1.Importing Necessary Libraries

In [37]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# Metrics for Classification technique

from sklearn.metrics import classification_report,confusion_matrix,accuracy_score

# Scaler

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import  RandomizedSearchCV, train_test_split

#Model building

from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
```

2.Creating Dataframe

In [11]:

```
df = pd.read_csv("heart.csv")
data.head(6) # Mention no of rows to be displayed from the top in the argument
```

Out[11]:

	age	sex	chestpain	bloodpressure	cholestoral	bloodsugar	ECG	Heartrate	Outcome
0	70	1	4	130	322	0	2	109	2
1	67	0	3	115	564	0	2	160	1
2	57	1	2	124	261	0	0	141	2
3	64	1	4	128	263	0	0	105	1
4	74	0	2	120	269	0	2	121	1
5	65	1	4	120	177	0	0	140	1

3.Exploratory Data Analysis

A. Observing Dataframe

B. Data Cleaning

C.Data Reduction

D.Data Transformation¶

E.Data Encoding

In [12]:

```
# printing shape of dataframe
print("Shape of Dataframe:",np.shape(df))
```

```
Shape of Dataframe: (1025, 14)
```

In [13]:

```
# the type of each feature that our dataset holds.

print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       1025 non-null   int64  
 1   sex       1025 non-null   int64  
 2   cp        1025 non-null   int64  
 3   trestbps  1025 non-null   int64  
 4   chol      1025 non-null   int64  
 5   fbs       1025 non-null   int64  
 6   restecg   1025 non-null   int64  
 7   thalach   1025 non-null   int64  
 8   exang     1025 non-null   int64  
 9   oldpeak   1025 non-null   float64
 10  slope     1025 non-null   int64  
 11  ca        1025 non-null   int64  
 12  thal      1025 non-null   int64  
 13  target    1025 non-null   int64  
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
None
```

In [15]:

```
# checking for missing value
print("missing values",df.isnull().sum())
```

```
missing values 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 [16]:

```
# Out of 14 features, we have 13 int types and only one with the float data types.
# Woah! Fortunately, this dataset doesn’t hold any missing values.
# As we are getting some information from each feature so let’s see how statistically the d

df.describe()
```

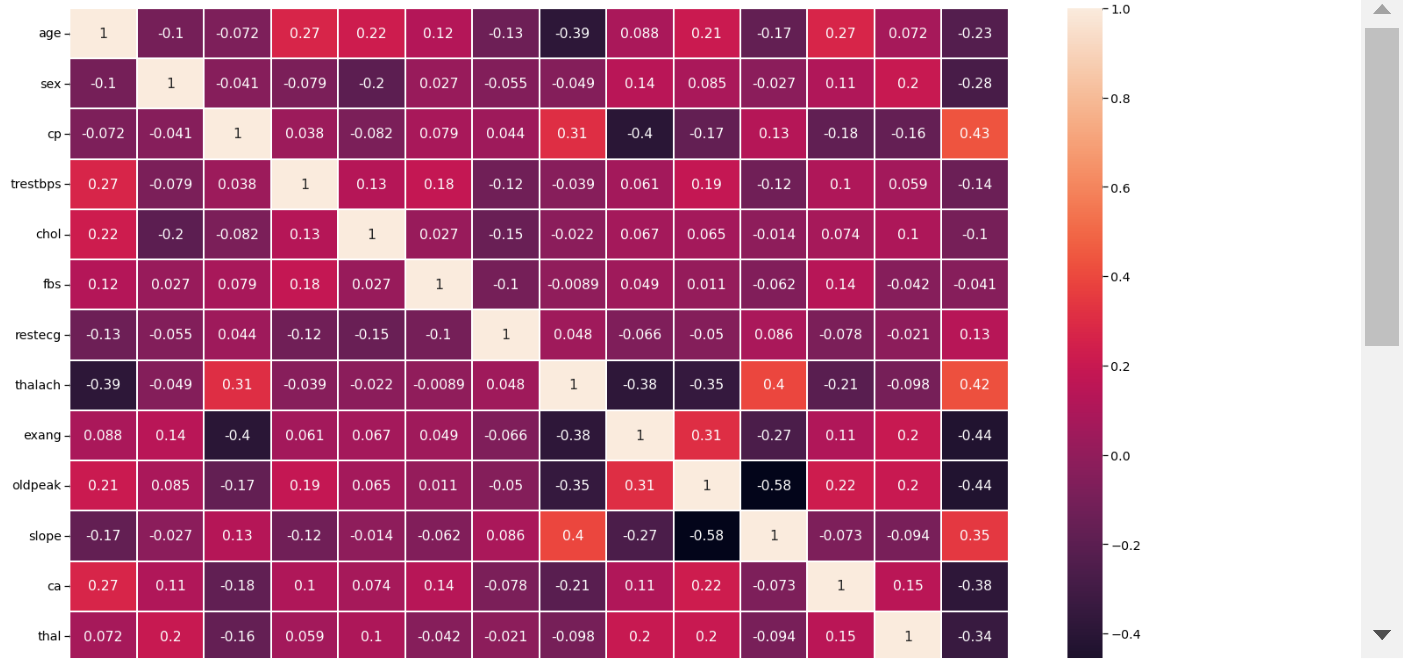
Out[16]:

	age	sex	cp	trestbps	chol	fbs	reste
count	1025.000000	1025.000000	1025.000000	1025.000000	1025.00000	1025.000000	1025.0000
mean	54.434146	0.695610	0.942439	131.611707	246.00000	0.149268	0.5297
std	9.072290	0.460373	1.029641	17.516718	51.59251	0.356527	0.5278
min	29.000000	0.000000	0.000000	94.000000	126.00000	0.000000	0.0000
25%	48.000000	0.000000	0.000000	120.000000	211.00000	0.000000	0.0000
50%	56.000000	1.000000	1.000000	130.000000	240.00000	0.000000	1.0000
75%	61.000000	1.000000	2.000000	140.000000	275.00000	0.000000	1.0000
max	77.000000	1.000000	3.000000	200.000000	564.00000	1.000000	2.0000

In [17]:

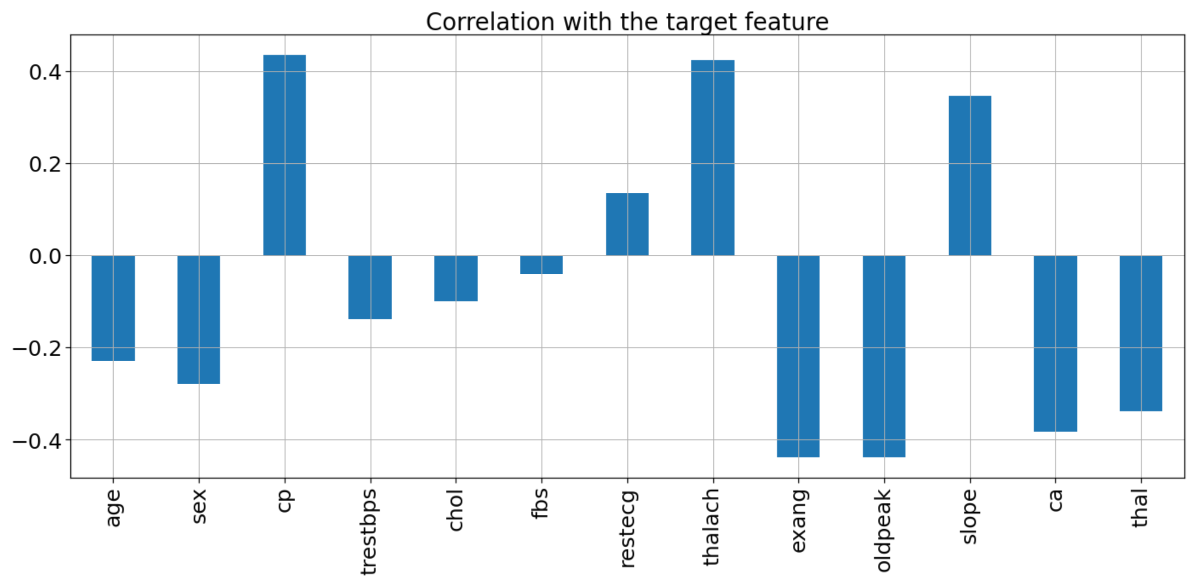
```
""" It is always better to check the correlation between the features so
 that we can analyze that which feature is negatively correlated and
 which is positively correlated so, Let’s check the correlation between various features.""

plt.figure(figsize=(20,12))
sns.set_context('notebook',font_scale = 1.3)
sns.heatmap(df.corr(),annot=True,linewidth =2)
plt.tight_layout()
```



In [19]:

```
# correlation of the target variable.
sns.set_context('notebook',font_scale = 2.3)
df.drop('target', axis=1).corrwith(df.target).plot(kind='bar', grid=True, figsize=(20, 10),
title="Correlation with the target feature")
plt.tight_layout()
```



In [ ]:

```
#Insights from the above graph are:
# Four feature( “cp”, “restecg”, “thalach”, “slope” ) are positively correlated with the ta
# Other features are negatively correlated with the target feature.
# So, we have done enough collective analysis now let’s go for the analysis of the individu
# which comprises both univariate and bivariate analysis.
```

Feature Engineering

In [27]:

```
# Feature Engineering
#Now we will see the complete description of the continuous data as well as the categorical

categorical_val = []
continous_val = []
for column in df.columns:
    print("--------------------")
    print(f"{column} : {df[column].unique()}")
    if len(df[column].unique()) <= 10:
        categorical_val.append(column)
    else:
        continous_val.append(column)
```

```
--------------------
age : [-0.26843658 -0.15815703  1.71659547  0.72407944  0.834359    0.393240
77
  0.06240209 -0.93011394 -0.04787747  1.82687503 -1.26095261 -2.25346864
 -0.37871614 -0.4889957   0.61379988  1.38575679 -1.04039349  0.94463856
 -1.37123217 -1.15067305  0.17268165  0.28296121  0.50352033  1.05491812
  1.16519768 -1.48151173  1.27547724 -1.8123504  -0.59927526 -0.70955482
 -2.80486643 -1.92262996 -0.81983438  1.49603635  2.37827282 -1.59179129
 -1.70207085  2.48855238  1.60631591 -2.14318908  2.1577137 ]
--------------------
sex : [1 0]
--------------------
cp : [0 1 2 3]
--------------------
trestbps : [-0.37763552  0.4791073   0.76468824  0.93603681  0.36487493 -1.8
0554022
 -1.00591359  1.62143107 -0.66321646 -0.54898408 -1.12014597  0.0221778
 -0.77744884 -0.20628695 -0.43475171 -1.4628431  -1.57707547  0.19352636
 -0.09205458  0.25064255  2.76375483 -0.14917077  1.05026919  2.64952245
  0.82180443 -0.83456502  1.16450156  1.27873394  2.19259295  0.13641017
  2.4210577   0.70757206 -1.34861072 -0.4918679  -1.23437834  0.59333968
 -0.32051933  3.44914909 -0.9487974  -2.14823735  3.90607859  1.90701201
 -1.69130785 -1.51995928  1.33585013  2.30682533  1.84989582  1.39296631
 -1.74842404]
--------------------
chol : [-6.59332089e-01 -8.33861171e-01 -1.39623266e+00  9.30821772e-01
  3.87842405e-02  1.39623266e+00  8.33861171e-01  5.81763608e-02
  7.75684810e-01 -1.88103566e+00  1.84225142e+00 -6.98116329e-01
  1.00839025e+00 -8.14469051e-01  1.20231146e+00  3.87842405e-01
 -3.87842405e-02 -6.78724209e-01 -1.18291934e+00 -4.46018766e-01
 -7.36900570e-01  1.16352722e-01 -7.17508449e-01  1.18291934e+00
 -2.52097563e-01  1.41562478e+00  1.93921203e-01  1.57076174e+00
 -1.49319326e+00 -2.23009383e+00  4.46018766e-01 -9.69606013e-01
 -2.90881804e-01 -6.39939968e-01  4.84803006e-01  3.29666044e-01
 -3.29666044e-01  2.21070171e+00  2.32705443e-01  1.62893810e+00
  1.86164354e+00 -3.87842405e-01 -3.49058165e-01  6.20547848e-01
 -3.10273924e-01  7.17508449e-01 -9.69606013e-02 -1.37684054e+00
 -1.12474297e+00 -5.62371487e-01 -1.02778237e+00 -1.93921203e-02
 -2.71489684e-01  1.02778237e+00  8.14469051e-01 -9.50213892e-01
  1.33805630e+00 -6.01155728e-01 -1.59015386e+00  1.55136962e+00
 -7.56292690e-01 -1.33805630e+00  2.13313323e-01  1.74529082e-01
 -1.14413509e+00 -8.72645411e-01 -5.04195127e-01  4.26626646e-01
  4.07234525e-01 -1.93921203e-01  1.10535085e+00  6.98116329e-01
 -2.32705443e+00  1.22170358e+00 -1.16352722e+00  5.62371487e-01
  6.78724209e-01 -7.75684810e-01  1.72589870e+00 -5.42979367e-01
  1.55136962e-01  9.50213892e-01  3.31605256e+00  2.71489684e-01
```

```
 -1.16352722e-01  1.08595873e+00 -1.04717449e+00 -4.07234525e-01
  1.53197750e+00 -2.13313323e-01  5.42979367e-01 -2.32705443e-01
 -1.24109570e+00 -1.53197750e+00 -1.43501690e+00  1.45440902e+00
  1.04717449e+00 -9.11429652e-01  6.16669424e+00 -1.72589870e+00
  1.12474297e+00 -4.65410886e-01 -1.20231146e+00  2.09434899e+00
 -1.66772234e+00  1.93921203e-02 -1.35744842e-01  3.46606213e-18
  3.16091560e+00  9.11429652e-01 -1.27987994e+00  7.75684810e-02
 -4.84803006e-01 -8.92037532e-01 -3.68450285e-01 -5.81763608e-02
  1.26048782e+00  2.90881804e-01 -7.75684810e-02 -7.95076930e-01
  1.16352722e+00 -5.23587247e-01  2.07495687e+00 -9.30821772e-01
  2.87003380e+00 -1.22170358e+00 -1.74529082e-01 -4.26626646e-01
  3.68450285e-01  1.29927206e+00  1.82285930e+00  2.52097563e-01
  4.65410886e-01 -5.81763608e-01  3.49058165e-01  5.81763608e-01
  1.47380114e+00 -6.20547848e-01  5.23587247e-01  1.35744842e-01
 -1.35744842e+00  7.36900570e-01  1.14413509e+00 -1.51258538e+00
  3.12213136e+00  8.53253291e-01  6.01155728e-01  3.10273924e-01
 -9.88998133e-01 -1.55136962e+00 -1.31866418e+00 -2.03617263e+00]
--------------------
fbs : [0 1]
--------------------
restecg : [1 0 2]
--------------------
thalach : [ 0.82132052  0.2559679  -1.04869198  0.51689988 -1.87497657 -1.17
915797
 -0.39636204 -0.17891872 -0.22240739 -1.44008994 -0.57031669  1.86504843
  0.29945657 -0.30938471 -1.74451058  0.56038854  0.69085453 -0.04845273
  0.99527517  1.03876384 -0.13543006  1.29969581  0.12550192 -1.39660128
 -1.48357861 -1.61404459  0.60387721 -0.0919414   1.4301618  -1.91846523
  0.03852459  0.08201325  0.86480918  0.7343432   1.25620715 -0.74427134
  0.47341122 -1.1356693  -0.4398507  -1.65753326  1.34318447  0.64736587
  2.29993506  0.34294523  0.42992256  0.90829785 -0.48333936  1.12574116
  0.38643389 -1.00520332 -0.26589605 -0.35287337  0.77783186 -2.35335186
  1.7780711  -1.35311262 -2.00544256  1.38667314 -1.78799925  1.21271849
 -0.65729401 -1.26613529  0.95178651 -0.00496407  0.21247924  0.16899058
 -2.6577725   1.0822525  -1.52706727  1.99551442 -0.70078268 -2.3098632
 -1.09218064 -0.78776     1.56062779  1.95202575 -0.91822599 -0.96171465
  1.60411645  1.51713913  1.69109378 -0.83124866 -3.39707977 -0.52682803
 -2.17939721 -1.22264663  1.64760511 -2.26637454 -2.57079518 -0.87473733
 -1.57055593]
--------------------
exang : [0 1]
--------------------
oldpeak : [-0.06088839  1.72713707  1.30141672 -0.91232909  0.70540823  2.83
400998
 -0.23117653  1.81228114  0.44997602  1.641993   -0.3163206   2.66372184
  0.36483196  0.96084044  0.02425568 -0.65689688 -0.57175281 -0.40146467
  1.98256928  1.47170486  0.10939975  1.55684893  2.15285742  0.27968789
 -0.74204095  0.7905523   3.85573881 -0.14603246  0.62026416  4.36660323
  2.4934337   1.21627265 -0.48660874 -0.82718502  0.87569637  1.13112858
  2.32314556  1.04598451  0.19454382  2.06771335]
--------------------
slope : [2 0 1]
--------------------
ca : [2 0 1 3 4]
--------------------
thal : [3 2 1 0]
--------------------
target : [0 1]
```

Now here first we will be removing the target column from our set of features

then we will categorize all the categorical variables using the get dummies method which will create a separate column for each category suppose X variable contains 2 types of unique values then it will create 2 different columns for the X variable.

In [28]:

```
categorical_val.remove('target')
df = pd.get_dummies(df, columns = categorical_val)
df.head(6)
```

Out[28]:

	age	trestbps	chol	thalach	oldpeak	target	sex_0	sex_1	cp_0	cp_1	...
0	-0.268437	-0.377636	-0.659332	0.821321	-0.060888	0	0	1	1	0	...
1	-0.158157	0.479107	-0.833861	0.255968	1.727137	0	0	1	1	0	...
2	1.716595	0.764688	-1.396233	-1.048692	1.301417	0	0	1	1	0	...
3	0.724079	0.936037	-0.833861	0.516900	-0.912329	0	0	1	1	0	...
4	0.834359	0.364875	0.930822	-1.874977	0.705408	0	1	0	1	0	...
5	0.393241	-1.805540	0.038784	-1.179158	-0.060888	1	1	0	1	0	...

6 rows × 31 columns

In [29]:

```
# Now we will be using the standard scaler method to scale down
# the data so that it won’t raise the outliers also dataset which
# is scaled to general units leads to having better accuracy.

sc = StandardScaler()
col_to_scale = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
df[col_to_scale] = sc.fit_transform(df[col_to_scale])
df.head(6)
```

Out[29]:

	age	trestbps	chol	thalach	oldpeak	target	sex_0	sex_1	cp_0	cp_1	...
0	-0.268437	-0.377636	-0.659332	0.821321	-0.060888	0	0	1	1	0	...
1	-0.158157	0.479107	-0.833861	0.255968	1.727137	0	0	1	1	0	...
2	1.716595	0.764688	-1.396233	-1.048692	1.301417	0	0	1	1	0	...
3	0.724079	0.936037	-0.833861	0.516900	-0.912329	0	0	1	1	0	...
4	0.834359	0.364875	0.930822	-1.874977	0.705408	0	1	0	1	0	...
5	0.393241	-1.805540	0.038784	-1.179158	-0.060888	1	1	0	1	0	...

6 rows × 31 columns

4.Model Building

In [42]:

```
#Modeling
#Splitting our Dataset
# Training and testing of dataset
X = df.drop('target', axis=1)
y = df.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

5. The KNN Machine Learning Algorithm

In [41]:

```
knn = KNeighborsClassifier(n_neighbors = 10)
knn.fit(X_train,y_train)
y_pred1 = knn.predict(X_test)
print("Accuracy of Model using KNN is :",accuracy_score(y_test,y_pred1))
```

```
Accuracy of Model using KNN is : 0.827922077922078
```

6. Random Forest Classifier

In [38]:

```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators=20)
model = classifier.fit(X_train, y_train)
# Checking Accuracy of Model
print("Random Forest Score:", model.score(X_train,y_train))
```

```
Random Forest Score: 1.0
```

In [40]:

```
#  Output: Accuracy = 1.0

#  So here we can see that on the training dataset our model is overfitted.

## Getting the accuracy score for Random Forest

from sklearn import metrics

predictions = classifier.predict(X_test)
print("Accuracy_Score for Random Forest Algorithm is =", format(metrics.accuracy_score(y_te
```

```
Accuracy_Score for Random Forest Algorithm is = 0.9805194805194806
```