In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
sns.set_theme(context='notebook', style='white', palette='deep', font='Lucida Calligraphy',
              font_scale=1.5, color_codes=True, rc=None)
```

In [3]:
```
#related to the figure
plt.rcParams['figure.figsize'] = (14, 8)
plt.rcParams['figure.facecolor'] = 'beige'
plt.rcParams['figure.titlesize'] = 'medium'
plt.rcParams['figure.dpi'] = 220
plt.rcParams['figure.edgecolor'] = 'green'
plt.rcParams['figure.frameon'] = True
plt.rcParams['figure.autolayout'] = True
```

In [4]:
```
# related to the axes
plt.rcParams['axes.facecolor']= '#F5F5DC'
plt.rcParams['axes.titlesize'] = 20
plt.rcParams['axes.titleweight'] = 'normal'
plt.rcParams['axes.titlecolor'] = 'Brown'
plt.rcParams['axes.edgecolor'] = 'red'
plt.rcParams['axes.linewidth'] = 2
plt.rcParams['axes.grid'] = True
plt.rcParams['axes.titlelocation'] = 'center'
plt.rcParams['axes.labelsize'] = 20
plt.rcParams['axes.labelpad'] = 2
plt.rcParams['axes.labelcolor'] = 'red'
plt.rcParams['axes.labelweight'] = 1
plt.rcParams['axes.axisbelow'] = False
plt.rcParams['axes.xmargin'] = .1
plt.rcParams['axes.ymargin'] = .1
```

In [5]:
```
#related to the xtick and ytick
plt.rcParams['xtick.bottom'] =True
plt.rcParams['ytick.left'] = True
plt.rcParams['ytick.right'] = False
plt.rcParams['xtick.color'] = '#A52A2A'
plt.rcParams['ytick.color'] = '#A52A2A'
```

In [6]:
```
# related to the grid
plt.rcParams['grid.color'] = 'green'
plt.rcParams['grid.linewidth'] = 0.5
plt.rcParams['grid.linestyle'] = '--'
plt.rcParams['grid.alpha'] = 0.5
```

In [7]:
```
# related to the legend
plt.rcParams['legend.loc'] = 'best'
plt.rcParams['legend.facecolor'] = 'NavajoWhite'
plt.rcParams['legend.edgecolor'] = 'pink'
plt.rcParams['legend.shadow'] = True
plt.rcParams['legend.edgecolor'] = 'Blue'
```

In [8]:
```
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.precision', 2)
```

In [9]:
```
data = pd.read_csv(filepath_or_buffer=r'E:\datasets\heart.csv')
```

In [10]:
```
data.shape
```

Out[10]: (1025, 14)

In [11]:
```
data.size
```

Out[11]: 14350

In [12]:
```
data.head().style.set_properties(**{'background-color' : 'blue', 'color' : 'red', 'border-color' : 'darkblack'})
```

Out[12]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.000000	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.100000	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.600000	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.000000	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.900000	1	3	2	0

In [13]:
```
print(f'There are {data.shape[0]} instances and {data.shape[1] - 1} features and 1 target variable')
```

```
There are 1025 instances and 13 features and 1 target variable
```

In [14]:
```
data.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
```

All the features numeric

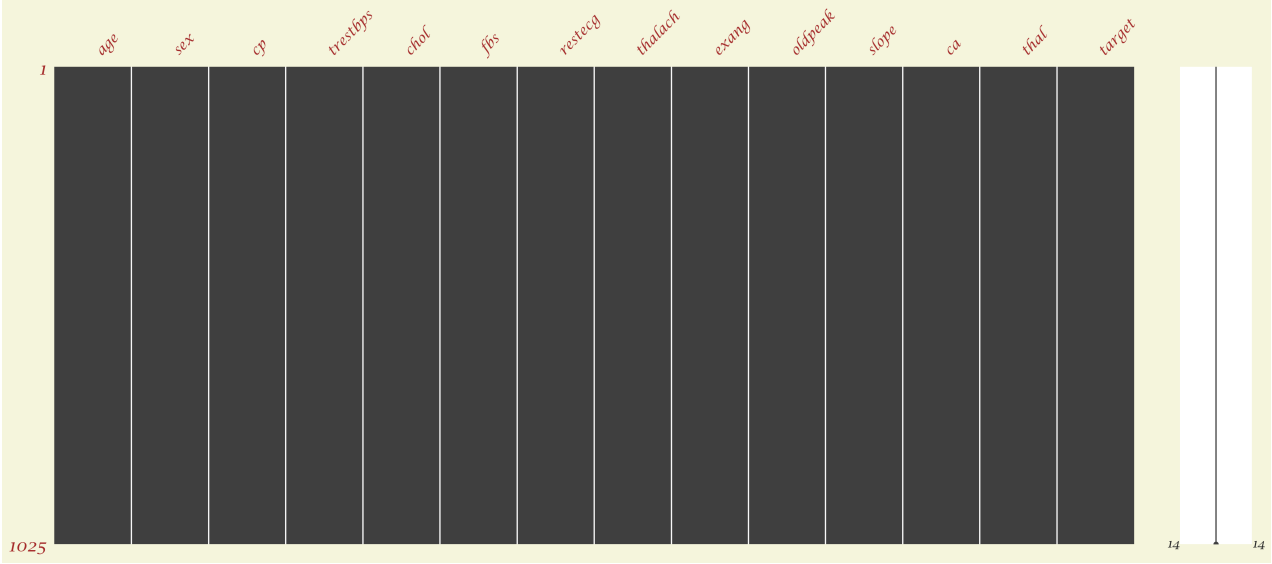
In [15]:
```
data.isnull().sum()
```

Out[15]:
```
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
```

There is no NAN values in the whole dataset

Lets confirm it by employing a library developed espacially for this very purpose

In [16]:
```
import missingno as msno
msno.matrix(data)
plt.show()
```



How much percentage of Males are suffering from heart disease

In [17]:
```
len(data.loc[data['sex'] == 1])
```

Out[17]:
```
713
```

Thre are 713 male records

In [18]:
```
males = data.loc[data['sex'] == 1]['target']
print(f'There are {len(males)} males in the dataset and {males.sum()} have heart disease which makes {((males.sum() / len(ma
```

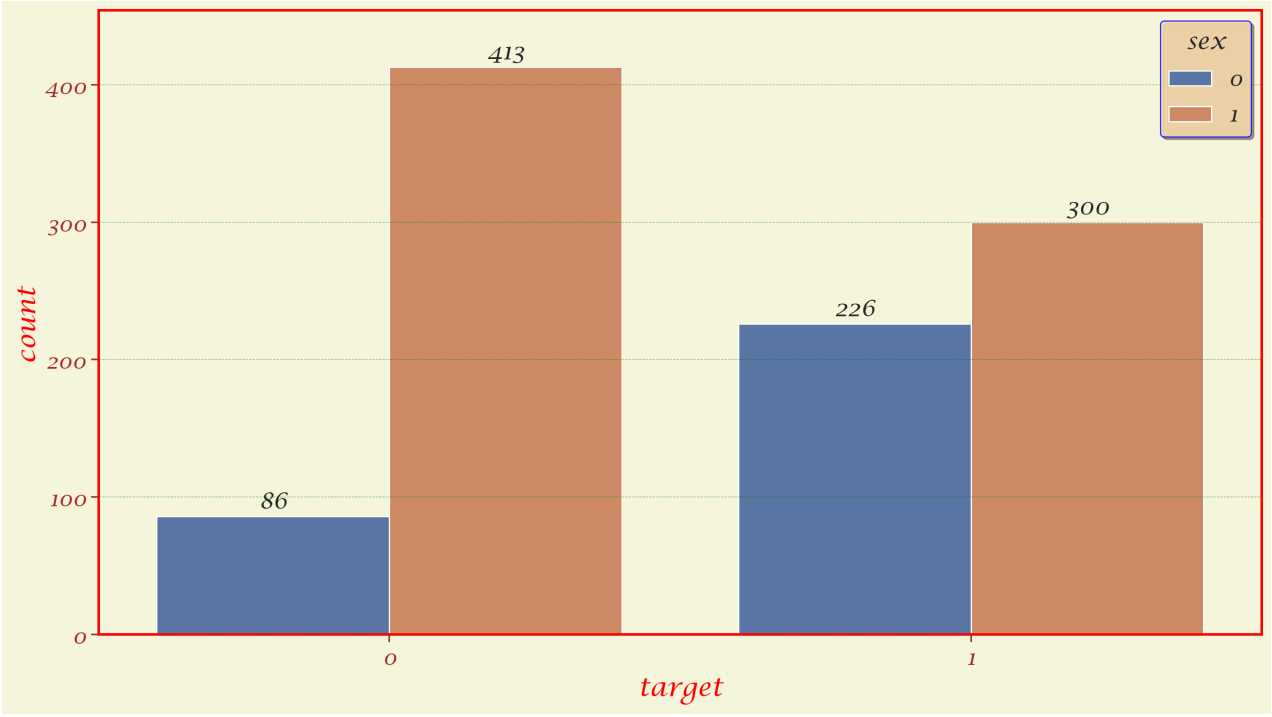
```
There are 713 males in the dataset and 300 have heart disease which makes 42.0 %
```

What about femlaes

In [19]:
```
females = data.loc[data['sex'] == 0]['target']
print(f'There are {len(females)} males in the dataset and {females.sum()} have heart disease which makes {((females.sum() /
```

```
There are 312 males in the dataset and 226 have heart disease which makes 72.0 %
```

In [20]:
```
ax = sns.countplot(data=data, x='target', hue='sex')
for container in ax.containers:
    ax.bar_label(container)
```



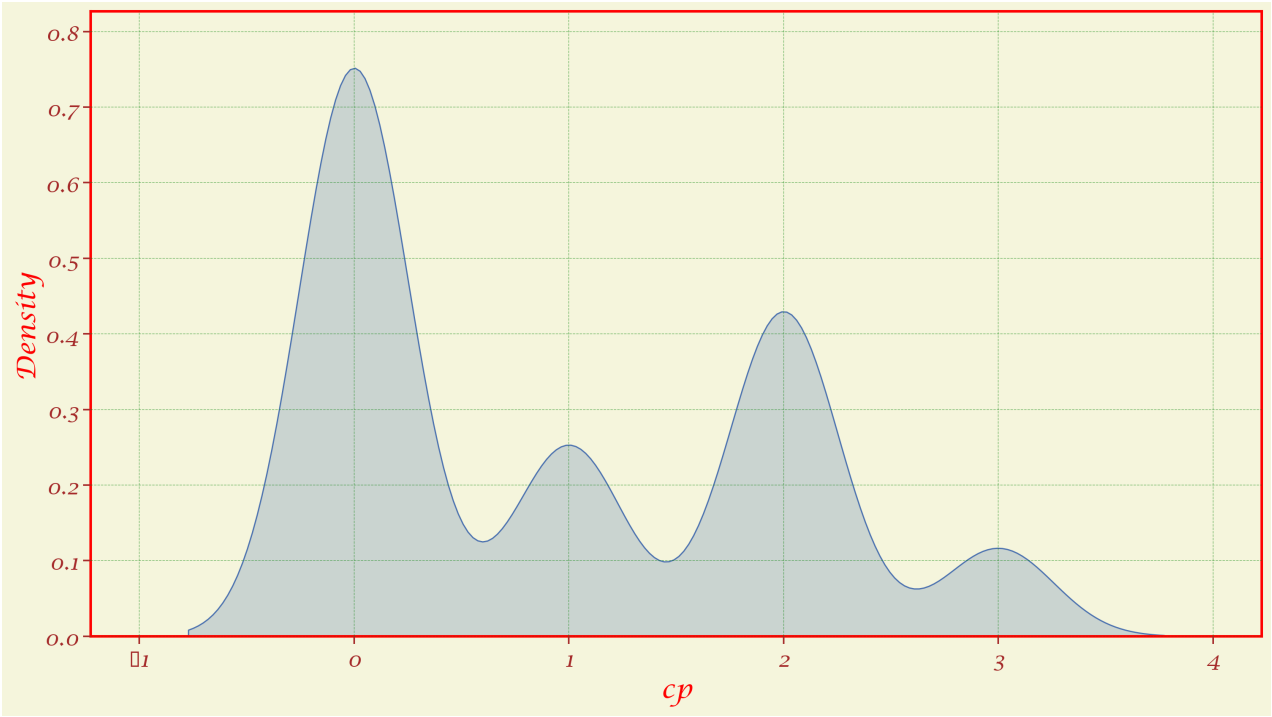
Above data revails that more women are prone to the heart diease than thier male counterparts

In [21]:
```
data['cp'].unique()
```

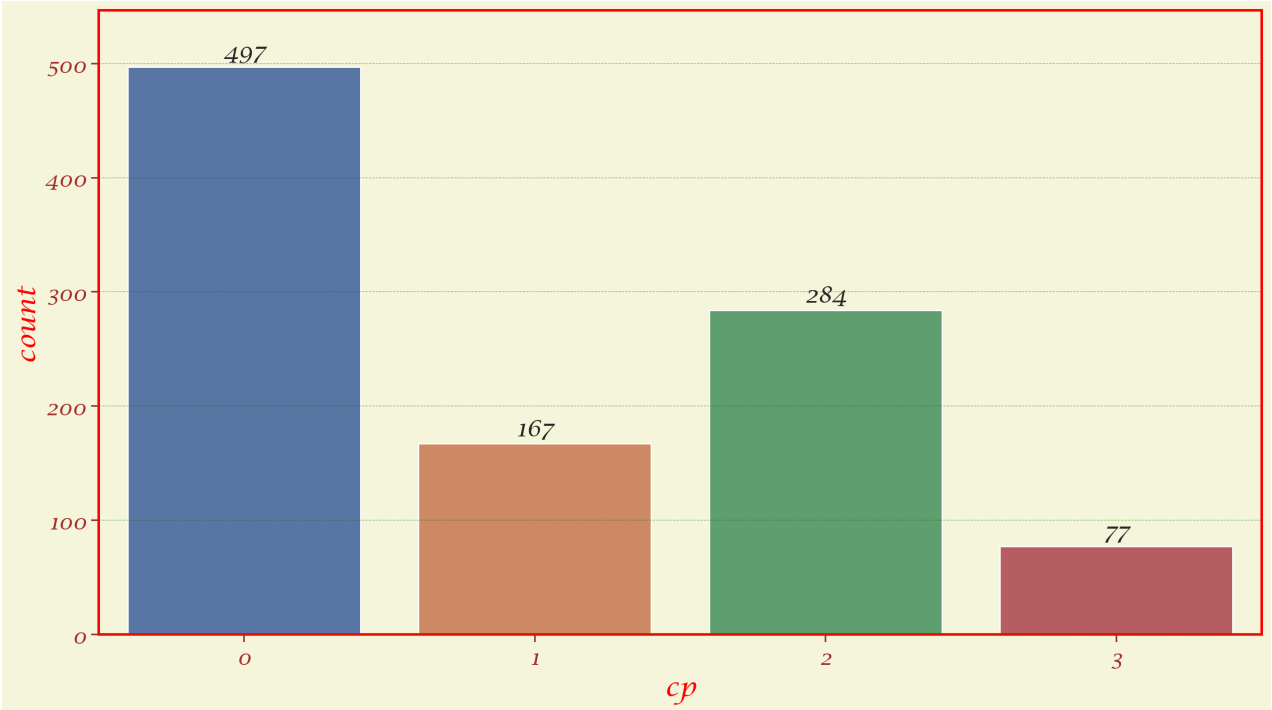
Out[21]:
```
array([0, 1, 2, 3], dtype=int64)
```

The column cp related to the types of the chest pain, which are four types

In [22]:
```
sns.kdeplot(data=data, x='cp', fill=True)
plt.show()
```



In [23]:
```
ax = sns.countplot(data=data, x='cp')
ax.bar_label(ax.containers[0])
plt.show()
```



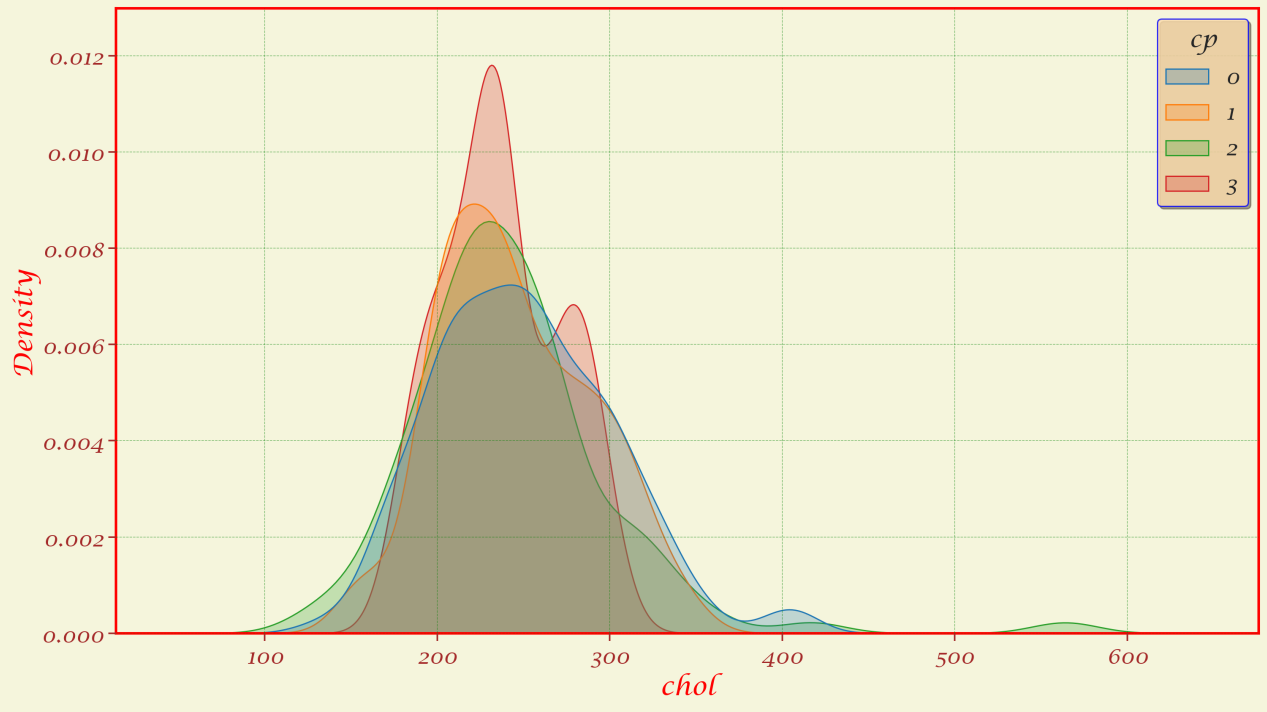
In [24]:
```
no_cp_issue=data.loc[data['cp'] == 0]['target'].sum()
total_no_cp=len(data.loc[data['cp'] == 0]['target'])
print(f'Total {total_no_cp} persons have no chest pain but {no_cp_issue} of them have heart disease which is {((no_cp_issue/
```

```
Total 497 persons have no chest pain but 122 of them have heart disease which is 25.0
```

It means that a person with no chest pain could be a heart patient.

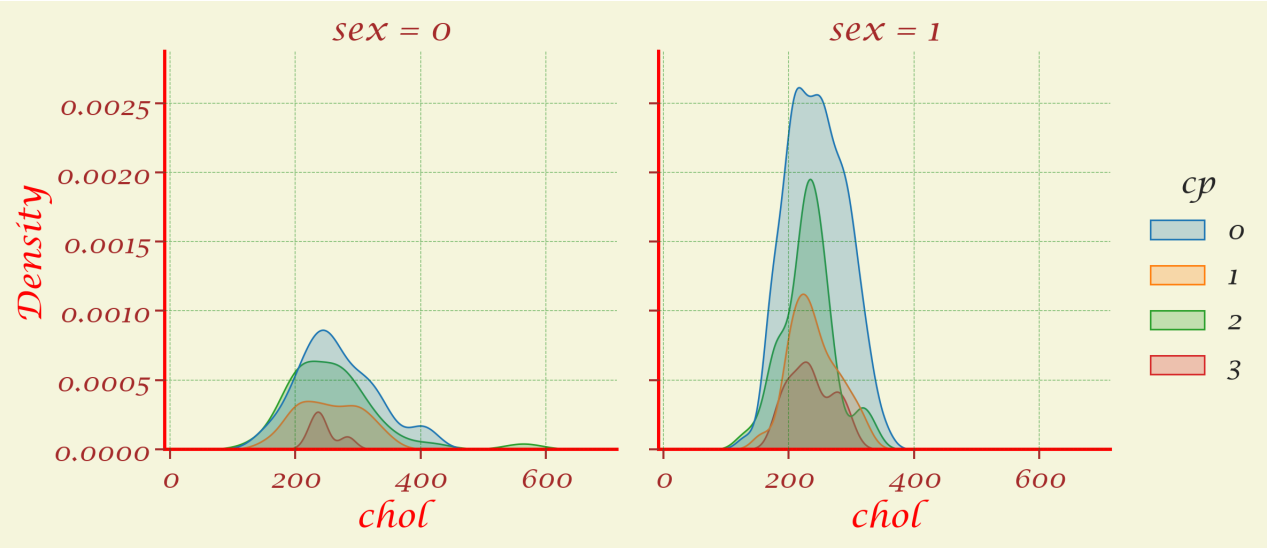
Is chalestrol the real culprit behind it?

In [25]:
```
sns.kdeplot(data=data, x='chol', hue='cp', fill=True, common_norm=False, palette='tab10')
plt.show()
```



The above plot shows that if you don't have chest pain but have a high level of chalestrol the you are prone to risk. And those who have high level of chalestrol also havdifferent types of chest pain. A very visible trend.

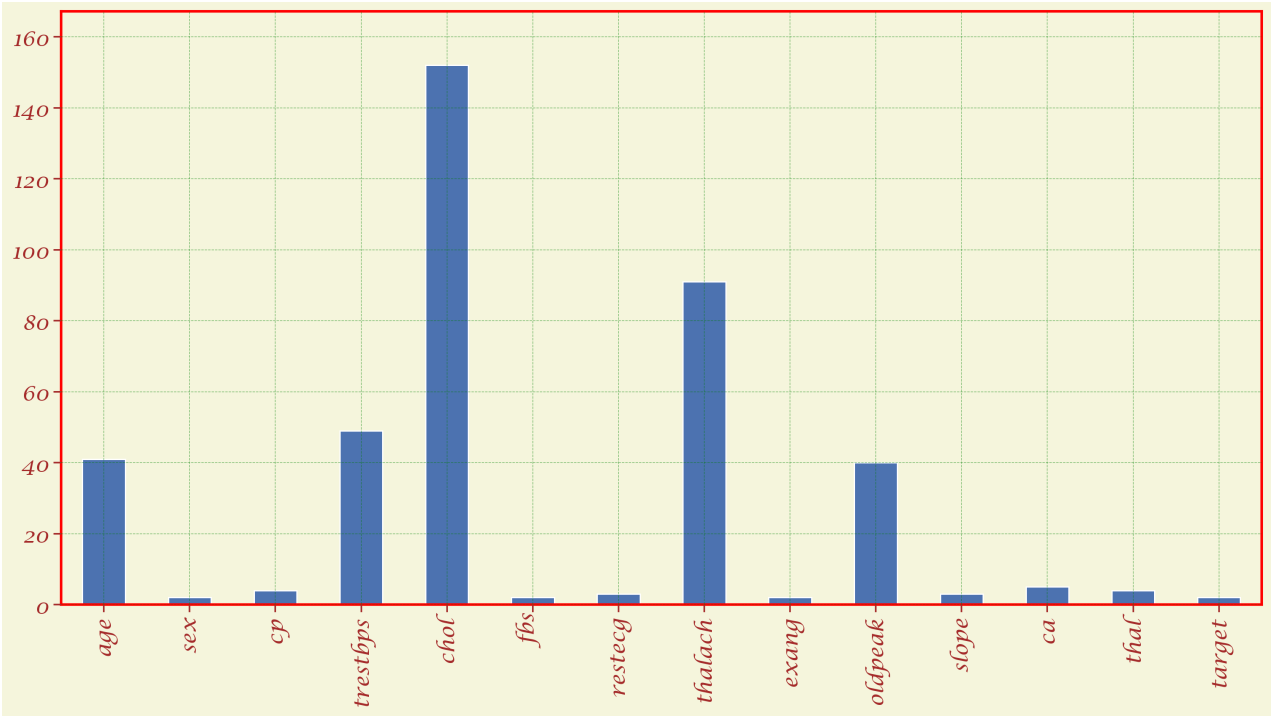
In [26]:
```
sns.displot(data=data, x='chol', hue='cp', kind='kde', col='sex', fill=True, palette='tab10')
plt.show()
```



In [27]:
```
data.nunique()
```

Out[27]:
```
age          41
sex           2
cp            4
trestbps     49
chol        152
fbs           2
restecg       3
thalach      91
exang         2
oldpeak      40
slope         3
ca            5
thal          4
target        2
dtype: int64
```

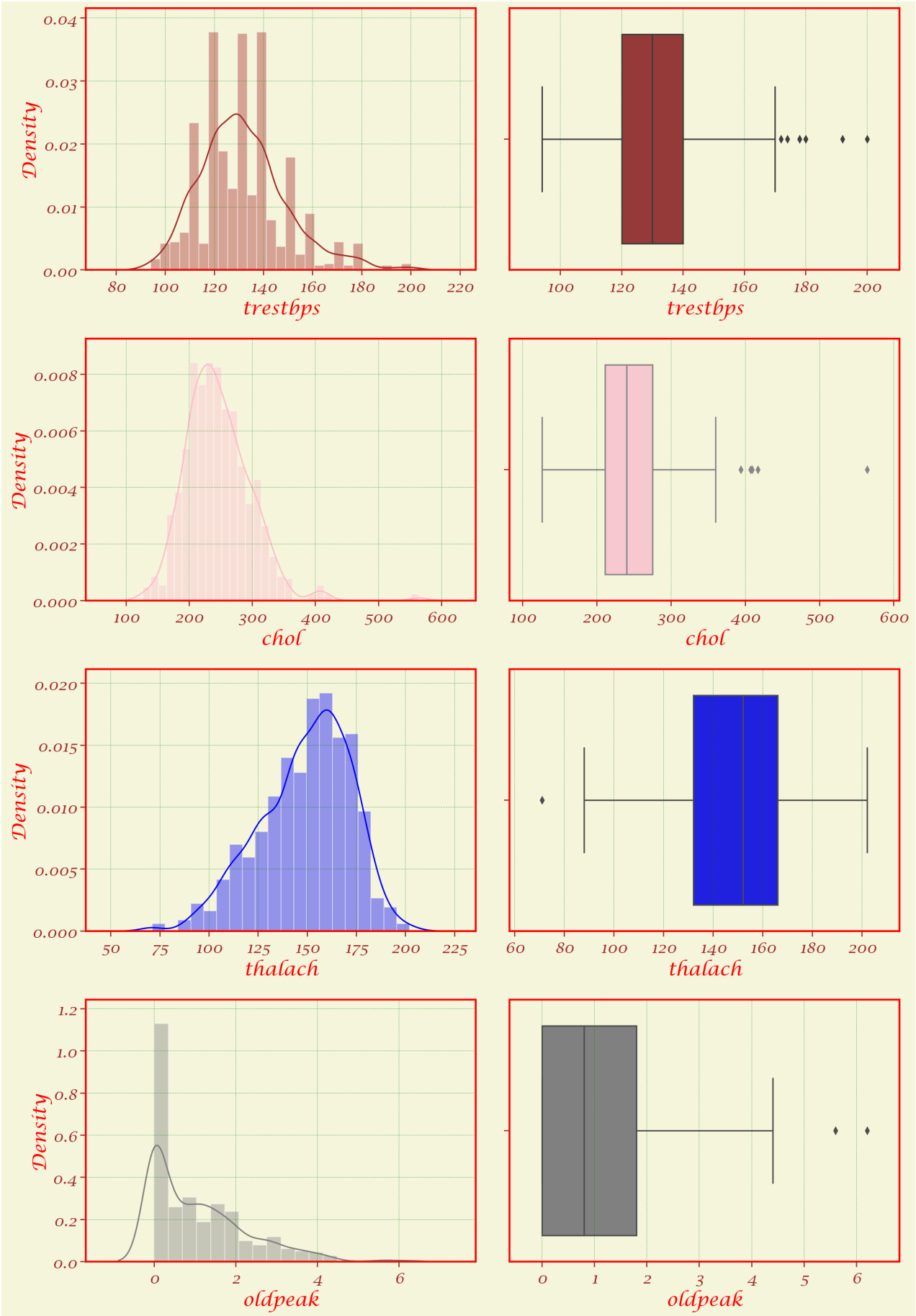
In [28]:
```
data.nunique().plot(kind='bar')
plt.show()
```



10 out of 14 are categorial columns

plot these four columns

In [29]:
```
plt.figure(figsize=(14, 20))
plt.subplot(4, 2, 1)
sns.distplot(data['trestbps'], color='brown')
plt.subplot(4, 2, 2)
sns.boxplot(data=data, x='trestbps', color='brown')
plt.subplot(4, 2, 3)
sns.distplot(data['chol'], color='pink')
plt.subplot(4, 2, 4)
sns.boxplot(data=data, x='chol', color='pink')
plt.subplot(4, 2, 5)
sns.distplot(data['thalach'], color='blue')
plt.subplot(4, 2, 6)
sns.boxplot(data=data, x='thalach', color='blue')
plt.subplot(4, 2, 7)
sns.distplot(data['oldpeak'], color='grey')
plt.subplot(4, 2, 8)
sns.boxplot(data=data, x='oldpeak', color='grey')
plt.tight_layout()
plt.show()
```



Data set has outliers and these are pretty visible. Removing them by using IQR(Inter Quartile Range)

In [30]:
```
Q1 = data.quantile(q=0.15)
Q1
```

Out[30]:
```
age          44.0
sex           0.0
cp            0.0
trestbps    112.0
chol        198.0
fbs           0.0
restecg       0.0
thalach     124.0
exang         0.0
oldpeak       0.0
slope         1.0
ca            0.0
thal          2.0
target        0.0
Name: 0.15, dtype: float64
```

In [31]:
```
Q3 = data.quantile(q=0.85)
Q3
```

Out[31]:
```
age          64.00
sex           1.00
cp            2.00
trestbps    150.00
chol        299.00
fbs           0.00
restecg       1.00
thalach     173.00
exang         1.00
oldpeak       2.34
slope         2.00
ca            2.00
thal          3.00
target        1.00
Name: 0.85, dtype: float64
```

In [32]:
```
IQR = Q3-Q1
IQR.sort_values(ascending=False)
```

Out[32]:
```
chol        101.00
thalach      49.00
trestbps     38.00
age          20.00
oldpeak       2.34
cp            2.00
ca            2.00
sex           1.00
restecg       1.00
exang         1.00
slope         1.00
thal          1.00
target        1.00
fbs           0.00
dtype: float64
```

In [33]:
```
data.shape
```

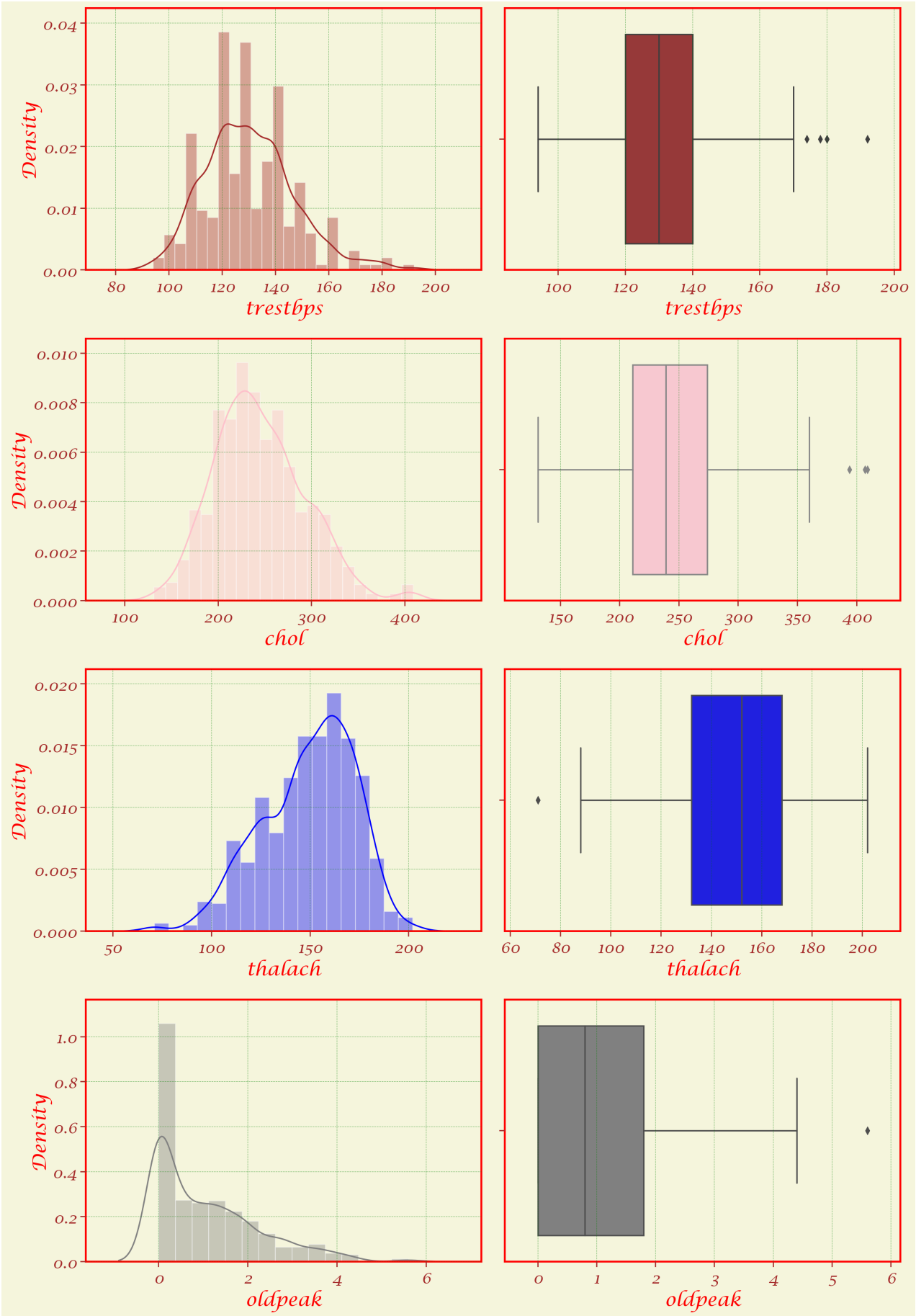
Out[33]:
```
(1025, 14)
```

In [34]:
```
new_data = data[~ ((data < (Q1 -1.5 * IQR)) | (data >(Q3 + 1.5 * IQR))).any(axis=1)]
```

In [35]:
```
new_data.shape
```

Out[35]:
```
(863, 14)
```

In [36]:
```
plt.figure(figsize=(14, 20))
plt.subplot(4, 2, 1)
sns.distplot(new_data['trestbps'], color='brown')
plt.subplot(4, 2, 2)
sns.boxplot(data=new_data, x='trestbps', color='brown')
plt.subplot(4, 2, 3)
sns.distplot(new_data['chol'], color='pink')
plt.subplot(4, 2, 4)
sns.boxplot(data=new_data, x='chol', color='pink')
plt.subplot(4, 2, 5)
sns.distplot(new_data['thalach'], color='blue')
plt.subplot(4, 2, 6)
sns.boxplot(data=new_data, x='thalach', color='blue')
plt.subplot(4, 2, 7)
sns.distplot(new_data['oldpeak'], color='grey')
plt.subplot(4, 2, 8)
sns.boxplot(data=new_data, x='oldpeak', color='grey')
plt.tight_layout()
plt.show()
```



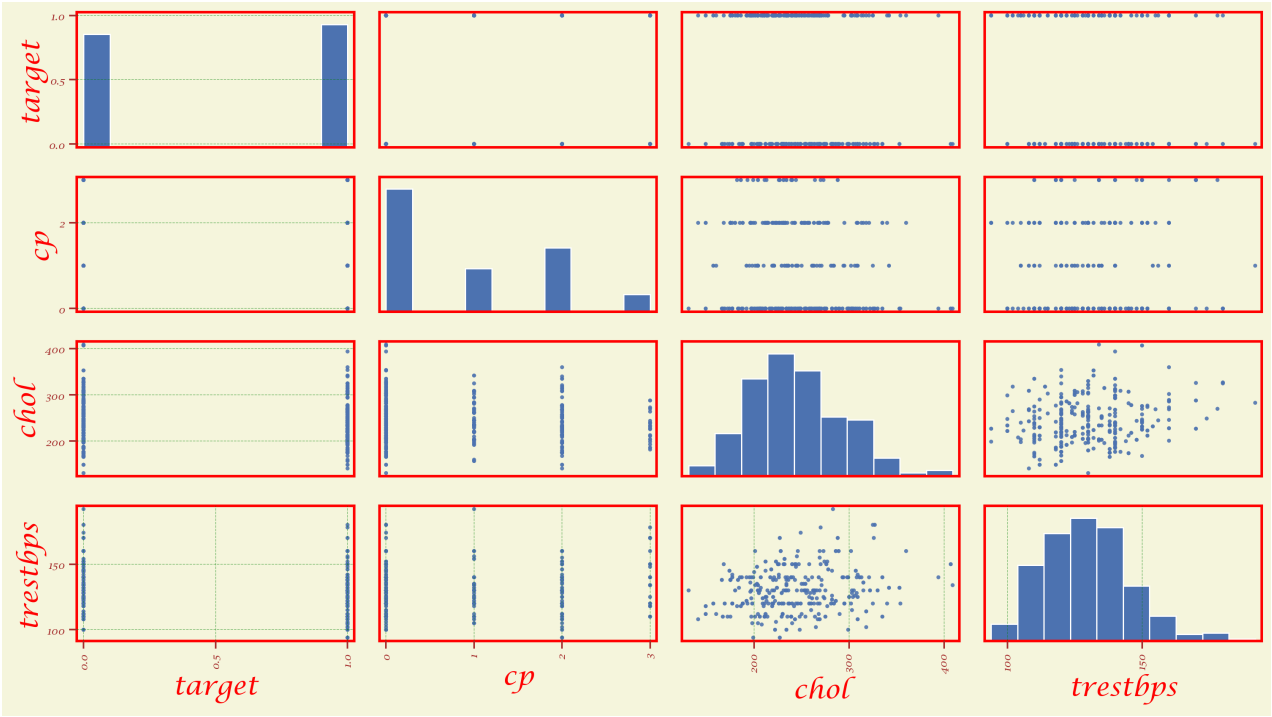
Now our data in much better shape

What about correlation?

In [37]:
```
from pandas.plotting import scatter_matrix
```

In [38]:
```
plt.figure(figsize=(14, 8))
atts = ['target', 'cp', 'chol', 'trestbps']
# Only those features that are not categorial
scatter_matrix(frame=new_data[atts])
plt.show()
```

```
<Figure size 3080x1760 with 0 Axes>
```



In [39]:
```
new_data[atts].corr()
```

Out[39]:

	target	cp	chol	trestbps
target	1.00	0.39	-0.12	-0.11
cp	0.39	1.00	-0.13	0.05
chol	-0.12	-0.13	1.00	0.19
trestbps	-0.11	0.05	0.19	1.00

Only Chest Pain feature has positive correlation with our target

What about other ones

In [40]:
```
data.corr().unstack().sort_values(ascending=False).drop_duplicates()
```

Out[40]:

age	age	1.00e+00
target	cp	4.35e-01
thalach	target	4.23e-01
slope	thalach	3.95e-01
target	slope	3.46e-01
exang	oldpeak	3.11e-01
cp	thalach	3.07e-01
age	ca	2.72e-01
trestbps	age	2.71e-01
ca	oldpeak	2.22e-01
chol	age	2.20e-01
age	oldpeak	2.08e-01
oldpeak	thal	2.03e-01
sex	thal	1.98e-01
exang	thal	1.97e-01
oldpeak	trestbps	1.87e-01
fbs	trestbps	1.82e-01
ca	thal	1.49e-01
sex	exang	1.39e-01
fbs	ca	1.37e-01
target	restecg	1.34e-01
cp	slope	1.32e-01
trestbps	chol	1.28e-01
age	fbs	1.21e-01
ca	sex	1.12e-01
exang	ca	1.08e-01
trestbps	ca	1.05e-01
thal	chol	1.00e-01
exang	age	8.82e-02
slope	restecg	8.61e-02
sex	oldpeak	8.47e-02
fbs	cp	7.93e-02
chol	ca	7.43e-02
age	thal	7.23e-02
exang	chol	6.74e-02
oldpeak	chol	6.49e-02
exang	trestbps	6.12e-02
thal	trestbps	5.93e-02
exang	fbs	4.93e-02
restecg	thalach	4.84e-02
cp	restecg	4.36e-02
trestbps	cp	3.82e-02
fbs	sex	2.72e-02
	chol	2.69e-02
	oldpeak	1.09e-02
	thalach	-8.87e-03
slope	chol	-1.42e-02
thal	restecg	-2.05e-02
thalach	chol	-2.18e-02
sex	slope	-2.67e-02
trestbps	thalach	-3.93e-02
sex	cp	-4.11e-02
fbs	target	-4.12e-02
thal	fbs	-4.22e-02
thalach	sex	-4.94e-02
oldpeak	restecg	-5.01e-02
sex	restecg	-5.51e-02
fbs	slope	-6.19e-02
exang	restecg	-6.56e-02
age	cp	-7.20e-02
ca	slope	-7.34e-02
	restecg	-7.81e-02
sex	trestbps	-7.90e-02
cp	chol	-8.16e-02
slope	thal	-9.41e-02
thal	thalach	-9.81e-02
target	chol	-1.00e-01
sex	age	-1.03e-01
fbs	restecg	-1.04e-01
slope	trestbps	-1.20e-01
trestbps	restecg	-1.24e-01
restecg	age	-1.33e-01
target	trestbps	-1.39e-01
chol	restecg	-1.47e-01
cp	thal	-1.63e-01
slope	age	-1.69e-01
cp	oldpeak	-1.75e-01
	ca	-1.76e-01
sex	chol	-1.98e-01
ca	thalach	-2.08e-01
target	age	-2.29e-01
slope	exang	-2.67e-01
target	sex	-2.80e-01
	thal	-3.38e-01
thalach	oldpeak	-3.50e-01
exang	thalach	-3.80e-01
ca	target	-3.82e-01
age	thalach	-3.90e-01
exang	cp	-4.02e-01
target	exang	-4.38e-01
oldpeak	target	-4.38e-01

```
slope     oldpeak    -5.75e-01
dtype: float64
```

In [41]:
```
corr = new_data.groupby(['cp'])[['age', 'trestbps']].corr()
```

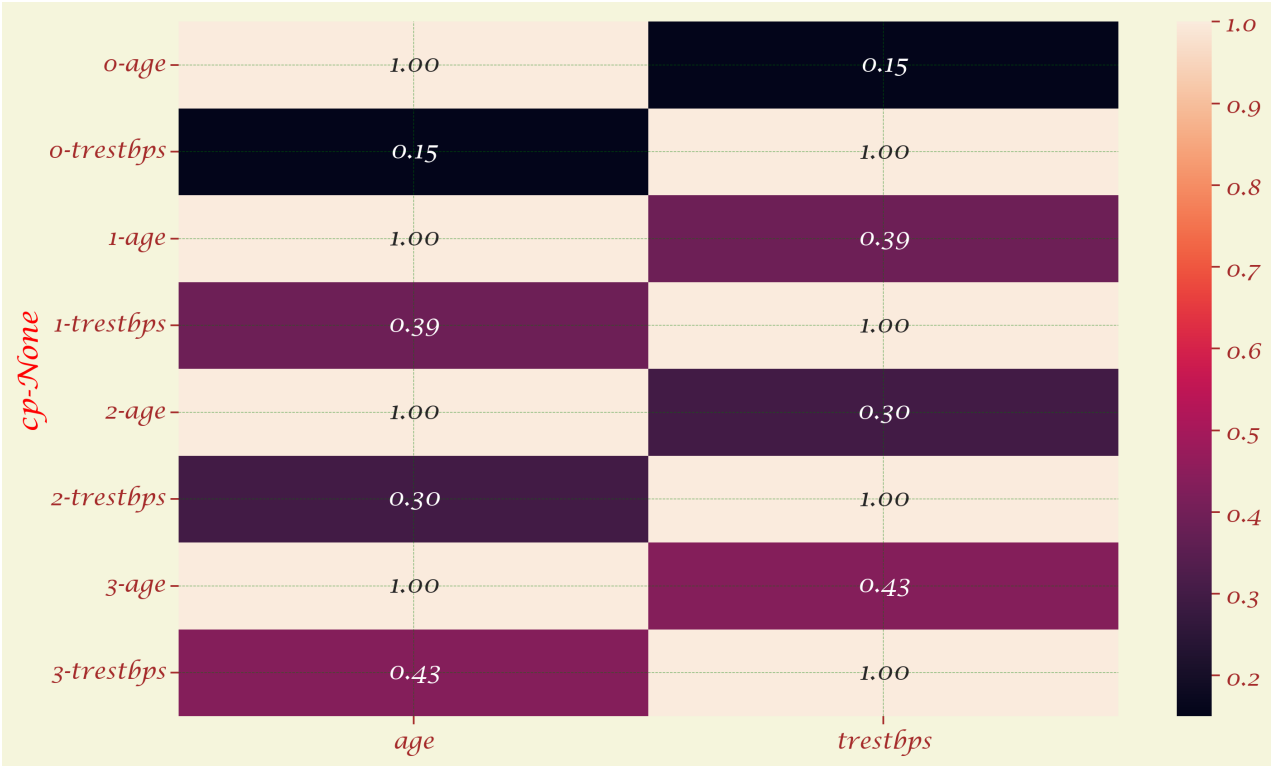
In [42]:
```
corr
```

Out[42]:

		age	trestbps
cp			
0	**age**	1.00	0.15
	trestbps	0.15	1.00
1	**age**	1.00	0.39
	trestbps	0.39	1.00
2	**age**	1.00	0.30
	trestbps	0.30	1.00
3	**age**	1.00	0.43
	trestbps	0.43	1.00

Chest pain is effecting the bllod pressure of a person

In [43]:
```
sns.heatmap(data=corr, annot=True, fmt='.2f')
plt.show()
```



In [44]:
```
print(f"Chest Pain in Women {new_data['cp'][new_data['sex'] == 0].value_counts(normalize=True) * 100}")
```

```
Chest Pain in Women cp
0    42.53
2    34.48
1    19.16
3     3.83
Name: proportion, dtype: float64
```

In [45]:
```
print(f"Chest Pain in Men {new_data['cp'][new_data['sex'] == 1].value_counts(normalize=True) * 100}")
```

```
Chest Pain in Men cp
0    52.66
2    22.09
1    16.78
3     8.47
Name: proportion, dtype: float64
```

Majority of the persons have no chest pain

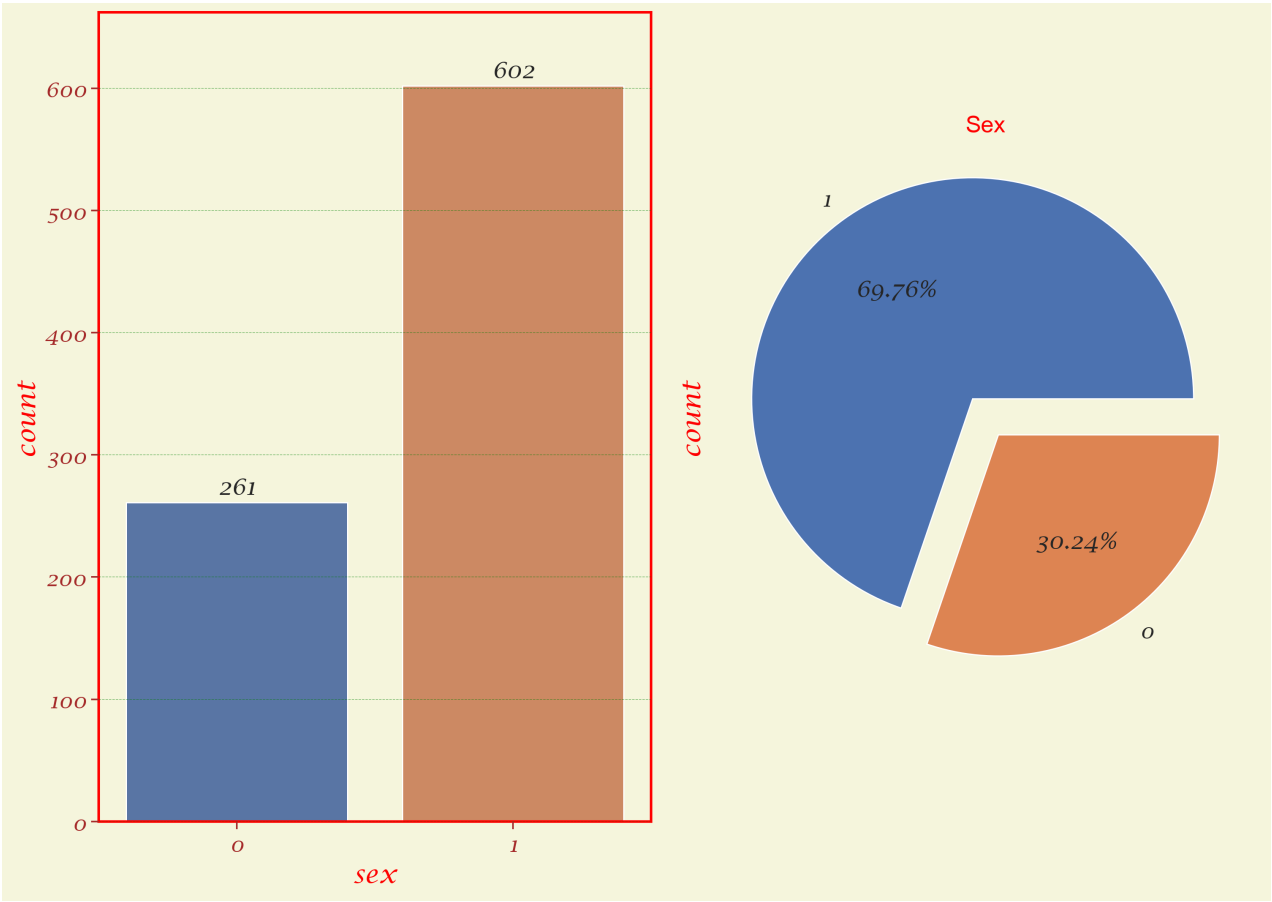
Now check out is there any wrong value

In [46]:
```
new_data.describe().T
```

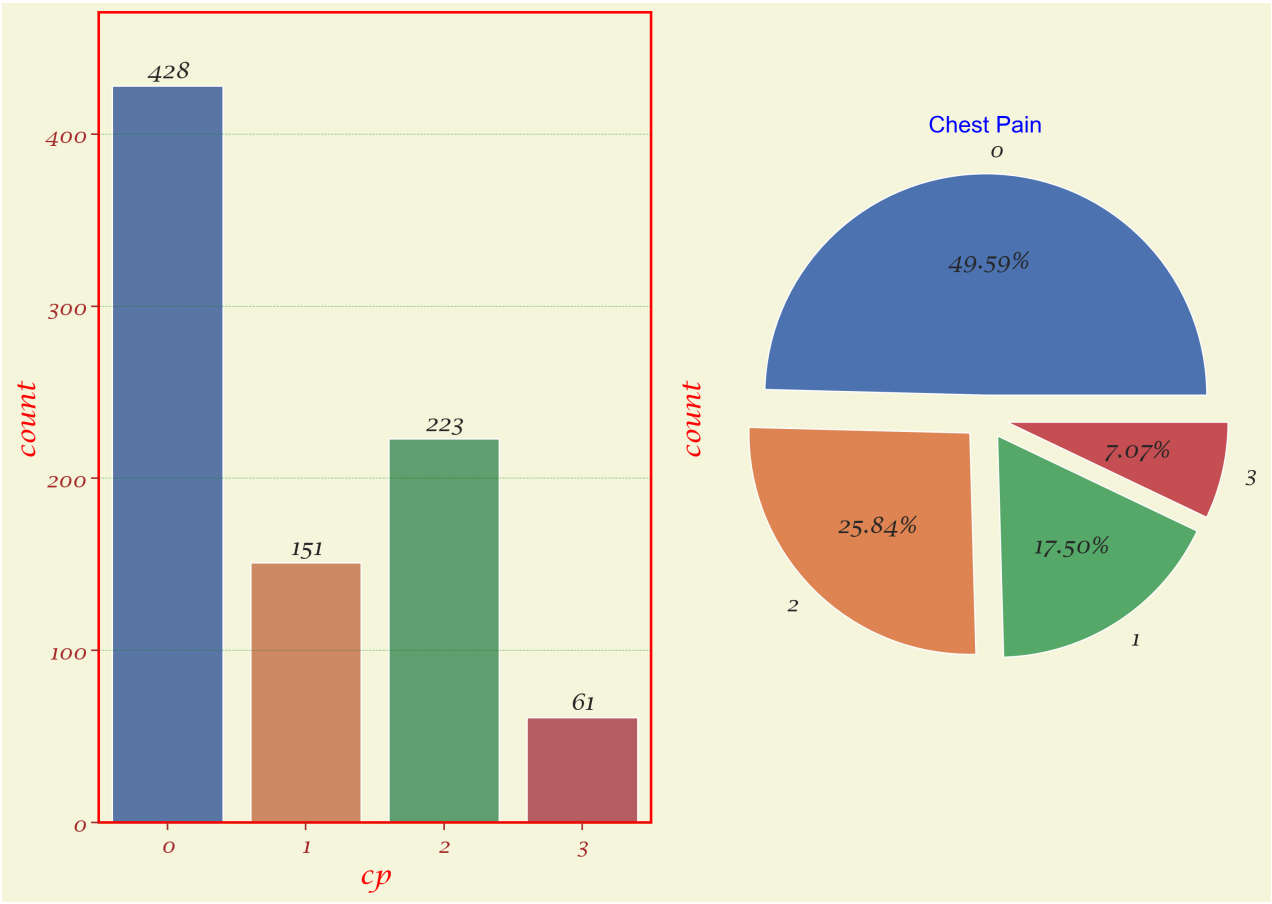
Out[46]:

	count	mean	std	min	25%	50%	75%	max
age	863.0	53.90	9.33	29.0	46.0	55.0	61.0	77.0
sex	863.0	0.70	0.46	0.0	0.0	1.0	1.0	1.0
cp	863.0	0.90	1.01	0.0	0.0	1.0	2.0	3.0
trestbps	863.0	130.24	16.61	94.0	120.0	130.0	140.0	192.0
chol	863.0	244.70	48.38	131.0	211.0	239.0	274.0	409.0
fbs	863.0	0.00	0.00	0.0	0.0	0.0	0.0	0.0
restecg	863.0	0.56	0.53	0.0	0.0	1.0	1.0	2.0
thalach	863.0	149.30	23.28	71.0	132.0	152.0	168.0	202.0
exang	863.0	0.33	0.47	0.0	0.0	0.0	1.0	1.0
oldpeak	863.0	1.05	1.16	0.0	0.0	0.8	1.8	5.6
slope	863.0	1.41	0.60	0.0	1.0	1.0	2.0	2.0
ca	863.0	0.69	1.00	0.0	0.0	0.0	1.0	4.0
thal	863.0	2.34	0.58	1.0	2.0	2.0	3.0	3.0
target	863.0	0.52	0.50	0.0	0.0	1.0	1.0	1.0

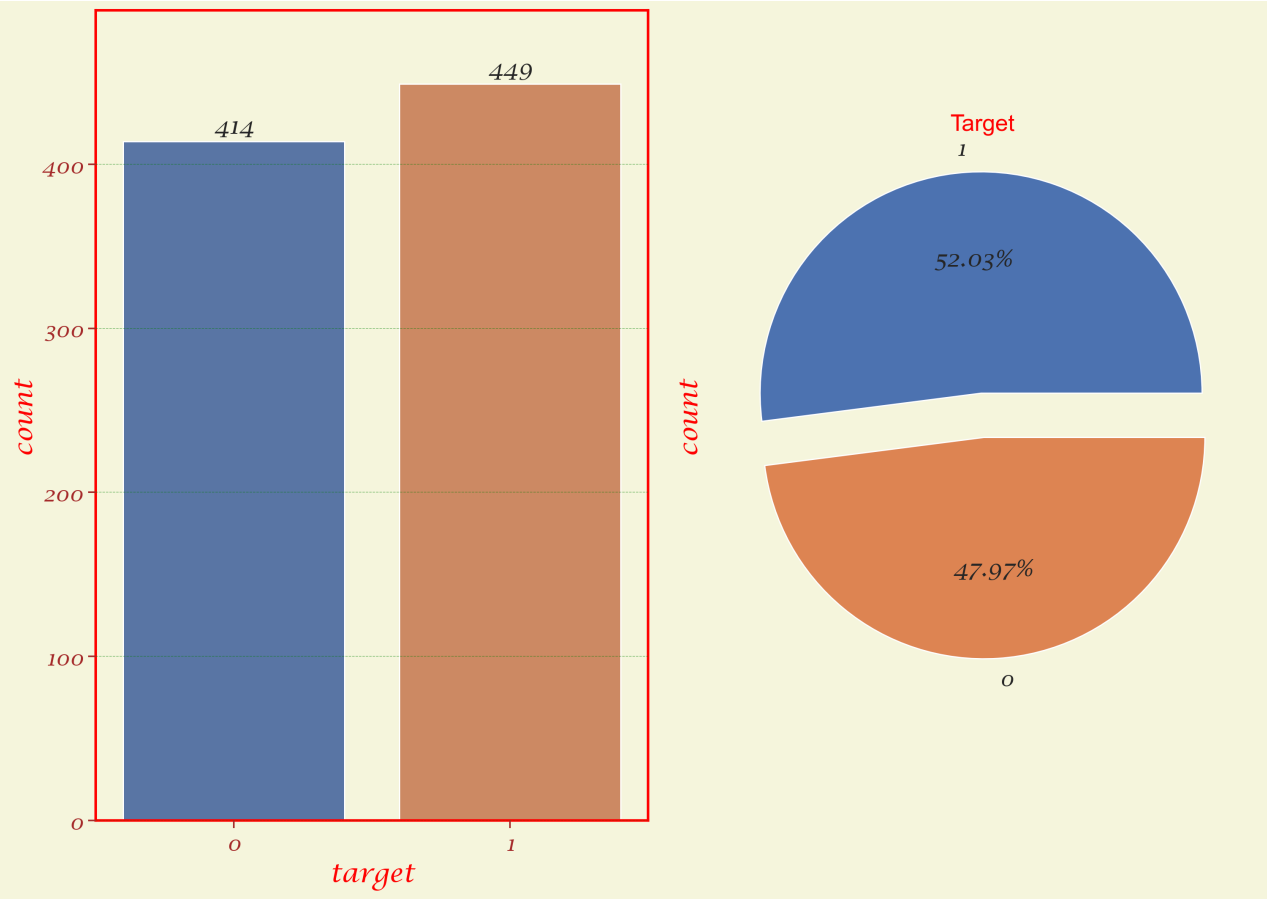
In [47]:
```
plt.figure(figsize=(14, 10))
ax = plt.subplot(1, 2, 1)
ax = sns.countplot(data=new_data, x='sex')
ax.bar_label(ax.containers[0])
plt.subplot(1, 2, 2)
ax = new_data['sex'].value_counts().plot(kind='pie', explode=[0.1, 0.1], autopct='%1.2f%%')
ax.set_title(label='Sex', fontsize=20, color='Red', font='Sans serif')
plt.show()
```



In [48]:
```
plt.figure(figsize=(14, 10))
ax = plt.subplot(1, 2, 1)
ax = sns.countplot(data=new_data, x='cp')
ax.bar_label(ax.containers[0])
plt.subplot(1, 2, 2)
ax = new_data['cp'].value_counts().plot(kind='pie', explode=[0.1, 0.1, 0.1, 0.1], autopct='%1.2f%%')
ax.set_title(label='Chest Pain', fontsize=20, color='Blue', font='Sans serif')
plt.show()
```



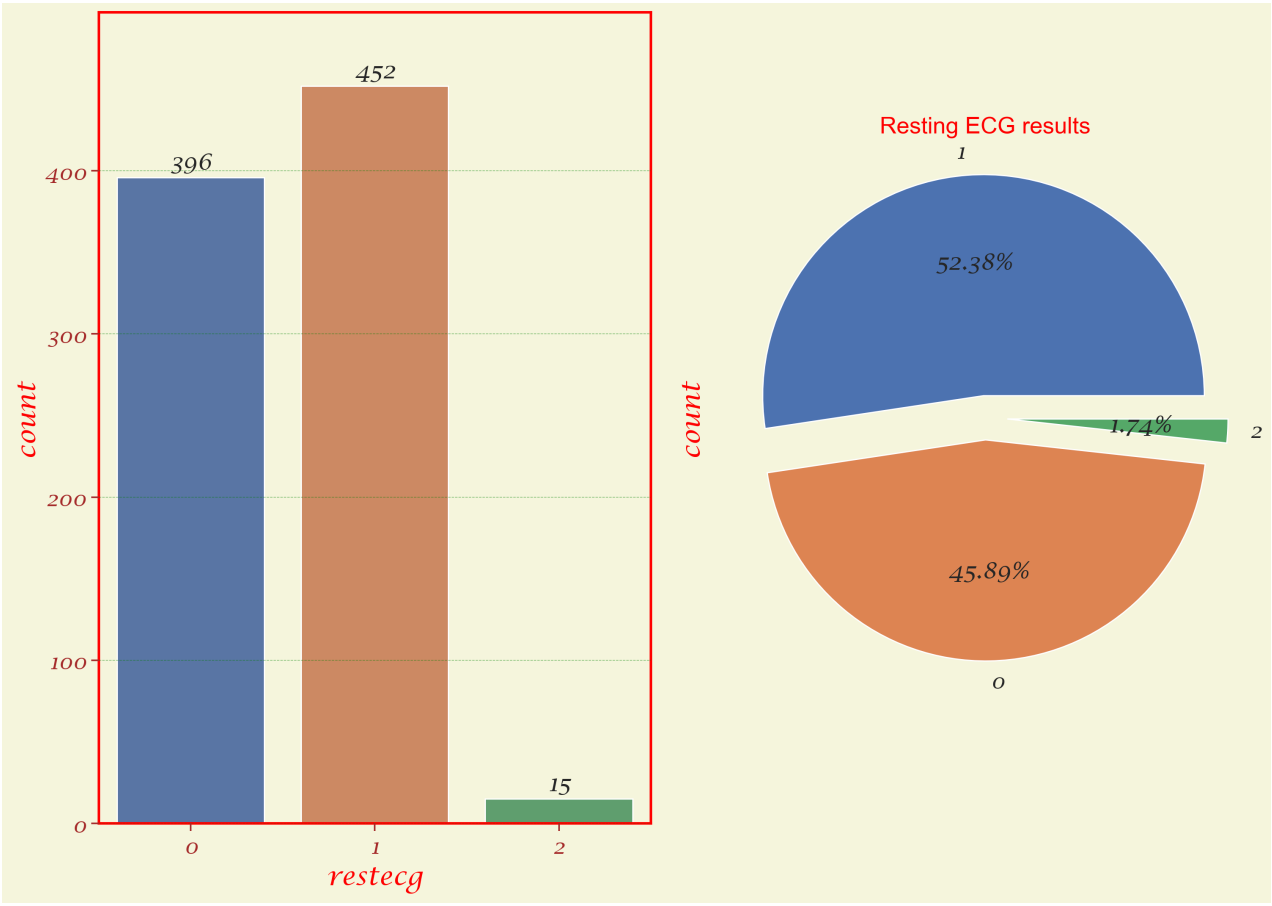
In [49]:
```
plt.figure(figsize=(14, 10))
ax = plt.subplot(1, 2, 1)
ax = sns.countplot(data=new_data, x='target')
ax.bar_label(ax.containers[0])
plt.subplot(1, 2, 2)
ax = new_data['target'].value_counts().plot(kind='pie', explode=[0.1, 0.1], autopct='%1.2f%%')
ax.set_title(label='Target', fontsize=20, color='Red', font='Sans serif')
plt.show()
```



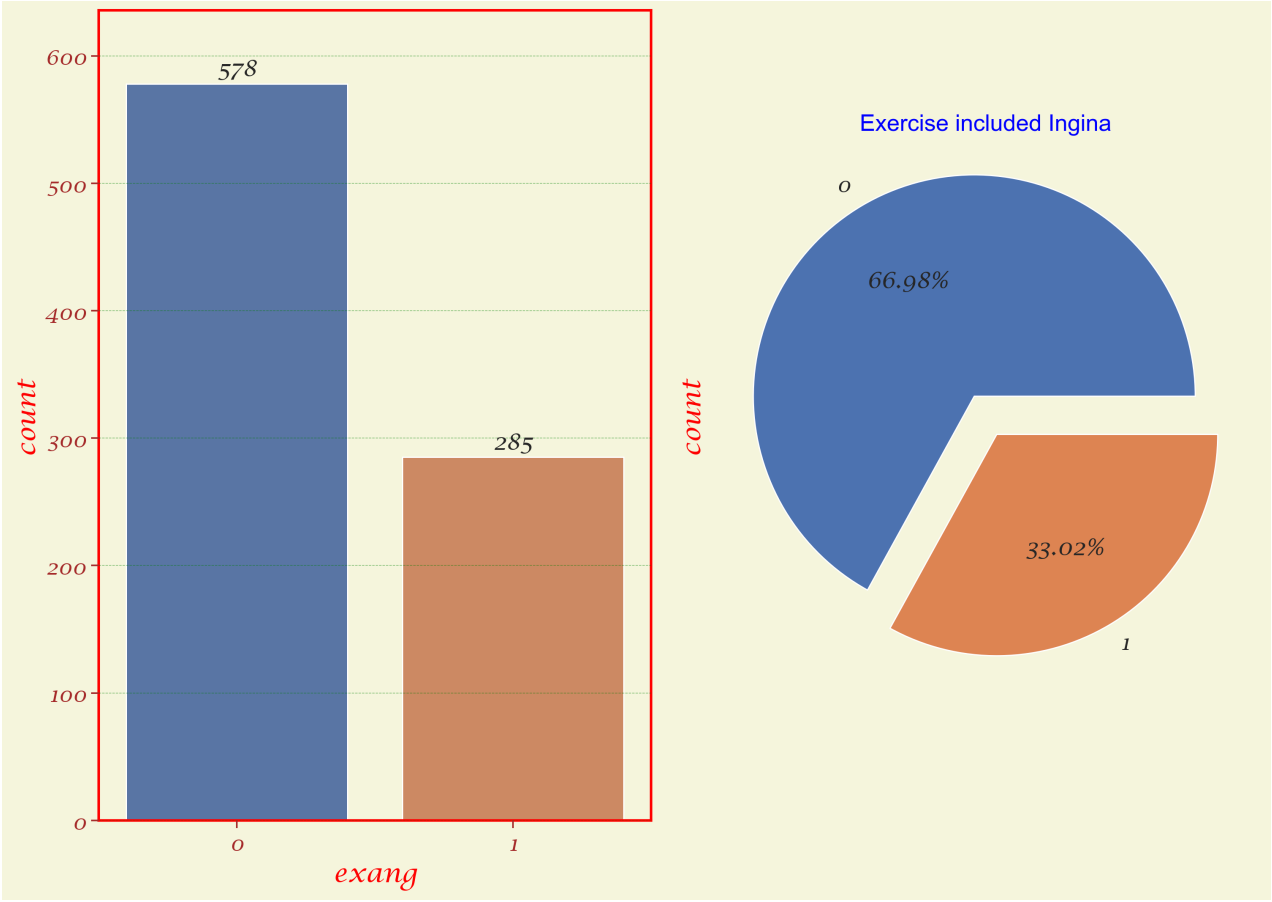
In [50]:
```
new_data['restecg'].value_counts()
```

Out[50]:
```
restecg
1    452
0    396
2     15
Name: count, dtype: int64
```

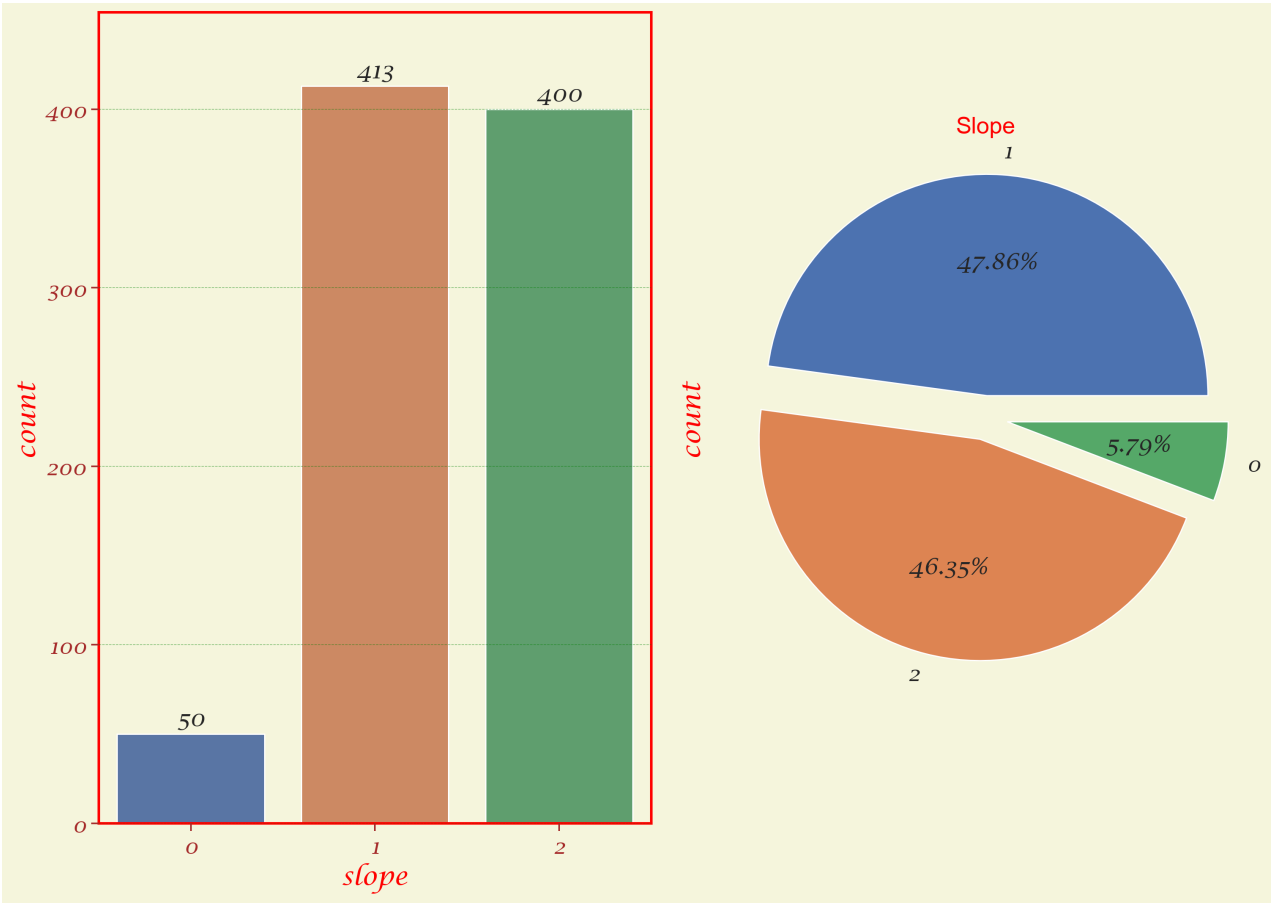
In [51]:
```
plt.figure(figsize=(14, 10))
ax = plt.subplot(1, 2, 1)
ax = sns.countplot(data=new_data, x='restecg')
ax.bar_label(ax.containers[0])
plt.subplot(1, 2, 2)
ax = new_data['restecg'].value_counts().plot(kind='pie', explode=[0.1, 0.1, 0.1], autopct='%1.2f%%')
ax.set_title(label='Resting ECG results', fontsize=20, color='Red', font='Sans serif')
plt.show()
```



In [52]:
```
plt.figure(figsize=(14, 10))
ax = plt.subplot(1, 2, 1)
ax = sns.countplot(data=new_data, x='exang')
ax.bar_label(ax.containers[0])
plt.subplot(1, 2, 2)
ax = new_data['exang'].value_counts().plot(kind='pie', explode=[0.1, 0.1], autopct='%1.2f%%')
ax.set_title(label='Exercise included Ingina', fontsize=20, color='Blue', font='Sans serif')
plt.show()
```



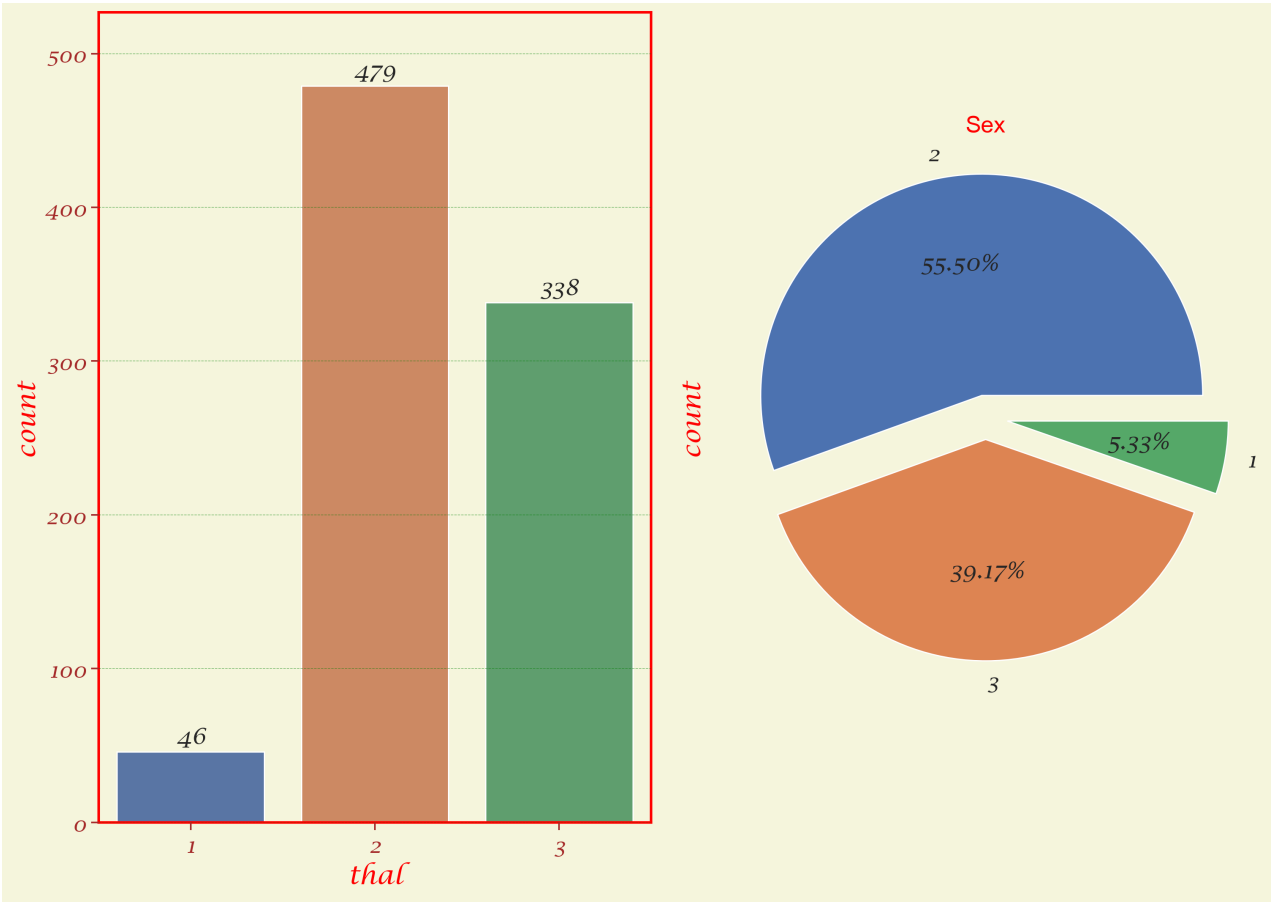
In [53]:
```
plt.figure(figsize=(14, 10))
ax = plt.subplot(1, 2, 1)
ax = sns.countplot(data=new_data, x='slope')
ax.bar_label(ax.containers[0])
plt.subplot(1, 2, 2)
ax = new_data['slope'].value_counts().plot(kind='pie', explode=[0.1, 0.1, 0.1], autopct='%1.2f%%')
ax.set_title(label='Slope', fontsize=20, color='Red', font='Sans serif')
plt.show()
```



In [54]:
```
new_data['thal'].value_counts()
```

Out[54]:
```
thal
2    479
3    338
1     46
Name: count, dtype: int64
```

In [55]:
```
plt.figure(figsize=(14, 10))
ax = plt.subplot(1, 2, 1)
ax = sns.countplot(data=new_data, x='thal')
ax.bar_label(ax.containers[0])
plt.subplot(1, 2, 2)
ax = new_data['thal'].value_counts().plot(kind='pie', explode=[0.1, 0.1, 0.1], autopct='%1.2f%%')
ax.set_title(label='Sex', fontsize=20, color='Red', font='Sans serif')
plt.show()
```



We have done data analysis pretty much.

Now do some feature engineering

The resting blood pressure and Chlestrol could be arranged into categorial features

In [56]:
```
new_data['chlestrol_by_category'] = pd.cut(new_data['chol'], bins=[0, 200, 230, 500], labels=['Normal', 'Borderline', 'High']
print('Value counts of Chlestrol column', new_data['chlestrol_by_category'].value_counts())
```

```
Value counts of Chlestrol column chlestrol_by_category
High          501
Borderline    214
Normal        148
Name: count, dtype: int64
```

In [57]:
```
new_data['blood_pressure_category'] = pd.cut(new_data.trestbps, bins=[0, 120, 129, 139, 200], labels=['Normal_BP', 'Eleved_BP
                                                                                                       'Hytertention_stage2'])
print('Value counts of Blood Column', new_data['blood_pressure_category'].value_counts())
```

```
Value counts of Blood Column blood_pressure_category
Normal_BP              306
Hytertention_stage2    261
Hypertension_stage1    198
Eleved_BP               98
Name: count, dtype: int64
```

In [58]:
```
new_data.chlestrol_by_category.dtype
```

Out[58]:
```
CategoricalDtype(categories=['Normal', 'Borderline', 'High'], ordered=True)
```

In [59]:
```
new_data.chlestrol_by_category = new_data.chlestrol_by_category.astype(object)
```

In [60]:
```
new_data.chlestrol_by_category.dtype
```

Out[60]:
```
dtype('O')
```

In [61]:
```
new_data.blood_pressure_category = new_data.blood_pressure_category.astype(object)
```

In [62]:
```
new_data.blood_pressure_category.dtype
```

Out[62]:
```
dtype('O')
```

In [63]:
```
num, cat = list(), list()
for label, content in new_data.items():
    if pd.api.types.is_numeric_dtype(content):
        num.append(label)
    else:
        cat.append(label)
```

In [64]:
```
num
```

Out[64]:
```
['age',
 'sex',
 'cp',
 'trestbps',
 'chol',
 'fbs',
 'restecg',
 'thalach',
 'exang',
 'oldpeak',
 'slope',
 'ca',
 'thal',
 'target']
```

In [65]:
```
cat
```

Out[65]:
```
['chlestrol_by_category', 'blood_pressure_category']
```

In [66]:
```
import category_encoders as ce
encorder = ce.OrdinalEncoder(cols=cat)
new_data = encorder.fit_transform(new_data)
```

In [67]:
```
new_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 863 entries, 0 to 1024
Data columns (total 16 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   age                      863 non-null    int64
 1   sex                      863 non-null    int64
 2   cp                       863 non-null    int64
 3   trestbps                 863 non-null    int64
 4   chol                     863 non-null    int64
 5   fbs                      863 non-null    int64
 6   restecg                  863 non-null    int64
 7   thalach                  863 non-null    int64
 8   exang                    863 non-null    int64
 9   oldpeak                  863 non-null    float64
 10  slope                    863 non-null    int64
 11  ca                       863 non-null    int64
 12  thal                     863 non-null    int64
 13  target                   863 non-null    int64
 14  chlestrol_by_category    863 non-null    int32
 15  blood_pressure_category  863 non-null    int32
dtypes: float64(1), int32(2), int64(13)
memory usage: 107.9 KB
```

In [68]:
```
X, y = new_data.drop('target', axis=1), new_data['target']
```

In [69]:
```
X.shape, y.shape
```

Out[69]:
```
((863, 15), (863,))
```

In [70]:
```
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_predict, cross_val_score, learning_curve
from sklearn.preprocessing import MinMaxScaler, OrdinalEncoder
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, f1_score, precision_recall_curve
from sklearn.metrics import recall_score, precision_score, roc_auc_score, roc_curve
```

In [71]:
```
minmax = MinMaxScaler(feature_range=(-1, 1))
X_new = minmax.fit_transform(X)
```

In [72]:
```
X = pd.DataFrame(data=X_new, columns=X.columns)
```

In [73]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [74]:
```
X_train.shape, X_train.shape, y_train.shape, y_test.shape
```

Out[74]:
```
((690, 15), (690, 15), (690,), (173,))
```

#1 Logistic Regression

In [75]:
```
lin_cfs = LogisticRegression()
lin_cfs.fit(X_train, y_train)
lin_cfs_score = lin_cfs.score(X_train, y_train)
lin_cfs_cross_val_accuracy = cross_val_score(lin_cfs, X_train, y_train, cv=5, scoring='accuracy', n_jobs=-1)
lin_cfs_cross_val_predict = cross_val_predict(lin_cfs, X_train, y_train, cv=5, n_jobs=-1)
lin_cfs_decision_score = cross_val_predict(lin_cfs, X_train, y_train, cv=5, n_jobs=-1, method='decision_function')
```

In [76]:
```
lin_cfs_score
```

Out[76]:
```
0.8608695652173913
```

In [77]:
```
lin_cfs_cross_val_accuracy
```

Out[77]:
```
array([0.83333333, 0.82608696, 0.84057971, 0.84057971, 0.88405797])
```

In [78]:
```
lin_cfs_cross_val_accuracy.mean()
```

Out[78]:
```
0.8449275362318842
```

In [79]:
```
confusion_matrix(y_train, lin_cfs_cross_val_predict)
```

Out[79]:
```
array([[278,  63],
       [ 44, 305]], dtype=int64)
```

In [80]:
```
precision_score(y_train, lin_cfs_cross_val_predict)
```

Out[80]:
```
0.8288043478260869
```

In [81]:
```
recall_score(y_train, lin_cfs_cross_val_predict)
```

Out[81]:
```
0.8739255014326648
```

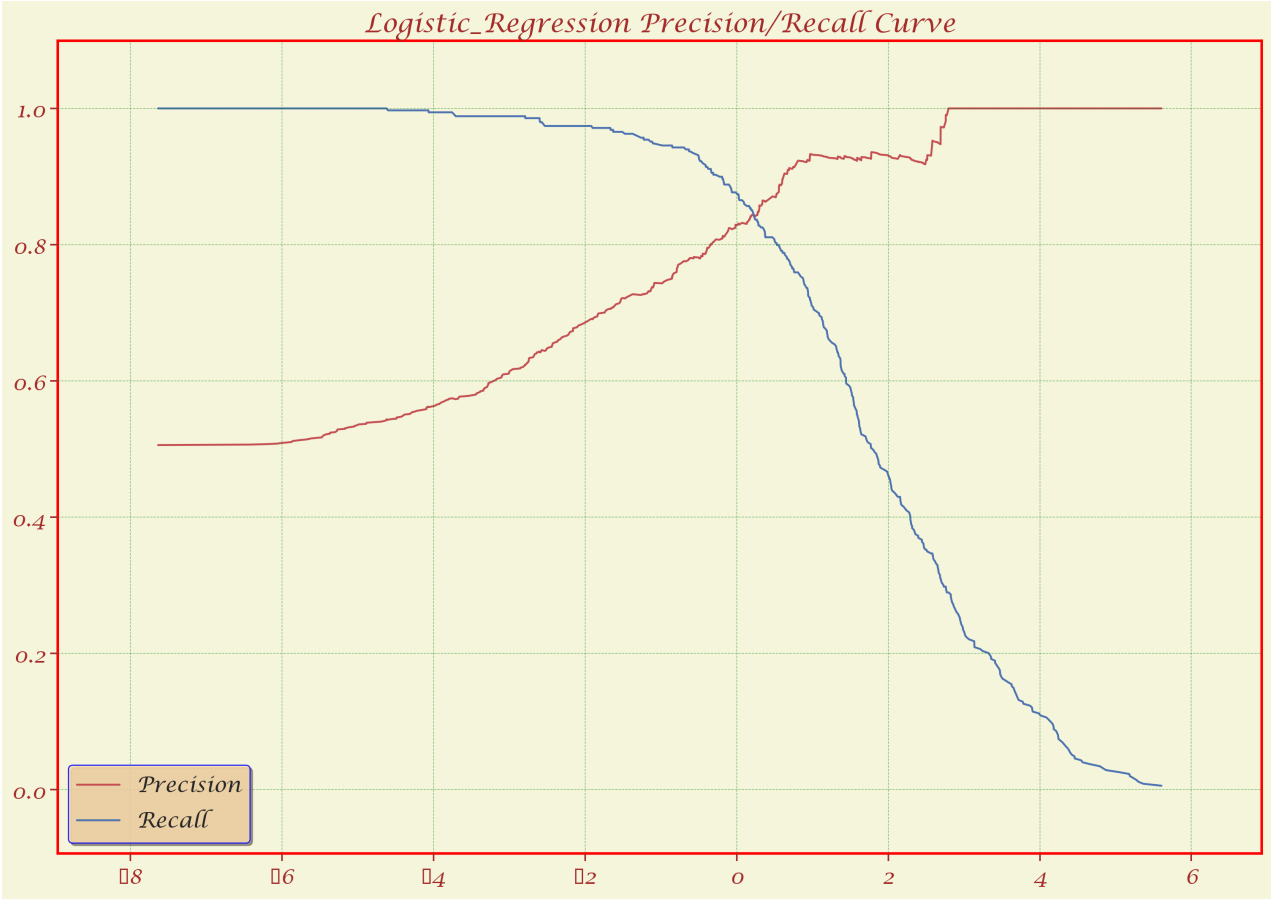
In [82]:
```
print(classification_report(y_train, lin_cfs_cross_val_predict))
```

```
              precision    recall  f1-score   support

           0       0.86      0.82      0.84       341
           1       0.83      0.87      0.85       349

    accuracy                           0.84       690
   macro avg       0.85      0.84      0.84       690
weighted avg       0.85      0.84      0.84       690
```

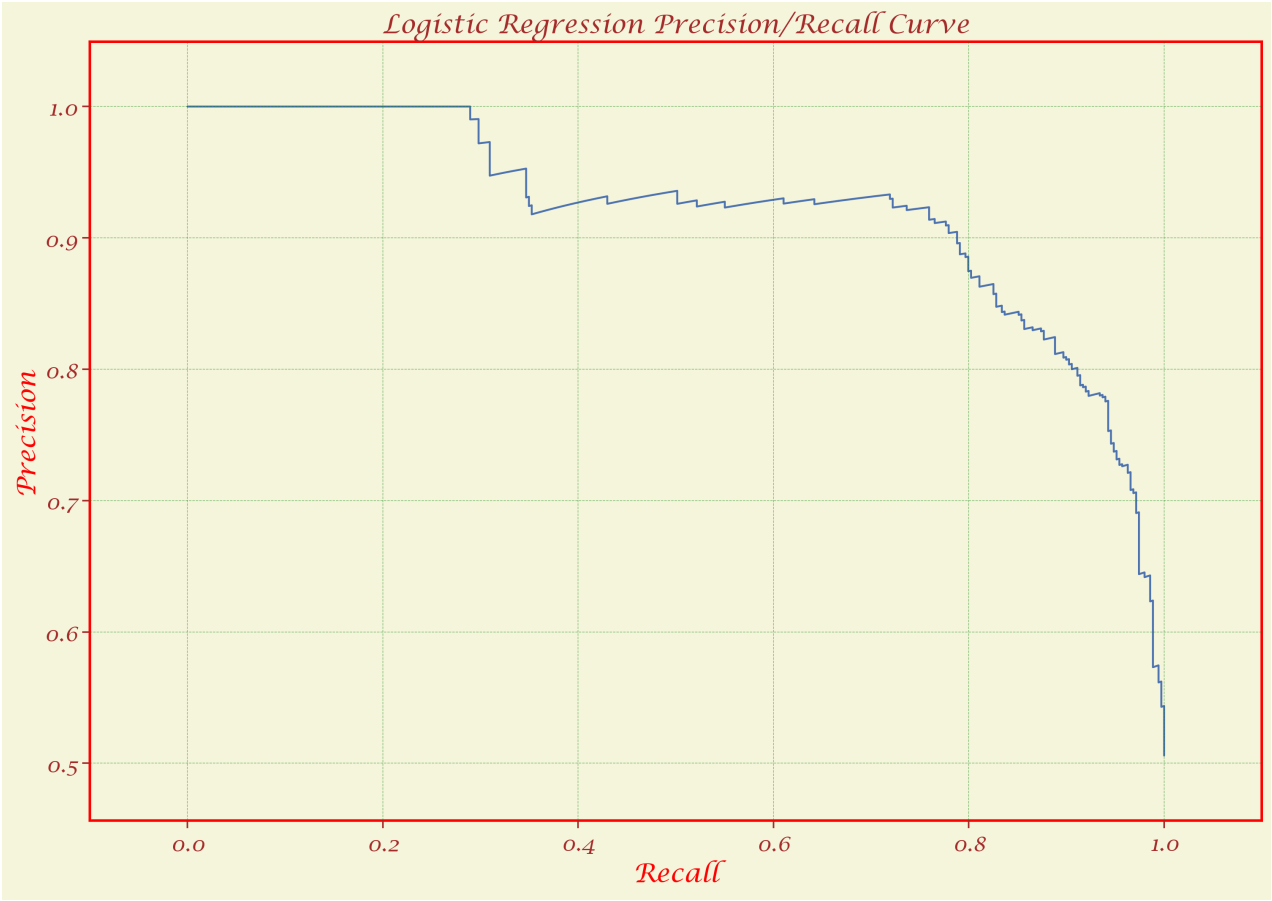
In [83]:
```
precision_, recall_, thresholds_ = precision_recall_curve(y_train, lin_cfs_decision_score)
fpr, tpr, threshold = roc_curve(y_train, lin_cfs_decision_score)
```

In [84]:
```
def p_r_t_curve(precision, recall, threshold, model_name):
    plt.figure(figsize=(14, 10))
    plt.title(f'{model_name} Precision/Recall Curve')
    plt.plot(threshold, precision[:-1], 'r-', label='Precision')
    plt.plot(threshold, recall[:-1], 'b-', label='Recall')

    plt.legend()
    plt.show()

def rc(precision, recall, model_name):
    plt.figure(figsize=(14, 10))
    plt.title(f'{model_name} Precision/Recall Curve')
    plt.plot(recall, precision)
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.show()
def roc_curve_plot(fpr, tpr, threshold, model_name):
    plt.figure(figsize=(14, 10))
    plt.title(f'{model_name} ROC Curve')
    plt.plot(fpr, tpr, label='ROC Curve')
    plt.plot([0, 1], [0, 1], 'r-+', label='Random Classifier\'s boundary')
    plt.axis([0, 1, 0, 1])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend()
    plt.show()
```

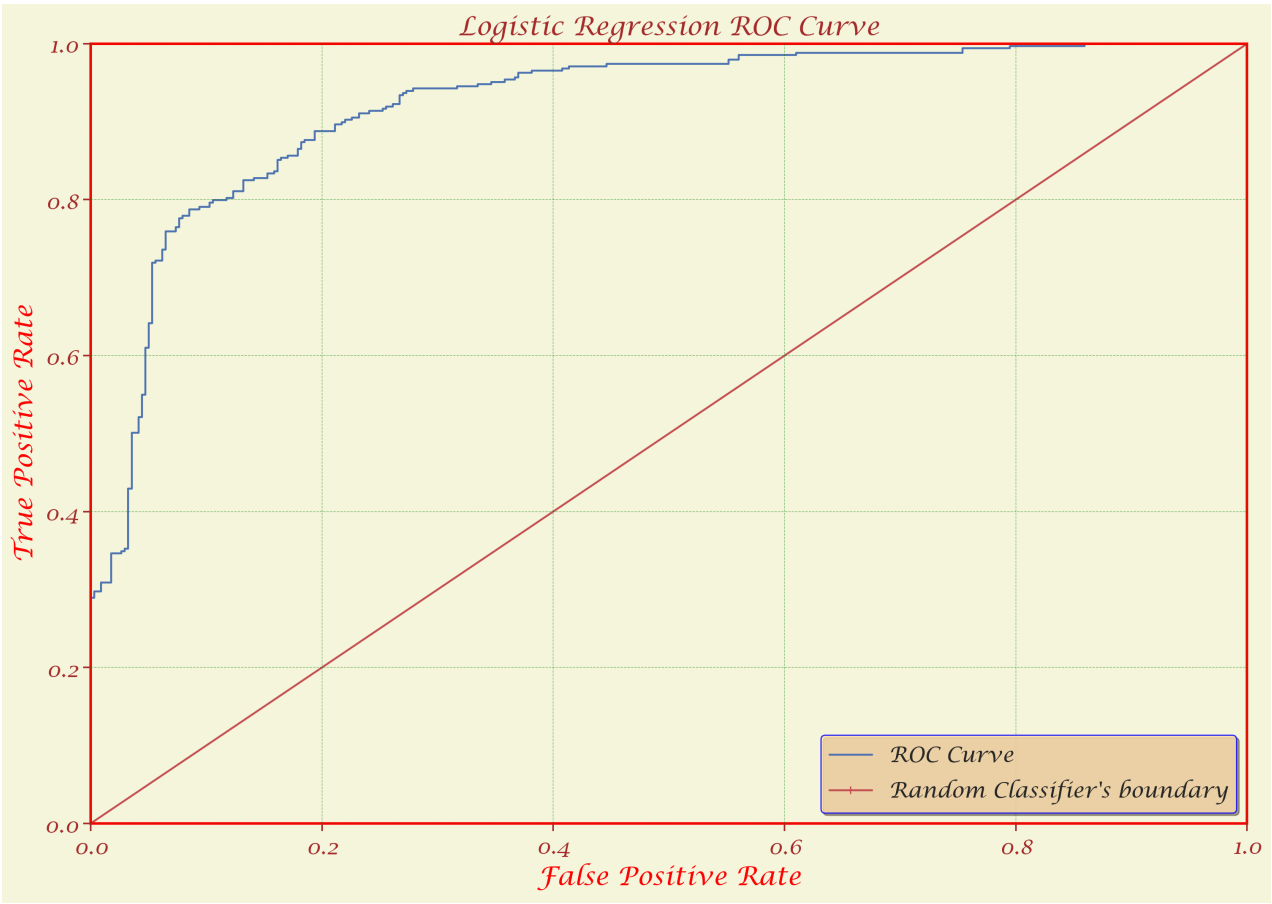
In [85]:
```
p_r_t_curve(precision_, recall_, thresholds_, 'Logistic_Regression')
```



In [86]:
```
rc(precision_, recall_, 'Logistic Regression')
```



In [87]:
```
roc_curve_plot(fpr, tpr, threshold, 'Logistic Regression')
```



#2 SGD Classifier

In [88]:
```
sgd_cfs = SGDClassifier()
sgd_cfs.fit(X_train, y_train)
sgd_cfs_score = sgd_cfs.score(X_train, y_train)
sgd_cfs_cross_val_accuracy = cross_val_score(sgd_cfs, X_train, y_train, cv=5, scoring='accuracy', n_jobs=-1)
sgd_cfs_cross_val_predict = cross_val_predict(sgd_cfs, X_train, y_train, cv=5, n_jobs=-1)
sgd_cfs_decision_score = cross_val_predict(sgd_cfs, X_train, y_train, cv=5, n_jobs=-1, method='decision_function')
```

In [89]:
```
precision_score(y_train, sgd_cfs_cross_val_predict)
```

Out[89]:
```
0.8461538461538461
```

In [90]:
```
recall_score(y_train, sgd_cfs_cross_val_predict)
```

Out[90]:
```
0.8510028653295129
```

In [91]:
```
print(classification_report(y_train, sgd_cfs_cross_val_predict))
```

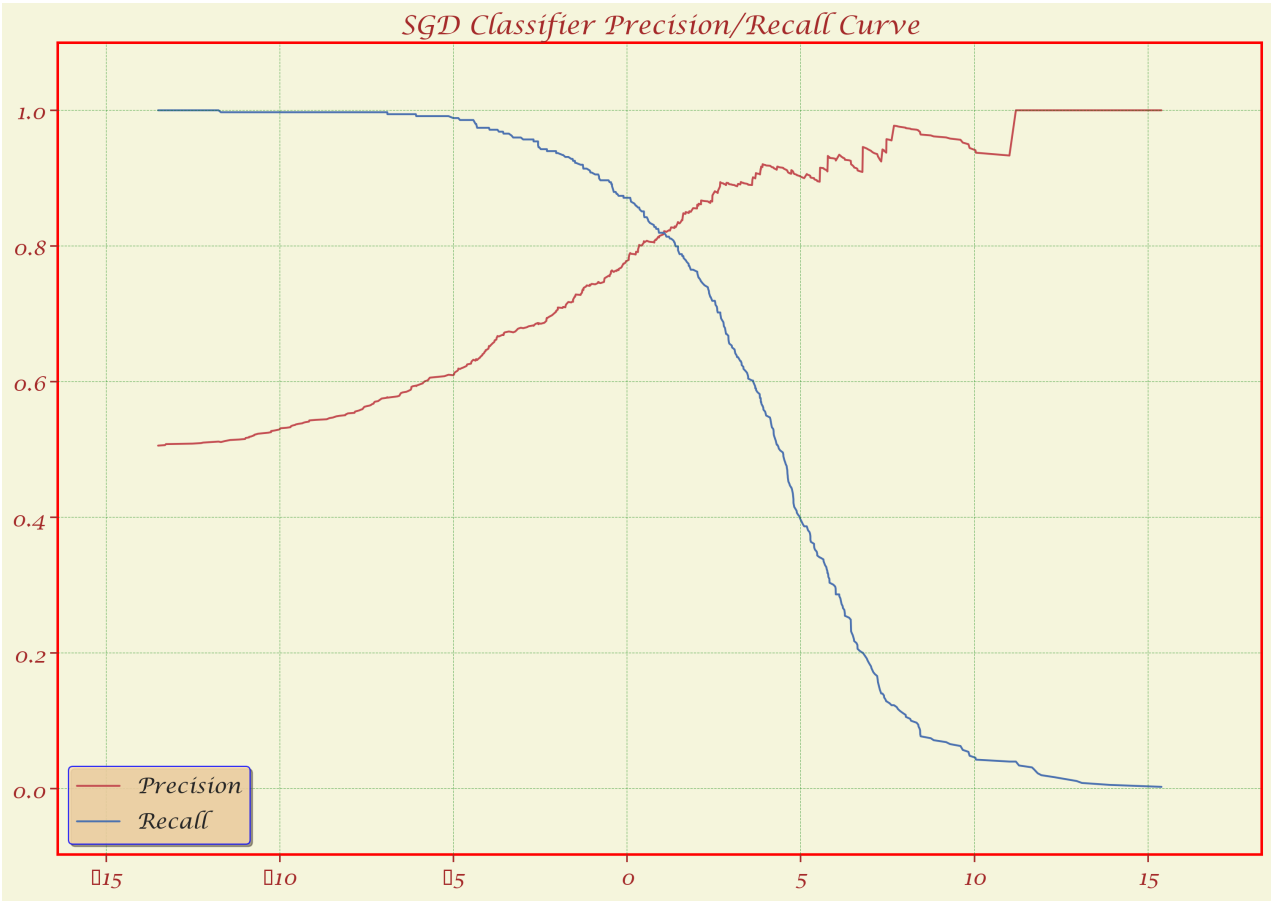
```
              precision    recall  f1-score   support

           0       0.85      0.84      0.84       341
           1       0.85      0.85      0.85       349

    accuracy                           0.85       690
   macro avg       0.85      0.85      0.85       690
weighted avg       0.85      0.85      0.85       690
```

In [92]:
```
precision_, recall_, thresholds_ = precision_recall_curve(y_train, sgd_cfs_decision_score)
fpr, tpr, threshold = roc_curve(y_train, sgd_cfs_decision_score)
```

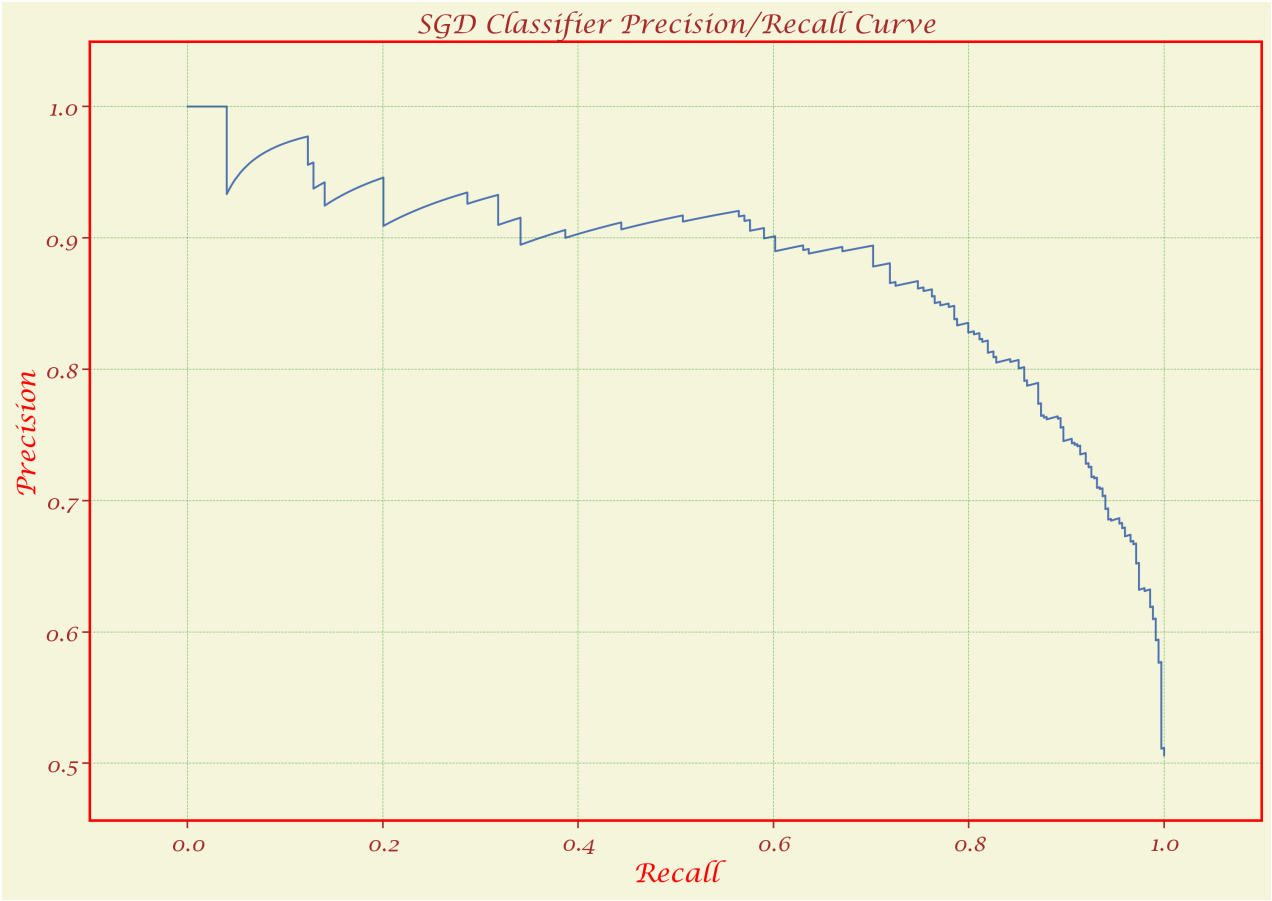
In [93]:
```
confusion_matrix(y_train, sgd_cfs_cross_val_predict)
```

Out[93]:
```
array([[287,  54],
       [ 52, 297]], dtype=int64)
```

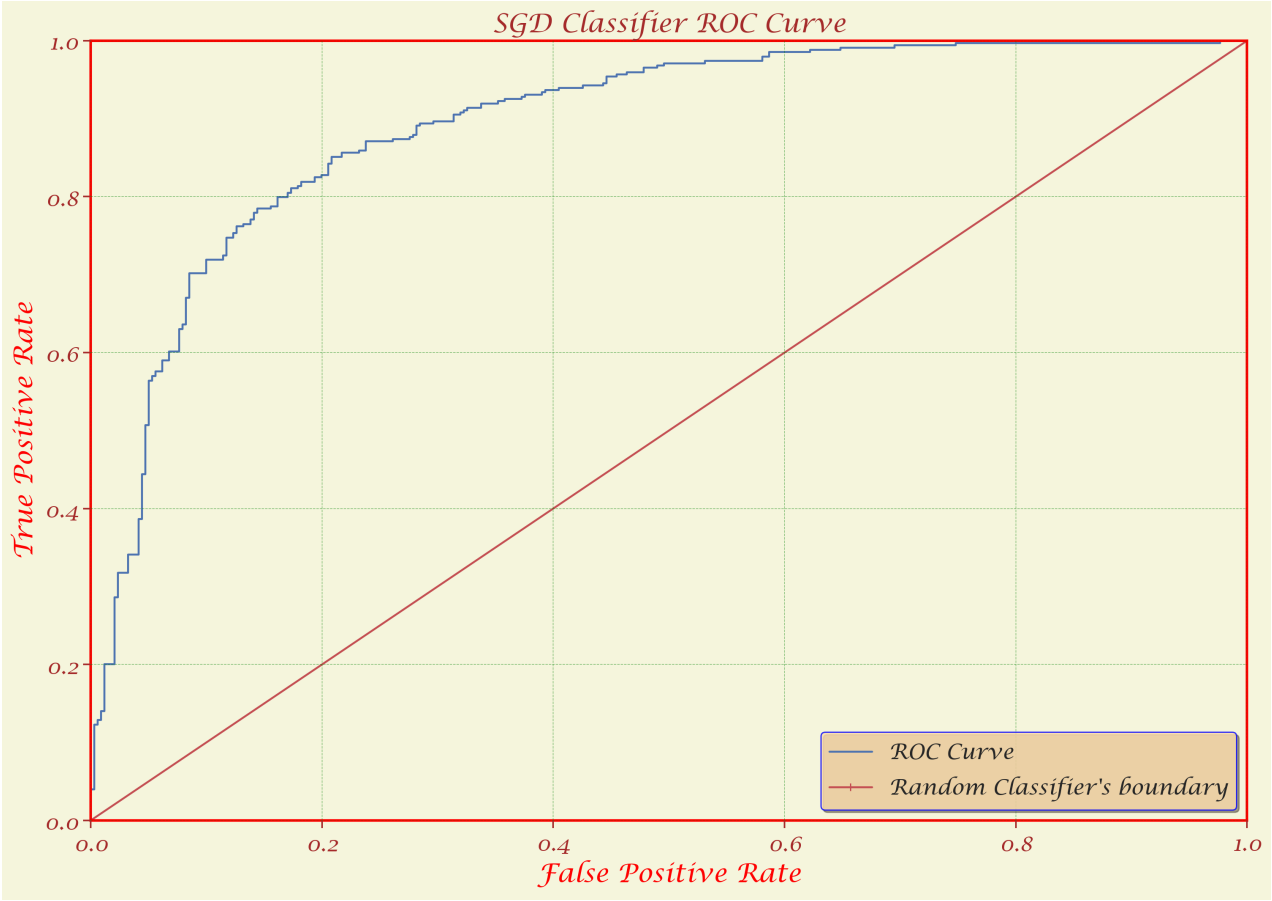
In [94]:
```
p_r_t_curve(precision_, recall_, thresholds_, 'SGD Classifier')
```



In [95]:
```
rc(precision_, recall_, 'SGD Classifier')
```



In [96]:
```
roc_curve_plot(fpr, tpr, threshold, 'SGD Classifier')
```



#3 KNN Classifier

In [97]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [98]:
```
knn_cfs = KNeighborsClassifier()
knn_cfs.fit(X_train, y_train)
knn_cfs_score = knn_cfs.score(X_train, y_train)
knn_cfs_cross_val_accuracy = cross_val_score(knn_cfs, X_train, y_train, cv=5, scoring='accuracy', n_jobs=-1)
knn_cfs_cross_val_predict = cross_val_predict(knn_cfs, X_train, y_train, cv=5, n_jobs=-1)
knn_cfs_decision_score = cross_val_predict(knn_cfs, X_train, y_train, cv=5, n_jobs=-1, method='predict_proba')
```

In [99]:
```
X.head()
```

Out[99]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	chlestrol_by_category	blood_pressure_category
0	-0.04	1.0	-1.0	-0.37	-0.42	-1.0	0.0	0.48	-1.0	-0.64	1.0	0.0	1.0	-1.0	-1.00
1	0.71	1.0	-1.0	0.04	-0.69	-1.0	0.0	-0.18	1.0	-0.07	-1.0	-1.0	1.0	0.0	-0.33
2	0.33	1.0	-1.0	0.10	-0.48	-1.0	0.0	0.37	-1.0	-1.00	1.0	-0.5	1.0	-1.0	-0.33
3	0.21	-1.0	-1.0	-0.88	-0.16	-1.0	-1.0	-0.22	-1.0	-0.64	0.0	-1.0	0.0	1.0	0.33
4	0.21	1.0	-1.0	-0.59	0.35	-1.0	1.0	0.05	-1.0	0.57	-1.0	0.5	-1.0	1.0	0.33

In [100]:
```
knn_cfs.predict(X_train[:1])
```

Out[100]:
```
array([1], dtype=int64)
```

In [101]:
```
knn_cfs.predict_proba(X_train[:1])
```

Out[101]:
```
array([[0., 1.]])
```

In [102]:
```
knn_cfs_decision_score
```

Out[102]:
```
array([[0. , 1. ],
       [0.8, 0.2],
       [0.2, 0.8],
       ...,
       [1. , 0. ],
       [0.6, 0.4],
       [0. , 1. ]])
```

In [103]:
```
precision_, recall_, thresholds_ = precision_recall_curve(y_train, knn_cfs_decision_score[:, 1])
fpr, tpr, threshold = roc_curve(y_train, knn_cfs_decision_score[:, 1])
```

In [104]:
```
confusion_matrix(y_train, knn_cfs_cross_val_predict)
```

Out[104]:
```
array([[282,  59],
       [ 69, 280]], dtype=int64)
```

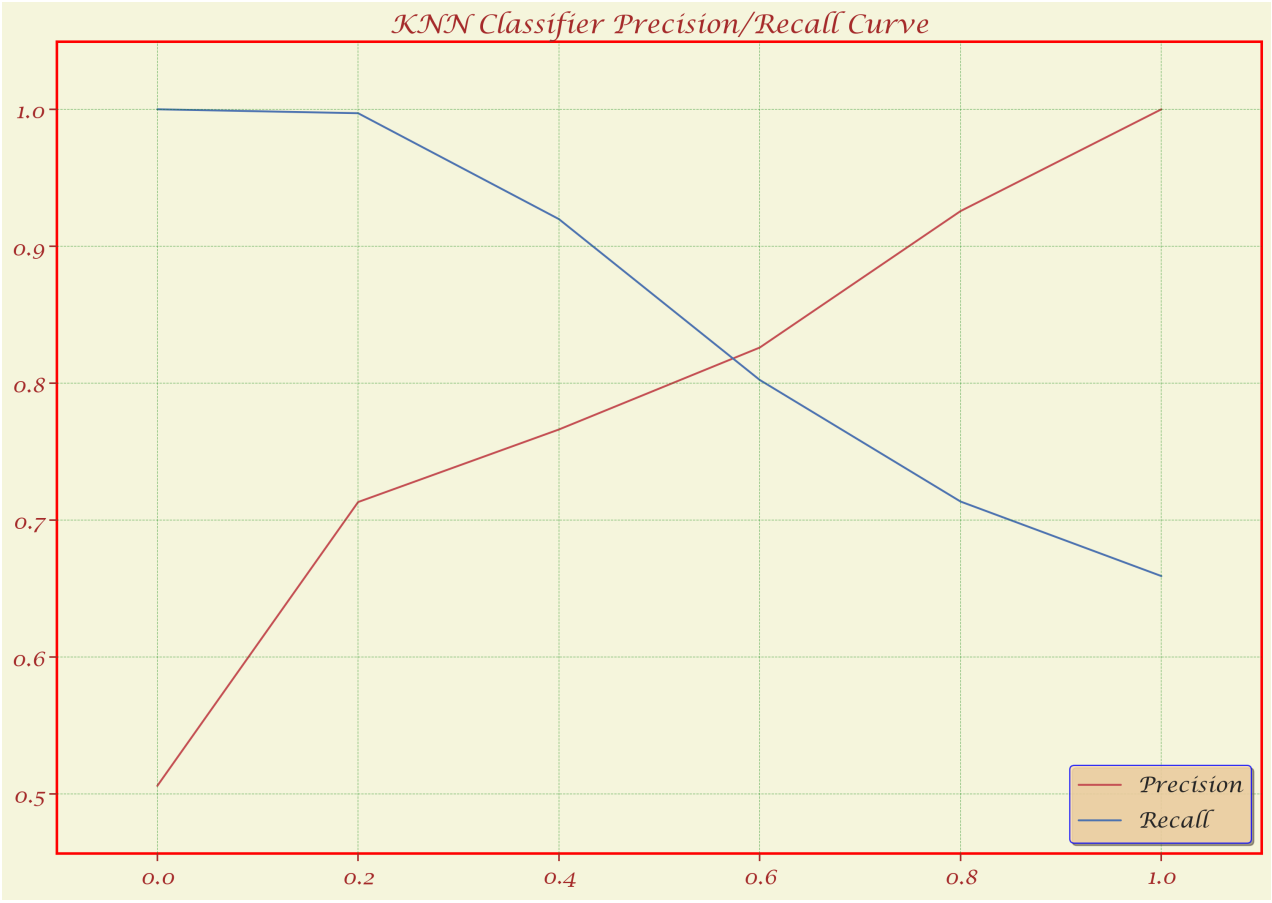
In [105]:
```
precision_score(y_train, knn_cfs_cross_val_predict)
```

Out[105]:
```
0.8259587020648967
```

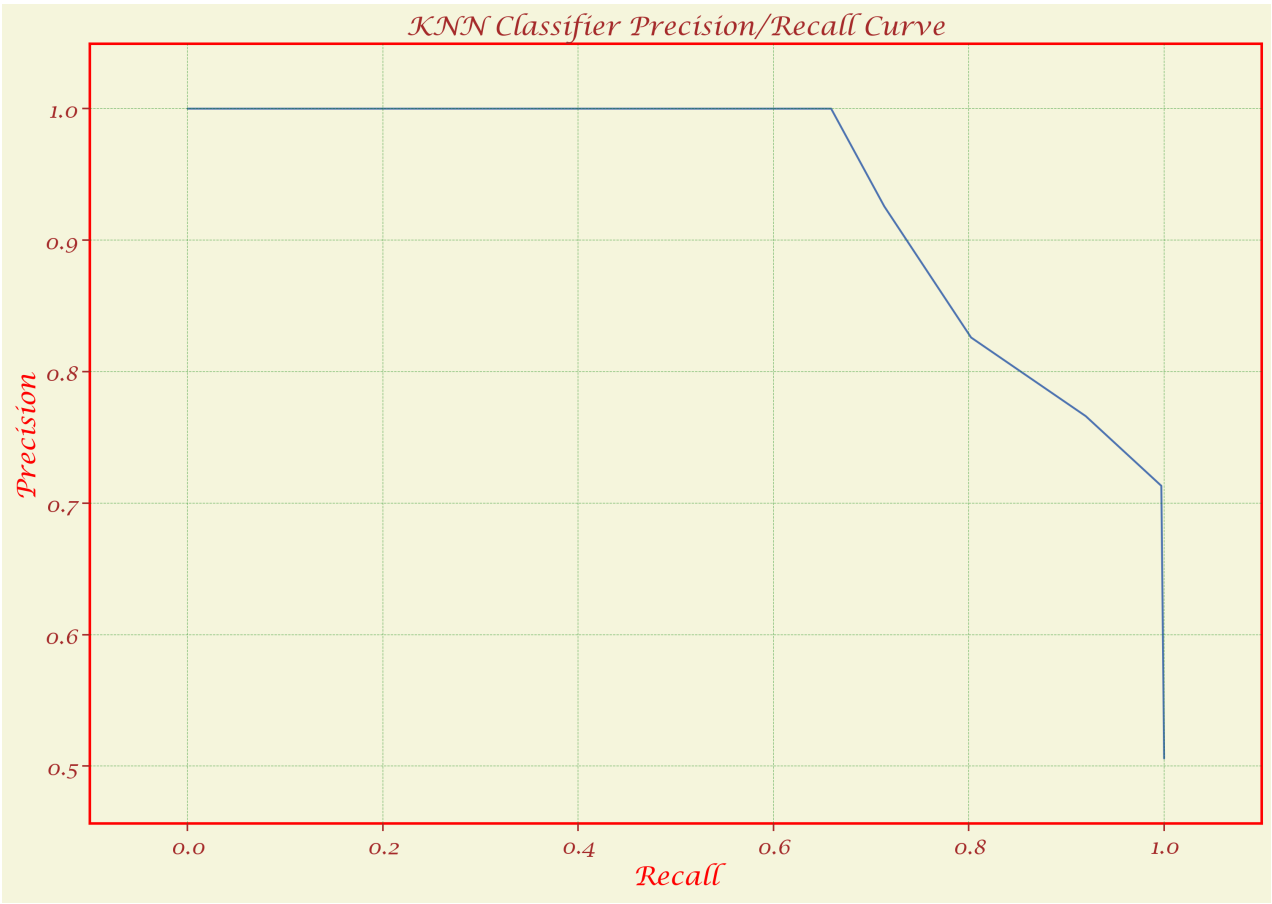
In [106]:
```
recall_score(y_train, knn_cfs_cross_val_predict)
```

Out[106]:
```
0.8022922636103151
```

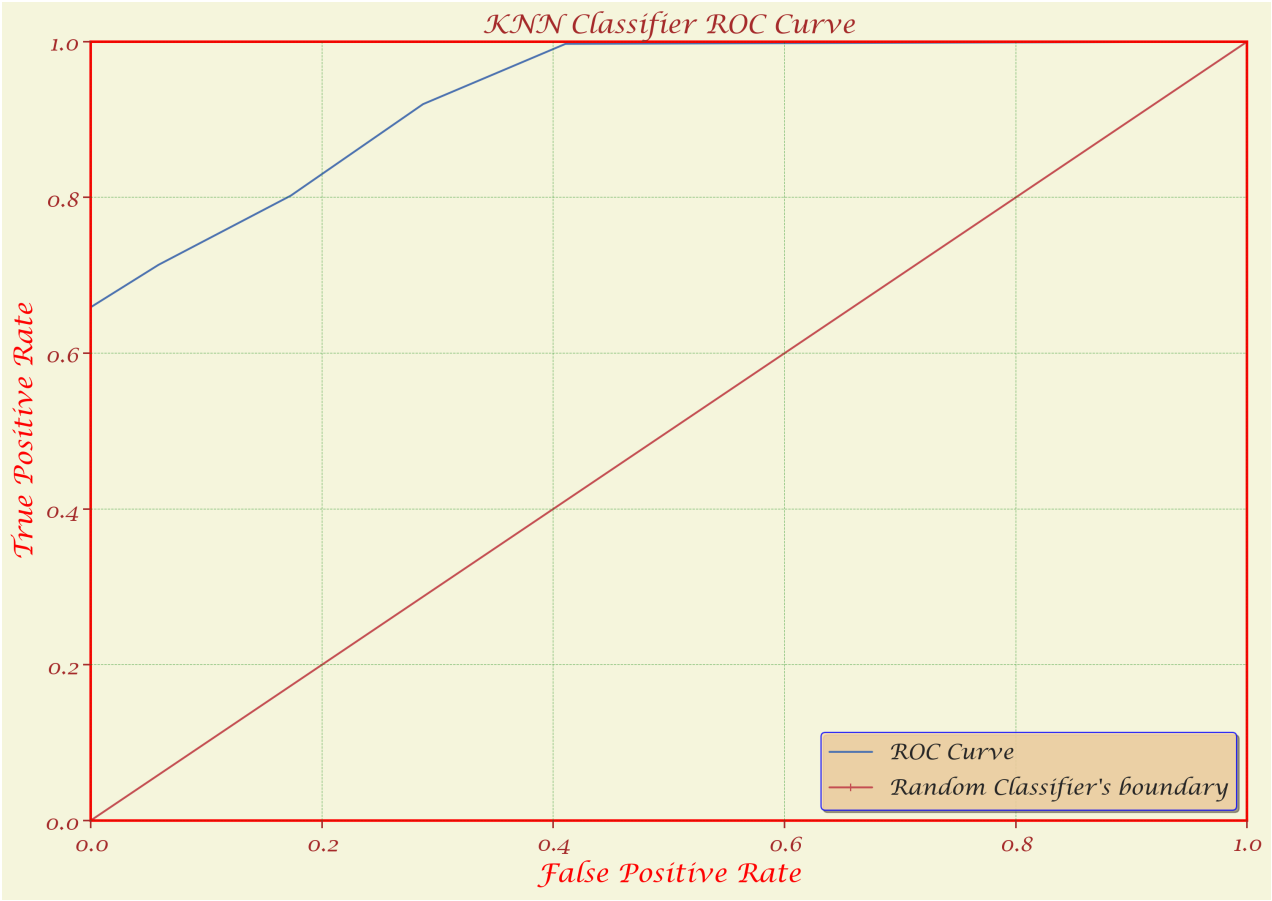
In [107]:
```
p_r_t_curve(precision_, recall_, thresholds_, 'KNN Classifier')
```



In [108]:
```
rc(precision_, recall_, 'KNN Classifier')
```



In [109]:
```
roc_curve_plot(fpr, tpr, threshold, 'KNN Classifier')
```



#4 Linear SVC

In [110]:
```
from sklearn.svm import LinearSVC, SVC
lin_svc_cfs = LinearSVC()
lin_svc_cfs.fit(X_train, y_train)
lin_svc_cfs_score = lin_svc_cfs.score(X_train, y_train)
lin_svc_cfs_cross_val_accuracy = cross_val_score(lin_svc_cfs, X_train, y_train, cv=5, scoring='accuracy', n_jobs=-1)
lin_svc_cfs_cross_val_predict = cross_val_predict(lin_svc_cfs, X_train, y_train, cv=5, n_jobs=-1)
lin_svc_cfs_decision_score = cross_val_predict(lin_svc_cfs, X_train, y_train, cv=5, n_jobs=-1, method='decision_function')
```

In [111]:
```
precision_, recall_, thresholds_ = precision_recall_curve(y_train, lin_svc_cfs_decision_score)
fpr, tpr, threshold = roc_curve(y_train, lin_svc_cfs_decision_score)
```

In [112]:
```
precision_score(y_train, lin_svc_cfs_cross_val_predict)
```

Out[112]:
```
0.820855614973262
```

In [113]:
```
recall_score(y_train, lin_svc_cfs_cross_val_predict)
```

Out[113]:
```
0.8796561604584527
```

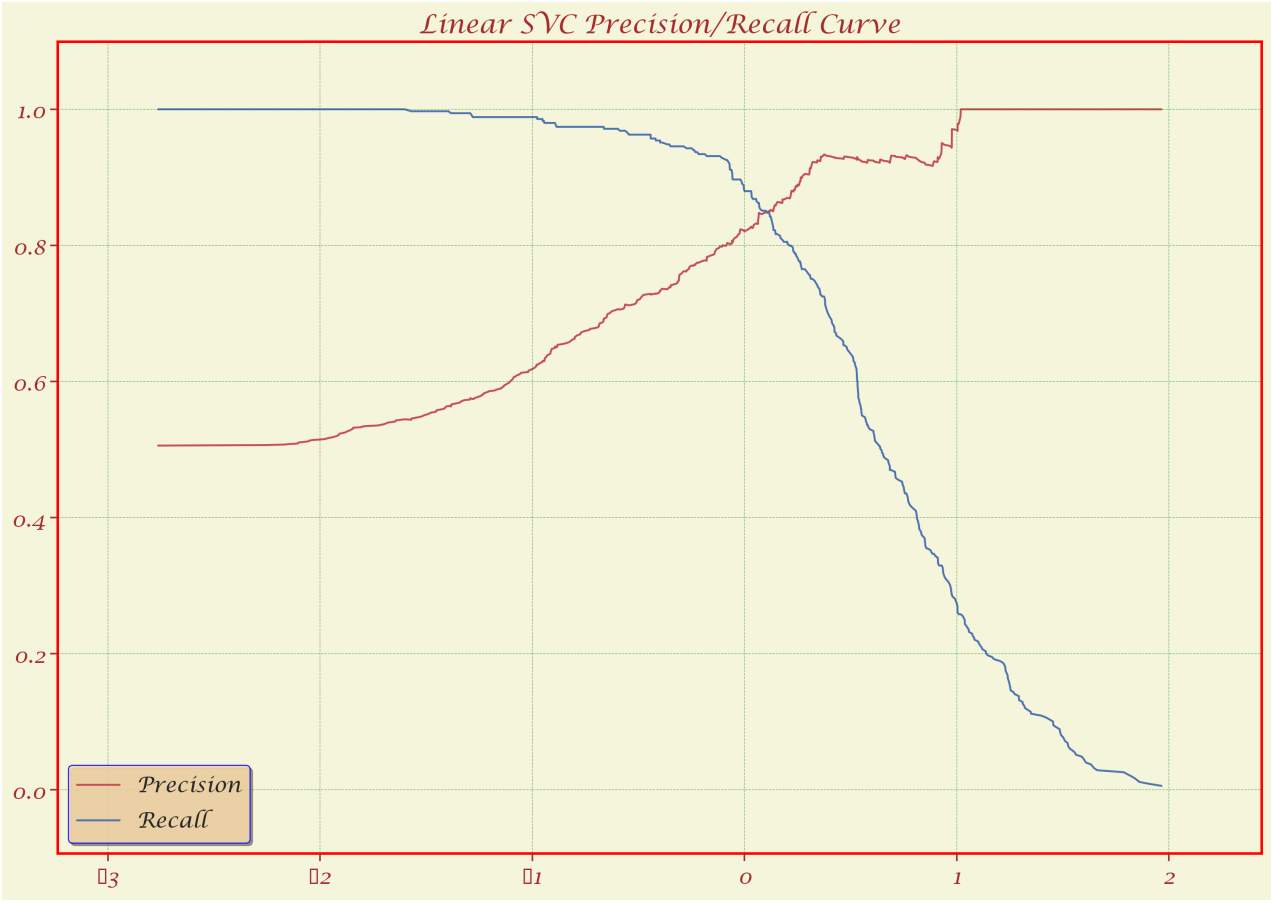
In [114]:
```
confusion_matrix(y_train, lin_svc_cfs_cross_val_predict)
```

Out[114]:
```
array([[274,  67],
       [ 42, 307]], dtype=int64)
```

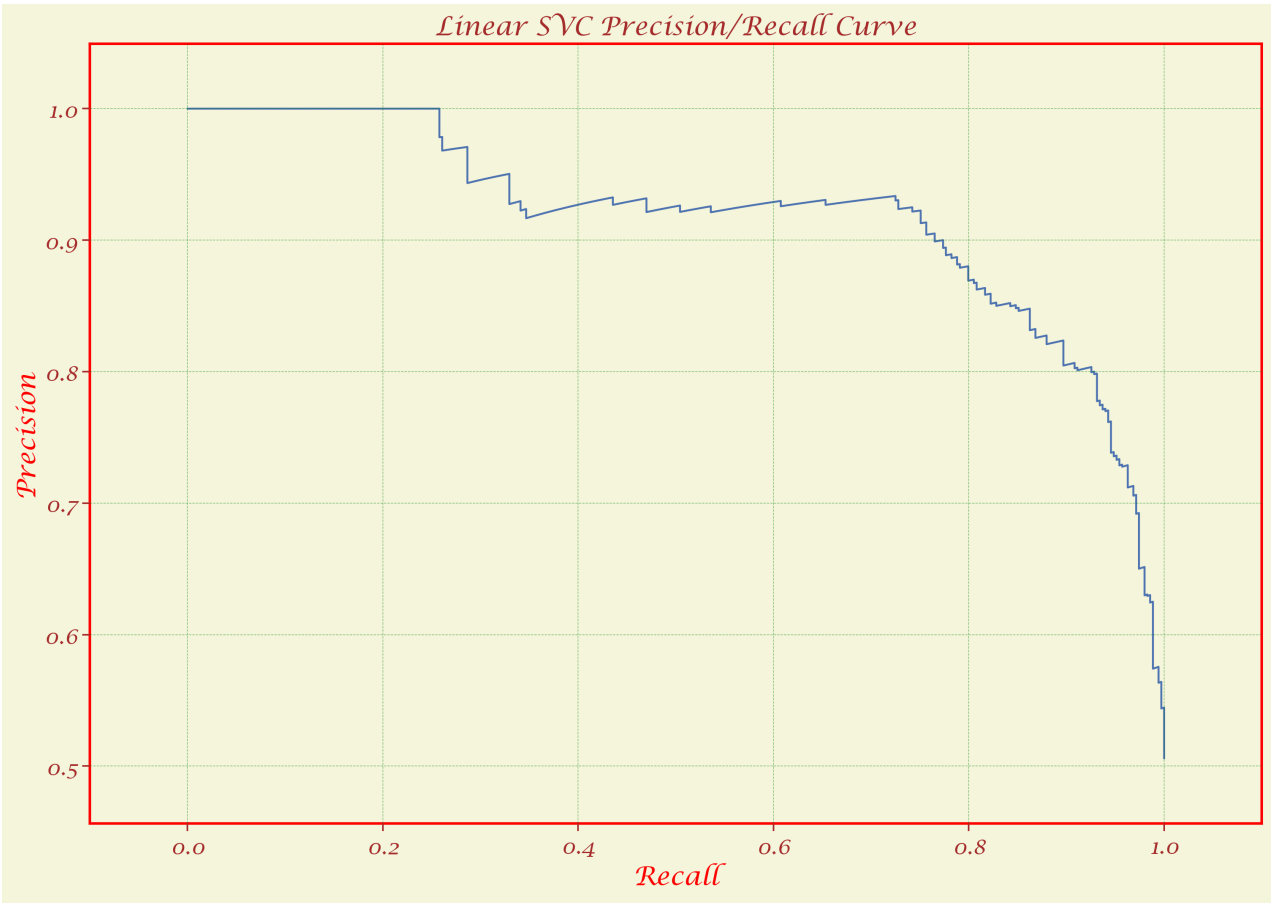
In [115]:
```
print(classification_report(y_train, lin_svc_cfs_cross_val_predict))
```

```
              precision    recall  f1-score   support

           0       0.87      0.80      0.83       341
           1       0.82      0.88      0.85       349

    accuracy                           0.84       690
   macro avg       0.84      0.84      0.84       690
weighted avg       0.84      0.84      0.84       690
```

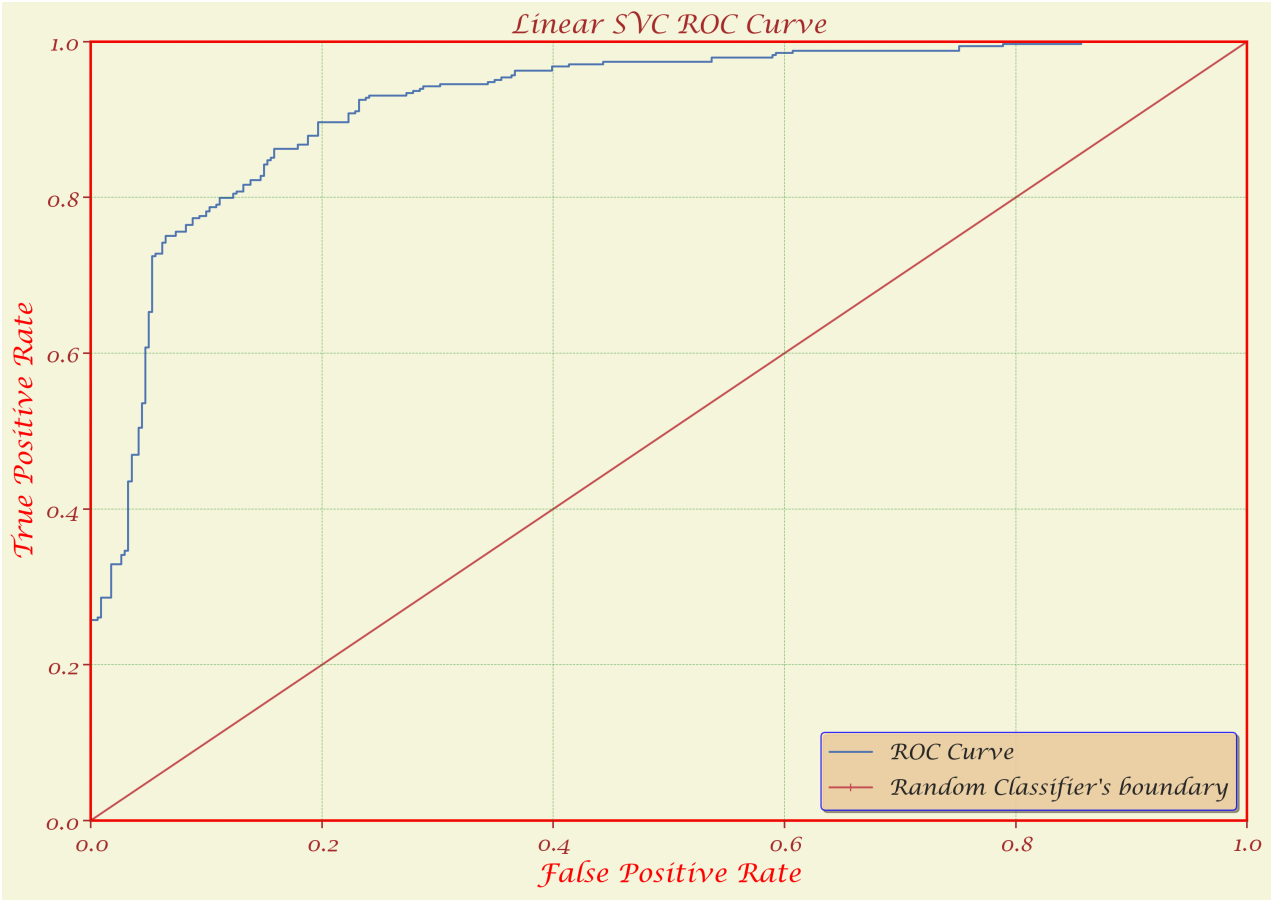
In [116]:
```
p_r_t_curve(precision_, recall_, thresholds_, 'Linear SVC')
```



In [117]:
```
rc(precision_, recall_, 'Linear SVC')
```



In [118]:
```
roc_curve_plot(fpr, tpr, threshold, 'Linear SVC')
```



#5 SVC

In [119]:
```
svc_cfs = SVC(probability=True)
svc_cfs.fit(X_train, y_train)
svc_cfs_score = svc_cfs.score(X_train, y_train)
svc_cfs_cross_val_accuracy = cross_val_score(svc_cfs, X_train, y_train, cv=5, scoring='accuracy', n_jobs=-1)
svc_cfs_cross_val_predict = cross_val_predict(svc_cfs, X_train, y_train, cv=5, n_jobs=-1)
svc_cfs_decision_score = cross_val_predict(svc_cfs, X_train, y_train, cv=5, n_jobs=-1, method='predict_proba')
```

In [120]:
```
precision_, recall_, thresholds_ = precision_recall_curve(y_train, svc_cfs_decision_score[:, 1])
fpr, tpr, threshold = roc_curve(y_train, svc_cfs_decision_score[:, 1])
```

In [121]:
```
precision_score(y_train, svc_cfs_cross_val_predict)
```

Out[121]:
```
0.8888888888888888
```

In [122]:
```
recall_score(y_train, svc_cfs_cross_val_predict)
```

Out[122]:
```
0.9169054441260746
```

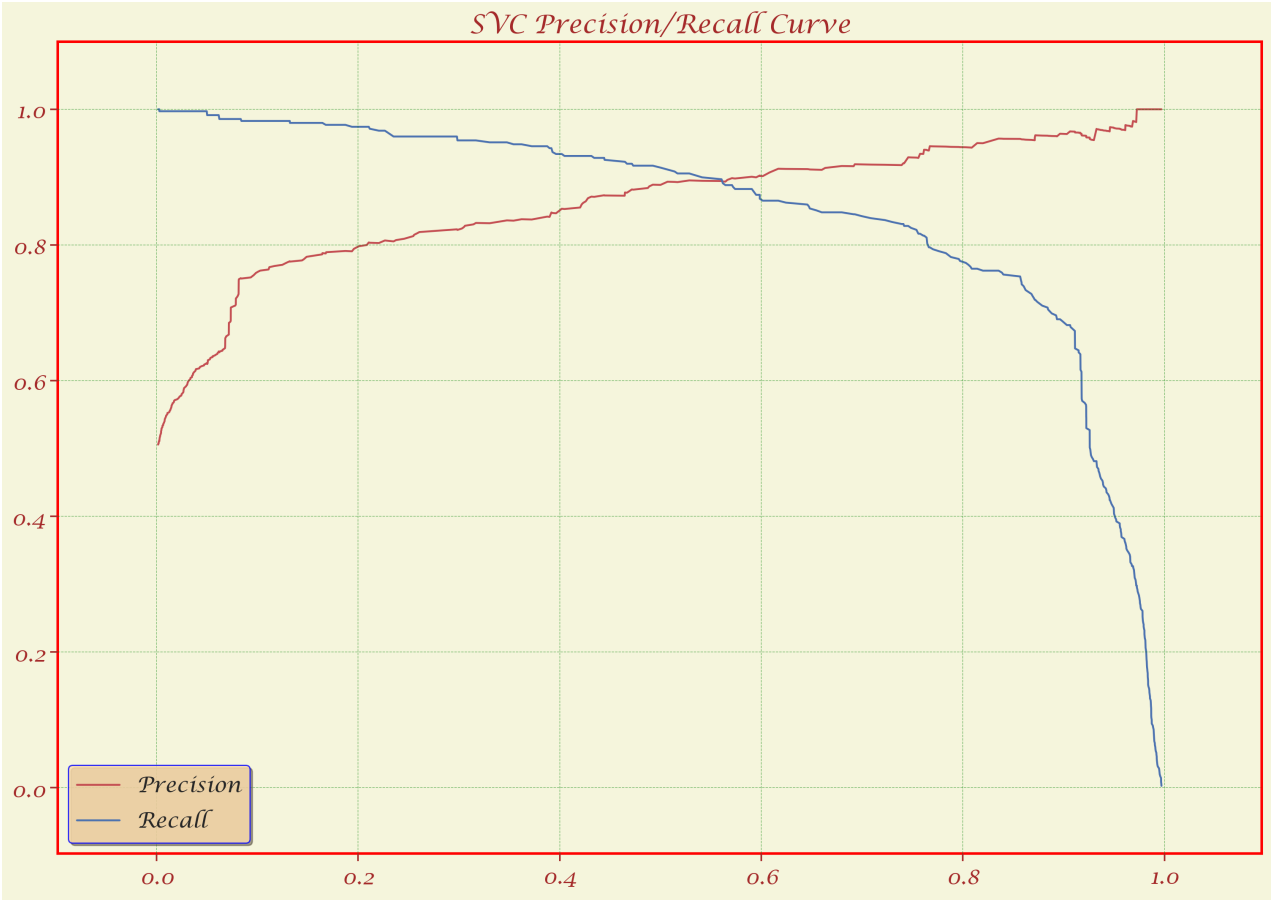
In [123]:
```
print(classification_report(y_train, svc_cfs_cross_val_predict))
```

```
              precision    recall  f1-score   support

           0       0.91      0.88      0.90       341
           1       0.89      0.92      0.90       349

    accuracy                           0.90       690
   macro avg       0.90      0.90      0.90       690
weighted avg       0.90      0.90      0.90       690
```

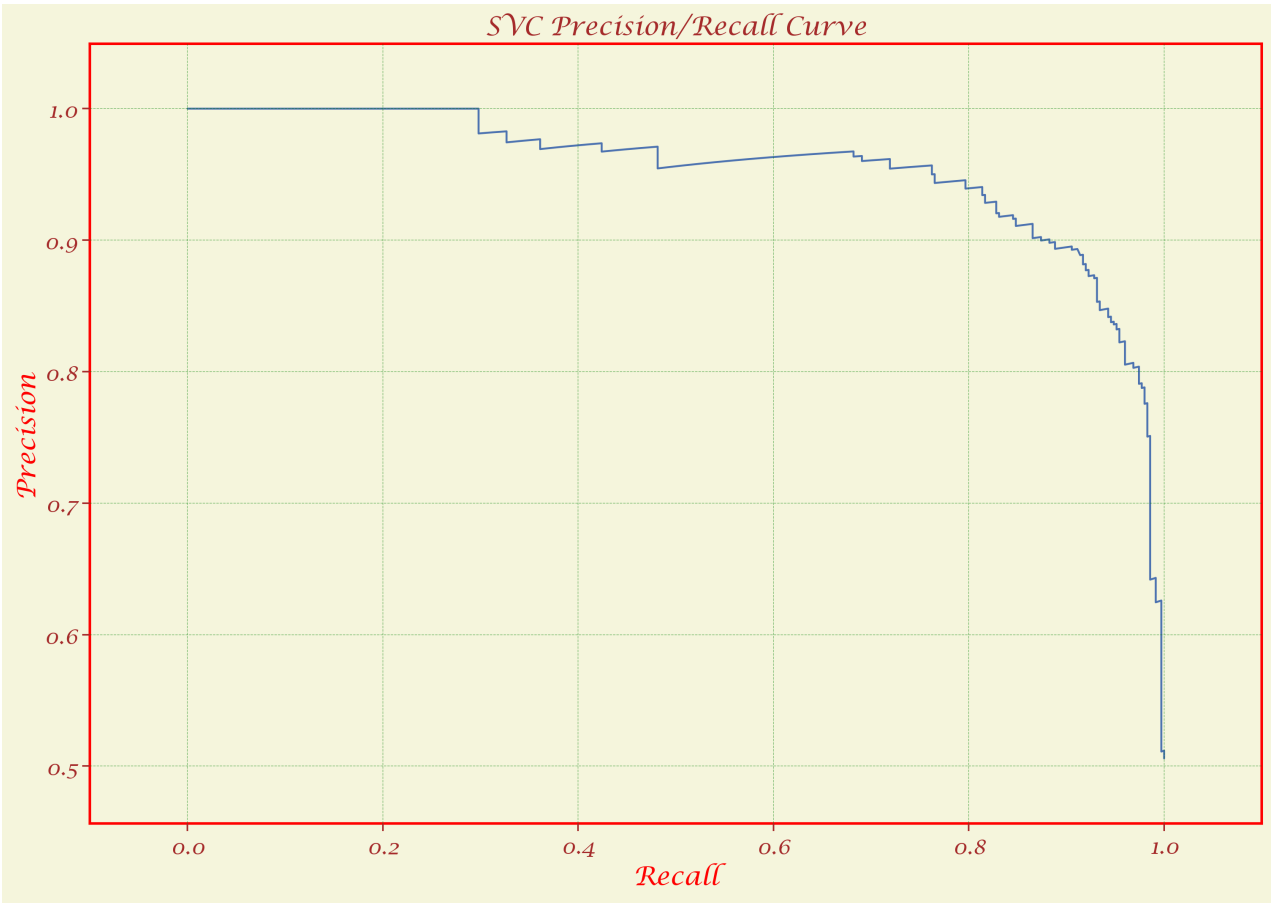
In [124]:
```
confusion_matrix(y_train, svc_cfs_cross_val_predict)
```

Out[124]:
```
array([[301,  40],
       [ 29, 320]], dtype=int64)
```

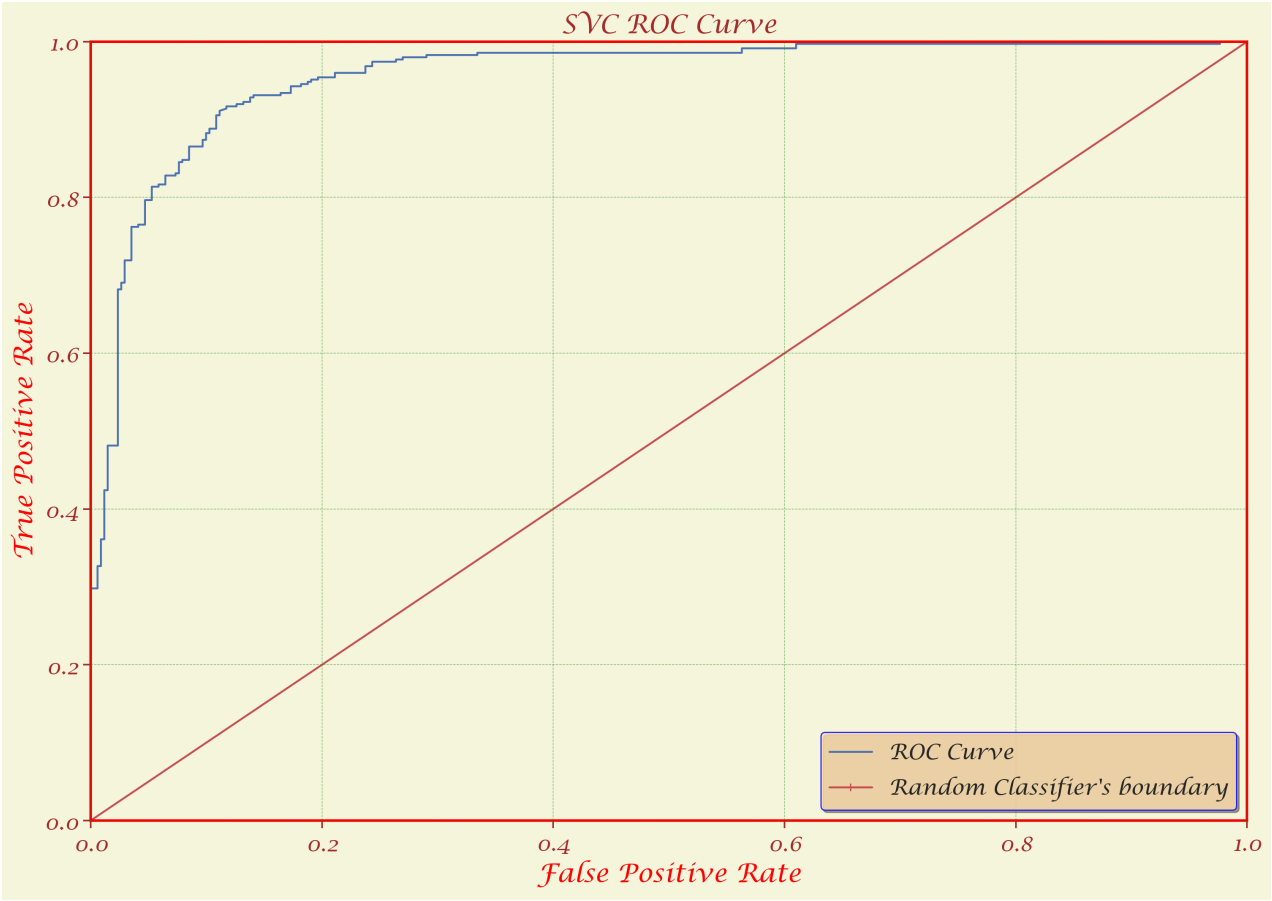
In [125]:
```
p_r_t_curve(precision_, recall_, thresholds_, 'SVC')
```



In [126]:
```
rc(precision_, recall_, 'SVC')
```



In [127]:
```
roc_curve_plot(fpr, tpr, threshold, 'SVC')
```



#6 Decision Tree Classifier

In [128]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [129]:
```
dec_cfs = DecisionTreeClassifier()
dec_cfs.fit(X_train, y_train)
dec_cfs_score = dec_cfs.score(X_train, y_train)
dec_cfs_cross_val_accuracy = cross_val_score(dec_cfs, X_train, y_train, cv=5, scoring='accuracy', n_jobs=-1)
dec_cfs_cross_val_predict = cross_val_predict(dec_cfs, X_train, y_train, cv=5, n_jobs=-1)
dec_cfs_decision_score = cross_val_predict(dec_cfs, X_train, y_train, cv=5, n_jobs=-1, method='predict_proba')
```

In [130]:
```
dec_cfs_score
```

Out[130]:
```
1.0
```

In [131]:
```
confusion_matrix(y_train, dec_cfs_cross_val_predict)
```

Out[131]:
```
array([[332,   9],
       [ 11, 338]], dtype=int64)
```

In [132]:
```
dec_cfs_cross_val_accuracy
```

Out[132]:
```
array([0.94927536, 0.96376812, 0.95652174, 0.98550725, 0.99275362])
```

In [133]:
```
precision_score(y_train, dec_cfs_cross_val_predict)
```

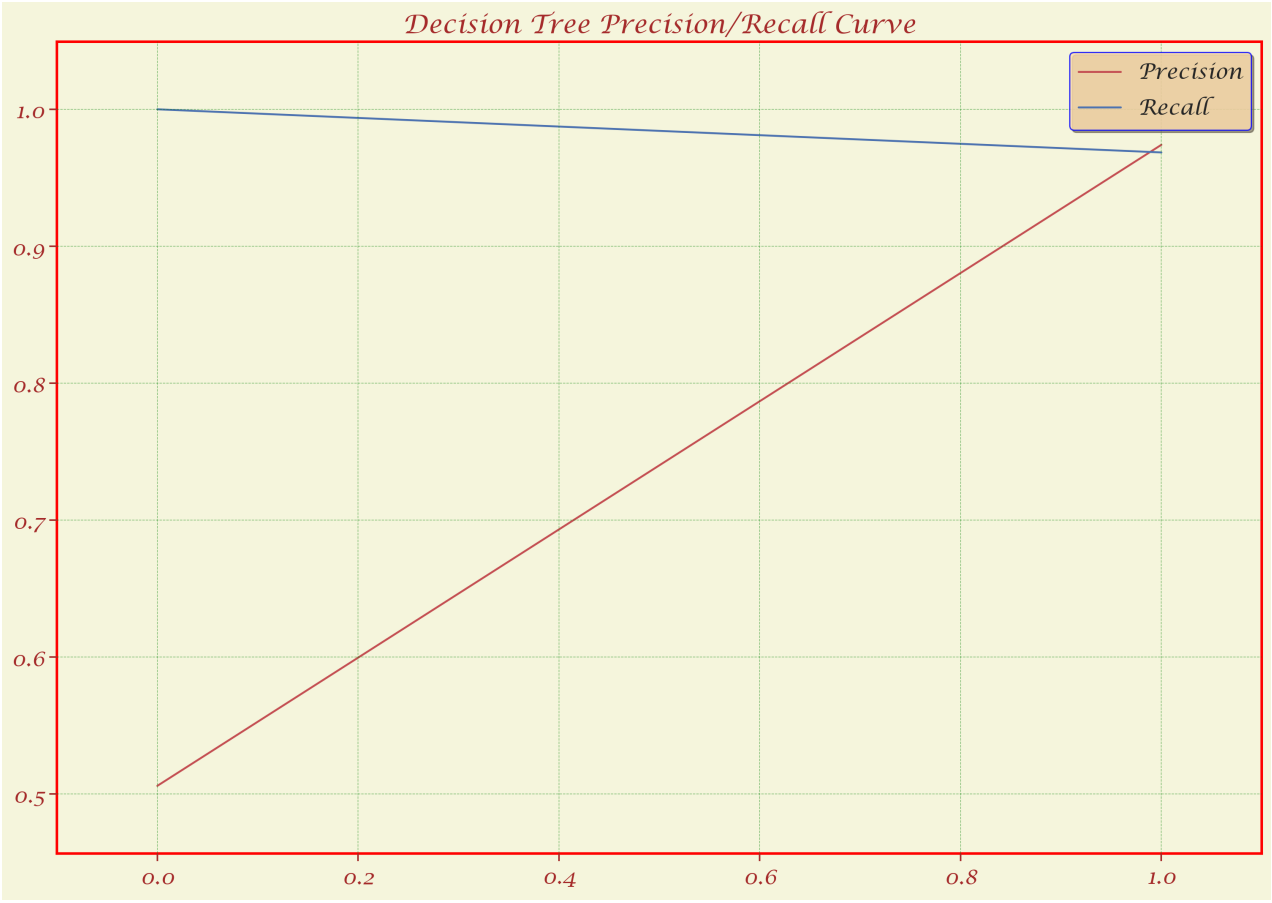
Out[133]:
```
0.9740634005763689
```

In [134]:
```
recall_score(y_train, dec_cfs_cross_val_predict)
```

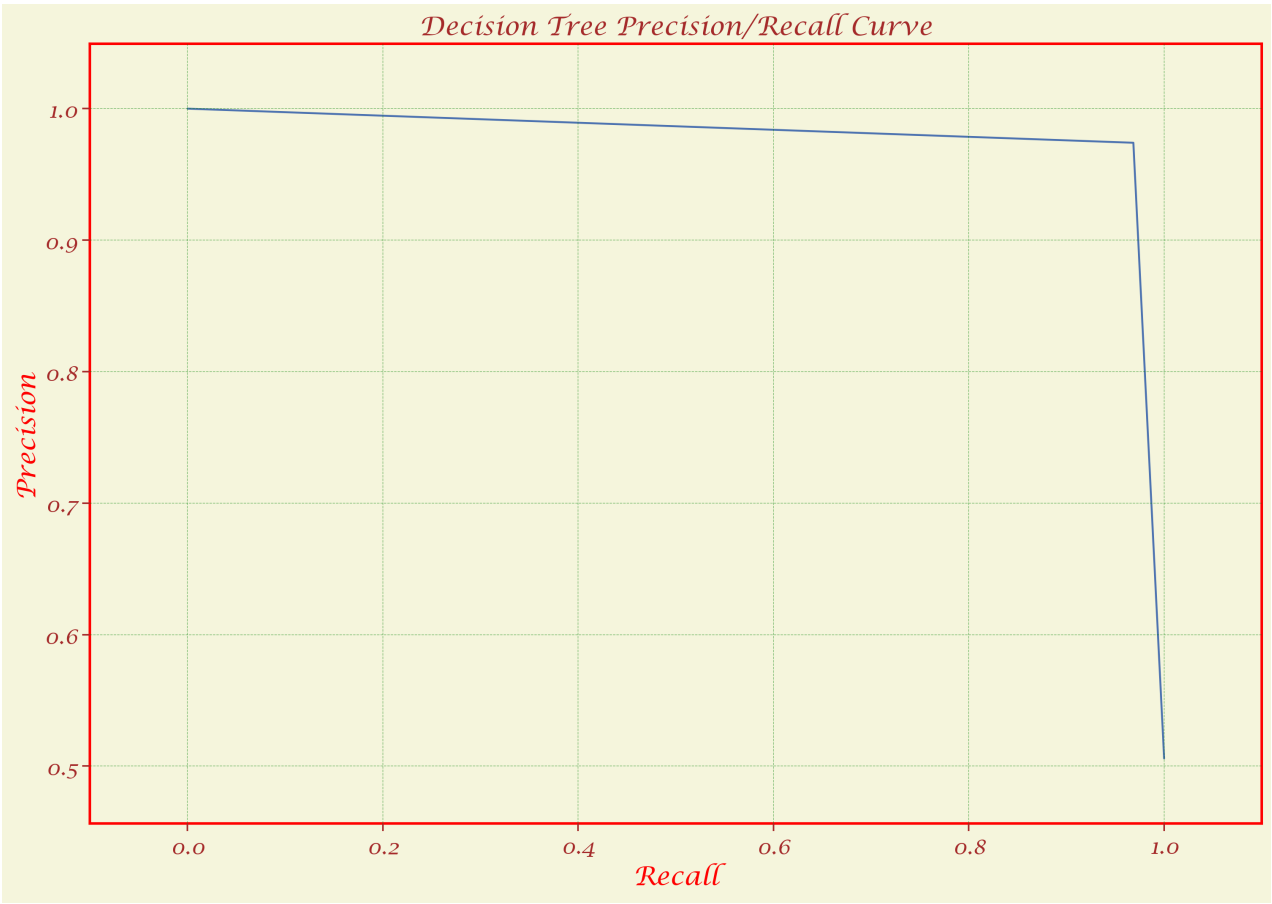
Out[134]:
```
0.9684813753581661
```

In [135]:
```
precision_, recall_, thresholds_ = precision_recall_curve(y_train, dec_cfs_decision_score[:, 1])
fpr, tpr, threshold = roc_curve(y_train, dec_cfs_decision_score[:, 1])
```

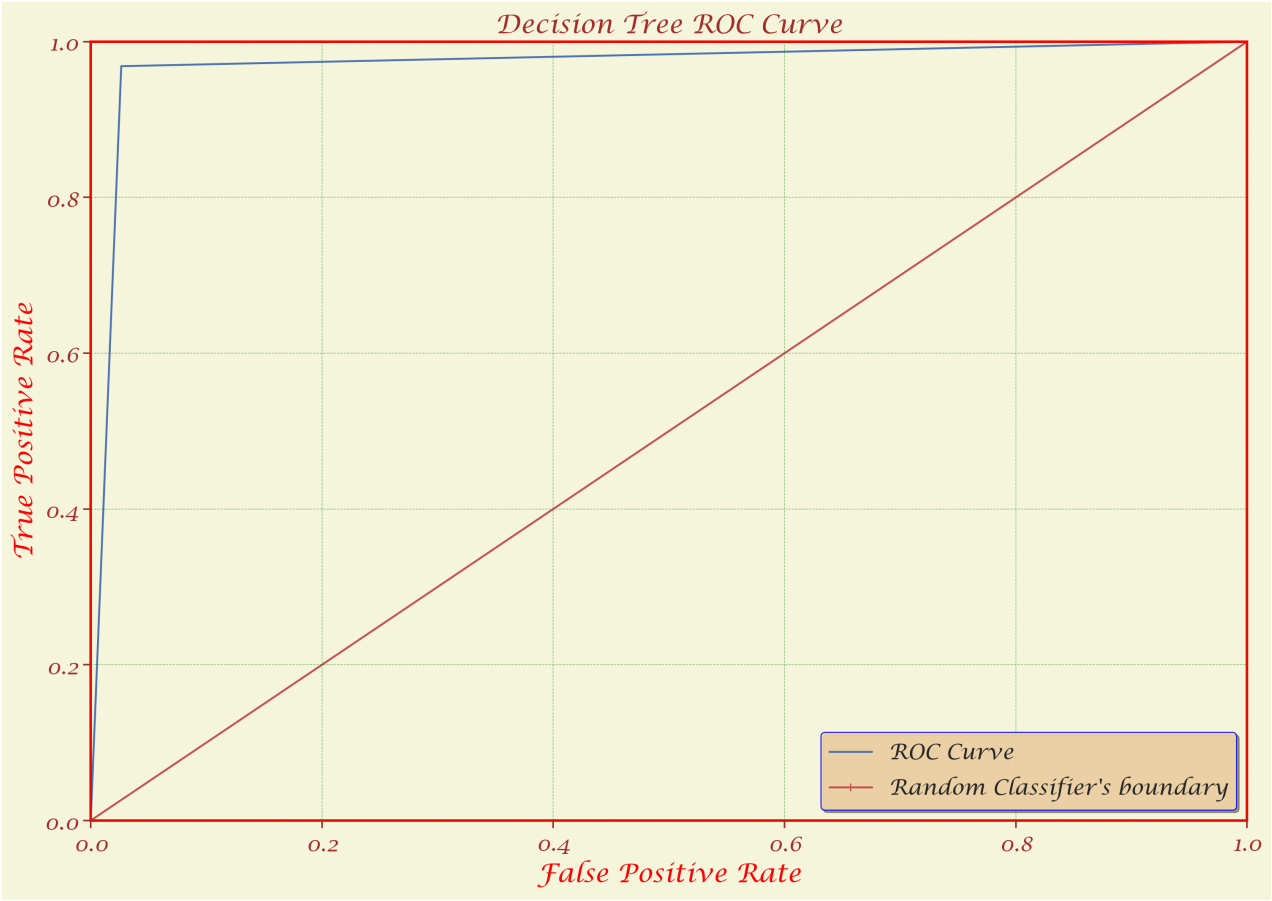
In [136]:
```
p_r_t_curve(precision_, recall_, thresholds_, 'Decision Tree')
```



In [137]:
```
rc(precision_, recall_, 'Decision Tree')
```



In [138]:
```
roc_curve_plot(fpr, tpr, threshold, 'Decision Tree')
```



#7 Random Forest Classifier

In [139]:
```
from sklearn.ensemble import RandomForestClassifier
rnd_cfs = RandomForestClassifier()
rnd_cfs.fit(X_train, y_train)
rnd_cfs_score = rnd_cfs.score(X_train, y_train)
rnd_cfs_cross_val_accuracy = cross_val_score(rnd_cfs, X_train, y_train, cv=5, scoring='accuracy', n_jobs=-1)
rnd_cfs_cross_val_predict = cross_val_predict(rnd_cfs, X_train, y_train, cv=5, n_jobs=-1)
rnd_cfs_decision_score = cross_val_predict(rnd_cfs, X_train, y_train, cv=5, n_jobs=-1, method='predict_proba')
```

In [140]:
```
rnd_cfs_score
```

Out[140]:
```
1.0
```

In [141]:
```
rnd_cfs_cross_val_accuracy
```

Out[141]:
```
array([0.97101449, 0.96376812, 0.98550725, 0.98550725, 1.        ])
```

In [142]:
```
precision_score(y_train, rnd_cfs_cross_val_predict)
```

Out[142]:
```
0.9829545454545454
```

In [143]:
```
recall_score(y_train, rnd_cfs_cross_val_predict)
```

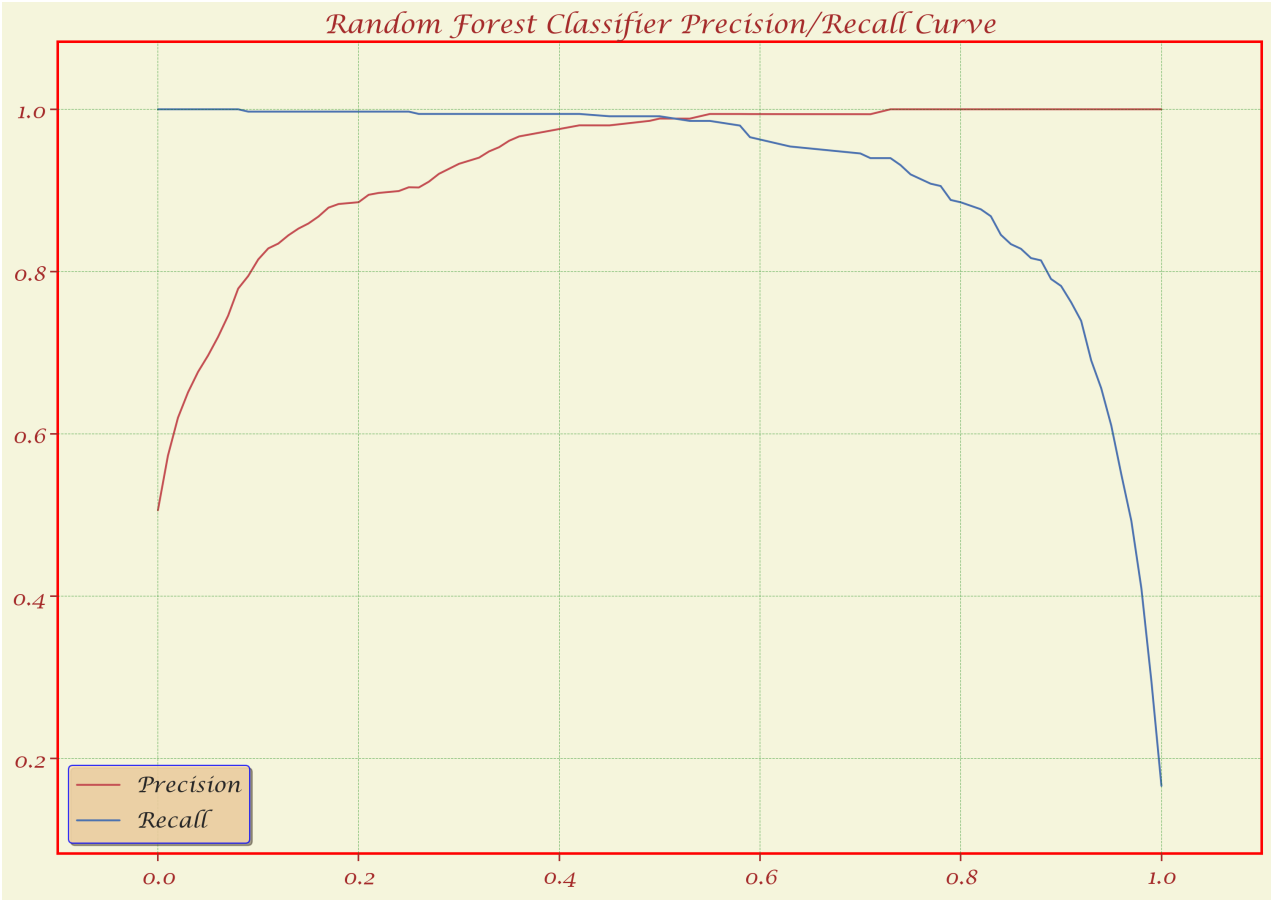
Out[143]:
```
0.9914040114613181
```

In [144]:
```
confusion_matrix(y_train, rnd_cfs_cross_val_predict)
```

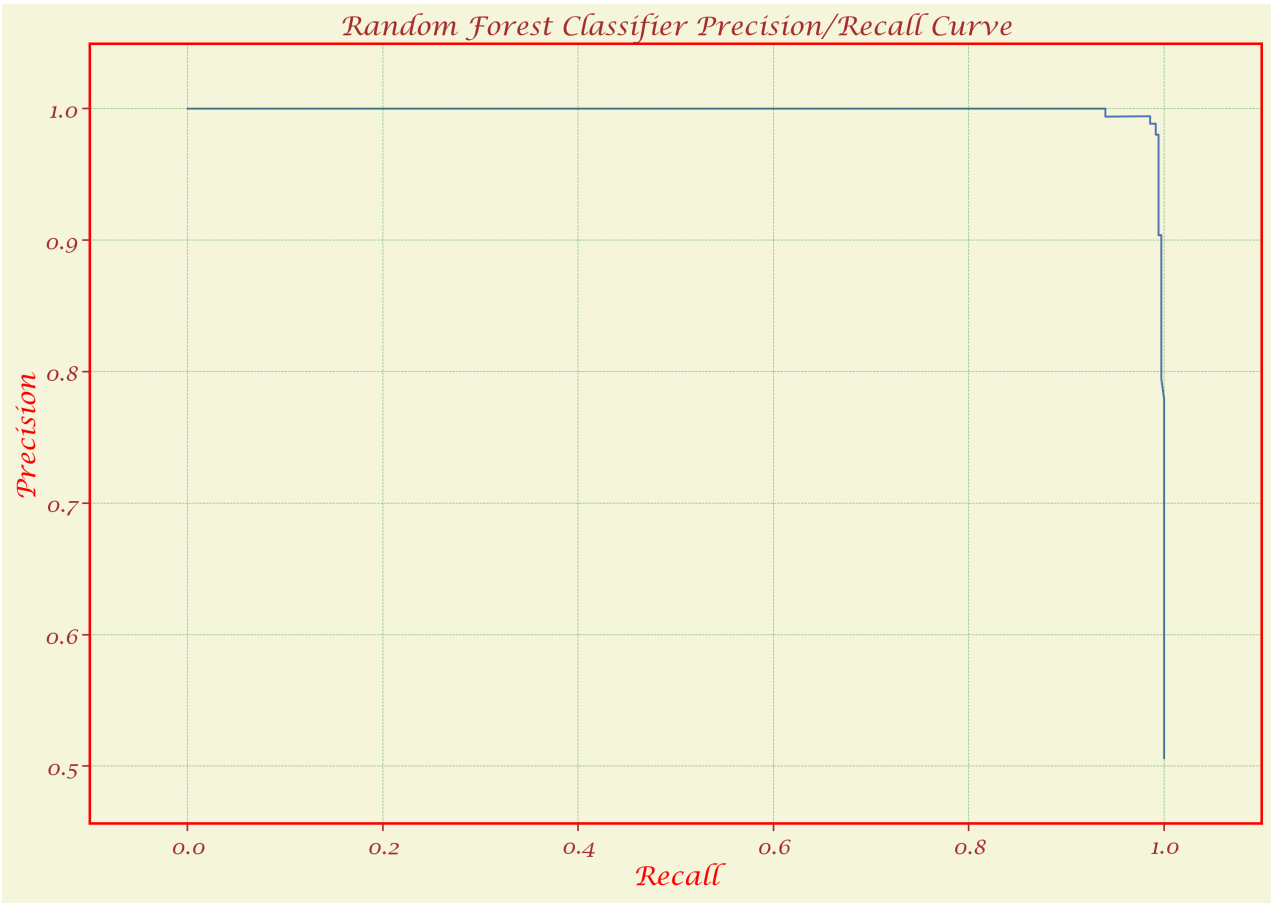
Out[144]:
```
array([[335,   6],
       [  3, 346]], dtype=int64)
```

In [145]:
```
precision_, recall_, thresholds_ = precision_recall_curve(y_train, rnd_cfs_decision_score[:, 1])
fpr, tpr, threshold = roc_curve(y_train, rnd_cfs_decision_score[:, 1])
```

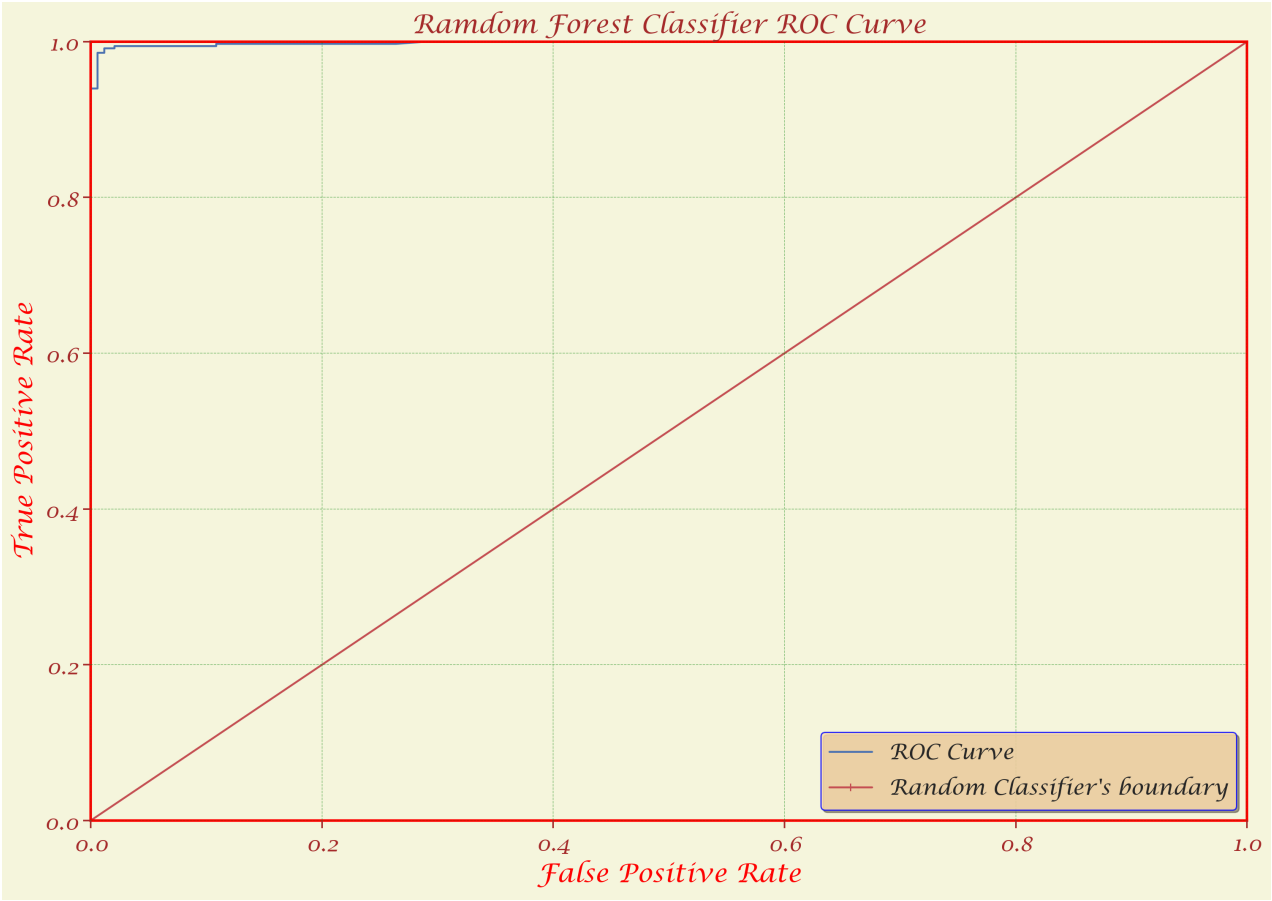
In [146]:
```
p_r_t_curve(precision_, recall_, thresholds_, 'Random Forest Classifier')
```



In [147]:
```
rc(precision_, recall_, 'Random Forest Classifier')
```



In [148]:
```
roc_curve_plot(fpr, tpr, threshold, 'Ramdom Forest Classifier')
```



#8 Extra Tree Classifier

In [149]:
```
from sklearn.ensemble import ExtraTreesClassifier
extra_cfs = ExtraTreesClassifier()
extra_cfs.fit(X_train, y_train)
extra_cfs_score = extra_cfs.score(X_train, y_train)
extra_cfs_cross_val_accuracy = cross_val_score(extra_cfs, X_train, y_train, cv=5, scoring='accuracy', n_jobs=-1)
extra_cfs_cross_val_predict = cross_val_predict(extra_cfs, X_train, y_train, cv=5, n_jobs=-1)
extra_cfs_decision_score = cross_val_predict(extra_cfs, X_train, y_train, cv=5, n_jobs=-1, method='predict_proba')
```

In [150]:
```
extra_cfs_score
```

Out[150]:
```
1.0
```

In [151]:
```
extra_cfs_cross_val_accuracy
```

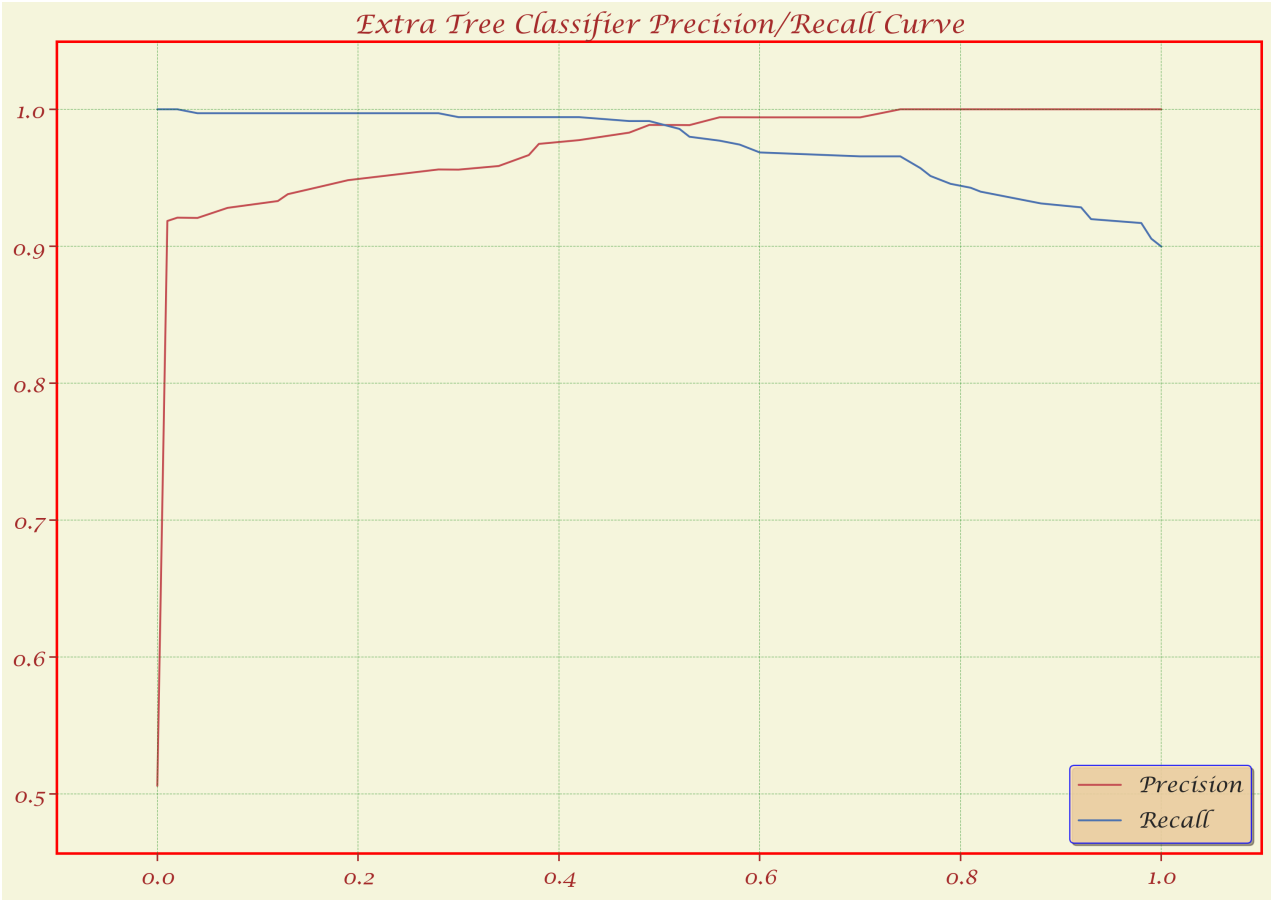
Out[151]:
```
array([1.        , 0.93478261, 1.        , 0.98550725, 1.        ])
```

In [152]:
```
confusion_matrix(y_train, extra_cfs_cross_val_predict)
```

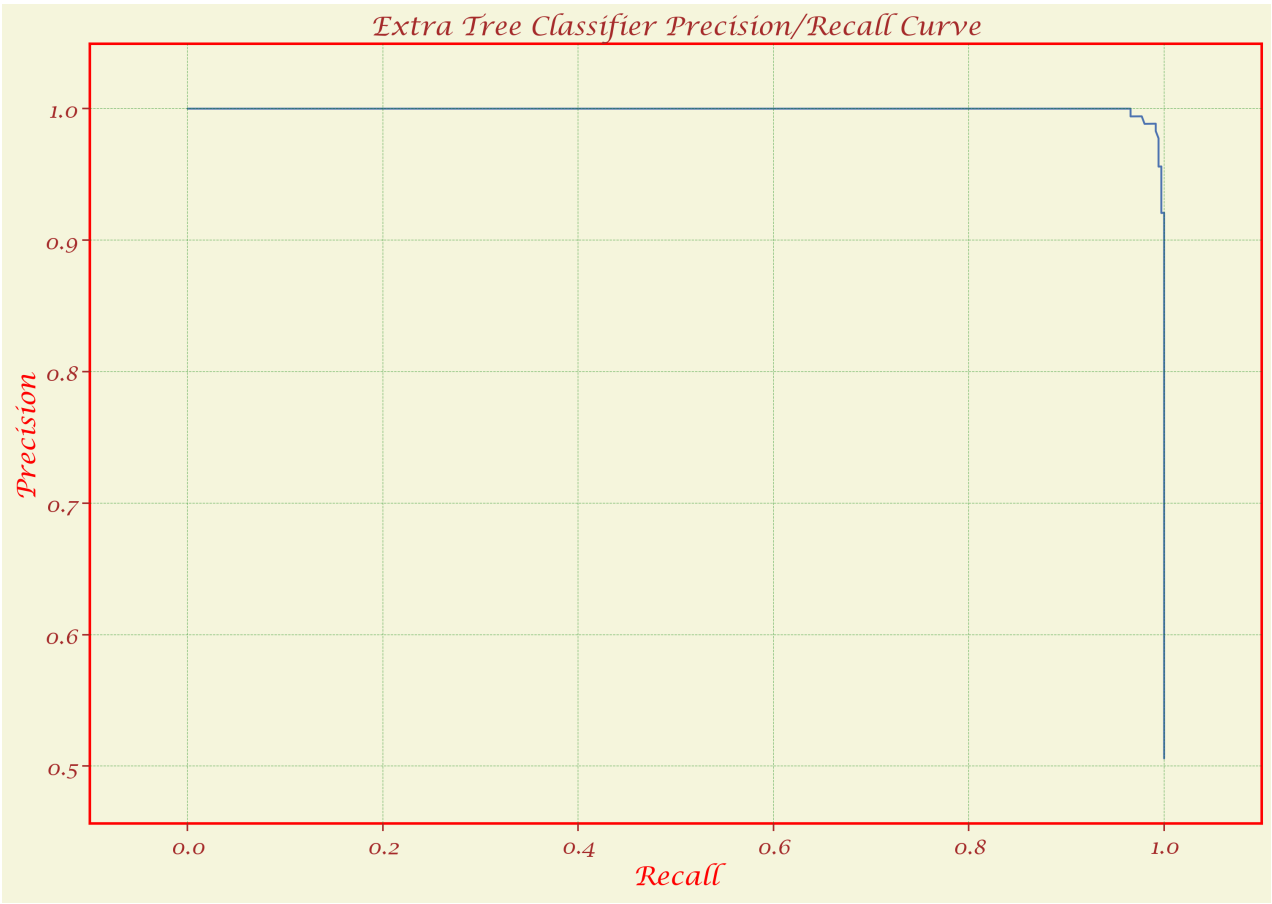
Out[152]:
```
array([[336,   5],
       [  5, 344]], dtype=int64)
```

In [153]:
```
precision_, recall_, thresholds_ = precision_recall_curve(y_train, extra_cfs_decision_score[:, 1])
fpr, tpr, threshold = roc_curve(y_train, extra_cfs_decision_score[:, 1])
```

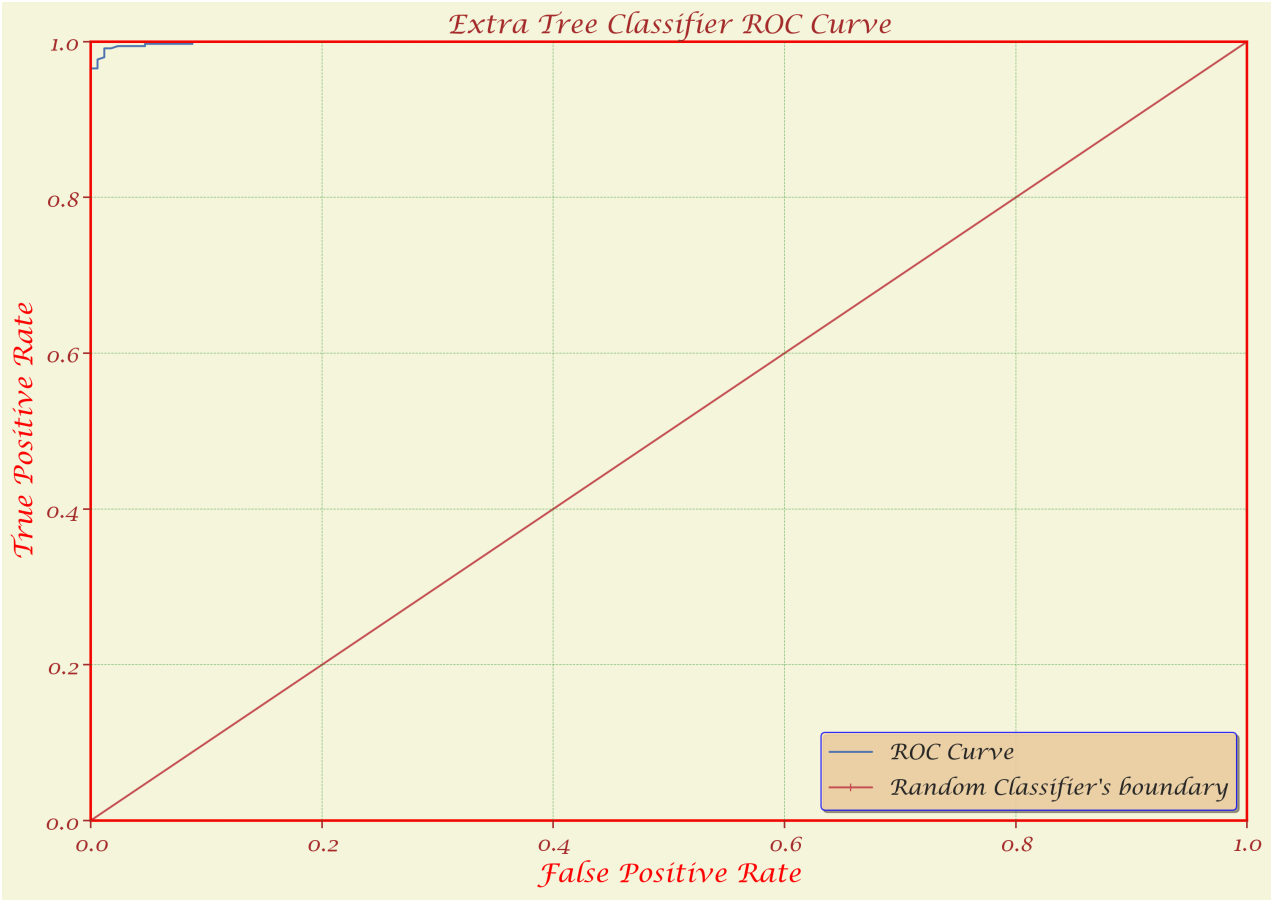
In [154]:
```
p_r_t_curve(precision_, recall_, thresholds_, 'Extra Tree Classifier')
```



In [155]:
```
rc(precision_, recall_, 'Extra Tree Classifier')
```



In [156]:
```
roc_curve_plot(fpr, tpr, threshold, 'Extra Tree Classifier')
```



#9 AdaBoost Classifier

In [157]:
```
from sklearn.ensemble import AdaBoostClassifier
ada_cfs = AdaBoostClassifier()
ada_cfs.fit(X_train, y_train)
ada_cfs_score = ada_cfs.score(X_train, y_train)
ada_cfs_cross_val_accuracy = cross_val_score(ada_cfs, X_train, y_train, cv=5, scoring='accuracy', n_jobs=-1)
ada_cfs_cross_val_predict = cross_val_predict(ada_cfs, X_train, y_train, cv=5, n_jobs=-1)
ada_cfs_decision_score = cross_val_predict(ada_cfs, X_train, y_train, cv=5, n_jobs=-1, method='predict_proba')
```

In [158]:
```
ada_cfs_score
```

Out[158]:
```
0.9492753623188406
```

In [159]:
```
ada_cfs_cross_val_accuracy
```

Out[159]:
```
array([0.89855072, 0.89855072, 0.92028986, 0.86231884, 0.94202899])
```

In [160]:
```
confusion_matrix(y_train, ada_cfs_cross_val_predict)
```

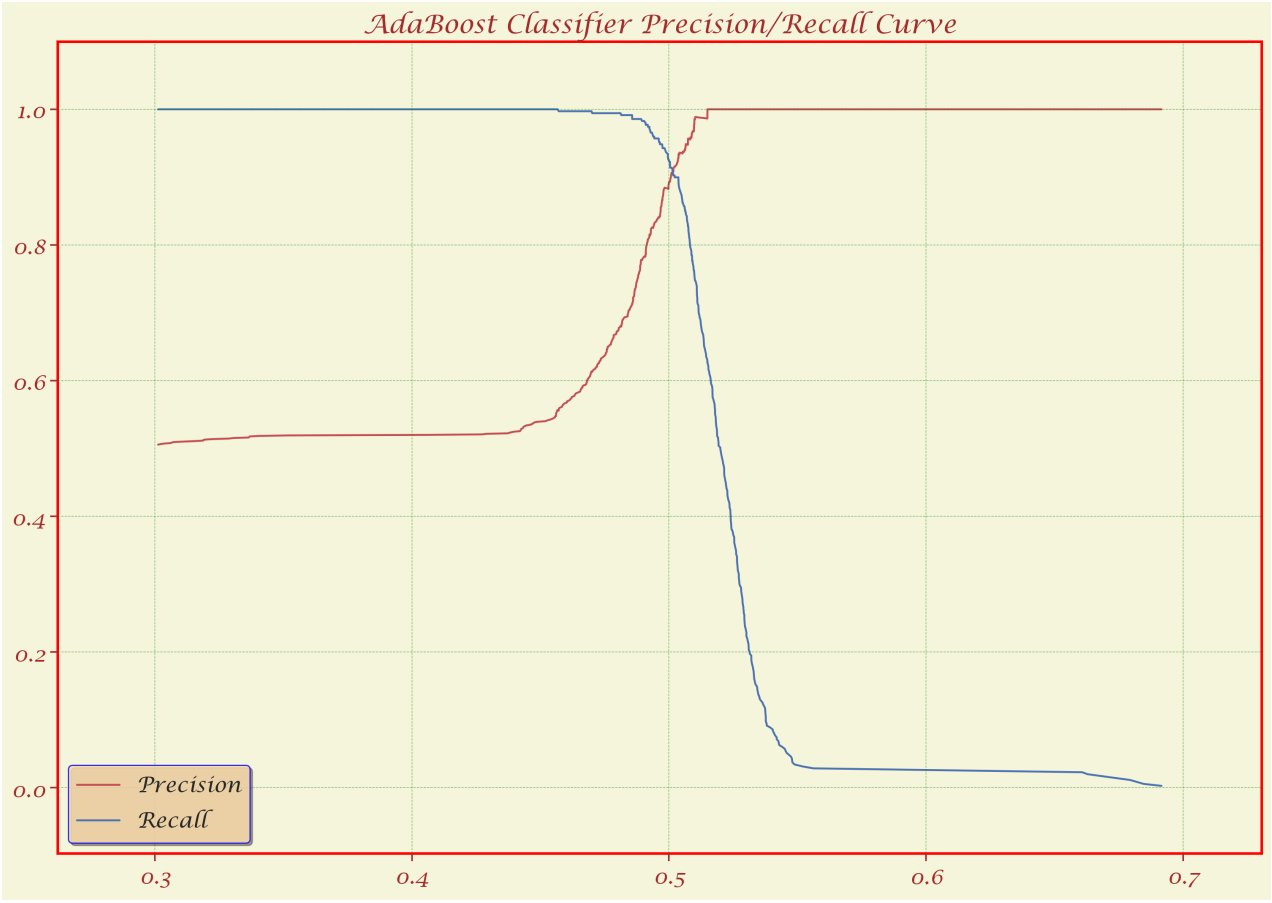
Out[160]:
```
array([[302,  39],
       [ 27, 322]], dtype=int64)
```

In [161]:
```
precision_score(y_train, ada_cfs_cross_val_predict)
```

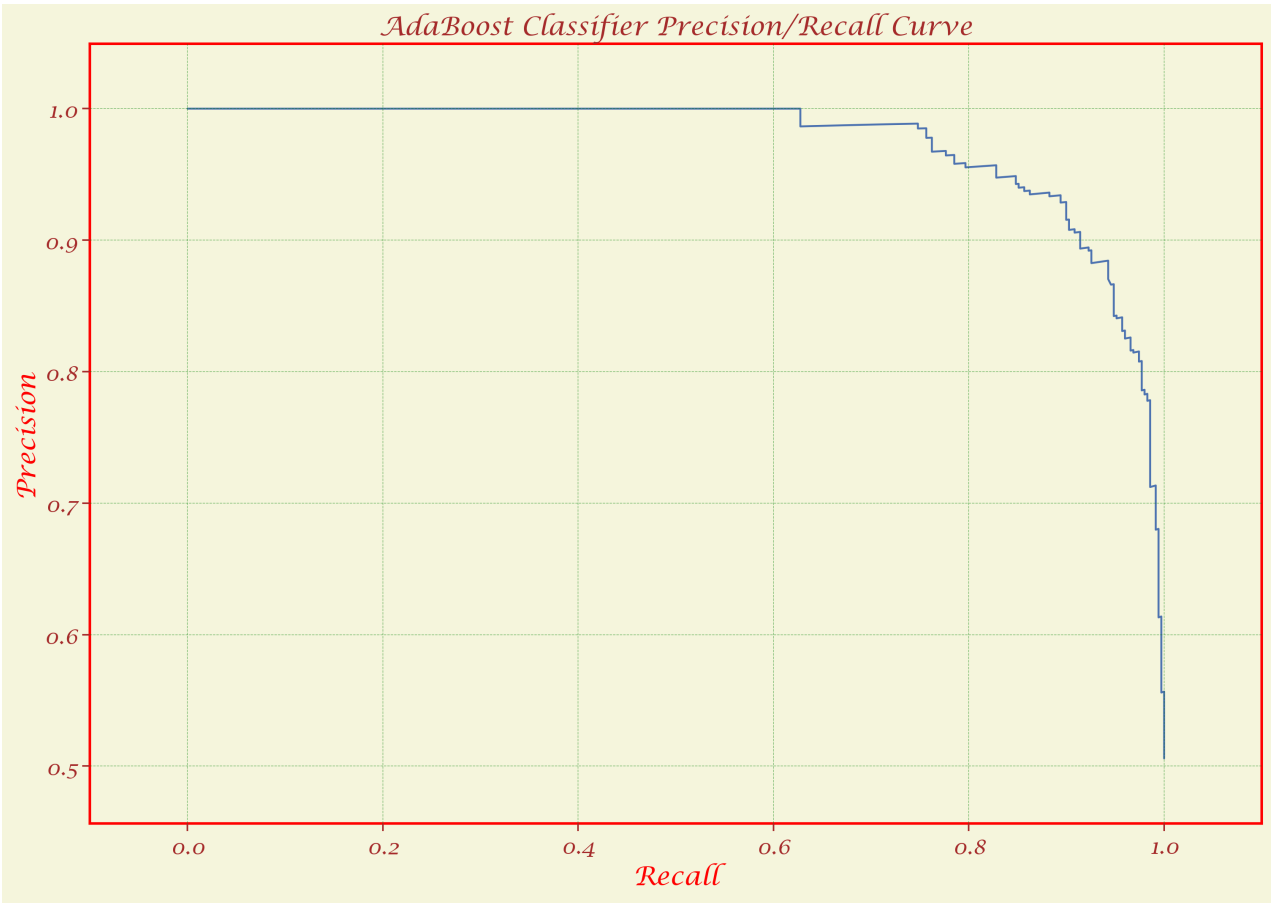
Out[161]:
```
0.8919667590027701
```

In [162]:
```
precision_, recall_, thresholds_ = precision_recall_curve(y_train, ada_cfs_decision_score[:, 1])
fpr, tpr, threshold = roc_curve(y_train, ada_cfs_decision_score[:, 1])
```

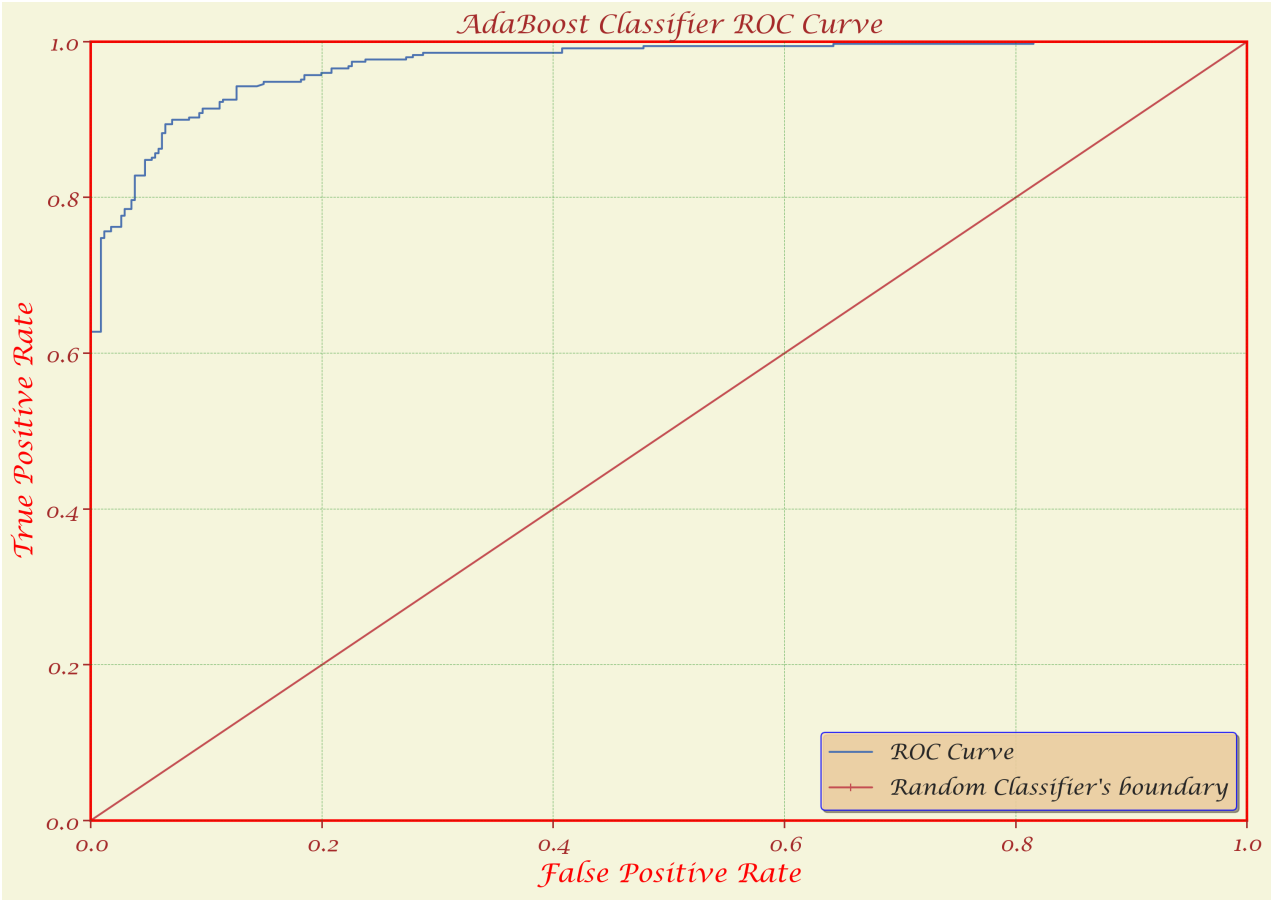
In [163]:
```
p_r_t_curve(precision_, recall_, thresholds_, 'AdaBoost Classifier')
```



In [164]:
```
rc(precision_, recall_, 'AdaBoost Classifier')
```



In [165]:
```
roc_curve_plot(fpr, tpr, threshold, 'AdaBoost Classifier')
```



#10 XGBoost Classifier

In [166]:
```
import xgboost as Xgb
xgb_cfs = Xgb.XGBClassifier()
xgb_cfs.fit(X_train, y_train)
xgb_cfs_score = xgb_cfs.score(X_train, y_train)
xgb_cfs_cross_val_accuracy = cross_val_score(xgb_cfs, X_train, y_train, cv=5, scoring='accuracy', n_jobs=-1)
xgb_cfs_cross_val_predict = cross_val_predict(xgb_cfs, X_train, y_train, cv=5, n_jobs=-1)
xgb_cfs_decision_score = cross_val_predict(xgb_cfs, X_train, y_train, cv=5, n_jobs=-1, method='predict_proba')
```

In [167]:
```
xgb_cfs_score
```

Out[167]:
```
1.0
```

In [168]:
```
xgb_cfs_cross_val_accuracy
```

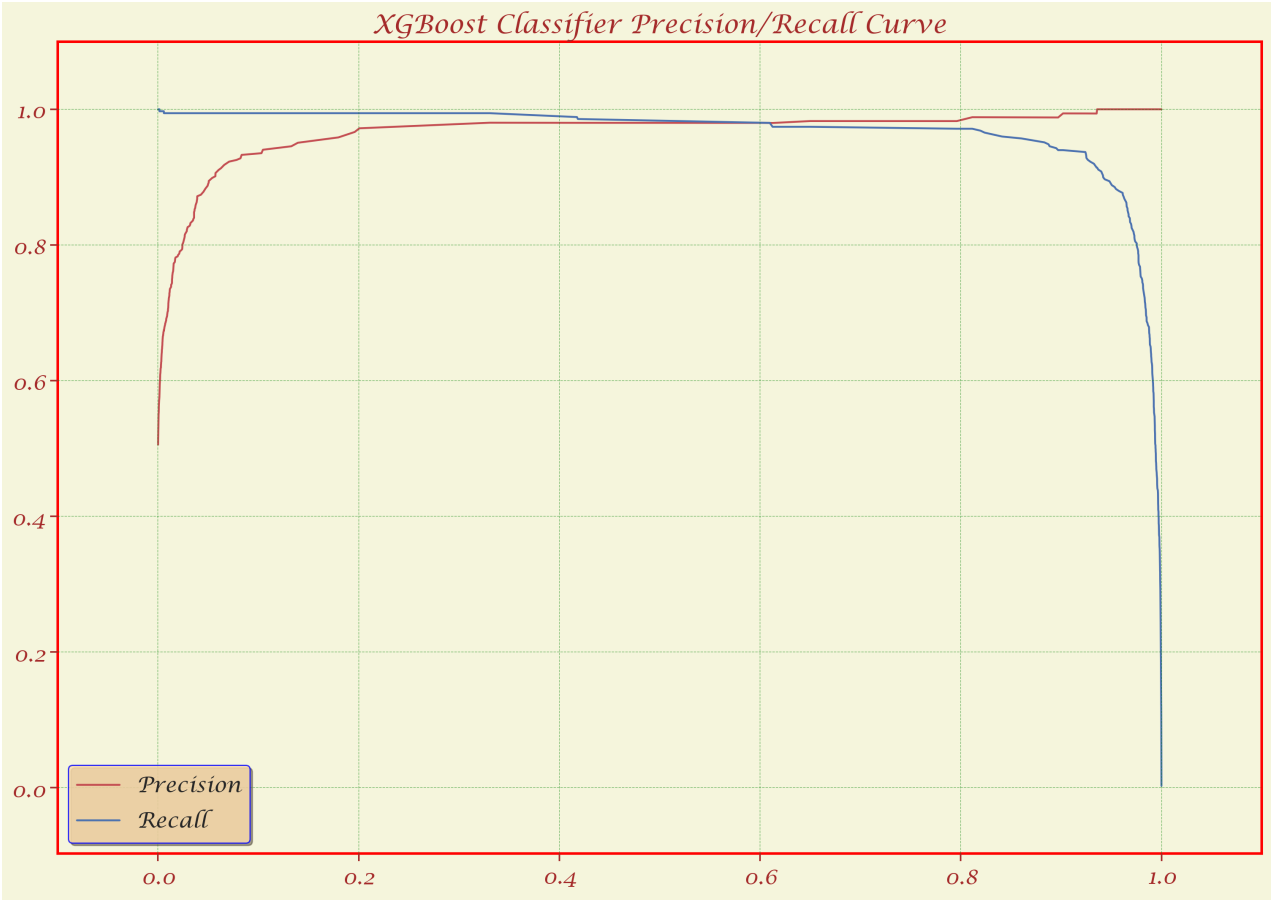
Out[168]:
```
array([0.95652174, 0.97826087, 0.98550725, 0.98550725, 0.99275362])
```

In [169]:
```
confusion_matrix(y_train, xgb_cfs_cross_val_predict)
```

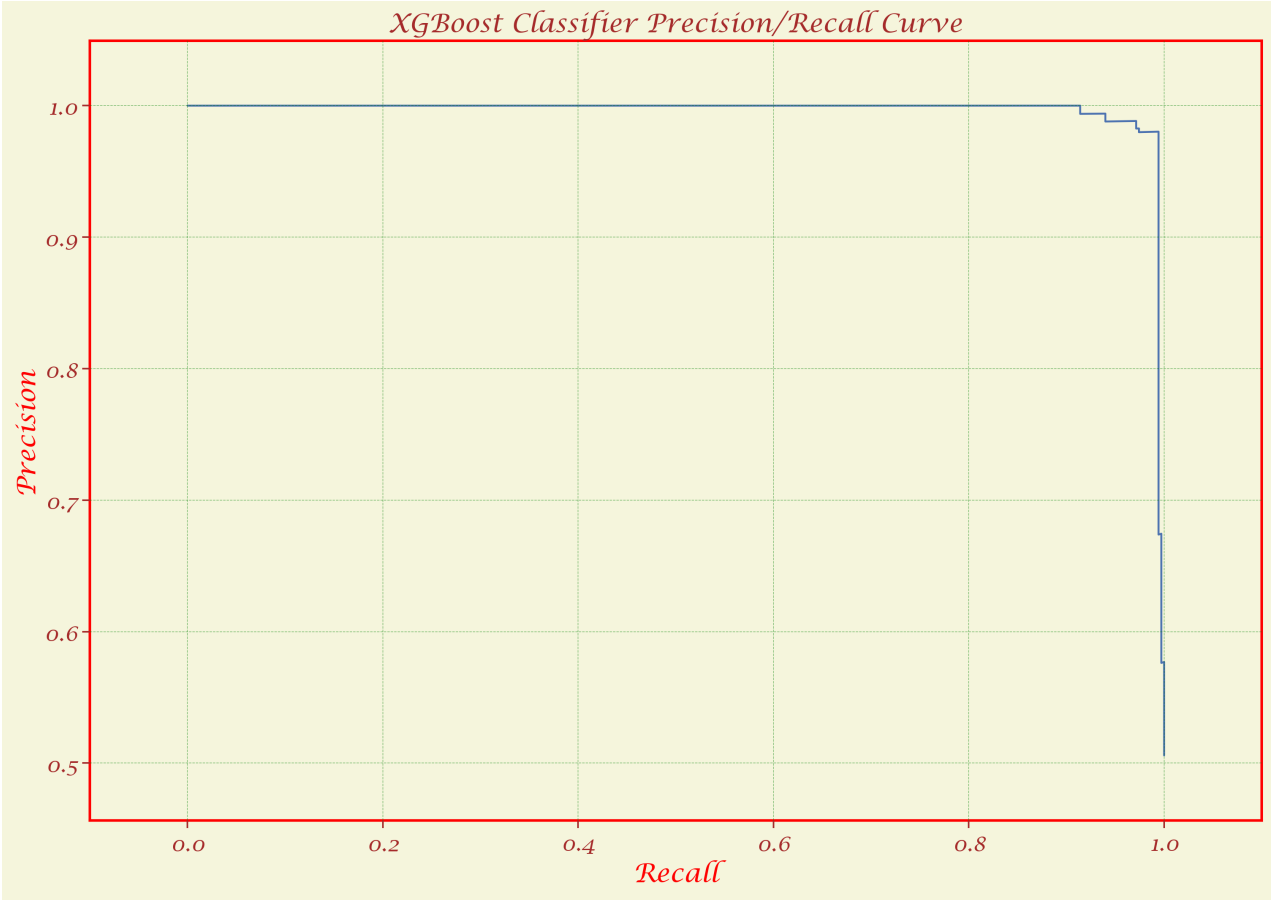
Out[169]:
```
array([[334,   7],
       [  7, 342]], dtype=int64)
```

In [170]:
```
precision_, recall_, thresholds_ = precision_recall_curve(y_train, xgb_cfs_decision_score[:, 1])
fpr, tpr, threshold = roc_curve(y_train, xgb_cfs_decision_score[:, 1])
```

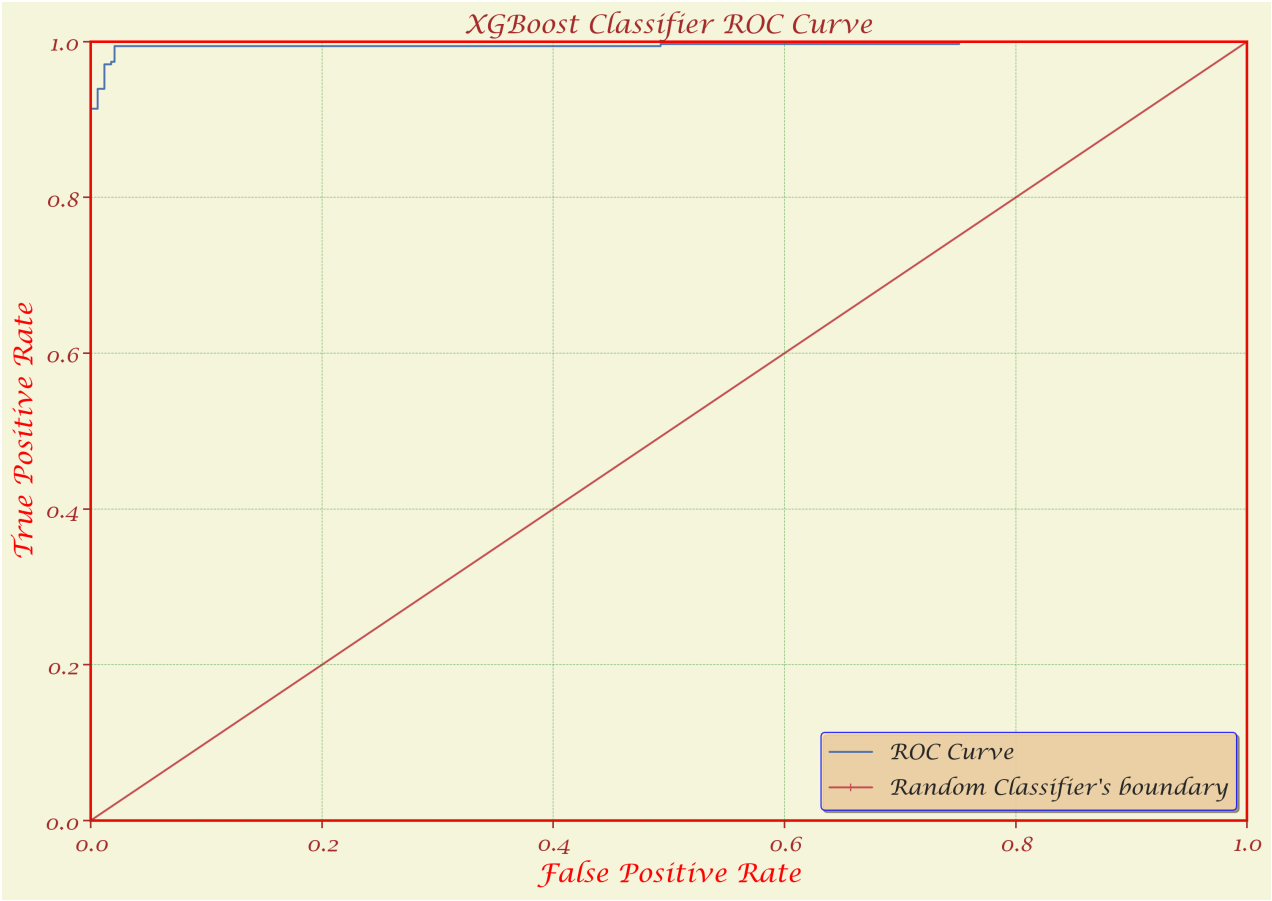
In [171]:
```
p_r_t_curve(precision_, recall_, thresholds_, 'XGBoost Classifier')
```



In [172]:
```
rc(precision_, recall_, 'XGBoost Classifier')
```



In [173]:
```
roc_curve_plot(fpr, tpr, threshold, 'XGBoost Classifier')
```



#11 Perception

In [174]:
```
from sklearn.linear_model import Perceptron
perp_cfs = Perceptron()
perp_cfs.fit(X_train, y_train)
perp_cfs_score = perp_cfs.score(X_train, y_train)
perp_cfs_cross_val_accuracy = cross_val_score(perp_cfs, X_train, y_train, cv=5, scoring='accuracy', n_jobs=-1)
perp_cfs_cross_val_predict = cross_val_predict(perp_cfs, X_train, y_train, cv=5, n_jobs=-1)
perp_cfs_decision_score = cross_val_predict(perp_cfs, X_train, y_train, cv=5, n_jobs=-1, method='decision_function')
```

In [175]:
```
perp_cfs_score
```

Out[175]:
```
0.8043478260869565
```

In [176]:
```
perp_cfs_cross_val_accuracy
```

Out[176]:
```
array([0.81884058, 0.80434783, 0.67391304, 0.77536232, 0.67391304])
```

In [177]:
```
confusion_matrix(y_train, perp_cfs_cross_val_predict)
```

Out[177]:
```
array([[298,  43],
       [130, 219]], dtype=int64)
```

Not worthwhile

#12 MLPClassifier

In [178]:
```
from sklearn.neural_network import MLPClassifier
mlp_cfs = MLPClassifier()
mlp_cfs.fit(X_train, y_train)
mlp_cfs_score = mlp_cfs.score(X_train, y_train)
mlp_cfs_cross_val_accuracy = cross_val_score(mlp_cfs, X_train, y_train, cv=5, scoring='accuracy', n_jobs=-1)
mlp_cfs_cross_val_predict = cross_val_predict(mlp_cfs, X_train, y_train, cv=5, n_jobs=-1)
mlp_cfs_decision_score = cross_val_predict(mlp_cfs, X_train, y_train, cv=5, n_jobs=-1, method='predict_proba')
```

In [179]:
```
mlp_cfs_score
```

Out[179]:
```
0.972463768115942
```

In [180]:
```
mlp_cfs_cross_val_accuracy
```

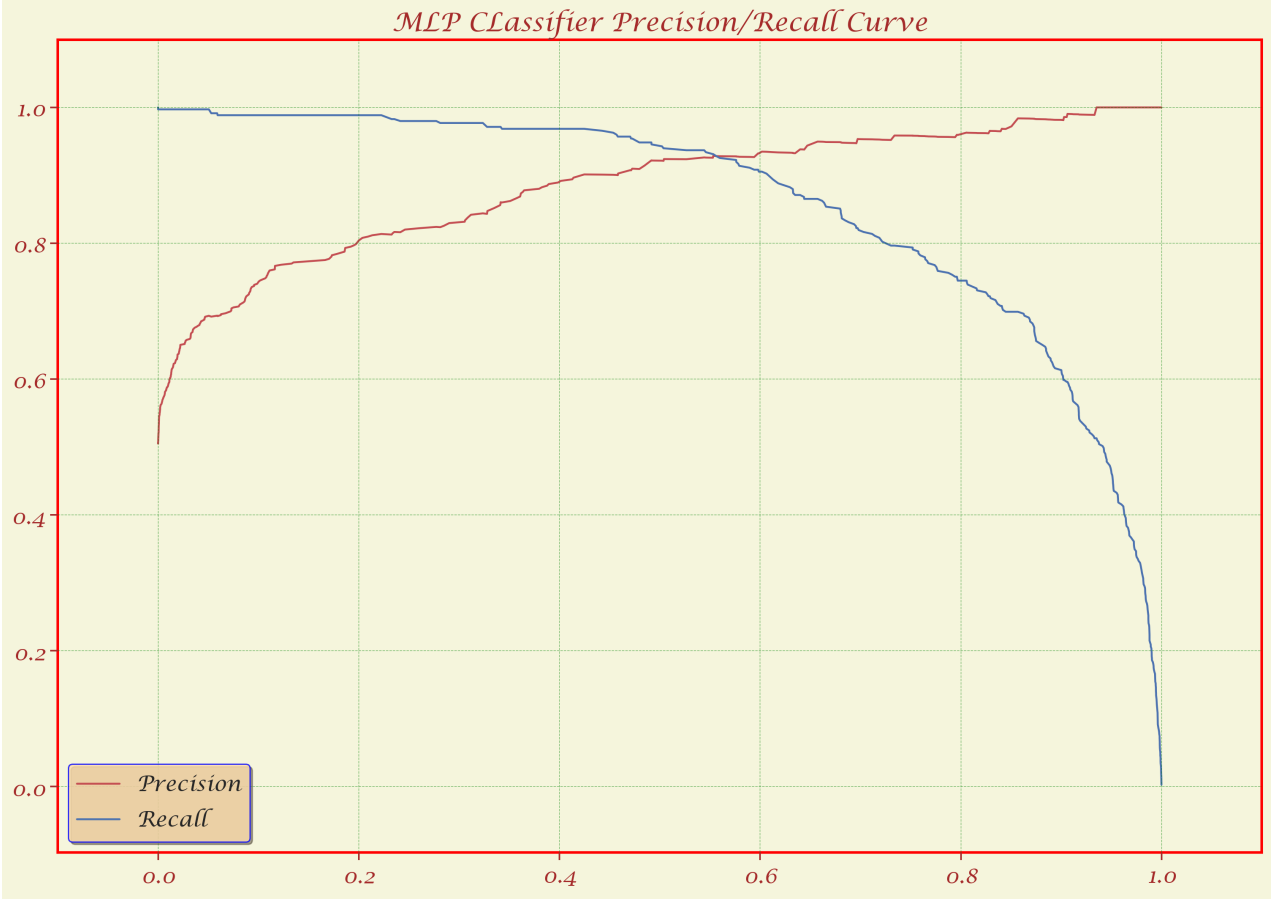
Out[180]:
```
array([0.91304348, 0.89855072, 0.92753623, 0.92753623, 0.94202899])
```

In [181]:
```
confusion_matrix(y_train, mlp_cfs_cross_val_predict)
```

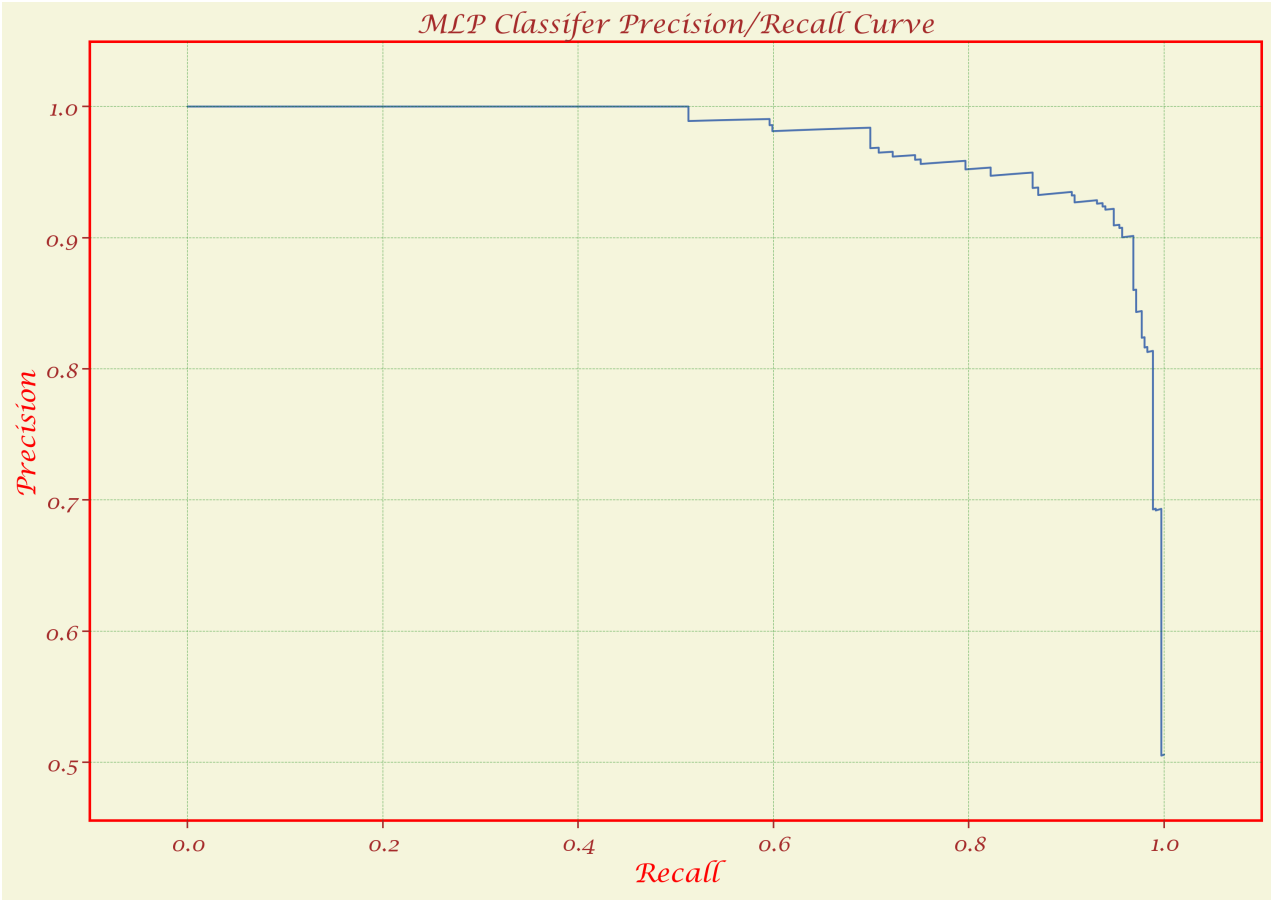
Out[181]:
```
array([[311,  30],
       [ 22, 327]], dtype=int64)
```

In [182]:
```
precision_, recall_, thresholds_ = precision_recall_curve(y_train, mlp_cfs_decision_score[:, 1])
fpr, tpr, threshold = roc_curve(y_train, mlp_cfs_decision_score[:, 1])
```

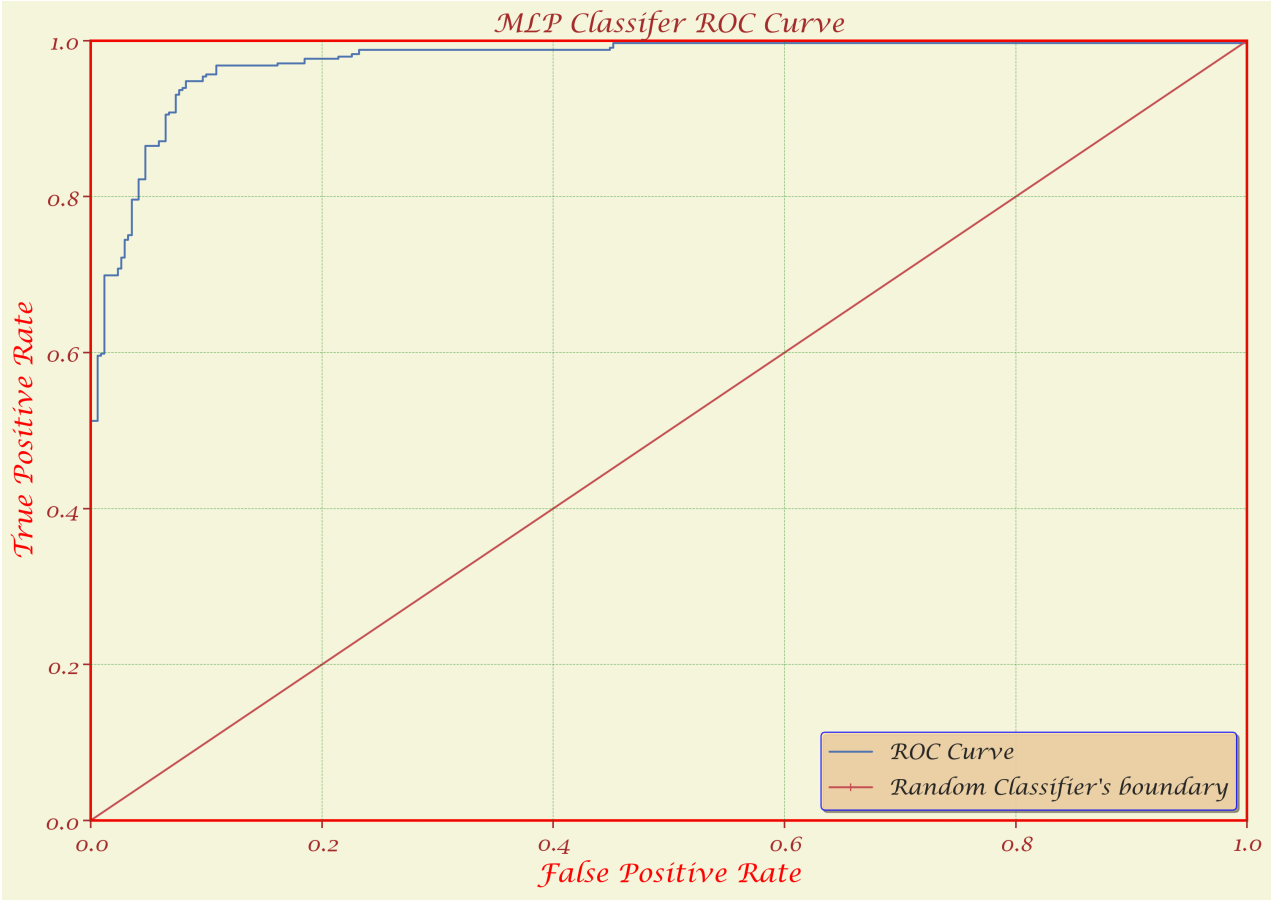
In [183]:
```
p_r_t_curve(precision_, recall_, thresholds_, 'MLP CLassifier')
```



In [184]:
```
rc(precision_, recall_, 'MLP Classifer')
```



In [185]:
```
roc_curve_plot(fpr, tpr, threshold, 'MLP Classifer')
```



#13 Voting Classifier

In [186]:
```
from sklearn.ensemble import VotingClassifier
voting_cfs = VotingClassifier([('ln', LogisticRegression()),
                               ('svc', SVC(probability=True)),
                               ('knn', KNeighborsClassifier()),
                               ('tree', DecisionTreeClassifier())], voting='soft')
voting_cfs.fit(X_train, y_train)
voting_cfs_score = voting_cfs.score(X_train, y_train)
voting_cfs_cross_val_accuracy = cross_val_score(voting_cfs, X_train, y_train, cv=5, scoring='accuracy', n_jobs=-1)
voting_cfs_cross_val_predict = cross_val_predict(voting_cfs, X_train, y_train, cv=5, n_jobs=-1)
voting_cfs_decision_score = cross_val_predict(voting_cfs, X_train, y_train, cv=5, n_jobs=-1, method='predict_proba')
```

In [187]:
```
voting_cfs_score
```

Out[187]:
```
0.9695652173913043
```

In [188]:
```
voting_cfs_cross_val_accuracy
```

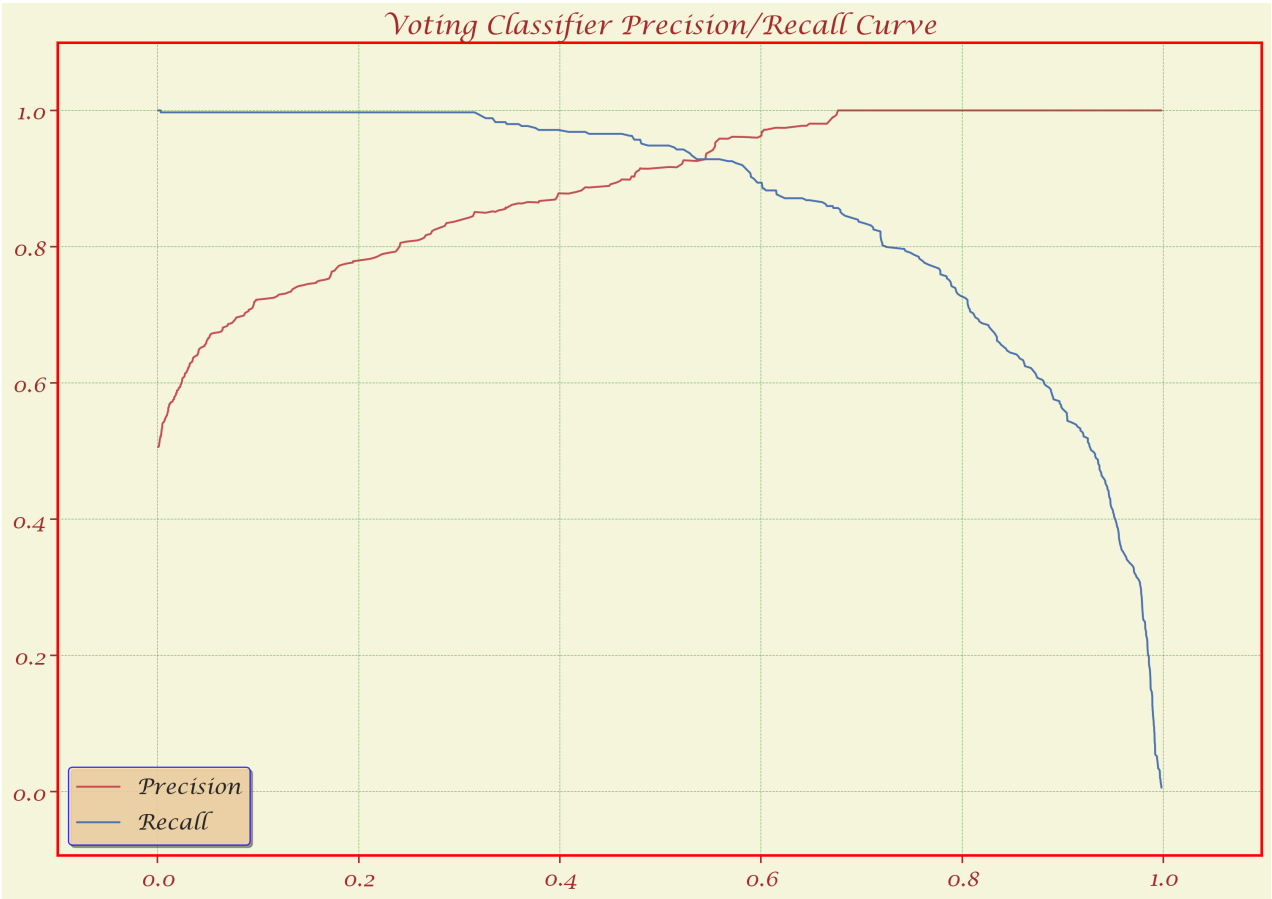
Out[188]:
```
array([0.89855072, 0.92753623, 0.92028986, 0.93478261, 0.97101449])
```

In [189]:
```
confusion_matrix(y_train, voting_cfs_cross_val_predict)
```

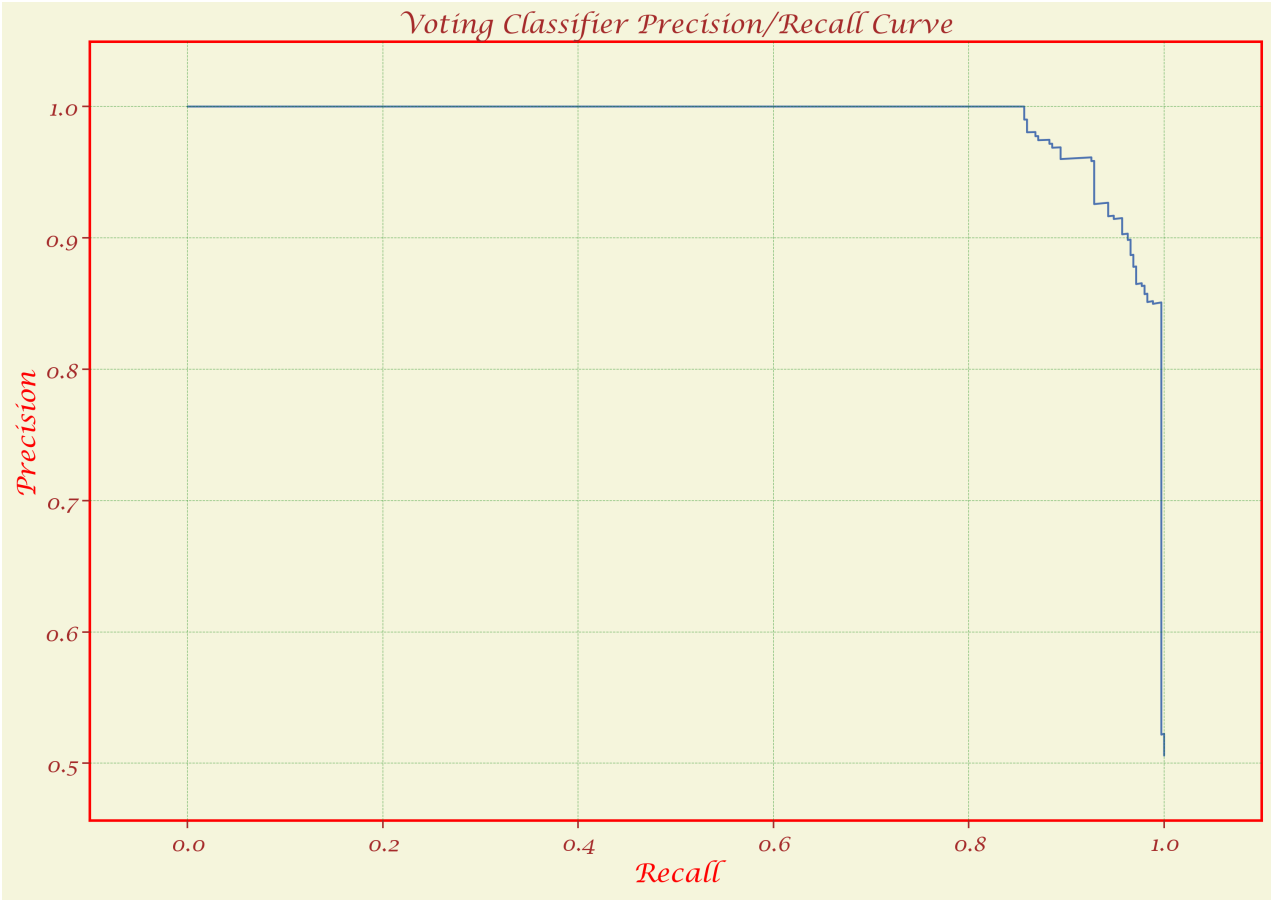
Out[189]:
```
array([[311,  30],
       [ 18, 331]], dtype=int64)
```

In [190]:
```
precision_, recall_, thresholds_ = precision_recall_curve(y_train, voting_cfs_decision_score[:, 1])
fpr, tpr, threshold = roc_curve(y_train, voting_cfs_decision_score[:, 1])
```

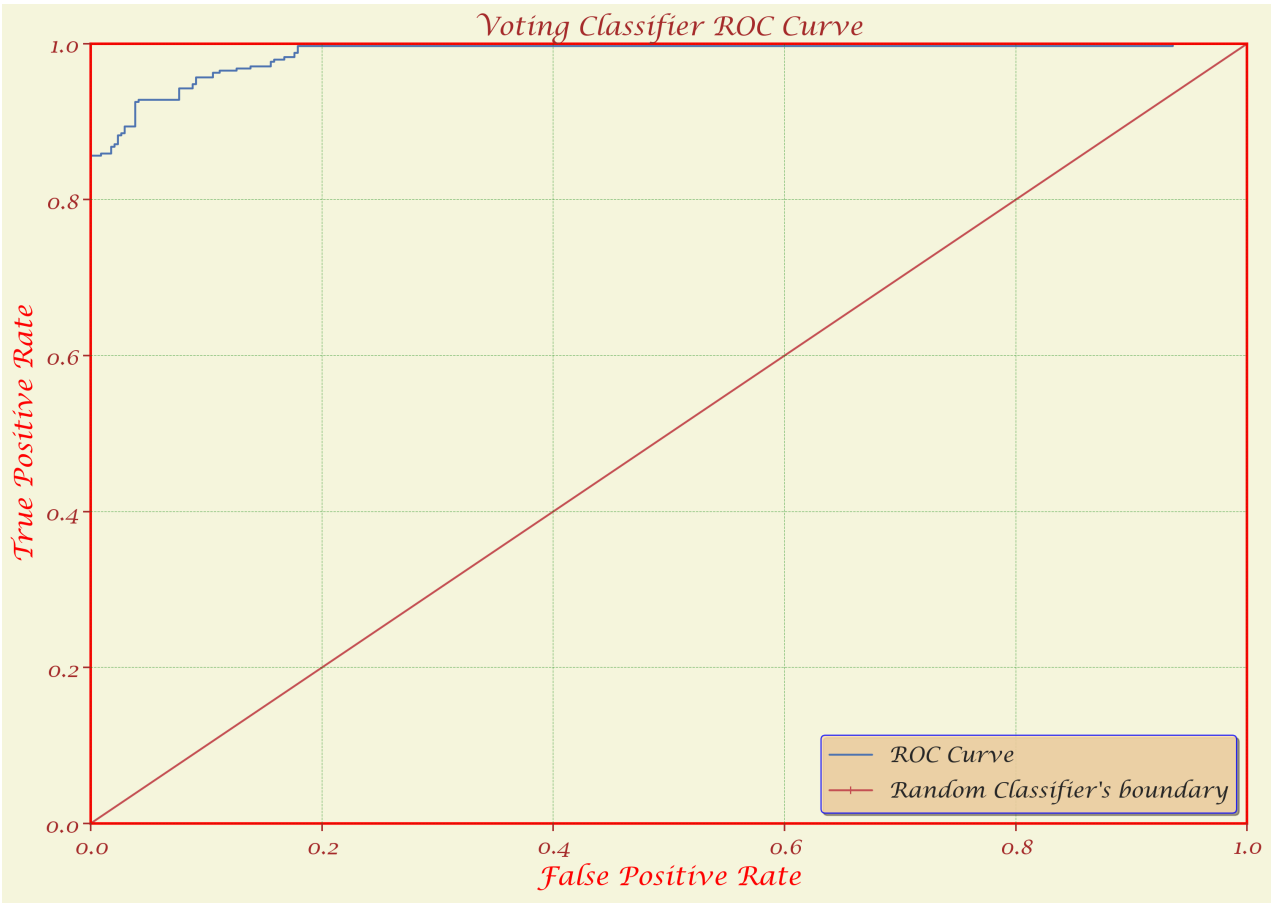
In [191]:
```
p_r_t_curve(precision_, recall_, thresholds_, 'Voting Classifier')
```



In [192]:
```
rc(precision_, recall_, 'Voting Classifier')
```



In [193]:
```
roc_curve_plot(fpr, tpr, threshold, 'Voting Classifier')
```



#14 Stack Classifier

In [194]:
```
%%time
from sklearn.ensemble import StackingClassifier
estimators = [('lr', LogisticRegression()), ('SVC', SVC(probability=True)), ('knn', KNeighborsClassifier()),
              ('extra', ExtraTreesClassifier(n_estimators=20)), ('mlp', MLPClassifier()), ('xg', Xgb.XGBClassifier())]
stack_cfs = StackingClassifier(estimators=estimators, final_estimator=RandomForestClassifier(n_estimators=20))
stack_cfs.fit(X_train, y_train)
stack_cfs_score = stack_cfs.score(X_train, y_train)
stack_cfs_cross_val_accuracy = cross_val_score(stack_cfs, X_train, y_train, cv=5, scoring='accuracy', n_jobs=-1)
stack_cfs_cross_val_predict = cross_val_predict(stack_cfs, X_train, y_train, cv=5, n_jobs=-1)
stack_cfs_decision_score = cross_val_predict(stack_cfs, X_train, y_train, cv=5, n_jobs=-1, method='predict_proba')
```

```
CPU times: total: 9.34 s
Wall time: 22.2 s
```

In [195]:
```
stack_cfs_score
```

Out[195]:
```
1.0
```

In [196]:
```
stack_cfs_cross_val_accuracy
```

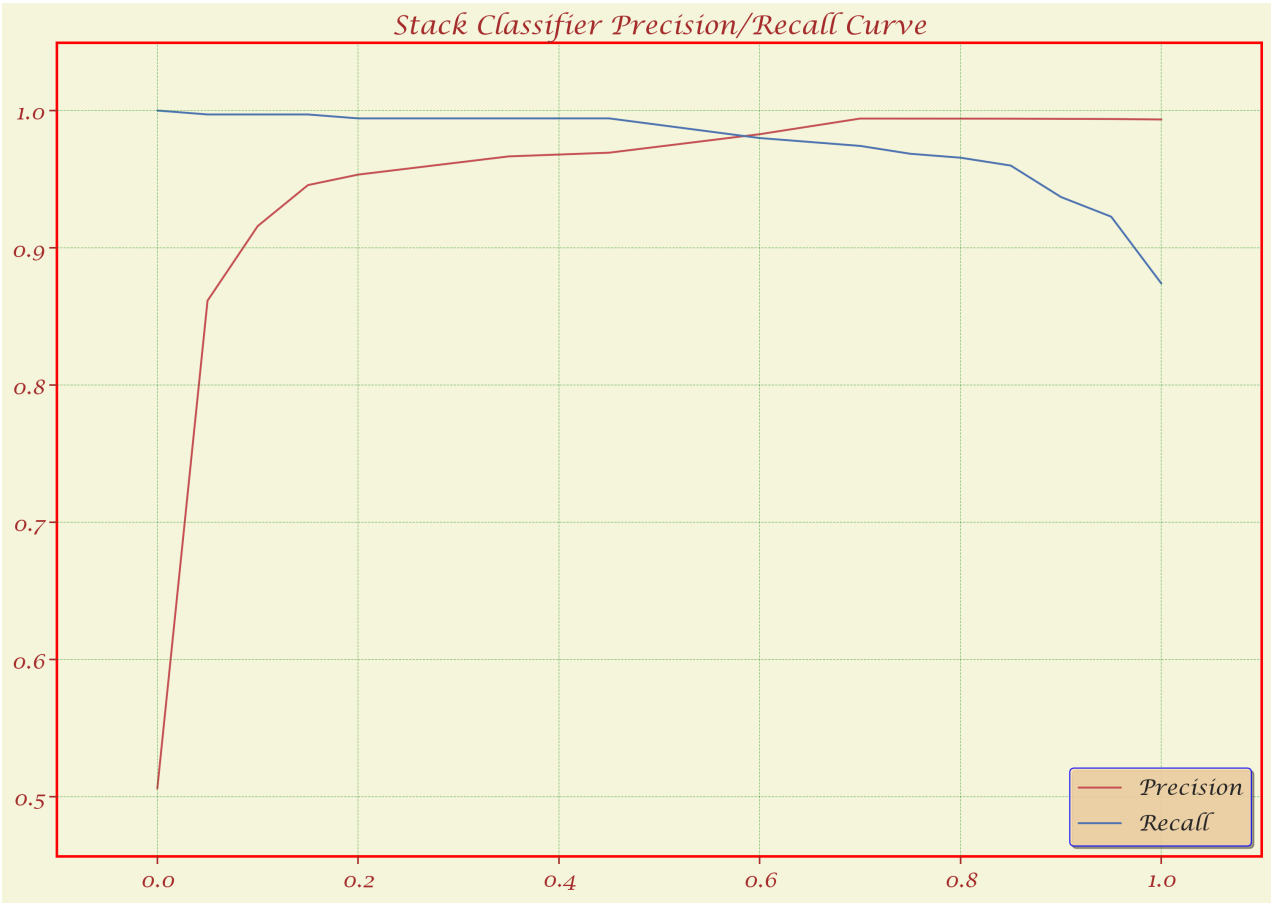
Out[196]:
```
array([0.97101449, 0.98550725, 0.98550725, 0.98550725, 1.        ])
```

In [197]:
```
confusion_matrix(y_train, stack_cfs_cross_val_predict)
```

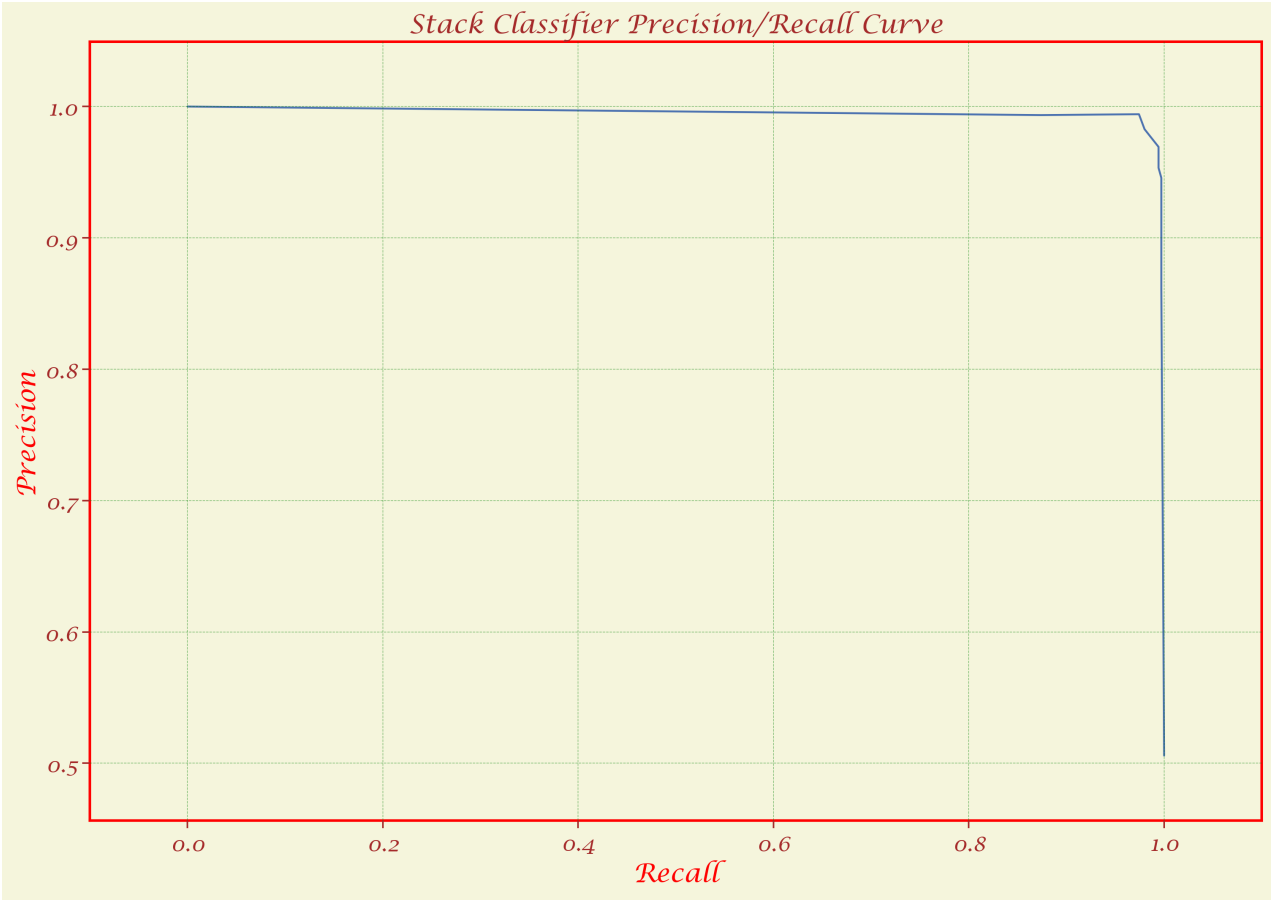
Out[197]:
```
array([[334,   7],
       [  9, 340]], dtype=int64)
```

In [198]:
```
precision_, recall_, thresholds_ = precision_recall_curve(y_train, stack_cfs_decision_score[:, 1])
fpr, tpr, threshold = roc_curve(y_train, stack_cfs_decision_score[:, 1])
```

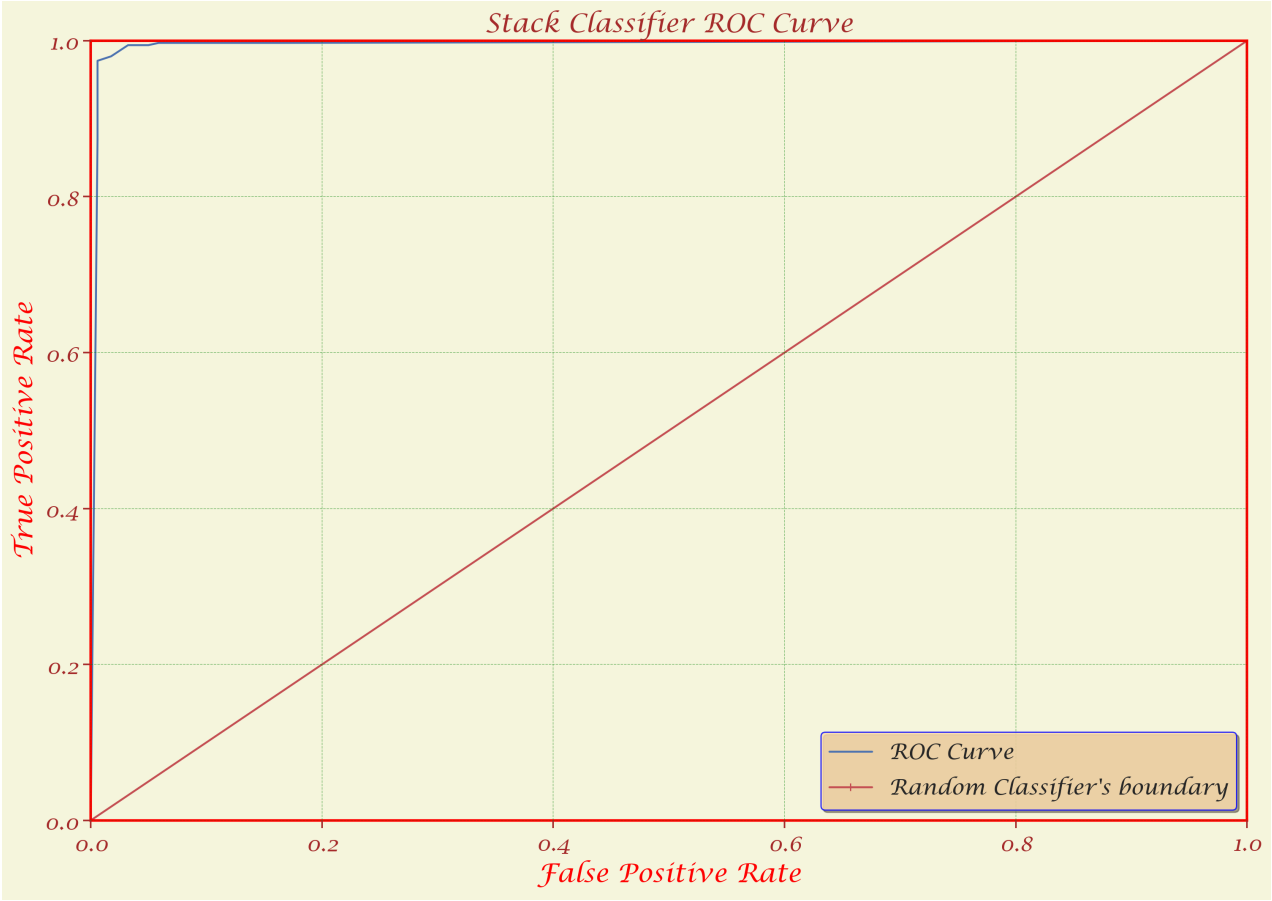
In [199]:
```
p_r_t_curve(precision_, recall_, thresholds_, 'Stack Classifier')
```



In [200]:
```
rc(precision_, recall_, 'Stack Classifier')
```



In [201]:
```
roc_curve_plot(fpr, tpr, threshold, 'Stack Classifier')
```



In [202]:
```
models = ['Logistic Regresssion', 'SGD Classifier', 'KNN', 'Linear SVC', 'SVC', 'Decision Tree', 'Random Forest',
          'Extra Tree', 'AdaBoost', 'XGBoost', 'Perception', 'MLP Classifier', 'Voting Classifier', 'Stacking Classifier']
model_accuracy =[lin_cfs_score, sgd_cfs_score, knn_cfs_score, lin_svc_cfs_score, svc_cfs_score, dec_cfs_score,
                 rnd_cfs_score, extra_cfs_score, ada_cfs_score, xgb_cfs_score, perp_cfs_score, mlp_cfs_score, voting_cfs_score
                 stack_cfs_score]
cross_val = [lin_cfs_cross_val_accuracy.mean(), sgd_cfs_cross_val_accuracy.mean(), knn_cfs_cross_val_accuracy.mean(),
             lin_svc_cfs_cross_val_accuracy.mean(), svc_cfs_cross_val_accuracy.mean(), dec_cfs_cross_val_accuracy.mean(),
             rnd_cfs_cross_val_accuracy.mean(), extra_cfs_cross_val_accuracy.mean(), ada_cfs_cross_val_accuracy.mean(),
             xgb_cfs_cross_val_accuracy.mean(), perp_cfs_cross_val_accuracy.mean(), mlp_cfs_cross_val_accuracy.mean(),
             voting_cfs_cross_val_accuracy.mean(), stack_cfs_cross_val_accuracy.mean()]
```

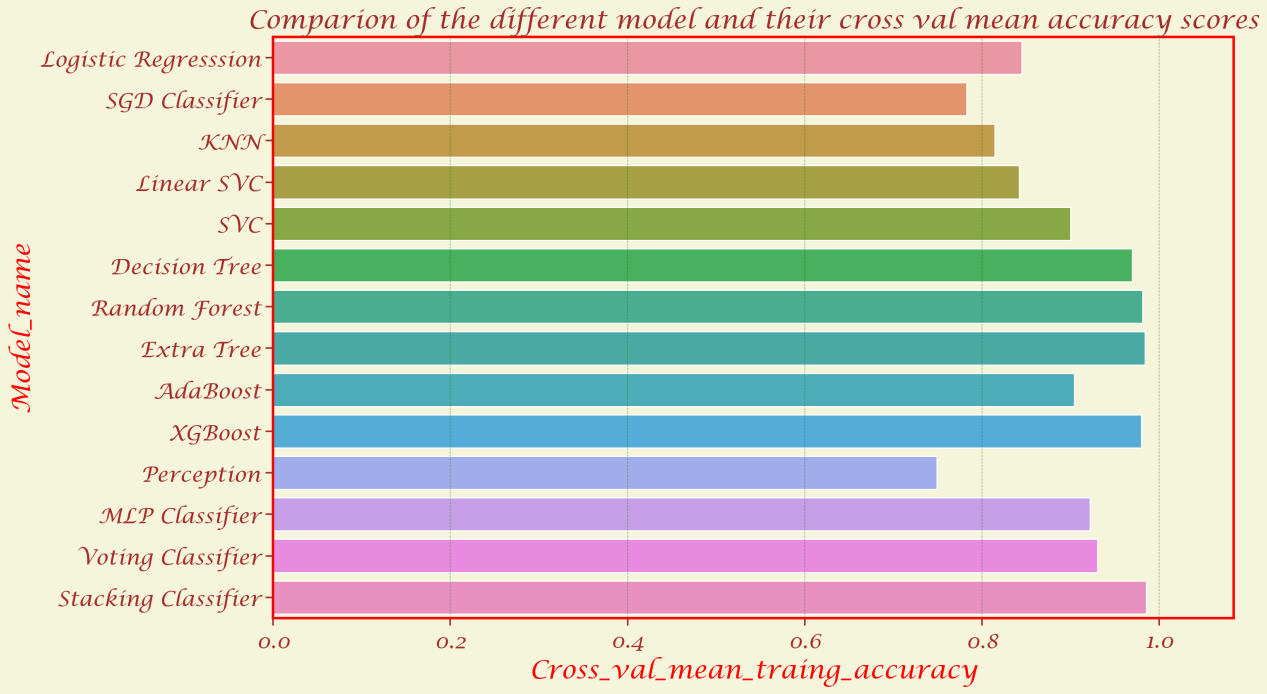
In [203]:
```
model_data = pd.DataFrame({'Model_name':models, 'Model_Accuracy':model_accuracy, 'Cross_val_mean_traing_accuracy':cross_val}
```

In [204]:
```
ta.sort_values(by='Cross_val_mean_traing_accuracy', ascending=False).style.background_gradient(cmap='hot').hide().set_propert
```

Out[204]:

Model_name	Model_Accuracy	Cross_val_mean_traing_accuracy
Stacking Classifier	1.000000	0.985507
Extra Tree	1.000000	0.984058
Random Forest	1.000000	0.981159
XGBoost	1.000000	0.979710
Decision Tree	1.000000	0.969565
Voting Classifier	0.969565	0.930435
MLP Classifier	0.972464	0.921739
AdaBoost	0.949275	0.904348
SVC	0.930435	0.900000
Logistic Regresssion	0.860870	0.844928
Linear SVC	0.860870	0.842029
KNN	0.959420	0.814493
SGD Classifier	0.750725	0.782609
Perception	0.804348	0.749275

In [205]:
```
plt.figure(figsize=(14, 8))
sns.barplot(data=model_data, x='Cross_val_mean_traing_accuracy', y='Model_name')
plt.title('Comparion of the different model and their cross val mean accuracy scores')
plt.show()
```

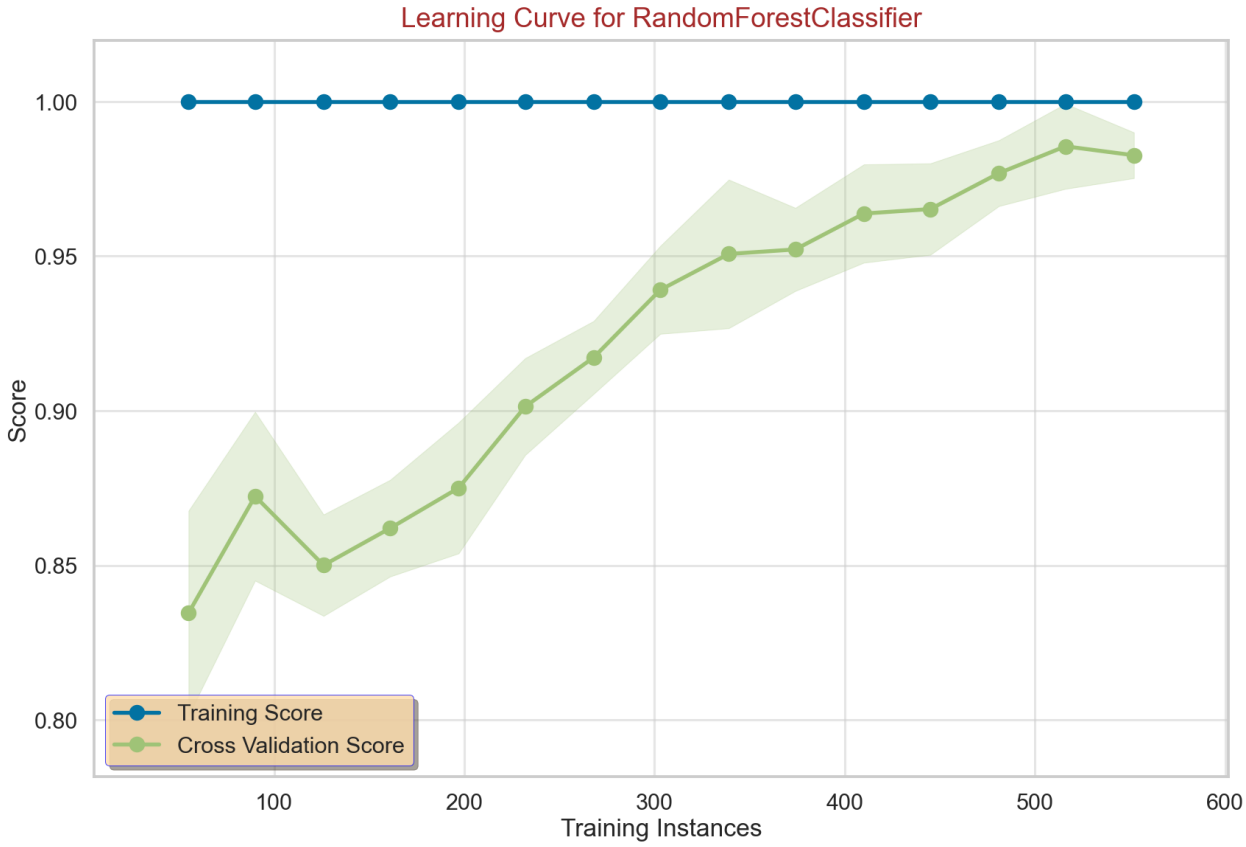


Stacking, EXtra Tree and Random Forest are the best models

To check their over and under fitness lets compare them by using Learing Curves

In [206]:
```
from sklearn.model_selection import learning_curve
train_size, train_score, test_score = learning_curve(rnd_cfs, X_train, y_train, cv=5,
                                                     train_sizes=np.linspace(0.1, 1, 20), n_jobs=-1, scoring='accuracy')
```

In [207]:
```
from yellowbrick.model_selection import LearningCurve
rand_curve = LearningCurve(estimator=RandomForestClassifier(), cv=5, train_sizes=np.linspace(0.1, 1, 15), n_jobs=-1, scoring=
rand_curve.fit(X_train, y_train)
rand_curve.show()
```

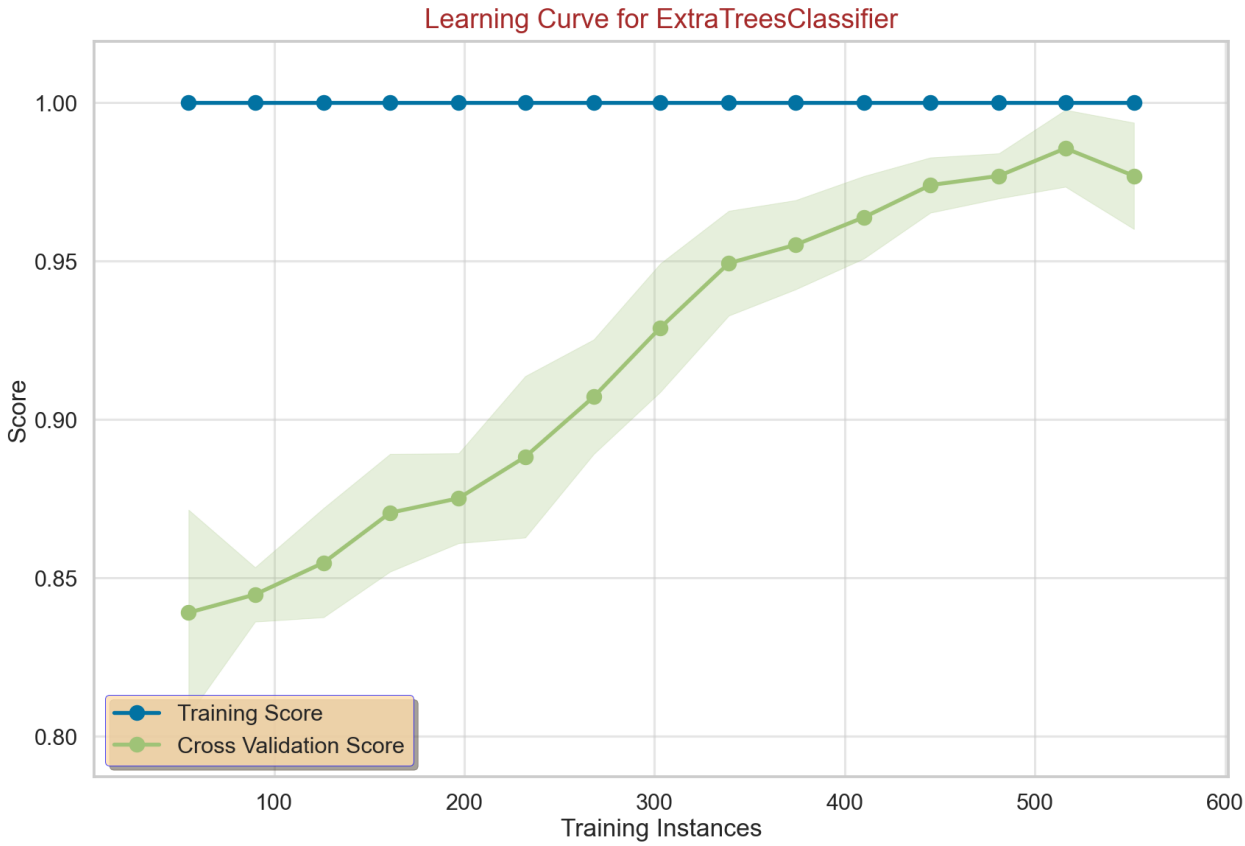


Out[207]: <Axes: title={'center': 'Learning Curve for RandomForestClassifier'}, xlabel='Training Instances', ylabel='Score'>

In [208]:
```
train_score[:-10]
```

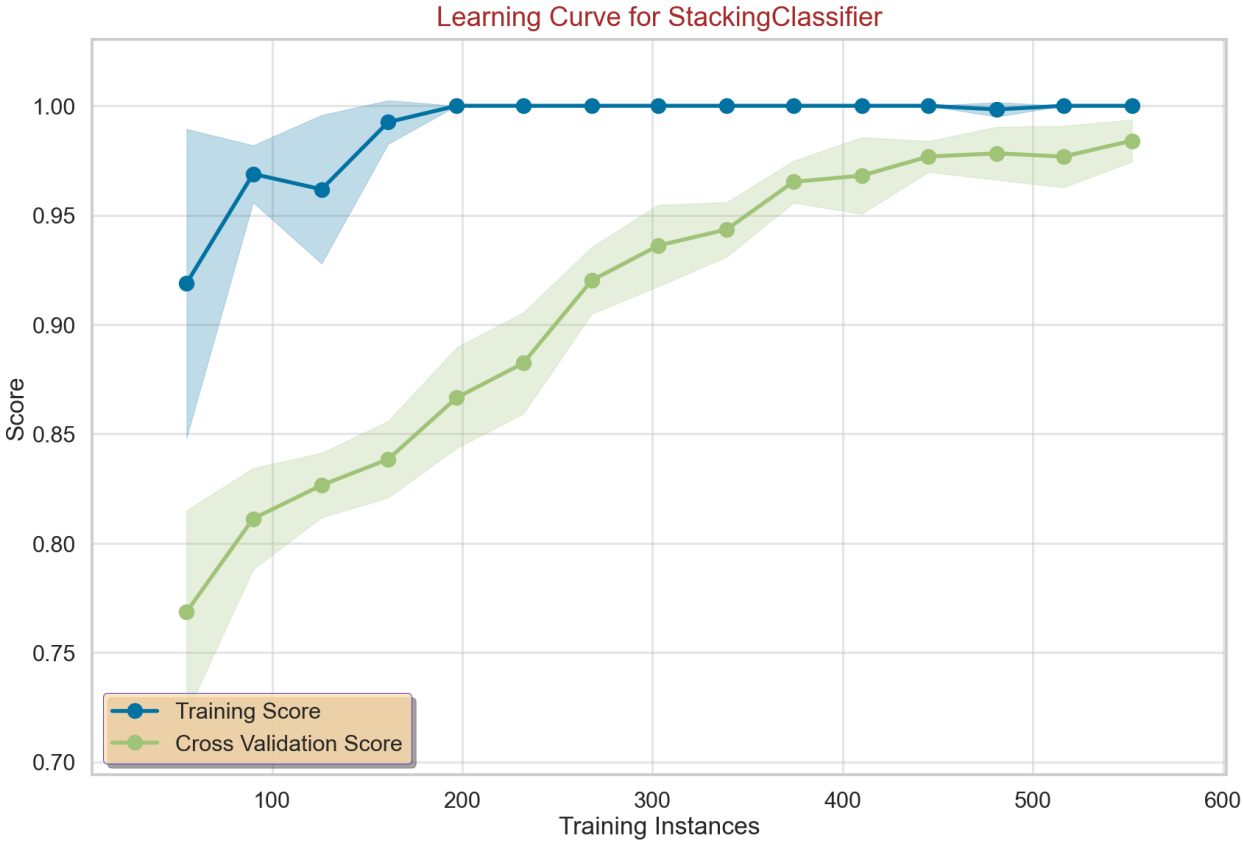
Out[208]:
```
array([[1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.]])
```

In [209]:
```
extra_curve = LearningCurve(estimator=ExtraTreesClassifier(), cv=5, train_sizes=np.linspace(0.1, 1, 15), n_jobs=-1, scoring=
extra_curve.fit(X_train, y_train)
extra_curve.show()
```



Out[209]: <Axes: title={'center': 'Learning Curve for ExtraTreesClassifier'}, xlabel='Training Instances', ylabel='Score'>

In [210]:
```
%%time
estimators = [('lr', LogisticRegression()), ('SVC', SVC(probability=True)), ('knn', KNeighborsClassifier()),
              ('extra', ExtraTreesClassifier(n_estimators=20)), ('mlp', MLPClassifier()), ('xg', Xgb.XGBClassifier())]
stack1 = StackingClassifier(estimators=estimators, final_estimator=RandomForestClassifier(n_estimators=20))
stack1_curve = LearningCurve(estimator=stack1, cv=5, train_sizes=np.linspace(0.1, 1, 15), n_jobs=-1, scoring='f1_weighted')
stack1_curve.fit(X_train, y_train)
stack1_curve.show()
```




```
CPU times: total: 6min 41s
Wall time: 2min 12s
```

Out[210]:
```
<Axes: title={'center': 'Learning Curve for StackingClassifier'}, xlabel='Training Instances', ylabel='Score'>
```

Above three models behaved similarly but stacking classifier fitted data more accurately than other two models

so pick this model for final test amd before doing that itis good to do some hypertuning

In [215]:
```
estimators = [('lr', LogisticRegression()), ('svc', SVC(probability=True)), ('knn', KNeighborsClassifier(n_jobs=-1)),
              ('extra', ExtraTreesClassifier(n_jobs=-1)), ('mlp', MLPClassifier()), ('xg', Xgb.XGBClassifier(n_jobs=-1))]
stack1 = StackingClassifier(estimators=estimators, final_estimator=RandomForestClassifier(n_estimators=40, max_samples=4, ma
```

In [ ]:
```
          'rand__n_estimators':[20, 30, 40, 50], 'rand__max_samples' :[4, 5, 6], 'rand__max_depth' :[2, 3], 'rand__max_featu
```

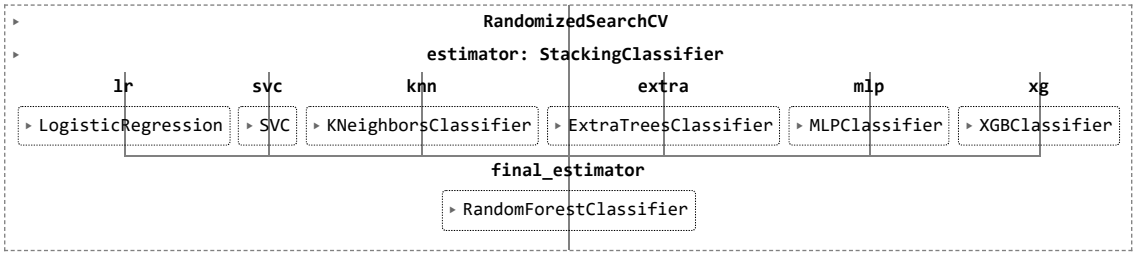
In [216]:
```
%%time
from sklearn.model_selection import RandomizedSearchCV
params_list = {'lr__penalty' :['l1', 'l2', 'elasticnet'], 'lr__solver': ['lbfgs', 'liblinear'],
               'svc__gamma':['scale', 'auto'],
              'knn__n_neighbors' : [2, 3, 4, 5, 6], 'knn__weights' :['uniform', 'distance'],
              'extra__n_estimators' : [30, 50, 100, 150, 200], 'extra__max_depth':[None, 2, 3], 'extra__min_samples_split':[
              'mlp__solver':['lbfgs', 'sgd', 'adam'], 'mlp__learning_rate':['constant', 'invscaling', 'adaptive'],
              'xg__n_estimators':[2, 4, 6, 8], 'xg__max_leaves' :[4, 6, 8], 'xg__depth' :[2, 4, 6]}
random_cv = RandomizedSearchCV(estimator=stack1, param_distributions=params_list, n_jobs=-1, scoring='f1_weighted')
```

```
CPU times: total: 0 ns
Wall time: 0 ns
```

In [217]:
```
%%time
random_cv.fit(X_train, y_train)
```

```
CPU times: total: 2.56 s
Wall time: 19.8 s
```

Out[217]:



In [218]:
```
random_cv.best_params_
```

Out[218]:
```
{'xg__n_estimators': 6,
 'xg__max_leaves': 4,
 'xg__depth': 4,
 'svc__gamma': 'auto',
 'mlp__solver': 'lbfgs',
 'mlp__learning_rate': 'adaptive',
 'lr__solver': 'liblinear',
 'lr__penalty': 'l1',
 'knn__weights': 'uniform',
 'knn__n_neighbors': 5,
 'extra__n_estimators': 200,
 'extra__min_samples_split': 6,
 'extra__max_features': 'log2',
 'extra__max_depth': None}
```

In [219]:
```
random_cv.score
```

Out[219]:
```
<bound method BaseSearchCV.score of RandomizedSearchCV(estimator=StackingClassifier(estimators=[('lr',
                                                                  LogisticRegression()),
                                                                 ('svc',
                                                                  SVC(probability=True)),
                                                                 ('knn',
                                                                  KNeighborsClassifier(n_jobs=-1)),
                                                                 ('extra',
                                                                  ExtraTreesClassifier(n_jobs=-1)),
                                                                 ('mlp',
                                                                  MLPClassifier()),
                                                                 ('xg',
                                                                  XGBClassifier(base_score=None,
                                                                                booster=None,
                                                                                callbacks=None,
                                                                                colsample_bylevel=None,
                                                                                colsample_bynode=None,
                                                                                colsample_bytree=...
                                        'knn__n_neighbors': [2, 3, 4, 5, 6],
                                        'knn__weights': ['uniform', 'distance'],
                                        'lr__penalty': ['l1', 'l2',
                                                        'elasticnet'],
                                        'lr__solver': ['lbfgs', 'liblinear'],
                                        'mlp__learning_rate': ['constant',
                                                               'invscaling',
                                                               'adaptive'],
                                        'mlp__solver': ['lbfgs', 'sgd', 'adam'],
                                        'svc__gamma': ['scale', 'auto'],
                                        'xg__depth': [2, 4, 6],
                                        'xg__max_leaves': [4, 6, 8],
                                        'xg__n_estimators': [2, 4, 6, 8]},
                   scoring='f1_weighted')>
```

In [222]:
```
estimators = [('lr', LogisticRegression(penalty='l1', solver='liblinear')),
              ('svc', SVC(probability=True, gamma='auto')), ('knn', KNeighborsClassifier(n_neighbors=5, weights='uniform',n_
             ('extra', ExtraTreesClassifier(n_estimators=200, min_samples_split=6, max_features='log2', max_depth=None, n_jo
              ('mlp', MLPClassifier(solver='lbfgs', learning_rate='adaptive')), ('xg', Xgb.XGBClassifier(n_estimators=6, max_
ideal = StackingClassifier(estimators=estimators, final_estimator=RandomForestClassifier(n_estimators=40, max_samples=4, max_
```

In [225]:
```
ideal.fit(X_train, y_train)
ideal_score = ideal.score(X_train, y_train)
ideal_cross_val_accuracy = cross_val_score(ideal, X_train, y_train, cv=5, scoring='accuracy', n_jobs=-1)
ideal_cross_val_predict = cross_val_predict(ideal, X_train, y_train, cv=5, n_jobs=-1)
ideal_decision_score = cross_val_predict(ideal, X_train, y_train, cv=5, n_jobs=-1, method='predict_proba')
```

In [226]:
```
ideal_score
```

Out[226]:
```
0.9898550724637681
```

In [227]:
```
ideal_cross_val_accuracy
```

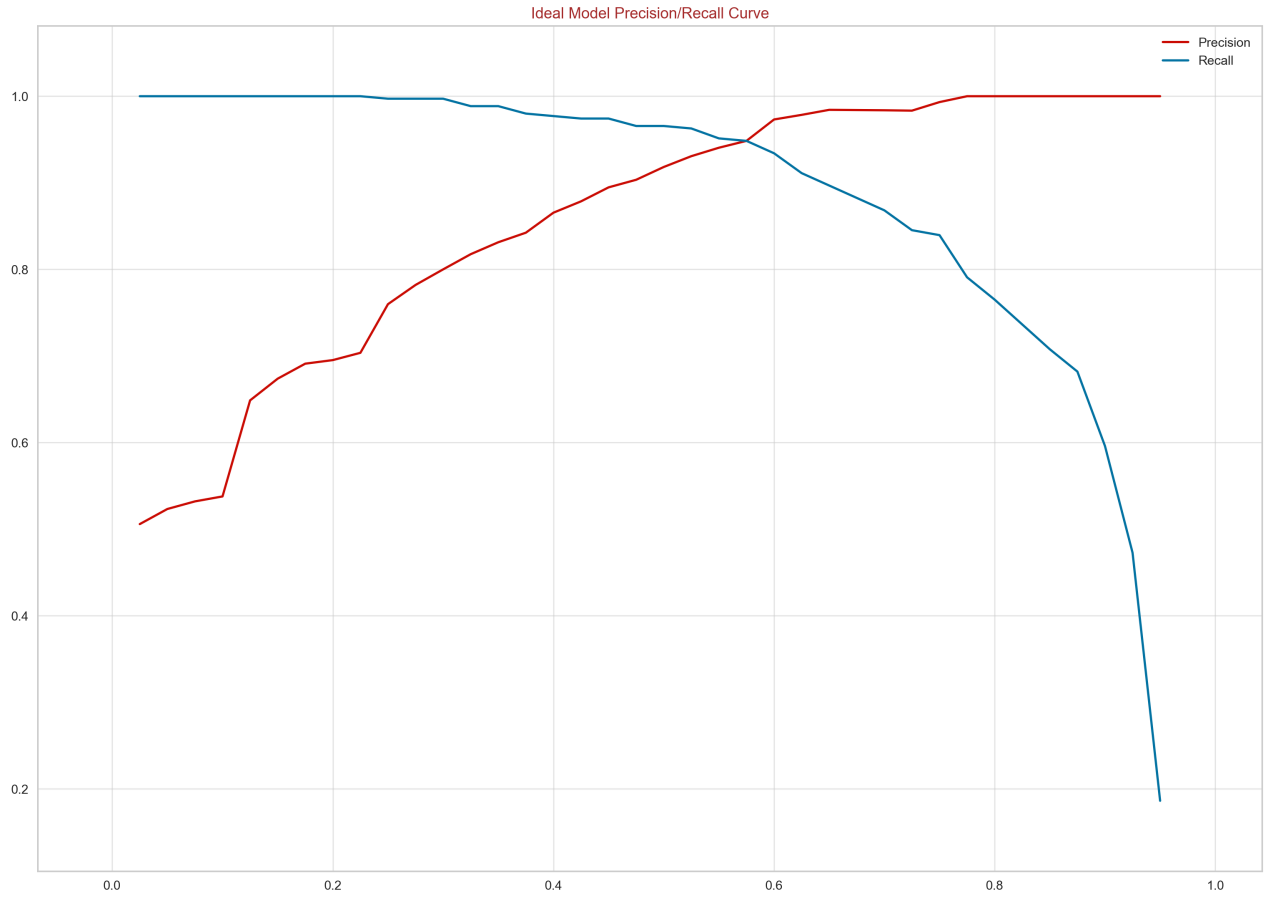
Out[227]:
```
array([0.94202899, 0.9057971 , 0.98550725, 0.91304348, 0.98550725])
```

In [233]:
```
confusion_matrix(y_train, ideal_cross_val_predict)
```

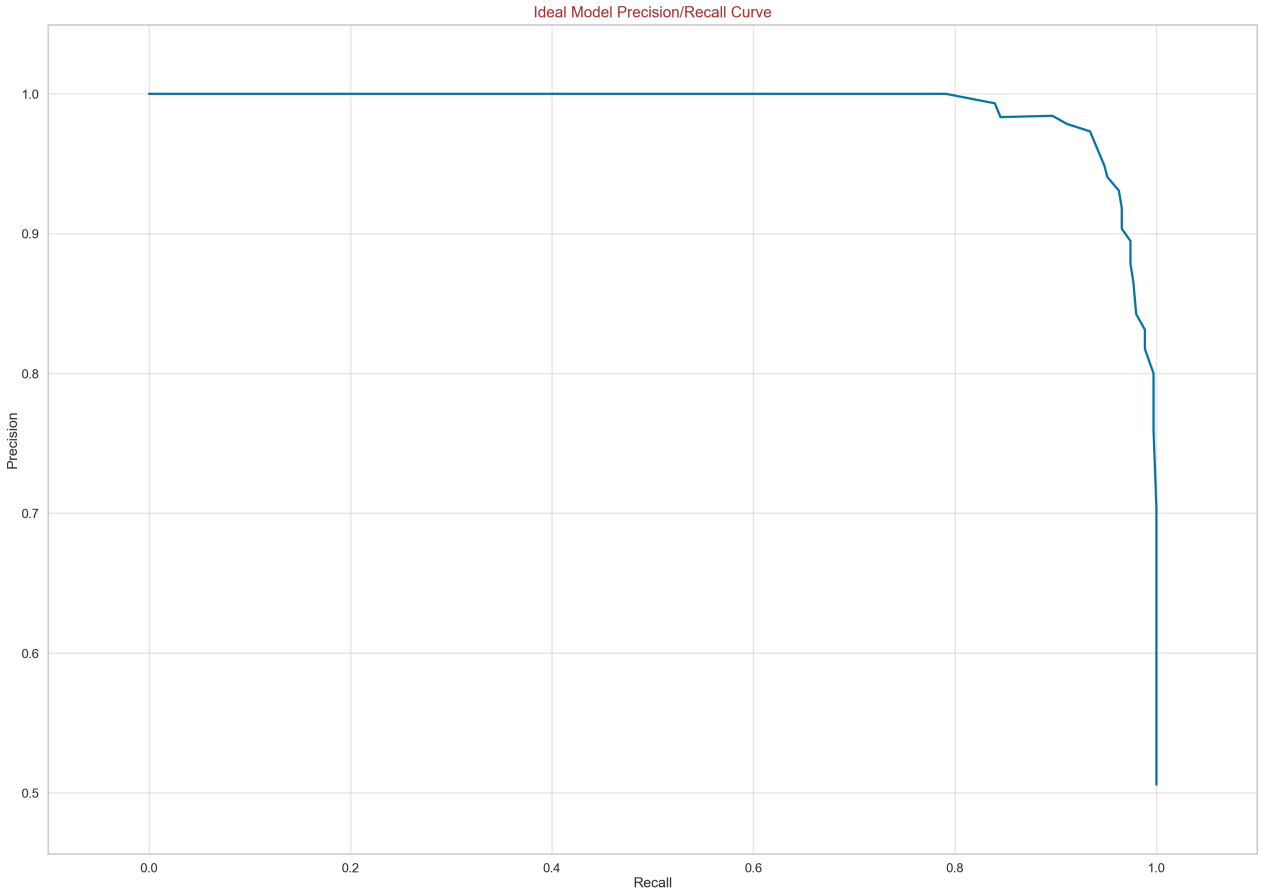
Out[233]:
```
array([[323,  18],
       [ 20, 329]], dtype=int64)
```

In [228]:
```
precision_, recall_, thresholds_ = precision_recall_curve(y_train, ideal_decision_score[:, 1])
fpr, tpr, threshold = roc_curve(y_train, ideal_decision_score[:, 1])
```

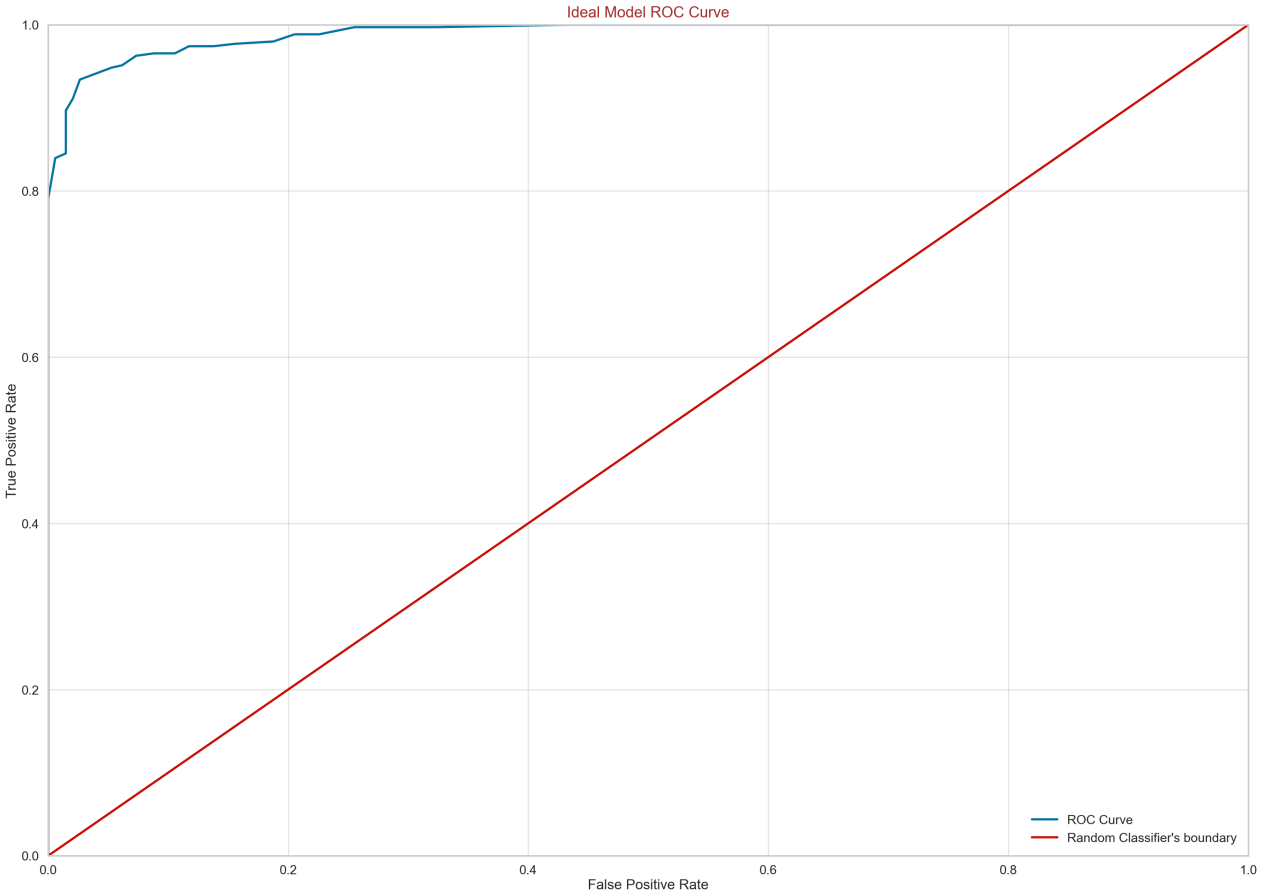
In [229]:
```
p_r_t_curve(precision_, recall_, thresholds_, 'Ideal Model')
```



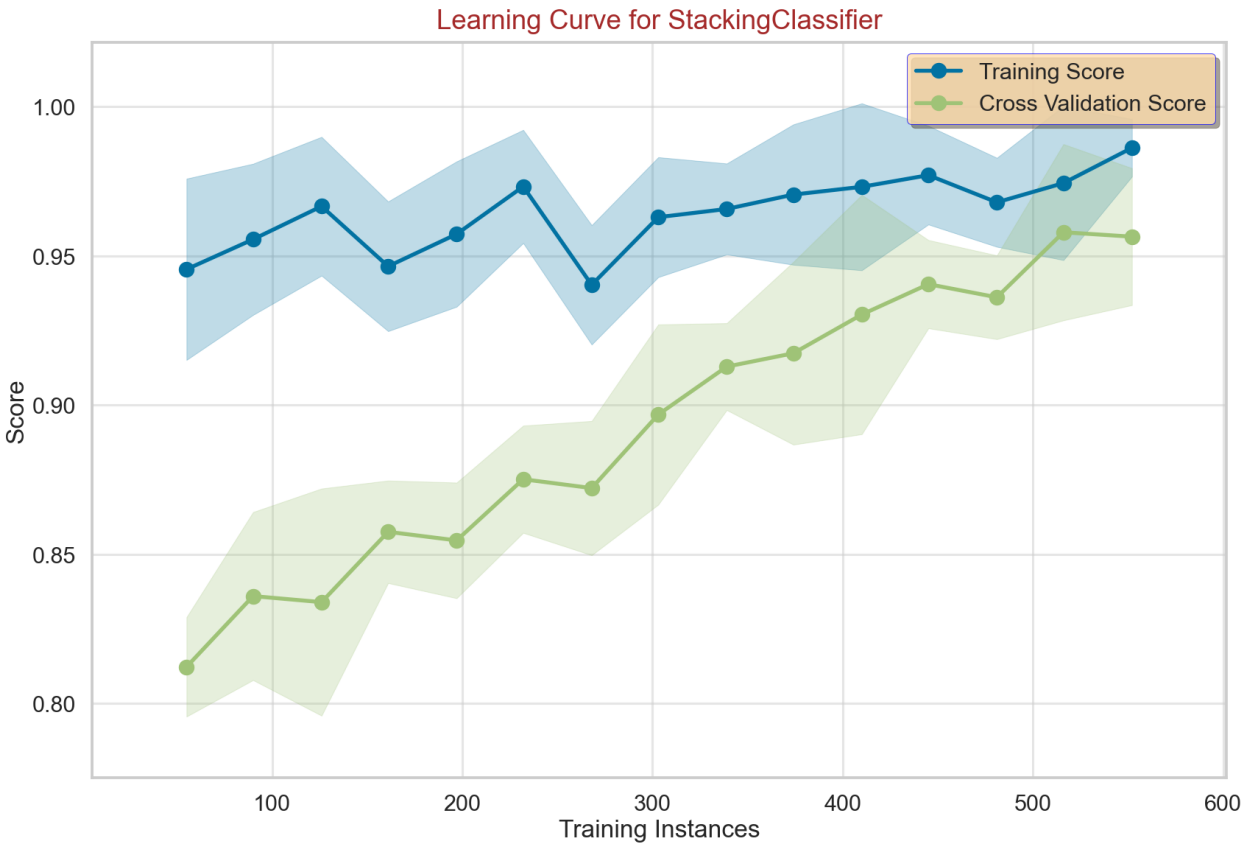
In [230]:
```
rc(precision_, recall_, "Ideal Model")
```



In [231]:
```
roc_curve_plot(fpr, tpr, threshold, 'Ideal Model')
```



In [232]:
```
ideal_curve = LearningCurve(estimator=ideal, cv=5, train_sizes=np.linspace(0.1, 1, 15), n_jobs=-1, scoring='f1_weighted')
ideal_curve.fit(X_train, y_train)
ideal_curve.show()
```



Out[232]:
```
<Axes: title={'center': 'Learning Curve for StackingClassifier'}, xlabel='Training Instances', ylabel='Score'>
```

The learning rate clearly depicts that this model is not overfit or underfit, the validation score improves as the new traing data feeds to the model

Lets test this model on the test data

In [234]:
```
y_preds = ideal.predict(X_test)
```

In [235]:
```
confusion_matrix(y_test, y_preds)
```

Out[235]:
```
array([[73,  0],
       [ 8, 92]], dtype=int64)
```

In [237]:
```
precision_score(y_test, y_preds)
```

Out[237]:
```
1.0
```

In [238]:
```
recall_score(y_test, y_preds)
```

Out[238]:
```
0.92
```

Our objective is to identify the persons who could have heart diease correctly so it is good to have high precision. But our model has 1.0 precision and 0.92 recall which is a good score.

In [239]:
```
print(y_preds)
```

```
[1 0 0 0 1 1 1 0 1 1 0 1 0 0 0 1 0 1 1 1 0 0 1 0 0 0 1 1 1 0 0 1 0 0 0 0 0
 1 0 1 1 1 0 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 0 0 1 0 0 0 1 1 1 0 1 1 0 1 0 1
 1 0 1 1 0 1 0 0 1 0 1 1 0 0 1 1 0 1 0 1 0 0 0 0 0 1 0 0 0 1 1 0 1 0 1 1 1 0 1
 1 0 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 0 1 0 0 0 0 1 1 1 0 0 0 0 1 0 1 0 1 1 1
 1 1 1 1 1 1 1 1 1 1 0 1 1 0 0 0 1 0 0 1 0 0 1 0 0]
```

In [240]:
```
actual_vs_predicted = pd.DataFrame({'Actual Values':y_test, 'Predicted Values' : y_preds})
```

In [247]:
```
actual_vs_predicted.sample(25).style.background_gradient(cmap='hot').hide()
```

Out[247]:

Actual Values	Predicted Values
1	1
1	1
0	0
1	1
0	0
1	1
1	1
0	0
0	0
0	0
0	0
1	1
1	1
1	1
1	1
1	1
0	0
1	1
1	0
1	1
1	1
0	0
1	1
1	1
0	0

In [ ]:
```

```