ניב לוי – 314628165

Assignment 1: Difference between Gradient Descent, Stochastic Gradient Descent, and Mini-Batch Gradient Descent

במטלה הראשונה של הקורס התבקשנו להציג את ההבדלים בין שלושת הדרכים הנפוצות לחישוב החיתוך והשיפוע של הרגרסיה ליניארית תחת שיטת Gradient Descent

ישנם עוד מספר שיטות או אמדים על מנת לחשב את החיתוך והשיפוע של רגרסיה ליניארית :

- Ordinary Least Squares (OLS),
  כאשר :

$$a = (\Sigma y * \Sigma x^2 - \Sigma x * \Sigma xy) / (n * \Sigma x^2 - (\Sigma x)^2)$$

$$b = (n * \Sigma xy - \Sigma x * \Sigma y) / (n * \Sigma x^2 - (\Sigma x)^2)$$

- Matrix method.

תחילה, על מנת בכלל להבין כיצד לגשת למטלה, חיפשתי מקורות/סרטונים שבהם שלבי האלגוריתם מוצגים שלב אחר שלב. לאחר שהבנתי איך האלגוריתם בנוי ניגשתי לכתוב את התוכנית בפיתון. הנחות המודל היו פשוטות :

1. לקחת את הנגזרת החלקית של הLoss Function בשביל החותך ובשביל השיפוע.
2. בכל צעד (איטרציה) להבין אילו ערכים אנו לוקחים (כל הדאטה, נקודה בודדת או מדגם).
3. להציב אותם בנגזרות החלקיות.
4. לחשב את הSteps Size עבור השיפוע והחותך (אחד בעבור כל אחד)
5. לחשב את החותך והשיפוע החדשים.
6. אם הSteps Size קטנים מספיק או שהגעתי למספר הצעדים המקסימאלי, להוציא פלט. אחרת לחזור על שלבים 3-6.

לאחר מכן, הרצתי מודל מובנה של Sklearn על מנת להעריך לאיזה ערכים אני צריך לכוון להגיע :

```python
from sklearn.linear_model import LinearRegression
import numpy as np

# Create some sample data
x = df['x'].values.reshape((-1, 1))
y = df['y'].values

# Create a linear regression object
model = LinearRegression()

# Fit the model to the data
model.fit(x, y)

# Print the intercept and slope
print('Intercept:', model.intercept_)
print('Slope:', model.coef_[0])
```

```
Intercept: 24.43095571592852
Slope: 5.113097853269347
```

כאשר ניגשתי לבנות את המודל חיפשתי ספרייה שתוכל לתת לי אף אפשרות לעבוד עם נגזרות חלקיות ולאחר שמצאתי ספרייה בשם "sympy" התחלתי לכתוב את התוכנית :

לצערי, למרות שלדעתי התוכנית ברורה יותר לקורא, זמן הריצה שלה היה כל כך ארוך שלא הצלחתי אפילו להוציא פלט בשביל לבדוק את תקינותה. לאחר בדיקה באינטרנט הבנתי שבשביל לקצר זמני ריצה, עליי לעבוד בחישובים וקטורים ובפרט לעבוד עם ספריית "Numpy"

```python
def gradient_descent(df, intercept=0, slope=0, min_step_size=0.0001, steps=1000):
    learning_rate = 0.0001
    step_size = 1
    num_steps = 0
    for i in range (steps) :
        d_SSE_intercept = 0
        d_SSE_slope = 0
        for line in range(len(df)):
            x = df.loc[line, "x"]
            y = df.loc[line, "y"]
            a, b = sympy.symbols('a b')
            # sum of squared residuals = (y-Y_pred)
            # Y_Pred = intercept(a) + slope(b)*x
            loss_functions = (y-(a+b*x))**2

            # Calculate partial derivatives
            df_da = sympy.diff(loss_functions, a)
            df_db = sympy.diff(loss_functions, b)

            # Evaluate the derivatives at a specific point
            d_SSE_intercept += float(df_da.evalf(subs={a: intercept, b: slope})/len(df))
            d_SSE_slope += float(df_db.evalf(subs={a: intercept, b: slope})/len(df))

        # Updating
        step_size_intercept = d_SSE_intercept * learning_rate
        intercept = intercept - step_size_intercept

        step_size_slope = d_SSE_slope * learning_rate
        slope = slope - step_size_slope

        num_steps += 1

        if abs(step_size_intercept) < min_step_size and abs(step_size_slope) < min_step_size:
            return intercept, slope, num_steps

    return (intercept, slope, num_steps)
```

<u>מודל הGD :</u>

התחלתי לבצע שיפורים במודל ולאחר ביצוע הטרנספורמציה למודל לעבודה בצורה וקטורית הגעתי לתוצאה :

```python
def gradient_descent(df, intercept=0, slope=0, min_step_size=0.0001, steps=1000):
    #Save our Parameters :
    inter_values = []
    slope_values = []
    loss_func_values = []


    # Set the learning rate
    learning_rate = 0.0001
    steps_count = 0 # Initialize the step count

    # Get the x and y values from the input dataframe
    x = df['x'].values
    y = df['y'].values

    # Loop through the specified number of steps
    for i in range(steps):
        # Calculate the predicted y values based on the current intercept and slope
        Y_pred = intercept + slope * x

        # Calculate the partial derivative of the SSE with respect to the intercept and slope
        loss_func = np.sum((y - Y_pred)**2)/len(x)
        d_SSE_intercept = -2 * np.sum(y - Y_pred)/len(x)
        d_SSE_slope = -2 * np.sum(x * (y - Y_pred))/len(x)

        # Calculate the step size for the intercept and slope
        step_size_intercept = d_SSE_intercept * learning_rate
        step_size_slope = d_SSE_slope * learning_rate

        # Update the intercept and slope based on the step sizes
        intercept -= step_size_intercept
        slope -= step_size_slope

        steps_count += 1

        #Save the parameters in order to generate the plots
        inter_values.append(intercept)
        slope_values.append(slope)
        loss_func_values.append(loss_func)


        # Check if the step sizes are smaller than the specified minimum step size
        # If the step sizes are small enough, return the current intercept, slope, and step count
        # If the maximum number of steps is reached, return the current intercept, slope, and step count

        if abs(step_size_intercept) < min_step_size and abs(step_size_slope) < min_step_size:
            return intercept, slope, steps_count, inter_values, slope_values, loss_func_values

    return intercept, slope, steps_count, inter_values, slope_values, loss_func_values

gradients_values = gradient_descent(df)
print(f'intercept: {gradients_values[0]}, Slope: {gradients_values[1]}, Steps : {gradients_values[2]}')
```

```
intercept: 1.2566132284026708, Slope: 5.40272148418424, Steps : 1000
```

ניב לוי – 314628165

ראיתי שהמודל משתמש בכל מספר הצעדים שהוגדרו לו, לכן ניסיתי להעלות את מספר הצעדים במספר סדרי גודל עד אשר המודל יממש את הדיוק בתנאי יציאה :

```python
def gradient_descent(df, intercept=0, slope=0, min_step_size=0.0001, steps=100000):
    #Save our Parameters :
    inter_values = []
    slope_values = []
    loss_func_values = []


    # Set the learning rate
    learning_rate = 0.0001
    steps_count = 0 # Initialize the step count

    # Get the x and y values from the input dataframe
    x = df['x'].values
    y = df['y'].values

    # Loop through the specified number of steps
    for i in range(steps):
        # Calculate the predicted y values based on the current intercept and slope
        Y_pred = intercept + slope * x

        # Calculate the partial derivative of the SSE with respect to the intercept and slope
        loss_func = np.sum((y - Y_pred)**2)/len(x)
        d_SSE_intercept = -2 * np.sum(y - Y_pred)/len(x)
        d_SSE_slope = -2 * np.sum(x * (y - Y_pred))/len(x)

        # Calculate the step size for the intercept and slope
        step_size_intercept = d_SSE_intercept * learning_rate
        step_size_slope = d_SSE_slope * learning_rate

        # Update the intercept and slope based on the step sizes
        intercept -= step_size_intercept
        slope -= step_size_slope

        steps_count += 1

        #Save the parameters in order to generate the plots
        inter_values.append(intercept)
        slope_values.append(slope)
        loss_func_values.append(loss_func)


        # Check if the step sizes are smaller than the specified minimum step size
        # If the step sizes are small enough, return the current intercept, slope, and step count
        # If the maximum number of steps is reached, return the current intercept, slope, and step count

        if abs(step_size_intercept) < min_step_size and abs(step_size_slope) < min_step_size:
            return intercept, slope, steps_count, inter_values, slope_values, loss_func_values

    return intercept, slope, steps_count, inter_values, slope_values, loss_func_values

gradients_values = gradient_descent(df)
print(f'intercept: {gradients_values[0]}, Slope: {gradients_values[1]}, Steps : {gradients_values[2]}')
```

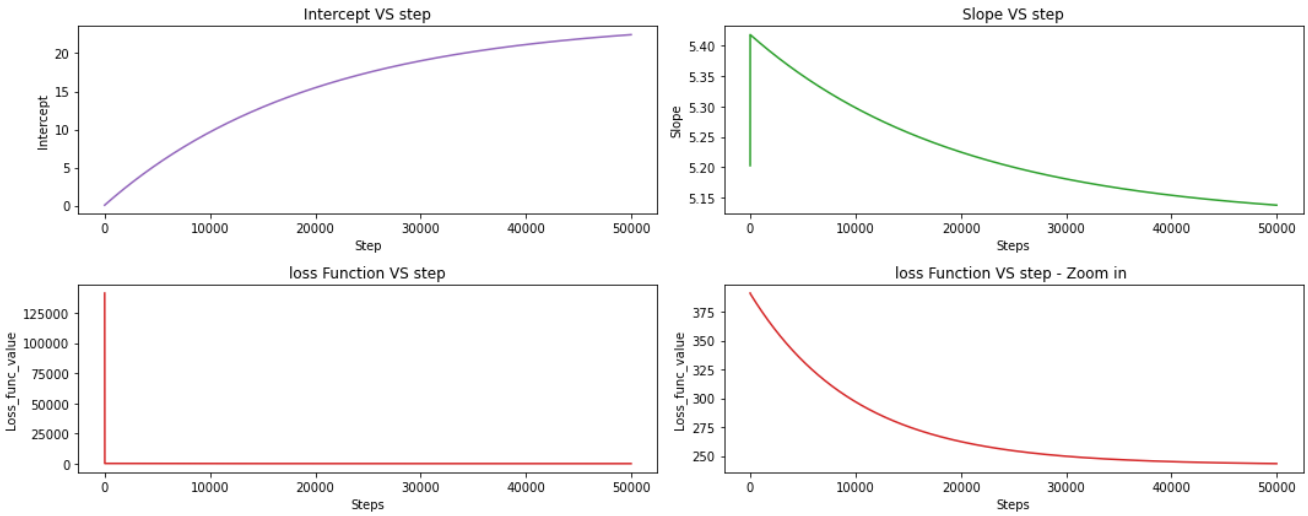
```
intercept: 22.432303718153996, Slope: 5.138076203737859, Steps : 49982
```

כפי שניתן לראות, התוצאות קרובות מאוד למודל המובנה של Sklearn.

ניב לוי – 314628165

<u>פלט GD:</u>

בחרתי להציג פלטים של מודל השני שהוצג למעלה, לאחר 49982 צעדים – שקיים את תנאי היציאה.



- Intercept VS Step – הנק' חיתוך גדלה בצורה שדומה מאוד לפונקציית שורש. מכיוון שכיילנו את הערך הראשון להיות 0, אנחנו רואים יציאה מראשית הצירים ונטייה של הפונקציה להיות אסימפטוטית לאחר 50000 צעדים.
- Slope VS Step – המודל בכמה ערכים הראשונים נותן לפונקציה שיפוע די גדול בסדר גודל של שאר השיפועים ולאחר מכן הוא דועך בצורה אסימפטוטית עד אשר מתקבע על שיפוע באזור 5.15.
- Loss Function VS Step – בשל הטעות המאוד גדולה יחסית בערכים הראשונים איננו יכולים לראות בצורה ראויה את התנהגות הפונקציה, לכן בחרתי להציג לוותר על ה2 ערכים הראשונים ולהציג יותר בקירוב בZoom in :
- Loss Function VS Step – Zoom in – בתצוגה הזאת של הגרף ניתן לראות התנהגות מונוטונית יורדת של הפונקציה שלאחר הרבה ריצות, ערך הLoss Function מתקבע קרוב ל243.

ניב לוי – 314628165

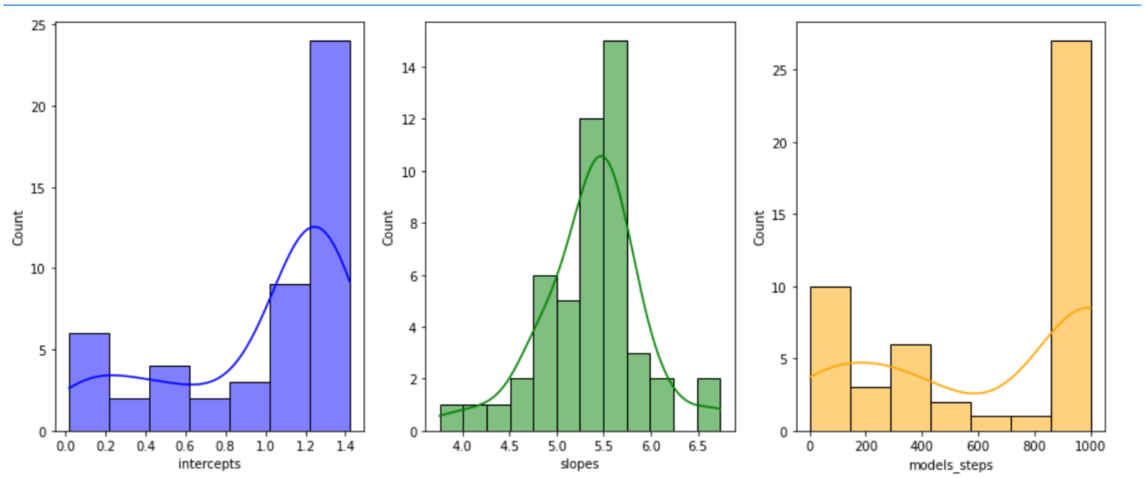
<u>מודל הSGD :</u>

ביצעתי מספר התאמות על מנת להתאים את המודל הקיים להיות מודל סטוכסטי. כידוע, במודל הסטוכסטי בכל צעד אנו בוחרים נקודה רנדומלית אחרת מהדאטה. השינויים שהייתי צריך לבצע בקוד השראינו קודם של הGD הם שבעת כל צעד כמות המידע שלנו תהיה רשומה בודדת. וההנחה השנייה היא שלקיחת הדאטה צריכה להתבצע בתוך הלולאה **כך שבכל צעד המודל יקח רשומה אחרת** ולא ירוץ בטעות על אותה רשומה כל הזמן.

```
def stoch_gradient_descent(df, intercept=0, slope=0, min_step_size=0.0001, steps=1000):
    #Save our Parameters :
    inter_values = []
    slope_values = []
    loss_func_values = []

    learning_rate = 0.0001
    steps_count = 0
    for i in range(steps):
        sample = df.sample(1) #We should take different sample point for each step
        x = sample['x'].values
        y = sample['y'].values
        Y_pred = intercept + slope * x

        loss_func = np.sum((y - Y_pred)**2)/len(x)
        d_SSE_intercept = -2 * (y - Y_pred)
        d_SSE_slope = -2 * (x * (y - Y_pred))

        step_size_intercept = d_SSE_intercept * learning_rate
        step_size_slope = d_SSE_slope * learning_rate

        intercept -= step_size_intercept
        slope -= step_size_slope

        steps_count += 1

        inter_values.append(float(intercept))
        slope_values.append(float(slope))
        loss_func_values.append(loss_func)

        if abs(step_size_intercept) < min_step_size and abs(step_size_slope) < min_step_size:
            return float(intercept), float(slope), steps_count, inter_values, slope_values, loss_func_values


    return float(intercept), float(slope), steps_count, inter_values, slope_values, loss_func_values

stoch_gradients_values = stoch_gradient_descent(df)
print(f'intercept: {stoch_gradients_values[0]}, Slope: {stoch_gradients_values[1]}, Steps : {stoch_gradients_values[2]}')
```

```
intercept: 0.142784316544361, Slope: 5.430280639244734, Steps : 41
```

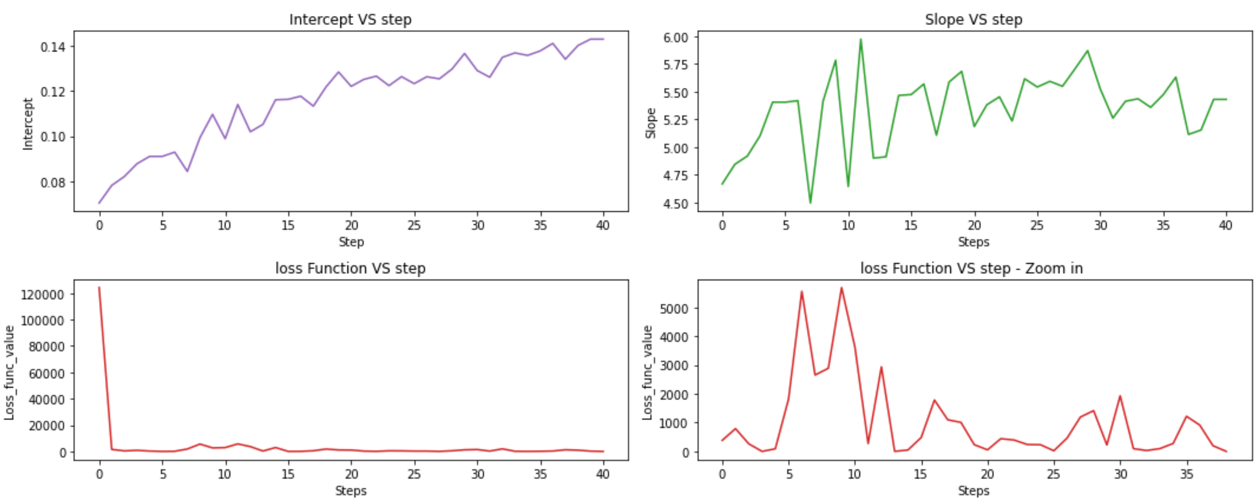
חשוב : מכיוון שהמודל אכן סטוכסטי, בכל הרצת מודל קיבלנו ערכים שונים לחיתוך, שיפוע ומספר הריצות. מאוד עניין אותי לראות את תוצאות המודל לאורך מספר מודלים ולכן בחנתי **50 מודלים שונים על מנת להשיג טווח ערכים** :



כפי שניתן לראות, קיים שוני בין תוצאות המודלים וערכיהם. לפחות ממה שנראה פה, המודל הסטוכסטי פחות עקבי בתוצאות ובביצועים שלו.

<u>פלטים :</u>

ניתן לראות כי המודל האחרון שהרצתי, נעצר לאחר 41 צעדים ולו נבצע ניתוח פלטים.



- Intercept VS Step – ניתן לראות פונקציה עולה, ללא מגמה ברורה. אני מניח שהשינוי התדיר בערכים נובע מסטוכסטיות המודל והשוני בין הנקודה הנלקחת בכל צעד.
- Slope VS Step – גם פה ניתן לראות מגמה לא ברורה שכנראה נובעת מסטוכסטיות המודל.
- Loss Function VS Step – שוב גם פה בשל הטעות המאוד גדולה יחסית בערכים הראשונים איננו יכולים לראות בצורה ראויה את התנהגות הפונקציה, לכן בחרתי להציג לוותר על ה2 ערכים הראשונים ולהציג יותר בקירוב.
- Loss Function VS Step – Zoom in – בשל הסטוכסטיות והידיעה כי המודל מחליף ערכים כל ריצה בצורה יחסית קיצונית, לא ניתן לראות מגמה ברורה בערכי הLoss Function, התוצאה מושפעת בכל צעד בהתאם לנקודה הנבחרת.

בסה"כ בנלמד, **המודל מושפע מערכים חריגים וקשה לו להתקבע על מגמה או תוצאה ברורה בהשוואה למודל הGD הרגיל**. לדעתי, המודל נותן תוצאות סבירות בסופו של דבר, אך אם הייתי רוצה להגיע לדיוק גבוה יותר, לא הייתי משתמש בו.

ניב לוי – 314628165

מודל Mini-Batch :

ביצעתי התאמות למודל הסטוכסטי כך שבמקום ערך בודד, כל צעד הוא יקח batch של 200 דגימות שונות באופן רנדומלי כל צעד (מספר הדגימות נתון לשינוי בפונקציה).

```
def mini_batch_gradient_descent(df, intercept=0, slope=0, min_step_size=0.0001, steps=1000, batch_size=200):
    #Save our Parameters :
    inter_values = []
    slope_values = []
    loss_func_values = []


    learning_rate = 0.0001
    steps_count = 0
    for i in range(steps):
        sample = df.sample(batch_size) #we should take different sample each step
        x = sample['x'].values
        y = sample['y'].values
        Y_pred = intercept + slope * x

        loss_func = np.sum((y - Y_pred)**2)/len(x)
        d_SSE_intercept = -2 * np.sum(y - Y_pred)/len(x)
        d_SSE_slope = -2 * np.sum(x * (y - Y_pred))/len(x)

        step_size_intercept = d_SSE_intercept * learning_rate
        step_size_slope = d_SSE_slope * learning_rate

        intercept -= step_size_intercept
        slope -= step_size_slope

        steps_count += 1
        inter_values.append(float(intercept))
        slope_values.append(float(slope))
        loss_func_values.append(loss_func)


        if abs(step_size_intercept) < min_step_size and abs(step_size_slope) < min_step_size:
            return float(intercept), float(slope), steps_count, inter_values, slope_values, loss_func_values


    return float(intercept), float(slope), steps_count, inter_values, slope_values, loss_func_values

mini_gradients_values = mini_batch_gradient_descent(df)
print(f'intercept: {mini_gradients_values[0]}, Slope: {mini_gradients_values[1]}, Steps : {mini_gradients_values[2]}')
```

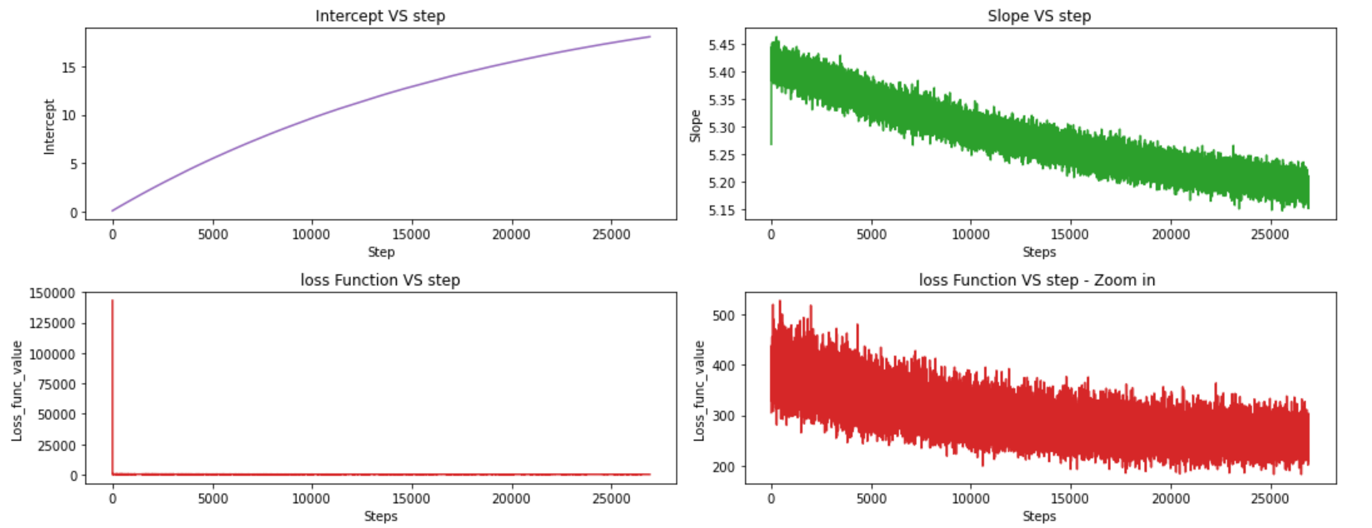
```
intercept: 1.2551456696839276, Slope: 5.397858017709886, Steps : 1000
```

גם פה לתוצאות די טובות, רציתי לבחון את תוצאות המודל ולהגדיל את מספר הצעדים עד אשר המודל יקיים את תנאי היציאה והגעתי לתוצאה הבאה :

```
def mini_batch_gradient_descent(df, intercept=0, slope=0, min_step_size=0.0001, steps=100000, batch_size=200):
    #Save our Parameters :
    inter_values = []
    slope_values = []
    loss_func_values = []


    learning_rate = 0.0001
    steps_count = 0
    for i in range(steps):
        sample = df.sample(batch_size) #we should take different sample each step
        x = sample['x'].values
        y = sample['y'].values
        Y_pred = intercept + slope * x

        loss_func = np.sum((y - Y_pred)**2)/len(x)
        d_SSE_intercept = -2 * np.sum(y - Y_pred)/len(x)
        d_SSE_slope = -2 * np.sum(x * (y - Y_pred))/len(x)

        step_size_intercept = d_SSE_intercept * learning_rate
        step_size_slope = d_SSE_slope * learning_rate

        intercept -= step_size_intercept
        slope -= step_size_slope

        steps_count += 1
        inter_values.append(float(intercept))
        slope_values.append(float(slope))
        loss_func_values.append(loss_func)


        if abs(step_size_intercept) < min_step_size and abs(step_size_slope) < min_step_size:
            return float(intercept), float(slope), steps_count, inter_values, slope_values, loss_func_values


    return float(intercept), float(slope), steps_count, inter_values, slope_values, loss_func_values

mini_gradients_values = mini_batch_gradient_descent(df)
print(f'intercept: {mini_gradients_values[0]}, Slope: {mini_gradients_values[1]}, Steps : {mini_gradients_values[2]}')
```

```
intercept: 18.087493350348787, Slope: 5.19647015683164, Steps : 26919
```

<u>פלט של MBGD:</u>



- Intercept VS Step – גם פה, הנק' חיתוך גדלה בצורה שדומה מאוד לפונקציית שורש. מכיוון שביילנו את הערך הראשון להיות 0, אנחנו רואים יציאה מראשית הצירים.
- Slope VS Step – ניתן לראות זגזוג רב בין צעד לצעד, המון עליות וירידות. אם נסתכל על הנתונים לאורך זמן במבט על, ניתן לראות מגמה יורדת בערכי השיפוע למרות הזגזוג המתמשך.
- Loss Function VS Step – גם כאן, בשל הטעות המאוד גדולה יחסית בערכים הראשונים איננו יכולים לראות בצורה ראויה את התנהגות הפונקציה, לכן בחרתי להציג לוותר על ה2 ערכים הראשונים ולהציג יותר בקירוב.
- Loss Function VS Step – Zoom in – גם פה ניתן לראות זגזוג גדול מאוד בין צעד לצעד, אי יכולת להגיע לתבנית או מגמה מסויימת ברורה בין צעד לצד. בדומה לשיפוע, גם פה ניתן לראות מגמה כוללת בירידה למרות הזגזוג.

<u>לסיכום :</u>

- תוצאות המודל המובנה של Sklearn : (Slope:5.11, Intercept:24.33)
- תוצאות מודל הGD : (Slope:5.13, Intercept:22.43, Steps: 49982)
- תוצאות מודל הSGD : (Slope:5.43 , Intercept:0.14, Steps: 41)
- תוצאות מודל הMBGD : (Slope:5.19 , Intercept:18.08, Steps: 26919)

מבחינת דיוק מודל הGD היה הקרוב ביותר ולאחריו MBGD, אך צריך לחשוב האם במקרה הזה שווה להשקיע ביותר מ20000 צעדים יותר על מנת להגיע לדיוק מעט יותר טוב.