lab8

September 2, 2024

[123]:
```
import pandas as pd
import numpy as np
```

[124]:
```
column_name=["Age","Sex","cp","trestbps","chol","fbs","restecg","thalach","exang","oldpeak","s
```

[125]:
```
dataset = pd.read_csv("processed.cleveland.data",names=column_name,header=None)
```

[126]:
```
df = pd.DataFrame(dataset)
df.tail()
```

[126]:

	Age	Sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	\
298	45.0	1.0	1.0	110.0	264.0	0.0	0.0	132.0	0.0	1.2	
299	68.0	1.0	4.0	144.0	193.0	1.0	0.0	141.0	0.0	3.4	
300	57.0	1.0	4.0	130.0	131.0	0.0	0.0	115.0	1.0	1.2	
301	57.0	0.0	2.0	130.0	236.0	0.0	2.0	174.0	0.0	0.0	
302	38.0	1.0	3.0	138.0	175.0	0.0	0.0	173.0	0.0	0.0	

	slope	ca	thal	num
298	2.0	0.0	7.0	1
299	2.0	2.0	7.0	2
300	2.0	1.0	7.0	3
301	2.0	1.0	3.0	1
302	1.0	?	3.0	0

[127]:
```
df.isnull().sum()
```

[127]:
```
Age         0
Sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
```

```
ca          0
thal        0
num         0
dtype: int64
```

[128]:
```
df['num'] = [1 if int(num) in [2,3,4] else 0 for num in df['num']]
```

[129]:
```
df['thal'].unique()
```

[129]:
```
array(['6.0', '3.0', '7.0', '?'], dtype=object)
```

[130]:
```
df['ca'] = [np.nan if val == '?' else val for val in df['ca']]
df['ca'] = [int(float(val)) if val in ['0.0', '3.0', '2.0', '1.0'] else val for␣
 ↪val in df['ca']]
df['ca'].fillna(value=df['ca'].mean(),inplace=True)
```

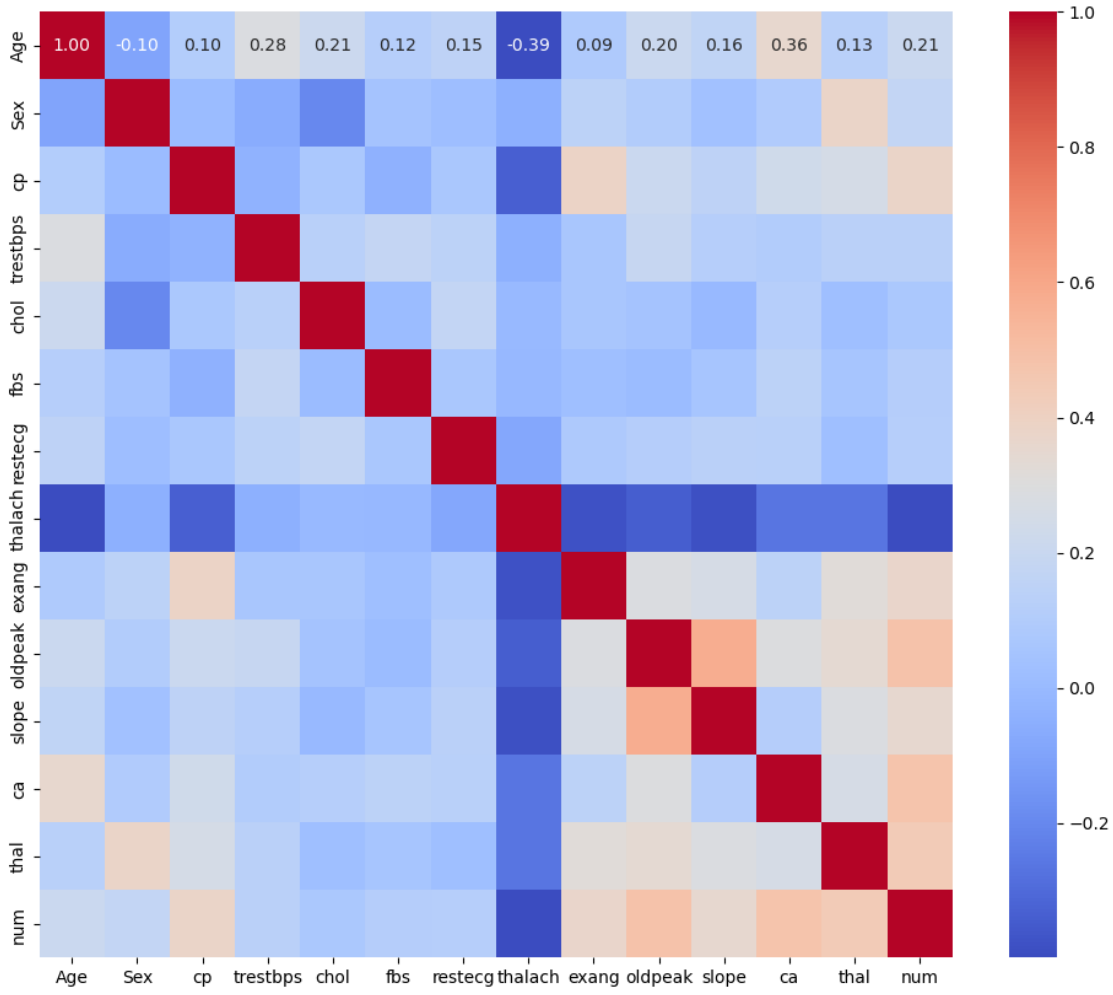
[131]:
```
df['thal'] = [np.nan if val == '?' else val for val in df['thal']]
df['thal'] = [int(float(val)) if val in ['6.0', '3.0', '7.0'] else val for val␣
 ↪in df['thal']]
df['thal'].fillna(value=df['ca'].mean(),inplace=True)
```

[133]:
```
# Separate features and target
X = df.drop(columns='num')
y = df['num']
```

[135]:
```
import seaborn as sns
import matplotlib.pyplot as plt

corr = df.corr()

# Plot the heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt='.2f')
plt.show()
```



[136]:
```
from sklearn.model_selection import train_test_split

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[139]:
```
from sklearn.preprocessing import StandardScaler

# Normalize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[143]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
```

```
model = LogisticRegression()
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)

print(f"Training Score: {model.score(X_train_scaled, y_train)}")

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Training Score: 0.9008264462809917
Confusion Matrix:
[[34  7]
 [ 7 13]]

Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.83      0.83        41
           1       0.65      0.65      0.65        20

    accuracy                           0.77        61
   macro avg       0.74      0.74      0.74        61
weighted avg       0.77      0.77      0.77        61
```

[144]:
```
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV

# Define parameter grid
param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear', 'saga']
}

# Randomized Search
random_search = RandomizedSearchCV(LogisticRegression(max_iter=1000),␣
 ↪param_distributions=param_grid, n_iter=10, cv=5, random_state=42)
random_search.fit(X_train_scaled, y_train)
print("Best parameters from RandomizedSearchCV:")
print(random_search.best_params_)

# Grid Search
grid_search = GridSearchCV(LogisticRegression(max_iter=1000), param_grid, cv=5)
```

```
grid_search.fit(X_train_scaled, y_train)
print("Best parameters from GridSearchCV:")
print(grid_search.best_params_)
```

```
Best parameters from RandomizedSearchCV:
{'solver': 'liblinear', 'penalty': 'l1', 'C': 10}
Best parameters from GridSearchCV:
{'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}
```