گزارش:

سوال 1 :

بخش اول:

-کاربرد اول(رفع نویز در عکس):

دسته‌ای از خودرمزگذارها که به آن‌ها رمزگذارهای رفع نویز نیز گفته می‌شود با یادگیری بازنمایی صحیحی از عکس سعی می‌کنند تا از دست این نویزها راحت شوند و در نهایت با ساختن دوباره‌ی عکس این بار بدون نویز این کار را انجام دهند در واقع برای آموزش این دسته از خودرمزگذارها معمولا از تعداد زیادی عکس در دو حالت با نویز و بدون نویز استفاده می‌کنند.

-کاربرد دوم(تولید عکس):

یک دسته‌ی خاصی از خودرمزگذارها که به آن‌ها خودرمزگذار متغیر گفته می‌شود به ما در تولید عکس‌های متنوع و جدیدی کمک کنند. در واقع این نوع مدل‌ها از دسته‌ی مدل سازنده می‌باشند که برای تولید عکس می‌توان از آن‌ها کمک گرفت. ایده‌ی کار به این ترتیب است که از روی تعدادی عکس ورودی برای مثال تعدادی عکس صورت انسان، می‌تواند عکس‌های جدیدی از صورت‌ها درست کند که بسیار طبیعی و شبیه به عکس‌های داده شده باشند ولی همچین آدم‌هایی یا همچین عکس‌هایی در حقیقت وجود ندارند.

بخش دوم:

احتمالاً مسأله اصلی این خطا در بازسازی ناکامل توسط اتوانکودر مربوط به اندازه فضای پنهان است. با افزایش اندازه فضای پنهان، اتوانکودر می‌تواند ویژگی‌های بیشتری از داده را نمایش دهد. اما اگر این اندازه زیاد شود، ممکن است اتوانکودر به جای یادگیری ویژگی های معنادار و اصلی، به یادگیری جزئیات نویزی یا غیرمعنادار بپردازد. تار بودن تصاویر بازسازی ممکن است به دلیل این باشد که اتوانکودر به جای نمایش دقیق ویژگی‌های اصلی تصویر، جزئیات نویزی یا بخش‌هایی از داده ورودی را در فضای پنهان نمایش داده است. این امر می‌تواند ناشی از اندازه نامناسب فضای پنهان یا نوع معماری اتوانکودر باشد.

بخش سوم:

آ)

```
from keras.datasets import mnist
(_, _), (test_images, _) = mnist.load_data()
test_images = test_images.reshape(test_images.shape[0], -1)
test_images = test_images.astype('float32') / 255.0

✓ 1.4s
```

ب)

```
import tensorflow as tf
autoencoder = tf.keras.models.load_model('mnist_AE.h5')
reconstructed_images = autoencoder.predict(test_images)
```

✓ 6.1s

```
313/313 [==============================] - 5s 14ms/step
```

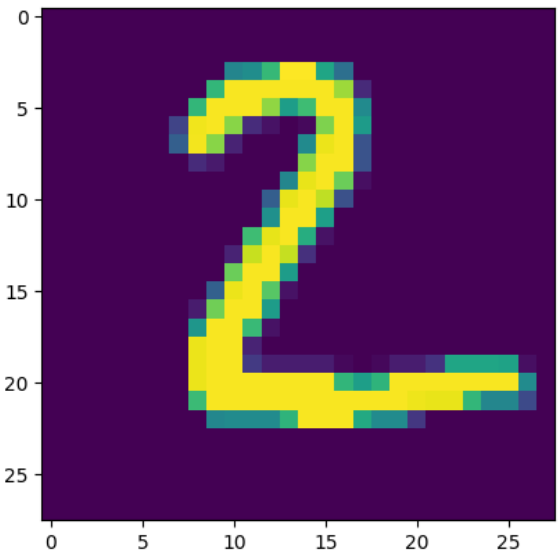
ج)

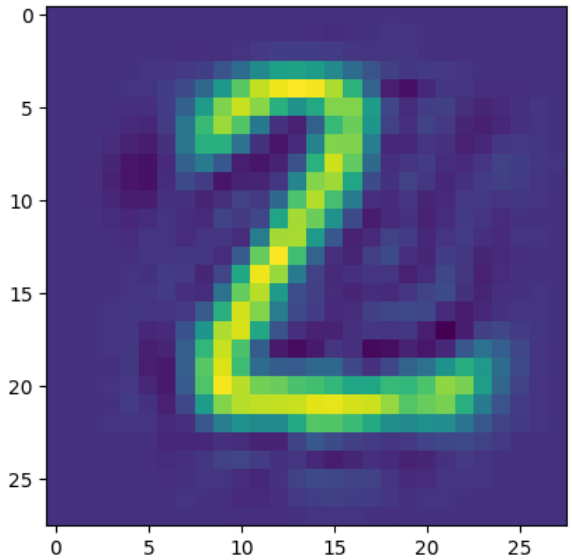
-فرم کلی کد:

```
import numpy as np
from matplotlib.pyplot import imshow
import matplotlib.pyplot as plt

test1=np.array(test_images[1])
test1 = test1.reshape((28,28))

imshow(test1)
```

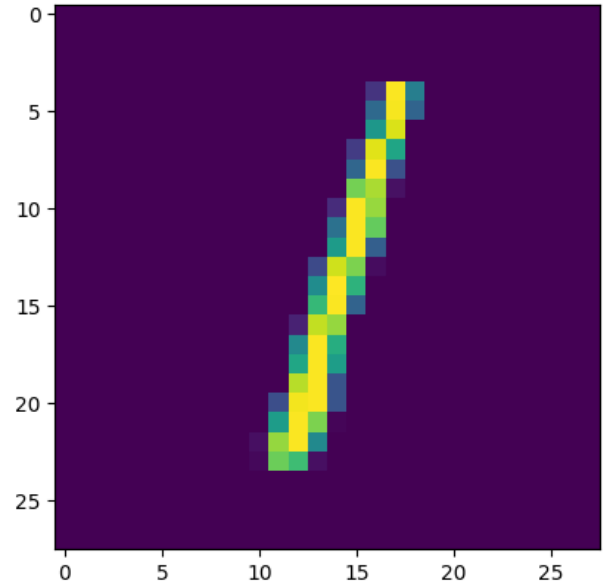
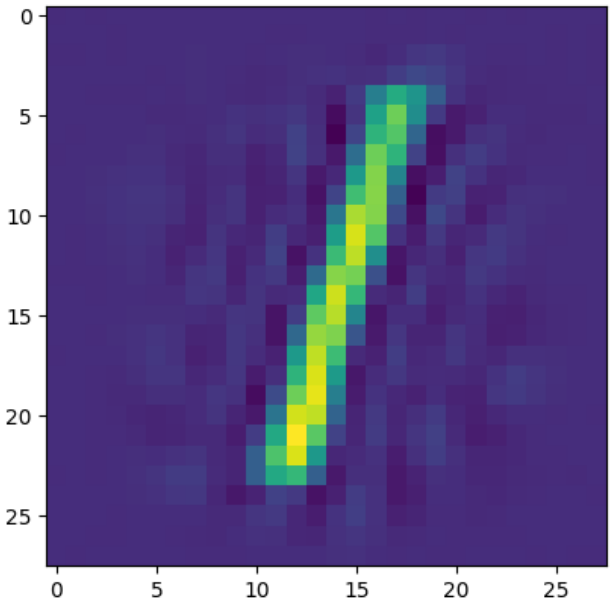
```
test1ec=np.array(reconstructed_images[1])
test1ec = test1ec.reshape((28,28))
imshow(test1ec)
```

-تست اول (i =۱) :

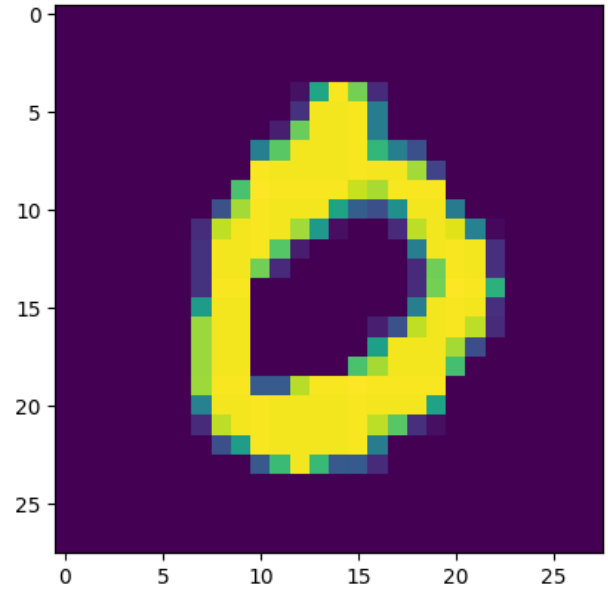
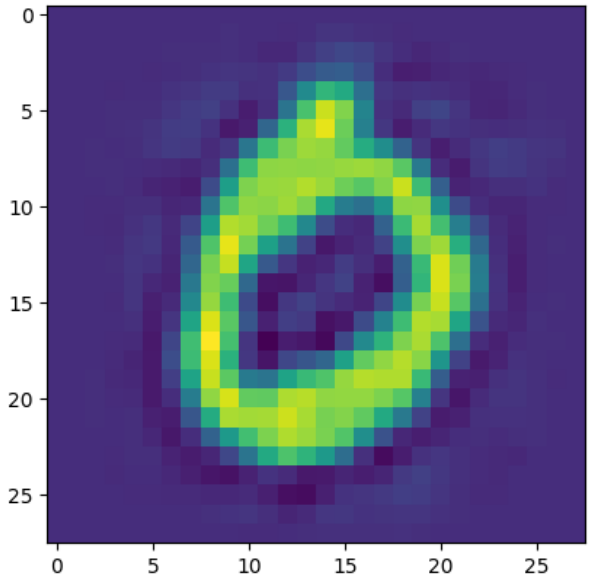




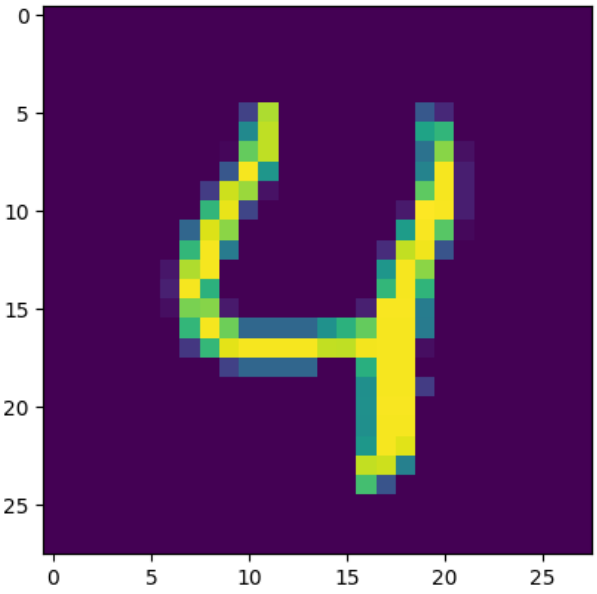
-تست دوم(i=2)

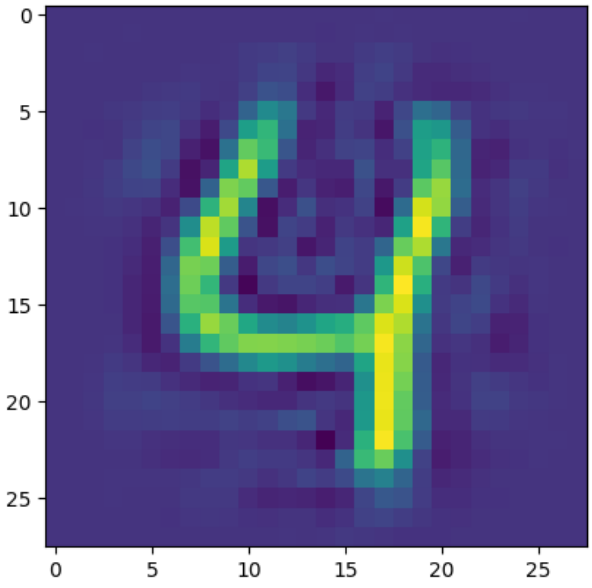



-تست سوم(i=3)

-تست چهارم(i=4)



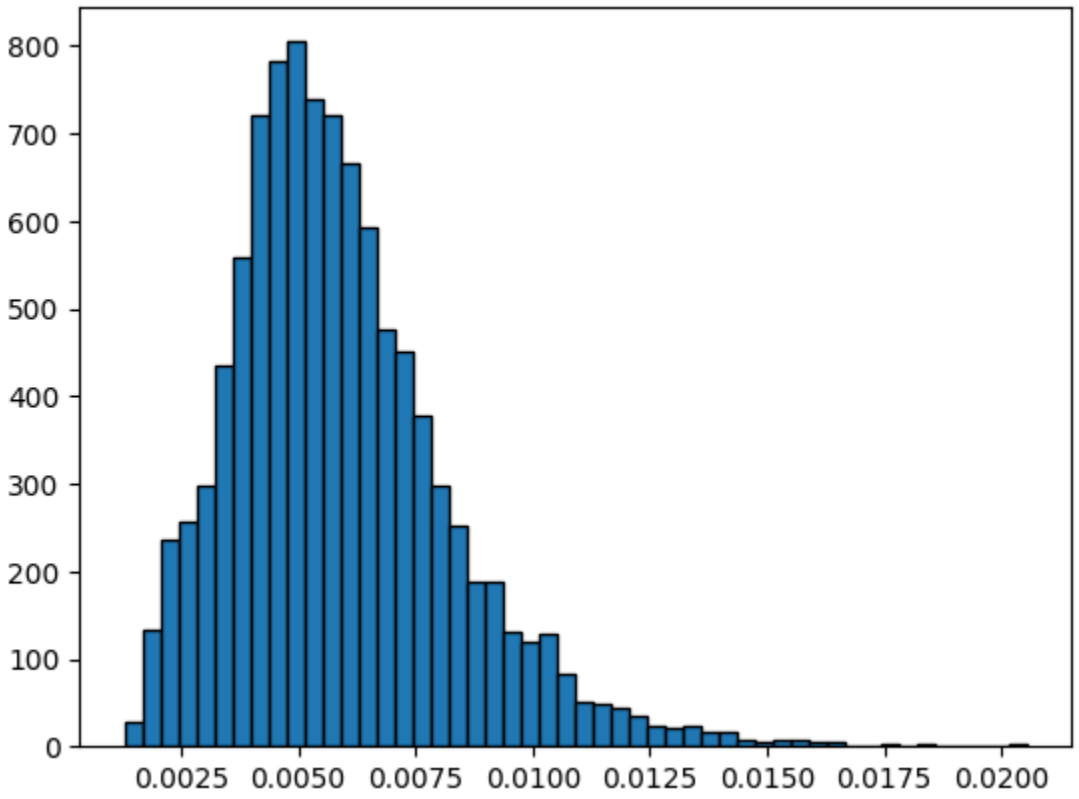


د)

```
def MSE(pixels,pixelsec):
    pixels=np.array(pixels)
    pixelsec=np.array(pixelsec)
    sum=0
    for i in range(pixels.size):
        sum=sum+((1/pixels.size)*(pixels[i]-pixelsec[i])**2)
    return sum

mse=np.array([])
for i in range(10000):
    mse=np.append(mse,MSE(test_images[i],reconstructed_images[i]))

mse
```

✓ 28.7s

```
plt.hist(mse,bins=50,edgecolor='black')
```



ه)

```
from scipy import stats
ks_statistic, p_value = stats.kstest(mse, cdf='norm', args=(np.average(mse), np.std(mse)))
p_value
```

✓ 0.0s

```
4.53929716593269e-43
```

- با توجه به اینکه مقدار p_value از 5% کمتر است پس میتوان گفت که فرض صفر رد میشود

سوال 2:

بخش اول:

-داده پرت یا به عبارتی نقطه دورافتاده نقطه ای است که مقدار ان(y) از سایر نقاط فاصله زیادی داشته باشد. این گونه داده ها به دلایل مختلفی از جمله اشتباه در جمع آوری داده ها، ابزار اندازه گیری نادرست، وجود افراد غیرمعمول در نمونه و... به وجود می آیند.
با در نظر گرفت این داده از انجا که y ان تفاوت زیادی دارد معادله خط رگرسیون ما دارای خطا میشود
-داده اهرمی داده ای است که x ان از بقیه داده ها دور افتاده است این داده حتی با اینکه y ان زیاد پرت نیست نسبت به بقیه داده ها اما از انجا که در قسمت تجمع داده ها قرار نگرفته تاثیر بیشتری نسبت به دیگر داده ها روی رگرسیون دارد یعنی خطای ان تاثیر بیشتری بر خطای رگرسیون دارد
-اگر در داده ها داده ای باشد که هم پرت باشد و هم اهرمی انگاه زیاد شدن همزمان x وy باعث میشود که تاثیر خطا کمتر شود یعنی خطای رگرسیون کاهش میابد

بخش دوم:

-ضریب تعیین نشان می‌دهد که چند درصد از تغییرات متغیر وابسته توسط متغیرهای مستقل توضیح داده می‌شود. به گونه‌ای که هدف آن یا پیش‌بینی خروجی‌های آینده است یا آزمودن فرضیه براساس سایر اطلاعات مرتبط.

Formula 2:

$$R^2 = 1 - \frac{\text{RSS}}{\text{TSS}}$$

Where:

- RSS = sum of squared residuals
- TSS = total sum of squares

$$RSS = \sum_{i=1}^{n} (y_i - f(x_i))^2$$

RSS = residual sum of squares
y_i = i^th value of the variable to be predicted
$f(x_i)$ = predicted value of y_i
n = upper limit of summation

$$\text{TSS} = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

TSS = total sum of squares
n = number of observations
y_i = value in a sample
$\bar{y}$ = mean value of a sample

بخش سوم:

-فرم کلی کد برای هر چهار حالت (برای عدم تکرار):

```
import matplotlib.pyplot as plt
import numpy as np
x_value = [-2.3, -1.1, 0.5, 3.2, 4.0, 6.7, 10.3, 11.5]
y_value = [ -9.6, -4.9, -4.1, 2.7, 5.9, 10.8, 18.9, 20.5]
plt.scatter(x_value, y_value, color='blue')
plt.xlabel('x_value')
plt.ylabel('y_value')
```

✓ 0.2s

-محاسبه رگرسیون:

فرمول رگرسیون: y=ax+b

```
x = x_value
y = y_value
def avg(data):
    my_temp = 0
    for i in range (0, len(x_value)):
        my_temp = my_temp + data[i]
    return(my_temp/len(x_value))

avg_x = avg(x)
avg_y = avg(y)

def areg():
    a_sorat = 0
    a_makhraj = 0
    for i in range (0, len(x_value)):
        a_sorat = a_sorat + ((x[i] - avg_x)*(y[i] - avg_y))
        a_makhraj = a_makhraj + ((x[i] - avg_x)**2)
    return(a_sorat/a_makhraj)

a = areg()
b = avg_y - (a * avg_x)

def regression(x):
    return(b + (a * x))


y_regression = []
for i in range(0, len(x_value)):
    y_regression.append(regression(x_value[i]))
```

✓ 0.0s

رسم نمودار:

```
plt.scatter(x_value, y_value, color='blue')
plt.scatter(x_value, y_regression, color='red')
plt.xlabel('x_value')
plt.ylabel('y_value')
plt.plot(x_value, y_regression, color='orange')
```

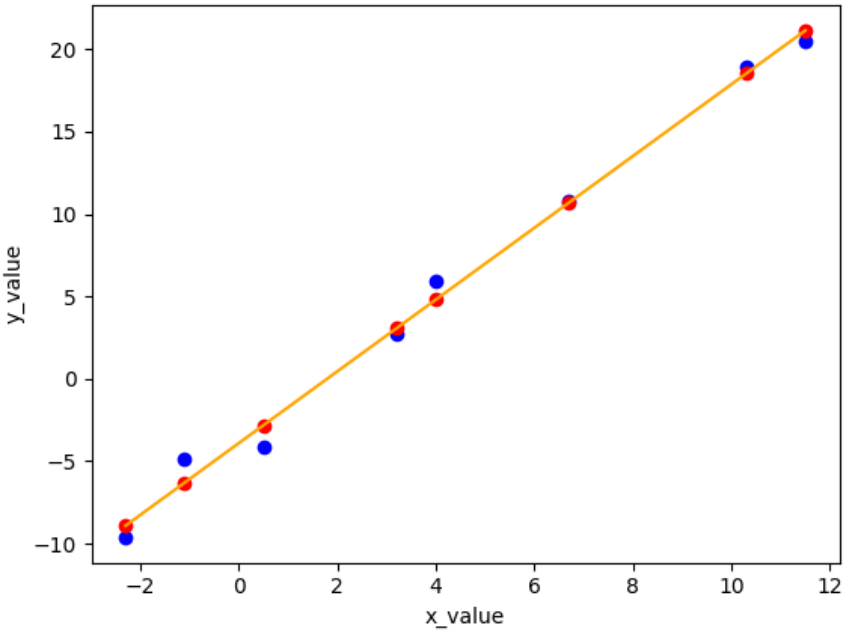
✓ 0.4s

-محاسبه ضریب تعیین:

```
rss = 0
for i in range(0, len(x_value)):
    rss=rss+(y_value[i]-regression(x_value[i]))**2

tss=0
for i in range(0, len(x_value)):
    tss=tss+(y_value[i]-np.average(y_value))**2

print(f"the R2 value is: {1-(rss/tss)}")
```

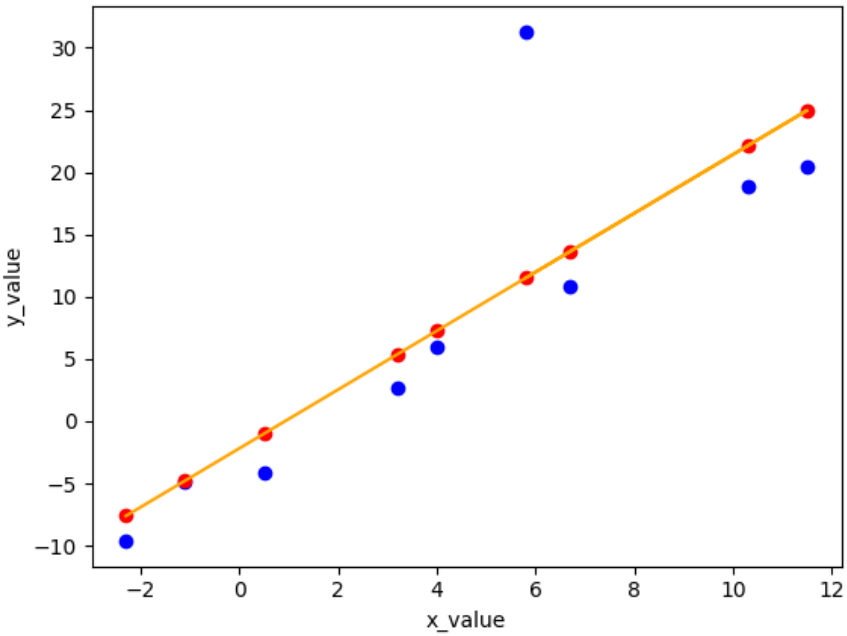
✓ 0.0s

-رگرسیون بر پایه هشت داده اصلی:



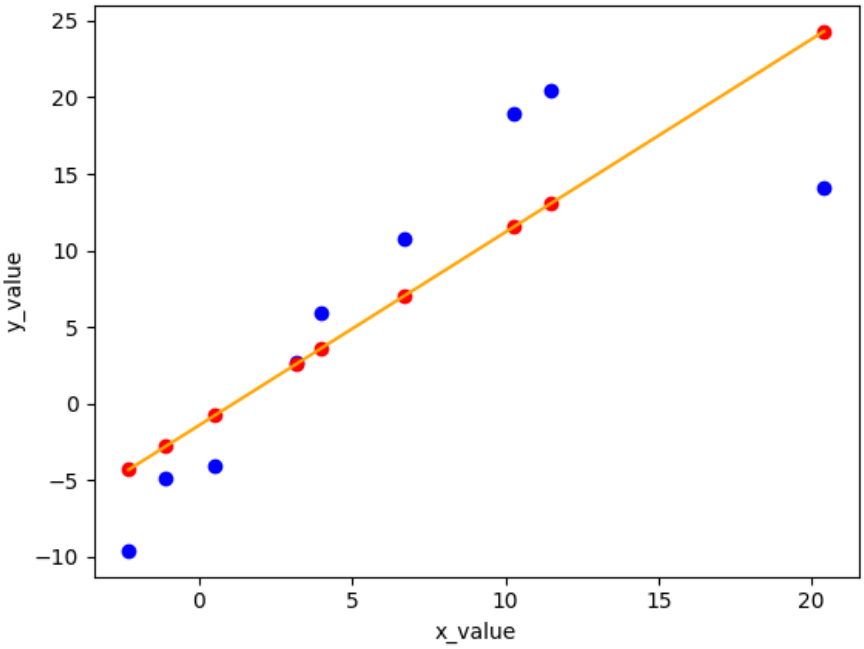

```
the R2 value is: 0.9931142293628097
```

-رگرسیون بر پایه هشت داده اصلی به اضافه نقطه دور افتاده:



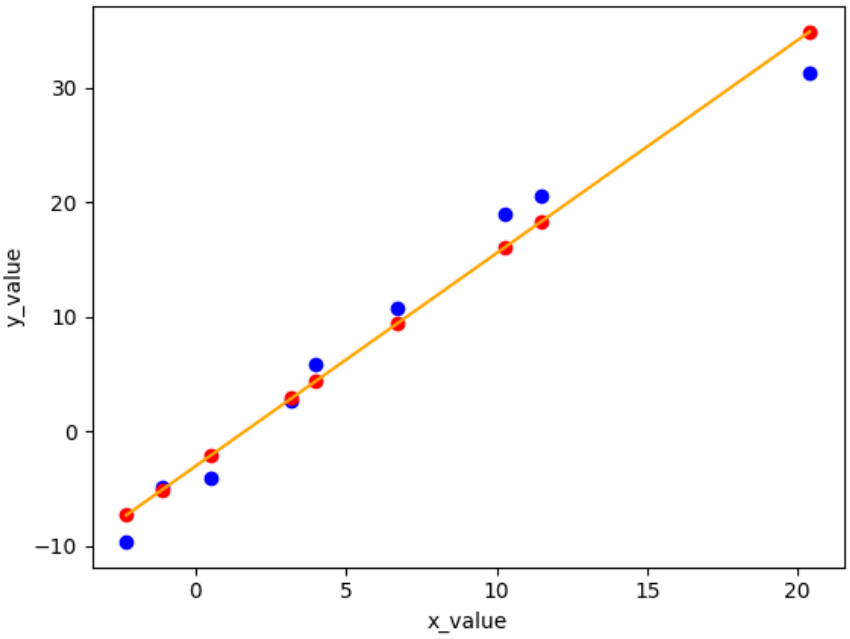

```
the R2 value is: 0.6943381680789323
```

-رگرسیون بر پایه هشت داده اصلی به اضافه نقطه اهرمی:




```
the R2 value is: 0.7069879724740986
```

- رگرسیون بر پایه هشت داده اصلی به اضافه نقطه دور افتاده-اهرمی:




```
the R2 value is: 0.9738367949787371
```

بخش چهارم:

-یک راه حذف داده های پرت و اهرمی به کمک نمودار جعبه‌ای و یا روش های دیگر است که با این کار خطای ما کاهش پیدا میکند و روش دیگر هم این است که مانند همین مثال نقاط پرت و اهرمی را با هم merge کنیم

سوال ۳:

بخش ۱:

-باتوجه به اینکه در کنار ستون های dribbling و pace ستون های دیگری در دیتاست وجود دارند که این ویژگی را در category های مختلف برسی میکند پس میتوان به جای not a number های میانگین category های مختلف ان ویژگی را قرار دهیم

```
import pandas as pd
import numpy
df = pd.read_csv('FIFA2020.csv', encoding = "ISO-8859-1")
```

-جایگزینی dribbling:

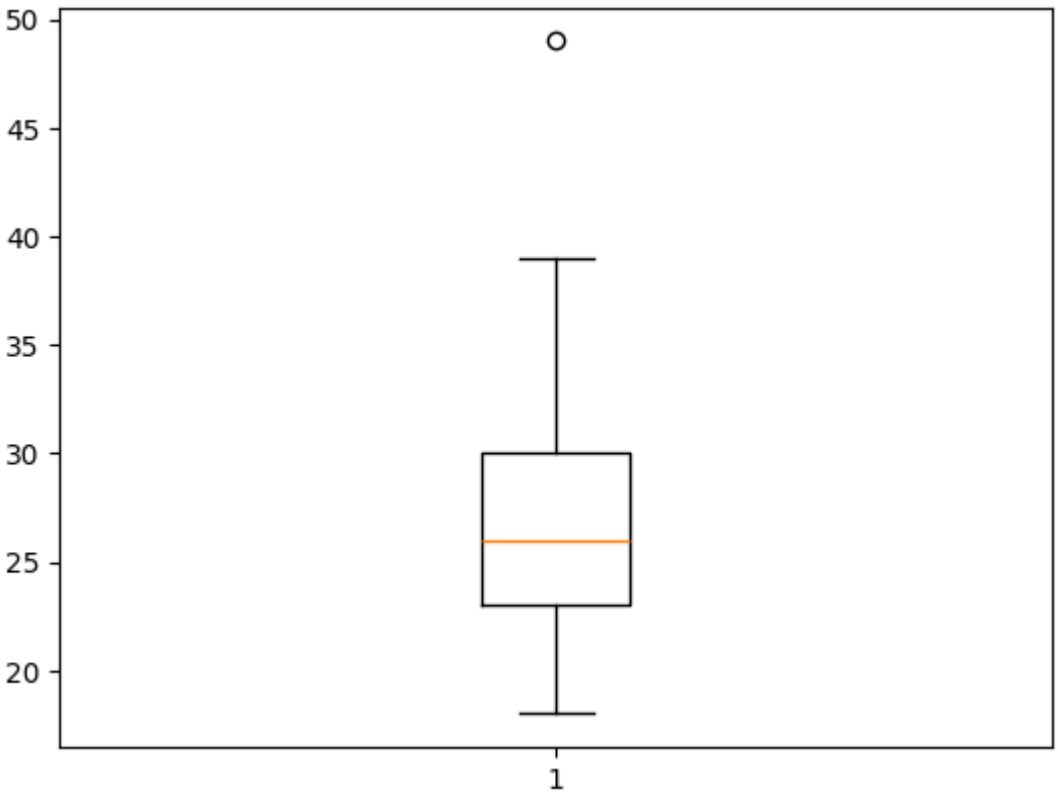
```
for i in range(len(df)):
    if numpy.isnan(df['dribbling'][i]):
        df['dribbling'][i]=numpy.average(df.loc[i,['drib_agility','drib_balance','drib_reactions','drib_ball_control','drib_dribbling','drib_composure']])
```

-جایگزینی pace:

```
for i in range(len(df)):
    if numpy.isnan(df['pace'][i]):
        df['pace'][i]=numpy.average(df.loc[i, ['pace_acceleration', 'pace_sprint_speed']])
```

بخش ۲:

```
import matplotlib.pyplot as plt
import numpy as np

# Creating dataset
np.random.seed(10)

data = np.random.choice(df['age'], 100, replace=False)

print(f"Max is: {np.max(data)}")
print(f"Min is: {np.min(data)}")
print(f"Q1 is: {np.percentile(data,25)}")
print(f"Q2 is: {np.percentile(data,50)}")
print(f"Q3 is: {np.percentile(data,75)}")
# Creating plot
plt.boxplot(data, autorange=True)
# show plot
plt.show()
```




```
Max is: 49
Min is: 18
Q1 is: 23.0
Q2 is: 26.0
Q3 is: 30.0
```

Max: بزرگترین داده

Min: کوچکترین داده

Q1: چارک اول (25% داده ها قبل از ان هستند)

Q2: چارک دوم(میانه) (50% داده ها قبل از ان هستند)

Q3: چارک سوم (75% داده ها قبل از ان هستند)

بخش ۳:

آ)

```
data = np.random.choice(df['weight'], 100, replace=False)
print(f"Average is: {np.average(data)}")
print(f"Variance is: {np.var(data)}")
print(f"Standard Variation is: {np.std(data)}")
```

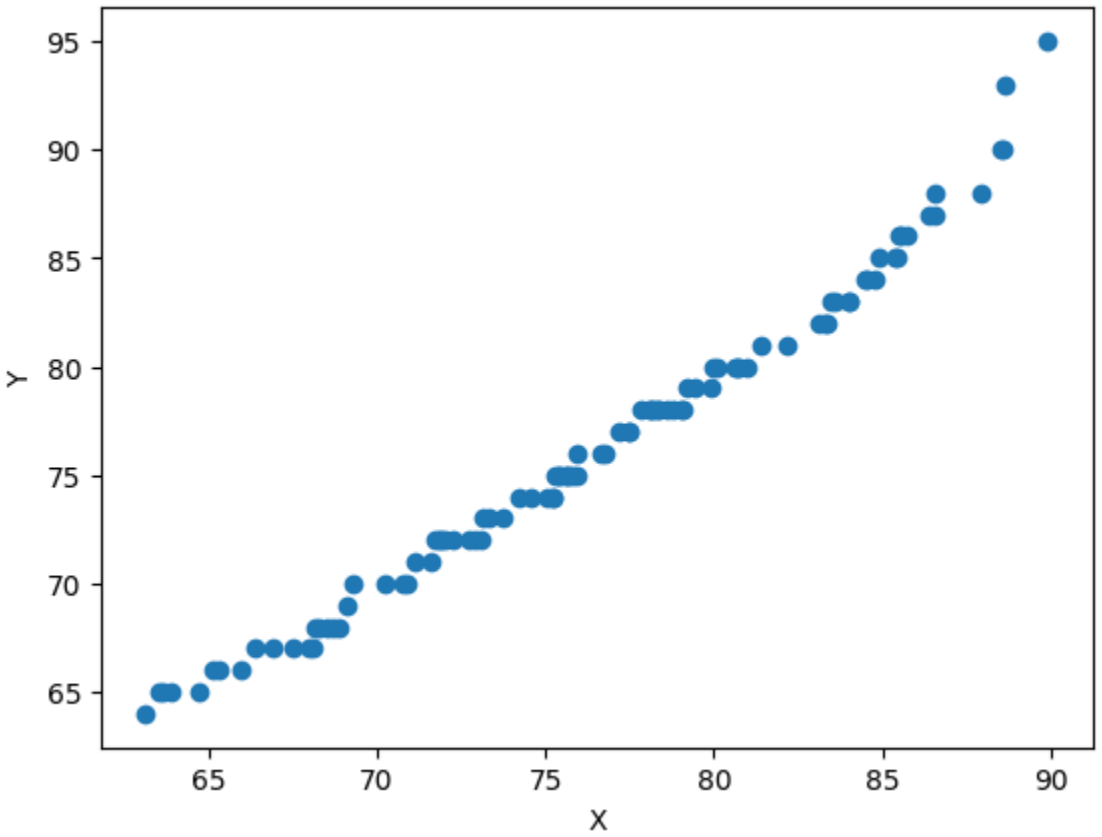
```
Average is: 75.48
Variance is: 50.70960000000001
Standard Variation is: 7.121067335729947
```

ب) نمودار Q-Q نموداری است که در ابتدا 2 تا دیتاست دریافت میکند و سپس هر کدام از انها را مرتب(sort) میکند و سپس به ترتیب جایگاه هر دیتا (چندک(quantiles)) انها را در 2 تا دیتاست بست اورده و کنار هم در نمودار رسم میکنیم

```
def Q_Q_two_sample(x, y):
# Quantile-quantile plot
    plt.figure()
    plt.scatter(np.sort(x), np.sort(y))
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.show()
    plt.close()
```

ج)

```
norm = np.random.normal(np.average(data), np.std(data), len(data))

Q_Q_two_sample(norm, data)
```



-همانطور که در نمودا روبرو مشاهده میکنید توزیع آماری وزن بازیکنان را میتوان با توزیع نرمال تقریب زد زیرا تا حد خوبی این دو داده نسبت خطی با یکدیگر دارند

د)

```
import scipy.stats as stats
statistic, p_value = stats.shapiro(data)
print(p_value)
```

✓ 0.0s

```
0.10022921115159988
```

همانطور که در بالا مشاهده میکنید مقدار p_value مقداری بزرگتر از 5% شده است بنابراین نمیتوان فرض صفر را رد کرد پس میتوان گفت داده ها از توزیع نرمال پیروی می کنند

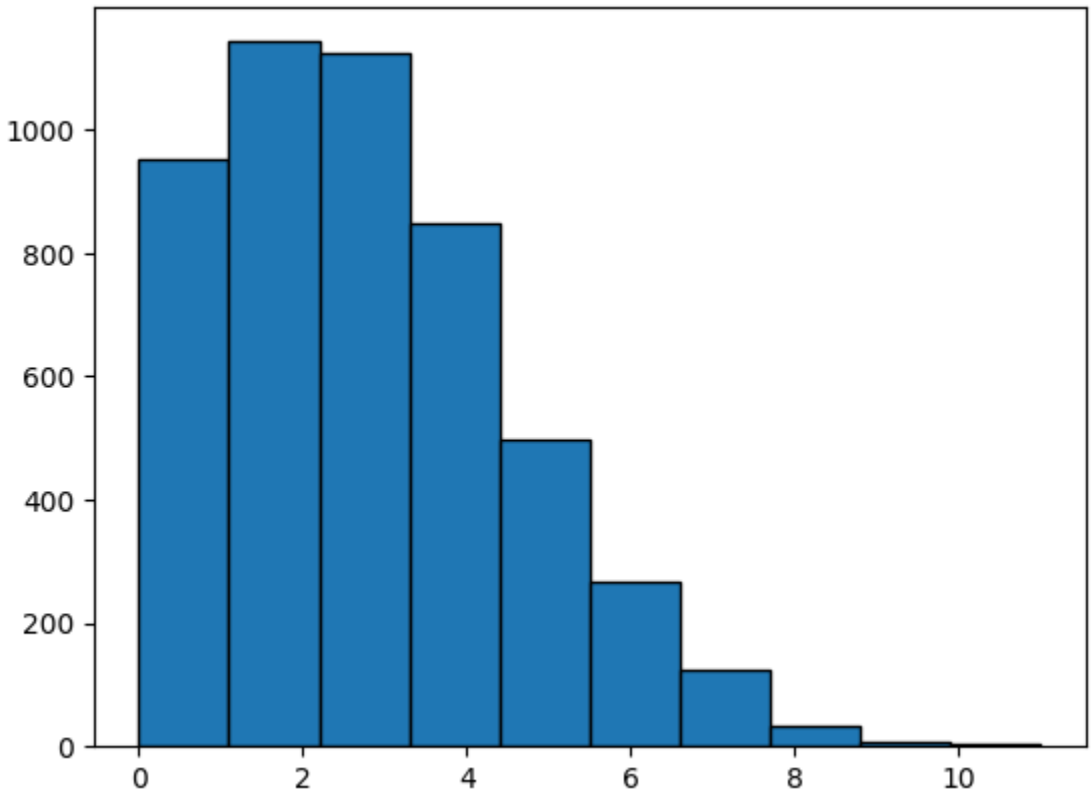
ه)

بخش ۴:

آ)

```
po_data = np.random.poisson(3, 5000)

plt.hist(po_data,bins=10,edgecolor='black')
```

✓ 0.2s



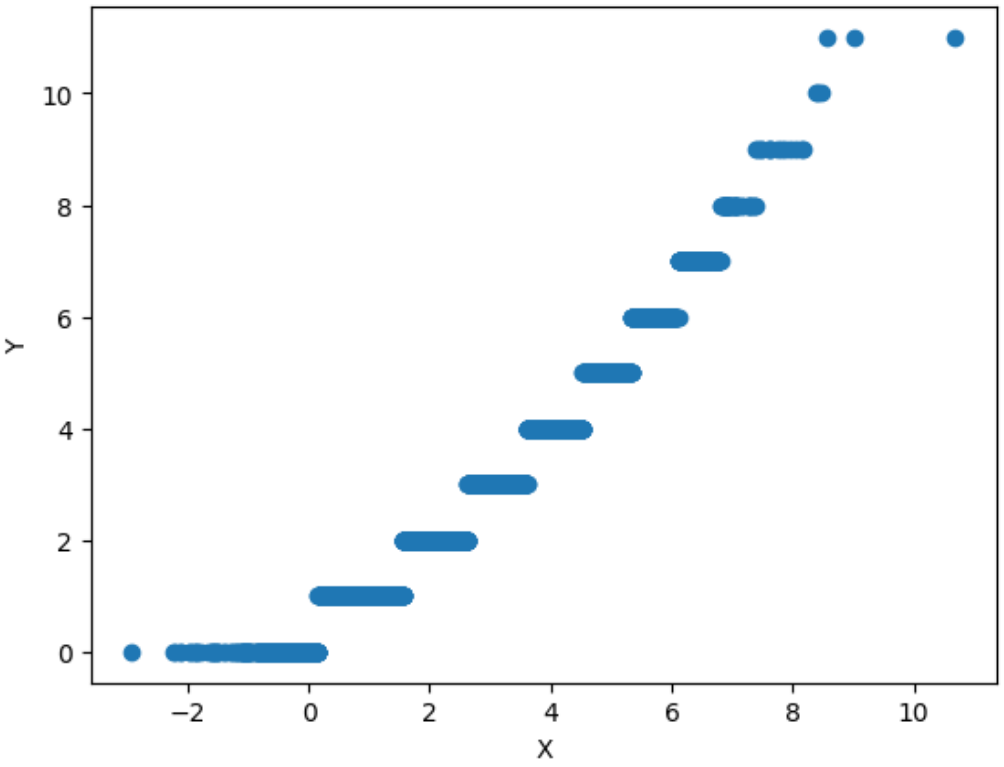
ب)

فرم کلی:

```
n = 5000
po_data = np.random.poisson(3, n)
norm = np.random.normal(np.mean(po_data), np.std(po_data), n)
Q_Q_two_sample(norm, po_data)
_, p_value = stats.shapiro(po_data)
print(p_value)
```

✓ 0.6s

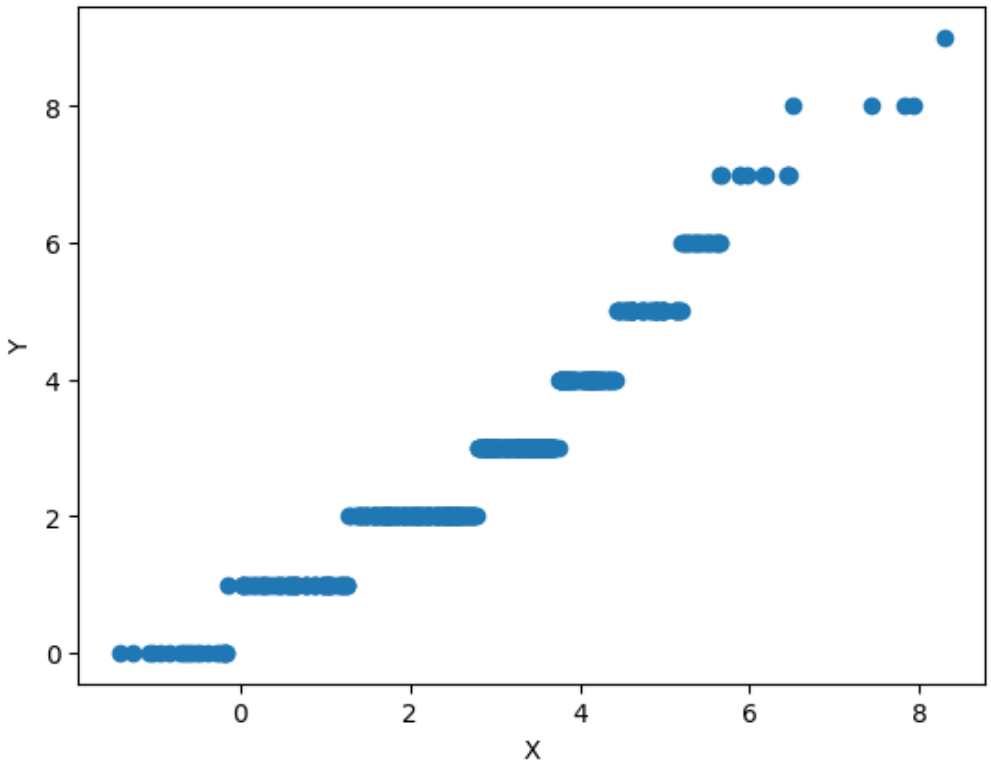
N=5000 :



7.68113485558709e-39

فرض صفر رد میشود

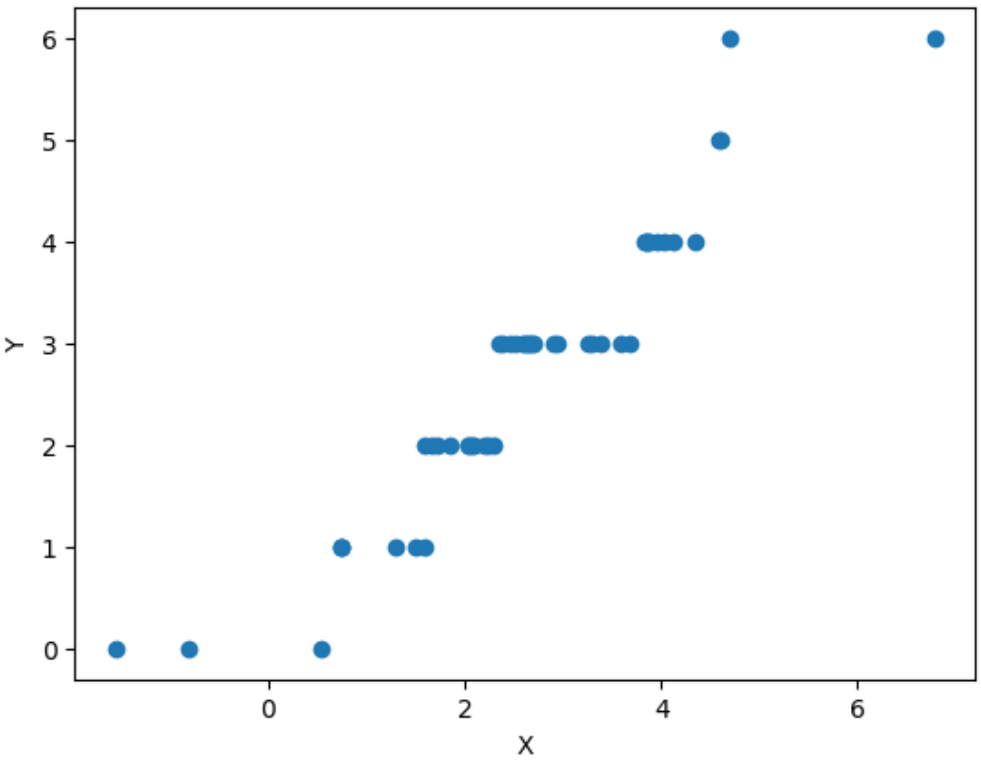
N=500 :



1.61168455426082e-13

فرض صفر رد میشود

N=50 :




```
0.020133780315518 38
```

فرض صفر رد نمیشود

-همانطور که مشاهده میکنید با کاهش n , p_value افزایش میابد

با تشکر از توجه شما ☺