STATS 3DA3

Homework Assignment 6

xinyan su 400409412 yangkai chen 400322838

2024-04-04

1.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.impute import SimpleImputer
from scipy import stats
```

```
/var/folders/8q/_2fpqc5j70b5k8r5kydwm3800000gn/T/ipykernel_21238/478234604.py:1: DeprecationWa
Pyarrow will become a required dependency of pandas in the next major release of pandas (panda
(to allow more performant data types, such as the Arrow string type, and better interoperabili
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/54466

  import pandas as pd
```

```
import seaborn as sns
```

```
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn. pipeline import Pipeline
```

```
kidney = pd.read_csv("/Users/jolena/Desktop/kidney_disease.csv")
```

```
kidney = kidney.drop("id",axis=1)
```

```
kidney.describe()
```

	age	bp	sg	al	su	bgr	bu	sc
count	391.000000	388.000000	353.000000	354.000000	351.000000	356.000000	381.000000	383.000000

	age	bp	sg	al	su	bgr	bu	sc
mean	51.483376	76.469072	1.017408	1.016949	0.450142	148.036517	57.425722	3.072454
std	17.169714	13.683637	0.005717	1.352679	1.099191	79.281714	50.503006	5.741126
min	2.000000	50.000000	1.005000	0.000000	0.000000	22.000000	1.500000	0.400000
25%	42.000000	70.000000	1.010000	0.000000	0.000000	99.000000	27.000000	0.900000
50%	55.000000	80.000000	1.020000	0.000000	0.000000	121.000000	42.000000	1.300000
75%	64.500000	80.000000	1.020000	2.000000	0.000000	163.000000	66.000000	2.800000
max	90.000000	180.000000	1.025000	5.000000	5.000000	490.000000	391.000000	76.000000

```
kidney.dtypes
```

```
age              float64
bp               float64
sg               float64
al               float64
su               float64
rbc               object
pc                object
pcc               object
ba                object
bgr              float64
bu               float64
sc               float64
sod              float64
pot              float64
hemo             float64
pcv               object
wc                object
rc                object
htn               object
dm                object
```

```
cad               object
appet             object
pe                object
ane               object
classification    object
dtype: object
```

2.

```
kidney
```

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wc	rc
0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	121.0	...	44	7800	5.2
1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	NaN	...	38	6000	Na
2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	423.0	...	31	7500	Na
3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	117.0	...	32	6700	3.9
4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	106.0	...	35	7300	4.6
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
395	55.0	80.0	1.020	0.0	0.0	normal	normal	notpresent	notpresent	140.0	...	47	6700	4.9
396	42.0	70.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	75.0	...	54	7800	6.2
397	12.0	80.0	1.020	0.0	0.0	normal	normal	notpresent	notpresent	100.0	...	49	6600	5.4
398	17.0	60.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	114.0	...	51	7200	5.9
399	58.0	80.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	131.0	...	53	6800	6.1

```
float_col = kidney.select_dtypes(include=['float64']).columns
object_col = kidney.select_dtypes(include=['object']).columns
#Divide the dataset into two categories one is "float64" and one is "object".
```

```
mappings = {
    'rbc': {'normal': 1, 'abnormal': 0},
    'pc': {'normal': 1, 'abnormal': 0},
    'pcc': {'present': 1, 'notpresent': 0},
    'ba': {'present': 1, 'notpresent': 0},
    'htn': {'yes': 1, 'no': 0},
    'dm': {'yes': 1, 'no': 0},
    'cad': {'yes': 1, 'no': 0},
    'pe': {'yes': 1, 'no': 0},
    'ane': {'yes': 1, 'no': 0},
    'appet': {'good': 1, 'poor': 0},

}
```

```
for column, mapping in mappings.items():
    kidney[column] = kidney[column].replace(mapping)
```

```
scale = StandardScaler()
kidney[float_col] = scale.fit_transform(kidney[float_col])
```

3.

```
kidney.describe()
```

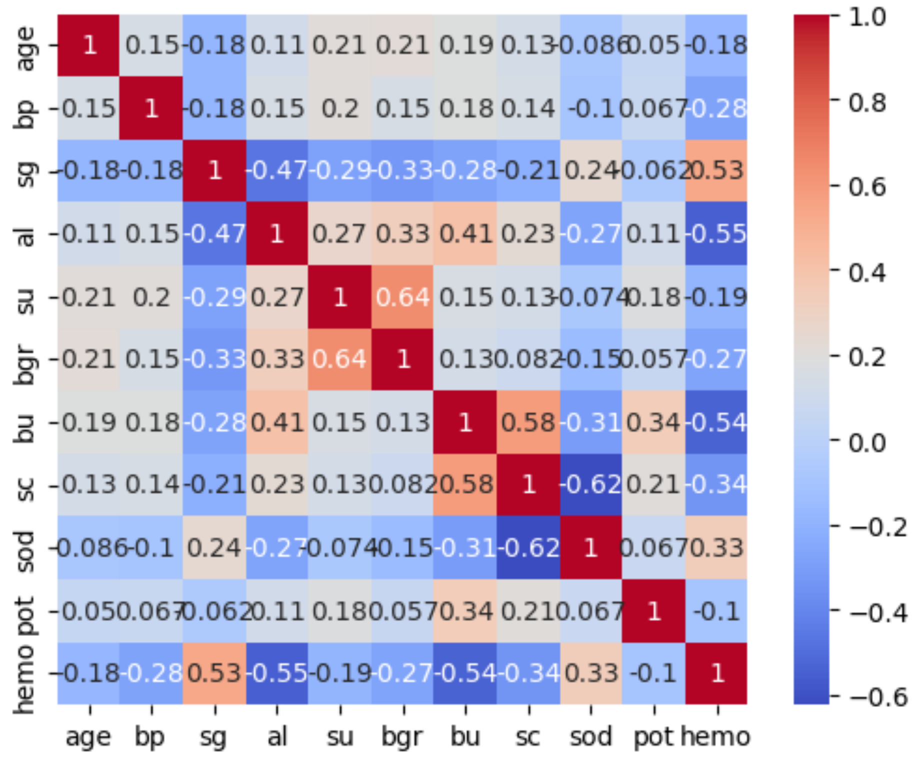
	age	bp	sg	al	su	rbc	pc
count	3.910000e+02	3.880000e+02	3.530000e+02	354.000000	351.000000	248.000000	335.000000
mean	9.994847e-17	-2.380684e-16	2.415443e-15	0.000000	0.000000	0.810484	0.773134
std	1.001281e+00	1.001291e+00	1.001419e+00	1.001415	1.001428	0.392711	0.419431
min	-2.885708e+00	-1.936857e+00	-2.173584e+00	-0.752868	-0.410106	0.000000	0.000000
25%	-5.530393e-01	-4.733701e-01	-1.297699e+00	-0.752868	-0.410106	1.000000	1.000000
50%	2.050779e-01	2.583733e-01	4.540705e-01	-0.752868	-0.410106	1.000000	1.000000
75%	7.590867e-01	2.583733e-01	4.540705e-01	0.727772	-0.410106	1.000000	1.000000
max	2.246163e+00	7.575807e+00	1.329955e+00	2.948733	4.145186	1.000000	1.000000

4.

```
df_float = Kidney[float_col]
df_float = df_float.apply(lambda x: x.fillna(x.mean()), axis=0)

co_matrix = df_float.corr()
co_matrix

sns.heatmap(co_matrix, annot=True, cmap='coolwarm', cbar=True, square=True)
```



5.

```
missing = kidney.isnull().sum()
missing
kidney_clean = kidney.dropna()
kidney_num = kidney_clean[float_col]
```

6.

```
z_scores = np.abs(stats.zscore(kidney_num))
outlier = np.where(z_scores > 3)
outlier
df_nooutlier =kidney_clean[(z_scores < 3).all(axis=1)]
outlier
```

```
(array([ 0,  1,  2,  6, 12, 18, 18, 19, 23, 23, 26, 31, 33, 34, 34, 36, 36,
        38, 38, 39, 40, 40, 40, 41, 41, 42, 42, 42]),
 array([ 8,  8,  5,  4,  7,  4,  9,  6,  6,  7,  8,  7,  7,  4,  7,  4,  5,
         7,  8,  4,  1,  6,  7,  4,  5,  6,  7, 10]))
```

7.

```
from sklearn.preprocessing import scale
from sklearn.decomposition import PCA, TruncatedSVD, FactorAnalysis
```

```
x = kidney_clean.drop('classification',axis=1)
```

```
pca_x = PCA()
pca_load = pd.DataFrame(pca_x.fit(x).components_.T, index=x .columns)
pca_load
```

	0	1	2	3	4	5	6	7	8
age	0.000044	0.020727	0.313500	-0.185826	-0.701004	-0.513627	0.273301	0.140848	-0.036657
bp	0.000002	0.035400	0.067652	0.046718	0.025120	0.407139	0.825058	-0.102388	-0.318178
sg	-0.000089	-0.069240	-0.349792	0.214633	-0.150371	0.007661	0.210613	-0.233942	0.372096
al	0.000105	0.086506	0.354798	-0.053853	0.245396	0.024628	0.049722	0.366635	-0.004227
su	0.000048	0.031270	0.400688	0.079648	-0.122333	0.389188	-0.105558	-0.128379	0.213501
rbc	-0.000011	-0.015295	-0.084867	0.066915	-0.038577	-0.061664	-0.031825	-0.004106	0.030667
pc	-0.000021	-0.031640	-0.083082	0.031712	-0.086061	0.050781	0.054878	-0.019013	0.036327
pcc	0.000013	0.016963	0.014968	-0.055576	0.046849	-0.010613	-0.009634	0.006357	-0.105535
ba	0.000014	0.011158	0.054319	-0.037182	0.024958	0.065924	-0.045905	0.035100	-0.091148

	0	1	2	3	4	5	6	7	8
bgr	0.000056	0.038115	0.443833	-0.095012	-0.201793	0.385907	-0.198867	-0.168903	0.207283
bu	0.000039	0.077062	0.184017	0.017902	0.353039	-0.296252	0.291868	-0.098321	0.599571
sc	0.000021	0.045384	0.096686	-0.044383	0.192678	-0.107065	0.186160	-0.052550	0.254785
sod	-0.000041	-0.045451	-0.108555	0.120149	-0.293493	0.043147	-0.032631	-0.391007	0.202584
pot	-0.000037	0.034069	0.238132	0.925938	-0.029697	-0.141365	-0.045239	0.123249	-0.126977
hemo	-0.000107	-0.090369	-0.160287	0.046926	-0.206142	0.178976	-0.009629	0.015822	0.042357
pcv	-0.001018	-0.975991	0.178541	-0.012622	0.099871	-0.046318	0.037977	-0.012559	-0.017505
wc	0.999999	-0.001040	-0.000003	0.000078	0.000032	-0.000009	0.000046	-0.000003	0.000011
rc	-0.000089	-0.081298	-0.223469	0.037148	-0.192072	0.298099	0.087366	0.741252	0.398699
htn	0.000030	0.034428	0.141817	-0.035623	-0.004630	0.003922	0.024883	0.000996	0.017065
dm	0.000035	0.026443	0.153101	-0.023298	-0.007350	0.026976	-0.039893	-0.014769	0.062120
cad	0.000002	0.011690	0.073261	-0.041899	-0.040208	0.048274	0.014196	-0.054750	0.011606
appet	-0.000034	-0.020677	-0.025783	0.053228	-0.060339	0.043245	0.019050	-0.003492	0.006716
pe	0.000030	0.021034	0.072622	-0.065014	0.069684	-0.057953	-0.027270	0.015960	0.041011
ane	0.000013	0.022616	0.004461	0.023172	0.096971	-0.078185	0.072109	0.004140	-0.007301

```
pc_score = pd.DataFrame(pca_x.fit_transform(x), index=x.index)
pc_score
```

	0	1	2	3	4	5	6	7	8
3	-1775.937410	12.277046	0.531766	-1.898433	1.606408	-0.467304	-0.989909	1.926934	-1.6827
9	3624.062410	9.510131	-0.930548	-0.411131	1.170972	-0.595923	1.413588	0.403137	-0.5305
11	-3975.936184	14.552231	1.909266	-1.708721	-0.669518	0.490025	-1.551092	0.242885	-0.4081
14	2524.076808	23.743046	-0.438817	-0.121706	-1.087686	0.015895	-0.771560	-0.255492	-0.7841
20	724.069302	17.572271	-0.506429	-0.386704	-0.123167	-0.798790	0.173911	-0.608048	0.3066
...	...	...	...	...	...	...	...	...	...
395	-1775.953785	-3.305705	0.422959	0.065001	-0.493713	-0.044379	0.425593	-0.643332	-0.1457
396	-675.961775	-11.542973	0.312832	-0.026240	0.646350	-0.042683	0.355971	0.448940	0.3878
397	-1875.955914	-5.251541	-0.331084	0.261720	1.673644	1.213478	-0.126453	-0.051150	-0.4963

	0	1	2	3	4	5	6	7	8
398	-1275.958300	-7.864024	-0.082125	0.452053	1.734872	0.316668	-0.801596	0.284339	0.71313
399	-1675.960104	-9.445848	0.804758	-0.316701	-0.289845	0.109278	0.905621	0.349483	0.06460

```
pc_score[0]
```

```
3     -1775.937410
9      3624.062410
11    -3975.936184
14     2524.076808
20      724.069302
          ...
395   -1775.953785
396    -675.961775
397   -1875.955914
398   -1275.958300
399   -1675.960104
Name: 0, Length: 158, dtype: float64
```

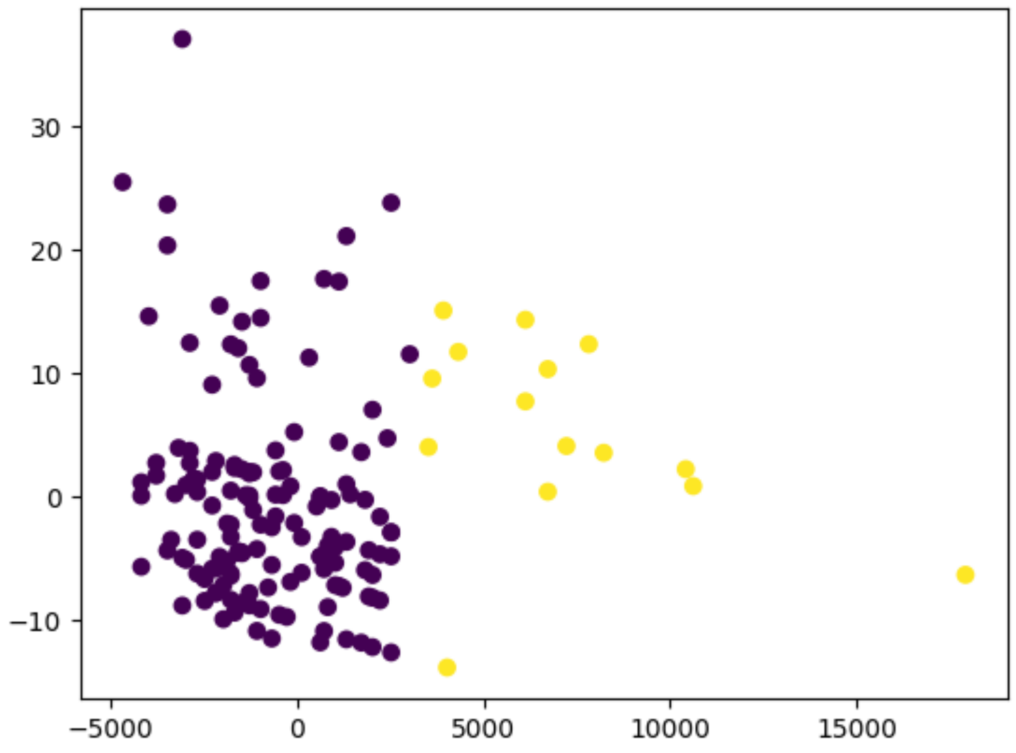
```
k_means = KMeans(n_clusters=2, n_init=20, random_state=0)
k_means.fit(x)
k_means.labels_
```

```
array([0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0,
       1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0], dtype=int32)
```

```
pd.Series(k_means.labels_).value_counts()
```

```
0    143
1     15
Name: count, dtype: int64
```

```
plt.scatter(pc_score[0], pc_score[1], c=k_means.labels_)
```



8.

```
from sklearn.model_selection import train_test_split
```

```
x_train, x_test, y_train, y_test = train_test_split(
    kidney_clean.drop('classification', axis=1),
    kidney_clean['classification'],
    test_size=0.3,
    random_state=1
)
```

9.

Random Forest: Offers high accuracy through ensemble learning, can handle various data types, and provides feature importance which is useful for interpretation in medical datasets. K-Nearest Neighbors (KNN): Simple and effective for small datasets, requires no model assumptions, and can quickly adapt to new data, which is advantageous in dynamic medical environments.

10.

Accuracy: This is a measure of the number of correct predictions made by the model divided by the total number of predictions. It's a general indicator of a model's performance. F1 Score: The F1 score is the harmonic mean of precision and recall, providing a balance between the two. It is particularly useful when the class distribution is uneven, as it accounts for both false positives and false negatives. Accuracy= Total Number of Predictions/Number of Correct Predictions = (TP+TN+FP+FN)/(TP+TN) F1 Score=2$\times$ (Precision+Recall)/Precision$\times$Recall Precision= (TP+FP)/TP Recall= (TP+FN)/TP where TP is True Positives, TN is True Negatives, FP is False Positives, and FN is False Negatives.

11.

With Random Forest, we can utilize the built-in feature importance to identify which features contribute most to the prediction. For KNN, feature selection is crucial because it relies on distance measurements; irrelevant features can disrupt its performance. Techniques like Sequential Feature Selector or using correlation metrics can be helpful.

12.

Logistic Regression: Accuracy of 97.5% and F1 Score of 0.975 K-Nearest Neighbors: Accuracy of 81.67% and F1 Score of 0.818 Random Forest: Accuracy of 100% and F1 Score of 1.0 The Random Forest classifier has achieved perfect scores on the test set, which suggests excellent performance. However, this could also be a sign of overfitting, and further investigation such as cross-validation would be recommended to confirm these results. The Logistic Regression also performed very well, with high accuracy and F1 score. The K-Nearest Neighbors classifier showed lower performance compared to the other two.

13.

The feature importances from the Random Forest classifier are as follows for the top five features: a. Hemoglobin (hemo): 11.59% importance b. Serum Creatinine (sc): 10.36% importance c. Specific Gravity (sg): 8.71% importance d. Albumin (al): 7.69% importance e. Hypertension No (htn_no): 7.65% importance These features are the most influential in predicting kidney disease according to the Random Forest model. Hemoglobin level is the most significant predictor, followed closely by serum creatinine, which is a waste product in the blood that kidneys filter out. Specific gravity is a measure related to urine concentration, and albumin is a protein that can be present in the urine and can indicate kidney health. The presence or absence of hypertensionl also plays a significant role in the model's predictions.

15.xinyan su 1-9 yangkai chen 10-13

16. https://github.com/Wasabixm/stats3da3-group-wasabi.git