Assignment 10

Heart Disease Data

This is a multivariate type of dataset which means providing or involving a variety of separate mathematical or statistical variables, multivariate numerical data analysis. It is composed of 14 attributes which are age, sex, chest pain type, resting blood pressure, serum cholesterol, fasting blood sugar, resting electrocardiographic results, maximum heart rate achieved, exercise-induced angina, oldpeak — ST depression induced by exercise relative to rest, the slope of the peak exercise ST segment, number of major vessels and Thalassemia. This database includes 76 attributes, but all published studies relate to the use of a subset of 14 of them. The Cleveland database is the only one used by ML researchers to date. One of the major tasks on this dataset is to predict based on the given attributes of a patient that whether that particular person has heart disease or not and other is the experimental task to diagnose and find out various insights from this dataset which could help in understanding the problem more.

Hint: heart_disease_uci.csv

Instructions:

1. Use Lifecycle of Data Sciece
2. Use necessary data Preprocess techniques
3. Use various Regression and Classification techniques for comparision
4. Use metrics for regression and classification when needed.
5. Use variosu Pipeline/Hyperparametr tuning techniques for improving performance

In [91]:
```
import numpy as np
import pandas as pd

import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.figure_factory as ff
import plotly.graph_objects as go
%matplotlib inline

data = pd.read_csv("Documents/Data science Material JNTU/Assignments/Assignment 10/heart_disease_uci.csv")
data.head()
```

Out[91]:

	id	age	sex	dataset	cp	trestbps	chol	fbs	restecg	thalch	exang	oldpeak	slope	ca	thal	num
0	1	63	Male	Cleveland	typical angina	145.0	233.0	True	lv hypertrophy	150.0	False	2.3	downsloping	0.0	fixed defect	0
1	2	67	Male	Cleveland	asymptomatic	160.0	286.0	False	lv hypertrophy	108.0	True	1.5	flat	3.0	normal	2
2	3	67	Male	Cleveland	asymptomatic	120.0	229.0	False	lv hypertrophy	129.0	True	2.6	flat	2.0	reversable defect	1
3	4	37	Male	Cleveland	non-anginal	130.0	250.0	False	normal	187.0	False	3.5	downsloping	0.0	normal	0
4	5	41	Female	Cleveland	atypical angina	130.0	204.0	False	lv hypertrophy	172.0	False	1.4	upsloping	0.0	normal	0

In [92]:
```
data.info()
```

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

In [93]:
```
data.columns
```

Out[93]:
```
Index(['id', 'age', 'sex', 'dataset', 'cp', 'trestbps', 'chol', 'fbs',
       'restecg', 'thalch', 'exang', 'oldpeak', 'slope', 'ca', 'thal', 'num'],
      dtype='object')
```

In [94]:
```
data.describe()
```

Out[94]:

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

Handling Missing Data

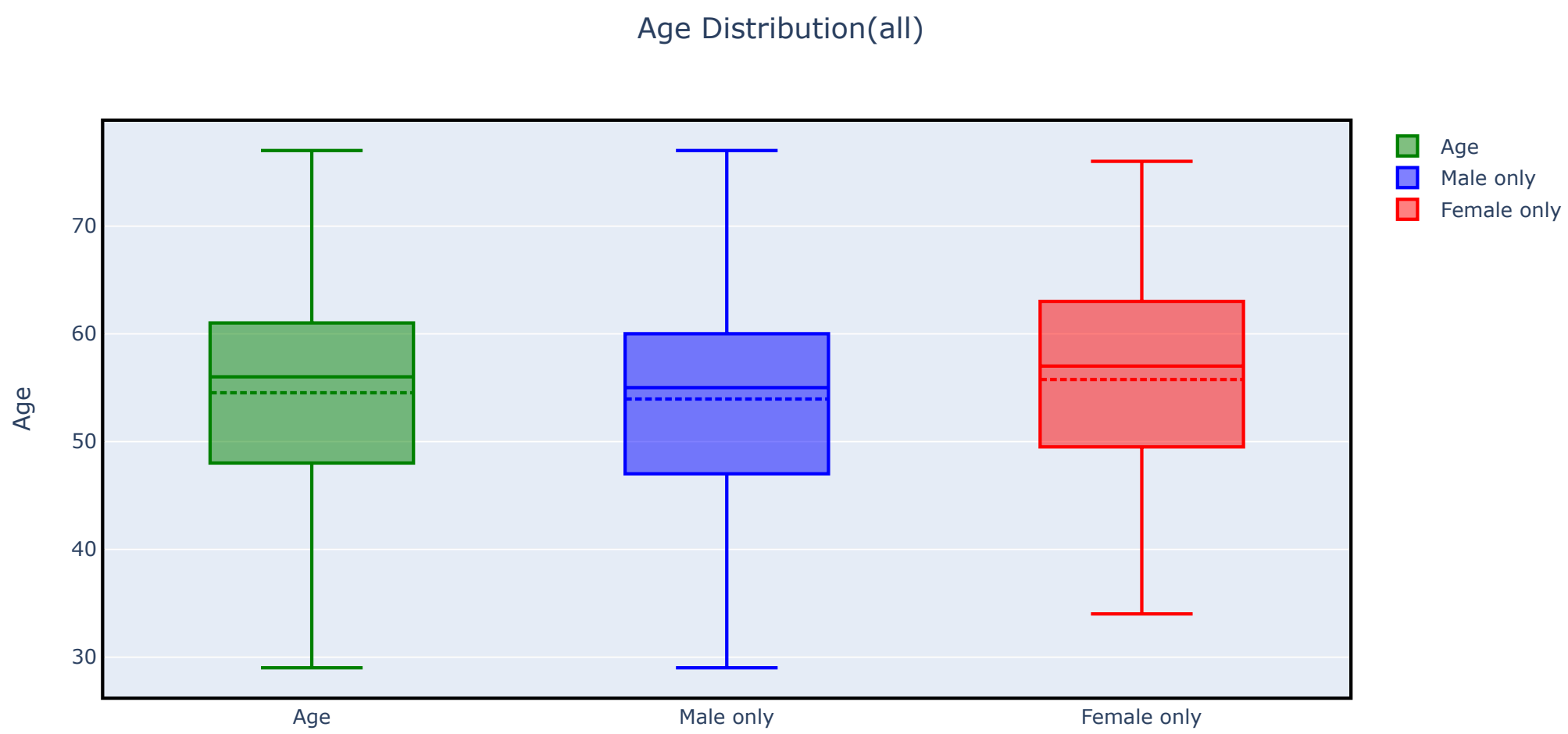
In [95]:
```
data.dropna(inplace = True)
from sklearn.utils import shuffle
data = shuffle(data)
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 299 entries, 177 to 273
Data columns (total 16 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   id        299 non-null    int64  
 1   age       299 non-null    int64  
 2   sex       299 non-null    object 
 3   dataset   299 non-null    object 
 4   cp        299 non-null    object 
 5   trestbps  299 non-null    float64
 6   chol      299 non-null    float64
 7   fbs       299 non-null    object 
 8   restecg   299 non-null    object 
 9   thalch    299 non-null    float64
 10  exang     299 non-null    object 
 11  oldpeak   299 non-null    float64
 12  slope     299 non-null    object 
 13  ca        299 non-null    float64
 14  thal      299 non-null    object 
 15  num       299 non-null    int64  
dtypes: float64(5), int64(3), object(8)
memory usage: 39.7+ KB
```

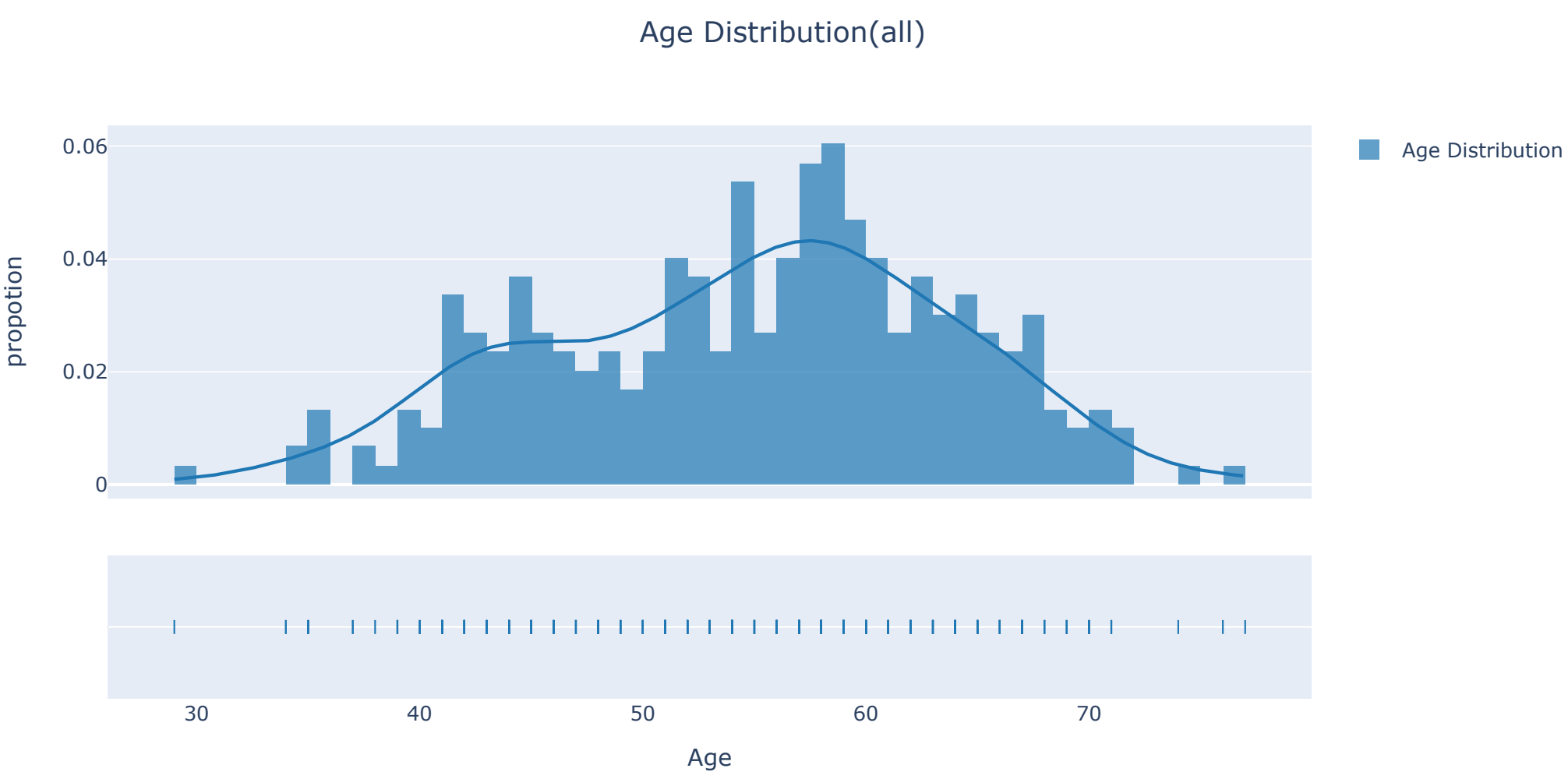
Data Visualization

1. Age Distribution:

In [96]:
```
fig = go.Figure()
fig.add_trace(go.Box(y=data['age'].values , name='Age', marker_color = 'green',boxmean=True))
fig.add_trace(go.Box(y=data[data['sex']=='Male']['age'].values, name ='Male only',
                     marker_color = 'blue', boxmean = True))
fig.add_trace(go.Box(y=data[data['sex']=='Female']['age'].values, name ='Female only',
                     marker_color = 'red', boxmean = True))
fig.update_layout(title = 'Age Distribution(all)', yaxis_title = 'Age', title_x = 0.5)
fig.update_xaxes(showline=True, linewidth=2, linecolor='black', mirror=True)
fig.update_yaxes(showline=True, linewidth=2, linecolor='black', mirror=True)
fig.show()
```

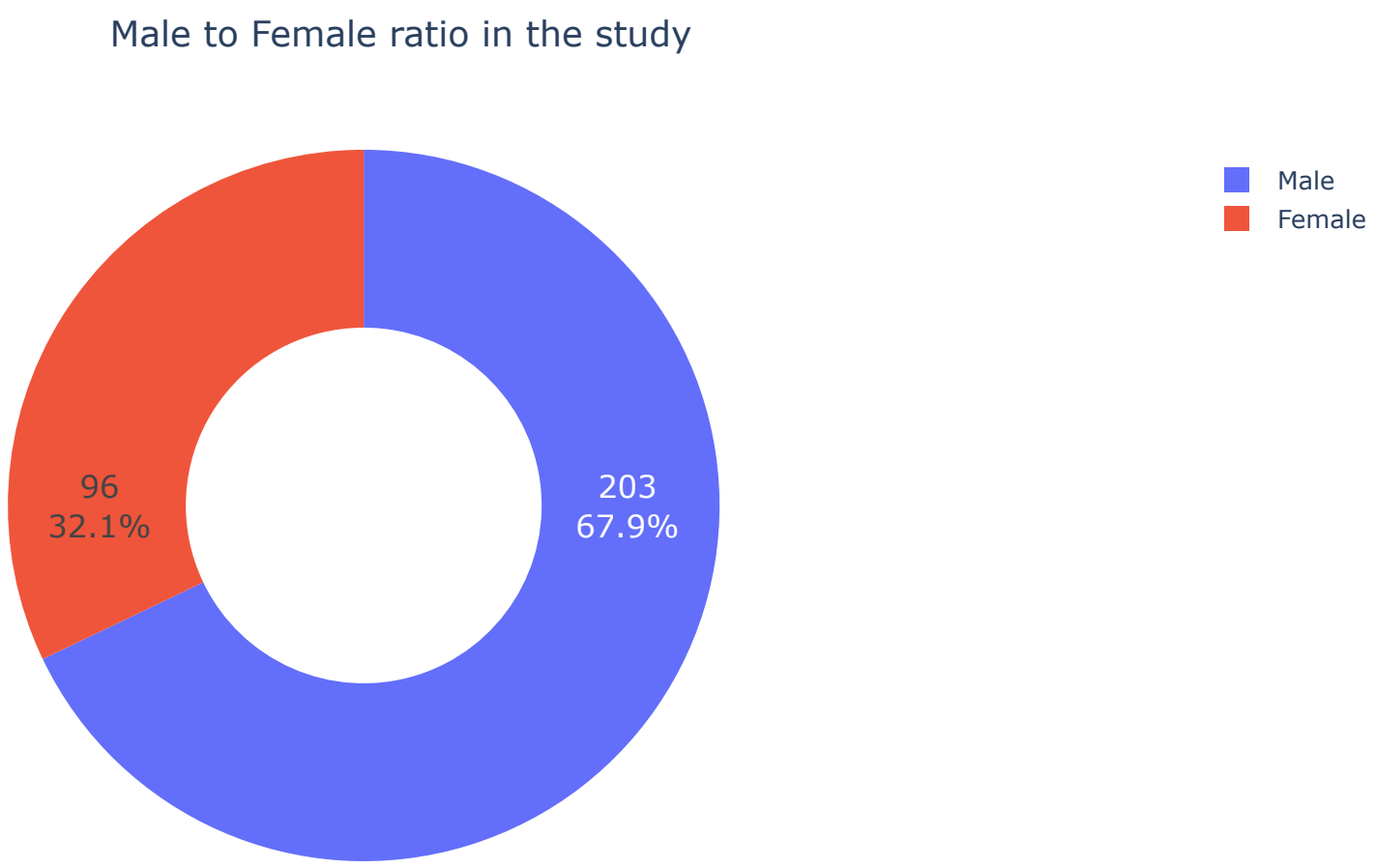


In [97]:
```
group_labels = ['Age Distribution'] # name of the dataset
fig = ff.create_distplot([data.age], group_labels)
fig.update_layout(title = 'Age Distribution(all)', yaxis_title = 'propotion', xaxis_title = 'Age', title_x = 0.5)
fig.show()
```



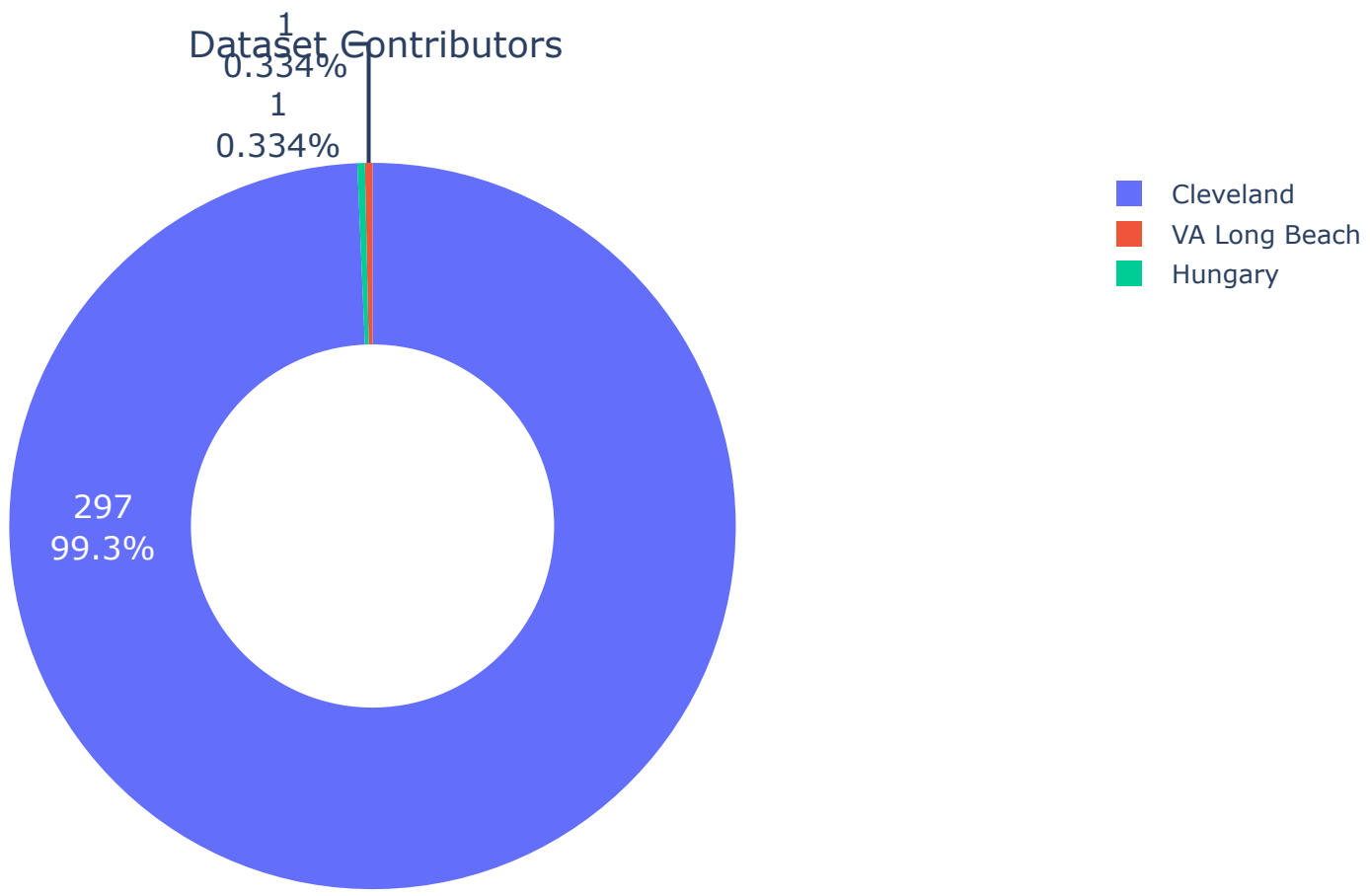
2. Male and Female Propotion

In [98]:
```
df=data['sex'].value_counts().reset_index().rename(columns={'index':'sex','sex':'count'})
fig = go.Figure([go.Pie(labels=['Male', 'Female'],values=df['count'], hole = 0.5)])
fig.update_traces(hoverinfo='label+percent', textinfo='value+percent', textfont_size=15,
                  insidetextorientation='radial')
fig.update_layout(title="Male to Female ratio in the study",title_x=0.5)
fig.show()
```



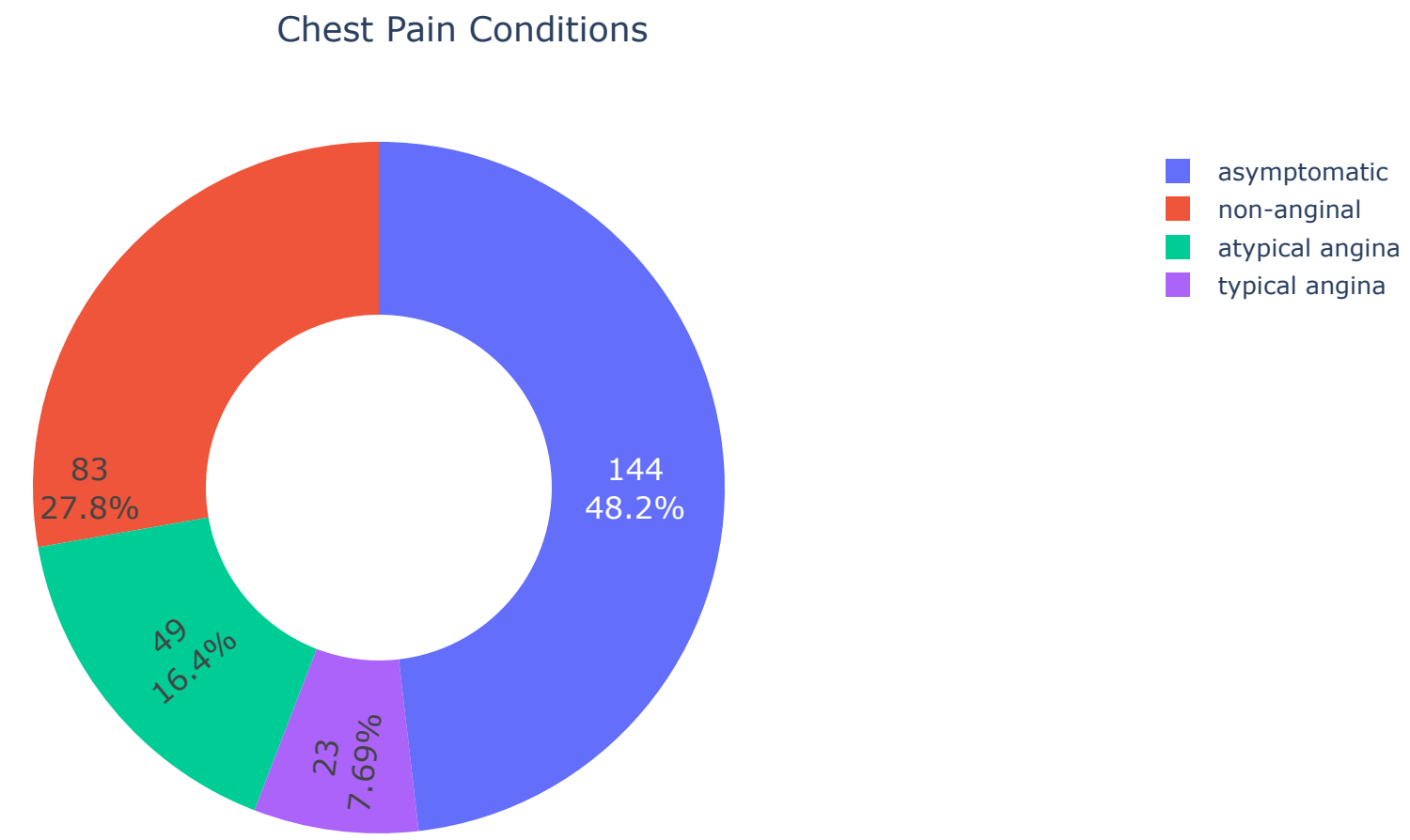
3. Dataset Contributors:

In [99]:
```
df=data['dataset'].value_counts().reset_index().rename(columns={'index':'dataset','dataset':'count'})
fig = go.Figure([go.Pie(labels=df['dataset'],values=df['count'], hole = 0.5)])
fig.update_traces(hoverinfo='label+percent', textinfo='value+percent', textfont_size=15,
                  insidetextorientation='radial')
fig.update_layout(title="Dataset Contributors",title_x=0.5)
fig.show()
```



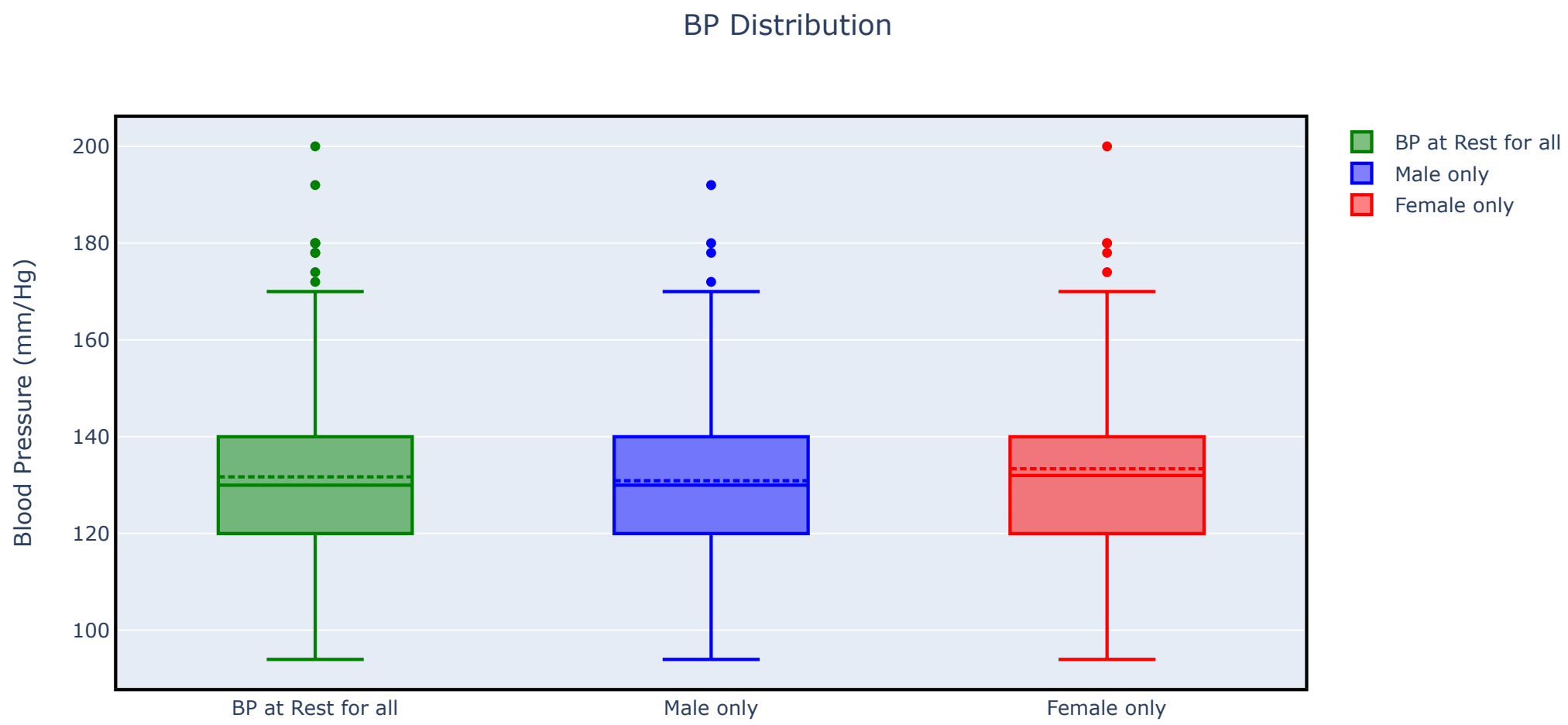
4. CP(Chest Pain Type) Propotions

In [100]:
```
df=data['cp'].value_counts().reset_index().rename(columns={'index':'cp','cp':'count'})
fig = go.Figure([go.Pie(labels=df['cp'],values=df['count'], hole = 0.5)])
fig.update_traces(hoverinfo='label+percent', textinfo='value+percent',
                  textfont_size=15,insidetextorientation='radial')
fig.update_layout(title="Chest Pain Conditions",title_x=0.5)
fig.show()
```



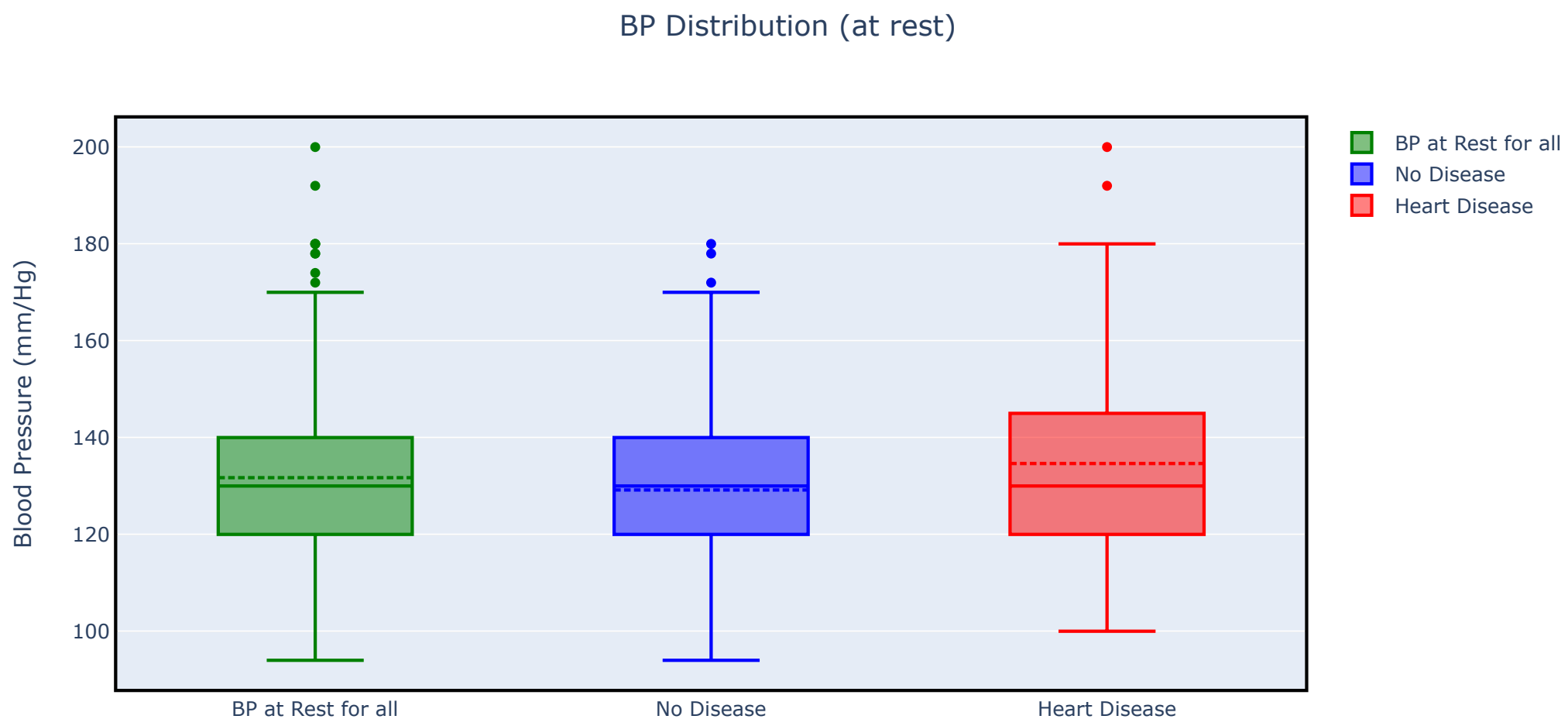
5. Resting Blood Pressure vs Gender:

In [101]:
```
fig = go.Figure()
fig.add_trace(go.Box(y=data['trestbps'].values , name='BP at Rest for all', marker_color = 'green',boxmean=True))
fig.add_trace(go.Box(y=data[data['sex']=='Male']['trestbps'].values, name ='Male only',
                     marker_color = 'blue', boxmean = True))
fig.add_trace(go.Box(y=data[data['sex']=='Female']['trestbps'].values, name ='Female only',
                     marker_color = 'red', boxmean = True))
fig.update_layout(title = 'BP Distribution', yaxis_title = 'Blood Pressure (mm/Hg)', title_x = 0.5)
fig.update_xaxes(showline=True, linewidth=2, linecolor='black', mirror=True)
fig.update_yaxes(showline=True, linewidth=2, linecolor='black', mirror=True)
fig.show()
```



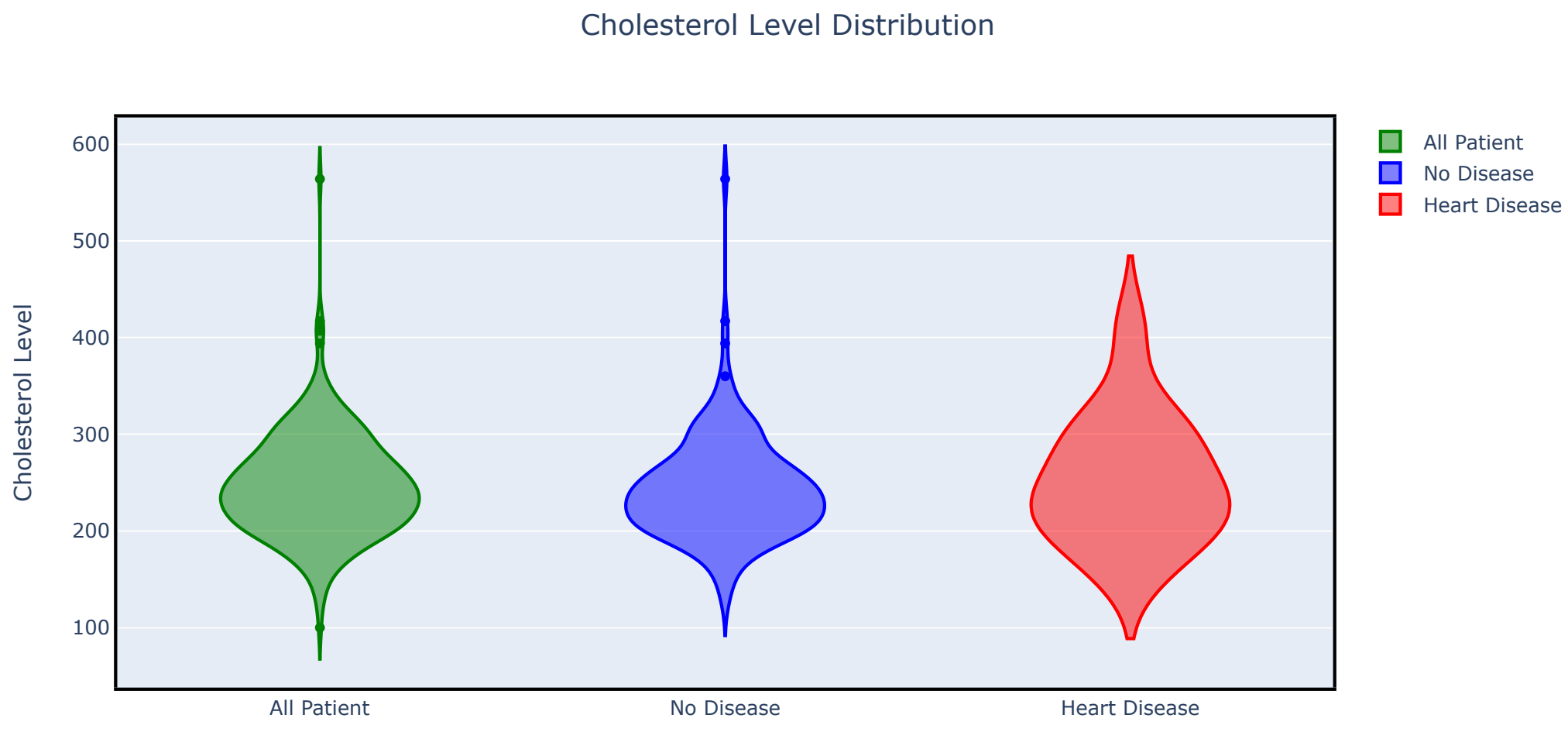
6. Resting Blood Prssure vs Disease

In [102]:
```
fig = go.Figure()
fig.add_trace(go.Box(y=data['trestbps'].values , name='BP at Rest for all', marker_color = 'green',boxmean=True))
fig.add_trace(go.Box(y=data[data['num']== 0]['trestbps'].values, name ='No Disease', marker_color = 'blue',
                     boxmean = True))
fig.add_trace(go.Box(y=data[data['num'] !=0]['trestbps'].values, name ='Heart Disease', marker_color = 'red',
                     boxmean = True))
fig.update_layout(title = 'BP Distribution (at rest)', yaxis_title = 'Blood Pressure (mm/Hg)', title_x = 0.5)
fig.update_xaxes(showline=True, linewidth=2, linecolor='black', mirror=True)
fig.update_yaxes(showline=True, linewidth=2, linecolor='black', mirror=True)
fig.show()
```



7. Cholesterol Level Distribution

In [103]:
```
fig = go.Figure()
fig.add_trace(go.Violin(y=data['chol'].values , name='All Patient', marker_color = 'green'))
fig.add_trace(go.Violin(y=data[data['num']== 0]['chol'].values, name ='No Disease', marker_color = 'blue'))
fig.add_trace(go.Violin(y=data[data['num'] ==4]['chol'].values, name ='Heart Disease', marker_color = 'red'))
fig.update_layout(title = 'Cholesterol Level Distribution', yaxis_title = 'Cholesterol Level', title_x = 0.5)
fig.update_xaxes(showline=True, linewidth=2, linecolor='black', mirror=True)
fig.update_yaxes(showline=True, linewidth=2, linecolor='black', mirror=True)
fig.show()
```



Classification

Pre-Processing

If we just look at the data, we will see some of the features have categorical values. So we have to do one hot encoding for them. Also the original dataset contains the target as 0, 1, 2, 3, 4. But for identifying simply the presence of disease, we will take binary classification. With that view in mind, we will covert all the target features in the num column into 1/0.

In [104]:
```
# In some of the features, there is space will will create problem later on.
# So we rename those attributes to handle problems in the future.

# data["restecg"].replace({"lv hypertrophy": "lv_hypertrophy","st-t abnormality": "stt_abnormality" }, inplace=True)
data['thal'].replace({'fixed defect':'fixed_defect' , 'reversable defect': 'reversable_defect' }, inplace =True)
data['cp'].replace({'typical angina':'typical_angina', 'atypical angina': 'atypical_angina' }, inplace =True)


data_tmp = data[['age','sex','cp', 'trestbps', 'chol', 'fbs',  'thalch', 'exang',
                 'oldpeak', 'slope', 'ca', 'thal']].copy()
data_tmp['target'] = ((data['num'] > 0)*1).copy()
data_tmp['sex'] = (data['sex'] == 'Male')*1
data_tmp['fbs'] = (data['fbs'])*1
data_tmp['exang'] = (data['exang'])*1

data_tmp.columns = ['age', 'sex', 'chest_pain_type', 'resting_blood_pressure',
              'cholesterol', 'fasting_blood_sugar',
              'max_heart_rate_achieved', 'exercise_induced_angina',
              'st_depression', 'st_slope_type', 'num_major_vessels',
              'thalassemia_type', 'target']
data_tmp.head(15)
```

Out[104]:

	age	sex	chest_pain_type	resting_blood_pressure	cholesterol	fasting_blood_sugar	max_heart_rate_achieved	exercise_induced_angina	st_depression	st_slo
177	56	1	asymptomatic	132.0	184.0	0	105.0	1	2.1	
217	46	0	asymptomatic	138.0	243.0	0	152.0	1	0.0	
748	56	1	asymptomatic	120.0	100.0	0	120.0	1	1.5	
131	51	1	non-anginal	94.0	227.0	0	154.0	1	0.0	u
40	65	0	asymptomatic	150.0	225.0	0	114.0	0	1.0	
142	52	1	atypical_angina	128.0	205.0	1	184.0	0	0.0	u
508	47	1	asymptomatic	150.0	226.0	0	98.0	1	1.5	
82	39	1	non-anginal	140.0	321.0	0	182.0	0	0.0	u
20	64	1	typical_angina	110.0	211.0	0	144.0	1	1.8	
180	48	1	asymptomatic	124.0	274.0	0	166.0	0	0.5	
183	59	1	typical_angina	178.0	270.0	0	145.0	0	4.2	dow
174	64	1	asymptomatic	145.0	212.0	0	132.0	0	2.0	
103	71	0	non-anginal	110.0	265.0	1	130.0	0	0.0	u
41	40	1	typical_angina	140.0	199.0	0	178.0	1	1.4	u
231	55	0	asymptomatic	180.0	327.0	0	117.0	1	3.4	

One-hot Encoding

In [105]:
```
data = pd.get_dummies(data_tmp, drop_first=False)
data.columns
```

```
/Users/nagendra/opt/anaconda3/lib/python3.9/site-packages/pandas/core/algorithms.py:798: FutureWarning:

In a future version, the Index constructor will not infer numeric dtypes when passed object-dtype sequences (matching
Series behavior)
```

Out[105]:
```
Index(['age', 'sex', 'resting_blood_pressure', 'cholesterol',
       'max_heart_rate_achieved', 'st_depression', 'num_major_vessels',
       'target', 'chest_pain_type_asymptomatic',
       'chest_pain_type_atypical_angina', 'chest_pain_type_non-anginal',
       'chest_pain_type_typical_angina', 'fasting_blood_sugar_0',
       'fasting_blood_sugar_1', 'exercise_induced_angina_0',
       'exercise_induced_angina_1', 'st_slope_type_downsloping',
       'st_slope_type_flat', 'st_slope_type_upsloping',
       'thalassemia_type_fixed_defect', 'thalassemia_type_normal',
       'thalassemia_type_reversable_defect'],
      dtype='object')
```

Logistic Regression:

Gathering Data let us separate the input and labels for the dataset and thus we will be able to put them in the training models.

In [106]:
```
from sklearn.model_selection import train_test_split
y = data['target']
X = data.drop('target', axis = 1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
print(f'Shape of X_train: {X_train.shape}')
print(f'Shape of y_train: {y_train.shape}')
print(f'Shape of X_test: {X_test.shape}')
print(f'Shape of y_test: {y_test.shape}')
```

```
Shape of X_train: (239, 21)
Shape of y_train: (239,)
Shape of X_test: (60, 21)
Shape of y_test: (60,)
```

Normalization

Min-Max Normalization method is used to Normalize the data. This method scales the data range to [0,1]. Standardization is also used on a feature-wise basis in most cases. Normalization is done by the following formula.

$xscaled=(x-xmin)/(xmax-xmin)$

In [107]:
```
X_train=(X_train-np.min(X_train))/(np.max(X_train)-np.min(X_train)).values
X_test=(X_test-np.min(X_test))/(np.max(X_test)-np.min(X_test)).values
```

```
/Users/nagendra/opt/anaconda3/lib/python3.9/site-packages/numpy/core/fromnumeric.py:84: FutureWarning:

In a future version, DataFrame.min(axis=None) will return a scalar min over the entire DataFrame. To retain the old b
ehavior, use 'frame.min(axis=0)' or just 'frame.min()'

/Users/nagendra/opt/anaconda3/lib/python3.9/site-packages/numpy/core/fromnumeric.py:84: FutureWarning:

In a future version, DataFrame.max(axis=None) will return a scalar max over the entire DataFrame. To retain the old b
ehavior, use 'frame.max(axis=0)' or just 'frame.max()'

/Users/nagendra/opt/anaconda3/lib/python3.9/site-packages/numpy/core/fromnumeric.py:84: FutureWarning:

In a future version, DataFrame.min(axis=None) will return a scalar min over the entire DataFrame. To retain the old b
ehavior, use 'frame.min(axis=0)' or just 'frame.min()'

/Users/nagendra/opt/anaconda3/lib/python3.9/site-packages/numpy/core/fromnumeric.py:84: FutureWarning:

In a future version, DataFrame.min(axis=None) will return a scalar min over the entire DataFrame. To retain the old b
ehavior, use 'frame.min(axis=0)' or just 'frame.min()'

/Users/nagendra/opt/anaconda3/lib/python3.9/site-packages/numpy/core/fromnumeric.py:84: FutureWarning:

In a future version, DataFrame.max(axis=None) will return a scalar max over the entire DataFrame. To retain the old b
ehavior, use 'frame.max(axis=0)' or just 'frame.max()'

/Users/nagendra/opt/anaconda3/lib/python3.9/site-packages/numpy/core/fromnumeric.py:84: FutureWarning:

In a future version, DataFrame.max(axis=None) will return a scalar max over the entire DataFrame. To retain the old b
ehavior, use 'frame.max(axis=0)' or just 'frame.max()'

/Users/nagendra/opt/anaconda3/lib/python3.9/site-packages/numpy/core/fromnumeric.py:84: FutureWarning:

In a future version, DataFrame.min(axis=None) will return a scalar min over the entire DataFrame. To retain the old b
ehavior, use 'frame.min(axis=0)' or just 'frame.min()'
```

In [108]:
```
X_test
```

Out[108]:

	age	sex	resting_blood_pressure	cholesterol	max_heart_rate_achieved	st_depression	num_major_vessels	chest_pain_type_asymptomatic	chest_pain_t
121	0.777778	0.0	0.651163	0.965636	0.614458	1.000	1.000000	1.0	
59	0.444444	1.0	0.360465	0.298969	0.265060	0.350	0.333333	0.0	
103	1.000000	0.0	0.186047	0.477663	0.325301	0.000	0.333333	0.0	
284	0.722222	1.0	0.627907	0.264605	0.698795	0.000	0.333333	1.0	
256	0.888889	0.0	0.139535	0.333333	0.469880	0.075	0.666667	1.0	
66	0.694444	1.0	0.534884	0.202749	0.626506	0.750	0.000000	0.0	
151	0.194444	0.0	0.093023	0.477663	0.228916	0.150	0.000000	1.0	
157	0.638889	1.0	0.360465	0.597938	0.819277	0.000	0.666667	1.0	
145	0.333333	1.0	0.162791	0.402062	0.590361	0.000	0.000000	0.0	
39	0.722222	1.0	0.651163	0.402062	0.409639	0.250	0.000000	0.0	
271	0.861111	1.0	0.767442	0.350515	0.421687	0.575	0.000000	1.0	
99	0.361111	1.0	0.325581	0.329897	1.000000	0.000	0.000000	1.0	
86	0.333333	1.0	0.511628	0.450172	0.638554	0.000	0.000000	0.0	
281	0.333333	1.0	0.418605	0.436426	0.915663	0.000	0.000000	0.0	
125	0.277778	0.0	0.418605	0.371134	0.867470	0.150	0.000000	0.0	
104	0.388889	1.0	0.302326	0.213058	0.433735	0.500	1.000000	0.0	
11	0.583333	0.0	0.534884	0.577320	0.602410	0.325	0.000000	0.0	
61	0.305556	0.0	0.558140	0.175258	0.686747	0.350	0.000000	0.0	
230	0.472222	0.0	0.488372	0.240550	0.795181	0.025	0.000000	0.0	
25	0.416667	0.0	0.302326	0.319588	0.662651	0.400	0.000000	0.0	
288	0.583333	1.0	0.418605	0.326460	0.722892	0.000	0.000000	0.0	

206	0.638889	1.0	0.395349	0.457045	0.325301	0.750	0.666667	1.0
222	0.111111	0.0	0.000000	0.250859	0.915663	0.000	0.000000	0.0
229	0.861111	1.0	0.209302	0.295533	0.349398	0.025	0.333333	1.0
134	0.222222	0.0	0.325581	0.298969	0.746988	0.050	0.000000	0.0
280	0.611111	1.0	0.186047	0.718213	0.481928	0.750	0.333333	1.0
239	0.194444	1.0	0.302326	0.580756	0.710843	0.000	0.000000	0.0
299	0.916667	1.0	0.581395	0.230241	0.457831	0.850	0.666667	1.0
83	0.916667	1.0	1.000000	0.508591	0.566265	0.400	0.000000	0.0
242	0.388889	0.0	0.418605	0.491409	0.722892	0.000	0.000000	1.0
14	0.472222	1.0	0.906977	0.250859	0.710843	0.125	0.000000	0.0
293	0.777778	1.0	0.534884	0.209622	0.493976	1.000	0.666667	1.0
20	0.805556	1.0	0.186047	0.292096	0.493976	0.450	0.000000	0.0
224	0.777778	0.0	0.162791	0.491409	0.795181	0.450	0.666667	1.0
6	0.750000	0.0	0.534884	0.487973	0.686747	0.900	0.666667	1.0
96	0.666667	1.0	0.186047	0.388316	0.469880	0.300	0.333333	1.0
48	0.833333	0.0	0.534884	1.000000	0.650602	0.200	0.333333	0.0
248	0.472222	1.0	0.360465	0.295533	0.783133	0.250	0.666667	1.0
273	1.000000	0.0	0.209302	0.079038	0.265060	0.400	0.000000	1.0
142	0.472222	1.0	0.395349	0.271478	0.975904	0.000	0.000000	0.0
228	0.527778	1.0	0.186047	0.274914	0.060241	0.000	0.333333	1.0
258	0.972222	1.0	0.720930	0.408935	0.481928	0.000	0.000000	0.0
70	0.833333	0.0	0.709302	0.491409	0.542169	0.200	0.000000	0.0
127	0.527778	1.0	0.186047	0.388316	0.277108	0.700	0.333333	1.0
76	0.694444	1.0	0.360465	0.453608	0.457831	0.700	0.333333	1.0
291	0.555556	0.0	0.441860	0.742268	0.759036	0.300	0.000000	0.0
211	0.083333	1.0	0.302326	0.360825	0.951807	0.950	0.000000	0.0
117	0.000000	0.0	0.511628	0.195876	0.951807	0.350	0.000000	1.0
82	0.111111	1.0	0.534884	0.670103	0.951807	0.000	0.000000	0.0
214	0.472222	1.0	0.209302	0.357388	0.686747	0.000	0.333333	1.0
81	0.500000	0.0	0.418605	0.474227	0.481928	0.100	0.000000	1.0
202	0.611111	1.0	0.651163	0.000000	0.843373	0.050	0.333333	0.0
300	0.611111	1.0	0.418605	0.017182	0.144578	0.300	0.333333	1.0
292	0.250000	1.0	0.302326	0.147766	0.493976	0.700	0.000000	1.0
168	0.000000	1.0	0.372093	0.536082	0.638554	0.000	0.000000	1.0
64	0.527778	1.0	0.302326	0.213058	0.120482	0.350	0.333333	1.0
246	0.638889	1.0	0.069767	0.371134	0.638554	0.025	0.333333	1.0
236	0.583333	1.0	0.418605	0.539519	0.000000	0.400	0.000000	1.0
249	0.750000	1.0	0.395349	0.281787	0.445783	0.000	0.000000	0.0
50	0.166667	0.0	0.127907	0.247423	0.783133	0.000	0.333333	0.0

60 rows × 21 columns

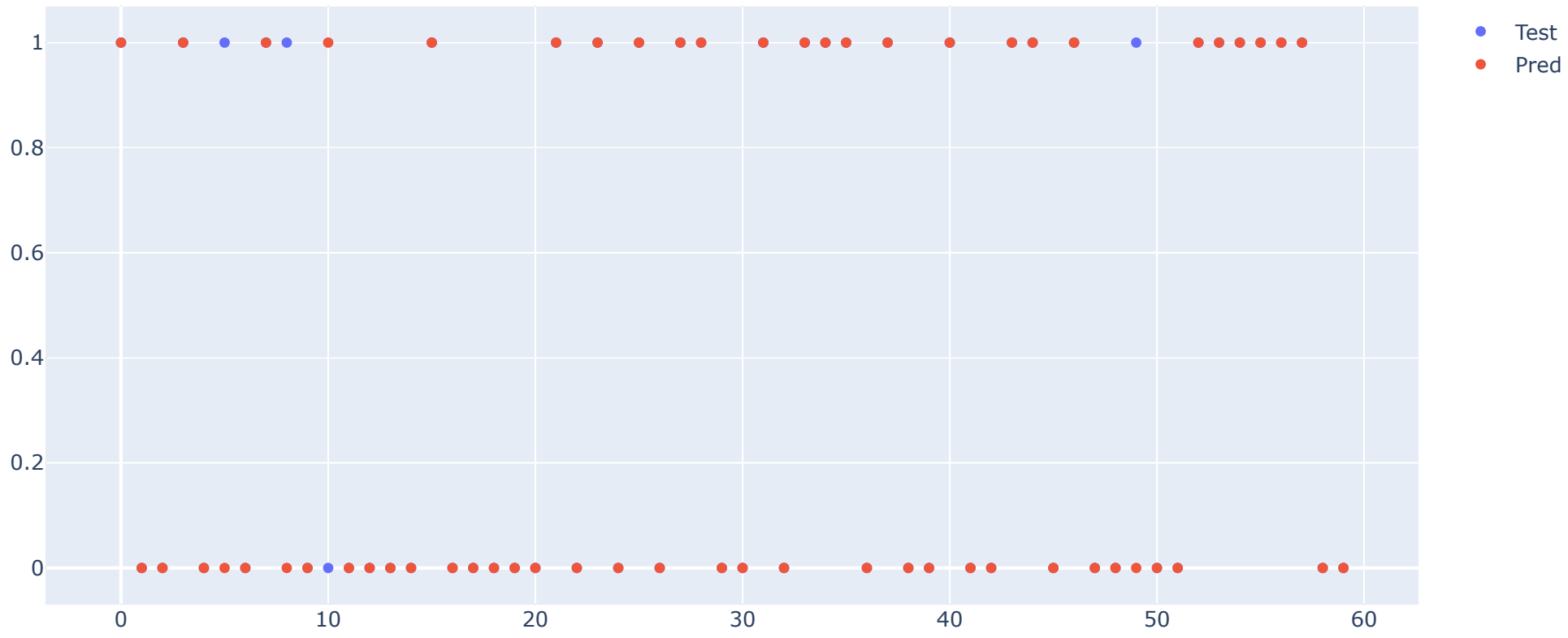
Fitting Into the Regression Model

In [109]:
```
from sklearn.linear_model import LogisticRegression
logre = LogisticRegression()
logre.fit(X_train,y_train)
```

Out[109]:
```
LogisticRegression()
```

Prediction

In [110]:
```
y_pred = logre.predict(X_test)
actual = []
predcition = []
for i,j in zip(y_test,y_pred):
    actual.append(i)
    predcition.append(j)

dic = {'Actual':actual,
       'Prediction':predcition }

result  = pd.DataFrame(dic)
import plotly.graph_objects as go

fig = go.Figure()
fig.add_trace(go.Scatter(x=np.arange(0,len(y_test)), y=y_test,    mode='markers',  name='Test'))
fig.add_trace(go.Scatter(x=np.arange(0,len(y_test)), y=y_pred,  mode='markers',  name='Pred'))
```



In the above figure, the red dots represent the predicted values that is either 0 or 1 and the blue line & and dot represents the actual value of that particular patient. In the places where the red dot and blue dot do not overlap are the wrong predictions and where the both dots overlap those are the right predicted values.

Model Evaluation

Accuracy

In [111]:
```
from sklearn.metrics import accuracy_score
print('The Accuracy Score is: ', accuracy_score(y_test,y_pred))
```

```
The Accuracy Score is:  0.9333333333333333
```

Precision, Recall, F1-Score, Support

In [112]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

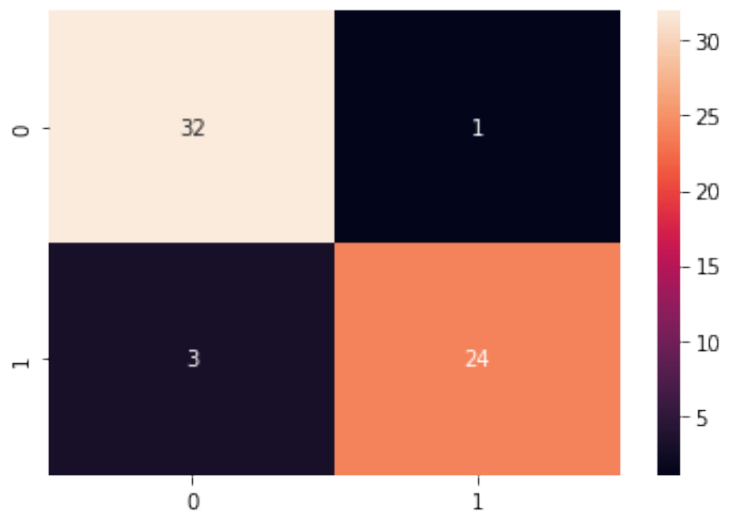
```
              precision    recall  f1-score   support

           0       0.91      0.97      0.94        33
           1       0.96      0.89      0.92        27

    accuracy                           0.93        60
   macro avg       0.94      0.93      0.93        60
weighted avg       0.93      0.93      0.93        60
```

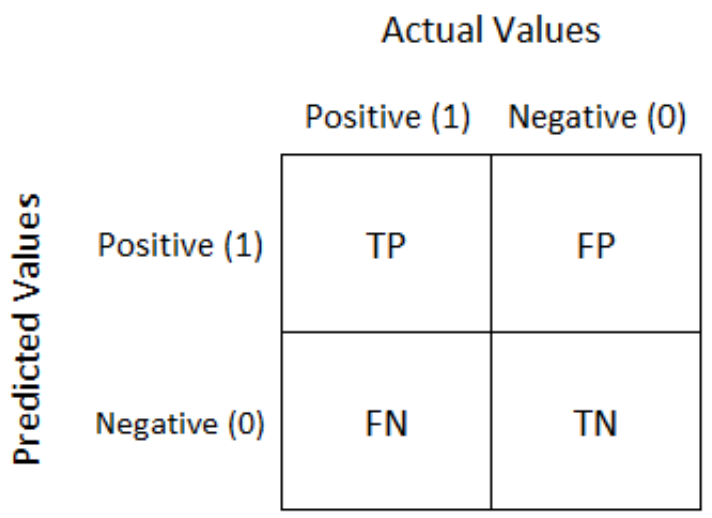
Confusion Matrix

In [113]:
```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test,y_pred))
sns.heatmap(confusion_matrix(y_test,y_pred),annot=True)
```

```
[[32  1]
 [ 3 24]]
```

Out[113]: <AxesSubplot:>



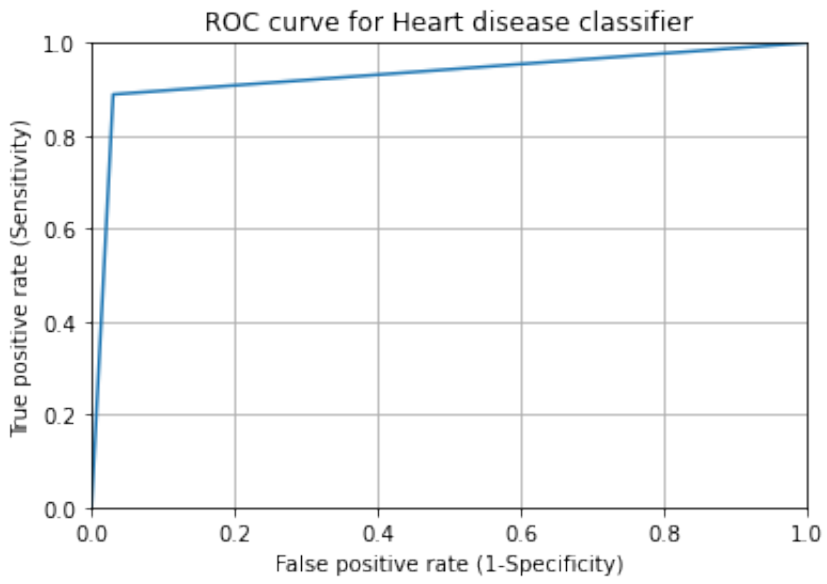


Area under ROC and ROC Curve

In [114]:
```
import sklearn
print('Area Under ROC-Curve: ', sklearn.metrics.roc_auc_score(y_test,y_pred))
```

```
Area Under ROC-Curve:  0.9292929292929294
```

In [115]:
```
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred, drop_intermediate = False)
plt.plot(fpr,tpr)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.title('ROC curve for Heart disease classifier')
plt.xlabel('False positive rate (1-Specificity)')
plt.ylabel('True positive rate (Sensitivity)')
plt.grid(True)
```



Analysis

Co-efficents:

Linear Regression actually calculates the total outcome by summing up the weighted sum of the different features. Let's have a look at those weights.

In [116]:
```
print(logre.intercept_)
plt.figure(figsize=(10,12))
coeffecients = pd.DataFrame(logre.coef_.ravel(),X.columns)
coeffecients.columns = ['Coeffecient']
coeffecients.sort_values(by=['Coeffecient'],inplace=True,ascending=False)
coeffecients
```

```
[-1.70969094]
```

Out[116]:

	Coeffecient
num_major_vessels	2.164922
resting_blood_pressure	1.216500
st_depression	1.014617
chest_pain_type_asymptomatic	0.915586
sex	0.799998
thalassemia_type_reversable_defect	0.723005
st_slope_type_flat	0.583592
age	0.434584
cholesterol	0.390178
exercise_induced_angina_1	0.315938
chest_pain_type_atypical_angina	0.160684
fasting_blood_sugar_0	0.023625
fasting_blood_sugar_1	-0.023598
st_slope_type_downsloping	-0.119785
thalassemia_type_fixed_defect	-0.135340
exercise_induced_angina_0	-0.315910
chest_pain_type_typical_angina	-0.445867
st_slope_type_upsloping	-0.463780
thalassemia_type_normal	-0.587637
chest_pain_type_non-anginal	-0.630376
max_heart_rate_achieved	-1.043557

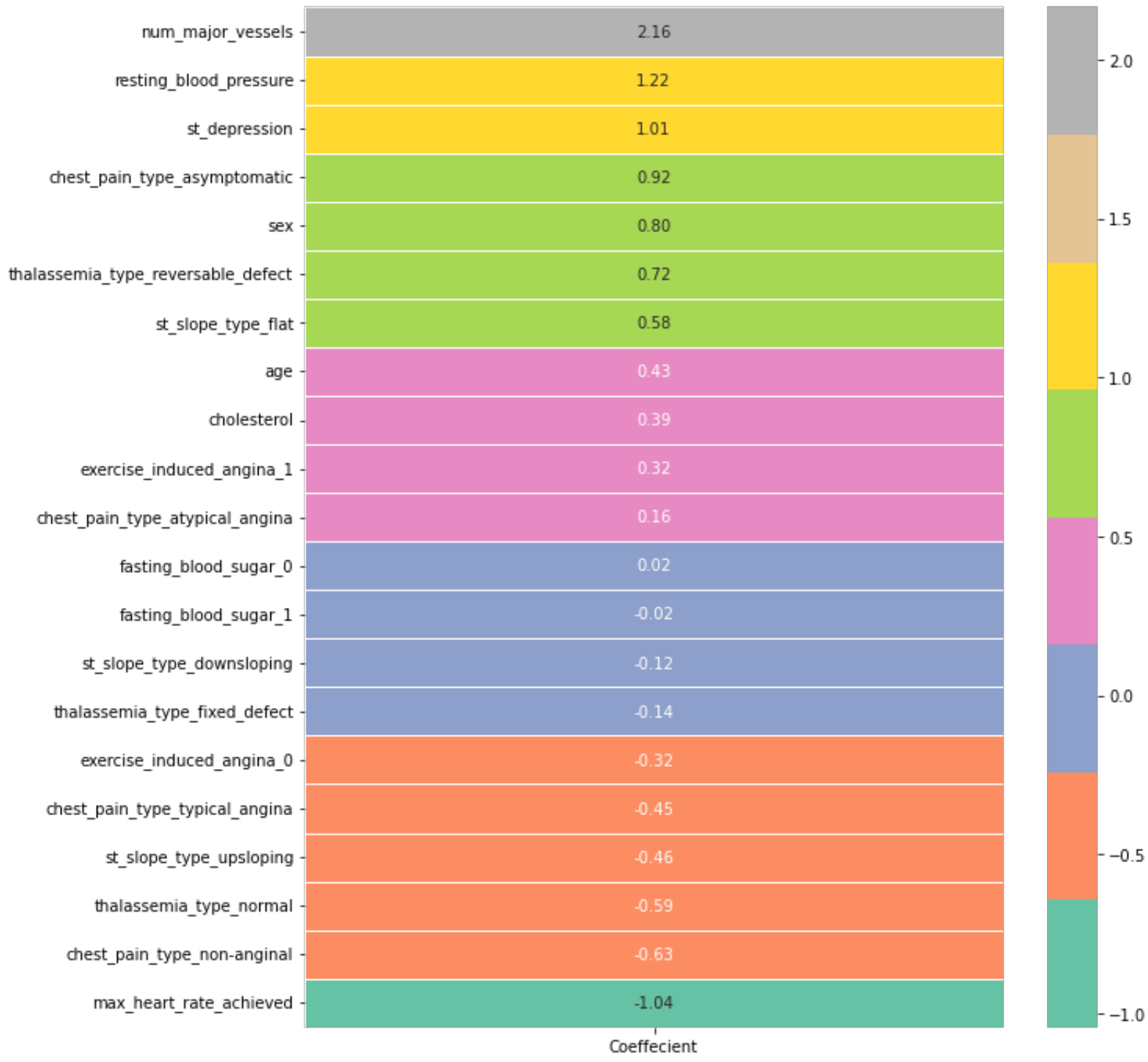
```
<Figure size 720x864 with 0 Axes>
```

Conclusion

1. The Area under the ROC curve is 92% which is somewhat satisfactory.
2. The model predicted with 93% accuracy. The model is more specific than sensitive.
3. According to this model the major features contributing in precision of predicting model
4. are shown in the heatmap in Ascending order.

In [117]:
```
plt.figure(figsize=(10,12))
coeffecients = pd.DataFrame(logre.coef_.ravel(),X.columns)
coeffecients.columns = ['Coeffecient']
coeffecients.sort_values(by=['Coeffecient'],inplace=True,ascending=False)
sns.heatmap(coeffecients,annot=True,fmt='.2f',cmap='Set2',linewidths=0.5)
```

Out[117]: <AxesSubplot:>



In [ ]: