Predicting heart disease using machine learning

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

We're going to take the following approach:

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

1. Problem Definition

In a statement,

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

2. Data

The original data came from the Cleavland data from UCI Machine Learning Repository. https://archive.ics.uci.edu/ml/datasets/heart+disease (https://archive.ics.uci.edu/ml/datasets/heart+disease)

There is also a version of it available on Kaggle. https://www.kaggle.com/ronitf/heart-disease-uci (https://www.kaggle.com/ronitf/heart-disease-uci)

3. Evaluation

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

4. Features

This is where you'll get different information about each of the features in your data.

Data dictionary

- age. The age of the patient.
- sex. The gender of the patient. (1 = male, 0 = female).
- cp. Type of chest pain. (1 = typical angina, 2 = atypical angina, 3 = non — anginal pain, 4 = asymptotic).
- trestbps. Resting blood pressure in mmHg.
- chol. Serum Cholestero in mg/dl.
- fbs. Fasting Blood Sugar. (1 = fasting blood sugar is more than 120mg/dl, 0 = otherwise).
- restecg. Resting ElectroCardioGraphic results (0 = normal, 1 = ST-T wave abnormality, 2 = left ventricular hyperthrophy).
- thalach. Max heart rate achieved.
- exang. Exercise induced angina (1 = yes, 0 = no).
- oldpeak. ST depression induced by exercise relative to rest.
- slope. Peak exercise ST segment (1 = upsloping, 2 = flat, 3 = downsloping).
- ca. Number of major vessels (0–3) colored by flourosopy.
- thal. Thalassemia (3 = normal, 6 = fixed defect, 7 = reversible defect).
- num. Diagnosis of heart disease (0 = absence, 1, 2, 3, 4 = present).

Preparing the tools

We're going to use Pandas, Matplotlib and Numpy for data analysis and manipulation.

In [125]:

```python
# Import all the tools we need

# Regular EDA(exploratory data analysis) and plotting libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

# We want our plots to appear inside the notebook
%matplotlib inline

# Models from Scikit-Learn
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier

# Model evaluation
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
from sklearn.metrics import plot_roc_curve, roc_curve, auc
```

executed in 24ms, finished 09-07-2021 22:58:56

Load data

In [2]:

```python
df = pd.read_csv("zero-to-mastery-ml-master/data/heart-disease.csv")
df.shape # (rows, columns)
```

executed in 35ms, finished 09-07-2021 19:38:26

Out[2]:

```
(303, 14)
```

Data Exploration (exploratory data analysis or EDA)

The goal here is to find out more about the data and become a subject matter expert on the dataset you're working with.

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

In [3]:

```python
df.head()
```

executed in 53ms, finished 09-07-2021 19:38:26

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]:

```python
df.tail()
```

executed in 47ms, finished 09-07-2021 19:38:26

Out[4]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

In [5]:

```python
# Let's find out how many of each class there
df["target"].value_counts()
```

executed in 42ms, finished 09-07-2021 19:38:26

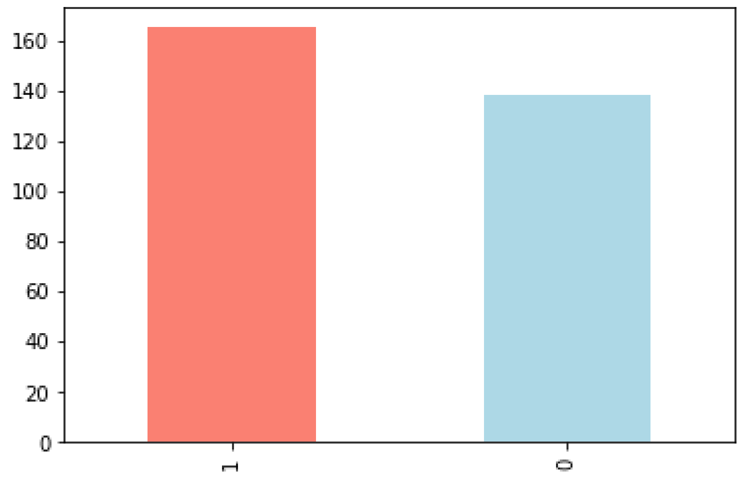
Out[5]:

```
1    165
0    138
Name: target, dtype: int64
```

In [6]:

```python
# Plot using Matplotlib
df["target"].value_counts().plot(kind="bar",color=["salmon", "lightblue"]);
```

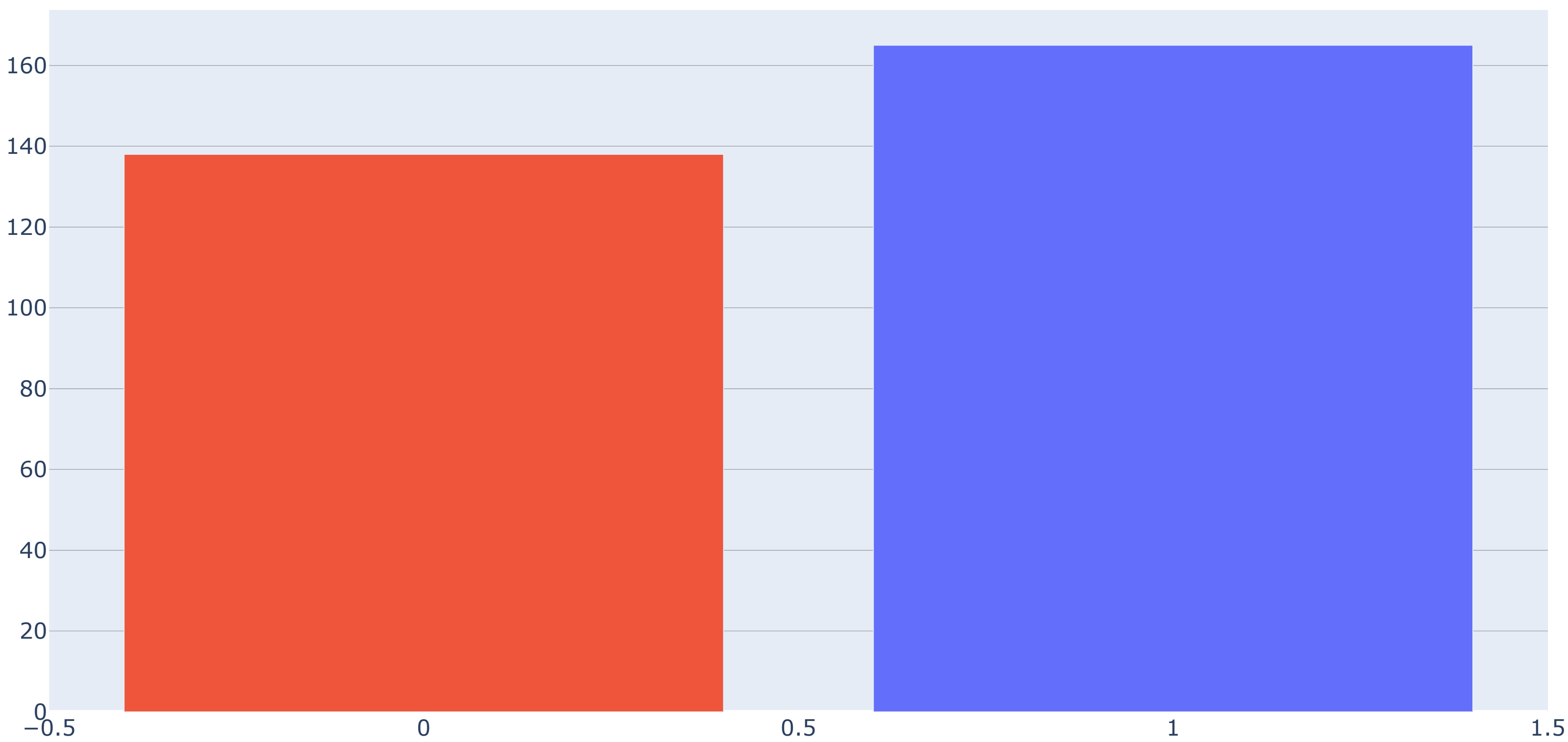
executed in 491ms, finished 09-07-2021 19:38:26



In [130]:

```python
# Plot using Plotly
fig = px.bar(df["target"].value_counts(),color=["1","0"])
fig.update_xaxes(title_text=" ")
fig.update_yaxes(title_text=" ")
fig.show()
```

executed in 148ms, finished 09-07-2021 23:04:06



In [8]:

```python
df.info()
```

executed in 32ms, finished 09-07-2021 19:38:28

```
<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 [9]:
```
# Are there any missing values?
df.isna().sum()
```
executed in 33ms, finished 09-07-2021 19:38:29

Out[9]:
```
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 [10]:
```
df.describe()
```
executed in 111ms, finished 09-07-2021 19:38:29

Out[10]:

	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	303.00
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.399340	0.72
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226	1.02
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.00
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000	0.00
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.000000	0.00
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000	1.00
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.00

Heart Disease Frequeny according to Sex

In [11]:
```
df.sex.value_counts()
```
executed in 16ms, finished 09-07-2021 19:38:29

Out[11]:
```
1    207
0     96
Name: sex, dtype: int64
```

In [12]:
```
# Compare target column with sex column
pd.crosstab(df.target,df.sex)
```
executed in 46ms, finished 09-07-2021 19:38:29

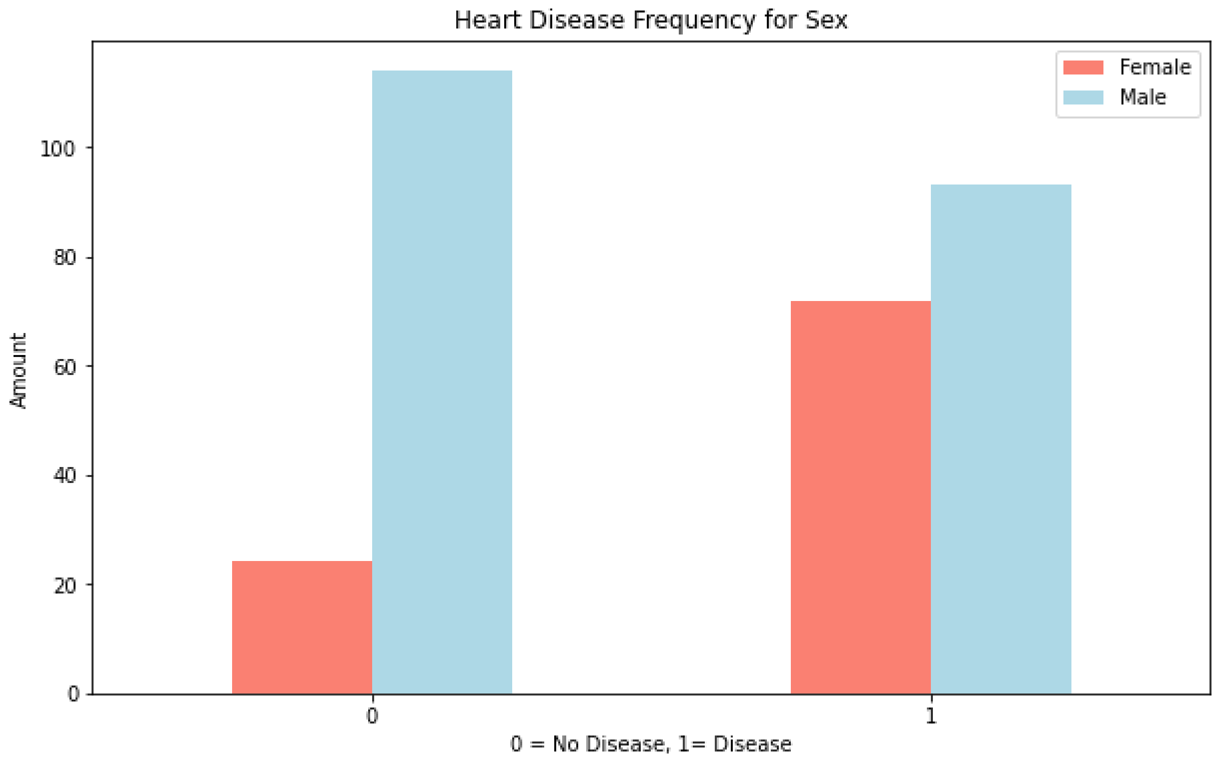
Out[12]:

sex	0	1
target		
0	24	114
1	72	93

In [13]:

```python
# Create a plot of crosstab using Matplotlib
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);
plt.show();
```

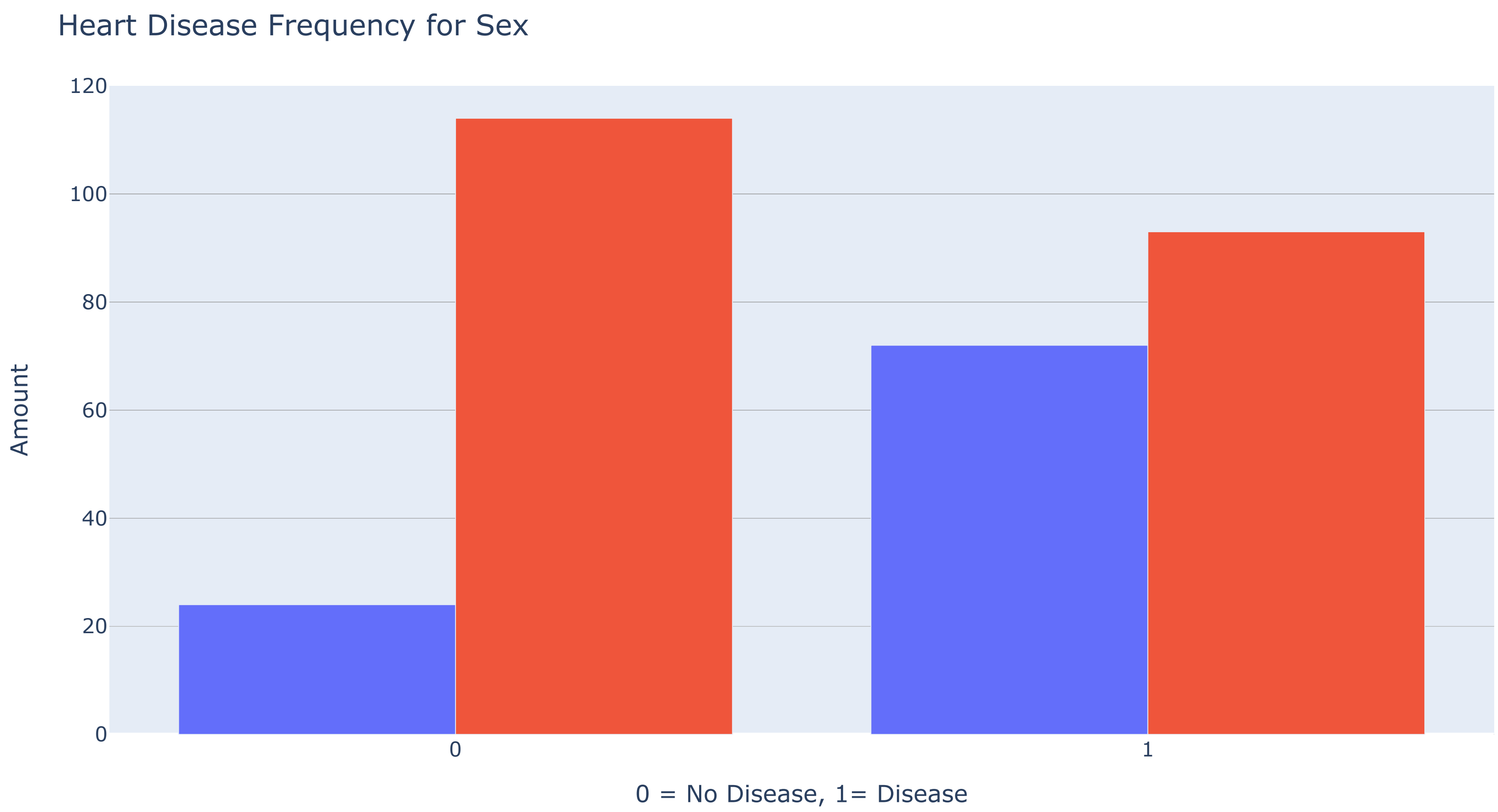
executed in 193ms, finished 09-07-2021 19:38:29



In [84]:

```python
# Create a plot of crosstab using Plotly
fig = px.bar(pd.crosstab(df.target,df.sex),barmode="group")
fig.update_xaxes(title_text="0 = No Disease, 1= Disease")
fig.update_yaxes(title_text="Amount")
fig.update_layout(title="Heart Disease Frequency for Sex")
fig.show()
```

executed in 101ms, finished 09-07-2021 20:33:20



Age vs. Max Heart Rate for Heart Disease

In [15]:

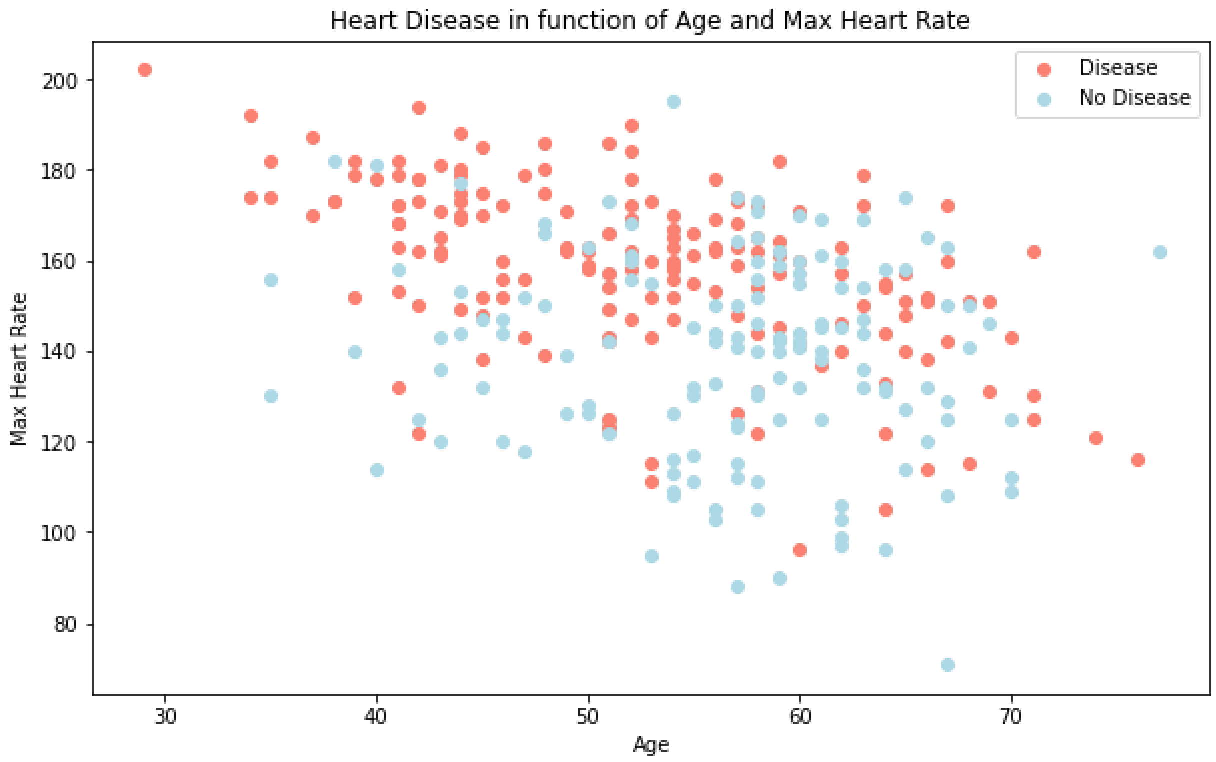
```
# Create another fig using Matplotlib
plt.figure(figsize=(10,6))

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

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

# Add some helpful info
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"]);
```

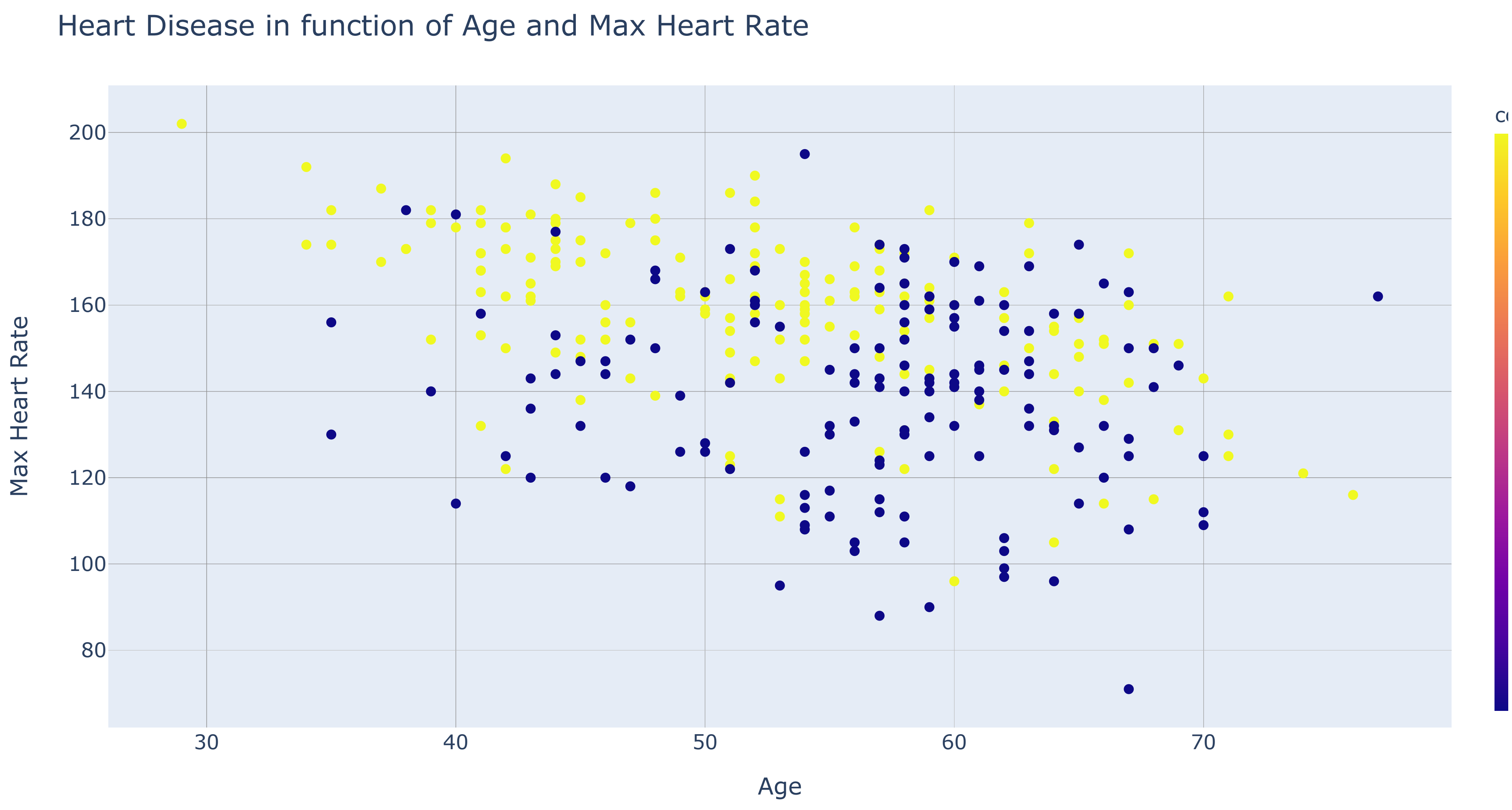
executed in 299ms, finished 09-07-2021 19:38:29



In [85]:

```
# Create another fig using Plotly
fig = px.scatter(x=df.age,y=df.thalach,color=df.target)
fig.update_xaxes(title_text="Age")
fig.update_yaxes(title_text="Max Heart Rate")
fig.update_layout(title="Heart Disease in function of Age and Max Heart Rate")
fig.show()
```

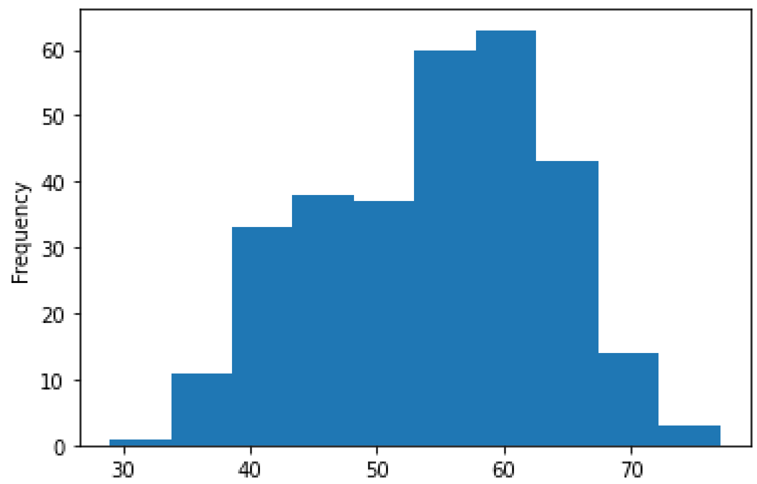
executed in 92ms, finished 09-07-2021 20:34:33



In [17]:

```
# Check the distribution of the age column with a histogram using Matplotlib
df.age.plot.hist();
```

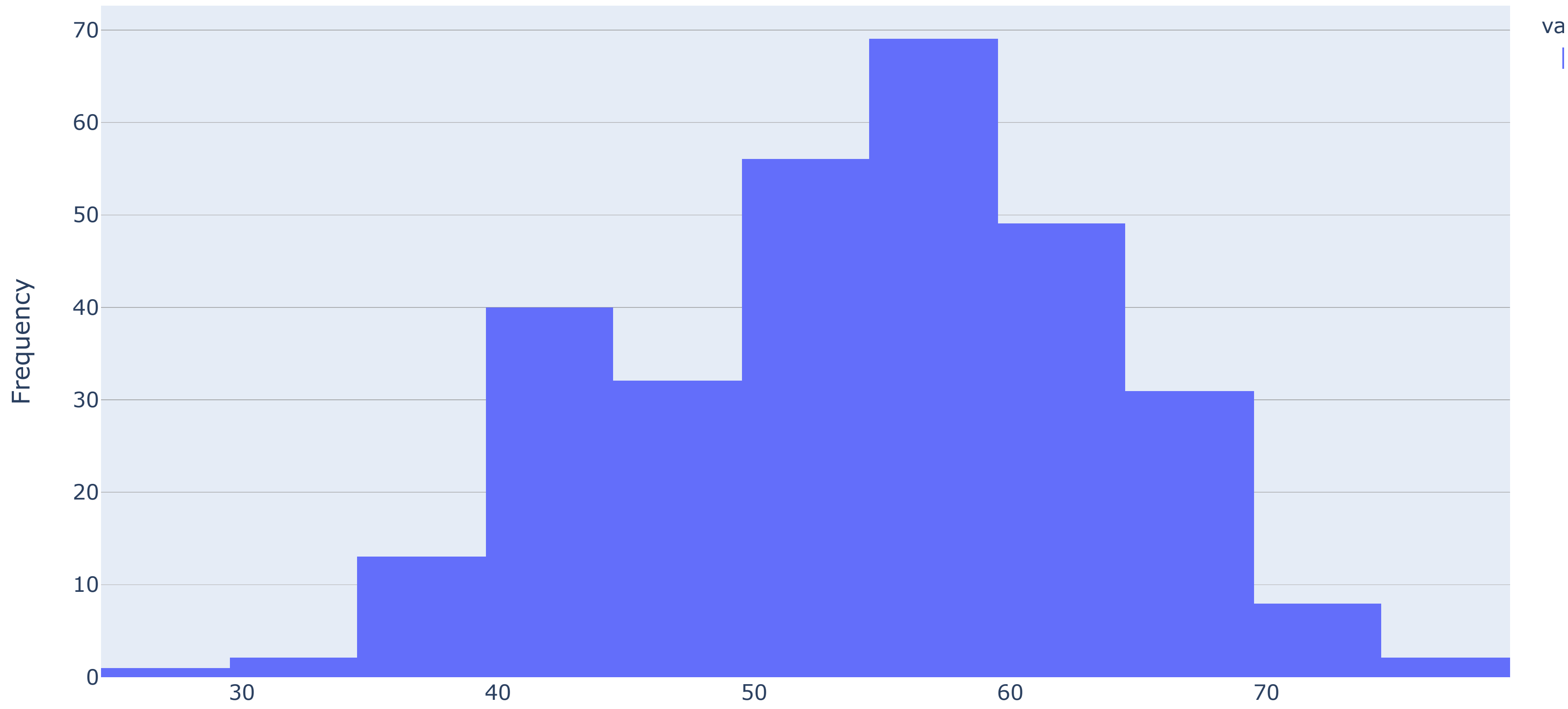
executed in 265ms, finished 09-07-2021 19:38:30



In [129]:

```
# Check the distribution of the age column with a histogram using Matplotlib
fig = px.histogram(df.age,nbins=10)
fig.update_xaxes(title_text=" ")
fig.update_yaxes(title_text="Frequency")
fig.show()
```

executed in 145ms, finished 09-07-2021 23:03:10



Heart Disease Frequency per Chest pain Type

cp. Type of chest pain.

- 0 = Typical angina
- 1 = Atypical angina
- 2 = Non-anginal pain
- 3 = Asymptomatic.

In [19]:

```
pd.crosstab(df.cp,df.target)
```

executed in 59ms, finished 09-07-2021 19:38:30

Out[19]:

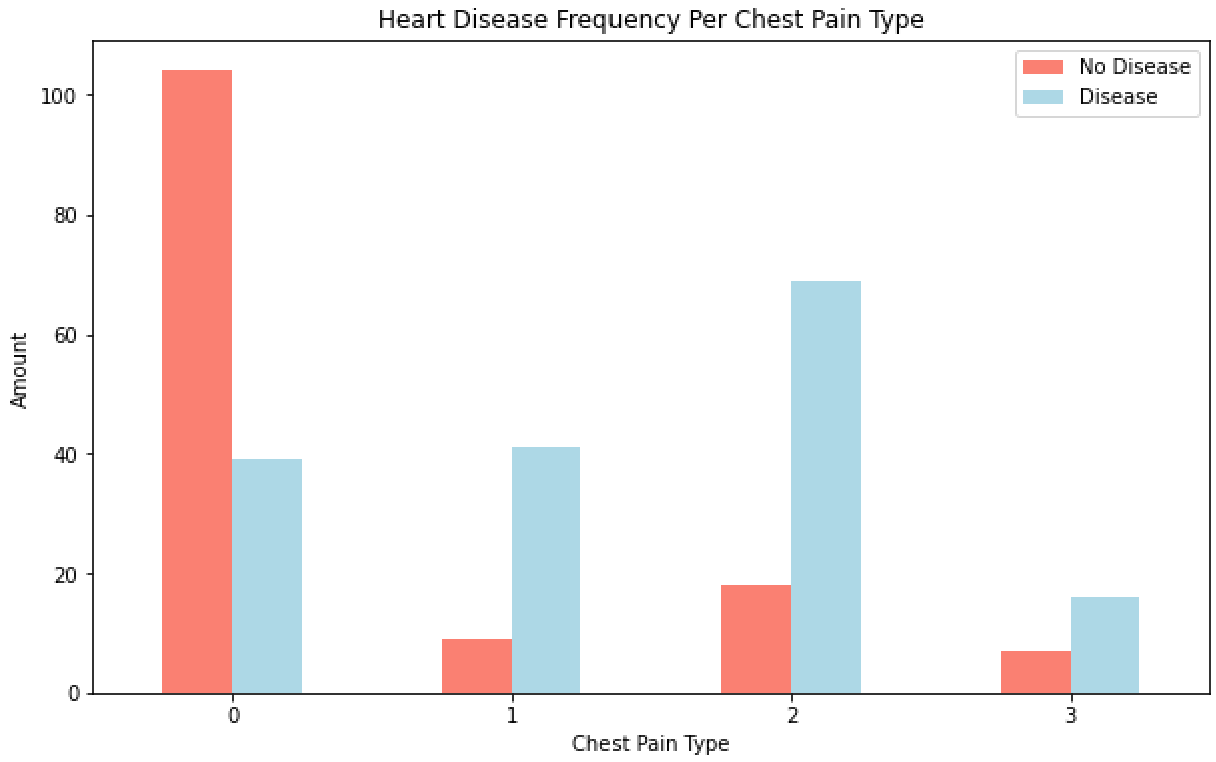
target	0	1
cp		
0	104	39
1	9	41
2	18	69
3	7	16

In [20]:

```
# Make the crosstab more visual using Matplotlib
pd.crosstab(df.cp,df.target).plot(kind="bar",
                                  figsize=(10,6),
                                  color=["salmon","lightblue"]);

# Add some communication
plt.title("Heart Disease Frequency Per Chest Pain Type");
plt.xlabel("Chest Pain Type");
plt.ylabel("Amount");
plt.legend(["No Disease","Disease"]);
plt.xticks(rotation=0);
```

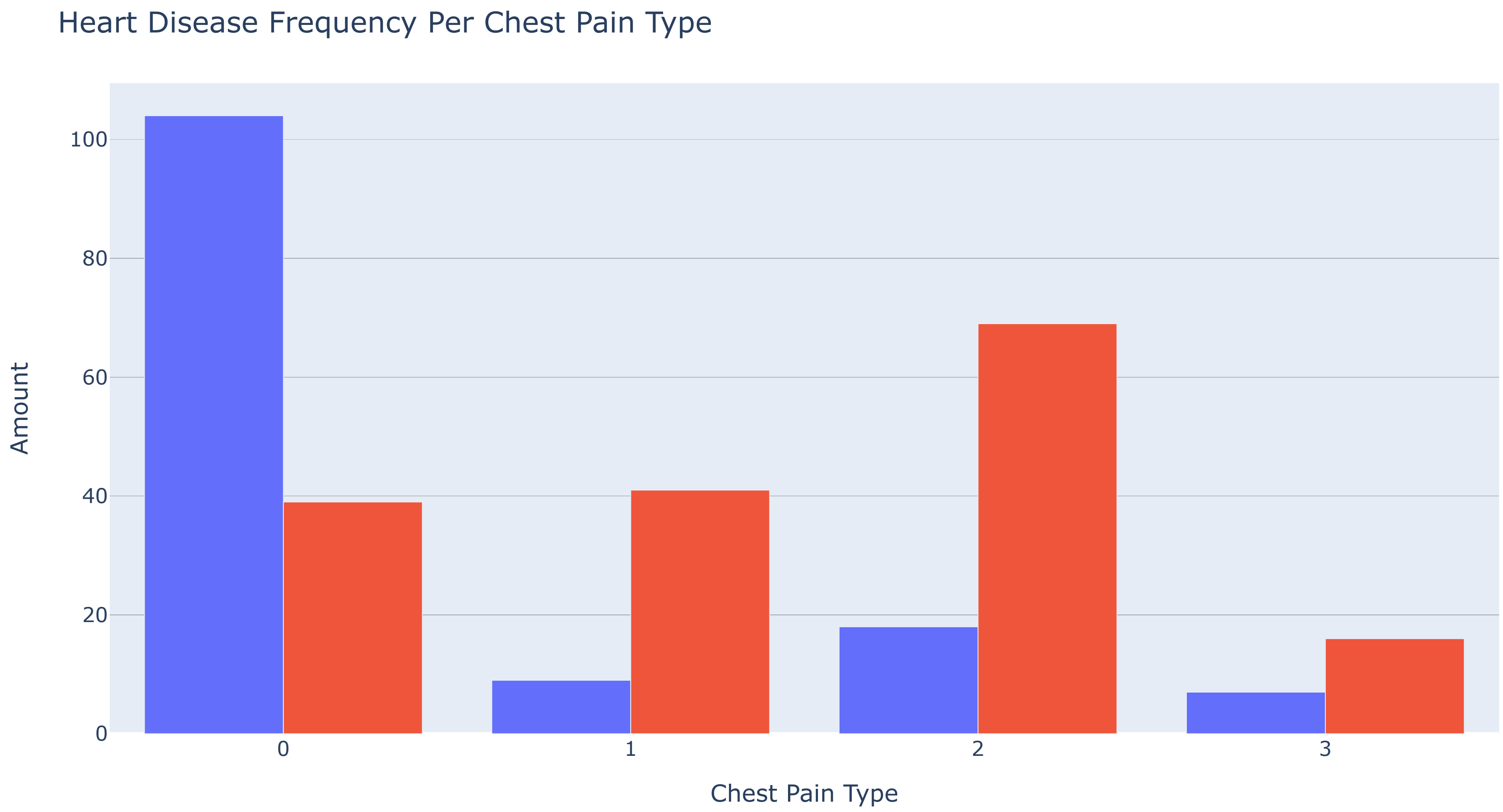
executed in 271ms, finished 09-07-2021 19:38:30



In [87]:

```
# Make the crosstab more visual using Plotly
fig = px.bar(pd.crosstab(df.cp,df.target),barmode="group")
fig.update_xaxes(title_text="Chest Pain Type")
fig.update_yaxes(title_text="Amount")
fig.update_layout(title="Heart Disease Frequency Per Chest Pain Type")
fig.show()
```

executed in 88ms, finished 09-07-2021 20:36:47



In [22]:

```
# Make a correlation matrix
df.corr()
```

executed in 60ms, finished 09-07-2021 19:38:31

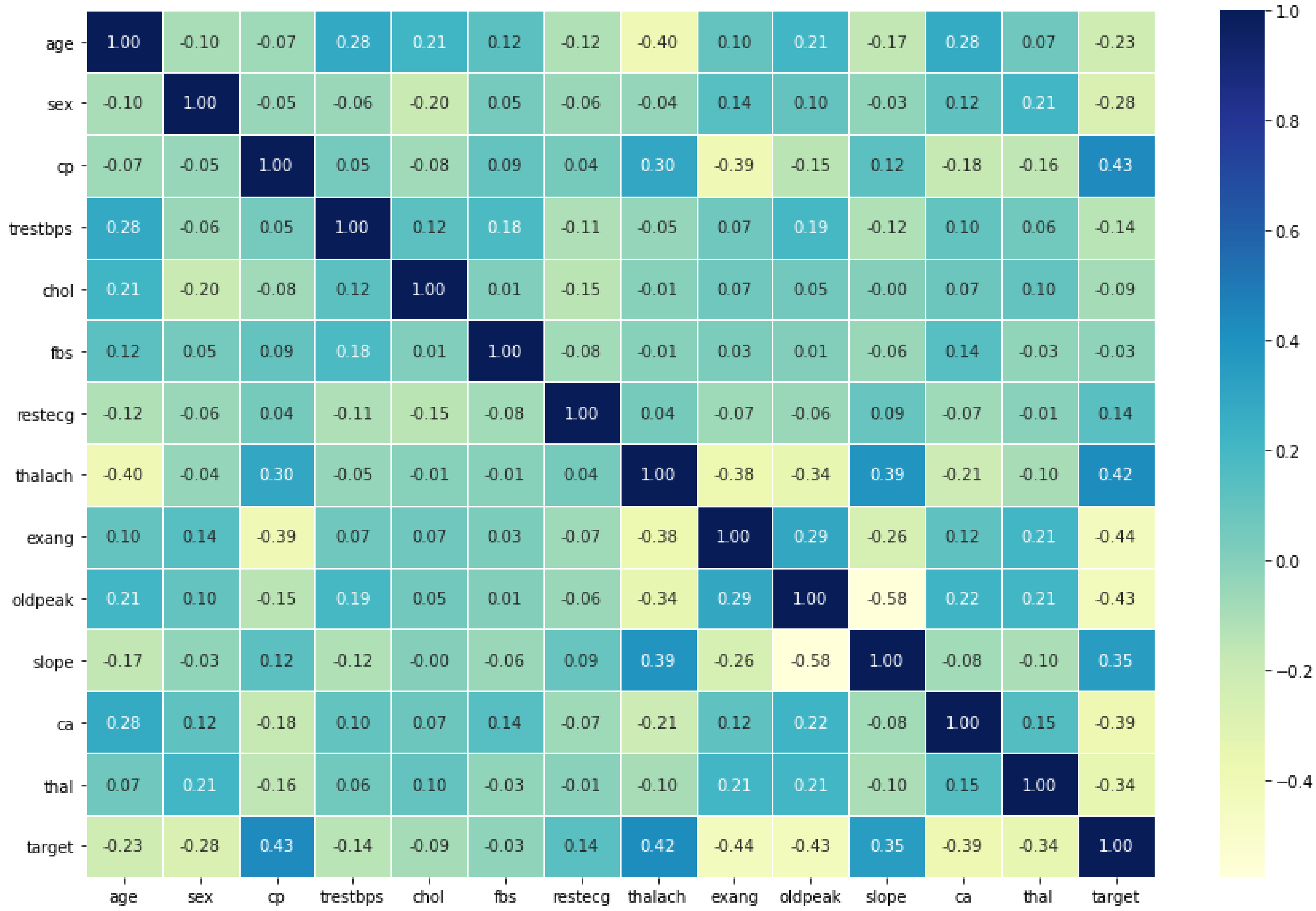
Out[22]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
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	0.068001
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	0.210041
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	-0.161736
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	0.062210
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	0.098803
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	-0.032019
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	-0.011981
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	-0.096439
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	0.206754
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	0.210244
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	-0.104764
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	0.151832
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	1.000000
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	-0.344029

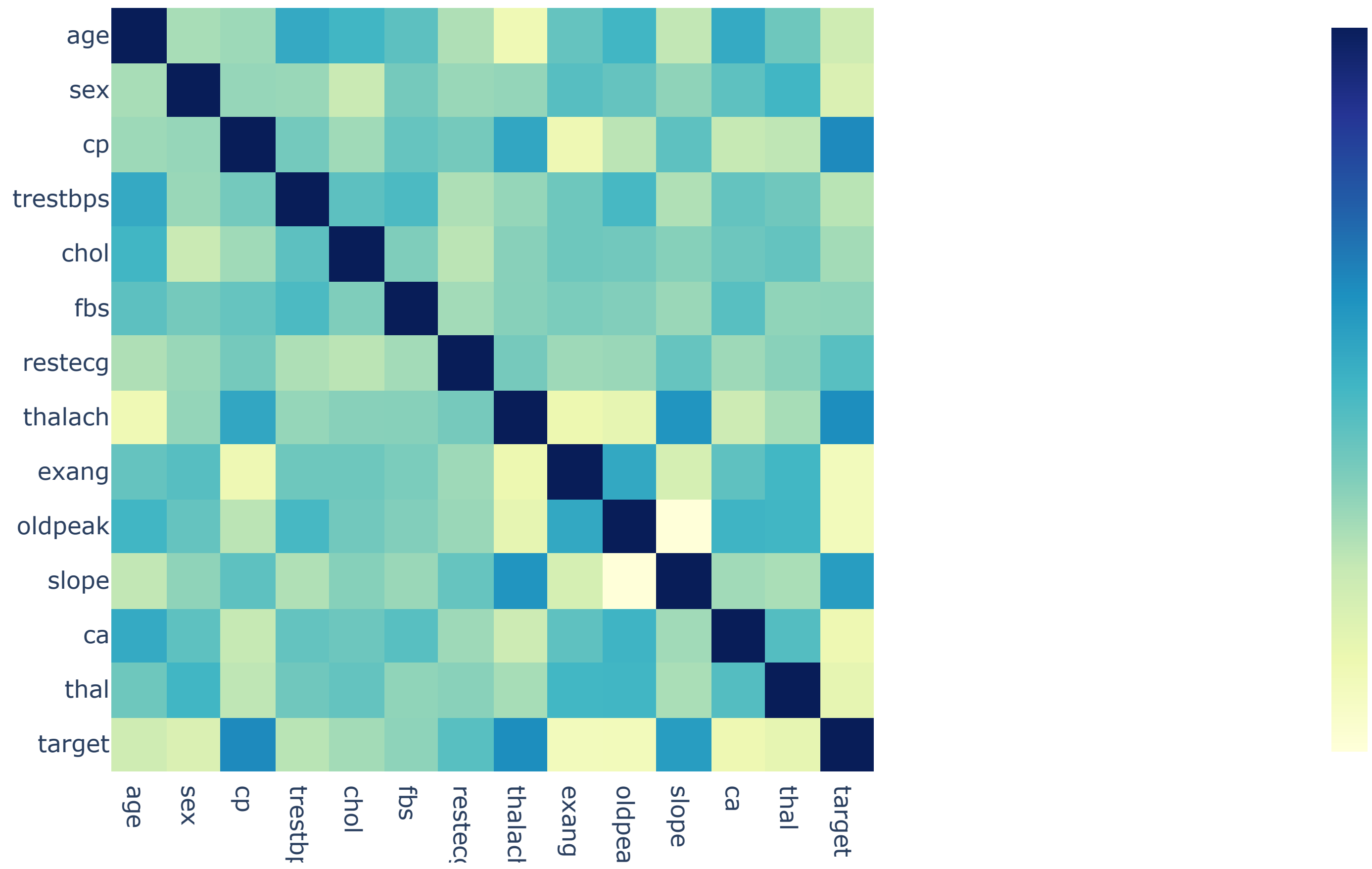
In [23]:

```
# Let's make our correlation matrix a little prettier using Matplotlib
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");
```

executed in 2.37s, finished 09-07-2021 19:38:33



In [24]:
```
fig = px.imshow(corr_matrix,color_continuous_scale="YlGnBu")
fig.show()
```
executed in 139ms, finished 09-07-2021 19:38:33



5. Modelling

In [25]:
```
df.head()
```
executed in 48ms, finished 09-07-2021 19:38:33

Out[25]:

	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 [26]:
```
# split data into X and y
X = df.drop("target",axis=1)
y = df["target"]
```
executed in 16ms, finished 09-07-2021 19:38:33

In [27]:
```
X
```
executed in 45ms, finished 09-07-2021 19:38:33

Out[27]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2

303 rows × 13 columns

In [28]:

```
y
```

executed in 31ms, finished 09-07-2021 19:38:33

Out[28]:

```
0      1
1      1
2      1
3      1
4      1
      ..
298    0
299    0
300    0
301    0
302    0
Name: target, Length: 303, dtype: int64
```

In [29]:

```
# Split data into train and test sets
np.random.seed(42)

# Split into train & test set
X_train, X_test, y_train, y_test = train_test_split(X,
                                                     y,
                                                     test_size=0.2)
```

executed in 23ms, finished 09-07-2021 19:38:33

In [30]:

```
X_train
```

executed in 52ms, finished 09-07-2021 19:38:33

Out[30]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
132	42	1	1	120	295	0	1	162	0	0.0	2	0	2
202	58	1	0	150	270	0	0	111	1	0.8	2	0	3
196	46	1	2	150	231	0	1	147	0	3.6	1	0	2
75	55	0	1	135	250	0	0	161	0	1.4	1	0	2
176	60	1	0	117	230	1	1	160	1	1.4	2	2	3
...	...	...	...	...	...	...	...	...	...	...	...	...	...
188	50	1	2	140	233	0	1	163	0	0.6	1	1	3
71	51	1	2	94	227	0	1	154	1	0.0	2	1	3
106	69	1	3	160	234	1	0	131	0	0.1	1	1	2
270	46	1	0	120	249	0	0	144	0	0.8	2	0	3
102	63	0	1	140	195	0	1	179	0	0.0	2	2	2

242 rows × 13 columns

In [31]:

```
y_train , len(y_train)
```

executed in 12ms, finished 09-07-2021 19:38:33

Out[31]:

```
(132    1
 202    0
 196    0
 75     1
 176    0
       ..
 188    0
 71     1
 106    1
 270    0
 102    1
 Name: target, Length: 242, dtype: int64,
 242)
```

Now we've got our data split into training and test sets, it's time to build a machine learning model.

We'll train it (find the patterns) on the training set.

And we'll test it (use the patterns) on the test set.

We're going to try 3 different machine learning models:

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

In [32]:

```
# 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 evaluates given machine learning models.
    models : a dict of different Scikit-Learn machine learning models
    X_train : training data (no labels)
    X_test : testing data (no labels)
    y_train : training labels
    y_test : test labels
    """
    # Set random seed
    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_scores
        model_scores[name] = model.score(X_test,y_test)
    return model_scores
```

executed in 27ms, finished 09-07-2021 19:38:33

In [33]:

```
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
```

executed in 441ms, finished 09-07-2021 19:38:34

```
C:\Users\Shreyash\miniconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: 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 (https://scikit-learn.org/stable/modules/preprocessing.h
tml)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modu
les/linear_model.html#logistic-regression)
```

Out[33]:

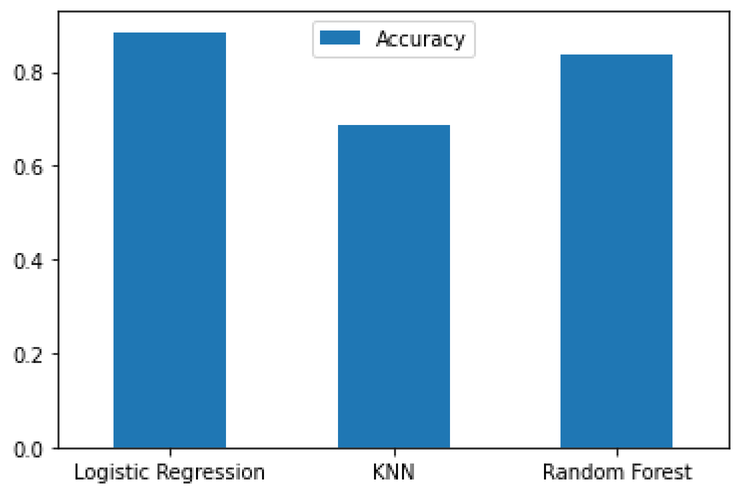
```
{'Logistic Regression': 0.8852459016393442,
 'KNN': 0.6885245901639344,
 'Random Forest': 0.8360655737704918}
```

Model Comparison

In [34]:

```
# Using Matplotlib
model_compare = pd.DataFrame(model_scores,index=["Accuracy"])
model_compare.T.plot.bar();
plt.xticks(rotation=0);
```

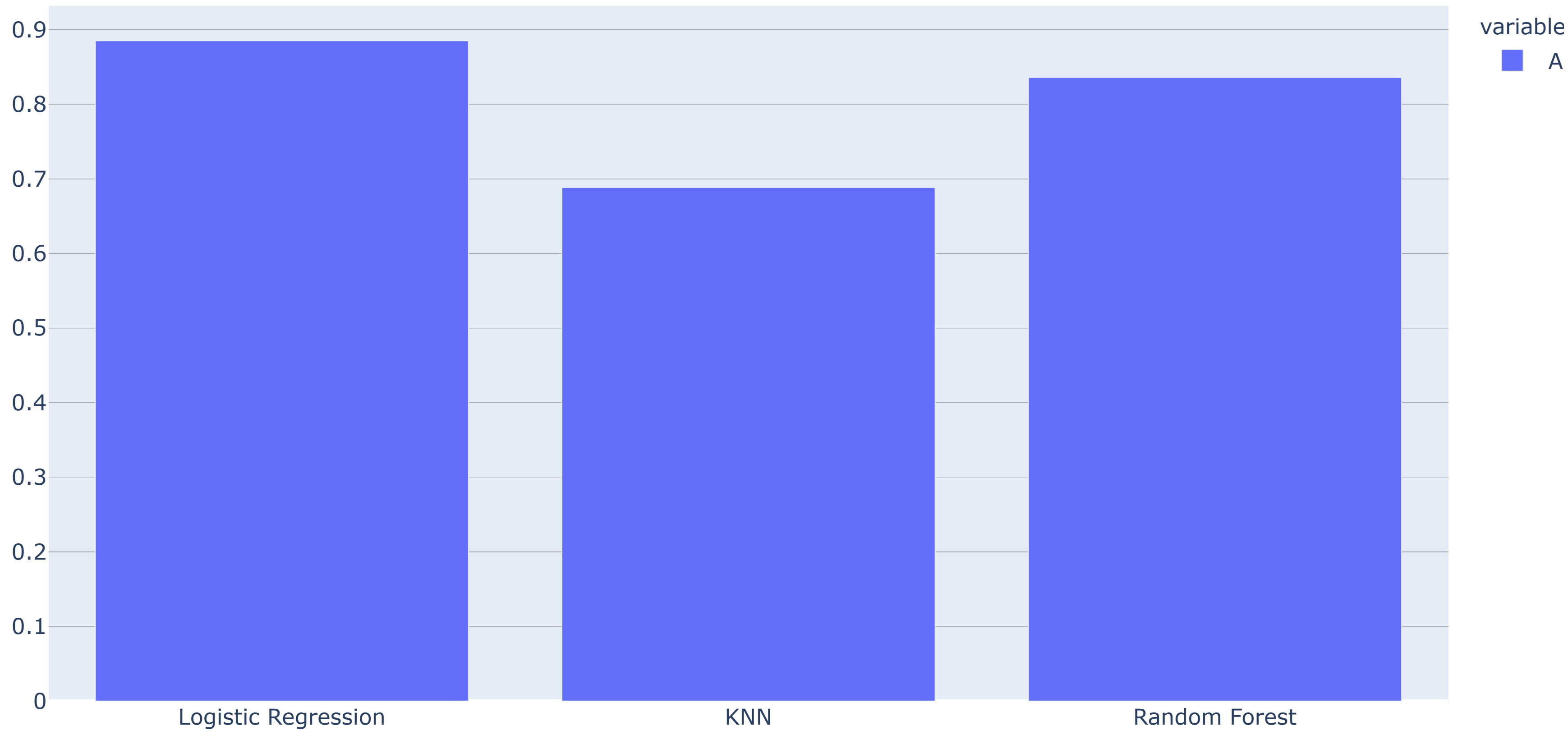
executed in 304ms, finished 09-07-2021 19:38:34



In [128]:

```
# Using Plotly
fig = px.bar(model_compare.T,barmode="group")
fig.update_xaxes(title_text = " ")
fig.update_yaxes(title_text = " ")
fig.show()
```

executed in 139ms, finished 09-07-2021 23:02:21



Now we've got a baseline model... and we know a model's first predictions aren't always what we should based 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 [36]:

```
# 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 scores list
    train_scores.append(knn.score(X_train,y_train))

    # Update the test scores list
    test_scores.append(knn.score(X_test,y_test))
```

executed in 774ms, finished 09-07-2021 19:38:35

In [37]:
```
train_scores
```
executed in 23ms, finished 09-07-2021 19:38:35

Out[37]:
```
[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]
```

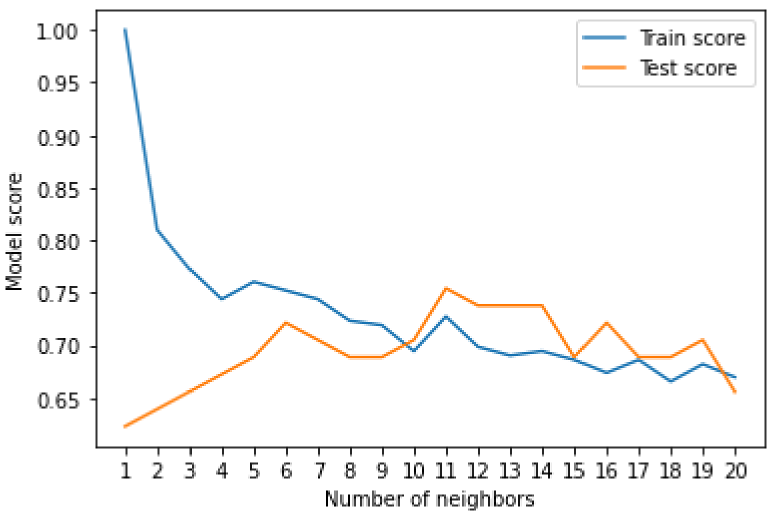
In [38]:
```
test_scores
```
executed in 12ms, finished 09-07-2021 19:38:35

Out[38]:
```
[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 [39]:
```
# Using Matplotlib
plt.plot(neighbors,train_scores,label="Train score")
plt.plot(neighbors,test_scores,label="Test score")
plt.xticks(np.arange(1,21,1))
plt.xlabel("Number of neighbors")
plt.ylabel("Model score")
plt.legend()

print(f"Maximum KNN score on the test data: {max(test_scores)*100:.2f}%")
```
executed in 590ms, finished 09-07-2021 19:38:36

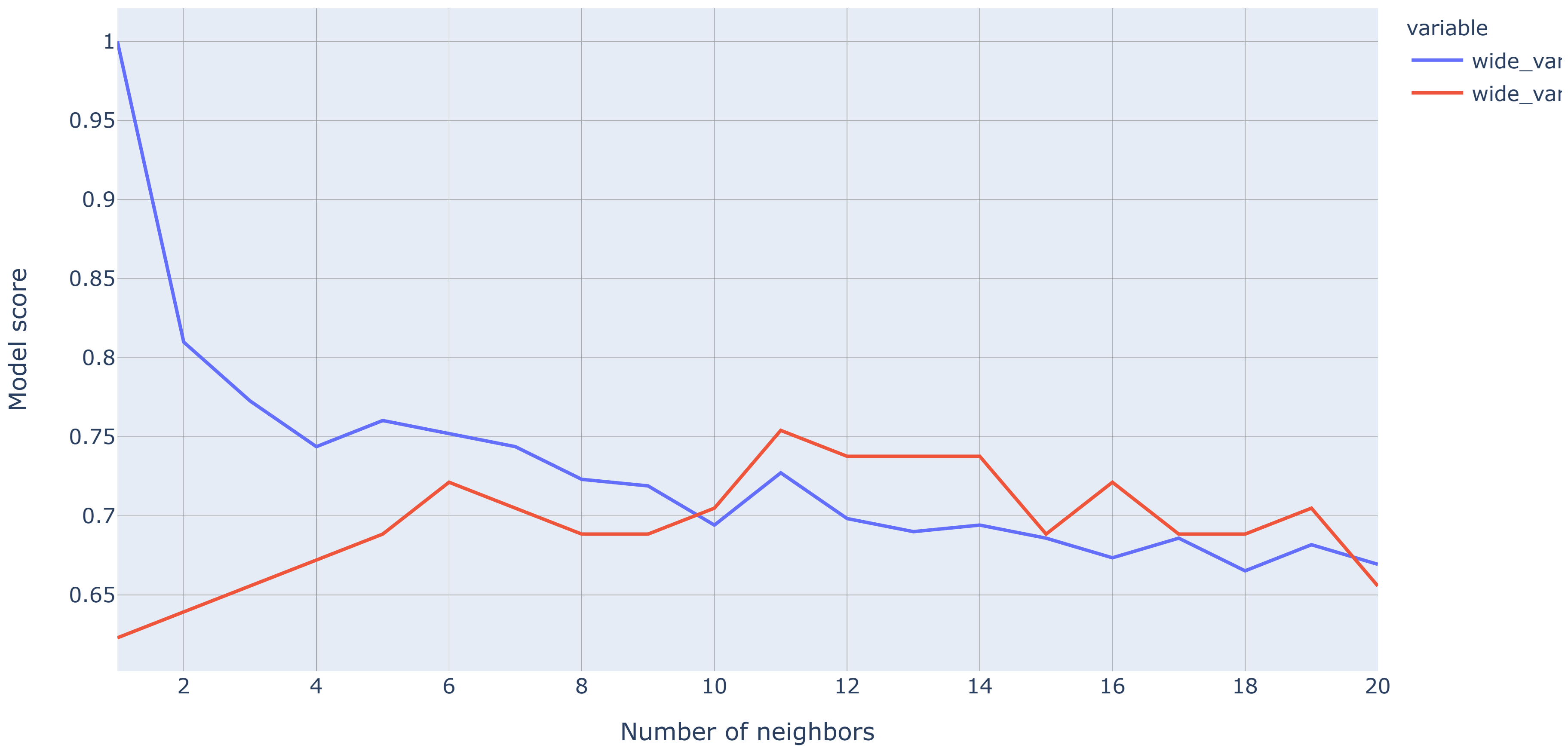
```
Maximum KNN score on the test data: 75.41%
```



In [75]:

```python
# Using Plotly
fig = px.line(x=neighbors,y=[train_scores,test_scores])
fig.update_yaxes(title_text="Model score")
fig.update_xaxes(title_text="Number of neighbors")
fig.show()
```

executed in 85ms, finished 09-07-2021 20:13:08



Hyperparameter tuning with RandomizedSearchCV

We're going to tune:

- LogisticRegression()
- RandomForestClassifier()

... using RandomizedSearchCV

In [41]:

```python
# Create a hyperparameter grid for LogisticRegression
log_reg_grid = {"C":np.logspace(-4,4,20),
                "solver":["liblinear"]}

# Create a hyperparameter grid for RandomForestClassifier
rf_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(1,20,2)}
```

executed in 15ms, finished 09-07-2021 19:38:36

Now we've got hyperparameter grids setup for each of our models,let's tune them using RandomizedSearchCV.

In [42]:
```python
# Tune LogisticRegression

np.random.seed(42)

# Setup rnadom 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 LogisticRegression
rs_log_reg.fit(X_train,y_train)
```
executed in 1.35s, finished 09-07-2021 19:38:37

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

Out[42]:
```
RandomizedSearchCV(cv=5, estimator=LogisticRegression(), n_iter=20,
                   param_distributions={'C': array([1.00000000e-04, 2.63665090e-04, 6.95192796e-04, 1.83298071e-03,
       4.83293024e-03, 1.27427499e-02, 3.35981829e-02, 8.85866790e-02,
       2.33572147e-01, 6.15848211e-01, 1.62377674e+00, 4.28133240e+00,
       1.12883789e+01, 2.97635144e+01, 7.84759970e+01, 2.06913808e+02,
       5.45559478e+02, 1.43844989e+03, 3.79269019e+03, 1.00000000e+04]),
                                        'solver': ['liblinear']},
                   verbose=True)
```

In [43]:
```python
rs_log_reg.best_params_
```
executed in 30ms, finished 09-07-2021 19:38:37

Out[43]:
```
{'solver': 'liblinear', 'C': 0.23357214690901212}
```

In [44]:
```python
rs_log_reg.score(X_test,y_test)
```
executed in 30ms, finished 09-07-2021 19:38:37

Out[44]:
```
0.8852459016393442
```

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

In [45]:
```python
# Setup random seed
np.random.seed(42)

# Setup random hyperparameter search for RandomForestClassifier
rs_rf = RandomizedSearchCV(RandomForestClassifier(),
                           param_distributions=rf_grid,
                           cv=5,
                           n_iter=20,
                           verbose=True)

# Fit random hyperparameter search model for RandomForestClassifier()
rs_rf.fit(X_train,y_train)
```
executed in 2m 21s, finished 09-07-2021 19:40:58

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

Out[45]:
```
RandomizedSearchCV(cv=5, estimator=RandomForestClassifier(), n_iter=20,
                   param_distributions={'max_depth': [None, 3, 5, 10],
                                        'min_samples_leaf': array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19]),
                                        'min_samples_split': array([ 2,  4,  6,  8, 10, 12, 14, 16, 18]),
                                        'n_estimators': array([ 10,  60, 110, 160, 210, 260, 310, 360, 410, 460, 510, 5
60, 610,
       660, 710, 760, 810, 860, 910, 960])},
                   verbose=True)
```

In [46]:
```python
# Find the best hyperparameters
rs_rf.best_params_
```
executed in 13ms, finished 09-07-2021 19:40:58

Out[46]:
```
{'n_estimators': 210,
 'min_samples_split': 4,
 'min_samples_leaf': 19,
 'max_depth': 3}
```

In [47]:
```python
# Evaluate the randomized search RandomForestClassifier model
rs_rf.score(X_test,y_test)
```
executed in 74ms, finished 09-07-2021 19:40:58

Out[47]:
```
0.8688524590163934
```

In [48]:
```
model_scores
```
executed in 22ms, finished 09-07-2021 19:40:58

Out[48]:
```
{'Logistic Regression': 0.8852459016393442,
 'KNN': 0.6885245901639344,
 'Random Forest': 0.8360655737704918}
```

Hyperparameter tuning with GridSearchCV

Since our LogisticRegression model provides the best scores so far, we'll try to improve them again using GridSearchCV...

In [49]:
```
# Different hyperparameters for our LogisticRegression model
log_reg_grid = {"C":np.logspace(-4,4,30),
                "solver":["liblinear"]}

# Setup grid 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)
```
executed in 1.74s, finished 09-07-2021 19:41:00

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

Out[49]:
```
GridSearchCV(cv=5, estimator=LogisticRegression(),
             param_grid={'C': array([1.00000000e-04, 1.88739182e-04, 3.56224789e-04, 6.72335754e-04,
       1.26896100e-03, 2.39502662e-03, 4.52035366e-03, 8.53167852e-03,
       1.61026203e-02, 3.03919538e-02, 5.73615251e-02, 1.08263673e-01,
       2.04335972e-01, 3.85662042e-01, 7.27895384e-01, 1.37382380e+00,
       2.59294380e+00, 4.89390092e+00, 9.23670857e+00, 1.74332882e+01,
       3.29034456e+01, 6.21016942e+01, 1.17210230e+02, 2.21221629e+02,
       4.17531894e+02, 7.88046282e+02, 1.48735211e+03, 2.80721620e+03,
       5.29831691e+03, 1.00000000e+04]),
                         'solver': ['liblinear']},
             verbose=True)
```

In [50]:
```
# Check the best hyperparameters
gs_log_reg.best_params_
```
executed in 13ms, finished 09-07-2021 19:41:00

Out[50]:
```
{'C': 0.20433597178569418, 'solver': 'liblinear'}
```

In [51]:
```
# Evaluate the grid search LogisticRegression model
gs_log_reg.score(X_test,y_test)
```
executed in 26ms, finished 09-07-2021 19:41:00

Out[51]:
```
0.8852459016393442
```

Evaluating our tuned machine learning classifier, beyond accuracy

- ROC curve and AUC score
- Confusion matrix
- Classification report
- Precision
- Recall
- F1-score

...and it would be great if cross-validation was used where possible.

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

In [52]:
```
# Make predictions with tuned model
y_preds = gs_log_reg.predict(X_test)
```
executed in 11ms, finished 09-07-2021 19:41:00

In [53]:
```
y_preds
```
executed in 33ms, finished 09-07-2021 19:41:00

Out[53]:
```
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 [54]:

```
y_test
```

executed in 31ms, finished 09-07-2021 19:41:00

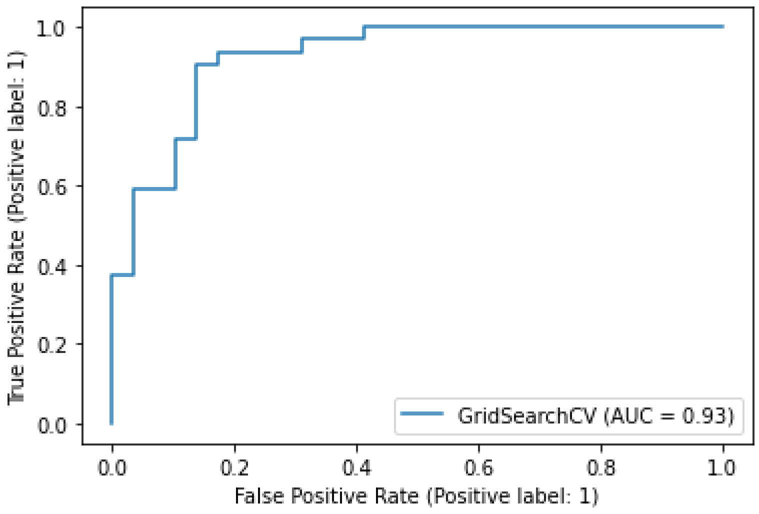
Out[54]:

```
179    0
228    0
111    1
246    0
60     1
      ..
249    0
104    1
300    0
193    0
184    0
Name: target, Length: 61, dtype: int64
```

In [55]:

```
# Plot ROC curve and calculate AUC metric using matplotlib
plot_roc_curve(gs_log_reg,X_test,y_test);
```

executed in 353ms, finished 09-07-2021 19:41:00



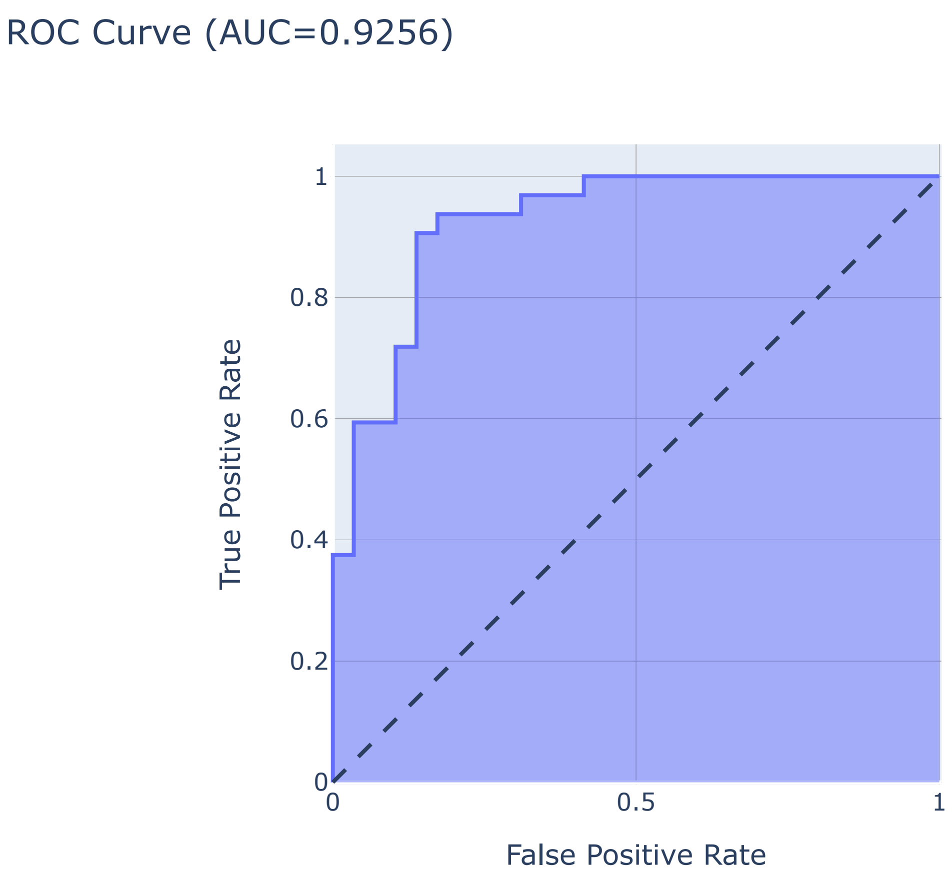
In [56]:

```
# Plot ROC curve and calculate AUC metric using Plotly
y_probs = gs_log_reg.predict_proba(X_test)
y_probs_positive = y_probs[:,1]
fpr,tpr,thresholds = roc_curve(y_test,y_probs_positive)
fig = px.area(x=fpr, y=tpr,
    title=f'ROC Curve (AUC={auc(fpr, tpr):.4f})',
    labels=dict(x='False Positive Rate', y='True Positive Rate'),
    width=700, height=500)

fig.add_shape(
    type='line', line=dict(dash='dash'),
    x0=0, x1=1, y0=0, y1=1
)

fig.update_yaxes(scaleanchor="x", scaleratio=1)
fig.update_xaxes(constrain='domain')
fig.show()
```

executed in 187ms, finished 09-07-2021 19:41:00



In [57]:

```python
# Confusion matrix
print(confusion_matrix(y_test,y_preds))
```

executed in 11ms, finished 09-07-2021 19:41:00

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

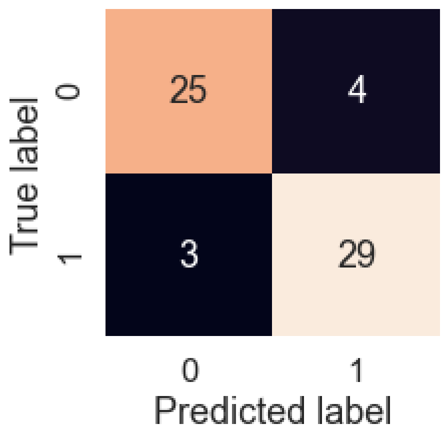
In [58]:

```python
# Using Matplotlib
sns.set(font_scale=1.5)

def plot_conf_mat(y_test,y_preds):
    """
    Plots a nice looking 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("Predicted label")
    plt.ylabel("True label")

plot_conf_mat(y_test,y_preds)
```

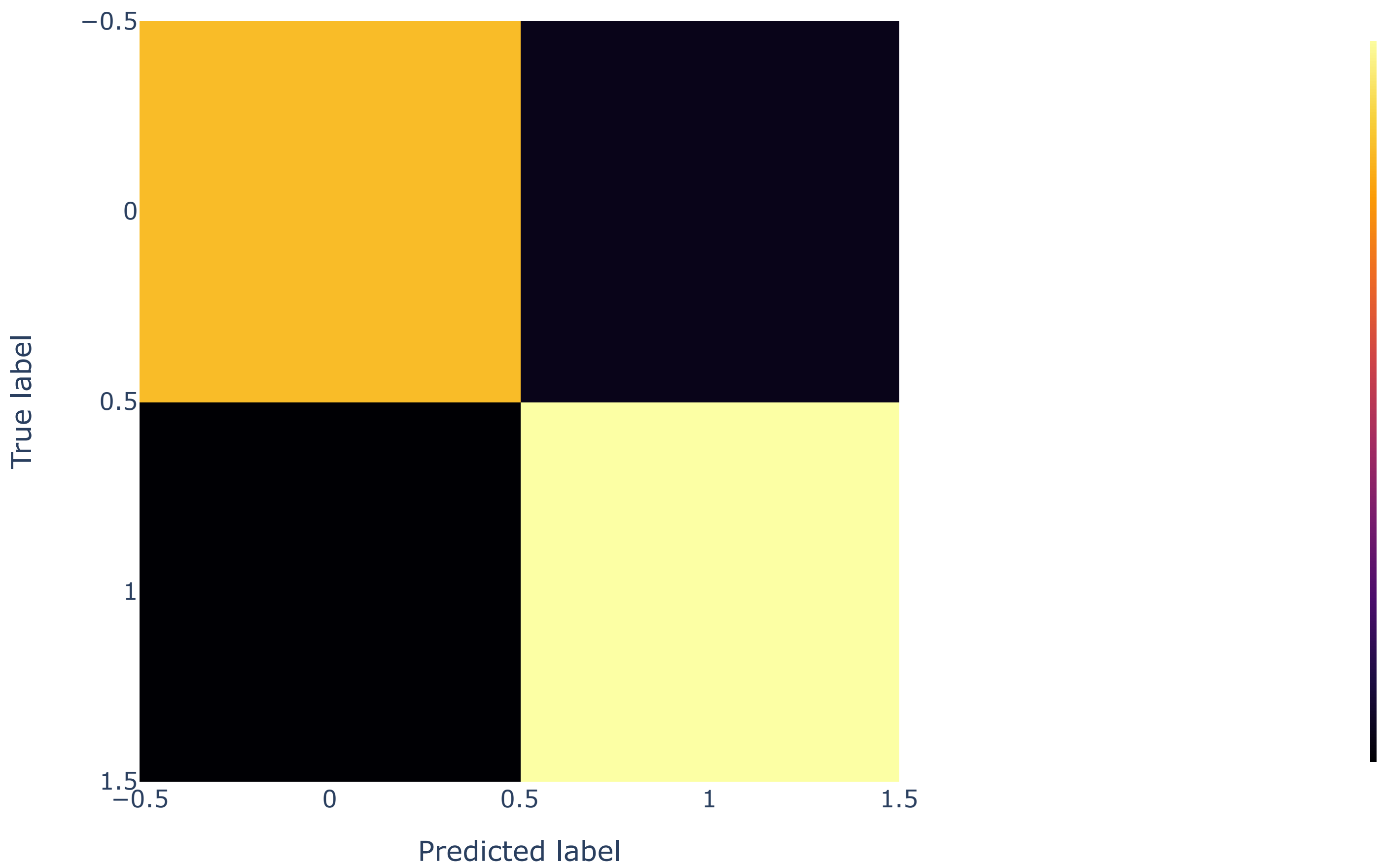
executed in 350ms, finished 09-07-2021 19:41:01



In [73]:

```python
# Using Plotly
fig = px.imshow(confusion_matrix(y_test,y_preds),color_continuous_scale="inferno")
fig.update_xaxes(title_text="Predicted label")
fig.update_yaxes(title_text="True label")
fig.show()
```

executed in 91ms, finished 09-07-2021 20:09:59



Now we've got a ROC curve, an AUC metric and a confusion matrix, let's get a classification report as well as cross-validated precision, recall and f1-score.

In [88]:
```
print(classification_report(y_test,y_preds))
```
executed in 28ms, finished 09-07-2021 21:49:02

```
              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
```

Calculate evaluation metrics using cross-validation

We're going to calculate accuracy, precision, recall and f1-score of our model using cross-validation and to do so we'll be using `cross_val_score()`

In [89]:
```
# Chceck best hyperparameters
gs_log_reg.best_params_
```
executed in 16ms, finished 09-07-2021 21:56:04

Out[89]:
```
{'C': 0.20433597178569418, 'solver': 'liblinear'}
```

In [90]:
```
# Create a new classifier with best parameters
clf = LogisticRegression(C=0.20433597178569418,
                         solver="liblinear")
```
executed in 6ms, finished 09-07-2021 21:57:27

In [92]:
```
# Cross-validated accuracy
cv_acc = cross_val_score(clf,
                         X,
                         y,
                         cv=5,
                         scoring="accuracy")
cv_acc
```
executed in 79ms, finished 09-07-2021 21:59:27

Out[92]:
```
array([0.81967213, 0.90163934, 0.86885246, 0.88333333, 0.75      ])
```

In [94]:
```
cv_acc = np.mean(cv_acc)
cv_acc
```
executed in 13ms, finished 09-07-2021 22:00:31

Out[94]:
```
0.8446994535519124
```

In [95]:
```
# Cross validated precision
cv_precision = cross_val_score(clf,
                         X,
                         y,
                         cv=5,
                         scoring="precision")
cv_precision = np.mean(cv_precision)
cv_precision
```
executed in 51ms, finished 09-07-2021 22:02:13

Out[95]:
```
0.8207936507936507
```

In [96]:
```
# Cross validated recall
cv_recall = cross_val_score(clf,
                         X,
                         y,
                         cv=5,
                         scoring="recall")
cv_recall = np.mean(cv_recall)
cv_recall
```
executed in 68ms, finished 09-07-2021 22:03:21

Out[96]:
```
0.9212121212121213
```

In [97]:

```python
# Cross validated f1-score
cv_f1 = cross_val_score(clf,
                        X,
                        y,
                        cv=5,
                        scoring="f1")
cv_f1 = np.mean(cv_f1)
cv_f1
```

executed in 52ms, finished 09-07-2021 22:04:04

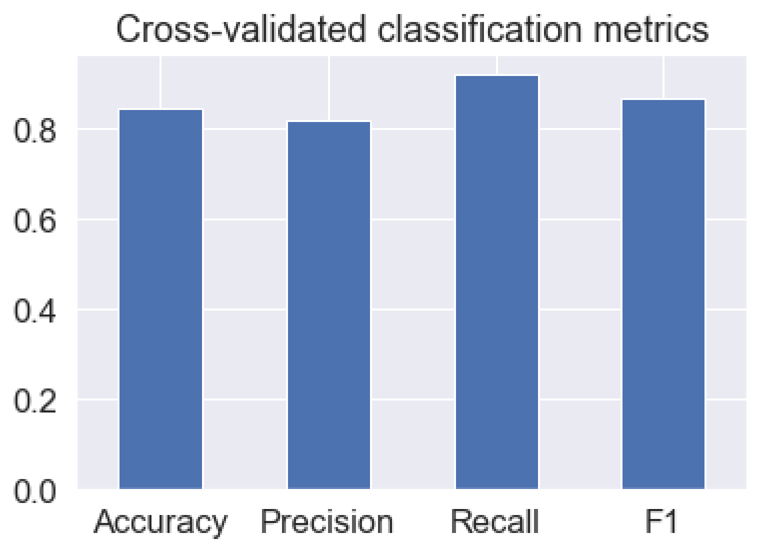
Out[97]: 0.8673007976269721

In [102]:

```python
# Visualise cross-validated metrics using Matplotlib
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);
plt.xticks(rotation=0);
```

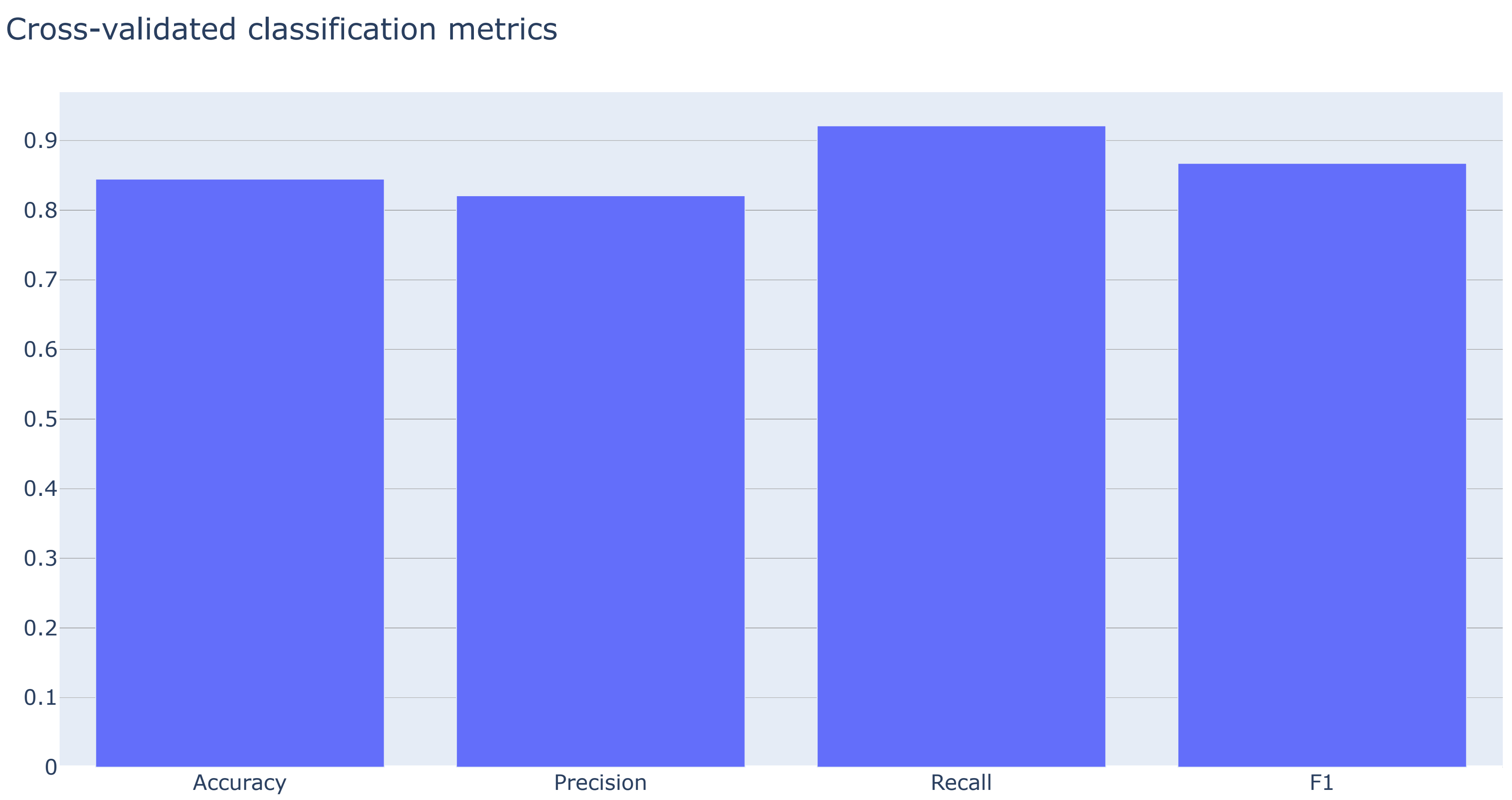
executed in 139ms, finished 09-07-2021 22:09:21



In [107]:

```python
# Visualise cross-validated metrics using Plotly
fig = px.bar(cv_metrics.T)
fig.update_xaxes(title_text=" ")
fig.update_yaxes(title_text=" ")
fig.update_layout(title="Cross-validated classification metrics")
fig.show()
```

executed in 83ms, finished 09-07-2021 22:12:22



Feature Importance

Feature importance is same as asking, "Which features contributed most to the outcomes of the 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 "(MODEL NAME) feature importance".

Let's find the feature importance for our LogisticRegression model...

In [110]:

```
# Fit an instance of LogisticRegression
clf = LogisticRegression(C=0.20433597178569418,
                         solver="liblinear")

clf.fit(X_train,y_train)
```

executed in 18ms, finished 09-07-2021 22:21:47

Out[110]:

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

In [112]:

```
# Check coef_
clf.coef_
```

executed in 14ms, finished 09-07-2021 22:22:46

Out[112]:

```
array([[ 0.00316728, -0.86044674,  0.66067031, -0.01156993, -0.00166375,
         0.04386101,  0.31275865,  0.02459362, -0.60413094, -0.56862789,
         0.45051632, -0.63609908, -0.67663375]])
```

In [114]:

```
df.head()
```

executed in 23ms, finished 09-07-2021 22:26:23

Out[114]:

	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 [113]:

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

executed in 21ms, finished 09-07-2021 22:25:36

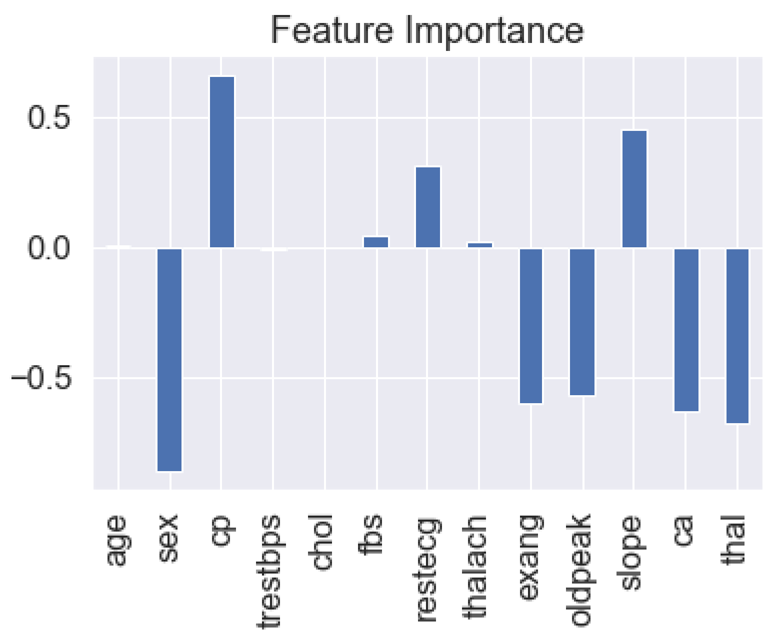
Out[113]:

```
{'age': 0.0031672830780218957,
 'sex': -0.8604467440762573,
 'cp': 0.6606703120090932,
 'trestbps': -0.011569932037408597,
 'chol': -0.00166374523064295,
 'fbs': 0.043861009724542044,
 'restecg': 0.3127586507840532,
 'thalach': 0.024593615555173243,
 'exang': -0.6041309439103262,
 'oldpeak': -0.5686278914396258,
 'slope': 0.4505163222528207,
 'ca': -0.6360990763634887,
 'thal': -0.6766337475895309}
```

In [117]:

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

executed in 209ms, finished 09-07-2021 22:29:22



In [120]:

```
# Visualize feature importance using Plotly
fig = px.bar(feature_df.T)
fig.update_xaxes(title_text=" ")
fig.update_yaxes(title_text=" ")
fig.update_layout(title="Feature Importance")
fig.show()
```

executed in 85ms, finished 09-07-2021 22:31:16

In [121]:

```
pd.crosstab(df["sex"],df["target"])
```

executed in 29ms, finished 09-07-2021 22:34:56

Out[121]:

target	0	1
sex		
0	24	72
1	114	93

In [122]:

```
pd.crosstab(df["slope"],df["target"])
```

executed in 42ms, finished 09-07-2021 22:35:27

Out[122]:

target	0	1
slope		
0	12	9
1	91	49
2	35	107

slope. Peak exercise ST segment

- 0 = upsloping
- 1 = flat
- 2 = downsloping.

Function to return the Calculated Metrics.

In [126]:
```python
def evaluate_preds(y_true,y_preds):
    """
    Performs evaluation comparison on y_true labels vs. y_pred labels
    on a classification model.
    """
    accuracy = accuracy_score(y_true,y_preds)
    precision = precision_score(y_true,y_preds)
    recall = recall_score(y_true,y_preds)
    f1 = f1_score(y_true,y_preds)
    metric_dict = {"accuracy": round(accuracy,2),
                   "precision": round(precision,2),
                   "recall": round(recall,2),
                   "f1": round(f1,2)}
    print(f"Acc:{accuracy * 100:.2f}%")
    print(f"Precision:{precision:.2f}")
    print(f"Recall:{recall:.2f}")
    print(f"F1 score:{f1:.2f}")

    return metric_dict
```
executed in 11ms, finished 09-07-2021 22:59:48

Exporting the model

Using `Joblib`

In [123]:
```python
from joblib import dump, load

# Save model to file
dump(clf, filename="Heart-Disease-Project.joblib")
```
executed in 15ms, finished 09-07-2021 22:54:27

Out[123]: ['Heart-Disease-Project.joblib']

In [124]:
```python
# Import a saved joblib model
loaded_job_model = load(filename = "Heart-Disease-Project.joblib")
```
executed in 19ms, finished 09-07-2021 22:55:09

In [127]:
```python
# Make and evaluate joblib predictions
joblib_y_preds = loaded_job_model.predict(X_test)
evaluate_preds(y_test,joblib_y_preds)
```
executed in 37ms, finished 09-07-2021 22:59:57

```
Acc:88.52%
Precision:0.88
Recall:0.91
F1 score:0.89
```

Out[127]: {'accuracy': 0.89, 'precision': 0.88, 'recall': 0.91, 'f1': 0.89}

6. Experimentation

If you haven't hit your evaluation metric yet... ask yourself...

- Could you collect more data?
- Could you try a better model? Like CatBoost or XGBoost?
- Could you improve the cuurent models? (beyond what we've done so far)
- If your model is good enough (you have hit your evaluation metric) how would you export it and share it with others?