In [280]:
```
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sbn
import warnings
warnings.filterwarnings('ignore')
```

In [281]:
```
#Load Data and descriptive analysis
```

In [282]:
```
df=pd.read_csv("data.csv")
df.shape
```

Out[282]:
```
(920, 16)
```

In [283]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 920 entries, 0 to 919
Data columns (total 16 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   id        920 non-null    int64  
 1   age       920 non-null    int64  
 2   sex       920 non-null    object 
 3   dataset   920 non-null    object 
 4   cp        920 non-null    object 
 5   trestbps  861 non-null    float64
 6   chol      890 non-null    float64
 7   fbs       830 non-null    object 
 8   restecg   918 non-null    object 
 9   thalch    865 non-null    float64
 10  exang     865 non-null    object 
 11  oldpeak   858 non-null    float64
 12  slope     611 non-null    object 
 13  ca        309 non-null    float64
 14  thal      434 non-null    object 
 15  num       920 non-null    int64  
dtypes: float64(5), int64(3), object(8)
memory usage: 115.1+ KB
```

In [284]:
```
print("=====Objects====")
print(df.select_dtypes(['object']).dtypes)
print("=====Numeric====")
print(df.select_dtypes(['number']).dtypes)
print("=====Bool====")
print(df.select_dtypes(['bool']).dtypes)
```

```
=====Objects====
sex         object
dataset     object
cp          object
fbs         object
restecg     object
exang       object
slope       object
thal        object
dtype: object
=====Numeric====
id             int64
age            int64
trestbps     float64
chol         float64
thalch       float64
oldpeak      float64
ca           float64
num            int64
dtype: object
=====Bool====
Series([], dtype: object)
```

In [285]:
```
df
```

Out[285]:

	id	age	sex	dataset	cp	trestbps	chol	fbs	restecg	thalch	exang	oldpeak	slope	ca	thal	num
0	1	63	Male	Cleveland	typical angina	145.0	233.0	True	lv hypertrophy	150.0	False	2.3	downsloping	0.0	fixed defect	0
1	2	67	Male	Cleveland	asymptomatic	160.0	286.0	False	lv hypertrophy	108.0	True	1.5	flat	3.0	normal	2
2	3	67	Male	Cleveland	asymptomatic	120.0	229.0	False	lv hypertrophy	129.0	True	2.6	flat	2.0	reversable defect	1
3	4	37	Male	Cleveland	non-anginal	130.0	250.0	False	normal	187.0	False	3.5	downsloping	0.0	normal	0
4	5	41	Female	Cleveland	atypical angina	130.0	204.0	False	lv hypertrophy	172.0	False	1.4	upsloping	0.0	normal	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
915	916	54	Female	VA Long Beach	asymptomatic	127.0	333.0	True	st-t abnormality	154.0	False	0.0	NaN	NaN	NaN	1
916	917	62	Male	VA Long Beach	typical angina	NaN	139.0	False	st-t abnormality	NaN	NaN	NaN	NaN	NaN	NaN	0
917	918	55	Male	VA Long Beach	asymptomatic	122.0	223.0	True	st-t abnormality	100.0	False	0.0	NaN	NaN	fixed defect	2
918	919	58	Male	VA Long Beach	asymptomatic	NaN	385.0	True	lv hypertrophy	NaN	NaN	NaN	NaN	NaN	NaN	0
919	920	62	Male	VA Long Beach	atypical angina	120.0	254.0	False	lv hypertrophy	93.0	True	0.0	NaN	NaN	NaN	1

920 rows × 16 columns

In [286]:
```
df.isna().sum()
```

Out[286]:
```
id            0
age           0
sex           0
dataset       0
cp            0
trestbps     59
chol         30
fbs          90
restecg       2
thalch       55
exang        55
oldpeak      62
slope       309
ca          611
thal        486
num           0
dtype: int64
```

In [287]:
```
df.describe()
```

Out[287]:

	id	age	trestbps	chol	thalch	oldpeak	ca	num
count	920.000000	920.000000	861.000000	890.000000	865.000000	858.000000	309.000000	920.000000
mean	460.500000	53.510870	132.132404	199.130337	137.545665	0.878788	0.676375	0.995652
std	265.725422	9.424685	19.066070	110.780810	25.926276	1.091226	0.935653	1.142693
min	1.000000	28.000000	0.000000	0.000000	60.000000	-2.600000	0.000000	0.000000
25%	230.750000	47.000000	120.000000	175.000000	120.000000	0.000000	0.000000	0.000000
50%	460.500000	54.000000	130.000000	223.000000	140.000000	0.500000	0.000000	1.000000
75%	690.250000	60.000000	140.000000	268.000000	157.000000	1.500000	1.000000	2.000000
max	920.000000	77.000000	200.000000	603.000000	202.000000	6.200000	3.000000	4.000000

In [ ]:
```
```

In [288]:
```
df['sex'].value_counts()
```

Out[288]:
```
Male      726
Female    194
Name: sex, dtype: int64
```

In [289]:
```
# for i in range(len(df.columns)):
#     print("unique Values:",df.columns[i])
#     print(df[df.columns[i]].value_counts())
```

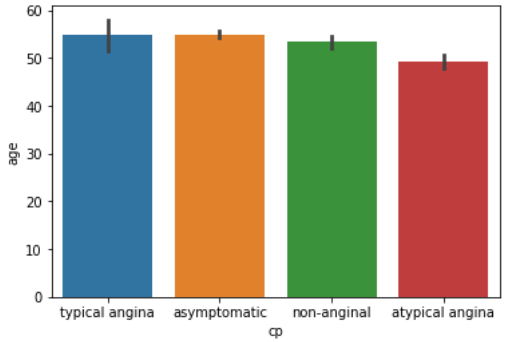
In [290]:
```
df['slope'].value_counts()
```

Out[290]:
```
flat           345
upsloping      203
downsloping     63
Name: slope, dtype: int64
```

In [291]:
```
#EDA
```

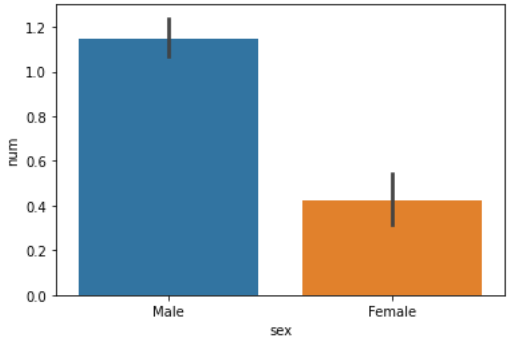
In [292]:
```
sbn.barplot(y=df['age'],x=df['cp'])
```

Out[292]: <AxesSubplot:xlabel='cp', ylabel='age'>



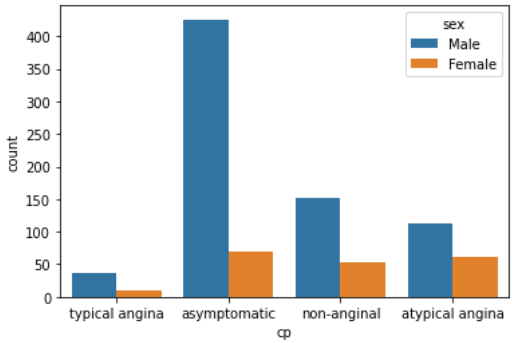
In [293]:
```
sbn.barplot(x=df['sex'],y=df['num'])
```

Out[293]: <AxesSubplot:xlabel='sex', ylabel='num'>



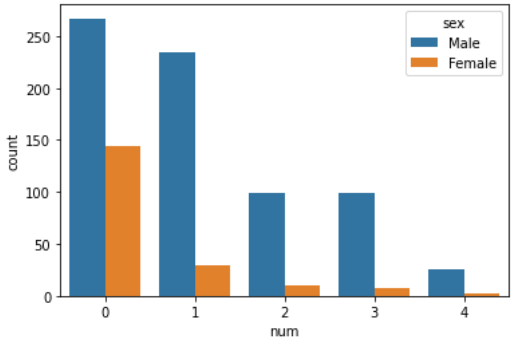
In [294]:
```
sbn.countplot(x=df['cp'],hue=df['sex'])
```

Out[294]: <AxesSubplot:xlabel='cp', ylabel='count'>



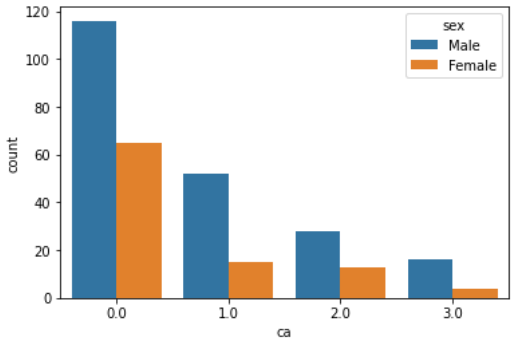
In [295]:
```
sbn.countplot(x=df['num'],hue=df['sex'])
```

Out[295]: <AxesSubplot:xlabel='num', ylabel='count'>



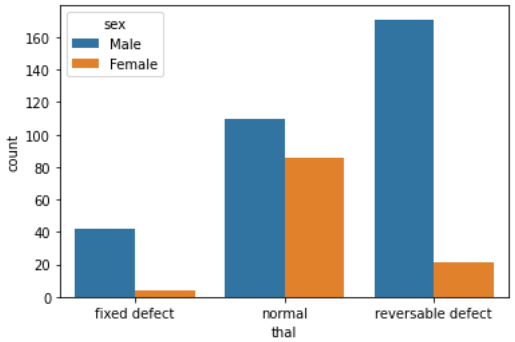
In [296]:
```
sbn.countplot(x=df['ca'],hue=df['sex'])
```

Out[296]: <AxesSubplot:xlabel='ca', ylabel='count'>



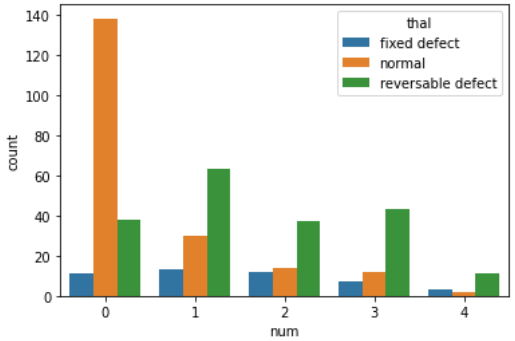
In [297]:
```
sbn.countplot(x=df['thal'],hue=df['sex'])
```

Out[297]: <AxesSubplot:xlabel='thal', ylabel='count'>



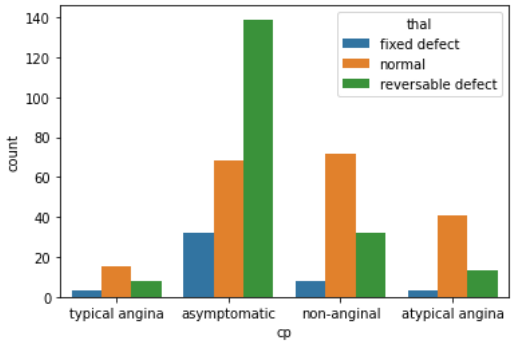
In [298]:
```
sbn.countplot(x=df['num'],hue=df['thal'])
```

Out[298]: <AxesSubplot:xlabel='num', ylabel='count'>



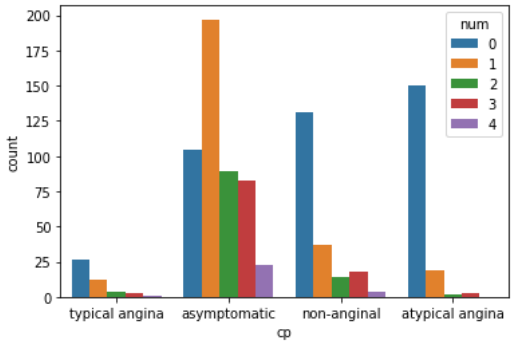
In [299]:
```
sbn.countplot(x=df['cp'],hue=df['thal'])
```

Out[299]: <AxesSubplot:xlabel='cp', ylabel='count'>



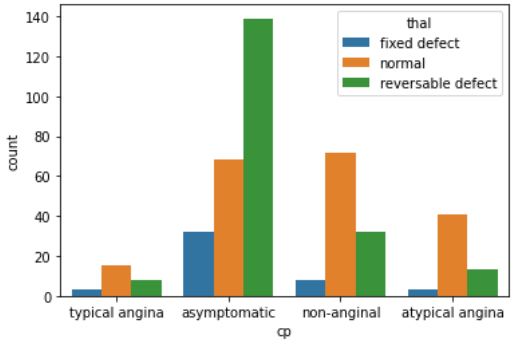
In [300]:
```
sbn.countplot(x=df['cp'],hue=df['num'])
```

Out[300]: <AxesSubplot:xlabel='cp', ylabel='count'>



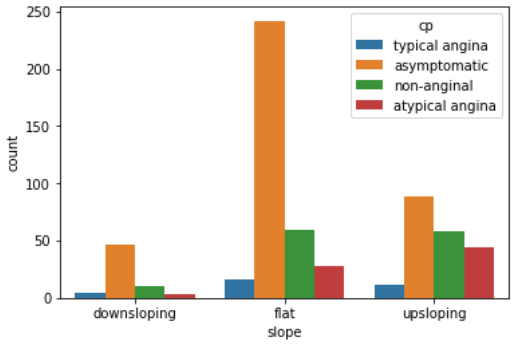
In [301]:
```
sbn.countplot(x=df['cp'],hue=df['thal'])
```

Out[301]: <AxesSubplot:xlabel='cp', ylabel='count'>



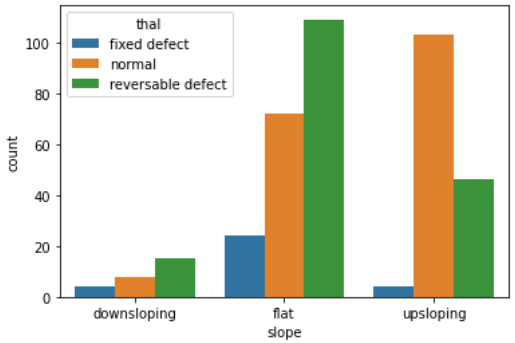
In [302]:
```
sbn.countplot(x=df['slope'],hue=df['cp'])
```

Out[302]: <AxesSubplot:xlabel='slope', ylabel='count'>



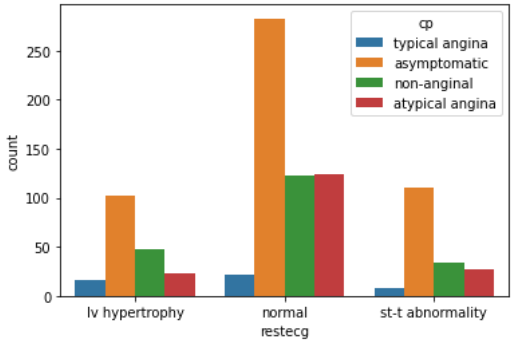
In [303]:
```
sbn.countplot(x=df['slope'],hue=df['thal'])
```

Out[303]: <AxesSubplot:xlabel='slope', ylabel='count'>



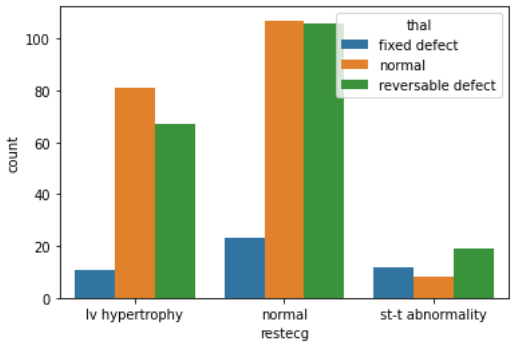
In [304]:
```
sbn.countplot(x=df['restecg'],hue=df['cp'])
```

Out[304]: <AxesSubplot:xlabel='restecg', ylabel='count'>



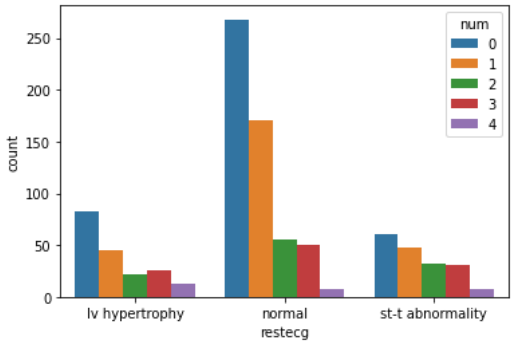
In [305]:
```
sbn.countplot(x=df['restecg'],hue=df['thal'])
```

Out[305]: <AxesSubplot:xlabel='restecg', ylabel='count'>



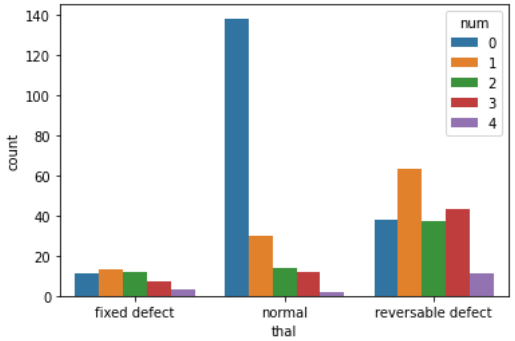
In [306]:
```
sbn.countplot(x=df['restecg'],hue=df['num'])
```

Out[306]:
```
<AxesSubplot:xlabel='restecg', ylabel='count'>
```



In [307]:
```
sbn.countplot(x=df['thal'],hue=df['num'])
```

Out[307]:
```
<AxesSubplot:xlabel='thal', ylabel='count'>
```

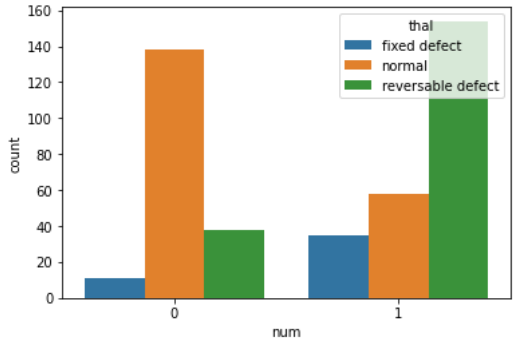


In [308]:
```
df2=df.replace({'num':{2:1,3:1,4:1}})
```

In [309]:
```
df2['num'].value_counts()
```

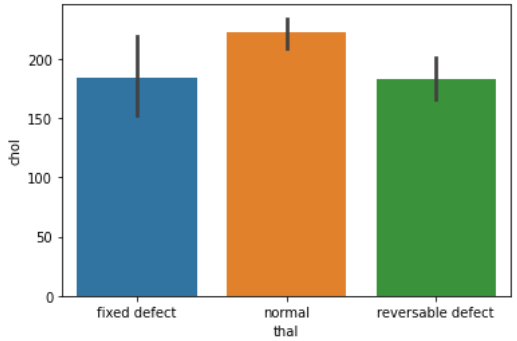
Out[309]:
```
1    509
0    411
Name: num, dtype: int64
```

In [310]:
```
sbn.countplot(x=df2['num'],hue=df2['thal'])
plt.show()
```



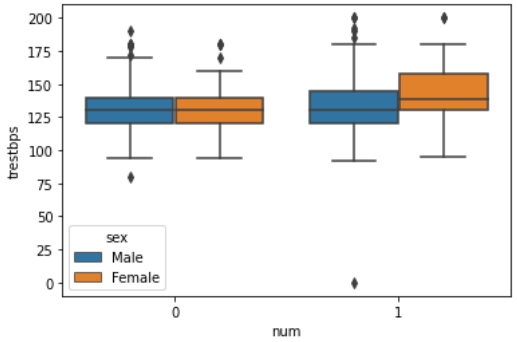
In [311]:
```
sbn.barplot(y=df2['chol'],x=df2['thal'])
```

Out[311]: <AxesSubplot:xlabel='thal', ylabel='chol'>



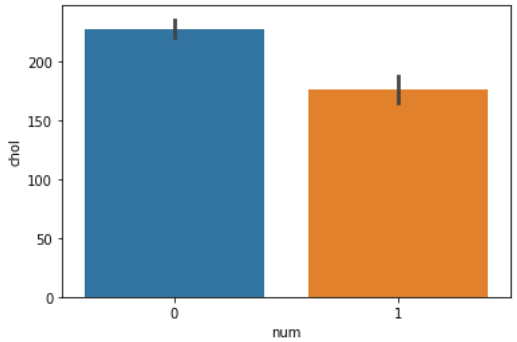
In [312]:
```
sbn.boxplot(y=df2['trestbps'],x=df2['num'],hue=df2['sex'])
```

Out[312]: <AxesSubplot:xlabel='num', ylabel='trestbps'>



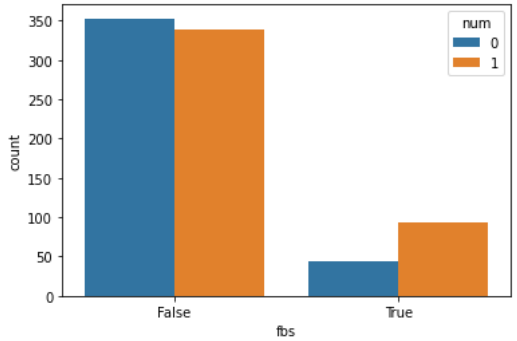
In [313]:
```
sbn.barplot(x=df2['num'],y=df2['chol'])
```

Out[313]: <AxesSubplot:xlabel='num', ylabel='chol'>



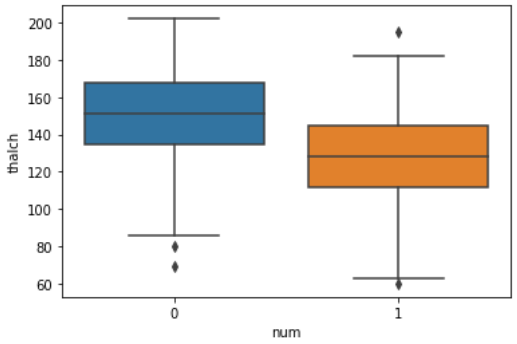
In [314]:
```
sbn.countplot(x=df2['fbs'],hue=df2['num'])
```

Out[314]: <AxesSubplot:xlabel='fbs', ylabel='count'>



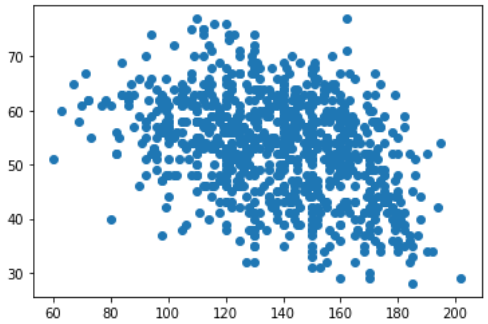
In [315]:
```
sbn.boxplot(x=df2['num'],y=df2['thalch'])
```

Out[315]: <AxesSubplot:xlabel='num', ylabel='thalch'>



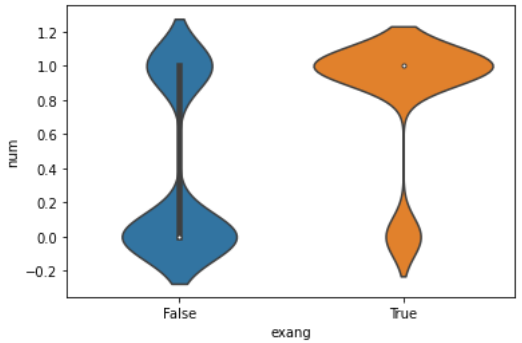
In [316]:
```
plt.scatter(data=df2,y='age',x='thalch')
```

Out[316]:
```
<matplotlib.collections.PathCollection at 0x216ec591da0>
```



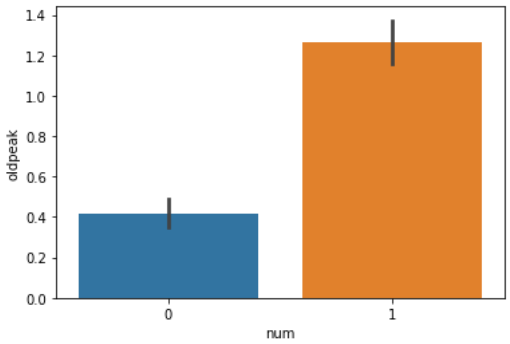
In [317]:
```
sbn.violinplot(y=df2['num'],x=df2['exang'])
# sbn.countplot(hue=df2['num'],x=df2['exang'])
```

Out[317]:
```
<AxesSubplot:xlabel='exang', ylabel='num'>
```



In [318]:
```
sbn.barplot(x=df2['num'],y=df2['oldpeak'])
```

Out[318]:
```
<AxesSubplot:xlabel='num', ylabel='oldpeak'>
```



In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

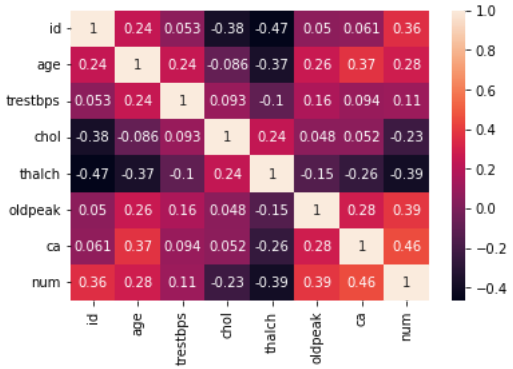
In [319]:
```
df2.corr()
```

Out[319]:

	id	age	trestbps	chol	thalch	oldpeak	ca	num
id	1.000000	0.239301	0.052924	-0.376936	-0.466427	0.049930	0.061433	0.356086
age	0.239301	1.000000	0.244253	-0.086234	-0.365778	0.258243	0.370416	0.282700
trestbps	0.052924	0.244253	1.000000	0.092853	-0.104899	0.161908	0.093705	0.106233
chol	-0.376936	-0.086234	0.092853	1.000000	0.236121	0.047734	0.051606	-0.230583
thalch	-0.466427	-0.365778	-0.104899	0.236121	1.000000	-0.151174	-0.264094	-0.394503
oldpeak	0.049930	0.258243	0.161908	0.047734	-0.151174	1.000000	0.281817	0.385528
ca	0.061433	0.370416	0.093705	0.051606	-0.264094	0.281817	1.000000	0.455599
num	0.356086	0.282700	0.106233	-0.230583	-0.394503	0.385528	0.455599	1.000000

In [320]:
```
sbn.heatmap(df2.corr(),annot=True)
```

Out[320]: <AxesSubplot:>



In [321]:
```
df2.isna().sum()
```

Out[321]:
```
id            0
age           0
sex           0
dataset       0
cp            0
trestbps     59
chol         30
fbs          90
restecg       2
thalch       55
exang        55
oldpeak      62
slope       309
ca          611
thal        486
num           0
dtype: int64
```

In [ ]:
```

```

In [322]:
```
df2['ca'].fillna(df2['ca'].median(),inplace=True)
```

In [323]:
```
df2['slope'].fillna(df2['slope'].mode()[0],inplace=True)
```

In [324]:
```
df2['thal'].fillna(df2['thal'].mode()[0],inplace=True)
```

In [325]:
```
df2['trestbps'].fillna(df2['trestbps'].median(),inplace=True)
df2['chol'].fillna(df2['chol'].mean(),inplace=True)
# df2['chol'].fillna(df2['chol'].mean(),inplace=True)
df2['fbs'].fillna(df2['fbs'].mode()[0],inplace=True)
df2['thalch'].fillna(df2['thalch'].median(),inplace=True)
df2['exang'].fillna(df2['exang'].mode()[0],inplace=True)
df2['oldpeak'].fillna(df2['oldpeak'].mean(),inplace=True)
df2['restecg'].fillna(df2['restecg'].mode()[0],inplace=True)


print(df2['restecg'].value_counts())
# df2['restecg'].mode()
```

```
normal              553
lv hypertrophy      188
st-t abnormality    179
Name: restecg, dtype: int64
```

In [326]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 920 entries, 0 to 919
Data columns (total 16 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   id        920 non-null    int64
 1   age       920 non-null    int64
 2   sex       920 non-null    object
 3   dataset   920 non-null    object
 4   cp        920 non-null    object
 5   trestbps  920 non-null    float64
 6   chol      920 non-null    float64
 7   fbs       920 non-null    bool
 8   restecg   920 non-null    object
 9   thalch    920 non-null    float64
 10  exang     920 non-null    bool
 11  oldpeak   920 non-null    float64
 12  slope     920 non-null    object
 13  ca        920 non-null    float64
 14  thal      920 non-null    object
 15  num       920 non-null    int64
dtypes: bool(2), float64(5), int64(3), object(6)
memory usage: 102.5+ KB
```

In [327]:
```
#DATA PREPROCESSING
```

In [328]:
```
a = pd.get_dummies(df2['cp'], prefix = "cp")
b = pd.get_dummies(df2['thal'], prefix = "thal")
c = pd.get_dummies(df2['slope'], prefix = "slope")
# d = pd.get_dummies(df2['sex'], prefix = "sex")
d = pd.get_dummies(df2['restecg'], prefix = "rest")
e = pd.get_dummies(df2['dataset'], prefix = "city")
```

In [329]:
```
ucols=[df2,a,b,c,d,e]
df3=pd.concat(ucols,axis=1)
df3 = df3.drop(columns = ['cp','thal', 'slope','restecg','dataset'])
df4=df3.replace({'fbs':{False:0,True:1},'exang':{False:0,True:1},'sex':{'Male':0,'Female':1}})
```

In [330]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
sccols=['age','chol','trestbps','thalch','oldpeak']
df4[sccols]=sc.fit_transform(df4[sccols])
df4
```

Out[330]:

	id	age	sex	trestbps	chol	fbs	thalch	exang	oldpeak	ca	...	slope_downsloping	slope_flat	slope_upsloping	rest_lv hypertrophy	rest_normal	rest_st-t abnormality	city_Cleveland	city_Hungary	city_Switzerland	city_VA Long Beach
0	1	1.007386	0	0.705176	0.311021	1	0.489727	0	1.349421	0.0	...	1	0	0	1	0	0	1	0	0	0
1	2	1.432034	0	1.518569	0.797713	0	-1.181478	1	0.589832	3.0	...	0	1	0	1	0	0	1	0	0	0
2	3	1.432034	0	-0.650479	0.274289	0	-0.345875	1	1.634267	2.0	...	0	1	0	1	0	0	1	0	0	0
3	4	-1.752828	0	-0.108217	0.467130	0	1.961979	0	2.488805	0.0	...	1	0	0	0	1	0	1	0	0	0
4	5	-1.328180	1	-0.108217	0.044717	0	1.365120	0	0.494884	0.0	...	0	0	1	1	0	0	1	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
915	916	0.051927	1	-0.270895	1.229308	1	0.648889	0	-0.834397	0.0	...	0	1	0	0	0	1	0	0	0	1
916	917	0.901224	0	-0.108217	-0.552169	0	0.091821	0	0.000000	0.0	...	0	1	0	0	0	1	0	0	0	1
917	918	0.158089	0	-0.542026	0.219192	1	-1.499803	0	-0.834397	0.0	...	0	1	0	0	0	1	0	0	0	1
918	919	0.476575	0	-0.108217	1.706817	1	0.091821	0	0.000000	0.0	...	0	1	0	1	0	0	0	0	0	1
919	920	0.901224	0	-0.650479	0.503861	0	-1.778337	1	-0.834397	0.0	...	0	1	0	1	0	0	0	0	0	1

920 rows × 28 columns

In [331]:
```
#Model Evalution
```

In [332]:
```
f1_scores={}
accscores={}
```

In [333]:
```
from sklearn.model_selection import train_test_split
```

In [334]:
```
y=df4['num']
x=df4.drop(['id','num'],axis=1)
```

In [335]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
```

In [336]:
```
from sklearn.naive_bayes import GaussianNB
nb=GaussianNB()
nb.fit(x_train,y_train)
y_pred=nb.predict(x_test)
acc=nb.score(x_test,y_test)
print(acc)
accscores['NB']=acc
```

```
0.8152173913043478
```

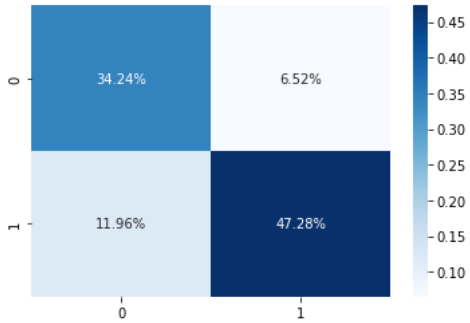
In [337]:
```
from sklearn.metrics import confusion_matrix,recall_score,precision_score,f1_score,accuracy_score
import scikitplot as scplot
```

In [338]:
```
cfm=confusion_matrix(y_test,y_pred)
print(cfm)
```

```
[[63 12]
 [22 87]]
```

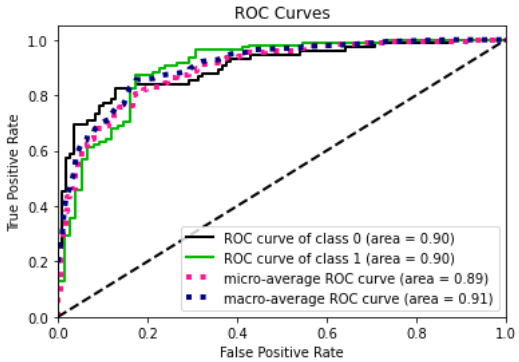
In [339]:
```
import numpy as np
sbn.heatmap(cfm/np.sum(cfm), annot=True,
            fmt='.2%', cmap='Blues')
```

Out[339]: <AxesSubplot:>



In [340]:
```
nb=GaussianNB()
nb.fit(x_train,y_train)

y_pred_prob=nb.predict_proba(x_test)
y_true=y_test
scplot.metrics.plot_roc(y_true,y_pred_prob)
```

Out[340]: <AxesSubplot:title={'center':'ROC Curves'}, xlabel='False Positive Rate', ylabel='True Positive Rate'>



In [341]:
```
rcscore=recall_score(y_test,y_pred)
rcscore
```

Out[341]:
```
0.7981651376146789
```

In [342]:
```
pecs=precision_score(y_test,y_pred)
pecs
```

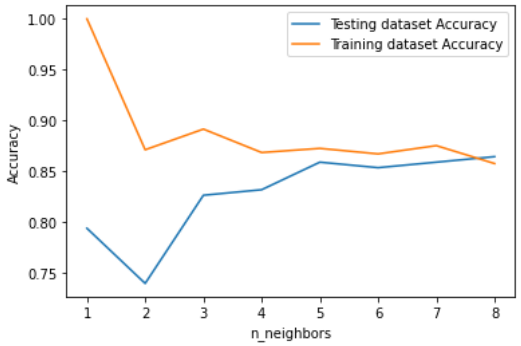
Out[342]:
```
0.8787878787878788
```

In [343]:
```
f1_scores={}
accscores={}
fnb=f1_score(y_test,y_pred)
f1_scores['NB']=fnb
nbacc=nb.score(x_test,y_test)
accscores['NB']=nbacc
```

In [344]:
```
from sklearn.neighbors import KNeighborsClassifier
for i in range(1,20):
    knn=KNeighborsClassifier(n_neighbors=i)
    knn.fit(x_train,y_train)
    y_kn_pred=knn.predict(x_test)
    print(i,"train",knn.score(x_train,y_train))
    print(i,"test",knn.score(x_test,y_test))
    print("====================")
```

```
1 train 1.0
1 test 0.7934782608695652
====================
2 train 0.8709239130434783
2 test 0.7391304347826086
====================
3 train 0.8913043478260869
3 test 0.8260869565217391
====================
4 train 0.8682065217391305
4 test 0.8315217391304348
====================
5 train 0.8722826086956522
5 test 0.8586956521739131
====================
6 train 0.8668478260869565
6 test 0.8532608695652174
====================
7 train 0.875
7 test 0.8586956521739131
====================
8 train 0.8573369565217391
8 test 0.8641304347826086
====================
9 train 0.8614130434782609
9 test 0.8695652173913043
====================
10 train 0.8505434782608695
10 test 0.8586956521739131
====================
11 train 0.8546195652173914
11 test 0.8695652173913043
====================
12 train 0.8464673913043478
12 test 0.8695652173913043
====================
13 train 0.8478260869565217
13 test 0.8695652173913043
====================
14 train 0.8396739130434783
14 test 0.8586956521739131
====================
15 train 0.8505434782608695
15 test 0.8532608695652174
====================
16 train 0.8464673913043478
16 test 0.8641304347826086
====================
17 train 0.8396739130434783
17 test 0.8641304347826086
====================
18 train 0.8355978260869565
18 test 0.875
====================
19 train 0.8383152173913043
19 test 0.8695652173913043
====================
```

In [345]:
```
neighbors = np.arange(1, 9)
train_accuracy = np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))

# Loop over K values
for i, k in enumerate(neighbors):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(x_train, y_train)

    # Compute training and test data accuracy
    train_accuracy[i] = knn.score(x_train, y_train)
    test_accuracy[i] = knn.score(x_test, y_test)

# Generate plot
plt.plot(neighbors, test_accuracy, label = 'Testing dataset Accuracy')
plt.plot(neighbors, train_accuracy, label = 'Training dataset Accuracy')

plt.legend()
plt.xlabel('n_neighbors')
plt.ylabel('Accuracy')
plt.show()
```

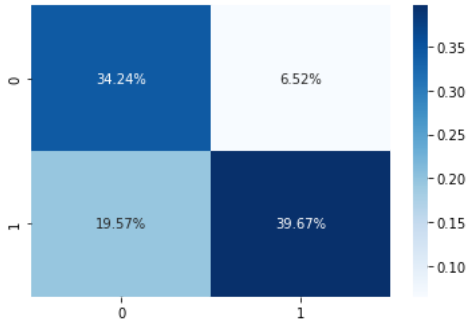


In [346]:
```
knn=KNeighborsClassifier(n_neighbors=2)
knn.fit(x_train,y_train)
y_kn_pred=knn.predict(x_test)
print(knn.score(x_train,y_train))
knnacc=knn.score(x_train,y_train)
accscores['KNN']=knnacc
```

```
0.8709239130434783
```

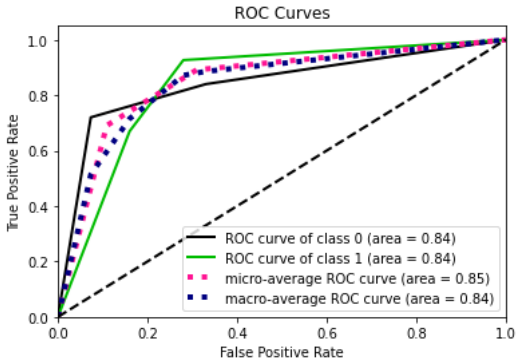
In [347]:
```
cf_matrix = confusion_matrix(y_test, y_kn_pred)

sbn.heatmap(cf_matrix/np.sum(cf_matrix), annot=True,
            fmt='.2%', cmap='Blues')
```

Out[347]: <AxesSubplot:>



In [348]:
```
fknn=f1_score(y_test, y_kn_pred)
f1_scores['KNN']=fknn
```

In [349]:
```
y_kn_pred_proba = knn.predict_proba(x_test)
y_kn_true = y_test
scplot.metrics.plot_roc(y_true, y_kn_pred_proba)
plt.show()
```

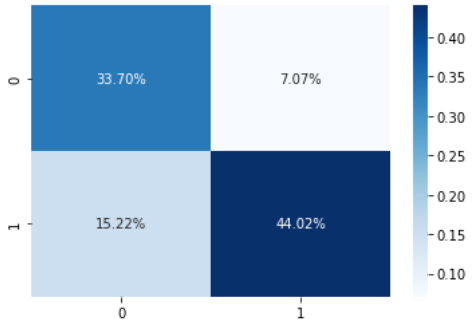


In [350]:
```
from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier(random_state=1)
dt.fit(x_train,y_train)
y_dt_pred=dt.predict(x_test)
print(dt.score(x_test,y_test))
dtacc=dt.score(x_test,y_test)
accscores['DT']=dtacc
```

```
0.7771739130434783
```

In [351]:
```
cf_matrix = confusion_matrix(y_test.T, y_dt_pred)

sbn.heatmap(cf_matrix/np.sum(cf_matrix), annot=True,
            fmt='.2%', cmap='Blues')
```

Out[351]: <AxesSubplot:>



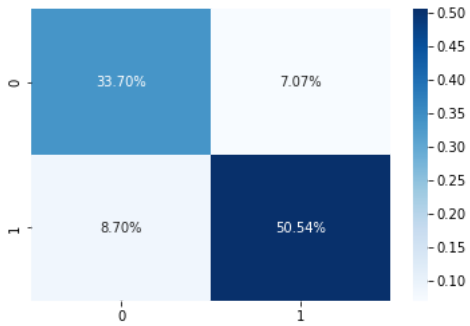
In [352]:
```
fdt=f1_score(y_test,y_dt_pred)
f1_scores['DT']=fdt
```

In [353]:
```
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()
lr.fit(x_train,y_train)
y_l_pred=lr.predict(x_test)
cfml=confusion_matrix(y_test,y_l_pred)
print(lr.score(x_test,y_test))
lracc=lr.score(x_test,y_test)
accscores['LR']=lracc
flr=f1_score(y_test,y_l_pred)
f1_scores['LR']=flr
print(cfml)
```

```
0.842391304347826
[[62 13]
 [16 93]]
```

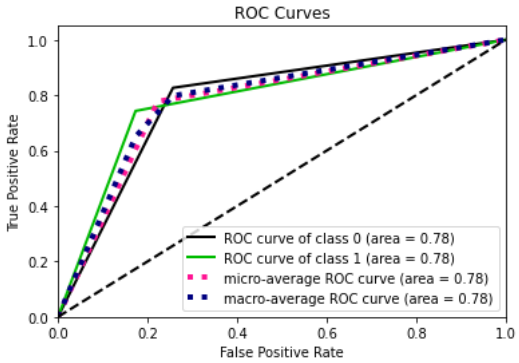
In [354]:
```
sbn.heatmap(cfml/np.sum(cfml), annot=True,
            fmt='.2%', cmap='Blues')
```

Out[354]: <AxesSubplot:>



In [355]:
```
fdt=f1_score(y_test,y_dt_pred)
f1_scores['DT']=fdt
```

In [356]:
```
y_dt_pred_proba = dt.predict_proba(x_test)
y_dt_true = y_test
scplot.metrics.plot_roc(y_dt_true, y_dt_pred_proba)
plt.show()
```

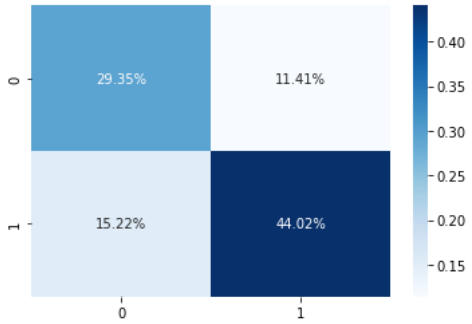


In [357]:
```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier(n_estimators=1,random_state=0)
rf.fit(x_train,y_train)
y_rf_pred=rf.predict(x_test)
print(rf.score(x_test,y_test))
rfacc=rf.score(x_test,y_test)
accscores['RF']=rfacc
```

```
0.7336956521739131
```

In [358]:
```
cfml=confusion_matrix(y_test,y_rf_pred)
sbn.heatmap(cfml/np.sum(cfml), annot=True,
            fmt='.2%', cmap='Blues')
```

Out[358]: <AxesSubplot:>



In [359]:
```
frt=f1_score(y_test,y_rf_pred)
f1_scores['RF']=fnb
```

In [360]:
```
#F1 SCORES
f1_scores
```

Out[360]:
```
{'NB': 0.8365384615384616,
 'KNN': 0.7525773195876289,
 'DT': 0.7980295566502462,
 'LR': 0.8651162790697675,
 'RF': 0.8365384615384616}
```

In [361]:
```
#ACCURACIES
accscores
```

Out[361]:
```
{'NB': 0.8152173913043478,
 'KNN': 0.8709239130434783,
 'DT': 0.7771739130434783,
 'LR': 0.842391304347826,
 'RF': 0.7336956521739131}
```

In [362]:
```
#Feature Engineering
```

In [363]:
```
ys=df4['num']
xs=df4.drop(['id','num'],axis=1)
```

In [364]:
```
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_selection import chi2,SelectKBest
X_norm = MinMaxScaler().fit_transform(xs)
test = SelectKBest(score_func=chi2, k=4)
fit = test.fit(X_norm, ys)
# X_new=test.fit_transform(X_norm, y)
# X_new
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(xs.columns)
#concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Specs','Score']
print(featureScores.nlargest(26,'Score'))
```

```
                    Specs       Score
10      cp_atypical angina  121.437148
9          cp_asymptomatic  112.777005
6                    exang  109.611826
24       city_Switzerland   72.504009
1                      sex   68.551621
15  thal_reversable defect   48.094615
23           city_Hungary   43.466863
11          cp_non-anginal   31.519125
25      city_VA Long Beach   29.748670
18         slope_upsloping   23.464482
14             thal_normal   19.494106
8                        ca   14.478627
16       slope_downsloping   12.848567
22          city_Cleveland   11.340951
4                      fbs    9.133272
21   rest_st-t abnormality    8.130710
13      thal_fixed defect    8.021673
5                   thalch    7.680281
0                      age    5.218891
3                     chol    4.659986
7                 oldpeak    4.635133
20            rest_normal    3.212138
12       cp_typical angina    2.612469
17              slope_flat    2.516157
2                 trestbps    0.121554
19     rest_lv hypertrophy    0.084964
```

In [365]:
```
from sklearn.feature_selection import RFE
rfe = RFE(lr)
fit = rfe.fit(xs, ys)
fit.n_features_
print("Num Features: ", fit.n_features_)
print("Selected Features:",fit.support_)
print("Feature Ranking: ",  fit.ranking_)
from operator import itemgetter
features = xs.columns.to_list()
for x1, y1 in (sorted(zip(rfe.ranking_ , features), key=itemgetter(0))):
    print(x1, y1)
```

```
Num Features:  13
Selected Features: [False  True False False  True False  True  True  True  True  True False
 False False  True  True False  True False False False False  True  True
  True False]
Feature Ranking:  [ 6  1  9  7  1  5  1  1  1  1  1  2 11  8  1  1  4  1  3 10 14 13  1  1
  1 12]
1 sex
1 fbs
1 exang
1 oldpeak
1 ca
1 cp_asymptomatic
1 cp_atypical angina
1 thal_normal
1 thal_reversable defect
1 slope_flat
1 city_Cleveland
1 city_Hungary
1 city_Switzerland
2 cp_non-anginal
3 slope_upsloping
4 slope_downsloping
5 thalch
6 age
7 chol
8 thal_fixed defect
9 trestbps
10 rest_lv hypertrophy
11 cp_typical angina
12 city_VA Long Beach
13 rest_st-t abnormality
14 rest_normal
```

In [366]:
```
from sklearn.ensemble import ExtraTreesClassifier

model = ExtraTreesClassifier()
model.fit(xs, ys)
print(model.feature_importances_)
```

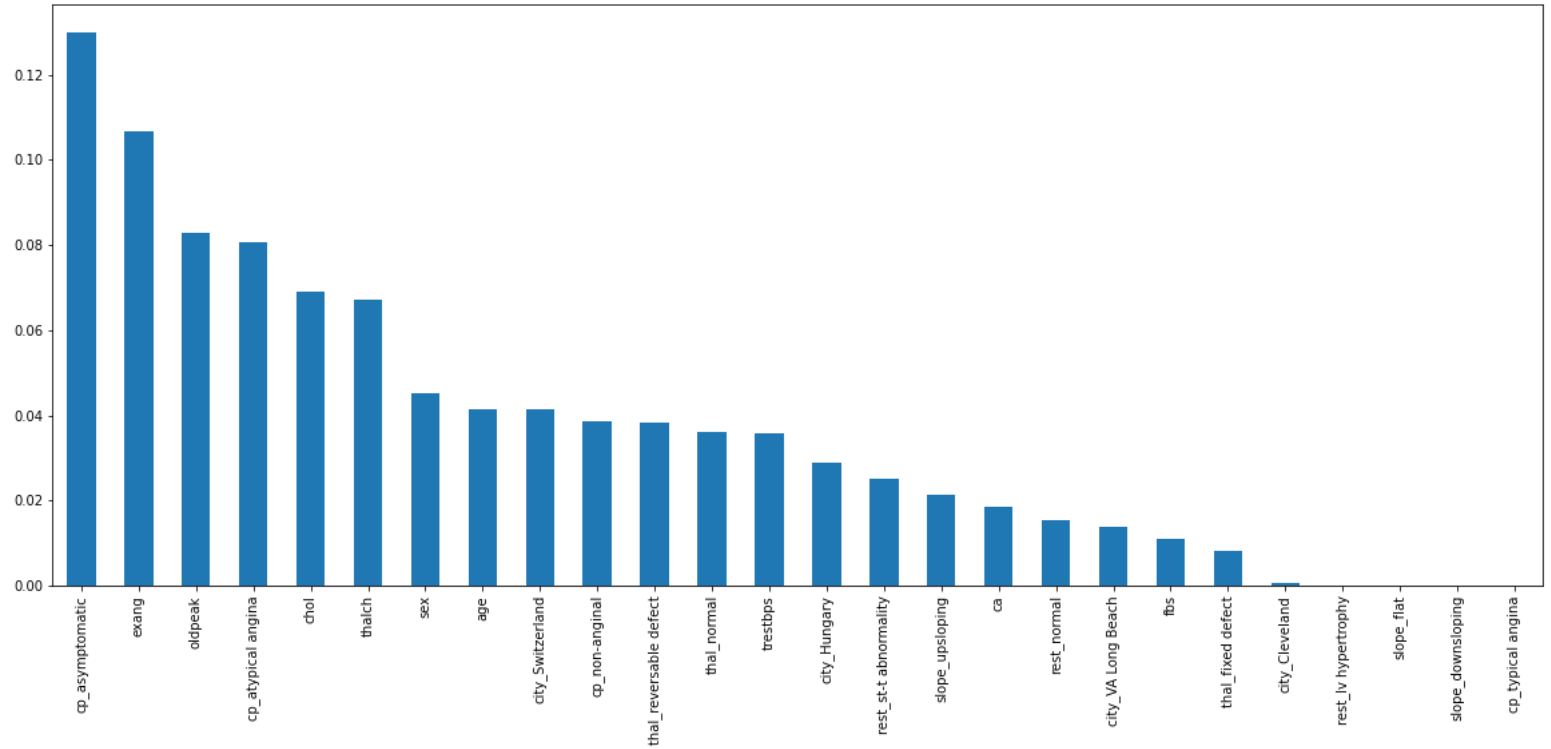
```
[0.07550891 0.04150283 0.0595734  0.07959308 0.02471638 0.06774862
 0.10402663 0.06876157 0.02787757 0.1068638  0.05414343 0.02092529
 0.01157017 0.00530935 0.03093414 0.01845048 0.00676999 0.01673392
 0.01787942 0.01438716 0.01834155 0.01338706 0.01767976 0.02142453
 0.04752651 0.02836444]
```

In [367]:
```
from sklearn.feature_selection import mutual_info_classif
# determine the mutual information
mutual_info = mutual_info_classif(xs, ys)
mutual_info
mutual_info = pd.Series(mutual_info)
mutual_info.index = xs.columns
mutual_info.sort_values(ascending=False)
```

Out[367]:
```
cp_asymptomatic           0.129954
exang                     0.106717
oldpeak                   0.082754
cp_atypical angina        0.080733
chol                      0.069091
thalch                    0.067161
sex                       0.045193
age                       0.041518
city_Switzerland          0.041358
cp_non-anginal            0.038596
thal_reversable defect    0.038336
thal_normal               0.036227
trestbps                  0.035629
city_Hungary              0.028854
rest_st-t abnormality     0.025249
slope_upsloping           0.021452
ca                        0.018530
rest_normal               0.015306
city_VA Long Beach        0.013759
fbs                       0.010967
thal_fixed defect         0.008177
city_Cleveland            0.000629
rest_lv hypertrophy       0.000000
slope_flat                0.000000
slope_downsloping         0.000000
cp_typical angina         0.000000
dtype: float64
```

In [368]:
```
mutual_info.sort_values(ascending=False).plot.bar(figsize=(20, 8))
```

Out[368]: <AxesSubplot:>



In [369]:
```
print(dict(zip(lr.get_params().keys(),lr.get_params().values())))
```

```
{'C': 1.0, 'class_weight': None, 'dual': False, 'fit_intercept': True, 'intercept_scaling': 1, 'l1_ratio': None, 'max_iter': 100, 'multi_class': 'auto', 'n_jobs': None, 'penalty': 'l2',
'random_state': None, 'solver': 'lbfgs', 'tol': 0.0001, 'verbose': 0, 'warm_start': False}
```

In [ ]:
```

```

In [370]:
```
#Logistic Regression HyperParameter Tuning
```

In [371]:
```
from sklearn.model_selection import GridSearchCV
parms={'penalty':['l1','l2'],'C':[0.001,.009,0.01,.09,1,5,10,25],}

logreg=LogisticRegression()
lrhyp=GridSearchCV(logreg,
                   param_grid=parms,
                   scoring='accuracy',cv=10)

lrhyp.fit(x_train,y_train)
print("Tuned Hyperparameters :", lrhyp.best_params_)
print("Accuracy :",lrhyp.best_score_)
```

```
Tuned Hyperparameters : {'C': 1, 'penalty': 'l2'}
Accuracy : 0.8289152165864495
```

In [ ]:
```

```

In [372]:
```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint
# Creating the hyperparameter grid

# Instantiating Decision Tree classifier
# tree = DecisionTreeClassifier()

# Instantiating RandomizedSearchCV object
tree_cv = RandomizedSearchCV(lr, parms, cv = 10)

tree_cv.fit(x_train, y_train)

# Print the tuned parameters and score
print("Tuned  Parameters: {}".format(tree_cv.best_params_))
print("Best score is {}".format(tree_cv.best_score_))
```

```
Tuned  Parameters: {'penalty': 'l2', 'C': 1}
Best score is 0.8289152165864495
```

In [373]:
```
#KNN Hyper
leaf_size = list(range(1,50))
n_neighbors = list(range(1,10))
p=[1,2]
#Convert to dictionary
knnparam = dict(leaf_size=leaf_size, n_neighbors=n_neighbors, p=p)


lrhyp=GridSearchCV(knn,
                   param_grid=knnparam,
                   scoring='accuracy',cv=10)

lrhyp.fit(x_train,y_train)
print("Tuned Hyperparameters :", lrhyp.best_params_)
print("Accuracy :",lrhyp.best_score_)
```

```
Tuned Hyperparameters : {'leaf_size': 1, 'n_neighbors': 9, 'p': 1}
Accuracy : 0.8398556090336913
```

In [374]:
```
tree_cv = RandomizedSearchCV(knn, knnparam, cv = 10)

tree_cv.fit(x_train, y_train)

# Print the tuned parameters and score
print("Tuned  Parameters: {}".format(tree_cv.best_params_))
print("Best score is {}".format(tree_cv.best_score_))
```

```
Tuned  Parameters: {'p': 2, 'n_neighbors': 5, 'leaf_size': 38}
Best score is 0.8302480562754535
```

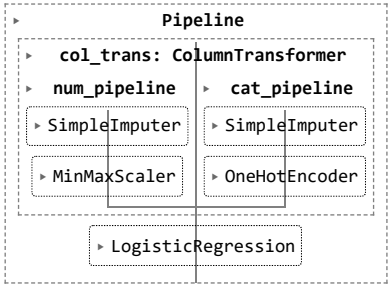
In [375]:
```
#XG Boost
```

In [376]:
```
from xgboost import XGBClassifier
xg = XGBClassifier()

# fit the model with the training data
xg.fit(x_train,y_train)

predict_train = xg.predict(x_train)
# print('\nTarget on train data',predict_train)

# Accuray Score on train dataset
accuracy_train = accuracy_score(y_train,predict_train)
print('\naccuracy_score on train dataset : ', accuracy_train)

# predict the target on the test dataset
predict_test = model.predict(x_test)
# print('\nTarget on test data',predict_test)

# Accuracy Score on test dataset
accuracy_test = accuracy_score(y_test,predict_test)
print('\naccuracy_score on test dataset : ', accuracy_test)
```

```
accuracy_score on train dataset :  1.0

accuracy_score on test dataset :  1.0
```

In [377]:
```
#Pipeline
```

In [380]:
```
def encode(df_pre):
    df2=df.replace({'num':{2:1,3:1,4:1}})

    return df_pre

df2 = encode(df)
num_cols = ['trestbps','chol','oldpeak','ca','thalch']

cat_cols = ['sex','dataset','cp','fbs','restecg','exang','slope','thal']
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler,StandardScaler,FunctionTransformer
from sklearn.pipeline import Pipeline,FeatureUnion
def get_dummies(X):
    return pd.get_dummies(X)
num_pipeline = Pipeline(steps=[
    ('impute', SimpleImputer(strategy='mean')),
    ('scale',MinMaxScaler())
])
cat_pipeline = Pipeline(steps=[
    ('impute', SimpleImputer(strategy='most_frequent')),
    ('one-hot',OneHotEncoder())
])
from sklearn.compose import ColumnTransformer

col_trans = ColumnTransformer(transformers=[
    ('num_pipeline',num_pipeline,num_cols),
    ('cat_pipeline',cat_pipeline,cat_cols)
    ],
    remainder='drop',
    n_jobs=-1)
from sklearn.linear_model import LogisticRegression

# clf=KNeighborsClassifier(n_neighbors=2)
clf = LogisticRegression(penalty= 'l2', C= 1)
clf_pipeline = Pipeline(steps=[
    ('col_trans', col_trans),
    ('model', clf)
])
from sklearn import set_config

set_config(display='diagram')
display(clf_pipeline)
from sklearn.model_selection import train_test_split

X = df[num_cols+cat_cols]
y = df['num']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state=142)
clf_pipeline.fit(X_train, y_train)

    # preds = clf_pipeline.predict(X_test)n
score = clf_pipeline.score(X_test, y_test)
print(i,f"Model score: {score}") # model accuracy
```




```
7 Model score: 0.657608695652174
```