# קורס הדמיית נתונים - סיכום

## תוכן

# שיעור 1

## הקדמה

1. אחרי שלמדנו הרבה ספריות ומודלים וכו' חשוב להבהיר מספר דברים – לא בהכרח מי שזוכר את כל הספריות בעל פה הוא מתכנת יותר טוב. ברור שזה יקצר לו את זמן העבודה כי הוא לא יצטרך להיכנס כל רגע לstuck overflow או לאתרים אחרים לבדוק איך עושים פעולות שלפעמים הן מאוד פשוטות, אבל העניין המהותי הוא לא רק לזכור פונקציות אלא להבין אותן באמת, אילו סוגי ערכים מודל מסוים מקבל ועם מה הוא לא יודע להתמודד, למה לבחור אלגוריתם אחד במקום אלגוריתם אחר וכו', אחד מהדברים שנעשה בקורס – ננסה לצלול ולהבין את העומק של הדברים. דבר נוסף – אנו לומדים הרבה דברים בתואר וחלקם חשובים מבחינת פיתוח כושר אנליטי אבל אין להן בהכרח השפעה ישירה על העבודה שלנו כdata scientists לכן מעבר לכל נושא התכנות וידיעת הספריות מי שרוצה לשלוט בכל תחום הmachine learning חייב להיות מבוסס היטב בתחומים של סטטיסטיקה, הסתברות, ואלגברה לינארית – אלו אבני הבניין המשמעותיים.

## מה נלמד בקורס

2. נושאים נוספים עליהם נלמד בקורס:

א. Pandas – יותר ממה שהתרגלנו אליו כמסדר טבלה שדומה לאקסל ותו לא – נכיר לעומק, למשל גרפים מתקדמים המתעדכנים און ליין או גרפים עם פרמטר שניתן להזיז וכו'.

ב. Time series – נבין מתי המיקום של שורה בpandas מהווה גורם משפיע. נוכל להבין את העיקרון הזה למשל ע"י ההבנה שניתן לחזות באופן טוב את מזג האוויר של מחר ע"י מזג האוויר של היום, זה בא לידי ביטוי בעוד תחומים רבים ואחד מהבולטים בו הוא למשל מניות – ולנתח את הסיבתיות של העלייה/ הירידה בהתאם למה שקדם אליה.
(במאמר מוסגר נציין שכאשר נבנה מודל על דאטה מסוג כזה הtrain שלנו יהיה כל החלק הראשון של הדאטה, והtest יהיה החלקים האחרונים, כי האירועים שקדמו הם אלו שהובילו לאלו באו אחריהם. בנוסף אם אין לי test ואני באמת רוצה לחזות אני יכול למשל לבנות מודל שמקבל את ה60 ימים האחרונים ומחזיר את החיזוי ליום הבא, אבל אז אני יכול להפעיל אותו על ה59 הישנים וה1 החדש, ולקבל חיזוי נוסף, וכך להמשיך ולחזות מספר ימים קדימה. יש בכך הרבה יתרון אך חשוב לזכור שה"מערכת" הזו מעולה, ויכולה לתת תוצאות נכונות, אך פעמים רבות החיזוי שלה הוא <u>חיזוי של העבר</u>, ובמציאות כמו שאנחנו יודעים יש דברים שאין אלגוריתם שיכול לחזות בגלל הכמות הגדולה כ"כ של משתנים).

ג. Ensemble learning - שיטות מורכבות – בתמצות – במקום להביא רעיון אחד פנומנלי שיכול לפתור חלק מאוד גדול מהמקרים – ניתן להשתמש בשיטה אחרת והיא הבאת המון רעיונות קטנים. נניח שיש לך מטבע לא הוגן, בעל סיכוי של 52% ליפול על עץ וסיכוי של 48% ליפול על פאלי. למעשה אם אתה רוצה לדעת האם הצד של ה- 52% הוא עץ או פאלי, הרי שלא תוכל להסתפק רק בזריקה אחת, כי מזריקה אחת לא תוכל ללמוד דבר. אולם, אם תטיל את המטבע 1,000 פעמים, הרי שישנה סבירות גבוהה שנראה עץ הרבה יותר פעמים מאשר מפאלי. אחת מהשיטות הכי מוכרות לזה הוא מודל הrandom forest. כאשר נפעיל אותו לרוב נשתמש בהמון עצי החלטה n_estimators אבל העומק של העצים (max depth) הוא נמוך (לרוב בין 1 ל3) – כלומר כמה "שאלות"/פיצולים אני נותן לכל עץ. במקום למפות ולמצוא מה

המילה הכי נפוצה שיכולה לעזור לי לסווג האם הודעה היא ספאם, הרעיון של random forest הוא לקחת המון כללים "טיפשיים" ולא בהכרח ממצים, ומחבר אותם – הדיוק שלו הוא גבוה. יתרון נוסף של עצי החלטה זה שניתן להסביר באופן יחסית ברור למה דווקא זו ההחלטה שהתקבלה, ולא לומר "המודל קבע שזה מה שנכון לעשות".

ד. Naïve base – הנחה (נאיבית) שאין תלות בין משתנים. כלומר בהינתן הרבה פיצ'רים ופרמטרים משפיעים – ניתן לחלק לכל אחד מהם בנפרד מהשני. (בעוד שפעמים רבות המציאות הוא שיש קשר- למשל שלא כדאי לשחק טניס אם מעונן, גשום, ויש רוח חזקה. יכולים להיות מצבים בהם יש רוח חזקה אבל לא מעונן ואין גשם וכד', ואז לשאול האם מומלץ לשחק או לא, אבל באמת אלו פרמטרים שקשורים קשר ישיר אחד לשני).
p(h/x1,x2,x3) = p(h/x1) * p(h/x2) * p(h/x3). עם זאת, למרות שהעולם הוא לא נאיבי – פעמים רבות מניחים הנחה נאיבית ובכל זאת מקבלים תוצאות ממש טובות.

ה. Clustering – (פירוש פשוט – מקבץ/ אשכול) עד עכשיו למדנו על סיווג במודלים של supervised learning, כלומר שאני נותן למודל את הדאטה עם "התשובות" לדאטה (labels) למשל את הגדלים של עלי הכותרת של אירוס ואיזה זן זה. אנו כיוון אחר ללמידת מכונה וזה באמצעות unsupervised learning, על מנת להצליח לסווג דאטה שאקבל מבלי שאקבל את הלייבלים. בדוגמה של האירוסים זה לקבל את הדאטה מבלי שאני יודע שיש באמת 3 זנים, איך אני מחלק? מה הפרמטרים?

ו. PCA – אנו מסוגלים לדמיין עד 3 מימדים – אורך, רוחב ועומק, כאשר יש לנו הרבה מימדים זו בעיה שקשה לעבוד איתה כי קשה להבין את המבנה הפנימי שלה, מה נמצא ליד מה וכו'. PCA מנסה 'להוריד' את המימד של הבעיה בצורה שיהיה ניתן לצייר אותה ועדיין לא לאבד את המהות של המידע. למשל אם יש לי נקודה על מרחב תלת מימדי x,y,z אני יכול לפשט את הבעיה יותר ולקבל מידע שמאפיין לי את המידע המקורי למשל ע"י רשימה של כל המרחקים של כל נקודה מראשית הצירים.

ז. T-SNE – אלגוריתמים המאפשרים לממש ניתוח ויזואלי, למשל זיהוי של מספרים שנכתבו בכתב יד. (למשל במובן של סיווג תמונה עם המספר 8 מבחינת האלגוריתם 8 קרוב יותר למספר 3 מאשר למספר 7, כי ויזואלית 8 ו3 דומים יותר ממה שדומים 8 ו7)

**<u>בריף פעולות בסיסיות בPandas</u>**

3. פקודת head וtail (ברירת מחדל ל5 הראשונים או האחרונים) למשל - titanic.head(20)
4. slicing ע"י אינדקס titanic_slice = titanic.iloc[:10,[2:3]] – לחיתוך ה10 שורות הראשונות, עם עמודות באינדקס 2 עד 3. (iloc = index location)
   או חיתוך ע"י שם הערך - titanic_slice = titanic.loc["name"]
   דגש חשוב – כאשר אני עושה חיתוך הוא ישנה לי אותו בדאטה סט האמיתי, לכן אם ארצה להימנע מזה אני מוסיף: titanic.iloc[:10,[2:3]].copy()
5. פונקציית groupby יותר מכל פונקציה אחרת יכולה לעזור בהבנת הנתונים, היא במובן מסוים מבצעת פעולות הדומות לpivot table באקסל.
   נבין כי כאשר אנחנו מבצעים השמה של groupby על data frame אנחנו לא יוצרים דאטה פריים אלא מעין אובייקט groupby. לכן אם נכתוב למשל: titanic_slice.groupby("sex"), הפלט יהיה: <pandas.core.groupby.generic.DataFrameGroupBy object> ועליו ניישם פקודות.

א. groupby.groups – על מנת לראות את הקבוצות של המשתנים, למשל:
gbo = titanic_slice.groupby("sex") , אם נריץ את הפקודה gbo.groups
הפלט יהיה: {'female': [1, 2, 3, 8, 9], 'male': [0, 4, 5, 6, 7]}

ב. l = list(gbo) – יצירת רשימה מכל אחת מהקבוצות.

ג. אנחנו מכירים שיש לנו יכולת לעשות סינונים ע"י בחירת המשתנים אבל ניתן גם ע"י groupby, למשל: titanic_slice.groupby( by='sex').age.mean( )
הפלט יהיה ממוצע הגילאים של כל מין. ניתן להציג במקום mean את describe /std /max.
היתרון של זה הוא שניתן להגיע לתוצאות מאוד מהירות ומאוד אינדיקטיביות, למשל- שבטיטאניק "עדיף" היה להיות אישה במחלקה ראשונה:

```
titanic.groupby(by=['sex','pclass']).survived.mean()
```

```
sex     pclass
female  1         0.968085
        2         0.921053
        3         0.500000
male    1         0.368852
        2         0.157407
        3         0.135447
Name: survived, dtype: float64
```

איך הקוד עובד? הוא מסנן לפי מין, ולכל מין לפי המחלקה (1 הכי גבוהה) וכל מי שאכן שרד מופיע בדאטה תחת survivred 1, ולזה הוא עושה ממוצע ביחס לכמות האנשים שתחת אותו קטגוריה ולא שרדו. יש לזה המון מסקנות, לדוג' שהמחלקה משחקת תפקיד פחות משמעותי מאשר המין.

ד. אם אני רוצה לקבל את כמות האיברים לכל תת פרמטר לפי אותה חלוקה של מין ומחלקה:
titanic.groupby(by=["sex","pclass"].size( )

ה. אם אני רוצה שהפלט שלי לא יהיה כמו שהוא מופיע בתמונה למעלה אלא שיוצג כצורה של טבלת pandas נכניס אותו למשתנה ונדפיס את המשתנה, למשל:

```
new_df = titanic.groupby("sex").mean()
```

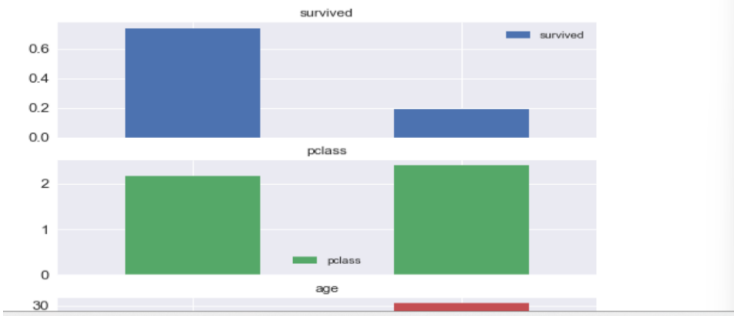
```
new_df
```

| sex | survived | pclass | age | sibsp | parch | fare |
|---|---|---|---|---|---|---|
| female | 0.742038 | 2.159236 | 27.915709 | 0.694268 | 0.649682 | 44.479818 |
| male | 0.188908 | 2.389948 | 30.726645 | 0.429809 | 0.235702 | 25.523893 |

הבאנו את הדוגמא הזו עם הmean לא רק כדי להבהיר איך עושים באופן ויזואלי אלא כי עכשיו עם הטבלה שהתקבלה הgroupby מאפשר לנו להשוות ממוצעים בכל אחד מהפרמטרים ולקבל הבנה על הדאטה. כמו כן ניתן להציג אותו בצורה גרפית:

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use("seaborn")
```

```
new_df.plot(kind = "bar", subplots = True, figsize = (8,15), fontsize = 13)
plt.show()
```



ו. sort_values – מיון הערכים שהתקבלו לי לפי פרמטר מסוים, למשל:
summer.groupby(by=["Country","Medal"]).Medal.count().sort_values(ascending=False)
מה שהקוד עושה זה לסנן לפי מדינה, וכל מדינה לפי המדליות שזכתה בהם – מה הכמות של המדליות בתת קטגוריה הזו, ואת כל מה שהתקבל תמיין בסדר יורד מהגבוה (ascending=False)

6. Type – מציאת סוג משתנה מסוים, למשל type(l[0]) יחזיר tuple למשל, ו type(l[0][0]) יחזיר str למשל.

7. nunique – מציאת כמות הערכים השונים תחת קטגוריה מסוימת, למשל בדאטה-סט של כל משתתפי האולימפיאדה אני רוצה לדעת כמה מדינות יש: summer.Country.nunique()

## שיעור 2

### המשך עבודה עם groupby

1. groupby עוזר לנו להבין את הדאטה בצורה יותר טובה, כך שמומלץ מאוד "לשחק" עם הנתונים שמכניסים אליו ולבדוק האם ישנן מסקנות/ עובדות מעניינות שיכולות לפתוח פתח לכיוון מחשבה משמעותי. למשל – מה הממוצע בכל אחד מהפרמטרים עבור גבר או אישה בטיטניק: למשל נוכל לראות שממוצע הנשים ששרדו אל מול ממוצע הגברים ששרדו הוא משמעותי מאוד.

```
new_df = titanic.groupby("sex").mean()
```

```
new_df
```

| sex | survived | pclass | age | sibsp | parch | fare |
|---|---|---|---|---|---|---|
| female | 0.742038 | 2.159236 | 27.915709 | 0.694268 | 0.649682 | 44.479818 |
| male | 0.188908 | 2.389948 | 30.726645 | 0.429809 | 0.235702 | 25.523893 |

2. הערה לגבי survived – הסיבה שמוצג לי כמות האחוזים ששרדו מהגברים ומהנשים הוא כי הוגדר ש'שרד' יסומן ב1 ו'לא שרד' יסומן ב0, ואז כל לבצע ממוצע.

3. יתרון נוסף שיש לgroupby היא היכולת לסנן בדיוק את המידע הרלוונטי בצורה מאוד מהירה, למשל סינון לפי מדינה – את ה20 שזכו בהכי הרבה מדליות ואיזה סוג זה היה. זה במקום לעשות sort_values ואז ascending=false ולזה לעשות head(20), הnlargest מקצר את כל הפעולות האלו לפעולה אחת.

```
medals_per_country = summer.groupby(by=["Country","Medal"]).Medal.count().nlargest(n = 20)
medals_per_country
```

```
Country  Medal
USA      Gold      2235
         Silver    1252
         Bronze    1098
URS      Gold       838
         Silver     627
GBR      Silver     621
URS      Bronze     584
GBR      Bronze     553
         Gold       546
FRA      Bronze     497
         Silver     491
```

4. הפרטמר של הגיל הוא פרמטר שמאוד קשה לעבוד איתו, כי הוא לרוב פרוס על ספקטרום גדול, וגם הוא לא בהכרח אינדיקטיבי. לכן אפשר לעשות פעולות מסויימת כדי לקבל אינפורמציה. למשל – אנו יכולים לשער שהמבוגרים דאגו יותר להציל את הילדים כך שנצפה לראות שאחוז הילדים ששרדו מול המבוגרים ששרדו הוא גדול משמעותית. נגדיר עמודה חדשה של מבוגר, ילד, או תינוק. ונראה את החלוקה באחוזים של מי ששרד – 70 אחוז מתוך 44 התינוקות שהיו שרדו, לעומת 36 אחוז מ781 המבוגרים שרדו. לעומת זאת מתוך הילדים 42 אחוז מ66

```
titanic["ad_chi"] = "adult"
titanic.loc[(titanic.age > 6) & (titanic.age<18) , "ad_chi"] = "child"
titanic.loc[titanic.age < 6 , "ad_chi"] = "toddler"
titanic.ad_chi.value_counts()
```

```
adult      781
child       66
toddler     44
Name: ad_chi, dtype: int64
```

```
titanic.groupby('ad_chi').survived.mean()
```

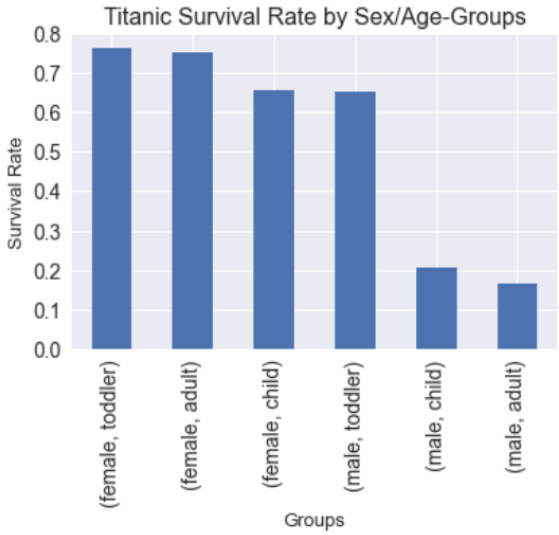
```
ad_chi
adult      0.362356
child      0.424242
toddler    0.704545
Name: survived, dtype: float64
```

הילדים שרדו, מכאן אני יכול לחשוב אולי יש trashold – רף - מסוים שהפך ילד למבוגר מבחינת הדאגה לו, למשל לנער בן 17/18 שלא התייחסו אליו כמו לילד בן 8.

כך שבסוף נוכל ליצור גרף המתאר לנו "מה עדיף היה להיות בטיטניק?" ונראה למשל להיות תינוקת או אישה מבוגרת היו סטטיסטית אלו ששרדו באחוזים הגבוהים ביותר – קרוב ל80 מהן.

```
w_and_c_first = titanic.groupby(["sex", "ad_chi"]).survived.mean().sort_values(ascending = False)
```

```
w_and_c_first.plot(kind = "bar", figsize = (6,4), fontsize = 14)
plt.xlabel("Groups", fontsize = 13)
plt.ylabel("Survival Rate", fontsize = 13)
plt.title("Titanic Survival Rate by Sex/Age-Groups", fontsize = 16)
plt.show()
```




**<u>ממוצע וההסתייגות ממנו</u>**

5. בשונה מmean – הממוצע, החציון (median) נותן את הערך של האיבר הנמצא בדיוק בחצי. <u>מתי נעדיף להשתמש בחציון במקום בממוצע? כאשר יש outliers גדולים</u>, כלומר כשיש איברים מסוימים עם ערכים גבוהים מאוד ביחס לאחרים כך שהם עלולים להסיט את התשובה מהאמת. אם נשאל לדוגמא מה המחיר הממוצע של חברה עם 100 עובדים המרוויחים בממוצע 7000 שקל, אם המנכ"ל והסמנכ"ל מרוויחים יחד 200,000 שקל הממוצע של כולם יהיה 8820 דבר שלא באמת מעיד על האמת.

לכן נבין ש<u>בדאטה שמתפלג בצורה נורמלית הממוצע והחציון דומים</u>.

6. הערה לגבי outliers – בכל דבר <u>שהטבע</u> יצר (גיל, משקל, גובה, מס' ילדים וכו') – לא יכולים להיות חריגות משמעותיות כל כך שישנו/ ישפיעו בצורה חריגה. ההפרש בין משקל האדם הממוצע למשקל האדם הכי שמן בעולם הוא לא יותר מאשר פי 3, כך שגם אם 200 קילו זה משקל חריג, אם המשקל הממוצע של 100 אנשים זה 80, אם נוציא אדם אחד ונכניס במקומו אדם ששוקל 200 הממוצע יהיה 81.2. כסף למשל זה לא דבר טבעי, לכן ההפרשים של בודדים יכולים להיות פי כמה יותר מאשר אחרים.

באופן כללי נוכל לומר ש<u>ניתן (או כדאי) לעבוד עם התפלגויות נורמליות כאשר הסטיית תקן היא בטווח של עד 3.5 / 4 סטיות תקן מהממוצע</u>. לכן כאשר אני בודק ממוצע, או מסתכל על האם קיימת התפלגות נורמלית אני חייב לדעת גם מהי סטיית התקן, כלומר האם הממוצע באמת מעיד לי על כך שרוב האנשים נמצאים סמוך לממוצע (לדוגמא 10 אנשים בחדר, חמישה בני 80, וחמישה

בני 10, הממוצע הוא 45 אבל אין לזה שום קשר לעובדות של מי שנמצא בחדר).
הסבר לחשיבות סטיית התקן ועל כך שלא ניתן לסמוך רק על ממוצע והתפלגות נורמלית נוכל ללמוד מהקריסה הכלכלית שהייתה ב2008, הקריסה הזו נבעה מכך שהבנקים בעולם ראו שישנה התפלגות נורמלית כך שבממוצע הרוב המוחלט של האנשים שלוקחים משכנתא גם מחזירים אותה, ולפי ההנחה על ההתפלגות הזו נתנו הלוואת משכנתא כמעט לכולם. אבל המציאות לא חייבת דין וחשבון להתפלגות הנורמלית, והמון אנשים לא הצליחו להחזיר את ההלוואות, והבנקים קרסו. היום זה ברור לנו כמו שברור לנו שאין סיבה לעשות תחזית מזג אוויר לעוד שנה.

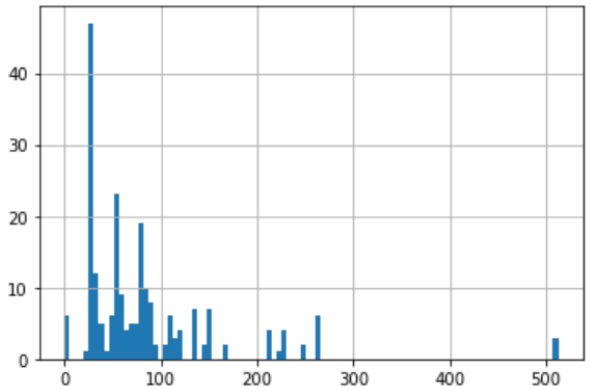
7. דוגמא נוספת מהדאטה סט של הטיטניק: נסתכל על כל מחלקה (pclass) ועל כמה שילם כל נוסע עבור הכרטיס (fare), נקבל את הנתונים הבאים: אנו רואים שהממוצע של הכרטיס במחלקה ראשונה היה 84 דולר, עבור סטיית התקן קיבלנו סטיית תקן של 78! כלומר שיכול להיות שהיו כאלו ששילמו 84 עבור הכרטיס אבל גם היה כמו לא מבוטלת בכלל של אנשים ששילמו מעט מאוד, ואנשים ששילמו הרבה יותר מאשר כולם. אם נציג זאת בגרף נוכל לראות זאת אף בצורה הרבה יותר מובחנת – ציר הy זה כמות האנשים, וציר הx זה מחיר הכרטיס. אנו רואים שמעל 40 אנשים שילמו כ20 דולר עבור הכרטיס, בעוד יש אנשים ששילמו 100/ 200 או אפילו 500 דולר עבור אותה מחלקה. אנחנו מבינים שממוצע של 84 דולר לא מצביע על מה שהיה באמת ובנוסף אנו יכולים להסיק מסקנות רבות שאחת מהן היא שאולי היו אנשים ש"סידרו" להם כרטיסים זולים משמעותית מאנשים אחרים שדרשו מהם מחיר מופקע ביחס למה שדרשו מאחרים. אנחנו חייבים לנהל דיאלוג עם הדאטה ואם לא נשאל שאלות עמוקות יותר לא נצליח להסיק מסקנות משמעותיות באמת.

```
titanic.groupby("pclass").fare.mean()
```

```
pclass
1    84.154687
2    20.662183
3    13.675550
Name: fare, dtype: float64
```

```
titanic.groupby("pclass").fare.std()
```

```
pclass
1    78.380373
2    13.417399
3    11.778142
Name: fare, dtype: float64
```




8. אם נשים לב – הצלחנו להוציא המון מסקנות רק מללמוד את הדאטה ועם מעט מחשבה, לא בנינו שום מודל עדיין. למה חשוב להבין את זה? כי למערכת אין ידע מוקדם, אבל לנו עוד לפני שהגענו להתעסק עם המודל, יש למשל את היכולת לחשוב על כך שהגיוני שיהיה תיעדוף בהצלה של ילדים ונשים – "women and children first" – נרצה קודם לראות את זה. והנחות עבודה אבסולוטיות למשל שאדם מת הוא מת ולא יחזור להיות חי – המערכת לא יודעת את זה ואנו נדרשים לקודד לה את זה.

# שיעור 3

## זיהוי outliers

1. כמו שראינו בשיעור קודם ישנה חשיבות רבה בבדיקת סטיית התקן ולא רק לסמוך על הממוצע, מאחר ויכולים להיות outliers שמשפיעים מאוד על הממוצע ומייצגים מצג שווא לפעמים. לכן לזיהוי הoutliers יש חשיבות רבה מאוד, גם כדי לנטרל אותם במידת הצורך, אבל גם ללמוד מהם ודרכם להגיע לתובנות חדשות.

נמשיך עם הדאטה סט של הטיטניק וניקח לדוגמא את ההסתברות לכך שהנוסע שרד בהתאם למין והמחלקה בה היה, למשל 96.8% מהנשים ממחלקה ראשונה שרדו, וכן רק 13.5%

```
titanic.groupby(["sex", "pclass"]).survived.mean()
```

```
sex     pclass
female  1         0.968085
        2         0.921053
        3         0.500000
male    1         0.368852
        2         0.157407
        3         0.135447
Name: survived, dtype: float64
```

```
titanic["probability_survived"]= titanic.groupby(["sex", "pclass"]).survived.transform("mean")
titanic.head()
```

| | survived | pclass | sex | age | fare | group_mean_age | provba_survived |
|---|---|---|---|---|---|---|---|
| **0** | 0 | 3 | male | 22.0 | 7.2500 | 26.507589 | 0.135447 |
| **1** | 1 | 1 | female | 38.0 | 71.2833 | 34.611765 | 0.968085 |
| **2** | 1 | 3 | female | 26.0 | 7.9250 | 21.750000 | 0.500000 |
| **3** | 1 | 1 | female | 35.0 | 53.1000 | 34.611765 | 0.968085 |
| **4** | 0 | 3 | male | 35.0 | 8.0500 | 26.507589 | 0.135447 |

מהגברים במחלקה שלישית שרדו בעוד 86.5% מהם לא שרדו. נוכל לבצע פעולת transform לעמודה חדשה ובכך נבצע השמה לכל אדם את האחוז שלו לכך ששרד בהתאם לנתונים. מכאן אנו יכולים לנסות לזהות outliers, למשל 96.8% מהנשים במחלקה ראשונה שרדו, אחוז קטן מאותן נשים בכל זאת לא שרד, האם יש ביניהן קשר? באותו אופן כלפי הגברים ממחלקה שלישית שכן שרדו – מה בכל זאת קושר אותם? אנו יכולים אולי לגלות שרוב אלו ש"ניצחו את הסטטיסטיקה" אלו אנשים שבsiblings שלהם מופיע 0/1 כלומר שלא בהכרח היו צריכים לדאוג למישהו והם היו יכולים לדאוג להציל את עצמם.

2. הערה – בנקודה זו חשוב מאוד לנסות לחדד מה מוגדר מבחינתי outlier, למשל מי ששרד למרות שהיה לו 30%? או 15%? או 5%? איפה הסף? זה עניין של כל מקרה לגופו, וצריך להשתמש בכלי החשוב והיעיל – מוח. (למשל אם אני מטיל מטבע 10,000 פעמים וב53% יצא עץ – מבחינתי זה מטבע לא מאוזן ושקרי, למרות שזה רק 3%, במספר נמוך של הטלות יכולים להתקבל חריגות , למשל בהטלה 5 פעמים הגיוני שייצא 4 פעמים עץ, אבל כאשר יש כמות גדולה מאוד ההתפלגות אמורה להתאזן, לעומת הדאטה סט של הטיטניק שהבדלים של 3% לא עושים הבדל תהומי)
3. הערה נוספת – אמרנו שאנו מעוניינים למצוא קשר בין החריגים, אבל העובדה היא שלא תמיד יש.

## Data cleaning

4. כמעט בכל דאטה שניקח אנו נקבל נתונים "מלוכלכים" זה יכול לקרות מהרבה סיבות, ניקח לדוגמא מסד נתונים שספר אנשים עובדים עליו וממלאים פרטים על לקוחות, בקטגורית מין יכולים לכתוב זכר/ ז או לפעמים נלחץ מקש לא נכון והוקלד 'זכער', וזה כאשר יש 2 אופציות, כאשר יש הרבה אפשרויות האופציות לא נגמרות. ניתן לייצר קובץ שלא מאפשר לסטות מהגדרות הפרמטרים אבל לרוב הדאטה מגיע עם חריגות רבות שלא מאפשרות לקרוא את הנתונים באופן חלק. הנחת עבודה היא שצריך לנקות את הדאטה ולזרוק נתונים שמפריעים, עם זאת כדי להימנע

מזריקה של נתונים סתם. למשל כאשר אנו בוחנים את הדאטה בinfo אנו יכולים לראות כמה תאים מלאים יש לנו, ולעיתים יש הרבה תאים ריקים - null, ועדיף להימנע במידת האפשר מdropnan – זריקת כל השורות המכילות ערכי null.

5. אחת הדרכים לטפל בתאים ריקים היא להכניס לאותם ערכים את הממוצע של כל העמודה, ובכך לא לפגוע בממוצע, מתוך הנחה שהדבר לא יפגע בהתפלגות של הנתונים.
אבל אפשר לעשות זאת בצורה יותר יסודית, למשל לייצר ממוצעים שונים בהתאם לפרמטרים מסוימים, למשל אם ממוצע הגילאים של הגברים הוא 70 וממוצע הגילאים של הנשים הוא 40 – יהיה מדויק יותר עבור התאים הריקים להשלים את הגיל בהתאם לממוצע לפי אותו מין. יותר מעמיק למשל מציאת ממוצע הגילאים לפי מין ולפי איזו מחלקה אותו נוסע היה, כמו שאנו רואים בתמונה. כך שמה שניתן לעשות הוא להגדיר עמודה חדשה עם transformה שלמדנו כך שלכל נוסע יופיע ממוצע הגיל לפי המין והמחלקה, ואז לבצע השמה ע"י fillna של age לתאים הריקים לפי העמודה הזו.

```
titanic.groupby(["sex", "pclass"]).age.mean()
```

```
sex     pclass
female  1         34.611765
        2         28.722973
        3         21.750000
male    1         41.281386
        2         30.740707
        3         26.507589
Name: age, dtype: float64
```

```
titanic["group_mean_age"] = titanic.groupby(["sex", "pclass"]).age.transform("mean")
```

```
titanic.head(6)
```

| | survived | pclass | sex | age | sibsp | parch | fare | embarked | deck | probability_survived | group_mean_age |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 3 | male | 22.0 | 1 | 0 | 7.2500 | S | NaN | 0.135447 | 26.507589 |
| **1** | 1 | 1 | female | 38.0 | 1 | 0 | 71.2833 | C | C | 0.968085 | 34.611765 |
| **2** | 1 | 3 | female | 26.0 | 0 | 0 | 7.9250 | S | NaN | 0.500000 | 21.750000 |
| **3** | 1 | 1 | female | 35.0 | 1 | 0 | 53.1000 | S | C | 0.968085 | 34.611765 |
| **4** | 0 | 3 | male | 35.0 | 0 | 0 | 8.0500 | S | NaN | 0.135447 | 26.507589 |
| **5** | 0 | 3 | male | NaN | 0 | 0 | 8.4583 | Q | NaN | 0.135447 | 26.507589 |

```
titanic.age.fillna(titanic.group_mean_age, inplace = True)
```

6. אנחנו לא צריכים להיות מקובעים לפעולות הבסיסיות של groupby, אנו יכולים לבנות פונקציות בעצמנו עם הסינונים שאנו רוצים ולהשתמש בapply לצורך זה. למשל פונקציה המחשבת את שלושת האנשים המבוגרים ביותר ששרדו. כמובן שניתן להכניס עוד פרמטרים בgroupby ובכך לקבל נתונים מפורטים יותר או לסבך את הפונקציה יותר ע"י if וכד'.

```
def five_oldest_surv(group):
    return group[group.survived == 1].nlargest(3, "age")
```

```
titanic.groupby("sex").apply(five_oldest_surv)
```

| | | survived | pclass | sex | age | fare |
|---|---|---|---|---|---|---|
| **sex** | | | | | | |
| **female** | **275** | 1 | 1 | female | 63.0 | 77.9583 |
| | **483** | 1 | 3 | female | 63.0 | 9.5875 |
| | **829** | 1 | 1 | female | 62.0 | 80.0000 |
| **male** | **630** | 1 | 1 | male | 80.0 | 30.0000 |
| | **570** | 1 | 2 | male | 62.0 | 10.5000 |
| | **587** | 1 | 1 | male | 60.0 | 79.2000 |

7. לפעמים אם נציג את הinfo של הדאטה נראה נתונים לא נוחים, למשל שהגיל הוא אובייקט (טקסט למשל, ואז לא ניתן לבצע עליו פעולות חשבון).
כמו כן, אנו מכירים את פעולת describe המציגה את המקסימום/ מינימום/ סטיית תקן וכו' עבור כל עמודה, אך ניתן להרחיב את הפעולה, למשל ע"י הוספת הפיצ'ר "o" = include (סימן לobject) ולמשל נוכל לראות שבsurvived מופיע unique=4, דבר שלא הגיוני כי אדם יכול או לשרוד או שלא, אין

| | Class | SipSp | ParCh |
|---|---|---|---|
| **count** | 894.000000 | 894.000000 | 894.000000 |
| **mean** | 2.309843 | 0.522371 | 0.381432 |
| **std** | 0.835370 | 1.101283 | 0.805171 |
| **min** | 1.000000 | 0.000000 | 0.000000 |
| **25%** | 2.000000 | 0.000000 | 0.000000 |
| **50%** | 3.000000 | 0.000000 | 0.000000 |
| **75%** | 3.000000 | 1.000000 | 0.000000 |
| **max** | 3.000000 | 8.000000 | 6.000000 |

```
titanic.describe(include ="O")
```

| | Survived | Gender | Age | Fare | Emb | Deck |
|---|---|---|---|---|---|---|
| **count** | 894 | 894 | 758 | 894 | 892 | 203 |
| **unique** | 4 | 2 | 92 | 248 | 3 | 7 |
| **top** | 0 | male | Missing Data | $8.05 | S | C |
| **freq** | 551 | 580 | 41 | 43 | 647 | 59 |

עוד אופציות. או למשל בfare – מחיר הכרטיס, מופיע גם מחיר ללא סימן מטבע, וגם $8.05.

נתבונן אם כן ערכים השונים של survived, ונראה כי מילאו גם yes וno, נוכל לשנות זאת ע"י פעולת replace עם הפיצ'ר של החלפה מרובית (על אותו עקרון של מילון, הראשון בto_replace מוחלף בראשון בvalue וכך הלאה, וחשוב להוסיף inplace כדי להגדיר שהערכים האלו באו להחליף את הערכים הקיימים ולעדכן את הטבלה. (inplace=false זה ברירת המחדל כך שאם לא נרשום true רק יוצג לנו איך זה נראה אחרי הפעולה שכתבנו אבל לא תעדכן את הטבלה לפי ההחלפה.

```
titanic.Survived.unique()
```

```
array(['0', '1', 'yes', 'no'], dtype=object)
```

```
titanic.Survived.value_counts()
```

```
0      551
1      341
no       1
yes      1
Name: Survived, dtype: int64
```

```
titanic.Survived.replace(to_replace= ["yes", "no"], value = [1, 0], inplace = True)
```

8. דגש חשוב – לאחר שהמרנו ערך המוצג כאובייקט שאמור להיות מספרי, לאחר שביצענו החלפה של הנתונים למשל כמו שעשינו כעת (מyes ל1 וכד') הערכים עדיין מוגדרים כאובייקט ויש לשנות אותם לערך מספרי ע"י פעולת astype.

```
titanic.Survived = titanic.Survived.astype('int64')
```

```
titanic.Survived.value_counts()
```

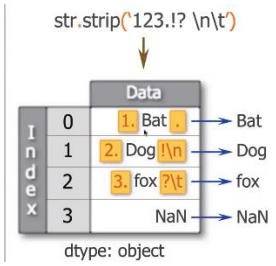
```
0    552
1    342
Name: Survived, dtype: int64
```

9. בעיות נוספות שאנו עלולים להיתקל בהן:

שם של עמודה מכיל רווח לפני השם ("name ") ואז נקבל שגיאה כשנקרא לעמודה הזו.

שמות הכתובים באותיות גדולות וכד' – ניתן להמיר לצורה של אות ראשונה גדולה והיתר קטנה ע"י בחירת העמודה ולאחר מכן ( )str.title. ועוד צורות רבות ע"י פעולה .str.

טקסט המכיל מספרים וסימנים או רווחים לפני ואחרי שלא רלוונטים – ניתן להשתמש בפקודת strip.




כמובן שיש עוד המון אך הבאנו את זה רק כדי להראות כמה פרטים קטנים יכולים להיות ולהקשות לנו על קריאת הנתונים באופן חלק. בגדול חשוב להבין שכמעט כל דאטה ניתן לנקות בעזרת פונקציות.

10. אנו יכולים לבדוק את כמות התאים הריקים בכל עמודה ע"י isna , דגש חשוב – אנו יודעים כי axis=0 הכוונה לשורות ו1 יחזיר לנו עמודות, כאשר אנו סוכמים את כמות הnull בעמודה נסמן בכל זאת axis=0 כי זה סוכם לי עבור העמודה לאחר שעובר שורה שורה – זה עניין מבלבל אבל מהותי.

```
titanic.isna().sum(axis = 0)
```

```
Survived      0
Class         0
Gender        0
Age         136
SipSp         0
ParCh         0
Fare          0
Emb           2
Deck        691
```

```
titanic[titanic.isna().any(axis = 1)]
```

| | Survived | Class | Gender | Age | SipSp | ParCh | Fare | Emb | Deck |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 3 | male | 22.0 | 1 | 0 | 7.2500 | S | NaN |
| 2 | 1 | 3 | female | 26.0 | 0 | 0 | 7.9250 | S | NaN |
| 4 | 0 | 3 | male | 35.0 | 0 | 0 | 8.0500 | S | NaN |
| 5 | 0 | 3 | male | Missing Data | 0 | 0 | 8.4583 | Q | NaN |

או למשל תציג לי את כל השורות (ולכן הפוך על הפוך – נרשום axis=1) המכילות בתוכן ערך של null. באותו אופן ניתן להגדיר notna ולראות מה שלא מכיל שום ערך null.

11. צורה נוספת בה ניתן להיפתר ערכים המכילים null זה ע"י האופן הבא:

```
titanic.dropna(axis = 1, how = "any").shape
```

```
(894, 6)
```

```
titanic.dropna(axis = 0, how = "all").shape
```

```
(894, 9)
```

```
titanic.dropna(axis = 1, how = "all").shape
```

```
(894, 9)
```

```
titanic.dropna(axis = 0, thresh = 8).shape
```

```
(736, 9)
```

בסינון הראשון אנו מורידים את כל העמודות המכילות איזשהו (any) ערך null, נראה כי ירדו לנו 3 עמודות. אם בhow נשנה לall נשים לב כי הדאטה לא השתנה כלל, מאחר ואין עמודה/שורה בה כל הערכים הם null, בנוסף ניתן לקבוע רף (thresh) ממנו יוריד, למשל – תמחק את כל השורות המכילות לפחות 8 ערכים שהם לא null, נראה כי יש 160 שורות שלא מכילות יותר מאשר 8 ערכים.

12. ניתן גם לסנן ערכים כפולים במקומות שאנו לא רוצים כפילויות, למשל בתעודת זהות, ולבחור באיזה מבין האופציות אנו מעוניינים לשמור (ע"י הפיצ'ר keep, ניתן היה

```
alphabet[alphabet.duplicated(keep = "first")]
```

גם לשנות לlast או לfalse ואז הוא יזרוק את כל השורות שמופיע בהן כפילויות, דרך אחרת – ע"י drop_duplicated במקום duplicated). כמו כן ניתן להוסיף בסוגריים את הפיצ'ר subset ולבחור כמה עמודות שאני מעוניין לסכום לפיהן לבדוק שאין כפילויות, למשל כמה כפילויות יש באנשים שהם גם גבר, וגם עם אותו שם.

13. ניתן לזרוק נתונים ע"י האינדקס הספציפי שלהם

```
titanic.drop(index = [891, 892, 893], inplace = True)
```

14. להוסיף את ההסבר על cut – חלוקה לפי הגדרות שאני מכניס לו – למשל בקפיצות של 100 וכו', וqcut – חלוקה למספר חלקים שווים כך שבכל חלק תהיה כמות שווה של אובייקטים_____

# שיעור 4

## Time series

1. חלק מאוד גדול מדאטה שנתעסק איתו יהיה כזה שיש משמעות לסדר השורות, ולכן יש חשיבות לסדרת זמן. כאשר יש לנו דאטה מסוג time series המידע בשורה אחת הוא קשור באופן מהותי לשורה שלפניו ולזו שאחריו, לכן כאשר נבנה מודל אסור יהיה לנו לעשות shuffle.
אני מבין שהיכולת שלי לחזור את מה שיקרה עוד רגע תלוי במה שקרה כמה שניות קודם לכן, והטמפרטורה מחר תלויה בזו שהייתה היום ואתמול ושלשום, ולכן אני יכול לקחת כמות מידע אחורה, אבל לא יותר מידי כי לפעמים דווקא בגלל עודף מידע משבש מהיכולת האמינה לחזות.

## Parse

2. פורמט התאריך הוא בעייתי כי יש צורות רבות לכתוב את אותו דבר – 29/3/22, 29/03/22, 03/29/2022 וכו'. לכן כאשר מייבאים את הקובץ אפשר להוסיף את הפיצ'ר parse_dates ולהגדיר

```
temp = pd.read_csv("temp.csv", parse_dates=["datetime"], index_col= "datetime")
```

לו איך למנות:

(עדיין חשוב לעבור על הדאטה לוודא שלא היו חריגות בהמרה, למשל אם מישהו מילא בטבלה "חסר" או "אין תאריך")

אופציה אחרת כאשר יש לי למשל עמודה עם סטרינגים של תאריך ניתן לבצע המרה כך שהעמודה תהיה time series. היעילות הרבה כל כך של ההמרה הזו היא שלא משנה באיזו דרך התאריך ייכתב (בתחום גבולות הגזרה ההגיוניים) פנד'ס יידע להמיר את זה לפורמט תאריך. למשל: 11/8/2015, 10:30:20, 20150520, 2015/05/20, 05 2015 20,

```
temp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35064 entries, 0 to 35063
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   datetime  35064 non-null  object
 1   LA        35062 non-null  float64
 2   NY        35064 non-null  float64
dtypes: float64(2), object(1)
memory usage: 821.9+ KB
```

```
temp.datetime[0]
print(pd.__version__)
```

```
1.2.4
```

```
pd.to_datetime(temp.datetime)
```

```
0       2013-01-01 00:00:00
1       2013-01-01 01:00:00
2       2013-01-01 02:00:00
```

```
pd.to_datetime(["2015-05-20", "Feb 20 2015", "Burik"], errors="coerce")
```

```
DatetimeIndex(['2015-05-20', '2015-02-20', 'NaT'], dtype='datetime64[ns]', freq=None)
```

May 2015 20,

"may 20th 2015", ועוד.

אם נכניס קלט לא תקין הוא לא ייתן לנו להריץ אך ניתן להתגבר על זה ע"י "errors="coerce.

אצלנו הפיצ'ר הזה נועד למצבים בהם מישהו הקליד למשל "אין תאריך" וכו'.

במקום הקלט הלא חוקי יחזיר NaT-not a time, כמו שעבור קלט שלא מספר יחזיר NaN.

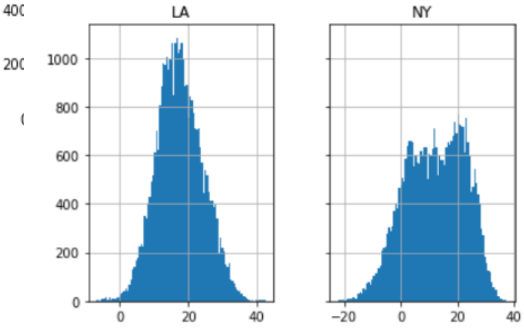
3. מה אפשר לעשות עם time series? נסתכל על דאטה סט המפרט את הטמפרטורה שנמדדה בכל יום במשך כמה שנים בלוס אנג'לס (LA) ובניו יורק (NY).

```
temp.hist(bins=100)
```

```
array([[<AxesSubplot:title={'center':'LA'}>,
        <AxesSubplot:title={'center':'NY'}>]], dtype=object)
```


```
temp.hist(bins=100, sharey=True)
```

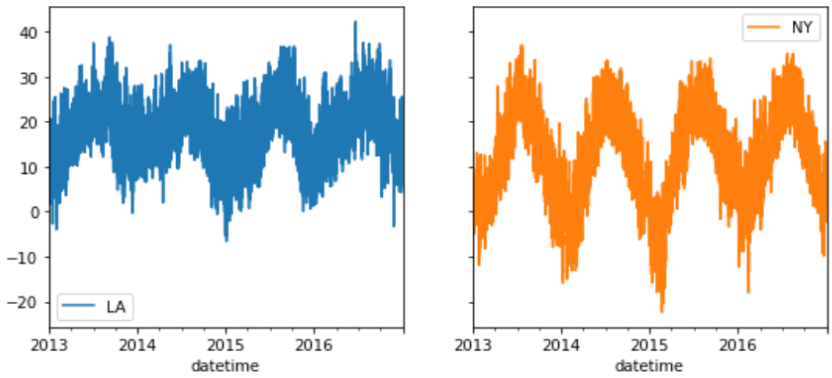
```
array([[<AxesSubplot:title={'center':'LA'}>,
        <AxesSubplot:title={'center':'NY'}>]], dtype=object)
```




למשל אנו יכולים להציג היסטוגרמה של הטמפרטורה (ציר x) שנממדה לפני כמות הימים/המופעים של הטמפרטורה הזו (ציר y). (bins זה כמות החלקים שאני מעוניין לחלק את ערכי הנתונים) אבל מה הבעיה שלנו בגרף שיצא לנו? היתרון המשמעותי כל כך של גרף ושל ויזואליזציה של הנתונים זה שהוויזואליזציה צריכה לזרוק עלינו את המידע ישר, בלי שנצטרך לנתח ולהתעמק בגרף. במבט מהיר על 2 הגרפים שקיבלנו על כמות הימים לפי הטמפרטורות שנמדדו בלוס אנג'לס ובניו יורק – <u>לא ניתן להבחין באופן מהיר שניו יורק יותר קרה מלוס אנג'לס</u>, רק כשנתעמק נראה שיש בניו יורק מדידות מתחת ל0, אבל במבט חטוף זה לא "צועק" את זה, לכן נבחר גרף אחר.

(הערה – ניתן להוסיף את הפיצ'ר Share y או share x אם אני רוצה ש2 הגרפים האלו למשל יסונכרנו בטווח הxים או הyים שלהם, ובכך לאפשר לאמוד את הנתונים טוב יותר, למשל עבור שיתוף ציר הy נקבל את הגרף הבא:)

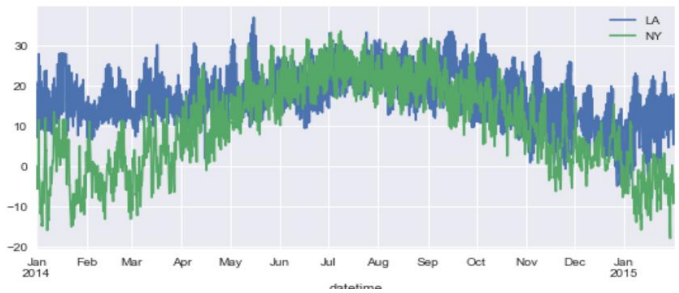
```
temp.plot(figsize = (9, 4), subplots=True, layout=(1, 2) , sharey=True)
plt.show()
```




4. נוכל למשך לבחור בצורת גרפים של plot ולקבל מידע שהוא קצת יותר אינדיקטיבי, למשל אפשר לראות שניו יורק מגיע לערכי y נמוכים יותר בגרף. נשים לב שמאחר ולקחנו את כל אלפי הערכים החלוקה שלנו בציר הx הייתה לשנים, time series יודע לחלק את ציר הx לפי הערכים שמקבל, זה עוד יתרון שלו. בנוסף,

```
temp.loc["01 2014":"01 2015"].plot(figsize = (9, 4), subplots=False, layout=(1, 2))
```

```
<AxesSubplot:xlabel='datetime'>
```



היינו יכולים לשנות את subplots לfalse ובמקום 2 גרפים נפרדים לקבל אותם אחד על השני ואז הפער בין 2 הערים הללו יהיה מובחן יותר.

אבל עדיין הגרף הזה נותן לנו מידע כללי, שבגדול - בקיץ חם ובחורף קר, אבל זה לא בהכרח יאפשר לי לחזות מה תהיה הטמפרטורה מחר/ שבוע הבא, לכן מה שאפשר לעשות זה לחתוך את המחרוזת ע"י loc לתאריכים. ניתן לעשות את הסלייסינג לכל תארי ושעה שנרצה.

## Data range

5. שימוש ראשון לDate_range מאפשר לייצר רשימה של תאריכים בקפיצות קבועות, עם התחלה וסוף, קצב הקפיצות (frequency) freq - שיכול לקבל כמות שונה מאוד של פרמטרים, למשל: יום – D, שבוע – W, חודש – M, שנה – Y, או אפילו שעה – H, או כמות מקטעים מוגדרת, או

```
pd.date_range(start = "2015-07-01", end = "2015-07-31", freq= "D")
```

```
DatetimeIndex(['2015-07-01', '2015-07-02', '2015-07-03', '2015-07-04',
               '2015-07-05', '2015-07-06', '2015-07-07', '2015-07-08',
               '2015-07-09', '2015-07-10', '2015-07-11', '2015-07-12',
               '2015-07-13', '2015-07-14', '2015-07-15', '2015-07-16',
               '2015-07-17', '2015-07-18', '2015-07-19', '2015-07-20',
               '2015-07-21', '2015-07-22', '2015-07-23', '2015-07-24',
               '2015-07-25', '2015-07-26', '2015-07-27', '2015-07-28',
               '2015-07-29', '2015-07-30', '2015-07-31'],
              dtype='datetime64[ns]', freq='D')
```

ללכת אחורנית ע"י הכנסה לסטארט את התאריך המאוחר יותר, או b-buisnessdays (שני עד שישי) או קפיצות בשבוע החל מיום ספציפי – למשל כל יום רביעי w-wed, או תחילת חודש MS, באותו אופן אפשר להציג רבעון – Q, או תחילת רבעון – QS, החוקיות היא אותה חוקיות – ברירת המחדל של כל זמן הוא סוף הזמן, ואפשר להוסיף S ולקבל את ההתחלה, כמו כן אפשר לבחור פרמטר מדידה ולהוסיף לו את התת זמן שעל פיו אני רוצה לבצע את המדידה כמו הדוגמה עם יום רביעי כל שבוע.

אופציה אחרת היא להגדיר במקום תאריך end את הפיצ'ר periods ואז להגדיר כמות, למשל החל מתאריך מסוים, 31 זמנים בהפרשים של 3 שעות.

```
pd.date_range(start = "2015-02-01", periods = 31, freq = "3H")
```

```
DatetimeIndex(['2015-02-01 00:00:00', '2015-02-01 03:00:00',
               '2015-02-01 06:00:00', '2015-02-01 09:00:00',
               '2015-02-01 12:00:00', '2015-02-01 15:00:00',
               '2015-02-01 18:00:00', '2015-02-01 21:00:00',
               '2015-02-02 00:00:00', '2015-02-02 03:00:00',
               '2015-02-02 06:00:00', '2015-02-02 09:00:00',
               '2015-02-02 12:00:00', '2015-02-02 15:00:00',
               '2015-02-02 18:00:00', '2015-02-02 21:00:00',
               '2015-02-03 00:00:00', '2015-02-03 03:00:00',
```

## Resample

6. פעולת resemble מאפשרת לבצע בחירה מסויימת כאשר יש לנו מספר שורות עבור ערכים דומים, למשל אם עבור מדידת הטמפרטורות יש לי 24 מדידות לכל יום, ואני רוצה לבחור את הערך

```
temp.resample("W").mean().plot(figsize = (10, 5), subplots=False, layout=(1, 2) , sharey=True)
```

```
<AxesSubplot:xlabel='datetime'>
```

המקסימלי, או עבור כל שעתיים תיקח את הראשון/האחרון/הממוצע וכד'. או אפילו יותר מגניב – תדגום מחדש לפי הממוצע בכל שבוע לתוך גרף

**אמינות הגרף**

7. יש בדיחה שאומרת שיש 3 סוגים של שקר – שקר, שקר וכזב, וסטטיסטיקה. כי באמת עם נתונים וגרפים ניתן לספר איזה סיפור שרוצים, וחשוב לוודא שהגרף מבטא את האמת.


ניקח לדוגמא גרף של מניות, שבמבט חטוף אפשר לראות שמחיר המניה של חברת המטוסים בוינג עלה באופן משמעותי ביחס לכל החברות האחרות, ולמשל באפל או IBM, אין מה להשקיע כי הן בקושי עלו. אבל האמת היא שהחשוב במניה זה האחוזים שהיא עלתה ,ואנשים קונים מניה לא לפי המחיר שלה אלא לפי האחוזים. אז מה אפשר לעשות? אני רוצה לנרמל את הנתונים לפי שער פתיחה מסוים ולפיו לראות את העליות/ירידות באחוזים. ניקח לדוגמא את המניה של אפל, ונחלק אותה בערך המניה ביום הראשון, ואז אני אוכל לקבל את השינוי באחוזים של מחיר הסגירה של המניה בכל יום מיום הפתיחה. ניתן לראות שהחל מ31.12.09 המניה עלתה ב578%!

```
close.AAPL.div(close.iloc[0,0])#.mul(100)
```

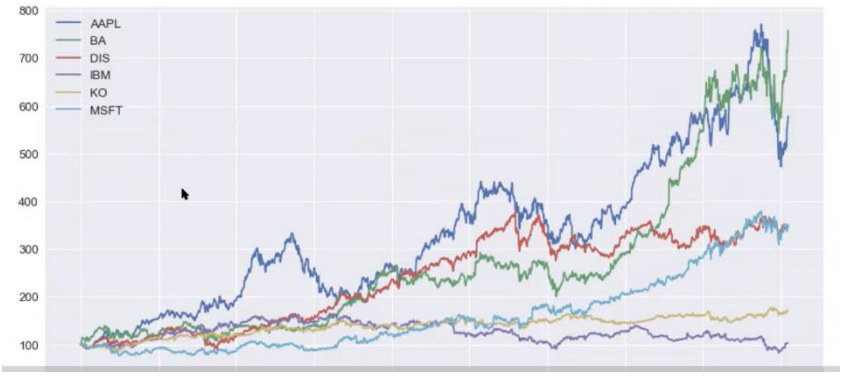
```
Date
2009-12-31    1.000000
2010-01-04    1.015565
2010-01-05    1.017321
2010-01-06    1.001139
2010-01-07    0.999288
                 ...
2019-01-30    5.489252
2019-01-31    5.528781
2019-02-01    5.531439
2019-02-04    5.688559
2019-02-05    5.785887
```

```
norm = close.div(close.iloc[0]).mul(100)
norm
```

| Date | AAPL | BA | DIS | IBM | KO |
|---|---|---|---|---|---|
| 2009-12-31 | 100.000000 | 100.000000 | 100.000000 | 100.000000 | 100.000000 |
| 2010-01-04 | 101.556498 | 103.787178 | 99.441860 | 101.184112 | 100.070177 |
| 2010-01-05 | 101.732082 | 107.186402 | 99.193798 | 99.961812 | 98.859646 |
| 2010-01-06 | 100.113899 | 110.437830 | 98.666666 | 99.312457 | 98.824565 |

באופו אופן אני יכול לעשות את זה על כל המניות (iloc[0] יחזיר לי רשימה של כל מניה עם הערך הראשון שלה, ואז כשנבצע חלוקה של הדאטה סט כולו בiloc הוא יחלק כל שורה בערך המקביל אליו).

כעת אם נציג את הכל בגרף אנחנו נקבל מצג שונה באופן מהותי ממה שחשבנו – וזה שמניית אפל השיגה לאורך הדרך הרבה יותר עליות מאשר מניית בוינג.



# שיעור 5

## המשך time serirs

### Shift

1. נמשיך להתעסק עם הדאטה של המניות, ונראה פונקציה שתעזור לנו מאוד – shift. ניקח רק את הנתונים של מניית apple לדאטה סט חדש, ברגע שנפעיל את פקודת shift עם period=1 אנחנו למעשה מזיזים לכל שורה את הערך של השורה אחת מעליה, כעת ניתן לראות את השינוי היומי מיום אחד לשני, ויהיה קל גם לחשב כמה היא עלתה בערך וכמה גם באחוזים.

```
aapl = close.AAPL.copy().to_frame()
```

```
aapl.head()
```

| | AAPL |
|---|---|
| Date | |
| 2018-12-31 | 39.435001 |
| 2019-01-02 | 39.480000 |
| 2019-01-03 | 35.547501 |
| 2019-01-04 | 37.064999 |
| 2019-01-07 | 36.982498 |

```
aapl["lag1"] = aapl.shift(periods = 1)
```

```
aapl.head(10)
```

| | AAPL | lag1 |
|---|---|---|
| Date | | |
| 2018-12-31 | 39.435001 | NaN |
| 2019-01-02 | 39.480000 | 39.435001 |
| 2019-01-03 | 35.547501 | 39.480000 |
| 2019-01-04 | 37.064999 | 35.547501 |
| 2019-01-07 | 36.982498 | 37.064999 |

נוסיף את לעמודה חדשה את הערך עם פקודת הshift. אילו היינו רושמים period=7, זה היה נותן את השינוי השבועי.

```
aapl["Diff"] = aapl.AAPL.sub(aapl.lag1)
```

```
aapl.head()
```

| | AAPL | lag1 | Diff |
|---|---|---|---|
| Date | | | |
| 2018-12-31 | 39.435001 | NaN | NaN |
| 2019-01-02 | 39.480000 | 39.435001 | 0.044998 |
| 2019-01-03 | 35.547501 | 39.480000 | -3.932499 |
| 2019-01-04 | 37.064999 | 35.547501 | 1.517498 |

נוסיף לעמודה חדשה בשם diff את ההשתנות מאתמול ע"י חיסור sub של מחיר המניה באותו יום את מחיר המניה בעמודת leg1. (למשל 39.48-39.435.

```
aapl.AAPL.div(aapl.lag1).sub(1).mul(100)
```

```
Date
2018-12-31          NaN
2019-01-02     0.114107
2019-01-03    -9.960737
2019-01-04     4.268930
2019-01-07    -0.222583
```

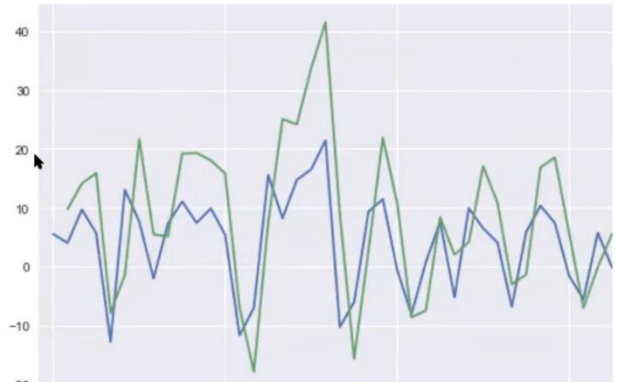
ואם נרצה את השינוי באחוזים נחלק את מחיר המניה במחיר של אתמול, ואז נוריד 1 ונכפיל ב100. הסיבה שמורידים 1 זה כי אם מחיר מניה עלה מ100 דולר ל200 דולר תוצאת החילוק תיתן 2 – גדל פי 2, אבל באחוזים – העלייה הייתה של 100 אחוז.

הערה – ישנה אפשרות אפילו לקצר את התהליך ולקבל את השינוי באחוזים לפי כמות שורות שנגדיר והיא ע"י הפיצ'ר pct_change במקום הshift שכתבנו, ואז להגדיר period כפי רצוננו.

2. נוכל לקחת למשל את כל ימי שני כל חודש (BM- business Monday) ועי"י פקודת resample שלמדנו לקחת את יום שני האחרון מביניהם, ולהציג את השינוי באחוזים בין כל חודש אחד (מסומן בכחול) וכן את השינוי כל 2 חודשים כאלו (ירוק). נשים לב שהגרף של 2 ימים הוא חד יותר מכיוון שאם היו שני חודשים רצופים שהסתיימו בעלייה – הקפיצה של חודשיים תהיה על חדה יותר.

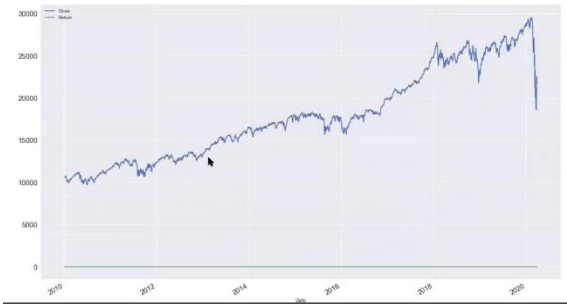
```
aapl.AAPL.resample("BM").last().pct_change(periods =1).mul(100).plot()
aapl.AAPL.resample("BM").last().pct_change(periods = 2).mul(100).plot()
```

```
<AxesSubplot:xlabel='Date'>
```

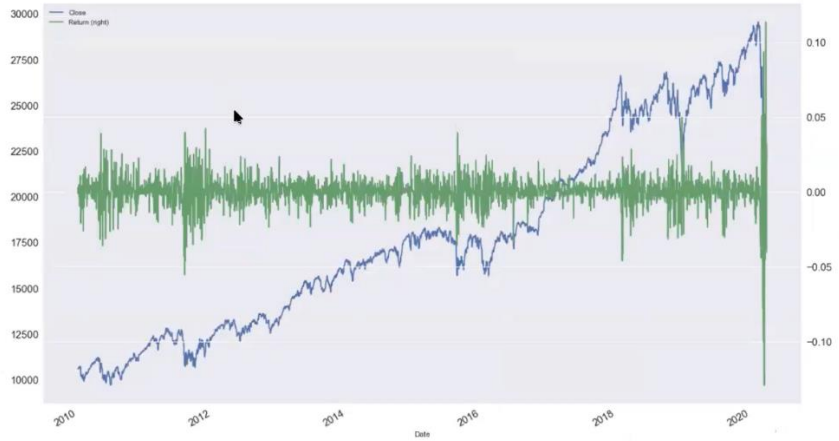



3. כלי נוסף שניתן להשתמש בו הוא הפיצ'ר secondary_y ניקח לדוגמא גרף שיש בו 2 עמודות, כך שבראשונה המספרים נעים במספרים קטנים, למשל סביב ה0, לעומת זאת בעמודה השניה ערכים גדולים מאוד למשל מחיר ממוצע לדירה, או מניה מאוד יקרה – ולכן אם נציג את 2 העמודות על גרף נראה גרף אחד עם עליות וירידות וגרף אחד ישר, כי הוא זניח לעומת השינויים של הגרף בעל המספרים הגדולים, בעוד שבפועל יכול להיות שהגרף הקטן מתנהג אחרת. לכן נשתמש ב secondary_y וכעת ישנו ציר y נוסף וניתן לראות את ההבדל בתנודיות והאם ישנן עליות ירידות או מגמות.

```
df.plot(figsize = (20, 12), mark_right = True, fontsize = 15)
plt.show()
```




```
df.plot(figsize = (20, 12), secondary_y = "Return", mark_right = True, fontsize = 15)
plt.show()
```




**<u>אסטרטגיות במניות</u>**

1. 2 הדרכים הפשוטות ביותר להחלטה על אסטרטגיית ההשקעה במניות הן: buy and hold, כלומר לקנות ולא למכור בכלל, ו"לספוג" את הירידות מתוך הבנה שהשוק סטטיסטית נמצא תמיד בעלייה, וכל מה שירד יעלה יותר. והשיטה השנייה היא simple momentum, כלומר אם אתמול הסתיים ברווח – קנה, אם אתמול הסתיים בהפסד – מכור. נרצה לכתוב תוכנית שבודקת מה

מבין האסטרטגיות האלו עדיפה.

נייצר טבלה עם אחוזי העלייה/ירידה כמו שראינו לפני כן, ואז נוסיף עמודה הקובעת את הסימן של האחוזים – 1 אם עלייה, ו1- אם ירידה.

```
df["Position"] = np.sign(df["DJI_Return"])
```

```
df["Strategy_Ret"] = df["Position"].shift() * df["DJI_Return"]
```

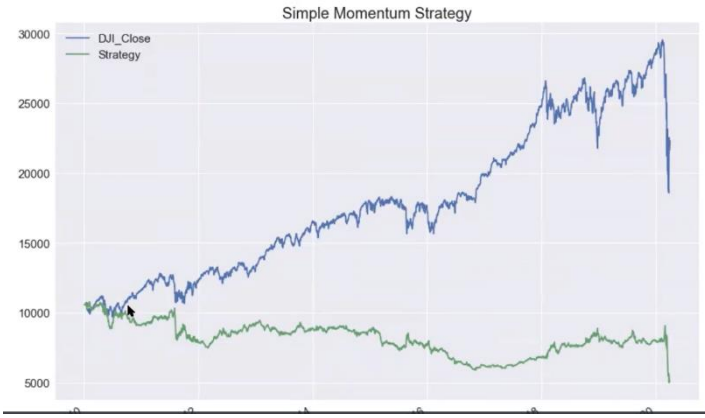
```
df
```

| | DJI_Close | DJI_Return | Position | Strategy_Ret |
|---|---|---|---|---|
| **Date** | | | | |
| **2010-01-05** | 10572.019531 | -0.001128 | -1.0 | NaN |
| **2010-01-06** | 10573.679688 | 0.000157 | 1.0 | -0.000157 |
| **2010-01-07** | 10606.860352 | 0.003138 | 1.0 | 0.003138 |
| **2010-01-08** | 10618.190430 | 0.001068 | 1.0 | 0.001068 |
| **2010-01-11** | 10663.990234 | 0.004313 | 1.0 | 0.004313 |

כעת אנחנו יכולים להגדיר עמודה חדשה שתחשב את עמודת הקניה או מכירה (+/-) כפול השינוי שהתרחש היום.

בסוף נקבל גרפים שמעידים יותר מכל דבר שהאסטרטגיה של המומנטום פשוט לא עובדת.

ומבין 2 האופציות נכון יותר לקנות ופשוט להחזיק כי באופן סטטיסטי הגרף בעלייה.




## Moving average

**4.** פעולת rollingה למעשה מאפשרת לתכלל מספר פעולות shift לשורה אחת, לצורך העניין בתמונה – לגלגל על 50 השורות שלפני כל שורה ולסכום אליו לפי פעולת חישוב מסויימת - הממוצע/ מקס'/ מיני' וכד' לבחירתנו. למשל בתמונה – כל שורה מקבלת אליה את הממוצע של 50 השורות שלפניו. זה נקרא ממוצא מתגלגל.

למה זה עוזר לנו? זה אמנם משבש לנו את הערכים האמיתיים של אותה שורה אבל זה מאפשר לנו לעשות **החלקה** של הגרף, מאחר וכאשר יש הרבה נתונים הגרף עולה ויורד בתנועות חדות כי בשורה מסויימת הערך

```
data
```

```
df = data.loc["2010":, "Close"].to_frame()
df["DJI_Return"] = df.Close.pct_change()
df.columns = ["DJI_Close", "DJI_Return"]
df.dropna(inplace = True)
df
```

```
df.DJI_Close.rolling(window = 50).mean()
```

```
df["SMA50"] = df.DJI_Close.rolling(window = 50).mean()
```

```
df
```

| | DJI_Close | DJI_Return | SMA50 |
|---|---|---|---|
| **Date** | | | |
| **2010-01-05** | 10572.019531 | -0.001128 | NaN |
| **2010-01-06** | 10573.679688 | 0.000157 | NaN |
| **2010-01-07** | 10606.860352 | 0.003138 | NaN |
| **2010-01-08** | 10618.190430 | 0.001068 | NaN |
| **2010-01-11** | 10663.990234 | 0.004313 | NaN |
| **...** | ... | ... | ... |
| **2020-03-24** | 20704.910156 | 0.113650 | 26708.547930 |
| **2020-03-25** | 21200.550781 | 0.023938 | 26554.417930 |
| **2020-03-26** | 22552.169922 | 0.063754 | 26426.667930 |

גבוה משמעותית מהממוצע, הוספה של איבר גדול מול הוצאה של איבר קטן (מסוף התור של ה50) יגדיל את הממוצע וכך "יחקה" את התנהגות הגרף.

```
df[["DJI_Close", "SMA50"]].plot(figsize = (15, 10), fontsize = 15)
plt.legend(fontsize = 15)
plt.show()
```


לקיחת הממוצע של 50 שורות מאפשר לגרף להיות רציף (ולינארי) יותר ככל שאנחנו שניקח ערך יותר גדול (כמובן שלא ניקח ערך מידי גדול כי אז הגרף לא יהיה מיצג אמיתי לגרף העובדות, כי הוא יהיה הרבה יותר ישר ללא הפיתולים).

## שיעור 6

### Naïve bayes classifier

1. למדנו בעבר על חוק בייס המדבר על הסתברות מותנית – מה ההסתברות להתרחשות אירוע מסוים, עפ"י ההסתברות של העובדות (E) מול התאוריה (H): $P(H,E) = \frac{p(E/H)\cdot P(H)}{P(E)}$

   P(E/H) – ההסתברות למה שאנו רוצים לקבל (תאוריה) בהנחה שעובדה מסויימת קרתה.

   P(H) – ההסתברות שמקרה התאוריה אכן התרחש.

   P(E) – סיגמא של כל ההסתברויות של כלל העובדות הקיימות.

   חוק בייס הנאיבי קובע לגבי מקרים בהם יש מספר עובדות ומחשב אותם כאילו הם לא קשורים ולא תלויים אחד בשני (למשל להניח שמחיר הבית נגזר מכמות החדרים, שטח הדירה, מספר חדרי שירותים וכו' אבל שכל אחד מהגורמים האלו הוא עובדה כשלעצמו ואין תלות אחד בשני למרות שברור שאם יש יותר חדרים אז השטח יותר גדול למשל, לכן אנו קוראים לו נאיבי). במצב זה בו אין תלות בין 2 גורמים הנוסחה תהיה כפולה של התאוריה בהנחה שעובדה i התרחשה: $P(H/E1,\ E2) = P(H/E1)\cdot P(H/H2)$.

   הדבר המפתיע בnaive bayes זה שאמנם היינו מצפים שהתוצאה תהיה לא קשורה למציאות אבל בפועל התוצאה יוצאת פעמים רבות קרובה מאוד לאמת.

2. נאיב בייס עובד מעולה בדאטה קטגוריאלי בעיקר, מאחר ודאטה קטגוריאלי מאפשר לי לעשות מיצוע של הרבה ערכים – למשל האם היום קר או חם, גשום או יבש, סטוזה או וירג'יניקה (בסוגים של האירוסים) וכו'.

3. יישום המודל בפועל:

```
>>> from sklearn.datasets import load_iris
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.naive_bayes import GaussianNB
>>> X, y = load_iris(return_X_y=True)
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)
>>> gnb = GaussianNB()
>>> y_pred = gnb.fit(X_train, y_train).predict(X_test)
>>> print("Number of mislabeled points out of a total %d points : %d"
...       % (X_test.shape[0], (y_test != y_pred).sum()))
Number of mislabeled points out of a total 75 points : 4
```

**Ensemble learning – למידת מכלול (/קבוצה)**

4. אחד הסגנונות מודלים המוכרים והשימושיים זה random forest ועצי החלטה. המודל מכניס את נתון הבדיקה לעץ ושואל שאלות, למשל האם כדאי לשחק טניס – האם חם? אם כן לך ימינה ואם לא שמאלה, אם קר למשל הוא ישאל לאחר מכן – האם יש רוחות חזקות? ואם כן למשל ישאל האם גשום? ואז יידע לסווג למשל שלא מומלץ כי קר יש רוחות וגשום. בנוסף זה יכול להיות עם ערכים מספריים – האם גדול מ150 וכו'. אבל השאלה המרכזית היא איך אני בוחר איזה שאלה לבחור קודם? התשובה היא שככל שיש לפרמטר/עמודה מסויימת ספקטרום אפשרויות קטן יותר – כלומר שיש לו פחות אופציות לבחור ביניהם – גבר/אישה, חי/מת וכד' כך בבחירה של אחד הפיצ'רים אנחנו מצמצמים כמות משמעותית יותר מהשורות.
בעצי החלטה אחת התכונות המשמעויות זה עומק (depth), כלומר כמה פיצולים / שאלות מבצע, ככל שהוא שואל יותר שאלות המודל על פניו יותר מורכב ויורד לרזולוציות מפורטות יותר וכך גם לחיזוי מדויק יותר (למשל בהינתן רכב – איזה שנה הוא? נפח מנוע? כמה מושבים? ארץ ייצור? וכו' עד שיכול להגיע לרכב הכי 'דומה' שיש בtrain).
Random forest לא יורד לעומק של רזולוציות עם הרבה שאלות אלא בודק כמות גדולה מאוד של עצים עם עומק נמוך מאוד – 2 או 3, כך שבהינתן תוצאה של עץ אחד אי אפשר לדעת בוודאות מה החיזוי, אבל מאחר ויש הרבה עצים כאלו נוצר מצב של מעין "חוכמת המונים" וכך לפי ה"מסקנה" של רוב העצים הוא קובע מה החיזוי המדויק.
זה המהות של ensemble learning – חוכמת ההמונים, לקיחת המון החלטות קטנות ושילובן.

**שיעור 7**

**ensemble learning**

1. אחד האתגרים הגדולים של למידת מכונה זה להתמודד עם מצבים של unlabel, כלומר שאין לי את התווית של המידע, למשל עבור תמונה של כלב – לומר "זה כלב", או זן של פרח שמוגדר כ"סטוזה". אבל לפעמים לא מופיע רק כלב או רק פרח אלא הרבה מעבר לזה ולכן למה זה חשוב להצליח לתת לייבלים לדברים? כי אז ניתן לבנות מכונות חכמות מאוד לדוגמא מצלמת כביש שיודעת לזהות מה בן אדם ולהתריע, מה תמרור ולהציג אותו וכו' וצובעת אותם לעצמה בצבעים שונים – דבר זה נקרא image segmentation.

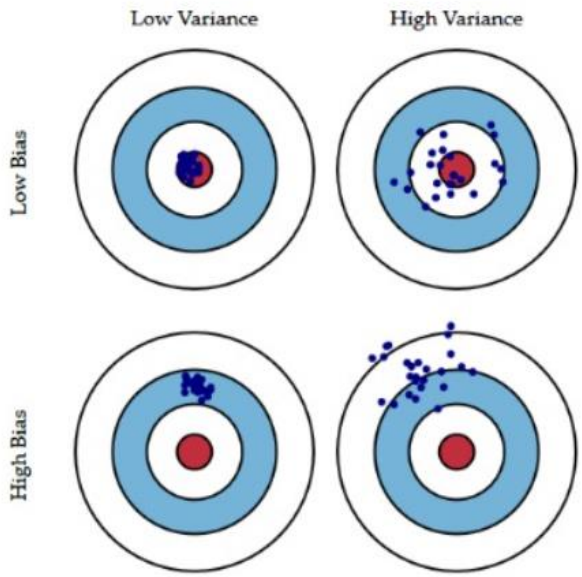
2. נלמד 2 מושגים חדשים – אינטרפולציה ואקסטרפולציה.
כאשר יש לי קטע מסוים עם מידע ידוע, למשל גרף של מחיר הדירה ביחס לכמות החדרים והמיקום. אינטרפולציה היא הבחינה של נקודה הנמצאת בתוך הקטע אבל לא הייתה קיימת לי לפני, ואקסטרפולציה משמעותה בחינה של נקודה הנמצאת מחוץ לטווח הידוע שלי, למשל אם כל הדירות בגרף מכילות רק 2 עד 5 חדרים, ופתאום אני נדרש לחזות/ לבחון על דירה בת 6 חדרים. כמובן שמבין שתי הפעולות הנ"ל אקסטרפולציה הינה פעולה מסובכת יותר כי יש

לפי פחות עם מה להתמודד או לבחון לעומתו. ניתן לחשוב שאם יש לי נקודה מחוץ לטווח אני יכול לבנו גרף לינארי ולחזות מה יהיה בהמשך, אבל זה לא תמיד נכון, למשל קפיץ – יש יחס לינארי שככל שאני מושך אותו יותר אחורה הוא יחזור במהירות גבוהה יותר, אבל יש נקודה שבה אם אני אמשוך מידי חזק הקפיץ ייהרס, לכן לא כל דבר שהולך בכיוון מסוים ימשיך כך תמיד. (לכן אומרים לפעמים שלמידת מכונה עוזרת לחזות טוב מאוד את <u>העבר</u>).

3. הכוח של אנסמבל זה במקום לנסות למצוא קשר בין כל הנקודות זה לקחת כל פעם מדגם קטן אל מול מה שאני מעוניין לחזות ולייצר המון מודלים קטנים שמתאימים לחלק מהדאטה, ואם נתמצת את ה"תורה" של אנסמבל זה – הרבה חוקים פשוטים עדיפים על חוק אחד מורכב וכללי.
4. דגש חשוב – כאשר נסביר מה היה תהליך בניית המודל שלנו, לומר "עשיתי אנסמבל" זה לא באמת לומר משהו, כי אנסמבל זו שיטה המתבססת על חיבור מודלים, צריך להדגיש <u>למה</u> עשיתי אנסמבל, זאת אומרת לאיזה מודלים, באיזה צורת חיבור קישרתי ביניהם.

## <u>Bias – נטייה</u>

5. בפשטות בייאס זה **<u>ממוצע השגיאה</u>**, בשונה למשל משונות, שמשמעותה זה עד כמה כל דוגמא רחוקה מהממוצע. כאשר אדם מרוויח בכל חודש 7,000 ₪ נוכל לומר שהשונות של השכר שלו נמוכה, ואם ירוויח בחודש אחד 84,000 ₪ ובכל יתר השנה לא יעבוד – הממוצע יהיה אותו ממוצע בדיוק, אבל השונות תהיה גדולה. בייאס גבוה משמעותו שהדעה/ אופן הבחינה מוטה / משוחד לכיוון מסוים ולכן לא מספק תשובת אמת. ניקח למשל לוח מטרה, כאשר השונות נמוכה וגם הבייאס נמוך – כל הפגיעות במרחק קטן מהמרכז וגם המרחק בין כל אותן פגיעות אחת מהשנייה קטן.




בעוד שבייאס גבוה ושונות נמוכה אומרת פגיעה באותו אזור במטרה מספר רב של פעמים, אבל רחוק מהמקום אליו כיוונתי – כל הפגיעות רחוקות מהמטרה. בייאס גבוה – נשק לא מאופס, שונות גבוהה - אדם שלא יודע לכוון.

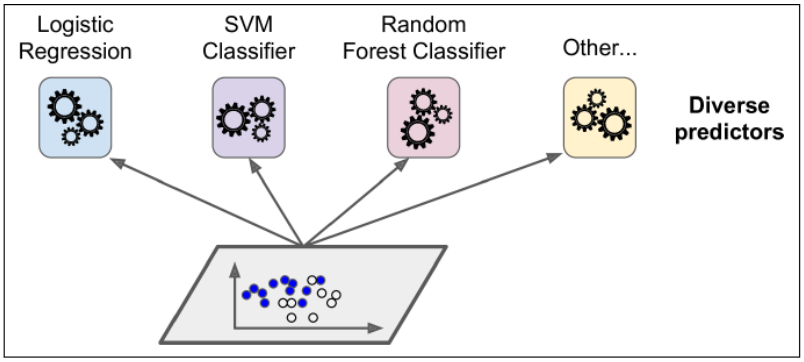
(הפירוש של bias זה נטייה/ משוא פנים/ דעה משוחדת)

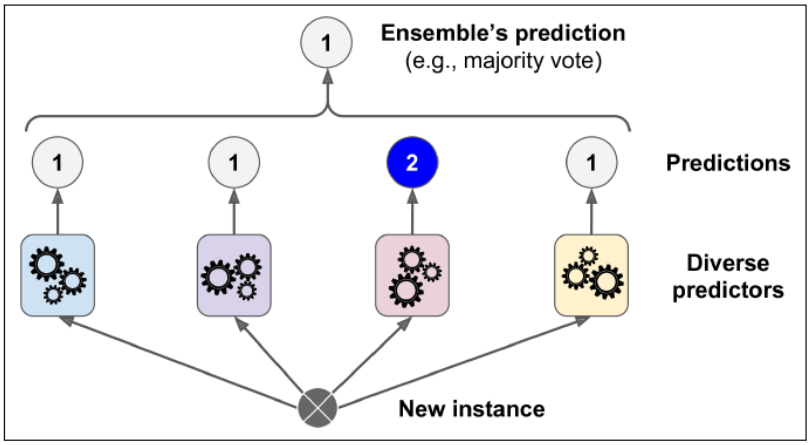
6. הקשר בין בייאס לאנסמבל זה שכאשר יש לנו דאטה סט שיש לנו הערכה/ תיאוריה עליו, אנחנו למעשה מניחים שהכל "שואף" למטרה מסויימת, כלומר שגם אם מודל קטן אחד נוטה לכיוון אחד ומודל קטן אחר נוטה לכיוון אחר - בסופו של דבר הבייאס, כלומר ממוצע השגיאה הוא 0 או קרוב אליו, איך זה יוצא? כי גם אם מופע בודד נמצא בסטייה גדולה יחסית לכיוון אחד, לרוב תהיה גם סטייה מקבילה מהכיוון השני כך שבסוף כאשר יש לי כמות גדולה מאוד של נתונים – נוצר איזון.

7. באותו אופן אנחנו ננסה להימנע מהטיה, שיכולה להיגרם למשל ע"י הכנסה לtrain של מודל המזהה תמונות של כלבים וחתולים 90% תמונות של כלבים ו10% של חתולים – ככל הנראה המודל יהיה מוטה לכלבים, נשים לב שבמצב זה גם מודל 0 שאומר תמיד כלב – הוא יקבל דיוק של 90%, זהו למעשה מצב של בייאס של המודל.
8. אנו מבינים כי בייאס הוא נתון מאוד משמעותי, כי בשונה ממצבים בינאריים – כלב/חתול וכד' – החיים הם לא בינאריים, ולכן כאשר אנחנו מנסים לנתח הסתברות מסויימת יש סיכוי שישנה הטיה שאנו לא שמים אליה לב. למשל אנו יכולים לנתח מה ההסתברות של אדם בבית משפט לצאת זכאי ואנו מנתחים את כל המקרים בהם זוכו ולא זוכו, אבל יכול להיות שחלק מאלו שזוכו – המשפט שלהם התנהל אחרי ארוחת צהריים, שהשופטים חזרו נינוחים רגועים ושבעים ולכן גם היו רחמנים יותר, הם היו מוטים, היה פה בייאס.
הסיבה השנייה שבייאס הוא נתון משמעותי כל כך הוא שבייאס הרבה יותר קל לתקן משונות, למשל לאזן את כמות התמונות שהכנסתי למודל כלבים וחתולים, לבחון מצבים שהנתונים החיצוניים שלהם (למשל שעת המשפט) כמה שיותר דומים וכו'.
9. קישור לסרטון שעוזר להבין את ההבדל בין שונות גבוהה ובייאס נמוך לעומת בייאס נמוך ושונות גבוהה. השאלה בסרטון היא מה עדיף - דיוק או מהימנות? במובן מסוים מקביל לבייאס אל מול שונות. אם נסתכל על התמונה עם 4 המטרות הוא אומר שעדיף את השמאלי התחתון של בייאס גבוה ושונות נמוכה, יורה טוב ונשק לא מאופס, מאשר הימני העליון – שונות גבוהה וביאס נמוך – נשק מאופס ויורה לא טוב, כי המהימנות של האירוע/ המבחן גבוהה יותר, ואז יהיה ניתן לשפר את הדיוק - את הנשק אפשר בקלות לאפס בעוד שאת היורה יהיה קשה לאמן. https://youtu.be/hRAFPdDppzs

## Voting classifier

10. כאשר אני משתמש באנסמבל אני למעשה יכול להשתמש במודלים פשוטים כך שבסוף אני יכול למשל לבחור hard voting ולקבוע את החיזוי/סיווג לפי התשובה של הרוב (גם אם רוב בפער קטן). בדוגמה הנ"ל יש אפילו אנסמבל של אנסמבל – בחירה של מספר מודלים אנסמבליים. האופציה השנייה



היא soft voting, להשתמש במודלים בpredict probability ומבין כל המודלים שמריץ במקביל – לא לומר לי מה הרוב אמרו אלא "להקשיב" ולבחור את המודל שהגיע לprobability הגבוה ביותר.

11. אופן המימוש של הקוד:

The following code creates and trains a voting classifier in Scikit-Learn, composed of three diverse classifiers (the training set is the moons dataset, introduced in Chapter 5):

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

log_clf = LogisticRegression()
rnd_clf = RandomForestClassifier()
svm_clf = SVC()

voting_clf = VotingClassifier(
    estimators=[('lr', log_clf), ('rf', rnd_clf), ('svc', svm_clf)],
    voting='hard')
voting_clf.fit(X_train, y_train)
```

Let's look at each classifier's accuracy on the test set:

```
>>> from sklearn.metrics import accuracy_score
>>> for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
...     clf.fit(X_train, y_train)
...     y_pred = clf.predict(X_test)
...     print(clf.__class__.__name__, accuracy_score(y_test, y_pred))
...
LogisticRegression 0.864
RandomForestClassifier 0.896
SVC 0.888
VotingClassifier 0.904
```

12. הבהרה מעניינת – אחד העקרונות החשובים בtrain test ובכללי בבדיקת ציון למודל הוא לא להגיע למצב של over fitting, כלומר שהמודל יותר מידי מאומן. איך זה יקרה? אם אני מכניס לשלב האימון חלקים מהטסט (או אפילו את כולו). כך המודל פשוט שולף לtest את התשובה שכבר ראה. עם זאת, למרות שלא נרצה לבצע over fitting אנחנו כן נהיה מעוניינים שתהיה את <u>האפשרות</u> לover fitting, כי במידה ונגיע למצב בו אנחנו לא מצליחים לעלות מעל score מסוים ככל הנראה המודל שלי פשוט מידי – למשל מודל החוזה הטלת מטבע מאוזן – לא ניתן לייצר מודל שינחש נכון 80 או 90% מההטלות. לכן האפשרות לover fitting היא חשובה כי היא מעידה על כך שהמודל מורכב יותר. ואם המודל מורכב יותר ככל שנשתמש ביותר מודלים, שכל אחד "תוקף" את המקרה מכיוון אחר – אנחנו נוכל להגיע לתוצאות טובות יותר ככל שאני מאמן יותר את המודל, כי כל מודל מצייר פונקציה של החיזוי שלו, ו"מזהה" נקודה אחרת או התנהגות שונה של הפונקציה שהמודלים האחרים לא זיהו, ומכניס אותו לפונקציה הכללית.

## <u>Bagging/ Pasting classifier</u>

13. מה ההסתברות לשלוף רביעיית אסים מחבילת קלפים רגילה? $\frac{4}{52}*\frac{3}{51}*\frac{2}{50}*\frac{1}{49}$.

אבל אם לאחר כל שליפה אחזיר את הקלף חזרה ואדגום מחדש? התשובה היא $\left(\frac{4}{52}\right)^4$ שזה סיכוי של כמעט פי 10 יותר. זהו הרציונל של Bagging/ Pasting classifier. הוא "משתמש"

במודל אחר, למשל decision tree, ועליו מבצע תהליך שבו הוא דוגם כל פעם קבוצה של נקודות, מייצר פונקציית חיזוי, כאשר הוא "יחזיר" את הנקודות שהשתמש בהן ובוחר קבוצת נקודות אחרת ועל ידיהן משפר את הפונקציה – זה ייקרא bagging. פעולת pasting זה כאשר המודל יבצע דגימה אקראית כל פעם של קבוצת נקודות אחרת ויבחר את הנקודות שהחזירו את החיזוי הטוב ביותר.

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

bag_clf = BaggingClassifier(
    DecisionTreeClassifier(), n_estimators=500,
    max_samples=100, bootstrap=True, n_jobs=-1)
bag_clf.fit(X_train, y_train)
y_pred = bag_clf.predict(X_test)
```

## שיעור 8

### סוגי טעויות - trade-off בין הvariance לbias

1. הבנה תיאורטית חשובה מאוד של למידת מכונה היא שכאשר יש לנו שגיאה בחיזוי, העובדה שיש לנו טעות כוללת של מודל לא חייבת לנבוע מסיבה אחת אלא יכולה להתבטא כסכום של שלוש שגיאות שונות מאוד:
   א. biasה יגרור טעות כאשר השגיאה נובעת מהנחות שגויות (זה המשמעות של bias – הטיה/שוחד, אני כאילו מוטה להנחה מסויימת), כגון הנחה שהנתונים ליניאריים כשהם למעשה ריבועיים. סביר להניח שמודל הטיה גבוה לא יתאים לנתוני האימון.
   ב. varianceה יגרור טעות כאשר השגיאה נובעת מהרגישות המוגזמת של המודל לשונות קטנה בנתוני הtrain. מודל עם דרגות רבות של חופש (כגון מודל פולינום בדרגה גבוהה – בו הוא "מכריח" את עצמו לעבור בכל נקודה ונקודה) עשוי להיות בעל שונות גבוהה, כי הוא לא יודע להתמודד עם מקרים שמחוץ לגרף שלו, הוא לא מזהה דפוס או מגמה בדאטה, כי הוא למעשה מתאים יותר מידי לtrain.
   ג. שגיאה בלתי ניתנת להפחתה (Irreducible error)
   חלק זה נובע מהרעש של הנתונים עצמם. הדרך היחידה לצמצם חלק זה של השגיאה היא לנקות את הנתונים (למשל, לתקן את מקורות הנתונים, כגון חיישנים שבורים, או לזהות ולהסיר חריגים).

2. הגדלת המורכבות של מודל תגדיל בדרך כלל את השונות שלו ותפחית את ההטיה שלו. לעומת זאת, הפחתת מורכבות המודל מגבירה את ההטיה שלו ומקטינה את השונות שלו. זו הסיבה שזה נקרא פשרות. הtrade-off זה בעצם במשחק הזה בין הגדלת/הקטנת השונות/הבייאס.

### Feature importance

3. בכל דאטה סט שנעבוד עליו אנו מניחים שיש משתנים שהשפעתם על המודל תהיה משמעותית יותר מאחרים, למשל במחיר דירה, אם נוסיף 3 מ"ר לבית זה פחות משמעותי מהוספה של עוד חדר או חניה צמודה. למה זה עוזר לי? כי למשל אם יש לי עשרות פרמטרים ככל הנראה יהיו חלק מהפיצ'רים שיהיו משמעותיים יותר ואותם נרצה לשמור, כך שיכול להיות שרמת הדיוק תרד אולי במעט אבל המודל יהיה פשוט הרבה יותר, או אפילו ממוקד הרבה יותר כך שיהיה ניתן

לעבוד עליו ולהגיע לתובנות חדשות שלא היה ניתן לראות אותן עם פיצ'רים רבים כ"כ. באותו אופן זה נכון לגבי תמונות, כשאני מבצע עיבוד של תמונה ישנם פיקסלים חשובים יותר מאשר אחרים (למשל אלו שבפינות/מסגרת וכו'). למשל בתמונה הנ"ל אנו רואים את החשיבות של כל פיקסל בדאטה סט של האירוסים, כעת אנו יכולים לצמצם משמעותית את כמות הפיקסלים שנבדוק, למשל 100 פיקסלים (10*10) במקום 784 (28*28), ובכללי לא רק לצמצם את זמן הריצה (כאשר יש לו כמות מאוד גדולה של תמונות לעבור עליהן), אלא גם להימנע מoutliers, שמסיחים את הדעת ומושכים את החיזוי לכיוון אחר.

```
>>> from sklearn.datasets import load_iris
>>> iris = load_iris()
>>> rnd_clf = RandomForestClassifier(n_estimators=500, n_jobs=-1)
>>> rnd_clf.fit(iris["data"], iris["target"])
>>> for name, score in zip(iris["feature_names"], rnd_clf.feature_importances_):
...     print(name, score)
...
sepal length (cm) 0.112492250999
sepal width (cm) 0.0231192882825
petal length (cm) 0.441030464364
petal width (cm) 0.423357996355
```

Similarly, if you train a Random Forest classifier on the MNIST dataset (introduced in Chapter 3) and plot each pixel's importance, you get the image represented in Figure 7-6.

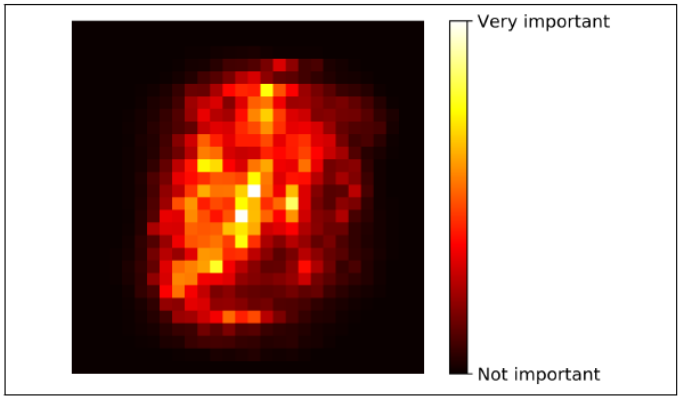


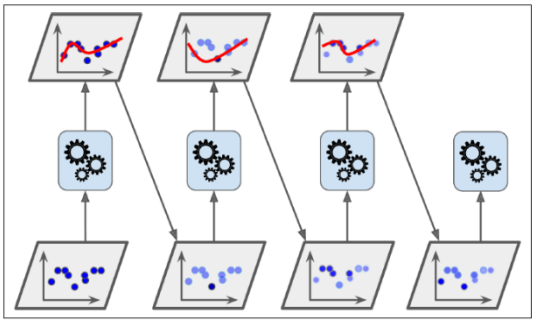

*Figure 7-6. MNIST pixel importance (according to a Random Forest classifier)*

## המונח Boosting

4. המונח boosting (חיזוק/ הגברה) מתייחס לכל שיטת אנסמבל שיכולה לשלב מספר לומדים חלשים ללומד חזק. הרעיון הכללי של רוב שיטות הboosting הוא לאמן מודלים של חיזוי ברצף, כל אחד מנסה לתקן את קודמו. ישנן שיטות חיזוק רבות זמינות, אך ללא ספק הפופולריות ביותר הן AdaBoost (קיצור של Adaptive Boosting – הגברת הסתגלות) ו- Gradient Boosting (הגברת שיפוע).

## Adaboost/ boostrap

1. כמו שראינו, כאשר משתמשים בbagging (ע"י הוספת הפיצ'ר boostrap=true), נוכל לבצע בחירה אקראית של נקודות כך שישפרו את הפונקציה. בשיטה של Adaboost אנו משתמשים שוב ברעיון הזה של מודל אחד שמשפר את קודמו, אלא שבשונה מbagging אנו לא מבצעים בחירה חלקית והחזרה של הנקודות לבחירה מחודשת אלא לוקחים מראש את כל הנקודות כך שמודל הניבוי החדש מתקן את קודמו ע"י כך שמקדיש קצת יותר תשומת לב (ע"י הגברת המשקל – המשמעות) למקרי האימון שהמודל הקודם לא התאים להם – כלומר טעה בהן/



היה רחוק מהן. והמודל השני מבצע את החיזוי שלו על הtrain המעודכן בו נקודות הטעות הן עם משקל משמעותי יותר, ועל הפלט והטעויות שלו – יעבוד המודל הבא. (הערה – הכלל הזה של העלאת המשקל של נקודה מסויימת יכול לעזור לנו בפעמים שנרצה "לחייב" את המודל "לעבור" דרך נקודה מסויימת, כאשר על פני השטח הנקודות שוות אבל אנו ניתחנו והגענו למסקנה שנקודה מסויימת מהווה גורם משפיע משמעותי).

5. דוגמה לאופן מימוש adaboost עם decision tree classifier:

```
from sklearn.ensemble import AdaBoostClassifier

ada_clf = AdaBoostClassifier(
    DecisionTreeClassifier(max_depth=1), n_estimators=200,
    algorithm="SAMME.R", learning_rate=0.5)
ada_clf.fit(X_train, y_train)
```

If your AdaBoost ensemble is overfitting the training set, you can try reducing the number of estimators or more strongly regularizing the base estimator.

## שיעור 9

### המשמעות של שגיאה

1. מה זה אומר דיוק של מודל? כמות החיזויים שאכן צדקתי בהם, לכן שגיאה זה היחס בין הטעויות לבין המספר הכולל. אבל ישנם מקרים רבים שיש imbalanced data ואז לא לכל דאטה יש את אותה הסתברות להופיע, ואז יכולות להיות נקודות שמשמעותיות יותר אם טעיתי בהן, או פחות. מצב זה יקרה או בטעות, שמראש הדאטה לא מאוזן, או שהמתכנן של הקוד בנה אותו ככה שהוא מעדיף שאם תהיה טעות היא תהיה בנקודות מסוימות ולא באחרות מתוך הבנה שגם אם יטעה בנקודות הלא חשובות – לאורך זמן הנקודות המשמעותיות הן הפיבוט המרכזי כך שהדיוק של המודל יהיה גבוה.
בהקשר זה הציין כי זה הכוח של adaboost שראינו בשיעור שעבר, שגם אם מבחינתנו התחלנו את התהליך במצב בו כל הנקודות הן בעלות אותו משקל יחסי, ובאמת הגרף שלנו הוא על קו אחד מעוגל ממעלה שניה למשל, וישנה נקודה אחת רחוקה על הגרף, אם היא תהיה בעלת אותו משקל כמו השאר היא בעצם תמשוך את הפונקציה לכיוונה, בadaboost לאחר השגיאה בנקודה הרחוקה הוא יגביר את המשקל של הנקודה הרחוקה, כך שבפעם הבאה היא תמשוך יותר את כל המודל אליה ותגרום כנראה לשגיאה רבה יותר – ואז היא תתוקן ע"י הגברת המשקל חזרה של הנקודות שטעה בהן עכשיו שביניהן ניתן להעביר את גרף הפונקציה. יוצא מצב שגם אם יש לנו כמות של נקודות שטעה בהן, עדיין ניתן להגיע לדיוק גבוה של המודל כי הוא מבין שיש נקודות משמעותיות יותר.

2. למדנו בעבר על מודל טיפש/ מודל 0 – dummy model, שהוא מודל הכי פחות מורכב שמחזיר score סביר. למשל בהטלת מטבע הוא אומר תמיד את אותה תשובה – לאורך זמן הוא יקבל דיוק של 50%.
3. Weak learner הוא גם מודל עצלן אבל הוא כזה שהוא טיפה יותר טוב ממודל טיפש.
4. הרעיון המהותי והפורץ דרך כל כך של למידת אנסמבל זה שאם ניקח כמות גדולה של לומדים חלשים – הם יכולים לייצר תוצאות טובות יותר מאשר לומד אחד טוב מאוד.
נסביר את הרעיון המתמטי – נניח שיש לי 3 שופטים שמדייקים כל אחד ב70% לעומת שופט שמדייק ב90% אחוז – את מי מהם עדיף לי לקחת? התשובה היא שאת ה90%, אבל אם נסתכל על המספרים נקבל מסקנה משמעותית. כאשר אני מבצע בחירה של הרוב קובע האפשרויות לקבלת דין צדק הן או שכולם צודקים, או ש2 צודקים ו1 טועה (כך 3 פעמים כל פעם אחד אחר טועה). אם כך נקבל: $(0.7^3)+(0.7^2*0.6)*3 = 0.784$, כלומר צירוף של 3 שופטים שמדייקים ב70% ייתן לי סיכוי דיוק של <u>מעל</u> 70%, נסתכל על הרכב של 5 שופטים: או שכולם צודקים, או שיש אחד שטועה (יש 5 אופציות כאלו), או ש2 טועים (יש 10 אופציות כאלו – בחירה של 5choose2) כך שנקבל: $(0.7^5)+(0.7^4*0.3)+(0.7^3*0.3^2)*10 = 0.836$. אם כך אם ניקח כמות גדולה יותר של שופטים כאלו – נוכל אפילו לעבור את ה90% שצודק השופט היחיד.

**<u>הסבר תיכנותי/מתמטי לadaboost</u>**

5. נניח שיש לי 4 פיצ'רים, למשל בטיטאניק – מין, מחלקה, גיל והאם שרד. לכל נקודה יש $x_i$ – הפיצ'רים , $y_i$ - החיזוי שלי – שרד/ לא שרד, שיסומן ב1/1- (וימשוך אותי למינוס כאשר ישנה שגיאה). הקוד של adaboost ירוץ מעין כך:
for t=1 to T – יבצע לולאה של כמות האיטרציות שadaboost יבצע
construct $D_t$– יבצע חלוקה כל פעם של נקודות אחרות – (הגדרת התפלגות להצבה זו)
find work classification $h_t(x)$ תמצא לומד חלש שמדייק לפחות ב51%.
with error $\sum t= Pr_{Dt} [h_t(x) \neq y_i]$ - חישוב הטעויות של הלומד החלש כך שאפשר לייצר פעם הבאה $D_t$ חדש. זהו בעצם חישוב של סך ההסתברויות לטעויות.
6. איך מייצרים התפלגות $D_t$ ? (להוסיף הדגמות משיעור 10.5.22 מדקה 35

_____טבלה_____

_____--)

א. $D_1(i) = \frac{1}{n}$ (n – מספר דוגמאות) זו החלוקה distribution הראשונה של הדוגמה הi, זו תהיה התפלגות אחידה - $\frac{1}{n}$, כל חלק הוא בעל אותו משקל ביחס לאחר. עבור $D_1(i)$ אני יכול לייצר $h_1(x)$ שזה בעצם המודל ( - הפונ' של המודל) עבור החלוקה הראשונה.

ב. איך מייצרים התפלגות נוספת $D_{t+1}(i)$? אני מעוניין להגדיל את המשקל של כל הנקודות שהייתה בהן שגיאה – miss classification.

כאמור $D_t(i)$ זו ההתפלגות הקודמת, ו$\sum t$ זו ההסתברות לטעויות בהתפלגות הקודמת.

ג. $D_{t+1}(i)=(D_t(i) * e^{-a_t * y_i * h_t(x_i)}) / Z_t$

ד. $\frac{Dt(i) * e-at * yi * ht(xi)}{zt}$

כאשר $z_t$ הוא הרכיב המנרמל כדי שסך ההסתברויות יהיה שווה ל1. (הערה – לרוב הרכיב המנרמל יהיה במכנה, למשל בחוק בייס אנו מנרמלים את הבחירה אל מול כל יתר הבחירות האפשריות – אם הבחירה שלי היא 100%, המכנה יהיה סכום כל האפשרויות שגם הוא 100% ואקבל 1). הערה – כאשר אני מחשב אחד ביחס לשני לא בהכרח שאהיה חייב להוסיף את הרכיב המנרמל אלא רק כשארצה לקבל את היחס שלו ביחס לכלל.
איך מחשבים את $z_t$? זה בעצם סיגמא של כל הdistributions.

ה. $a_t$ יביא לכך שככל שהטעות גדולה יותר כך הוא ייתן יותר משקל לדוגמאות עם הטעויות.

$$a_t = \frac{1-\sum t}{\sum t}$$

ו. $e^{-a_t * y_i * h_t(x_i)}$ נסתכל על החזקה - $y_i$ הוא הלייבל של האיבר הi – למשל שרד/לא שרד ויסומן ב1/1-, כנל $h_t$, לכן אם המודל צודק אז נקבל 1, כי זה מכפלה של אותו סימן, אם הם עם סימנים הפוכים נקבל 1-. לכן אם הוא צודק נקבל $\frac{Dt^{(i)}}{e^{\alpha t}}$

ז. הערה חשובה – אנו יכולים לשאול ובצדק למה יש רק 2 אפשרויות ללייבלים, הרי לא תמיד הדאטה שלנו היא בינארית, למשל לזהות חיה מסויימת. התשובה היא שגם דאטה שהוא לא בינארי קל להפוך להיות בינארי למשל ע"י בדיקה האם זה כלב או לא כלב/ חתול או לא חתול וכו'. כלומר גם multiclass יכול להיות מיוצג ע"י סיווג בינארי.

## <u>טכניקות Unsupervised learning</u>

7. כאשר אנו מדברים על החיים האמיתיים שלנו רוב התופעות הן לא לינאריות וגם לא בהכרח שיהיו בעלי לייבל, למשל כשמופיע במחשב תמונה של חתול – המחשב לא יודע שהוא חתול אלא אם מתחת לתמונה כתוב איזו חיה זו - חתול. אבל יותר מזה, הדוגמא הכי נפוצה לUnsupervised learning זה יוטיוב/נטפליקס, שמציע לך סרט שאתה עשוי לאהוב – אין פה נכון או לא נכון מובהק.
8. הרעיון של Unsupervised learning מביא אותנו לרעיון שנקרא clustering (מקבץ) שאם נראה גרף שבו בפינה אחת מרוכזות קבוצה אחת של נקודות, ובפינה שניה קבוצה מרוכזת של נקודות – באופן אינטואיטיבי נאמר שאלו 2 קבוצות של דברים שונים. כאשר נראה את זה בעיניים נאמר זאת ישר, אבל המחשב, כמו הדוגמא עם החתול, לא יידע לומר שאלו 2 קבוצות, המחשב דבר ראשון צריך לדעת לכמה קבוצות אני רוצה לחלק.

דוגמא להבהרה – כמה סוגי בני אדם יש? אולי 2 – גבר ואישה, אבל יש גם התפלגות של לבנים/שחורים, וגם תחת כל אחת מהן יש עוד קטגוריות למשל תחת שחורים יש המון ארצות מוצא – הודים, אפריקאים, דרום אמריקאים וכו'. זו הסיבה שצריך להחליט את אופן החלוקה, כלומר לכמה קלאסים אני רוצה לחלק.

9. תחת Unsupervised learnin נעסוק ב3 שיטות/תפיסות מרכזיות:

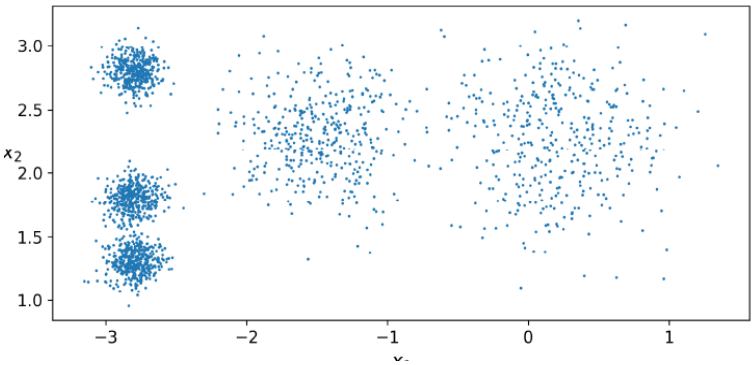
א. Clustering - הניסיון לסווג דברים שונים לקבוצות. הרעיון הזה יוצא מתוך ההבנה שדברים דומים נמצאים קרוב אחד לשני. אני נכנס לכיתה ורואה קבוצות של חברים יושבות יחד, ככל הנראה שלאלו בכל קבוצה יש קשר אחד לשני, כך שאם אדם מסוים חבר של אחד האנשים, הגיוני שהוא יהיה חבר גם של עוד חברים שבאותה קבוצה. אהבת את הסרטון הזה – הרבה אנשים שאהבו את הסרטון הזה גם ראו את הסרטון הבא.

ב. Anomaly detection (זיהוי אנומליות/ זרות)– הניסיון להבין מה זה "נורמלי", אם חברת האשראי רואה חיוב של 7000 שקל באלי אקספרס או מאיזו חנות לא קשורה בחו"ל שעד היום לא קניתי בה – סביר להניח שהיא תתקשר אליי להבין אם אני ביצעתי את הקנייה. אבל אם יהיה חיוב של 7000 שקל במוסך – היא לא תתקשר, כי זה חיוב שהוא הגיוני. הרעיון של Anomaly detection הוא לזהות את החריגות הללו.

ג. Density estimation (אומדן צפיפות) – זיהוי של חריגים ע"י אבחנה של הצפיפות של הנקודה עם נקודות אחרות, אם יש לנו קבוצה של נקודות קרובות ונקודה אחת רחוקה – כנראה שהנקודה הזו אנומלית/זרה להן.

## K means

10. אלגוריתם k-means הוא אלגוריתם שפועל תחת שיטת הclustering, נסתכל על הגרף הבא – נוכל לטעון כי מדובר ב5 קבוצות, אבל כמו שהסברנו לפני כן – ישנן דרכים רבות להגדיר קבוצות, אנו יכולים גם להגדיר שיש פה 4 קבוצות – 3 בעלות צפיפות גבוהה, בעלות קשר מובהק, וקבוצה של כל אלו בעלי צפיפות נמוכה שאולי הקשר ביניהם לא משמעותי.




אלגוריתם הk-means למעשה בוחר k נקודות אקראיות, ניקח לדוגמה k=5, ואז "צובע" כל נקודה בצבע של אחת מ5 הנקודות שנבחרו שהכי קרובה אליה. לאחר מכן ייקח שוב 5 נקודות אקראיות אחרות ויצבע שוב את כל הנקודות בצבע של אחת מ5 הנקודות שהכי קרובה אליה. כך ימשיך עד שלמרות בחירה שונה של נקודות – ה"מפה" תישאר ללא שינוי.

```
from sklearn.cluster import KMeans
k = 5
kmeans = KMeans(n_clusters=k)
y_pred = kmeans.fit_predict(X)
```

השיטה הזו טובה עבור סיווג לקבוצות בהנחה שאני יודע בכמה קבוצות מדובר, למשל אני יודע שיש בקמפוס 8 תאי שירותים ואני רוצה להציג בכל נקודה בקמפוס מה תא השירותים הכי קרוב אליך. אבל לא בהכרח שנשתמש בשיטה רק כאשר הk ידוע – כי אולי דווקא ע"י

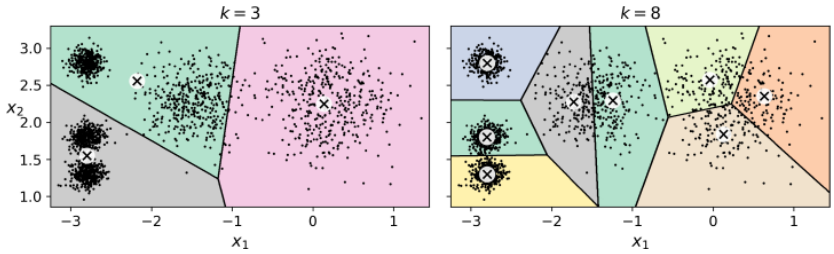
שיטה זו ניתן להבין משהו חדש על הדאטה – אם הנחתי שיש 5 קבוצות אבל אני מקבל חיזוי מדויק יותר אם אתייחס לזה כ4 או 8 קבוצות.

11. אני יכול גם להגדיר את הנקודות הספציפיות שאני מעוניין לקבוע את "צביעת" המודל ביחס אליהן, בהנחה שאני יודע שהן בוודאי נקודות מk קבוצות שונות. אופן הביצוע בקוד יהיה:

```
good_init = np.array([[-3, 3], [-3, 2], [-3, 1], [-1, 2], [0, 2]])
kmeans = KMeans(n_clusters=5, init=good_init, n_init=1)
```

12. פתרון נוסף הוא להפעיל את האלגוריתם מספר פעמים עם אתחולים אקראיים שונים ולשמור על הפתרון הטוב ביותר. זה נשלט על ידי הפיצ'ר n_init (כברירת מחדל הוא שווה ל-10, כלומר שכל האלגוריתם מהתמונה הראשונה של הקוד שצירפנו פועל 10 פעמים) לאחר שנקרא לfit(), Scikit-Learn שומר על הפתרון הטוב ביותר. אבל איך בדיוק הוא יודע איזה פתרון הוא הטוב ביותר? זה נקרא <u>האינרציה של המודל</u>: זהו המרחק הממוצע בריבוע בין כל נקודה למרכז הקרוב ביותר שלו.

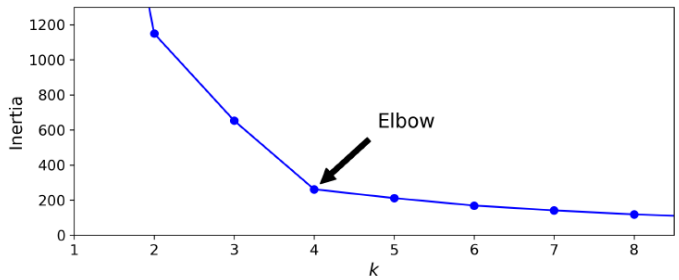
הגדרנו את מספר הקבוצות k ל-5 כי זה היה ברור בהסתכלות על הנתונים שזהו המספר הנכון של הקבוצות. אבל באופן כללי, זה לא יהיה כל כך קל לדעת איך להגדיר k, והתוצאה עלולה להיות גרועה למדי אם נגדיר אותו לערך הלא נכון. לדוגמה, כפי שניתן לראות בתמונה הנ"ל הגדרת k ל-3 או 8 מביאה למודלים גרועים למדי: אז איך נוכל פשוט לבחור




את הדגם עם האינרציה הנמוכה ביותר? למרבה הצער, זה לא כל כך פשוט. האינרציה עבור k=3 היא 653.2, וזה הרבה יותר גבוה מאשר עבור k=5 (שהיה 211.6), אבל עם k=8, האינרציה היא רק 119.1. חשוב להבהיר כי עלולים לטעות ולחשוב שאני שואף לאינרציה כמו שיותר נמוכה. אבל האינרציה **אינה** מדד לביצועים טובים יותר, מאחר ומובן לנו שככל שנבחר יותר נקודות כך המרחק אליהן יהיה נמוך יותר, למשל עבור כל הנקודות האינרציה תהיה 0. אנו שואפים לאינרציה נמוכה אבל כזו שתשקף באמת את אופן ההתנהגות של המודל – ולחלק לקבוצות כפי שבאמת המציאות הכתיבה זאת בגרף, בשאיפה לחזות אל מה שיהיה בעתיד לפי הקבוצות האמיתיות.

ניתן לשרטט את האינרציה כפונקציה של k:

כפי שניתן לראות, האינרציה יורדת מהר מאוד ככל שאנו מגדילים את k עד 4, אך אז היא פוחתת הרבה יותר לאט ככל שאנו ממשיכים להגדיל את k. לעקומה זו יש בערך צורה




*Selecting the number of clusters k using the "elbow rule"*

של זרוע, ויש "מרפק" ב-k=4, כך שאם לא היינו יודעים טוב יותר כמה קבוצות יש - 4 תהיה בחירה טובה: כל ערך k נמוך יותר יהיה יביא לאינרציה גבוהה (ולמרחק רב של הנקודות מנקודות הבחירה), בעוד שכל ערך גבוה יותר (ולמעשה חלוקה לקבוצה נוספת) לא ישפיע יותר מידי, ויכול להיות שאנחנו מפצלים קבוצות לשניים ללא סיבה מוצדקת.

## שיעור 10

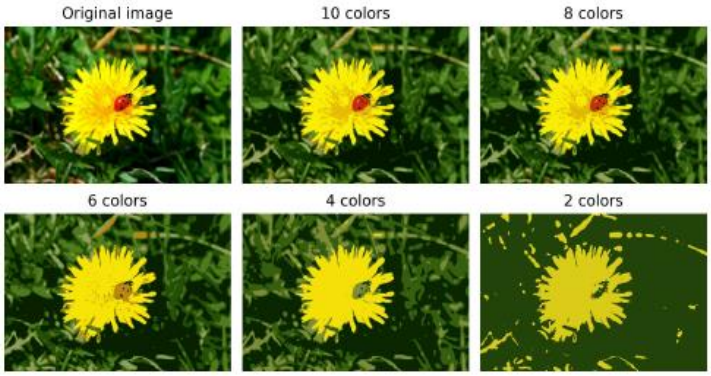
### שימוש בclustering לניתוח תמונה (image segmentation)

1. image segmentation הוא השם הכללי לחלוקת תמונה למספר פלחים לצורך עיבודה.

א. בsemantic segmentation , לכל הפיקסלים שהם חלק מאותו סוג אובייקט מוקצים אליהם אותו קטע. לדוגמה, כל הפיקסלים שהם חלק מתמונה של הולך רגל עשוי להיות מוקצה לקטע "הולך רגל" וכל הולכי הרגל ייצבעו "באותו הצבע".

ב. בinstance segmentation , כל הפיקסלים שהם חלק מאותו אובייקט בודד מוקצים לאותו אובייקט, אבל במקרה זה תהיה התייחסות שונה לכל הולך רגל. האבחון וה"צביעה" של כל אובייקט כשלעצמו בסגמנטציה סמנטית מושגת כיום באמצעות ארכיטקטורות מורכבות המבוססות על רשתות נוירונים קונבולוציוניות (CNN – convolution neural network).

אנחנו הולכים לעשות משהו הרבה יותר פשוט: color segmentation. אנו פשוט נקצה פיקסלים ל- אותו קטע אם יש להם צבע דומה. ביישומים מסוימים, זה עשוי להספיק, למשל אם אתה רוצה לנתח תמונות לוויין כדי למדוד כמה יער כולל אזור שיש באזור, פילוח הצבע עשוי להיות יעיל.




נסתכל למשל על התמונה הזו – אני יכול לייחס לתמונה כאילו היא 10 קבוצות צבעים, או 8, או אפילו 2. למה החיפושית נעלמה? כי הנפח שלה אל מול יתר הצבעים הוא זניח, כך שלמרות שהצבע שלה שונה ובולט מאוד בתמונה, K Means לא מצליח להקדיש לה אשכול/ קבוצת סיווג משלה. כפי שהוזכר קודם לכן, K-Means מעדיף אשכולות של גדלים דומים – ומעשה את הרוב. יש יתרון בהקטנה של כמות הקבוצות בכך שיותר "קל" לסווג עבור פיקסל נתון האם הוא חלק מהפרח או לא – כי זו שאלה בינארית – צהוב או לא צהוב, אבל אנחנו מפסידים חלק מהמהות של clustering – שדברים קרובים קשורים זה לזה, ולמעשה "מפספסים" את העובדה שמעבר לדשא ופרח יש פה גם חיפושית.

המימוש בקוד:

```
>>> from matplotlib.image import imread  # you could also use `imageio.imread()`
>>> image = imread(os.path.join("images","clustering","ladybug.png"))
>>> image.shape
(533, 800, 3)
```

```
X = image.reshape(-1, 3)
kmeans = KMeans(n_clusters=8).fit(X)
segmented_img = kmeans.cluster_centers_[kmeans.labels_]
segmented_img = segmented_img.reshape(image.shape)
```

**שימוש בclustering לpreprocessing**

2. clustering יכול להיות גישה יעילה להורדת מימדים (dimensionality reduction), למשל אני מבין שכדי להבין שמופיע כלב בתמונה – עיקר הפיקסלים יהיו לרוב במרכז, ולא בצדדים, ואני יכול "לזרוק" חלק מהמידע כדי לייעל את הלמידה ולא להיות מוסח ממידע אמיתי אבל לא בהכרח רלוונטי שמצביע לי על מה מופיע בתמונה. במיוחד כשלב עיבוד מקדים לפני שימוש במודל unsupervised. לדוגמה, נטפל בדאטה סט של הספרות המכיל 1,797 תמונות בגווני אפור בגודל 8×8 המייצגים את הספרות 0 עד 9. עם מודל פשוט של logistic regression הגענו לדיוק של 96.6%, בוא נראה אם נוכל להשתפר על ידי שימוש ב-KMeans כשלב עיבוד מקדים.

```
from sklearn.datasets import load_digits

X_digits, y_digits = load_digits(return_X_y=True)
```

Now, let's split it into a training set and a test set:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_digits, y_digits)
```

Next, let's fit a Logistic Regression model:

```
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression(random_state=42)
log_reg.fit(X_train, y_train)
```

Let's evaluate its accuracy on the test set:

```
>>> log_reg.score(X_test, y_test)
0.9666666666666667
```

3. לפני שנסביר איך לייעל עם clustering נכיר מושג חדש – pipline, במשמעות המילולית זה צינור, אבל המהות שלו זה ליצור תהליך מסוים, שהפונקציה תידרש לעבור דרכו שלב אחרי שלב. בשונה מפונקציה שמקבלת ערך ומחזירה משהו, pipline יכולה להגדיר גם פעולות שונות שלא בהכרח קשורות אחת לשניה ואני מעוניין שיבוצעו בסדר כרונולוגי שאקבע. ניצור pipline שתחילה יקבץ את מערך הtrain ל-50 clusters, ע"י הגרלה של 50 נקודות במרחב בפעולת הk-mean, ולאחר מכן יישם מודל רגרסיה לוגיסטי. הערה חשובה - למרות שמפתה להגדיר את מספר האשכולות ל-10, מכיוון שיש 10 ספרות שונות, לא סביר שהוא יצליח, **כי יש כמה דרכים שונות לכתוב כל ספרה**. יכול להיות שתהיה קבוצה מסויימת של המספר 7 שנראית כמו 7, קבוצה של 1 שנראית כמו 1, וקבוצה של

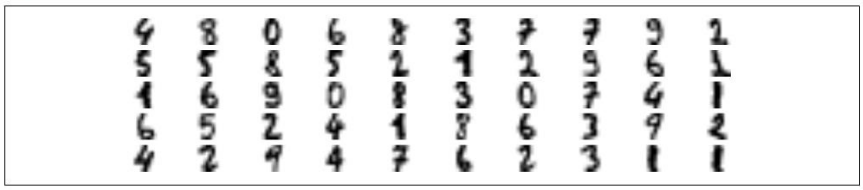
```
from sklearn.pipeline import Pipeline

pipeline = Pipeline([
    ("kmeans", KMeans(n_clusters=50)),
    ("log_reg", LogisticRegression()),
])
pipeline.fit(X_train, y_train)
```

Now let's evaluate this classification pipeline:

```
>>> pipeline.score(X_test, y_test)
0.9822222222222222
```

```
k = 50
kmeans = KMeans(n_clusters=k)
X_digits_dist = kmeans.fit_transform(X_train)
representative_digit_idx = np.argmin(X_digits_dist, axis=0)
X_representative_digits = X_train[representative_digit_idx]
```

Figure 9-13 shows these 50 representative images:



מספר שנראה או 1 או 7, אם באמת הרוב בtrain של קבוצה זו הוא 7 – הוא יתייחס אליו כ7. נשים לב שאם נכניס לו 50 קבוצות, ואז נבקש ממנו להדפיס לנו נציג מכל קבוצה – נראה למשל שהספרה 1 מופיעה 6 פעמים, שמסווג אותה כספרה שונה, בגלל הבדלים מסוימים וכד', לעומת 0 שמופיע רק 3 פעמים.

4. אבל עדיין נוכל לטעון שניתן לשפר, כי בחרנו את מספר האשכולות k=50 באופן שרירותי לחלוטין, ואולי יש בחירה טובה יותר. מכיוון ש-K-Means הוא רק שלב עיבוד מקדים ב classification pipeline, כעת מציאת ערך טוב עבור k היא הרבה יותר פשוטה מאשר קודם לכן, ע"י cross validation. נשתמש ב-GridSearchCV כדי למצוא את המספר האופטימלי של אשכולות. אני

```
from sklearn.model_selection import GridSearchCV

param_grid = dict(kmeans__n_clusters=range(2, 100))
grid_clf = GridSearchCV(pipeline, param_grid, cv=3, verbose=2)
grid_clf.fit(X_train, y_train)
```

Let's look at best value for *k*, and the performance of the resulting pipeline:

```
>>> grid_clf.best_params_
{'kmeans__n_clusters': 90}
>>> grid_clf.score(X_test, y_test)
0.9844444444444445
```

With *k*=90 clusters, we get a small accuracy boost, reaching 98.4% accuracy on the test set. Cool!

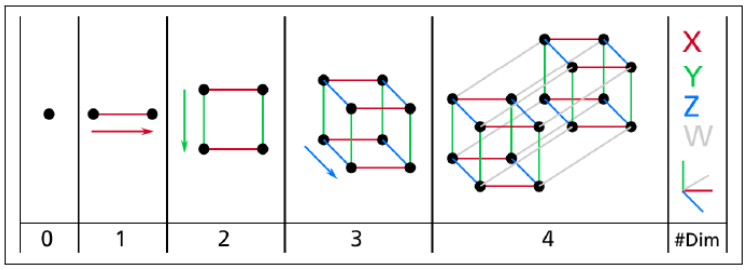
למעשה לא מסתפק רק במיקום של כל הנקודה על הגרף ועל פיה לחזות מה המספר, אלא גם לאחר השיבוץ בגרף – לנסות להבין האם ניתן לחלק את הנקודות על הגרף לקבוצות נושא – לפי אותו רעיון מרכזי שדברים דומים נמצאים קרוב אחד לשני. אז מספרים שברור שהם 3, יהיו במקבץ מאוד קרוב, ומספר שהוא בין 3 ל8, יהיה בין 2 הקבוצות הללו, והמודל הזה עוזר לנו "להחליט" ע"י השאלה- לאיזה קבוצה הוא קרוב <u>יותר</u>.

5. הערה חשובה – שמנו לב שעברנו מדיוק של 96.6% לדיוק של 98.44% שעל פניו אנו יכולים שלא לייחס לכך משמעות, אבל יש לכך משמעות מאוד גדולה כי למעשה העלייה של 1.8% בדיוק הוא למעשה הקטנה של הerror שלנו כמעט פי 2! זאת אומרת יש לנו פי 2 פחות טעויות.

## <u>הורדת מימדים - dimensionality reduction</u>

6. בעיות רבות של למידת מכונה כוללות אלפי או אפילו מיליוני פיצ'רים (תכונות). לא רק שזה הופך את האימונים לאיטיים ביותר, זה גם יכול להקשות בהרבה על מציאת פתרון טוב, כפי שנראה. בעיה זו מכונה לעתים קרובות קללת הdimensionality . אם יש לי עבור פיצ'ר מסוים 100 אופציות – אז עבור בחירה מסויימת יש 100 אופציות, אבל אם יש לי עוד פיצ'ר עם 100 אופציות יש לנו כבר בסך הכל 10,000 בחירות אפשריות, עבור 5 פיצ'רים כאלו יהיו לנו 10,000,000,000 אופציות, זה קצב גידול גבוה מאוד, כי נוצרו לנו זה המון מימדים.

7. מצד אחד אם ניקח מימד אחד – אותו פיצ'ר וקנה מידה ביחס לכולם- המידע שאוכל לקבל מזה הוא זעום, למשל רק הגילאים של הנוסעים בטיטאניק. אך מצד שני אם ניקח המון פיצ'רים – כלומר אם כל פיצ'ר-תכונה/ כל כיוונון או שינוי קטן מאובייקט אחד לאחר יהווה יישות אחרת וקטגוריה אחרת, בפני עצמו כל אחד מהם יהיה למעשה מקרה




קיצוני באותה תכונה ביחס לחבריו. זה המהות של מימדים, אם נסתכל בגרף הקוביות למעלה –

אפשר לצייר קובייה במספר מימדים, והשינוי במימדים יאפשר לנקודה מסויימת שניקח על הקוביה להיות קרובה יותר או פחות לקודקוד (לענייננו – קטגוריה, תכונה) אבל אם יש יותר מידי מימדים, (אנחנו לא יכולים לדמיין את זה) כל נקודה תהיה רחוקה מכל הנקודות – כי בתכונה הזו, שאני מייחס אותה כדבר שונה מהותית – פיצ'ר נפרד – אין אף אחת כמוה, כלומר כל נקודה תהיה קיצונית. אם לבן אדם יש המון תכונות, תחומי עניין, משימות והתעסקויות – סביר להניח שישנו תחום אחד/ תכונה אחת שהוא קיצוני בו, כלומר שהוא חורג מגדר הנורמה הממוצעת ביחס ליתר האנשים. הסיכוי של נקודה להיות קיצונית אנומלית הולך ועולה, מכיוון שהמימדים מעין מקופלים בתוך עצמם – כל נקודה שניקח למעשה תהיה קרובה לאיזשהי פינה. מה זה משפיע עליי? העלאת מימדים פורסת לנו את המודל ומאפשרת לזהות קיצוניים – חריגים, שלא מעידים על האמת, כי המודל נועד "לנבא" בשביל האמצע, ולא בשביל הקצוות. ולכן אם יש יותר מידי פיצ'רים כל דוגמה הופכת להיות קיצונית באיזשהו אספקט. במרחב עם המון מימדים נוצר לנו מרחב עצום, שרובו-אפשר לומר "ריק", כמו לדוגמא החוות בארצות הברית שיכולים להיות מספר קילומטרים בין בית אחד לאחר – אין שום יכולת לקשר ביניהם.

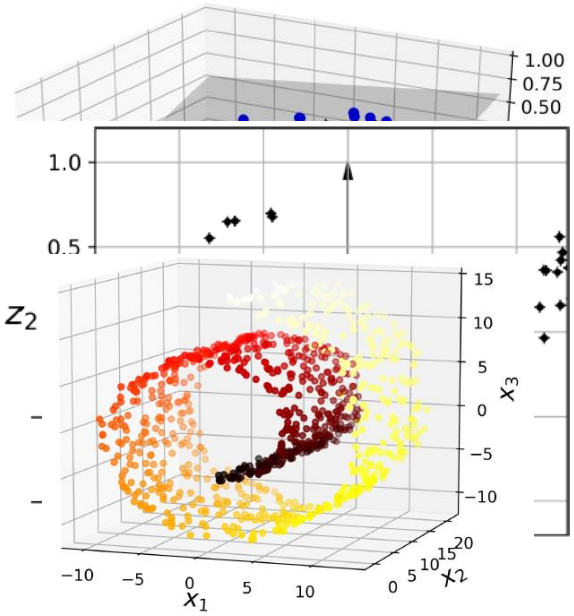
## שיעור 11

### המשך הורדת מימדים - dimensionality reduction

1. כאמור מימדים בשיח שלנו הוא פיצ'רים – תכונות, של האובייקטים בשורות, והבנו שככל שיש לנו יותר פיצ'רים אנחנו עלולים להיות בבעיה. בדיקת דם למשל מכילה מאות פרמטרים, כדי לאבחן מחלות מסוימות נדרש לוותר על חלק ולהצליח להתמקד באלו שההשפעה שלהם היא המשמעותית ביותר. כי אם נכניס את כולם למודל נקבל curse of dimensionality – מרחב מאוד דליל (sparse), ויהיה קשה לייצר ו אלגוריתם שמבין דפוסים וכד'. לכן אנחנו מוכרחים להוריד את מספר הפיצ'רים.

### Projection

2. השיטה הראשונה להורדת מימדים היא Projection – היטל/ הקרנה. ברוב הבעיות בעולם האמיתי, הנקודות שבtrain אינם מפוזרות באופן אחיד על פני כל הממדים. תכונות רבות לרוב עומדות לעצמן, בעוד שאחרות נמצאות בקורלציה גבוהה. כתוצאה מכך, כל מופעי הtrain למעשה נמצאים בתוך (או קרוב ל) תת-מרחב בעל מימד נמוך בהרבה של המרחב הגבוה. זה נשמע מאוד מופשט, נסתכל על דוגמה, בתמונה ניתן לראות מערך נתונים תלת מימדי המיוצג על ידי המעגלים. ישנו מישור מסוים, לאו דווקא אופקי שמקביל למישור X1,X2, בו כל מופעי האימון נמצאים קרוב לאותו מישור: זהו תת-מרחב במימד נמוך יותר (2D) של המרחב הגבוה (3D). כעת אם נטיל כל נקודה כזו בניצב על תת-מרחב זה (כפי שמיוצג על ידי הקווים הקצרים המחברים את

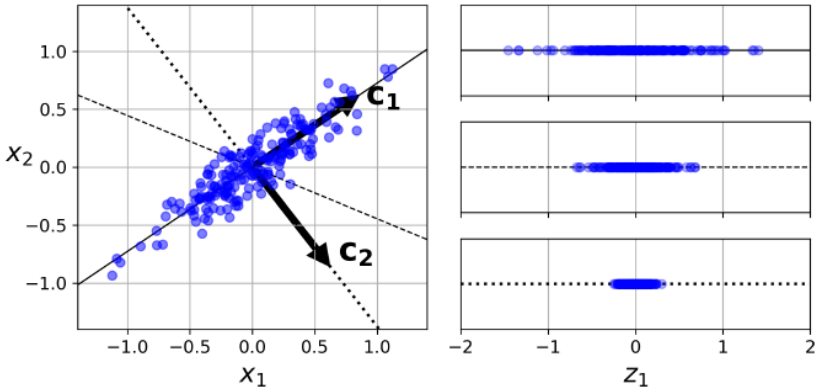


המופעים למישור), נקבל את מערך הנתונים הדו-ממדי החדש. זה עתה צמצמנו את הממדיות של מערך הנתונים מתלת-ממד ל2D. שימו לב שהצירים תואמים למאפיינים החדשים z1 ו-z2 (הקואורדינטות של ההקרנות במישור). עם זאת, Projection היא לא תמיד הגישה הטובה ביותר להפחתת מימד. במקרים רבים תת המרחב עשוי להתפתל ולהסתובב, כמו במערך הנתונים המפורסם Swiss roll toy dataset, שבו ההיטל של הנקודות למישור של 2D לא יעיד באמת על התנהגות הגרף.

## PCA

3. השיטה השנייה שנלמד היא PCA - Principal Component Analysis - ניתוח רכיבים ראשיים. PCA הוא ללא ספק האלגוריתם הפופולרי ביותר להפחתת הממדיות. נניח ויש לי 3 משתנים – פיצ'רים – מישורים, איך אפשר להחליף את שלושת המשתנים האלה במשתנה אחד? אפשרות אחת היא להחליף את המשתנים בממוצע שלהם, אבל יש כאן בעיה – איבדנו אינפורמציה. PCA מנסה למצוא וקטור - הערך האופטימלי (ייקרא גם הערך העצמי) של כל מישור – כלומר פיצ'ר – כך ששקלול שלושת המשתנים יביא לכך שאיבוד האינפורמציה יהיה מינימלי.

הוא בעצם יבחר מישורים – לאו דווקא המישורים מהפיזור המקורי, תחילה הוא יזהה את המישורים בהם מרחק הנקודות הוא הקרוב ביותר לנתונים, ולאחר מכן הוא יקרין עליהם את הנתונים – כמו בprojection, ויבדוק מהי השונות של הנקודות על כל אחד מהם. המישורים




אותם הוא יבחר אלו מישורים אחרים, שמהווים קומבינציות שונות של הפיצ'רים המקוריים (למשל כמו ההפיכה של צבעוני לשחור לבן שלוקחים אחוזים שונים מכל צבע בRGB או למשל אם ניקח את הדאטה סט של הטיטאניק מישור חדש יכול להיות 0.2*מחלקה+0.5*גיל+0.3*שרד) כך שמישורים חדשים אלו ישמרו על כמות השונות המקסימלית, כדי שלא יטשטשו את ההבדלים שבין הנקודות ויגרום לאיבוד מידע. לאחר מכן אחרי שיבחר את הנקודות הוא ירכיב וקטור מסוים של שלושת המישורים, למעשה בכך הוא מקטין את כמות הפיצ'רים שנדרש להתבסס עליהם. דרך נוספת להצדיק את הבחירה הזו היא שהציר הוא שממזער את המרחק הממוצע בריבוע ($R^2$) בין מערך הנתונים המקורי וההקרנה שלו על הציר הזה.

4. עם זאת – ישנם כללים לשימוש בPCA, אם תחשבו עבור מישהו את הממוצע של גובהו ומנת המשכל שלו, מה תקבלו? איזה משמעות יש לזה? לכן הכלל הראשון: יש להשתמש ב-PCA אך ורק לשקלול משתנים המבטאים אספקטים שונים של אותו הדבר, כך שלשקלול שלהם תהיה משתמעות. הכלל השני – אין משמעות למשתנה שמי, כלומר אם סימנתי גבר ב0 ואישה ב1, אין לייחס להם ערכים מספריים אלא סיווגיים, לכן הכלל השני: PCA נועד למשתנים כמותיים בלבד. הכלל השלישי: תמיד יש לבדוק האם ניתן להתייחס לכל הנתונים כמקשה אחת, או שמא יש בנתונים תת קבוצות של נתונים, כאשר בכל אחת מהן יש קשרים שונים בין המשתנים. למשל

מדידת אספקטים רפואיים אצל אנשים, כך שנחלק בין נשים לגברים שהנתונים הפיזיולוגיים והתגובות הפיזיולוגיות השונות אינן אותו דבר.

## Explained variance ratio

5. מעבר להורדת המישורים אני מעוניין לקבל הסבר ליחס של השונות בין המישורים שנלקחו מתוך שאיפה להגיע לפיזור שונות של 100%. הפעולה הזו מתבצעת ע"י explained variance ratio. זה אומר לך ש-84.2% מהשונות של הדאטה סט נמצאים לאורך המישור הראשון, ו-14.6% נמצאים לאורך המישור השני. זה משאיר פחות מ-1.2% למישור השלישי, אותו הורדנו, כך שסביר להניח שהוא נושא מעט מידע.

```
>>> pca.explained_variance_ratio_
array([0.84248607, 0.14631839])
```

6. מה זה אומר לי שונות של 100%? המושג קורלציה מתאר את המצב של עד כמה הדברים משתנים יחד – כלומר אם יש קורלציה של 100%, כלומר 1, זה מצב בו כאשר הראשון עולה גם השני עולה ולהפך, בעוד ש1- זה קורלציה הפוכה, כשהראשון יעלה השני ירד ולהפך. קורלציה 0 זה שום קשר אחד על השני. חשוב לדעת – הקשר בין קורלציה לשונות זה שקורלציה זה בעצם שונות מנורמלת. <u>קורלציה של 100% אומר ש100% מהשונות בפיצ'ר מסוים מוסבר ע"י המשתנה הזה.</u>

למשל התפלגות הטיפים במסעדה ביום מסוים הוא בין 10 ל100 דולר, ושעות הפתיחה של המסעדה זה מהבוקר לערב. קורלציה 1 בין הפיצ'רים הללו תסביר שיש התאמה מדויקת בין השעה לבין גובה הטיפ. בעוד שקורלציה נמוכה תצביע על כך שלא העובדה שהיום הלך והתקדם הוא זה שהשפיע על גובה הטיפ.

נניח שהקורלציה בין הטיפ למין במלצר יוצא 0.6, <u>נפעיל על הקורלציה $R^2$</u> – ממוצע הפרשי הנקודות מהפונקציה הלינארית בריבוע - ונקבל כי <u>ההסבר</u> של השונות הוא 0.36. כלומר 36% אחוז מתוך טווח התפלגות השונות קשורים למין המלצר. יתר ה64% יוסבר ע"י משתנים אחרים.

7. דוגמא לאופן המימוש בקוד:

```
from sklearn.decomposition import PCA

pca = PCA(n_components = 2)
X2D = pca.fit_transform(X)
```

כאשר אני כותב n_components = 2 הכוונה היא להפעיל את הPCA כך שנבחר רק את 2 הפיצ'רים – מישורים – שמספקים לי את אחוז יחס השונות הגבוה ביותר. (חשוב לשים לב שלמרות שהם ה2 הראשונים יכול להיות שיחד הם מספקים מענה רק ל60% מהדאטה).

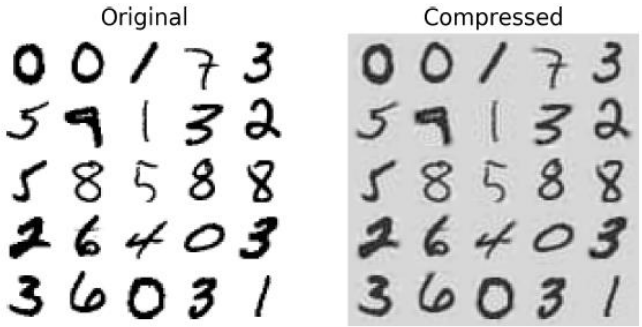
אופציה אחרת היא לרשום n_components = 0.95 כלומר תביא לי כמות פיצ'רים שיחד מספקים הסבר ל95% מהשונות.

8. הקוד הבא מחשב את ה-PCA מבלי להפחית את הממדיות, ולאחר מכן מחשב את המספר המינימלי של מימדים הנדרשים כדי לשמר 95% מהשונות של trainn בדאטה סט:

```
pca = PCA()
pca.fit(X_train)
cumsum = np.cumsum(pca.explained_variance_ratio_)
d = np.argmax(cumsum >= 0.95) + 1
```

9. בדאטה סט של הMNIST digits – אנו נדרשים לזהות את המספרים מבין אלפי תמונות בגודל של 28*28 פיקסלים, כלומר בסה"כ 784 פיצ'רים, שלעבור עליהם יהיה אולי מדויק אבל לא יעיל בכלל, כי כאמור ודאי שיש פיקסלים שמשפיעים מאוד על ההבנה וכאלו שלא. הורדת המימדים היא למעשה לקחת הרבה פחות פיצ'רים, בצורה כזו שהexplained variance יישאר גבוה. אם נפעיל עליו את הPCA כך שיוסבר 95% מהשונות נוכל לוותר על רוב הפיקסלים ולהישאר עם 154 הפיקסלים המשמעותיים! כמובן ישנו הבדל מבחינה חזותית בין המספרים הרגילים לבין אלו הדחוסים, אך כמו שגם ניתן לראות – הם לא מפריעים לנו בזיהוי של אילו מספרים אלו.

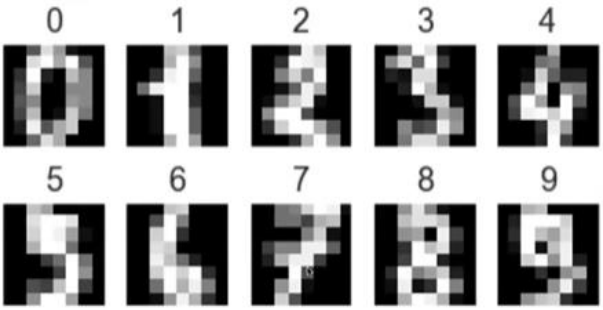
```
pca = PCA(n_components = 154)
X_reduced = pca.fit_transform(X_train)
X_recovered = pca.inverse_transform(X_reduced)
```

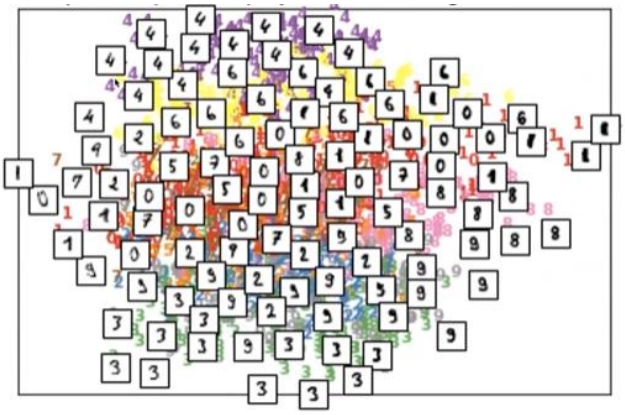



## שיעור 12

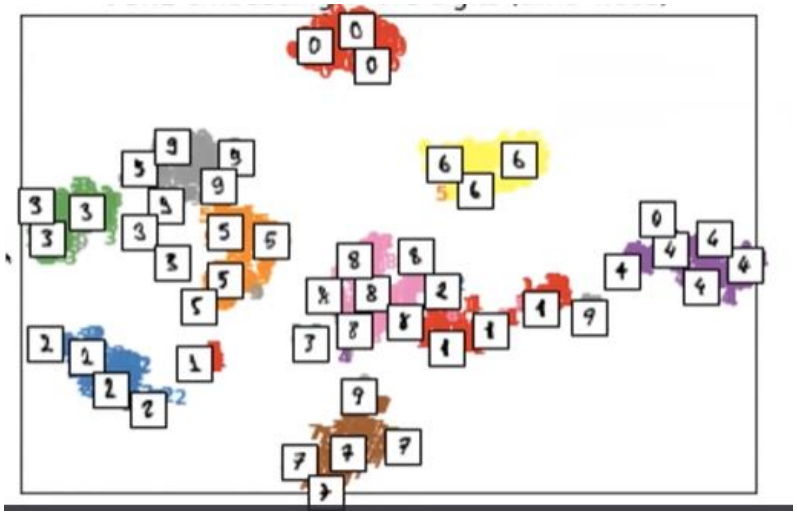
### T-SNE

1. גם כאשר נתונה לנו תמונה של בסך הכל 8 על 8 פיקסלים – עדיין מדובר על 64 מימדים, שמהם אני אצטרך לסנן ולעבוד עליהם על מנת למצוא את הרלוונטיים ביותר. לכן כאשר מדובר ביותר פיקסלים, כמו שיש בתמונה רגילה – יתקבלו אלפי מימדים. ראינו את האלגוריתם של PCA שלמעשה מוריד מימדים ע"י יצירת קומבינציה לינארית של המימדים כך שנוכל להוריד מימדים אבל עדיין לשמור על השונות של האובייקטים כך שיהיו פרוסים בגרף חדש לפי אשכולות – קלסטרים. אנו יכולים לראות בציור איך הוא חילק, ונראה כי 8,9,5 קרובים אחד לשני בגלל הדמיון ביניהם, בשונה מ3 ו4 שנמצאים ב2 קצוות שונים. האם אני יודע לומר משהו על ציר הx? לא, וזה גם לא משנה לי, PCA עוזר להוריד מימדים ולהגיד מה דומה למה, אבל לא לסווג לקלאסים, את זה המודל יצטרך לעשות בהמשך. הציור למעלה כן מספק לי מושג שיש איזשהו דפוס – pattern – אבל הויזואליזציה לא מאובחנת באופן מהותי, למשל אנו יכולים למצוא 6 גם בפינה הימנית העליונה ליד 0 ו1, אבל גם בקצה השני, כמעט פינה השמאלית ליד 2,5. לכן PCA לא עוזר לי בויזואליזציה.



2. T-sne היא מטודה שלרוב אנשים טועים ואומרים שהיא מורידה מימדים, אך זה לא מדויק. מה שהיא עושה זה שהיא לוקחת מימד גבוה ומציירת אותו במימד נמוך יותר. כלומר לא מורידה מימדים אלא מייצרת לי ויזואליזציה קליטה יותר. נסתכל על התמונה הזו של התוצאה מTSNE, אמנם יש אי דיוקים, למשל שיחד עם קבוצת ה7 יש גם 9 אחד, אבל הדברים הרבה יותר ברורים, כי אכן יש דמיון בין 7 ל9.



3. השאיפה של t-sne זה לייצר לאחר השימוש בPCA חלוקה כך שבתוך כל קלאס תהיה מינימום שונות, אבל בין קלאסים שונים – מקסימום שונות.
4. סרטון שממחיש יפה את הרעיון של האלגוריתם -
https://www.youtube.com/watch?v=NEaUSP4YerM
מחברת גיטהאב -
https://github.com/shivanichander/tSNE/blob/master/Code/tSNE%20Code.ipynb