף של בעיות הוא בעיית קלסיפיקציה (Classification), אשר גם כן נכלל תחת למידה מונחית (supervised). הרעיון העיקרי של מודל מסוג זה הוא שיוך מידע לקבוצה אליה הוא שייך על בסיס נקודות דימיון ושוני בינו לקבוצות השונות. בניגוד לרגרסיה מאופיינת במספר תוצאות נמוך.

בבעיית קלסיפיקציה, על האלגוריתם לשייך מידע חדש אל הקבוצה המתאימה לו מתוך הקבוצות המוגדרות לאלגוריתם מראש (מאגר המידע המסומן הקיים לו).

נשתמש במודל קלסיפיקציה כאשר מדובר בבעיה אשר מספר התוצאות האפשריות עבורה הוא אמנם מוגדר אך קטן מאוד, בעצם, כאשר מדובר בשיוך מידע למספר קבוצות קבועות מראש. למשל, הבעיה שתוארה קודם - שיוך המייל לשתי קבוצות אפשריות - ספאם או לא ספאם, בה יש שתי תוצאות אפשריות בלבד.



ניתן לראות בתמונה דוגמה ספציפית למודל קלסיפיקציה של חלוקה לשתי קבוצות. כאשר הנקודות הירוקות הבהירות מסמלות מידע מקבוצה אחת והנקודות הירוקות הכהות מסמלות את המידע מקבוצה אחרת. הקו השחור הוא הפונקציה המחלקת בין חיזוי שתי הקבוצות.

עבור כל אוסף נתונים חדש, אם הנקודה של הנתונים תהיה בצד אחד - הנתון יהיה שייך לקבוצה אחת, ואם מהצד השני של הפונקציה אז יהיה שייך לקבוצה האחרת.

4.3.3 ניתוח אשכולות (Clustering)

לשם כתיבת תת פרק זה, נעזרתי במאמר "Clustering in Machine Learning" של חברת Google.

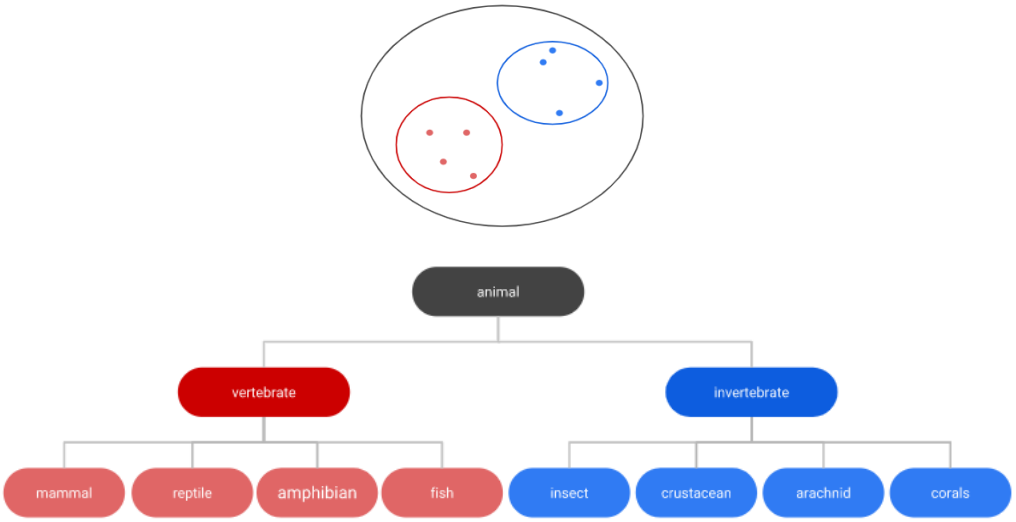
סוג נוסף של בעיות הוא בעיית ניתוח אשכולות (Clustering), אשר נכלל תחת למידה בלתי מונחית (unsupervised). הרעיון העיקרי של מודל מסוג זה הוא חילוק מאגרי המידע הקיימים אצלו לקבוצות על פי נקודות דימיון ושוני שהוא מוצא.

בבעיית ניתוח אשכולות, על האלגוריתם לשייך מידע חדש אל הקבוצה המתאימה לו מתוך הקבוצות שהגדיר לעצמו לפני על פי נקודות דימיון ושוני לקבוצות עצמן.

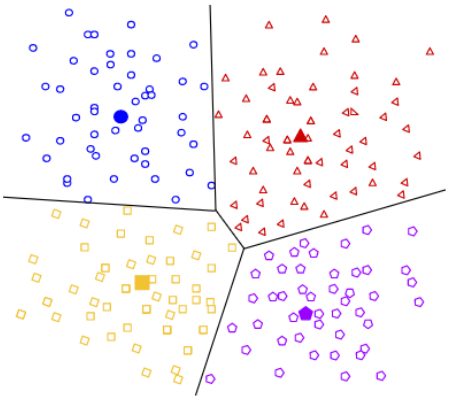
נשתמש במודל ניתוח אשכולות כאשר מדובר בבעיית חילוק מידע לקבוצות כאשר התוצאות האפשריות עבורה לא מוגדרות מראש. למשל, האלגוריתם מקבל אוסף ספרים שונים אשר אותם יחלק על פי מאפיינים דומים ושונים למשל על פי כותב הספר או שנת הוצאת הספר ועוד, וכאשר יקבל ספר חדש ישייך אותו בהתאם לקבוצה היותר דומה עבורו.

ישנן כמה וכמה שיטות שונות של ניתוח אשכולות, ביניהן: מודל היררכי (Hierarchical), מודל מבוסס מרכז כובד (Centroid-based), מודל מבוסס צפיפות (Density-based) ומודל מבוסס הפצה (Distribution-based).

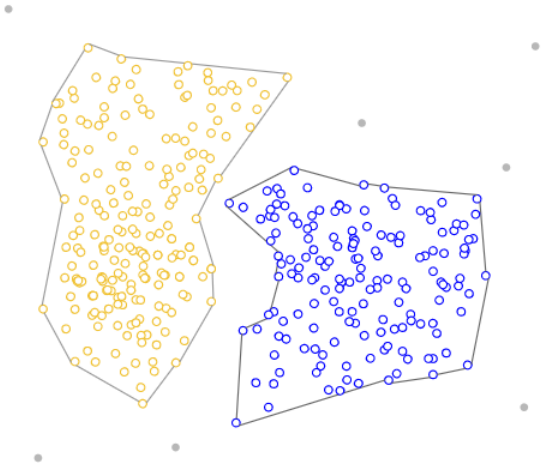
מודל היררכי - בהתאם להגדרת מרחק בין הנתונים השונים, המודל מאתר אשכולות באופן היררכי על פי מרחק הנקודות אחד מהשני ויוצר עץ אשכולות. כך, כל אוסף הנקודות נמצא באשכול גדול אחד, אשר מתחלק למספר אשכולות שונים על פי קרבת נקודות, וכך הלאה, עד מצב שבו גם כל נקודה נחשבת לאשכול עצמאי. כך, על ידי בחירת השלב בעץ, אנו מקבלים את סוגי האשכולות השונים באותה רמה אליה משתייכת הנקודה.



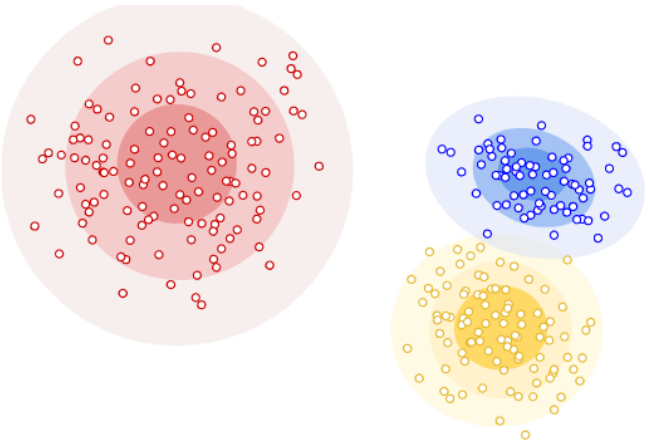
מודל מבוסס מרכז כובד - בשיטה זו עבור כל אשכול מוגדר מרכז כובד שעל פיו מגדיר האלגוריתם את האשכול של כל נתון.



מודל מבוסס צפיפות - בשיטה זו המידע מחולק על פי צפיפות עם נקודות אחרות, אוסף נקודות קרובות הוא אשכול, ונקודה שלא נמצאת בקרבת אף נקודה אחרת לא בתוך אשכול.



מודל מבוסס הפצה - בשיטה זו המודל משתמש במודל מתמטי הפצתי, לדוגמה במודל גאוס. כך ככל שנקודה רחוקה יותר מהמרכז של הפצה כלשהי כך הסיכוי שהיא שייכת אליו קטן ולהיפך.



4.4 פירוט מונחי יסוד

לכתיבת תת פרק זה, נעזרתי בחומרי שיעור של הקורס "Machine Learning" של אוניברסיטת סטנפורד בהנחיית אנדרו נייג.

4.4.1 פונקציית ההיפותזה (hypothesis)

היא בעצם המודל המתמטי המתאר את חיזוי התשובה לשאלה בהתאם לנתונים. בתנאים הנכונים, ככל שיהיו יותר נתונים, כך החיזוי יהיה יותר מדויק. פונקציה זו תסומן בתור פונקציית h(x).

דוגמה ל-hypothesis של רגרסיה לינארית:

$$h_\theta(x) = \theta_0 + \theta_1 x$$

4.4.2 מאפיינים (Features)

המאפיינים הם משתני המודל, בהתאם למשתנים אלו יקרה החיזוי של המודל, חיזוי המודל תלוי בערכים של מאפיינים אלו. לרוב המאפיינים יסומנו באות X, כאשר כל מאפיין יקבל מספר משלו, X0, X1, X2, X3 והלאה..

נסמן באות **n** את כמות המאפיינים. כמו כן, נסמן באות **j** את מספר המאפיין.

למשל, ניקח דוגמה בסיסית בחיזוי ערך בתים שבה יש רק מאפיין אחד שהוא שטח הבית. אם נתאר את מודל הפונקציה כגרף, אז השטח יהווה משתנה של הפונקציה אותו נסמן ב-X1 (המשתנה X0 תמיד יהיה שווה 1), כך בעצם יווצר מודל לחיזוי ערך הבית בתלות בשטחו. לצורך הדגמה:

$$h_\theta(x) = \theta_0 + \theta_1 x$$

ההיפותזה להעיל היא מודל של רגרסיה לינארית, לצורך העניין נגיד ש-X הוא השטח של הבית, כך בעצם נוצר יחס ישיר בין השטח להיפותזה אשר ערכה בהתאם ל-X יהיה חיזוי ערך הבית. אך מה הם שאר הסימנים? זאת נבין בהמשך...

תהליך קביעת והגדרת המאפיינים בצורה שהמחשב יבין וישתמש בהם בצורה האופטימלית ביותר נקרא **Feature Engineering**. חשוב להדגיש שהגדרת מאפיינים נכונים בצורה האופטימלית היא המפתח לכל אלגוריתם בלמידת מכונה. על ידי שינוי המאפיינים ואופן שימושם במודל, פונקציית ההיפותזה משתנה ועם כך משתנים ערכי החיזוי עבורה.

4.4.3 מאגר נתונים (data set)

ברגרסיה נקרא לרוב למאגר נתונים זה **סט אימונים (training set)**. סט אימונים הוא בעצם טבלה בעלת נתונים על קיימים על המאפיינים, כאשר כל שורה מהווה **דוגמת אימון (training example)**.

כאשר נרצה להציג בגרף נקודות מתוך מאגר המידע אז כל נקודה מהווה דוגמת אימון. מספר דוגמאות האימונים תהיה תמיד מסומנת באות **m**, ומספר דוגמת האימון הספציפית תסומן באות i (דוגמת האימון בשורה i). כל דוגמת אימון תיוצג לרוב כך:

$$(x^{(i)}, y^{(i)})$$

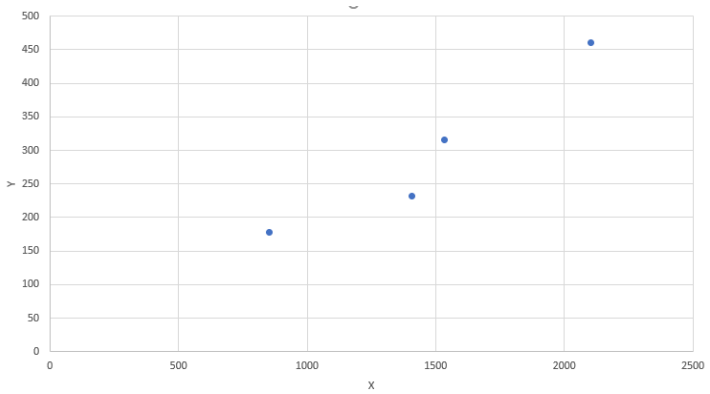
דוגמה לטבלת סט אימונים של רגרסיה לינארית בעלת מאפיין אחד אחד:

Size in feet² (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
…	…

בדוגמה זו ניתן לראות טבלה של סט אימונים מסומן בעבור מודל רגרסיה, בה קיימים נתונים אך ורק למאפיין אחד המסומן ב-X שהוא גודל השטח של בית בריבוע, וערך הבית בהתאם לגודל זה על פי נתונים שנאספו המסומן ב-Y. כל שורה הוא דוגמת אימון, אוסף נתונים עבור מקרה ספציפי שנאסף.

כך למשל ערך ה-X של סט אימונים בעבור i=1 יהיה 2104.

כך הנתונים במקרה זה יראו על צירים:



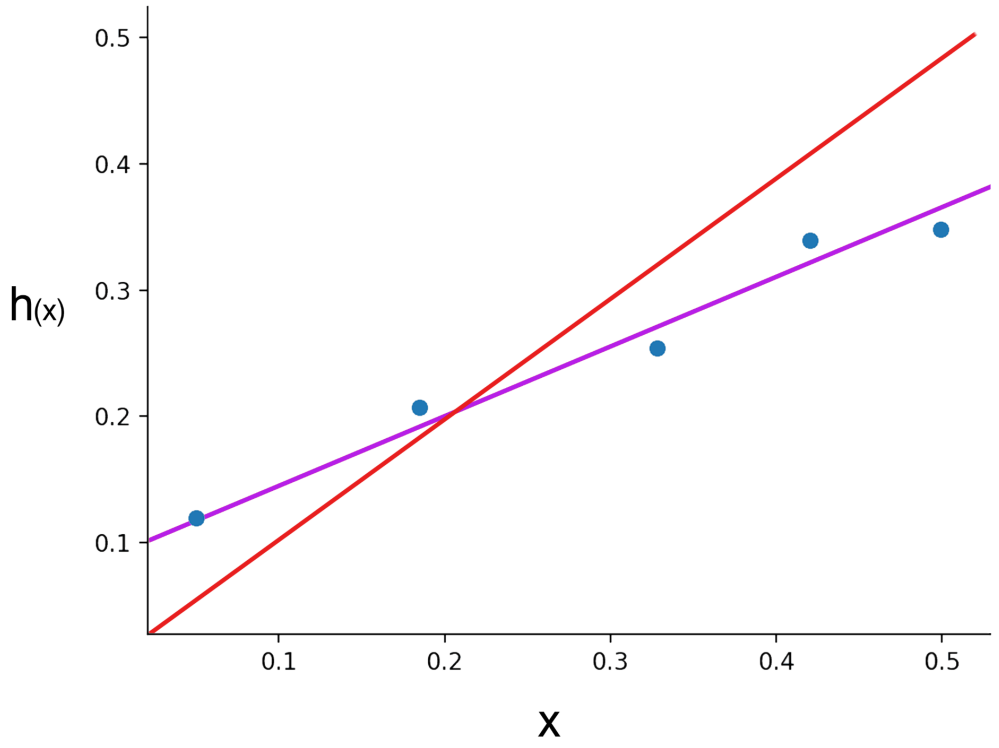
הנקודות הכחולות מסמלות את דוגמאות האימון.

4.4.4 טטה (theta)

טטה הוא קבוע אותו צריך המודל למצוא על ידי תהליך הנקרא אימון(עליו אסביר בהמשך). המודל ברגרסיה לינארית בעצם משלב בין המאפיינים לטטה, ועל ידי מציאת הטטה המתאימים הוא בעצם מגדיר את היחס שבין המאפיינים להיפותזה עצמה. לדוגמה, אם נתאר את ההיפותזה של רגרסיה לינארית עם מאפיין אחד:

$$h_\theta(x) = \theta_0 + \theta_1 x$$

כפי שניתן לראות, יש לנו את ה-h(x) שזו ההיפותזה, יש לנו את ה-X שזהו המאפיין ויש לנו את θ_0 ו-θ_1 אשר אותם המודל ינסה למצוא. כפי שניתן לראות במצב זה θ_0 הוא ערך ה-y בנקודת החיתוך עם ציר ה-y ו-θ_1 זה שיפוע הגרף. על מנת להמחיש את המשמעות של שינוי ה-θ (טטה) אדגים את שינוי הגרף בהתאם לשינוי הערכים בהתאם לדוגמה להעיל.

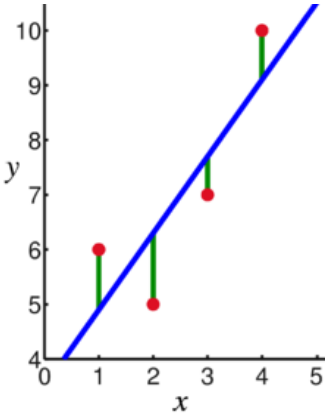


הנקודות הכחולות מסמלות את מאגר הנתונים. כפי שניתן לראות, הגרף האדום והגרף הסגול שניהם מתארים את אותו המודל אך עם ערכים שונים של θ (טטה). הגרף האדום מתאר את המודל כאשר θ_0 שווה ל-0 ו-θ_1 שווה ל-1. כפי שניתן לראות, במצב זה הקו לא מקורב במיוחד לנקודות הכחולות, משמע הערכים לא מתאימים למודל במצב זה. מנגד, ניתן לראות שהגרף הסגול, בו θ_0 שווה ל-0.1 ו-θ_1 שווה

ל-0.5 הרבה יותר מדויק עבור הנקודות הכחולות.

מכך בעצם אפשר להבין כי המטרה הסופית של האלגוריתם למצוא את ה-theta הנכונים על ידי אימון.

4.4.5 פונקצית עלות (Cost function)



פונקציית עלות היא פונקציה מתמטית אשר בעזרתה ניתן לחשב את סכום ההפרשים שבין המודל חיזוי (hypothesis) למאגרי המידע (datasets). הפונקצייה לרוב מסומנת באות האנגלית J.

לרוב האלגוריתם ינסה לגרום לערך פונקציה זו להיות כמה שיותר קטן בשביל למצוא את ה-θ (טטה) האופטימלי ברגרסיה. ככל שערך פונקציה זו קטן יותר משמעו שההפרש בין פונקציית החיזוי לנתונים קטנה יותר משמע המודל מתאים יותר ויותר עבור הנתונים.

הנוסחה של פונקציית העלות:

$$\boxed{J(\theta_0,\theta_1)} = \frac{1}{2m}\boxed{\sum_{i=1}^{m}}\boxed{\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2}$$

$$\underset{\theta_0,\theta_1}{\text{minimize}}\ J(\theta_0,\theta_1)$$

נפרק את הנוסחה לחלקים.

- **מסגרת אדומה** - מסמן כי הפונקציה מסומנת ב-J ותלוייה בערכי ה-θ (טטה) המשתנים מפעם לפעם. הפונקציה מחזירה ערכים בהתאם ל-θ (טטה).
- **האות m** - האות m מסמלת את מספר הנקודות של מאגר המידע, מספר ה-datasets.
- **האות i** - האות i מסמלת את מספר דוגמת האימון הנוכחית, נקרא לאות זו משתנה פנימי עבור מעבר על כלל הנקודות, על כלל דוגמאות האימון.
- **מסגרת ירוקה** - פה ניתן לראות את האות סיגמא (Sigma) המסמלת סכום, מתחתיה מוגדרת האות i שמתחילה מהערך 1, ומלמעלה מוגדר הערך האחרון של i שהוא m. הסיגמה מסמלת שצריך את סכום הביטוי שאחריה (מה שמסומן במסגרת הכחולה) עבור כל ערך של i אשר בין 1 ל-m בקפיצות של אחד.
- **מסגרת כחולה** - בתוך הסוגריים ניתן ההפרש בין ערך ה-hypothesis עבור ה-X של נקודה מספר i, לבין ערך ה-y של אותה נקודה. הביטוי בסוגריים בריבוע על מנת שהתוצאה תהיה

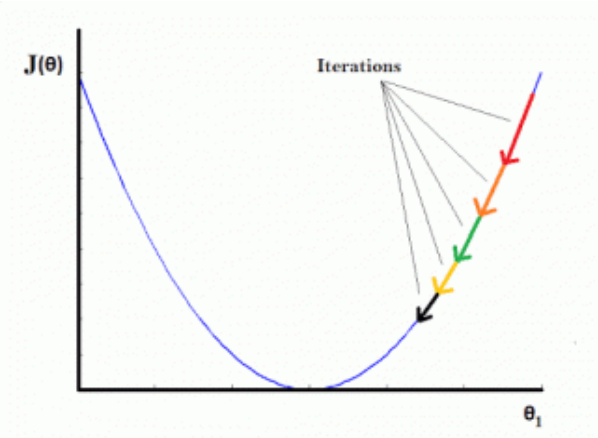
חיובית. עבור המודל לא חשוב לאיזה צד ההפרש, עם מודל החיזוי מעל לנקודה או מתחתיה לכן נעלה את ההפרש בריבוע לצורך הסרת שליליות.

- כך אנו מבינים שערך פונקציית העלות מחושב בתור סכום ההפרשים בריבוע בין ערך ההיפותזה בעבור X דוגמאות האימון לבין ערך ה-Y של אותן נקודות כפול חצי חלקי m (כמות דוגמאות האימון). המטרה היא למצוא את הטטה בעבורו ערך פונקציית העלות מינימלית.

4.4.6 אימון - תהליך בו המודל "לומד", במהלך האימון מנסה המודל למצוא את ה-θ (טטה) האופטימלי. לצורך האימון יש מספר שיטות שונות כיצד למצוא את ה-θ. אנחנו נתמקד ברגרסיה, לכן נדבר על שיטות מציאת ה-θ השונות בסוג בעיות זה.

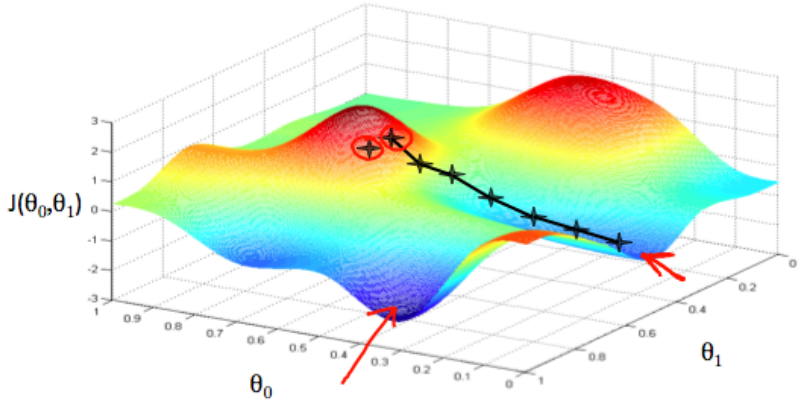
4.4.7 מורד הגרדיאנט (Gradient Descent)

בשיטה זו, המודל מנסה להגיע בצעדים קטנים (באיטרציות - Iterations) לעבר נקודת הקיצון המינימום של פונקציית העלות. ננסה להבין זאת בצורה מוחשית יותר. ניקח לדוגמה מצב של מורד גרדיאנט בעבור טטה אחד בלבד. כאשר אנו מסתכלים על גרף פונקציית העלות בתלות בטטה, נראה מצב זה:



במצב זה, ניתן לראות כי בכל איטרציה המודל מתקרב יותר ויותר לעבר נקודת הקיצון המינימום. אך מדוע המודל שואף לנקודה זו? השאיפה לנקודה זו נובעת מהסיבה שנקודה זו היא בעצם הנקודה שמסמלת את המקום עבורו פונקציית העלות מינימלית, וכפי שניתן לזכור, ככל שערך פונקציית העלות קטן יותר, כך המודל מדויק יותר בעבור מאגר הנתונים.

אך זהו מצב די מופשט, בואו ננסה לראות מצב מסובך קצת יותר אשר יותר דומה למצב איתו אתמודד.



במצב זה, ניתן לראות הצגה מרחבית של פונקציית העלות כתלות בשני טטה: טטה אפס וטטה אחת. בהצגה זו, פונקציית העלות נראית כיריעת בד. ככל שאזור בצבע חם יותר כך הוא גבוה יותר, ולהפך, ככל שאזור בצבע קר יותר הוא מסמל אזור נמוך יותר. החצים האדומים מצביעים על נקודות המינימום של הפונקציה, כפי שניתן לראות יש יותר מאחת. כמו כן, ניתן לראות בשחור את הצעדים שמבצע המודל. כל איקס שחור מסמל נקודה בה נמצא המודל בכל איטרציה. עם כל איטרציה עובר מתקדם המודל לעבר נקודת המינימום. אך כיצד בעצם מתבצעות האיטרציות? בואו נבין את נוסחת המורד הגרדיאנט. זוהי נוסחת הגרדיאנט בעבור מאפיינים מרובים, אך היא פועלת גם בעבור מאפיין אחד.

$$\text{repeat until convergence: } \{$$
$$\theta_j := \theta_j - \alpha \boxed{\frac{1}{m} \boxed{\sum_{i=1}^{m}} \boxed{(h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}}} \qquad \boxed{\text{for j := 0...n}}$$
$$\}$$

נזכור, כי j מסמל את מספר המאפיין, m את כמות הדוגמאות האימון, ו-i את מספר דוגמת האימון.

מסגרת אדומה - מסמן כי הפעולה כולה מתבצעת עבור כל j מ-0 ועד n (מספר המאפיינים, כאשר לכל מאפיין טטה משלו).

מסגרת ירוקה - פה ניתן לראות את האות סיגמא (Sigma) המסמלת סכום, מתחתיה מוגדרת האות i שמתחילה מהערך 1, ומלמעלה מוגדר הערך האחרון של i שהוא m. הסיגמה מסמלת שצריך את סכום הביטוי שאחריה (מה שמסומן במסגרת הכחולה) עבור כל ערך של i אשר בין 1 ל-m בקפיצות של אחד.

מסגרת כחולה - בתוך הסוגריים ניתן ההפרש בין ערך ה-hypothesis עבור ערכי המאפיינים של דוגמת אימון מספר i, לבין ערך ה-y של אותה נקודה. ההפרש בסוגריים מוכפל בערך מאפיין ה-j של דוגמת האימון מספר ה-i.

האות α (אלפא) - מסמל את קצב הלמידה, מספר קבוע הנבחר על ידי המתכנת. קצב למידה גבוה יותר יגרום לצעדים גדולים יותר, קצב למידה נמוך מדי יגרום לפעולה איטית יותר של האלגוריתם. אחד הקשיים הוא למצוא את קצב הלמידה האופטימלי. על כך אסביר בהמשך.

מסגרת סגולה - הביטוי בתוך מסגרת זו הוא שווה ערך לשיפוע פונקציית העלות. על ידי כך שהשיפוע יכול להיות חיובי או שלילי, הצעד מתבצע לכיוון נקודת המינימום מכיוון שמשמאל לנקודת המינימום הפונקציה בירידה (שיפוע שלילי) לכן ה-θ (טטה) צריך לגדול, ומימין לנקודת המינימום הפונקציה עולה (שיפוע חיובי) לכן ה-θ (טטה) צריך לקטון. בעקבות זאת, צעד האיטרציה תמיד יהיה לכיוון נקודת המינימום.

משמעות הביטוי כולו - בעבור כל איטרציה, עבור כל θ (טטה), ה-θ שווה לאותו ה-θ פחות קצב הלמידה חלקי m כפול סכום ההפרשים בעבור כלל דוגמאות האימון כפול אותם ערך המאפיין של אותו ה-θ בעבור כל דוגמת אימון. האיטרציות מתבצעות עד להגעה לנקודת מינימום.

יש מספר דרכים שבאמצעותם ניתן לייעל מורד גרדיאנט. ביניהן Feature Scaling ו-Mean Normalization.

Feature Scaling and Mean Normalization

על מנת להאיץ את המורד הגרדיאנט, אנו יכולים לייעל את הנתונים שלנו בצורה שיהיה מהיר יותר לעבוד איתם על ידי הגבלתם בתוך טווח מספרים משותף קטן יותר. הסיבה לכך היא ש-θ יירד במהירות בטווחים קטנים ולאט יותר בטווחים גדולים, וכך יתנדנד בצורה לא יעילה עד לנקודת המינימום כאשר המשתנים מאוד לא אחידים.

הדרך למנוע זאת היא שימוש הגבלת הנתונים בעבור כל מאפיין בטווח. באופן אידיאלי בטווח מספרים בין 1- ל-1 או 0.5- ל-0.5. טווח זה הוא לא חובה, המטרה שלנו היא רק להכניס את כל הנתונים לטווח אחד לצורך ייעול.

$$-1 \le x_{(i)} \le 1$$

$$-0.5 \le x_{(i)} \le 0.5$$

שתי השיטות בהן נשתמש על מנת להכניס את הנתונים לטווחים אלו הם Feature Scaling ו-Mean Normalization. ב-Feature Scaling נחלק את הנתונים של כל מאפיין בטווחו (הערך המקסימלי פחות המינימלי), כתוצאה מכך ייוצרו לנו מספרים בטווח של 1. ב-Mean Normalization נחסיר את הממוצע של כל מאפיין על מנת שיווצר טווח חדש בו הממוצע החדש הוא אפס בעבור על מאפיין. נממש את שתי השיטות בצורה הבאה:

$$x_i := \frac{x_i - \mu_i}{s_i}$$

μ_i - מסמל את הממוצע

s_i - מסמל את הטווח

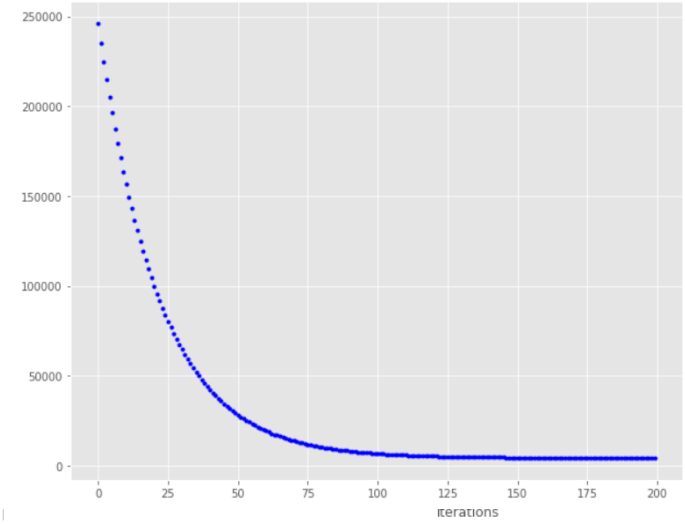
המאפיין שווה למאפיין פחות הממוצע של אותו מאפיין חלקי טווחו (ההפרש שבין הערך המקסימלי למינימלי).

דיבוג של מורד גרדיאנט

אחת שיטות הדיבוג של מורד גרדיאנט על מנת לברר כמה איטרציות צריך האלגוריתם לבצע, הוא ליצור גרף של פונקציית העלות כתלות במספר האיטרציות. הגרף יציג לנו מספר דברים:

א. האם ערך פונקציית העלות יורד ככל שיש יותר איטרציות? כיוון שבאופן פעולה תקין, ערך פונקציית העלות צריך לרדת עם כל איטרציה.

ב. את מספר האיטרציות אחריו ניתן להפסיק כיוון שערך פונקציית העלות אינו משתנה או אינו משתנה משמעותית.



ישנה דרך למצוא מתי ניתן לעצור את האיטרציות באופן אוטומטי, והיא הגדרת הגעה לערך פונקצית עלות מינימלי כאשר ערך פונקציית העלות קטן בפחות ממספר מוגדר מראש E (מספר קטן מאוד, למשל 0.001) באיטרציה מסויימת. אך, הבעיה העיקרית בשיטה זו היא הגדרת אותו מספר, כיוון שהוא שונה עם כל סיטואציה.

במידה ומורד הגרדיאנט לא עובד באופן תקין, מה עלולה להיות הבעיה? התשובה היא קצב למידה מוגדר לא באופן תקין.

קצב למידה (Learning rate)

קצב הלמידה מסומן באות α (אלפא). קצב הלמידה הוא מספר קבוע הנבחר על ידי המתכנת. קצב למידה גבוה יותר יגרום לצעדים גדולים יותר, קצב למידה נמוך מדי יגרום לפעולה איטית יותר של האלגוריתם. כפי שהסברתי לפני כן, אחד הקשיים הוא למצוא את קצב הלמידה האופטימלי.

במצב בו קצב הלמידה גבוה מדי, עלול להווצר מצב בו ערך פונקציית העלות עם כל איטרציה יגדל במקום להקטן או יגדל ויקטן ויגדל ויקטן הלוך חזור. במצבים אלו עלינו להקטין את קצב הלמידה. אך

מצד שני, איננו רוצים שקצב הלמידה יהיה נמוך מדי, כיוון שכאשר קצב הלמידה נמוך מדי, אז ייקח למורד הגרדיאנט זמן הרבה יותר גדול להגעה לערך פונקציית עלות מינימלי.

4.4.8 Normal Equation

מלבד מורד גרדיאנט יש עוד שיטות רבות למציאת ה-θ (טטה), אחת מהן היא Normal Equation. בשיטה זו, על ידי ביטוי אחד, אנו יכולים למצוא את כל ה-θ. ביטוי זה משתמש באלגברה לינארית.

θ - וקטור הטטה

X - מטריצת המאפיינים

y - וקטור ערכי התוצאה של כל דוגמאות האימונים

$$\Theta = (X^T X)^{-1} X^T y$$

שיטה זו, לעומת Gradient Descent לא דורשת Feature Scaling ו-Mean Normalization.

לשיטה זו היתרונות והחסרונות שלה. על מנת להבין מתי להשתמש בכל אחת מן השיטות נבצע השוואה.

4.4.9 השוואה בין Normal Equation ו-Gradient Descent

Normal Equation	Gradient Descent
אין צורך בבחירת קצב למידה (α).	יש לבחור קצב למידה (α).
אין צורך בביצוע איטרציות.	יש צורך באיטרציות רבות.
יש צורך לחשב את $X^T X$, חישוב אשר נתון בסיבוכיות $O(N^3)$. כתוצאה מכך, לוקח זמן רב אם כמות המאפיינים גדולה.	עובד בצורה טובה מאוד אפילו כאשר כמות המאפיינים גדולה מאוד.

מספר גדול במקרה זה אינו 100, ואף אינו 1000. בימים שלנו למחשב לא ייקח זמן רב לחישוב כאשר N שווה למספרים אלו. מדובר במקרים בו כמות המאפיינים הוא למשל 10^6.

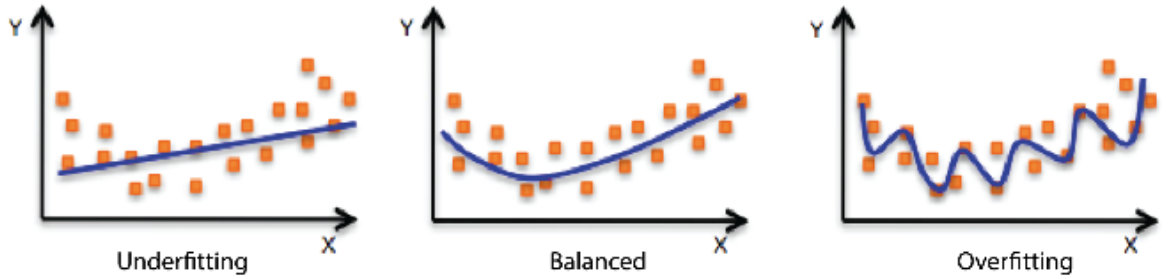
4.4.10 התאמת יתר/התאמת חסר

כאשר מדובר בבעית רגרסיה ישנם כמה בעיות נפוצות. כפי שכבר הסברתי, המטרה הסופית בעת יצירת מודל, היא ליצור מודל המקורב ככל היותר לסט האימונים אותו הוא מקבל. אך כתוצאה מכך עלולה להווצר בעיה בשם **התאמת יתר (Overfitting)**.

לרוב כאשר מדובר במאגר נתונים גדול בעל מאפיינים רבים אם המודל ינסה לדייק ככל היותר עבור דוגמאות אימון קיימות, הדבר עלול לפגוע ביכולת חיזוי המודל - בעיה זו נקראת **התאמת יתר**. כפי שהשם מרמז, מדובר במצב בו המודל מתאים לסט האימונים יתר על המידה. במצב זה המודל עלול להציג תמונה כללית לא נכונה, למשל אם בסט האימונים נכללו **נתוני 'רעש'** - נתונים היוצאים מן הכלל ומבטאים יותר סיכוי קלוש מאשר את האמת. כך שבסופו של דבר היא לא תתאים עבור נתונים חדשים. על כן, נרצה להימנע מדיוק יתר על המידה על הנתונים הקיימים, ולנסות ליצור מודל מופשט יותר.

מצד שני, עלול לקרות מצב הפוך, בו המודל מופשט מדי עבור סט האימונים, מצב זה נקרא **התאמת חסר (Underfitting)**. מצב זה גם כן יפגע ביכולת החיזוי של המודל בכך שיתאר מצב אידיאלי מדי על כן לא אמיתי ולא מתאים בעבור הנתונים הקיימים.

למשל במקרה שלנו, בעת חיזוי ערך נדל"ן, יהיה מדובר במאגר נתונים גדול בעל מאפיינים רבים, בעל ערכים מאוד שונים מפעם לפעם. אם המודל ינסה לדייק ככל היותר עבור דוגמאות האימון הקיימות, עלול להיווצר מצב של התאמת יתר. מצד שני, אם ננסה לחזות כל ידי רגרסיה לינארית, הדבר עלול להוביל לחיזוי לא מציאותי עבור הנתונים. להלן דוגמה המציגה את העיקרון בצורה מוחשית:



כפי שניתן להבין, אחת המטרות של המודל היא למצוא את עמק השווה, למצוא מצב מדויק ככל האפשר בעבור נתונים קיימים, ללא פגיעה ביכולת החיזוי.

המודל צריך לגבור על הקושי העיקרי במודלים המשתמשים במאפיינים רבים כמו במצב של חיזוי ערך של נדל"ן. מצב בו מספר רב מדי של מאפיינים עלול לגרום לסיכוי רב יותר של הופעת התאמת יתר, אותה ניתן למנוע במספר דרכים. כאשר אנו מנסים לפתור בעיית התאמת יתר, אנחנו יכולים ללכת לשתי גישות עיקריות: הוצאת מאפיינים, וביצוע רגולריזציה (Regularization) עבורם.

הוצאת מאפיינים

הוצאת מאפיינים יכולה להתבצע בשתי דרכים:

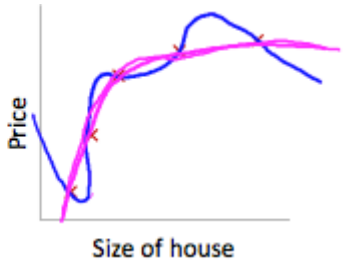
באופן ידני - יוצר המודל עצמו בוחר באילו מאפיינים להשתמש ובאילו לא, זאת ועוד נעשה כחלק מתהליך הנקרא Feature Engineering.

על ידי שימוש באלגוריתם לבחירת מודל - עליו אפרט בהמשך העבודה.

ביצוע רגולריזציה (Regularization)

בשיטה זו ניתן לשמור על כלל המאפיינים אך לשנות את חשיבותם. השיטה עובדת היטב כאשר יש שימוש במאפיינים רבים, כאשר כל אחד תורם במעט לחיזוי ערך ה-y.

רגולריזציה פועלת לפי העיקרון שערכים קטנים יותר של טטה גורמים להיפותזה "מופשטת" יותר אשר פחות נוטה לפיתוח בעיית התאמת יתר.



על מנת לבצע רגולריזציה, נשנה את פונקציית העלות שלנו כדי שתראה כך:

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \boxed{\lambda \sum_{j=1}^{n}\theta_j^2}\right]$$

כאשר החלק אשר נוסף (המסומן במסגרת כחולה) נקרא **ביטוי רגולריזציה** ומטרתו להקטין את ערכי הטטה על מנת לבצע רגולריזציה.

האות λ (למבדא) - האות λ אשר נמצאת במסגרת הסגולה, היא פרמטר רגולריזציה. מספר גדול מאוד שמטרתו להקטין את ההשפעה של הטטה ובכך להפוך את ההיפותזה ל-"פשוטה" יותר.

אסור שהערך של λ יהיה גדול מדי כיוון שאם יהיה גדול מדי, אז ערכי הטטה ישאפו ל-0 מה שיגרום לכך שהשפעת המאפיינים תהיה מינימלית וההיפותזה תהיה שווה אך ורק לטטה מספר 0.

רגולריזציה במורד הגרדיאנט (Gradient Descent)

הנוסחה המחודשת:

$$
\begin{aligned}
&\text{Repeat } \{ \\
&\quad \theta_0 := \theta_0 - \alpha \ \frac{1}{m} \ \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)} \\
&\quad \theta_j := \theta_j - \alpha \left[\left(\frac{1}{m} \ \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \right) \boxed{+ \frac{\lambda}{m} \theta_j} \right] \qquad j \in \{1, 2 ... n\} \\
&\}
\end{aligned}
$$

כפי שניתן לראות, הנוסחה השתנתה במעט, נוסף ביטוי (במסגרת הכחולה). כפי שניתן לראות, עבור טטה מספר 0 לא השתנתה הנוסחה, לא נבצע רגולריזציה על טטה 0.

על ידי מניפולציה על הנוסחה, הנוסחה תראה כך בעבור כל טטה שהיא לא טטה 0:

$$\theta_j := \theta_j \boxed{(1 - \alpha \frac{\lambda}{m})} - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

הביטוי במסגרת הסגולה תמיד יהיה קטן מ1 וכך בעצם תתבצע הקטנתו הנוספת.

רגולריזציה ב-Normal Equation

הנוסחה המחודשת:

$$\theta = \left(X^T X + \lambda \cdot L \right)^{-1} X^T y$$

$$\text{where } \ L = \begin{bmatrix} 0 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix}$$

L - היא מטריצה שבה 0 בפינה השמאלית למעלה, כאשר המשך האלכסון הוא הספרות 1, ושאר האיברים במטריצה הם 0.

4.4.11 כיצד לשפר את ההיפותזה?

זהו הנושא הסופי בפרק זה. חלק זה יעסוק בחלק בלתי נפרד מהעבודה כולה, והוא שיפור המודל.

כעת נדון בדיבוג מודל למידת מכונה. למשל, במצב שבו אנחנו מבצעים רגולריזציה של מודל רגרסיה, והוא לא עובד כפי שצריך.

במצב זה, אנו יכולים לנסות מספר דברים, למשל, להשיג יותר דוגמאות אימון, לנסות להפחית את מספר המאפיינים, או אולי להפך, להוסיף מאפיינים נוספים. כמו כן, אנו יכולים לנסות להוסיף מאפיינים פולינומים, להקטין את פרמטר הרגולרזיציה ולהגדילו. כל אחת מאופציות אלו יכולה לעזור, אך כיצד לבחור באיזו לנסות? ובכן, יש שיטה אשר יכולה לעזור לנו לפחות להוריד חלק ממקרים אלו.

על מנת לאבחן את ההיפותזה שלנו, ניקח את מאגר הנתונים שלנו, ונחלק אותו לשני חלקים: **סט אימון (Training set)**, **וסט בדיקה (Test set)**. כ-70% מהנתונים יהוו חלק מסט האימון, וכ-30% יהוו חלק מסט הבדיקה. לפני החילוק נדאג שהנתונים שאספנו מאורגנים באופן רנדומלי, על מנת לשפר את אמינות הבדיקה.

כעת, נמצא את ה-θ (טטה) בעבור סט האימון, ולאחר מכן נחשב את ערך פונקציית העלות בעבור סט האימון וה-θ שמצאנו. לאחר מכן, ניקח את אותו ה-θ ונחשב את ערך פונקציית העלות בעבור ה-θ שמצאנו קודם. באופן אידיאלי, ערכי שתי בדיקות פונקציית העלות יהיו נמוכים וקרובים אחד לשני, מכיוון שמצב בו הערך נמוך בעבור סט האימון אך גבוה בעבור סט הבדיקה, מהווה סימן לקח שישנה התאמת יתר של המודל כלפי סט האימון מה שפוגע בחיזויו בעבור נתונים חדשים.

אך נובעת השאלה, כיצד למצוא את המודל הטוב ביותר? כיצד להבין מודל באיזה חזקה נרצה בעבור חיזוי מדויק ככל האפשר עבור מידע נתון ומידע חדש. אחת השיטות הנפוצות לכך, אשר בין היתר פותרת בעיית התאמת יתר היא אלגוריתם לבחירת מודל (Model Selection Algorithm).

אלגוריתם לבחירת מודל (Model Selection Algorithm)

בשיטה זו, תחילה נחלק את המאגר נתונים (dataset) לשלושה חלקים במקום שניים(בהמשך אסביר מדוע): **סט אימון (Training set)**, **סט הצלבה (Cross Validation set)** **וסט בדיקה (Test set)**.

סט אימון כ-60%, סט הצלבה כ-20% וסט בדיקה כ-20%.

1. $h_\theta(x) = \theta_0 + \theta_1 x$
2. $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$
3. $h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_3 x^3$

$\vdots$

10. $h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_{10} x^{10}$

ניקח פונקציות היפותזה מכל מעלה החל ממעלה ראשונה, ונמצא בעבורם את ה-θ תוך שימוש אך ורק בסט האימון.

לאחר מכן, נחשב את ערך פונקציית העלות של כל היפותזה בעבור סט ההצלבה וה-θ שמצאנו. נבחר את ההיפותזה בעלת ערך פונקציית העלות המינימלית במקרה זה. כעת נוכל **למדוד את השגיאה** באופן מציאותי של המודל על ידי חישוב ערך פונקציית העלות בעבור סט הבדיקה וה-θ שמצאנו.

לרוב, השגיאה שנמדוד בעבור סט ההצלבה תהיה קטנה מזו שבעבור סט הבדיקה כיוון שמדידת השגיאה בסט ההצלבה תלוייה בפרמטר נוסף, למשל בדוגמה מלפני הפרמטר הנוסף הוא החזקה של המודל. זו הסיבה שחילקנו את מאגר הנתונים לשלושה חלקים. במידה והיינו מחלקים את הנתונים לשני חלקים של אימון ובדיקה, ובוחרים את המודל שבעבורו ערך פונקציית העלות הקטן ביותר בעבור סט הבדיקה, היינו מקבלים תמונה יותר אידיאלית מהמציאות כיוון שבחרנו בכוונה את המודל בעל אחוז השגיאה הנמוך ביותר בעבור מאגר המידע שלנו. כתוצאה מכך, היינו מקבלים תוצאה נמוכה יותר מהמציאות. כאשר אנו בוחרים את המודל לפי הקבוצה השנייה - ההצלבה, אך מודדים את השגיאה בעבור סט נתונים חדש, כך מתקבלת שגיאה יותר מציאותית.

4.5 המודל שלי

מטרת המודל שלי היא חיזוי ערך הנדל"ן בארה"ב. כפי שניתן להבין המטרה היא חיזוי ערך - הסימון שלנו, והתוצאות האפשריות הן בטווח כל המספרים החיוביים (ערך מספרי).

גם הבעיה וגם התוצאות האפשריות מוגדרות לכן מדובר בבעיה הנמצאת תחת למידה מונחית (Supervised). כמו כן, התוצאות האפשריות הן טווח מספרי גדול ולא מספר אפשרויות מצומצם, לכן, מדובר בבעיית regression.

כעת, כאשר הבנו את סוג הבעיה, עלינו להבין יותר לעומק איזה סוג רגרסיה תתאים יותר לפתרון הבעיה. זאת אבין מהתבוננות על הנתונים וכמותם, אך סביר להניח כי יהיה מדובר ברגרסיה לינארית (Linear Regression).

אמנם מודל של non-liner regression באופן פוטנציאלי יכול להיות הרבה יותר מדויק ביחס לדוגמאות האימון הרבות השונות, אך כיוון שאיננו רוצים להגיע למצב של התאמת יתר, מודל של רגרסיה לינארית יתאים לנו יותר. על אף שניתן לומר שמודל לינארי מתאר מצב אידיאלי של יחס אידיאלי בין המחיר למאפיינים, אך ברצוננו לא להתאים את המודל באופן אידיאלי למאגר המידע הנתון, אלא לערכים עבורם יצטרך לחזות את המחיר. המודל לעולם לא יחזה באופן מדויק לגמרי עבור, אך הוא יתן תמונה הדומה ככל האפשר. על מנת לאפשר למודל לחזות באופן מדויק ככל שאפשר, נשתמש ברגרסיה לינארית עם מספר מאפיינים רב.

מטרתי לבדוק את השימוש בשיטות שונות למציאת הטטה האופטימלי - Normal Equation ו-Gradient Descent, ולממש אלגוריתם למציאת מודל - גם אם רק לצורך בדיקה עבור המאפיינים שיהיו לי.

פרק חמש - מבוא לנדל"ן

לשם כתיבת פרק זה, התייעצתי עם מנכל חברת 8Estate, סער ליטמנוביץ', העוסק בנדל"ן בארה"ב.

המודל אותו אבנה קשור מאוד בתחום הנדל"ן, הרי בסופו של דבר המודל יצטרך לחזות מחיר של נדל"ן על פי מגוון נתונים. כפי שתיארתי בפרק הקודם, המודל משתמש במאפיינים שונים, ובסופו של דבר אשתמש במאפיינים לפי מאגרי נתונים אותם אוכל למצוא באינטרנט.

כאשר אנו מדברים על נדל"ן, אנחנו יכולים להתייחס למגוון נתונים. במודל שלי, אנסה ליצור איזון בין הכמות לאיכות. מצד אחד, מספר מאפיינים גדול מדי עלול לגרום למודל לעבוד באופן איטי יותר, כאשר מצד שני, אם המאפיין ישפר את חיזוי המודל אז אעדיף כן להתייחס אליו.

בסופו של דבר, הכל מסתכם באילו ובכמה מאגרי מידע אצליח לאסוף, לכן בשלב זה של העבודה איני יכול לקבוע את המיקום הגיאוגרפי הספציפי עבורו יחזה המודל, זאת אעשה בשלב מציאת המידע. מטרת הפרויקט היא ליצור קוד עבור מודל שיוכל לחזות כמעט עבור כל מידע שיקבל בצורה הטובה ביותר. לכן, המיקום הספציפי עבורו יחזה המודל, הוא לא הגורם החשוב בלמידה עצמה במקרה זה.

בשביל ליצור תמונה כללית ונכונה יותר עליי להבין אילו קריטריונים חשובים בנדל"ן.

בין המאפיינים יש לנו, מאפיינים של הנדל"ן עצמו, ומאפיינים של הסביבה. המאפיינים של הנדל"ן עצמו לא משתנים במיוחד מאזור לאזור, לעומת הגורמים הסביבתיים. במודל אנסה לשלב גם אלו וגם אלו אך ההחלטה הסופית בעניין זה תתבצע בחלק המעשי.

מאפיינים של הנדל"ן עצמו אלו מאפיינים כגון: שטח הנדל"ן, מספר חדרים, מספר קומות, מספר חדרי שינה, האם יש גינה, האם יש בריכה ועוד.

מאפיינים של הסביבה יכולים להיות: מספר בתי ספר קרובים ואיכותם, האם יש קרבה לאזורי תיירות חשובים, קרבה לחנויות, קרבה למסעדות ופנאי, כמות הפשע באזור, קרבה למרכז העיר, אחוז המסים של אותו אזור וכדומה.

בארה"ב קיימים רישומים רשמיים של הנדל"ן הנמכר אשר ניתן להסתכל בהם באינטרנט. למשל, ניקח לצורך דוגמה את אתר משרד האוצר של עיריית ניו-יורק בו קיימים רישומים רשמיים אשר מתעדכנים כל שנה בנוגע לנדל"ן שנמכר באותה שנה. לכן, אוכל אולי להשתמש במידע זה לצורך המודל.

- חלק מעשי -

1. כתיבת האב-טיפוס ב-Matlab

1.1 הסיבה לכתיבת האב-טיפוס

לאחר שצברתי את הידע התיאורטי, נשאלת השאלה: מדוע לכתוב פרוטוטייפ ב-Matlab?

אב-טיפוס ב-Matlab יאפשר לי להבין טוב יותר את החישובים אותם אבצע לאורך עבודת חקר זו - ביישום המודל בקוד הפייתון, על ידי מעבר ממתמטיקה לכתיבת קוד מתמטי. כמו כן, כתיבת אב-טיפוס זה בשפה זו מאפשרת לי להתנסות טוב יותר בשפה אשר השימוש בה נפוץ מאוד בתחום ה-Data Science. בנוסף, שפה זו שמה דגש על חישובים מתמטיים - חלק שחשוב לי מאוד בפרוייקט זה, ההבנה המתמטית של למידת מכונה.

מטרתי בחלק זה היא להבין טוב יותר את הנוסחאות של השיטות השונות ודברים נוספים הנעשים על ידי חישובי מטריצות ווקטורים.

1.2 הצגת כתיבת השיטות וכדומה באופן מתמטי ב-Matlab

- **ההיפותזה**

הקוד:

```
function h = hypothesis(X, theta)
%HYPOTHESIS Compute the hypothesis for linear regression with multiple variables

h = sum(X'*theta);
end
```

הפעולה מחשבת את הערך ההיפותזה של מודלי רגרסיה. היא מקבלת את הפרמטרים הבאים:

- X - וקטור של מאפיינים עבור דוגמת אימון ספציפית
- theta - וקטור של טטה.

היא מבצעת להם הכפלה ולוקחת את סכום התאים מוקטור התוצאה. נשתמש במכפלת מטריצות - דבר היעיל יותר מביצוע לולאות מיותרות.

- **Cost Function**

הקוד:

```
function J = computeCost(X, y, theta)
%COMPUTECOST Compute cost for linear regression with multiple variables
%   J = COMPUTECOST(X, y, theta) computes the cost of using theta as the
%   parameter for linear regression to fit the data points in X and y

% Initialize some useful values
m = length(y); % number of training examples

J = sum(((X*theta)-y).^2)/(2*m);
end
```

הפעולה מוצאת את ערך פונקציית העלות. היא מקבלת את הפרמטרים הבאים:

- X - מטריצה של מאפיינים
- y - סימון דוגמאות האימון, הערך אותו אנחנו מנסים לחזות בפועל
- theta - ווקטור של טטה

הפונקציה מבצעת כפל בין מטריצת המאפיינם ווקטור הטטה, כך היא מקבלת וקטור ההיפותזה עבור כלל דומאות האימון, מכך היא מחסירה את וקטור ה-y ומעלה את הוקטור בחזקת שתיים. לבסוף היא לוקחת את הסכום של כלל התאים ומחלקת ב2 כפול מספר דוגמאות האימון. לבסוף, היא מחזירה את ערך העלות שקיבלה (J).

- **Gradient Descent**

הקוד:

```
function [theta, J_history] = gradientDescentMulti(X, y, theta, alpha, num_iters)
%GRADIENTDESCENTMULTI Performs gradient descent to learn theta
%   theta = GRADIENTDESCENTMULTI(x, y, theta, alpha, num_iters) updates theta by
%   taking num_iters gradient steps with learning rate alpha


% Initialize some useful values
m = length(y); % number of training examples
J_history = zeros(num_iters, 1);

for iter = 1:num_iters
    theta = theta-(alpha*((X*theta)-y)'*X/m)';
    J_history(iter) = computeCost(X, y, theta);
end
end
```

הפעולה מוצאת את ערך הטטה האופטימלי בשיטת מורד הגרדיאנט (Gradient Descent). היא מקבלת את הפרמטרים הבאים:

- X - מטריצה של מאפיינים
- theta - ווקטור של טטה התחלתי
- alpha - קצת הלמידה
- num_iters - מספר האיטרציות שברצוננו לבצע. יכולנו גם לא לקבל פרמטר זה ולבצע במקום בדיקה של שינוי ערך העלות ולעצור כאשר השינוי קטן מאוד.

הפונקציה מבצעת את מספר האיטרציות כאשר בכל פעם משנה את הטטה בהתאם לקצב הלמידה כפי שהוסבר בפרק העיוני על שיטת הגרדיאנט . כך היא מחשבת את הטטה האופטימלי אותו היא מחזירה (theta) יחד עם היסטוריית ערכי פונקציית העלות - J_history. פונקציונליות אותה כנראה אממש בפרוייקט הפייתון.

- **Feature Scaling and Mean Normalization**

```
function [X_norm, mu, sigma] = featureNormalize(X)
%FEATURENORMALIZE Normalizes the features in X
%   FEATURENORMALIZE(X) returns a normalized version of X where
%   the mean value of each feature is 0 and the standard deviation
%   is 1. This is often a good preprocessing step to do when
%   working with learning algorithms.

mu = mean(X);
sigma = std(X)
iter = size(X)

for i = 1:iter(2)
    X_norm(:, i) = (X(:,i)-mu(i))./sigma(i);
end
end
```

הפעולה מוצאת את ערך פונקציית העלות. היא מקבלת פרמטר אחד בלבד:

- X - מטריצה של מאפיינים אותם אנו רוצים להכניס לטווח ערכים של 1- עד 1.

הפונקציה משתמשת בפונקציונליות נוחה של Matlab. היא לוקחת את הממוצע (mu), לוקחת את טווח מבצעת כפל בין מטריצת המאפיינם ווקטור הטטה, כך היא מקבלת וקטור ההיפותזה עבור כלל דומאות האימון, מכך היא מחסירה את וקטור ה-y ומעלה את הוקטור בחזקת שתיים. לבסוף היא לוקחת את הסכום של כלל התאים ומחלקת ב2 כפול מספר דוגמאות האימון. לבסוף, היא מחזירה את ערך העלות שקיבלה (J).

- **Normal Equation**

הקוד:

```
function [theta] = normalEqn(X, y)
theta = zeros(size(X, 2), 1);
theta = pinv(X'*X)*X'*y
end
```

הפעולה מוצאת את ערך הטטה האופטימלי בשיטת Normal Equation. היא מקבלת את הפרמטרים הבאים:

- X - מטריצה של מאפיינים אותם
- y - סימון דוגמאות האימון, המאפיין אותו אנחנו מנסים לחזות בפועל

הפונקציה מבצעת חישוב כפי שתואר בפרק על Normal Equation ולבסוף מחזירה וקטור טטה אופטימלי.

1.3 מסקנות מהאב-טיפוס

לאחר כתיבת האב טיפוס, אני מרגיש שהשגתי את מטרותיי לפרק זה.

- למדתי להשתמש בשפת MATLAB - הרחבתי את הידע שלי בשפה זו, וצברתי ניסיון בכתיבת קוד בשפה זו.
- הצלחתי להבין את המתמטיקה העומדת מאחורי נוסחאות למידת המכונה, ולמדתי את אופן יישומם באופן שיעבוד בדרך היעילה ביותר - אופן מתמטי על ידי חישובים בעזרת מטריצות ווקטורים.
- יצרתי אב-טיפוס ממנו למדתי כיצד אוכל ליישם ידע תיאורטי אותו צברתי בקוד הפייתון אותו אכתוב בהמשך.

כעת, אני מרגיש שאני מוכן לשלב הבא - כתיבת יישום המודל בפייתון.

2. עבודה בפייתון - python

2.1 מחלקת המודל - Model - התחלתי

תחילה יצרתי מחלקה מופשטת של המודל ששמה Model, בעל התכונות הבאות:

- **all_data** - מטריצה (אובייקט מסוג ndarray של numpy), שומרת בתוכה את כל ה-dataset מתוך קובץ csv אותו מקבלת הפעולה הבונה (__init__). המטריצה מכילה גם את הסימונים של המידע.
- **features** - מטריצה (אובייקט מסוג ndarray של numpy), שומרת בתוכה את כל הנתונים מתוך all_data ללא הסימונים.
- **y** - וקטור (אובייקט מסוג ndarray של numpy), השומר בעצמו את כל הסימונים של ה-dataset עבור כל דוגמה.
- **theta** - וקטור (אובייקט מסוג ndarray של numpy), בעל גודל כמספר המאפיינים השומר בתוכו את הטטה של המודל.

גרסה ראשונה של הפעולה הבונה:

```
def __init__(self, data):
        self.all_data = data.to_numpy()
        self.features = self.all_data[:, 0:-1]
        self.y = self.all_data[:, -1]
        self.theta = np.ones(np.size(self.features, 1))
```

הפעולה מקבלת את data שהוא אובייקט מסוג DataFrame של pandas. בעצם, data שומר את המידע מתוך הקובץ csv של מאגר הנתונים.

כמו כן, לצורך תחילת העבודה, נוצרו הפונקציות הבאות:

- **debug_print** - פונקציה המדפיסה את המידע של כל תכונות אובייקט המודל, לצורך דיבוג.

הקוד:

```
def debug_print(self):
        print("all: \n", self.all_data)
        print("features: \n", self.features)
        print("head: \n", self.features[:50, :])
        print("y: \n", self.y)
```

הפונקציה מדפיסה את כל מאגר המידע של המודל, את מטריצת המאפיינים, את 50 המאפיינים הראשונים, ואת וקטור ה-y שהוא מחירי הנכסים.

- **hypothesis** - פונקציה המחשבת את ערך ההיפותזה בעבור דוגמת אימון ספציפית.

הקוד:

```
def hypothesis(self, x, theta):
        sum = np.sum(x.T * theta)
        return sum
```

הפונקציה מקבלת את הפרמטרים הבאים:

- x - וקטור ה-X של דוגמת האימון הספציפית עבורה חוזים.
- theta - וקטור הטטה בעבורה מחשבים את ערך החיזוי הנוכחי.

הפוקנציה מבצעת כפל וקטורים של X ב-theta, מכך נוצר וקטור שבו כל תא הוא X[i] * Theta[i].

הפונקציה לוקחת את סכום של כל תאי וקטור זה ובכך נוצר ערך החיזוי.

- **error** - פונקציה המחשבת את ערך השגיאה בעבור דוגמת אימון.

הקוד:

```
def error(self, x, y, theta):
        hypothesis = self.hypothesis(x, theta)
        err = pow(hypothesis - y , 2)
        return err
```

הפונקציה מקבלת את הפרמטרים הבאים:

- x - וקטור ה-X של דוגמת האימון הספציפית עבורה חוזים.
- y - ערך ה-y של דוגמת האימון הנוכחית
- theta - וקטור הטטה בעבורה מחשבים את ערך החיזוי הנוכחי.

הפוקנציה מחשבת קודם כל את החיזוי (ערך ההיפותזה) בעבור ה-X והטטה אותם קיבלה, ולאחר מכן, מבצעת חישוב של ערך ההיפותזה פחות ה-y. וכל זה בריבוע. לאחר מכן מחזירה את הערך של השגיאה.

- **cost_function** - פונקציה המחשבת את העלות של המודל בעבור טטה ומאגר נתונים מסויים.

הקוד:

```
def cost_function(self, x, theta):
        cost = 0
        for row in range(np.size(x, 0)):
            cost += self.error(x[row, :], self.y[row], theta)
        cost /= (2*np.size(x, 0))
        return cost
```

הפונקציה מקבלת את הפרמטרים הבאים:

- x - מטריצת ה-X של כלל דוגמאות האימון עבורה חוזים.
- theta - וקטור הטטה בעבורו מחשבים את ערכי החיזוי.

הפונקציה עוברת על כלל דוגמאות האימון ומחשבת עבור כל אחת את השגיאה על ידי הפונקציה error ומוסיפה זאת לעלות. כאשר מסיימת לעבור על כלל השורות, היא מחלקת את התוצאה ב2*מספר דוגמאות האימון. לבסוף, מוחזר ערך זה - ערך העלות הנוכחי.

- **normal_equation** - פונקציה המחזירה טטה אופטימלי באמצעות שיטת ה-Normal Equation.

הקוד:

```
def normal_equation(self):
        X_transpose = self.features.T
        _x = X_transpose.dot(self.features)
        self.theta = np.linalg.pinv(_x).dot(X_transpose).dot(self.y)
        return self.theta
```

הפונקציה מחשבת את הטטה האופטימלי על ידי ביצוע נוסחת ה-Normal Equation. היא מכפילה את מטריצת המאפיינים, בטרנספוזיציה שלה. הופכת את מטריצת התוצאה, ומכפילה שוב בטרנספוזיציה של המאפיינים ובוקטור y. בכך, מקבלת את וקטור הטטה אותו מחזירה.

- **scale_norm** - פונקציה לצורך ביצוע Feature Scaling ו-Mean Normalization בעבור שיפור ביצועי מורד הגרדיאנט.

הקוד:

```
def scale_norm(self, data):
        new_data = data
        for col in range(np.size(data, 1)):
            max = np.amax(data[:, col])
            min = np.amin(data[:, col])
            av = np.average(data[:, col])
            print(max, " ", min, " ", av)
            scale = max - min
            for row in range(np.size(data, 0)):
                new_data[row][col] -= av
                new_data[row][col] /= scale
        return new_data
```

הפונקציה מקבלת את הפרמטרים הבאים:

- data - מטריצת המאפיינים של כלל דוגמאות האימון עבורה חוזים.

הפונקציה עוברת על כלל עמודות המאפיינים, בעבור כל עמודה היא מוצאת את הערך המקסימלי, המינימלי והממוצע. כך, היא מוצאת את הטווח scale ובעבור כל איבר במטריצה היא מבצעת לו Feature Scaling ו-Mean Normalization, זאת על ידי הורדת הממוצע ממנו, וחילוק שלו בטווח scale. כך כל הערכים הם בין 1 ל-1-.

- **gradient_descent** - פונקציה המחזירה טטה אופטימלי באמצעות שיטת ה-Gradient Descent.

הקוד:

```
def gradient_descent(self, learning_rate, theta, scale_norm = False):
      if scale_norm:
            data = self.scale_norm(self.features)
      else:
            data = self.features
      last_cost = self.cost_function(data, theta) + 1
      cost = self.cost_function(data, theta)
      count = 1

      while last_cost - cost > 0.00001:
            print("iter: ", count, " cost: ", self.cost_function(data,
theta))
            count +=1
            last_cost = cost
            for col in range(np.size(data, 1)):
                  row_sum = 0
                  for row in range(np.size(data, 0)):
                        row_sum += (self.hypothesis(data[row, :], theta) -
self.y[row]) * data[row, col]

                  theta[col] -= learning_rate/np.size(data, 0) * row_sum
             cost = self.cost_function(data, theta)
      self.theta = theta
      return theta
```

הפונקציה מקבלת את הפרמטרים הבאים:

- learning_rate - קצב הלמידה. אלפא בנוסחת מורד הגרדיאנט.
- theta - וקטור הטטה בעבורו מחשבים את ערכי החיזוי.
- scale_norm - פרמטר בוליאני המתאר האם לבצע Scaling ו-Normalization או לא.

הפונקציה מבצעת את שיטת מורד הגרדיאנט עד להגעה לערך טטה אופטימלי.

2.2 בחירת שיטה - Gradient Descent vs Normal Equation

מטרתי כעת להראות את מסקנות בחירת השיטה לאחר יישומם.

כפי שהסברתי בחלק העיוני בפרק הלמידת מכונה, Normal Equation היא שיטה היותר מתאימה במקרה זה. כתוצאה מכמות קטנה של מאפיינים (קטנה מ-100,000), אנו יכולים להשתמש בשיטה זו ולא בשיטת מורד הגרדיאנט (Gradient Descent) אשר חסרונה העיקרי הוא לקיחת זמן רב וקביעת קצב למידה.

זאת, ראיתי גם לאחר הרצת השיטות בלאחר שיישמתי אותם בקוד. בעוד שהרצת ה-Normal Equation לקחה לי פחות משניות ספורות, מורד הגרדיאנט לקח זמן רב של אפילו מספר שעות בשביל להגיע לתוצאה דומה. בנוסף, דבר ההקשה על העבודה עם מורד הגרדיאנט הוא קביעת קצב הלמידה (learning rate) נכון. אך לאחר מספר ניסיונות, למדתי לעשות זאת באופן נכון יותר.

חשוב להבין, שעל אף שמורד הגרדיאנט לא התאים למודל שלי, הוא יתאים במצב בו מספר המאפיינים גדול מאוד.

2.3 מציאת מאגר המידע

תחילה ניסיתי למצוא מאגרי נתונים איכותיים איתם אוכל לעבוד. חיפשתי באתרים שונים ביניהם Kaggle, אתר בו מועלים מאגרי מידע רבים לצורך למידת מכונה. מצאתי datasets שונים רבים עבור נדל"ן במדינות שונות, אך החלטתי לבחור במאגר בו כמות המאפיינים לא קטנה ולא גדולה, ובו מספר דוגמאות האימון רבות. בסופו של דבר, חיפשתי מאגר שישלב מאפיינים מגוונים, יהיה בעל מספר רב של דוגמאות אימון ושיוכל לשמש אותי בצורה הטובה ביותר וידמה מצב אמת ככל האפשר.

לבסוף, לקחתי את מאגר נתונים אשר יש בו פרטים בנוגע לכל נכס אשר נמכר בניו יורק לאורך תקופה של 12 חודשים בשנים 2016-2017. המאגר כולל מאפיינים שונים אשר להם אבצע Feature Engineering, ולאחר מכן אבצע Data Cleaning למידע כולו.
מאגר המידע מבוסס על רישומים רשמיים של משרד האוצר של עיריית ניו יורק לשנת 2016-2017, אך מנוקה במעט. כיוון שהמידע משנה משנה נשאר באותו פורמט, מטרתי הסופית היא שאלגוריתם מציאת המודל יעבוד עבור כל מאגר שאתן לו בפורמט זה ויוכל לעבוד איתו, דבר היאפשר לאלגוריתם לעבוד עבור כל שנה עבורה קיימים נתונים.

הסיבה לשימוש בנתונים מהשנים 2016-2017 כיוון ששנים אלו הם לפני תחילת הקורונה, מגפה אשר השפיעה רבות על כל הקשור לנדל"ן בשנים האחרונות. לכן, לקחתי מאגר נתונים משנים בהם שוק הנדל"ן היה יותר ברור ופחות באירועים חריגים.

2.4 המאפיינים - Feature Engineering

2.4.1 מאגר המידע ההתחלתי - סיקור

מאגר המידע מחולק ל-21 עמודות שונות, ביניהן אחליט באיזה עמודות להשתמש וכיצד להעביר אותם לפורמט של מאפיין בו יוכל המודל להשתמש. אשתמש במסמך הרשמי של משרד האוצר של עיריית ניו-יורק "Glossary of Terms for Property Sales Files" בו רשומים הסברים בנוגע לעמודות השונות.

1. **BOROUGH - רובע**

עמודה זו כוללת את הרובע בו הנכס נמצא. ניו-יורק מחולקת ל-5 רובעים, ביניהם: ברונקס, ברוקלין, מנהטן, קווינס וסטייטן-איילנד. במאגר המידע, הערכים בעמודה זו הם בין 1-5 בהתאם לרובעים השונים.

2. **NEIGHBORHOOD - שכונה**

עמודה זו כוללת את שמות השכונות בהם נמצאים הנכסים. השכונה של הנכס משפיעה על מחיר הנכס.

3. **BUILDING CLASS CATEGORY - קטגוריית בניין**

עמודה זו כוללת את סוג המבנה כפי שקבעה זו משרד האוצר של עיריית ניו-יורק. זהו תחום הנכלל על מנת לאפשר שמשתמשי קבצים אלו יוכלו לזהות בקלות נכסים דומים לפי שימוש נרחב (למשל, One Family Homes) מבלי לחפש כיתות בניין בודדות. הקבצים ממוינים לפי רובע, שכונה, קטגוריית בניין, בלוק וחלקה.

4. **TAX CLASS AT PRESENT - קטגוריית המס בזמן הנוכחי**

עמודה זו כוללת את סוג המס על הנכס כפי שקבע אותו משרד האוצר של עיריית ניו-יורק בזמן הפצת מאגר מידע זה (כפי שציינתי לפני כן, מאגר זה מועלה באופן מעודכן מדי שנה). כל נכס בעיר משויך לאחת מארבע דרגות מס (דרגות 1, 2, 3 ו-4), בהתאם לשימוש בנכס.

- סוג 1: כולל את רוב הנכסים למגורים של עד שלוש יחידות (כגון בתים חד, דו ותלת-משפחתיים וחנויות קטנות או משרדים עם דירה צמודה אחת או שתיים), קרקע פנויה המיועדת למגורים, וכן רוב הבתים המשותפים שהם לא יותר משלוש קומות.
- סוג 2: כולל את כל הנכסים האחרים שעיקרם מגורים, כגון בתים משותפים.
- סוג 3: כולל נכס עם ציוד בבעלות חברת גז, טלפון או חשמל.
- סוג 4: כולל את כל שאר הנכסים שאינם נכללים בסיווג 1,2 ו-3, כגון משרדים, מפעלים, מחסנים, בנייני מוסכים וכו'.

5. BLOCK - בלוק

עמודה זו כוללת את מספר הבלוק של הנכס. בלוק מס הוא תת חטיבה של הרובע שעליו נמצאים נכסים. משרד האוצר משתמש בסיווג רובע-בלוק-חלקה כדי לתייג את כל הנדל"ן בעיר. בעוד שכתובות מתארות את מיקום הרחוב של נכס, הבלוק והחלקה מבדילים יחידה אחת של נכס לאחרת, כגון הבתים המשותפים השונים בבניין בודד. כמו כן, בלוק ומגרשים אינם כפופים לשינויי שמות בהתאם לאיזה צד הבניין שם את הכניסה שלו.

6. LOT - חלקה

עמודה זו כוללת את מספר חלקה של הנכס. חלקת מס היא חלק משנה של בלוק מס ומייצג את המיקום הייחודי של הנכס.

7. EASE-MENT - זיקת הנאה

זיקת הנאה היא זכות, כגון זכות הדרך, המאפשרת לגוף לעשות שימוש מוגבל בנכס של אחר. לדוגמה: פסי רכבת MTA העוברים על פני חלק מנכס אחר.

8. BUILDING CLASS AT PRESENT - קטגוריית הבניין בזמן הנוכחי

עמודה זו כוללת את סוג המבנה כפי שקבעה זו משרד האוצר של עיריית ניו-יורק, אך באופן מפורט יותר המתאר את שימוש הנכס בזמן הנוכחי.

קטגוריית הבניין משמשת לתיאור השימוש הקונסטרוקטיבי של הנכס. המיקום הראשון של קטגוריית הבניין היא אות המשמשת לתיאור סוג כללי של נכסים (לדוגמה "A" מסמל בתים חד-משפחתיים, "O" מסמל בנייני משרדים. "R" מסמל בתים משותפים). העמדה השנייה, היא מספר, המוסיפה מידע ספציפי יותר על השימוש או סגנון הבנייה של הנכס (באמצעות הדוגמאות הקודמות שלנו "A0" הוא בית משפחתי אחד בסגנון קייפ קוד, "O4" הוא בניין משרדים מסוג מגדל ו-"R5" הוא יחידה מסחרית משותפת). המונח Building Class המשמש את משרד האוצר ניתן להחלפה עם המונח Building Code המשמש את משרד הבניינים.

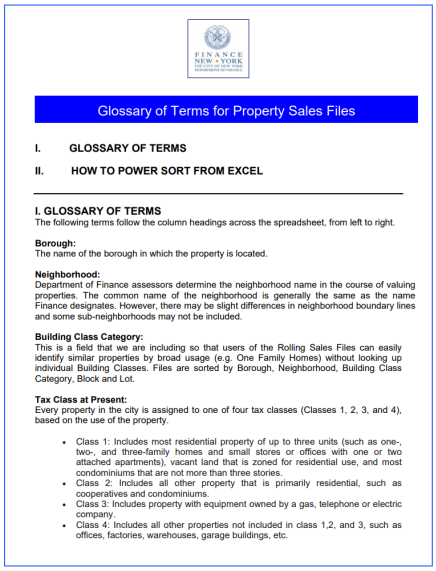
על פי המסמך הרשמי של משרד האוצר של ניו-יורק בנוגע לקטגוריות הבניינים, ישנם 22 אותיות (מ-A ל-Z) ועבור כל אות ישנם בין 4 ל-21 מספרים או אותיות.

9. ADDRESS - כתובת

עמודה זו כוללת את כתובת הנכס. כתובת זו עלולה לכלול גם את מספר הדירה.

10. APARTMENT NUMBER - מספר הדירה

עמודה זו כוללת את מספר הדירה של הנכס.



Glossary of Terms for Property Sales Files

I. GLOSSARY OF TERMS

II. HOW TO POWER SORT FROM EXCEL

I. GLOSSARY OF TERMS
The following terms follow the column headings across the spreadsheet, from left to right.

Borough:
The name of the borough in which the property is located.

Neighborhood:
Department of Finance assessors determine the neighborhood name in the course of valuing properties. The common name of the neighborhood is generally the same as the name Finance designates. However, there may be slight differences in neighborhood boundary lines and some sub-neighborhoods may not be included.

Building Class Category:
This is a field that we are including so that users of the Rolling Sales Files can easily identify similar properties by broad usage (e.g. One Family Homes) without looking up individual Building Classes. Files are sorted by Borough, Neighborhood, Building Class Category, Block and Lot.

Tax Class at Present:
Every property in the city is assigned to one of four tax classes (Classes 1, 2, 3, and 4), based on the use of the property.

- Class 1: Includes most residential property of up to three units (such as one-, two-, and three-family homes and small stores or offices with one or two attached apartments), vacant land that is zoned for residential use, and most condominiums that are not more than three stories.
- Class 2: Includes all other property that is primarily residential, such as cooperatives and condominiums.
- Class 3: Includes property with equipment owned by a gas, telephone or electric company.
- Class 4: Includes all other properties not included in class 1,2, and 3, such as offices, factories, warehouses, garage buildings, etc.

11. ZIP CODE - מיקוד הנכס

עמודה זו כוללת את מספר המיקוד של הנכס.

12. RESIDENTIAL UNITS - יחידות דיור

עמודה זו כוללת את מספר יחידות הדיור של הבניין בו נמצא הנכס.

13. COMMERCIAL UNITS - יחידות מסחריות

עמודה זו כוללת את מספר היחידות המסחריות של הבניין בו נמצא הנכס.

14. TOTAL UNITS - מס' יחידות כולל

עמודה זו כוללת את סך מספר היחידות של הבניין בו נמצא הנכס.

15. LAND SQUARE FEET - שטח הנכס

עמודה זו כוללת את שטח הנכס ב-feet בריבוע.

16. GROSS SQUARE FEET - שטח כולל

השטח הכולל של כל קומות הבניין כפי שנמדד מהמשטחים החיצוניים של הבניין קירות חיצוניים של הבניין, לרבות שטח הקרקע והחלל בתוך כל בניין או מבנה על הנכס.

17. YEAR BUILT - שנת בנייה

עמודה זו כוללת את השנה בה הנכס נבנה.

18. TAX CLASS AT TIME OF SALE - קטגוריית המס בזמן המחירה

עמודה זו כוללת את סוג המס על הנכס כפי שהיה כאשר נמכר הנכס.

19. BUILDING CLASS AT TIME OF SALE - קטגוריית הבניין בזמן המחירה

עמודה זו כוללת את קטגוריית הבניין כפי שהיה כאשר נמכר הנכס.

20. SALE PRICE - מחיר בו נמכר הנכס

עמודה זו כוללת את המחיר בו נמכר הנכס בדולרים. מחיר עלול להיות שווה ל-0 כאשר מדובר בהעברת בעלות ללא תשלום. למשל, כאשר הורים מעבירים לילדיהם את הנכס.

21. SALE DATE - תאריך מכירת הנכס

עמודה זו כוללת את התאריך בו נמכר הנכס.

2.4.2 קביעת המאפיינים

לאחר שעברנו על מאגר המידע, אפרט על תהליך קביעת המאפיינים (Feature Engineering).

עמודות לא בשימוש

תחילה, אפרט על העמודות בהן לא אשתמש ומדוע. חשוב לזכור, שבסופו של דבר מטרת המודל לחזות את ערך הנדל"ן, על מנת לעשות את זה, אנו לא זקוקים למידע שלא נכון עבור זמן המכירה עצמו. למשל, קטגוריית הבניין וקטגוריית המס שלא בזמן במכירה אינן חשובות לנו כיוון שאין להן השפעה על ערך הנדל"ן עצמו בזמן המכירה. מטרתינו לשמור אך ורק את המידע אשר משפיע על מחיר הנכס, משמע מידע התואם את זמן המכירה.

כמו כן, לא נשתמש בכתובת מכמה סיבות. קודם כל, ישנו קושי להעביר את הנתונים של הכתובת לערך מספרי שבו נוכל להשתמש במודל באופן שבו כל כתובת תהיה מספר ייחודי משלה. הסיבה השנייה, היא שמבחינתנו כל נכס ניתן לתיאור תוך שילוב של רובע, בלוק וחלקה כפי שתיארתי לעיל, או אפילו בעזרת מיקוד, לכן ניתן לוותר על כתובת ואולי אף על מיקוד. על כן, גם מספר דירה לא נחוץ כאן כיוון שכל נכס מוגדר בנפרד על ידי רובע, בלוק, וחלקה, כאשר בכתובת אחת עלולים להיות מספר נכנסים אותם מבדילים על ידי מספר דירה לכן כתובת ומספר דירה לא נחוצים.

בנוסף לכך, העמודה של מחיר המכירה יהווה את הסימון של מאגר המידע.

כמו כן, TOTAL UNITS הוא פשוט סכום של RESIDENTIAL UNITS ו-COMMERCIAL UNITS לכן כנראה גם בו לא אעשה שימוש אך אראה את ההשפעה שלו במהלך אימון המודל.

כמו כן, לא אשתמש בעמודה EASE-MENT כיוון שאיני רואה כיצד יכולה להשפיע על מחיר הנכס, וניתן לראות שאין עבורה מידע בטבלה.

עמודות אותן אשנה

כעת, אעבור על עמודות בעיתיות להן אבצע Feature Engineering. העמודה NEIGHBORHOOD לדעתי כן תשפיע על יעילות המודל. במהלך העבודה אראה את השפעתה. הבעיה בעמודה זו היא שהיא בעלת ערכים לא מספריים לכן אעביר אותה אליהם. במקום שכל שם של שכונה ישמר בתור אותו השם, אגדיר עבור כל שכונה ערך משלה. אעשה זאת על ידי הגדרת dictionary אשר בו יהיו שמורים כל השכונות. עבור כל שכונה, יהיה מוגדר הערך שלה במודל. כך יווצר לנו טווח מספרי בו כל שכונה היא ערך אחר, מידע אותו המודל מסוגל לקבל.

תהליך דומה אבצע עבור קטגוריית הבניין (בזמן המכירה). אשר עבורו אגדיר לכל קטגורייה ערך על ידי הגדרת dictionary שבו עבור כל שם יהיה ערך מספרי.

בנוסף, לא אשתמש בעמודת התאריך באופן המלא בו היא שמורה. כעת שמורים שם התאריך עצמו והשעה בו נמכר הנכס. על מנת לפשט את העמודה, אגדיר עבור כל דוגמת אימון לא את כל התאריך אלא רק את החודש (ערכים בטווח 1-12).

2.4.3 מאפיינים סופיים לתחילת העבודה

לאחר שהגדרנו את המאפיינים אתאר כאן את המאפיינים כולם.

1. **X0 - מאפיין מקום 0**

תמיד שווה ל-1 ונמצא עבור נקודת חיתוך עם ציר X.

2. **X1 - רובע**

מאפיין זה מסמל את הרובע בו הנכס נמצא. ניו-יורק מחולקת ל-5 רובעים, ביניהם: ברונקס, ברוקלין, מנהטן, קווינס וסטייטן-איילנד. הערכים האפשריים הם בין 1-5 בהתאם לרובעים השונים.

3. **X2 - שכונה**

מאפיין זה מסמל את השכונות השונות בהם נמצאים הנכסים. עבור כל שכונה יוגדר ערך משלה. טבלה מלאה ניתן לראות בעמוד X.

4. **X3 - בלוק**

מאפיין זה מסמל את מספר הבלוק של הנכס. בלוק מס הוא תת חטיבה של הרובע שעליו נמצאים נכסים. משרד האוצר משתמש בסיווג רובע-בלוק-חלקה כדי לתייג את כל הנדל"ן בעיר.

5. **X4 - חלקה**

מאפיין זה מסמל את מספר החלקה של הנכס. חלקת מס היא חלק משנה של בלוק מס ומייצג את המיקום הייחודי של הנכס. משרד האוצר משתמש בסיווג רובע-בלוק-חלקה כדי לתייג את כל הנדל"ן בעיר.

6. **X5 - מיקוד**

מאפיין זה מסמל את מספר המיקוד של הנכס.

7. **X6 - יחידות דיור**

מאפיין זה מסמל את מספר יחידות הדיור של הבניין.

8. **X7 - יחידות מסחריות**

מאפיין זה מסמל את מספר היחידות המסחריות של הבניין.

9. X8 - יחידות סה"כ

מאפיין זה מסמל את סך מספר היחידות של הבניין.

10. X9 - שטח

מאפיין זה מסמל את שטח הנכס ב-feet בריבוע.

11. X10 - שטח כולל

מאפיין זה מסמל את שטח הכולל של הבניין ב-feet בריבוע (של כלל הקומות).

12. X11 - שנת בנייה

מאפיין זה מסמל את שנת בניית הנכס.

13. X12 - קטגוריית מס

מאפיין זה מסמל את סוג המס על הנכס כפי שקבע אותו משרד האוצר של עיריית ניו-יורק בזמן הפצת מאגר מידע זה (כפי שציינתי לפני כן, מאגר זה מועלה באופן מעודכן מדי שנה). את קטגוריות המס השונות תיארתי בהסבר על העמודות השונות במאגר המידע המקורי. הטווח מ1-4.

14. X13 - קטגוריית בניין

מאפיין זה מסמל את סוג המבנה כפי שקבעה זו משרד האוצר של עיריית ניו-יורק, אך באופן מפורט יותר המתאר את שימוש הנכס בזמן המכירה. את קטגוריות הבניין השונות תיארתי בהסבר על העמודות השונות במאגר המידע המקורי.

X14 - חודש המכירה

עמודה זו כוללת את החודש בו נמכר הנכס, בטווח 1-12 בהתאם לחודש.

15. Y - מחיר הנכס

עמודה זו כוללת את המחיר בו נמכר הנכס בדולרים.

אלו הם המאפיינים איתם אתחיל את העבודה, מאפיינים אלו סביר להניח שישתנו רבות בהמשך.

2.4.4 תהליך הניקיון (Data Cleaning)

על מנת לנקות את המאגר מידע, נמחק קודם כל את כל דוגמאות האימון בהן אחד הערכים של המאפיינים שקבענו שווה ל-0, לתו " - " או פשוט ריק.

קוד הניקוי

```
def clean(self, data):
      #setting the columns that need to be checked
      check_col = ['TOTAL UNITS', 'LAND SQUARE FEET',
                   'GROSS SQUARE FEET', 'SALE PRICE']

      #flag for the check_col loop
      rmv_flag = False

      #going through the data
      for row in range(np.size(data, 0)):
            #going through all the check_col columns
            for e in check_col:
                  if data.loc[row].at[e] == ' -  ' or
                     data.loc[row].at['SALE PRICE'] == ' -  ' or
                     int(data.loc[row].at[e]) <= 0 or
                     int(data.loc[row].at['SALE PRICE']) <= 1000:
                        rmv_flag = True
                              break

            #if row need to be removed
            if rmv_flag:
                  rmv_flag = False
                  data.drop(row, axis=0, inplace=True)
            print(row) #for debug

      #removing numbering columns
      data.drop(columns=['Unnamed: 0'], inplace=True)

      #saving new data
      data.to_csv(r'C:\Users\Dan\Desktop\ML-NYC\new-data.csv')
      return data
```

ה-data אותו מקבלת הפעולה, מתקבל משורת קוד אחת אשר בה הקוד לוקח את מאגר המידע ומכניס אותו כ-Dataframe של pandas:

```
df = pd.read_csv (r'C:\Users\Dan\Desktop\ML-NYC\nyc-rolling-sales.csv')
```

לפני הניקוי היה לנו 84,549 דוגמאות אימון כאשר לאחר מכן יש ברשותנו 28,351 דוגמאות אימון.

הפעולה מבצעת מעבר על כל השורות בטבלה, אשר במהלכה בודקת עבור כל שורה האם הערכים בעמודות אשר שמותיהן ברשימה check_col לא שווים למחרוזת " - " (המסמלת ערך NULL) ונמצאים בטווח המתאים. אם כן, נכנסים לתנאי שאחרי הלולאה, ומוחקים את השורה על ידי הפעולה drop של pandas.

לצורך דיבוג מבצעת הפעולה הדפסת מספר השורה עבור כל שורה אשר נבדקה.

לאחר מכן, מוחקים את העמודה "Unnamed: 0" מתוך הטבלה הנקייה שלנו. עמודה זאת שומרת בעצמה את המספר המקורי עבור כל שורה בטבלה המקורית.

לאחר מכן, בודקים ומדפיסים את כמות הכפילויות במאגר המידע, אם לא שווה ל-0 אז מוחקים את כל הכפילויות.

לבסוף, שומרים את הטבלה הנקייה בתור קובץ new-data.csv אשר בו יישמר הdata על מנת לא לבצע ניקוי מחדש כאשר קיים קובץ זה.

Feature Engineering 2.4.5

בשלב זה ננסה לראות כיצד נגדיר את המידע שלנו בתוך מאפיינים בהם ישתמש המודל.

קוד התחלתי בעבור הגדרת המאפיינים

תחילה מוגדר משתנה saved_data אשר שומר את הDataframe של מאגר הנתונים. על משתנה זה יבוצעו השינויים.

```
def feature_prepare(self, data):
      saved_data = data
```

לאחר מכן, מורידים ממאגר הנתונים את העמודות אשר קבענו כלא נחוצות.

```
#removing unnecessary columns
saved_data.drop(columns = ['BUILDING CLASS CATEGORY', 'TAX CLASS AT
      PRESENT', 'EASE-MENT', 'BUILDING CLASS AT PRESENT', 'ADDRESS',
      'APARTMENT NUMBER'], inplace=True)
```

כעת, בודקים אם לא יצרנו כבר מאגר נתונים נקי. אם לא אז מבצעים ניקוי ואם כן, אז לוקחים את הקובץ של מאגר הנתונים הנקי.

```
#cleaning
if not os.path.isfile(r'C:\Users\Dan\Desktop\ML-NYC\new-data.csv'):
      saved_data = self.clean(saved_data)
else:
      saved_data = pd.read_csv(r'C:\Users\Dan\Desktop\ML-NYC\new-data.csv')
      saved_data = saved_data.iloc[: , 1:]
```

לאחר מכן, משנים את הערכים בעמודת תאריך המכירה, כך שעבור כל דוגמת אימון יהיה בעמודה זו רק ערך החודש של המכירה.

```
#change SALE DATE to the months
for row in range(np.size(saved_data, 0)):
      value = int(saved_data.at[row, 'SALE DATE'][5:7])
      saved_data.at[row, 'SALE DATE'] = value
      print(row) #for debug
```

אז, מגדירים מילון ריק עבור ערכי השכונות השונות, ורשימה עבור כל קטגוריות הבניינים.

```
self.neighborhood = {}
self.building_classes = ['A0' ,'A1' ,'A2' ,'A3' ,'A4' ,'A5' ,'A6', 'A7',
      'A8' ,'A9' ,'B1' ,'B2' ,'B3' ,'B9' ,'C0' ,'C1' ,'C2' ,'C3' ,'C4' ,
      'C5' ,'C6' ,'C7' ,'C8' ,'C9' ,'CM' ,'D0' ,'D1' ,'D2' ,'D3' ,'D4' ,
      'D5' ,'D6' ,'D7' ,'D8' ,'D9' ,'E1' ,'E2' ,'E3' ,'E4' ,'E7' ,'E9' ,
      'F1' ,'F2' ,'F4' ,'F5' ,'F8' ,'F9' ,'G0' ,'G1' ,'G2' ,'G3' ,'G4' ,
      'G5' ,'G6' ,'G7' ,'G8' ,'G9' ,'GU' ,'GW' ,'HB' ,'HH' ,'HR' ,'HS' ,
      'H1' ,'H2' ,'H3' ,'H4' ,'H5' ,'H6' ,'H7' ,'H8' ,'H9' ,'I1' ,'I2' ,
      'I3' ,'I4' ,'I5' ,'I6' ,'I7' ,'I9' ,'J1' ,'J2' ,'J3' ,'J4' ,'J5' ,
      'J6' ,'J7' ,'J8' ,'J9' ,'K1' ,'K2' ,'K3' ,'K4' ,'K5' ,'K6' ,'K7' ,
      'K8' ,'K9' ,'L1' ,'L2' ,'L3' ,'L8' ,'L9' ,'M1' ,'M2' ,'M3' ,'M4' ,
      'M9' ,'N1' ,'N2' ,'N3' ,'N4' ,'N9' ,'O1' ,'O2' ,'O3' ,'O4' ,'O5' ,
      'O6' ,'O7' ,'O8' ,'O9' ,'P1' ,'P2' ,'P3' ,'P4' ,'P5' ,'P6' ,'P7' ,
      'P8' ,'P9' ,'Q1' ,'Q2' ,'Q3' ,'Q4' ,'Q5' ,'Q6' ,'Q7' ,'Q8', 'Q9' ,
      'RA' ,'RB' ,'RG' ,'RH' ,'RK' ,'RP' ,'RR' ,'RS' ,'RT' ,'RW' ,'R0' ,
      'R1' ,'R2' ,'R3' ,'R4' ,'R5' ,'R6' ,'R7' ,'R8' ,'R9' ,'RR' ,'S0' ,
      'S1' ,'S2' ,'S3' ,'S4' ,'S5' ,'S9' ,'T1' ,'T2' ,'T9' ,'U0' ,'U1' ,
      'U2' ,'U3' ,'U4' ,'U5' ,'U6' ,'U7' ,'U8' ,'U9' ,'V0' ,'V1' ,'V2' ,
      'V3' ,'V4' ,'V5' ,'V6' ,'V7' ,'V8' ,'V9' ,'W1' ,'W2' ,'W3' ,'W4' ,
      'W5' ,'W6' ,'W7' ,'W8' ,'W9' ,'Y1' ,'Y2' ,'Y3' ,'Y4' ,'Y5' ,'Y6' ,
      'Y7' ,'Y8' ,'Y9' ,'Z0' ,'Z1' ,'Z2' ,'Z3' ,'Z4' ,'Z5' ,'Z7' ,'Z8' ,
      'Z9']
```

לאחר מכן מגדירים סופית את שני מילונים סופיים עבור השכונות השונות וקטגוריות הבניינים השונים.

עבור כל מילון מגדירים ערכים בסדר עולה (1,2,3...)

```
#create dictionary for NEIGHBORHOOD
for row in range(np.size(saved_data, 0)):
      key_n = saved_data.at[row, 'NEIGHBORHOOD']
      self.neighborhood[key_n] = 0
      print(row) #for debug

#create dictionary for BUILDING CLASS AT TIME OF SALE
self.building_classes = dict.fromkeys(self.building_classes)

#create values for NEIGHBORHOOD dictionary
counter = 1
for key in self.neighborhood:
```

```
        self.neighborhood[key] = counter
        counter+=1

#create values for BUILDING CLASS AT TIME OF SALE dictionary
counter = 1
for key in self.building_classes:
        self.building_classes[key] = counter
        counter+=1
```

כעת, מעדכנים את כלל הערכים בעמודות קטוגריית הבניין והשכונה.

```
#change NEIGHBORHOOD and BUILDING CLASS AT TIME OF SALE to the values
for row in range(np.size(saved_data, 0)):
        saved_data.at[row, 'NEIGHBORHOOD'] =
            self.neighborhood[saved_data.at[row, 'NEIGHBORHOOD']]
        saved_data.at[row, 'BUILDING CLASS AT TIME OF SALE'] =
            self.building_classes[saved_data.at[row, 'BUILDING CLASS AT TIME
                OF SALE']]
        print(row) #for debug
```

לבסוף מכניסים את כלל המידע לתוך תכונה ששמה all_data, את כלל המידע ללא הסימון עם מאפיין 0 לתוך תכונה features, ואת הסימון לתכונה y. פירוט מלא יהיה בתת פרק הבא.

```
self.all_data = saved_data.to_numpy()
self.features = np.c_[np.ones(np.size(self.all_data, 0)),
        self.all_data[:,0:-2], self.all_data[:, -1]] #adding X0
self.y = self.all_data[:, -2]
```

לאחר ניסיון הרצת Normal Equation נראה כי המודל לא עובד טוב בלשון המעטה. ערך העלות המוחזר לנו גבוה מאוד, ושווה ל-59,988,637,634,732 שזוהי שגיאה של כ-10.8 מיליון בממוצע בעבור כל דוגמת אימון.

זאת מעיד, שנתקלנו בבעיית **התאמת חסר**.

2.4.6 התאמת חסר - המשך עבודה - נסיונות פתרון הבעיה ושיפור המודל

כפי שניתן להבין, נתקלנו בבעית התאמת חסר, בעיה נפוצה עבור מודלים מסוג לינארי.

2.4.6.1 אלגוריתם לבחירת מודל - Model Selection Algorithm

בתור התחלה, ניסיתי להשתמש ב-model selection algorithm (אלגוריתם לבחירת מודל) אשר ימצא בעבורי את השילוב הטוב ביותר של המאפיינים בעבור רגרסיה לינארית אשר בעבורה יהיה ערך העלות הנמוך ביותר. זאת בשביל לוודא, האם כל המאפיינים שבחרתי נחוצים למודל.

קוד האלוגריתם:

```
def model_selection(model):
    m_feat = model.features
    x_num = np.size(model.features, 1)

    min_cost = LARGE_NUM
    min_theta = []
    min_comb = []

    possible_col = range(x_num)

    all_combinations = []
    for r in range(len(possible_col) + 1):
        combinations_object = itertools.combinations(possible_col, r)
        combinations_list = list(combinations_object)
        all_combinations += combinations_list

    for comb in all_combinations:
        model.features = m_feat[:, comb]
        np.set_printoptions(suppress=True)
        theta = model.normal_equation()
        cost = model.cost_function(model.features, theta)
        print(comb,"\ncost: ",cost,"\n\nmin comb:",min_comb,"\nmin cost: ",
min_cost)
        if cost < min_cost:
            min_cost = cost
            min_theta = theta
            min_comb = comb
    print("min comb: ", min_comb, "min cost: ", min_cost)
```

האלגוריתם לא עובד באופן המדויק אותו תיארתי לפני כן, בו מחלקים את מאגר המידע לשלושה חלקים לצורך בדיקה מדויקת, כיוון שמטרתי לראות את ההבדל בערך העלות תוך שימוש בשילובי מאפיינים שונים.

לאחר מספר שעות של עבודת האלגוריתם, הוא החזיר כי השילוב הטוב ביותר הוא של כל המאפיינים, מה שמחזיר אותנו לנקודת ההתחלה, בה ערך העלות גבוה מאוד.

```
min comb:  (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14) min cost:  599886376347 32.06
```

אך בדיקה זאת לא הייתה לשווא. בדיקה זו הראתה על הצורך בשינוי גישה אל הבעיה. לכן, החלטתי כי אעבור שוב על מאגר המידע עצמו.

2.4.6.2 מעבר על מאגר המידע

כעת נעבור על מאגר הנתונים ונראה אם נוכל להסיק מסקנות כלשהן.

נעשה זאת באמצעות הפעולה describe של pandas.

```
#seeing the data
data_info = data.describe()
data_info.to_csv(r'C:\Users\Dan\Desktop\ML-NYC\data-info.csv')
print(data_info)
```

זה המידע שקיבלנו:

	BOROUGH	BLOCK	LOT	ZIP CODE	RESIDENTIAL UNITS	COMMERCIAL UNITS	TOTAL UNITS	LAND SQUARE FEET	GROSS SQUARE FEET	YEAR BUILT	SALE PRICE
count	28351	28351	28351	28351	28351	28351	28351	28351	28351	28351	28351
mean	3.5414623	5564.50	61.59	10995.7944	2.988642376	0.31579133	3.3029875	4016.97	4332.866	1940.0	1685172.557
std	1.01482852	3777.17	117.5	513.912243	19.13012006	14.01843297	23.861673	32205.75	33329.04	44.61	17273299.48
min	1	5	1	0	0	0	1	200	120	0	1162
25%	3	2702.5	19	10462	1	0	1	2000	1360	1920	435000
50%	4	4931	38	11221	2	0	2	2500	1870	1930	635000
75%	4	7850	64	11373	2	0	2	4000	2655	1959	970000
max	5	16319	3710	11694	1844	2261	2261	4228300	3750565	2017	2210000000

הנקודות שעלו מהמידע

- ישנם דוגמאות אימון אשר ערך המיקוד שלהם הוא 0. דבר שאינו אפשרי.
- ישנם בתים בעלי ערך שנת בנייה 0. דבר שצריך לנקות אותו.
- כ-75% מבין כל הבתים הם בעלי ערך 970,000 ומטה, אך בו זמנית קיימת דוגמת אימון בעלת המחיר 2,210,000,000 ודוגמאות אימון בעלות המחיר 1162.
- כ-75% מבין כל הבתים אינם בעלי יחידות מסחריות.
- כ-75% מבין כל הבתים בעלי ערך של LAND SQUARE FEET של 4000 ומטה ו-GROSS SQUARE FEET של 2655 ומטה, כאשר ישנם דוגמאות אימון בעלות ערכים 4,228,300 ו-3,750,565 בהתאם.

מסקנות להמשך וביצוע שינויים

בהתאם לנתונים ניתן להבין שתהליך הניקוי לא היה מאוד מוצלח. עלינו לבצע ניקוי יסודי יותר בהתחשב בכמה מאפיינים.

נתחיל מלעבור על הסימון של המידע (מחיר הנכס), ואז נמשיך לעבור בין המאפיינים העיקריים, איתם היו בעיות וננתח את המידע שלהם על ידי גרפים מסוגים שונים.

2.4.6.3 ניתוח, ניקוי והגדרת מאפיינים

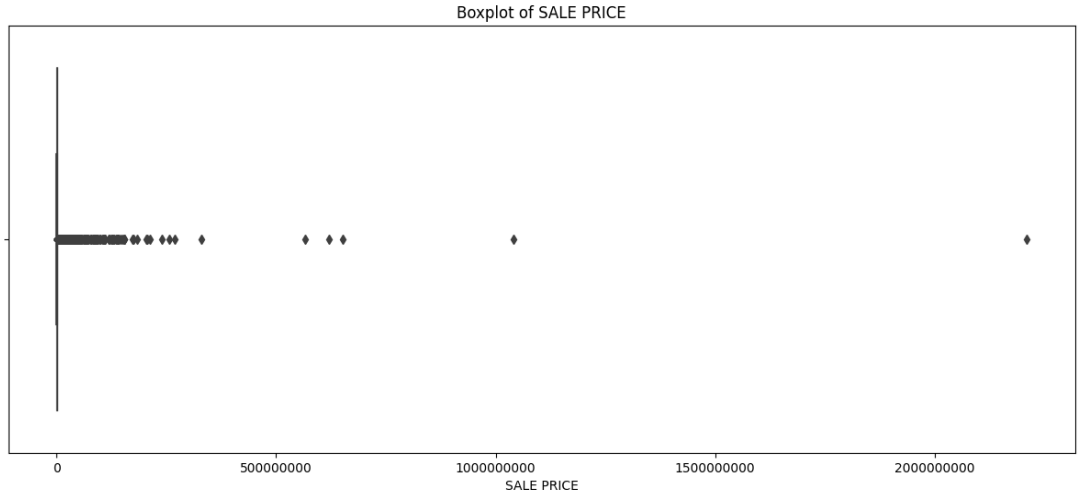
SALE PRICE - מחיר הנכס

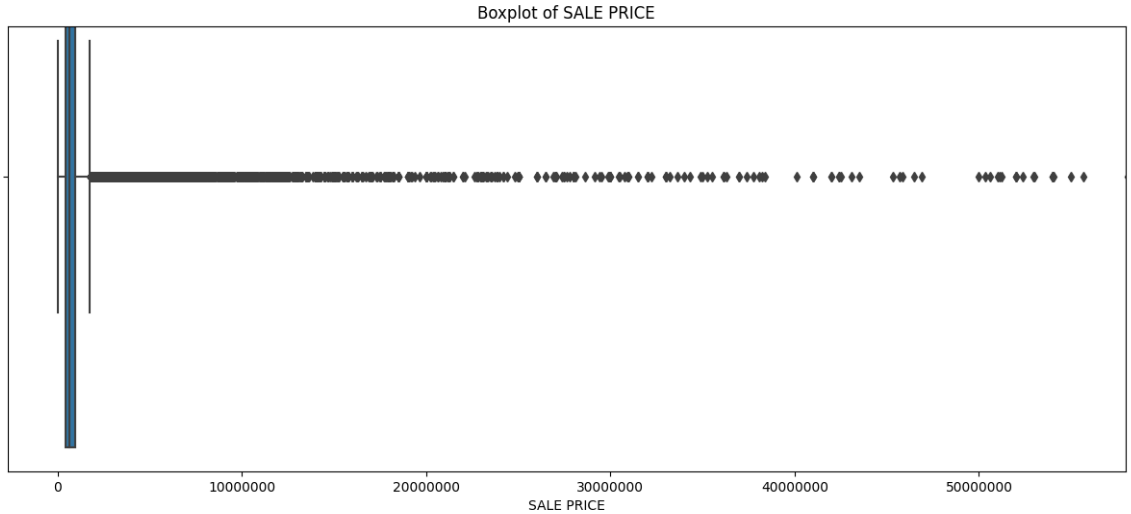
נתחיל מניקוי על פי המאפיין החשוב ביותר, הסימון - מחיר הנכס. קודם כל ננסה ליצור מספר גרפים שיתארו עבורנו באופן הטוב ביותר את כמות הדוגמאות בעבור כל מחיר. קודם כל נעשה זאת על יד גרף מסוג Boxplot מתוך הספרייה Seaborn.

זאת נעשה על ידי הוספת הקוד הבא לפונקציית הניקוי clean:

```
#Set the size of the plot
plt.figure(figsize=(15,6))

# Plot the data and configure the settings
sns.boxplot(x='SALE PRICE', data=data)
plt.ticklabel_format(style='plain', axis='x')
plt.title('Boxplot of SALE PRICE')
plt.show()
```

גרף ה-Boxplot:





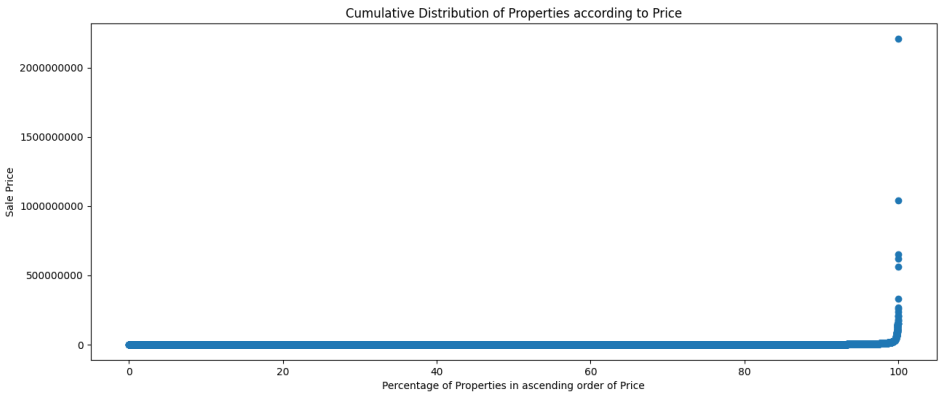
ניתן לראות כי מרבית הנכנסים הרבה יותר קטנים אפילו מ-500,000,000. וכאשר אנחנו מתקרבים ניתן לראות שמרבית הנכסים ערכם לא גדול מ-6,000,000.

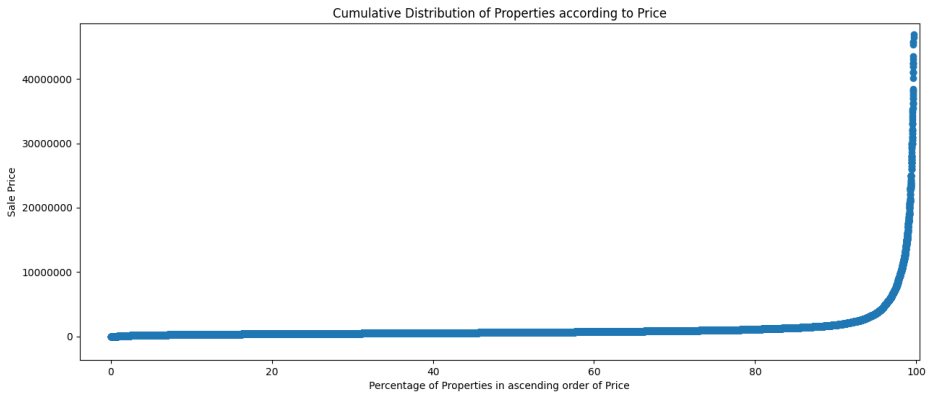
לכן ננסה לחקור זאת יותר לעומק על ידי גרף מסוג Cumulative distribution (התפלגות) אשר יציג את אחוז הנכסים מתחת לכל מחיר.

זאת נעשה על ידי הוספת הקוד הבא לפונקציית הניקוי clean:

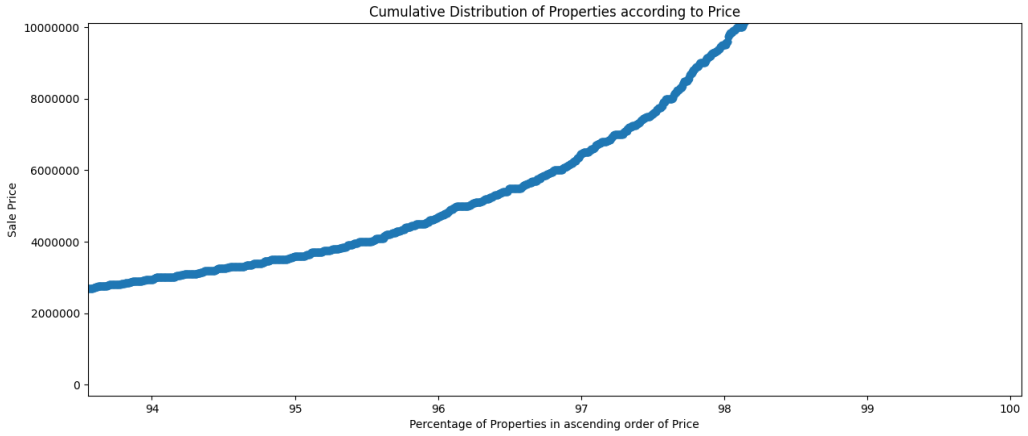
```
#Set the size of the plot
plt.figure(figsize=(15,6))
#Get the data and format it
x = data[['SALE PRICE']].sort_values(by='SALE PRICE').reset_index()
x['PROPERTY PROPORTION'] = 1
x['PROPERTY PROPORTION'] = x['PROPERTY PROPORTION'].cumsum()
x['PROPERTY PROPORTION'] = 100* x['PROPERTY PROPORTION'] /
      len(x['PROPERTY PROPORTION'])

# Plot the data and configure the settings
plt.plot(x['PROPERTY PROPORTION'],x['SALE PRICE'], linestyle='None',
marker='o')
plt.title('Cumulative Distribution of Properties according to Price')
plt.xlabel('Percentage of Properties in ascending order of Price')
plt.ylabel('Sale Price')
plt.ticklabel_format(style='plain', axis='y')
plt.show()
```

גרף ההתפלגות:



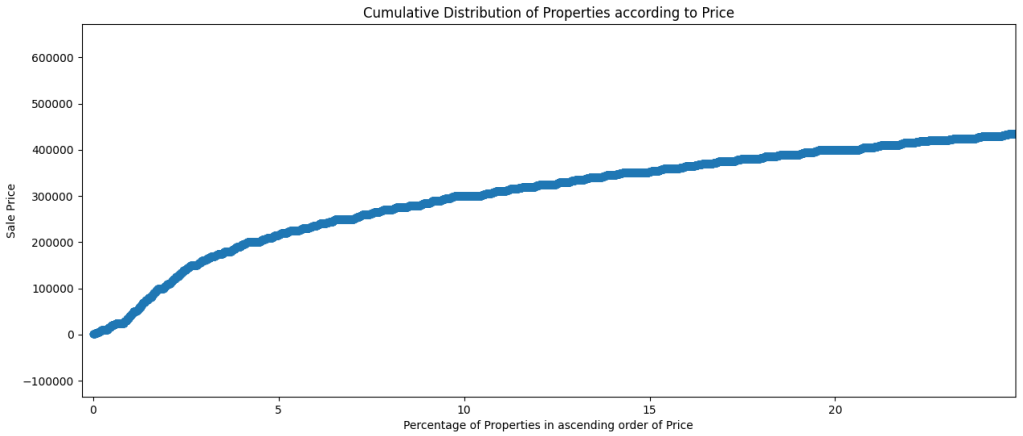


כפי שניתן לראות, הגרף נמצא בשיפוע קטן ביחס לעלייה הדרסטית הקיימת מ-90%. לכן נראה אזור זה יותר לעומק.

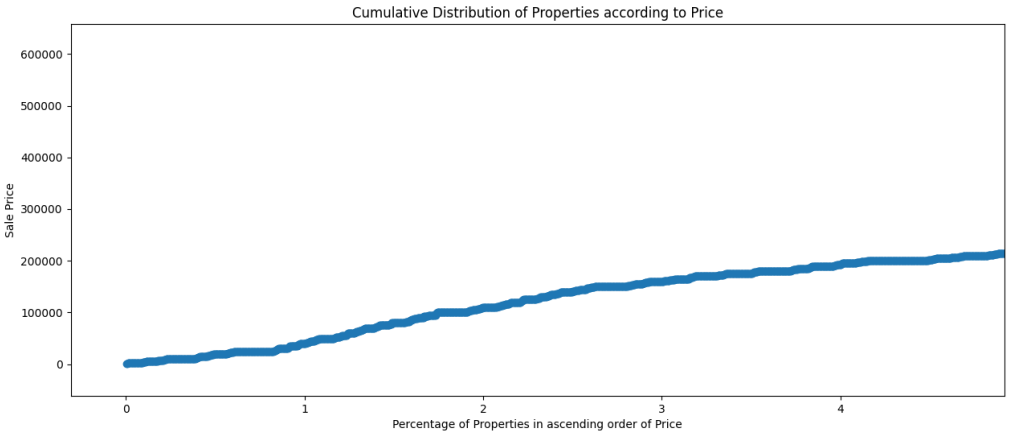


נראה ונבין, כי כ-97% מהנכסים בעלי מחיר של כ-6,000,000.

כמו כן, נתבונן ב25% התחתון של הגרף.



נראה כי הגרף ישר בערך מה-5%. לכן, נמקד את החיפוש ב5% אלו בשביל למצוא מאיזה מחיר נשמור את דוגמאות האימון.



נראה כי כ-2.5% מהנכסים בעלי מחיר 150,000 ומטה.

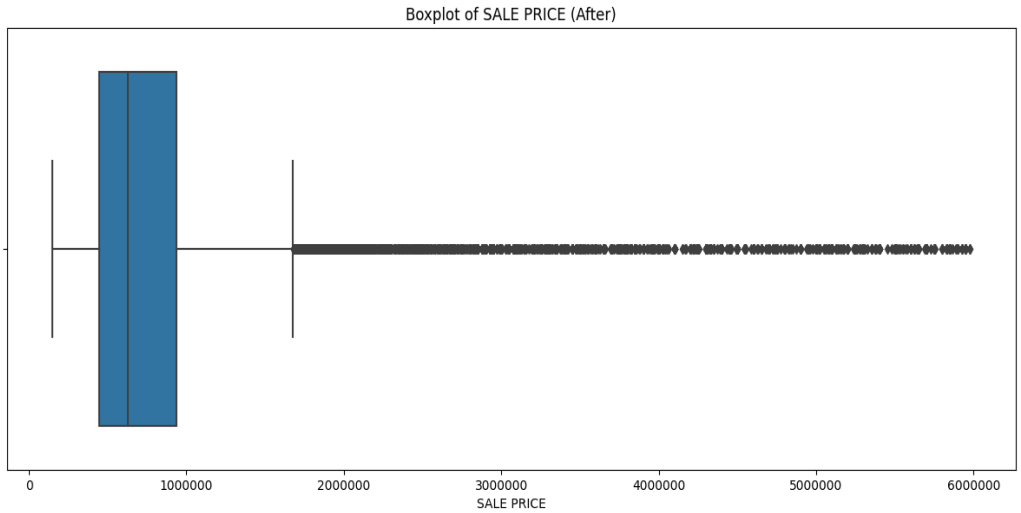
לכן, נבצע ניקוי של מאגר המידע כך שכל דוגמאות האימון חייבות להיות בעלות מחיר בין 150,000 ל-6,000,000.

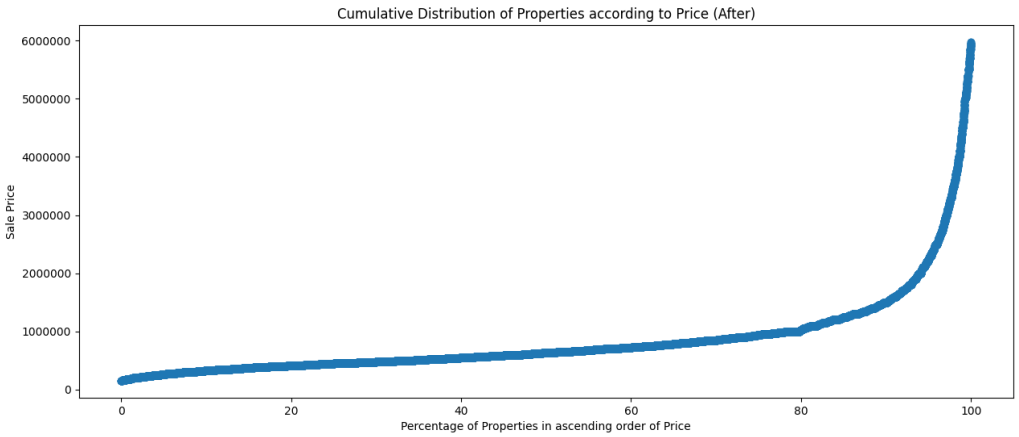
נבצע ניקוי זה על ידי הוספת השורה הבאה לפעולה clean:

```
data = data[(data['SALE PRICE'] > 150000) & (data['SALE PRICE'] <
6000000)]
```

לאחר ניקוי זה, נשארו לנו **26,652 דוגמאות אימון**.

נראה את הגרפים לאחר השינוי.





כפי שניתן לראות, כעת מאגר המידע הרבה יותר נקי, ומתאר מצב יותר אידיאלי אך מציאותי, ללא התייחסות למקרי קיצון מועטים הפוגעים בהתאמת המודל.

לאחר ניקוי זה, ירד ערך פונקציית העלות מ-59,988,637,634,732 ל-206,521,407,805 שזה שגיאה של כ-635,000 לכל דוגמת אימון במקום 10,843,326. זוהי ירידה מאוד משמעותית, אך כעת ננסה להוריד אותה אף יותר.

חשוב להבין שמכאן והלאה הירידה עומדת להיות פחות דרסטית אך עדיין יש מקום לשיפור.

ננסה כעת לעבוד על המידע של המאפיינים עצמם.

ZIP CODE - מיקוד

כעת נמשיך. בתור התחלה נעבור שוב על המידע לאחר הניקוי הנוסף שביצענו.

נעשה זאת באמצעות הפעולה describe של pandas.

```
#seeing the data
data_info = data.describe()
data_info.to_csv(r'C:\Users\Dan\Desktop\ML-NYC\data-info-after.csv')
print(data_info)
```

זה המידע שקיבלנו:

	BOROUGH	BLOCK	LOT	ZIP CODE	RESIDENTIAL UNITS	COMMERCIAL UNITS	TOTAL UNITS	LAND SQUARE FEET	GROSS SQUARE FEET	YEAR BUILT	SALE PRICE
count	26652	26652	26652	26652	26652	26652	26652	26652	26652	26652	26652
mean	3.599242	5684.1	61.85	11013	2.151808	0.217357	2.367702	3636.34	2762.57	1940	846277.6
std	0.961581	3743.4	114.71	503.6	13.63766	13.94722	19.57813	32177.5	25381.9	45.29	752466.8
min	1	5	1	0	0	0	1	200	120	0	150500
25%	3	2892	20	10465	1	0	1	2000	1350	1920	448000
50%	4	5049	39	11223	2	0	2	2500	1828	1930	630000
75%	4	7903.2	65	11375	2	0	2	3880	2531.25	1960	939000
max	5	16319	3597	11694	1844	2261	2261	4228300	3750565	2017	5976252

כפי שניתן לראות עדיין יש בעיה עם הנתונים של המיקוד - ישנם דוגמאות אימון בעלות ערך מיקוד 0. לכן, ננקה דוגמאות האלו.

על פי אתר ששמו משהו, טווח המיקודים בניו-יורק נע בין 10001 ל-14975. לכן ננקה כל דוגמת אימון בה המיקוד לא נמצא בטווח זה.

נעשה זאת על ידי הוספת השורה הבאה לפעולה clean:

```
data = data[(data['ZIP CODE'] > 10000) & (data['ZIP CODE'] <= 14975)]
```

דבר שהוריד את מאגר הנתונים ל-26,643 דוגמאות אימון.

על מנת שיהיה נוח יותר לעבוד עם הנתונים בעמודה זו, נעביר את הערכים של העמודה כך שיתחילו מ-0, נעשה זאת באופן הבא:

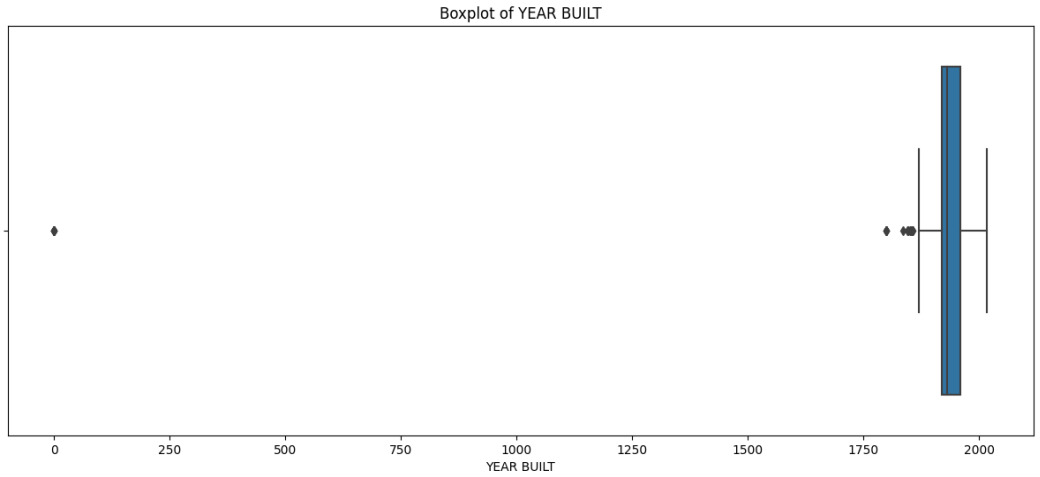
```
data['ZIP CODE'] = data['ZIP CODE'] - 10001
```

YEAR BUILT - שנת בנייה

כעת ננסה לראות את כל השנים בהם נבנו בתים על ידי גרף מסוג Boxplot. נעשה זאת על ידי הקוד הבא:

```
#Set the size of the plot
plt.figure(figsize=(15,6))

# Plot the data and configure the settings
sns.boxplot(x='YEAR BUILT', data=data)
plt.ticklabel_format(style='plain', axis='x')
plt.title('Boxplot of YEAR BUILT')
plt.show()
```

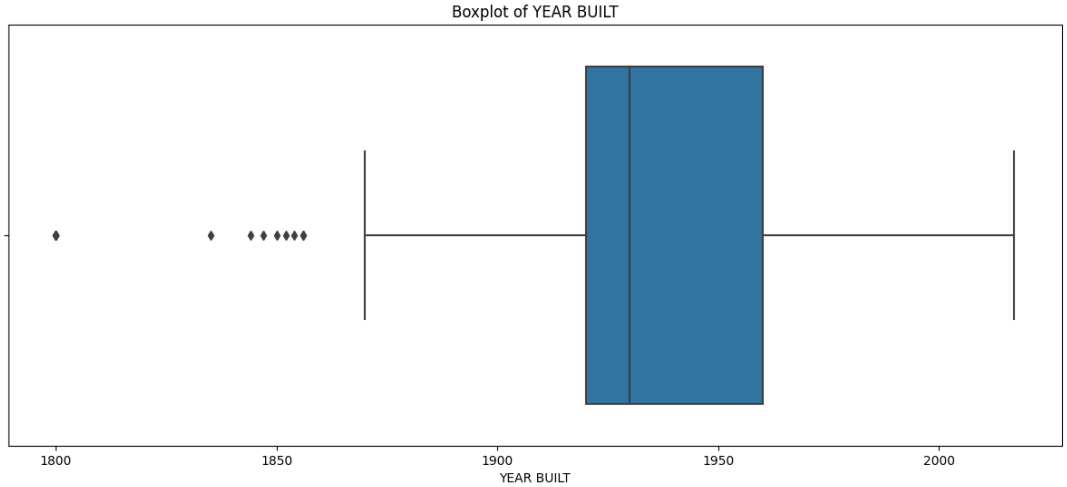
הגרף נראה כך:



כפי שניתן לראות ישנם מספר דוגמאות אימון בעלות ערך YEAR BUILT ששווה לאפס. נתקן זאת על ידי הוספת השורה הבאה:

```
data = data[data['YEAR BUILT'] > 0]
```

לאחר שינוי זה, מספר דוגמאות האימון ירד ל-26635, וניתן לראות כי טווח השנים הוא בין 1800 ל-2017.



לכן עלה הרעיון, למה שלא לשנות את המאפיין ממספר השנה בו נבנה הנכס לגיל הנכס. אמנם שני המאפיינים בעלי מהות דומה, אך השני יותר פרקטי ויותר מתאים בעבור מודל מתמטי.

לכן נוסיף את הקוד הבא:

```
data['BUILDING AGE'] = 2017 - data['YEAR BUILT']
del data['YEAR BUILT']
```

ונציג גרף של regplot בין המחיר לגיל הנכס באמצעות הקוד הבא:

```
#Set the size of the plot
plt.figure(figsize=(15,6))

# Plot the data and configure the settings
sns.regplot(x='BUILDING AGE', y='SALE PRICE', data=data, fit_reg=False,
scatter_kws={'alpha':0.3})
plt.title('BUILDING AGE vs SALE PRICE')
plt.show()
```

הגרף המוצג:

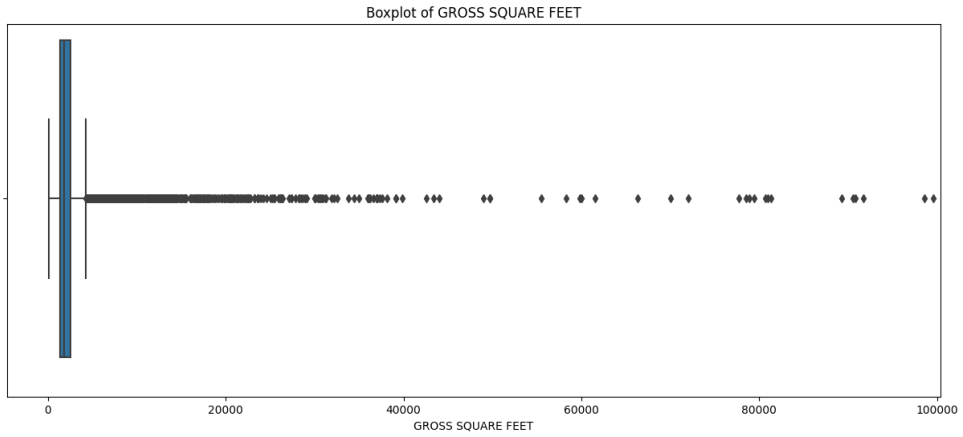
כפי שניתן לראות, כך נוצרת לנו תמונה מציאותית. ככל שנכס ישן יותר, כך הוא נראה יקר יותר. כמו כן, ניתן לשים לב, שהגודל במחיר ביחס לגיל הנכס נראה שגדל באופן פולינומיאלי. לכן יכול להיות שננסה להגדיר זאת במודל הסופי.

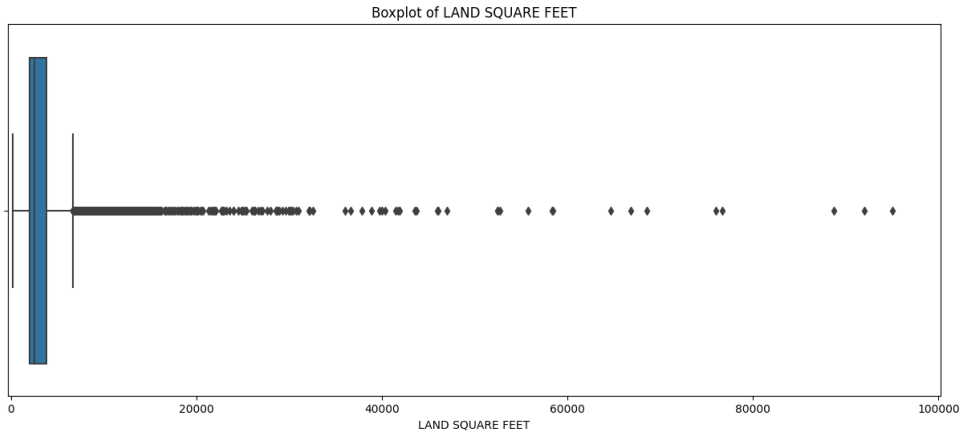
GROSS SQUARE FEET-ו LAND SQUARE FEET

כעת ננסה לראות את ה-Boxplot של שני מאפיינים אלו. נעשה זאת על ידי הקוד הבא:

```
#Set the size of the plot
plt.figure(figsize=(15,6))

# Plot the data and configure the settings
sns.boxplot(x='GROSS SQUARE FEET', data=data)
plt.ticklabel_format(style='plain', axis='x')
plt.title('Boxplot of GROSS SQUARE FEET')
plt.show()

#Set the size of the plot
plt.figure(figsize=(15,6))

# Plot the data and configure the settings
sns.boxplot(x='LAND SQUARE FEET', data=data)
plt.ticklabel_format(style='plain', axis='x')
plt.title('Boxplot of LAND SQUARE FEET')
plt.show()
```

הגרפים נראים כך:





כפי שניתן לראות ישנם דוגמאות אימון מעטות בעלות ערכים גבוהים במאפיינים אלו, לכן נבצע ניקוי.

נתקן זאת על ידי הוספת השורה הבאה:

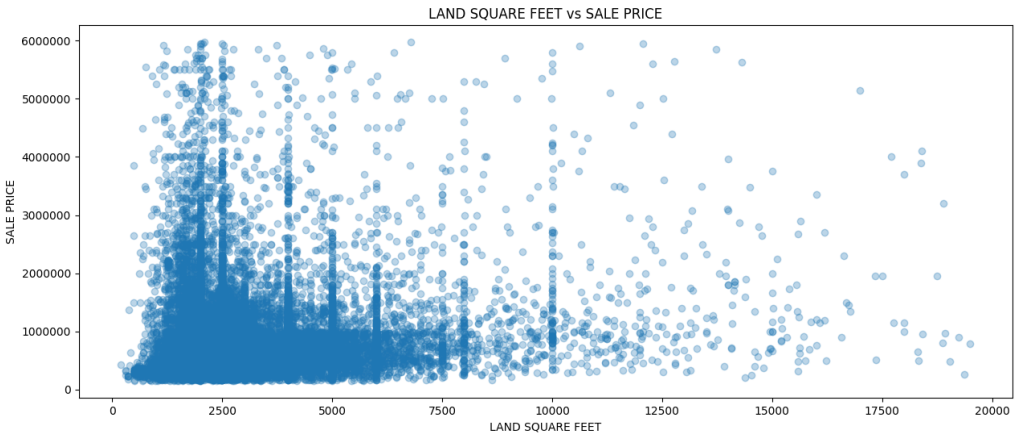
```
data = data[(data['GROSS SQUARE FEET'] < 20000) & (data['LAND SQUARE
FEET'] < 20000)]
```

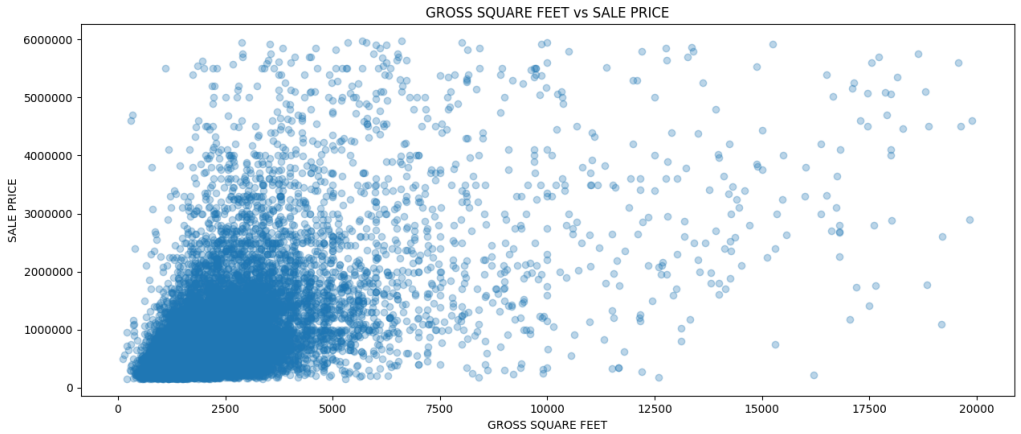
לאחר שינוי זה, מספר דוגמאות האימון ירד ל-26430.

כעת נראה את היחס שבין כל מאפיין למחיר הנדל"ן על ידי גרף מסוג Regplot. על ידי הקוד הבא:

```
#Set the size of the plot
plt.figure(figsize=(15,6))

# Plot the data and configure the settings
sns.regplot(x='GROSS SQUARE FEET', y='SALE PRICE', data=data,
fit_reg=False, scatter_kws={'alpha':0.3})
plt.title('GROSS SQUARE FEET vs SALE PRICE')
plt.show()

#Set the size of the plot
plt.figure(figsize=(15,6))

# Plot the data and configure the settings
sns.regplot(x='LAND SQUARE FEET', y='SALE PRICE', data=data,
fit_reg=False, scatter_kws={'alpha':0.3})
plt.title('LAND SQUARE FEET vs SALE PRICE')
plt.show()
```

הגרפים המוצגים:





מכאן, ניתן לראות תמונה מעניינת, ככל ש-GROSS SQUARE FEET גדל, גם המחיר גדל. כמו כן, LAND SQUARE FEET מצטייר באופן כללי בו ככל שהוא גדל, כך גם המחיר גדל, אמנם יש קפיצה חדה בטווח השטח שבין 0 ל–2500. נראה מה נעשה עם נתונים אלו בהמשך.

RESIDENTIAL UNITS-ו TOTAL UNITS, COMMERCIAL UNITS

כעת נעבור על מאפייני היחידות. בעבור מאפיינים אלו שמתי לב כי מרבית הנכסים לא מגיעים אפילו ל-50 יחידות דיור, כאשר יש מספר דוגמאות אימון בעלות ערך של אלפים. לכן, הגדיר את הטווח באמצעות הקוד הבא:

```
data = data[(data['TOTAL UNITS'] > 0) & (data['TOTAL UNITS'] < 50)]
```

ונגדיר שסך כל היחידות חייב להיות שווה לסכום יחידות הדיור והיחידות המסחריות. נעשה זאת בקוד הבא:

```
data = data[(data['TOTAL UNITS'] == data['COMMERCIAL UNITS'] +
data['RESIDENTIAL UNITS'])]
```

BUILDING CLASS - סוג בניין

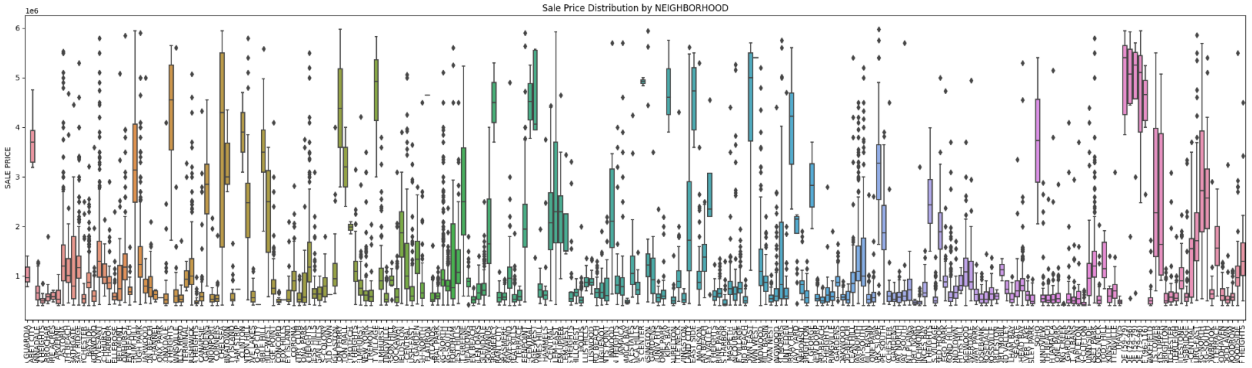
ננסה לראות מה ניתן להבין בנוגע למאפיין BUILDING CLASS AT TIME OF SALE או שנקרא לו בקיצור BUILDING CLASS. לשם כך נציג את הגרף שלו:

בגרף אני מציג רק את הסוגים הנמצאים במאגר המידע, אם הסוג לא קיים, אין מידע להציג בשבילו ואין צורך להציגו בגרף.

לצערי, הגרף לא מציג לי מידע חדש אותו לא ידעתי אשר אוכל להשתמש בו, לכן אני עובר למאפיין הבא.

NEIGHBORHOOD - שכונה

ננסה לראות מה ניתן להבין בנוגע למאפיין NEIGHBORHOOD. לשם כך נציג את הגרף שלו:

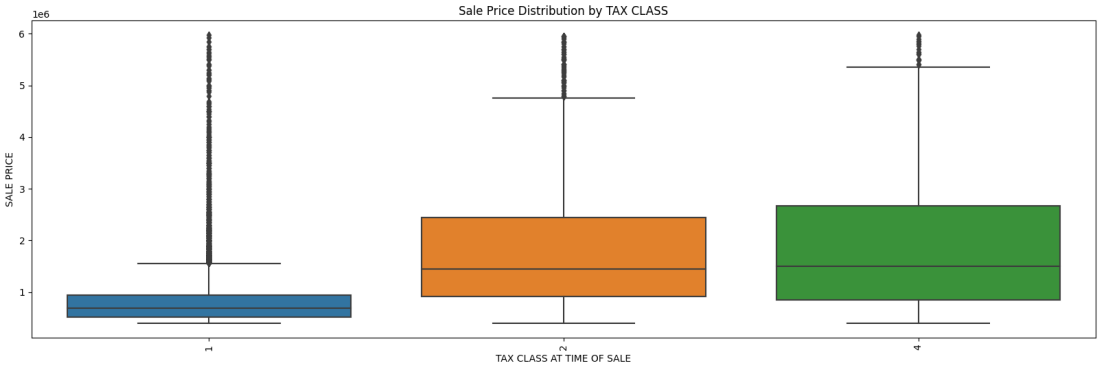


בגרף אני מציג רק את השכונות הנמצאות במאגר המידע, אם הסוג לא קיים, אין מידע להציג בשבילו ואין צורך להציגו בגרף.

לצערי, הגרף עמוס מדי ולא מציג לי מידע חדש אותו לא ידעתי אשר אוכל להשתמש בו, לכן אני עובר למאפיין הבא.

TAX CLASS - קטגוריית מס על הנכס

ננסה לראות מה ניתן להבין בנוגע למאפיין TAX CLASS. לשם כך נציג את הגרף שלו:

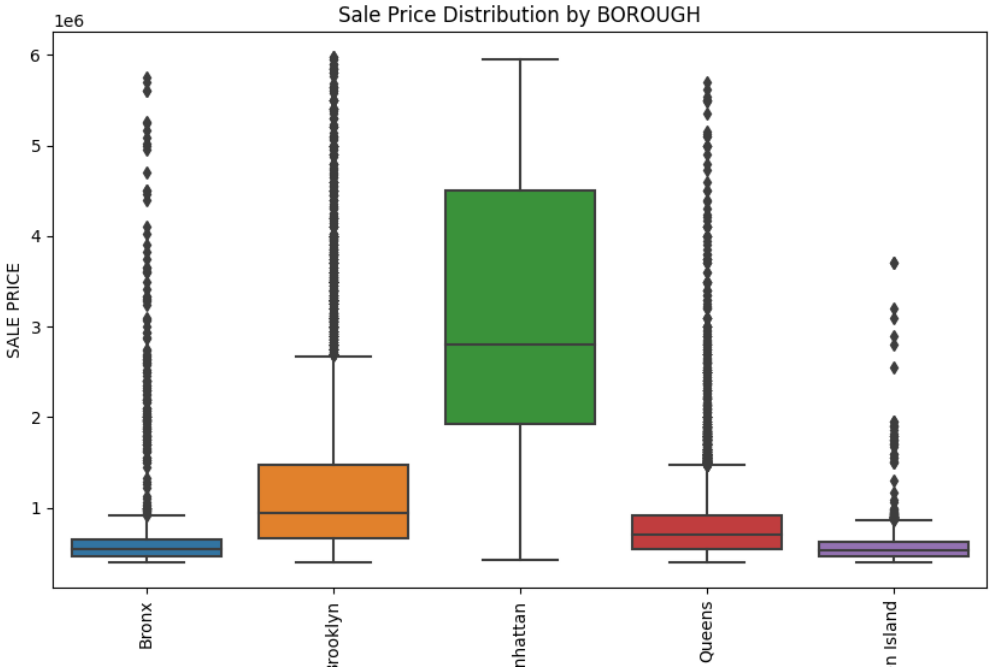


בגרף אני מציג רק את הקטגוריות הנמצאות במאגר המידע, אם הסוג לא קיים, אין מידע להציג בשבילו ואין צורך להציגו בגרף.

לצערי, הגרף לא מציג לי מידע חדש אותו לא ידעתי אשר אוכל להשתמש בו, לכן אני עובר למאפיין הבא.

BOROUGH - מחוז

ננסה לראות מה ניתן להבין בנוגע למאפיין BOROUGH. לשם כך נציג את הגרף שלו:



הגרף מציג לי תמונה מעניינת, בעבור כל מחוז, ניתן לראות טווח מחירים עיקרי שונה. נראה מה יהיה אפשר לעשות עם מידע זה בהמשך.

2.4.6.4 הגדרת מאפיינים - הפרדת מאפיינים קטגוריאלים

על ידי התבוננות על המידע, הבנתי בעיה הקיימת במודל. כעת, יש לנו מספר מאפיינים שהם קטגוריות: קטוגריית מס, שכונה, קטגוריית בניין. כל אלו עובדים באופן בו מוגדר מאפיין אחד, בו לכל קטגוריה ערך משלה.

אך, כך נוצרת בעיה מתמטית. על ידי כך שאנחנו נותנים לקטגורייה מסוימת ערך גדול יותר מאחרת, אנו אומרים שבעבור אותו טטה, קטגוריה אחת תקבל מחיר גבוה יותר אף שזה לא בהכרח נכון. בסופו של דבר, הרי המודל הוא שצריך למצוא בעבור איזה מאפיין ניתן יתקבל מחיר גבוה יותר.

הפתרון אותו מצאתי הוא פשוט. כל מאפיין קטגוריאלי - בעל מספר קטגוריות, נחלק למספר מאפיינים בוליאנים. כך, רק הטטה הוא שישפיע על היחס שבין כל קטגוריה למחיר הנדל"ן, במקום יחס למאפיין של כלל הקטגוריות. כך, רק הטטה ישפיע על מחיר הנדל"ן כפי שצריך להיות.

נעדכן את clean באופן הבא:

הקוד ייצור מאפיינים חדשים אך ורק לקטגוריות הקיימות במאגר המידע שיקבל, כיוון שאין צורך במאפיין שכולו מלא באפסים.

```
 # The Categorical features that need to be seperate
one_hot_features = ['BOROUGH', 'TAX CLASS AT TIME OF SALE',
'NEIGHBORHOOD', 'BUILDING CLASS AT TIME OF SALE']
```

תחילה, הגדרנו מי הם המאפיינים אשר צריכים להתפצל. הכנסנו את שמותיהם ל one_hot_features.

```
# For each categorical column, find the unique number of categories.
This tells us how many columns we are adding to the dataset.
longest_str = max(one_hot_features, key=len)
total_num_unique_categorical = 0
for feature in one_hot_features:
      num_unique = len(data[feature].unique())
      print('{col:<{fill_col}} : {num:d} unique categorical
values.'.format(col=feature, fill_col=len(longest_str),num=num_unique))
      total_num_unique_categorical += num_unique
      print('{total:d} columns will be added during one-hot
encoding.'.format(total=total_num_unique_categorical))
```

לאחר מכן, אנו מדפיסים כמה קטגוריות יחודיות יש לכל מאפיין וכמה מאפיינים חדשים יווצרו.

```
one_hot_encoded = pd.get_dummies(data_model[one_hot_features])
one_hot_encoded.info(verbose=True, memory_usage=True, null_counts=True)
```

פירקנו את המאפיינים למאפיינים נפרדים.

```
# Delete the old columns
data_model = data_model.drop(one_hot_features, axis=1)

# Add the new one-hot encoded variables
data_model = pd.concat([data_model, one_hot_encoded], axis=1)
data_model.head()
data_model['SALE PRICE'] = data['SALE PRICE']

# Saving the changes
data = data_model
```

לבסוף, מוחקים את המאפיינים באופן הישן שלהם, מוסיפים את החדשים אל המאגר הכללי, ומעדכנים את המאגר.

מאפיינים חדשים יופיעו במאגר המידע כך:

FEATURE NAME_SPECIFIC-CATEGORY

משמאל שם המאפיין הכללי, ומימין הקטגוריה הספציפית אותה הוא מייצג.

הערכים בעמודות אלו יהיו 0 ו-1 בדומה למשתנה בוליאני - 0 אם הוא לא בקטגוריה, 1 אם הוא כן.

2.4.6.5 חישוב שגיאה ממוצעת, סיקור ביצועים ושיפור חיזוי המודל

על מנת לבדוק את המודל, יצרתי שתי פונקציות: run_check ו-average_error.

הראשונה, בודקת את המודל בכך שמחפשת מהו אחוז הנכסים במאגר המידע בהם אחוז השגיאה קטן מ-50%, 30%, 20%, 10% ו-5%.

השנייה, מחשבת את אחוז השגיאה הממוצעת של המודל, ואת מחיר השגיאה הממוצע.

כעת, אציג את הקוד של כל אחת. נתחיל ב-**run_check**:

הפעולה, יוצרת מילון בו עבור כל מפתח המהווה את האחוז - ערכו הוא מונה דוגמאות האימון שאחוז שגיאותיהם פחות מערך זה.

הפעולה עוברת דוגמה דוגמה, ומעלה את המונה בהתאם עבור כל אחוז.

```
def run_check(self, theta):
    # dictionary to save the amount of encounters per percentage
    dict_below = {}

    # list of the percentage numbers
    below_percent_lst = [50, 40, 30, 20, 10, 5]

    # initializing dictionary
    for e in below_percent_lst:
        dict_below[e] = 0

    # amount of training examples
    m = np.size(self.features, 0)

    # going through all the training sets
    for row in range(m):
        # x vector
        x = self.features[row]
        error_size = self.error(x, self.y[row], theta)

        h = self.hypothesis(x, theta)
        #going through all the percentages
        for percent in below_percent_lst:
```

```
                if error_size < pow(percent/100 * h, 2):
                        dict_below[percent] += 1

        # print all the encounters per percentage
        for key in dict_below:
            print('difference below {col}% : {num} from {amount}
training example, ({global_percent}%).'.format(col=key,
num=dict_below[key], amount = m, global_percent = dict_below[key]/m *
100))
```

הפעולה עוברת על כל דוגמאות האימון. בכל מעבר, היא בוחנת האם הדוגמת אימון נמצאת מתחת לאחוז מסויים, אם כן, היא מוסיפה 1 למונה של אותו האחוז (שמור במילון dict_below).

לבסוף, מדפיסה עבור כל אחוז, מהי כמות דוגמאות האימון התואמות לקריטריון, וכמה אחוז זה מהווה מכלל דוגמאות האימון.

כעת, אציג את הקוד של **average_error**:

הפעולה, יוצרת DataFrame חדש, שבו ישמר אחוז השגיאה והמחיר של כל נכס.

הפעולה עוברת על כל דוגמאות האימון אחת אחת, ומחשבת את אחוז השגיאה הממוצעת של כלל דוגמאות האימון, ואת הפרש המחיר הממוצע של כלל דוגמאות האימון.

לבסוף, היא מדפיסה נתונים אלו, ומציגה גרף של אחוז השגיאה ביחס למחיר.

```
def average_error(self, theta):
        '''
        This function returns the average error of the model.
        In percentage and in price
        '''
        error_data = pd.DataFrame({'ERROR %' : [], 'PRICE' : []})

        #initializing average error
        av_error_percent = 0
        av_price_error = 0

        # amount of training examples
        m = np.size(self.features, 0)
```

```
        #going through all the training sets
        for row in range(m):
            # x vector
            x = self.features[row]
            error_size = self.error(x, self.y[row], theta)
            error_sqrt = math.sqrt(error_size)
            av_price_error += error_sqrt

            av_error_percent += error_sqrt / self.y[row]

            a_row = {'ERROR %' : error_sqrt / self.y[row] * 100,
'PRICE' : self.y[row]}
            error_data = error_data.append(a_row, ignore_index = True)

        av_error_percent /= m
        av_price_error /= m

        print(error_data)
        print('average error percentage :
{num}%'.format(num=av_error_percent * 100))
        print('average error in price :
{num}'.format(num=av_price_error))

        Graph.regplot(error_data,'ERROR %', 'PRICE')

error_data.to_csv(r'C:\Users\Dan\Desktop\ML-NYC\error_data.csv')
        return (av_error_percent, av_price_error)
```

כעת, נעזר בפונקציות אלו, על מנת לנתח את המצב הקיים.

לאחר הרצת run_check קיבלנו את התוצאה הבאה:

```
difference below 50% : 23453 from 26407 training example, (88.81357215889726%).
difference below 40% : 21689 from 26407 training example, (82.13352520165108%).
difference below 30% : 18860 from 26407 training example, (71.4204566970879%).
difference below 20% : 14444 from 26407 training example, (54.69761805581853%).
difference below 10% : 8010 from 26407 training example, (30.332866285454614%).
difference below 5% : 4177 from 26407 training example, (15.817775589805732%).
```

תוצאה זו לא רעה, אך לא מספיקה, כ12% מדוגמאות האימון הם בשגיאה מעל 50%!, כאשר רק כ-70% בשגיאה של כ-30%. מעבר לזה, השגיאה הממוצעת היא על כ-31%, והפרש המחיר הממוצע הוא כ-210,000. תוצאה זו לא רעה, אפילו טובה אבל ברצוני לשפר את המודל אפילו יותר.

```
average error precentage : 31.31453572594589%
average error in price : 210302.02334759463
```

אך חשוב להבין, המטרה היא לא להגיע לפחות מכ-15%, כי אז עלול להיות מצב של התאמת יתר אשר יפגע אפילו יותר בחיזוי בעבור דוגמאות אימון חדשות.

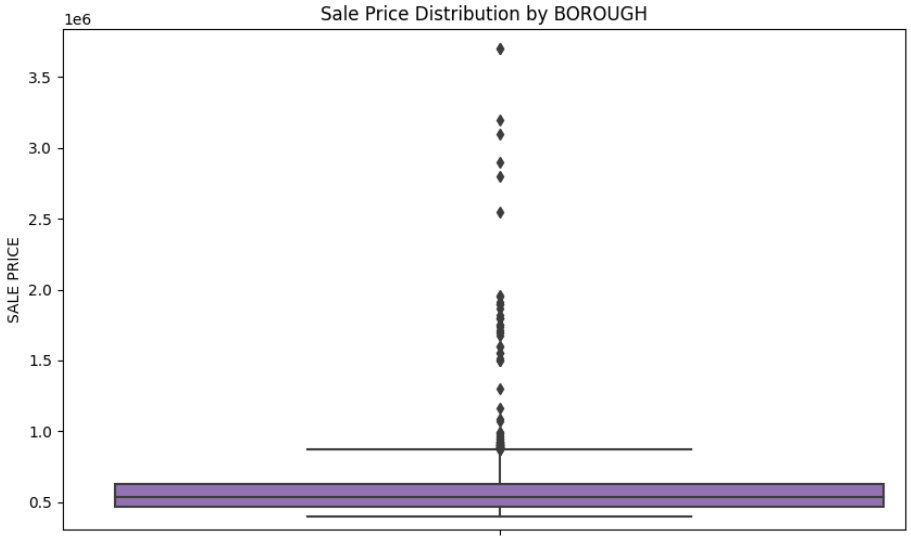
לכן עלה לי רעיון נוסף לניקוי, והוא שבכל מחוז יש טווח מחירים עיקרי, אשר השאר פוגעים בחיזוי המודל. לכן, על ידי ניתוח טווחים אלו וניקויים, בתקווה נוביל לשיפור משמעותי וניכר.

לכן ננתח כעת כל מחוז, נעבור מחוז מחוז וננסה להבין את הטווח המתאים לו.

נתחיל ב-ברונקס (Bronx)

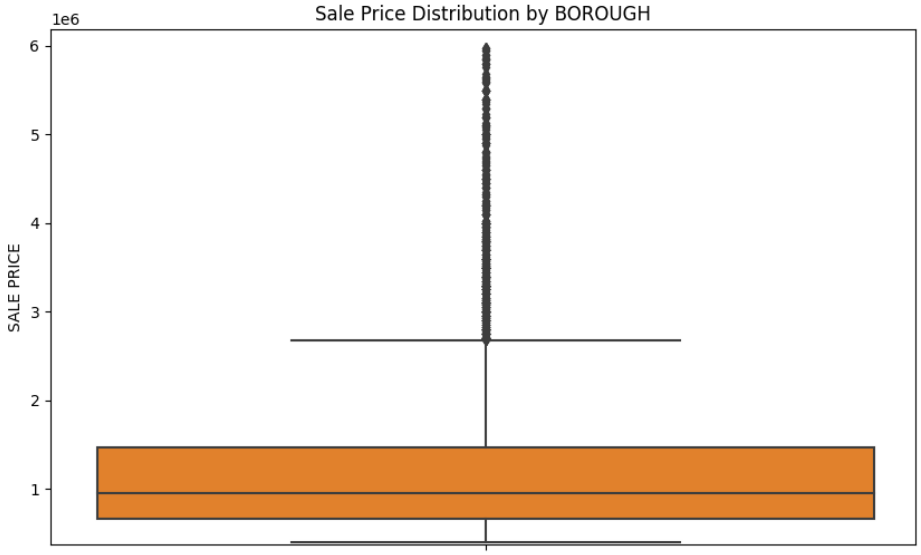
כפי שניתן לראות טווח המחירים הראשי של Bronx נע עד 1,000,000, ניתן לו טווח של עד 3,000,000 כדי לא למחוק יותר מדי דוגמאות אימון.

נמשיך ב-סטייטן איילנד (Staten Island)



כפי שניתן לראות טווח המחירים הראשי של Staten Island נע עד 1,000,000, ניתן לו טווח של עד 2,000,000 כדי לא למחוק יותר מדי דוגמאות אימון.

נמשיך ב-ברוקלין (Brooklyn)

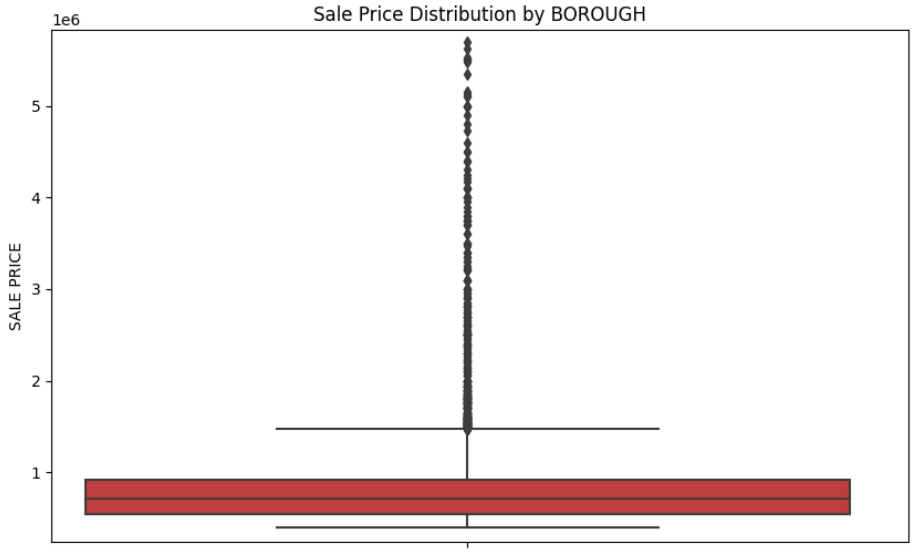


כפי שניתן לראות טווח המחירים הראשי של Brooklyn נע עד 3,000,000, ניתן לו טווח של עד 4,000,000 כדי לא למחוק יותר מדי דוגמאות אימון.

נמשיך ב-מנהטן (Manhattan)

כפי שניתן לראות טווח המחירים הראשי של Manhattan נע בעיקר מ-1,500,000. נכסים מתחת למחיר זה במנהטן לא ממש הגיוניים.

נסיים ב-קווינס (Queens)



כפי שניתן לראות טווח המחירים הראשי של Queens נע בעיקר עד 1,500,000. ניתן לו טווח של עד 3,000,000 כדי לא למחוק יותר מדי דוגמאות אימון.

בשביל לוודא שלא מחקתי מספר דוגמאות אימון רב מדי, כך שכבר פגעתי עד כדי כך שניקיתי נתונים שהם לא 'נתוני רעש', כתבתי קוד קצר לבדיקת כמות הדוגמאות שמחקנו מכל מחוז:

תחילה מגדירים את כלל הערכים, בעבור מנהטן - הערך מעליו יהיו כל הנכסים, בעבור השאר - הערך מתחתיו יהיו כל הנכסים. שומרים ערכים אלו במשתנים num הממוספרים מ1 עד 5.

```
num1,num2,num3,num4,num5 = (1500000, 3000000, 4000000, 3000000,
2000000)
```

לאחר מכן, שומרים במשתנים len, הממוספרים מ1 עד 5 בהתאם לכל מחוז, את כמות הנכסים מתחת או מעל כל טווח.

```
len1 = len(data[(data['BOROUGH'] == 'Manhattan') & (data['SALE PRICE']
<= num1)])
len2 = len(data[(data['BOROUGH'] == 'Bronx') & (data['SALE PRICE'] >=
num2)])
len3 = len(data[(data['BOROUGH'] == 'Brooklyn') & (data['SALE PRICE']
>= num3)])
len4 = len(data[(data['BOROUGH'] == 'Queens') & (data['SALE PRICE'] >=
num4)])
len5 = len(data[(data['BOROUGH'] == 'Staten Island') & (data['SALE
PRICE'] >= num5)])
```

לבסוף מדפיסים את התוצאה:

```
print('''
      Manhattan below {num1}: {ans1}
      Bronx above {num2}: {ans2}
      Brooklyn above {num3}: {ans3}
      Queens above {num4}: {ans4}
      Staten Island above {num5}: {ans5}''').format(num1=num1,num2=num2,
      num3=num3,num4=num4,num5=num5,ans1=len1,ans2=len2,ans3=len3,ans4=
len4,ans5=len5))
```

תוצאת הבדיקה:

```
Manhattan below 1500000: 58
Bronx above 3000000: 38
Brooklyn above 4000000: 153
Queens above 3000000: 81
Staten Island above 2000000: 7
```

כפי שניתן לראות, ירדה כמות קטנה של דוגמאות אימון.

הקוד להגבלת הטווח:

```
# data for every Borough
manhattan_data = data[(data['BOROUGH'] == 'Manhattan') & (data['SALE
PRICE'] >= num1)]
bronx_data = data[(data['BOROUGH'] == 'Bronx') & (data['SALE PRICE'] <=
num2)]
brooklyn_data = data[(data['BOROUGH'] == 'Brooklyn') & (data['SALE
PRICE'] <= num3)]
queens_data = data[(data['BOROUGH'] == 'Queens') & (data['SALE PRICE']
<= num4)]
staten_island_data = data[(data['BOROUGH'] == 'Staten Island') &
(data['SALE PRICE'] <= num5)]

dataframes = [manhattan_data, bronx_data, brooklyn_data, queens_data,
staten_island_data]

# connecting the data for every borough
data = pd.concat(dataframes, axis=0)
```

הקוד שומר את מאגר המידע המוגבל בטווח למשתנים חדשים, ולבסוף מחבר אותם באמצעות הפעולה concat של pandas.

כמו כן, לאחר שהגבלנו את הטווחים, הבנתי על בעיה נוספת הנבעה מהגרף. ככל שהמחיר נמוך יותר, כך אחוז השגיאה גבוה יותר. תופעה זו הגיונית, כי אם הפרש המחירים הממוצע הוא כ-200,000 אז המחיר אז בבתים שמחירם גבוהה, הפרש זה יהווה אחוז הרבה יותר קטן, מנכסים יותר זולים.

יש פתרון לתופעה זו כך שהמודל יחזה טוב יותר והוא מיקוד המחיר.
חשוב להבין, שהתופעה היא לא בעיה חמורה במצב רגיל, ואפילו אם היינו משאירים את המודל כך היה המודל נחשב למוצלח מאוד. אבל מיקוד המחירים במקרה זה הוא על מנת לנקות נתונים לא נכונים. בניו יורק, נכסים שמחירם פחות מ-400,000 ואפילו יותר, לא מאוד נפוצים על גבול החוסר קיום. מטרתנו היא לתקן את השגיאה הזו בנתונים של מאגר המידע.

מיקוד המחיר יהווה תוספת לשיפור. התחלנו את מיקוד המחירים כבר בתיקון הקודם בו הגבלנו את טווחי המחירים בכל מחוז. אך עדיין נשאר המצב שבו בניו יורק, נכסים שמחירם פחות מ-400,000 ואפילו יותר, לא מאוד נפוצים על גבול החוסר קיום.

כפי שלאחר מכן שמתי לב, מעבר לכך שמחירים אלו לא סבירים, דוגמאות האימון במחירים אלו, היו בחלקם גם בעלי שטח עצום עד מצב שלא הגיוני. לכן, נבצע מספר שינויים שיתקנו מצב זה.

נקטין את טווח המחירים כך שיתחיל מ-400,000 באמצעות השינוי הבא בקוד:

```
data = data[(data['SALE PRICE'] > 400000) & (data['SALE PRICE'] <
6000000)]
```

וניצור הגבלה של מחיר ביחס לשטח בבתים שמחירים נמוך מאוד.

```
# limiting the price-square feet ratio

data = data[(data['SALE PRICE'] < 1500000) & (data['GROSS SQUARE FEET']
< 5000) & (data['LAND SQUARE FEET'] < 5000)].append(data[(data['SALE
PRICE'] >= 1500000)])
```

2.4.6.6 סיכום ביצועים

בפרק זה, אסכם סופית את הביצועים של המודל.

בסוף, החלטתי לוותר על מספר מאפיינים והשארתי אך ורק את המאפיינים הבאים:

- מאפיינים אותם פיצלתי - מחוז, קטגוריית בניין, קטגוריית מס, שכונה.
- מאפיינים רגילים - יחידות מסחריות, יחידות מגורים, גיל הבניין, שני מאפייני השטחים, ומספר המיקוד.

לאחר כל השינויים, הגיע המודל לביצועים כאלה:

```
difference below 50% : 17980 from 18892 training example, (95.17255981367775%).
difference below 40% : 17214 from 18892 training example, (91.11793351683252%).
difference below 30% : 15688 from 18892 training example, (83.04044039805208%).
difference below 20% : 12517 from 18892 training example, (66.25555790810925%).
difference below 10% : 7228 from 18892 training example, (38.25958077493119%).
difference below 5% : 3787 from 18892 training example, (20.045521914037685%).
```

פחות מ-5% מכלל דוגמאות האימון בעלות שגיאה מעל 50 אחוז. כ-90% מכלל דוגמאות האימון בשגיאה של פחותה מ-40%. כ-85% מכלל דוגמאות האימון בשגיאה פחותה מ-30%! לפני כן, השינויים האחרונים, המצב היה שכ-20% מכלל דוגמאות האימון היו בשגיאה של יותר מ-50%. כפי שניתן לראות המצב השתפר מאוד.

כמו כן, השגיאה הממוצעת עברה לפחות מ-20%! שיפור מאוד משמעותי של כ-12% מתת הפרק הקודם, והפרש המחיר הממוצע עבר ל-180,000!

```
average error percentage : 19.66766576458657%
average error in price : 180351.1369547054$
```

כמו כן, אם נראה את אחוז השגיאה ביחס למחיר, שהיחס בין המחיר לאחוז השגיאה הוא טוב מאוד.

בנוסף לכך, למדנו מספר דברים.

למשל, המחיר המינימלי בו אנחנו בוחרים משפיע מאוד. כפי שמצאנו, יש הרבה מידע לא מדויק כשמדובר בנכסים שמחירם נמוך. זאת מכיוון שנכסים במחירים נמוכים לא מאוד נפוצים בניו יורק, ובמאגר המידע הם מייצגים נתוני 'רעש' - דוגמאות אימון בעלות מידע שככל הנראה הייתה בו טעות, או לא מייצגים את המחירים האמיתיים של הבתים.

לכן, כשהמודל שלנו, אשר בעבורו המאפיינים החשובים ביותר הם ה-LAND SQUARE FEET ו-GROSS SQUARE FEET, רואה את ערכים אלו גבוהים, הוא מחזה מחיר גבוה מאוד, על אף שזה לא תואם את הדוגמאות אימון.

אם המודל היה מקבל יותר בתים במחירים אלו שבהם ערכי מאפיינים אלו גבוהים, הוא היה מחזה נכון יותר עבור מחירים אלו, אך כיוון שאין הרבה מקרים כאלה - ניתן להסיק שאלו נתוני רעש ויש לנקות אותם. לכן, ניקינו דוגמאות אימון אלו המהוות שגיאות סטטיסטיות.

הרעיון העיקרי של המודל הוכח - ככל שהמודל יקבל יותר נתונים נקיים, הוא ילמד נכון יותר וכך התוצאות שלו יהיו יותר ויותר מדויקות. לאחר ניקוי יסודי מצדנו, הצלחנו לפתור את בעיית התאמת החסר שהייתה לנו, ולשפר רבות את חיזויי המודל עד למצב של אחוז שגיאה ממוצעת של פחות מ-20%!

2.5 ייעול וסידור העבודה - הקוד הסופי לאחר שימוש בטכניקות שונות

2.5.1 קבועים ושילוב ספריות

חלק זה הוא תחילת הקוד, בו זימנתי את כלל הספריות והגדרתי מספר קבועים חשובים.

שילוב הספריות והמודולים בקוד:

```
from asyncio.windows_events import NULL
import csv
from distutils.command.build import build
import os
from matplotlib.style import use
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pyparsing import col
import seaborn as sns
import time
import math
import PySimpleGUI as sg
import os.path
import copy
import itertools
```

הקבועים השונים:

```
LARGE_NUM = 10000000000000000
BUILDING_CLASSES = ['A0','A1','A2','A3','A4','A5','A6','A7','A8','A9',
                    ...
                    ...
                    ...
                    'Z0','Z1','Z2','Z3','Z4','Z5','Z7','Z8','Z9']
BOROUGH = ['manhattan', 'bronx', 'brooklyn', 'queens', 'staten island']
```

הגדרתי מספר קבועים להם נזדקק בעבודה.

- LARGE_NUM - כפי שהשם מרמז, הקבוע שומר בעצמו מספר גדול מאוד. זהו מספר בו נשתמש באלגוריתם בחירת המודל בכך שנשווה את ערך העלות ההתחלתי איתו. זאת ניתן יהיה לראות בהמשך.
- BUILDING_CLASSES - רשימה השומרת את כלל קטגוריות הבניין השונות הקיימות. נשתמש בה בשביל לבצע בדיקה האם המידע שיוכנס בעבור קטגוריית הבניין על ידי המשתמש, הוא קטגוריית בניין אפשרית.
- BOROUGH - רשימה השומרת את כלל המחוזים השונים. נשתמש ברשימה זו בשביל לבדוק אם מחוז שהמשתמש הכניס הוא מחוז קיים.

2.5.2 המחלקה Graph

מחלקה זו נועדה על מנת לספק את כל הגרפים הנחוצים לצורך חקירת המודל.

המחלקה כוללת את הפונקציות הבאות:

- **2.5.2.1 Boxplot** - גרף המתאר בעבור המאפיין את קבוצות היישוב, ההתפשטות וההטיה של נתונים מספריים דרך הרבעונים שלהם. כך למשל, אם ניקח מאפיין מתוך מאגר הנתונים ונציג Boxplot שלו, נראה את כמות דוגמאות האימון בעבור כל ערך של המאפיין.

הקוד:

```
def boxplot(data, column, add_to_title = NULL):
      '''
      Boxplot creation function
      '''
      # Setting the size of the plot
      plt.figure(figsize=(15,6))

      column = str(column)

      # Plot the data and configure the settings
      sns.boxplot(x=column, data=data)
      plt.ticklabel_format(style='plain', axis='x')
      if(add_to_title != NULL):
            plt.title('Boxplot of ' + column + ' ' + add_to_title)
      else:
            plt.title('Boxplot of ' + column)
      plt.show()
```

הפונקציה מקבלת את הפרמטרים הבאים:

- data - מאגר המידע, הפרמטר מסוג DataFrame של הספרייה pandas.
- column - שם המאפיין בעבורו אנחנו רוצים את הגרף, הפרמטר מסוג מחרוזת.
- add_to_title - מידע אותו אנחנו רוצים להוסיף לכותרת, הפרמטר מסוג מחרוזת.

הפונקציה מגדירה את גודל המסך בו יוצג הגרף לגודל של 15 על 6, ולאחר מכן, יוצרת גרף boxplot מהספרייה Seaborn. לבסוף מציגה את הגרף.

- **2.5.2.2 Cumulative Distribution (גרף התפלגות)** - גרף בו ציר ה-X הוא האחוז וציר ה-Y הוא ערכי המאפיין. כך בעבור כל אחוז מוצג הערך תחתיו נמצאים אחוז זה מהמקרים.

הקוד:

```
def cumulative_distribution(data, column, add_to_title = NULL):
      '''
      Cumulative Distribution creation function
      '''
      # Setting the size of the plot
      plt.figure(figsize=(15,6))

      column = str(column)

      # Get the data and format it
      x = data[[column]].sort_values(by=column).reset_index()
      x['PROPERTY PROPORTION'] = 1
      x['PROPERTY PROPORTION'] = x['PROPERTY PROPORTION'].cumsum()
      x['PROPERTY PROPORTION'] = 100* x['PROPERTY PROPORTION'] /
len(x['PROPERTY PROPORTION'])

      # Plot the data and configure the settings
      plt.plot(x['PROPERTY PROPORTION'],x[column], linestyle='None',
marker='o')

      if(add_to_title != NULL):
            plt.title('Cumulative Distribution of Properties according
to ' + column + ' ' + add_to_title)
      else:
            plt.title('Cumulative Distribution of Properties according
to ' + column)
      plt.xlabel('Percentage of Properties in ascending order of ' +
column)
      plt.ylabel(column)
      plt.ticklabel_format(style='plain', axis='y')
      plt.show()
```

הפונקציה מקבלת את הפרמטרים הבאים:

- data - מאגר המידע, הפרמטר מסוג DataFrame של הספרייה pandas.

- column - שם המאפיין בעבורו אנחנו רוצים את הגרף, הפרמטר מסוג מחרוזת.
- add_to_title - מידע אותו אנחנו רוצים להוסיף לכותרת, הפרמטר מסוג מחרוזת.

הפונקציה מגדירה את גודל המסך בו יוצג הגרף לגודל של 15 על 6. לאחר מכן, מגדירה את ציר הערכים של ציר ה-X להיות אחוזים המייצגים את אחוז המידע המגיע הנמצא מתחת לערך ה-Y של אותה נקודה. לבסוף מציגה את הגרף.

- **Regplot 2.5.2.3** - גרף רגיל המתאר את כלל הנקודות עבור שני ערכים שונים. עבור כל ערך X את ערכי ה-Y השונים.

הקוד:

```
def regplot(data, y_title, x_title, add_to_title = NULL):
      '''
      Regplot creation function
      '''
      # Setting the size of the plot
      plt.figure(figsize=(15,6))

      # Plot the data and configure the settings
      plt.ticklabel_format(style='plain', axis='y',useOffset=False)
      plt.ticklabel_format(style='plain', axis='x',useOffset=False)
      sns.regplot(x=x_title, y=y_title, data=data, fit_reg=False,
scatter_kws={'alpha':0.3})
      if(add_to_title != NULL):
            plt.title(x_title + ' vs ' + y_title + ' ' + add_to_title)
      else:
            plt.title(x_title + ' vs ' + y_title)
      plt.show()
```

הפונקציה מקבלת את הפרמטרים הבאים:

- data - מאגר המידע, הפרמטר מסוג DataFrame של הספרייה pandas.
- y_title - שם המאפיין שיהיה על ציר ה-Y, הפרמטר מסוג מחרוזת.
- x_title - שם המאפיין שיהיה על ציר ה-X, הפרמטר מסוג מחרוזת.

הפונקציה מגדירה את גודל המסך בו יוצג הגרף לגודל של 15 על 6. לאחר מכן, יוצרת גרף regplot מתוך הספריה Seaborn בעבור ערכי ה-X וה-Y שהגדרנו. לבסוף הפונקציה מציגה את הגרף.

2.5.3 המחלקה Model

כאן אציג את המחלקה לאחר השינויים שביצענו בה, כולל הפעולות clean וfeature_prepare ועוד. בין היתר, פעולות רבות עודכנו כך שיעבדו אף יותר יעיל על ידי חישובים של מטריצות ווקטורים בניגוד למעבר על ידי לולאות וכדומה.

2.5.3.1 פעולה בונה __init__

```
def __init__(self, data, filename):
        # This is a constructor for the Model object
        self.file_dir = filename

        # setting all the Object members
        self.feature_prepare(data)

        # initializing the theta vector
        self.theta = np.ones(np.size(self.features, 1))
        #debug print
        self.debug_print()
```

זוהי הפעולה הבונה של מחלקת המודל. הפעולה, מקבלת שני פרמטרים:

- data - מאגר הנתונים עליו יתאמן המודל, שהוא מסוג DataFrame של pandas.
- filename - שם הקובץ של מאגר המידע, יחד עם ה-Directory המלא אליו. פרמטר זה ישמש אותנו לשמירת כל המידע הנחוץ במהלך ריצת הקוד באותה תיקיה יחד עם קובץ מאגר המידע ובשם דומה.

הפעולה מכינה את כלל התכונות לה היא זקוקה. חלק בפעולה feature_prepare, בה היא מנקה את מאגר המידע ומסדרת אותו. כך שבסוף מגדירה את התכונות הבאות:

- all_data - מטריצה מסוג ndarray של NumPy, המכילה בעצמה את כלל מאגר המידע המנוקה באופן של מטריצה - המאפיינים והסימון יחדיו.
- features - מטריצה מסוג ndarray של NumPy, המכילה בעצמה אך ורק את עמודות מאפייני המודל - כולל המאפיין X0.
- y - וקטור מסוג ndarray של NumPy, המכיל בעצמו אך ורק את עמודת הסימון המודל - מחיר הנדל"ן בדולר.
- data - אובייקט DataFrame של pandas, המכיל את כלל מאגר המידע הנקי של המודל.

לאחר מכן, היא מגדירה בעצמה את התכונה הבאה:

- theta - וקטור מסוג ndarray של NumPy, המכיל בעצמו את הטטה עבור כל מאפיין של המודל.

לבסוף, הפעולה מריצה את פעולת הדיבוג, עליה אסביר כעת.

2.5.3.2 פעולת דיבוג debug_print

```
def debug_print(self):
        # This is a debug function of the Model class
        print("all: \n", self.all_data)
        print("features: \n", self.features)
        print("head: \n", self.features[:50, :])
        print("y: \n", self.y)
```

זוהי פעולת הדיבוג של מחלקת המודל. הפעולה מדפיסה מידע על כלל תכונות מחלקת המודל הקשורות למאגר המידע - all_data, features, ו-y. בפעולה נשתמש אך ורק לצרכי דיבוג ובדיקה בסיסית שהמודל הגדיר את כלל התכונות כפי שצריך.

2.5.3.3 פעולת ההיפותזה

```
def hypothesis(self, x, theta):
        '''
        This function is an implementation of regression hypothesis
        It returns the hypothesis value for specific training example.
        or the hypothesis vector for number of training examples
        '''
        h = x.dot(theta)
        return h
```

זוהי פעולת חישוב ההיפותזה של מחלקת המודל. הפעולה משתמשת בחישובים של מטריצות ווקטורים ובכך מחזירה את ערך ההיפותזה בעבור דוגמת אימון ספציפית, או וקטור של מספר היפותזות בעבור מספר דוגמאות אימון.

הפרמטרים אותם מקבלת:

- x - וקטור או מטריצה מסוג ndarray של NumPy, המכילים את ערכי המאפיינים של מספר דוגמאות אימון או אחת ספציפית.
- theta - וקטור מסוג ndarray של NumPy, המכיל בעצמו את הטטה עבור כל מאפיין של המודל.

2.5.3.4 פעולת השגיאה לדוגמת אימון אחת

```
def error(self, x, y, theta):
        # Returns the squared error for specific training example.
        hypothesis = self.hypothesis(x, theta)
        err = pow(hypothesis - y , 2)
        return err
```

זוהי פעולת חישוב השגיאה של מחלקת המודל. הפעולה, מקבלת שלושה פרמטרים:

- x - וקטור מסוג ndarray של NumPy, המכיל את ערכי המאפיינים של דוגמת אימון ספציפית.
- y - הסימון של דוגמת אימון ספציפית.
- theta - וקטור מסוג ndarray של NumPy, המכיל בעצמו את הטטה עבור כל מאפיין של המודל.

הפעולה מחשבת ומחזירה את השגיאה הריבועית בעבור דוגמת אימון ספציפית.

2.5.3.5 פעולת פונקציית העלות

```
def cost_function(self, x, theta):
        # Implementation of Cost Function
        m = np.size(self.features, 0)
        h = x.dot(theta)
        cost = np.sum(np.power(h-self.y, 2))/(2*m)
        return cost
```

זוהי פעולת פונקציית העלות של מחלקת המודל. הפעולה מקבלת שני פרמטרים:

- x - מטריצה מסוג ndarray של NumPy, המכילה בעצמה את ערכי מאפייני המודל בעבור כל דוגמאות האימון עליהם נרצה לחשב את העלות.
- theta - וקטור מסוג ndarray של NumPy, המכיל בעצמו את הטטה עבור כל מאפיין של המודל.

הפעולה מחשבת את ערך פונקציית העלות על פי הנוסחה, ומחזירה. על מנת לייעל את הביצועים, השתמשתי בשיטה ללא לולאות אלא רק חישובים בעזרת מטריצות כפי שעשיתי באב-טיפוס ב-MATLAB.

2.5.3.6 פעולת בדיקת הכפילויות - check_duplicates

```
def check_duplicates(self, data):
        # method for checking for duplicates and removing them
        dup_num = sum(data.duplicated(data.columns))
        print("Duplicates before:", dup_num)

        # delete duplicates if there are duplicates
        if dup_num != 0:
            data = data.drop_duplicates(data.columns, keep='last')
            print("Duplicates after:",
sum(data.duplicated(data.columns)))

        return data
```

זוהי פעולת בדיקת הכפילויות של מחלקת המודל. הפעולה, מקבלת את הפרמטר הבא:

- data - מאגר הנתונים עליו יתאמן המודל, שהוא מסוג DataFrame של pandas.

הפעולה בודקת האם קיימים מספר דוגמאות אימון שהן אותו דבר - אם כן, מוחקת את הכפילויות.

2.5.3.7 פעולת ה-Normal Equation

```
def normal_equation(self):
        '''
        This function is a Normal Equation Algorithm
        It searches the optimal theta vector
        '''
        X_transpose = self.features.T
        _x = X_transpose.dot(self.features)
        self.theta = np.linalg.pinv(_x).dot(X_transpose).dot(self.y)
        return self.theta
```

זוהי פעולת שיטת השגת הטטה האופטימלי - Normal Equation של מחלקת המודל. הפעולה משתמשת ישירות בתכונות המודל.

הפעולה מחשבת את הטטה האופטימלי על פי הנוסחה, ומחזירה.

2.5.3.8 פעולת מורד הגרדיאנט

זוהי פעולת שיטת השגת הטטה האופטימלי - Gradient Descent של מחלקת המודל. הפעולה, מקבלת ארבעה פרמטרים:

- learning rate - קצב הלמידה, פרמטר מסוג float.
- iter_num - מספר האיטרציות ברצוננו לבצע. אם לא מכניסים ערך, ערכו הברירת מחדל הוא NULL. אם הוא NULL, הוא יבצע את הלולאה עד שהפרש העלויות בין שתי איטרציות יהיה קטן מאוד.
- theta - וקטור מסוג ndarray של NumPy, המכיל בעצמו את הטטה עבור כל מאפיין של המודל.
- scale_norm - פרמטר בוליאני המסמל אם ברצוננו לבצע Feature Scaling ו-Mean Normalization.

הפעולה מחשבת את הטטה האופטימלי על פי הנוסחה, ומחזירה כולל הזמן שלקח לה. כמו כן, היא שומרת ב-DataFrame את העלות בכל איטרציה, ולבסוף מציגה גרף של שינוי ערך העלות עם כל איטרציה.

```
def gradient_descent(self, learning_rate, iter_num = NULL,  theta =
NULL, scale_norm = False):
        '''
        This function is a Gradient Descent Algorithm
        It searches the optimal theta vector
        '''
        # initializing theta
        if theta == NULL:
            theta = np.ones(np.size(self.features, 1))
        # initializing cost history DataFrame
        cost_data = pd.DataFrame({'Cost' : [], 'Iteretion' : []})

        # saving starting time
        start_time = time.time()

        # checking if Scaling and Normalization is needed
        if scale_norm:
            data = self.scale_norm(self.features)
        else:
```

```
            data = self.features

        # initializing starting cost and last cost and m
        last_cost = self.cost_function(data, theta) + 1
        cost = self.cost_function(data, theta)
        m = np.size(data, 0)

        # looping
        if iter_num != NULL:
            for iter in range(1,iter_num + 1):
                print("Iteretion: ", iter, " Cost: ",
self.cost_function(data, theta))

                # hypothesis calculation
                h = self.hypothesis(data, theta)
                theta = theta -( learning_rate/m * (h -
self.y).T.dot(data))

                # entering current cost to cost history
                cost = self.cost_function(data, theta)
                a_row = {'Cost' : cost, 'Iteration' : iter}
                cost_data = cost_data.append(a_row, ignore_index =
True)

        else:
            # count - iteration variable
            count = 1

            while last_cost - cost > 0.00001:
                print("Iteretion: ", count, " Cost: ",
self.cost_function(data, theta))

                # updating last cost
                last_cost = cost

                # hypothesis calculation
                h = self.hypothesis(data, theta)
```

```
                theta = theta -( learning_rate/m * (h -
self.y).T.dot(data))

                # entering current cost to cost history
                cost = self.cost_function(data, theta)
                a_row = {'Cost' : cost, 'Iteration' : count}
                cost_data = cost_data.append(a_row, ignore_index =
True)
                count += 1

        self.theta = theta

        Graph.regplot(cost_data,'Cost', 'Iteration')
        cost_data.to_csv(self.file_dir[0:-5] +
"_gradient-descent-data.csv")

        # saving ending time
        end_time = time.time()

        result = 'Gradient Descent\nTime spended : ' + str(int(end_time
- start_time)) + " s"
        print(result)
        return (theta, result)
```

2.5.3.9 פעולת Feature Scaling ו-Mean Normalization ששמה scale_norm

זוהי פעולת ה-Feature Scaling ו-Mean Normalization. הפעולה, מקבלת את הפרמטר הבא:

- data - מטריצה מסוג ndarray של NumPy, המכילה בעצמה אך ורק את עמודות מאפייני המודל - כולל המאפיין X0.

הפעולה לוקחת את המטריצה, ועושה אופטימיזציה עבור כל מאפיין - היא מגבילה כל מאפיין לטווח קטן, שמאפשר לבצע את שיטת מורד הגרדיאנט הרבה יותר מהר. לצורך ייעול הפעולה השתמשתי בחישובי מטריצות ווקטורים. לבסוף, הפעולה מחזירה את המטריצה שעברה אופטימיזציה.

הפונקציה עוברת על כלל עמודות המאפיינים, בעבור כל עמודה היא מוצאת את הערך המקסימלי, המינימלי והממוצע. כך, היא מוצאת את הטווח scale ובעבור כל איבר במטריצה היא מבצעת לו

Feature Scaling ו-Mean Normalization, זאת על ידי הורדת הממוצע ממנו, וחילוק שלו בטווח scale.

כך כל הערכים הם בין 1 ל-1-.

```
def scale_norm(self, data):
        '''
        This function is an implemintation of Feature Scaling
        and Mean Normalization.
        It return the data set after those changes
        '''
        new_data = data

        # going through all the columns
        for col in range(np.size(data, 1)):
            max = np.amax(data[:, col])
            min = np.amin(data[:, col])
            av = np.average(data[:, col])

            # debug
            print('''
            For column #{num}:
            \tmax value : {max}
            \tmin value : {min}
            \taverage value : {av}
            '''.format(num = col, max = max, min = min, av = av))

            scale = max - min
            new_data[:][col] -= av
            new_data[:][col] /= scale

        return new_data
```

2.5.3.10 פעולת הכנת מאגר המידע והמאפיינים feature_prepare

זוהי פעולת הכנת מאגר המידע והמאפיינים. הפעולה, מקבלת את הפרמטר הבא:

- data - מאגר הנתונים עליו יתאמן המודל, שהוא מסוג DataFrame של pandas.

הפעולה, מצפה שמאגר המידע אותו תקבל, תהיה בפורמט הנכון של משרד האוצר של ניו-יורק. הפעולה, מוחקת את העמודות הלא נחוצות, ובודקת האם קיים כבר קובץ בו שמור מאגר המידע באופן מנוקה. אם כן, היא לוקחת אותו, אם לא - היא מבצעת את הניקוי על ידי הפעולה clean, עליה אסביר בהמשך. לבסוף, הפעולה מכינה את כלל תכונות מחלקת המודל, ביניהן: data, all_data, features ו-y.

```
def feature_prepare(self, data):
        '''
        Feature Engineering Prepare function
        '''

        saved_data = data
        #removing unnecessary columns
        saved_data.drop(columns = ['BUILDING CLASS CATEGORY', 'TAX
CLASS AT PRESENT', 'EASE-MENT', 'BUILDING CLASS AT PRESENT', 'ADDRESS',
'APARTMENT NUMBER'], inplace=True)
        #cleaning
        if not os.path.isfile(self.file_dir[0:-5] + '_new.csv'):
            saved_data = self.clean(saved_data)
        else:
            saved_data = pd.read_csv (self.file_dir[0:-5] + '_new.csv')
            saved_data = saved_data.iloc[: , 1:]

        self.data = saved_data
        saved_data = saved_data.astype('float64')
        self.all_data = saved_data.to_numpy()
        self.features = np.c_[np.ones(np.size(self.all_data, 0)),
self.all_data[:, 0:2], self.all_data[:, 3:-1]] #adding X0
        self.y = self.all_data[:, 2]
```

2.5.3.11 פעולת הניקוי נתונים clean

זוהי פעולת ניקוי המידע - Data Cleaning. הפעולה, מקבלת את הפרמטר הבא:

- data - מאגר הנתונים עליו יתאמן המודל, שהוא מסוג DataFrame של pandas.

הפעולה, מצפה שמאגר המידע אותו תקבל, תהיה בפורמט הנכון של משרד האוצר של ניו-יורק. הפעולה, מתחילה מהגדרת המידע שבתוך המאגר לטיפוסים המתאימים. לאחר מכן, היא ממשיכה בניקוי כלל מאגר המידע, באופן אותו תיארתי בפרקים הקודמים. הפעולה, מבצעת את השינויים אותם תיארתי, ולפני ואחרי כל שינוי מציגה גרף המתאר את המצב של מאגר המידע, תוך שמירת נתונים חשובים בקבצים נפרדים.

הפעולה, כחלק מתהליך הניקוי מבצעת גם את תהליך ה-Feature Engineering - למשל הפרדת מאפיינים קטגוריאלים למספר מאפיינים נפרדים. לבסוף, הפעולה שומרת את המידע המנוקה, ומחזירה אותו.

```
def clean(self, data):
        '''
        Data Cleaning function
        '''

        # set the right types for the features
        data['SALE PRICE'] = pd.to_numeric(data['SALE PRICE'],
errors='coerce')
        data['LAND SQUARE FEET'] = pd.to_numeric(data['LAND SQUARE
FEET'], errors='coerce')
        data['GROSS SQUARE FEET']= pd.to_numeric(data['GROSS SQUARE
FEET'], errors='coerce')
        data['SALE DATE'] = pd.to_datetime(data['SALE DATE'],
errors='coerce')
        data['TAX CLASS AT TIME OF SALE'] = data['TAX CLASS AT TIME OF
SALE'].astype('category')

        # seeing the data before cleaning
        data_info = data.describe()
        data_info.to_csv(self.file_dir[0:-5] + '_data-info.csv')
        print(data_info)
```

```
        #setting the columns that need to be checked
        check_col = ['TOTAL UNITS', 'LAND SQUARE FEET','GROSS SQUARE
FEET', 'SALE PRICE']

        for col in data.columns:
            data = data[data[col].notnull()]

        data = data[data['SALE PRICE'] > 1000]

        for col in check_col:
            data = data[data[col] > 0]

        #removing numbering columns
        del data['Unnamed: 0']

        #checking for duplicates and removing them
        data = self.check_duplicates(data)

        #For BoxPlot
        Graph.boxplot(data, 'SALE PRICE')

        #For Cumulative Distribution
        Graph.cumulative_distribution(data, 'SALE PRICE')

        data = data[(data['SALE PRICE'] > 400000) & (data['SALE PRICE']
< 6000000)]

        #For BoxPlot
        Graph.boxplot(data, 'SALE PRICE', '(After)')

        #For Cumulative Distribution
        Graph.cumulative_distribution(data, 'SALE PRICE', '(After)')

        #For BoxPlot
        Graph.boxplot(data, 'GROSS SQUARE FEET')

        #For BoxPlot
        Graph.boxplot(data, 'LAND SQUARE FEET')
```

```
        data = data[(data['GROSS SQUARE FEET'] < 20000) & (data['LAND
SQUARE FEET'] < 20000)]
        print(len(data))

        #For BoxPlot
        Graph.boxplot(data, 'GROSS SQUARE FEET', '(After)')

        #For BoxPlot
        Graph.boxplot(data, 'LAND SQUARE FEET', '(After)')

        # limiting the TOTAL UNITS and check the sum of all the units
accordingly
        data = data[(data['TOTAL UNITS'] > 0) & (data['TOTAL UNITS'] <
50)]
        data = data[(data['TOTAL UNITS'] == data['COMMERCIAL UNITS'] +
data['RESIDENTIAL UNITS'])]
        print(len(data))

        # deleting zip codes that cannot be
        data = data[(data['ZIP CODE'] > 10000) & (data['ZIP CODE'] <=
14975)]
        print(len(data))

        # making the zipcode start from 0
        data['ZIP CODE'] = data['ZIP CODE'] - 10001

        data = data[data['YEAR BUILT'] > 0]
        print(len(data))

        # creating BUILDING AGE feature instead of YEAR BUILT
        data['BUILDING AGE'] = 2017 - data['YEAR BUILT']

        del data['YEAR BUILT']

        #For Regplot
        Graph.regplot(data, 'SALE PRICE', 'BUILDING AGE')
```

```
        #For BUILDING CLASS AT TIME OF SALE Boxplot
        plt.figure(figsize=(20,6))
        order = sorted(data['BUILDING CLASS AT TIME OF SALE'].unique())
        sns.boxplot(x='BUILDING CLASS AT TIME OF SALE', y='SALE PRICE',
data=data, order=order)
        plt.xticks(rotation=90)
        plt.title('Sale Price Distribution by BUILDING CLASS')
        plt.show()

        #For TAX CLASS AT TIME OF SALE Boxplot
        plt.figure(figsize=(20,6))
        order = sorted(data['TAX CLASS AT TIME OF SALE'].unique())
        sns.boxplot(x='TAX CLASS AT TIME OF SALE', y='SALE PRICE',
data=data, order=order)
        plt.xticks(rotation=90)
        plt.title('Sale Price Distribution by TAX CLASS')
        plt.show()

        #For NEIGHBORHOOD class Boxplot
        plt.figure(figsize=(30,6))
        order = sorted(data['NEIGHBORHOOD'].unique())
        sns.boxplot(x='NEIGHBORHOOD', y='SALE PRICE', data=data,
order=order)
        plt.xticks(rotation=90)
        plt.title('Sale Price Distribution by NEIGHBORHOOD')
        plt.show()

        # updating the values in the BOROUGH feature to the actual
boroughs
        data['BOROUGH'][data['BOROUGH'] == 1] = 'Manhattan'
        data['BOROUGH'][data['BOROUGH'] == 2] = 'Bronx'
        data['BOROUGH'][data['BOROUGH'] == 3] = 'Brooklyn'
        data['BOROUGH'][data['BOROUGH'] == 4] = 'Queens'
        data['BOROUGH'][data['BOROUGH'] == 5] = 'Staten Island'

        # Boxplot of the different boroughs
        plt.figure(figsize=(10,6))
        order = sorted(data['BOROUGH'].unique())
```

```
        sns.boxplot(x='BOROUGH', y='SALE PRICE', data=data,
order=order)
        plt.xticks(rotation=90)
        plt.title('Sale Price Distribution by BOROUGH')
        plt.show()

        # limiting the price-square feet ratio
        data = data[(data['SALE PRICE'] < 1500000) & (data['GROSS
SQUARE FEET'] < 5000) & (data['LAND SQUARE FEET'] <
5000)].append(data[(data['SALE PRICE'] >= 1500000)])

        # initializing bounderies for every borough
        num1,num2,num3,num4,num5 = (1500000, 3000000, 4000000, 3000000,
2000000)

        # amount below\above the bounderies for every borough
        len1 = len(data[(data['BOROUGH'] == 'Manhattan') & (data['SALE
PRICE'] <= num1)])
        len2 = len(data[(data['BOROUGH'] == 'Bronx') & (data['SALE
PRICE'] >= num2)])
        len3 = len(data[(data['BOROUGH'] == 'Brooklyn') & (data['SALE
PRICE'] >= num3)])
        len4 = len(data[(data['BOROUGH'] == 'Queens') & (data['SALE
PRICE'] >= num4)])
        len5 = len(data[(data['BOROUGH'] == 'Staten Island') &
(data['SALE PRICE'] >= num5)])

        # debug
        print('''
        Manhattan below {num1}: {ans1}
        Bronx above {num2}: {ans2}
        Brooklyn above {num3}: {ans3}
        Queens above {num4}: {ans4}
        Staten Island above {num5}: {ans5}''''.format(num1=num1,
num2=num2, num3=num3,num4=num4,num5=num5,
                      ans1=len1,
ans2=len2,ans3=len3,ans4=len4,ans5=len5))
```

```
        # data for every Borough
        manhattan_data = data[(data['BOROUGH'] == 'Manhattan') &
(data['SALE PRICE'] >= num1)]
        bronx_data = data[(data['BOROUGH'] == 'Bronx') & (data['SALE
PRICE'] <= num2)]
        brooklyn_data = data[(data['BOROUGH'] == 'Brooklyn') &
(data['SALE PRICE'] <= num3)]
        queens_data = data[(data['BOROUGH'] == 'Queens') & (data['SALE
PRICE'] <= num4)]
        staten_island_data = data[(data['BOROUGH'] == 'Staten Island')
& (data['SALE PRICE'] <= num5)]

        dataframes = [manhattan_data, bronx_data, brooklyn_data,
queens_data, staten_island_data]

        # connecting the data for every borough
        data = pd.concat(dataframes, axis=0)

        # The features that will stay
        columns = ['BOROUGH', 'BUILDING CLASS AT TIME OF SALE',
'COMMERCIAL UNITS','BUILDING AGE', 'SALE PRICE', 'GROSS SQUARE FEET',
'LAND SQUARE FEET', 'RESIDENTIAL UNITS', 'NEIGHBORHOOD', 'TAX CLASS AT
TIME OF SALE', 'ZIP CODE'] #better without month or date - write about
this
        data_model = data.loc[:,columns]


        # The Categorical features that need to be seperate
        one_hot_features = ['BOROUGH', 'TAX CLASS AT TIME OF SALE',
'NEIGHBORHOOD', 'BUILDING CLASS AT TIME OF SALE']

        # For each categorical column, find the unique number of
categories. This tells us how many columns we are adding to the
dataset.
        longest_str = max(one_hot_features, key=len)
        total_num_unique_categorical = 0
        for feature in one_hot_features:
            num_unique = len(data[feature].unique())
```

```
            print('{col:<{fill_col}} : {num:d} unique categorical
values.'.format(col=feature,

fill_col=len(longest_str),

num=num_unique))
            total_num_unique_categorical += num_unique
        print('{total:d} columns will be added during one-hot
encoding.'.format(total=total_num_unique_categorical))

        one_hot_encoded = pd.get_dummies(data_model[one_hot_features])
        one_hot_encoded.info(verbose=True, memory_usage=True,
null_counts=True)

        # Delete the old columns
        data_model = data_model.drop(one_hot_features, axis=1)

        # Add the new one-hot encoded variables
        data_model = pd.concat([data_model, one_hot_encoded], axis=1)
        data_model.head()
        data_model['SALE PRICE'] = data['SALE PRICE']

        # Saving the changes
        data = data_model

        #seeing the data
        data_info = data.describe()
        data_info.to_csv(self.file_dir[0:-5] + '_data-info-after.csv')
        print(data_info)

        #saving new data
        data.to_csv(self.file_dir[0:-5] + '_new.csv')
        return data
```

2.5.3.12 פעולת השגיאה הממוצעת - average_error

זוהי פעולת חישוב השגיאה הממוצעת של המודל במחלקת המודל. הפעולה, מקבלת את הפרמטר הבא:

- theta - וקטור מסוג ndarray של NumPy, המכיל בעצמו את הטטה עבור כל מאפיין של המודל.

הפעולה מחשבת את אחוז השגיאה הממוצעת של המודל, ואת מחיר השגיאה הממוצע.

הפעולה, יוצרת DataFrame חדש, שבו ישמר אחוז השגיאה והמחיר של כל נכס. לאחר מכן, היא עוברת על כל דוגמאות האימון אחת אחת, ומחשבת את אחוז השגיאה הממוצעת של כלל דוגמאות האימון, ואת הפרש המחיר הממוצע של כלל דוגמאות האימון.

לבסוף, היא מדפיסה נתונים אלו, ומציגה גרף של אחוז השגיאה ביחס למחיר.

```
def average_error(self, theta):
        '''
        This function returns the average error of the model.
        In percentage and in price
        '''
        error_data = pd.DataFrame({'ERROR %' : [], 'PRICE' : []})

        #initializing average error
        av_error_percent = 0
        av_price_error = 0

        # amount of training examples
        m = np.size(self.features, 0)

        #going through all the training sets
        for row in range(m):
            # x vector
            x = self.features[row]
            error_size = self.error(x, self.y[row], theta)
            error_sqrt = math.sqrt(error_size)
            av_price_error += error_sqrt

            av_error_percent += error_sqrt / self.y[row]
```

```
            a_row = {'ERROR %' : error_sqrt / self.y[row] * 100,
'PRICE' : self.y[row]}
            error_data = error_data.append(a_row, ignore_index = True)

        av_error_percent /= m
        av_price_error /= m

        print(error_data)
        result = 'average error percentage :
{num}%\n'.format(num=av_error_percent * 100)
        result += 'average error in price :
{num}$'.format(num=av_price_error)
        print(result)

        Graph.regplot(error_data,'ERROR %', 'PRICE')
        error_data.to_csv(self.file_dir[0:-5] + '_error-data.csv')
        return (result, av_error_percent, av_price_error)
```

2.5.3.13 פעולת בדיקת השגיאות של כלל המידע - run_check

זוהי פעולת בדיקת המודל של מחלקת המודל. הפעולה, מקבלת את הפרמטר הבא:

- theta - וקטור מסוג ndarray של NumPy, המכיל בעצמו את הטטה עבור כל מאפיין של המודל.

הפעולה בודקת את המודל בכך שמחפשת מהו אחוז הנכסים במאגר המידע בהם אחוז השגיאה קטן מ-50%, 30%, 20%, 10% ו-5%.

הפעולה עוברת על כל דוגמאות האימון. בכל מעבר, היא בוחנת האם הדוגמת אימון נמצאת מתחת לאחוז מסויים, אם כן, היא מוסיפה 1 למונה של אותו האחוז (שמור במילון dict_below).

```
def run_check(self, theta):
        # dictionary to save the amount of encounters per percentage
        dict_below = {}

        # list of the percentage numbers
        below_percent_lst = [50, 40, 30, 20, 10, 5]

        # initializing dictionary
        for e in below_percent_lst:
            dict_below[e] = 0

        # amount of training examples
        m = np.size(self.features, 0)

        #going through all the training sets
        for row in range(m):
            # x vector
            x = self.features[row]
            error_size = self.error(x, self.y[row], theta)

            h = self.hypothesis(x, theta)
            #going through all the percentages
            for percent in below_percent_lst:
                if error_size < pow(percent/100 * h, 2):
                    dict_below[percent] += 1
```

```
        # print all the encounters per percentage
        result = ''

        for key in dict_below:
            result += 'difference below {col}% : {num} from {amount}
training example, ({global_percent}%).\n'.format(col=key,
num=dict_below[key], amount = m, global_percent = dict_below[key]/m *
100)

        print(result)
        return result
```

Model Selection Algorithm 2.5.4

כפי שהסברתי, אלגוריתם לבחירת מודל הוא אלגוריתם שמטרתו למצוא את המודל הטוב ביותר בעבור מטרה מסוימת. הבעיה העיקרית באלגוריתמים מסוג זה, היא זמני העבודה שלהם הלוקחים זמן רב.

קוד משופר זה, תוך שיפור פעולות המחלקה Model באופן מתמטי - חישוב על ידי מטריצות ווקטורים, האיץ את אופן פעולת האלגוריתם, ממספר שעות למספר שניות!

במהלך העבודה הבנתי כי המודל אותו ברצוני לבנות הוא לינארי - דרך המונעת בדרך הטובה ביותר התאמת יתר חמורה לנתונים עליהם מתאמן המודל עד לפגיעה בחיזוי עבור נתונים חדשים. לכן, המודל, על ידי שילוב בדיקת הצלבה, מוצא את קומבינציית המאפיינים הטובה ביותר.

האלגוריתם, כמו כן, מודד את זמני ביצועיו ובסוף פעילותו מדפיסם על המסך.

כעת נעבור על הקוד:

האלגוריתם מתחיל מאתחול המשתנה start_time השומר את זמן תחילת ריצת האלגוריתם.

```
def model_selection(model):
    '''
    This function is a Model Selection Algorithm
    It searches the best feature combination
    '''
    start_time = time.time()
```

לאחר מכן, האלגוריתם, מערבב את המידע, באופן שימנע מהאלגוריתם ללמוד מאוסף נתונים ספציפיים הלא מציג את כלל התמונה, ומחלק את המידע לשלוש קבוצות: סט אימונים (60%), סט הצלבה (20%), וסט בדיקה (20%).

```
every = np.c_[model.features, model.y]
every_save = every
theta_save = model.theta
m_feat = model.features
```

```
m_y = model.y

# shuffle

np.random.shuffle(every)


m_feat = every[:, 0:np.size(m_feat, 1)]

m_y = every[:, -1]


# amount of training examples

m = np.size(m_feat, 0)

training_size = int(0.6 * m)

csv_size = int(0.2 * m)

test_size = m - training_size - csv_size


# training set

training_set = m_feat[0:training_size]

training_y = m_y[0:training_size]


# cross validation set

csv_set = m_feat[training_size:training_size+csv_size]

csv_y = m_y[training_size:training_size+csv_size]


# test set

test_set = m_feat[training_size + csv_size:m]

test_y = m_y[training_size + csv_size:m]
```

לאחר מכן, האלגוריתם, יוצר לעצמו רשימה המהווה אוסף קומבינציות שונות בין מאפיינים שונים, כל שכל מאפיין מצויין בתור מספר האינדקס שלו.

```
# amount of features
x_num = np.size(model.features, 1)

min_cost = LARGE_NUM
min_theta = np.ones(x_num)
min_comb = []

possible_col = range(7)
always_col = list(range(7, x_num))

# creating a list of all combinations
all_combinations = []
for r in range(len(possible_col) + 1):
      combinations_object = itertools.combinations(possible_col, r)
      combinations_list = list(combinations_object)
      all_combinations += combinations_list
```

בשלב זה, האלגוריתם, בעבור כל קומבינציה של מאפיינים, מוצא את הטטה האופטימלי באמצעות אימון על סט האימון, ומחשב את ערך העלות בעבור אותו אוסף מאפיינים על גבי .

```
# checking every features combination
for comb in all_combinations:
      model.features = training_set[:, list(comb) + always_col]
      model.y = training_y
      np.set_printoptions(suppress=True)

      # finding theta
      theta = model.normal_equation()
      model.y = csv_y
      cost = model.cost_function(csv_set[:, list(comb) + always_col],
theta)
```

```
        print(comb, "\ncost: ", cost, "\n\nmin comb:", min_comb, "\nmin cost: ", min_cost)


        # updating if better combination
        if cost < min_cost:
            min_cost = cost
            min_theta = theta
            min_comb = comb

    end_time = time.time()

    model.y = test_y
```

לבסוף, האלגוריתם יוצר את המחרוזת אותה יחזיר עם תשבותו, ומחזיר את ערכי כל המשתנים אותה שינה לערכים המקוריים.

```
    result = '''Min comb : {comb}
                Min cost (Cross-Validation Set): {cost}
                Real cost (Test Set) : {r_cost}
                Time spended : {time}
               '''.format(comb = min_comb, cost = min_cost, r_cost = model.cost_function(test_set[:, list(min_comb) + always_col], min_theta), time = end_time - start_time)


    model.features = every_save[:, 0:np.size(m_feat, 1)]
    model.y = every_save[:, -1]
    model.theta = theta_save
    print(result)
    return result
```

GUI 2.5.5

נתחיל מלעבור על המבנה הכללי של המחלקה-GUI (Graphical User Interface), או בעברית - ממשק משתמש גרפי.

ה-GUI מורכב משלושה חלונות עיקריים: מסך הפתיחה, המסך הראשי - מסך האימון, ומסך החיזוי.

2.5.5.1 הפעולה הבונה __init__

הפעולה הבונה מגדירה שלושה Layouts שונים עבור כל אחד משלושת החלונות השונים. כל Layout ישמר כתכונה של המחלקה. כמו כן, לפעולה תהיה תכונה ששמה model - אובייקט מסוג Model בו בעצם ישתמש המשתמש.

בעבור כל חלון, הפעולה מגדירה מספר listים - אחד בעבור כל עמודה בחלון. לאחר מכן, היא מחברת אותם ל-list אחד של אובייקטים שונים של המחלקה PySimpleGUI. למשל, האובייקט Column - אובייקט לצורך עמודה.

בכל list של עמודה בתוך ה-Layout, נשים את ה-widgetים השונים בהם נשתמש: תיבות טקסט, כפתורים, וכדומה. כל אלו, אובייקטים מוגדרים של הספרייה PySimpleGUI.

בכל Layout היא תשתמש בעבור חלון נפרד. על ה-Layout עצמו של כל חלון נעבור בתתי הפרקים הבאים על החלונות עצמם. כעת אכניס את הקוד של קוד הגדרת Layout.

הגדרת Layout מסך הפתיחה:

```
# First Column - choosing folder and file viewer
file_list_column = [
      [sg.Text("Data Folder"),
      sg.In(size=(25, 1), enable_events=True, key="-FOLDER-"),
      sg.FolderBrowse()],
      [sg.Listbox(
      values=[], enable_events=True, size=(40, 20), key="-FILE LIST-")
      ]]
```

```
# Second Column - Text about the file and confirmation
data_viewer_column = [
      [sg.Text("Choose a data from list on left:")],
      [sg.Text(size=(40, 1), key="-TOUT-")],
      [sg.Button("Confirm", visible=False, key="-CONFIRM-")]]

# ----- Full layout -----
self.layout = [
      [
            sg.Column(file_list_column),
            sg.VSeperator(),
            sg.Column(data_viewer_column),
      ]]
```

הגדרת Layout המסך הראשי - מסך האימון:

```
# First Column - training methods
control_column = [
      [sg.Button("Normal Equation", key = "-NORMAL-")],
      [sg.Button("Gradient Descent", key = "-GD-")],
      [sg.Text("Enter Learning rate:")],
      [sg.In(size=(10, 1), enable_events=True, key="-LEARNING RATE-")],
      [sg.Text("Enter Iteration number:")],
      [sg.In(size=(10, 1), enable_events=True, key="-ITERATION-")],
      [sg.Button("Show theta", key = "-SHOW THETA-")],
      [sg.Button("Reset theta", key = "-RESET THETA-")],
      [sg.Text("")],
      [sg.Button("Clear", key = "-CLEAR-")],
      [sg.Button("Predict", key = "-PREDICT-", visible=False)]]

# Seconds Column - Model Checking methods and Result
check_column = [
      [sg.Button("Run Check", key = "-RUNCHECK-")],
      [sg.Button("Average Error", key = "-AVERROR-")],
      [sg.Button("Model Selection Algorithm", key = "-MSA-")],
      [sg.Text("Result:")],
      [sg.Text("", key = "-RESULT-")]]
```

```
# ----- Main Menu Layout -----
self.main_layout = [
      [
            sg.Column(control_column),
            sg.VSeperator(),
            sg.Column(check_column)
      ]]
```

הגדרת Layout מסך החיזוי:

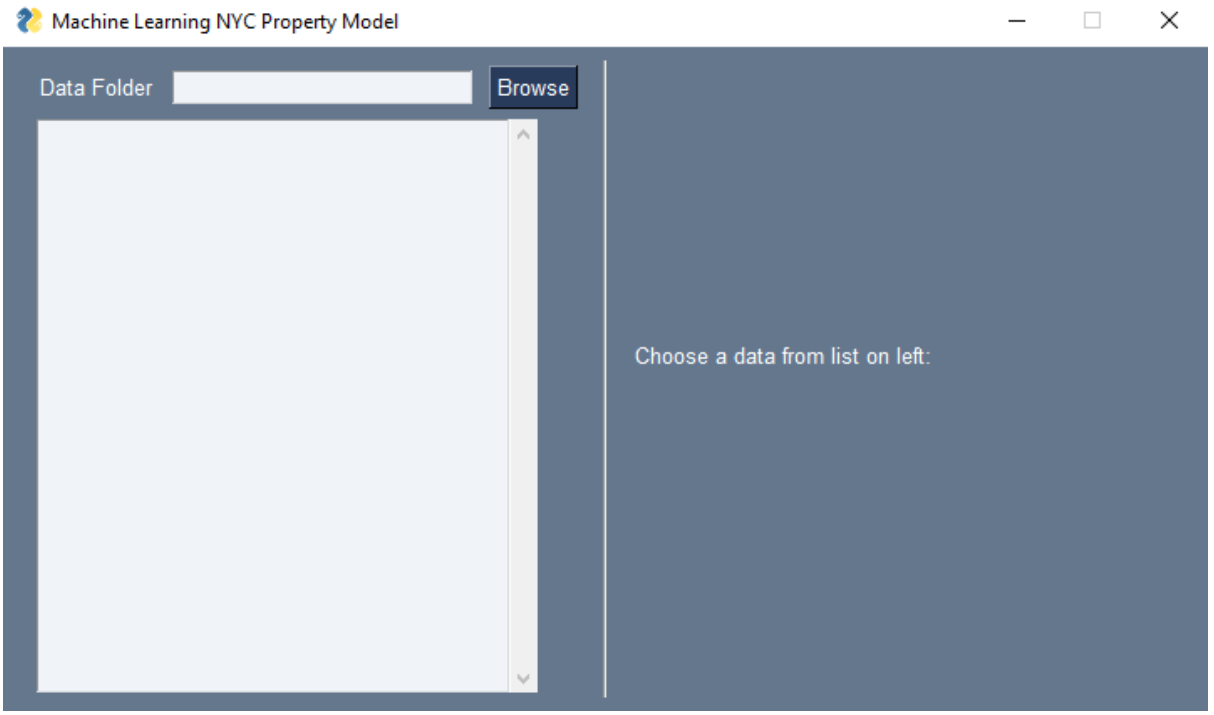
```
# First Column - Titles for inputs
enter_data_text_column = [
      [sg.Text("Enter the following data:")],
      [sg.Text("Enter Borough:")],
      [sg.Text("Enter Building Class:")],
      [sg.Text("Enter Tax Class:")],
      [sg.Text("Enter Neighborhood:")],
      [sg.Text("Enter year build:")],
      [sg.Text("Enter Zip Code:")],
      [sg.Text("Enter Commercial Units:")],
      [sg.Text("Enter Residential Units:")],
      [sg.Text("Enter Gross Square feet:")],
      [sg.Text("Enter Land square feet:")],
      [sg.Button("Predict", key = "-CHECK-")]]

# Second Column - The Input Boxes
enter_data_column = [
      [sg.In(size=(15, 1), enable_events=True, key="-BOROUGH-")],
      [sg.In(size=(15, 1), enable_events=True, key="-BUILDING-")],
      [sg.In(size=(15, 1), enable_events=True, key="-TAX-")],
      [sg.In(size=(15, 1), enable_events=True, key="-NEIGHBORHOOD-")],
      [sg.In(size=(15, 1), enable_events=True, key="-YEAR-")],
      [sg.In(size=(15, 1), enable_events=True, key="-ZIPCODE-")],
      [sg.In(size=(15, 1), enable_events=True, key="-COMMERCIAL-")],
      [sg.In(size=(15, 1), enable_events=True, key="-RESIDENTIAL-")],
      [sg.In(size=(15, 1), enable_events=True, key="-GROSS-")],
      [sg.In(size=(15, 1), enable_events=True, key="-LAND-")]]
```

```
# Third Column - The results
result_column = [
      [sg.Text("Result:")],
      [sg.Text("", key= "-PRESULT-")]
]

# ----- Full layout -----
self.data_enter_layout = [
      [
            sg.Column(enter_data_text_column),
            sg.Column(enter_data_column),
            sg.VSeparator(),
            sg.Column(result_column)]]
```

כעת נעבור על כל מסך באופן מעמיק.

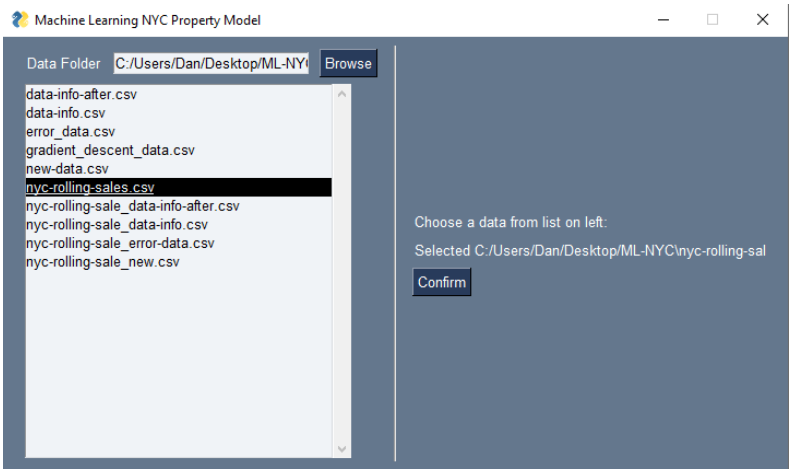
2.5.5.2 מסך הפתיחה

כאשר אנו מריצים את התוכנה, נפתח מסך הפתיחה:



בחלון זה, יכול המשתמש לבחור לעצמו תיקיה בה נמצא מאגר המידע, על ידי לחיצה על הכפתור Browse.

כאשר הוא לוחץ על כפתור זה, משמאל מופיעה רשימת קבצי ה-csv. כאשר הוא בוחר קובץ, אשר חייב להיות בפורמט הקבצים הרשמיים של משרד האוצר של ניו יורק, מופיע לו כפתור האישור באופן הבא:



לאחר שהקוד מנקה את מאגר המידע והכין את המודל, הוא עובר למסך הבא שהוא מסך האימון.

הקוד - הפעולה start_window

תחילה, הקוד יוצר אובייקט מסוג window - אובייקט של הספרייה PySimpleGUI האחראי על פתיחת חלון. אחד הפרמטרים להם הוא זקוק לשם יצירת החלון הוא ה-Layout, אותו הכנו קודם לכן בפעולה הבונה.

```
window = sg.Window("Machine Learning NYC Property Model", self.layout)
```

לאחר מכן, הפעולה מאתחלת שני משתנים: self.model - אובייקט המודל של המחלקה GUI, ו-data_filename - מחרוזת ריקה בה ישמר לאחר מכן ה-directory של קובץ מאגר המידע.

```
data_filename = ""
self.model = NULL
```

לאחר אתחול זה, מתחילה לולאת while אינסופית אשר בה בכל איטרציה, תחילה נשמרים כל הערכים של כל ה-widgetים וכל ה-"אירועים" שקרו בחלון, במשתנים values ו-event בהתאם. לאחר מכן, בכל איטרציה, נבדק אם בעצם קרא אירוע ספציפי - אירוע מסמל אינטרקציה של המשתמש עם widget מסויים במסך - למשל, לחיצה על כפתור.

```
# Run the Event Loop
while True:
      event, values = window.read()
```

הלולאה בודקת את המקרים הבאים:

- אם המסך נסגר - במצב זה יוצא מהלולאה ובכך סוגר את האובייקט window ויוצא מהפעולה.

```
if event == "Exit" or event == sg.WIN_CLOSED:
       break
```

- אם נבחרה תיקיה - הפעולה מנסה להשיג את רשימת קבצי ה-csv הנמצאים בתיקיה, ומציגה אותם ברשימת הקבצים על המסך.

```
# Folder name was filled in, make a list of files in the folder
if event == "-FOLDER-":
      folder = values["-FOLDER-"]
      try:
            # Get list of files in folder
```

```
            file_list = os.listdir(folder)
      except:
            file_list = []
            fnames = [
            f
            for f in file_list
            if os.path.isfile(os.path.join(folder, f))
            and f.lower().endswith(".csv")]

            window["-FILE LIST-"].update(fnames)
```

- אם נבחר קובץ - הפעולה לוקחת את ה-directory המלא של הקובץ יחד עם התיקיות בהן הוא נמצא, ומעדכנת מימין כיתוב בו רשום איזה קובץ נבחר. כמו כן, היא מציגה את הכפתור לאישור הבחירה.

```
elif event == "-FILE LIST-":  # A file was chosen from the listbox
      try:
            filename = os.path.join(
            values["-FOLDER-"], values["-FILE LIST-"][0]
            )
            window["-TOUT-"].update("Selected " + filename)
            data_filename = filename
            window["-CONFIRM-"].update(visible=True)
      except:
            pass
```

- אם נלחץ כפתור האישור - פותחת הפעולה את הקובץ לתוך DataFrame, יוצרת את אובייקט המודל של המחלקה, ויוצאת מהלולאה האינסופית.

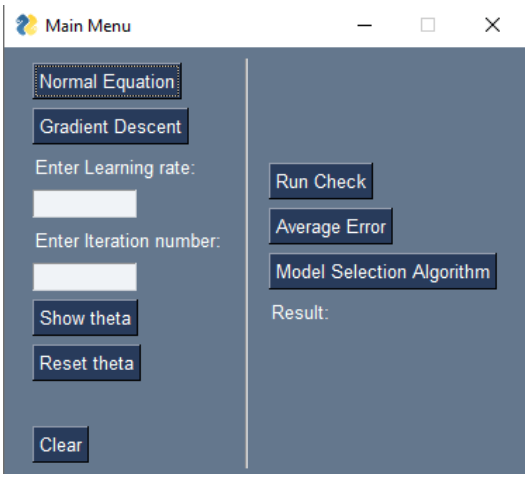
```
elif event == "-CONFIRM-":
      df = pd.read_csv (data_filename)
      self.model = Model(df, data_filename)
      break
```

לאחר יציאה מהלולאה, החלון נסגר.

```
window.close()
```

2.5.5.3 המסך הראשי - מסך האימון

כאשר יצאנו ממסך הפתיחה, נפתח מסך האימון:

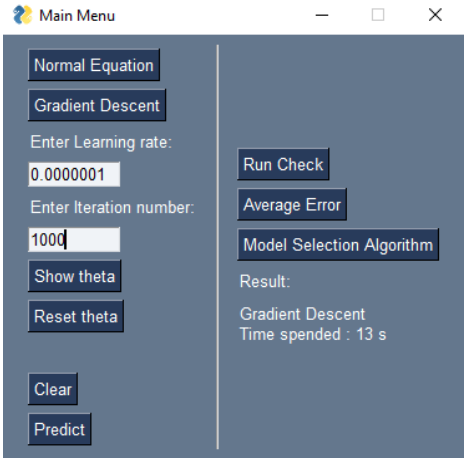


חלון זה מחולק לשני חלקים:

- החלק השמאלי - אפשרויות אימון המודל על ידי שיטות Normal Equation ו-Gradient Descent, ואתחול האימון על ידי אתחול הטטה.
- החלק הימני - אפשריות לבדיקת המודל: בדיקת השגיאה הממוצעת, הרצת בדיקה, ואלגוריתם לבחירת מודל אשר בודק את שילוב המאפיינים האידיאלי.

התוצאה לאחר לחיצה על כל כפתור מופיעה תחת הכיתוב "Result".

בחלון זה, על מנת לאמן את המודל בשיטת Gradient Descent, המשתמש צריך להקליד את קצב הלמידה. כמו כן, הוא יכול להקליד כמות איטרציות ספציפית אותם יבצע האלגוריתם, או להשאיר ריק והאלגוריתם ירוץ עד שהפרש העלויות בין שתי איטרציות יהיה קטן מאוד.



כאשר המודל יאומן בשיטה כלשהי, מופיע כפתור מעבר לעמוד החיזוי - עמוד בו יוכל המשתמש להכניס מידע של נכס, ולחזות את מחירו על פי המודל שאימן.

הקוד - הפעולה main_menu

תחילה, הקוד יוצר אובייקט מסוג window - אובייקט של הספרייה PySimpleGUI האחראי על פתיחת חלון. אחד הפרמטרים להם הוא זקוק לשם יצירת החלון הוא ה-Layout, אותו הכנו קודם לכן בפעולה הבונה.

```
window = sg.Window("Main Menu", self.main_layout)
```

לאחר מכן, הפעולה מאתחלת שני משתנים: iter - מספר האיטרציות אותו יבצע אלגוריתם מורד הגרדיאנט, ו-learning_rate - קצב הלמידה של אלגוריתם מורד הגרדיאנט.

```
iter = NULL
learning_rate = 0.0000001
```

לאחר אתחול זה, מתחילה לולאת while אינסופית אשר בה בכל איטרציה, תחילה נשמרים כל הערכים של כל ה-widget-ים וכל ה-"אירועים" שקרו בחלון, במשתנים values ו-event בהתאם. לאחר מכן, בכל איטרציה, נבדק אם בעצם קרא אירוע ספציפי - אירוע מסמל אינטרקציה של המשתמש עם widget מסויים במסך - למשל, לחיצה על כפתור.

```
while True:
      event, values = window.read()
```

הלולאה בודקת את המקרים הבאים:

- אם המסך נסגר - במצב זה יוצא מהלולאה ובכך סוגר את האובייקט window ויוצא מהפעולה.

```
if event == "Exit" or event == sg.WIN_CLOSED:
      break
```

- אם הכפתור של Normal Equation נלחץ - במצב זה הפעולה מבצעת Normal Equation וגורמת לכפתור המעבר לעמוד החיזוי להופיע.

```
if event == "-NORMAL-":
      self.model.normal_equation()
      window["-PREDICT-"].update(visible = True)
```

- אם הכפתור של Gradient Descent נלחץ - במצב זה הפעולה מבצעת את שיטת מורד הגרדיאנט תוך שימוש בערכי learning_rate ו-iter תוך בדיקת ערכיהם. לבסוף, מעדכנת את זמן ריצתה מתחת לכיתוב "Result" וגורמת לכפתור המעבר לעמוד החיזוי להופיע.

```
elif event == "-GD-":
      print(learning_rate)
      print(iter)
      if type(learning_rate) == float and type(iter) == int or iter ==
NULL:
      window["-RESULT-"].update(self.model.gradient_descent(learning_ra
te,iter_num=iter)[1])
      window["-PREDICT-"].update(visible = True)
      else:
            sg.popup("Iteration number and Learning rate should contain
only numbers and dot!")
```

- אם הכפתור של Model Selection Algorithm נלחץ - במצב זה הפעולה מריצה את האלגוריתם לבחירת המודל ומעדכנת את תוצאותיה מתחת לכיתוב "Result".

```
elif event == "-MSA-":
      window["-RESULT-"].update(model_selection(self.model))
```

- אם עודכן ערך ה-learning rate - הערך מתעדכן במשתנה learning_rate תוך בדיקה של נכונות הערך שהוכנס.

```
elif event == "-LEARNING RATE-":
      learning_rate = values["-LEARNING RATE-"]
      if learning_rate == '':
            learning_rate = 0.0000001
            continue
      try:
            learning_rate = float(learning_rate)
      except:
            pass
```

- אם עודכן ערך ה-iteration number - הערך מתעדכן במשתנה בהתאם תוך בדיקה של נכונות הערך שהוכנס.

```
elif event == "-ITERATION-":
      iter = values["-ITERATION-"]
      if iter == '':
            iter = NULL
            continue
      try:
            iter = int(iter)
            if iter <= 0:
                  iter = NULL
      except:
            pass
```

- אם כפתור אתחול הטטה נלחץ - הטטה של אובייקט המודל מאותחל, ומוחבא כפתור המעבר למסך החיזוי.

```
elif event == "-RESET THETA-":
      self.model.theta = np.ones(np.size(self.model.features, 1))
      window["-PREDICT-"].update(visible = False)
```

- אם כפתור הרצת הבדיקה נלחץ - מורצת פעולת הבדיקה של המודל, והתוצאה מתעדכנת מתחת לכיתוב "Result".

```
elif event == "-RUNCHECK-":
      window["-RESULT-"].update(self.model.run_check(self.model.theta))
```

- אם כפתור חישוב השגיאה הממוצעת נלחץ - מורצת פעולת חישוב השגיאה הממוצעת, והתוצאה מתעדכנת מתחת לכיתוב "Result".

```
elif event == "-AVERROR-":
      window["-RESULT-"].update(self.model.average_error(self.model.the
ta)[0])
```

- אם כפתור הצגת הטטה נלחץ - מופיעים ערכי הטטה מתחת לכיתוב "Result".

```
elif event == "-SHOW THETA-":
      window["-RESULT-"].update("Theta:\n" + str(self.model.theta))
```

- אם כפתור הניקוי נלחץ - הטקסט מתחת לכיתוב "Result" נמחק מהמסך.

```
elif event == "-CLEAR-":
      window["-RESULT-"].update("")
```

- אם כפתור החיזוי נלחץ - נפתח מסך החיזוי על ידי הרצת פעולתו - data_enter_window.

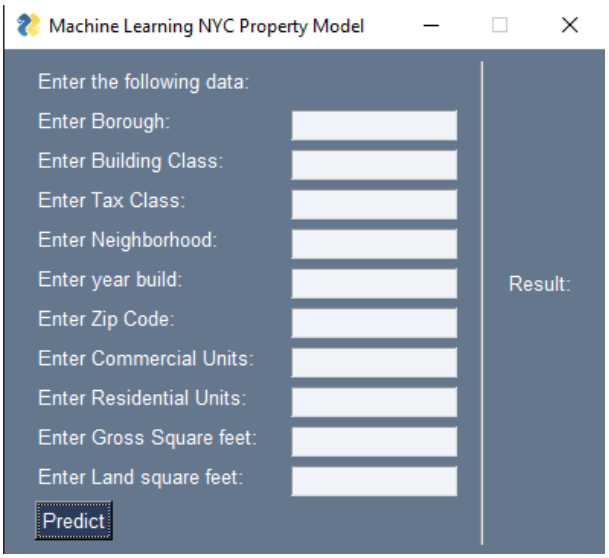
```
elif event == "-PREDICT-":
      self.data_enter_window()
```

לאחר יציאה מהלולאה, החלון נסגר.

```
window.close()
```

2.5.5.4 מסך הכנסת המידע - מסך החיזוי

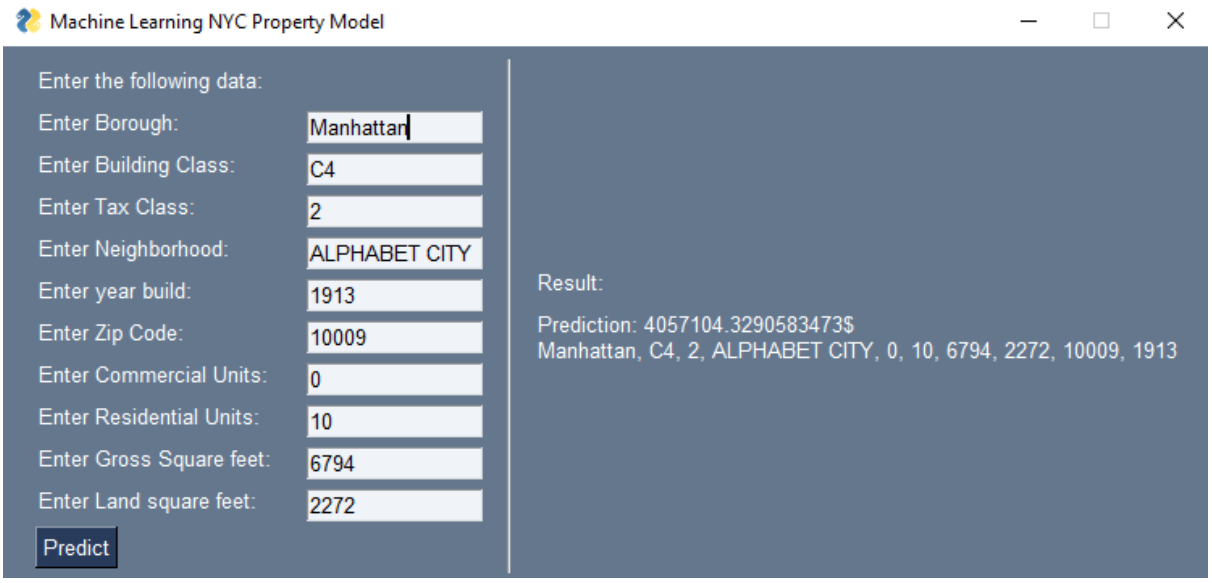
כאשר לחצנו על כפתור ה-Predict במסך האימון, נפתח מסך החיזוי:



חלון זה מחולק לשני חלקים:

- החלק השמאלי - מקומות להכנסת הערכים למאפיינים השונים
- החלק הימני - מקום להצגת תוצאת החיזוי

התוצאה לאחר הלחיצה על כפתור ה-Predict מופיעה תחת הכיתוב "Result", באופן הבא:



הקוד - הפעולה data_enter_window

תחילה, הקוד יוצר אובייקט מסוג window - אובייקט של הספרייה PySimpleGUI האחראי על פתיחת חלון. אחד הפרמטרים להם הוא זקוק לשם יצירת החלון הוא ה-Layout, אותו הכנו קודם לכן בפעולה הבונה.

```
def data_enter_window(self):
      layout = copy.deepcopy(self.data_enter_layout)
      window = sg.Window("Machine Learning NYC Property Model", layout)

      sg.Popup("Be sure you trained the model as you wished before you
check here the prediction!")
```

לאחר קפיצת הודעה זו, מתחילה לולאת while אינסופית אשר בה בכל איטרציה, תחילה נשמרים כל הערכים של כל ה-widgetים וכל ה-"אירועים" שקרו בחלון, במשתנים values ו-event בהתאם. לאחר מכן, בכל איטרציה, נבדק אם בעצם קרא אירוע ספציפי - אירוע מסמל אינטרקציה של המשתמש עם widget מסויים במסך - למשל, לחיצה על כפתור.

```
# Run the Event Loop
while True:
      event, values = window.read()
```

הלולאה בודקת את המקרים הבאים:

- אם המסך נסגר - במצב זה יוצא מהלולאה ובכך סוגר את האובייקט window ויוצא מהפעולה.

```
if event == "Exit" or event == sg.WIN_CLOSED:
      break
```

- אם הכפתור Predict נלחץ

```
if event == "-CHECK-":
```

  - תחילה, מכניסה את כלל הערכים שהוכנסו למשתנים המתאימים.

```
(borough, building, neighborhood, tax, c_units, r_units, g_feet,
l_feet, zipcode, year) = (values["-BOROUGH-"], values["-BUILDING-"],
values["-NEIGHBORHOOD-"], values["-TAX-"], values["-COMMERCIAL-"],
```

```
values["-RESIDENTIAL-"], values["-GROSS-"], values["-LAND-"],
values["-ZIPCODE-"], values["-YEAR-"])
```

- לאחר מכן, נעשית בדיקת כלל הערכים שהוכנסו, וקופצות הודעות במידה וערך כלשהו שהוכנס אינו מתאים.

```
containOther = False
for c in [tax, c_units, r_units, l_feet, g_feet, zipcode, year]:
      if not str(c).isdecimal():
            sg.Popup("Tax Class, Commercial Units, Residential Units,
Zipcode, Year, Land Square feet and Gross Square feet should only
contain numbers!")
            containOther = True

if containOther:
      continue

tax = int(tax)
c_units = int(c_units)
r_units = int(r_units)
l_feet = int(l_feet)
g_feet = int(g_feet)
zipcode = int(zipcode)
year = int(year)

if str(borough).lower() not in BOROUGH:
      sg.Popup("Enter Valid Borough!")
      continue

if str(borough).lower() not in BOROUGH:
      sg.Popup("Enter Valid Borough!")
      continue

if str(building).upper() not in BUILDING_CLASSES:
      sg.Popup("Enter Valid Building Class!")
      continue
```

```
if tax not in range(1,5):
      sg.Popup("Enter Valid Tax Class (1-4)!")
      continue

if year <= 0:
      sg.Popup("Enter Valid Year (Above 0)!")
      continue

year = 2017 - year

if zipcode <= 10000 or zipcode > 14975:
      sg.Popup("Enter Valid NYC Zipcode (10001-14975)!")
      continue

zipcode -= 10001

use_data = self.model.data

neigh_feature = "NEIGHBORHOOD_" + neighborhood.upper()
b_class_feature = "BUILDING CLASS AT TIME OF SALE_" + building.upper()
tax_class_feature = "TAX CLASS AT TIME OF SALE_" + str(tax)

one_hot_features = list(use_data)[7:-1]

if neigh_feature not in one_hot_features:
      sg.Popup("Sorry, There is no enough data for such neighborhood!")
      continue

if b_class_feature not in one_hot_features:
      sg.Popup("Sorry,  There  is  no  enough  data  for  such  building
class!")
      continue

if tax_class_feature not in one_hot_features:
      sg.Popup("Sorry, There is no enough data for such tax class!")
      continue
```

- לבסוף, מכינים את הערכים שהוכנו לוקטור בפורמט המתאים של מאגר המידע, ומחשבים באמצעותו את ערך ההיפותזה.

```
use_data = use_data.astype('float64')
for col in use_data.columns:
      use_data[col].values[:] = 0
use_data["BOROUGH_" + borough] = 1
use_data[neigh_feature] = 1
use_data[tax_class_feature] = 1
use_data[b_class_feature] = 1
use_data['RESIDENTIAL UNITS'] = r_units
use_data['COMMERCIAL UNITS'] = c_units
use_data['ZIP CODE'] = zipcode
use_data['BUILDING AGE'] = year
use_data['GROSS SQUARE FEET'] = g_feet
use_data['LAND SQUARE FEET'] = l_feet
use_data = use_data.to_numpy()
example  =  np.c_[np.ones(np.size(use_data,  0)),  use_data[:,  0:2],
use_data[:, 3:-1]]
example = example[0][:]

hypothesis = self.model.hypothesis(example, self.model.theta)
```

- מציגים את המחיר ואת ערכי כלל המאפיינים מתחת לכיתוב "Result".

```
window["-PRESULT-"].update("Prediction:  {h}$\n{b},  {b_c},  {tax},  {n},
{c},  {r},  {g},  {l},  {z},  {y}".format(h=hypothesis,  b=borough,  b_c  =
building, tax=tax, c = c_units, r = r_units, g = g_feet, l= l_feet, z =
zipcode + 10001, y = 2017 - year, n = neighborhood))
```

לאחר יציאה מהלולאה, החלון נסגר.

```
window.close()
```

2.5.5.5 הפעולה Main

בפעולה main, תחילה נוצר אובייקט מסוג GUI. לאחר מכן, מריצים את חלון הפתיחה. בודקים אם המודל נוצר כמו שצריך, ואם כן, ממשיכים למסך הראשי.

```
def main():
    ui = GUI()

    ui.start_window()

    if ui.model == NULL:
        exit()

    ui.main_menu()

if __name__ == "__main__":
    main()
```

מסקנות - סיכום וסקירה כללית של התשובות והתוצאות לשאלה

בפרויקט זה, הצבתי את השאלה, "האם ניתן לחזות ערך של נדל"ן?". זמן קצר לאחר שאילת שאלה זו, הבנתי שאכן, קיימת אפשרות זו. כתוצאה מכך השאלה התפתחה לשאלה נוספת - "כיצד?". בעבודה זו, ניסיתי לענות על שאלה זו.

על מנת לענות על השאלה, התחלתי מלחקור את כל התחומים להם אזדקק. חקרתי על אלגברה לינארית, תחום הנדל"ן, שפות התכנות השונות בהן אשתמש ועוד. לאחר שלב זה, המשכתי את תהליך המחקר במחקר על למידת מכונה - סוגי למידת מכונה, סוגי הבעיות השונות, השיטות השונות ללמידה, והבעיות שעלולות להופיע.

הבנתי שעל מנת ליצור מודל חיזוי לערך נדל"ן, הבעיה אותה הגדרתי היא בעיית רגרסיה לינארית תחת למידה מפוקחת.

המשכתי בייצור אב-טיפוס בשפת MATLAB, בעזרת אב-טיפוס זה, הרחבתי את ההבנה שלי במתמטיקה והאלגוריתמיקה העומדות מאחורי למידת מכונה, והבנתי את אופן פעולת התוצר העתידי שלי בשפת Python.

התחלתי בייצור התוצר הסופי בשפת Python. בשלב זה, הבנתי לעומק יותר תכנות מונחה עצמים, ואופטימיזציות שונות. בשלב זה, הבנתי לעומק יותר את הבדלי השיטות מורד הגרדיאנט ומשוואת הנורמל (Normal Equation).

כמו כן, למדתי וצברתי ניסיון באופן יישומי בתחום חקר הנתונים, ושלבי ה-Data Cleaning ו-Feature Engineering. למדתי על ההשפעה של תהליך ניקוי הנתונים והגדרת המאפיינים על דיוק החיזוי של המודל, ועסקתי רבות בפתירת בעיית התאמת חסר ברגרסיה לינארית ומניעת בעיית התאמת יתר, בדרכים שונות ביניהן ניקוי יסודי מאוד של מאגר הנתונים והגדרת המאפיינים.

בתהליך פתירת בעיות אלו, יישמתי כמו כן, אלגוריתם לבחירת מודל - Model Selection Algorithm, אשר עזר לי להבין את אופן בחירת המאפיינים, ולבדוק את המאפיינים אותם בחרתי.

לבסוף, הגעתי לתוצר סופי מרשים במיוחד, העונה על שאלת המחקר. המודל בעל שגיאת חיזוי ממוצעת של פחות מ-20% והפרש מחירים ממוצע של כ-$180,000 בהתחשב בכך שהמחירים נעים בין 400 אלף ל-6 מיליון!

במהלך העבודה למדתי דברים רבים:

- למדתי רבות על תחום למידת המכונה וחקר הנתונים.
- צברתי ניסיון במחקר והבנה של שיטות שונות לבניית מודל מתמטי מדויק ככל האפשר לפתרון בעיה שהוגדרה.
- התנסתי בכתיבת אלגוריתם מורכב בתחום למידת מכונה.
- הרחבתי את הידע שלי באלגברה לינארית ויישומה במודלים מורכבים של למידת מכונה לצורך עבודה יעילה יותר עם נתונים.
- למדתי את שפת MATLAB ועל השימוש הרחב שלה בעולם שלנו. התנסתי בכתיבת אב-טיפוס בשפה זו.
- למדתי על שפת Python, והתנסתי בכתיבת פרויקט גדול בשפה זו - הכוללת יישומים מתמטיים מורכבים בלמידת מכונה ויצירת ממשק משתמש גרפי (GUI).
- יישמתי מודל אשר עונה על בעיה מהעולם האמיתי תוך שימוש במידע אמיתי ככל האפשר - רישומים של משרד האוצר של עיריית ניו יורק.
- לבסוף, הגעתי למטרתי העיקרית, מימשתי את המחקר ליצירת תוצר העונה על השאלה.

על ידי התוצר הסופי עניתי על מספר שאלות, בינהן:

- מדוע השגיאה הממוצעת של המודל לא שואפות ל-0%? התשובה לכך, היא שמטרתו להגיע לחיזוי הטוב ככל האפשר עבור דוגמאות אימון חדשות. בעצם, מטרתנו לא לשאוף לאחוז שגיאה של 0% כיוון שבמצב זה החיזוי עבוד דוגמאות אימון חדשות יפגע מאוד.

- האם על ידי שימוש ברגרסיה לא לינארית, לא היינו לאחוזי שגיאה נמוכים יותר? התשובה לכך, שכן, היינו מגיעים לאחוזי שגיאה נמוכים יותר, אך כפי שהסברתי, אז היה עלול להפגע החיזוי בעבור דוגמאות אימון חדשות. זה לא אומר שאי אפשר היה לעשות גם מודל לא לינארי שיחזה טוב גם בעבור דוגמאות אימון חדשות, אך הדבר היה דורש הרבה יותר זמן וכוח עבודה.

כיצד אני יכול לסכם פרויקט חקר זה. אני מחשיב פרויקט זה להצלחה גדולה עבורי. כחלק מהעבודה על הפרויקט למדתי רבות על תחומים שונים ומגוונים. יצרתי תוצר סופי העונה על השאלה אותה הצבתי לעצמי בתחילת העבודה, שאלה מהעולם האמיתי אשר בעלי מקצוע רבים עוסקים בה ביום-יום. כמו כן, התנסתי במחקר ברמה אקדמית. לבסוף, הגעתי לתוצר סופי מרשים ביותר שמצליח לחזות עם אחוזי שגיאה ממוצעת נמוכים מאוד (גם בעבור דוגמאות אימון חדשות) את ערכו של נכס בניו-יורק.

הרעיון העיקרי של המודל הוכח - ככל שהמודל יקבל יותר נתונים נקיים, הוא ילמד נכון יותר וכך התוצאות שלו יהיו יותר ויותר מדויקות. לאחר ניקוי יסודי מצדנו, הצלחנו לפתור את בעיית התאמת החסר שהייתה לנו, ולשפר רבות את חיזויי המודל עד למצב של אחוז שגיאה ממוצעת של פחות מ-20%!

לבסוף, יש לי מספר רעיונות להמשך העבודה. אם היה לי עוד זמן, הייתי רוצה לחקור מודלים לא ליניאריים מסוגים שונים ולנסותם. למשל, לממש מודל מבוסס רשתות נוירונים או אפילו להגדיר את הבעיה בתור בעית clustering ולא רגרסיה.

ביבליוגרפיה

קורס

- חומרי קורס "Machine Learning" של אוניברסיטת סטנפורד (ארה"ב), מנחה: אנדרו נייג.

למידת מכונה

- Smola A. and Vishwanathan S.V.N. (2008), Cambridge University Press. “Introduction to Machine Learning”, Chapter 1.

- Maimon O. and Rokach L. (2005), Tel Aviv University. “Data Mining and Knowledge Discovery Handbook”, Chapter 8: “Supervised Learning”, pages 133-147.

- Google. “Clustering in Machine Learning”. https://developers.google.com/machine-learning/clustering/clustering-algorithms

- Sharma R. (2020), UpGrad. “What is Clustering and Different Types of Clustering Methods”. https://www.upgrad.com/blog/clustering-and-types-of-clustering-methods/

- IBM Cloud Education (2020). “Machine Learning”. https://www.ibm.com/cloud/learn/machine-learning

- Wakefield K. SAS. “A guide to the types of machine learning algorithms and their applications” https://www.sas.com/en_gb/insights/articles/analytics/machine-learning-algorithms.html

אלגברה לינארית

- האוניברסיטה הפתוחה. (1998), "אלגברה לינארית I".

MATLAB

- MathWorks. MATLAB Documentation.
  https://www.mathworks.com/help/matlab/

פייתון - Python

- Python 3.8 Documentation
  https://docs.python.org/3.8/

- Pandas Library Documentation
  https://pandas.pydata.org/docs/

- NumPy User guide
  https://numpy.org/doc/stable/user/index.html#user

- Matplotlib User Guide
  https://matplotlib.org/stable/users/index

- Seaborn Tutorial
  https://seaborn.pydata.org/tutorial.html

- PySimpleGui Documentation
  https://pysimplegui.readthedocs.io/en/latest/

נדל"ן

- שיחת ייעוץ עם סער ליטמנוביץ', מנכ"ל חברת 8Estate.

- The City of New York Department of Finance. (2021), “Glossary of Terms for Property Sales Files”.
  https://www1.nyc.gov/assets/finance/downloads/pdf/07pdf/glossary_rsf071607.pdf

- The City of New York Department of Finance. (2021), “Building Classification | City of New York”.
  https://www1.nyc.gov/assets/finance/jump/hlpbldgcode.html