**تمرین 2**

در این تمرین، ما به بررسی مجموعه داده‌ی Breast Cancer Wisconsin پرداختیم، و با استفاده از انتخاب ویژگی و روش آماری T-Test اقدام به مرتب سازی ویژگی‌ها بر اساس اهمّیّت کردیم. سپس با استفاده از الگوریتم KNN اقدام به کلاسبندی این داده‌ها به دو کلاس ['malignant', 'benign'] کردیم. مراحل کار در ادامه ارائه می‌گردد.

**1. ساخت DataFrame:**

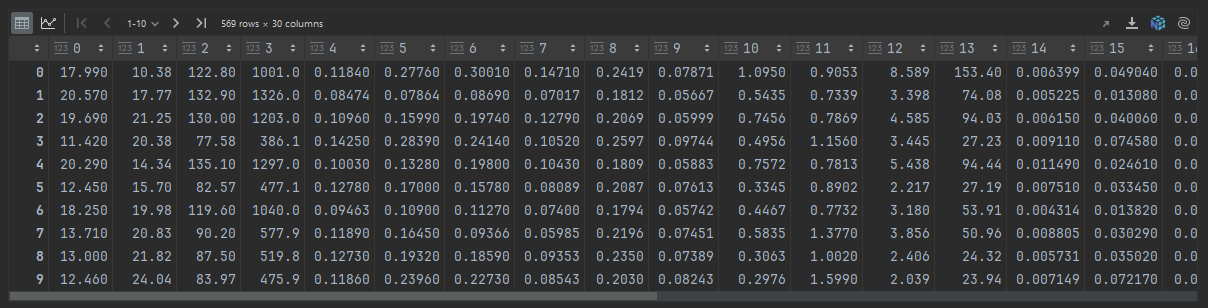
برای این منظور، ابتدا مجموعه‌ی داده را با استفاده از کتابخانه‌ی Pandas به یک DataFrame تبدیل کردیم. روش کار به صورت زیر می‌باشد:

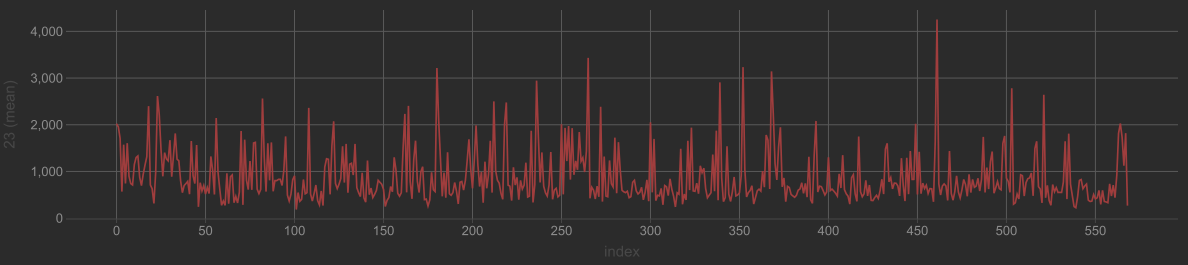
1. data = datasets.load\_breast\_cancer()

2. data["data"]

3.

Output:





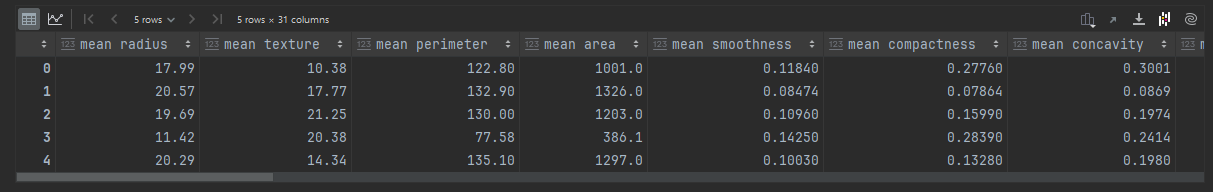
1. df = pd.DataFrame(data["data"] , columns = data["feature\_names"])

2. df["target"] = data["target"]

3. df.head()

4.

Output:



1. x = df.drop(columns = "target").copy()  
y = df["target"]

2.

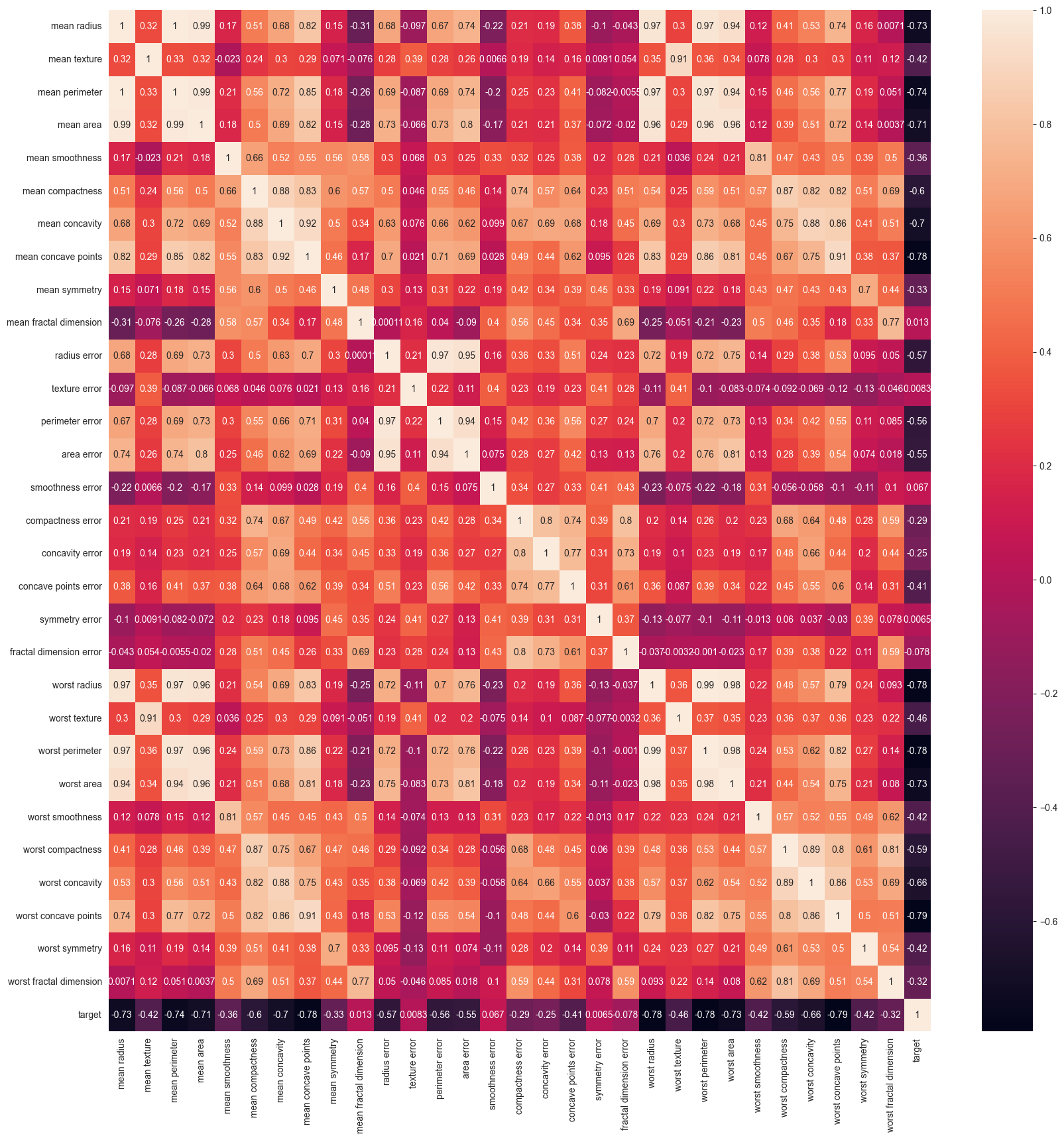
**2. رسم نمودار روابط:**

در این بخش، با استفاده از کتابخانه‌ی Seaborn اقدام به ترسیم نمودارهای تناسب بین ویژگی‌ها و کلاس‌ها نمودیم. روش کار به صورت زیر می‌باشد:

1. plt.figure(figsize = (20 , 20))  
sns.heatmap(df.corr() , annot = True)

2.

Output:

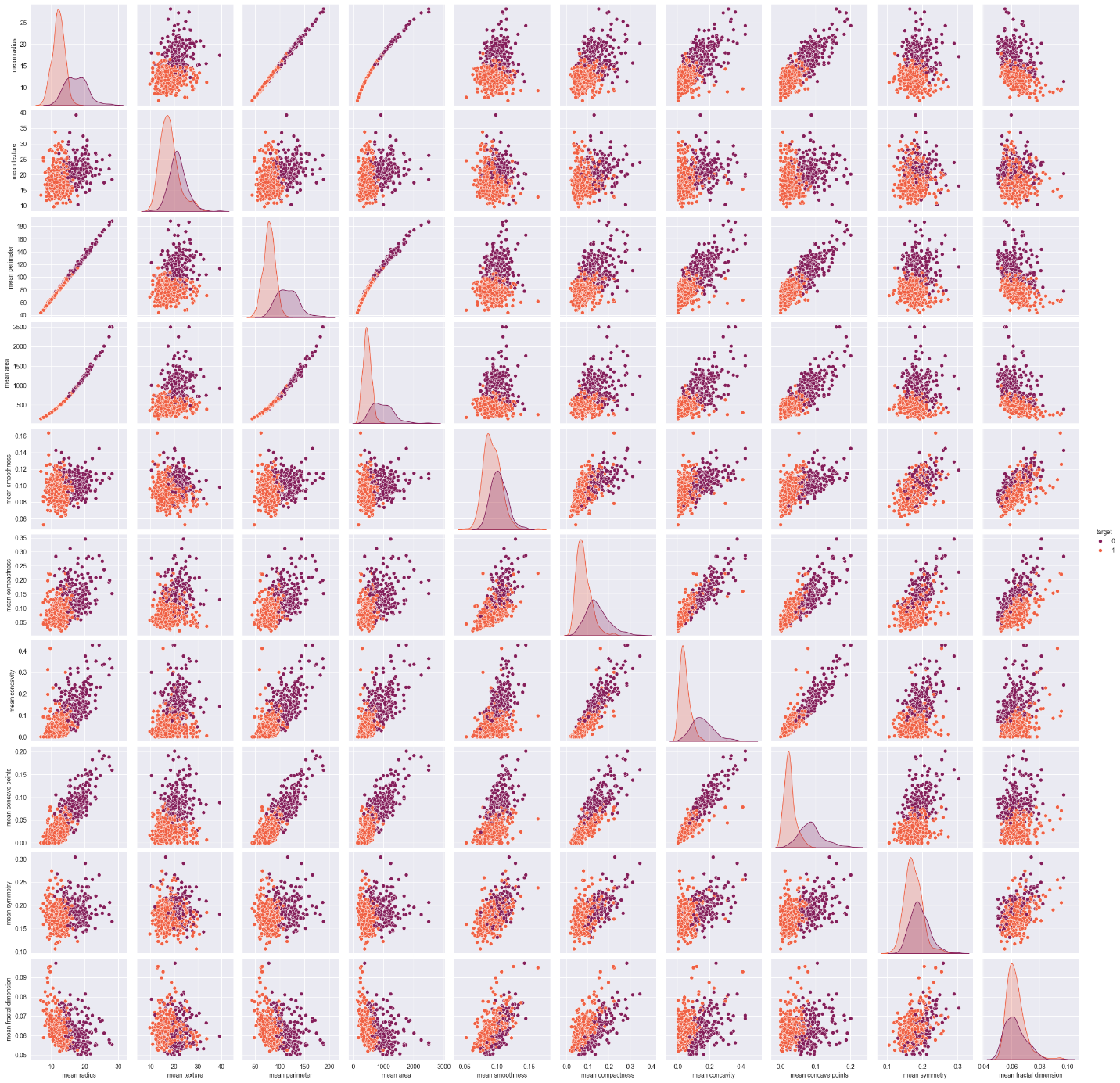


1. # Generate A Scatter Plot Matrix With The "mean" Columns.  
2. cols = ['target' ,  
3. 'mean radius' ,  
4. 'mean texture' ,  
5. 'mean perimeter' ,  
6. 'mean area' ,  
7. 'mean smoothness' ,  
8. 'mean compactness' ,  
9. 'mean concavity' ,  
10. 'mean concave points' ,  
11. 'mean symmetry' ,  
12. 'mean fractal dimension']  
13.

14. sns.pairplot(data = df[cols] , hue = "target" , palette = "rocket")

15.

Output:



1. data["target\_names"]

2.

Output:

array(['malignant', 'benign'], dtype='<U9')

1. # Generate And Visualize The Correlation Matrix.

2. corr = df.corr().round(2)

3.

4. # Mask For The Upper Triangle.

5. mask = np.zeros\_like(corr , dtype = np.bool\_)

6. mask[np.triu\_indices\_from(mask)] = True

7.

8. # Set Figure Size.

9. f , ax = plt.subplots(figsize = (20 , 20))

10. # Define Custom Colormap.

11. cmap = sns.diverging\_palette(220 , 10 , as\_cmap = True)

12.

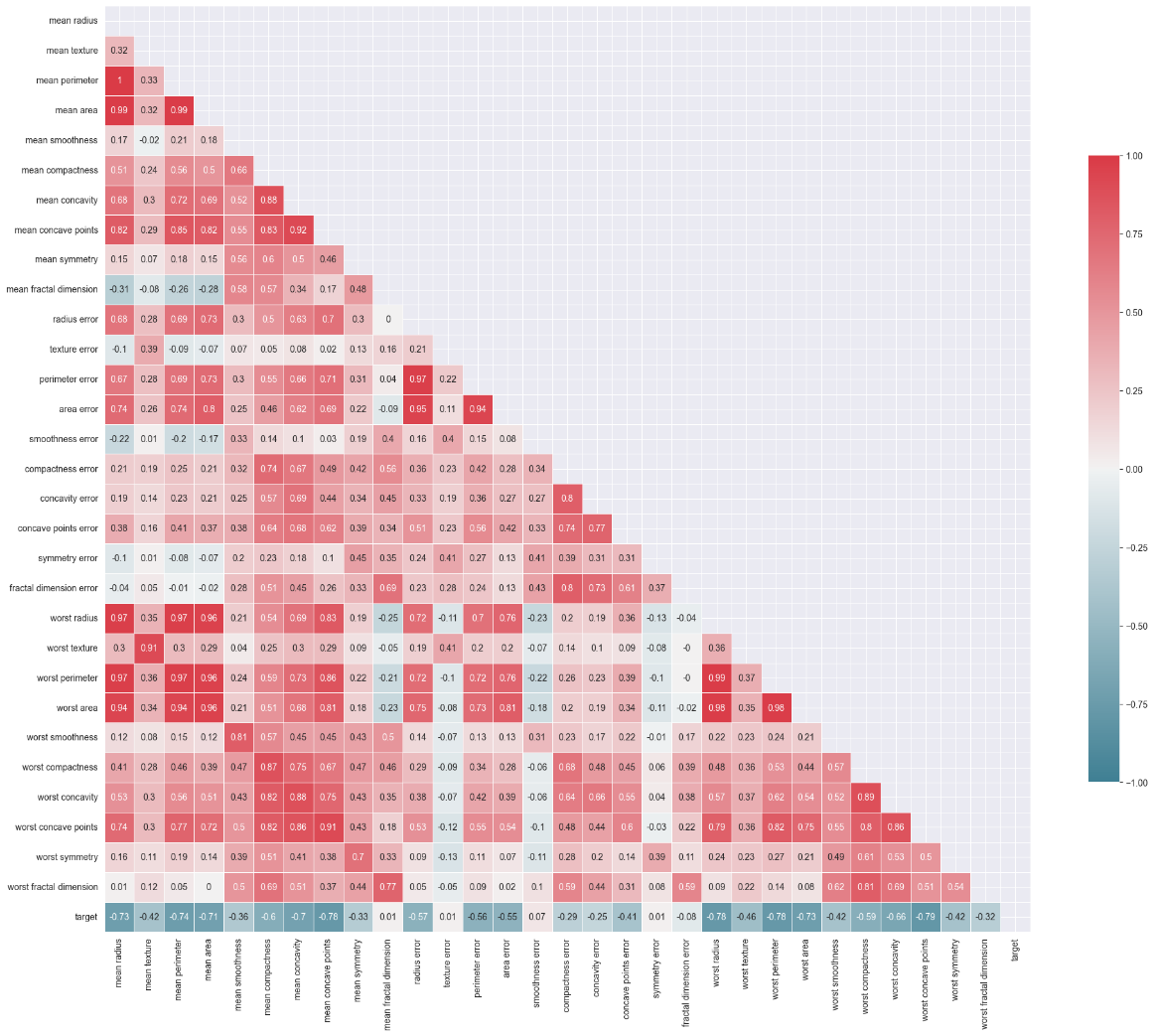
13. # Draw The Heatmap.

14. sns.heatmap(corr , mask = mask , cmap = cmap , vmin = -1 , vmax = 1 , center = 0 ,  
15. square = True , linewidths = 0.5 , cbar\_kws = {"shrink" : 0.5} , annot = True)

16. plt.tight\_layout()

17.

Output:



**3. انتخاب ویژگی:**

در این بخش، با توجّه به ماتریس همبستگی بالا و میزان ارتباط میان داده‌ها، بهترین ویژگی‌ها را از میان ویژگی‌های موجود انتخاب می‌کنیم. روش کار به صورت زیر است:

1. # First , Drop All "worst" Columns.

2. cols = ['worst radius' ,

3. 'worst texture' ,

4. 'worst perimeter' ,

5. 'worst area' ,

6. 'worst smoothness' ,

7. 'worst compactness' ,

8. 'worst concavity' ,

9. 'worst concave points' ,

10.   'worst symmetry' ,

11. 'worst fractal dimension']

12. df = df.drop(cols , axis = 1)

13.

14. # Then , Drop All Columns Related To The "perimeter" And "area" Attributes.

15. cols = ['mean perimeter' ,

16. 'perimeter error' ,

17. 'mean area' ,

18. 'area error']

19. df = df.drop(cols , axis = 1)

20.

21. # Lastly , Drop All Columns Related To The "concavity" And "concave points" Attributes.

22. cols = ['mean concavity' ,

23. 'concavity error' ,

24. 'mean concave points' ,

25. 'concave points error']

26. df = df.drop(cols , axis = 1)

27.

28. # Verify Remaining Columns.

29. df.columns

30.

Output:

Index(['mean radius', 'mean texture', 'mean smoothness', 'mean compactness',

'mean symmetry', 'mean fractal dimension', 'radius error',

'texture error', 'smoothness error', 'compactness error',

'symmetry error', 'fractal dimension error', 'target'],

dtype='object')

1. # Draw The Heatmap Again , With The New Correlation Matrix.  
2. # Generate And Visualize The Correlation Matrix.

3. corr = df.corr().round(2)

4.

5. # Mask For The Upper Triangle.

6. mask = np.zeros\_like(corr , dtype = np.bool\_)

7. mask[np.triu\_indices\_from(mask)] = True

8.

9. # Set Figure Size.

10. f , ax = plt.subplots(figsize = (20 , 20))

11.

12. # Define Custom Colormap.

13. cmap = sns.diverging\_palette(220 , 10 , as\_cmap = True)

14.

15. # Draw The Heatmap.

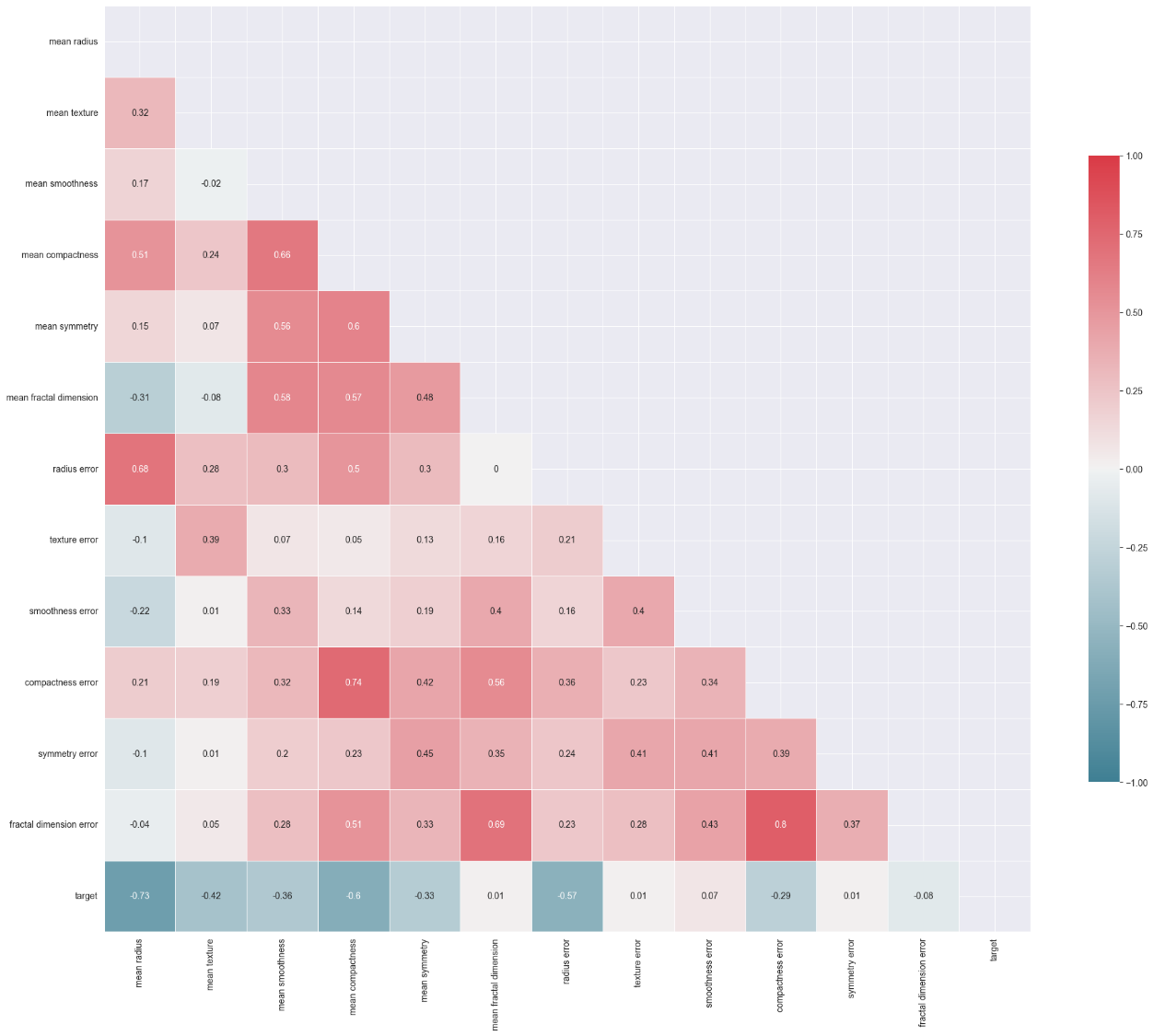
16. sns.heatmap(corr , mask = mask , cmap = cmap , vmin = -1 , vmax = 1 , center = 0 ,

17. square = True , linewidths = 0.5 , cbar\_kws = {"shrink" : 0.5} , annot = True)

18. plt.tight\_layout()

19.

Output:



**4. فیلتر T-Test:**

در این بخش، برای کاهش هرچه بیشتر ابعاد ویژگی‌ها، ما از فیلتر T-Test استفاده می‌کنیم. روش کار به صورت زیر می‌باشد:

1. def twoSampT(x1, x2, sd1, sd2, n1, n2):  
2. pooledSE = sqrt(sd1 \*\* 2 / n1 + sd2 \*\* 2 / n2)  
3. t\_test = (x1 - x2) / pooledSE p\_value = 2 \* (1 - norm.cdf(abs(t\_test)))  
4. return round(t\_test , 3) , p\_value

5.

1. cols = ['mean radius' ,  
2. 'mean texture' ,  
3. 'mean smoothness' ,  
4. 'mean compactness' ,  
5. 'mean symmetry' ,  
6. 'mean fractal dimension' ,  
7. 'radius error' ,  
8. 'texture error' ,  
9. 'smoothness error' ,  
10. 'compactness error' ,  
11. 'symmetry error' ,  
12. 'fractal dimension error']

13.

1. p\_values = {}  
2. for feature in cols:

3. # Filter the dataset for the two species we want to compare:

4. malignant\_mean = df.loc[df['target'] == 0 , feature].mean()

5. benign\_mean = df.loc[df['target'] == 1 , feature].mean()

6. malignant\_std = df.loc[df['target'] == 0 , feature].std()

7. benign\_std = df.loc[df['target'] == 1 , feature].std()

8. no\_of\_malignant = df.loc[df['target'] == 0 , feature].count()

9. no\_of\_benign = df.loc[df['target'] == 1 , feature].count()

10.

11. # Perform the t-test:

12. t\_test , p\_value = twoSampT(malignant\_mean ,benign\_mean , malignant\_std , benign\_std ,

13. no\_of\_malignant , no\_of\_benign)

14. print('T-Test =' , t\_test ,'P-Value =' , p\_value)

15.

16. # Interpret the results:

17. if p\_value < 0.025:

18. p\_values[feature] = p\_value

19. print(f"we reject null hypothesis for feature {feature}")

20. else:

21. df = df.drop(feature , axis = 1)

22. print(f"we accept null hypothesis for feature {feature}")

23.

Output:

T-Test = 22.209 P-Value = 0.0

we reject null hypothesis for feature mean radius

T-Test = 11.022 P-Value = 0.0

we reject null hypothesis for feature mean texture

T-Test = 9.297 P-Value = 0.0

we reject null hypothesis for feature mean smoothness

T-Test = 15.818 P-Value = 0.0

we reject null hypothesis for feature mean compactness

T-Test = 8.112 P-Value = 4.440892098500626e-16

we reject null hypothesis for feature mean symmetry

T-Test = -0.297 P-Value = 0.7665689002696578

we accept null hypothesis for feature mean fractal dimension

T-Test = 13.301 P-Value = 0.0

we reject null hypothesis for feature radius error

T-Test = -0.208 P-Value = 0.8353343609804973

we accept null hypothesis for feature texture error

T-Test = -1.623 P-Value = 0.10461735657236004

we accept null hypothesis for feature smoothness error

T-Test = 7.083 P-Value = 1.4144241333724494e-12

we reject null hypothesis for feature compactness error

T-Test = -0.142 P-Value = 0.8870364255851693

we accept null hypothesis for feature symmetry error

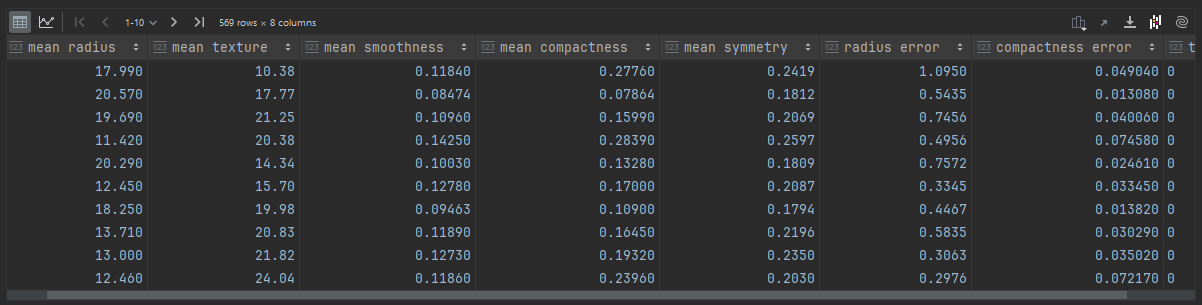
T-Test = 2.036 P-Value = 0.041726613882844266

we accept null hypothesis for feature fractal dimension error

1. df

2.

Output:



1. # Draw The Heatmap Again , With The New Correlation Matrix.  
2. # Generate And Visualize The Correlation Matrix.

3. corr = df.corr().round(2)

4.

5. # Mask For The Upper Triangle.

6. mask = np.zeros\_like(corr , dtype = np.bool\_)

7. mask[np.triu\_indices\_from(mask)] = True

8.

9. # Set Figure Size.

10. f , ax = plt.subplots(figsize = (20 , 20))

11.

12. # Define Custom Colormap.

13. cmap = sns.diverging\_palette(220 , 10 , as\_cmap = True)

14.

15. # Draw The Heatmap.

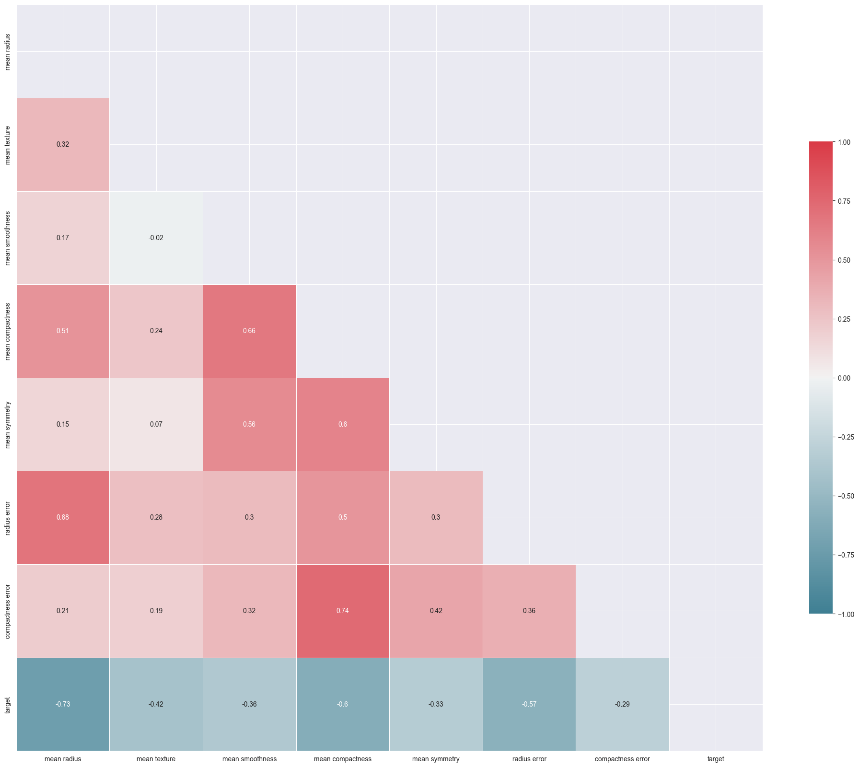
16. sns.heatmap(corr , mask = mask , cmap = cmap , vmin = -1 , vmax = 1 , center = 0 ,

17. square = True , linewidths = 0.5 , cbar\_kws = {"shrink" : 0.5} , annot = True)

18. plt.tight\_layout()

19.

Output:



**5. تقسیم بندی داده‌ها به دو بخش آموزش و آزمون (Train-Test Split):**

در این بخش، برای ساخت مدل مورد نظر، داده‌ها را به دو بخش آموزش و آزمون تقسیم می‌کنیم. روش کار به صورت زیر می‌باشد:

1. x\_train , x\_test , y\_train , y\_test = train\_test\_split(x , y , test\_size = 0.25 , random\_state =

2. 33)

1. ss = StandardScaler()  
2. x\_train = ss.fit\_transform(x\_train)  
3. x\_test = ss.fit\_transform(x\_test)

4.

**6. ساخت مدل KNN:**

در این بخش ما مدل نهایی خود را با استفاده از طبقه بند KNN ساخته و در انتها آن را ارزیابی می‌کنیم. روش کار به صورت زیر می‌باشد:

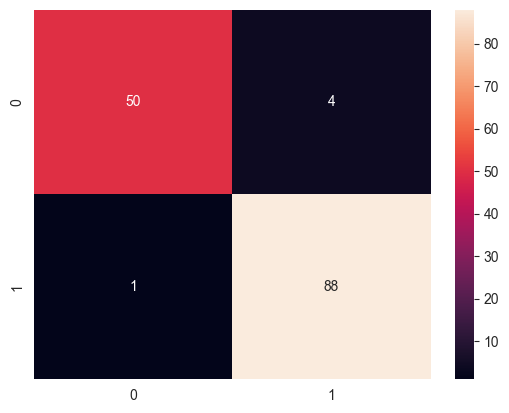
1. knn = neighbors.KNeighborsClassifier()  
2.  
3. model4 = knn.fit(x\_train , y\_train)  
4. y\_test\_pred4 = model4.predict(x\_test)  
5.  
6. cm = confusion\_matrix(y\_test , y\_test\_pred4)

7.

1. sns.heatmap(cm , annot = True)  
plt.savefig("h.png")

2.

Output:



1. accuracy = accuracy\_score(y\_test , y\_test\_pred4)  
2. precision = precision\_score(y\_test , y\_test\_pred4 , average = 'weighted')  
3. recall = recall\_score(y\_test , y\_test\_pred4 , average = 'weighted')  
4. f1 = f1\_score(y\_test , y\_test\_pred4 , average = 'weighted')

5.

1. report = classification\_report(y\_test , y\_test\_pred4 , target\_names = data["target\_names"])  
2. print(f"Accuracy = {accuracy \* 100 : .2f}%")  
3. print(f"Precision = {precision \* 100 : .2f}%")  
4. print(f"Recall = {recall \* 100 : .2f}%")  
5. print(f"F1 - Score = {f1 \* 100 : .2f}%")  
6. print("\nClassification Report:\n" , report)

7.

Output:

Accuracy = 96.50%

Precision = 96.55%

Recall = 96.50%

F1 - Score = 96.48%

Classification Report:

precision recall f1-score support

malignant 0.98 0.93 0.95 54

benign 0.96 0.99 0.97 89

accuracy 0.97 143

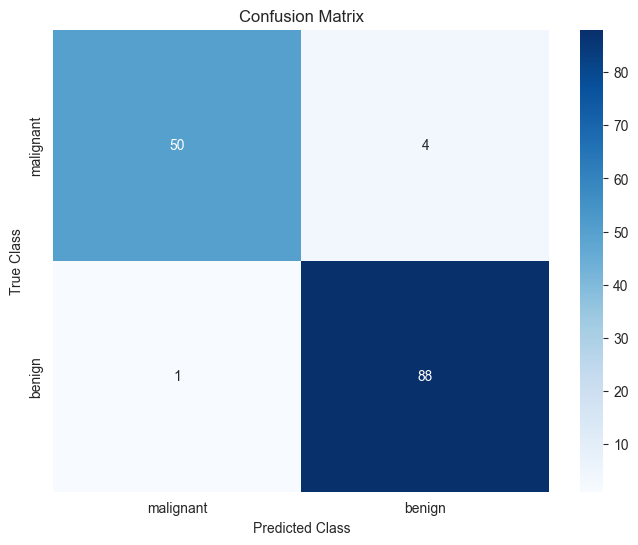
macro avg 0.97 0.96 0.96 143

weighted avg 0.97 0.97 0.96 143

1. plt.figure(figsize=(8 , 6))  
2. sns.heatmap(cm , annot = True , fmt = "d" , cmap = "Blues" , xticklabels = data["target\_names"] , 3. yticklabels = data["target\_names"])  
4. plt.title("Confusion Matrix")  
5. plt.xlabel("Predicted Class")  
6. plt.ylabel("True Class")  
7. plt.show()

8.

Output:



1. # Plotting The ROC curve

2. fpr,tpr, threshold = roc\_curve(y\_test , y\_test\_pred4)

3. roc\_auc\_ = auc(fpr,tpr)

4. plt.title('Receiver Operating Characteristics Curve')

5. plt.plot(fpr,tpr,'r',label = 'AUC = %0.3f' % roc\_auc\_)

6. plt.legend(loc='lower right')

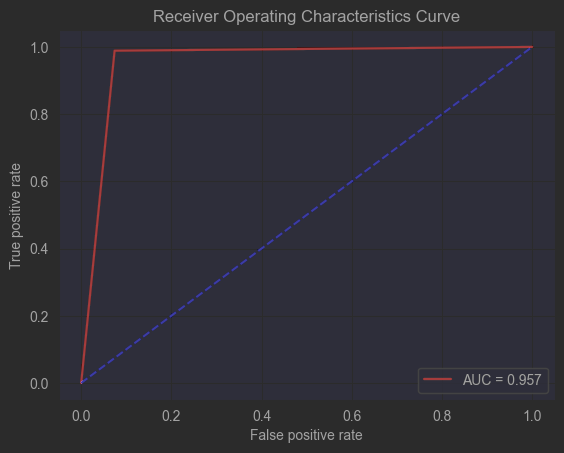
7. plt.plot([0,1],[0,1],'b--')

8. plt.ylabel('True positive rate')

9. plt.xlabel('False positive rate')

10.

Output:



1. knn\_score = knn.score(x\_train , y\_train)  
2. print(f"KNN Score is:{(knn\_score \* 100) : .2f}%")

3.

Output:

KNN Score is: 97.18%

1. ne\_neighbors = range(1 , 15)  
2. training\_accuracy = []  
3. testing\_accuracy = []  
4.   
5. for n in ne\_neighbors:  
6. knn = neighbors.KNeighborsClassifier(n\_neighbors = n)  
7. knn.fit(x\_train , y\_train)  
8. training\_accuracy.append(knn.score(x\_train , y\_train))  
9. testing\_accuracy.append(knn.score(x\_test , y\_test))

10.

Output:

