Visualization

Programming for Data Science (ID2214)

Amir H. Rahnama (Twitter: @_ambodi)

Ph.D. Candidate

Key *takeaways* from this lecture:

- Understanding key visualizations in a Machine Learning project
- Getting started and becoming comfortable working with a dataset
- Learn basics of Matplotlib, Scikit-Learn library
- Learn about the problem of imbalanced classes in Machine Learning

Loading the Wisconsin Breast Cancer dataset

Let us load the dataset and look at the

- features
- shape of the data
- classes
- class labels

In [2]:
```
from sklearn.datasets import load_breast_cancer

d = load_breast_cancer()

d.feature_names
```

Out[2]:
```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension'], dtype='|S23')
```

In [3]:
```
(d.data.shape, len(d.target))
```

Out[3]:
```
((569, 30), 569)
```

In [5]:
```
d.target_names
```

Out[5]:
```
array(['malignant', 'benign'], dtype='|S9')
```

In [6]:
```
import pandas as pd

data = pd.DataFrame(data= d.data, columns=d.feature_names)

malignant_filter = d.target == 0
benign_filter = d.target == 1

malignant = data[malignant_filter]
benign = data[benign_filter]
```

In [7]:
```
data.head()
```

Out[7]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980

5 rows × 30 columns

Visualization with Matplotlib

Our main focus while discussing visualization will be the Matplotlib library, however we will show examples of Plotly and Seaborn library.

Rendering a Matplotlib visualization is a 4-step process:

- Import the pyplot module from matplotlib (I am using the styles imported from ggplot2, a data visualization package for the statistical programming language R):

In [8]:
```
import matplotlib.pyplot as plt
plt.style.use('ggplot')
```

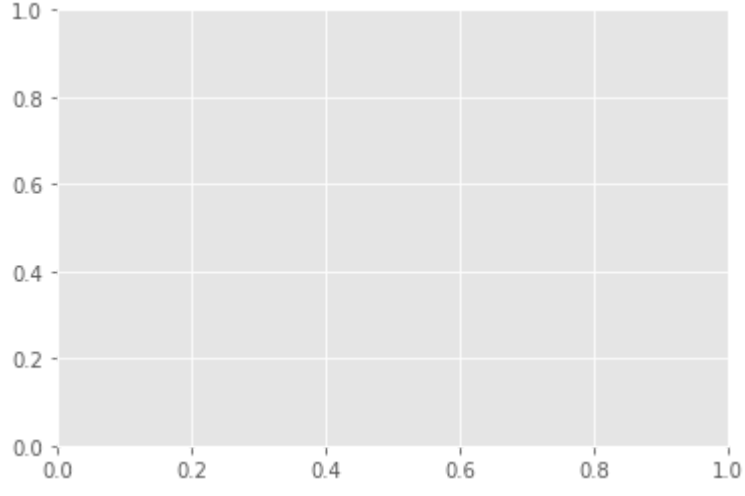
- Create a figure object from the pyplot:

In [9]:
```
fig = plt.figure()
```

```
<Figure size 432x288 with 0 Axes>
```

- Create axes from the pyplot

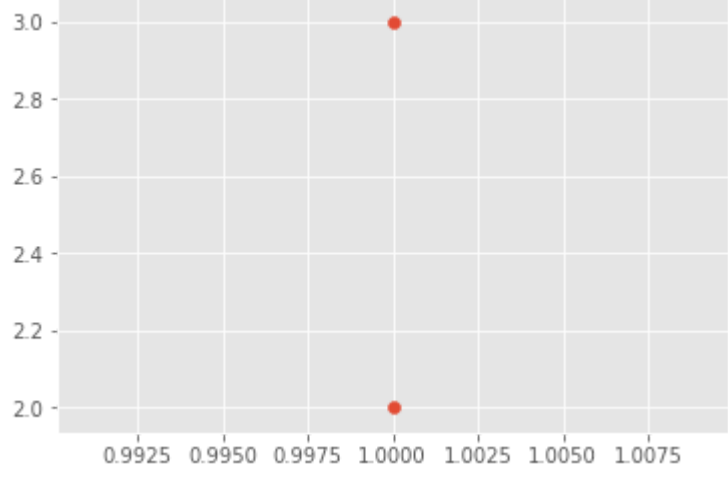
In [10]:
```
ax = plt.axes()
```



- call the type of plot you are using on the top of the plot object:

In [11]:
```
plt.scatter([1,1],[2,3])
```

Out[11]:
```
<matplotlib.collections.PathCollection at 0x11f008910>
```



Exploratory analysis

Visualization plays an important part in exploratory analysis. When performing exploratory analysis, we are interested to summarize the data but with the help of our vision.

Histogram

One of many ways that we can look at frequency of an attribute in the data is to visualize its histogram. In the histogram, we need to set a parameter called "bins" that basically divides the data into intervals and counts the frequency in each bin. Let us look at histogram of the attribute "mean radius" across both classes. Another parameter alpha sets transpareny level of the rendering.

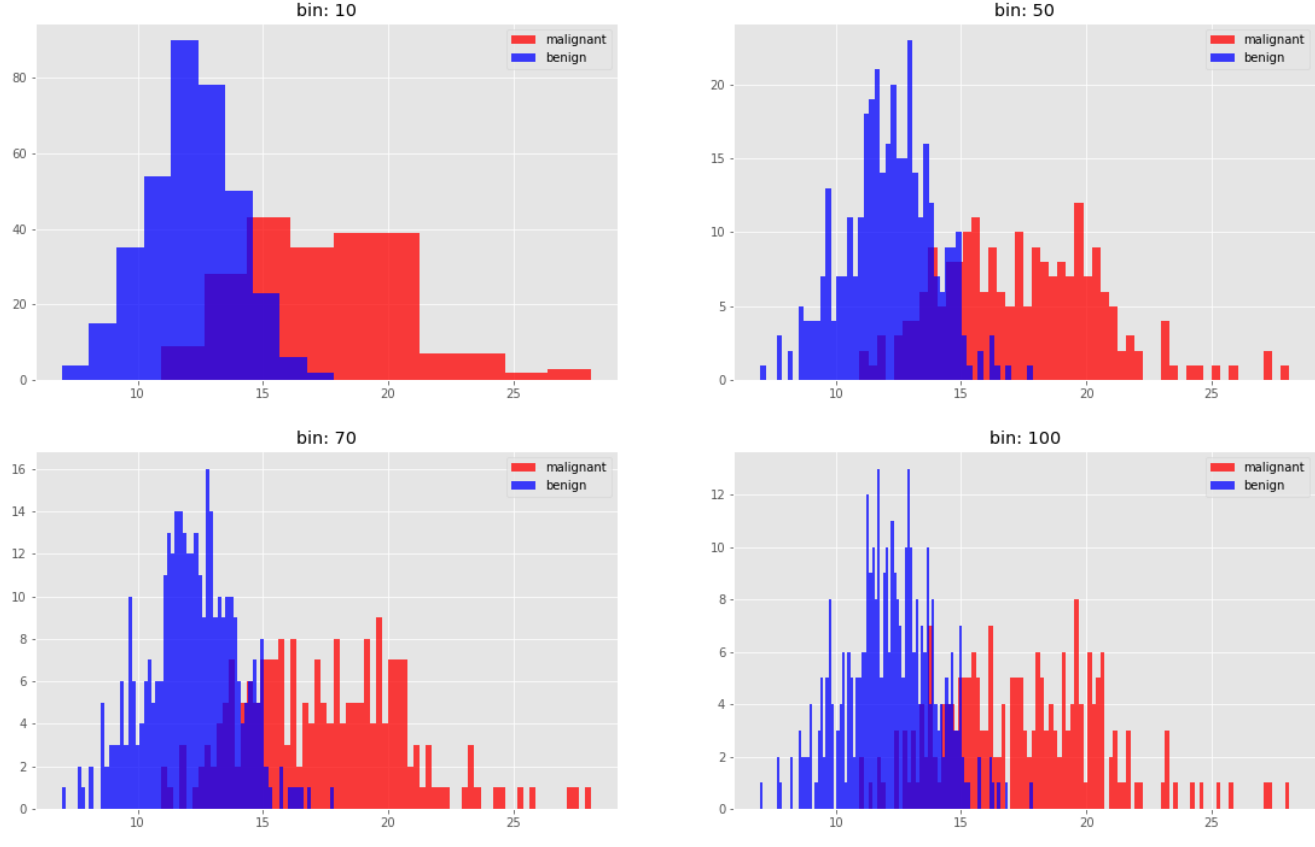
Let us have a look at some bin values:

In [12]:

```
import numpy as np
import matplotlib.pyplot as plt

fig, ax = plt.subplots(2,2, figsize=(19, 12))

ax[0, 0].hist(malignant['mean radius'], bins=10, alpha=0.75, facecolor='red', label="malignant")
ax[0, 0].hist(benign['mean radius'], bins=10, alpha=0.75, facecolor='blue', label="benign")
ax[0, 0].legend()
ax[0, 0].set_title('bin: 10')

ax[0, 1].hist(malignant['mean radius'], bins=50, alpha=0.75, facecolor='red', label="malignant")
ax[0, 1].hist(benign['mean radius'], bins=50, alpha=0.75, facecolor='blue', label="benign")
ax[0, 1].legend()
ax[0, 1].set_title('bin: 50')

ax[1, 0].hist(malignant['mean radius'], bins=70, alpha=0.75, facecolor='red', label="malignant")
ax[1, 0].hist(benign['mean radius'], bins=70, alpha=0.75, facecolor='blue', label="benign")
ax[1, 0].legend()
ax[1, 0].set_title('bin: 70')

ax[1, 1].hist(malignant['mean radius'], bins=100, alpha=0.75, facecolor='red', label="malignant")
ax[1, 1].hist(benign['mean radius'], bins=100, alpha=0.75, facecolor='blue', label="benign")
ax[1, 1].legend()
ax[1, 1].set_title('bin: 100')

fig.suptitle('Histogram: Mean Radius')
```

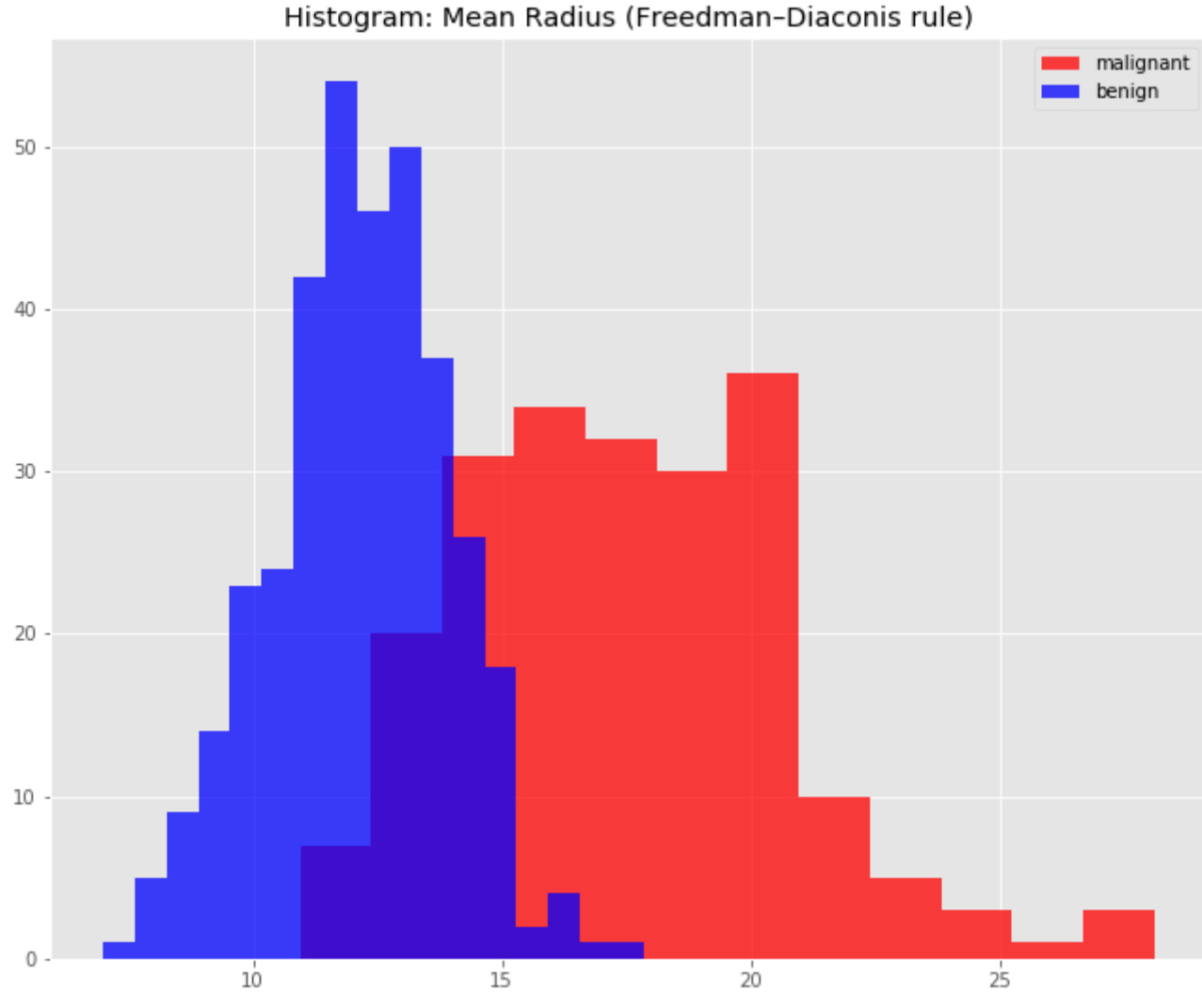
```
plt.savefig('./hist_original.png')
plt.close()
```

Histogram: Mean Radius

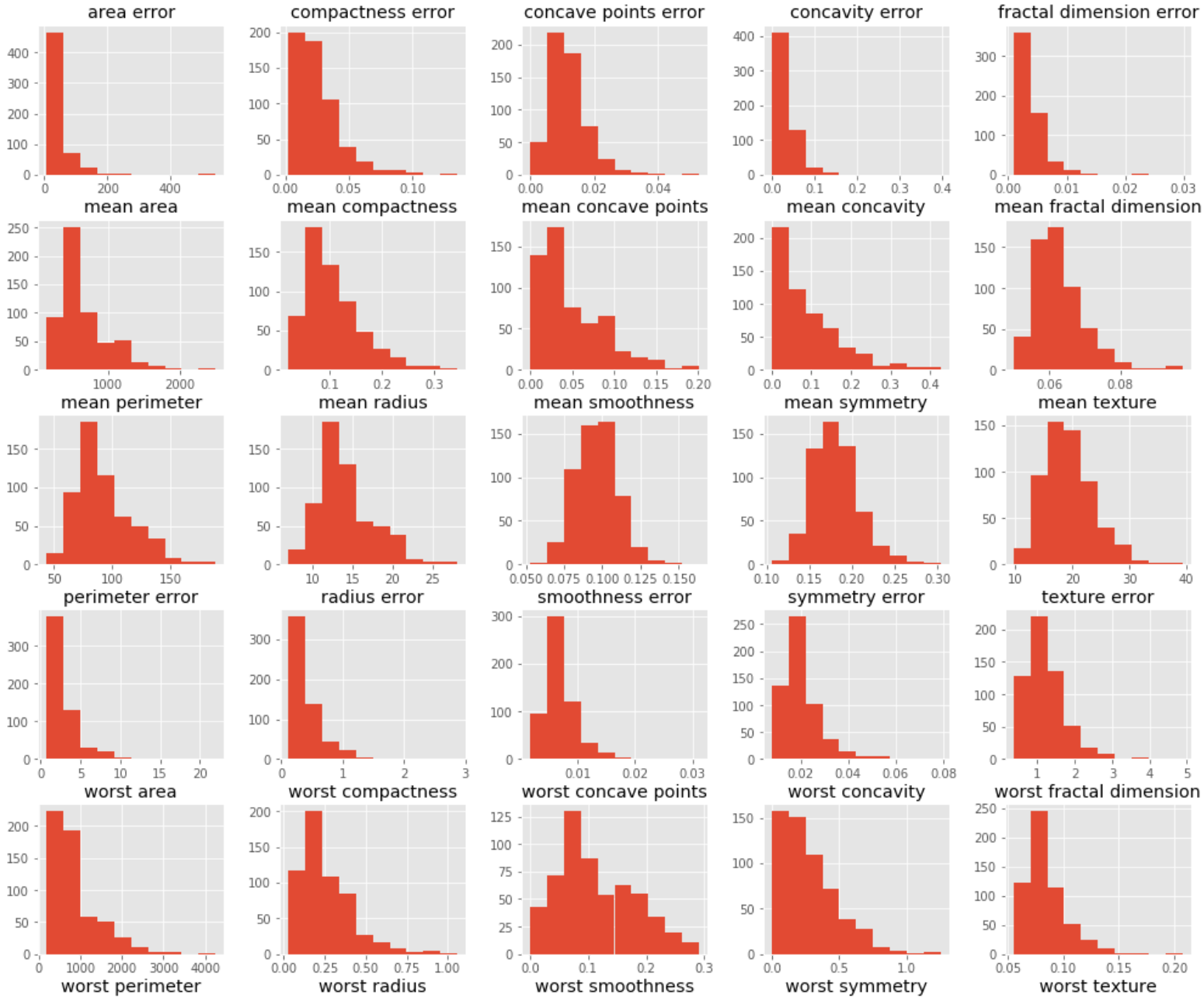


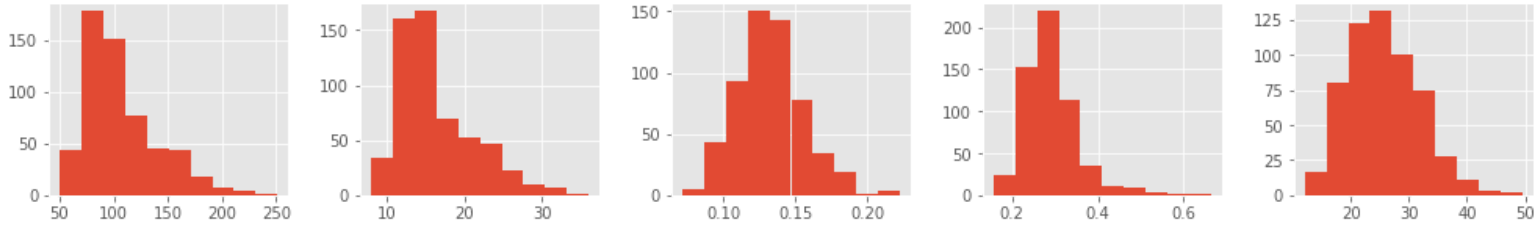
In [13]:

```
import numpy as np
import matplotlib.pyplot as plt

plt.ioff()
fig, ax = plt.subplots(1, 1, figsize=(11, 9))

ax.hist(malignant['mean radius'], bins='fd', alpha=0.75, facecolor='red', label="m
alignant")
ax.hist(benign['mean radius'], bins='fd', alpha=0.75, facecolor='blue', label="ben
ign")
ax.legend()
ax.set_title(u'Histogram: Mean Radius (Freedman–Diaconis rule)')

plt.savefig('./hist_fd.png')
plt.close()
```


Histogram: Mean Radius (Freedman–Diaconis rule)
malignant
benign
0
10
20
30
40
50
10
15
20
25

In [14]:
```
histog = data.hist(bins=10, figsize=(17,17))
```

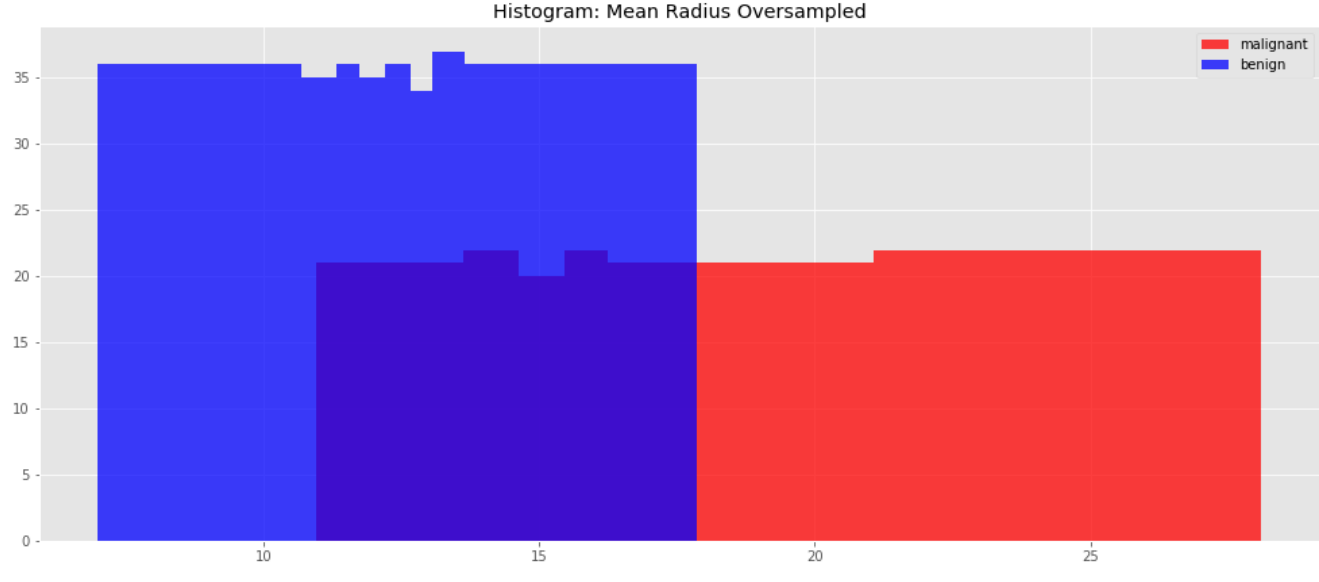



150
100
50
0
50 100 150 200 250
150
100
50
0
10 20 30
150
100
50
0
0.10 0.15 0.20
200
150
100
50
0
0.2 0.4 0.6
125
100
75
50
25
0
20 30 40 50

Histogram with Equal Size

In [15]:
```
### deciles

ser_mal, mal_bins = pd.qcut(malignant['mean radius'], 10, retbins=True, labels=Fal
se)
ser_ben, ben_bins = pd.qcut(benign['mean radius'], 10, retbins=True, labels=False)
```

In [16]:
```
import numpy as np
import matplotlib.pyplot as plt

plt.ioff()
fig, ax = plt.subplots(1, 1, figsize=(17, 7))

ax.hist(malignant['mean radius'], bins=mal_bins, alpha=0.75, facecolor='red', labe
l="malignant")
ax.hist(benign['mean radius'], bins=ben_bins, alpha=0.75, facecolor='blue', label=
"benign")
ax.legend()
ax.set_title('Histogram: Mean Radius Oversampled')

plt.savefig('./hist_mean_radius_qcut.png')
plt.close()
```


Histogram: Mean Radius Oversampled
malignant
benign
35
30
25
20
15
10
5
0
10
15
20
25

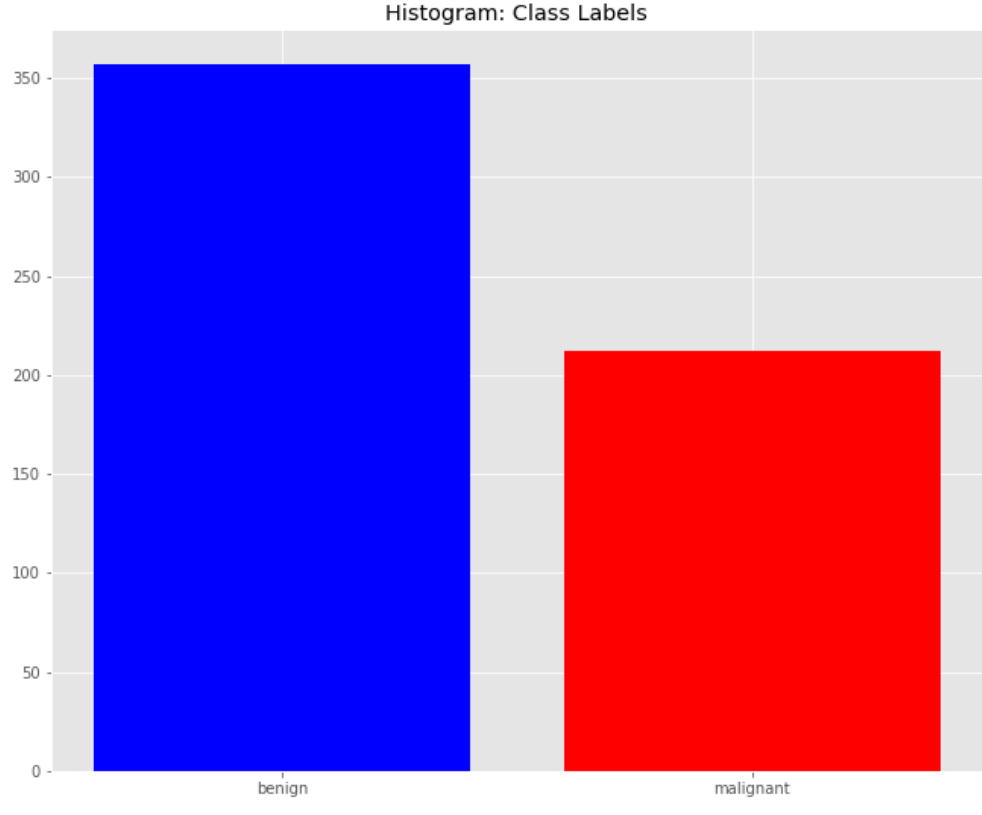
Bar chart

Bar chart are one of the most popular visualization methods. They are used for all types of attributes, categorical or numerical. One of the ways you can visualize a categorical variable is to simply count how many of each there is and show that count.

A histogram lets you do exactly this. In our dataset, we can treat the diagnosis label as a categorical variable:

In [17]:

```
import matplotlib.pyplot as plt

type_data = {'malignant': len(malignant), 'benign': len(benign)}

plt.ioff()
fig = plt.figure(figsize=(11, 9))

plt.bar(type_data.keys(), type_data.values(), color=['blue', 'red'])
plt.title('Histogram: Class Labels')

plt.savefig('./bar_classes.png')
plt.close()
```


Histogram: Class Labels
350
300
250
200
150
100
50
0
benign
malignant

When confronted with class-imbalance problem, there are two approaches researchers and practitioners take:

- Over-sampling: increase the number of instances from the class with lower number of instances
- Under-sampling: decrease the number of instances from the class with higher number of instances
- AUC: consider a performance metric that is less affected by class imbalance with AUC

Oversampling

We will be using *imblearn* package for both cases. We will install the package as it is not included in Anaconda by default:

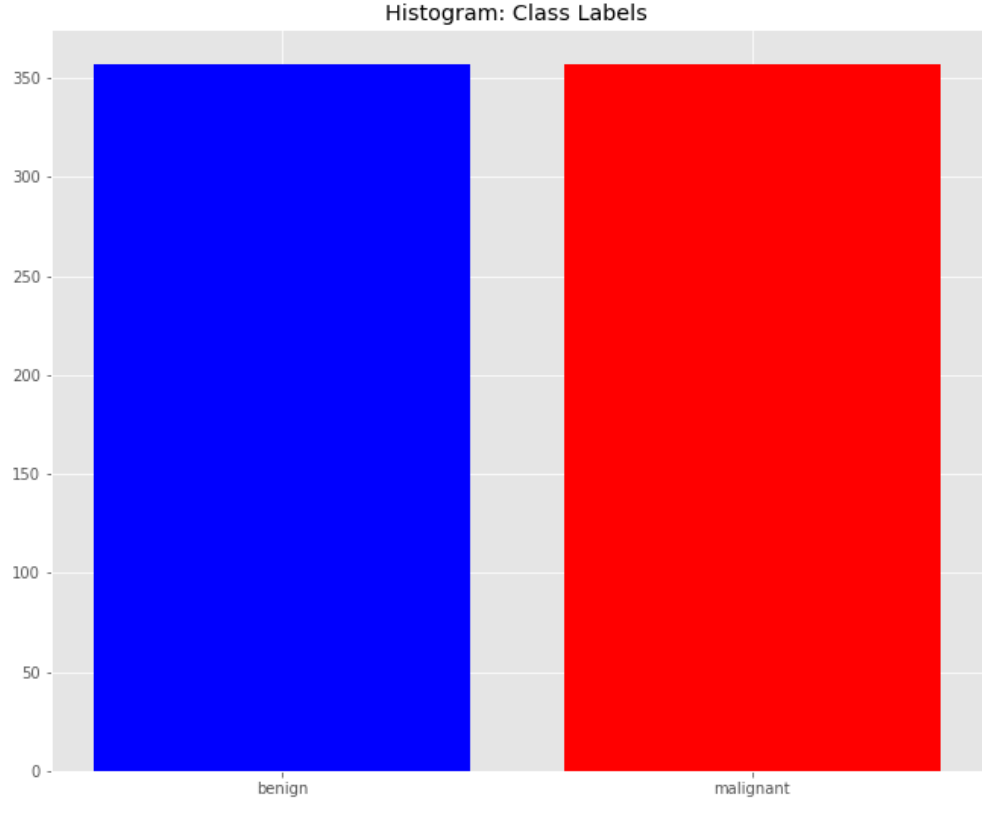
In [18]:
```
!pip install imblearn
```

```
Requirement already satisfied: imblearn in /Users/amirrahnama/anaconda3/envs/t
f/lib/python2.7/site-packages (0.0)
Requirement already satisfied: imbalanced-learn in /Users/amirrahnama/anaconda
3/envs/tf/lib/python2.7/site-packages (from imblearn) (0.4.3)
Requirement already satisfied: numpy>=1.8.2 in /Users/amirrahnama/anaconda3/en
vs/tf/lib/python2.7/site-packages (from imbalanced-learn->imblearn) (1.15.4)
Requirement already satisfied: scipy>=0.13.3 in /Users/amirrahnama/anaconda3/e
nvs/tf/lib/python2.7/site-packages (from imbalanced-learn->imblearn) (1.1.0)
Requirement already satisfied: scikit-learn>=0.20 in /Users/amirrahnama/anacon
da3/envs/tf/lib/python2.7/site-packages (from imbalanced-learn->imblearn) (0.2
0.0)
You are using pip version 10.0.1, however version 18.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
```

In [19]:
```
from imblearn.over_sampling import RandomOverSampler

X_oversampled, y_oversampled = RandomOverSampler().fit_resample(X, y)
```

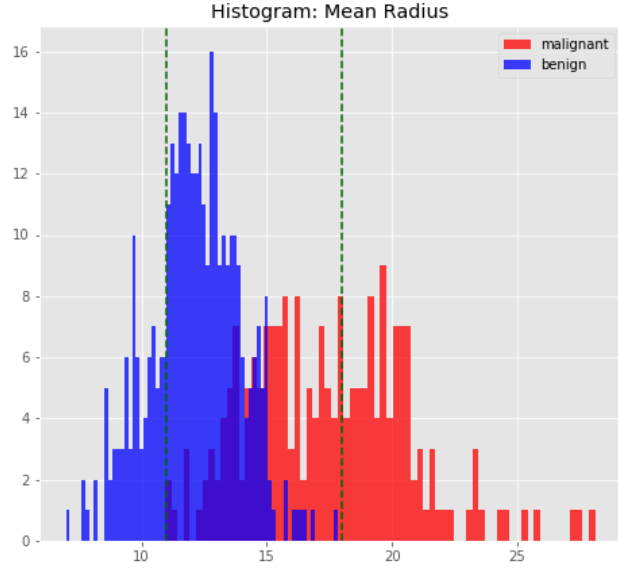
In [20]:
```
data_resample = pd.DataFrame(data= X_oversampled, columns=d.feature_names)

malignant_filter_oversampled = y_oversampled == 0
benign_filter_oversampled = y_oversampled == 1

malignant_oversampled = data_resample[malignant_filter_oversampled]
benign_oversampled = data_resample[benign_filter_oversampled]
```

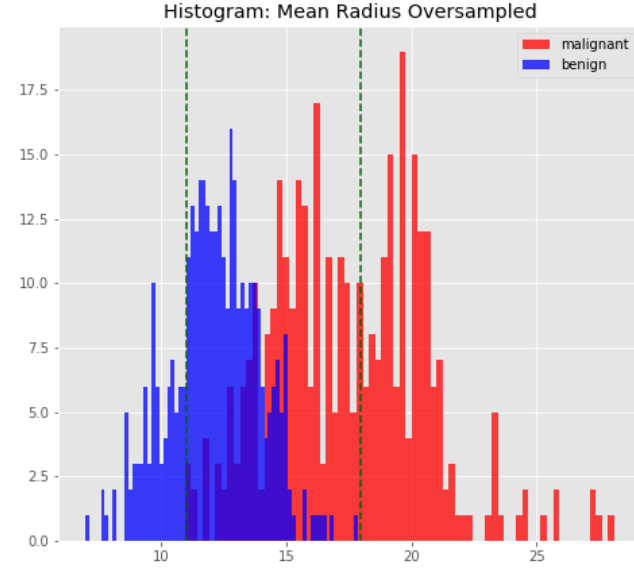
In [21]:
```
import matplotlib.pyplot as plt

type_data = {'malignant': len(malignant_oversampled), 'benign': len(benign_oversam
pled)}


plt.ioff()
fig = plt.figure(figsize=(11, 9))

plt.bar(type_data.keys(), type_data.values(), color=['blue', 'red'])
plt.title('Histogram: Class Labels')

plt.savefig('./bar_classes_oversampled.png')
plt.close()
```


Histogram: Class Labels
350
300
250
200
150
100
50
0
benign
malignant

In [22]:

```
import numpy as np
import matplotlib.pyplot as plt

plt.ioff()
fig, ax = plt.subplots(1,2, figsize=(17, 7))

ax[0].hist(malignant['mean radius'], bins=70, alpha=0.75, facecolor='red', label=
"malignant")
ax[0].hist(benign['mean radius'], bins=70, alpha=0.75, facecolor='blue', label="be
nign")
ax[0].axvline(x=11, color="darkgreen", linestyle='--')
ax[0].axvline(x=18, color="darkgreen", linestyle='--')
ax[0].legend()
ax[0].set_title('Histogram: Mean Radius')

ax[1].hist(malignant_oversampled['mean radius'], bins=70, alpha=0.75, facecolor='r
ed', label="malignant")
ax[1].hist(benign_oversampled['mean radius'], bins=70, alpha=0.75, facecolor='blu
e', label="benign")
ax[1].axvline(x=11, color="darkgreen", linestyle='--')
ax[1].axvline(x=18, color="darkgreen", linestyle='--')
ax[1].legend()
ax[1].set_title('Histogram: Mean Radius Oversampled')

plt.savefig('./hist_radius_oversampled.png')
plt.close()
```

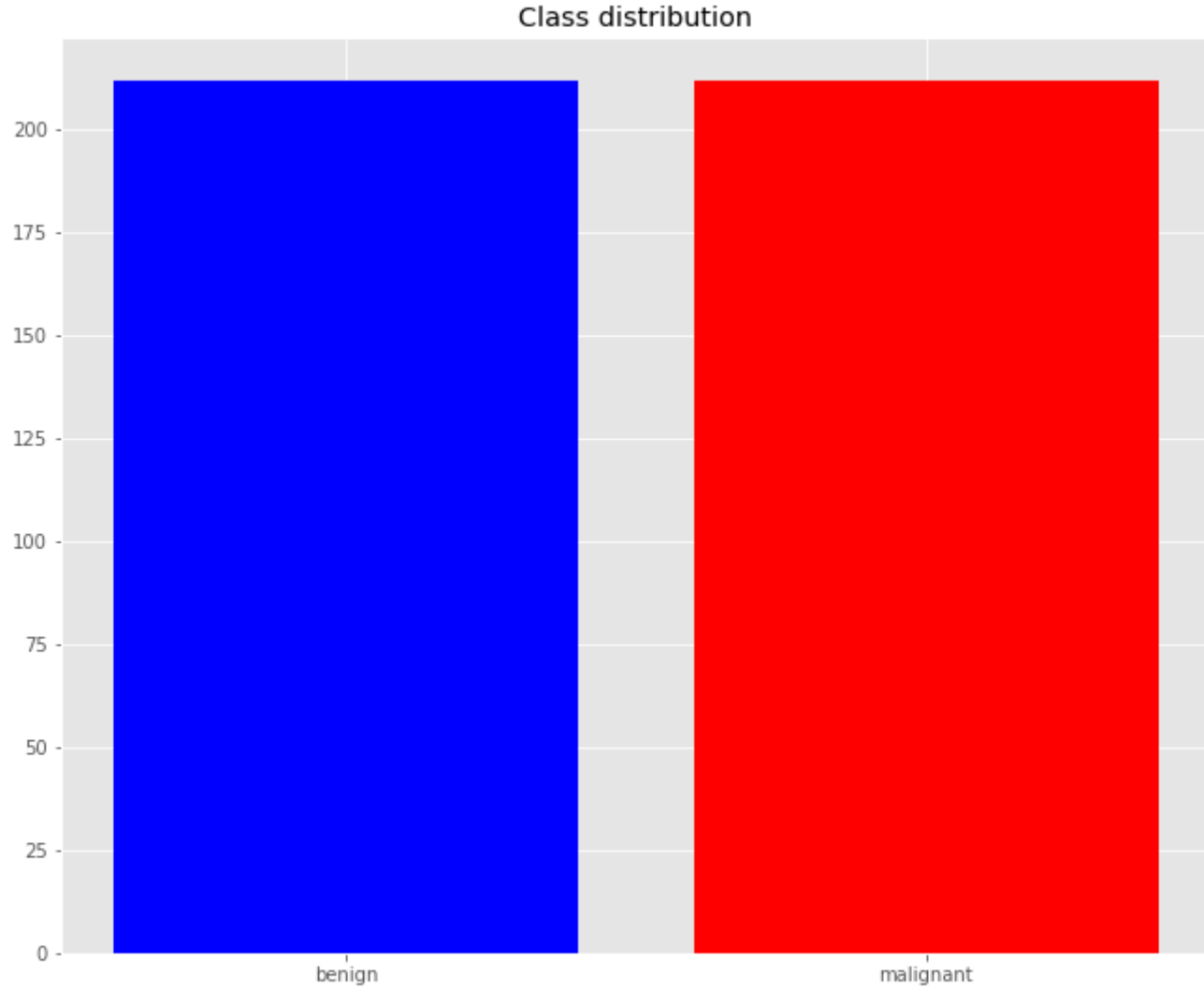

Histogram: Mean Radius
malignant
benign
0
2
4
6
8
10
12
14
16
10
15
20
25


Histogram: Mean Radius Oversampled
malignant
benign
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
10
15
20
25

Undersampling

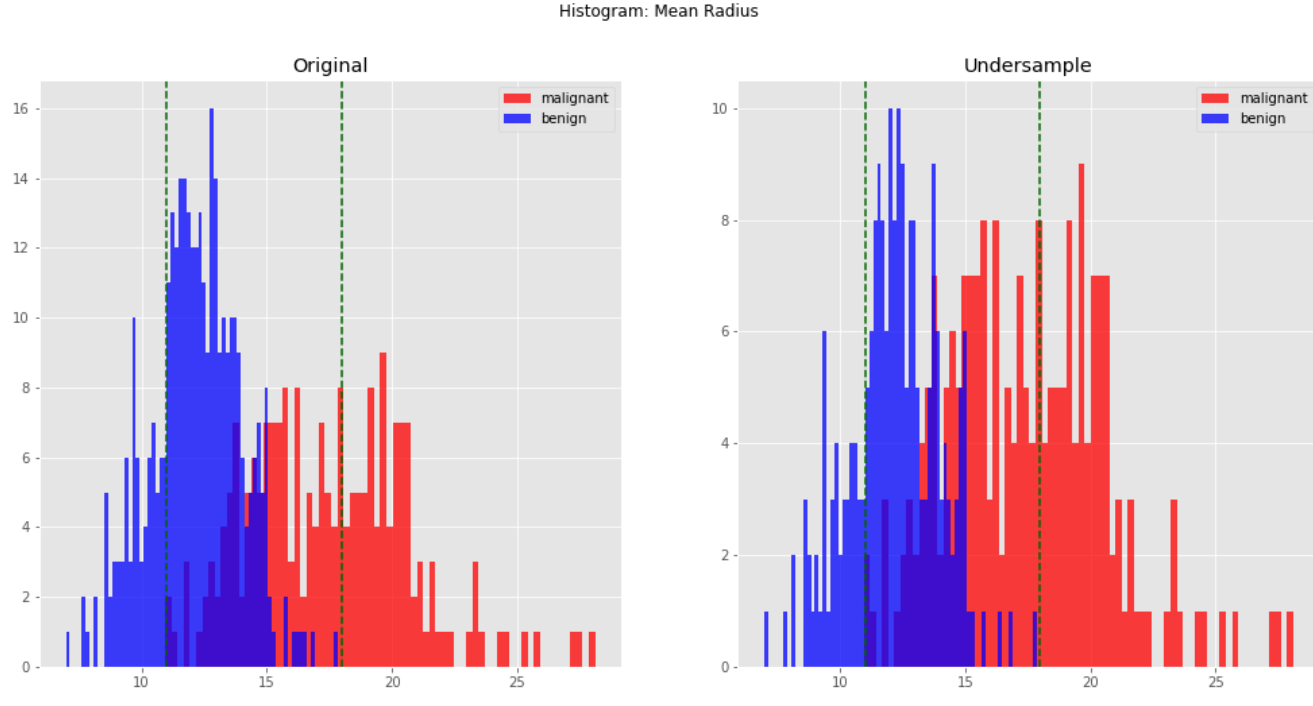
In [23]:
```
from imblearn.under_sampling import RandomUnderSampler

rus = RandomUnderSampler(random_state=0)
X_undersampled, y_undersampled = rus.fit_resample(X, y)
```

In [25]:
```
import matplotlib.pyplot as plt

type_data = {'malignant': len(malignant_undersampled), 'benign': len(benign_unders
ampled)}

plt.ioff()

plt.figure(figsize=(11,9))
plt.bar(type_data.keys(), type_data.values(), color=['blue', 'red'])
plt.title('Class distribution')

plt.savefig('./bar_classes_oversample.png')
plt.close()
```


Class distribution
200
175
150
125
100
75
50
25
0
benign
malignant

In [26]:

```
import numpy as np
import matplotlib.pyplot as plt

plt.ioff()

fig, ax = plt.subplots(1,2, figsize=(17, 8))

# Original data
ax[0].hist(malignant['mean radius'], 70, alpha=0.75, facecolor='red', label="malig
nant")
ax[0].hist(benign['mean radius'], 70, alpha=0.75, facecolor='blue', label="benign"
)

ax[0].axvline(x=11, color="darkgreen", linestyle='--')
ax[0].axvline(x=18, color="darkgreen", linestyle='--')

ax[0].legend()
ax[0].set_title('Original')


# Undersampled data
ax[1].hist(malignant_undersampled['mean radius'], 70, alpha=0.75, facecolor='red',
 label="malignant")
ax[1].hist(benign_undersampled['mean radius'], 70, alpha=0.75, facecolor='blue', l
abel="benign")

ax[1].axvline(x=11, color="darkgreen", linestyle='--')
ax[1].axvline(x=18, color="darkgreen", linestyle='--')

ax[1].legend()
ax[1].set_title('Undersample')

fig.suptitle('Histogram: Mean Radius')

plt.savefig('./hist_undersample.png')
plt.close()
```


Histogram: Mean Radius
Original
malignant
benign
0
2
4
6
8
10
12
14
16
10
15
20
25
Undersample
malignant
benign
0
2
4
6
8
10
10
15
20
25

Kolmogorov-Smirnov statistic

The Kolmogorov–Smirnov statistic quantifies a distance between the empirical distribution function of the sample and the cumulative distribution function of the reference distribution, or between the empirical distribution functions of two samples. The null hypothesis is that the distributions are the same:

In [27]:
```
from scipy import stats

stats.ks_2samp(malignant['mean radius'], malignant_undersampled['mean radius'])
```

Out[27]:
```
Ks_2sampResult(statistic=0.0, pvalue=1.0)
```

In [28]:
```
from scipy import stats

stats.ks_2samp(malignant['mean radius'], malignant_oversampled['mean radius'])
```

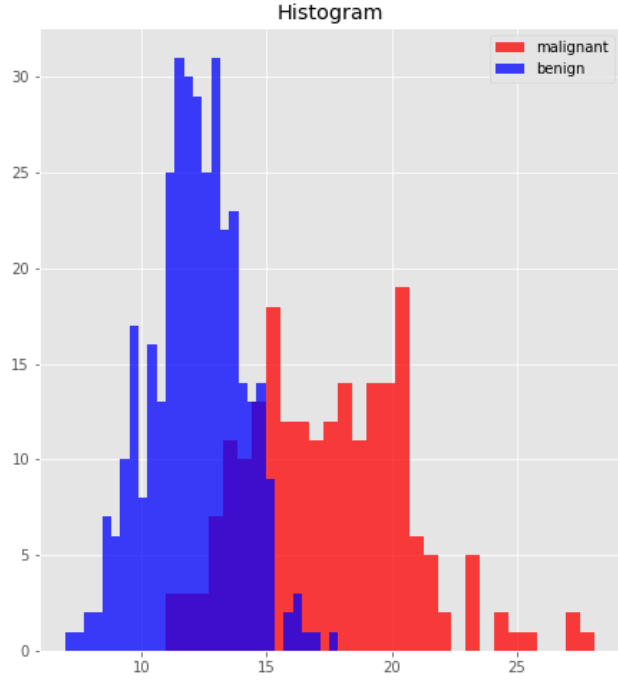
Out[28]:
```
Ks_2sampResult(statistic=0.02075735954759264, pvalue=0.9999999925389874)
```

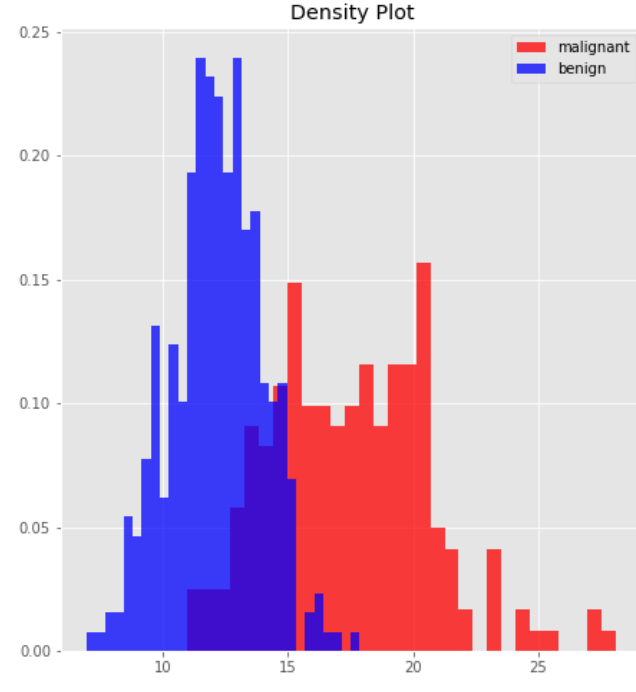
Density Plots

Sometimes instead of using counts, you can visualize the probability density function instead, the key is to pass **density=True** inside the function. Now we get probablity values and not only that, the area under the chart will sum up to 1, just like a Probability Density Function (PDF):

In [29]:

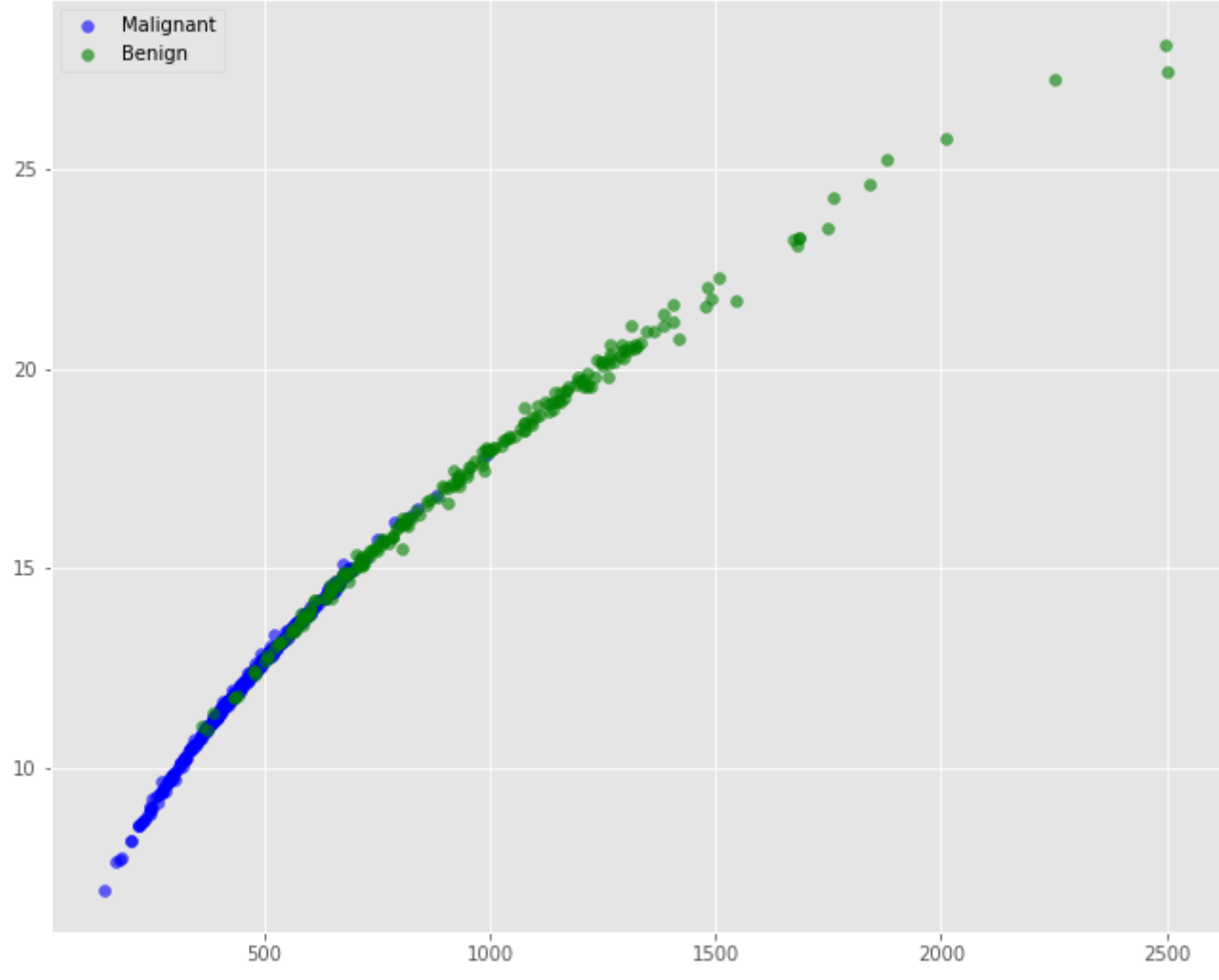
```
import numpy as np
import matplotlib.pyplot as plt

plt.ioff()

fig, ax = plt.subplots(1,2, figsize=(16, 8))

# Histogram
ax[0].hist(malignant['mean radius'], bins=30, alpha=0.75, facecolor='red', label=
"malignant")
ax[0].hist(benign['mean radius'], bins=30, alpha=0.75, facecolor='blue', label="be
nign")
ax[0].legend()
ax[0].set_title('Histogram')

# Density Plot
ax[1].hist(malignant['mean radius'], bins=30, alpha=0.75, density=True, facecolor=
'red', label="malignant")
ax[1].hist(benign['mean radius'], bins=30, alpha=0.75, density=True, facecolor='bl
ue', label="benign")
ax[1].legend()
ax[1].set_title('Density Plot')


plt.savefig('./desnity_plot.png')
plt.close()
```


Histogram
malignant
benign
30
25
20
15
10
5
0
10
15
20
25


Density Plot
malignant
benign
0.25
0.20
0.15
0.10
0.05
0.00
10
15
20
25

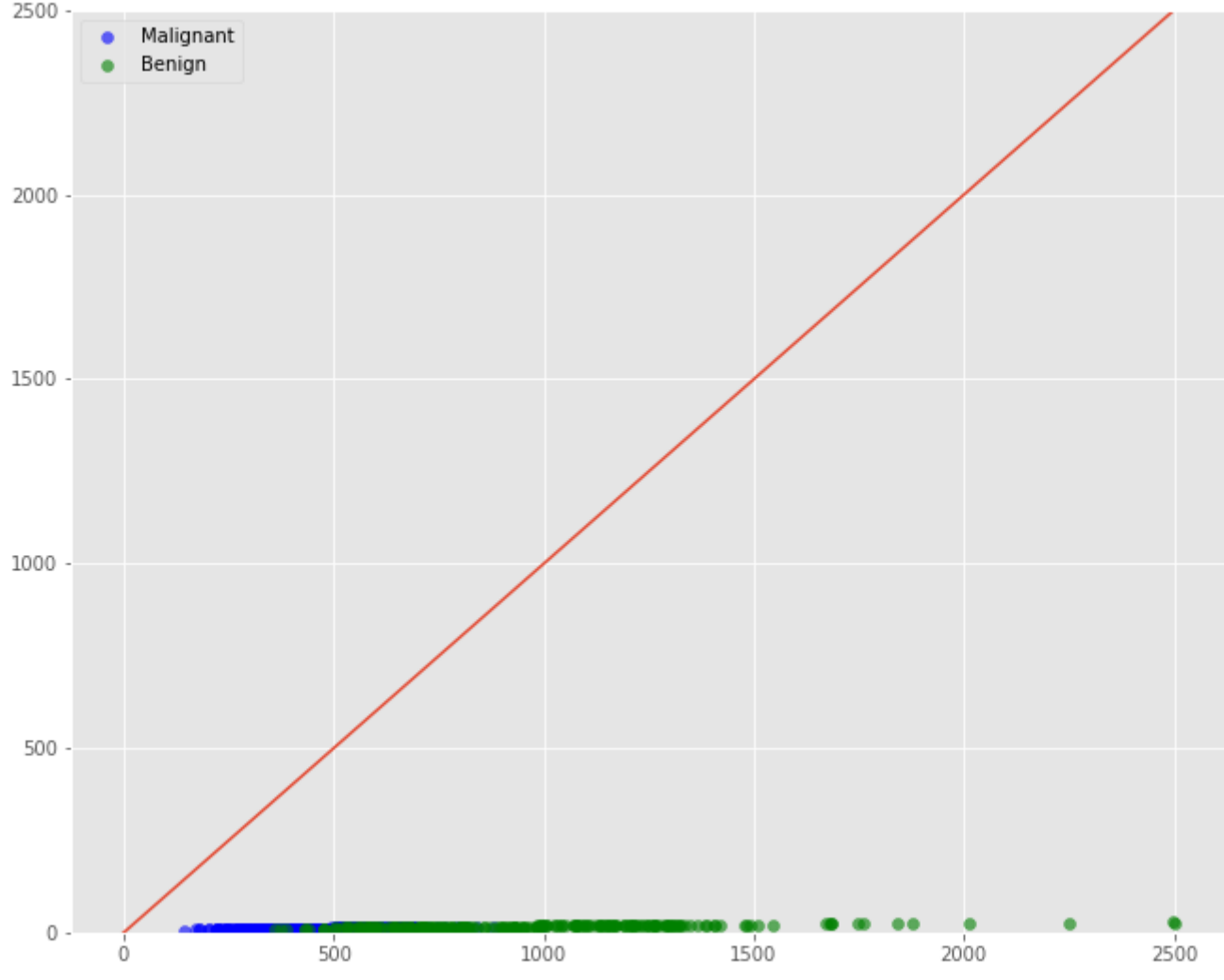
Paired grid plot

In [30]:
```
import numpy as np
import matplotlib.pyplot as plt

plt.ioff()

plt.figure(figsize=(11, 9))

plt.scatter(benign['mean area'], benign['mean radius'], c='blue', label='Malignan
t', alpha=0.6)
plt.scatter(malignant['mean area'], malignant['mean radius'], c='green', label='Be
nign', alpha=0.6)

plt.legend()

plt.savefig('./pair_area_radius.png')
plt.close()

plt.show()
```


Malignant
Benign
25
20
15
10
500
1000
1500
2000
2500

In [31]:

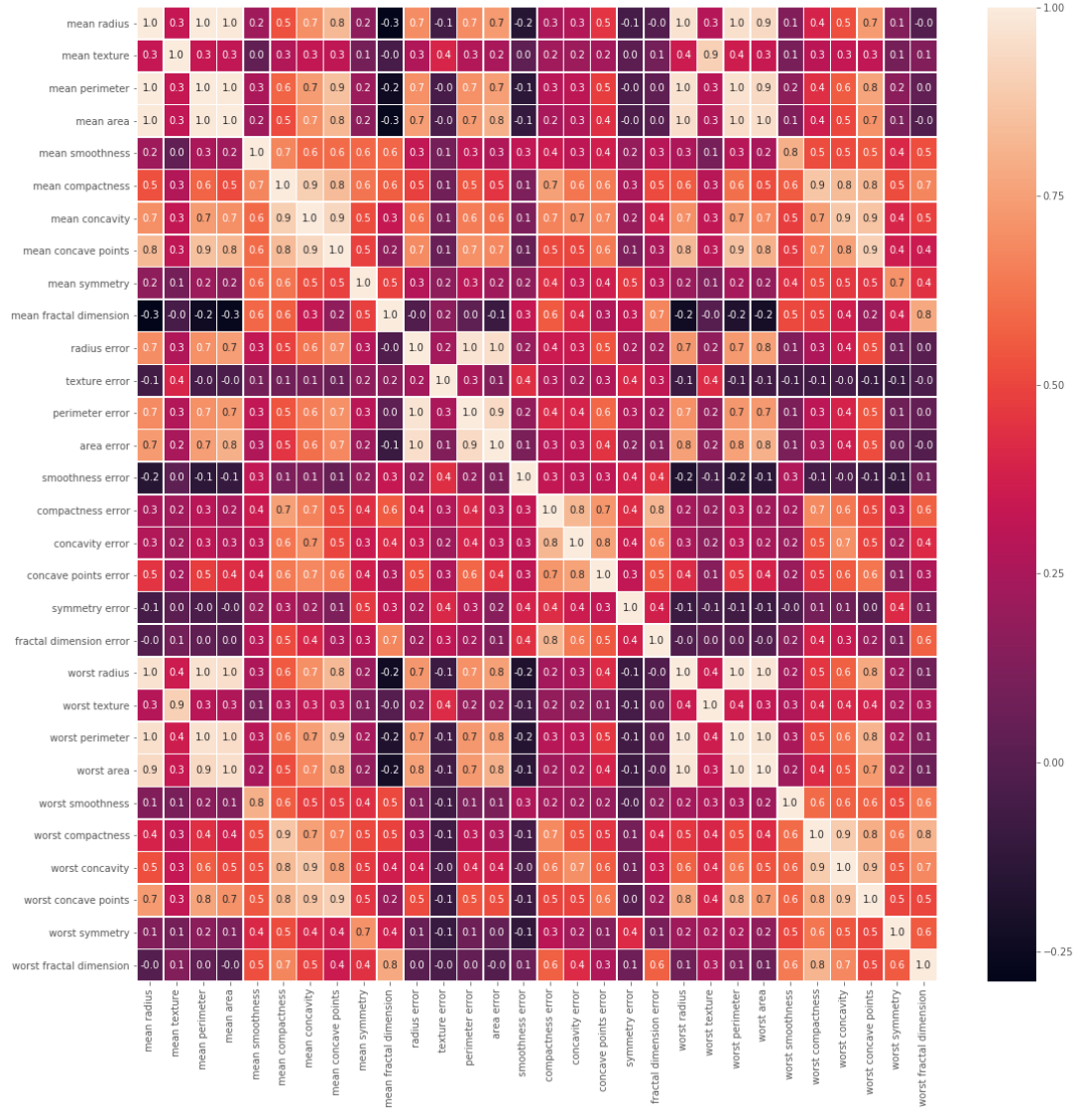
```
import numpy as np
import matplotlib.pyplot as plt


plt.ioff()

plt.figure(figsize=(11, 9))
plt.scatter(benign['mean area'], benign['mean radius'], c='blue', label='Malignan
t', alpha=0.6)
plt.scatter(malignant['mean area'], malignant['mean radius'], c='green', label='Be
nign', alpha=0.6)

x = np.arange(0, 2500, 1)
plt.plot(x, x)

plt.ylim(0, 2500)
plt.ylim(0, 2500)

plt.legend()

plt.savefig('./pair_area_radius_scale.png')
plt.close()
```


Malignant
Benign
2500
2000
1500
1000
500
0
0
500
1000
1500
2000
2500

Heatmap of correlations

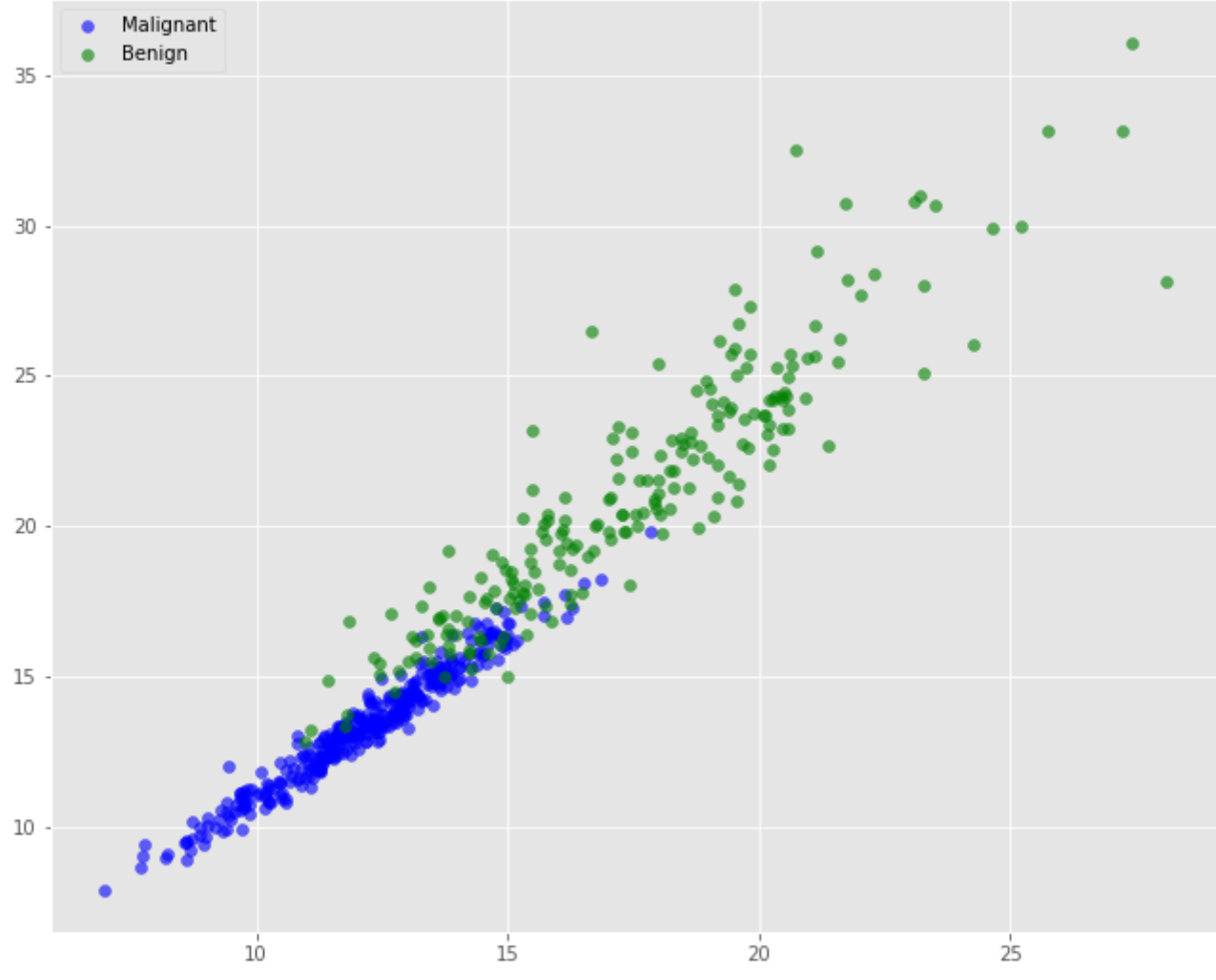
According to Wikipedia, "Heatmap is a graphical representation of data where the individual values contained in a matrix are represented as colors". In this lecture, we are using the Seaborn package (https://seaborn.pydata.org/ (https://seaborn.pydata.org/)).

In [32]:
```
import seaborn as sns

plt.ioff()

f,ax = plt.subplots(figsize=(18, 18))

heatmap = sns.heatmap(data_undersampled.corr(), annot=True, linewidths=.5, fmt= '.
1f',ax=ax)

fig = heatmap.get_figure()
fig.savefig("heatmap.png")

plt.close()
```


mean radius
mean texture
mean perimeter
mean area
mean smoothness
mean compactness
mean concavity
mean concave points
mean symmetry
mean fractal dimension
radius error
texture error
perimeter error
area error
smoothness error
compactness error
concavity error
concave points error
symmetry error
fractal dimension error
worst radius
worst texture
worst perimeter
worst area
worst smoothness
worst compactness
worst concavity
worst concave points
worst symmetry
worst fractal dimension
1.00
0.75
0.50
0.25
0.00
−0.25

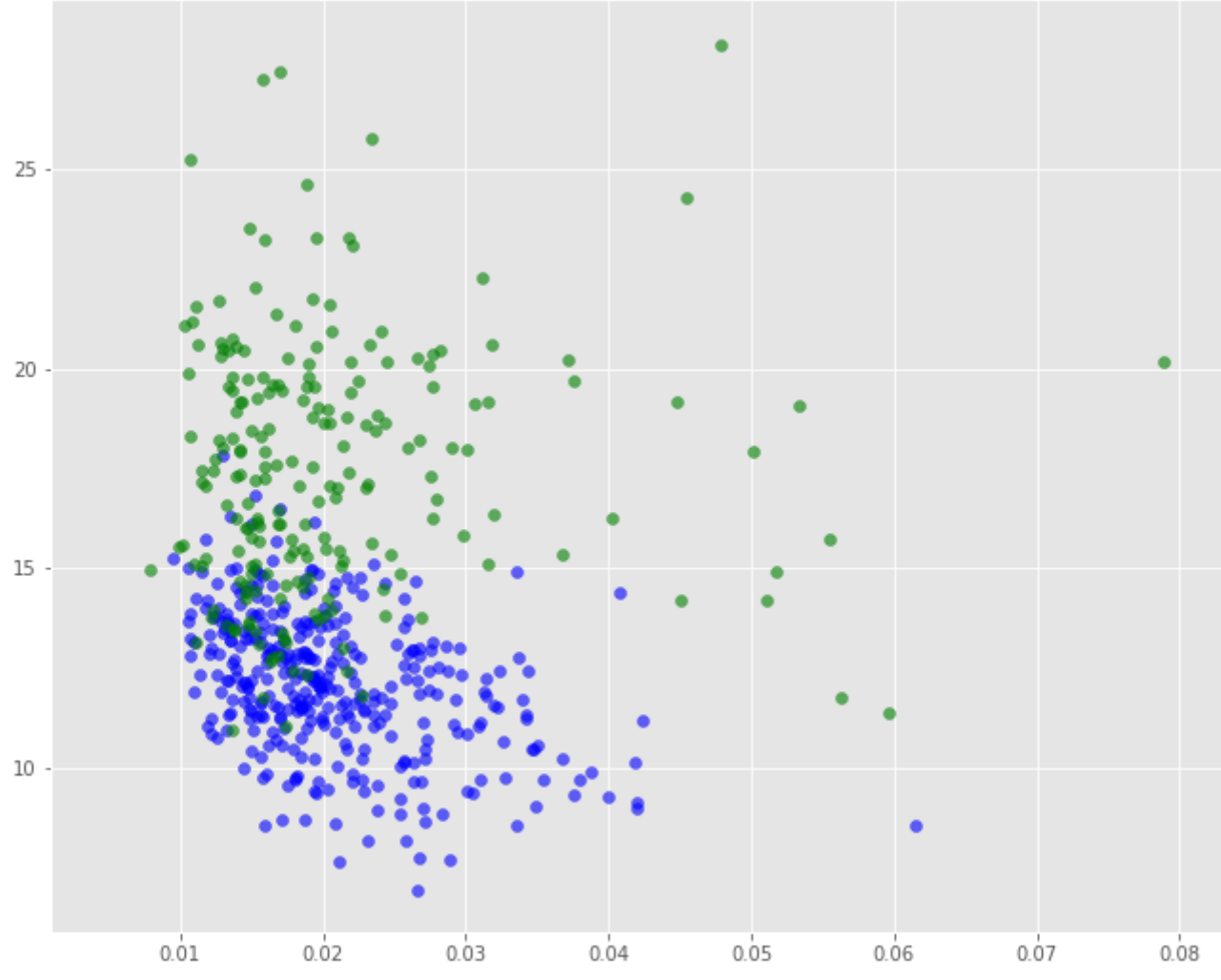
In [33]:

```
import numpy as np
import matplotlib.pyplot as plt

plt.ioff()

plt.figure(figsize=(11, 9))
plt.scatter(benign['mean radius'], benign['worst radius'], c='blue', label='Malign
ant', alpha=0.6)
plt.scatter(malignant['mean radius'], malignant['worst radius'], c='green', label=
'Benign', alpha=0.6)
plt.legend()

plt.savefig('paired_mean_worse_radius.png')
plt.close()
```


Malignant
Benign
35
30
25
20
15
10
10
15
20
25

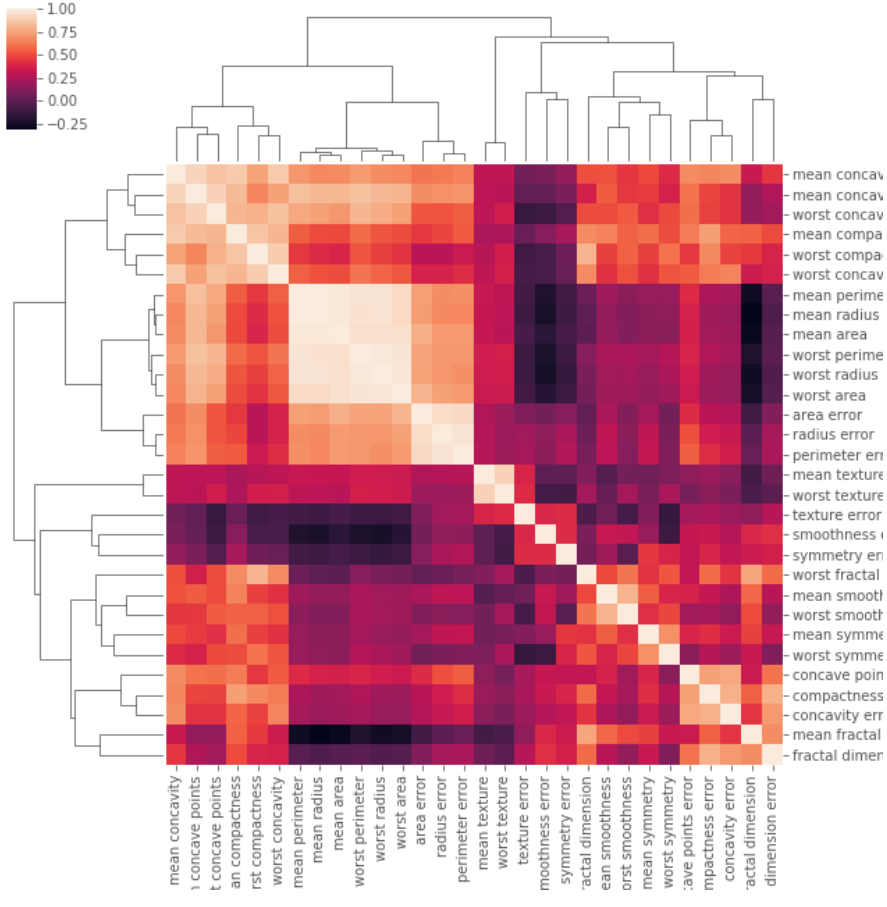
In [34]:

```
import numpy as np
import matplotlib.pyplot as plt

plt.ioff()

plt.figure(figsize=(11, 9))
plt.scatter(benign['symmetry error'], benign['mean radius'], c='blue', label='Mali
gnant', alpha=0.6)
plt.scatter(malignant['symmetry error'], malignant['mean radius'], c='green', labe
l='Benign', alpha=0.6)

plt.savefig('./paired_symmery_error_radius.png')
plt.close()
```


25
20
15
10
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08

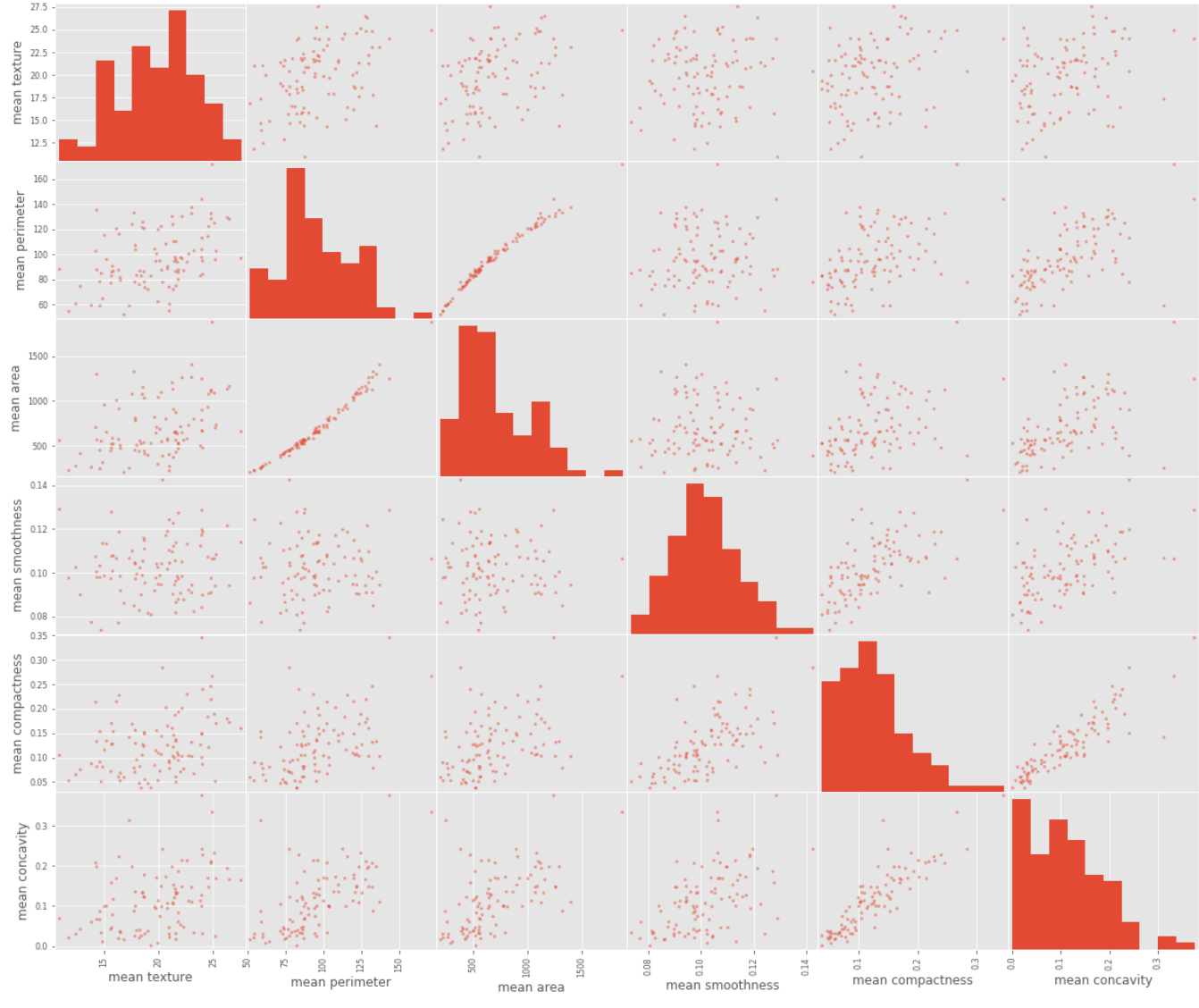
Clustermap

In [35]:
```
import seaborn as sns

cmap = sns.clustermap(data.corr())

cmap.fig.savefig("cmap.png")
plt.close()
```


1.00
0.75
0.50
0.25
0.00
−0.25
mean concav
mean concav
worst concav
mean compa
worst compa
worst concav
mean perime
mean radius
mean area
worst perime
worst radius
worst area
area error
radius error
perimeter err
mean texture
worst texture
texture error
smoothness
symmetry er
worst fractal
mean smooth
worst smooth
mean symme
worst symme
concave poin
compactness
concavity err
mean fractal
fractal dimer
mean concavity
concave points
concave points
compactness
concavity
mean perimeter
mean radius
mean area
worst perimeter
worst radius
worst area
area error
radius error
perimeter error
mean texture
worst texture
texture error
smoothness error
symmetry error
fractal dimension
smoothness
symmetry
worst symmetry
points error
compactness error
concavity error
fractal dimension
dimension error

ScatterMatrix

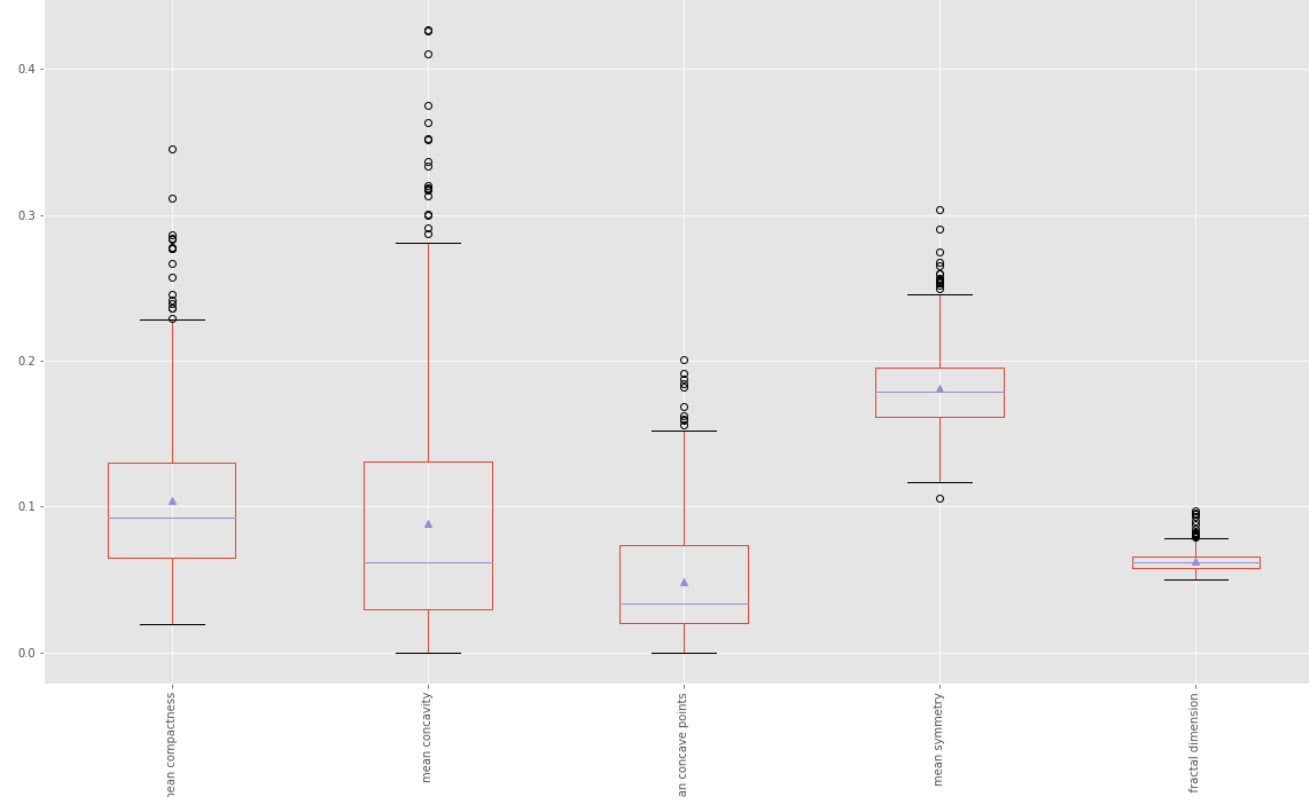
In [36]:
```
from pandas.plotting import scatter_matrix
import matplotlib.pyplot as plt

scatter_matrix(data.iloc[1:100, 1:7], figsize=(20, 17))

plt.savefig('./scatter_matrix_7.png')
plt.close()
```


mean texture
mean perimeter
mean area
mean smoothness
mean compactness
mean concavity

Boxplot

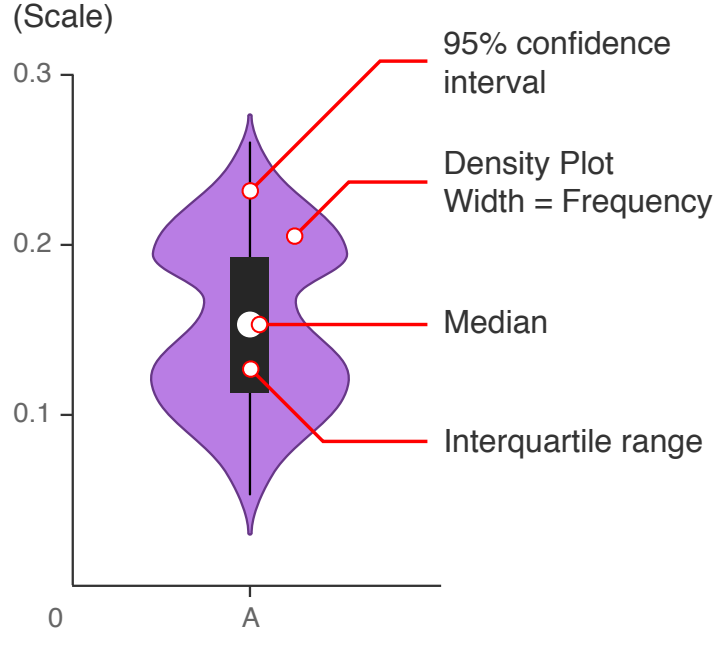
Boxplots are one of many visualizations that can show a wide range of information in a small plot: mean, median, quartiles and more. They are very useful for you to get an idea about the range of data you are dealing with. Have a look yourself:

In [37]:
```
import matplotlib.pyplot as plt

plt.ioff()

data.iloc[:, 5:10].boxplot(figsize=(20, 11), rot=90, showmeans=True)

plt.savefig('./boxplot_panda.png')
plt.close()
```

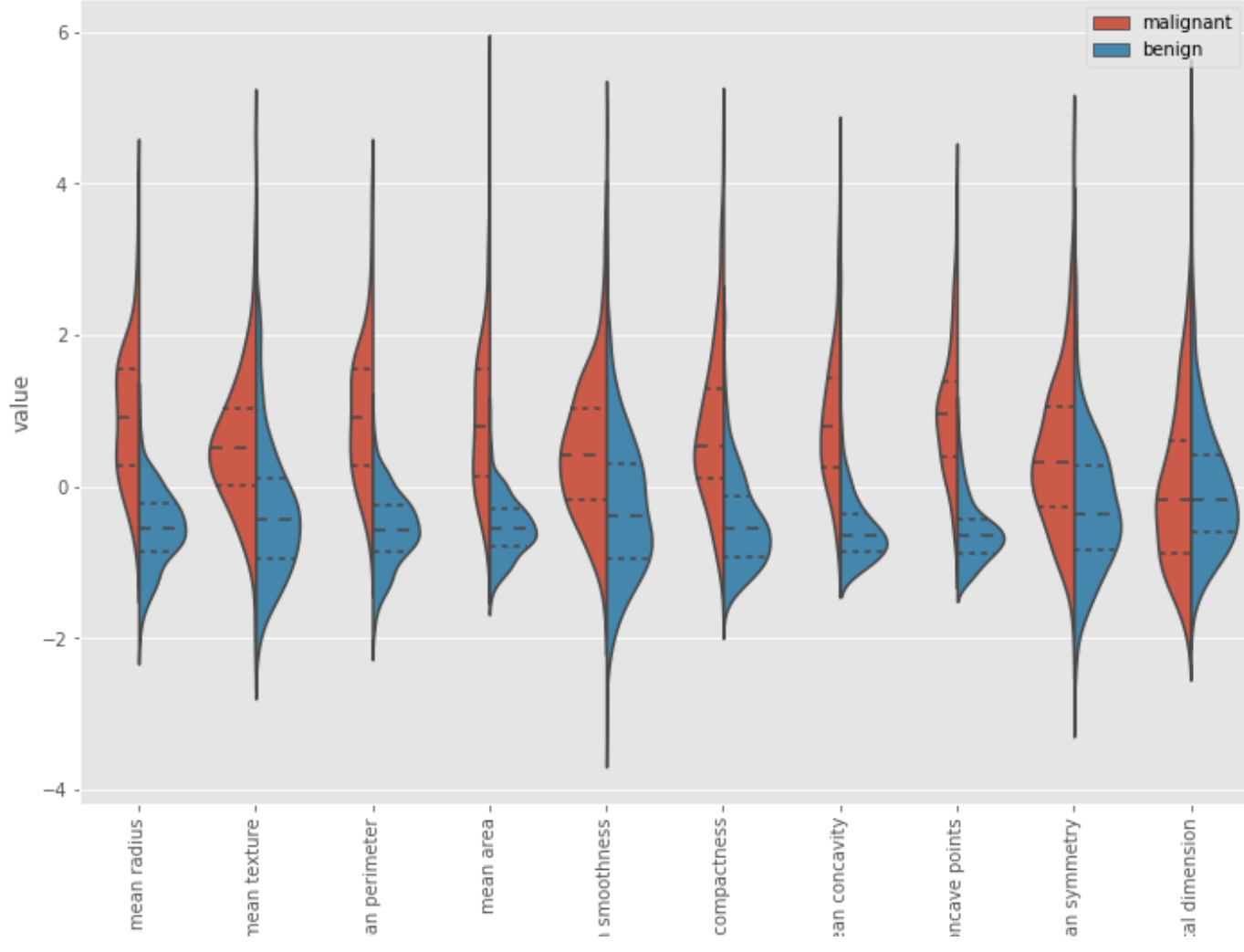

0.4
0.3
0.2
0.1
0.0
ean compactness
mean concavity
an concave points
mean symmetry
fractal dimension

Violin plot (Density, Quantiles and even more)

Violin plots are a new generation of plots that embed many different plots in a single visualizations. In order to understand how you can translate violin chart's features, we can look at the following image from Data Visualization Catalogue (https://datavizcatalogue.com):

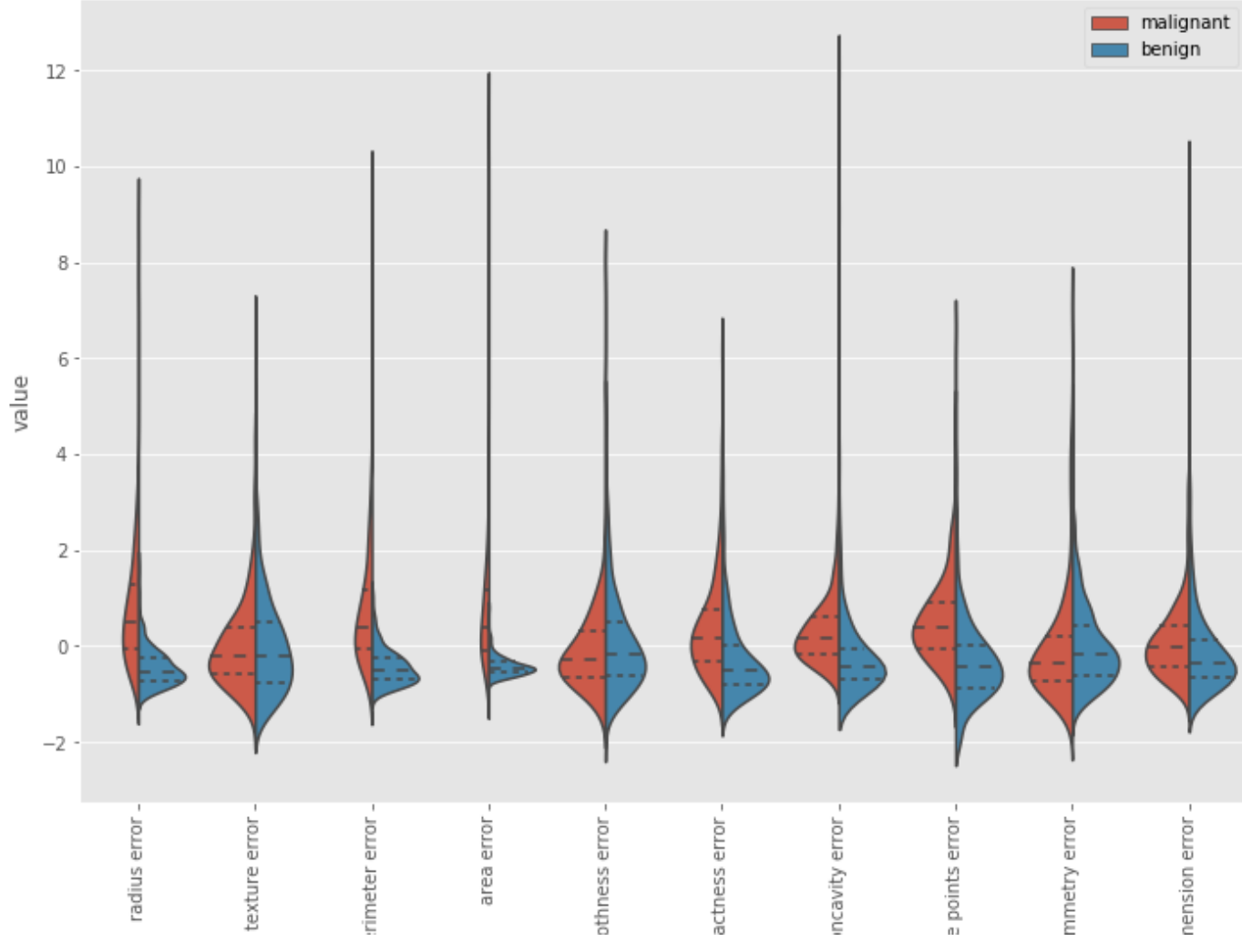


In [38]:
```
from matplotlib import pyplot

a4_dims = (11.7, 8.27)

data_normalized = data.copy()
data_normalized = (data_normalized - data_normalized.mean()) / data_normalized.std
()

data_first_ten = pd.concat([pd.Series(d.target_names[d.target]), data_normalized.i
loc[:,0:10]],axis=1)
data_first_ten_flat = pd.melt(data_first_ten, id_vars=0, var_name='features', valu
e_name='value')

plt.ioff()
fig, ax = pyplot.subplots(figsize=a4_dims)
vplot = sns.violinplot(x="features", y="value", hue=0, data=data_first_ten_flat, s
plit=True, inner='quart')

plt.xticks(rotation=90)
fig.savefig('v_0_10.png')
plt.close()
```


malignant
benign
value
6
4
2
0
−2
−4
mean radius
mean texture
an perimeter
mean area
smoothness
compactness
an concavity
ncave points
an symmetry
al dimension

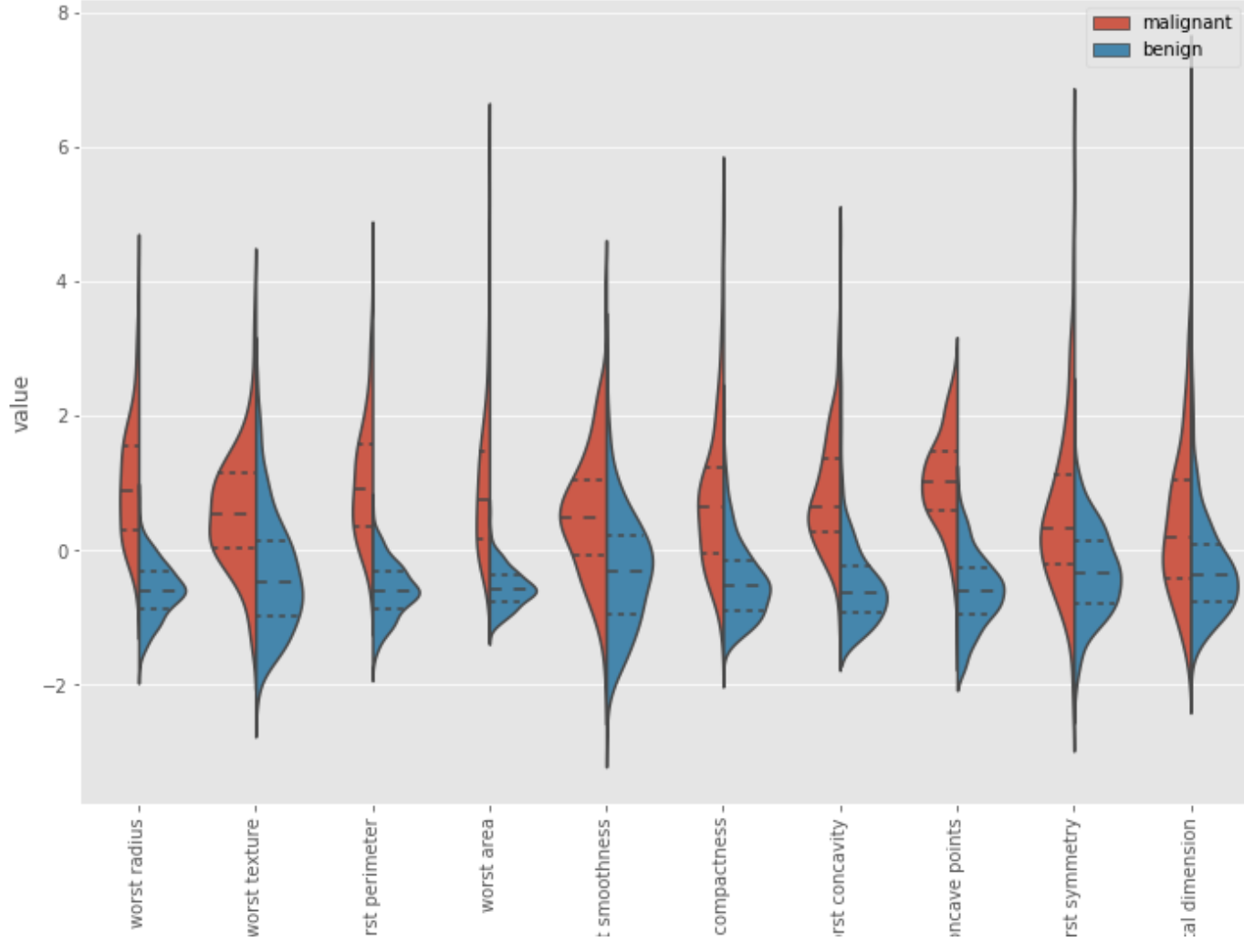
In [39]:

```
from matplotlib import pyplot

a4_dims = (11.7, 8.27)

data_second_ten = pd.concat([pd.Series(d.target_names[d.target]), data_normalized.
iloc[:,10:20]],axis=1)
data_second_ten_flat = pd.melt(data_second_ten, id_vars=0, var_name='features', va
lue_name='value')

plt.ioff()
fig, ax = pyplot.subplots(figsize=a4_dims)

sns.violinplot(x="features", y="value", hue=0, data=data_second_ten_flat, split=Tr
ue, inner='quart')
plt.xticks(rotation=90)

fig.savefig('v_10_20.png')
plt.close()
```


malignant
benign
value
12
10
8
6
4
2
0
−2
radius error
texture error
:rimeter error
area error
othness error
actness error
ncavity error
e points error
mmetry error
nension error

In [40]:

```
from matplotlib import pyplot

a4_dims = (11.7, 8.27)

data_third_ten = pd.concat([pd.Series(d.target_names[d.target]), data_normalized.i
loc[:,20:31]],axis=1)
data_third_ten_flat = pd.melt(data_third_ten, id_vars=0, var_name='features', valu
e_name='value')

fig, ax = pyplot.subplots(figsize=a4_dims)
sns.violinplot(x="features", y="value", hue=0, data=data_third_ten_flat, split=Tru
e, inner='quart')
plt.xticks(rotation=90)

fig.savefig('v_20_31.png')
plt.close()
```

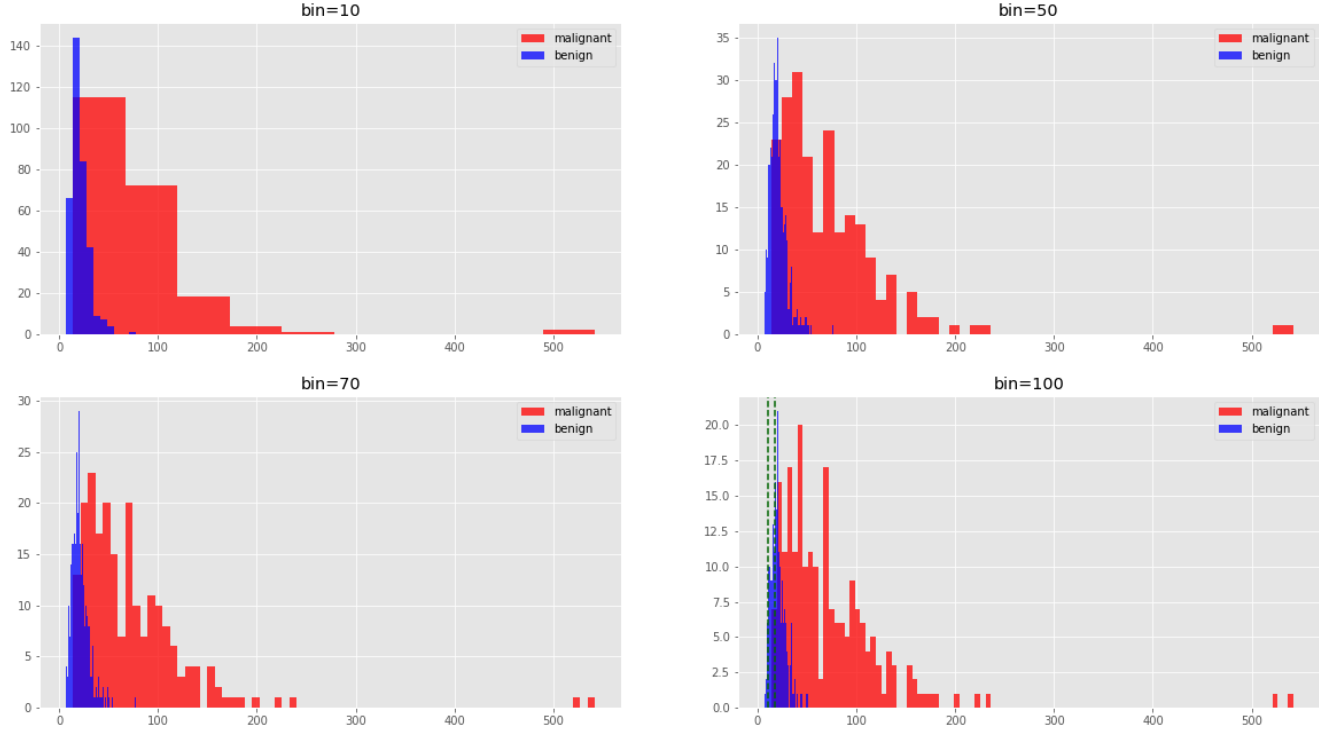

malignant
benign
value
8
6
4
2
0
−2
worst radius
worst texture
rst perimeter
worst area
t smoothness
compactness
rst concavity
ncave points
rst symmetry
al dimension

In [41]:

```
import numpy as np
import matplotlib.pyplot as plt

plt.ioff()

fig, ax = plt.subplots(2,2, figsize=(20,11))


ax[0, 0].hist(malignant['area error'], bins=10, alpha=0.75, facecolor='red', label
="malignant")
ax[0, 0].hist(benign['area error'], bins=10, alpha=0.75, facecolor='blue', label=
"benign")
ax[0, 0].legend()
ax[0, 0].set_title('bin=10')

ax[0, 1].hist(malignant['area error'], bins=50, alpha=0.75, facecolor='red', label
="malignant")
ax[0, 1].hist(benign['area error'], bins=50, alpha=0.75, facecolor='blue', label=
"benign")
ax[0, 1].legend()
ax[0, 1].set_title('bin=50')

ax[1, 0].hist(malignant['area error'], bins=70, alpha=0.75, facecolor='red', label
="malignant")
ax[1, 0].hist(benign['area error'], bins=70, alpha=0.75, facecolor='blue', label=
"benign")
ax[1, 0].legend()
ax[1, 0].set_title('bin=70')

ax[1, 1].hist(malignant['area error'], bins=100, alpha=0.75, facecolor='red', labe
l="malignant")
ax[1, 1].hist(benign['area error'], bins=100, alpha=0.75, facecolor='blue', label=
"benign")
ax[1, 1].axvline(x=11, color="darkgreen", linestyle='--')
ax[1, 1].axvline(x=18, color="darkgreen", linestyle='--')
ax[1, 1].legend()
```

```
ax[1, 1].set_title('bin=100')

fig.suptitle('Histogram: Area Error')

plt.savefig('hist_mean_area.png')
plt.close()
```

Histogram: Area Error



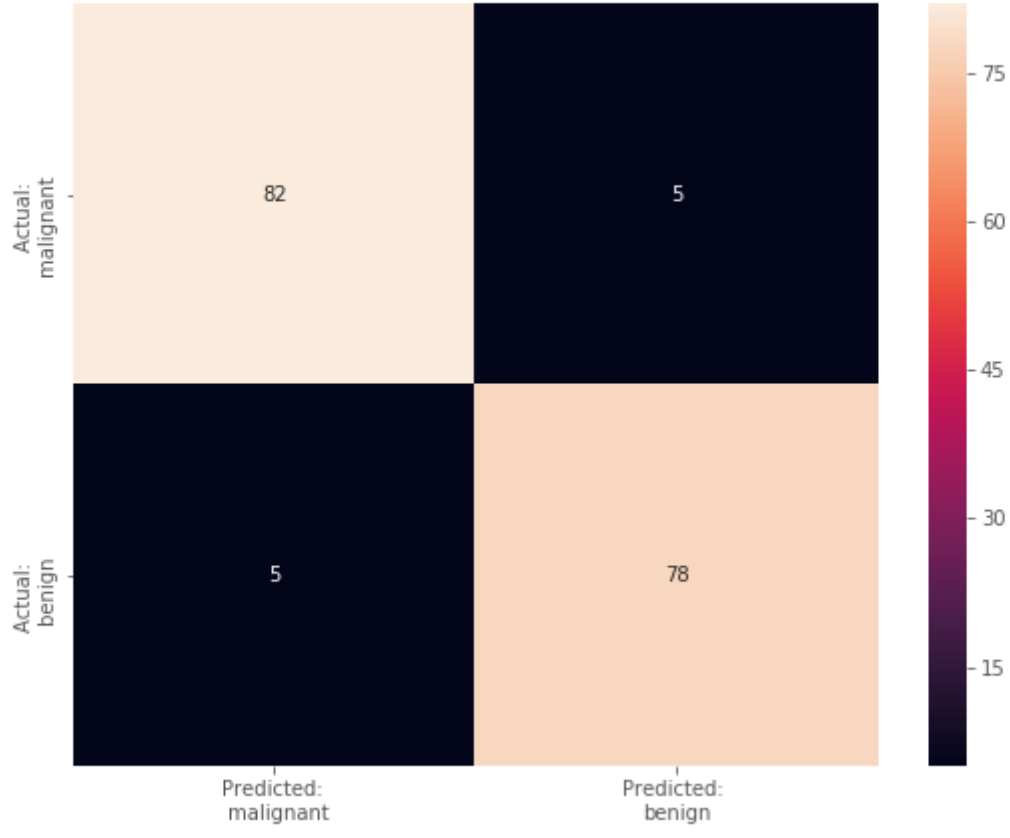
Visualizations for Modelling

Predictions

In [42]:
```
from sklearn import tree

clf = tree.DecisionTreeClassifier(min_samples_leaf=5)
```

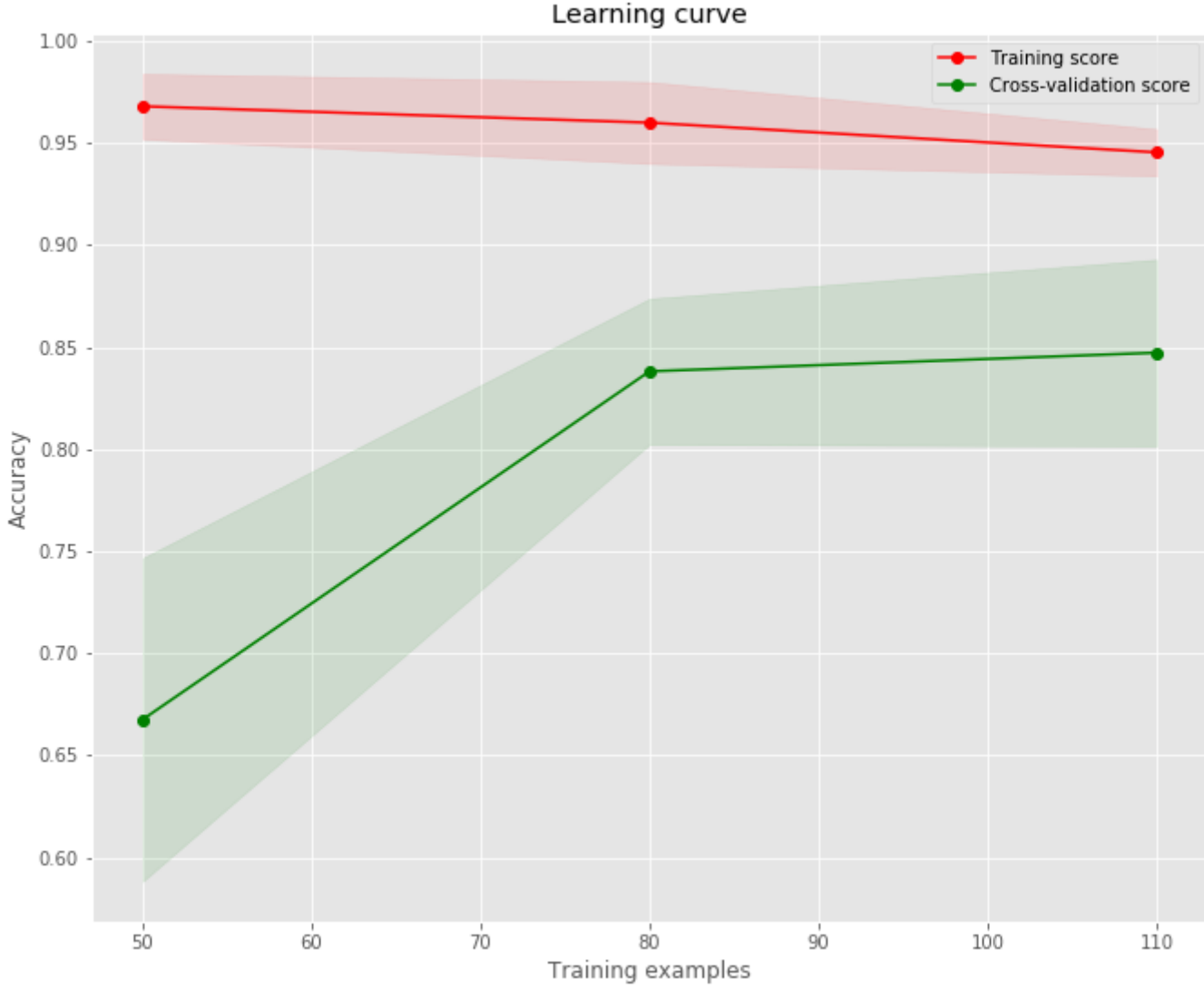
In [43]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix

X_train, X_test, y_train, y_test = train_test_split(X_undersampled,
                                                    y_undersampled, test_size=0.4,
 random_state=0)

clf = clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
```

In [44]:
```
fig, ax = plt.subplots(figsize=(9, 7))

sns.heatmap(cm,annot=True,fmt="d", xticklabels=['Predicted: \n malignant', 'Predic
ted: \nbenign'],
            yticklabels=['Actual: \n malignant', 'Actual: \n benign'])

fig.savefig('./confusion_matrix.png')
plt.close()
```


82
5
5
78
Actual: malignant
Actual: benign
Predicted: malignant
Predicted: benign
75
60
45
30
15

Learning curve

In [45]:
```
from sklearn.model_selection import ShuffleSplit
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.datasets import load_digits
from sklearn.model_selection import learning_curve


plt.ioff()

plt.figure(figsize=(11,9))
plt.title("Learning curve")
plt.xlabel("Training examples")
plt.ylabel("Accuracy")

train_sizes, train_scores, valid_scores = learning_curve(clf, X, y, train_sizes=[5
0, 80, 110], cv=5)
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
valid_scores_mean = np.mean(valid_scores, axis=1)
valid_scores_std = np.std(valid_scores, axis=1)

plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                 train_scores_mean + train_scores_std, alpha=0.1,
                 color="r")
plt.fill_between(train_sizes, valid_scores_mean - valid_scores_std,
                 valid_scores_mean + valid_scores_std, alpha=0.1, color="g")
plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
         label="Training score")
plt.plot(train_sizes, valid_scores_mean, 'o-', color="g",
         label="Cross-validation score")

plt.legend(loc="best")

plt.savefig('learningcurve.png')
plt.close()
```

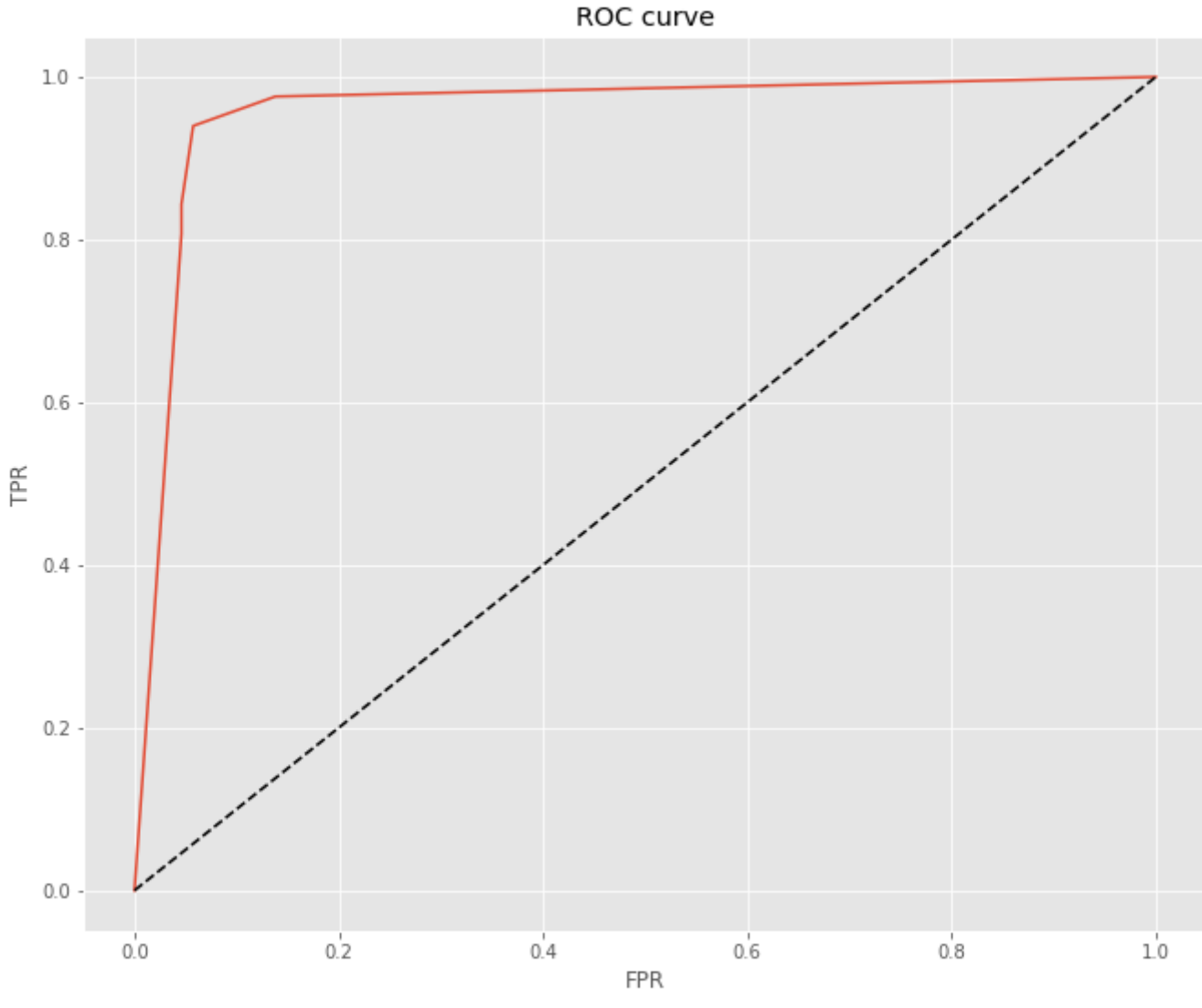

Learning curve
Training score
Cross-validation score
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
Accuracy
50
60
70
80
90
100
110
Training examples

ROC

According to Wikipedia, "a receiver operating characteristic curve, i.e., ROC curve, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied". We can use methods like **predict_proba** on a classifier and use the **roc_curve**:

In [46]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_curve, roc_auc_score
from matplotlib import pyplot as plt

predictions = clf.predict_proba(X_test)
fpr, tpr, _ = roc_curve(y_test, predictions[:,1])

plt.ioff()
plt.figure(figsize=(11,9))

plt.clf()
plt.plot(fpr, tpr)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC curve')

plt.savefig('./roc.png')
plt.close()
```


ROC curve
TPR
FPR
0.0
0.2
0.4
0.6
0.8
1.0

Graph Visualization of a model

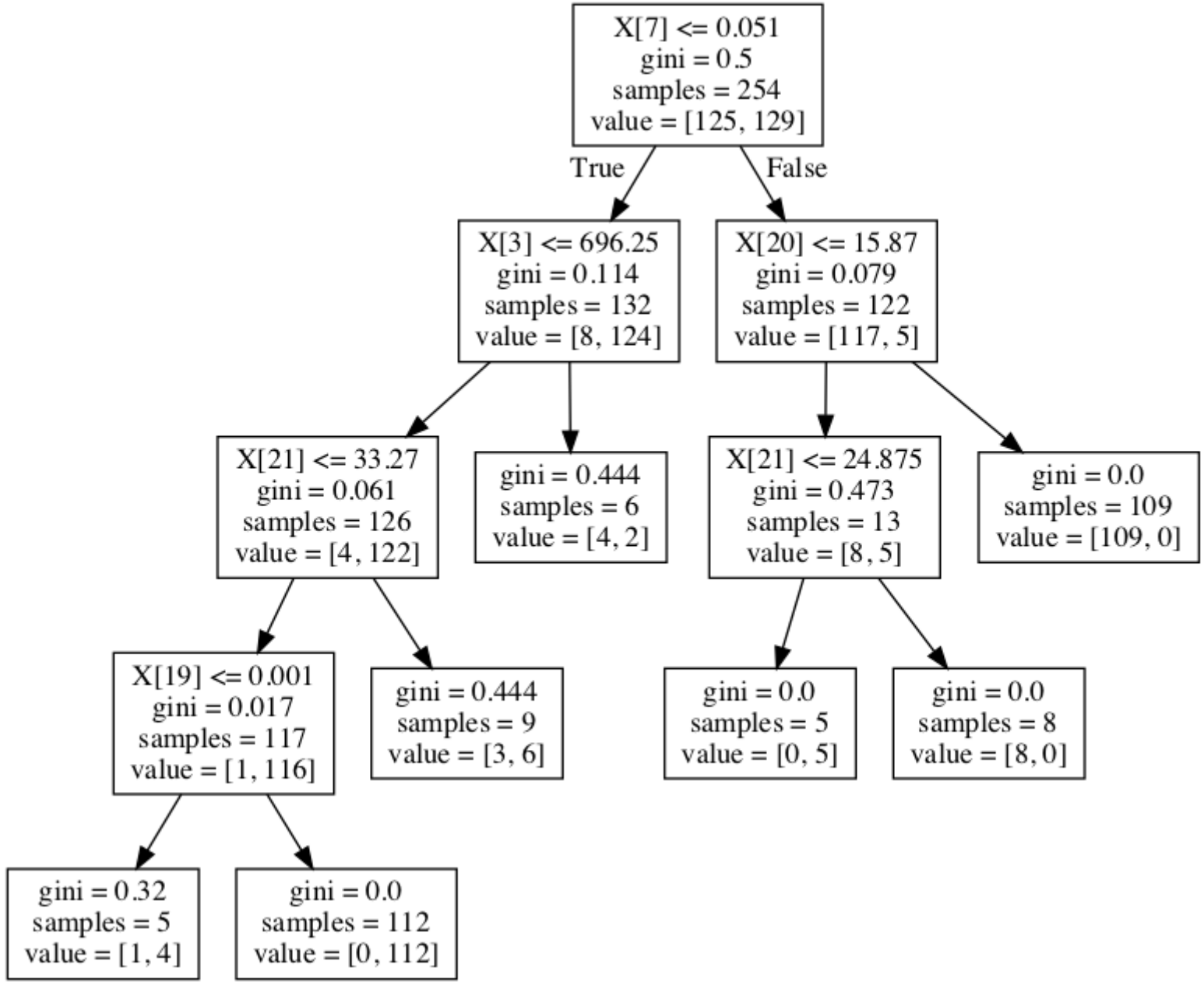
When working with specific set of models, you can have a visualization of the model itself. The good news is that decision trees are one of those models. Let us try to see what would a typical model looks like:

In [47]:
```
from sklearn import tree

tree.export_graphviz(clf, out_file='tree.dot')
```

In order to read the dot files, you need to have **GraphViz** installed in your computer. You can find a guide on how to install graphviz here: https://graphviz.gitlab.io/download/ (https://graphviz.gitlab.io/download/).

After installing graphviz, you can open your command line and run the following to convert the dot file to a PNG image file:

In [48]:
```
!dot tree.dot -Tpng -o tree.png
```

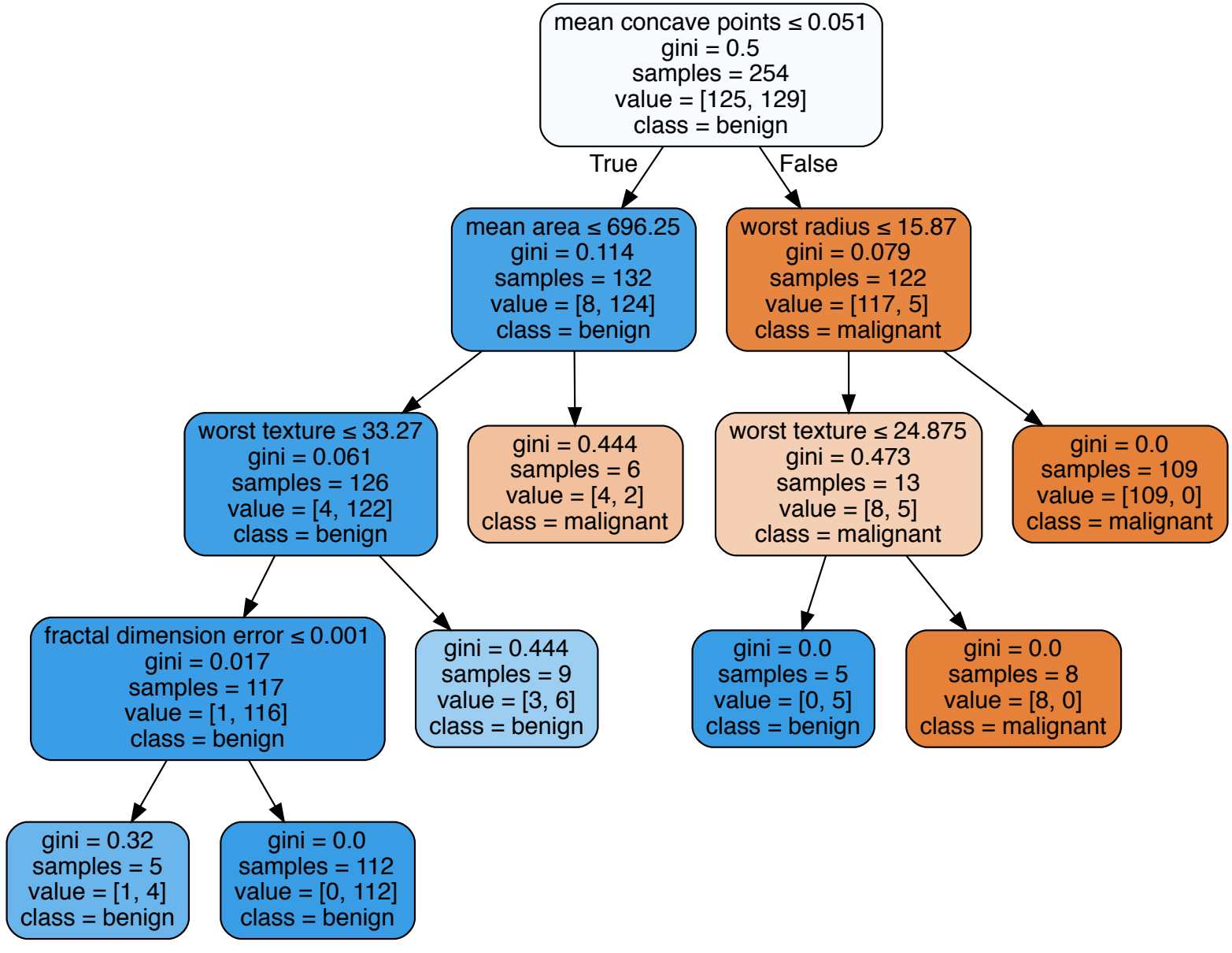

X[7] <= 0.051
gini = 0.5
samples = 254
value = [125, 129]
True
False
X[3] <= 696.25
gini = 0.114
samples = 132
value = [8, 124]
X[20] <= 15.87
gini = 0.079
samples = 122
value = [117, 5]
X[21] <= 33.27
gini = 0.061
samples = 126
value = [4, 122]
gini = 0.444
samples = 6
value = [4, 2]
X[21] <= 24.875
gini = 0.473
samples = 13
value = [8, 5]
gini = 0.0
samples = 109
value = [109, 0]
X[19] <= 0.001
gini = 0.017
samples = 117
value = [1, 116]
gini = 0.444
samples = 9
value = [3, 6]
gini = 0.0
samples = 5
value = [0, 5]
gini = 0.0
samples = 8
value = [8, 0]
gini = 0.32
samples = 5
value = [1, 4]
gini = 0.0
samples = 112
value = [0, 112]

In [49]:
```
!pip install graphviz
```

```
Requirement already satisfied: graphviz in /Users/amirrahnama/anaconda3/envs/t
f/lib/python2.7/site-packages (0.10.1)
You are using pip version 10.0.1, however version 18.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
```

In [50]:
```
import graphviz

dot_data = tree.export_graphviz(clf, out_file=None,
                                feature_names=d.feature_names,
                                class_names=d.target_names,
                                filled=True, rounded=True,
                                special_characters=True)
graph = graphviz.Source(dot_data)
graph
```

Out[50]:


mean concave points ≤ 0.051
gini = 0.5
samples = 254
value = [125, 129]
class = benign
True
False
mean area ≤ 696.25
gini = 0.114
samples = 132
value = [8, 124]
class = benign
worst radius ≤ 15.87
gini = 0.079
samples = 122
value = [117, 5]
class = malignant
worst texture ≤ 33.27
gini = 0.061
samples = 126
value = [4, 122]
class = benign
gini = 0.444
samples = 6
value = [4, 2]
class = malignant
worst texture ≤ 24.875
gini = 0.473
samples = 13
value = [8, 5]
class = malignant
gini = 0.0
samples = 109
value = [109, 0]
class = malignant
fractal dimension error ≤ 0.001
gini = 0.017
samples = 117
value = [1, 116]
class = benign
gini = 0.444
samples = 9
value = [3, 6]
class = benign
gini = 0.0
samples = 5
value = [0, 5]
class = benign
gini = 0.0
samples = 8
value = [8, 0]
class = malignant
gini = 0.32
samples = 5
value = [1, 4]
class = benign
gini = 0.0
samples = 112
value = [0, 112]
class = benign

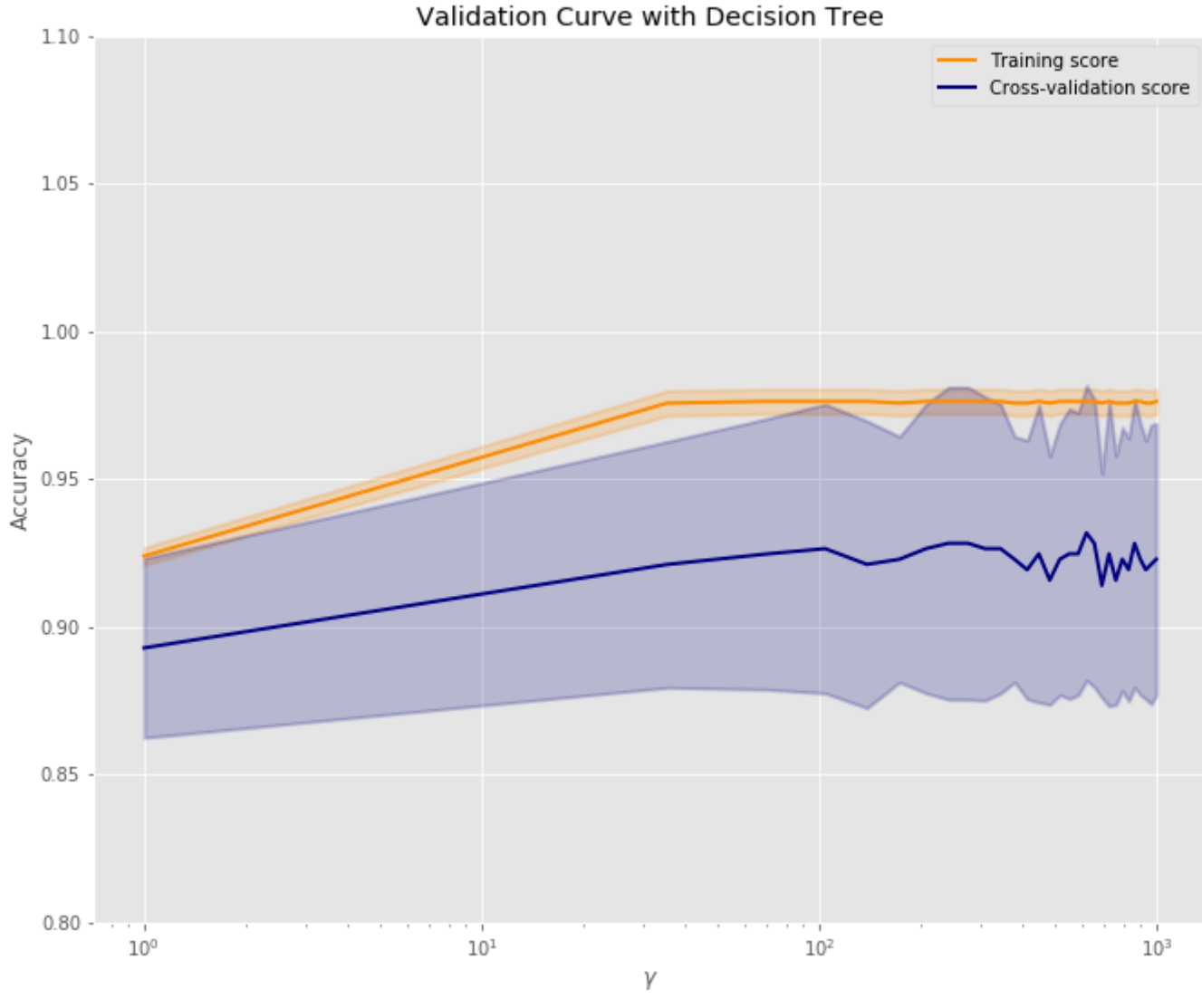
Validation Curve

In [51]:
```
clf.get_params().keys()
```

Out[51]:
```
['presort',
 'splitter',
 'min_impurity_decrease',
 'max_leaf_nodes',
 'min_samples_leaf',
 'min_samples_split',
 'min_weight_fraction_leaf',
 'criterion',
 'random_state',
 'min_impurity_split',
 'max_features',
 'max_depth',
 'class_weight']
```

In [56]:

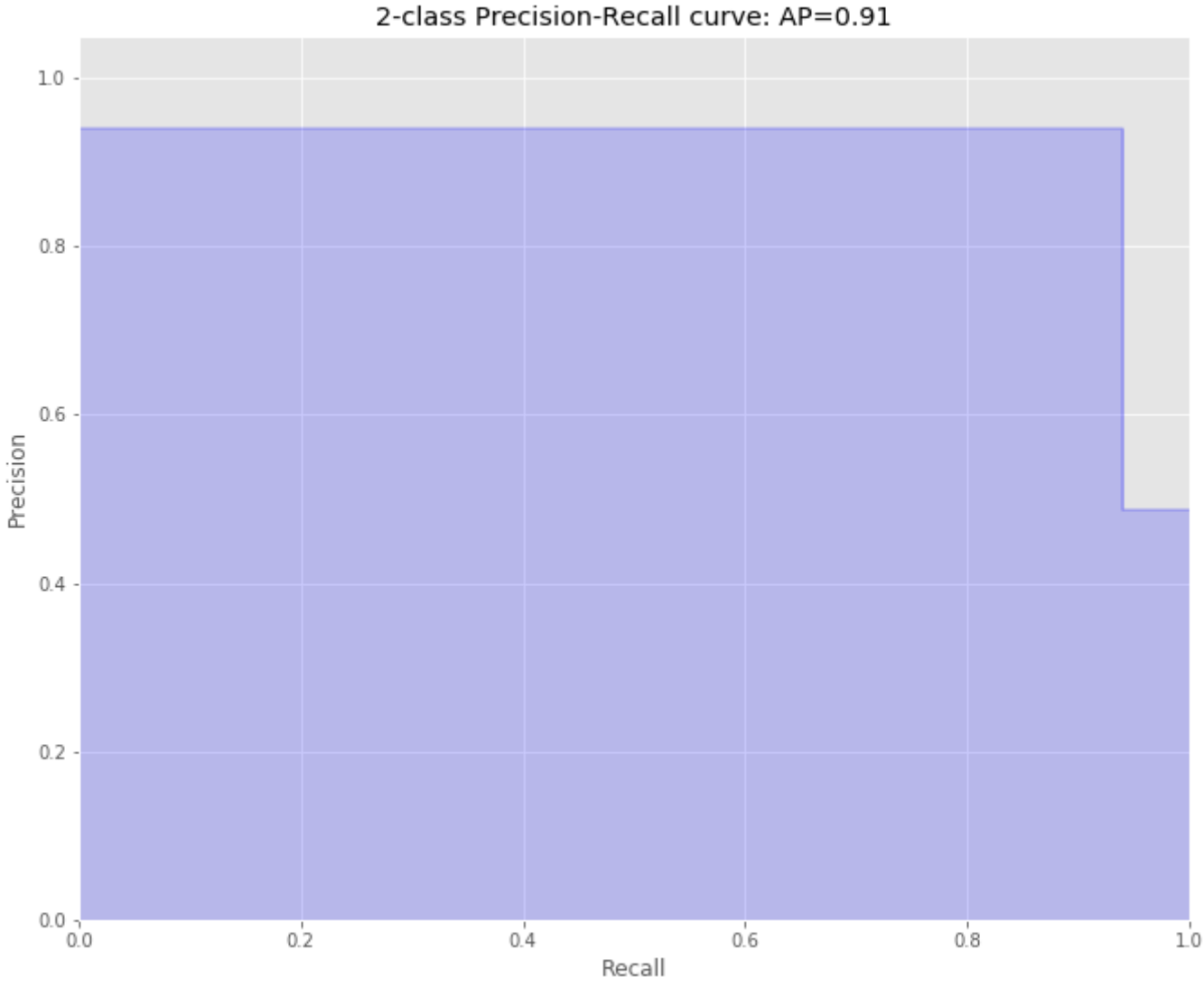
```
from sklearn.model_selection import validation_curve


param_range = np.linspace(1, 1000, 30)
train_scores, test_scores = validation_curve(
    clf, X, y, param_name="max_depth", param_range=param_range,
    cv=10, scoring="accuracy", n_jobs=1)
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)


plt.figure(figsize=(11,9))
plt.title("Validation Curve with Decision Tree")
plt.xlabel("$\gamma$")
plt.ylabel("Accuracy")
plt.ylim(0.8, 1.1)
lw = 2
plt.semilogx(param_range, train_scores_mean, label="Training score",
             color="darkorange", lw=lw)
plt.fill_between(param_range, train_scores_mean - train_scores_std,
                 train_scores_mean + train_scores_std, alpha=0.2,
                 color="darkorange", lw=lw)
plt.semilogx(param_range, test_scores_mean, label="Cross-validation score",
             color="navy", lw=lw)
plt.fill_between(param_range, test_scores_mean - test_scores_std,
                 test_scores_mean + test_scores_std, alpha=0.2,
                 color="navy", lw=lw)
plt.legend(loc="best")

plt.savefig('./validation_curve.png')
plt.close()
```


Validation Curve with Decision Tree
Training score
Cross-validation score
Accuracy
1.10
1.05
1.00
0.95
0.90
0.85
0.80
10^0
10^1
10^2
10^3
γ

Precision Recall

Precision-Recall is a useful measure of success of prediction when the classes are very imbalanced. In information retrieval, precision is a measure of result relevancy, while recall is a measure of how many truly relevant results are returned. In Sklearn, we have the ability to visualize the curve very easily:

In [53]:
```
from sklearn.metrics import precision_recall_curve
import matplotlib.pyplot as plt
from sklearn.utils.fixes import signature
from sklearn.metrics import average_precision_score

average_precision = average_precision_score(y_test, y_pred)

precision, recall, _ = precision_recall_curve(y_test, y_pred)

step_kwargs = ({'step': 'post'}
               if 'step' in signature(plt.fill_between).parameters
               else {})
plt.figure(figsize=(11,9))
plt.step(recall, precision, color='b', alpha=0.2,
         where='post')
plt.fill_between(recall, precision, alpha=0.2, color='b', **step_kwargs)

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title('2-class Precision-Recall curve: AP={0:0.2f}'.format(
          average_precision))

plt.savefig('./precision_recall.png')
plt.close()
```

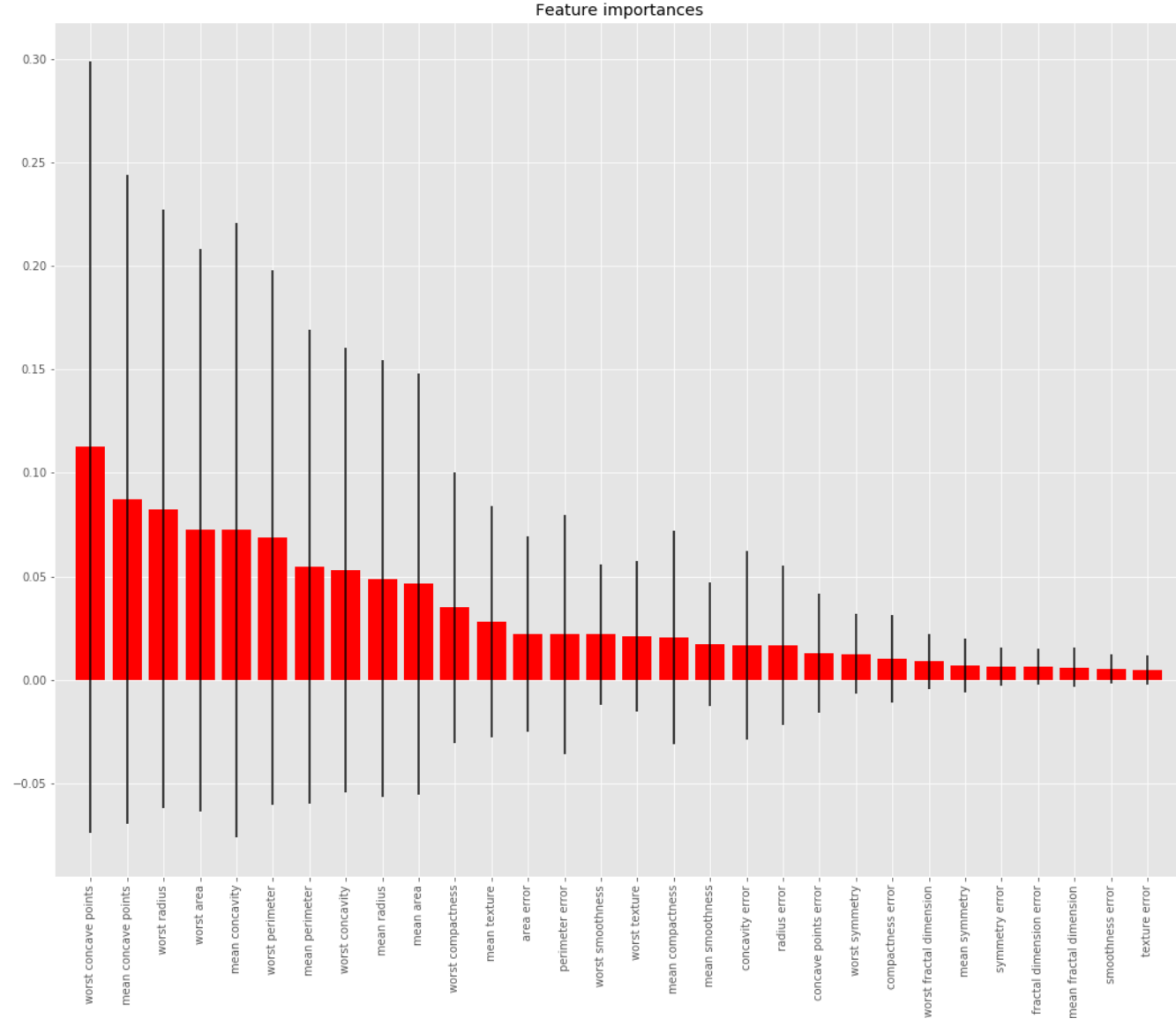

2-class Precision-Recall curve: AP=0.91
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0

Feature importance for a random forest

Some Machine Learning models like Random Forest (RF) not only predict but also show which features in their learning method played a role. Let us have a look at a scenario to model the data using a RandomForest:

In [54]:

```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.ensemble import ExtraTreesClassifier

# Build a forest and compute the feature importances
forest = ExtraTreesClassifier(n_estimators=250,
                              random_state=0)

forest.fit(X_train, y_train)
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]

# Plot the feature importances of the forest
plt.figure(figsize=(18, 14))
plt.title("Feature importances")
plt.bar(range(X_train.shape[1]), importances[indices],
       color="r", yerr=std[indices], align="center")
plt.xticks(range(X_train.shape[1]), data_undersampled.columns[indices], rotation=9
0)
plt.xlim([-1, X_train.shape[1]])

plt.savefig('./rf_feature_importance.png')
plt.close()
```


Feature importances
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−0.05
worst concave points
mean concave points
worst radius
worst area
mean concavity
worst perimeter
mean perimeter
worst concavity
mean radius
mean area
worst compactness
mean texture
area error
perimeter error
worst smoothness
worst texture
mean compactness
mean smoothness
concavity error
radius error
concave points error
worst symmetry
compactness error
worst fractal dimension
mean symmetry
symmetry error
fractal dimension error
mean fractal dimension
smoothness error
texture error

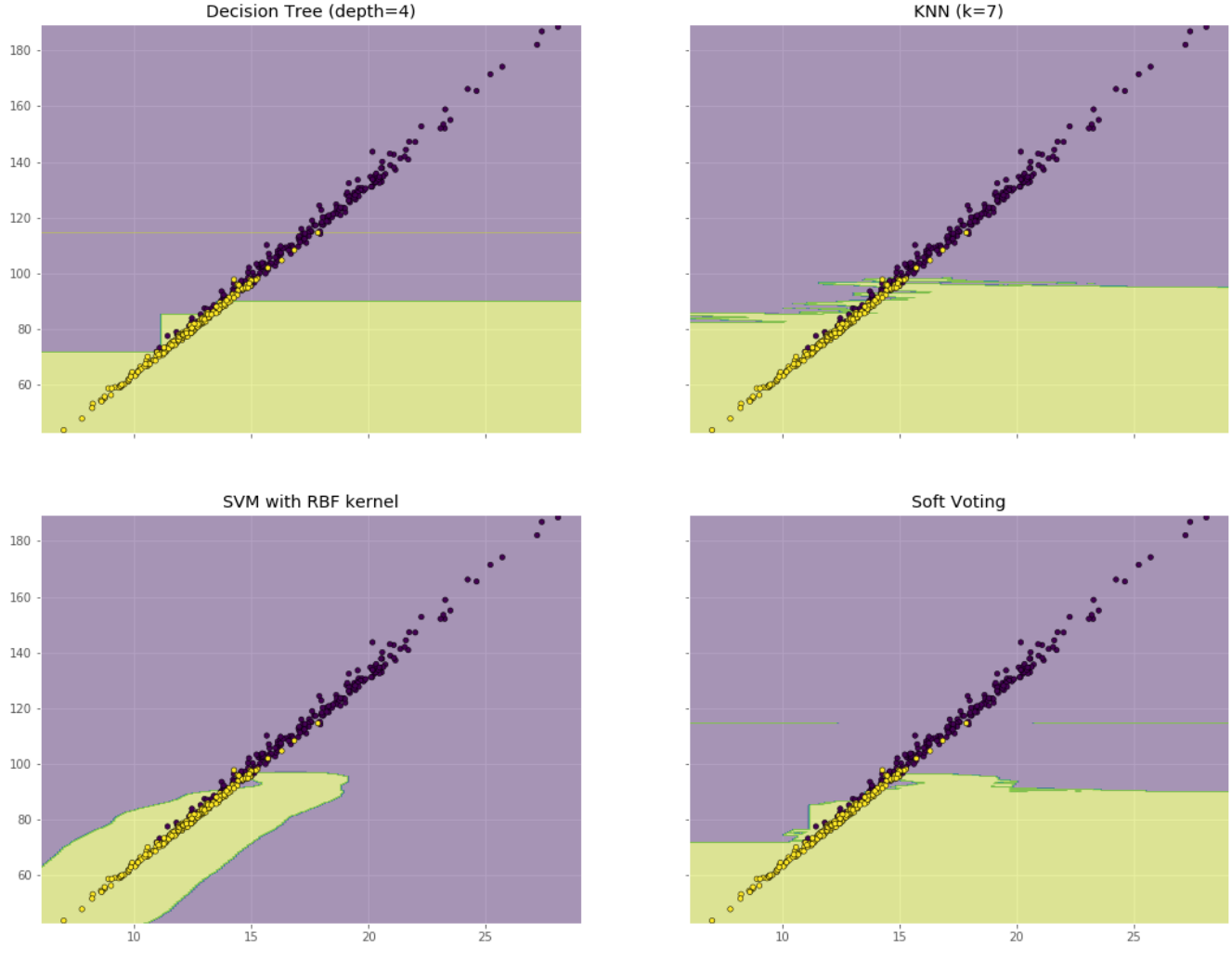
Decision Boundary Analysis

A Machine Learning model learns to divide the feature space with a hyperplane. If we reduce our problem to a two or three dimensional problem, that hyperplane will become visible. In our example, we reduce our input to only the "mean radius" and "mean perimeter" and show the hyperplane that different classifiers learn to create to partition the data:

In [55]:

```
from itertools import product
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier


X_decision = X_undersampled[:, [0, 2]]
y_decision = y_undersampled

# Training classifiers
clf1 = DecisionTreeClassifier(max_depth=4)
clf2 = KNeighborsClassifier(n_neighbors=7)
clf3 = SVC(gamma=.1, kernel='rbf', probability=True)
eclf = VotingClassifier(estimators=[('dt', clf1), ('knn', clf2),
                                    ('svc', clf3)],
                        voting='soft', weights=[2, 1, 2])

clf1.fit(X_decision, y_decision)
clf2.fit(X_decision, y_decision)
clf3.fit(X_decision, y_decision)
eclf.fit(X_decision, y_decision)

# Plotting decision regions
x_min, x_max = X_decision[:, 0].min() - 1, X_decision[:, 0].max() + 1
y_min, y_max = X_decision[:, 1].min() - 1, X_decision[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                     np.arange(y_min, y_max, 0.1))

f, axarr = plt.subplots(2, 2, sharex='col', sharey='row', figsize=(18, 14))

for idx, clff, tt in zip(product([0, 1], [0, 1]),
                         [clf1, clf2, clf3, eclf],
```

```
                          ['Decision Tree (depth=4)', 'KNN (k=7)',
                           'SVM with RBF kernel', 'Soft Voting']):

    Z = clff.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    axarr[idx[0], idx[1]].contourf(xx, yy, Z, alpha=0.4)
    axarr[idx[0], idx[1]].scatter(X_decision[:, 0], X_decision[:, 1], c=y_decision
,
                                  s=20, edgecolor='k')
    axarr[idx[0], idx[1]].set_title(tt)

plt.savefig('./decision_boundary.png')
plt.close()
```


Decision Tree (depth=4)
KNN (k=7)
SVM with RBF kernel
Soft Voting
180
160
140
120
100
80
60
10
15
20
25

Libraries

A lot of the examples I covered and I showed you how to code them, you can find in this package/library: https://github.com/reiinakano/scikit-plot (https://github.com/reiinakano/scikit-plot)

Thank you for your attention!

In the next lecture, we will cover libraries such as **Tensorflow**, Neural Networks, RandomForests and GPU computation using these libraries!