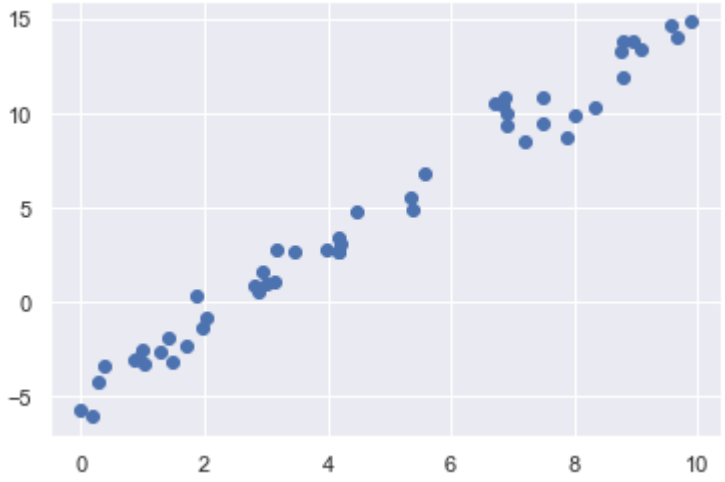
In [9]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
import numpy as np

#Import library yang dibutuhkan
```

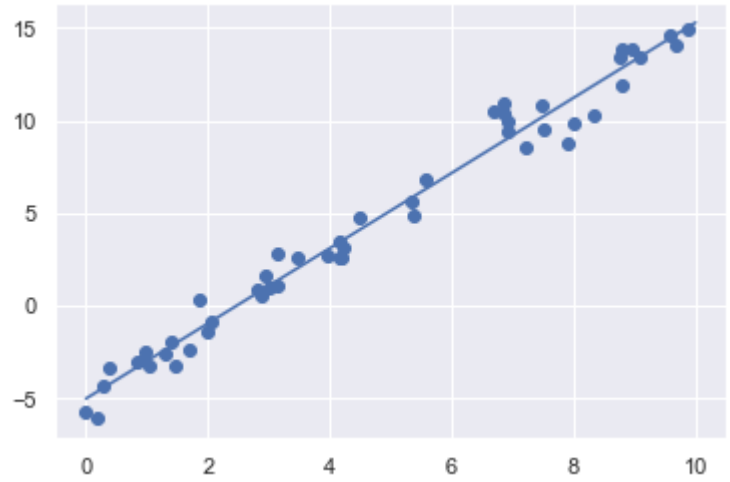
In [10]:
```
rng = np.random.RandomState(1)
x = 10 * rng.rand(50)
y = 2 * x - 5 + rng.randn(50)
plt.scatter(x,y)

#menampilkan scatter dari x dan y
```

Out[10]: <matplotlib.collections.PathCollection at 0x215de117f70>



In [11]:
```
from sklearn.linear_model import LinearRegression
model = LinearRegression(fit_intercept=True)

model.fit(x[:,np.newaxis],y)

xfit=np.linspace(0,10,1000)
yfit=model.predict(xfit[:,np.newaxis])

plt.scatter(x,y)
plt.plot(xfit,yfit)
```

Out[11]: [<matplotlib.lines.Line2D at 0x215de9aa2b0>]



In [12]:
```
print("Model slope:    ",model.coef_[0])
print("Model intercept:",model.intercept_)
```

```
Model slope:     2.027208810360695
Model intercept: -4.998577085553202
```

In [13]:
```
rng=np.random.RandomState(1)
X = 10 * rng.rand(100,3)
y = 0.5 + np.dot(X,[1.5,-2.,1.])

model.fit(X,y)
print(model.intercept_)
print(model.coef_)

#Menampilkan model intercept dan coef
```

```
0.5000000000000064
[ 1.5 -2.   1. ]
```

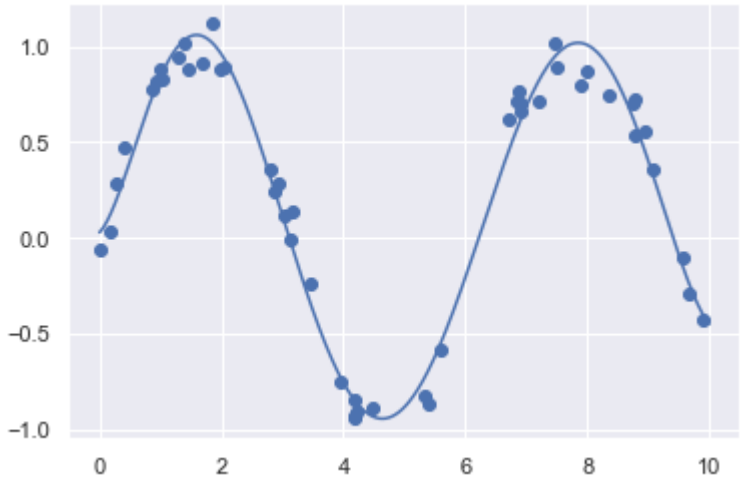
In [14]:
```
from sklearn.preprocessing import PolynomialFeatures
x=np.array([2,3,4])
poly = PolynomialFeatures(3,include_bias=False)
poly.fit_transform(x[:,None])
```

Out[14]:
```
array([[ 2.,  4.,  8.],
       [ 3.,  9., 27.],
       [ 4., 16., 64.]])
```

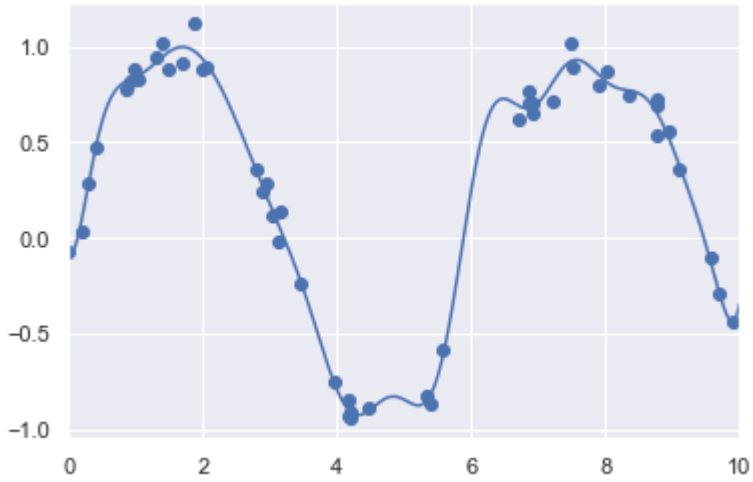
In [15]:
```
from sklearn.pipeline import make_pipeline
poly_model = make_pipeline(PolynomialFeatures(7),LinearRegression())
```

In [16]:
```
rng= np.random.RandomState(1)
x = 10*rng.rand(50)
y = np.sin(x) + 0.1 * rng.randn(50)
poly_model.fit(x[:,np.newaxis],y)
yfit = poly_model.predict(xfit[:,np.newaxis])

plt.scatter(x,y)
plt.plot(xfit,yfit)
```

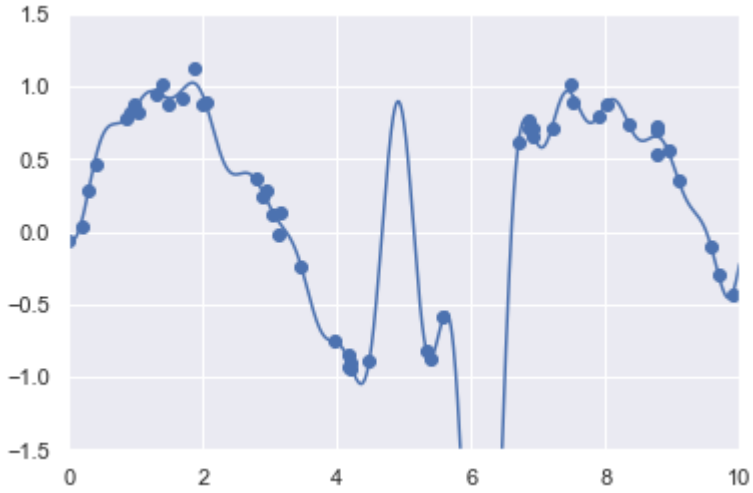
Out[16]:
```
[<matplotlib.lines.Line2D at 0x215dea26160>]
```



In [17]:
```
from sklearn.base import BaseEstimator, TransformerMixin

class GaussianFeatures(BaseEstimator, TransformerMixin):

    def __init__(self, N, width_factor=2.0):
        self.N = N
        self.width_factor = width_factor

    @staticmethod
    def _gauss_basis(x, y, width, axis=None):
        arg = (x - y) / width
        return np.exp(-0.5 * np.sum(arg ** 2, axis))

    def fit(self, X, y=None):
        # create N centers spread along the data range
        self.centers_ = np.linspace(X.min(), X.max(), self.N)
        self.width_ = self.width_factor * (self.centers_[1] - self.centers_[0])
        return self

    def transform(self, X):
        return self._gauss_basis(X[:, :, np.newaxis], self.centers_,
                                 self.width_, axis=1)

gauss_model = make_pipeline(GaussianFeatures(20),
                            LinearRegression())
gauss_model.fit(x[:, np.newaxis], y)
yfit = gauss_model.predict(xfit[:, np.newaxis])

plt.scatter(x, y)
plt.plot(xfit, yfit)
plt.xlim(0, 10);
```

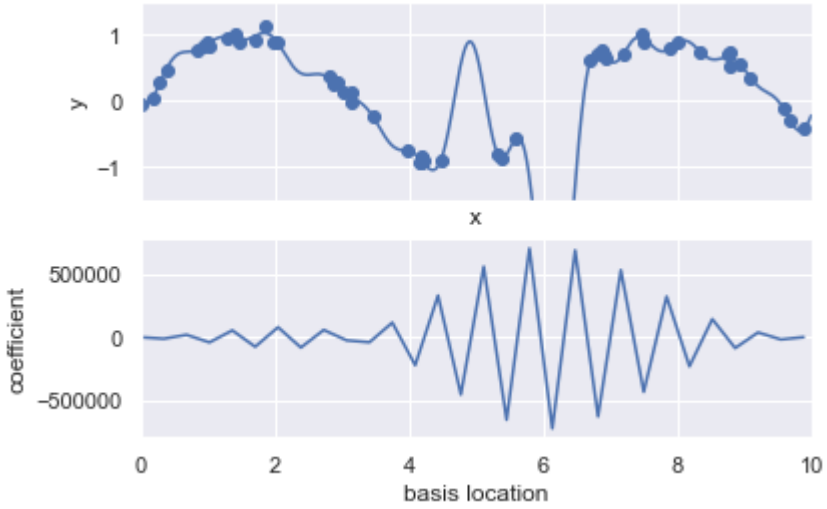


In [18]:
```
model = make_pipeline(GaussianFeatures(30),
                      LinearRegression())
model.fit(x[:, np.newaxis], y)

plt.scatter(x, y)
plt.plot(xfit, model.predict(xfit[:, np.newaxis]))

plt.xlim(0, 10)
plt.ylim(-1.5, 1.5);
```

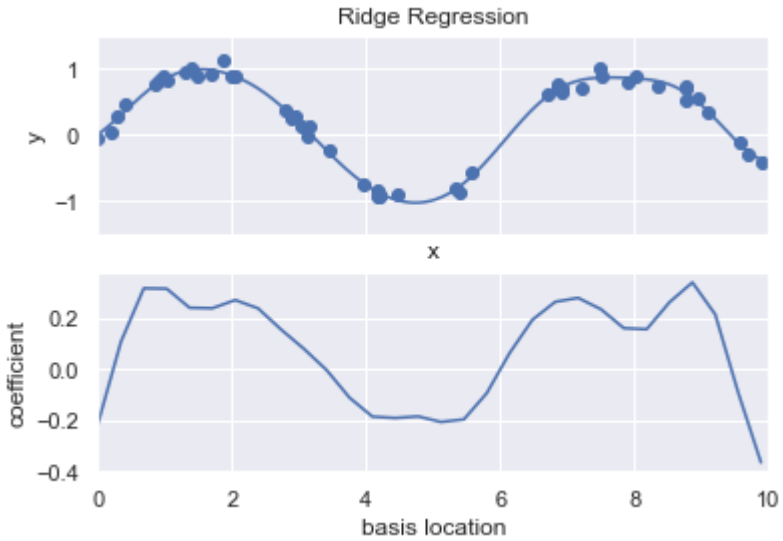


In [19]:
```
def basis_plot(model, title=None):
    fig, ax = plt.subplots(2, sharex=True)
    model.fit(x[:, np.newaxis], y)
    ax[0].scatter(x, y)
    ax[0].plot(xfit, model.predict(xfit[:, np.newaxis]))
    ax[0].set(xlabel='x', ylabel='y', ylim=(-1.5, 1.5))

    if title:
        ax[0].set_title(title)

    ax[1].plot(model.steps[0][1].centers_,
               model.steps[1][1].coef_)
    ax[1].set(xlabel='basis location',
              ylabel='coefficient',
              xlim=(0, 10))
```

```
model = make_pipeline(GaussianFeatures(30), LinearRegression())
basis_plot(model)
```

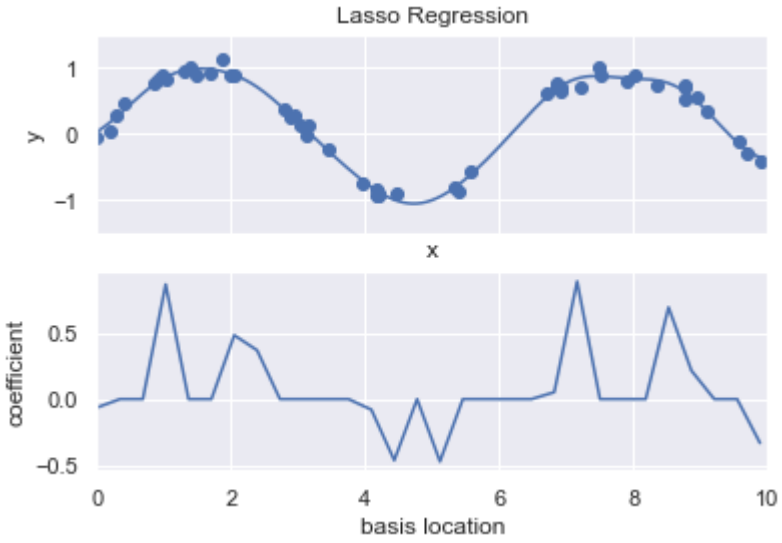


In [20]:
```
from sklearn.linear_model import Ridge
model = make_pipeline(GaussianFeatures(30), Ridge(alpha=0.1))
basis_plot(model, title='Ridge Regression')
```



In [21]:
```
from sklearn.linear_model import Lasso
model = make_pipeline(GaussianFeatures(30), Lasso(alpha=0.001))
basis_plot(model, title='Lasso Regression')
```

```
D:\anaconda\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:529: Convergen
ceWarning: Objective did not converge. You might want to increase the number of iteratio
ns. Duality gap: 0.002135815870486668, tolerance: 0.002065280097246271
  model = cd_fast.enet_coordinate_descent(
```



In [34]:
```
import pandas as pd
counts = pd.read_csv("FremontBridge.csv",index_col="Date",parse_dates=True)
weather = pd.read_csv("BicycleWeather.csv",index_col="DATE",parse_dates=True)

#Memuat dataset CSV
#Dataset saya dapatkan dari E-Learning
```

In [35]:
```
counts.head()
#Menampilkan isi dataset counts
```

Out[35]:

Date	Fremont Bridge East Sidewalk	Fremont Bridge West Sidewalk
2019-01-01 00:00:00	0.0	9.0
2019-01-01 01:00:00	2.0	22.0
2019-01-01 02:00:00	1.0	11.0
2019-01-01 03:00:00	1.0	2.0
2019-01-01 04:00:00	2.0	1.0

In [36]:
```
weather.head()
#Menampilkan dataset weather
```

Out[36]:

DATE	STATION	STATION_NAME	PRCP	SNWD	SNOW	TMAX	TMIN	AWND	WDF2	WI
2012-01-01	GHCND:USW00024233	SEATTLE TACOMA INTERNATIONAL AIRPORT WA US	0	0	0	128	50	47	100	
2012-01-02	GHCND:USW00024233	SEATTLE TACOMA INTERNATIONAL AIRPORT WA US	109	0	0	106	28	45	180	

DATE	STATION	STATION_NAME	PRCP	SNWD	SNOW	TMAX	TMIN	AWND	WDF2	WI
2012-01-03	GHCND:USW00024233	SEATTLE TACOMA INTERNATIONAL AIRPORT WA US	8	0	0	117	72	23	180	
2012-01-04	GHCND:USW00024233	SEATTLE TACOMA INTERNATIONAL AIRPORT WA US	203	0	0	122	56	47	180	
2012-01-05	GHCND:USW00024233	SEATTLE TACOMA INTERNATIONAL AIRPORT WA US	13	0	0	89	28	61	200	

5 rows × 25 columns

In [37]:
```
daily = counts.resample('d').sum()
daily['Total'] = daily.sum(axis=1)
daily = daily[['Total']]
```

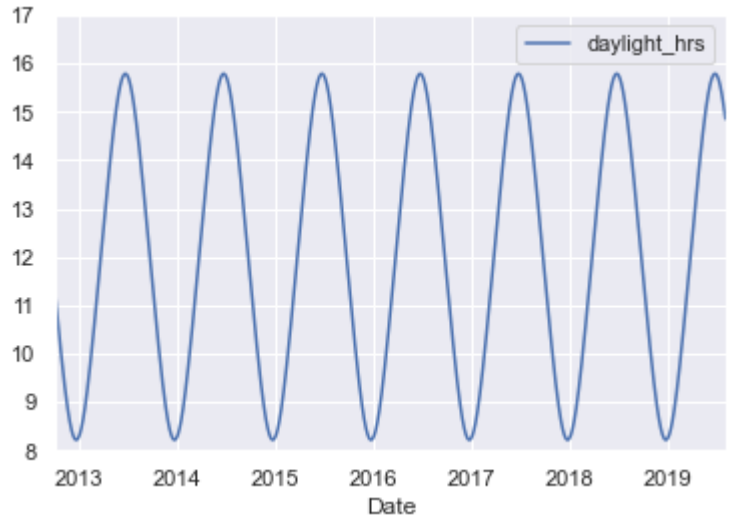
In [38]:
```
days = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
for i in range(7):
    daily[days[i]] = (daily.index.dayofweek == i).astype(float)
```

In [39]:
```
from pandas.tseries.holiday import USFederalHolidayCalendar
cal = USFederalHolidayCalendar()
holidays = cal.holidays('2012', '2016')
daily = daily.join(pd.Series(1, index=holidays, name='holiday'))
daily['holiday'].fillna(0, inplace=True)
```

In [40]:
```
def hours_of_daylight(date, axis=23.44, latitude=47.61):
    days = (date - pd.datetime(2000, 12, 21)).days
    m = (1. - np.tan(np.radians(latitude))
         * np.tan(np.radians(axis) * np.cos(days * 2 * np.pi / 365.25)))
    return 24. * np.degrees(np.arccos(1 - np.clip(m, 0, 2))) / 180.

daily['daylight_hrs'] = list(map(hours_of_daylight, daily.index))
daily[['daylight_hrs']].plot()
plt.ylim(8, 17)
```

```
<ipython-input-40-f555813b579d>:2: FutureWarning: The pandas.datetime class is deprecate
d and will be removed from pandas in a future version. Import from datetime module inste
ad.
  days = (date - pd.datetime(2000, 12, 21)).days
```

Out[40]: (8.0, 17.0)



In [41]:
```
weather['TMIN'] /= 10
weather['TMAX'] /= 10
weather['Temp (C)'] = 0.5 * (weather['TMIN'] + weather['TMAX'])

weather['PRCP'] /= 254
weather['dry day'] = (weather['PRCP'] == 0).astype(int)

daily = daily.join(weather[['PRCP', 'Temp (C)', 'dry day']])
```

In [42]:
```
daily['annual'] = (daily.index - daily.index[0]).days / 365.
```

In [43]:
```
daily.head()
```

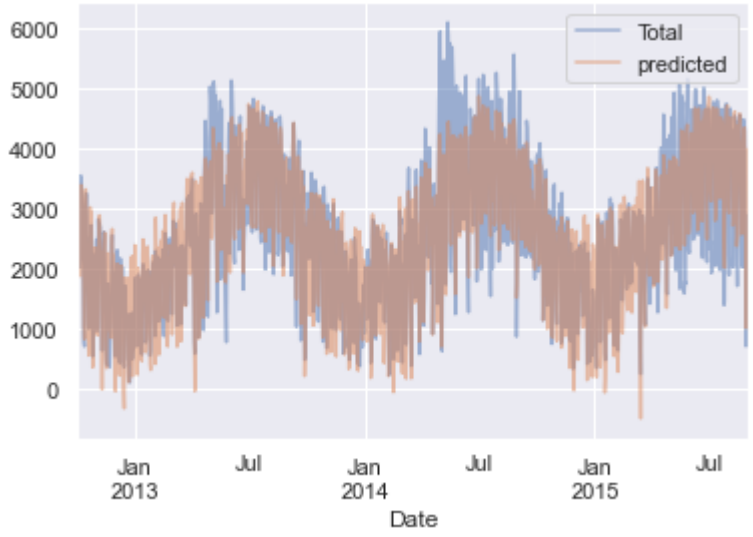
Out[43]:

Date	Total	Mon	Tue	Wed	Thu	Fri	Sat	Sun	holiday	daylight_hrs	PRCP	Temp (C)	dry day	annual
2012-10-03	3521.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	11.277359	0.0	13.35	1.0	0.000000
2012-10-04	3475.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	11.219142	0.0	13.60	1.0	0.002740
2012-10-05	3148.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	11.161038	0.0	15.30	1.0	0.005479
2012-10-06	2006.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	11.103056	0.0	15.85	1.0	0.008219
2012-10-07	2142.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	11.045208	0.0	15.85	1.0	0.010959

In [44]:
```
daily.dropna(axis=0, how='any', inplace=True)

column_names = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun', 'holiday',
                'daylight_hrs', 'PRCP', 'dry day', 'Temp (C)', 'annual']
X = daily[column_names]
y = daily['Total']

model = LinearRegression(fit_intercept=False)
```

```
model.fit(X, y)
daily['predicted'] = model.predict(X)
```

In [45]:
```
daily[['Total', 'predicted']].plot(alpha=0.5);
```



In [46]:
```
params = pd.Series(model.coef_, index=X.columns)
params
```

Out[46]:
```
Mon              504.882756
Tue              610.233936
Wed              592.673642
Thu              482.358115
Fri              177.980345
Sat            -1103.301710
Sun            -1133.567246
holiday        -1187.401381
daylight_hrs     128.851511
PRCP            -664.834882
dry day          547.698592
Temp (C)          65.162791
annual            26.942713
dtype: float64
```

In [47]:
```
from sklearn.utils import resample
np.random.seed(1)
err = np.std([model.fit(*resample(X, y)).coef_
              for i in range(1000)], 0)
```

In [48]:
```
print(pd.DataFrame({'effect': params.round(0),
                    'error': err.round(0)}))
```

```
              effect  error
Mon            505.0   86.0
Tue            610.0   83.0
Wed            593.0   83.0
Thu            482.0   85.0
Fri            178.0   81.0
Sat          -1103.0   80.0
Sun          -1134.0   83.0
holiday      -1187.0  163.0
daylight_hrs   129.0    9.0
PRCP          -665.0   62.0
dry day        548.0   33.0
```

```
Temp (C)        65.0    4.0
annual          27.0   18.0
```

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

```
Collecting graphviz
  Downloading graphviz-0.16-py2.py3-none-any.whl (19 kB)
Installing collected packages: graphviz
Successfully installed graphviz-0.16
```

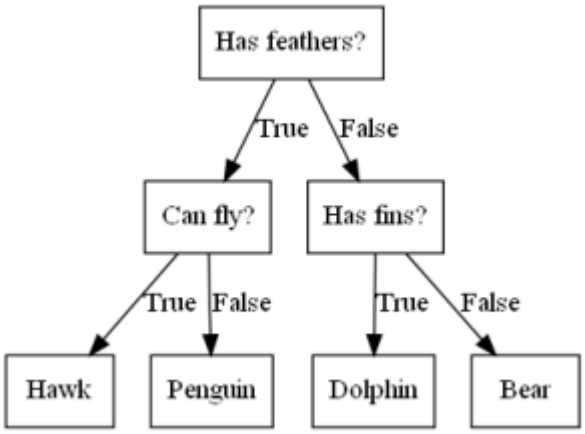
In [50]:
```
import os
os.environ["PATH"] +=os.pathsep + r'C:\Program Files (x86)\Graphviz\bin'

#Membuka path dari Graphviz yang berada pada storage kita
```

In [51]:
```
!pip install mglearn

#install mglearn
```

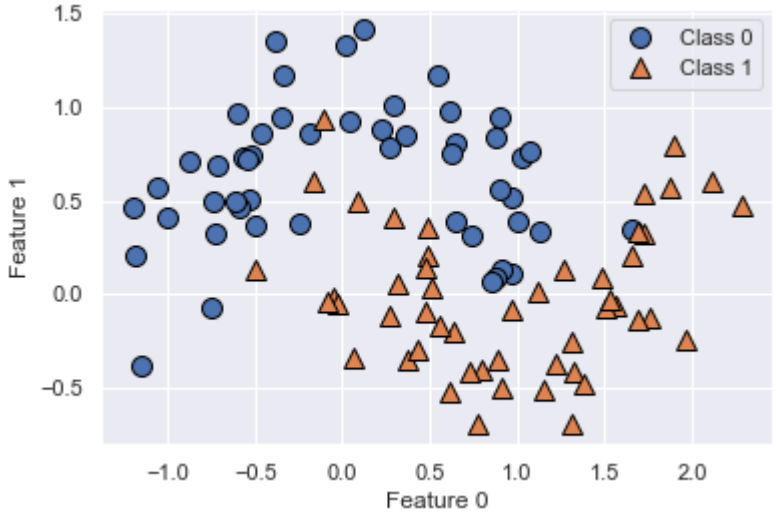
```
Collecting mglearn
  Downloading mglearn-0.1.9.tar.gz (540 kB)
Requirement already satisfied: numpy in d:\anaconda\lib\site-packages (from mglearn) (1.
19.2)
Requirement already satisfied: matplotlib in d:\anaconda\lib\site-packages (from mglear
n) (3.3.2)
Requirement already satisfied: scikit-learn in d:\anaconda\lib\site-packages (from mglea
rn) (0.23.2)
Requirement already satisfied: pandas in d:\anaconda\lib\site-packages (from mglearn)
(1.1.3)
Requirement already satisfied: pillow in d:\anaconda\lib\site-packages (from mglearn)
(8.0.1)
Requirement already satisfied: cycler in d:\anaconda\lib\site-packages (from mglearn)
(0.10.0)
Requirement already satisfied: imageio in d:\anaconda\lib\site-packages (from mglearn)
(2.9.0)
Requirement already satisfied: joblib in d:\anaconda\lib\site-packages (from mglearn)
(0.17.0)
Requirement already satisfied: kiwisolver>=1.0.1 in d:\anaconda\lib\site-packages (from
matplotlib->mglearn) (1.3.0)
Requirement already satisfied: certifi>=2020.06.20 in d:\anaconda\lib\site-packages (fro
m matplotlib->mglearn) (2020.6.20)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in d:\anaconda\l
ib\site-packages (from matplotlib->mglearn) (2.4.7)
Requirement already satisfied: python-dateutil>=2.1 in d:\anaconda\lib\site-packages (fr
om matplotlib->mglearn) (2.8.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in d:\anaconda\lib\site-packages (fr
om scikit-learn->mglearn) (2.1.0)
Requirement already satisfied: scipy>=0.19.1 in d:\anaconda\lib\site-packages (from scik
it-learn->mglearn) (1.5.2)
Requirement already satisfied: pytz>=2017.2 in d:\anaconda\lib\site-packages (from panda
s->mglearn) (2020.1)
Requirement already satisfied: six in d:\anaconda\lib\site-packages (from cycler->mglear
n) (1.15.0)
Building wheels for collected packages: mglearn
  Building wheel for mglearn (setup.py): started
  Building wheel for mglearn (setup.py): finished with status 'done'
  Created wheel for mglearn: filename=mglearn-0.1.9-py2.py3-none-any.whl size=582645 sha
256=6d4c25fb0313998a9b0a074b5af43f9d4bbb4e8f3970099f6e7d6e6849270c48
  Stored in directory: c:\users\rizal\appdata\local\pip\cache\wheels\87\75\37\404e66d0c4
bad150f101c9a0914b11a8eccc2681559936e7f7
Successfully built mglearn
Installing collected packages: mglearn
Successfully installed mglearn-0.1.9
```

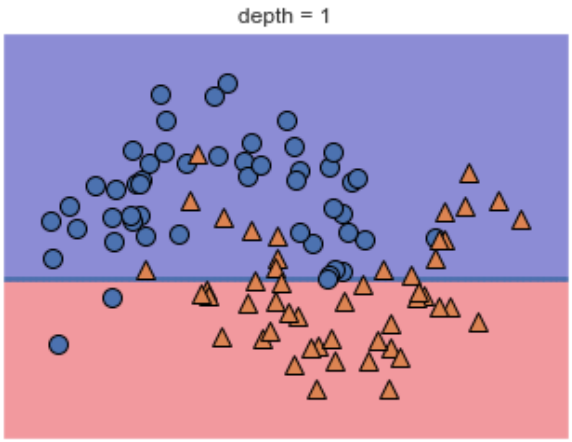
In [66]:
```
import sklearn.datasets
import mglearn
import matplotlib.pyplot as plt
from matplotlib import *
```

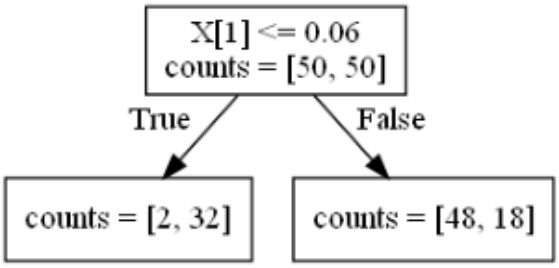
In [67]:
```
mglearn.plots.plot_animal_tree()
```

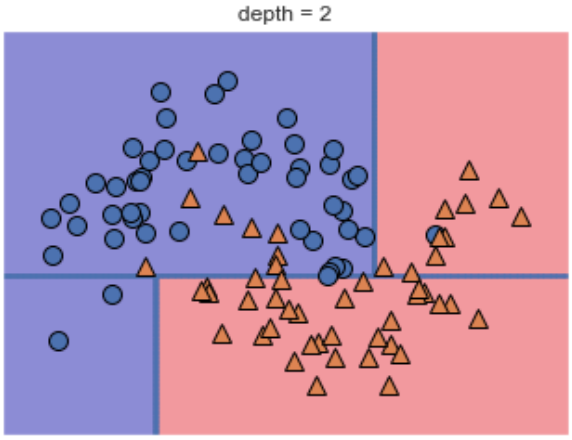


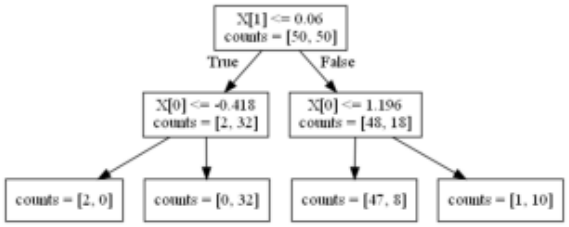
In [68]:
```
mglearn.plots.plot_tree_progressive()
```

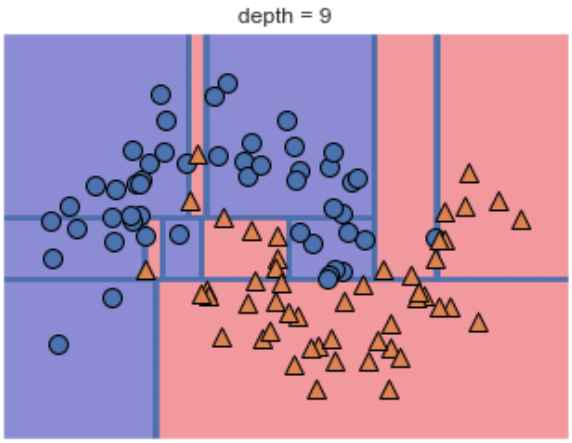


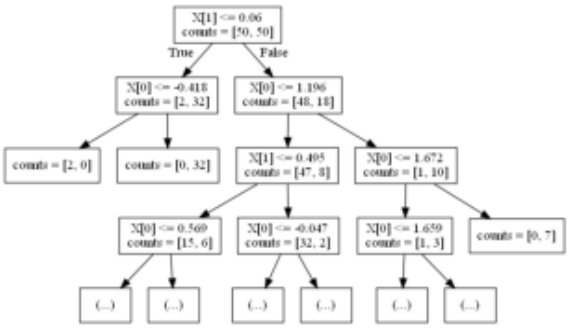












In [69]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
```

In [70]:
```
cancer = load_breast_cancer()
x_train,x_test,y_train,y_test = train_test_split(
cancer.data,cancer.target,stratify = cancer.target,random_state=42)
tree = DecisionTreeClassifier(random_state=0)
tree.fit(x_train,y_train)
print("Accuracy on training set : {:.3f}".format(tree.score(x_train,y_train)))
print("Accuracu on test set: {:.3f}".format(tree.score(x_test,y_test)))
```

```
Accuracy on training set : 1.000
Accuracu on test set: 0.937
```

In [71]:
```
tree = DecisionTreeClassifier(max_depth=4, random_state=0)
tree.fit(x_train,y_train)

print("Accuracy on training set : {:.3f}".format(tree.score(x_train,y_train)))
print("Accuracu on test set: {:.3f}".format(tree.score(x_test,y_test)))
```

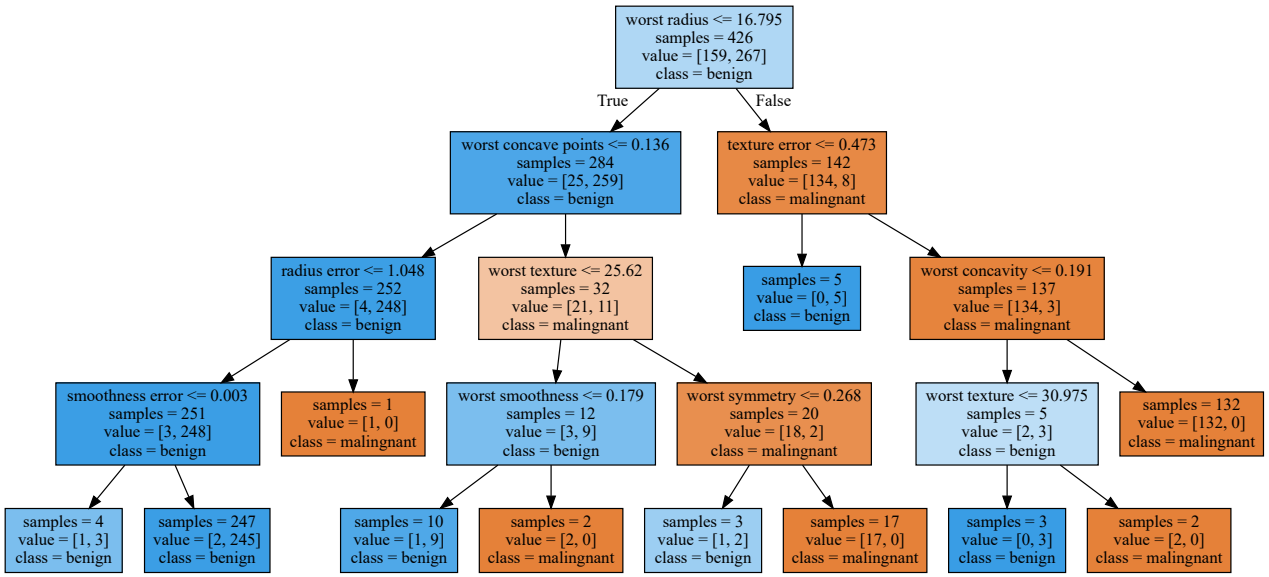
```
Accuracy on training set : 0.988
Accuracu on test set: 0.951
```

In [72]:
```
from sklearn.tree import export_graphviz
export_graphviz(tree,out_file="tree.dot",class_names=["malingnant","benign"],feature_na
```

In [73]:
```
import graphviz

with open("tree.dot") as f:
```

```
    dot_graph = f.read()
graphviz.Source(dot_graph)
```

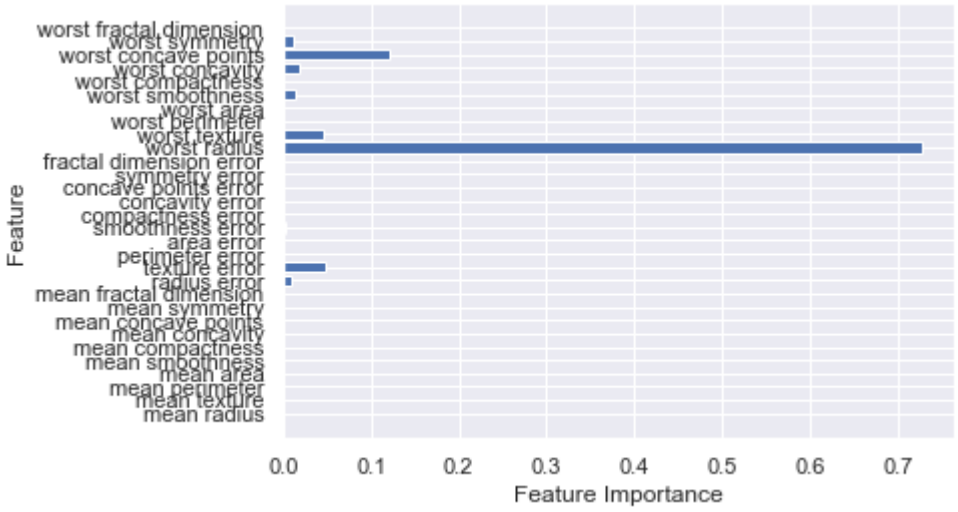
Out[73]:



In [74]:
```
print("Feature importances : \n{}".format(tree.feature_importances_))
```

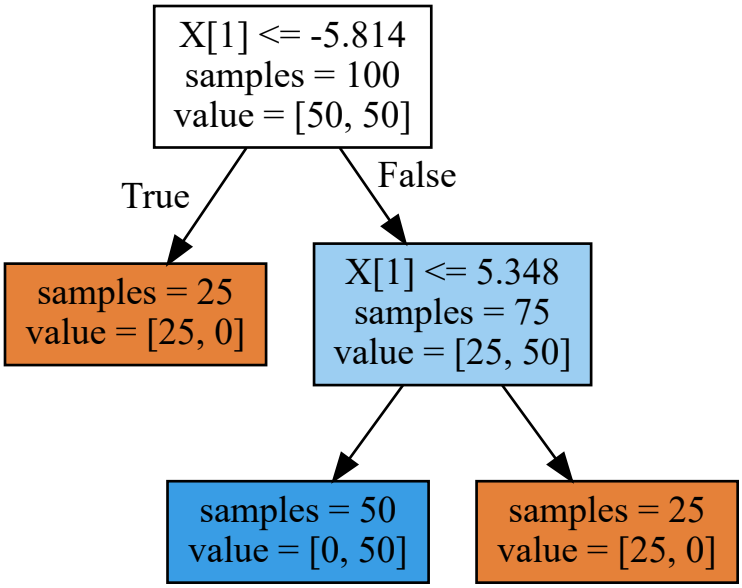
```
Feature importances :
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.01019737 0.04839825
 0.         0.         0.0024156  0.         0.         0.
 0.         0.         0.72682851 0.0458159  0.         0.
 0.0141577  0.         0.018188   0.1221132  0.01188548 0.        ]
```

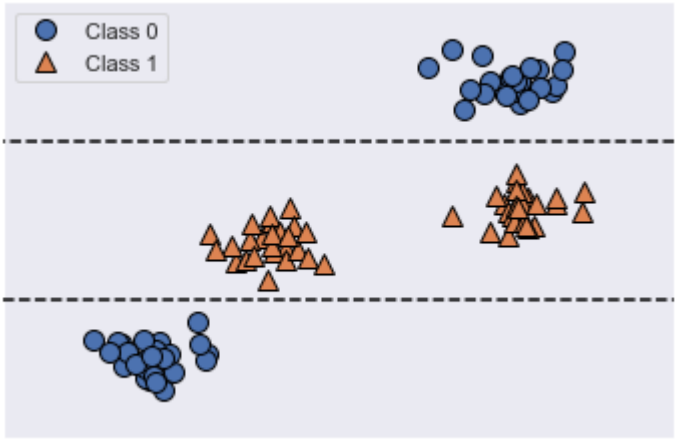
In [78]:
```
def plot_feature_importances_cancer(model):
    n_features = cancer.data.shape[1]
    plt.barh(range(n_features), model.feature_importances_,align='center')
    plt.yticks(np.arange(n_features),cancer.feature_names)
    plt.xlabel("Feature Importance")
    plt.ylabel("Feature")

plot_feature_importances_cancer(tree)
```



In [79]:
```
tree = mglearn.plots.plot_tree_not_monotone()
display(tree)
```

```
Feature importances: [0. 1.]
```

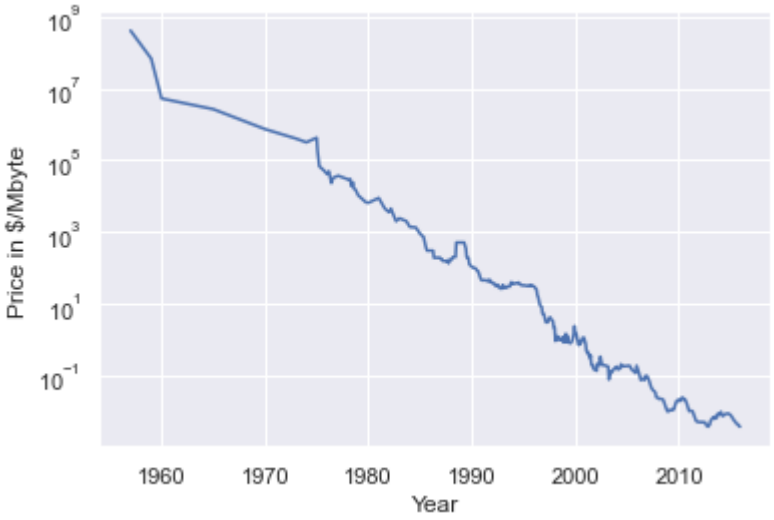




In [83]:
```
ram_prices = pd.read_csv(os.path.join(mglearn.datasets.DATA_PATH,"ram_price.csv"))

#Membaca dataset ram_price.csv
#dataset CSV saya dapatkan dari mglearn dataset
```

In [84]:
```
plt.semilogy(ram_prices.date,ram_prices.price)
plt.xlabel("Year")
plt.ylabel("Price in $/Mbyte")
```

Out[84]:
```
Text(0, 0.5, 'Price in $/Mbyte')
```

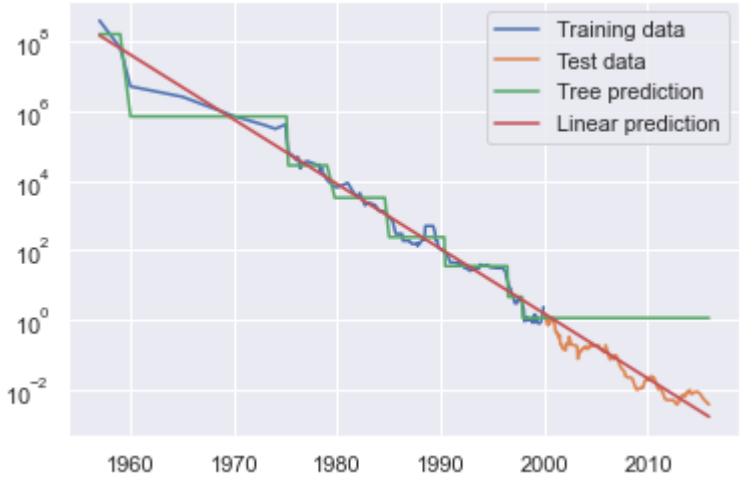


In [88]:
```
from sklearn.tree import DecisionTreeRegressor

data_train = ram_prices[ram_prices.date < 2000 ]
data_test = ram_prices[ram_prices.date >= 2000 ]


X_train = data_train.date[:, np.newaxis]
y_train = np.log(data_train.price)

tree = DecisionTreeRegressor(max_depth = 3 ).fit(X_train, y_train)
linear_reg = LinearRegression().fit(X_train, y_train)

X_all = ram_prices.date[:, np.newaxis]

pred_tree = tree.predict(X_all)
pred_lr = linear_reg.predict(X_all)

price_tree = np.exp(pred_tree)
price_lr = np.exp(pred_lr)
```

```
<ipython-input-88-798c1b514e09>:7: FutureWarning: Support for multi-dimensional indexing
(e.g. `obj[:, None]`) is deprecated and will be removed in a future version.  Convert to
a numpy array before indexing instead.
  X_train = data_train.date[:, np.newaxis]
<ipython-input-88-798c1b514e09>:13: FutureWarning: Support for multi-dimensional indexin
g (e.g. `obj[:, None]`) is deprecated and will be removed in a future version.  Convert
to a numpy array before indexing instead.
  X_all = ram_prices.date[:, np.newaxis]
```

In [89]:
```
plt.semilogy(data_train.date, data_train.price, label= "Training data" )
plt.semilogy(data_test.date, data_test.price, label= "Test data" )
plt.semilogy(ram_prices.date, price_tree, label= "Tree prediction" )
plt.semilogy(ram_prices.date, price_lr, label= "Linear prediction" )
plt.legend()
```

Out[89]:
```
<matplotlib.legend.Legend at 0x215e0a356a0>
```

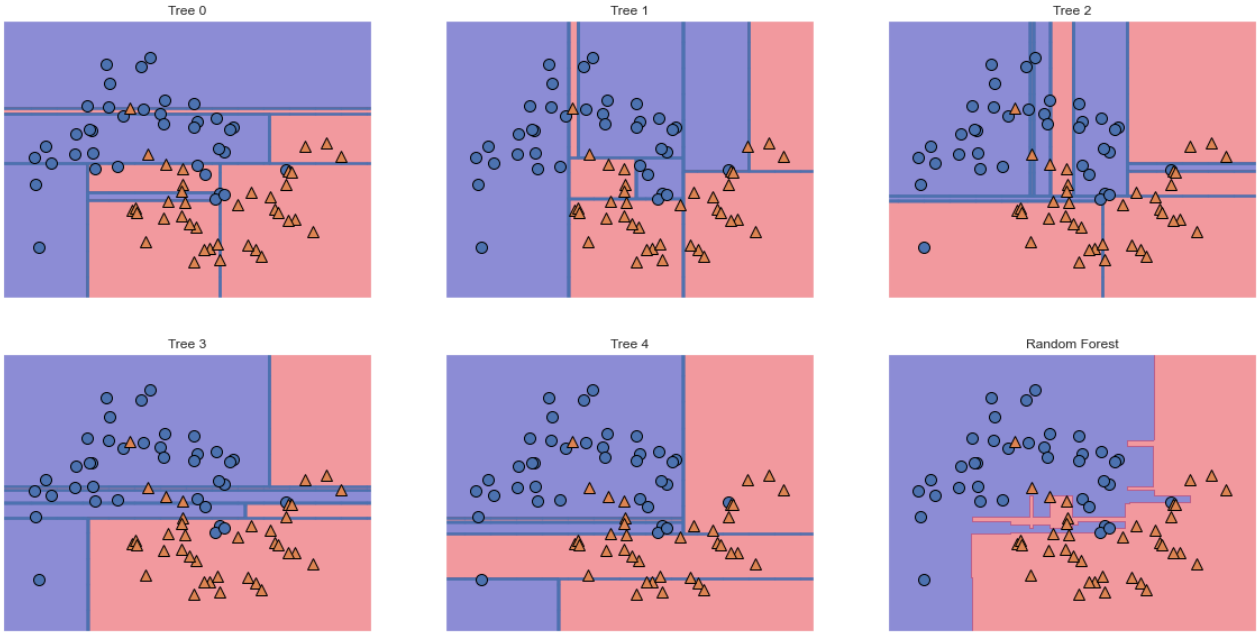


In [91]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_moons

x, y = make_moons(n_samples = 100, noise = 0.25, random_state = 3)
x_train, x_test, y_train, y_test = train_test_split(x, y, stratify = y, random_state =

forest = RandomForestClassifier(n_estimators = 5, random_state = 2)
forest.fit(x_train, y_train)
```

Out[91]:
```
RandomForestClassifier(n_estimators=5, random_state=2)
```

In [92]:
```
fig, axes = plt.subplots(2, 3, figsize=(20,10))
for i, (ax, tree) in enumerate(zip(axes.ravel(), forest.estimators_)):
    ax.set_title("Tree {}".format(i))
    mglearn.plots.plot_tree_partition(x_train, y_train, tree, ax=ax)

mglearn.plots.plot_2d_separator(forest, x_train, fill = True, ax=axes[-1, -1], alpha =
axes[-1, -1].set_title("Random Forest")
mglearn.discrete_scatter(x_train[:, 0], x_train[:, 1], y_train)
```

Out[92]:
```
[<matplotlib.lines.Line2D at 0x215e05aae80>,
 <matplotlib.lines.Line2D at 0x215e05aa4c0>]
```



In [93]:
```
x_train, x_test, y_train, y_test = train_test_split(cancer.data, cancer.target, random_
forest = RandomForestClassifier(n_estimators = 100, random_state = 0)
forest.fit(x_train, y_train)
print("Accuracy on training set : {:.3}".format(forest.score(x_train, y_train)))
print("Accuracu on test set : {:.3}".format(forest.score(x_test, y_test)))
```

```
Accuracy on training set : 1.0
Accuracu on test set : 0.972
```

In [ ]:
```

```