```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import numpy as np
import pandas as pd

import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline


import plotly.graph_objects as go
import plotly.express as px

import seaborn as sns

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

## Model evaluators
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
from sklearn.metrics import RocCurveDisplay
```

```
df = pd.read_csv("/content/drive/MyDrive/heart_disease_project/heart_disease/heart-disease.csv")
df.shape
```

```
(303, 14)
```

```
df.head(10)
```

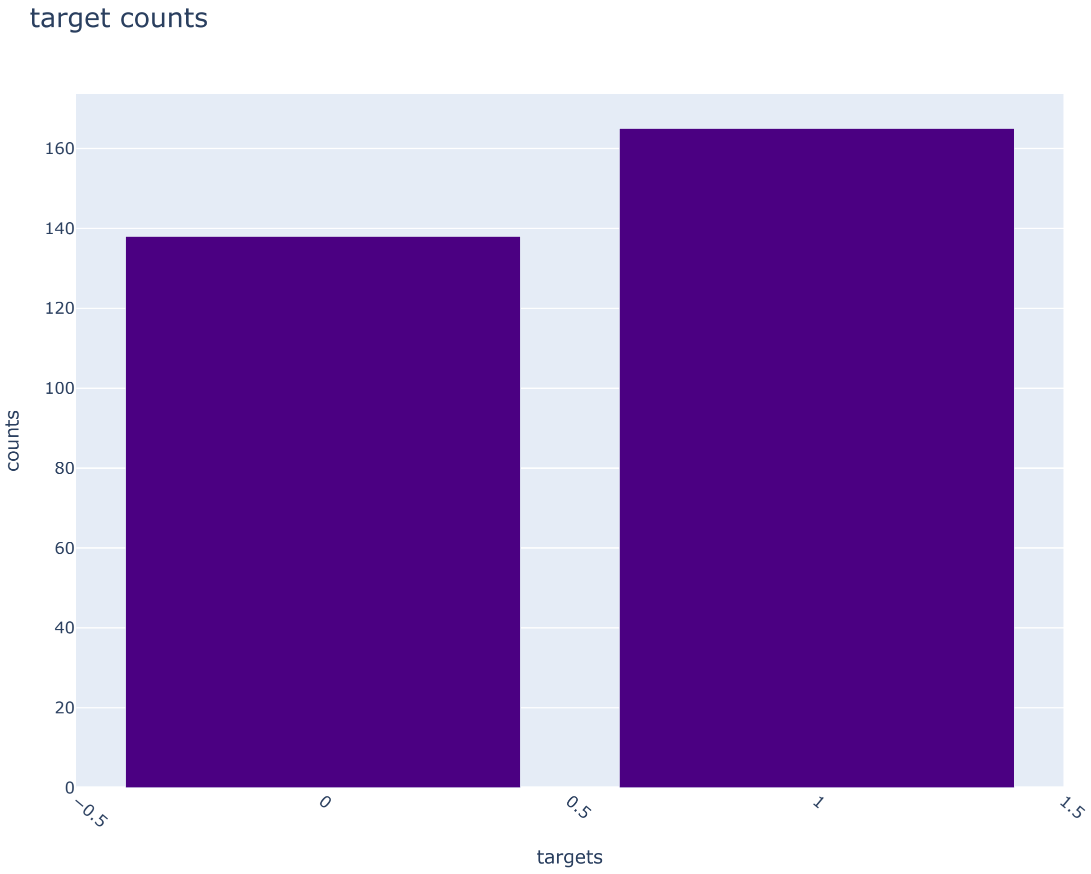
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
5	57	1	0	140	192	0	1	148	0	0.4	1	0	1	1
6	56	0	1	140	294	0	0	153	0	1.3	1	0	2	1
7	44	1	1	120	263	0	1	173	0	0.0	2	0	3	1
8	52	1	2	172	199	1	1	162	0	0.5	2	0	3	1
9	57	1	2	150	168	0	1	174	0	1.6	2	0	2	1

```
df.target.value_counts()
```

	count
target	
1	165
0	138

dtype: int64

```
data_to_plot = df['target'].value_counts().reset_index()
data_to_plot.columns = ['target', 'count']


fig = go.Figure(
    data=[
        go.Bar(
            x=data_to_plot['target'],
            y=data_to_plot['count'],
            marker_color='indigo'
        )
    ],
    layout_title_text="target counts "
)

fig.update_layout(
    xaxis_title="targets",
    yaxis_title="counts",
    title_font_size=20,
    xaxis_tickangle=40,
    height=700,
    width=900
)

fig.show()
```



```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       303 non-null    int64  
 1   sex       303 non-null    int64  
 2   cp        303 non-null    int64  
 3   trestbps  303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalach   303 non-null    int64  
 8   exang     303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64  
 11  ca        303 non-null    int64  
 12  thal      303 non-null    int64  
 13  target    303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

```
df.describe()
```

	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	303.00000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.39934
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.61622
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.00000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.00000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.00000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	2.00000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.00000

comapring features

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

	count
sex	
1	207
0	96

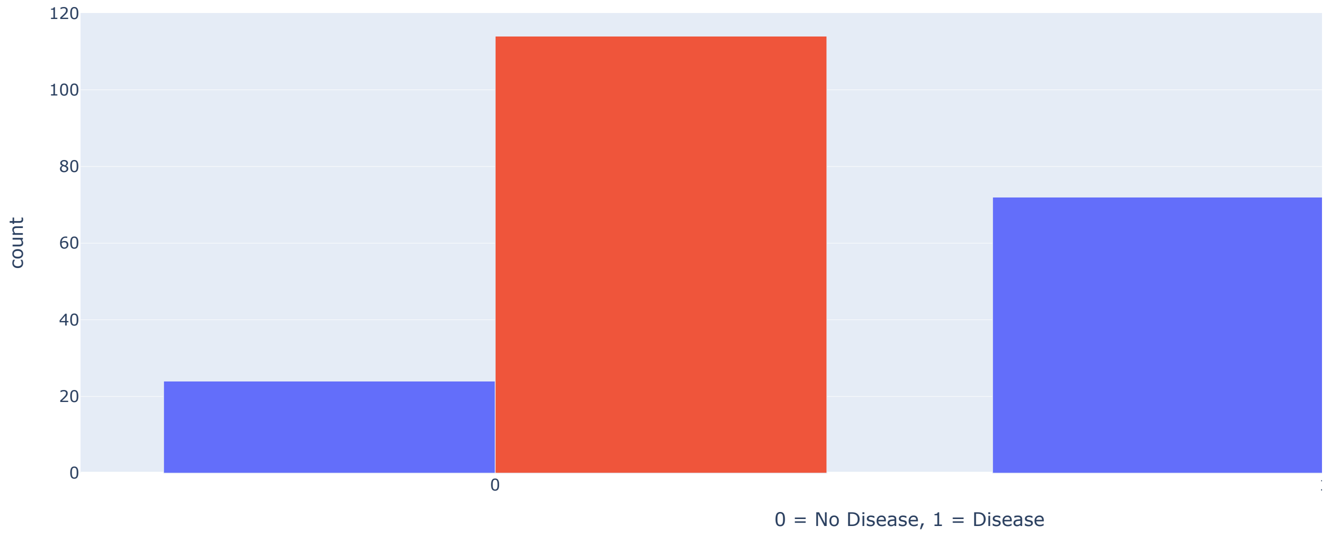
dtype: int64

```
pd.crosstab(index=df.target, columns=df.sex)
```

sex	0	1
target		
0	24	114
1	72	93

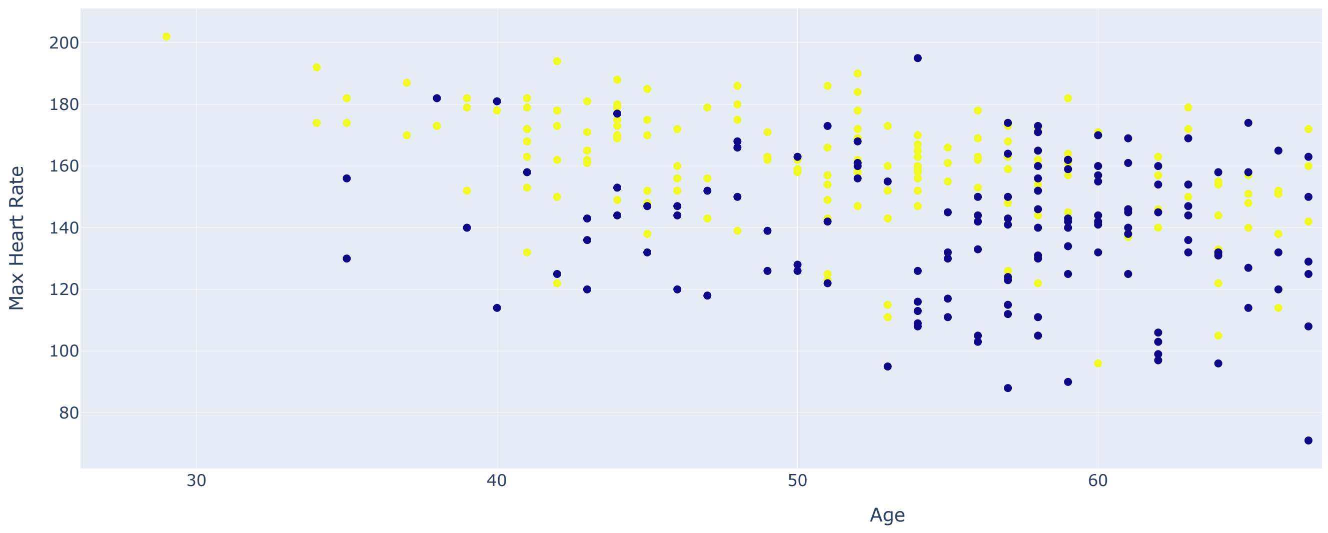
```
crosstab = pd.crosstab(index=df['target'], columns=df['sex'])

crosstab = crosstab.reset_index()


crosstab_melted = crosstab.melt(id_vars='target', var_name='sex', value_name='count')


fig = px.bar(crosstab_melted, x='target', y='count', color='sex', barmode='group', title = "Heart Disease Frequency as per Sex")
fig.update_layout(
    xaxis_title=("0 = No Disease, 1 = Disease"),
    yaxis_title='count'
)
fig.show()
```

Heart Disease Frequency as per Sex



```
fig = px.scatter(
    df,
    x='age',
    y='thalach',
 color='target',
    color_discrete_map={'Disease': 'salmon', 'No Disease': 'lightblue'},
    labels={
        'age': 'Age',
        'thalach': 'Max Heart Rate',
        'target': 'Condition'
    },
    title='Heart Disease in Function of Age and Max Heart Rate'
)

fig.show()
```

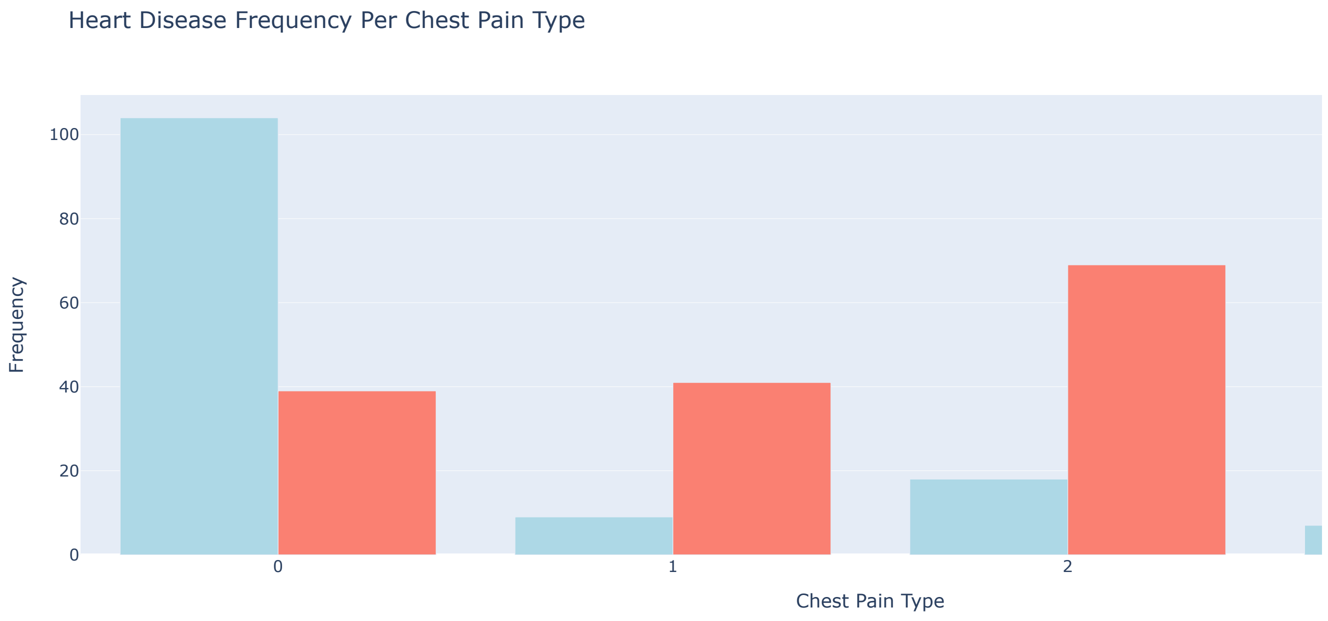
Heart Disease in Function of Age and Max Heart Rate



```
df.head()
```

	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

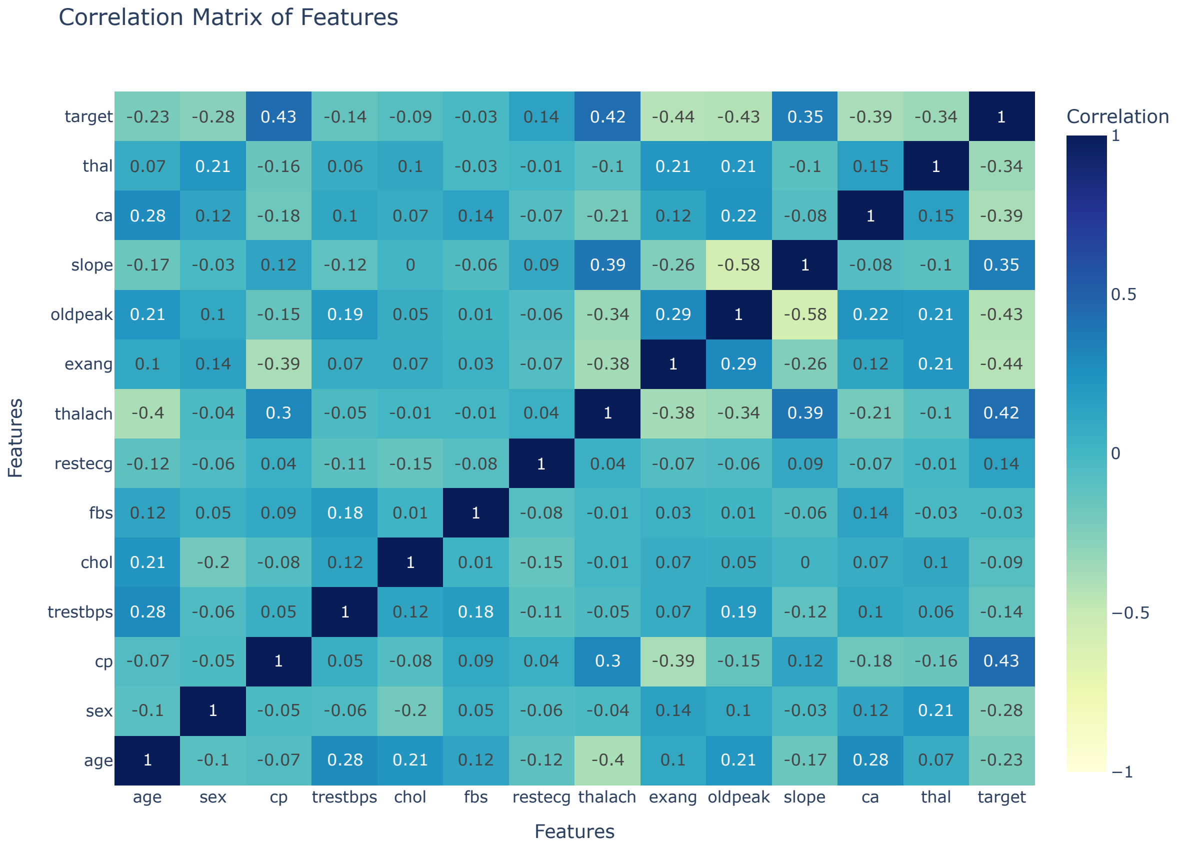
```
cp_target_ct = pd.crosstab(df['cp'], df['target'])


cp_target_ct = cp_target_ct.reset_index()

cp_target_ct.columns = ['Chest Pain Type', 'No Disease', 'Disease']

cp_target_melted = cp_target_ct.melt(id_vars='Chest Pain Type',
                                     value_vars=['No Disease', 'Disease'],
                                     var_name='Heart Disease',
                                     value_name='Frequency')

fig = px.bar(cp_target_melted,
             x='Chest Pain Type',
             y='Frequency',
             color='Heart Disease',
             barmode='group',
             title='Heart Disease Frequency Per Chest Pain Type',
             color_discrete_map={'No Disease': 'lightblue', 'Disease': 'salmon'})

fig.update_layout(xaxis_title='Chest Pain Type',
                  yaxis_title='Frequency')

fig.show()
```



```
corr_matrix = df.corr()
corr_matrix
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	
age	1.000000	-0.098447	-0.068653	0.279351	0.213678	0.121308	-0.116211	-0.398522	0.096801	0.210013	-0.168814	0.276326	0
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
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
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
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
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
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
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
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
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
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
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
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
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

```
corr_matrix = df.corr()

fig = go.Figure(data=go.Heatmap(
    z=corr_matrix.values,
    x=corr_matrix.columns,
    y=corr_matrix.index,
    colorscale='YlGnBu',
    zmin=-1, zmax=1,
    text=np.round(corr_matrix.values, 2),
    texttemplate="%{text}",
    colorbar=dict(title="Correlation")
))

fig.update_layout(
    title="Correlation Matrix of Features",
    xaxis_title="Features",
    yaxis_title="Features",
    width=900,
    height=700,
)

fig.show()
```



Correlation Matrix of Features

modelling

```
X = df.drop(labels="target", axis=1)
y = df.target.to_numpy()
```

```
X.head()
```

	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

```
y,type(y)
```

```
(array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]),
 numpy.ndarray)
```

```
np.random.seed(42)

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size = 0.2)
```

```
X_train.head()
```

	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

```
X_test.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
179	57	1	0	150	276	0	0	112	1	0.6	1	1	1
228	59	1	3	170	288	0	0	159	0	0.2	1	0	3
111	57	1	2	150	126	1	1	173	0	0.2	2	1	3
246	56	0	0	134	409	0	0	150	1	1.9	1	2	3
60	71	0	2	110	265	1	0	130	0	0.0	2	1	2

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

```
(array([1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1,
        1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0,
        1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1,
        0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0,
        0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0,
        1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1,
        1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1,
        1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0,
        0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1,
        1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1,
        1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1]),
 242)
```

```
y_test, len(y_test)
```

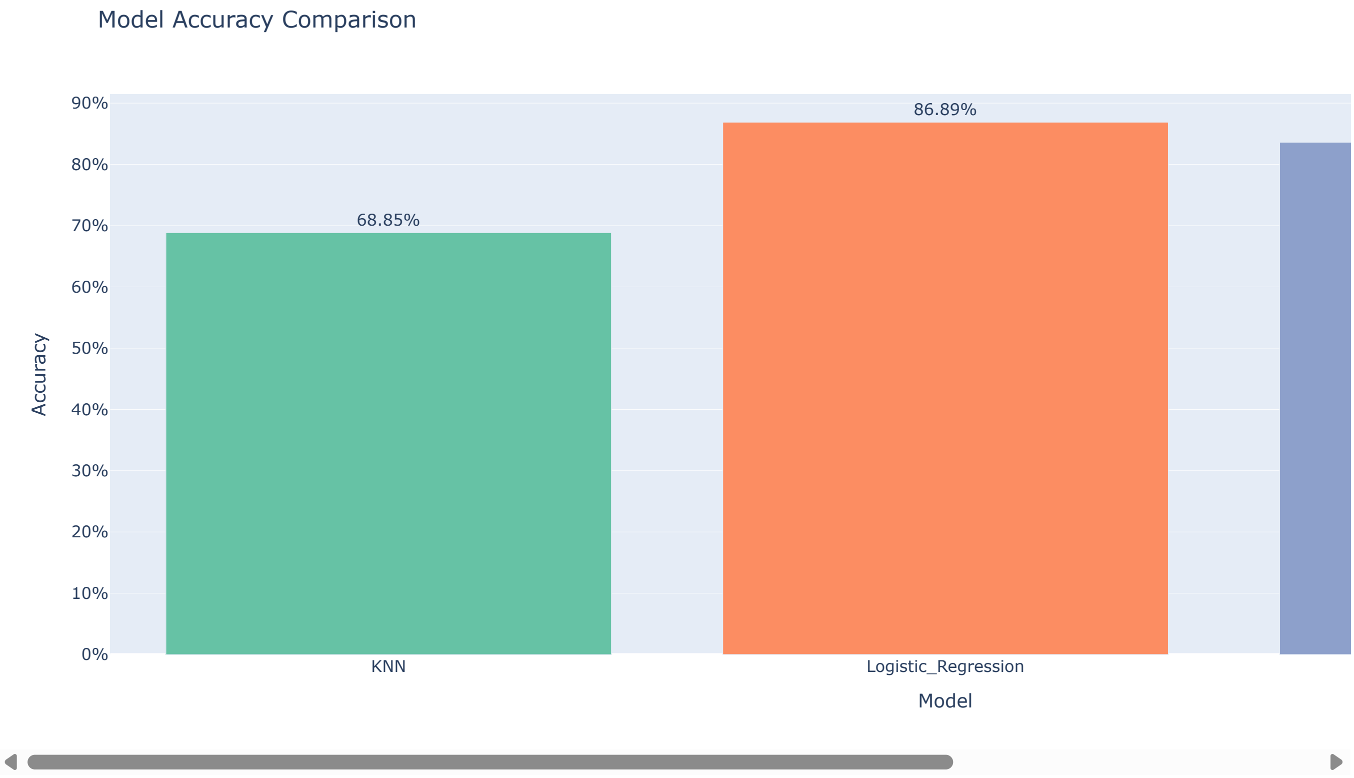
```
(array([0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0,
        0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0]),
 61)
```

```
models = {"KNN": KNeighborsClassifier(),
          "Logistic_Regression": LogisticRegression(max_iter=50,solver='liblinear'),
          "Random_Forest": RandomForestClassifier()}


def fit_and_score(models, X_train, X_test, y_train, y_test):
    np.random.seed(42)
    model_scores = {}

    for name, model in models.items():

        model.fit(X_train, y_train)


        model_scores[name] = model.score(X_test, y_test)
    return model_scores
```

```
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
```

```
{'KNN': 0.6885245901639344,
 'Logistic_Regression': 0.8688524590163934,
 'Random_Forest': 0.8360655737704918}
```

```
model_compare = pd.DataFrame(model_scores, index=['accuracy']).T.reset_index()
model_compare.columns = ['Model', 'Accuracy']


fig = px.bar(model_compare,
             x='Model',
             y='Accuracy',
             title='Model Accuracy Comparison',
             text='Accuracy',
             color='Model',
             color_discrete_sequence=px.colors.qualitative.Set2)

fig.update_traces(texttemplate='%{text:.2%}', textposition='outside')
fig.update_layout(yaxis=dict(tickformat=".0%"), showlegend=False)
fig.update_layout(
    yaxis=dict(tickformat=".0%"),
    showlegend=False,
    height=600
)
fig.show()
```



```
train_scores = []

test_scores = []

neighbors = list(range(1, 30))

knn = KNeighborsClassifier()

for i in neighbors:
    knn.set_params(n_neighbors = i)

    knn.fit(X_train, y_train)

    train_scores.append(knn.score(X_train, y_train))

    test_scores.append(knn.score(X_test, y_test))
```

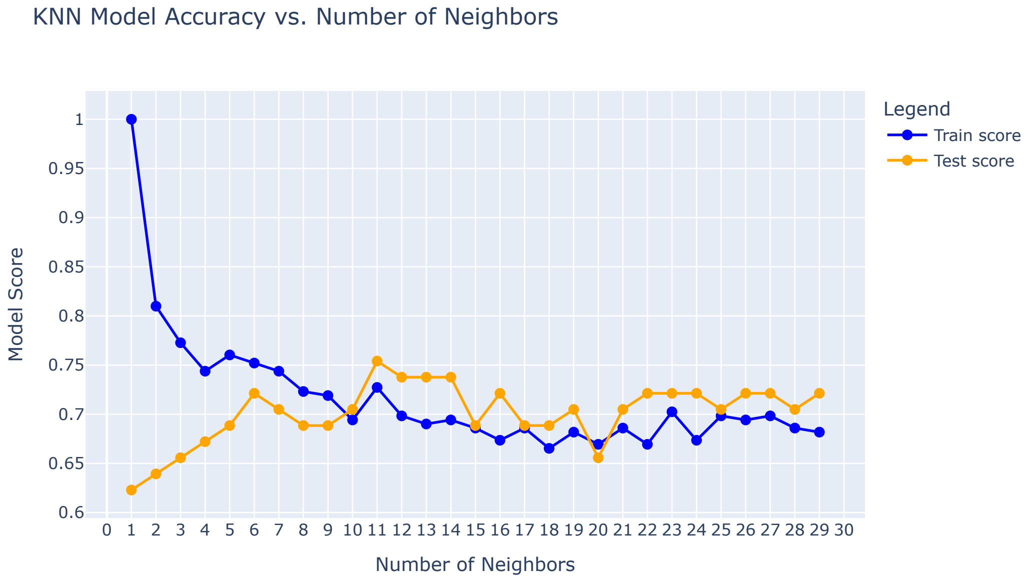
```
train_scores
```

```
[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,
 0.6859504132231405,
 0.6694214876033058,
 0.7024793388429752,
 0.6735537190082644,
 0.6983471074380165,
 0.6942148760330579,
 0.6983471074380165,
 0.6859504132231405,
 0.6818181818181818]
```

```
test_scores
```

```
[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,
 0.7049180327868853,
 0.7213114754098361,
 0.7213114754098361,
 0.7213114754098361,
 0.7049180327868853,
 0.7213114754098361,
 0.7213114754098361,
 0.7049180327868853,
 0.7213114754098361]
```

```


fig = go.Figure()



fig.add_trace(go.Scatter(
    x=neighbors,
    y=train_scores,
    mode='lines+markers',
    name='Train score',
    line=dict(color='blue'),
    marker=dict(size=8)
))



fig.add_trace(go.Scatter(
    x=neighbors,
    y=test_scores,
    mode='lines+markers',
    name='Test score',
    line=dict(color='orange'),
    marker=dict(size=8)
))


fig.update_layout(
    title="KNN Model Accuracy vs. Number of Neighbors",
    xaxis=dict(title="Number of Neighbors", tickmode='linear', dtick=1),
    yaxis=dict(title="Model Score"),
    width=800,
    height=500,
    legend=dict(title="Legend"),
)

fig.show()

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



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

```
# using randomizedsearchcv for model params tunning
log_reg_grid = {"C": np.logspace(-4, 4, 20),
                "solver": ["liblinear"]}

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)}
```

```
#logistic regression
np.random.seed(42)

rs_log_reg = RandomizedSearchCV(LogisticRegression(),
                                param_distributions=log_reg_grid,
                                cv=5,
                                n_iter=20,
                                verbose=True)

rs_log_reg.fit(X_train, y_train);
```

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

```
rs_log_reg.best_params_
```

```
{'solver': 'liblinear', 'C': np.float64(0.23357214690901212)}
```

```
log_reg_accuracy = round(rs_log_reg.score(X_test, y_test)*100,2)
```

```
#random forest
np.random.seed(42)

rs_rf = RandomizedSearchCV(RandomForestClassifier(),
                           param_distributions=rf_grid,
                           cv=5,
                           n_iter=20,
                           verbose=True)

rs_rf.fit(X_train, y_train);
```

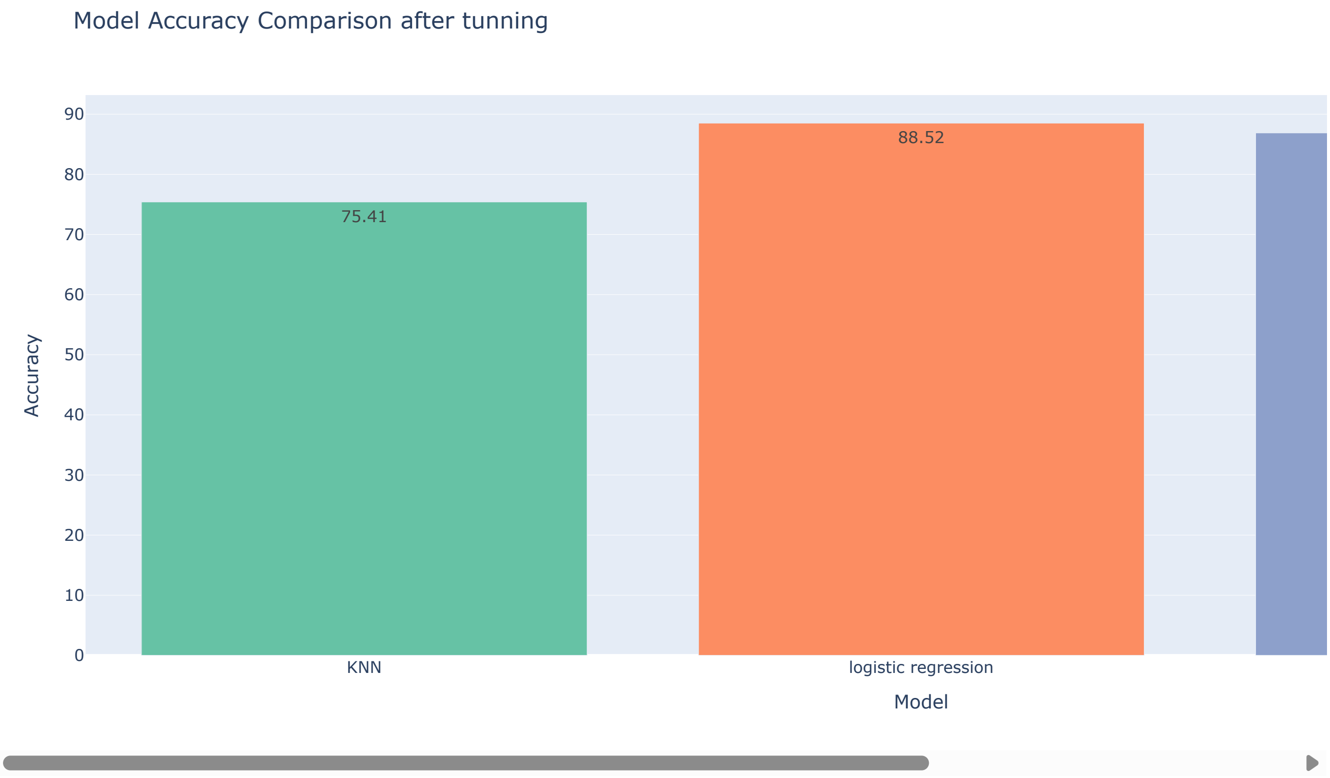
```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
```

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

```
{'n_estimators': np.int64(210),
 'min_samples_split': np.int64(4),
 'min_samples_leaf': np.int64(19),
 'max_depth': 3}
```

```
# Evaluating the randomized search random forest model
rf_accuracy = round(rs_rf.score(X_test, y_test)*100,2)
```

```
model_scores = {
     "KNN": knn_accuracy,
     "logistic regression":log_reg_accuracy,
    "random forest":rf_accuracy
}

model_compare = pd.DataFrame(model_scores, index=['accuracy']).T.reset_index()
model_compare.columns = ['Model', 'Accuracy']


fig = px.bar(model_compare,
             x='Model',
             y='Accuracy',
             title='Model Accuracy Comparison after tunning',
             text='Accuracy',
             color='Model',
             color_discrete_sequence=px.colors.qualitative.Set2)


fig.update_layout(
    yaxis=dict(tickformat=".-0%"),
    showlegend=False,
    height=600
)
fig.show()
```



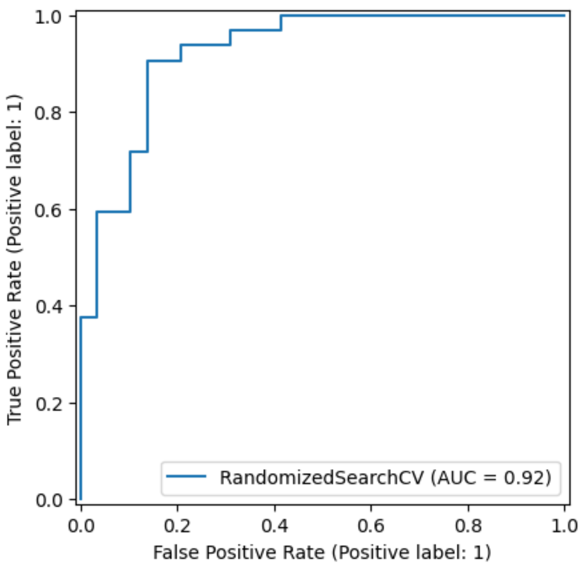
```
y_preds = rs_log_reg.predict(X_test)
y_preds
```

```
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])
```

```
y_test
```

```
array([0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0,
       0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0])
```

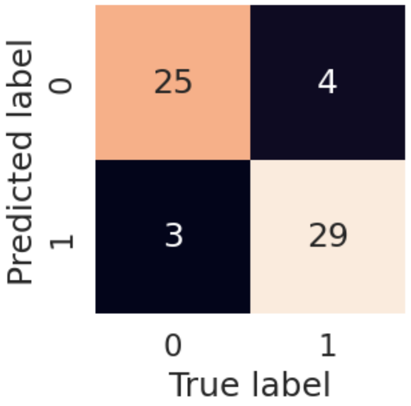
```
#model evaluation
RocCurveDisplay.from_estimator(rs_log_reg, X_test, y_test)
plt.show()
```



```
print(confusion_matrix(y_test, y_preds))
```

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

```
sns.set(font_scale=1.5)

def plot_conf_mat(y_test, y_preds):

    fig, ax = plt.subplots(figsize=(3, 3))
    ax = sns.heatmap(confusion_matrix(y_test, y_preds),
                     annot=True,
                     cbar=False)
    plt.xlabel("True label")
    plt.ylabel("Predicted label")

    bottom, top = ax.get_ylim()
    ax.set_ylim(bottom, top )
    plt.show()

plot_conf_mat(y_test, y_preds)
```



```
print(classification_report(y_test, y_preds))
```

```
              precision    recall  f1-score   support

           0       0.89      0.86      0.88        29
           1       0.88      0.91      0.89        32

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

```
#instanciating log_reg wit best params to check which feature helps most in model training

clf = LogisticRegression(C=0.23357214690901212,
                         solver="liblinear")
```