American International University-Bangladesh

Department of Computer Science

Faculty of Science &Technology (FST)

Fall: 23-24

DATA WAREHOUSING AND DATA MINING [B]

Project Name: Heart Attack Analysis & Prediction. (Feature selection by ARM)

Name: Md. Minhazul Bari Fahim

ID: 20-42176-1; **Section**: B

Project Overview:

The project involves the analysis and prediction of heart attacks based on a dataset containing various health-related features. The dataset includes information such as age, sex, exercise-induced angina, number of major vessels, chest pain type, resting blood pressure, cholesterol levels, fasting blood sugar, resting electrocardiographic results, maximum heart rate achieved, and a target variable indicating the likelihood of a heart attack (0 for less chance, 1 for more chance). The goal is to build a predictive model that can accurately classify individuals into these two categories, helping in early detection and prevention of heart attacks. The project will utilize Association Rule Mining (ARM) for feature selection, aiming to identify the most relevant features that contribute to the prediction of heart attacks. The selected features will then be used to train machine learning models for heart attack prediction.

Dataset Overview:

The dataset consists of health-related attributes collected from individuals, with a focus on factors associated with heart health. The features include demographic information such as age and sex, lifestyle indicators like exercise-induced angina and fasting blood sugar, as well as medical measurements like resting blood pressure, cholesterol levels, and maximum heart rate achieved. The dataset is labelled, with the target variable indicating whether an individual has a higher or lower chance of experiencing a heart attack. The diverse set of features provides a comprehensive overview of potential risk factors associated with heart health. The use of Association Rule Mining (ARM) will help in identifying patterns and relationships among these features, contributing to the selection of key variables for building an effective predictive model for heart attack risk assessment.

Source:https://www.kaggle.com/datasets/rashikrahmanpritom/heart-attack-analysis-prediction-dataset/data

Importing all the necessary libraries and then loading our dataset into jupyter notebook.

```
[1]  import numpy as np
     import pandas as pd
     import os
     for dirname, _, filenames in os.walk('/kaggle/input'):
         for filename in filenames:
             print(os.path.join(dirname, filename))
```

```
[74] pip install scikeras
```

```
[73] pip install kmodes
```

```
[62] import matplotlib.pyplot as plt
     import seaborn as sns
     from mlxtend.frequent_patterns import apriori
     from mlxtend.frequent_patterns import association_rules
     from mlxtend.preprocessing import TransactionEncoder
     from sklearn.model_selection import train_test_split
     import time
     from sklearn.metrics import accuracy_score
```

```
     # import libraries of models
     from sklearn.model_selection import GridSearchCV
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.tree import DecisionTreeClassifier
     from sklearn.neighbors import KNeighborsClassifier
     from scikeras.wrappers import KerasClassifier
     from sklearn.linear_model import LogisticRegression
     from kmodes.kmodes import KModes


     # keras
     from keras.models import Sequential
     from keras.layers import Dense
```

```
[70] df_const = pd.read_csv('heart.csv')
     df = df_const.copy(deep=True)
```

Overview of the dataset

```
df_const.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       303 non-null    int64
 1   sex       303 non-null    int64
 2   cp        303 non-null    int64
 3   trtbps    303 non-null    int64
 4   chol      303 non-null    int64
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalachh  303 non-null    int64
 8   exng      303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slp       303 non-null    int64
 11  caa       303 non-null    int64
 12  thall     303 non-null    int64
 13  output    303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

Missing Values

```
df_const.isnull().sum()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/i
  and should_run_async(code)
age         0
sex         0
cp          0
trtbps      0
chol        0
fbs         0
restecg     0
thalachh    0
exng        0
oldpeak     0
slp         0
caa         0
thall       0
output      0
dtype: int64
```

The data is very clean already and there are no missing data.

Sample

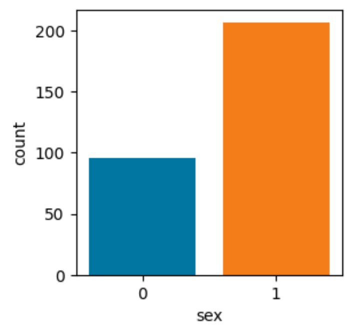
```
df_const.sample(10)
```

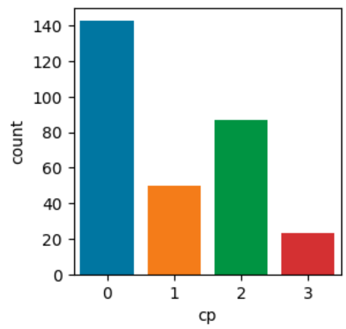
```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipykernel.py:283: DeprecationWarning: `should_run_async`
  and should_run_async(code)
```

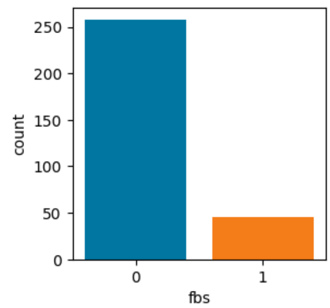
	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
14	58	0	3	150	283	1	0	162	0	1.0	2	0	2	1
141	43	1	0	115	303	0	1	181	0	1.2	1	0	2	1
121	59	1	0	138	271	0	0	182	0	0.0	2	0	2	1
19	69	0	3	140	239	0	1	151	0	1.8	2	2	2	1
45	52	1	1	120	325	0	1	172	0	0.2	2	0	2	1
154	39	0	2	138	220	0	1	152	0	0.0	1	0	2	1
251	43	1	0	132	247	1	0	143	1	0.1	1	4	3	0
89	58	0	0	100	248	0	0	122	0	1.0	1	0	2	1
163	38	1	2	138	175	0	1	173	0	0.0	2	4	2	1
94	45	0	1	112	160	0	1	138	0	0.0	1	0	2	1

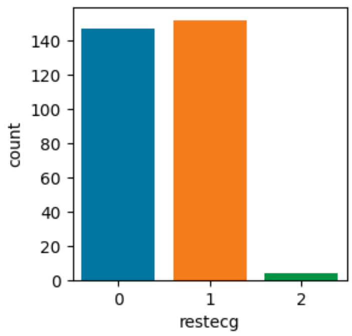
Understand the data better by data visualization.

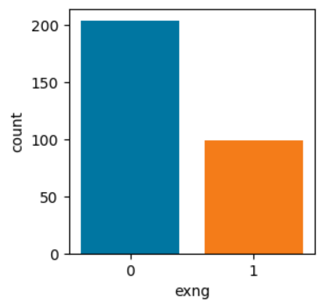
```
for i in df_const[['sex', 'cp', 'fbs', 'restecg',
          'exng', 'slp', 'caa', 'thall', 'output']]:
    plt.figure(figsize=(3,3))
    sns.countplot(x=i,data=df_const)
plt.show()
```

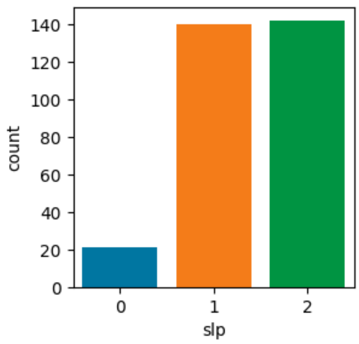


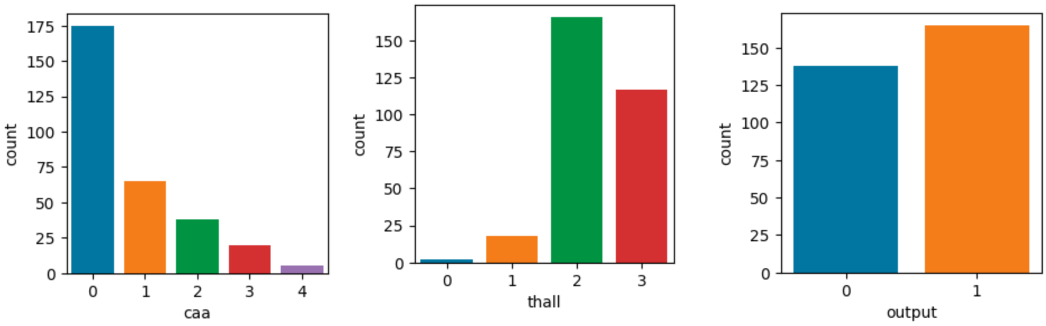




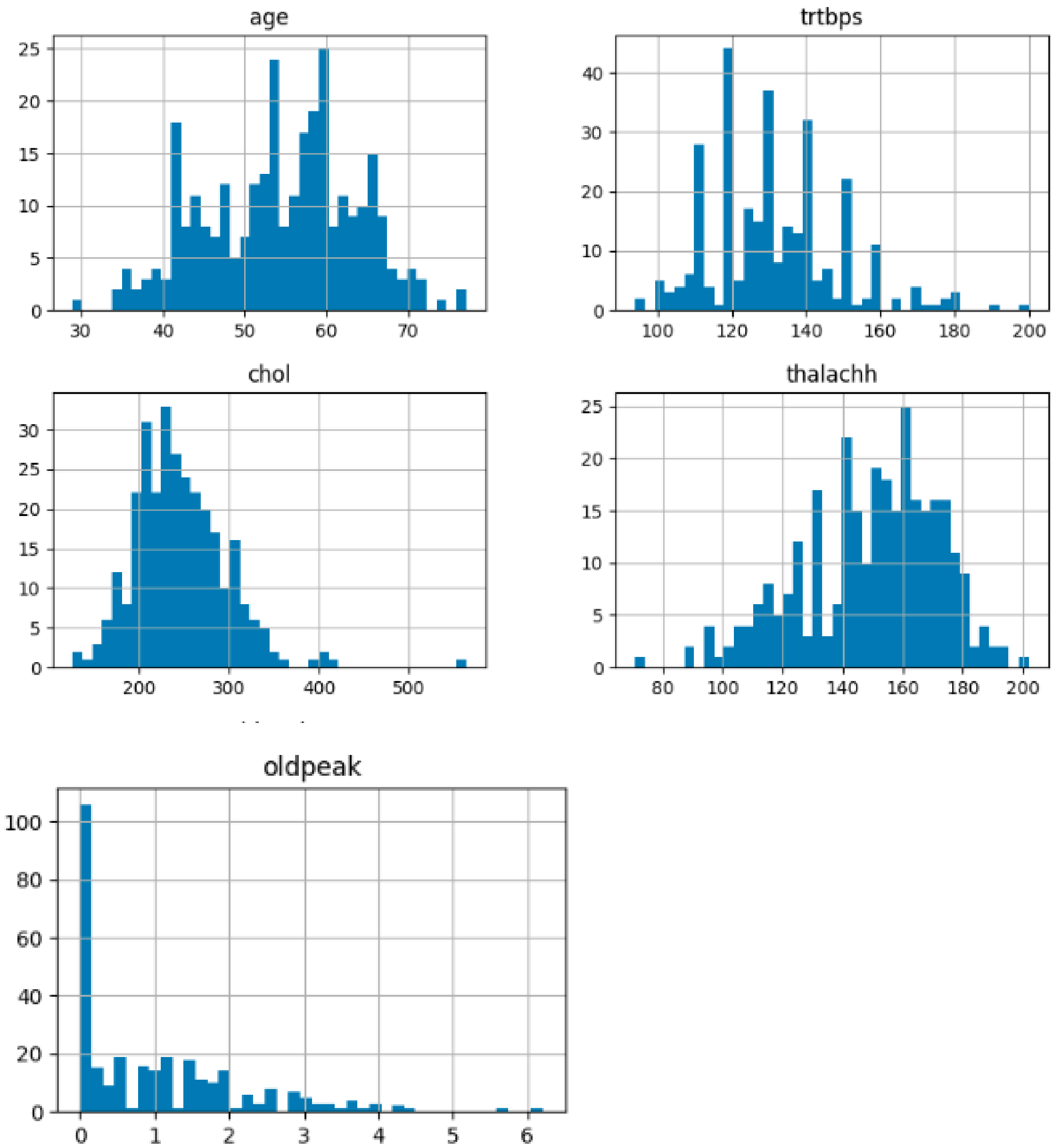






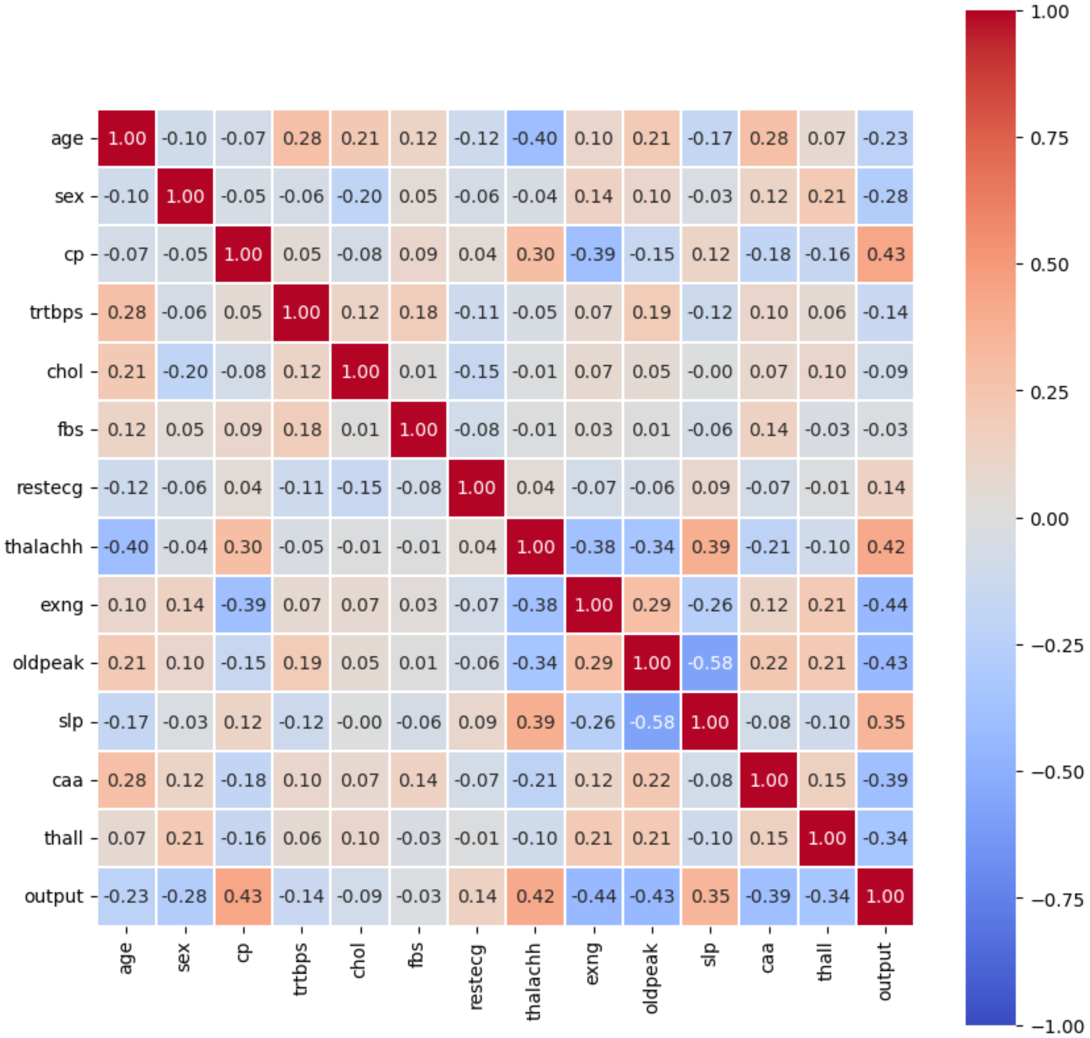



```
df_const[['age', 'trtbps', 'chol', 'thalachh', 'oldpeak']].hist(bins=40 , figsize=(10,10))
plt.show()
```



Correlation heatmap

```
[12] corr = df_const.corr()
     plt.figure(figsize=(10,10))
     ax = sns.heatmap(corr, vmin=-1, vmax=1, cmap="coolwarm", linewidths=.1, square=True, annot=True ,fmt=".2f")
     plt.yticks(rotation=0)
     plt.show()
```




```
corr['output'].sort_values(ascending=False)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkerne
  and should_run_async(code)
output      1.000000
cp          0.433798
thalachh    0.421741
slp         0.345877
restecg     0.137230
fbs        -0.028046
chol       -0.085239
trtbps     -0.144931
age        -0.225439
sex        -0.280937
thall      -0.344029
caa        -0.391724
oldpeak    -0.430696
exng       -0.436757
Name: output, dtype: float64
```

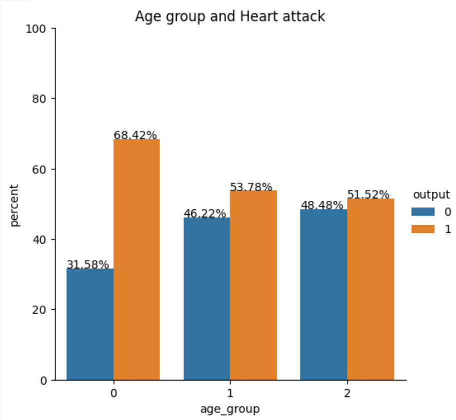
Feature Engineering

The numeric features should be removed and transformed through

➢ Age

```
[78] def encode_age(age):
         #adults < 40
         if age <= 40.0:
             return 0
         # middle age 41-65
         elif 40.0 < age <= 65.0:
             return 1
         # eldery > 65
         else:
             return 2

     # encode the Age to Age_group
     df['age_group'] = df['age'].apply(encode_age)
```

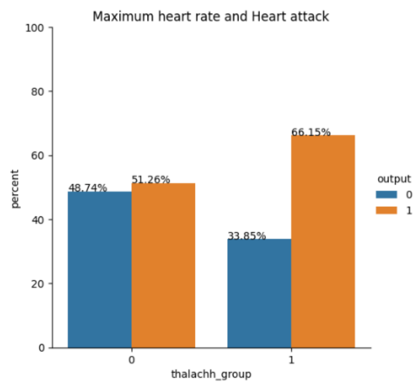
```
x,y = 'age_group', 'output'
df1 = df.groupby(x)[y].value_counts(normalize=True)
df1 = df1.mul(100)
df1 = df1.rename('percent').reset_index()

g = sns.catplot(x=x,y='percent',hue=y,kind='bar',data=df1)
g.ax.set_ylim(0,100)
plt.title('Age group and Heart attack')

for p in g.ax.patches:
    txt = str(p.get_height().round(2)) + '%'
    txt_x = p.get_x()
    txt_y = p.get_height()
    g.ax.text(txt_x,txt_y,txt)
```



➢ Maximum Heart rate

```
[19] def encode_thalachh(age, thalachh):
         max_heartrate = 220.0-age
         # normal
         if thalachh <= max_heartrate:
             return 0
         # abnormal
         else:
             return 1

     # encode the thalachh to thalachh_group
     df['thalachh_group'] = df.apply(lambda x: encode_thalachh(x['age'], x['thalachh']), axis=1)
```

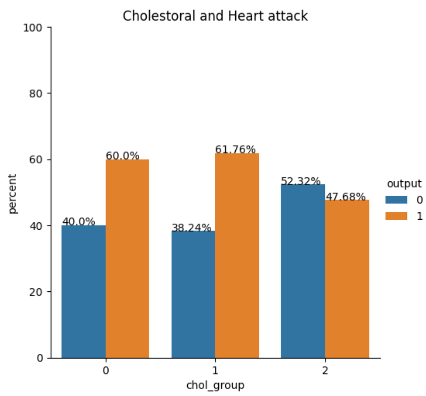
```python
[20] x,y = 'thalachh_group', 'output'
     df1 = df.groupby(x)[y].value_counts(normalize=True)
     df1 = df1.mul(100)
     df1 = df1.rename('percent').reset_index()

     g = sns.catplot(x=x,y='percent',hue=y,kind='bar',data=df1)
     g.ax.set_ylim(0,100)
     plt.title('Maximum heart rate and Heart attack')

     for p in g.ax.patches:
         txt = str(p.get_height().round(2)) + '%'
         txt_x = p.get_x()
         txt_y = p.get_height()
         g.ax.text(txt_x,txt_y,txt)
```



- **Cholesterol.**

```python
def encode_chol(chol):
    # normal
    if chol < 200.0:
        return 0
    # borderline high
    elif 200.0 <= chol <= 240.0:
        return 1
    # abnormal
    else:
        return 2

# encode the cholesterol to cholesterol_group
df['chol_group'] = df['chol'].apply(encode_chol)
```

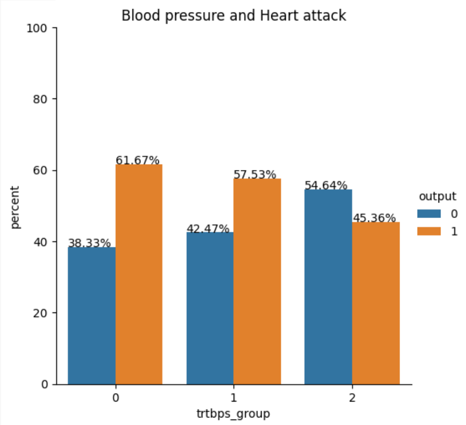
```python
x,y = 'chol_group', 'output'
df1 = df.groupby(x)[y].value_counts(normalize=True)
df1 = df1.mul(100)
df1 = df1.rename('percent').reset_index()

g = sns.catplot(x=x,y='percent',hue=y,kind='bar',data=df1)
g.ax.set_ylim(0,100)
plt.title('Cholestoral and Heart attack')

for p in g.ax.patches:
    txt = str(p.get_height().round(2)) + '%'
    txt_x = p.get_x()
    txt_y = p.get_height()
    g.ax.text(txt_x,txt_y,txt)
```



- **Blood Pressure**

```python
def encode_trtbps(trtbps):
    # normal
    if trtbps < 120.0:
        return 0
    # pre-hypertension
    elif 120.0 <= trtbps < 140.0:
        return 1
    # hypertension
    else:
        return 2

# encode the cholesterol to cholesterol_group
df['trtbps_group'] = df['trtbps'].apply(encode_trtbps)
```

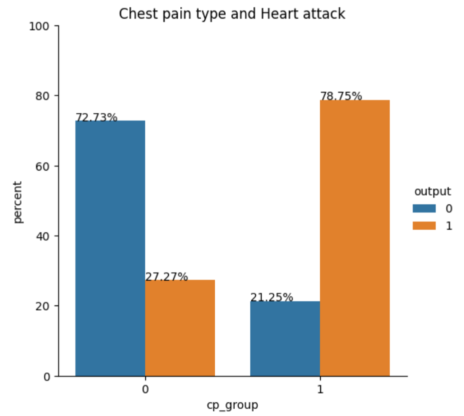
```python
x,y = 'trtbps_group', 'output'
df1 = df.groupby(x)[y].value_counts(normalize=True)
df1 = df1.mul(100)
df1 = df1.rename('percent').reset_index()

g = sns.catplot(x=x,y='percent',hue=y,kind='bar',data=df1)
g.ax.set_ylim(0,100)
plt.title('Blood pressure and Heart attack')

for p in g.ax.patches:
    txt = str(p.get_height().round(2)) + '%'
    txt_x = p.get_x()
    txt_y = p.get_height()
    g.ax.text(txt_x,txt_y,txt)
```



➢ Chest Pain

```python
[25] def encode_cp(cp):
       # typical angina
       if cp == 0:
           return 0
       # the other 3
       else:
           return 1

     # encode the cholesterol to cholesterol_group
     df['cp_group'] = df['cp'].apply(encode_cp)
```

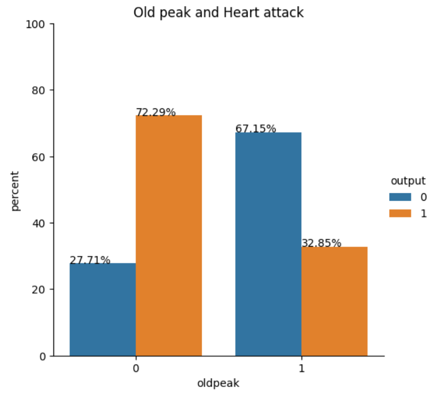
```python
x,y = 'cp_group', 'output'
df1 = df.groupby(x)[y].value_counts(normalize=True)
df1 = df1.mul(100)
df1 = df1.rename('percent').reset_index()

g = sns.catplot(x=x,y='percent',hue=y,kind='bar',data=df1)
g.ax.set_ylim(0,100)
plt.title('Chest pain type and Heart attack')

for p in g.ax.patches:
    txt = str(p.get_height().round(2)) + '%'
    txt_x = p.get_x()
    txt_y = p.get_height()
    g.ax.text(txt_x,txt_y,txt)
```



➢ Old Peak

```python
def encode_oldpeak(oldpeak):
    # normal
    if oldpeak < 1.0:
        return 0
    #abnormal
    else:
        return 1

# repalce the oldpeak column
df['oldpeak'] = df['oldpeak'].apply(encode_oldpeak)
```

```python
x,y = 'oldpeak', 'output'
df1 = df.groupby(x)[y].value_counts(normalize=True)
df1 = df1.mul(100)
df1 = df1.rename('percent').reset_index()

g = sns.catplot(x=x,y='percent',hue=y,kind='bar',data=df1)
g.ax.set_ylim(0,100)
plt.title('Old peak and Heart attack')

for p in g.ax.patches:
    txt = str(p.get_height().round(2)) + '%'
    txt_x = p.get_x()
    txt_y = p.get_height()
    g.ax.text(txt_x,txt_y,txt)
```



Drop unwanted column

```python
df.drop(columns=['age', 'chol', 'thalachh', 'trtbps'], inplace=True)
```

```python
[30] df.head()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell
  and should_run_async(code)
```

	sex	cp	fbs	restecg	exng	oldpeak	slp	caa	thall	output	age_group	thalachh_group	chol_group	trtbps_group	cp_group
0	1	3	1	0	0	1	0	0	1	1	1	0	1	2	1
1	1	2	0	1	0	1	0	0	2	1	0	1	2	1	1
2	0	1	0	0	0	1	2	0	2	1	1	0	1	1	1
3	1	1	0	1	0	0	2	0	2	1	1	1	1	1	1
4	0	0	0	1	1	0	2	0	2	1	1	0	2	1	0

Feature selection by Association Rule Mining

➢ Data Preparation

```python
df_nohead=df.copy(deep=True)

# add column as prefix for each value to indentify them in rules
for x in df_nohead.columns.values.tolist():
    df_nohead[x] = x + '_' + df_nohead[x].astype(str)

df_nohead.columns = range(df_nohead.shape[1])
df_nohead
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	sex_1	cp_3	fbs_1	restecg_0	exng_0	oldpeak_1	slp_0	caa_0	thall_1	output_1	age_group_1	thalachh_group_0	chol_group_1	trtbps_group_2	cp_group_1
1	sex_1	cp_2	fbs_0	restecg_1	exng_0	oldpeak_1	slp_0	caa_0	thall_2	output_1	age_group_0	thalachh_group_1	chol_group_2	trtbps_group_1	cp_group_1
2	sex_0	cp_1	fbs_0	restecg_0	exng_0	oldpeak_1	slp_2	caa_0	thall_2	output_1	age_group_1	thalachh_group_0	chol_group_1	trtbps_group_1	cp_group_1
3	sex_1	cp_1	fbs_0	restecg_1	exng_0	oldpeak_0	slp_2	caa_0	thall_2	output_1	age_group_1	thalachh_group_1	chol_group_1	trtbps_group_1	cp_group_1
4	sex_0	cp_0	fbs_0	restecg_1	exng_1	oldpeak_0	slp_2	caa_0	thall_2	output_1	age_group_1	thalachh_group_0	chol_group_2	trtbps_group_1	cp_group_0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	sex_0	cp_0	fbs_0	restecg_1	exng_1	oldpeak_0	slp_1	caa_0	thall_3	output_0	age_group_1	thalachh_group_0	chol_group_2	trtbps_group_2	cp_group_0
299	sex_1	cp_3	fbs_0	restecg_1	exng_0	oldpeak_1	slp_1	caa_0	thall_3	output_0	age_group_1	thalachh_group_0	chol_group_2	trtbps_group_0	cp_group_1
300	sex_1	cp_0	fbs_1	restecg_1	exng_0	oldpeak_1	slp_1	caa_2	thall_3	output_0	age_group_2	thalachh_group_0	chol_group_0	trtbps_group_2	cp_group_0
301	sex_1	cp_0	fbs_0	restecg_1	exng_1	oldpeak_1	slp_1	caa_1	thall_3	output_0	age_group_1	thalachh_group_0	chol_group_0	trtbps_group_1	cp_group_0
302	sex_0	cp_1	fbs_0	restecg_0	exng_0	oldpeak_0	slp_1	caa_1	thall_2	output_0	age_group_1	thalachh_group_1	chol_group_1	trtbps_group_1	cp_group_1

```python
te = TransactionEncoder()
transactions = df_nohead.values.tolist()
te_ary = te.fit_transform(transactions)
result_df = pd.DataFrame(te_ary, columns=te.columns_)
result_df
```

	age_group_0	age_group_1	age_group_2	caa_0	caa_1	caa_2	caa_3	caa_4	chol_group_0	chol_group_1	...	slp_2	thalachh_group_0	thalachh_group_1	thall_0	thall_1	thall_2	thall_3	trtbps_group_0	trtbps_group_1	trtbps_group_2
0	False	True	False	True	False	False	False	False	False	True	...	False	True	False	False	True	False	False	False	False	True
1	True	False	False	True	False	False	False	False	False	False	...	False	False	True	False	False	True	False	False	True	False
2	False	True	False	True	False	False	False	False	False	True	...	True	True	False	False	False	True	False	False	True	False
3	False	True	False	True	False	False	False	False	False	True	...	True	False	True	False	False	True	False	False	True	False
4	False	True	False	True	False	False	False	False	False	False	...	True	True	False	False	False	True	False	False	True	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	False	True	False	True	False	False	False	False	False	False	...	False	True	False	False	False	False	True	False	False	True
299	False	True	False	True	False	False	False	False	False	False	...	False	True	False	False	False	False	True	True	False	False
300	False	False	True	False	False	True	False	False	True	False	...	False	True	False	False	False	False	True	False	False	True
301	False	True	False	False	True	False	False	False	True	False	...	False	True	False	False	False	False	True	False	True	False
302	False	True	False	False	True	False	False	False	False	True	...	False	False	True	False	False	True	False	False	True	False

➢ Apriori Algorithm

```python
freq_items = apriori(result_df, min_support=0.3, use_colnames=True)
freq_items.head(10)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipykernel.py:283: D
  and should_run_async(code)
```

	support	itemsets
0	0.828383	(age_group_1)
1	0.577558	(caa_0)
2	0.336634	(chol_group_1)
3	0.498350	(chol_group_2)
4	0.471947	(cp_0)
5	0.471947	(cp_group_0)
6	0.528053	(cp_group_1)
7	0.673267	(exng_0)
8	0.326733	(exng_1)
9	0.851485	(fbs_0)

```
rules = association_rules(freq_items, metric="support", min_threshold=0.3)
rules.sort_values("support", ascending=False).head(5)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automatica
  and should_run_async(code)
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
13	(age_group_1)	(fbs_0)	0.828383	0.851485	0.699670	0.844622	0.991939	-0.005686	0.955826	-0.045210
12	(fbs_0)	(age_group_1)	0.851485	0.828383	0.699670	0.821705	0.991939	-0.005686	0.962548	-0.051878
140	(fbs_0)	(thalachh_group_0)	0.851485	0.785479	0.676568	0.794574	1.011579	0.007744	1.044274	0.077073
141	(thalachh_group_0)	(fbs_0)	0.785479	0.851485	0.676568	0.861345	1.011579	0.007744	1.071107	0.053358
32	(thalachh_group_0)	(age_group_1)	0.785479	0.828383	0.650165	0.827731	0.999213	-0.000512	0.996217	-0.003657

➤ Data Filtering and visualization

```
[36] rules['consequents_len'] = rules['consequents'].apply(lambda x: len(x))
     rules
```

```
[37] sorted_rules = rules[(rules['consequents_len'] == 1)]
     sorted_rules = sorted_rules.sort_values("lift", ascending=False)
```

```
[38] sorted_rules.drop_duplicates(subset=['antecedents', 'consequents'], inplace=True)
```

```
[39] sorted_rules[sorted_rules['consequents'].astype(str).str.contains("output")].head(10)
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric	consequents_len
634	(caa_0, thall_2)	(output_1)	0.376238	0.544554	0.336634	0.894737	1.643062	0.131752	4.326733	0.627451	1
753	(cp_group_1, thall_2)	(output_1)	0.376238	0.544554	0.333333	0.885965	1.626954	0.128451	3.993907	0.617790	1
1186	(fbs_0, caa_0, thall_2)	(output_1)	0.343234	0.544554	0.303630	0.884615	1.624476	0.116721	3.947195	0.585318	1
1242	(cp_group_1, exng_0, thall_2)	(output_1)	0.343234	0.544554	0.300330	0.875000	1.606818	0.113420	3.643564	0.575018	1
555	(cp_group_1, caa_0)	(output_1)	0.363036	0.544554	0.316832	0.872727	1.602645	0.119139	3.578501	0.590350	1
80	(cp_group_0)	(output_0)	0.471947	0.455446	0.343234	0.727273	1.596838	0.128288	1.996700	0.707813	1
72	(cp_0)	(output_0)	0.471947	0.455446	0.343234	0.727273	1.596838	0.128288	1.996700	0.707813	1
650	(cp_0, cp_group_0)	(output_0)	0.471947	0.455446	0.343234	0.727273	1.596838	0.128288	1.996700	0.707813	1
991	(exng_0, age_group_1, caa_0)	(output_1)	0.359736	0.544554	0.310231	0.862385	1.583653	0.114335	3.309571	0.575620	1
574	(exng_0, caa_0)	(output_1)	0.432343	0.544554	0.369637	0.854962	1.570021	0.134203	3.140177	0.639587	1

```
sorted_rules[(sorted_rules['antecedents'].astype(str).str.contains("thalachh")
              | sorted_rules['antecedents'].astype(str).str.contains("slp")
              | sorted_rules['antecedents'].astype(str).str.contains("restecg"))
             & sorted_rules['consequents'].astype(str).str.contains("output")].head(10)
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric	consequents_len
820	(exng_0, slp_2)	(output_1)	0.382838	0.544554	0.310231	0.810345	1.488088	0.101755	2.401440	0.531460	1
969	(thalachh_group_0, thall_2)	(output_1)	0.392739	0.544554	0.306931	0.781513	1.435141	0.093063	2.084539	0.499299	1
169	(slp_1)	(output_0)	0.462046	0.455446	0.300330	0.650000	1.427174	0.089893	1.555870	0.556395	1
177	(slp_2)	(output_1)	0.468647	0.544554	0.353135	0.753521	1.383739	0.097932	1.847808	0.521913	1
903	(fbs_0, slp_2)	(output_1)	0.402640	0.544554	0.300330	0.745902	1.369747	0.081070	1.792399	0.451885	1
963	(thalachh_group_0, sex_1)	(output_0)	0.531353	0.455446	0.320132	0.602484	1.322846	0.078130	1.369895	0.520764	1
1255	(fbs_0, thalachh_group_0, exng_0)	(output_1)	0.429043	0.544554	0.300330	0.700000	1.285455	0.066693	1.518152	0.388935	1
627	(thalachh_group_0, caa_0)	(output_1)	0.448845	0.544554	0.313531	0.698529	1.282754	0.069111	1.510746	0.399937	1
825	(thalachh_group_0, exng_0)	(output_1)	0.488449	0.544554	0.339934	0.695946	1.278010	0.073947	1.497910	0.425243	1
173	(restecg_1)	(output_1)	0.501650	0.544554	0.316832	0.631579	1.159809	0.043656	1.236209	0.276490	1

The best features selected by ARM are: caa, thall, cp_group, fbs, exng, cp, age_group

Modelling

```
validation_results = pd.DataFrame(columns=["model", "accuracy"])
```

Data Preparation

```
[43] df_final_arm = df[['caa', 'thall', 'cp_group', 'fbs', 'exng', 'cp', 'age_group','output']].copy(deep=True)
     df_final_corr = df[['cp_group', 'cp', 'slp', 'restecg', 'thalachh_group','output']].copy(deep=True)

     # final dataset of original/ unprocessed data
     df_final_original = df_const.copy(deep=True)
```

```
df_final = df_final_arm.copy(deep=True)
```

```
[47] X = df_final.drop(columns=['output'])
     y = df['output']

     X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.25, random_state=0)
     X_train
```

	caa	thall	cp_group	fbs	exng	cp	age_group
173	2	3	1	0	0	2	1
261	1	2	0	0	0	0	1
37	0	3	1	0	0	2	1
101	0	3	1	0	0	3	1
166	2	3	0	0	1	0	2
...	...	...	...	...	...	...	...
251	4	3	0	1	1	0	1
192	1	3	0	0	0	0	1
117	0	3	1	0	0	3	1
47	0	2	1	0	0	2	1
172	0	2	1	0	0	1	1

```
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 227 entries, 173 to 172
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   caa        227 non-null    int64
 1   thall      227 non-null    int64
 2   cp_group   227 non-null    int64
 3   fbs        227 non-null    int64
 4   exng       227 non-null    int64
 5   cp         227 non-null    int64
 6   age_group  227 non-null    int64
dtypes: int64(7)
```

Classification

```
[49] def create_model(units=128, activation='relu', learning_rate=0.001, optimizer='adam'):
         model = Sequential()
         model.add(Dense(units, input_dim=X_train.shape[1], activation=activation))
         model.add(Dense(1, activation='sigmoid'))  # Assuming binary classification
         model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
         return model
```

```
estimator = {
    'RandomForest': RandomForestClassifier(random_state=0),
    'DecisionTree': DecisionTreeClassifier(),
    'Knn': KNeighborsClassifier(),
    'Keras': KerasClassifier(model=create_model, verbose=0, random_state=0),
    'LogisticRegression': LogisticRegression(max_iter=100000, random_state=0),
}

param_grid = {
    'RandomForest': {
        'n_estimators': range(10, 110, 10),
        'max_depth': range(3, 22, 2),
    },
    'DecisionTree': {
        'max_depth': range(3, 22, 2),
    },
    'Knn': {
        'n_neighbors': range(3, 110, 3),
        'weights': ['uniform', 'distance'],
    },
    'Keras': {
        'optimizer': ['adam', 'rmsprop'],
        'batch_size': [32, 64],
        'epochs': [10, 20, 30, 40, 50],
    },
    'LogisticRegression': {
        'solver': ['lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky', 'sag', 'saga'],
    },
}
```

```
fit_models = {}
# loop all the classifiers
for algo, classifier in estimator.items():
    print(f'Training the {algo} model...')
    # create grid search CV
    clf = GridSearchCV(estimator=classifier, param_grid=param_grid[algo], n_jobs=-1, cv=None)

    # train the model
    t0 = time.time()
    clf.fit(X_train, y_train)

    # store the trained result
    fit_models[algo] = clf

    print(f'The {algo} model is trained. Time taken = {time.time()-t0}')
```

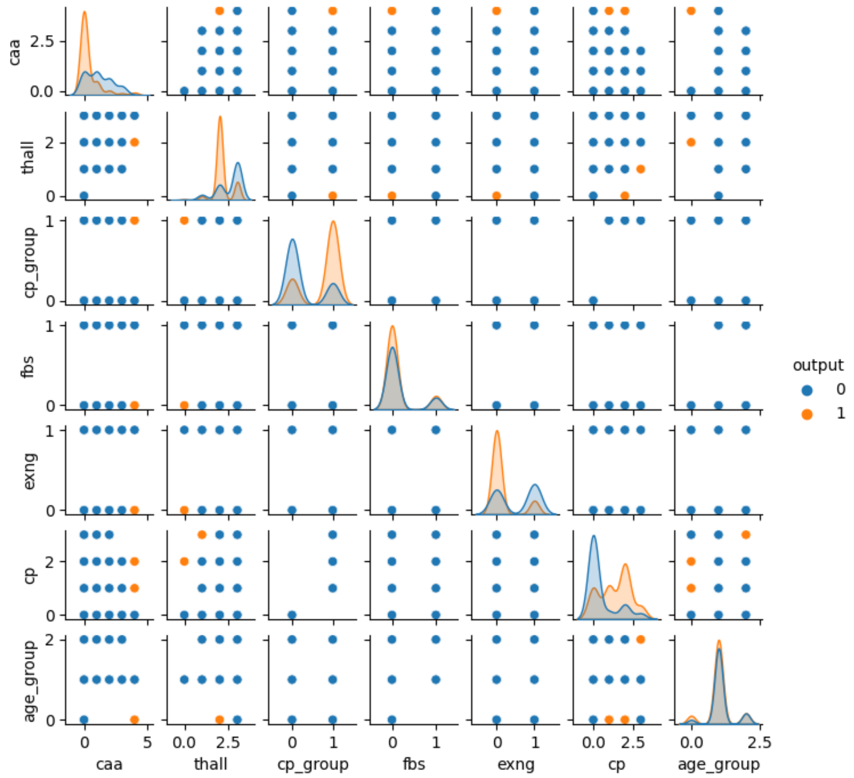
```
Training the RandomForest model...
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automa
  and should_run_async(code)
The RandomForest model is trained. Time taken = 48.503103494644165
Training the DecisionTree model...
The DecisionTree model is trained. Time taken = 0.22644901275634766
Training the Knn model...
The Knn model is trained. Time taken = 1.9948561191558838
Training the Keras model...
/usr/local/lib/python3.10/dist-packages/joblib/externals/loky/process_executor.py:752: UserWarning: A worker stopped while some jobs were given
  warnings.warn(
The Keras model is trained. Time taken = 117.9610641002655
Training the LogisticRegression model...
The LogisticRegression model is trained. Time taken = 0.2695605754852295
```

```python
for algo, classifier in fit_models.items():
    y_pred = classifier.predict(X_valid)
    accuracy = accuracy_score(y_valid, y_pred)
    print(f'Model: {algo}')
    print(f'Accuracy score: {accuracy}\n')
    validation_results.loc[len(validation_results.index)] = [algo, accuracy]
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipykernel.py:283: DeprecationWar
  and should_run_async(code)
Model: RandomForest
Accuracy score: 0.8289473684210527

Model: DecisionTree
Accuracy score: 0.8289473684210527

Model: Knn
Accuracy score: 0.8552631578947368

Model: Keras
Accuracy score: 0.7763157894736842

Model: LogisticRegression
Accuracy score: 0.8157894736842105
```

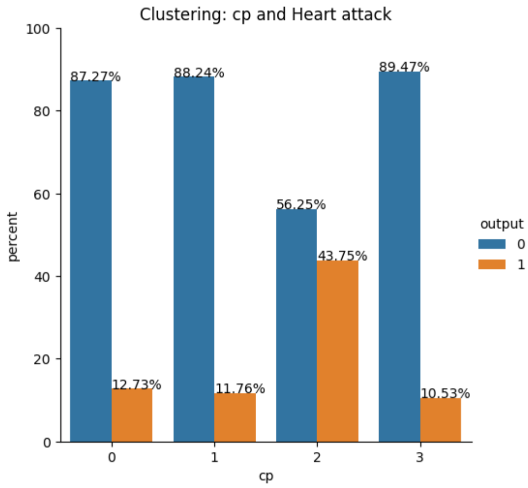
Clustering

```python
[54] X_cluster=X_train.copy(deep=True)
     sns.pairplot(df_final, hue='output', height=1)
     plt.show()
```

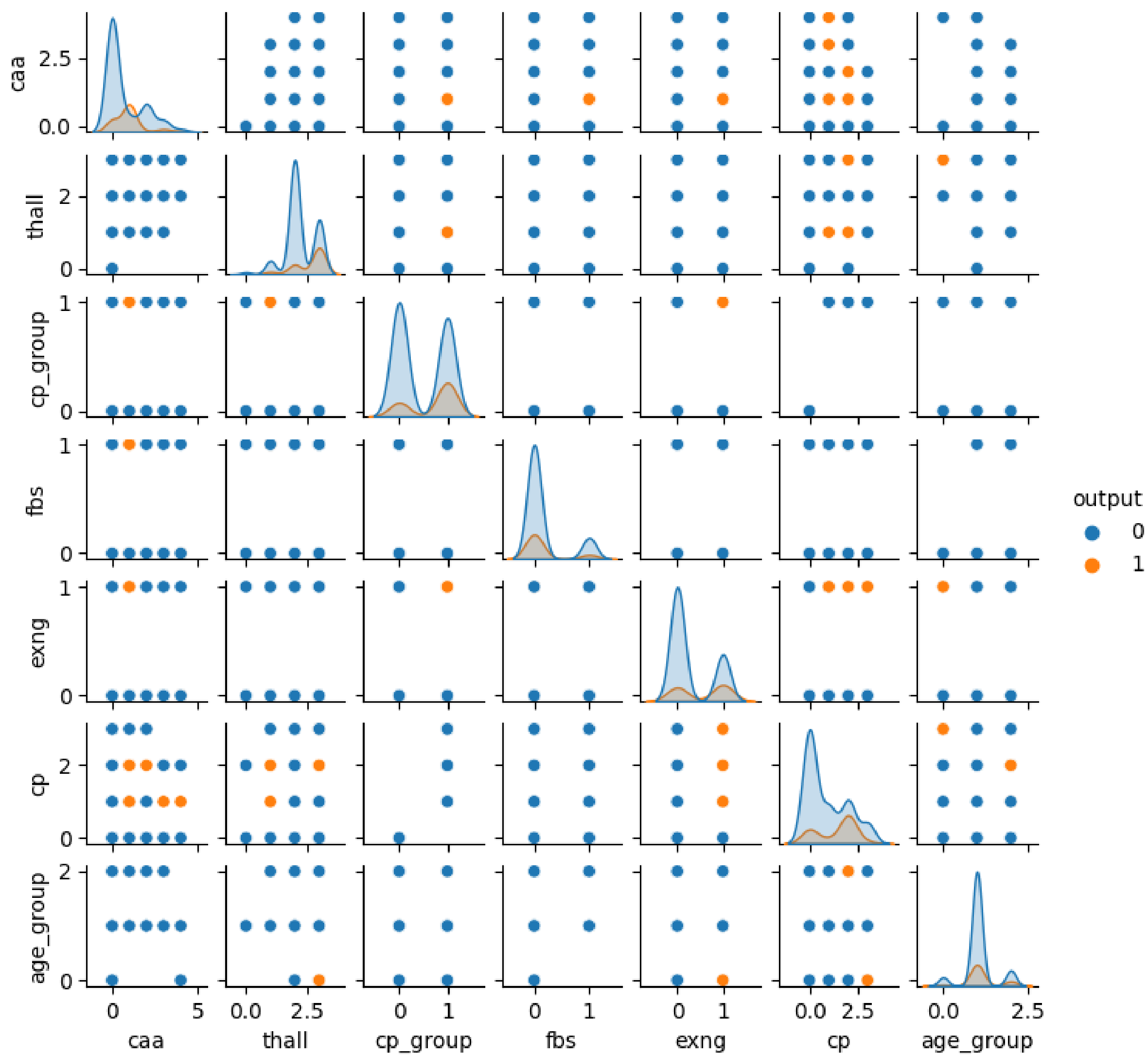


```
[63] km=KModes(n_clusters=2)
     cluster_labels=km.fit_predict(X_cluster)
     X_cluster['output'] = cluster_labels
```

```
x,y = 'cp', 'output'
df9 = X_cluster.groupby(x)[y].value_counts(normalize=True)
df9 = df9.mul(100)
df9 = df9.rename('percent').reset_index()

g = sns.catplot(x=x,y='percent',hue=y,kind='bar',data=df9)
g.ax.set_ylim(0,100)
plt.title('Clustering: cp and Heart attack')

for p in g.ax.patches:
    txt = str(p.get_height().round(2)) + '%'
    txt_x = p.get_x()
    txt_y = p.get_height()
    g.ax.text(txt_x,txt_y,txt)
```




```
[66] sns.pairplot(X_cluster, hue='output', height=1)
     plt.show()
```



Results Visualization

```
[67] validation_results
```

```
/usr/local/lib/python3.10/dist-packages/ipy
  and should_run_async(code)
```

	model	accuracy
0	RandomForest	0.828947
1	DecisionTree	0.828947
2	Knn	0.855263
3	Keras	0.776316
4	LogisticRegression	0.815789

- **Sorting results by Accuracy**

```
[69] validation_results.sort_values(by="accuracy", ascending=False)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
  and should_run_async(code)
```

	model	accuracy
2	Knn	0.855263
0	RandomForest	0.828947
1	DecisionTree	0.828947
4	LogisticRegression	0.815789
3	Keras	0.776316