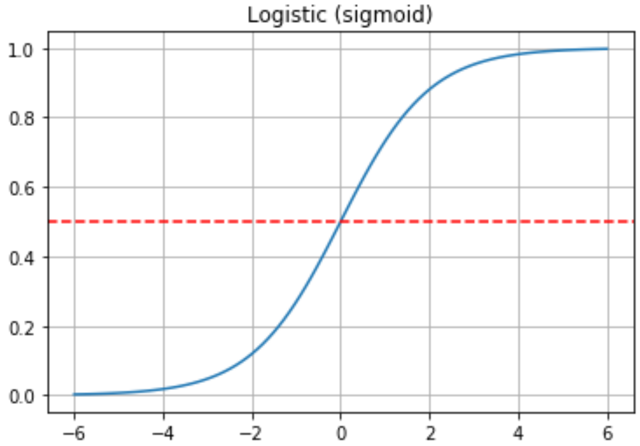
Logistic Regression

- Logistic regression is a linear model for classification rather than regression
- Sigmoid function: $p = \frac{1}{(1+e^{-y})}$
- Linear regression equation: $\hat{y}(w, x) = w_0 + w_1 x_1 + w_2 x_2 + ... + w_p x_p$
- Applying sigmoid function: $p = \frac{1}{(1+e^{-(w_0+w_1 x_1+w_2 x_2+...+w_p x_p)})}$
- Types of Logistic Regression
  - Binary Logistic Regression: target variable has only two possible outcomes
  - Multinomial Logistic Regression: Target variable has three or more nominal categories
  - Ordinal Logistic Regression: Target variable has three of more ordinal categories

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

np.set_printoptions(precision=4, suppress=True)
```

In [2]:
```
def logistic(x):
    return 1 / (1 + np.exp(-x))

x = np.linspace(-6, 6, 100)
plt.plot(x, logistic(x))
plt.axhline(.5, c='r', ls='--')
plt.grid(True)
plt.title('Logistic (sigmoid)');
```



In [3]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression

from sklearn import datasets

from sklearn import metrics
```

Logistic versus Linear Regression

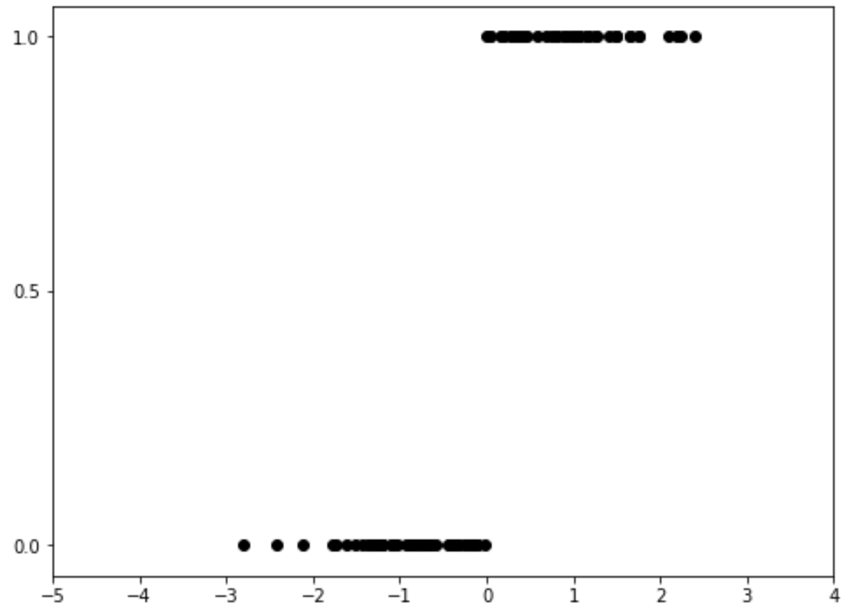
In [4]:
```
# Generate a sample dataset - a straight line with some Gaussian noise:

np.random.seed(123)

xmin, xmax = -5, 5
n_samples = 100

X = np.random.normal(size=n_samples)
y = (X > 0).astype(np.float)

# Noise commented out

# X[X > 0] *= 4
# X += .3 * np.random.normal(size=n_samples)

X = X[:, np.newaxis]  # or X.reshape(-1,1)
X[:10]
```

Out[4]:
```
array([[-1.0856],
       [ 0.9973],
       [ 0.283 ],
       [-1.5063],
       [-0.5786],
       [ 1.6514],
       [-2.4267],
       [-0.4289],
       [ 1.2659],
       [-0.8667]])
```

In [5]:
```
y[:10]
```

Out[5]:
```
array([0., 1., 1., 0., 0., 1., 0., 0., 1., 0.])
```

In [6]:
```
plt.figure(figsize=(8, 6))

plt.scatter(X.ravel(), y, color='k')
plt.xticks(range(-5, 5))
plt.yticks([0, 0.5, 1]);
```



In [7]:
```
# Fit the Logistic classifier and the Linear regression

clf = LogisticRegression(C=1e5, solver='lbfgs')
clf.fit(X, y)

print(clf.coef_)
print(clf.intercept_)
```

```
[[230.1743]]
[1.0355]
```

In [8]:
```
ols = LinearRegression()
ols.fit(X, y)

print(ols.coef_)
print(ols.intercept_)
```

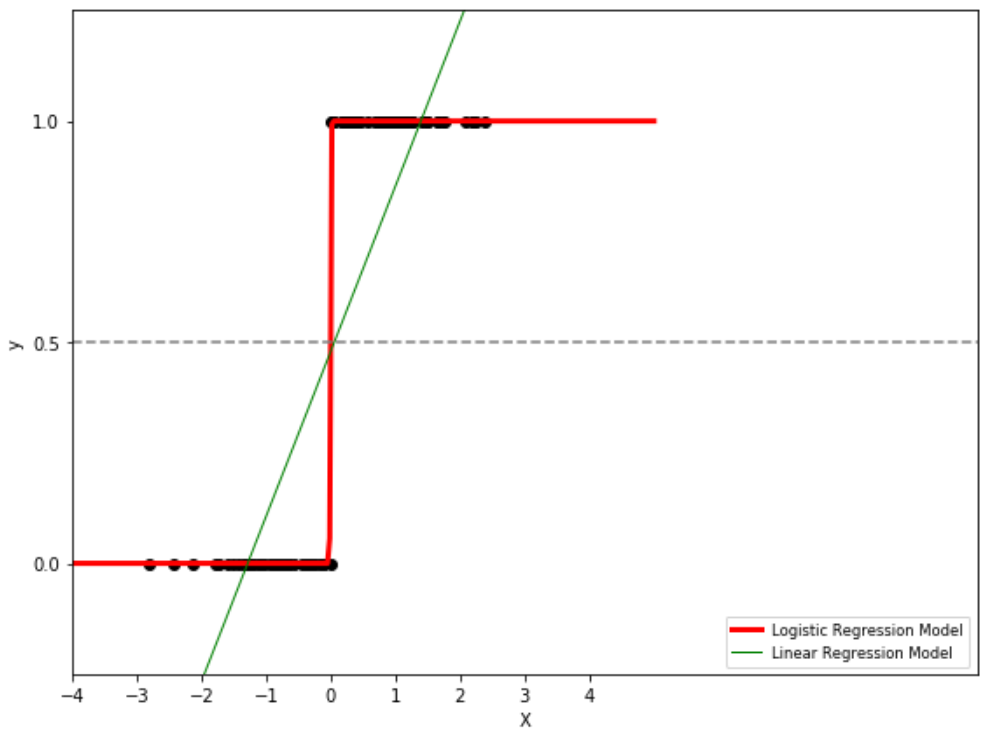
```
[0.3731]
0.47988653768628625

/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklearn/linear_model/base.py:485:
RuntimeWarning: internal gelsd driver lwork query error, required iwork dimension not returned. This is likely
the result of LAPACK bug 0038, fixed in LAPACK 3.2.2 (released July 21, 2010). Falling back to 'gelss' driver.
  linalg.lstsq(X, y)
```

In [9]:
```python
def model(x):
    return 1 / (1 + np.exp(-x))

X_test = np.linspace(-5, 5, 300)

loss = model(X_test * clf.coef_ + clf.intercept_).ravel()

loss
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/ipykernel_launcher.py:2: RuntimeW
arning: overflow encountered in exp
```

Out[9]:
```
array([0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    ,
       0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    ,
       0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    ,
       0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    ,
       0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    ,
       0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    ,
       0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    ,
       0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    ,
       0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    ,
       0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    ,
       0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    ,
       0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    ,
       0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    ,
       0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    ,
       0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    ,
       0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    ,
       0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    ,
       0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    ,
       0.    , 0.    , 0.    , 0.    , 0.    , 0.0566, 0.9925, 1.    ,
       1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    ,
       1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    ,
       1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    ,
       1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    ,
       1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    ,
       1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    ,
       1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    ,
       1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    ,
       1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    ,
       1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    ,
       1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    ,
       1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    ,
       1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    ,
```

In [10]:
```
plt.figure(figsize=(8, 6))

plt.scatter(X.ravel(), y, color='k')

plt.plot(X_test, loss, color='r', linewidth=3)

plt.plot(X_test, ols.coef_ * X_test + ols.intercept_, linewidth=1, color='g')

plt.axhline(.5, c='gray', ls='--')

plt.ylabel('y')
plt.xlabel('X')
plt.xticks(range(-5, 5))
plt.yticks([0, 0.5, 1])
plt.ylim(-.25, 1.25)
plt.xlim(-4, 10)
plt.legend(('Logistic Regression Model', 'Linear Regression Model'),
           loc="lower right", fontsize='small')
plt.tight_layout()
```



In [ ]:

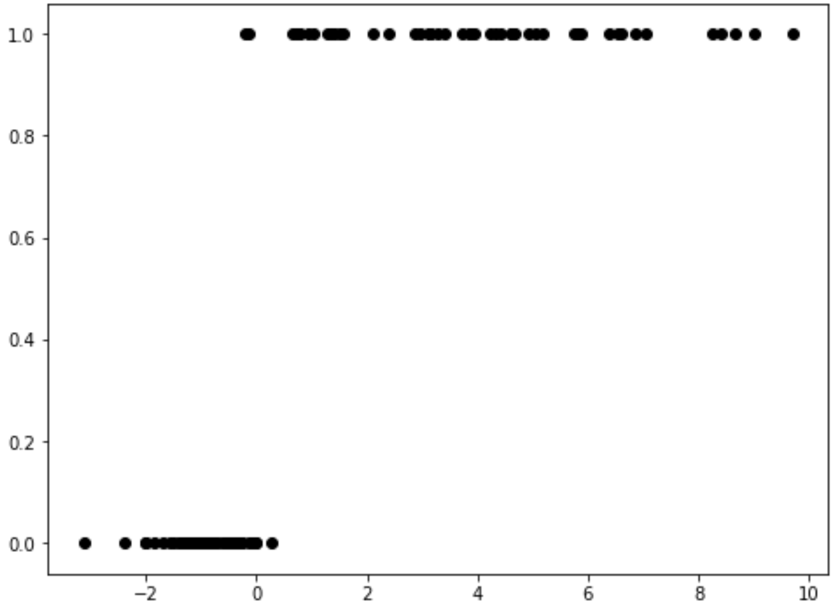
In [11]:
```
# Generate a sample dataset - a straight line with some Gaussian noise:

np.random.seed(123)

xmin, xmax = -5, 5
n_samples = 100

X = np.random.normal(size=n_samples)
y = (X > 0).astype(np.float)

# With Noise

X[X > 0] *= 4
X += .3 * np.random.normal(size=n_samples)

np.min(X), np.max(X)
```

Out[11]:
```
(-3.1286020392469336, 9.706342332311573)
```

In [12]:
```
X = X[:, np.newaxis]  # or X.reshape(-1,1)
X[:10]
```

Out[12]:
```
array([[-0.893 ],
       [ 3.396 ],
       [ 1.3456],
       [-0.7268],
       [-0.586 ],
       [ 6.616 ],
       [-2.3728],
       [-0.9875],
       [ 5.1916],
       [-1.3484]])
```

In [13]:
```
plt.figure(figsize=(8, 6))

plt.scatter(X.ravel(), y, color='k');
```



In [14]:
```
# Fit the Logistic classifier and the Linear regression

clf = LogisticRegression(C=1e5, solver='lbfgs')
clf.fit(X, y)

print(clf.coef_)
print(clf.intercept_)
```

```
[[5.4187]]
[-1.1437]
```

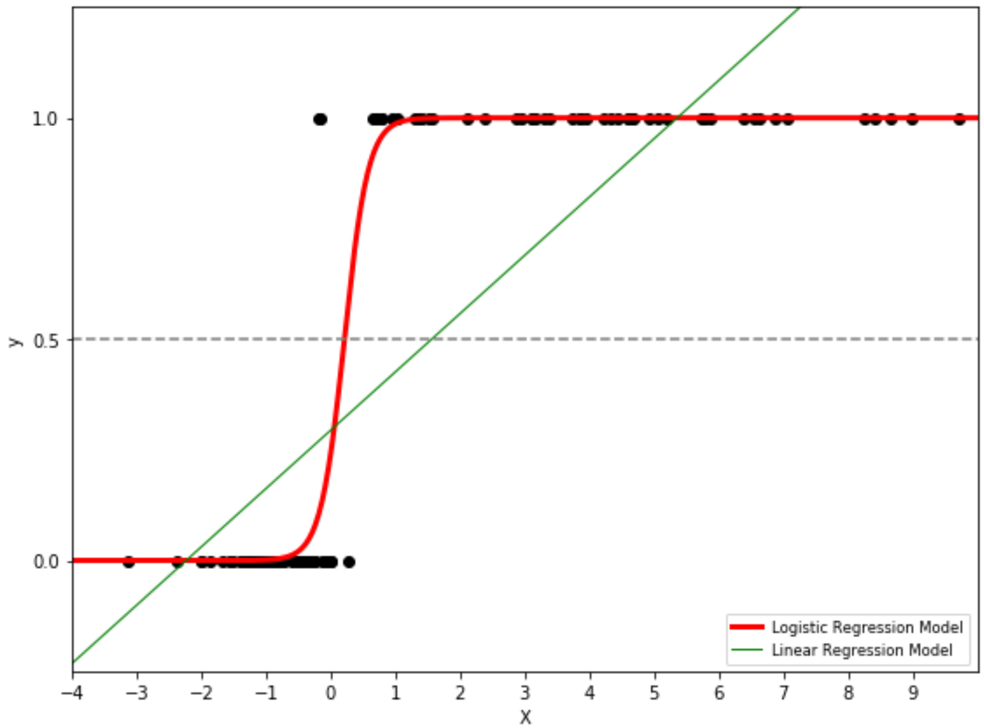
In [15]:
```
ols = LinearRegression()
ols.fit(X, y)

print(ols.coef_)
print(ols.intercept_)
```

```
[0.1317]
0.29434790257445975
```

In [16]:
```
def model(x):
    return 1 / (1 + np.exp(-x))

X_test = np.linspace(-5, 10, 300)

loss = model(X_test * clf.coef_ + clf.intercept_).ravel()

loss
```

Out[16]:
```
array([0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    ,
       0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    ,
       0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    ,
       0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    ,
       0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    ,
       0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    ,
       0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    ,
       0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    ,
       0.    , 0.    , 0.    , 0.    , 0.0001, 0.0001, 0.0001, 0.0001,
       0.0002, 0.0002, 0.0003, 0.0004, 0.0005, 0.0007, 0.0009, 0.0012,
       0.0015, 0.002 , 0.0026, 0.0034, 0.0045, 0.0059, 0.0077, 0.0101,
       0.0132, 0.0172, 0.0225, 0.0293, 0.0381, 0.0495, 0.0639, 0.0822,
       0.1052, 0.1337, 0.1684, 0.21  , 0.2586, 0.314 , 0.3753, 0.4409,
       0.5086, 0.5759, 0.6406, 0.7005, 0.7543, 0.8011, 0.8409, 0.874 ,
       0.9011, 0.9228, 0.9401, 0.9537, 0.9643, 0.9726, 0.979 , 0.9839,
       0.9877, 0.9906, 0.9928, 0.9945, 0.9958, 0.9968, 0.9976, 0.9981,
       0.9986, 0.9989, 0.9992, 0.9994, 0.9995, 0.9996, 0.9997, 0.9998,
       0.9998, 0.9999, 0.9999, 0.9999, 0.9999, 1.    , 1.    , 1.    ,
       1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    ,
       1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    ,
       1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    ,
       1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    ,
       1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    ,
       1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    ,
       1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    ,
       1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    ,
       1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    ,
       1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    ,
       1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    ,
       1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    ,
       1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    ,
       1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    ,
       1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    ,
       1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    ,
       1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    , 1.    ,
```

In [17]:
```
plt.figure(figsize=(8, 6))

plt.scatter(X.ravel(), y, color='k')

plt.plot(X_test, loss, color='r', linewidth=3)

plt.plot(X_test, ols.coef_ * X_test + ols.intercept_,
         linewidth=1, color='g')

plt.axhline(.5, c='gray', ls='--')

plt.ylabel('y')
plt.xlabel('X')
plt.xticks(range(-5, 10))
plt.yticks([0, 0.5, 1])
plt.ylim(-.25, 1.25)
plt.xlim(-4, 10)
plt.legend(('Logistic Regression Model', 'Linear Regression Model'),
           loc="lower right", fontsize='small')
plt.tight_layout()
```



In [ ]:

```

```

Iris Dataset

In [18]:

```
iris = datasets.load_iris()
```

In [19]:
```
print(iris.DESCR)
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field and
is referenced frequently to this day.  (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers to a
```

In [20]:
```
X = iris.data[:, :2]  # Using the first two features.
y = iris.target
```

In [21]:
```
X[:5]
```

Out[21]:
```
array([[5.1, 3.5],
       [4.9, 3. ],
       [4.7, 3.2],
       [4.6, 3.1],
       [5. , 3.6]])
```

In [22]:
```
np.unique(y)
```

Out[22]:
```
array([0, 1, 2])
```

Binary Logistic Regression

In [23]:
```
# Use only the first two classes

X = X[y != 2]
y = y[y != 2]
```

In [24]:
```
logreg = LogisticRegression()

logreg.fit(X, y);
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:
432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warnin
g.
  FutureWarning)
```

In [25]:
```
logreg.score(X, y)
```

Out[25]:
```
0.99
```

In [26]:
```
y_pred = logreg.predict(X)
y_pred
```

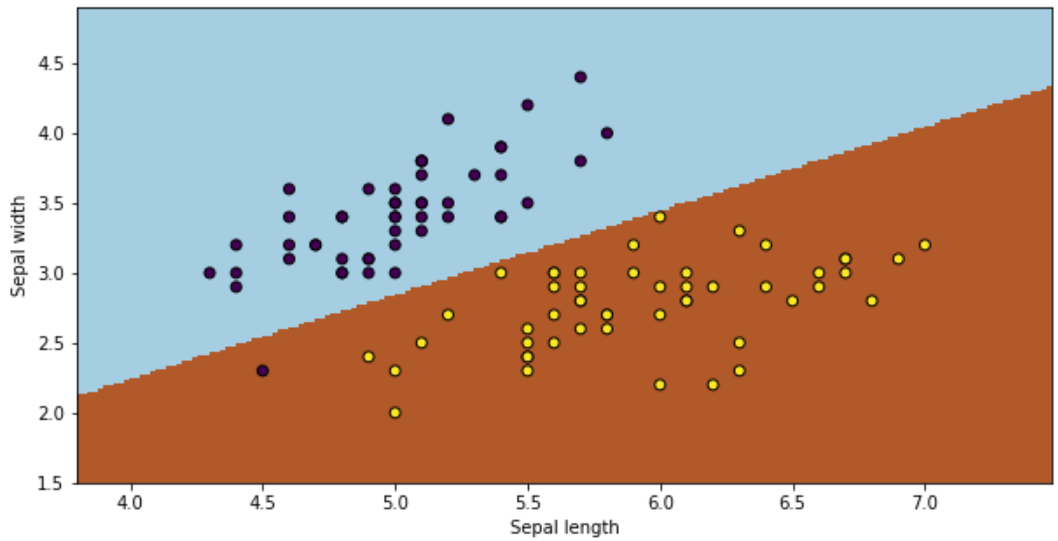
Out[26]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

In [27]:
```
metrics.confusion_matrix(y, y_pred)
```

Out[27]:
```
array([[49,  1],
       [ 0, 50]])
```

In [28]:
```
# Plot the decision boundary. For that, we will assign a color to each
# point in the mesh [x_min, x_max] by [y_min, y_max].

x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5

h = .02  # step size in the mesh

xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))

Z = logreg.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
```

In [29]:
```
plt.figure(1, figsize=(10, 5))
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)

# Plot also the training points
plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k')

plt.xlabel('Sepal length')
plt.ylabel('Sepal width')

plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max());
```



In [ ]:

Multinomial Logistic Regression

In [30]:
```
# Using All the features
```

In [31]:
```
logreg = LogisticRegression(solver='lbfgs', multi_class='multinomial')
```

In [32]:
```
logreg.fit(iris.data, iris.target)
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:
757: ConvergenceWarning: lbfgs failed to converge. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
```

Out[32]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='multinomial',
          n_jobs=None, penalty='l2', random_state=None, solver='lbfgs',
          tol=0.0001, verbose=0, warm_start=False)
```

In [33]:
```
iris_probs = logreg.predict_proba(iris.data)
iris_probs[:5]
```

Out[33]:
```
array([[0.9818, 0.0182, 0.    ],
       [0.9717, 0.0283, 0.    ],
       [0.9854, 0.0146, 0.    ],
       [0.9763, 0.0237, 0.    ],
       [0.9854, 0.0146, 0.    ]])
```

In [34]:
```
iris_pred = logreg.predict(iris.data)
iris_pred[:5]
```

Out[34]:
```
array([0, 0, 0, 0, 0])
```

In [35]:
```
metrics.confusion_matrix(iris.target, iris_pred)
```

Out[35]:
```
array([[50,  0,  0],
       [ 0, 47,  3],
       [ 0,  1, 49]])
```

In [36]:
```
iris_pred_df = pd.DataFrame(iris_probs, columns=iris.target_names).round(4)
iris_pred_df['predicted_class'] = iris.target_names[iris_pred]
iris_pred_df['target_class'] = iris.target_names[iris.target]
iris_pred_df.sample(12)
```

Out[36]:

	setosa	versicolor	virginica	predicted_class	target_class
11	0.9754	0.0246	0.0000	setosa	setosa
64	0.0743	0.9152	0.0105	versicolor	versicolor
146	0.0002	0.2503	0.7495	virginica	virginica
111	0.0001	0.1369	0.8631	virginica	virginica
116	0.0001	0.1232	0.8767	virginica	virginica
68	0.0018	0.7993	0.1989	versicolor	versicolor
122	0.0000	0.0047	0.9953	virginica	virginica
139	0.0000	0.0934	0.9066	virginica	virginica
45	0.9739	0.0261	0.0000	setosa	setosa
34	0.9687	0.0313	0.0000	setosa	setosa
93	0.1217	0.8753	0.0031	versicolor	versicolor
133	0.0005	0.4759	0.5235	virginica	virginica

In [ ]:

In [37]:
```
logreg.score(iris.data, iris.target)
```

Out[37]: 0.9733333333333334

In [38]:
```
iris_pred_df[iris_pred != iris.target]
```

Out[38]:

	setosa	versicolor	virginica	predicted_class	target_class
70	0.0023	0.4404	0.5573	virginica	versicolor
77	0.0006	0.4811	0.5183	virginica	versicolor
83	0.0004	0.3496	0.6500	virginica	versicolor
106	0.0057	0.5119	0.4824	versicolor	virginica

In [39]:
```
print("Accuracy:",metrics.accuracy_score(iris.target, iris_pred))
```

```
Accuracy: 0.9733333333333334
```

In [ ]:
```

```

Model Validation

- Divide the dataset into a training set and a test set

In [40]:
```
from sklearn.model_selection import train_test_split
```

In [41]:
```
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.3, random_state=0)
```

In [42]:
```
X_train.shape, X_test.shape
```

Out[42]:
```
((105, 4), (45, 4))
```

In [43]:
```
logreg = LogisticRegression(solver='lbfgs', multi_class='multinomial')
```

In [44]:
```
logreg.fit(X_train, y_train)
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:
757: ConvergenceWarning: lbfgs failed to converge. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
```

Out[44]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='multinomial',
          n_jobs=None, penalty='l2', random_state=None, solver='lbfgs',
          tol=0.0001, verbose=0, warm_start=False)
```

In [45]:
```
y_pred = logreg.predict(X_test)
```

In [46]:
```
logreg.score(X_test, y_test)
```

Out[46]:
```
0.9777777777777777
```

In [47]:
```
metrics.confusion_matrix(y_test, y_pred)
```

Out[47]:
```
array([[16,  0,  0],
       [ 0, 17,  1],
       [ 0,  0, 11]])
```

In [48]:
```
metrics.accuracy_score(y_test, y_pred)
```

Out[48]:
```
0.9777777777777777
```

In [ ]:

Case Study - Predicting Credit Card Default

- [https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients (https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients)](https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients)
- [http://inseaddataanalytics.github.io/INSEADAnalytics/CourseSessions/ClassificationProcessCreditCardDefault.html (http://inseaddataanalytics.github.io/INSEADAnalytics/CourseSessions/ClassificationProcessCreditCardDefault.html)](http://inseaddataanalytics.github.io/INSEADAnalytics/CourseSessions/ClassificationProcessCreditCardDefault.html)

In [49]:
```
ccd = pd.read_csv('http://people.bu.edu/kalathur/datasets/credit_card_default.csv',
                  index_col="ID")
ccd.head()
```

Out[49]:

	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_1	PAY_2	PAY_3	PAY_4	PAY_5	...	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AM
ID																
1	20000	2	2	1	24	2	2	-1	-1	-2	...	0	0	0	0	[illegible]
2	120000	2	2	2	26	-1	2	0	0	0	...	3272	3455	3261	0	1
3	90000	2	2	2	34	0	0	0	0	0	...	14331	14948	15549	1518	1
4	50000	2	2	1	37	0	0	0	0	0	...	28314	28959	29547	2000	2
5	50000	1	2	1	57	-1	0	-1	0	0	...	20940	19146	19131	2000	36

5 rows × 24 columns

In [50]:
```
len(ccd)
```

Out[50]:
```
30000
```

In [51]:
```
ccd.rename(columns=lambda x: x.lower(), inplace=True)
ccd.rename(columns={'default payment next month':'default'}, inplace=True)

ccd.head().T
```

Out[51]:

ID	1	2	3	4	5
limit_bal	20000	120000	90000	50000	50000
sex	2	2	2	2	1
education	2	2	2	2	2
marriage	1	2	2	1	1
age	24	26	34	37	57
pay_1	2	-1	0	0	-1
pay_2	2	2	0	0	0
pay_3	-1	0	0	0	-1
pay_4	-1	0	0	0	0
pay_5	-2	0	0	0	0
pay_6	-2	2	0	0	0
bill_amt1	3913	2682	29239	46990	8617
bill_amt2	3102	1725	14027	48233	5670
bill_amt3	689	2682	13559	49291	35835
bill_amt4	0	3272	14331	28314	20940
bill_amt5	0	3455	14948	28959	19146
bill_amt6	0	3261	15549	29547	19131
pay_amt1	0	0	1518	2000	2000
pay_amt2	689	1000	1500	2019	36681
pay_amt3	0	1000	1000	1200	10000
pay_amt4	0	1000	1000	1100	9000
pay_amt5	0	0	1000	1069	689
pay_amt6	0	2000	5000	1000	679
default	1	1	0	0	0

In [52]:
```
# getting the groups of features

bill_amt_features = ['bill_amt'+ str(i) for i in range(1,7)]

pay_amt_features = ['pay_amt'+ str(i) for i in range(1,7)]

numerical_features = ['limit_bal','age'] + bill_amt_features + pay_amt_features
```

In [53]:
```
numerical_features
```

Out[53]:
```
['limit_bal',
 'age',
 'bill_amt1',
 'bill_amt2',
 'bill_amt3',
 'bill_amt4',
 'bill_amt5',
 'bill_amt6',
 'pay_amt1',
 'pay_amt2',
 'pay_amt3',
 'pay_amt4',
 'pay_amt5',
 'pay_amt6']
```

In [54]:
```
ccd.sex.unique()
```

Out[54]:
```
array([2, 1])
```

In [55]:
```
ccd.sex.value_counts()
```

Out[55]:
```
2    18112
1    11888
Name: sex, dtype: int64
```

In [56]:
```
ccd.education.unique()
```

Out[56]:
```
array([2, 1, 3, 5, 4, 6, 0])
```

In [57]:
```
ccd.education.value_counts()
```

Out[57]:
```
2    14030
1    10585
3     4917
5      280
4      123
6       51
0       14
Name: education, dtype: int64
```

In [58]:
```
# Creating  some binary features

ccd['male'] = (ccd['sex'] == 1).astype('int')

ccd['grad_school'] = (ccd['education'] == 1).astype('int')
ccd['university'] = (ccd['education'] == 2).astype('int')

ccd['married'] = (ccd['marriage'] == 1).astype('int')
```

In [59]:
```
ccd.head().T
```

Out[59]:

ID	1	2	3	4	5
limit_bal	20000	120000	90000	50000	50000
sex	2	2	2	2	1
education	2	2	2	2	2
marriage	1	2	2	1	1
age	24	26	34	37	57
pay_1	2	-1	0	0	-1
pay_2	2	2	0	0	0
pay_3	-1	0	0	0	-1
pay_4	-1	0	0	0	0
pay_5	-2	0	0	0	0
pay_6	-2	2	0	0	0
bill_amt1	3913	2682	29239	46990	8617
bill_amt2	3102	1725	14027	48233	5670
bill_amt3	689	2682	13559	49291	35835
bill_amt4	0	3272	14331	28314	20940
bill_amt5	0	3455	14948	28959	19146
bill_amt6	0	3261	15549	29547	19131
pay_amt1	0	0	1518	2000	2000
pay_amt2	689	1000	1500	2019	36681
pay_amt3	0	1000	1000	1200	10000
pay_amt4	0	1000	1000	1100	9000
pay_amt5	0	0	1000	1069	689
pay_amt6	0	2000	5000	1000	679
default	1	1	0	0	0
male	0	0	0	0	1
grad_school	0	0	0	0	0

In [60]:
```
# simplifying pay features

pay_features= ['pay_' + str(i) for i in range(1,7)]
```

In [61]:
```
pay_features
```

Out[61]:
```
['pay_1', 'pay_2', 'pay_3', 'pay_4', 'pay_5', 'pay_6']
```

In [62]:
```
ccd['pay_1'].unique()
```

Out[62]:
```
array([ 2, -1,  0, -2,  1,  3,  4,  8,  7,  5,  6])
```

In [63]:
```
ccd.loc[ccd['pay_1'] > 0].T
```

Out[63]:

ID	1	14	16	19	20	23	27	32	39	51	...	29963	29967	29974	29975	29977	29982	29992	29995
limit_bal	20000	70000	50000	360000	180000	70000	60000	50000	50000	70000	...	50000	150000	230000	50000	40000	50000	210000	80000
sex	2	1	2	2	2	2	1	1	1	1	...	1	1	1	1	1	1	1	1
education	2	2	3	1	1	2	1	2	1	3	...	2	5	2	2	2	2	2	2
marriage	1	2	3	1	2	2	2	2	2	2	...	2	2	1	1	2	1	1	2
age	24	30	23	49	29	26	27	33	25	42	...	30	31	35	37	47	44	34	34
pay_1	2	1	1	1	1	2	1	2	1	1	...	1	2	1	1	2	1	3	2
pay_2	2	2	2	-2	-2	0	-2	0	-1	2	...	-1	0	-2	2	2	2	2	2
pay_3	-1	2	0	-2	-2	0	-1	0	-1	2	...	2	0	-2	2	3	2	2	2
pay_4	-1	0	0	-2	-2	2	-1	0	-2	2	...	-1	0	-2	2	2	2	2	2
pay_5	-2	0	0	-2	-2	2	-1	0	-2	2	...	-1	-2	-2	0	2	0	2	2
pay_6	-2	2	0	-2	-2	2	-1	0	-2	0	...	-2	-2	-2	0	2	0	2	2
bill_amt1	3913	65802	50614	0	0	41087	-109	30518	0	37042	...	-264	134866	0	10904	52358	38671	2500	72557
bill_amt2	3102	67369	29173	0	0	42445	-425	29618	780	36171	...	264	136692	0	9316	54892	36772	2500	77708
bill_amt3	689	65701	28116	0	0	45020	259	22102	0	38355	...	264	91815	0	4328	53415	33101	2500	79384
bill_amt4	0	66782	28771	0	0	44006	-57	22734	0	39423	...	7300	0	0	2846	51259	28192	2500	77519
bill_amt5	0	36137	29531	0	0	46905	127	23217	0	38659	...	0	0	0	1585	47151	22676	2500	82607
bill_amt6	0	36894	30211	0	0	46012	-189	23680	0	39362	...	0	0	0	1324	46934	14647	2500	81158
pay_amt1	0	3200	0	0	0	2007	0	1718	780	0	...	528	4633	0	0	4000	2300	0	7000
pay_amt2	689	0	1500	0	0	3582	1000	1500	0	3100	...	0	2000	0	3000	0	1700	0	3500
pay_amt3	0	3000	1100	0	0	0	0	1000	0	2000	...	7300	0	0	0	2000	0	0	0
pay_amt4	0	3000	1200	0	0	3601	500	1000	0	0	...	0	0	0	0	0	517	0	7000
pay_amt5	0	1500	1300	0	0	0	0	1000	0	1500	...	0	0	0	1000	3520	503	0	0
pay_amt6	0	0	1100	0	0	1820	1000	716	0	1500	...	0	0	0	1000	0	585	0	4000
default	1	1	0	0	0	1	1	1	1	1	...	0	1	1	1	1	0	1	1
male	0	1	0	0	0	0	1	1	1	1	...	1	1	1	1	1	1	1	1
grad_school	0	0	0	1	1	0	1	0	1	0	...	0	0	0	0	0	0	0	0

In [64]:
```
for x in pay_features:
    ccd.loc[ccd[x] <= 0, x] = 0
```

In [65]:
```
# simplifying delayed features

delayed_features = ['delayed_' + str(i) for i in range(1,7)]
```

In [66]:
```
for pay, delayed in zip(pay_features, delayed_features):
    ccd[delayed] = (ccd[pay] > 0).astype(int)
```

In [67]:
```
# creating a new feature: months delayed
ccd['months_delayed'] = ccd[delayed_features].sum(axis=1)
```

In [68]:
```
ccd.head().T
```

Out[68]:

ID	1	2	3	4	5
limit_bal	20000	120000	90000	50000	50000
sex	2	2	2	2	1
education	2	2	2	2	2
marriage	1	2	2	1	1
age	24	26	34	37	57
pay_1	2	0	0	0	0
pay_2	2	2	0	0	0
pay_3	0	0	0	0	0
pay_4	0	0	0	0	0
pay_5	0	0	0	0	0
pay_6	0	2	0	0	0
bill_amt1	3913	2682	29239	46990	8617
bill_amt2	3102	1725	14027	48233	5670
bill_amt3	689	2682	13559	49291	35835
bill_amt4	0	3272	14331	28314	20940
bill_amt5	0	3455	14948	28959	19146
bill_amt6	0	3261	15549	29547	19131
pay_amt1	0	0	1518	2000	2000
pay_amt2	689	1000	1500	2019	36681
pay_amt3	0	1000	1000	1200	10000
pay_amt4	0	1000	1000	1100	9000
pay_amt5	0	0	1000	1069	689
pay_amt6	0	2000	5000	1000	679
default	1	1	0	0	0
male	0	0	0	0	1
grad_school	0	0	0	0	0

In [69]:
```
ccd['months_delayed'].value_counts()
```

Out[69]:
```
0    19931
1     4426
2     1899
6     1341
3     1154
4      951
5      298
Name: months_delayed, dtype: int64
```

Splitting the dataset

In [70]:
```
numerical_features = numerical_features + ['months_delayed']
binary_features = ['male','married','grad_school','university']

X = ccd[numerical_features + binary_features]
y = ccd['default'].astype(int)
```

In [71]:
```
X[:10]
```

Out[71]:

ID	limit_bal	age	bill_amt1	bill_amt2	bill_amt3	bill_amt4	bill_amt5	bill_amt6	pay_amt1	pay_amt2	pay_amt3	pay_amt4	pay_amt5	pay_amt6	month
1	20000	24	3913	3102	689	0	0	0	0	689	0	0	0	0	
2	120000	26	2682	1725	2682	3272	3455	3261	0	1000	1000	1000	0	2000	
3	90000	34	29239	14027	13559	14331	14948	15549	1518	1500	1000	1000	1000	5000	
4	50000	37	46990	48233	49291	28314	28959	29547	2000	2019	1200	1100	1069	1000	
5	50000	57	8617	5670	35835	20940	19146	19131	2000	36681	10000	9000	689	679	
6	50000	37	64400	57069	57608	19394	19619	20024	2500	1815	657	1000	1000	800	
7	500000	29	367965	412023	445007	542653	483003	473944	55000	40000	38000	20239	13750	13770	
8	100000	23	11876	380	601	221	-159	567	380	601	0	581	1687	1542	
9	140000	28	11285	14096	12108	12211	11793	3719	3329	0	432	1000	1000	1000	
10	20000	35	0	0	0	0	13007	13912	0	0	0	13007	1122	0	

In [72]:
```
y[:10]
```

Out[72]:
```
ID
1     1
2     1
3     0
4     0
5     0
6     0
7     0
8     0
9     0
10    0
Name: default, dtype: int64
```

In [73]:
```
# 1. Import the class you will use
from sklearn.preprocessing import StandardScaler

# 2. Create an instance of the class
scaler = StandardScaler()

# 3. Use the fit method of the instance
scaler.fit(X[numerical_features])

X[:][numerical_features] = scaler.transform(X[numerical_features])
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/sklearn/preprocessing/data.py:61
7: DataConversionWarning: Data with input dtype int64 were all converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/ipykernel_launcher.py:10: DataCon
versionWarning: Data with input dtype int64 were all converted to float64 by StandardScaler.
  # Remove the CWD from sys.path while we load stuff.
```

In [74]:
```
len(X)
```

Out[74]:
```
30000
```

In [75]:
```
X[:10]
```

Out[75]:

ID	limit_bal	age	bill_amt1	bill_amt2	bill_amt3	bill_amt4	bill_amt5	bill_amt6	pay_amt1	pay_amt2	pay_amt3	pay_amt4	pay_amt5	pay_amt6	month
1	20000	24	3913	3102	689	0	0	0	0	689	0	0	0	0	
2	120000	26	2682	1725	2682	3272	3455	3261	0	1000	1000	1000	0	2000	
3	90000	34	29239	14027	13559	14331	14948	15549	1518	1500	1000	1000	1000	5000	
4	50000	37	46990	48233	49291	28314	28959	29547	2000	2019	1200	1100	1069	1000	
5	50000	57	8617	5670	35835	20940	19146	19131	2000	36681	10000	9000	689	679	
6	50000	37	64400	57069	57608	19394	19619	20024	2500	1815	657	1000	1000	800	
7	500000	29	367965	412023	445007	542653	483003	473944	55000	40000	38000	20239	13750	13770	
8	100000	23	11876	380	601	221	-159	567	380	601	0	581	1687	1542	
9	140000	28	11285	14096	12108	12211	11793	3719	3329	0	432	1000	1000	1000	
10	20000	35	0	0	0	0	13007	13912	0	0	0	13007	1122	0	

In [76]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=5/30, random_state=43)
```

In [77]:
```
len(X_train), len(y_train)
```

Out[77]: (25000, 25000)

In [78]:
```
X_train[:10]
```

Out[78]:

ID	limit_bal	age	bill_amt1	bill_amt2	bill_amt3	bill_amt4	bill_amt5	bill_amt6	pay_amt1	pay_amt2	pay_amt3	pay_amt4	pay_amt5	pay_amt6	mo
20794	20000	34	390	390	780	780	0	0	390	780	0	0	0	0	
27423	170000	31	172012	167929	116189	192082	120077	92593	168019	5000	6000	7125	5000	4500	
1376	30000	50	0	0	0	0	7092	6832	0	0	0	7092	0	0	
25681	290000	31	284583	274119	275169	189354	193163	197303	10000	10490	6620	6700	7000	7150	
20200	250000	44	23438	0	3850	0	32690	37141	0	3850	0	32690	5000	5000	
26945	370000	30	333930	280727	285705	295747	250158	255956	13000	11000	15000	10000	10000	12000	
3050	210000	37	1890	3037	2429	823	1089	1451	3037	1200	0	1089	1201	1031	
17952	170000	36	158954	139482	139869	139956	141431	149946	7	6513	6548	5300	11001	0	
8839	220000	41	6516	194	3619	7069	4092	0	0	3655	7069	4092	0	0	
7848	100000	25	91842	40308	41800	0	0	0	1809	2301	0	0	0	0	

In [ ]:

In [79]:
```
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression(C=1e5, solver='lbfgs')

logreg.fit(X_train['months_delayed'].values.reshape(-1, 1), y_train)
```

Out[79]:
```
LogisticRegression(C=100000.0, class_weight=None, dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=None, penalty='l2', random_state=None,
          solver='lbfgs', tol=0.0001, verbose=0, warm_start=False)
```

In [80]:
```
print("W0: {}, W1: {}".format(logreg.intercept_[0], logreg.coef_[0][0]))
```

```
W0: -1.8333177651924735, W1: 0.5287693448719608
```

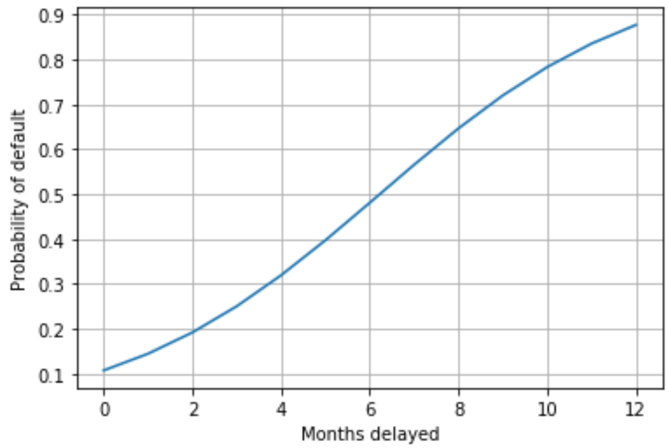
In [81]:
```
def get_probs(months_delayed):
    m = scaler.mean_[-1]
    std = scaler.var_[-1]**.5
    x = (months_delayed - m)/std
    prob_default = 1/(1+np.exp(-logreg.intercept_[0] + -logreg.coef_[0][0]*x))
    return prob_default
```

In [82]:
```
months = np.arange(13)
pred_probs = get_probs(months)
pd.DataFrame({'months': months, 'pred_probs':pred_probs})
```

Out[82]:

	months	pred_probs
0	0	0.107444
1	1	0.144685
2	2	0.192055
3	3	0.250395
4	4	0.319449
5	5	0.397450
6	6	0.481035
7	7	0.565694
8	8	0.646687
9	9	0.720050
10	10	0.783285
11	11	0.835499
12	12	0.877107

In [83]:
```
plt.plot(months, pred_probs)
plt.xlabel('Months delayed')
plt.ylabel('Probability of default')
plt.grid()
```



In [84]:
```
np.unique(y_train, return_counts=True)
```

Out[84]:
```
(array([0, 1]), array([19507,  5493]))
```

In [85]:
```
y_pred = logreg.predict(X_train['months_delayed'].values.reshape(-1, 1))
np.unique(y_pred, return_counts=True)
```

Out[85]:
```
(array([0, 1]), array([22862,  2138]))
```

In [86]:
```
accuracy_logreg = metrics.accuracy_score(y_train, y_pred)
accuracy_logreg
```

Out[86]:
```
0.80412
```

In [87]:
```
metrics.confusion_matrix(y_train, y_pred)
```

Out[87]:
```
array([[18736,   771],
       [ 4126,  1367]])
```

In [88]:
```
# Using test data
```

In [89]:
```
np.unique(y_test, return_counts=True)
```
Out[89]:
```
(array([0, 1]), array([3857, 1143]))
```

In [90]:
```
y_pred = logreg.predict(X_test['months_delayed'].values.reshape(-1, 1))
np.unique(y_pred, return_counts=True)
```
Out[90]:
```
(array([0, 1]), array([4548,  452]))
```

In [91]:
```
metrics.accuracy_score(y_test, y_pred)
```
Out[91]:
```
0.7978
```

In [92]:
```
metrics.confusion_matrix(y_test, y_pred)
```
Out[92]:
```
array([[3697,  160],
       [ 851,  292]])
```

In [ ]:
```

```