بسم الله الرحمن الرحیم

پروژه ۳ آمار و احتمالات مهندسی

دکتر توسلی پور

مهدی وجهی

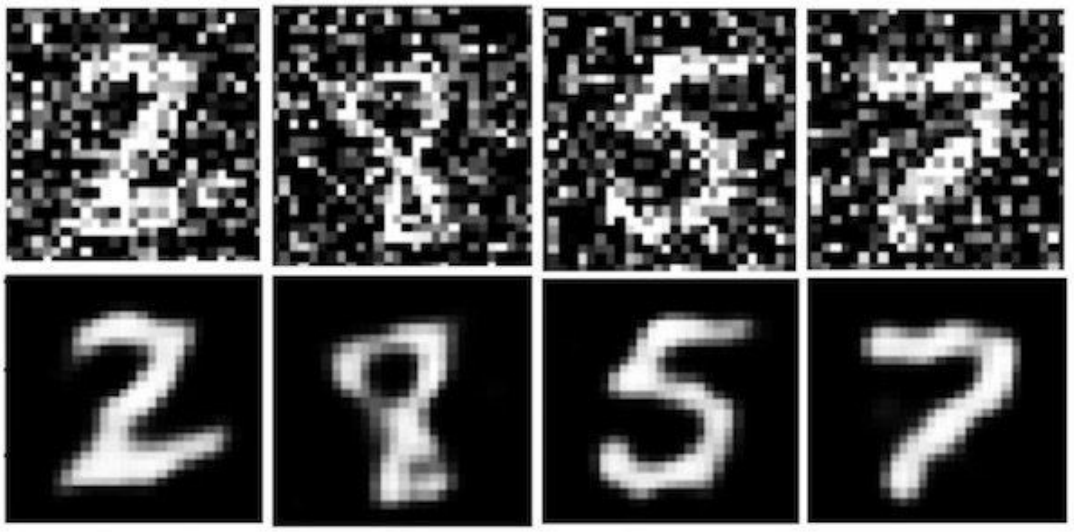
۸۱۰۱۰۱۵۵۸

فهرست

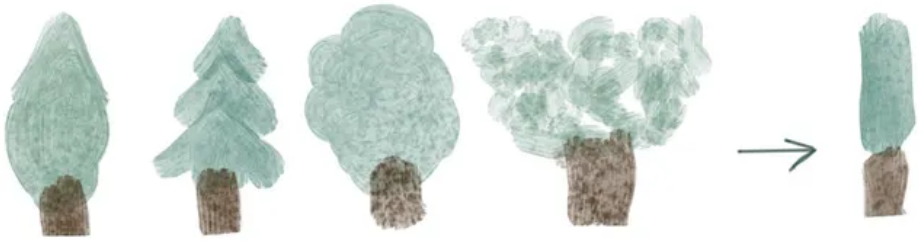
سوال ۱

بخش ۱

1. حذف داده های پرت از داده های آماری: با استفاده از این روش می توان داده های پرت را از داده های آماری خود حذف کرد زیرا این مدل مجبور است در فشرده سازی فقط الگو های اصلی ذخیره می شود و موارد نامربوط و پرت از داده ها حذف می شود
2. حذف نویز از صوت و تصویر: این مدل کمک می کند که نویز از تصاویر حذف شود دلیل این موضوع همان فشرده سازی ای است که انجام می گیرد. مانند اعداد دست نویس زیر:



3. تولید تصویر: این روش با یادگیری الگوی کلی تصاویر قادر است تصویری با حفظ کلیت مفهوم خلق کند.



بخش ۲

دلیل این موضوع فشرده سازی داده است. با فشرده سازی داده بخشی از آن حذف می شود بنابراین واضح است که با تولید مجدد تصویر بخشی از داده ها مجدد باید تولید شود و این روند به خاطر از دست دادن با خطا همراه است. طبیعی است که با افزایش فضای پنهان فشرده سازی کمتری صورت می گیرد و داده های کمتری از دست می روند پس تولید مجدد تصاویر با خطای کمتری همراه خواهد بود.

بخش ۳

قطعه کد های داده شده را اجرا می کنیم و در آخر برای نمایش از کد زیر استفاده می کنیم:

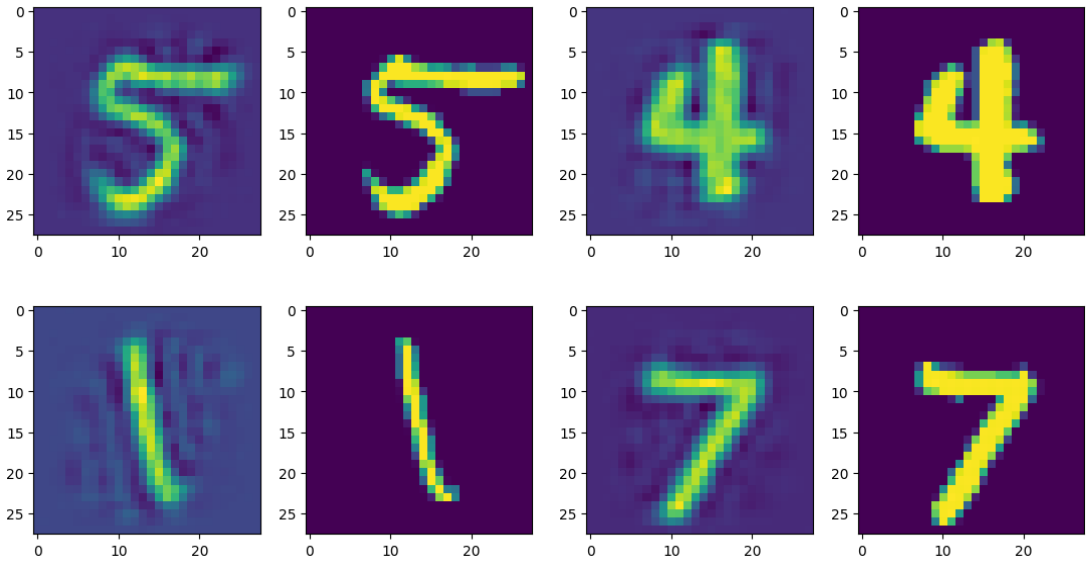
```
import matplotlib.pyplot as plt
rand = np.random.randint(0, len(reconstructed_images))
random_case = reconstructed_images[rand]
random_case2 = test_images[rand]
img_m = np.reshape(random_case, (28,28))
img_m2 = np.reshape(random_case2, (28,28))

plt.subplot(1, 2, 1)
plt.imshow(img_m)

plt.subplot(1, 2, 2)
plt.imshow(img_m2)

plt.show()
```

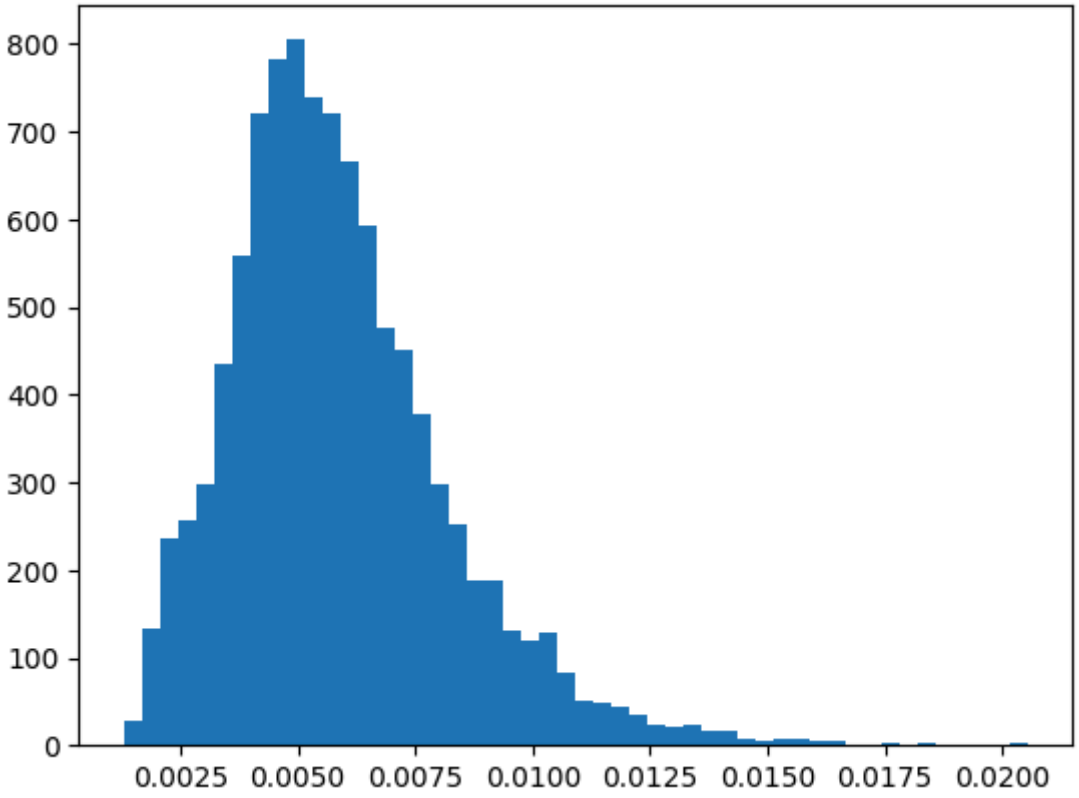
خروجی ها به صورت زیر است: (راست تصویر اصلی و چپ تصویر تولیدی)



حال mse را برای تصاویر حساب می کنیم.

```
mse = [0] * len(test_images)
for i in range(len(test_images)):
   for j in range(len(test_images[i])):
       mse[i] += (test_images[i][j] - reconstructed_images[i][j]) ** 2
for i in range(len(mse)):
   mse[i] /= len(test_images[i])
```

نمودار آن به شکل زیر است:



حال باید آزمون نیکویی برازش کولموگروف - اسمیرنوف را انجام دهیم:

```
import statistics as st
m_v_mse = (st.mean(mse), st.variance(mse))
from scipy import stats
ks_statistic, p_value = stats.kstest(mse, cdf='norm', args=(m_v_mse[0],
m_v_mse[1]))
```

نتیجه ۰ است. چون از ۰.۰۵ کمتر است پس رد می شود. اما این تست واقعا تست خوبی نیست زیرا حتی مثلا برای کد زیر که مقایسه با توزیع نرمال هست هم خروجی ۲٪ می دهد!

```
x = stats.norm.rvs(size=10000)
stats.kstest(x, stats.norm.cdf)
```

اما اگر خودمان بخواهیم تحلیل کنیم ما ۱۰۰۰۰ تصویر مستقل از داریم اگر mse آنها را حساب کنیم این مقدار برای هر کدام از آنها یک متغیر تصادفی مستقل است که طبق قضیه حد مرکزی باید به توزیع نرمال میل بکند که در نمودار نیز اینگونه مشاهده می شود.

سوال ۲

بخش ۱

پرت y دور است اما x نزدیک است ولی برای اهرمی برعکس و پرت از الگو پیروی نمی کند و اهرمی روی نتیجه موثر است

به طور کلی به داده هایی که از الگوی کلی پیروی نمی کنند پرت می گویند و به آنهایی که در الگوی کلی تاثیر اساسی دارند اهرمی می گویند. اگر هم عنصری هم پرت باشد و هم اهرمی هم از الگویی کلی پیروی نمی کند و هم روی نتیجه نهایی تاثیر گذار است. به طور دقیق تر به داده ای که y آن دور است و در اهرمی x آن. ممکن است داده های پرت به خاطر بعضی از خطا های ما ایجاد شده باشد بنابراین حذف آنها می تواند دقت نتیجه را بالا ببرد اما همیشه این طور نیست پس ما باید بررسی کنیم و دلیل ایجاد داده پرت را متوجه شویم تا بفهمیم که چه اقدامی برای آن باید انجام دهیم. به طور کلی به نقطه ای اهرمی خوب می گویند که x آن از طرح کلی پیروی نکند اما y از طرح کلی حمایت کند در این صورت در نتیجه تاثیری منفی ای را شاهد نیستیم ولی اگر y از الگوی کلی حمایت نکند می تواند در نتیجه تاثیر بد داشته باشد و به آن اهرمی بد می گویند.

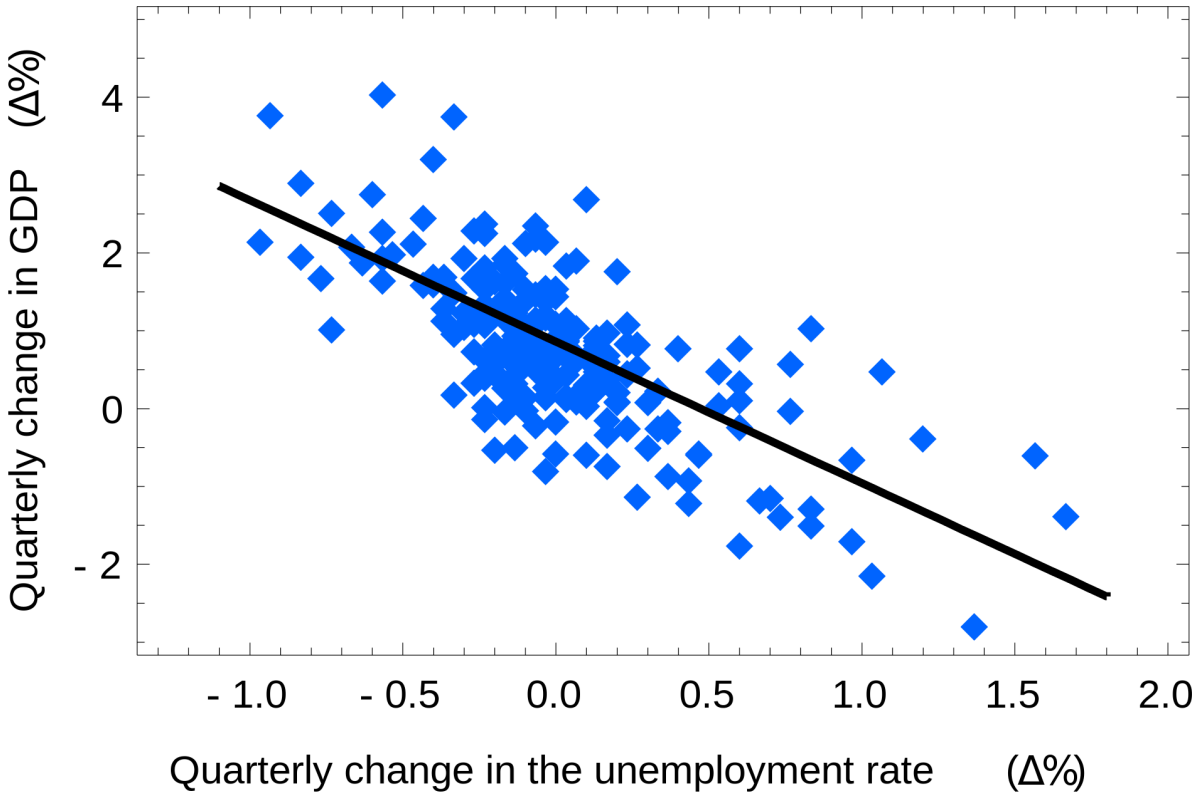
بخش ۲

ضریب تعیین به نسبتی از واریانس بر حسب متغیر وابسته می گویند که از متغیر مستقل قابل پیش‌بینی باشد. فرمول آن به شکل زیر است:

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

R^2 = coefficient of determination
RSS = sum of squares of residuals
TSS = total sum of squares

ای ضریب در واقع به ما می گویند که اون مدل ما چقدر خطا دارد. هر چقدر مقدار این ضریب به یک نزدیک تر باشد یعنی مدل دقیق تر است و بر الگو تطابق دارد اما اگر به صفر میل کند یعنی مدل به درستی طراحی نشده و خطای بالایی دارد. در نمودار زیر ضریب تعیین داده ها را مشاهده می کنید:



بخش ۳

نتیجه رگرسیون Y=β0+β1X است. حال باید β1 , β0 ها را محاسبه کرد. روی کاغذ با مشتق گیری و سایر محاسبات قابل به دست آوردن هستند اما با جست و جویی در اینترنت جواب پارامتری β1 , β0 را پیدا می کنیم (البته که به دست آوردن آن به همان روش دستی و با استفاده از مشتق از مربع خطا بوده):

$$\widehat{\beta_0} = \bar{y} - \widehat{\beta_1}\bar{x}$$

$$\widehat{\beta_1} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}$$

حال رگرسیون را پیاده می کنیم.

```
def regression(x, y):
    b1 = cal_b1(x, y)
    b0 = cal_b0(b1, x, y)
    r2 = cal_r2(b0, b1, x, y)
    return b1, b0, r2
```

```
def cal_b1(x, y):
    return (np.sum(x*y)-len(x)*np.mean(x)*np.mean(y)) /
```

```
(np.sum(x*x)-len(x)*np.mean(x)*np.mean(x))
```

```
def cal_b0(b1, x, y):
   return np.mean(y)-b1*np.mean(x)
```

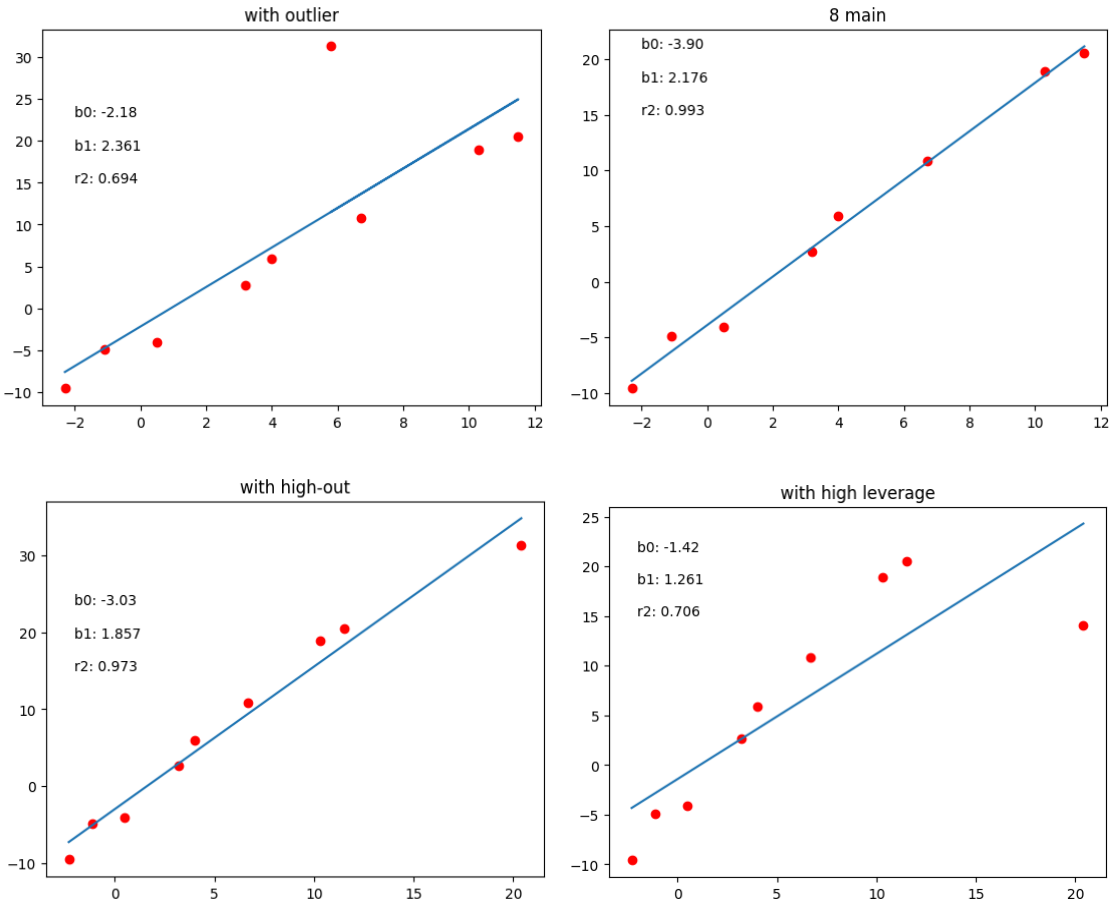
```
def cal_r2(b0, b1, x, y):
   Y = b1 * x + b0
   return 1 - ((np.sum((y - Y)**2)) / (np.sum((y - np.mean(y))**2)))
```

سپس برای موارد خواسته شده باید رگرسیون بدست بیاوریم تابعی برای این کار می نویسیم:

```
def regres_and_plot(x_main, y_main, x_other, y_other, title):
   x = np.array(list(x_main) + list(x_other))
   y = np.array(list(y_main) + list(y_other))
   b1, b0, r2 = regression(x, y)
   Y = b1 * x + b0
   print('b0: ', b0)
   print('b1: ', b1)
   print('r2: ', r2)
   plt.scatter(x, y, color = "red")
   plt.plot(x, Y)
   plt.text(-2, 15, 'b0: ' + str(b0)[:5] + '\n\nb1: ' + str(b1)[:5] +
'\n\nr2: ' + str(r2)[:5])
   plt.title(title)
```

سپس برای هر یک از موارد تابع مربوطه را صدا می زنیم:

```
x_main = np.array([-2.3, -1.1, 0.5, 3.2, 4.0, 6.7, 10.3, 11.5])
y_main = np.array([-9.6, -4.9, -4.1, 2.7, 5.9, 10.8, 18.9, 20.5])
regres_and_plot(x_main, y_main, np.array([]), np.array([]), "8 main")
regres_and_plot(x_main, y_main, np.array([5.8]), np.array([31.3]), "with
outlier")
regres_and_plot(x_main, y_main, np.array([20.4]), np.array([14.1]), "with
high leverage")
regres_and_plot(x_main, y_main, np.array([20.4]), np.array([31.3]), "with
high-out")
```



بخش ۴

یک راهی که به نظر می رسد شاید بد نباشد استفاده از مدل سوال قبل است با این کار می توانیم داده های نامربوط را کمرنگ تر بکنیم و نتایج متعادل تر بشود. می توان از روش DBSCAN نیز استفاده که به این صورت که این روش ما را در خوشه بندی داده ها کمک می کند و با این کار می توانیم داده های پرت را شناسایی و حذف کنیم. راهکار دیگری که به نظر می آید این است که ما به اختلاف مربعات خود وزن اضافه کنیم به این صورت که نقاطی که به خط نزدیک تر هستند موثر تر باشد که باعث این می شود که داده های دور تر کم ارزش تر شوند.

سوال ۳

بخش ۱

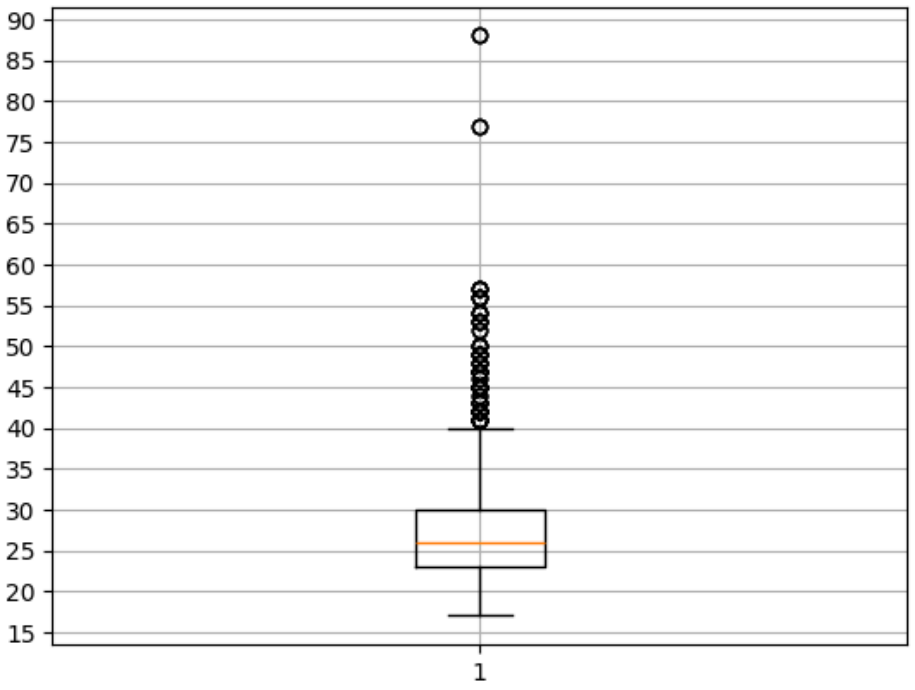
سه راه به نظر می رسد یک راه ساده این است که مقدار آن ها را برابر میانگین سایر داده ها قرار دهیم و راه دیگر که پیچیده تر است این است که با توجه سایر ستون ردیفی مشابه با ردیفی که خانه ی نامعلوم دارد پیدا کنیم و از مقدار آن استفاده کنیم یا یک رابطه ای بین باقی خانه ها با آن خانه با کمک باقی ردیف ها پیدا کنیم و با استفاده از آن مقدار این خانه را پیشبینی کنیم. اما ما همان راه ساده یعنی میانگین را استفاده می کنیم 😂.

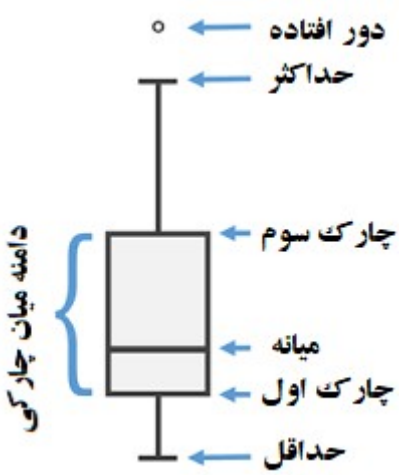
```
mean = df['pace'].mean()
df['pace'] = df['pace'].fillna(mean)
mean = df['dribbling'].mean()
df['dribbling'] = df['dribbling'].fillna(mean)
```

بخش ۲

در این قسمت به شکل زیر نمودار جعبه ای را می کشیم:

```
plt.boxplot(df['age'])
plt.yticks(range(15, 91, 5))
plt.grid(True)
```





حداکثر برابر ۴۰ و چارک ۳ برابر ۳۰ و میانه برابر ۲۶ و چارک ۱ برابر ۲۳ و حداقل برابر ۱۷ است. این مقادیر بعد از حذف داده های پرت به صورت زیر تعریف می شوند:

- حداکثر: بیشترین مقدار در داده ها
- چارک سوم: ۷۵٪ داده ها مقداری کمتر از این مقدار دارند.
- میانه (چارک دوم): ۵۰٪ داده ها مقداری کمتر از این مقدار دارند.
- چارک اول: ۲۵٪ داده ها مقداری کمتر از این مقدار دارند.
- حداقل: کمترین مقدار در داده ها

بخش ۳

نمونه برداری می کنیم:

```
random_data = df['weight'].sample(n=100, replace=False)
```

زیر بخش آ

```
print("mean:",np.mean(random_data))
print("var:",np.var(random_data))
print("std:",np.std(random_data))
```

```
mean: 76.7
var: 53.07
std: 7.284915922644544
```

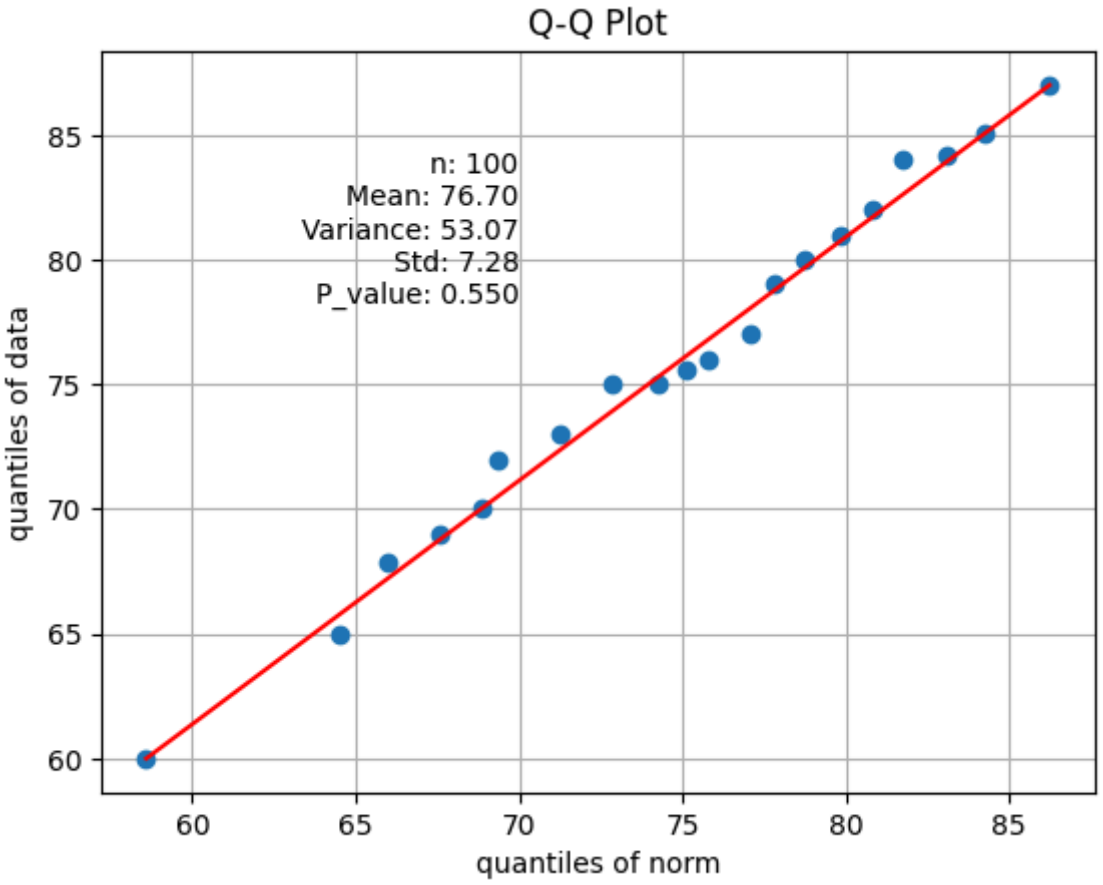
زیر بخش ب

این نمودار به منظور مقایسه دو توزیع استفاده می شود همچنین می توان دید که داده ها بر یک توزیع خاص تطابق دارند یا خیر. اگر نقاط نمودار روی یک خط راست قرار داشته باشند یعنی دو توزیع رابطه خطی دارند و اگر این خط بر x=y منطبق باشد یعنی آنها یکسانند.

زیر بخش ج

```
sample = np.random.normal(np.mean(random_data), np.std(random_data), 500)
quantiles_norm = np.percentile(sample, np.arange(0, 100, 5))
quantiles_data = np.percentile(random_data, np.arange(0, 100, 5))

plt.plot(quantiles_norm, quantiles_data, marker='o', linestyle='None')
plt.plot([min(quantiles_norm), max(quantiles_norm)], [min(quantiles_data),
max(quantiles_data)], color='red')
plt.xlabel('quantiles of norm')
plt.ylabel('quantiles of data')
plt.title('Q-Q Plot')
plt.grid(True)
plt.show()
```

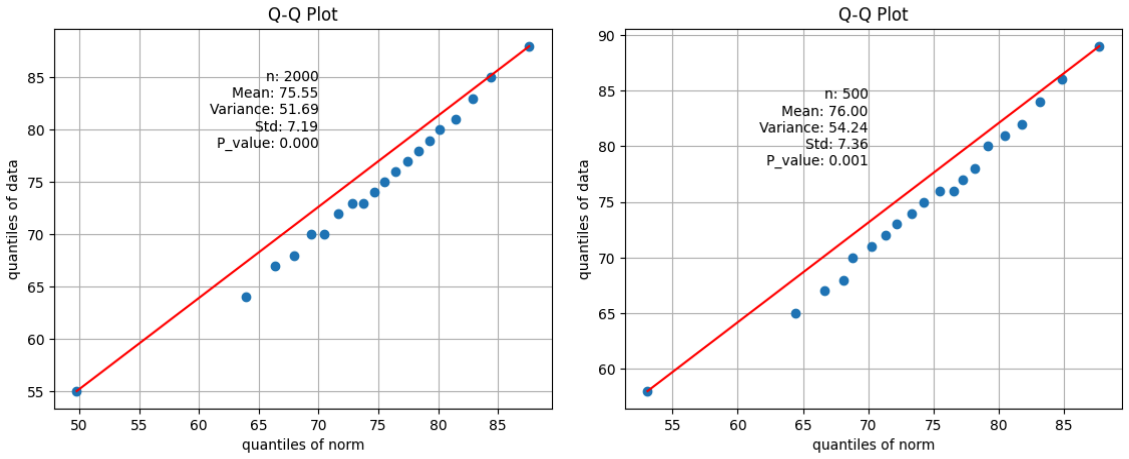


همانطور که مشاهده می شود نقاط تقریبا روی خط راست قرار دارند که یعنی رابطه خطی دارند. نسبتا هم به خط x=y نزدیک است یعنی می توان گفت این دو توزیع تقریبا یکسانند

زیر بخش د

مقدار p_value برابر 0.0013 است که یعنی نرمال بودن رد می شود.

زیر بخش ه

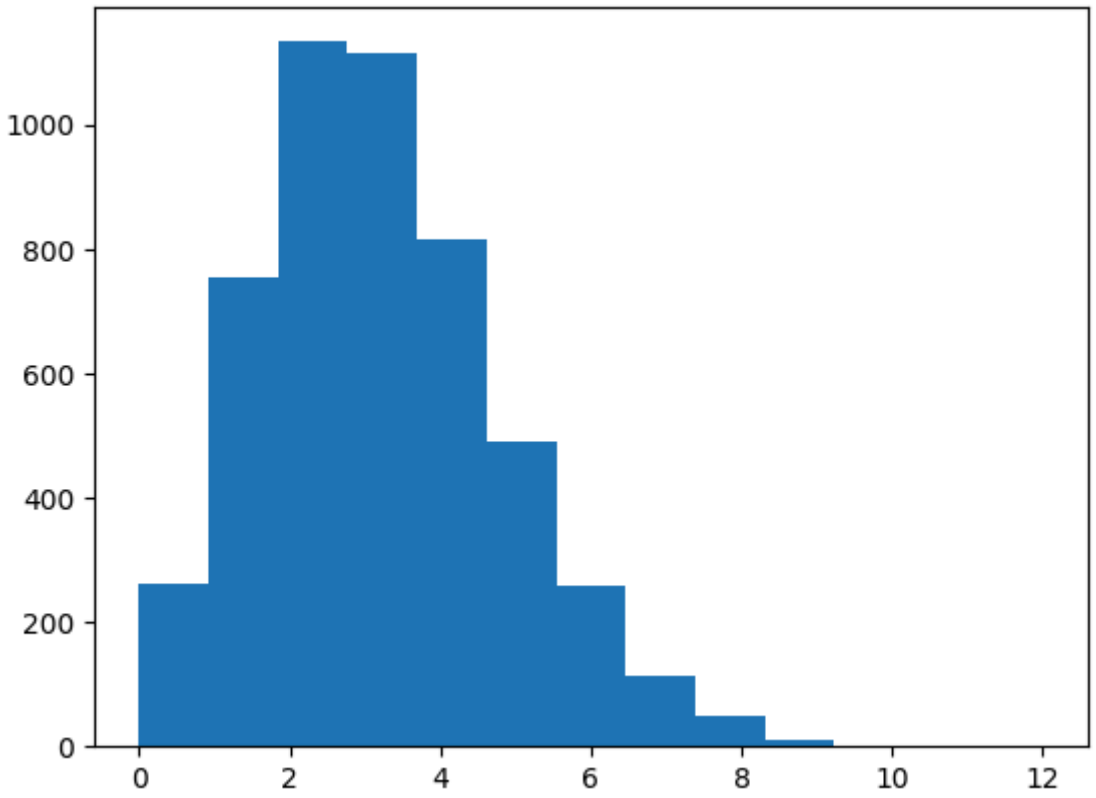


می بینیم انحراف بیشتر و بیشتر می شود و تطابق دو توزیع از بین می رود.

بخش ۴

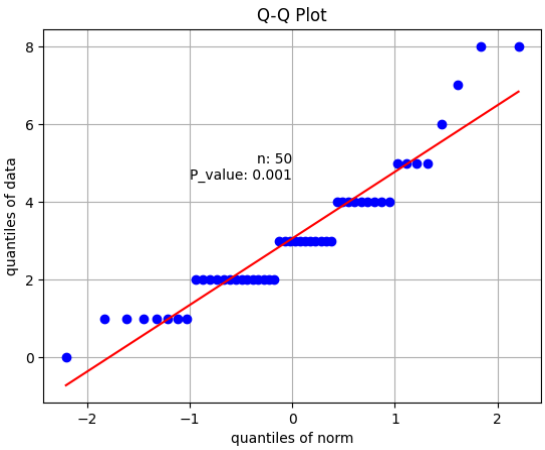
نمودار بخش آ را می کشیم فقط بنده جایی رو پیدا نکردم به بتونیم تنظیم کنیم انتخاب با جایگذاری باشه یا بدون جایگذاری:

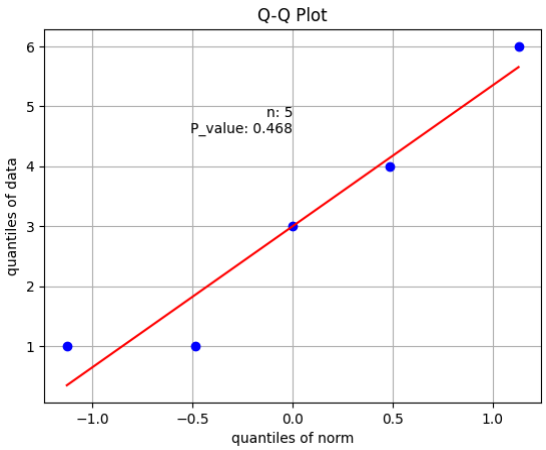
```
set_seed(810109203)
random_data = np.random.poisson(3, 5000)
_ = plt.hist(random_data, bins=13)
```

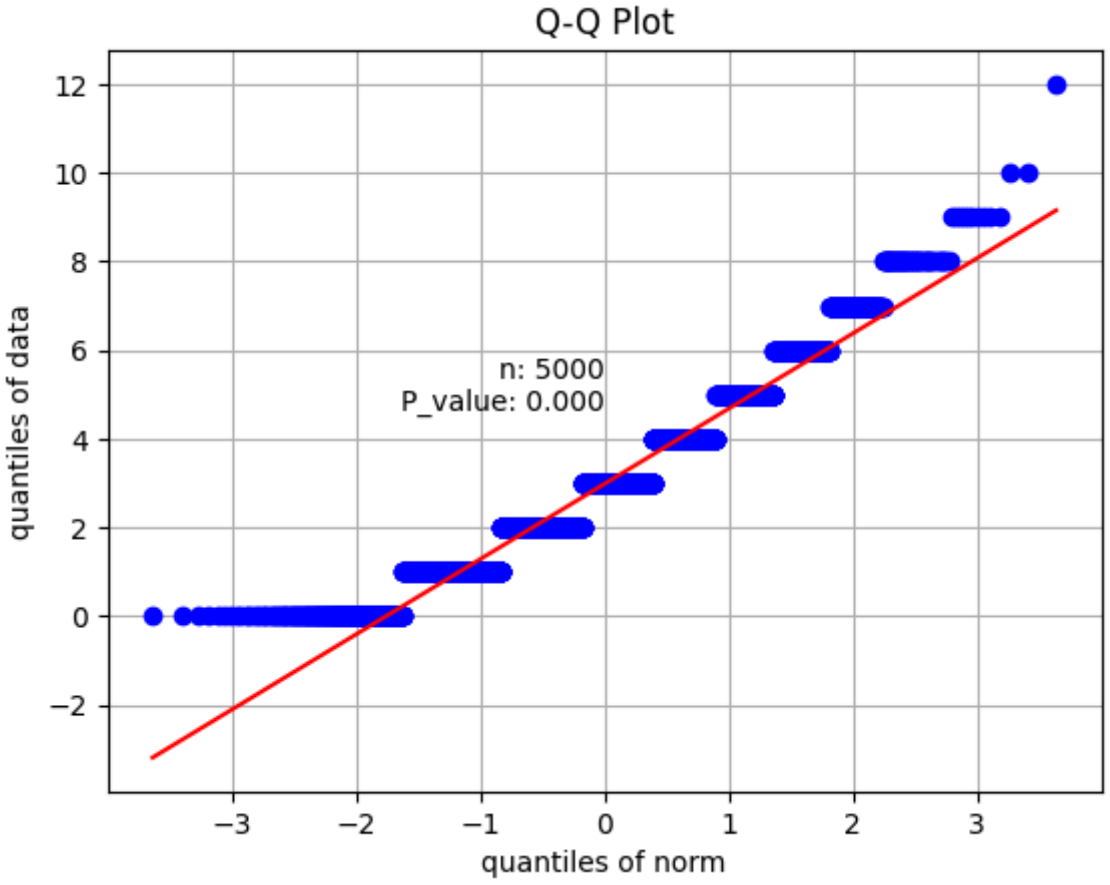


برای قسمت ب کد زیر را داریم:

```
def test(n):
   set_seed(810109203)
   random_data = np.random.poisson(3, n)
   statistic, p_value = stats.shapiro(random_data)
   stats.probplot(random_data, dist="norm", plot=plt)
   plt.xlabel('quantiles of norm')
   plt.ylabel('quantiles of data')
   plt.title('Q-Q Plot')
   plt.text(0, 4.5, f'n: {n}\nP_value: {p_value:.3f}', ha='right',
va='bottom')
   plt.grid(True)
   plt.show()
```

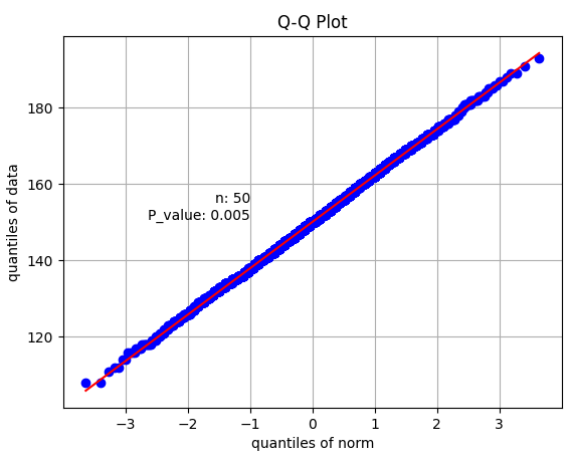


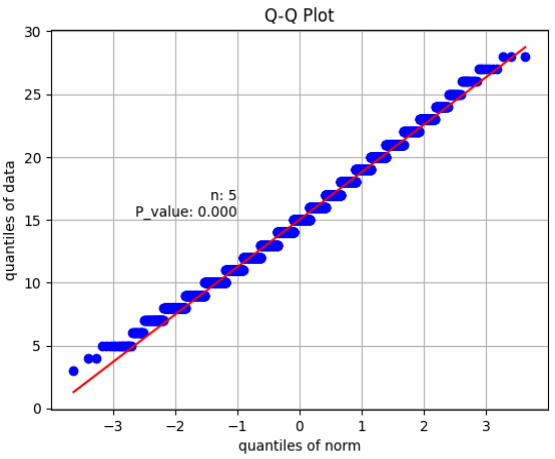


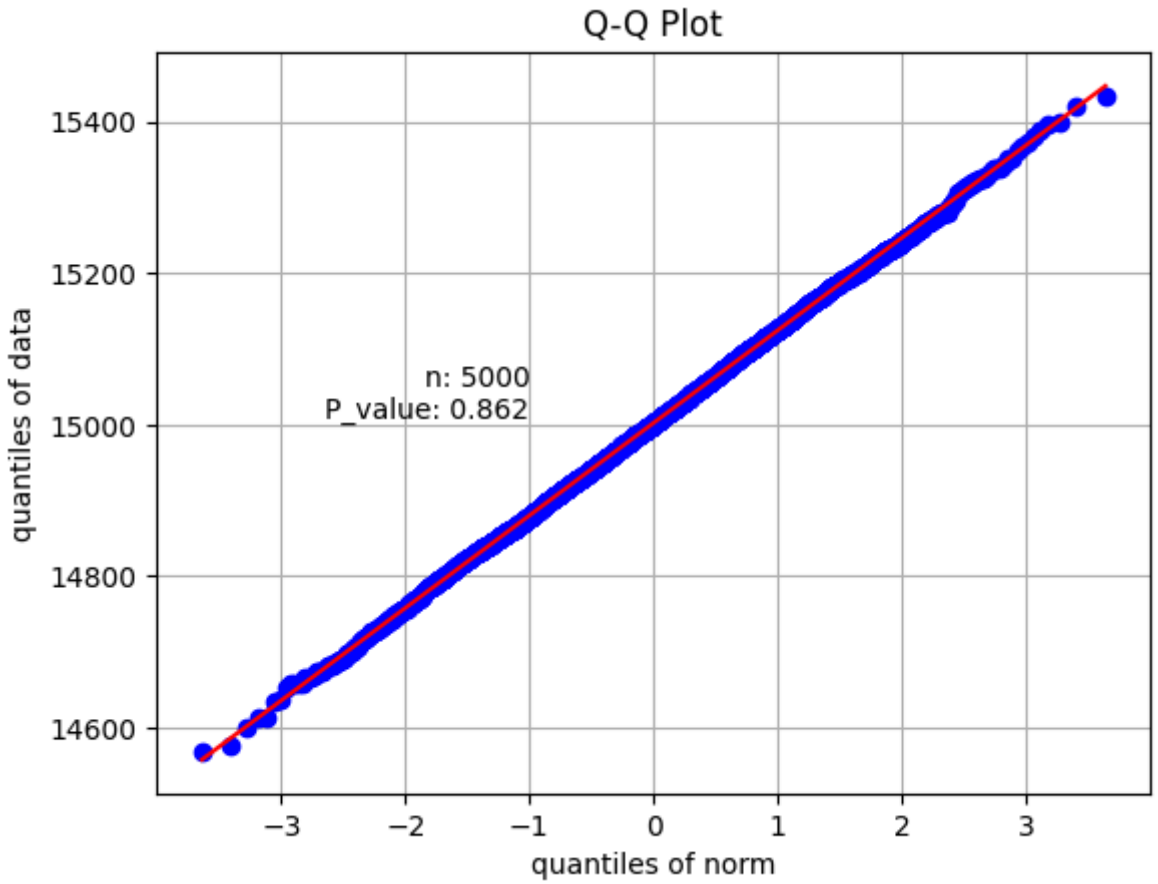


که با حد مرکزی قابل توجیه نیست اما اگر به صورت زیر پیاده کنیم درست کار می کند.

```
random_data = np.random.poisson(3 * n, 5000)
```







که در این حالت به نرمال میل می کند زیرا پواسون با پارامتر ۱۵ معادل جمع ۵ پواسون با پارامتر ۳ است در مرحله بعد جمع ۵۰ تا در مرحله آخر جمع ۵۰۰۰ تا. چون ما ۵۰۰۰ تا پواسون که مستقل از هم هستند را با هم جمع زدیم طبق قضیه حد مرکزی باید به توزیع نرمال میل کند.