notebookb1028bd394

September 7, 2020

**The main aim of the project is to apply logistic regression and random forest machine learning technique to study about their accuracies. The dataset is about factors on which heart disease depend. I have taken classification data over here,therefore applying logistic and random forest over here.

The features of dataset are as folows:

1.age

2.sex

3.chest pain type (4 values)

4.resting blood pressure

5.serum cholestoral in mg/dl

6.fasting blood sugar > 120 mg/dl

7.resting electrocardiographic results (values 0,1,2)

8.maximum heart rate achieved

9.exercise induced angina

10.oldpeak = ST depression induced by exercise relative to rest

11.the slope of the peak exercise ST segment

12.number of major vessels (0-3) colored by flourosopy 13.thal: 3 = normal; 6 = fixed defect; 7 = reversable defect 14.target-0(no heart disease)-1(have heart disease)

Importing different libraries for working with data.The dataset was a winzip file , therefore zipfile libraray is imported**

[1]:
```
# # This Python 3 environment comes with many helpful analytics libraries␣
↪installed
# # It is defined by the kaggle/python Docker image: https://github.com/kaggle/
↪docker-python
# # For example, here's several helpful packages to load

# import numpy as np # linear algebra
# import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# # Input data files are available in the read-only "../input/" directory
```

```
# # For example, running this (by clicking run or pressing Shift+Enter) will␣
↪list all files under the input directory

# import os
# for dirname, _, filenames in os.walk('/kaggle/input/heart-csv'):
#     for filename in filenames:
#         print(os.path.join(dirname, filename))

# # You can write up to 5GB to the current directory (/kaggle/working/) that␣
↪gets preserved as output when you create a version using "Save & Run All"
# # You can also write temporary files to /kaggle/temp/, but they won't be␣
↪saved outside of the current session
```

[2]:
```
import pandas as pd
import numpy as np
import zipfile
import seaborn as sns
import matplotlib.pyplot as plt
```

[3]:
```
df = pd.read_csv('heart.csv')
df.head()
```

[3]:
```
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   63    1   3       145   233    1        0      150      0      2.3      0
1   37    1   2       130   250    0        1      187      0      3.5      0
2   41    0   1       130   204    0        0      172      0      1.4      2
3   56    1   1       120   236    0        1      178      0      0.8      2
4   57    0   0       120   354    0        1      163      1      0.6      2

   ca  thal  target
0   0     1       1
1   0     2       1
2   0     2       1
3   0     2       1
4   0     2       1
```

[4]:
```
df.shape
```

[4]:
```
(303, 14)
```

[5]:
```
df.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   trestbps  303 non-null    int64
 4   chol      303 non-null    int64
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalach   303 non-null    int64
 8   exang     303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64
 11  ca        303 non-null    int64
 12  thal      303 non-null    int64
 13  target    303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

We see that one feature is float64, other features are int64. With this same method, we can easily see if there are any missing values. Here, there are none because each column contains 303 observations, the same number of rows we saw before with shape.

[6]:
```
df.describe().transpose()
```

[6]:

	count	mean	std	min	25%	50%	75%	max
age	303.0	54.366337	9.082101	29.0	47.5	55.0	61.0	77.0
sex	303.0	0.683168	0.466011	0.0	0.0	1.0	1.0	1.0
cp	303.0	0.966997	1.032052	0.0	0.0	1.0	2.0	3.0
trestbps	303.0	131.623762	17.538143	94.0	120.0	130.0	140.0	200.0
chol	303.0	246.264026	51.830751	126.0	211.0	240.0	274.5	564.0
fbs	303.0	0.148515	0.356198	0.0	0.0	0.0	0.0	1.0
restecg	303.0	0.528053	0.525860	0.0	0.0	1.0	1.0	2.0
thalach	303.0	149.646865	22.905161	71.0	133.5	153.0	166.0	202.0
exang	303.0	0.326733	0.469794	0.0	0.0	0.0	1.0	1.0
oldpeak	303.0	1.039604	1.161075	0.0	0.0	0.8	1.6	6.2
slope	303.0	1.399340	0.616226	0.0	1.0	1.0	2.0	2.0
ca	303.0	0.729373	1.022606	0.0	0.0	0.0	1.0	4.0
thal	303.0	2.313531	0.612277	0.0	2.0	2.0	3.0	3.0
target	303.0	0.544554	0.498835	0.0	0.0	1.0	1.0	1.0

**The describe method shows basic statistical characteristics of each numerical feature (int64 and float64 types): number of non-missing values, mean, standard deviation, range, median, 0.25 and 0.75 quartiles.With this I can get a general idea about feature mean ,standard deviation,min and max values , median adn total count of each numeric dataset

Count plot and distribution plot

Count plot is a nice way of getting to know about frequency of different features.So here I have use countplot with distplot to get a better idea about our dataset**

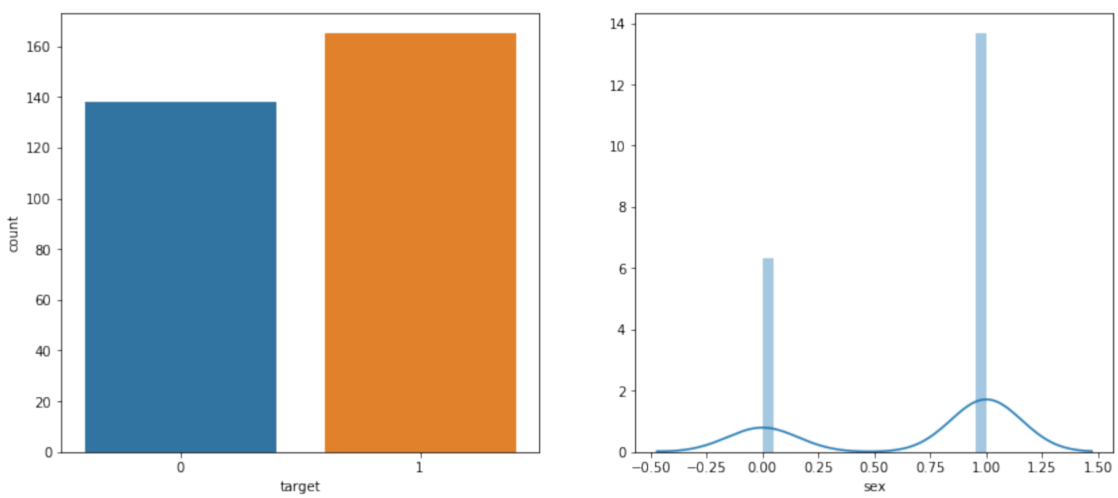
[7]:
```python
df.isnull().sum()
```

[7]:
```
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
target      0
dtype: int64
```

[8]:
```python
df.columns
```

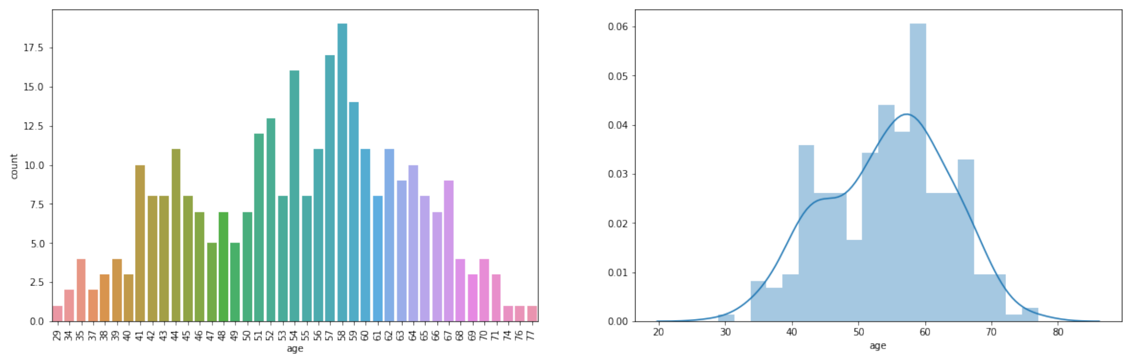
[8]:
```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

[9]:
```python
# Visualizing target variable Clicked on Ad
plt.figure(figsize = (14, 6))
plt.subplot(1,2,1)
sns.countplot(x = 'target', data = df)
plt.subplot(1,2,2)
sns.distplot(df["sex"], bins = 20)
plt.show()
```



So there are slightly more people with heart diseases. 1- Yes, 0 - No .

[10]:
```python
# Visualizing target variable Clicked on Ad
plt.figure(figsize = (20, 6))
plt.subplot(1,2,1)
chart=sns.countplot(data = df,x='age')
plt.subplot(1,2,2)
sns.distplot(df["age"], bins = 20)
chart.set_xticklabels(chart.get_xticklabels(),rotation=90)
plt.show()
```



Let's check which gender has the maximum heart diseases ???

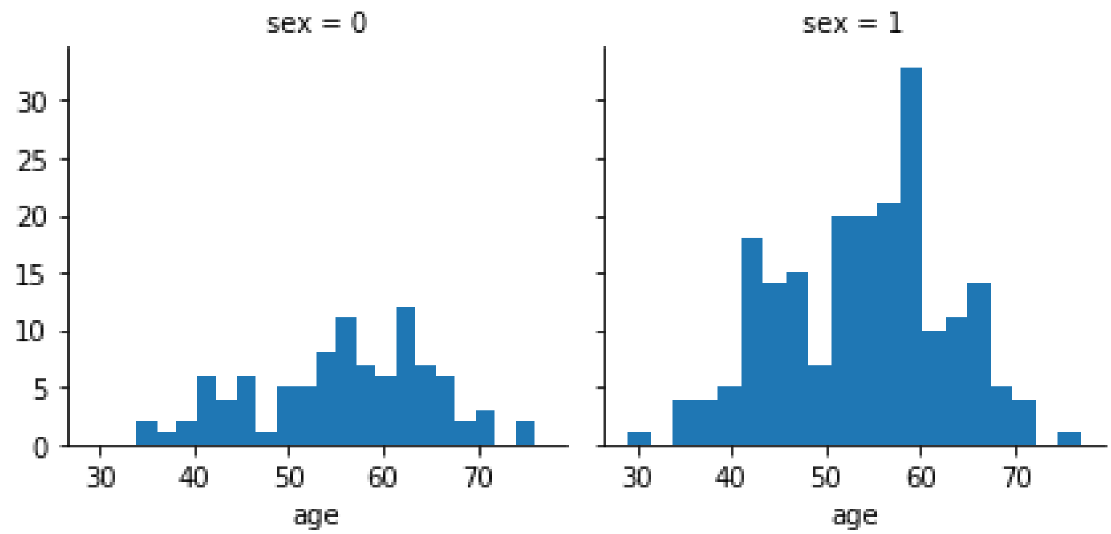
[11]:
```python
df[["target", "sex"]].groupby(['sex'], as_index=False).mean().
↪sort_values(by='target', ascending=False)
```

[11]:

	sex	target
0	0	0.750000
1	1	0.449275

[12]:
```python
g = sns.FacetGrid(df, col='sex')
g.map(plt.hist, 'age', bins=20)
```

[12]:
```
<seaborn.axisgrid.FacetGrid at 0x121b08bf708>
```

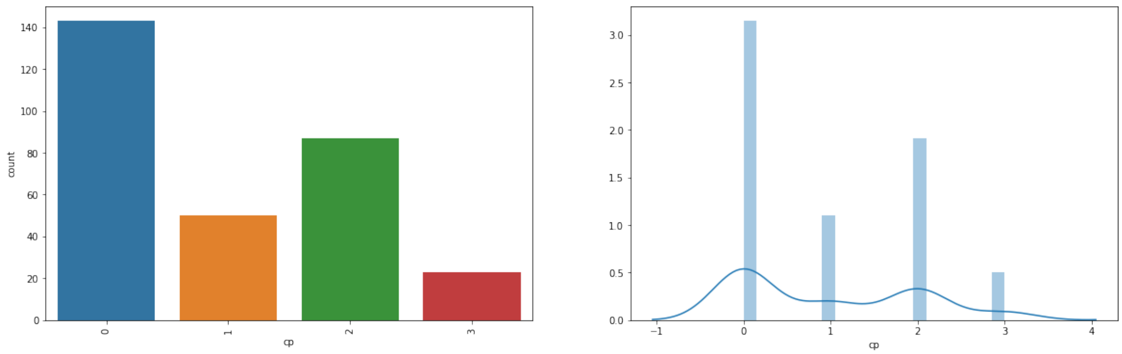


Male has less heart diseases !!! Savage !!! Was not expecting If we see the count , for females between age (40-70), the count is averaged at 7.5 , but for men the count is high between the same age group !!!

[13]:
```
df.columns
```

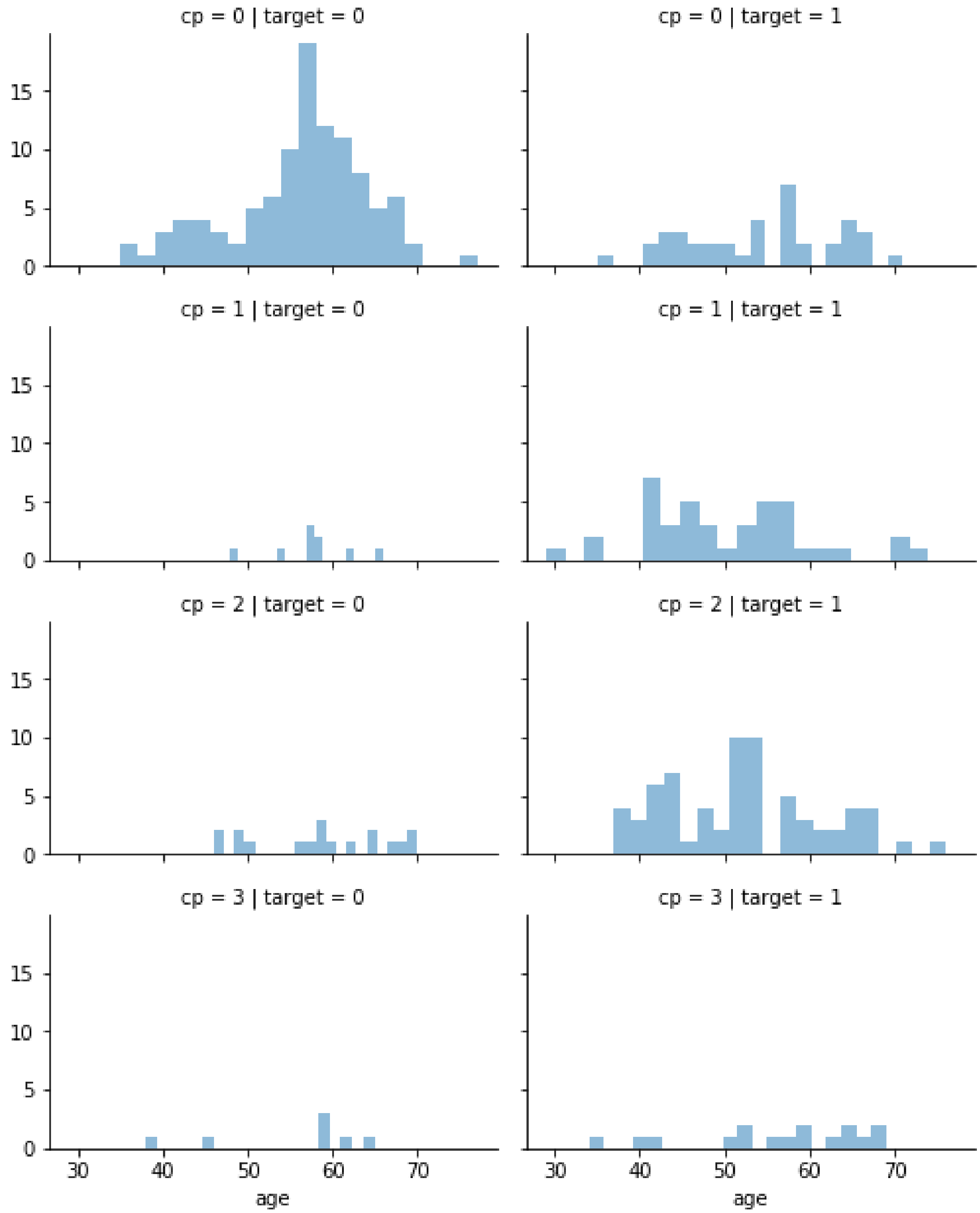
[13]:
```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

[14]:
```
plt.figure(figsize = (20, 6))
plt.subplot(1,2,1)
chart=sns.countplot(data = df,x='cp')
plt.subplot(1,2,2)
sns.distplot(df["cp"], bins = 20)
chart.set_xticklabels(chart.get_xticklabels(),rotation=90)
plt.show()
```



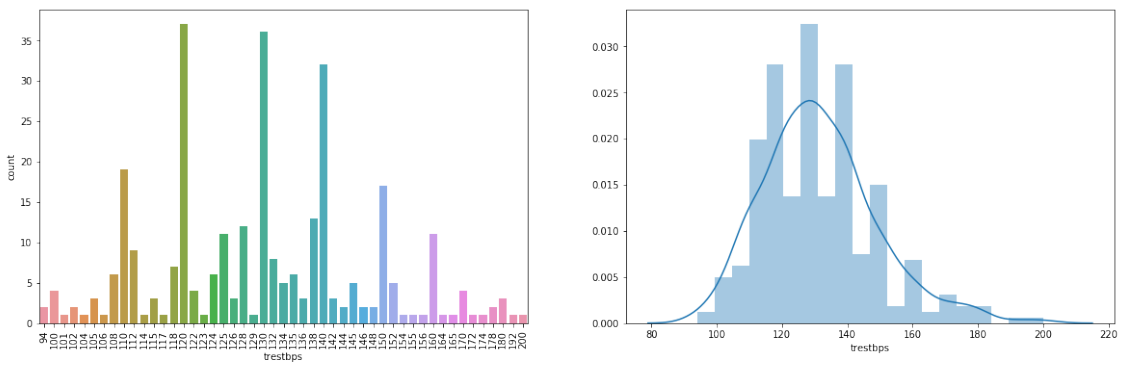
[15]:
```
grid = sns.FacetGrid(df, col='target', row='cp', size=2.2, aspect=1.6)
grid.map(plt.hist, 'age', alpha=.5, bins=20)
grid.add_legend();
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\axisgrid.py:243: UserWarning:
The `size` parameter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```



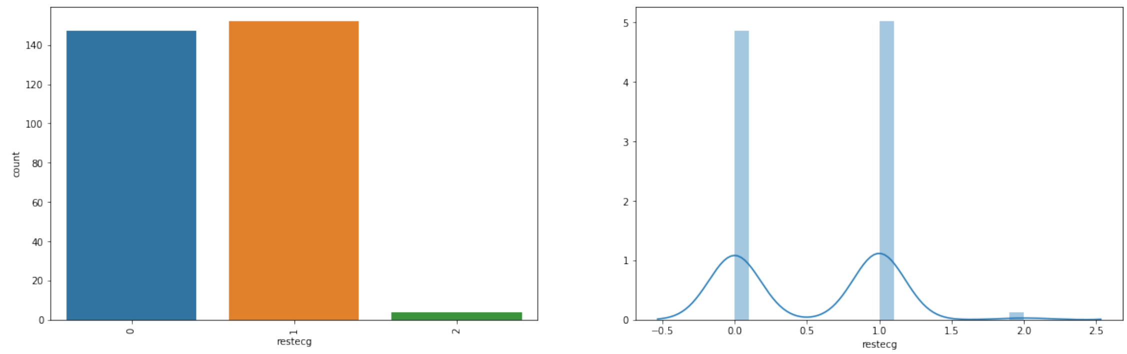
Seeing from the chart it is observed that woman suffer majority from typical angina and from men is mostly - atypical angina, non-anginal pain.

[16]:
```
plt.figure(figsize = (20, 6))
plt.subplot(1,2,1)
chart=sns.countplot(data = df,x='trestbps')
plt.subplot(1,2,2)
sns.distplot(df["trestbps"], bins = 20)
chart.set_xticklabels(chart.get_xticklabels(),rotation=90)
plt.show()
```

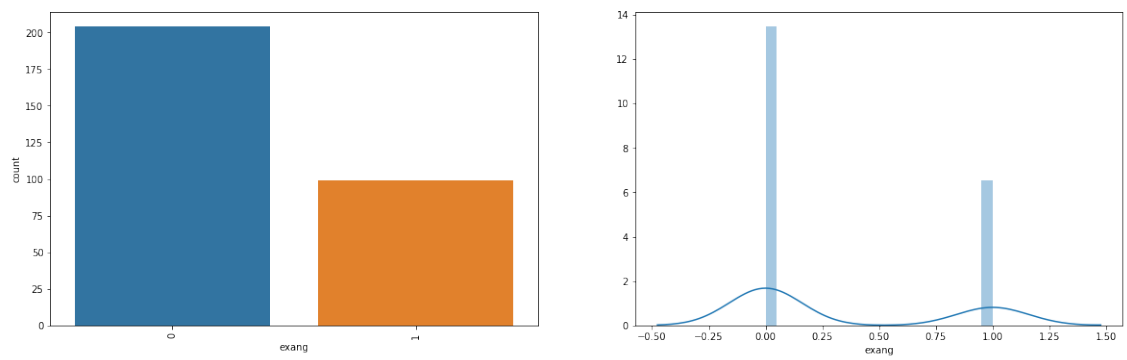


[17]:
```
# plt.figure(figsize = (20, 6))
# plt.subplot(1,2,1)
# chart=sns.countplot(data = df,x='fbs')
# plt.subplot(1,2,2)
# sns.distplot(df["fbs"], bins = 20)
# chart.set_xticklabels(chart.get_xticklabels(),rotation=90)
# plt.show()
```

[18]:
```
plt.figure(figsize = (20, 6))
plt.subplot(1,2,1)
chart=sns.countplot(data = df,x='restecg')
plt.subplot(1,2,2)
sns.distplot(df['restecg'], bins = 20)
chart.set_xticklabels(chart.get_xticklabels(),rotation=90)
plt.show()
```



[19]:
```
plt.figure(figsize = (20, 6))
plt.subplot(1,2,1)
chart=sns.countplot(data = df,x='exang')
plt.subplot(1,2,2)
sns.distplot(df['exang'], bins = 20)
chart.set_xticklabels(chart.get_xticklabels(),rotation=90)
plt.show()
```



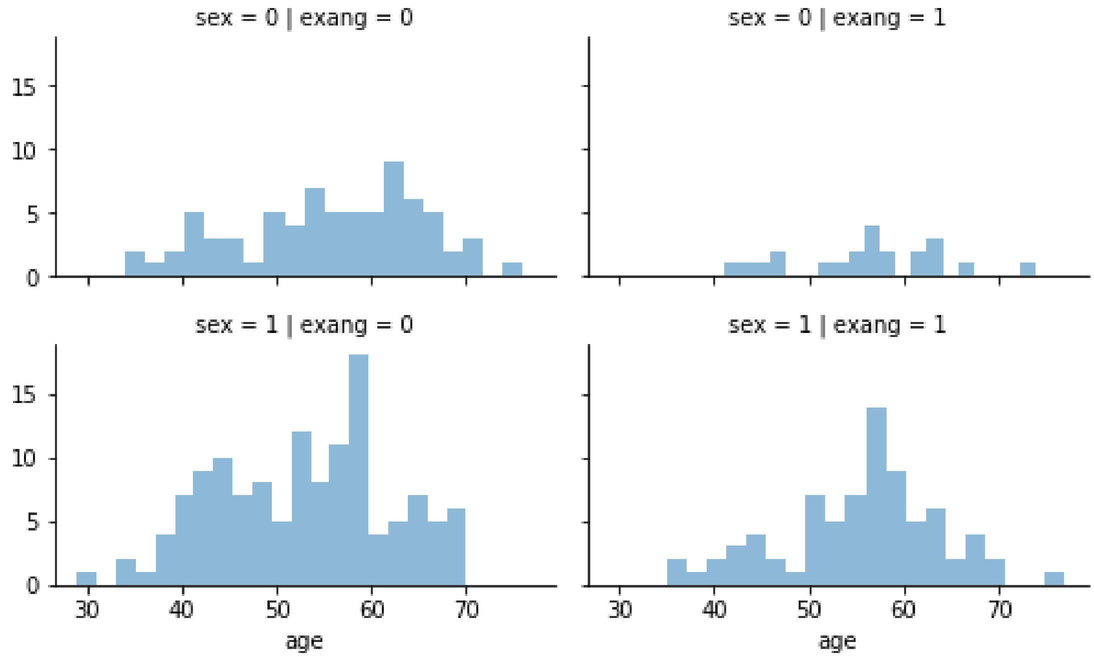
[20]:
```
df[["sex", "exang"]].groupby(['exang'], as_index=False).mean().
 ↪sort_values(by='sex', ascending=False)
```

[20]:

	exang	sex
1	1	0.777778
0	0	0.637255

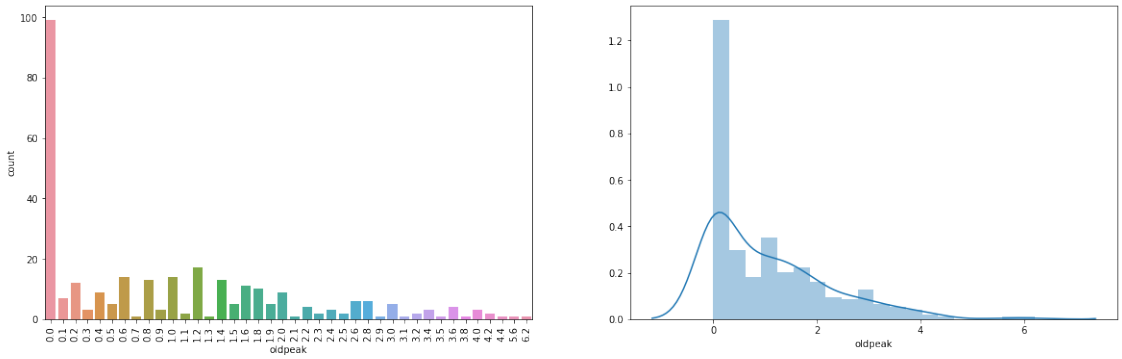
[21]:
```
grid = sns.FacetGrid(df, col='exang', row='sex', size=2.2, aspect=1.6)
grid.map(plt.hist, 'age', alpha=.5, bins=20)
grid.add_legend();
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\axisgrid.py:243: UserWarning:
The `size` parameter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```



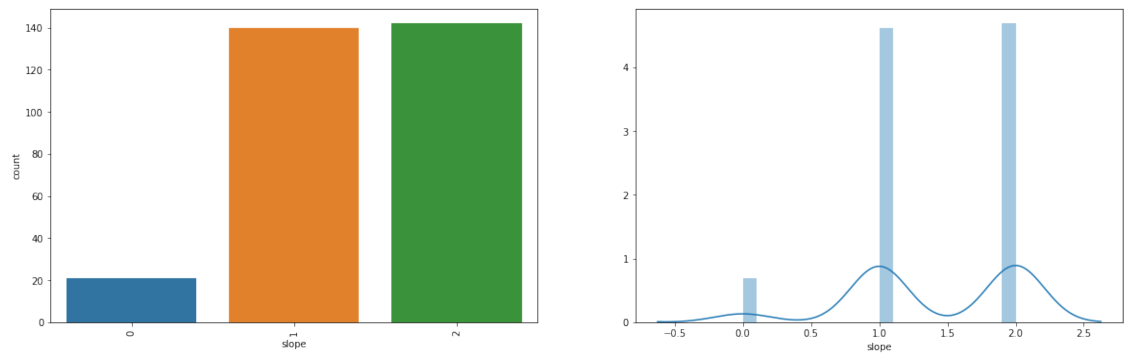
Based on the above two distributions we can see the exersized induced pain is high for men compared to woman.

[22]:
```
plt.figure(figsize = (20, 6))
plt.subplot(1,2,1)
chart=sns.countplot(data = df,x='oldpeak')
plt.subplot(1,2,2)
sns.distplot(df['oldpeak'], bins = 20)
chart.set_xticklabels(chart.get_xticklabels(),rotation=90)
plt.show()
```



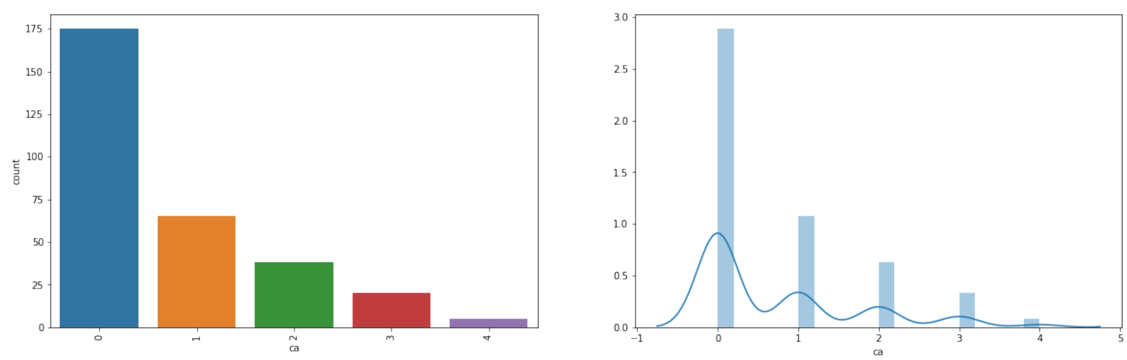
[23]:

```
plt.figure(figsize = (20, 6))
plt.subplot(1,2,1)
chart=sns.countplot(data = df,x='slope')
plt.subplot(1,2,2)
sns.distplot(df['slope'], bins = 20)
chart.set_xticklabels(chart.get_xticklabels(),rotation=90)
plt.show()
```



[24]:

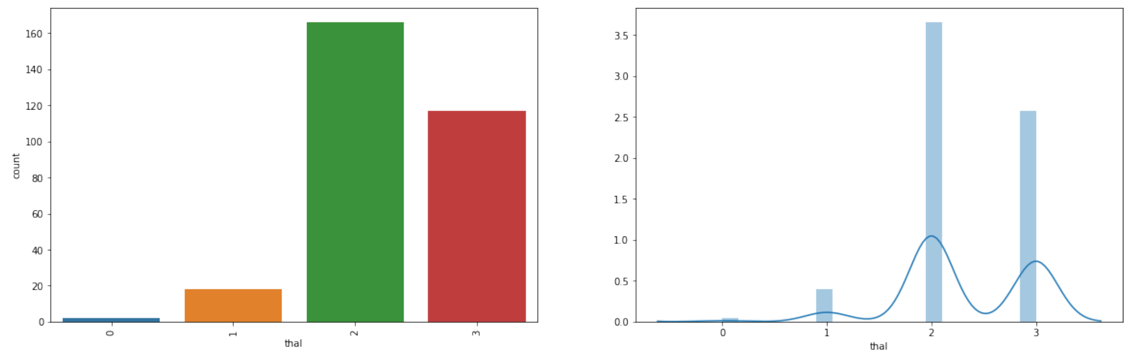
```
plt.figure(figsize = (20, 6))
plt.subplot(1,2,1)
chart=sns.countplot(data = df,x='ca')
plt.subplot(1,2,2)
sns.distplot(df['ca'], bins = 20)
chart.set_xticklabels(chart.get_xticklabels(),rotation=90)
plt.show()
```



[25]:

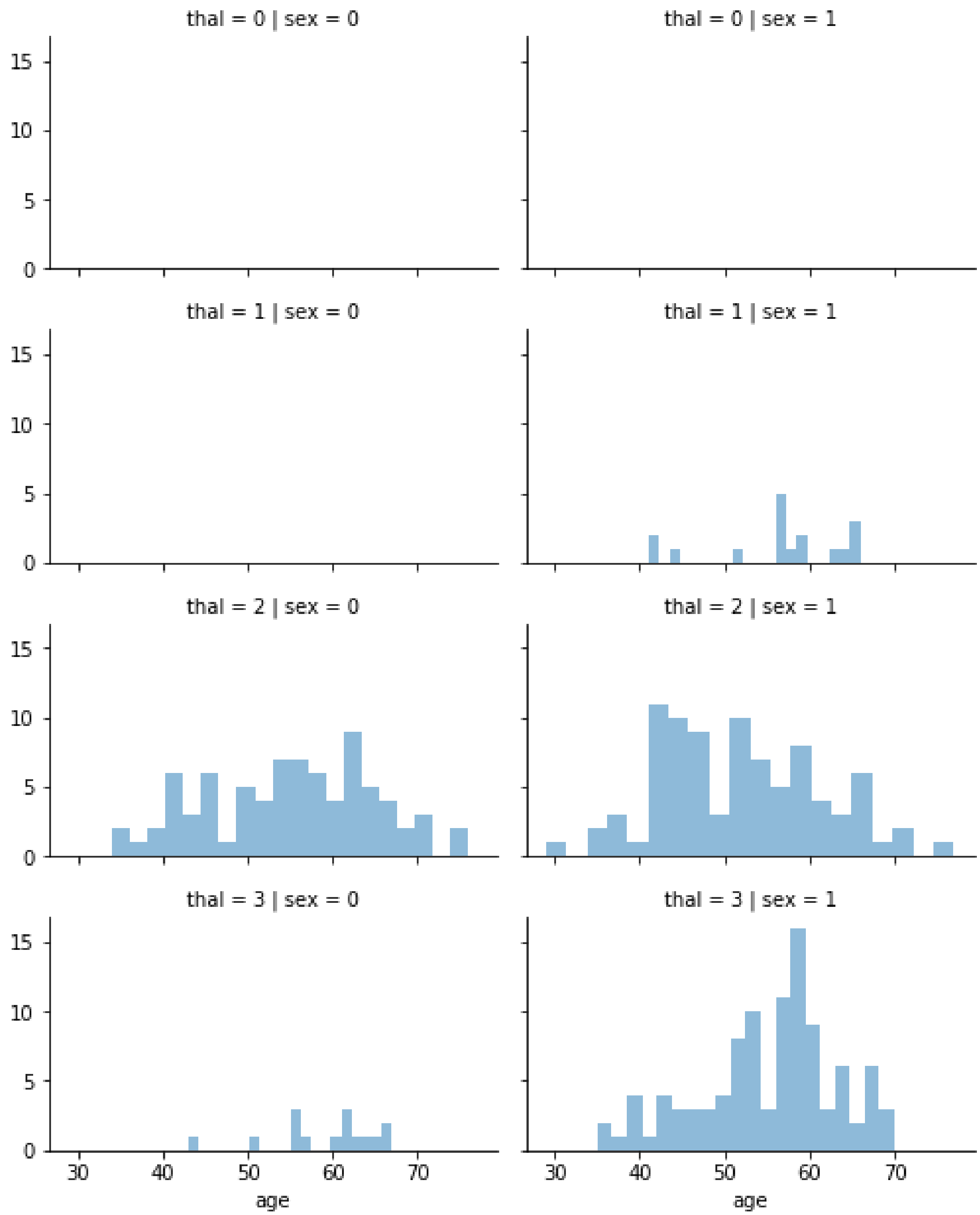
```
plt.figure(figsize = (20, 6))
plt.subplot(1,2,1)
chart=sns.countplot(data = df,x='thal')
plt.subplot(1,2,2)
sns.distplot(df['thal'], bins = 20)
```

```
chart.set_xticklabels(chart.get_xticklabels(),rotation=90)
plt.show()
```



[26]:
```
grid = sns.FacetGrid(df, col='sex', row='thal', size=2.2, aspect=1.6)
grid.map(plt.hist, 'age', alpha=.5, bins=20)
grid.add_legend();
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\axisgrid.py:243: UserWarning:
The `size` parameter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```



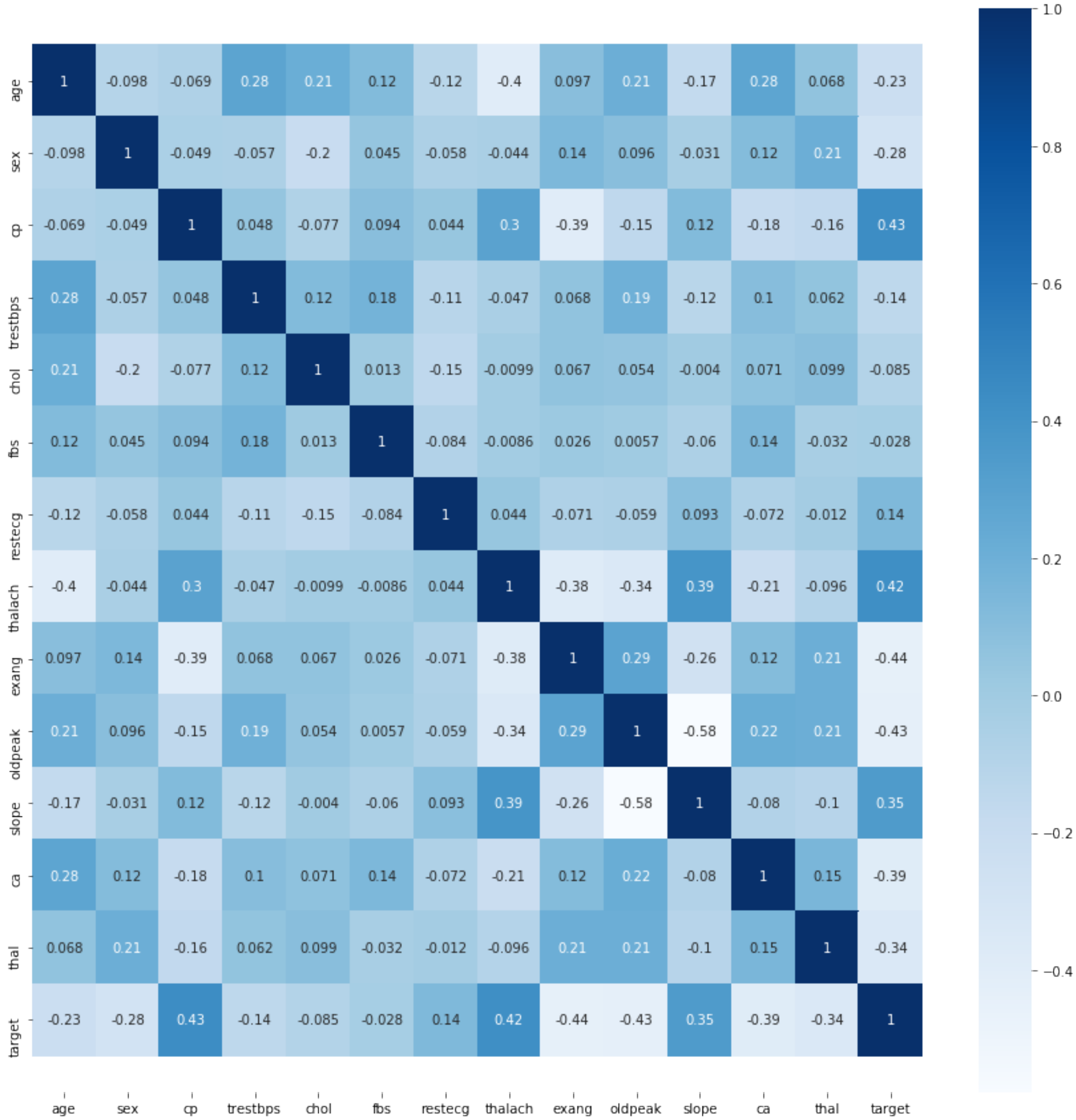
Based on the observations men and woman suffers the most on thalassemia with reversible defect

Let's use heatmaps.....

[27]:
```
fig,ax=plt.subplots(figsize=(15,15))
sns.heatmap(df.corr(), cmap='Blues', annot = True)
```

```
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

[27]: (14.5, -0.5)



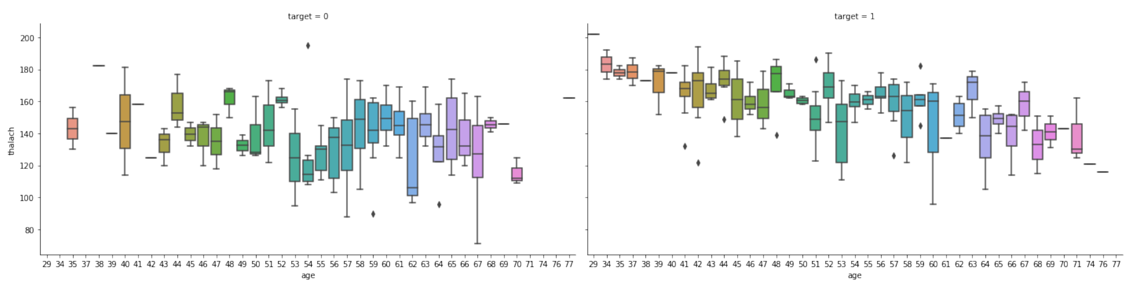
As one can see from above heatmap,dark color represent high correlation between features. 1.Target depend highly on chest pain,maximum heart rate achieved(thalach) 2.Target have negative corelation with depend on thal,ca,exang and oldpeak

Factor plot analysis

[28]:
```
sns.factorplot(x="age", y="thalach", col="target", data=df, kind="box",size=5,␣
↪aspect=2.0)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\categorical.py:3669:
UserWarning: The `factorplot` function has been renamed to `catplot`. The
original name will be removed in a future release. Please update your code. Note
that the default `kind` in `factorplot` (`'point'`) has changed `'strip'` in
`catplot`.
  warnings.warn(msg)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\categorical.py:3675:
UserWarning: The `size` parameter has been renamed to `height`; please update
your code.
  warnings.warn(msg, UserWarning)
```

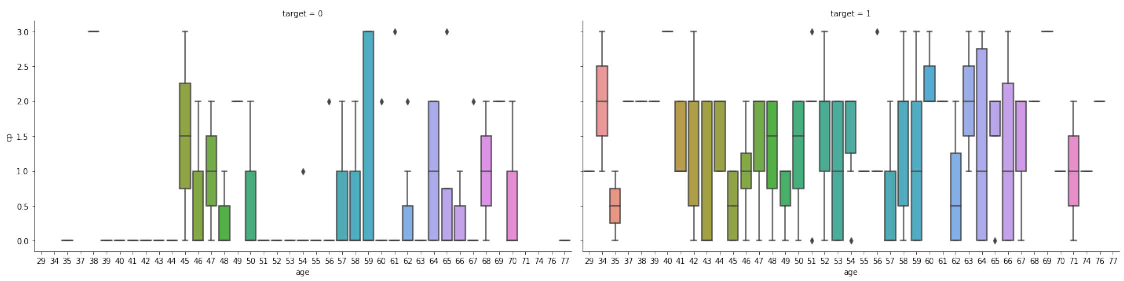
[28]: <seaborn.axisgrid.FacetGrid at 0x121b0899c88>



[29]:
```
sns.factorplot(x="age", y="cp", col="target", data=df, kind="box",size=5,␣
↪aspect=2.0)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\categorical.py:3669:
UserWarning: The `factorplot` function has been renamed to `catplot`. The
original name will be removed in a future release. Please update your code. Note
that the default `kind` in `factorplot` (`'point'`) has changed `'strip'` in
`catplot`.
  warnings.warn(msg)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\categorical.py:3675:
UserWarning: The `size` parameter has been renamed to `height`; please update
your code.
  warnings.warn(msg, UserWarning)
```

[29]: <seaborn.axisgrid.FacetGrid at 0x121b078de08>



[30]:

```
# Importing train_test_split from sklearn.model_selection family
from sklearn.model_selection import train_test_split
# Import LogisticRegression from sklearn.linear_model family
from sklearn.linear_model import LogisticRegression
# Assigning Numerical columns to X & y only as model can only take numbers
X = df[['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach','exang', 'oldpeak', 'slope', 'ca', 'thal']]
y = df['target']
# Splitting the data into train & test sets
# test_size is % of data that we want to allocate & random_state ensures a specific set of random splits on our data because
#this train test split is going to occur randomly
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
# We dont have to use stratify method in train_tst_split to handle class distribution as its not imbalanced and does contain equal number of classes i.e 1's and 0's
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(203, 13) (203,)
(100, 13) (100,)
```

[31]:

```
# Instantiate an instance of the linear regression model (Creating a linear regression object)
logreg = LogisticRegression()
# Fit the model on training data using a fit method
model = logreg.fit(X_train,y_train)
model
```

```
C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
```

```
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

[31]:
```
LogisticRegression(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)
```

[32]:
```
# The predict method just takes X_test as a parameter, which means it just␣
↪takes the features to draw predictions
predictions = logreg.predict(X_test)
# Below are the results of predicted click on Ads
predictions[0:20]
```

[32]:
```
array([0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0],
      dtype=int64)
```

[33]:
```
# Importing classification_report from sklearn.metrics family
from sklearn.metrics import classification_report

# Printing classification_report to see the results
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.75      0.79      0.77        42
           1       0.84      0.81      0.82        58

    accuracy                           0.80       100
   macro avg       0.79      0.80      0.80       100
weighted avg       0.80      0.80      0.80       100
```

[34]:
```
# Importing a pure confusion matrix from sklearn.metrics family
from sklearn.metrics import confusion_matrix

# Printing the confusion_matrix
print(confusion_matrix(y_test, predictions))
```

```
[[33  9]
 [11 47]]
```

[35]:

```python
import  statsmodels.api  as sm
from scipy import stats

X2   = sm.add_constant(X_train)
est  = sm.OLS(y_train, X2)
est2 = est.fit()
print(est2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 target   R-squared:                       0.558
Model:                            OLS   Adj. R-squared:                  0.527
Method:                 Least Squares   F-statistic:                     18.33
Date:                Mon, 07 Sep 2020   Prob (F-statistic):           4.25e-27
Time:                        13:39:11   Log-Likelihood:                -64.228
No. Observations:                 203   AIC:                             156.5
Df Residuals:                     189   BIC:                             202.8
Df Model:                          13
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.5596      0.343      1.630      0.105      -0.118       1.237
age            0.0036      0.003      1.096      0.274      -0.003       0.010
sex           -0.1491      0.057     -2.623      0.009      -0.261      -0.037
cp             0.1098      0.027      4.056      0.000       0.056       0.163
trestbps      -0.0011      0.002     -0.700      0.485      -0.004       0.002
chol       -7.306e-05      0.000     -0.152      0.879      -0.001       0.001
fbs            0.0251      0.078      0.323      0.747      -0.128       0.179
restecg        0.0472      0.047      0.998      0.320      -0.046       0.141
thalach        0.0021      0.001      1.592      0.113      -0.001       0.005
exang         -0.1600      0.062     -2.595      0.010      -0.282      -0.038
oldpeak       -0.0310      0.029     -1.076      0.283      -0.088       0.026
slope          0.1190      0.048      2.466      0.015       0.024       0.214
ca            -0.1807      0.030     -5.954      0.000      -0.241      -0.121
thal          -0.1688      0.043     -3.884      0.000      -0.255      -0.083
==============================================================================
Omnibus:                        4.284   Durbin-Watson:                   2.256
Prob(Omnibus):                  0.117   Jarque-Bera (JB):                4.333
Skew:                          -0.352   Prob(JB):                        0.115
Kurtosis:                       2.870   Cond. No.                     4.64e+03
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 4.64e+03. This might indicate that there are
```

strong multicollinearity or other numerical problems.

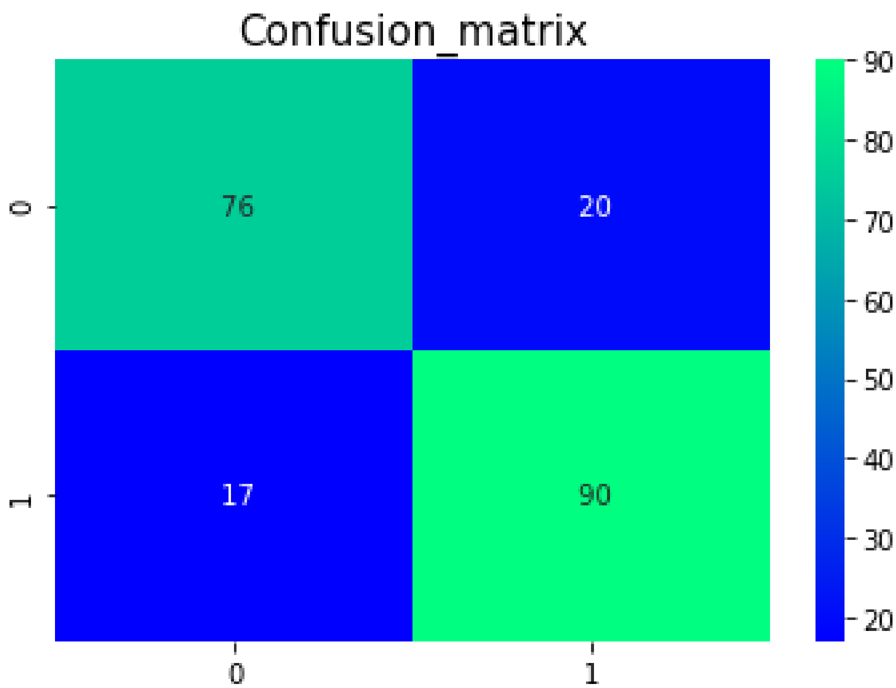
[36]:
```
#Creating K fold Cross-validation
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
kf = KFold(n_splits=10, shuffle=True, random_state=42)
scores = cross_val_score(model, # model
                         X_train, # Feature matrix
                         y_train, # Target vector
                         cv=kf, # Cross-validation technique
                         scoring="accuracy", # Loss function
                         n_jobs=-1) # Use all CPU scores
print('10 fold CV accuracy: %.3f +/- %.3f' % (np.mean(scores), np.std(scores)))
```

```
10 fold CV accuracy: 0.863 +/- 0.060
```

[37]:
```
from sklearn.metrics import roc_auc_score, accuracy_score, confusion_matrix
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(criterion='gini', n_estimators=400,
                            min_samples_split=10,min_samples_leaf=1,
                            max_features='auto',oob_score=True,
                            random_state=42,n_jobs=-1)
rf.fit(X_train,y_train)
# Predict using model
rf_training_pred = rf.predict(X_train)
rf_training_prediction = accuracy_score(y_train, rf_training_pred)

print("Accuracy of Random Forest training set:", ␣
 ↪round(rf_training_prediction,3))

from sklearn.model_selection import cross_val_predict
print('The cross validated score for Random Forest Classifier is:',round(scores.
 ↪mean()*100,2))
y_pred = cross_val_predict(rf,X_train,y_train,cv=10)
sns.heatmap(confusion_matrix(y_train,y_pred),annot=True,fmt='3.
 ↪0f',cmap="winter")
plt.title('Confusion_matrix', y=1.05, size=15)
```

```
Accuracy of Random Forest training set: 0.961
The cross validated score for Random Forest Classifier is: 86.26
```

[37]:
```
Text(0.5, 1.05, 'Confusion_matrix')
```



[38]:
```
new_df = df.copy() # just to keep the original dataframe unchanged
# Assigning Numerical columns to X & y only as model can only take numbers
X1 = df[[ 'sex', 'cp', 'chol', 'fbs', 'restecg', 'thalach', 'slope', 'thal']]
y1 = df['target']
# Splitting the data into train & test sets
# test_size is % of data that we want to allocate & random_state ensures a␣
↪specific set of random splits on our data because
#this train test split is going to occur randomly
X_train1, X_test1, y_train1, y_test1 = train_test_split(X1, y1, test_size=0.33,␣
↪random_state=42)
# We dont have to use stratify method in train_tst_split to handle class␣
↪distribution as its not imbalanced and does contain equal number of classes␣
↪i.e 1's and 0's
print(X_train1.shape, y_train1.shape)
print(X_test1.shape, y_test1.shape)
logreg = LogisticRegression()
# Fit the model on training data using a fit method
model1 = logreg.fit(X_train1,y_train1)
model1
# The predict method just takes X_test as a parameter, which means it just␣
↪takes the features to draw predictions
predictions1 = logreg.predict(X_test1)
# Below are the results of predicted click on Ads
predictions1[0:20]
```

```
# Printing classification_report to see the results
print(classification_report(y_test1, predictions1))
```

```
(203, 8) (203,)
(100, 8) (100,)
              precision    recall  f1-score   support

           0       0.77      0.79      0.78        42
           1       0.84      0.83      0.83        58

    accuracy                           0.81       100
   macro avg       0.80      0.81      0.81       100
weighted avg       0.81      0.81      0.81       100


C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

[39]:
```
# Importing a pure confusion matrix from sklearn.metrics family
from sklearn.metrics import confusion_matrix

# Printing the confusion_matrix
print(confusion_matrix(y_test1, predictions1))
```

```
[[33  9]
 [10 48]]
```

[40]:
```
import  statsmodels.api  as sm
from scipy import stats

X21   = sm.add_constant(X_train1)
est1 = sm.OLS(y_train1, X21)
est21 = est1.fit()
print(est21.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 target   R-squared:                       0.443
Model:                            OLS   Adj. R-squared:                  0.420
```

```
Method:                 Least Squares   F-statistic:                     19.30
Date:                Mon, 07 Sep 2020   Prob (F-statistic):           3.14e-21
Time:                        13:39:23   Log-Likelihood:                -87.600
No. Observations:                 203   AIC:                             193.2
Df Residuals:                     194   BIC:                             223.0
Df Model:                           8
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.1717      0.251      0.685      0.494      -0.323       0.666
sex           -0.2283      0.060     -3.785      0.000      -0.347      -0.109
cp             0.1464      0.028      5.151      0.000       0.090       0.202
chol          -0.0004      0.001     -0.842      0.401      -0.001       0.001
fbs           -0.0150      0.083     -0.181      0.857      -0.179       0.149
restecg        0.0563      0.052      1.091      0.277      -0.046       0.158
thalach        0.0046      0.001      3.504      0.001       0.002       0.007
slope          0.1504      0.046      3.261      0.001       0.059       0.241
thal          -0.1963      0.047     -4.168      0.000      -0.289      -0.103
==============================================================================
Omnibus:                        3.297   Durbin-Watson:                   2.288
Prob(Omnibus):                  0.192   Jarque-Bera (JB):                2.736
Skew:                          -0.175   Prob(JB):                        0.255
Kurtosis:                       2.551   Cond. No.                     2.77e+03
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 2.77e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

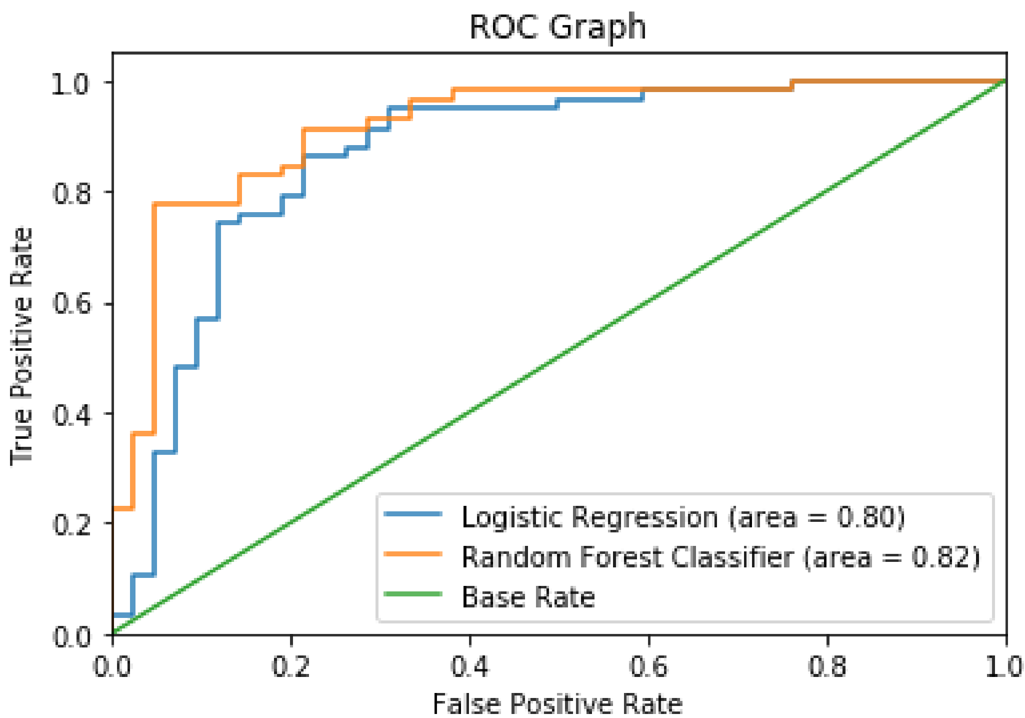
[41]:
```
print ("\n\n ---Logistic Regression Model---")
lr_auc = roc_auc_score(y_test, model.predict(X_test))

print ("Logistic Regression AUC = %2.2f" % lr_auc)
print(classification_report(y_test, model.predict(X_test)))

print ("\n\n ---Logistic Regression Model deleting some features---")
lr_auc1 = roc_auc_score(y_test1, model1.predict(X_test1))

print ("Logistic Regression AUC = %2.2f" % lr_auc)
print(classification_report(y_test1, model1.predict(X_test1)))

print ("\n\n ---Random Forest Model---")
rf_roc_auc = roc_auc_score(y_test, rf.predict(X_test))
```

```python
print ("Random Forest AUC = %2.2f" % rf_roc_auc)
print(classification_report(y_test, rf.predict(X_test)))
```

```
 ---Logistic Regression Model---
Logistic Regression AUC = 0.80
              precision    recall  f1-score   support

           0       0.75      0.79      0.77        42
           1       0.84      0.81      0.82        58

    accuracy                           0.80       100
   macro avg       0.79      0.80      0.80       100
weighted avg       0.80      0.80      0.80       100



 ---Logistic Regression Model deleting some features---
Logistic Regression AUC = 0.80
              precision    recall  f1-score   support

           0       0.77      0.79      0.78        42
           1       0.84      0.83      0.83        58

    accuracy                           0.81       100
   macro avg       0.80      0.81      0.81       100
weighted avg       0.81      0.81      0.81       100



 ---Random Forest Model---
Random Forest AUC = 0.82
              precision    recall  f1-score   support

           0       0.80      0.79      0.80        42
           1       0.85      0.86      0.85        58

    accuracy                           0.83       100
   macro avg       0.83      0.82      0.82       100
weighted avg       0.83      0.83      0.83       100
```

[42]:
```python
# Create ROC Graph
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, model.predict_proba(X_test)[:,1])
rf_fpr, rf_tpr, rf_thresholds = roc_curve(y_test, rf.predict_proba(X_test)[:,1])
```

```
plt.figure()

# Plot Logistic Regression ROC
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % lr_auc)

# Plot Random Forest ROC
plt.plot(rf_fpr, rf_tpr, label='Random Forest Classifier (area = %0.2f)' %␣
 ↪rf_roc_auc)


# Plot Base Rate ROC
plt.plot([0,1], [0,1],label='Base Rate')

plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Graph')
plt.legend(loc="lower right")
plt.show()
```



[43]:
```
columns = X.columns
train = pd.DataFrame(np.atleast_2d(X_train), columns=columns) # Converting␣
↪numpy array list into dataframes
```

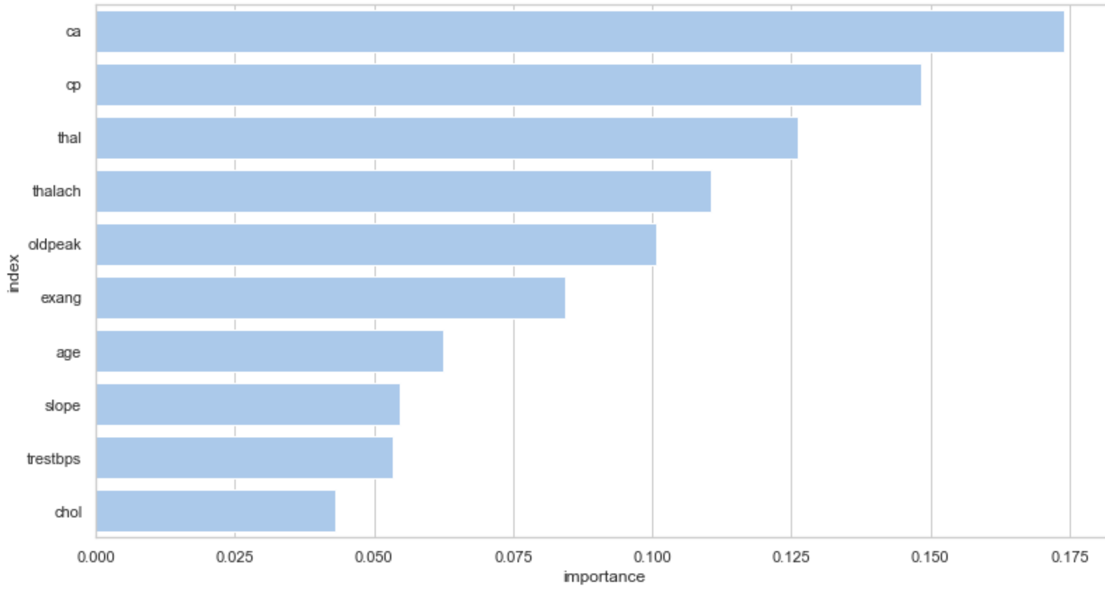
[44]:
```
# Get Feature Importances
feature_importances = pd.DataFrame(rf.feature_importances_,
                                   index = train.columns,
                                    columns=['importance']).
↪sort_values('importance', ascending=False)
feature_importances = feature_importances.reset_index()
feature_importances.head(10)
```

[44]:

	index	importance
0	ca	0.173951
1	cp	0.148279
2	thal	0.126150
3	thalach	0.110456
4	oldpeak	0.100775
5	exang	0.084257
6	age	0.062424
7	slope	0.054618
8	trestbps	0.053496
9	chol	0.043117

[45]:
```
sns.set(style="whitegrid")

# Initialize the matplotlib figure
f, ax = plt.subplots(figsize=(13, 7))

# Plot the Feature Importance
sns.set_color_codes("pastel")
sns.barplot(x="importance", y='index', data=feature_importances[0:10],
            label="Total", color="b")
```

[45]: <matplotlib.axes._subplots.AxesSubplot at 0x121b55bbb08>



ca
cp
thal
thalach
oldpeak
exang
age
slope
trestbps
chol
index
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
importance