IDS-Report

November 30, 2020

1 Introduction to Data Science (CSE 327) project report on

Application of Machine Learning Classification algorithms on heart failure clinical records data

Project's team members :

- Abhigyan Agarwala - 18ucc035
- Eshaan Rathi - 18ucc089
- Mradul Bohra - 18ucc011
- Kartikey Sharma - 18ucc158

link to code:
https://colab.research.google.com/drive/1fkbDp6XX-vJtpG8BXNVuEdlLKtxog9i8?usp=sharing

2 Structure of work:

- Load dataset and libraries in environment
- First look at the dataset
- Dataset exploration and visualisation
- Feature selection
- Machine learning modelling and classification
- Conclusions and learning outcomes
- References

3 A brief background of dataset and the project:

The dataset consists of several features on which Heart failure depends, and depending on those features, it tells that whether the person suffered fatality or not. The following work aims to understand the dataset, relation between various features and Death_Event, and use Machine learning algorithm(s) to predict whether a person will die or not, given a set of entries for those attributes.

Dataset source : https://archive.ics.uci.edu/ml/datasets/Heart+failure+clinical+records

4 Load dataset and libraries in environment

[ ]:
```
#We stored the dataset in our Google drive. Will be mounting our G-drive to use␣
 ↪it.
from google.colab import drive
drive.mount('/content/drive')
```

[ ]:
```
import pandas as pd #for dataframe manipulation
import matplotlib.pyplot as plt #for figures
%matplotlib inline
#for figures

import numpy as np #n-d array manipulation

import plotly.express as px #for plots
import plotly.graph_objects as go # for plots
import plotly.io as pio
pio.renderers
pio.renderers.default = "jpg"

import seaborn as sns #for plots

from sklearn import preprocessing #for preprocessing
import statsmodels.api as sm #for feature selection
from sklearn.feature_selection import SelectKBest, f_classif #for feature␣
 ↪selection

#machine learning dependencies:
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, confusion_matrix,␣
 ↪accuracy_score, f1_score, recall_score, precision_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
```

[ ]:
```
#dependencies for ORCA and colab
!pip install plotly>=4.0.0
!wget https://github.com/plotly/orca/releases/download/v1.2.1/orca-1.2.1-x86_64.
 ↪AppImage -O /usr/local/bin/orca
!chmod +x /usr/local/bin/orca
!apt-get install xvfb libgtk2.0-0 libgconf-2-4
```

We now load dataset and store it in a dataframe called "data".

[325]:
```
data = pd.read_csv('/content/drive/MyDrive/ids-project/heart.csv')
```

5 First look at the dataset

Concise summary of the Dataset:

[326]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 299 entries, 0 to 298
Data columns (total 13 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   age                       299 non-null    float64
 1   anaemia                   299 non-null    int64
 2   creatinine_phosphokinase  299 non-null    int64
 3   diabetes                  299 non-null    int64
 4   ejection_fraction         299 non-null    int64
 5   high_blood_pressure       299 non-null    int64
 6   platelets                 299 non-null    float64
 7   serum_creatinine          299 non-null    float64
 8   serum_sodium              299 non-null    int64
 9   sex                       299 non-null    int64
 10  smoking                   299 non-null    int64
 11  time                      299 non-null    int64
 12  DEATH_EVENT               299 non-null    int64
dtypes: float64(3), int64(10)
memory usage: 30.5 KB
```

We observe that there are 299 tuples, each of 13 attributes in the dataset. 10 attributes have data of int64 type and 3 have that of float64 type. Moreover, none of the entries are null.

Meanings, measurement units, and intervals of each feature of the dataset (given by original donors of the dataset) :

[327]:
```
from IPython.display import Image
Image(filename="/content/drive/MyDrive/ids-project/table_meanings.png")
```

[327]:

Feature	Explanation	Measurement	Range
Age	Age of the patient	Years	[40,..., 95]
Anaemia	Decrease of red blood cells or hemoglobin	Boolean	0, 1
High blood pressure	If a patient has hypertension	Boolean	0, 1
Creatinine phosphokinase	Level of the CPK enzyme in the blood	mcg/L	[23,..., 7861]
(CPK)			
Diabetes	If the patient has diabetes	Boolean	0, 1
Ejection fraction	Percentage of blood leaving	Percentage	[14,..., 80]
	the heart at each contraction		
Sex	Woman or man	Binary	0, 1
Platelets	Platelets in the blood	kiloplatelets/mL	[25.01,..., 850.00]
Serum creatinine	Level of creatinine in the blood	mg/dL	[0.50,..., 9.40]
Serum sodium	Level of sodium in the blood	mEq/L	[114,..., 148]
Smoking	If the patient smokes	Boolean	0, 1
Time	Follow-up period	Days	[4,...,285]
(target) death event	If the patient died during the follow-up period	Boolean	0, 1

mcg/L: micrograms per liter. mL: microliter. mEq/L: milliequivalents per litre

Overview of the first 5 records in the dataset:

[328]:
```
data.head()
```

[328]:
```
    age  anaemia  creatinine_phosphokinase  ...  smoking  time  DEATH_EVENT
0  75.0        0                       582  ...        0     4            1
1  55.0        0                      7861  ...        0     6            1
2  65.0        0                       146  ...        1     7            1
3  50.0        1                       111  ...        0     7            1
4  65.0        1                       160  ...        0     8            1

[5 rows x 13 columns]
```

Overview of the last 5 records in the dataset:

[329]:
```
data.tail()
```

[329]:
```
      age  anaemia  creatinine_phosphokinase  ...  smoking  time  DEATH_EVENT
294  62.0        0                        61  ...        1   270            0
295  55.0        0                      1820  ...        0   271            0
296  45.0        0                      2060  ...        0   278            0
297  45.0        0                      2413  ...        1   280            0
298  50.0        0                       196  ...        1   285            0

[5 rows x 13 columns]
```

Descriptive statistics of data: (i.e. count, central tendency, quartiles, max and min)

[330]:
```
data.describe()
```

[330]:
```
                age     anaemia  ...        time  DEATH_EVENT
count   299.000000  299.000000  ...  299.000000    299.00000
mean     60.833893    0.431438  ...  130.260870      0.32107
std      11.894809    0.496107  ...   77.614208      0.46767
min      40.000000    0.000000  ...    4.000000      0.00000
25%      51.000000    0.000000  ...   73.000000      0.00000
50%      60.000000    0.000000  ...  115.000000      0.00000
75%      70.000000    1.000000  ...  203.000000      1.00000
max      95.000000    1.000000  ...  285.000000      1.00000

[8 rows x 13 columns]
```

Q. Are there any duplicate records (rows)? If found, should they be removed?

They will take space and consume time while running the algorithms later. Hence if found, we will remove them.

[331]:
```
duplicate_rows = data[data.duplicated()]
print("number of duplicate rows: ", duplicate_rows.shape[0])
```

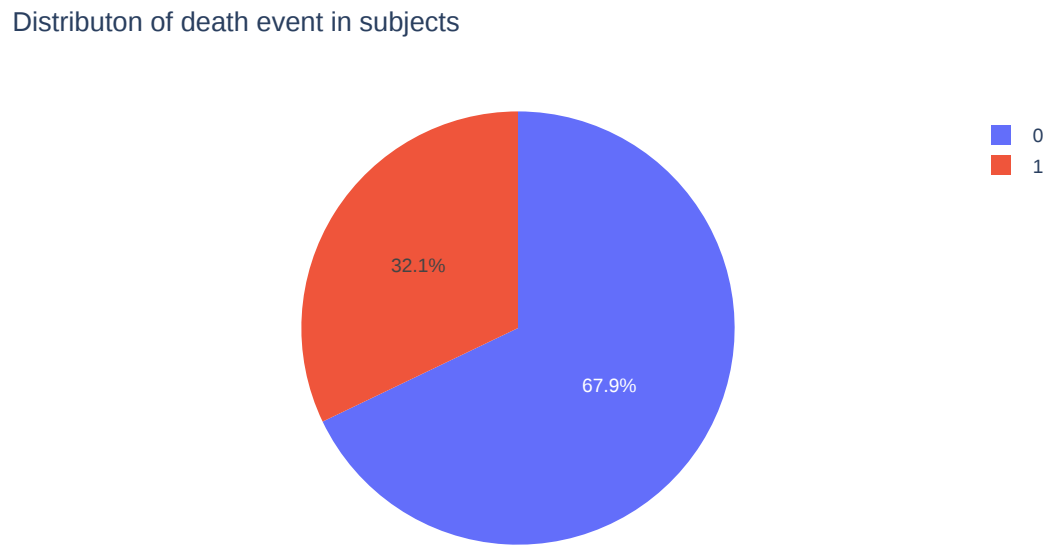
```
number of duplicate rows:  0
```

No duplicate rows were found in the dataset.

6 Dataset exploration and visualisation

Q. What is the ratio of sujects who died to the total subjects evaluated?

[332]:
```
fig = px.pie(data, names='DEATH_EVENT', title='Distributon of death event in␣
 ↪subjects', width = 500, height=500)
fig.show(renderer="svg")
```

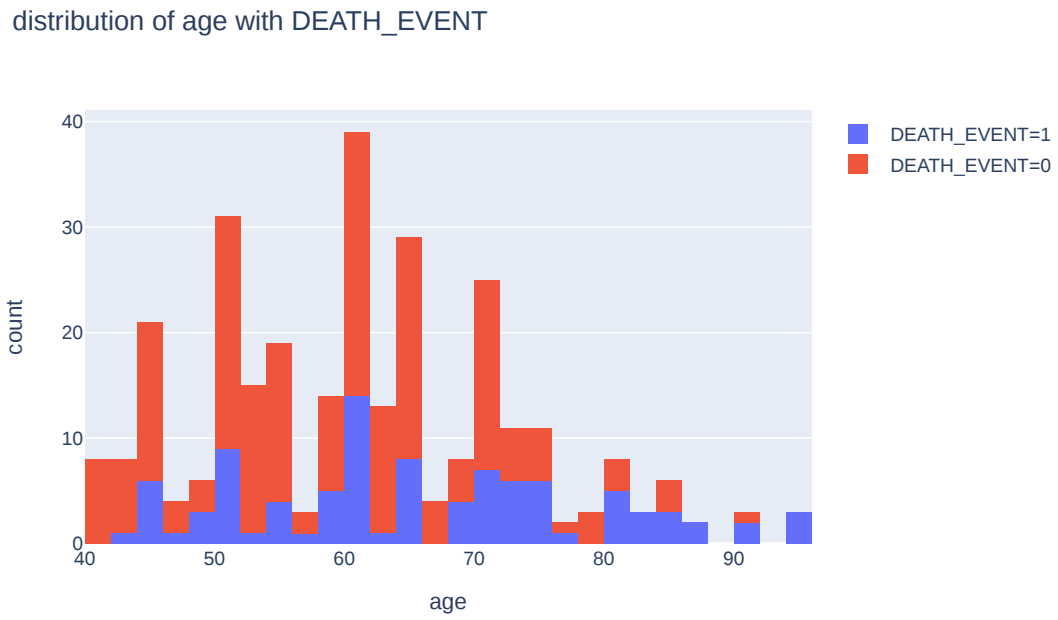


There's an imbalance in the distribution of DEATH_EVENT. Fatal and Non-fatal cases are roughly in 1:2 ratio.

7 Now we look at various features, their inter-relation and impact on target(i.e. DEATH_EVENT)

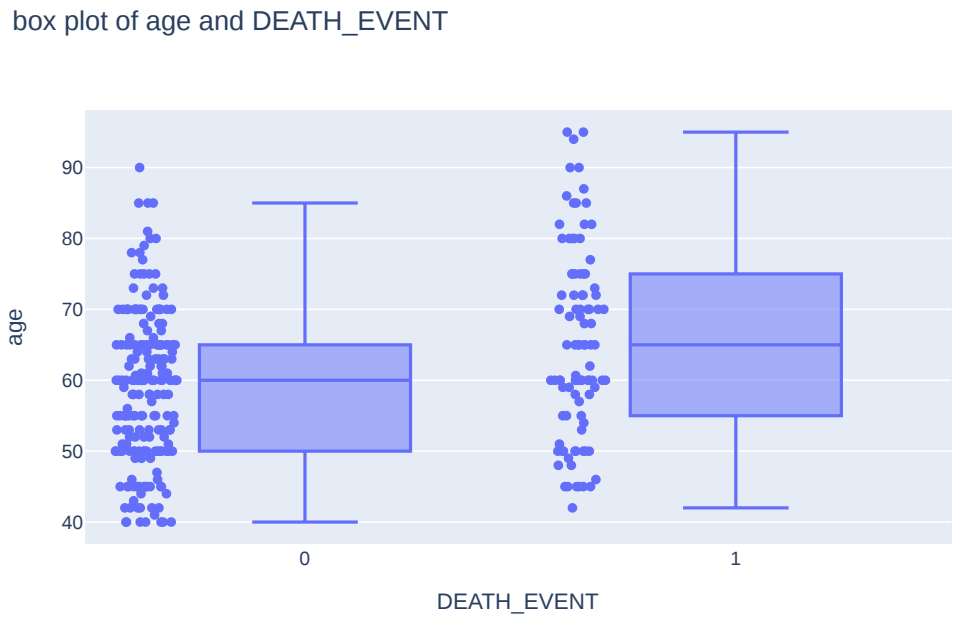
Q. What is the distribution of subjects with their age? How many died at a particular age?

[333]:
```
fig = px.histogram(data, title='distribution of age with DEATH_EVENT', x="age",␣
 ↪color="DEATH_EVENT", nbins = 50,width=800,height=500)
fig.show(renderer="svg")
```



Q. is there any anomaly present in the distribution of age of subjects? If yes, should it be removed?

[334]:
```
fig = px.box(
    data,
    x="DEATH_EVENT",
    y="age",
    points='all',
    title='box plot of age and DEATH_EVENT',
    width=400,
    height=600
)

fig.show(renderer="svg")
```



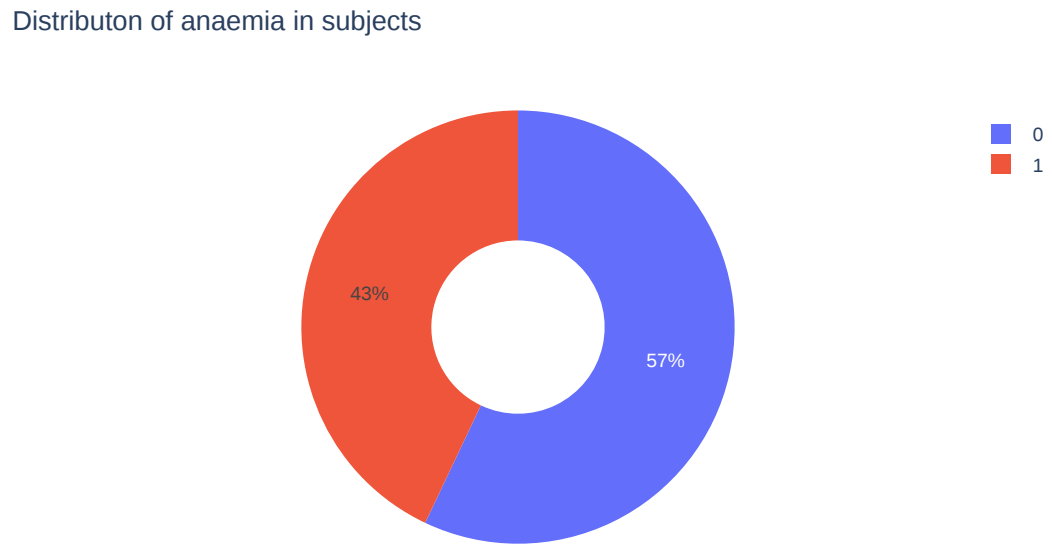
We observe that the quartiles of age for DEATH_EVENT = 1 are higher than DEATH_EVENT = 0. Moreover, it lies away from the cluster of the corresponding observations. We remove this outlier from the dataset.

[335]:
```
print("shape of data frame with anomaly: ", data.shape)
data = data.drop(data[(data['age'] > 85) & (data['DEATH_EVENT'] == 0)].index)
print("shape of data frame after removing anomaly: ", data.shape)
```

```
shape of data frame with anomaly:  (299, 13)
shape of data frame after removing anomaly:  (298, 13)
```

Q. What is the ratio of the subjects who had anaemia?

[336]:
```
fig = px.pie(data, names='anaemia', title='Distributon of anaemia in subjects',␣
 ↪width = 400, height=400, hole = .4)
fig.show(renderer="svg")
```

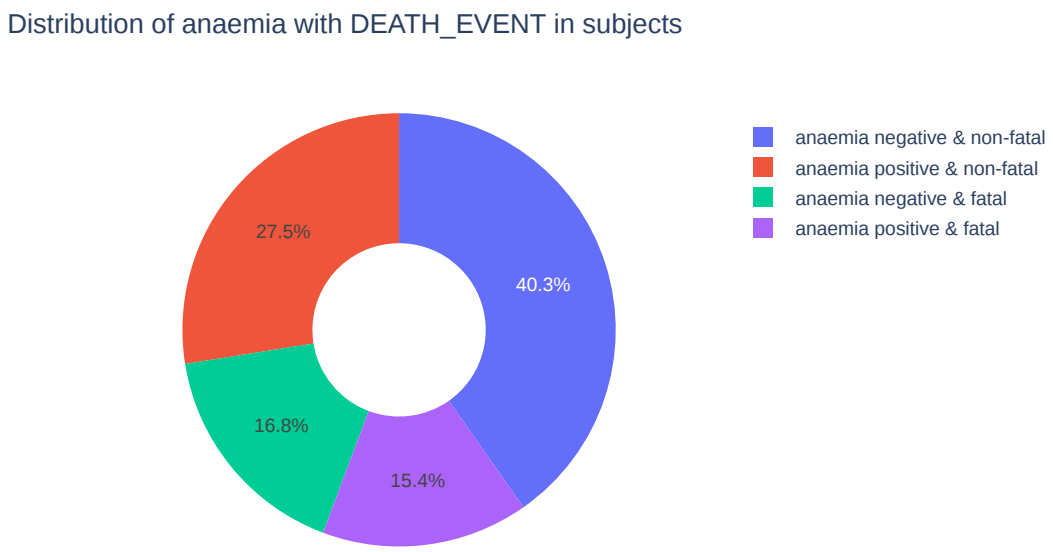


Q. How many died out of those who had anaemia? How many didn't? What can be inferred?

[337]:
```
anaemia_p = data[data["anaemia"]==1]
anaemia_n = data[data["anaemia"]==0]

anaemia_p_non_f = anaemia_p[data["DEATH_EVENT"]==0]
anaemia_p_f = anaemia_p[data["DEATH_EVENT"]==1]
anaemia_n_non_f = anaemia_n[data["DEATH_EVENT"]==0]
anaemia_n_f = anaemia_n[data["DEATH_EVENT"]==1]

VALUES = [len(anaemia_p_non_f), len(anaemia_p_f), len(anaemia_n_non_f),␣
 ↪len(anaemia_n_f)]
LABELS = ["anaemia positive & non-fatal", "anaemia positive & fatal", "anaemia␣
 ↪negative & non-fatal", "anaemia negative & fatal"]

fig = px.pie(data, values=VALUES, names=LABELS, title='Distribution of anaemia␣
 ↪with DEATH_EVENT in subjects', width = 500, height=400, hole = .4)
fig.show(renderer="svg")
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: UserWarning:

Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:5: UserWarning:
```

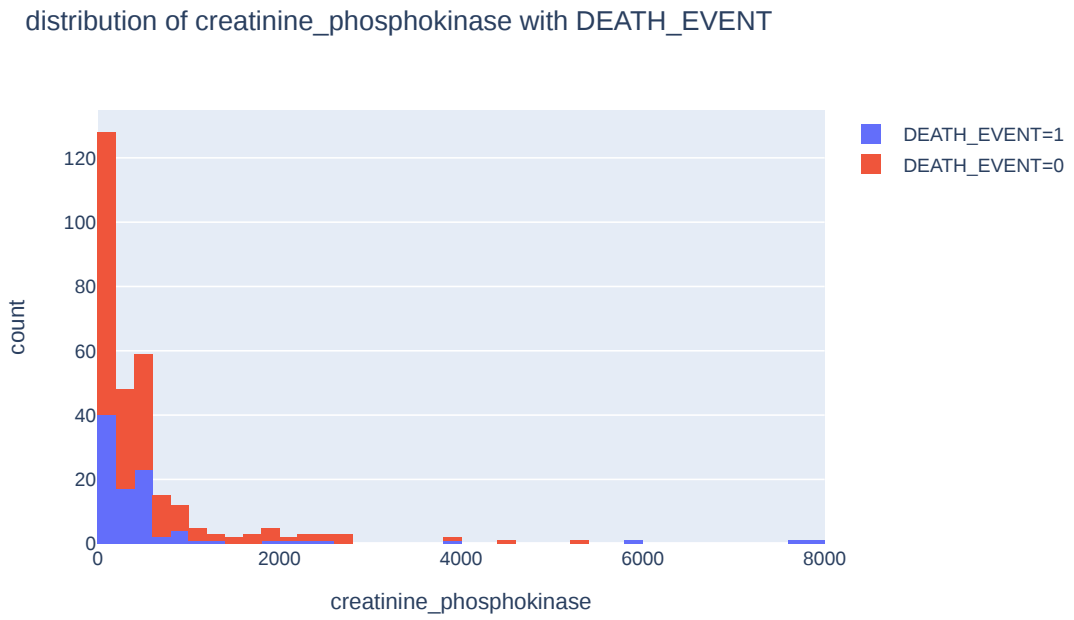
```
Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:6: UserWarning:

Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: UserWarning:

Boolean Series key will be reindexed to match DataFrame index.
```



We observe that fatality vs non-fatality ratio was found to be slightly higher in anaemia postive subjects than anaemia negative subjects. Therefore, anaemia can be a good feature for classifying DEATH_EVENT. We will verify this later.

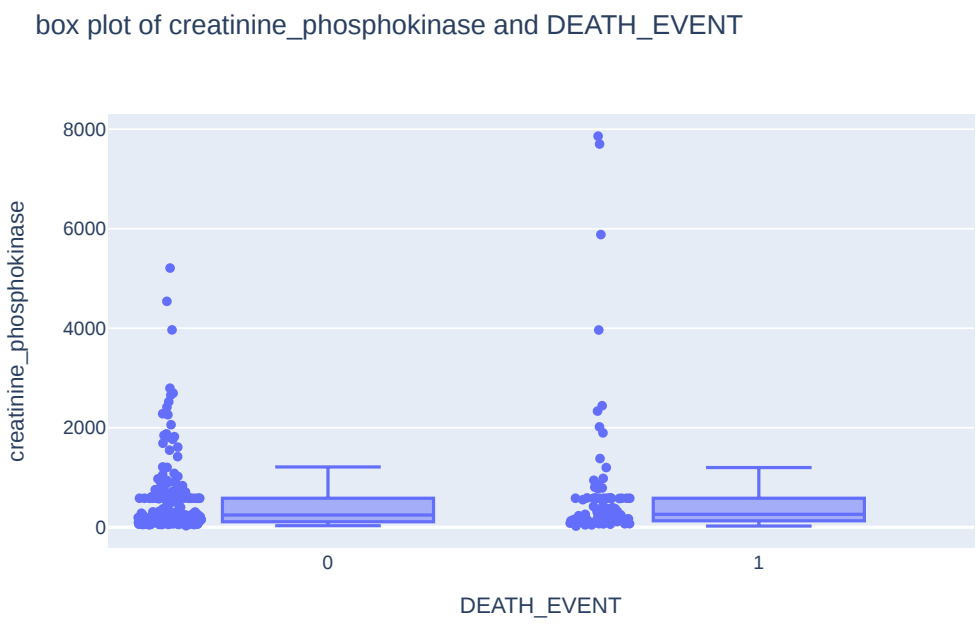
Q. What is the distribution of subjects with creatinine_phosphokinase content? How many died at a particular level of creatinine_phosphokinase content?

[338]:
```
fig = px.histogram(data, title='distribution of creatinine_phosphokinase with␣
↪DEATH_EVENT', x="creatinine_phosphokinase", color="DEATH_EVENT", nbins=50,␣
↪width=1000,height=500)
fig.show(renderer="svg")
```



Q. is there any anomaly present in the distribution of creatinine_phosphokinase in subjects? If yes, should it be removed?

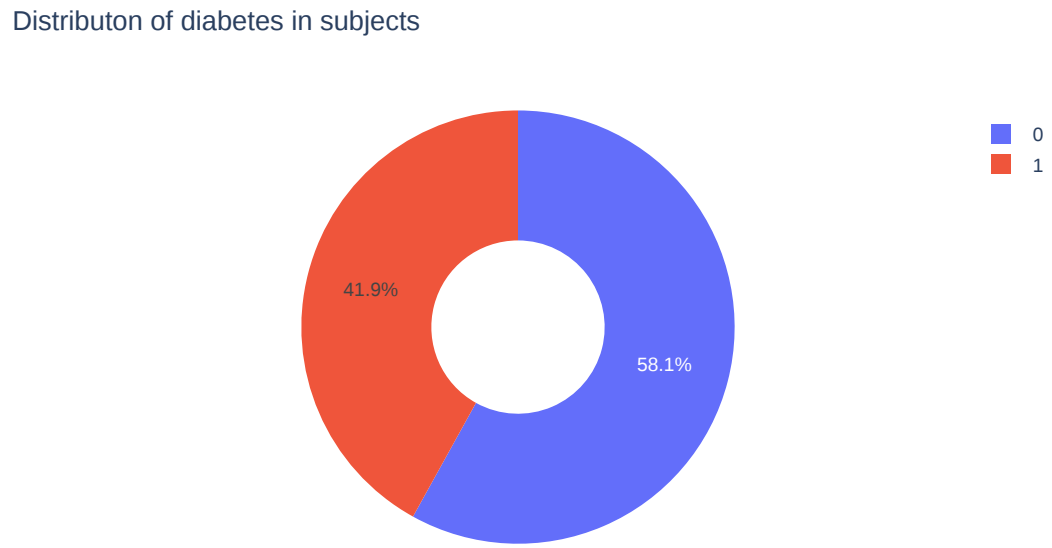
[339]:
```
fig = px.box(
    data,
    x="DEATH_EVENT",
    y="creatinine_phosphokinase",
    points='all',
    title='box plot of creatinine_phosphokinase and DEATH_EVENT',
    width=600,
    height=600
)

fig.show(renderer="svg")
```



We observe outliers in DEATH_EVENT = 1 and creatinine_phosphokinase > 7000, but we don't have robust reason to remove them. It's possible that they might tell us something critical about data. We leave them untouched.

Q. What is the ratio of the subjects who had diabetes?

[340]:
```
fig = px.pie(data, names='diabetes', title='Distributon of diabetes in␣
 ↪subjects', width = 400, height=400, hole = .4)
fig.show(renderer="svg")
```

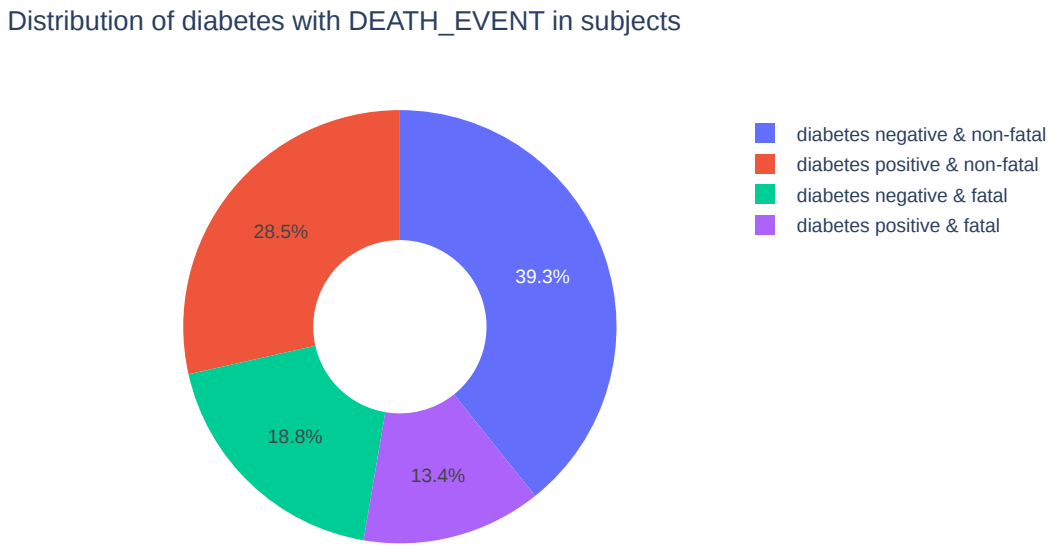


Q. How many died out of those who had diabetes? How many didn't? What can be inferred?

[341]:
```
p = data[data["diabetes"]==1]
n = data[data["diabetes"]==0]

p_non_f = p[data["DEATH_EVENT"]==0]
p_f = p[data["DEATH_EVENT"]==1]
n_non_f = n[data["DEATH_EVENT"]==0]
n_f = n[data["DEATH_EVENT"]==1]

VALUES = [len(p_non_f), len(p_f), len(n_non_f), len(n_f)]
LABELS = ["diabetes positive & non-fatal", "diabetes positive & fatal",␣
 ↪"diabetes negative & non-fatal", "diabetes negative & fatal"]

fig = px.pie(data, values=VALUES, names=LABELS, title='Distribution of diabetes␣
 ↪with DEATH_EVENT in subjects', width = 500, height=400, hole = .4)
fig.show(renderer="svg")
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: UserWarning:

Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:5: UserWarning:

Boolean Series key will be reindexed to match DataFrame index.
```

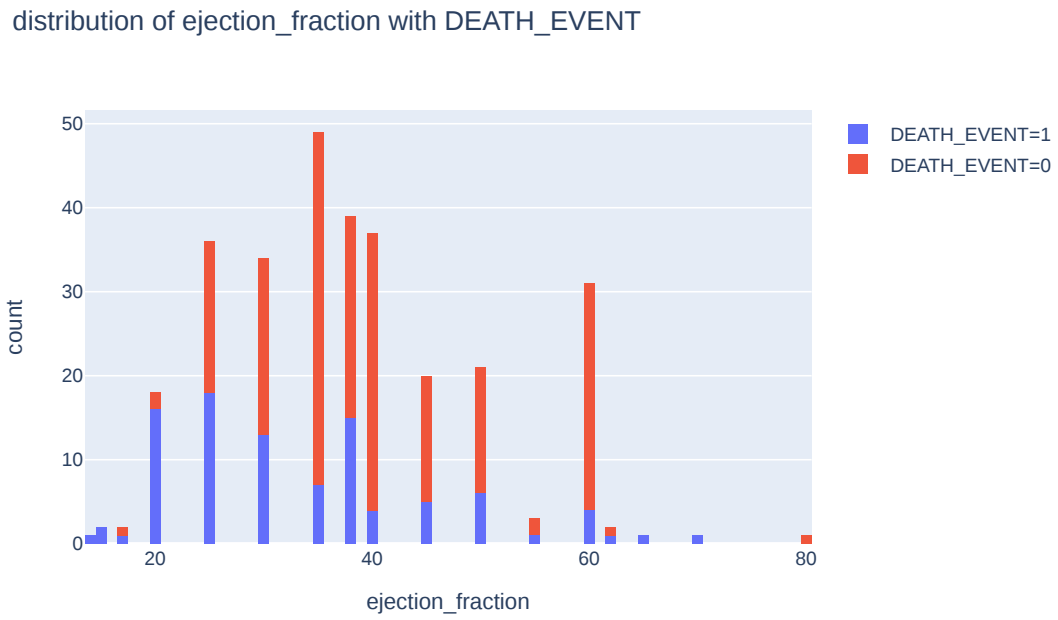
```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:6: UserWarning:

Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: UserWarning:

Boolean Series key will be reindexed to match DataFrame index.
```



We observe that fatality vs non-fatality ratio was found to be similar in both the types of subjects, i.e. diabetes postive and dibetes negative.(roughly diving fatal and non-fatal subjects in 1:2 ratio in both). Therefore, diabetes doesn't seem to be a good feature for classifying DEATH_EVENT.

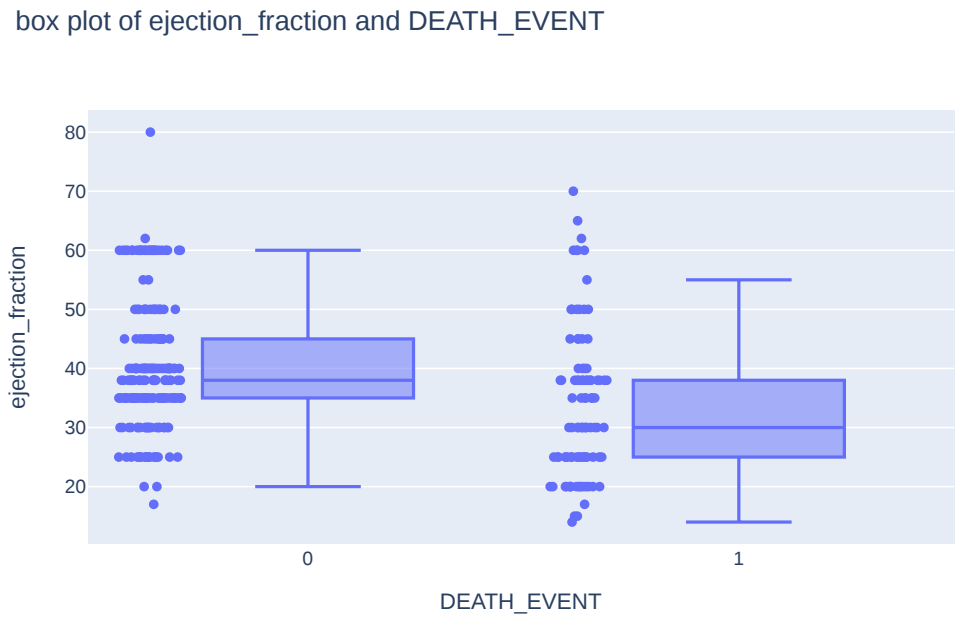
Q. What is the distribution of subjects with content of ejection_fraction? How many died at a particular level of ejection_fraction content?

[342]:
```
fig = px.histogram(data, title='distribution of ejection_fraction with␣
 ↪DEATH_EVENT', x="ejection_fraction", color="DEATH_EVENT", nbins=80,␣
 ↪width=1000,height=500)
fig.show(renderer="svg")
```



Q. is there any anomaly present in the distribution of ejection_fraction in subjects? If yes, should it be removed?

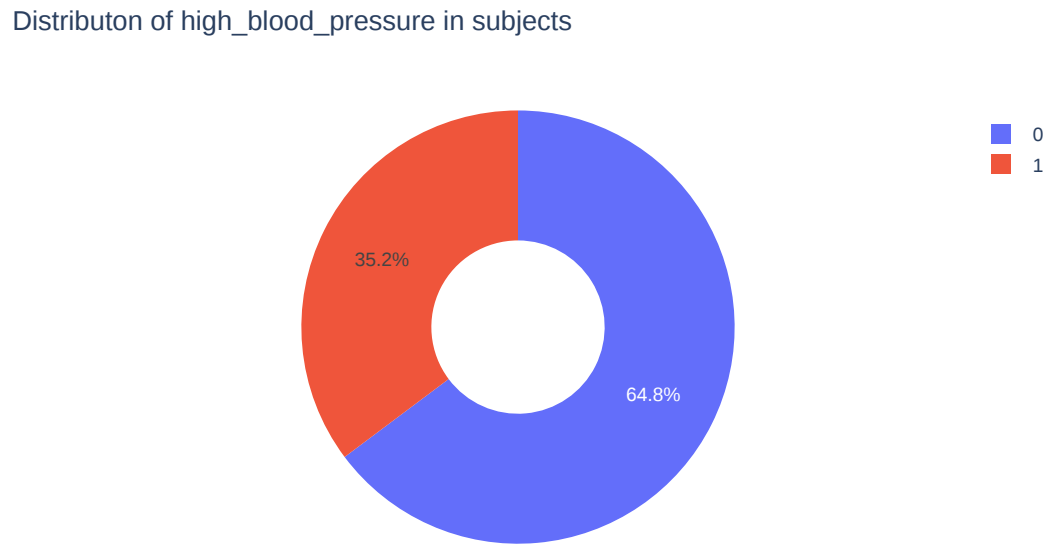
[343]:
```
fig = px.box(
    data,
    x="DEATH_EVENT",
    y="ejection_fraction",
    points='all',
    title='box plot of ejection_fraction and DEATH_EVENT',
    width=600,
    height=600
)

fig.show(renderer="svg")
```



We observe that the ejection_fraction's Median, Q1 and Q3 are higher when Death_EVENT is 0. The record with ejection_fraction = 80 and DEATH_EVENT = 0 is is in accordance to the above observation inspite of being away from it's corresponding cluster. We don't have robust reasoning to remove this record. We keep it untouched.

Q. What is the ratio of the subjects who had high blood pressure?

[344]:
```
#high_blood_pressure
fig = px.pie(data, names='high_blood_pressure', title='Distributon of␣
 ↪high_blood_pressure in subjects', width = 400, height=400, hole = .4)
fig.show(renderer="svg")
```

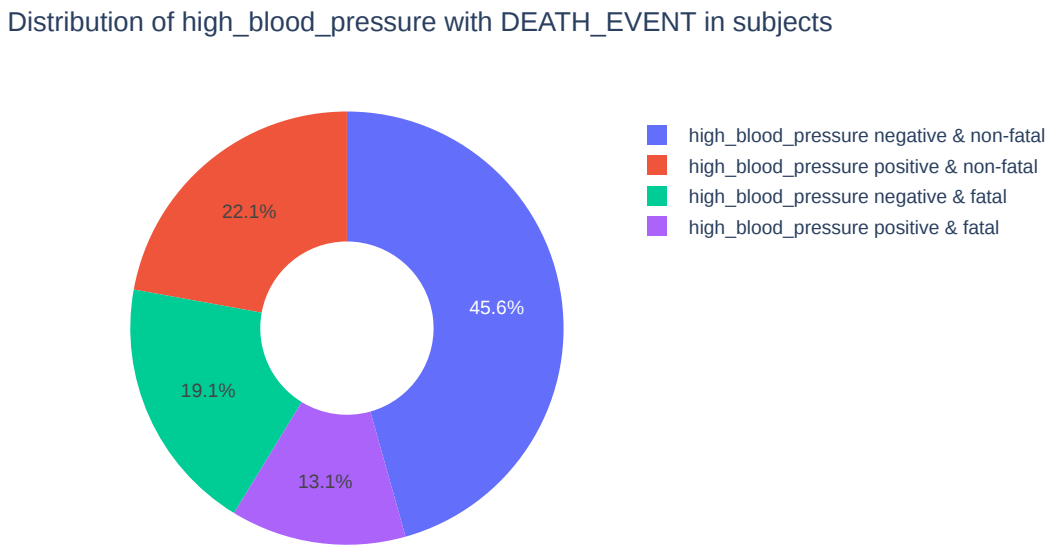


Q. How many died out of those who had high blood pressure? How many didn't? What can be inferred?

[345]:
```
p = data[data["high_blood_pressure"]==1]
n = data[data["high_blood_pressure"]==0]

p_non_f = p[data["DEATH_EVENT"]==0]
p_f = p[data["DEATH_EVENT"]==1]
n_non_f = n[data["DEATH_EVENT"]==0]
n_f = n[data["DEATH_EVENT"]==1]

VALUES = [len(p_non_f), len(p_f), len(n_non_f), len(n_f)]
LABELS = ["high_blood_pressure positive & non-fatal", "high_blood_pressure␣
 ↪positive & fatal", "high_blood_pressure negative & non-fatal",␣
 ↪"high_blood_pressure negative & fatal"]

fig = px.pie(data, values=VALUES, names=LABELS, title='Distribution of␣
 ↪high_blood_pressure with DEATH_EVENT in subjects', width = 600, height=400,␣
 ↪hole = .4)
fig.show(renderer="svg")
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: UserWarning:

Boolean Series key will be reindexed to match DataFrame index.
```

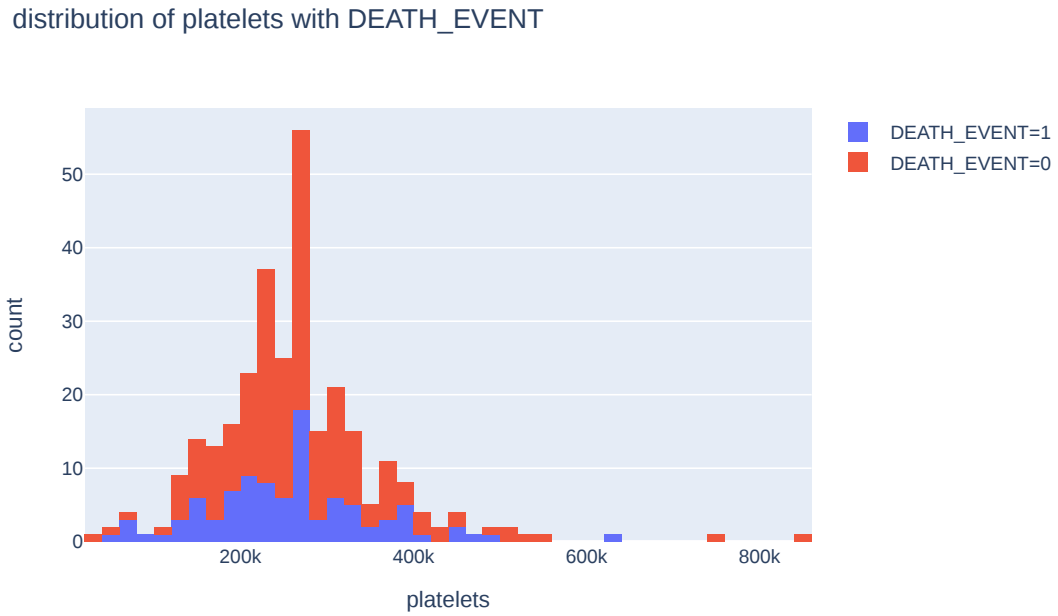
```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:5: UserWarning:

Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:6: UserWarning:

Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: UserWarning:

Boolean Series key will be reindexed to match DataFrame index.
```



We observe that fatality vs non-fatality ratio was found to be slightly higher in high_blood_pressure postive subjects than high_blood_pressure negative subjects. Therefore, anaemia can be a good feature for classifying DEATH_EVENT. We will verify this later.

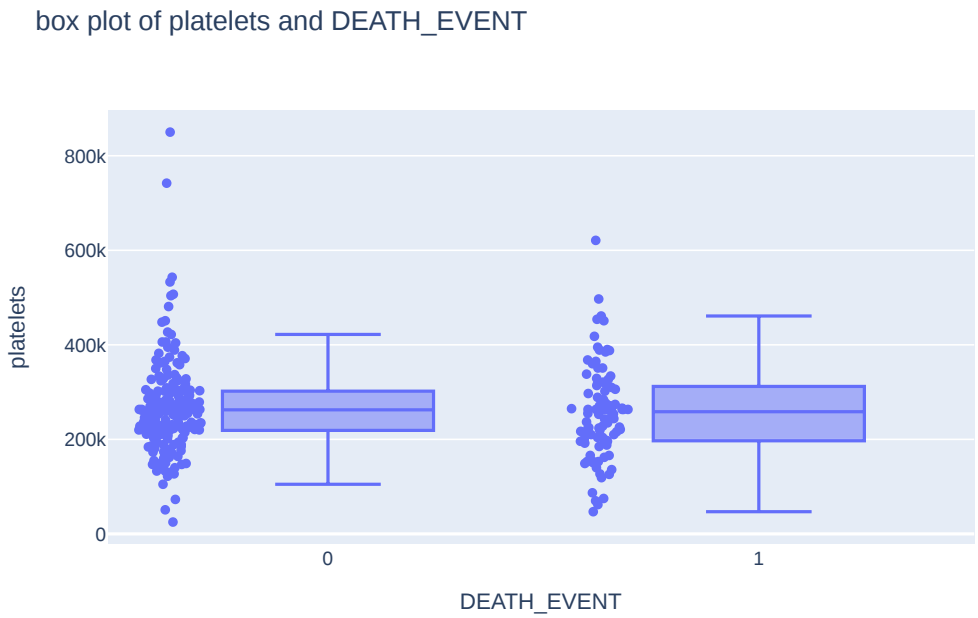
Q. What is the distribution of subjects with their platelet content? How many died at a particular level of platelet content?

[346]:
```
#platelets
fig = px.histogram(data, title='distribution of platelets with DEATH_EVENT',␣
 ↪x="platelets", color="DEATH_EVENT", nbins=80, width=1000,height=400)
fig.show(renderer="svg")
```



Q. Is there any anomaly present in the distribution of platelet content in subjects? If yes, should it be removed?

[347]:
```
fig = px.box(
    data,
    x="DEATH_EVENT",
    y="platelets",
    points='all',
    title='box plot of platelets and DEATH_EVENT',
    width=500,
    height=600
)

fig.show(renderer="svg")
```



The quartiles of platelets for both DEATH_EVENT categories are almost same. The only difference in the distribution is that, DEATH_EVENT = 0 has a compact distribution (near it's box) than DEATH_EVENT = 1.

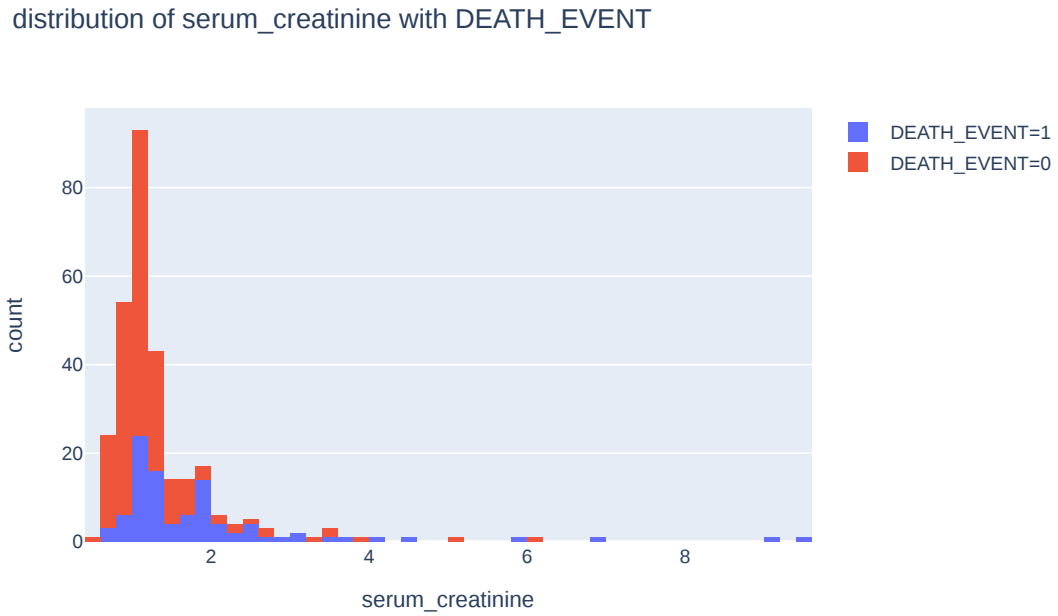
Clearly, the records with plateletsn > 700000 and DEATH_EVENT = 0 are anomalies. We can safely remove them. Similarly for the record with platelets > 600000 and DEATH_EVENT = 1.

[348]:
```
print("shape of data frame with anomaly: ", data.shape)
data = data.drop(data[(data['platelets'] > 700000) & (data['DEATH_EVENT'] ==␣
 ↪0)].index)
data = data.drop(data[(data['platelets'] > 600000) & (data['DEATH_EVENT'] ==␣
 ↪1)].index)
print("shape of data frame after removing anomaly: ", data.shape)
```

```
shape of data frame with anomaly:  (298, 13)
shape of data frame after removing anomaly:  (295, 13)
```

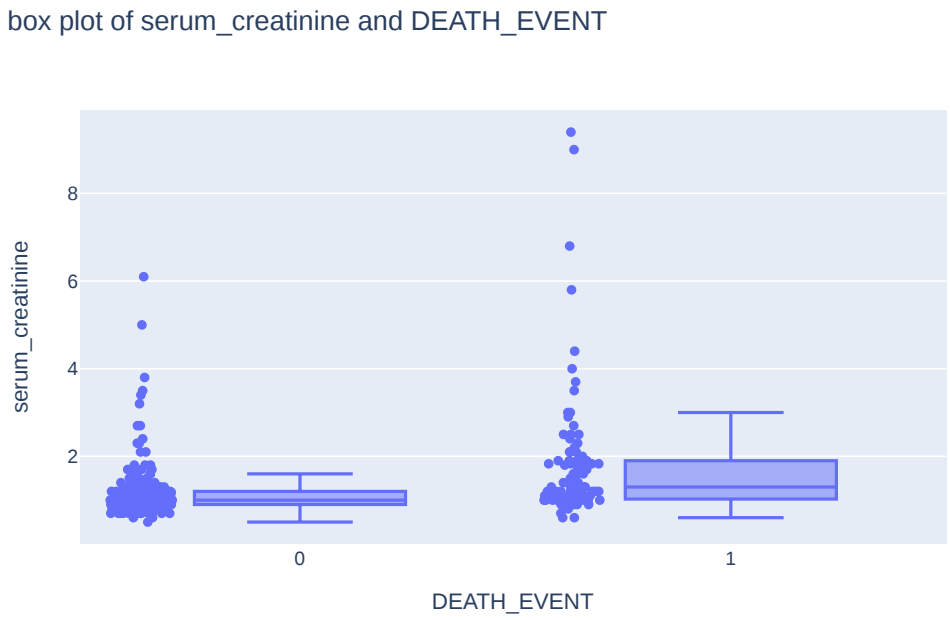
Q. What is the distribution of subjects with their serum_creatinine content? How many died at a particular level of serum_creatinine content?

[349]:
```
#serum_creatinine
fig = px.histogram(data, title='distribution of serum_creatinine with␣
 ↪DEATH_EVENT', x="serum_creatinine", color="DEATH_EVENT", nbins=50,␣
 ↪width=1000,height=400)
fig.show(renderer="svg")
```



Q. Is there any anomaly present in the distribution of serum_creatinine content in subjects? If yes, should it be removed?

[350]:
```
fig = px.box(
    data,
    x="DEATH_EVENT",
    y="serum_creatinine",
    points='all',
    title='box plot of serum_creatinine and DEATH_EVENT',
    width=500,
    height=500
)

fig.show(renderer="svg")
```



The box for DEATH_EVENT = 1 is more wider than that of DEATH_EVENT = 0 and has higher median and upper fences.

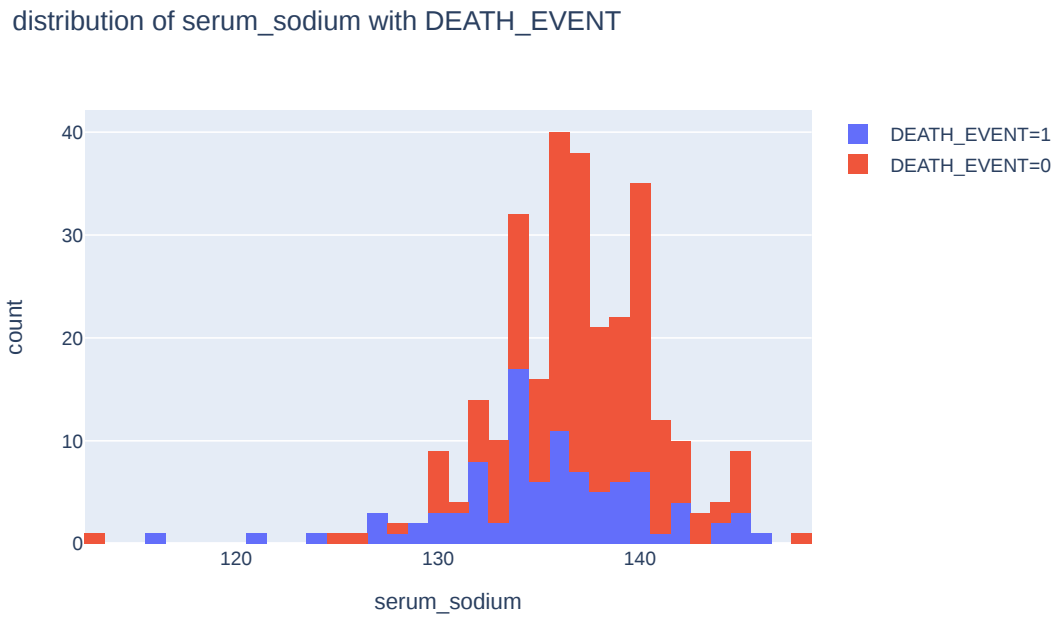
The record with serum_creatinine = 9.4 and DEATH_EVENT = 1 is in accordance to the above rule inspite of being abnormally away from it's cluster. We don't have enough reasoning to remove this record. Whereas the record with serum_creatinine = 6.1 and DEATH_EVENT = 0 is anomalous. We remove it.

[351]:
```
print("shape of data frame with anomaly: ", data.shape)
data = data.drop(data[(data['serum_creatinine'] > 6) & (data['DEATH_EVENT'] ==␣
 ↪0)].index)
print("shape of data frame after removing anomaly: ", data.shape)
```

```
shape of data frame with anomaly:  (295, 13)
shape of data frame after removing anomaly:  (294, 13)
```

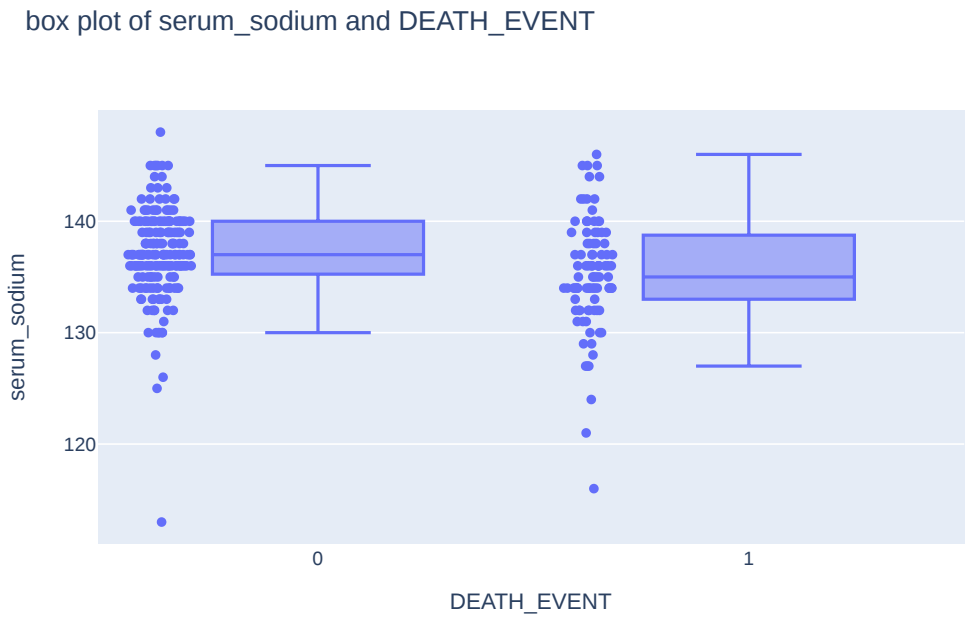
Q. What is the distribution of subjects with their serum_sodium content? How many died at a particular level of serum_sodium content?

[352]:
```
#serum_sodium
fig = px.histogram(data, title='distribution of serum_sodium with DEATH_EVENT',␣
 ↪x="serum_sodium", color="DEATH_EVENT", nbins=50, width=1000,height=400)
fig.show(renderer="svg")
```



Q. Is there any anomaly present in the distribution of serum_sodium content in subjects? If yes, should it be removed?

[353]:
```
fig = px.box(
    data,
    x="DEATH_EVENT",
    y="serum_sodium",
    points='all',
    title='box plot of serum_sodium and DEATH_EVENT',
    width=500,
    height=500
)

fig.show(renderer="svg")
```



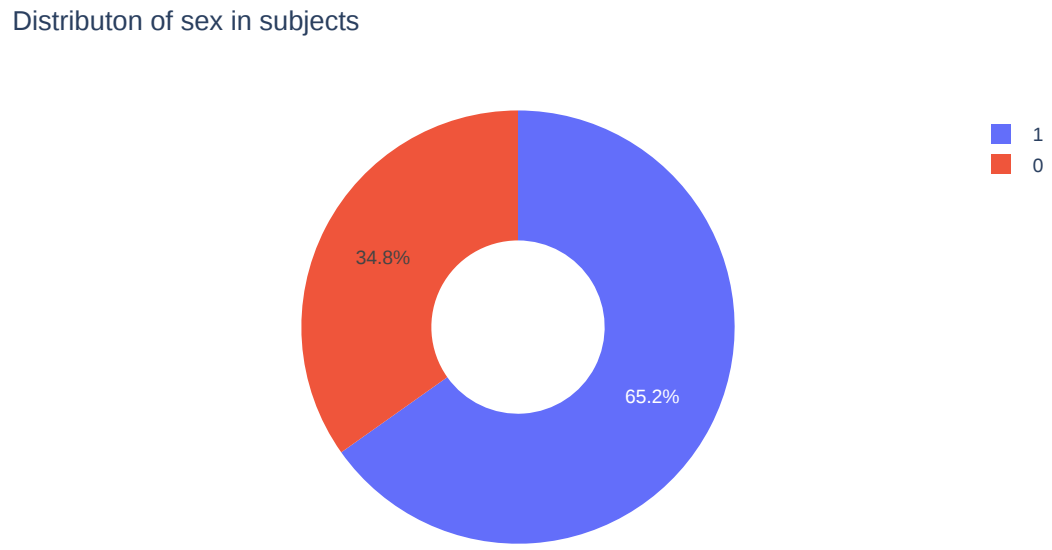
We observe that, cluster for serum_sodium has higher values for DEATH_EVENT = 0 than DEATH_EVENT = 1. But The record with serum_sodium = 113 and DEATH_EVENT = 0 is clearly violating the above trend and it is also very far away from its cluster. We remove this anomaly.

[354]:
```
print("shape of data frame with anomaly: ", data.shape)
data = data.drop(data[(data['serum_sodium'] < 115) & (data['DEATH_EVENT'] ==␣
 ↪0)].index)
print("shape of data frame after removing anomaly: ", data.shape)
```

```
shape of data frame with anomaly:  (294, 13)
shape of data frame after removing anomaly:  (293, 13)
```

Q. What is the ratio of the subjects who are male (or female)?

[355]:
```
#sex
fig = px.pie(data, names='sex', title='Distributon of sex in subjects', width =␣
 ↪400, height=400, hole = .4)
fig.show(renderer="svg")
```

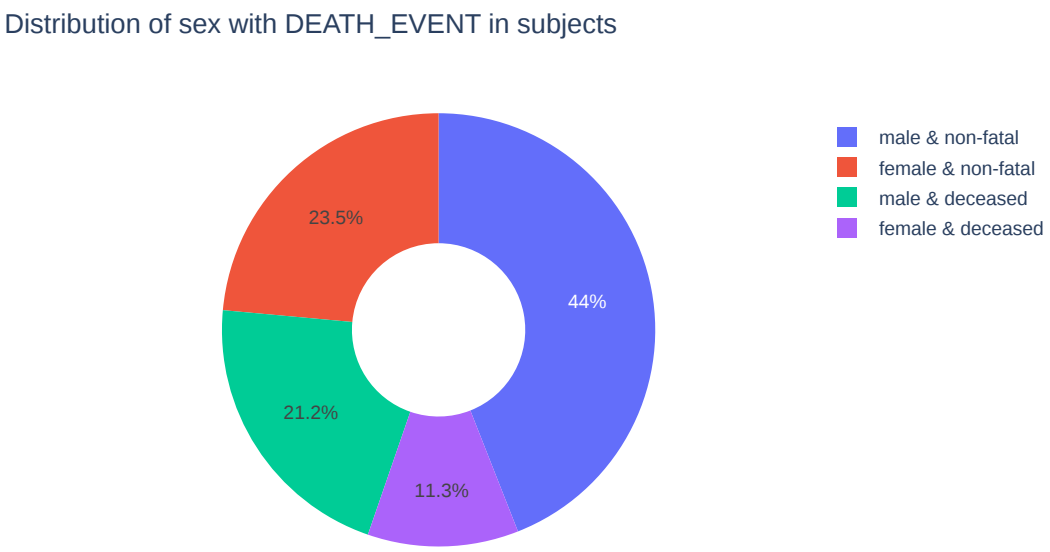


Q. How many died out of those who were male (or female)? How many didn't? What can be inferred?

[356]:
```
p = data[data["sex"]==1]
n = data[data["sex"]==0]

p_non_f = p[data["DEATH_EVENT"]==0]
p_f = p[data["DEATH_EVENT"]==1]
n_non_f = n[data["DEATH_EVENT"]==0]
n_f = n[data["DEATH_EVENT"]==1]

VALUES = [len(p_non_f), len(p_f), len(n_non_f), len(n_f)]
LABELS = ["male & non-fatal", "male & deceased", "female & non-fatal", "female␣
 ↪& deceased"]

fig = px.pie(data, values=VALUES, names=LABELS, title='Distribution of sex with␣
 ↪DEATH_EVENT in subjects', width = 500, height=400, hole = .4)
fig.show(renderer="svg")
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: UserWarning:

Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:5: UserWarning:
```

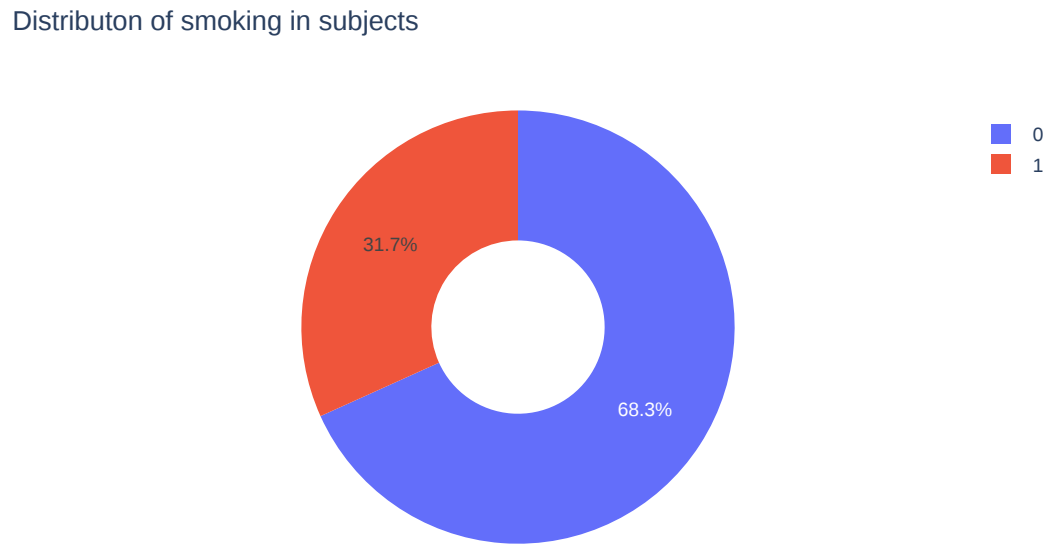
```
Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:6: UserWarning:

Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: UserWarning:

Boolean Series key will be reindexed to match DataFrame index.
```



For male subjects, fatality and non fatality is in 1:2 ratio. This is almost same to that in female subjects. Therefore, sex doesn't seem to be a good feature to classify the DEATH_EVENT.

Q. What is the ratio of the subjects who smoked?

[357]:
```
#smoking
fig = px.pie(data, names='smoking', title='Distributon of smoking in subjects',␣
 ↪width = 500, height=400, hole = .4)
fig.show(renderer="svg")
```

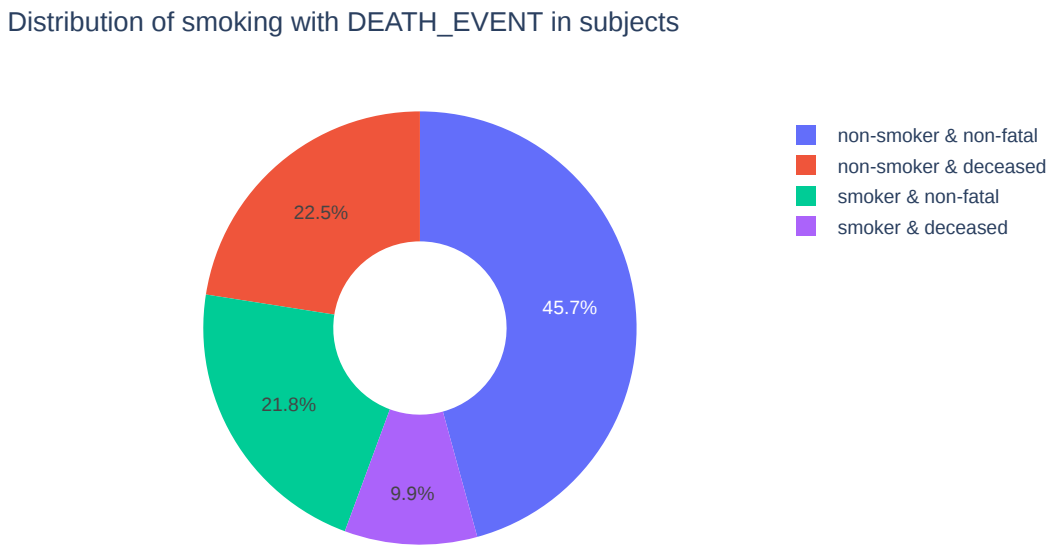


Q. How many died out of those who smoked? How many didn't? What can be inferred?

[358]:
```
p = data[data["smoking"]==1]
n = data[data["smoking"]==0]

p_non_f = p[data["DEATH_EVENT"]==0]
p_f = p[data["DEATH_EVENT"]==1]
n_non_f = n[data["DEATH_EVENT"]==0]
n_f = n[data["DEATH_EVENT"]==1]

VALUES = [len(p_non_f), len(p_f), len(n_non_f), len(n_f)]
LABELS = ["smoker & non-fatal", "smoker & deceased", "non-smoker & non-fatal",␣
 ↪"non-smoker & deceased"]

fig = px.pie(data, values=VALUES, names=LABELS, title='Distribution of smoking␣
 ↪with DEATH_EVENT in subjects', width = 500, height=400, hole = .4)
fig.show(renderer="svg")
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: UserWarning:

Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:5: UserWarning:

Boolean Series key will be reindexed to match DataFrame index.
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:6: UserWarning:

Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: UserWarning:

Boolean Series key will be reindexed to match DataFrame index.
```



For subjects who smoke, fatality and non fatality is in 1:2 ratio. This is almost same to that in non-smoker subjects. Therefore, smoking doesn't seem to be a good feature to classify the DEATH_EVENT.

We drop 'time' attribute, since it signifies time (in days) of death or discharging. But if one has to use model for prediction, he won't be able to provide value of time, since he doesn't know when will subject get discharged in future.

[359]:
```
data = data.drop(['time'], axis=1)
```

Brief overview of data till now

[360]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 293 entries, 0 to 298
Data columns (total 12 columns):
 #   Column                    Non-Null Count  Dtype
```

```
---  ------                    --------------  -----
 0   age                       293 non-null    float64
 1   anaemia                   293 non-null    int64
 2   creatinine_phosphokinase  293 non-null    int64
 3   diabetes                  293 non-null    int64
 4   ejection_fraction         293 non-null    int64
 5   high_blood_pressure       293 non-null    int64
 6   platelets                 293 non-null    float64
 7   serum_creatinine          293 non-null    float64
 8   serum_sodium              293 non-null    int64
 9   sex                       293 non-null    int64
 10  smoking                   293 non-null    int64
 11  DEATH_EVENT               293 non-null    int64
dtypes: float64(3), int64(9)
memory usage: 39.8 KB
```

8 Feature selection

Q. Do we need all features?

No, we must discard (possibly none) those features who have partial (or poor) dependencies on the target (DEATH_EVENT).

Q. Why should we remove them?

Removing useless features would save space, will be computationally cheaper and give higher accuracy while applying classification algorithms.

In the above section we observed histograms, box plots, pie charts to get insights, anomalies and common trends in the data. But we didn't strongly say that whether the feature is good enough to classify DEATH_EVENT correctly or not, because we need more reasoning for it.

We will use stronger technique(s) for that purpose.

Q. Should the features be brought on same scale(range). How?

First we will scale each feature in [0,1] so that we can assess them on same scale. Feature scaling also speeds up the computation while running ML algorithms. We didn't do this before, because we had to visualise how the original data looks.

[361]:
```
cols_to_norm = data.columns[0:11].tolist()
data[cols_to_norm] = data[cols_to_norm].apply(lambda x: (x - x.min()) / (x.
 ↪max() - x.min()))
```

Overview of Data frame after scaling :

[362]:
```
data.head()
```

[362]:

	age	anaemia	creatinine_phosphokinase	...	sex	smoking	DEATH_EVENT
0	0.636364	0.0	0.071319	...	1.0	0.0	1
1	0.272727	0.0	1.000000	...	1.0	0.0	1
2	0.454545	0.0	0.015693	...	1.0	1.0	1
3	0.181818	1.0	0.011227	...	1.0	0.0	1
4	0.454545	1.0	0.017479	...	0.0	0.0	1

[5 rows x 12 columns]

Q. Which features should you remove? How do you choose them?
We employ two methods for selecting features.

1. **Wrapper method** : It uses a greedy approach. It makes a subset of combinations of features from all, runs dummy ML algorithm on them and iteratively (or recursively) selects optimal features us. Performance metric used is pvalue.

2. **Select K best (sklearn)** : It internally uses functions like anova, chi2, mutual_info_regression etc and select features according to the k highest scores. Since we are dealing with numerical data which has continuous values for a field and we have multiple features in the data set, we can use **ANOVA**.

Wrapper method :

[363]:
```
X = data.iloc[:,0:11]
y = data.iloc[:,-1]
cols = list(X.columns)
pmax = 1
while (len(cols)>0):
    p= []
    X_1 = X[cols]
    X_1 = sm.add_constant(X_1)
    model = sm.OLS(y,X_1).fit()
    p = pd.Series(model.pvalues.values[1:],index = cols)
    pmax = max(p)
    feature_max = p.idxmax()
    if(pmax>0.05):
        cols.remove(feature_max)
    else:
        break
print(cols)
```

```
['age', 'ejection_fraction', 'serum_creatinine', 'serum_sodium']
```

Select K best :

[364]:
```
best = SelectKBest(score_func=f_classif, k=10) #f_classif computes the ANOVA␣
↪F-value for the provided sample
fit = best.fit(X,y)

scores = pd.DataFrame(fit.scores_)
col = pd.DataFrame(X.columns)

feature = pd.concat([col,scores],axis=1)
feature.columns = ['Feature','Score']  #naming the dataframe columns
print(feature.nlargest(10,'Score'))
```

```
                  Feature      Score
7           serum_creatinine  32.192078
4          ejection_fraction  21.940906
```

```
0                         age  20.196586
8                serum_sodium  14.723903
2   creatinine_phosphokinase   1.427646
5        high_blood_pressure   1.242954
1                     anaemia   1.089413
6                   platelets   0.598897
10                    smoking   0.095053
3                    diabetes   0.037637
```

From both of the above methods we see that serum_creatinine, ejection_fraction and age are best possible features to classify DEATH_EVENT.

9 Machine learning modelling and classification

Train test split :

[381]:
```
features = ["serum_creatinine", "ejection_fraction", "age"]
x = data[features]
y = data["DEATH_EVENT"]
acc = []
acc_t = []
train_x, test_x, train_y, test_y = train_test_split(x, y, test_size=0.25)
```

Helper function for plotting confusion matrix :

[382]:
```
def plot_cm(y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred)
    ax= plt.subplot()
    sns.heatmap(cm, annot=True, ax=ax)
    ax.set_xlabel('Predicted labels')
    ax.set_ylabel('True labels')
```

Decision tree classifier:

- Q. What?

It is a tree based algorithm which splits the source set, constituting the root node of the tree, into subsets—which constitute the successor children. The splitting is based on a set of splitting rules based on classification features.

- Q. Why?

Because it can solve classification problem. Its easy to employ and one of the foremost ML algorithms which is computationally cheap as well.

- Q.How?

We use sklearn.tree module to build the classifier.

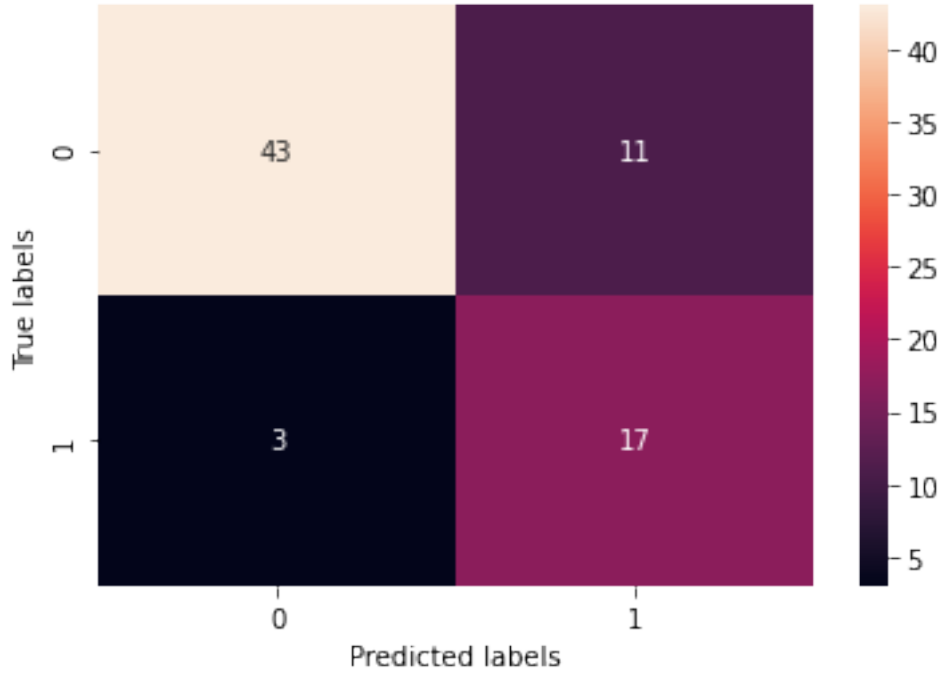
[383]:
```
model = DecisionTreeClassifier(max_depth = 3, criterion='entropy',␣
 ↪max_leaf_nodes=3)
model.fit(train_x, train_y)
```

```
pred = model.predict(test_x)
acc_ = accuracy_score(test_y, pred)*100
acc.append(acc_)
acc_t_ = model.score(train_x, train_y)*100
acc_t.append(acc_t_)
print("train accuracy with Decision Tree Classifier = ", acc_t_)
print("test accuracy with Decision Tree Classifier = ", acc_)
```

```
train accuracy with Decision Tree Classifier =  74.88584474885845
test accuracy with Decision Tree Classifier =  81.08108108108108
```

Confusion matrix for DecisionTreeClassifier :

[384]:
```
plot_cm(test_y, pred)
```



Random forest :

- Q. What?

It employs multiple decision trees and combines them into one ensemble model.

- Q. Why?

Because it can solve classification problem. It does not rely on the feature importance given by a single decision tree, it rather looks into multiple decison trees. Therefore gives better results compared to a decison tree.

- Q.How?

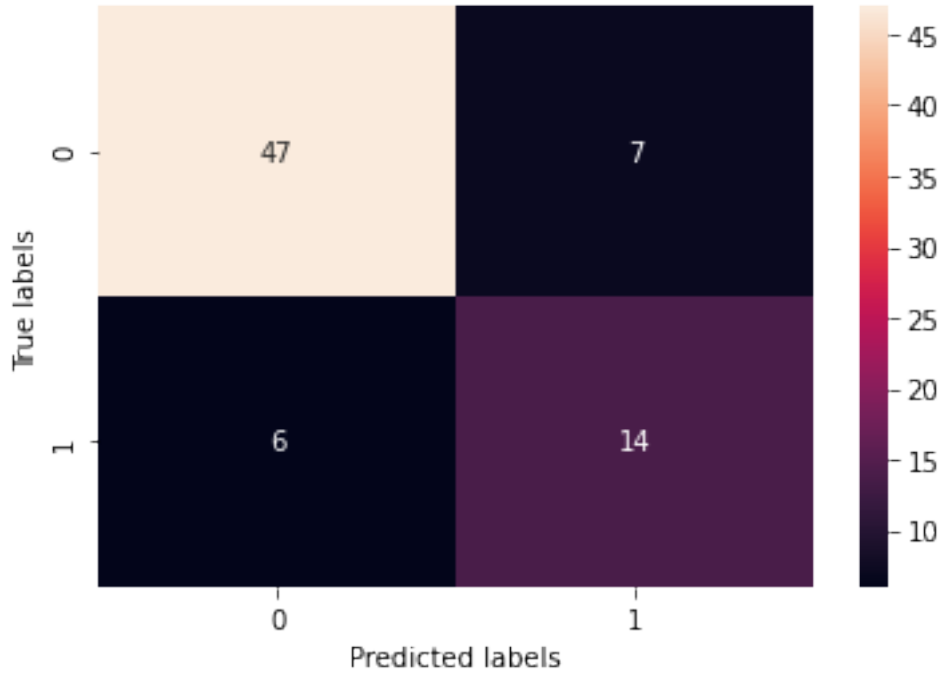
We use sklearn.ensemble class to build the classifier.

[385]:
```
model = RandomForestClassifier()
model.fit(train_x, train_y)
pred = model.predict(test_x)
acc_ = accuracy_score(test_y, pred)*100
acc.append(acc_)
acc_t_ = model.score(train_x, train_y)*100
acc_t.append(acc_t_)
print("train accuracy with Random Forest Classifier = ", acc_t_)
print("test accuracy with Random Forest Classifier = ", acc_)
```

```
train accuracy with Random Forest Classifier =  99.08675799086758
test accuracy with Random Forest Classifier =  82.43243243243244
```

Confusion matrix for Random Forest Classifier

[386]:
```
plot_cm(test_y, pred)
```



K nearest neighbours classifier:

- Q. What?

The algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other.

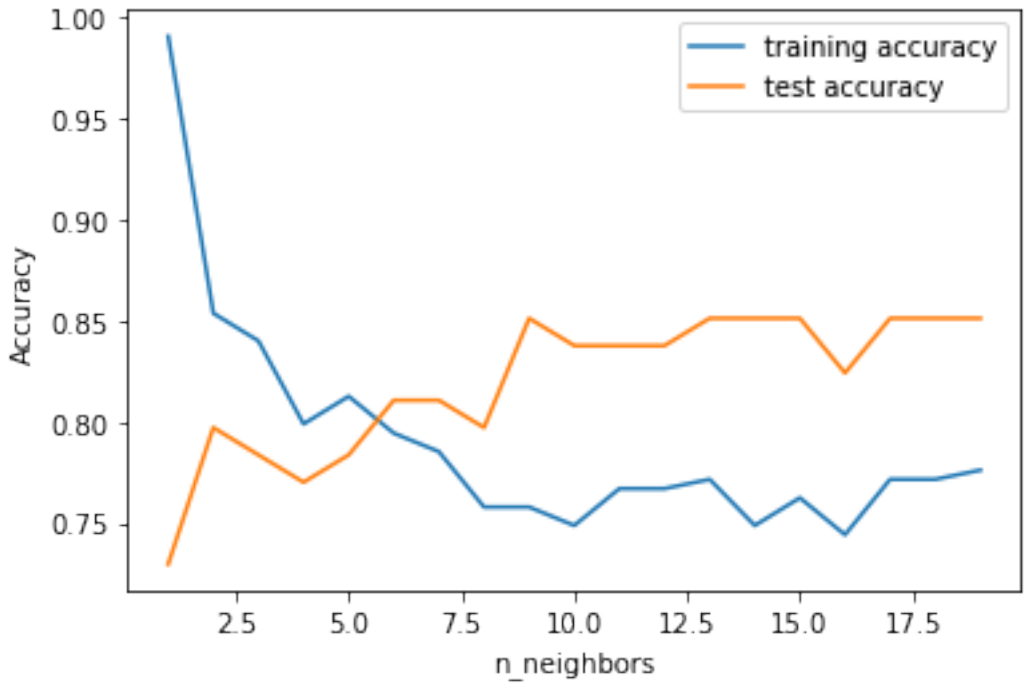
- Q. Why?

Because it can solve classification problem and works good enough for fewer features.

- Q.How?

We use sklearn.neighbors module to build the classifier. We will look for ideal value of k say from [1,20].

[387]:
```
training_accuracy = []
test_accuracy = []
neighbors_settings = range(1, 20)
for n_neighbors in neighbors_settings:
    knn = KNeighborsClassifier(n_neighbors=n_neighbors)
    knn.fit(train_x,train_y)
    training_accuracy.append(knn.score(train_x,train_y))
    test_accuracy.append(knn.score(test_x,test_y))
plt.plot(neighbors_settings, training_accuracy, label="training accuracy")
plt.plot(neighbors_settings, test_accuracy, label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("n_neighbors")
plt.legend()
plt.savefig('knn_compare_model')
```



Accuracies diverge from k = 5, we can choose this value for k , for this split of data.

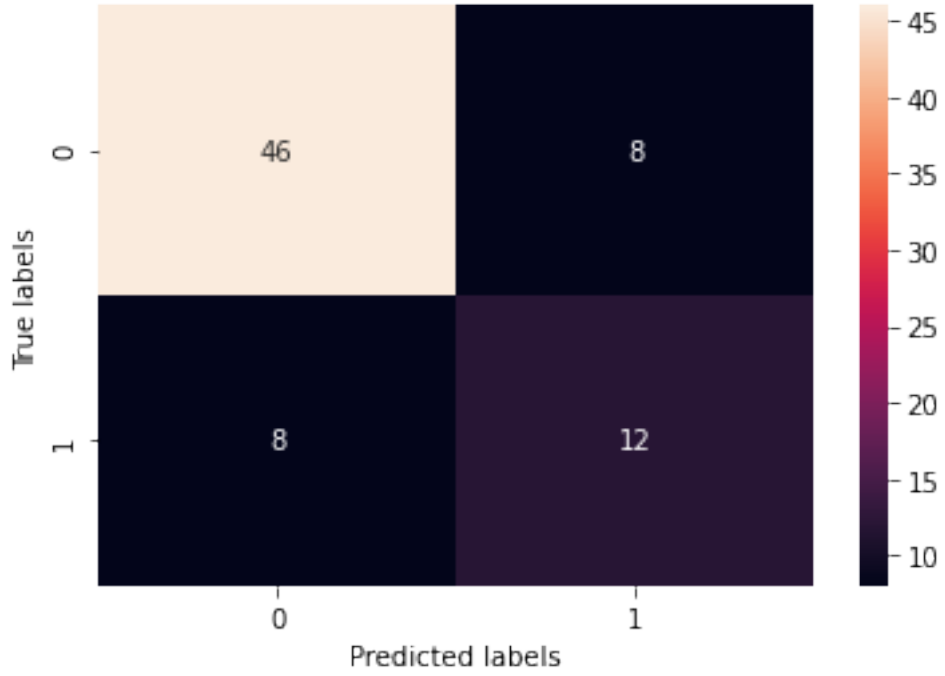
[388]:
```
model = KNeighborsClassifier(n_neighbors=5)
model.fit(train_x, train_y)
pred = model.predict(test_x)
acc_ = accuracy_score(test_y, pred)*100
acc.append(acc_)
```

```
acc_t_ = model.score(train_x, train_y)*100
acc_t.append(acc_t_)
print("train accuracy with K nearest neighbours Classifier = ", acc_t_)
print("test accuracy with K Nearest Neighbors Classifier = ", acc_)
```

```
train accuracy with K nearest neighbours Classifier =  81.27853881278538
test accuracy with K Nearest Neighbors Classifier =  78.37837837837837
```

Confusion matrix for Random Forest Classifier

[389]:
```
plot_cm(test_y, pred)
```



Naive Bayes classifier :

- Q. What? It is a probabilistic classifier based on Bayes' theorem with strong (naïve) independence assumptions between the features.
- Q. Why?

Because it can solve classification problem and works good for small dataset like this and small number of features.

- Q.How?

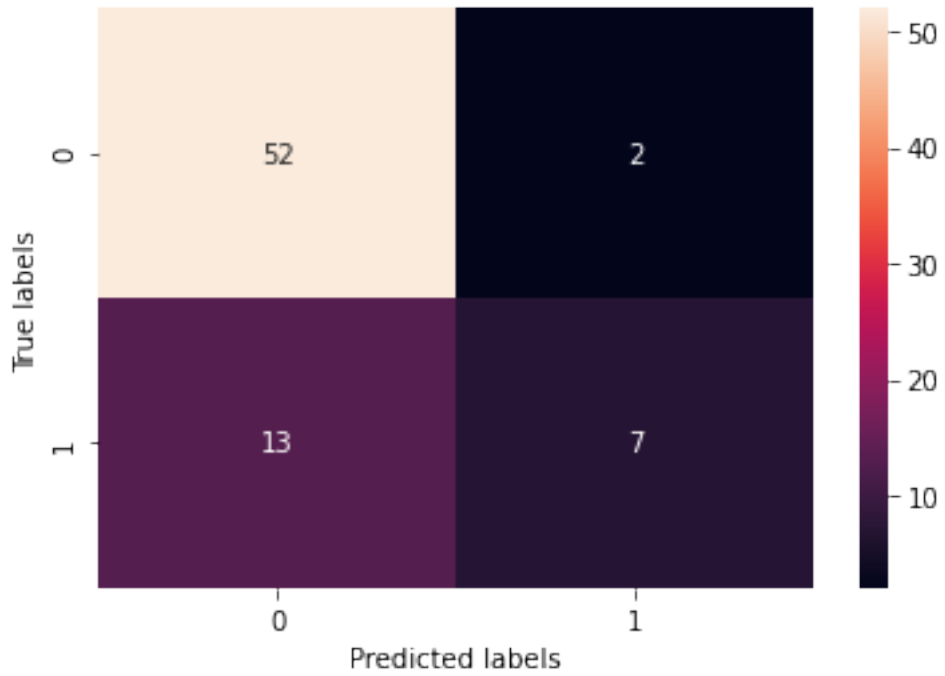
We use sklearn.naive_bayes module to build the classifier.

[390]:
```
model = GaussianNB()
model.fit(train_x, train_y)
pred = model.predict(test_x)
acc_ = accuracy_score(test_y, pred)*100
acc.append(acc_)
acc_t_ = model.score(train_x, train_y)*100
acc_t.append(acc_t_)
print("train accuracy with Naive Bayes = ", acc_t_)
print("test accuracy with Naive Bayes classifier = ", acc_)
```

```
train accuracy with Naive Bayes =  72.14611872146118
test accuracy with Naive Bayes classifier =  79.72972972972973
```

Confusion matrix for naive bayes

[391]:
```
plot_cm(test_y, pred)
```



Support vector machine classification:

- Q. What? This model (linear in nature) is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible.
- Q. Why?

Because it can solve classification problem and works good for numerical data type, like this dataframe.

- Q.How?

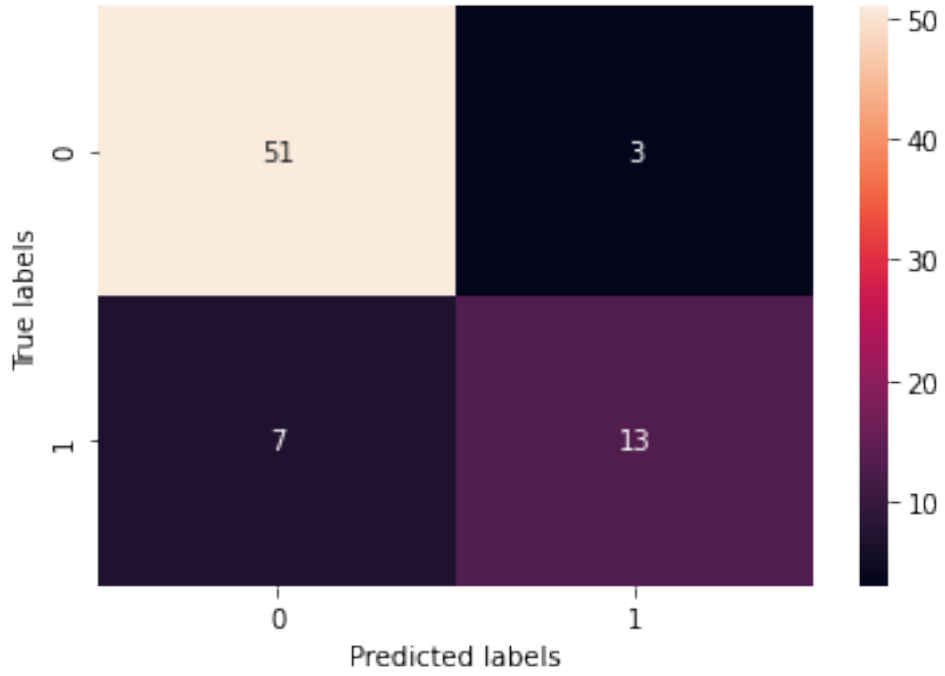
We use sklearn.svm module to build the classifier.

[392]:
```
model = SVC()
model.fit(train_x, train_y)
pred = model.predict(test_x)
acc_ = accuracy_score(test_y, pred)*100
acc.append(acc_)
acc_t_ = model.score(train_x, train_y)*100
acc_t.append(acc_t_)
print("train accuracy with Support vector machine Classifier = ", acc_t_)
print("test accuracy with Support vector machine classifier = ", acc_)
```

```
train accuracy with Support vector machine Classifier =  77.1689497716895
test accuracy with Support vector machine classifier =  86.48648648648648
```

Confusion matrix for Support vector machine classifier

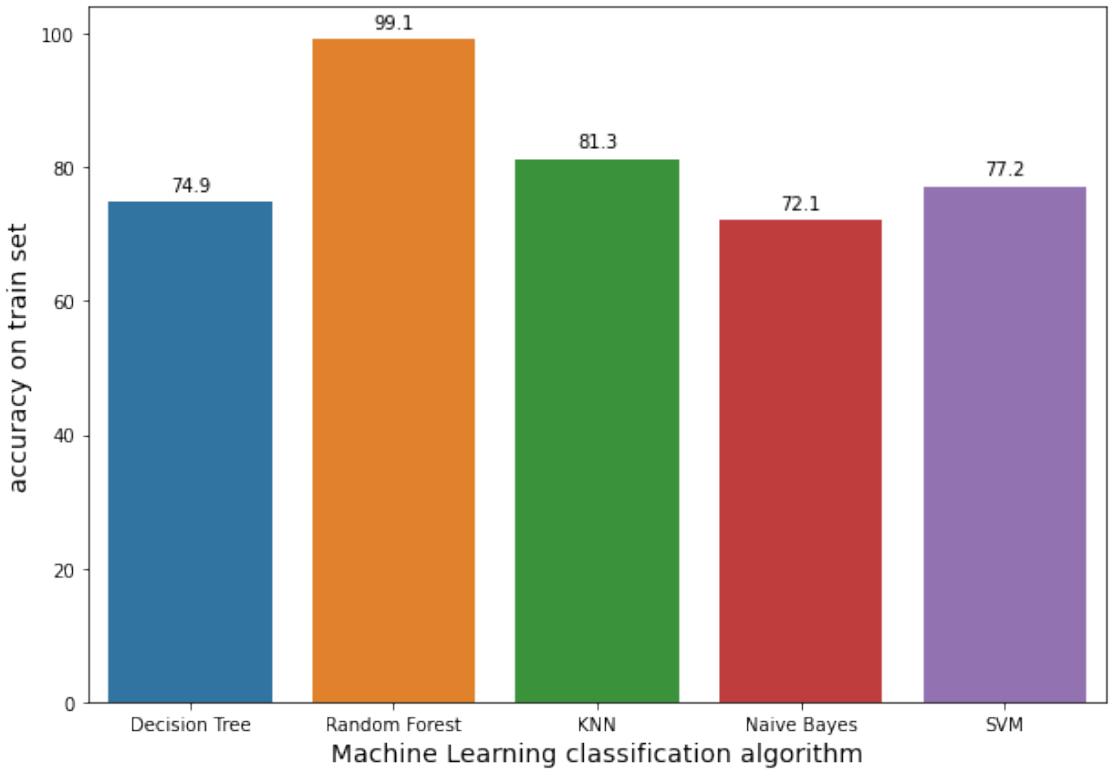
[393]:
```
plot_cm(test_y, pred)
```



Comparisions of Machine learning algorithms used:

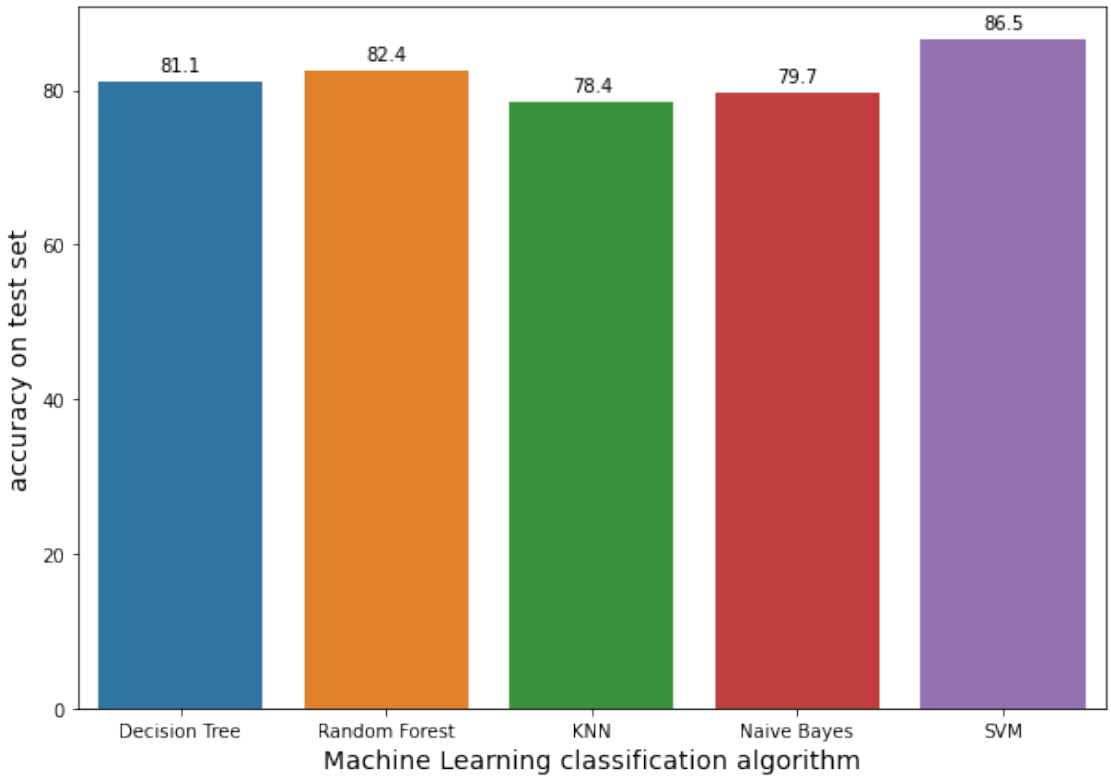
[394]:
```
classifiers=["Decision Tree", "Random Forest", "KNN", "Naive Bayes", "SVM"]
acc_graph = pd.DataFrame({
  "classifiers": classifiers,
  "acc_t": acc_t,
})
```

```
plt.figure(figsize=(10, 7))
splot=sns.barplot(x='classifiers',y='acc_t',data=acc_graph,ci=None)
for p in splot.patches:
    splot.annotate(format(p.get_height(), '.1f'),
                   (p.get_x() + p.get_width() / 2., p.get_height()),
                   ha = 'center', va = 'center',
                   xytext = (0, 9),
                   textcoords = 'offset points')
plt.xlabel("Machine Learning classification algorithm", size=14)
plt.ylabel("accuracy on train set", size=14)
plt.show()
```



[395]:
```
classifiers=["Decision Tree", "Random Forest", "KNN", "Naive Bayes", "SVM"]
acc_graph = pd.DataFrame({
  "classifiers": classifiers,
  "acc": acc,
})
plt.figure(figsize=(10, 7))
splot=sns.barplot(x='classifiers',y='acc',data=acc_graph,ci=None)
for p in splot.patches:
    splot.annotate(format(p.get_height(), '.1f'),
                   (p.get_x() + p.get_width() / 2., p.get_height()),
                   ha = 'center', va = 'center',
```

```
                    xytext = (0, 9),
                    textcoords = 'offset points')
plt.xlabel("Machine Learning classification algorithm", size=14)
plt.ylabel("accuracy on test set", size=14)
plt.show()
```



We observe that for train set, Random forest gave best results, whereas for test set all algorithms gave almost same accuracies. We know that Random forest combines multiple decision trees and are henceforth give promising results.

10 Conclusions and learning outcomes

- We were able to understand the features in Heart failure clinical records Data Set, it's shape and datatypes.
- We learnt about distribution of values in various features, their central tendencies, deviations and anomalies.
- We were able to remove few anomalies and cleaned the data.
- We carried out feature selection via two methods.
- We were able to use several Machine Learning algorithms and understood which worked best for this data.

11 References

https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
https://scikit-learn.org/stable/modules/naive_bayes.html
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html
https://scikit-learn.org/stable/modules/feature_selection.html
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html?highlig
https://github.com/brpy/colab-pdf
https://plotly.com/python/pie-charts/
https://plotly.com/python/bar-charts/
https://plotly.com/python/histograms/
https://seaborn.pydata.org/generated/seaborn.barplot.html

12 For downloading this colab notebook :

[ ]:
```
!apt-get install inkscape
```

[319]:
```
%%capture
#!wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
#from colab_pdf import colab_pdf
colab_pdf('IDS-Report.ipynb')
```