Predicting heart disease using machine learning

This notebooks looks into using various python based machine learning and Data science libraries in an attempt to build machine learning model capable of predicting whether or not someone has heart disease based on their medical attributes.

We're going to take the followiing approach:

1. Problem definiation
2. Data
3. Evaluation
4. Features
5. Modelling
6. Experimentation

1. Problem Definition

In our case, the problem we will be exploring is binary classification (a sample can only be one of two things).

This is because we're going to be using a number of differnet features (pieces of information) about a person to predict whether they have heart disease or not.

In a statement,

Given clinical parameters about a patient, can we predict whether or not they have heart disease?

2. Data

What you'll want to do here is dive into the data your problem definition is based on. This may involve, sourcing, defining different parameters, talking to experts about it and finding out what you should expect.

The original data came from the Cleveland database from UCI Machine Learning Repository.

Howevever, we've downloaded it in a formatted way from Kaggle.

The original database contains 76 attributes, but here only 14 attributes will be used. Attributes (also called features) are the variables what we'll use to predict our target variable.

Attributes and features are also referred to as independent variables and a target variable can be referred to as a dependent variable.

We use the independent variables to predict our dependent variable.

Or in our case, the independent variables are a patients different medical attributes and the dependent variable is whether or not they have heart disease.

3. Evaluation

The evaluation metric is something you might define at the start of a project.

Since machine learning is very experimental, you might say something like,

If we can reach 95% accuracy at predicting whether or not a patient has heart disease during the proof of concept, we'll pursure this project.

The reason this is helpful is it provides a rough goal for a machine learning engineer or data scientist to work towards.

However, due to the nature of experimentation, the evaluation metric may change over time.

4. Features

Features are different parts of the data. During this step, you'll want to start finding out what you can about the data.

One of the most common ways to do this, is to create a data dictionary.

Heart Disease Data Dictionary A data dictionary describes the data you're dealing with. Not all datasets come with them so this is where you may have to do your research or ask a subject matter **EXPERT** (someone who knows about the data) for more.

The following are the features we'll use to predict our target variable (heart disease or no heart disease).

1. age - age in years
2. sex - (1 = male; 0 = female)
3. cp - chest pain type
   `0: Typical angina: chest pain related decrease blood supply to the heart`

```
1: Atypical angina: chest pain not related to heart
2: Non-anginal pain: typically esophageal spasms (non heart related)
3: Asymptomatic: chest pain not showing signs of disease
```

4. trestbps - resting blood pressure (in mm Hg on admission to the hospital)
   ```
   -anything above 130-140 is typically cause for concern
   ```
5. chol - serum cholestoral in mg/dl
   ```
   serum = LDL + HDL + .2 * triglycerides
   above 200 is cause for concern
   ```
6. fbs - (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
   ```
   '>126' mg/dL signals diabetes
   ```
7. restecg - resting electrocardiographic results
   ```
   0: Nothing to note
   1: ST-T Wave abnormality
       can range from mild symptoms to severe problems
       signals non-normal heart beat
   2: Possible or definite left ventricular hypertrophy
       Enlarged heart's main pumping chamber
   ```
8. thalach - maximum heart rate achieved
9. exang - exercise induced angina (1 = yes; 0 = no)
10. oldpeak - ST depression induced by exercise relative to rest
    ```
    looks at stress of heart during excercise
    unhealthy heart will stress more
    ```
11. slope - the slope of the peak exercise ST segment
    ```
    0: Upsloping: better heart rate with excercise (uncommon)
    1: Flatsloping: minimal change (typical healthy heart)
    2: Downslopins: signs of unhealthy heart
    ```
12. ca - number of major vessels (0-3) colored by flourosopy
    ```
    colored vessel means the doctor can see the blood passing through
    the more blood movement the better (no clots)
    ```
13. thal - thalium stress result
    ```
    1,3: normal
    6: fixed defect: used to be defect but ok now
    7: reversable defect: no proper blood movement when excercising
    ```
14. target - have disease or not (1=yes, 0=no) (= the predicted attribute)

Note: No personal identifiable information (PPI) can be found in the dataset.

It's a good idea to save these to a Python dictionary or in an external file, so we can look at them later without coming back here.

Preparing the tools At the start of any project, it's custom to see the required libraries imported in a big chunk like you can see below.

However, in practice, your projects may import libraries as you go. After you've spent a couple of hours working on your problem, you'll probably want to do some tidying up. This is where you may want to consolidate every library you've used at the top of your notebook (like the cell below).

In [ ]:

```
```

In [1]:

```
# Import all the tools we need
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline


# Models from Sklearn
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier

# Model Evaluations
from sklearn.model_selection import train_test_split, cross_val_score, RandomizedSearchCV, GridSearchCV
from sklearn.metrics import precision_score, f1_score, recall_score,confusion_matrix, classification_report, pl
```

Load data

In [2]:

```
df= pd.read_csv("heart-disease.csv")
df.shape
```

Out[2]:

```
(303, 14)
```

Data Exploration (Exploratory Data Analysis or EDA)

The goal is to know more about the data and become a subject matter export on the dataset you're working with

1. What question are you trying to solve?
2. What type of data do we have and how do we treat different types?
3. What's missing from the data and how to deal with it
4. Where are the outliers and why should we care about them?
5. How can you add, change or remove features to get more out of your data?

In [3]:
```
df.head()
```

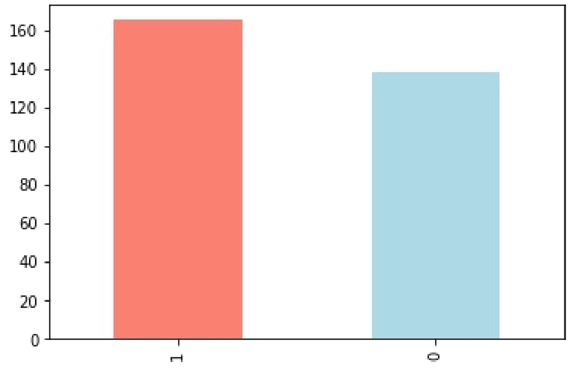
Out[3]:

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

In [4]:
```
# Let's find out how many of each classes there are
df["target"].value_counts()
```

Out[4]:
```
1    165
0    138
Name: target, dtype: int64
```

In [5]:
```
df["target"].value_counts().plot(kind="bar",color=["salmon","lightblue"]);
```



In [6]:
```
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
```

In [7]:
```
# Are there any missing data
df.isna().sum()
```

Out[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
```

In [8]:

```
df.describe(include="all")
```

Out[8]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.399340
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.000000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000

Heart disease frequency according to sex

In [9]:

```
df.sex.value_counts()
```

Out[9]:

```
1    207
0     96
Name: sex, dtype: int64
```

In [10]:

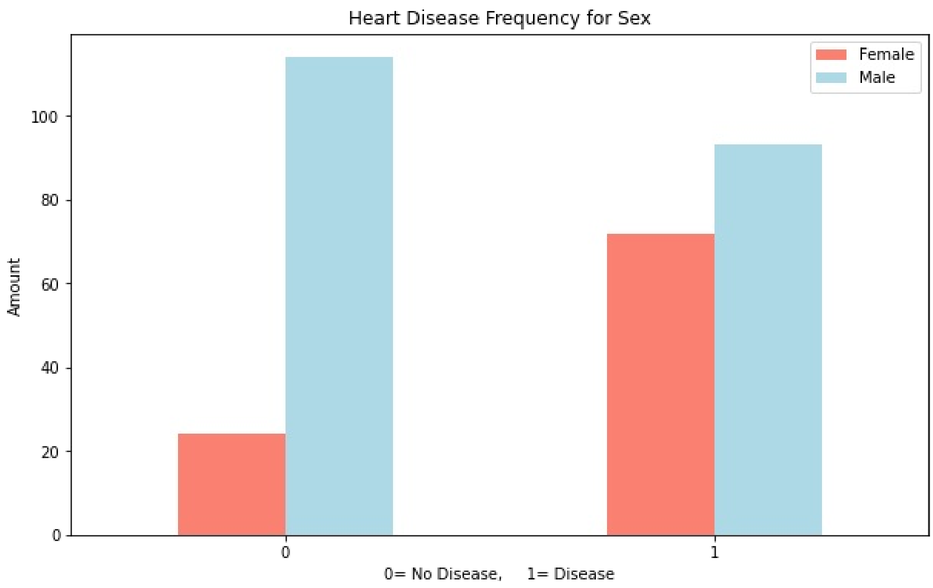
```
# Compare target column with sex column
pd.crosstab(df.target, df.sex)
```

Out[10]:

sex	0	1
target		
0	24	114
1	72	93

In [11]:

```
# Create a plot of crosstab
pd.crosstab(df.target, df.sex).plot(kind="bar",
                                    figsize=(10,6),
                                    color= ["salmon", "lightblue"])
plt.title("Heart Disease Frequency for Sex")
plt.xlabel("0= No Disease,     1= Disease")
plt.ylabel("Amount")
plt.legend(["Female","Male"])
plt.xticks(rotation=0);
```

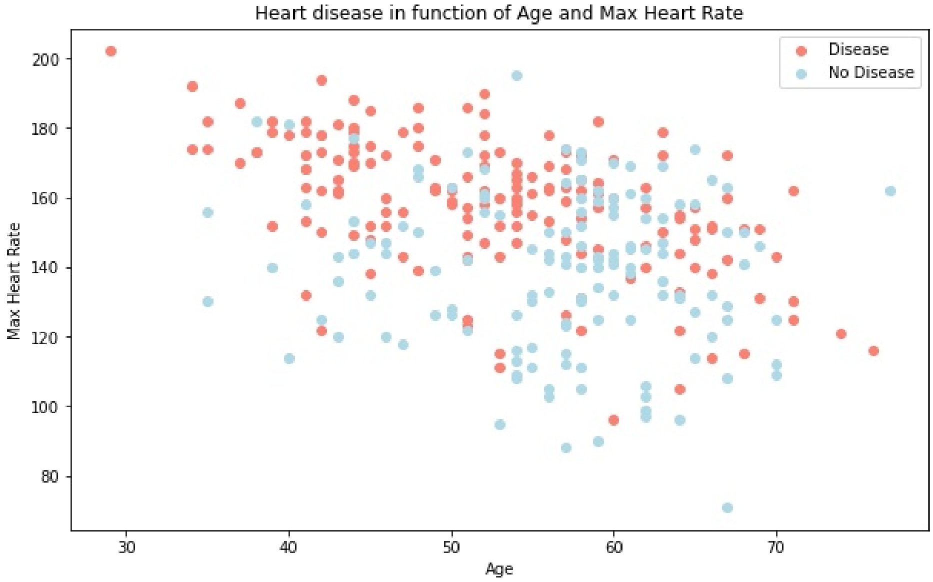


Age vs. Max Heart Rate (thalach) for Heart Disease

In [12]:

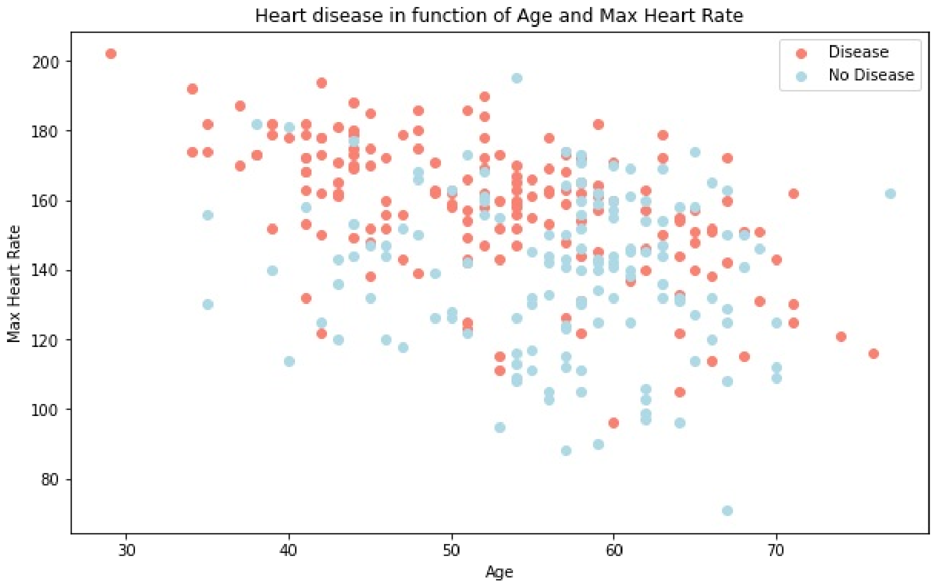
```
#Create another figure
plt.figure(figsize=(10,6))

#Scatter with positive examples
plt.scatter(df.age[df.target==1],# we're taking the Age column from our data were Target ==1
           df.thalach[df.target==1],# we're taking the Age column from our data were Target ==1
           color="salmon")

#Scattter with Negative examples
plt.scatter(df.age[df.target==0],
           df.thalach[df.target==0],
           color="lightblue");
```

```python
plt.title("Heart disease in function of Age and Max Heart Rate")
plt.xlabel("Age")
plt.ylabel("Max Heart Rate")
plt.legend(["Disease","No Disease"]);
```



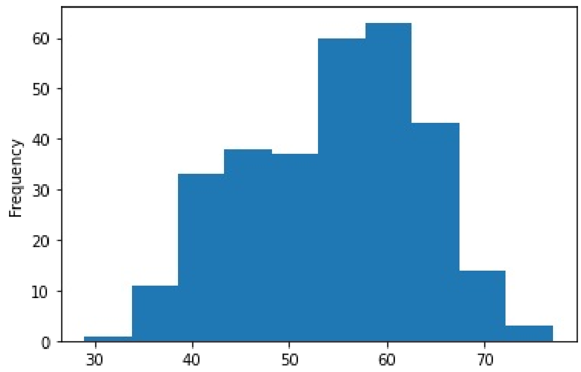
In [13]:

```python
#Create another figure
plt.figure(figsize=(10,6))

#Scatter with positive examples
plt.scatter(df["age"][df["target"]==1],
           df["thalach"][df["target"]==1],
           color="salmon")

#Scatter with negative example
plt.scatter(df["age"][df["target"]==0],
           df["thalach"][df["target"]==0],
           color="lightblue");

plt.title("Heart disease in function of Age and Max Heart Rate")
plt.xlabel("Age")
plt.ylabel("Max Heart Rate")
plt.legend(["Disease","No Disease"]);
```

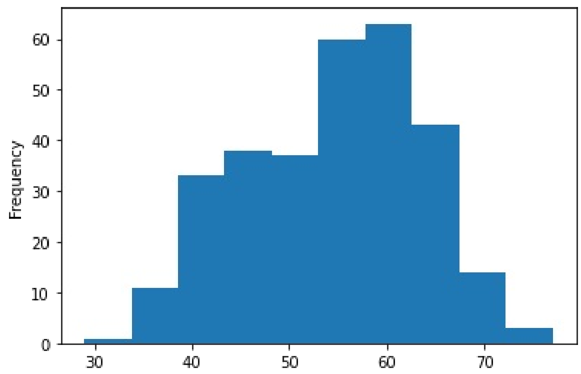


In [34]:

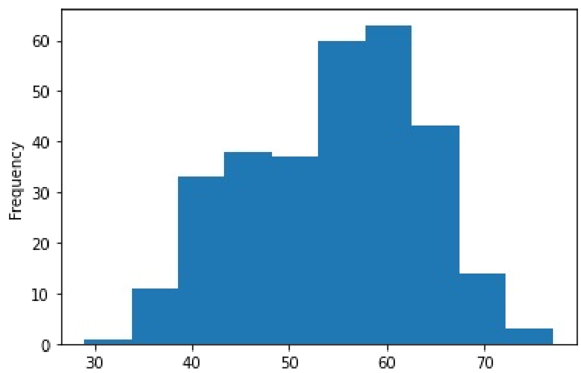
```python
#Check the distribution of the Age column with histogram
df.age.plot(kind="hist");
```



In [37]:
```
df["age"].plot.hist();
```



In [38]:
```
df.age.plot.hist();
```



Heart Disease Frequency per Chest Pain Type

CP- chest pain type

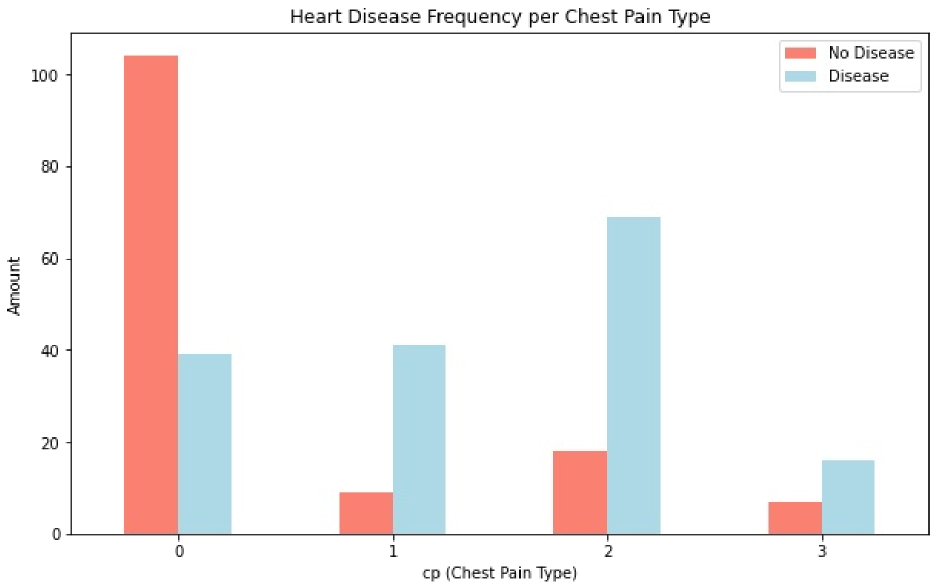
1. Typical angina: Chest pain related, decrease blood supply to the heart
2. Atypical angina: Chest pain not related to the heart
3. Non-anginal pain: typical esophageal spam (not heart related)
4. Asymptomatic: Chest pain not showing sing of disease

In [14]:
```
pd.crosstab(df.cp,df.target)
```

Out[14]:

target	0	1
cp		
0	104	39
1	9	41
2	18	69
3	7	16

In [15]:
```
pd.crosstab(df.cp,df.target).plot(kind="bar",
                                   figsize=(10,6),
                                   color=["salmon","lightblue"])

plt.title("Heart Disease Frequency per Chest Pain Type")
plt.xlabel("cp (Chest Pain Type)")
plt.ylabel("Amount")
plt.legend(["No Disease", "Disease"])
plt.xticks(rotation=0);
```

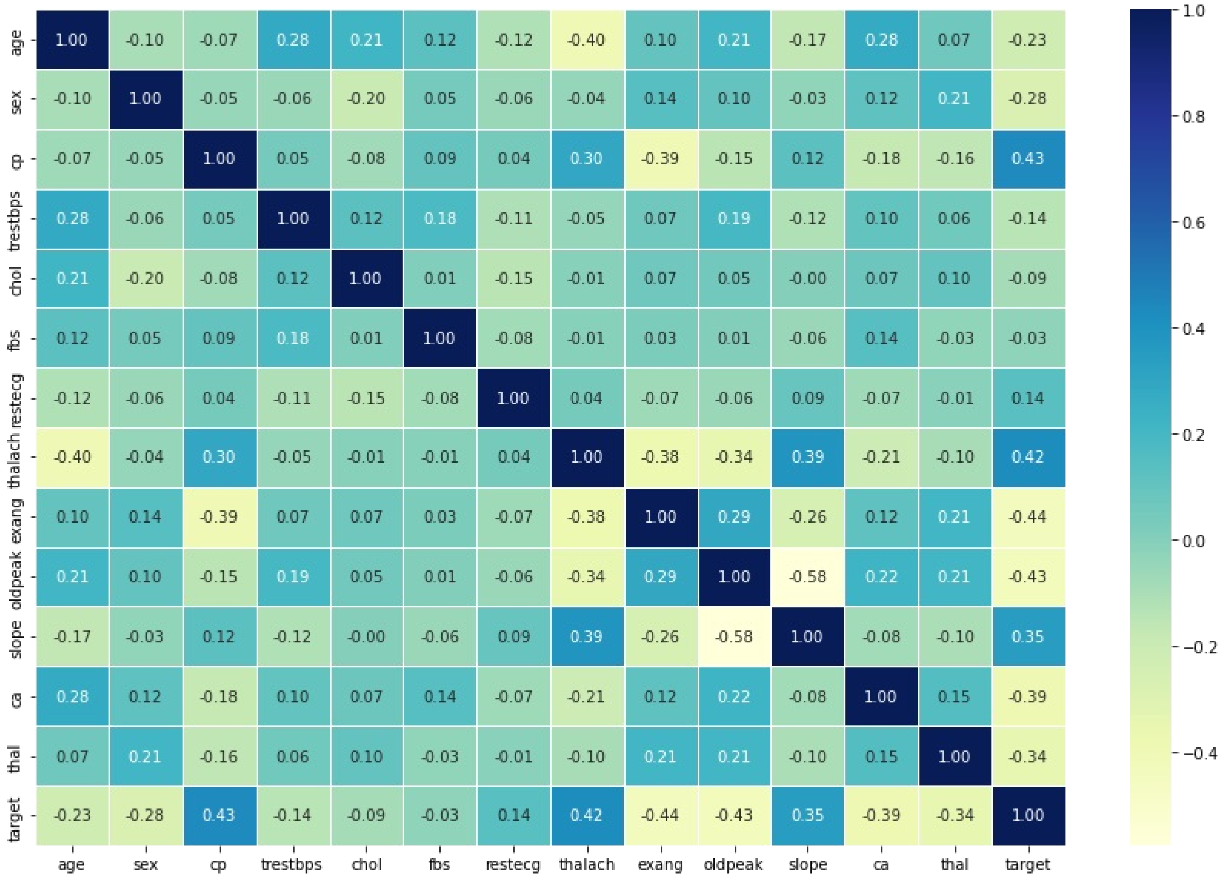


In [16]:
```
#Compare the Independent variables to each other, let's make a correlation matrix
df.corr()
```

Out[16]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	
age	1.000000	-0.098447	-0.068653	0.279351	0.213678	0.121308	-0.116211	-0.398522	0.096801	0.210013	-0.168814	0.276326	0.0
sex	-0.098447	1.000000	-0.049353	-0.056769	-0.197912	0.045032	-0.058196	-0.044020	0.141664	0.096093	-0.030711	0.118261	0.2
cp	-0.068653	-0.049353	1.000000	0.047608	-0.076904	0.094444	0.044421	0.295762	-0.394280	-0.149230	0.119717	-0.181053	-0.1
trestbps	0.279351	-0.056769	0.047608	1.000000	0.123174	0.177531	-0.114103	-0.046698	0.067616	0.193216	-0.121475	0.101389	0.0
chol	0.213678	-0.197912	-0.076904	0.123174	1.000000	0.013294	-0.151040	-0.009940	0.067023	0.053952	-0.004038	0.070511	0.0
fbs	0.121308	0.045032	0.094444	0.177531	0.013294	1.000000	-0.084189	-0.008567	0.025665	0.005747	-0.059894	0.137979	-0.0
restecg	-0.116211	-0.058196	0.044421	-0.114103	-0.151040	-0.084189	1.000000	0.044123	-0.070733	-0.058770	0.093045	-0.072042	-0.0
thalach	-0.398522	-0.044020	0.295762	-0.046698	-0.009940	-0.008567	0.044123	1.000000	-0.378812	-0.344187	0.386784	-0.213177	-0.0
exang	0.096801	0.141664	-0.394280	0.067616	0.067023	0.025665	-0.070733	-0.378812	1.000000	0.288223	-0.257748	0.115739	0.2
oldpeak	0.210013	0.096093	-0.149230	0.193216	0.053952	0.005747	-0.058770	-0.344187	0.288223	1.000000	-0.577537	0.222682	0.2
slope	-0.168814	-0.030711	0.119717	-0.121475	-0.004038	-0.059894	0.093045	0.386784	-0.257748	-0.577537	1.000000	-0.080155	-0.1
ca	0.276326	0.118261	-0.181053	0.101389	0.070511	0.137979	-0.072042	-0.213177	0.115739	0.222682	-0.080155	1.000000	0.1
thal	0.068001	0.210041	-0.161736	0.062210	0.098803	-0.032019	-0.011981	-0.096439	0.206754	0.210244	-0.104764	0.151832	1.0
target	-0.225439	-0.280937	0.433798	-0.144931	-0.085239	-0.028046	0.137230	0.421741	-0.436757	-0.430696	0.345877	-0.391724	-0.3

In [17]:
```
# Lets make our correlation matrix more better
corr_matrix= df.corr()

fig,ax= plt.subplots(figsize=(15,10))
ax= sns.heatmap(corr_matrix,
               annot=True,
               linewidths=0.5,
               fmt=".2f",
               cmap="YlGnBu");
```



In [ ]:

5. Modelling

In [54]:
```
df.head()
```

Out[54]:

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

In [18]:
```
# Create data
x= df.drop("target",axis=1)
y= df["target"]
```

In [19]:
```
#Split data into training and test sets
np.random.seed(42)

x_train,x_test,y_train,y_test= train_test_split(x,y,test_size=0.2)
```

Now lets build our machine learning model

We're going to try out 3 different machine learning model

1. Logistic Regression
2. K-Nearest Neighbor Classifier
3. Random Forest

In [20]:
```
# Put models in a dictionary
models= {"Logistic Regression":LogisticRegression(),
        "KNN":KNeighborsClassifier(),
        "Random Forest":RandomForestClassifier()}
# Create a function to fit and score models
def fit_and_score(models,x_train,x_test,y_train,y_test):
    """
    Fits and evaluate machine learning models.
    models: different sklearn machine learning models.
    x_train: training data(no labels)
    x_test: testing data(no labels)
    y_train: train labels
```

```
    y_test: test labels
    """
    #Set up random score
    np.random.seed(42)
    #Make a dictionary to keep model scores
    model_scores = {}
    #Loop through models
    for name,model in models.items():
        #Fit the model to the data
        model.fit(x_train, y_train)
        #Evaluate the model and append it's score to model_score
        model_scores[name]= model.score(x_test,y_test)
    return model_scores
```

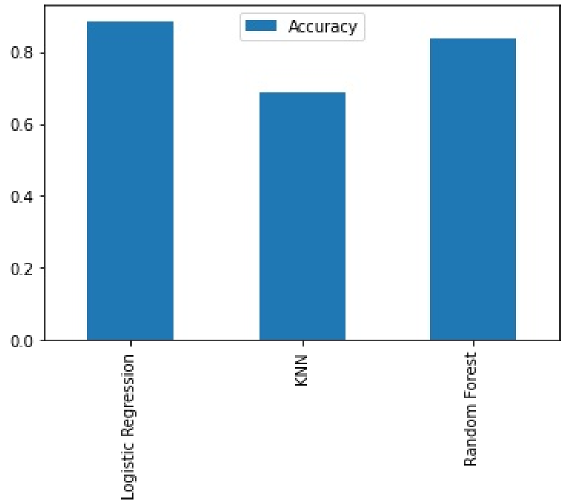
In [21]:
```
model_scores= fit_and_score(models= models,
                            x_train= x_train,
                            x_test= x_test,
                            y_train= y_train,
                            y_test= y_test)

model_scores
```

```
C:\Users\Aboya\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:444: ConvergenceWarning: lbfgs fai
led 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
  n_iter_i = _check_optimize_result(
```

Out[21]:
```
{'Logistic Regression': 0.8852459016393442,
 'KNN': 0.6885245901639344,
 'Random Forest': 0.8360655737704918}
```

Model comparison

In [22]:
```
model_compare= pd.DataFrame(model_scores, index=["Accuracy"])
model_compare.T.plot.bar();
```



Now we've got a basseline model.... and we know that a model's first prediction is not what we should base our next steps off. what should we do?

let's look at the following:

- Hyperparameter Tuning
- Feature Importance
- Confusion Matrix
- Cross_validation
- Precision
- Recall
- F1 Score
- Classification Report
- ROC Curve
- Area Under The Curve (AUC)

Hyperparameter Tuning (by hand)

In [23]:
```
# Let's tune KNN

train_scores= []
```

```
test_scores= []

#Create a list of different values for n_neighbors

neighbors= range(1,21)

#setup KNN instance
knn= KNeighborsClassifier()

#Loop through different n_neighbors
for i in neighbors:
    knn.set_params(n_neighbors=i)

    #fit the algorithm
    knn.fit(x_train,y_train)

    #Update the training score
    train_scores.append(knn.score(x_train,y_train))

    #Update the test score
    test_scores.append(knn.score(x_test,y_test))
```

In [24]:

```
train_scores
```

Out[24]:

```
[1.0,
 0.8099173553719008,
 0.7727272727272727,
 0.743801652892562,
 0.7603305785123967,
 0.7520661157024794,
 0.743801652892562,
 0.7231404958677686,
 0.71900826446281,
 0.6942148760330579,
 0.7272727272727273,
 0.6983471074380165,
 0.6900826446280992,
 0.6942148760330579,
 0.6859504132231405,
 0.6735537190082644,
 0.6859504132231405,
 0.6652892561983471,
 0.6818181818181818,
 0.6694214876033058]
```

In [25]:

```
test_scores
```

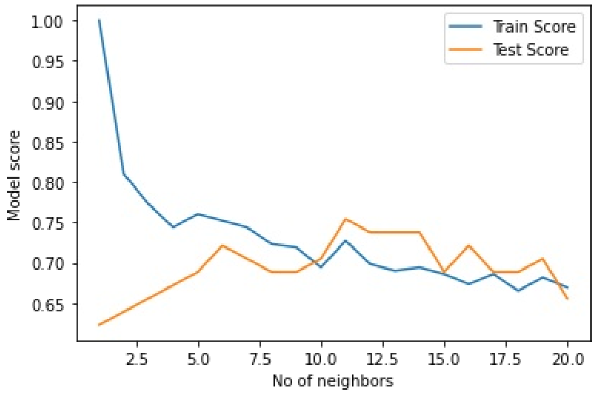
Out[25]:

```
[0.6229508196721312,
 0.639344262295082,
 0.6557377049180327,
 0.6721311475409836,
 0.6885245901639344,
 0.7213114754098361,
 0.7049180327868853,
 0.6885245901639344,
 0.6885245901639344,
 0.7049180327868853,
 0.7540983606557377,
 0.7377049180327869,
 0.7377049180327869,
 0.7377049180327869,
 0.6885245901639344,
 0.7213114754098361,
 0.6885245901639344,
 0.6885245901639344,
 0.7049180327868853,
 0.6557377049180327]
```

In [26]:

```
plt.plot(neighbors, train_scores, label="Train Score")
plt.plot(neighbors, test_scores, label="Test Score")
plt.xlabel("No of neighbors")
plt.ylabel("Model score")
plt.legend()

print(f" Maximum KNN score on the test data:{max(test_scores)*100:.2f}%")
```

```
 Maximum KNN score on the test data:75.41%
```



Hyperparameter Tuning with RandomizedSearchCV

We're going to tune:

- Logistic Regression
- RandomForestClassifier

....using RandomizedSearchCV

In [27]:
```
#Create a hyperparameter grid for LogisticRegression

log_reg_grid = {"C":np.logspace(-4,4,20),
                "solver":["liblinear"]}

#Create a hyperparameter grid for RandomForestClassifier
rg_grid = {"n_estimators":np.arange(10,1000,50),
           "max_depth":[None,3,5,10],
           "min_samples_split":np.arange(2,20,2),
           "min_samples_leaf":np.arange(2,20,2)}
```

In [28]:
```
#Tune logistic Regression
np.random.seed(42)
#Set up random hyperparameter search for LogisticRegression
rs_log_reg= RandomizedSearchCV(LogisticRegression(),
                              param_distributions=log_reg_grid,
                              cv=5,
                              n_iter=20,
                              verbose=True)

#Fit random hyperparameter search model for logistic regression
rs_log_reg.fit(x_train,y_train)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
```

Out[28]:

```
▸ RandomizedSearchCV
▸ estimator: LogisticRegression
  ▸ LogisticRegression
```

In [29]:
```
rs_log_reg.best_params_
```

Out[29]:
```
{'solver': 'liblinear', 'C': 0.23357214690901212}
```

In [30]:
```
#Evaluate Randomized search Logistic Regression model
rs_log_reg.score(x_test,y_test)
```

Out[30]:
```
0.8852459016393442
```

Now we've tuned LogisticRegression(), let's do the same for RandomForestClassifier()

In [31]:
```
#Setup random seed
np.random.seed(42)

#Setup hyperparameter search for RandomForestClassifier
rs_rf = RandomizedSearchCV(RandomForestClassifier(),
                          param_distributions=rg_grid,
```

```
                          cv=5,
                          n_iter=20,
                          verbose=True)

#Fit Random hyperparamter search model for RandomForestClassifier()
rs_rf.fit(x_train,y_train)
```

Fitting 5 folds for each of 20 candidates, totalling 100 fits

Out[31]:

```
▸ RandomizedSearchCV
▸ estimator: RandomForestClassifier
    ▸ RandomForestClassifier
```

In [32]:

```
#Find the best parameters
rs_rf.best_params_
```

Out[32]:

```
{'n_estimators': 560,
 'min_samples_split': 12,
 'min_samples_leaf': 18,
 'max_depth': 3}
```

In [97]:

```
#Evaluate Randomized search Logistic Regression model
rs_rf.score(x_test, y_test)
```

Out[97]:

```
0.8688524590163934
```

Hyperparameter tuning by GridSearchCV()

Since our GridSearchCV provides the best result so far, we're goint to and improve them again using GridSearchCV

In [34]:

```
#Different Hyperparameter tuning for Our LogisticRegression Model()


log_reg_grid = {"C":np.logspace(-4,4,30),
               "solver":["liblinear"]}

np.random.seed(42)
#Set up random hyperparameter search for LogisticRegression
gs_log_reg= GridSearchCV(LogisticRegression(),
                              param_grid=log_reg_grid,
                              cv=5,
                              verbose=True)

#Fit grid hyperparameter search model
gs_log_reg.fit(x_train, y_train);
```

Fitting 5 folds for each of 30 candidates, totalling 150 fits

In [35]:

```
#Check the best hyperparameter
gs_log_reg.best_params_
```

Out[35]:

```
{'C': 0.20433597178569418, 'solver': 'liblinear'}
```

In [36]:

```
#Evaluate the grid search Logistic Regresssion model
gs_log_reg.score(x_test, y_test)
```

Out[36]:

```
0.8852459016393442
```

Evaluating Our Tuned Machine Learning Classifier, Beyond Accuracy

- Ruc Curve And AUC
- Confusion Matrix
- Classification Report
- Precision
- Recall
- F1 Score

.... It would be great if cross validation is used were possible

To make comparisons and evaluate our trained model, first we need to make predictions

In [37]:

```
#Make predictions with tuned model
y_preds= gs_log_reg.predict(x_test)
y_preds
```

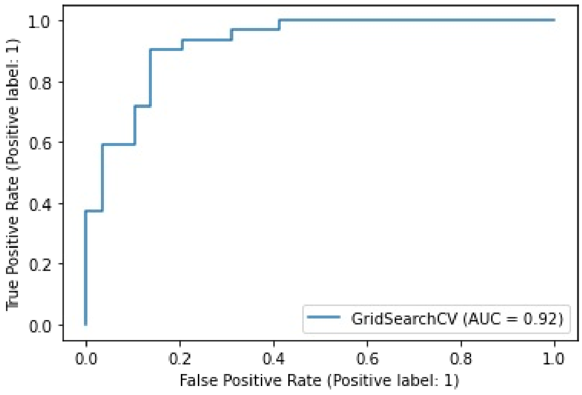
Out[37]:

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

In [38]:

```
#Plot ROC and Calculate AUC metrics
plot_roc_curve(gs_log_reg, x_test, y_test);
```

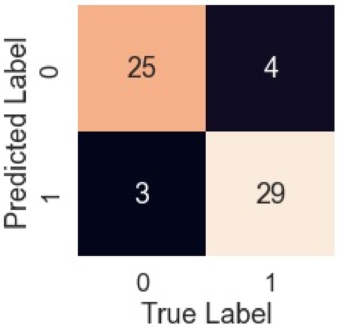
```
C:\Users\Aboya\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_roc_cu
rve is deprecated; Function :func:`plot_roc_curve` is deprecated in 1.0 and will be removed in 1.2. Use one of
the class methods: :meth:`sklearn.metrics.RocCurveDisplay.from_predictions` or :meth:`sklearn.metrics.RocCurveD
isplay.from_estimator`.
  warnings.warn(msg, category=FutureWarning)
```



In [39]:
```
#Confusion Matrix
print(confusion_matrix(y_test,y_preds))
```

```
[[25  4]
 [ 3 29]]
```

In [40]:
```
sns.set(font_scale=1.5)

def plot_conf_mat(y_test,y_preds):
    """
    Plots a nice confusion_matrix using seaborn's heatmap()
    """
    fig,ax= plt.subplots(figsize=(3,3))
    ax= sns.heatmap(confusion_matrix(y_test,y_preds),
                   annot= True,
                   cbar= False)
    plt.xlabel("True Label")
    plt.ylabel("Predicted Label")


plot_conf_mat(y_test,y_preds)
```



We've gotten Confusion_Matrix ROC Cureve and AUC, now Let's get Classification Report as well as Cross Validated precision,Recall and F1 Score

In [41]:
```
print(classification_report(y_test,y_preds))
```

```
              precision    recall  f1-score   support

           0       0.89      0.86      0.88        29
           1       0.88      0.91      0.89        32

    accuracy                           0.89        61
   macro avg       0.89      0.88      0.88        61
weighted avg       0.89      0.89      0.89        61
```

We're going to calculate Accuracy, Precision, Recall and F1 using cross_val_score()

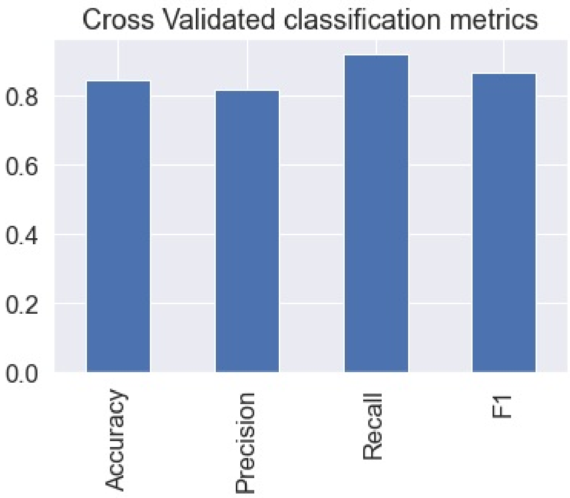
In [42]:
```
#Check Best params
gs_log_reg.best_params_
```

Out[42]:
```
{'C': 0.20433597178569418, 'solver': 'liblinear'}
```

In [43]:
```
#Create a new classifier with best params

clf= LogisticRegression(C=0.20433597178569418,
                        solver='liblinear')
```

In [44]:
```
#Cross validated Accuracy
```

In [44]:
```
#Cross_validated Accuracy
cv_acc= cross_val_score(clf,
                        x,
                        y,
                        cv=5,
                        scoring="accuracy")
cv_acc
```
Out[44]:
```
array([0.81967213, 0.90163934, 0.86885246, 0.88333333, 0.75      ])
```
In [45]:
```
cv_acc= np.mean(cv_acc)
cv_acc
```
Out[45]:
```
0.8446994535519124
```
In [46]:
```
#Cross_validated Precision
cv_precision= cross_val_score(clf,
                        x,
                        y,
                        cv=5,
                        scoring="precision")
cv_precision
```
Out[46]:
```
array([0.775     , 0.88571429, 0.85714286, 0.86111111, 0.725     ])
```
In [47]:
```
cv_precision= np.mean(cv_precision)
cv_precision
```
Out[47]:
```
0.8207936507936507
```
In [48]:
```
#Cross_validated Recall
cv_Recall= cross_val_score(clf,
                        x,
                        y,
                        cv=5,
                        scoring="recall")
cv_Recall
```
Out[48]:
```
array([0.93939394, 0.93939394, 0.90909091, 0.93939394, 0.87878788])
```
In [49]:
```
cv_Recall= np.mean(cv_Recall)
cv_Recall
```
Out[49]:
```
0.9212121212121213
```
In [50]:
```
#Cross_validated F1-score
cv_F1= cross_val_score(clf,
                        x,
                        y,
                        cv=5,
                        scoring="f1")
cv_F1
```
Out[50]:
```
array([0.84931507, 0.91176471, 0.88235294, 0.89855072, 0.79452055])
```
In [51]:
```
cv_F1= np.mean(cv_F1)
cv_F1
```
Out[51]:
```
0.8673007976269721
```
In [54]:
```
#Visualize Cross_validated Metrics

cv_metrics = pd.DataFrame({"Accuracy":cv_acc,"Precision":cv_precision,"Recall":cv_Recall,"F1":cv_F1},index=[0])


cv_metrics.T.plot.bar(title="Cross Validated classification metrics",legend=False)
```
Out[54]:
```
<AxesSubplot:title={'center':'Cross Validated classification metrics'}>
```



Feature importance

Feature importance is another way of asking 'which feature contributed most to the outcome of our model and how did they contribute?' Finding feature importance is different for each machine learning model. One way to find feature importance is to search for (NODEL NAME) feature importance'.

Let's find the feature importance for LogisticRegression()

In [55]:
```
#Fitan instance on Logistic Regression
clf= LogisticRegression(C=0.20433597178569418,
                        solver='liblinear')

clf.fit(x_train, y_train)
```

Out[55]:
```
▾                    LogisticRegression
LogisticRegression(C=0.20433597178569418, solver='liblinear')
```

In [56]:
```
#Check Coefficient
clf.coef_
```

Out[56]:
```
array([[ 0.00320769, -0.86062049,  0.66001432, -0.01155971, -0.00166496,
         0.04017236,  0.31603405,  0.02458922, -0.60470171, -0.56795456,
         0.45085392, -0.63733328, -0.67555094]])
```

In [58]:
```
# Match coef's of features to columns
feature_dict= dict(zip(df.columns, list(clf.coef_[0])))
feature_dict
```

Out[58]:
```
{'age': 0.0032076883508599633,
 'sex': -0.8606204883695241,
 'cp': 0.660014324982524,
 'trestbps': -0.011559706005500047,
 'chol': -0.0016649614843449207,
 'fbs': 0.040172360271308105,
 'restecg': 0.31603405294617176,
 'thalach': 0.02458922341328129,
 'exang': -0.604701713592625,
 'oldpeak': -0.5679545646616215,
 'slope': 0.4508539209693025,
 'ca': -0.6373332766360461,
 'thal': -0.6755509369619848}
```

In [60]:
```
#Visualize feature importance
feature_df= pd.DataFrame(feature_dict, index=[0])
feature_df.T.plot.bar(title="Feature Importance", legend=False);
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js