Logistic Function

Example 6-1: The logistic function in Python for one independent variable

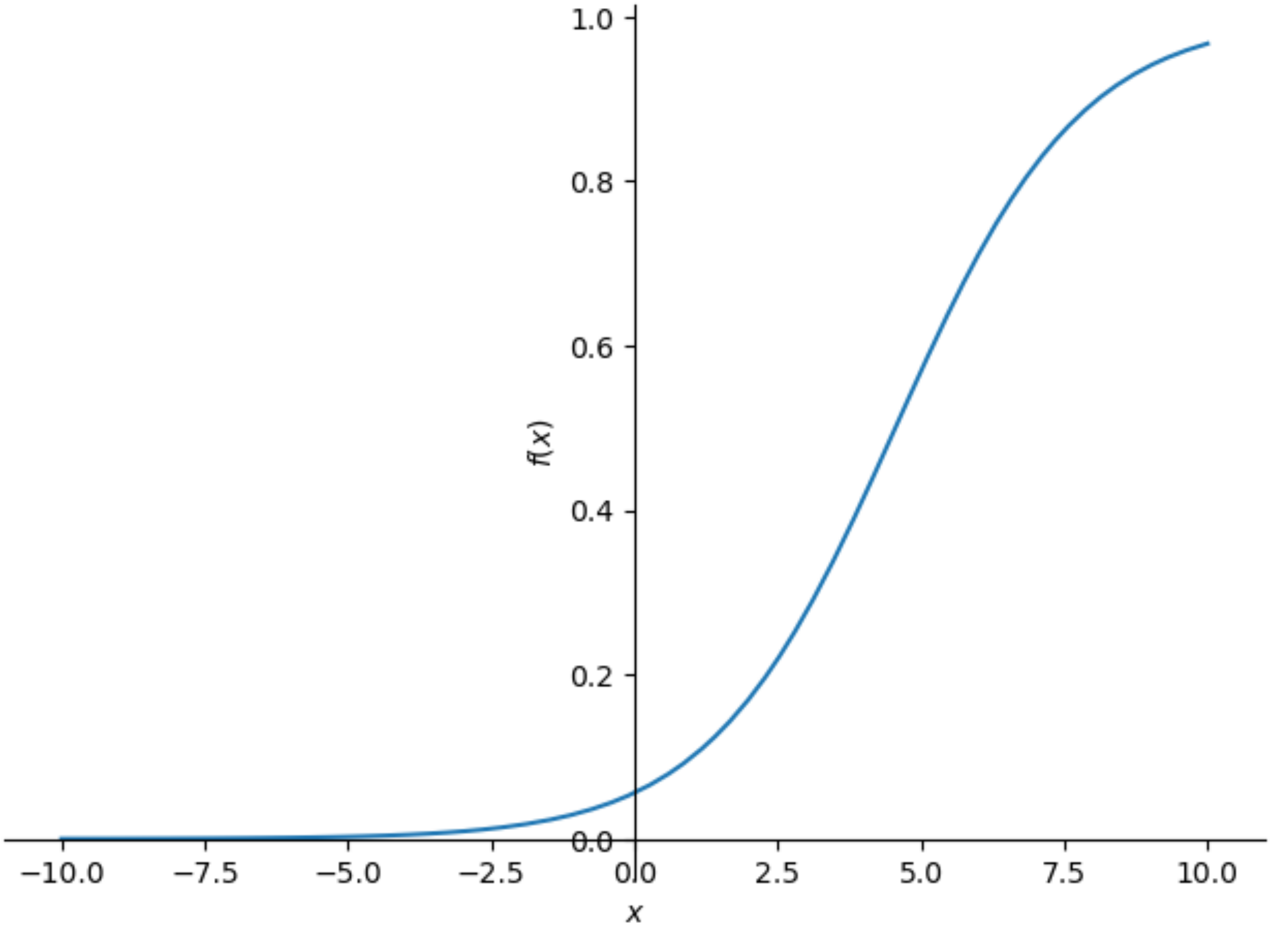
In [111…

```
import math

def predict_probability(x, b0, b1):
    p = 1 / (1 + math.exp(-(b0 + b1 * x)))
    return p
```

Example 6-2: Using Sympy to plot a logistic function

In [112…

```
import math
from sympy import *

x, b0, b1 = symbols('x b0 b1')
p = 1 / (1 + exp(-(b0 + b1 * x)))
p = p.subs(b0, -2.823)
p = p.subs(b1, 0.620)

plot(p); # the plot is from sympy, it can handle both 2D and 3D (plot3d)
```



Using Scipy

Example 6-3. Usig a plain logistic regression in Scipy

In [112…

```
import pandas as pd
from sklearn.linear_model import LogisticRegression

#Load the data
df = pd.read_csv('https://bit.ly/33ebs2R', delimiter=',')

X = df.values[:, :-1]
Y = df.values[:, -1]

model = LogisticRegression(penalty=None)
model.fit(X,Y)
```

Out[112…

```
▾ LogisticRegression
LogisticRegression(penalty=None)
```

In [112…

```
b1 = model.coef_.flatten()[0]
```

In [113…

```
b0 = model.intercept_[0]
```

Using Maximum Likelihood and Gradient Descent

Example 6-4: Calculating the joint likelihood of observing all the points for a given logistic regression

In [113…

```
# Using if functions
import math
import pandas as pd

patient_data = pd.read_csv('https://bit.ly/33ebs2R', delimiter=',').itertupl

def logistic_function(x):
    p = 1 / (1 + math.exp(-(b0 + b1 * x)))
    return p

joint_likelihood = 1

for p in patient_data:
    if p.y == 1:
        joint_likelihood *= logistic_function(p.x)
    elif p.y == 0:
        joint_likelihood *= (1 - logistic_function(p.x))

print(joint_likelihood)
```

```
4.79111802216874e-05
```

In [113…

```
# Using multiply
import math
import pandas as pd

patient_data = pd.read_csv('https://bit.ly/33ebs2R', delimiter=',').itertupl

def logistic_function(x):
    p = 1 / (1 + math.exp(-(b0 + b1 * x)))
    return p

joint_likelihood = 1

for p in patient_data:
    joint_likelihood *= (logistic_function(p.x) ** p.y * (1 - logistic_funct
#    if p.y == 1:
#        joint_likelihood *= logistic_function(p.x)
#    elif p.y == 0:
#        joint_likelihood *= (1 - logistic_function(p.x))
```

```
print(joint_likelihood)
```

```
4.79111802216874e-05
```

In [114…

```
# Using log functions - avoiding the floating point underflow
import math
import pandas as pd

patient_data = pd.read_csv('https://bit.ly/33ebs2R', delimiter=',').itertupl

def logistic_function(x):
    p = 1 / (1 + math.exp(-(b0 + b1 * x)))
    return p

joint_likelihood = 0

for p in patient_data:
    if p.y == 1:
        joint_likelihood += math.log(logistic_function(p.x))
    elif p.y == 0:
        joint_likelihood += math.log(1 - logistic_function(p.x))

print(math.exp(joint_likelihood))
```

```
4.791118022168739e-05
```

In [114…

```
# Calculate the logarithmic addition
import math
import pandas as pd

patient_data = pd.read_csv('https://bit.ly/33ebs2R', delimiter=',').itertupl

def logistic_function(x):
    p = 1 / (1 + math.exp(-(b0 + b1 * x)))
    return p

joint_likelihood = 0.0

for p in patient_data:
    joint_likelihood += math.log(logistic_function(p.x) ** p.y * \
                                 (1.0 - logistic_function(p.x)) ** (1.0 - p.

joint_likelihood = math.exp(joint_likelihood)
print(joint_likelihood)
```

```
4.791118022168739e-05
```

Example 6-8: Using gradient descent on logistic regression

In [114…

```
from sympy import *
```

```
import pandas as pd

points = list(pd.read_csv('https://tinyurl.com/y2cocoo7').itertuples())

b1, b0, i, n = symbols('b1 b0 i n')
x, y = symbols('x y', cls=Function)

joint_likelihood = Sum(log((1.0 / (1.0 + exp(-(b0 + b1 * x(i))))) ** y(i) \
        * (1.0 - (1.0 / (1.0 + exp(-(b0 + b1 * x(i)))))) ** (1 - y(i))), (i,

d_b1 = diff(joint_likelihood, b1) \
    .subs(n, len(points)-1).doit() \
    .replace(x, lambda i: points[i].x) \
    .replace(y, lambda i: points[i].y)

d_b0 = diff(joint_likelihood, b0) \
    .subs(n, len(points)-1).doit() \
    .replace(x, lambda i: points[i].x) \
    .replace(y, lambda i: points[i].y)

d_b1 = lambdify([b1, b0], d_b1)
d_b0 = lambdify([b1, b0], d_b0)

b1 = 0.01
b0 = 0.01
L = 0.01

for j in range(10_000):
    b1 += d_b1(b1, b0) * L
    b0 += d_b0(b1, b0) * L

print(b1, b0)
```

```
0.6926693075370819 -3.1757515504098244
```

Multivariable Logistic Regression

Example 6-9: Doing a multivariable logistic regression on employee data

In [115…

```
import pandas as pd
from sklearn.linear_model import LogisticRegression

employee_data = pd.read_csv("https://tinyurl.com/y6r7qjrp")

inputs = employee_data.iloc[:, :-1]
output = employee_data.iloc[:, -1]

fit = LogisticRegression(penalty=None).fit(inputs, output)
```

```
print(employee_data.columns)
print('coefficients:{0}'.format(fit.coef_.flatten()))
```

```
Index(['SEX', 'AGE', 'PROMOTIONS', 'YEARS_EMPLOYED', 'DID_QUIT'], dtype='obj
ect')
coefficients:[ 0.03213405  0.03682453 -2.50410028  0.9742266 ]
```

In [116…

```
# Interact and test with new employee data
def predict_employee_will_stay(sex, age, promotions, years_employed):
    prediction = fit.predict([[sex, age, promotions, years_employed]]) # due
    probabilities = fit.predict_proba([[sex, age, promotions, years_employed
    if prediction == [[1]]:
        return 'will leave: {0}'.format(probabilities)
    else:
        return 'will stay: {0}'.format(probabilities)
```

In [116…

```
# Test a prediction
n = input("Predict employee will stay or leave {sex}, {age}, {promotions}, {
(sex, age, promotions, years_employed) = n.split(',')
print(predict_employee_will_stay(int(sex), int(age), int(promotions), int(ye
```

```
will leave: [[0.28570264 0.71429736]]
```

```
/opt/anaconda3/lib/python3.11/site-packages/sklearn/base.py:439: UserWarnin
g: X does not have valid feature names, but LogisticRegression was fitted wi
th feature names
  warnings.warn(
/opt/anaconda3/lib/python3.11/site-packages/sklearn/base.py:439: UserWarnin
g: X does not have valid feature names, but LogisticRegression was fitted wi
th feature names
  warnings.warn(
```

R-Squared

1) Using statsmodel to get the R_Squared

In [117…

```
import statsmodels.api as sm
#Load the data
df = pd.read_csv('https://bit.ly/33ebs2R', delimiter=',')
```

In [120…

```
x = df.values[:,:-1]
y = df.values[:, -1]
x = sm.add_constant(x)

# fit the logistic regression model
model = sm.Logit(y, x)
result = model.fit()
```

```
# Get the Logistic Regression Results
result.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.473627
         Iterations 6
```

Out[120…

Logit Regression Results

Dep. Variable:	y	**No. Observations:**	21
Model:	Logit	**Df Residuals:**	19
Method:	MLE	**Df Model:**	1
Date:	Sun, 26 May 2024	**Pseudo R-squ.:**	0.3065
Time:	21:05:47	**Log-Likelihood:**	-9.9462
converged:	True	**LL-Null:**	-14.341
Covariance Type:	nonrobust	**LLR p-value:**	0.003029

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
const	-3.1758	1.553	-2.045	0.041	-6.220	-0.131
x1	0.6927	0.297	2.331	0.020	0.110	1.275

2) Plain Python to calculate the R_Squared

In [121…

```
import pandas as pd
import math
#Load the data
points = list(pd.read_csv('https://bit.ly/33ebs2R', delimiter=',').itertuple
```

# Calculate the log likelihood fit

In [122…

```
# - Method 1
## define the likelihood_fit
def likelihood(x):
    p = 1 / (1 + math.exp(-(b0 + b1 * x)))
    return p

log_likelihood_fit = 0

for p in points:
    if p.y == 1:
        log_likelihood_fit += log(likelihood(p.x))
    elif p.y ==0:
        log_likelihood_fit += log(1 - likelihood(p.x))
```

```
log_likelihood_fit
```

Out[122…

```
$\displaystyle -9.94616167318397$
```

In [122…

```
# - Method 2

## define the likelihood_fit
def likelihood(x):
    p = 1 / (1 + math.exp(-(b0 + b1 * x)))
    return p

log_likelihood_fit = 0

log_likelihood_fit =sum(log(likelihood(p.x) ** p.y) + log((1 - likelihood(p.

log_likelihood_fit
```

Out[122…

```
$\displaystyle -9.94616167318397$
```

In [126…

```
# - Method 3

## define the likelihood_fit
def likelihood(x):
    p = 1 / (1 + math.exp(-(b0 + b1 * x)))
    return p

log_likelihood_fit = 0

log_likelihood_fit = sum(log(likelihood(p.x)) * p.y + log(1 - likelihood(p.x

log_likelihood_fit
```

Out[126…

```
-9.946161673231583
```

Calculate the log likelihood

In [122…

```
# - Method 1
likelihood = sum(p.y for p in points) / len(points)

log_likelihood = 0

for p in points:
    if p.y == 1:
        log_likelihood += log(likelihood)
    elif p.y == 0:
        log_likelihood += log(1 - likelihood)
```

```
log_likelihood
```

Out[122…

$\displaystyle -14.3410701987099$

In [122…

```
# - Method 2
likelihood = sum(p.y for p in points) / len(points)

log_likelihood = 0

for p in points:
    log_likelihood += log(likelihood) * p.y + log(1 - likelihood) * (1 - p.y
log_likelihood
```

Out[122…

$\displaystyle -14.3410701987099$

In [122…

```
# - Method 3
likelihood = sum(p.y for p in points) / len(points)

log_likelihood = 0

log_likelihood = sum(log(likelihood) * p.y + log(1 - likelihood) * (1 - p.y)

log_likelihood
```

Out[122…

$\displaystyle -14.3410701987099$

In [126…

```
# - Method 4
likelihood = sum(p.y for p in points) / len(points)

log_likelihood = 0

log_likelihood = sum(log(likelihood ** p.y) + log((1 - likelihood) ** (1 - p

log_likelihood
```

Out[126…

```
-14.341070198709906
```

Calculate the R_Squared

In [123…

```
r_squared = (log_likelihood - log_likelihood_fit) / log_likelihood
print('{0:.4f}'.format(r_squared))
```

```
0.3065
```

In [123…

```
# compare with the statsmodel result
print('{0:.4f}'.format(result.prsquared))
```

```
0.3065
```

P_Value

In [126…
```
from scipy.stats import chi2
```

In [126…
```
chi2_input = 2 * (log_likelihood_fit - log_likelihood)
```

In [126…
```
p_value
```

Out[126…
```
<scipy.stats._distn_infrastructure.rv_continuous_frozen at 0x318248410>
```

In [126…
```
log_likelihood_fit
```

Out[126…
```
-9.946161673231583
```

In [126…
```
log_likelihood
```

Out[126…
```
-14.341070198709906
```

In [126…
```
chi2_input
```

Out[126…
```
8.789817050956646
```

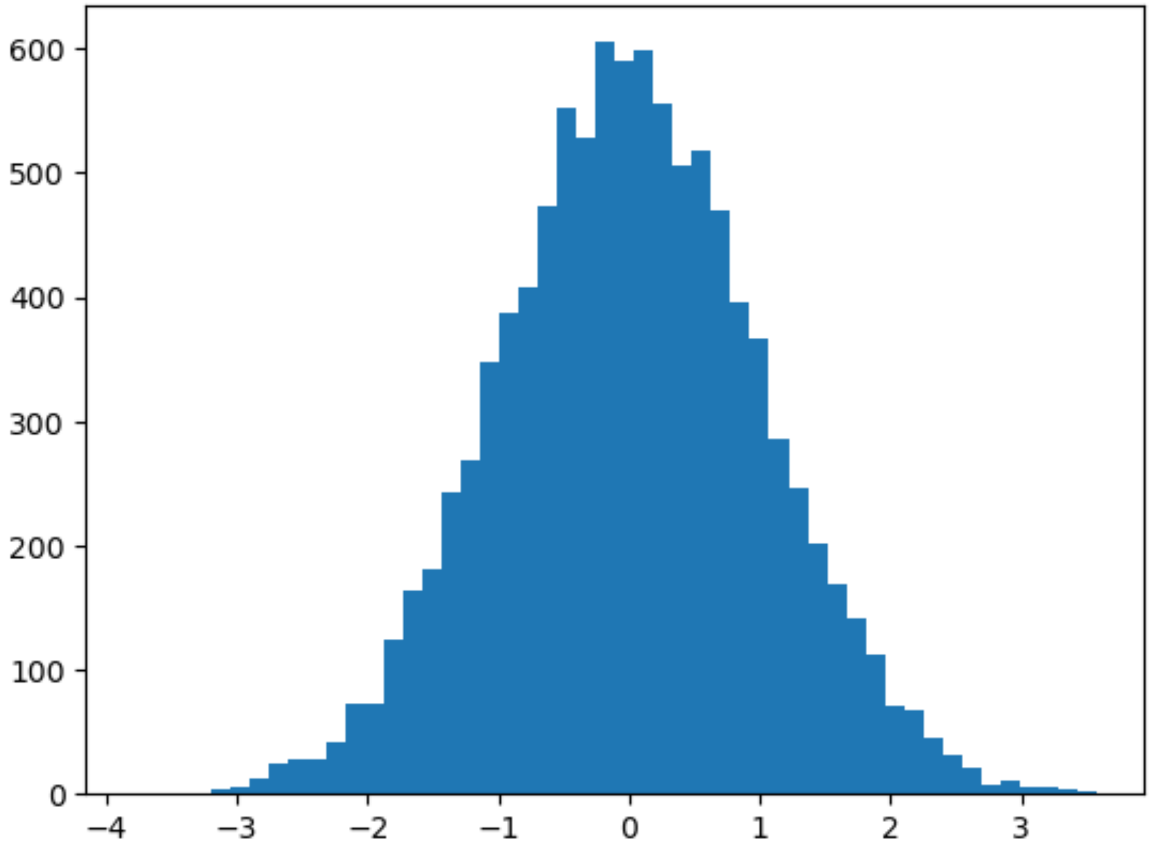
In [127…
```
chi2.pdf(chi2_input,1)
```

Out[127…
```
0.0016604875618753787
```

Check Chi-Square distribution is the square of each value in standard normal distribution

In [130…
```
a = np.random.standard_normal(10000)
```

In [130…
```
plt.hist(a,bins=50);
```

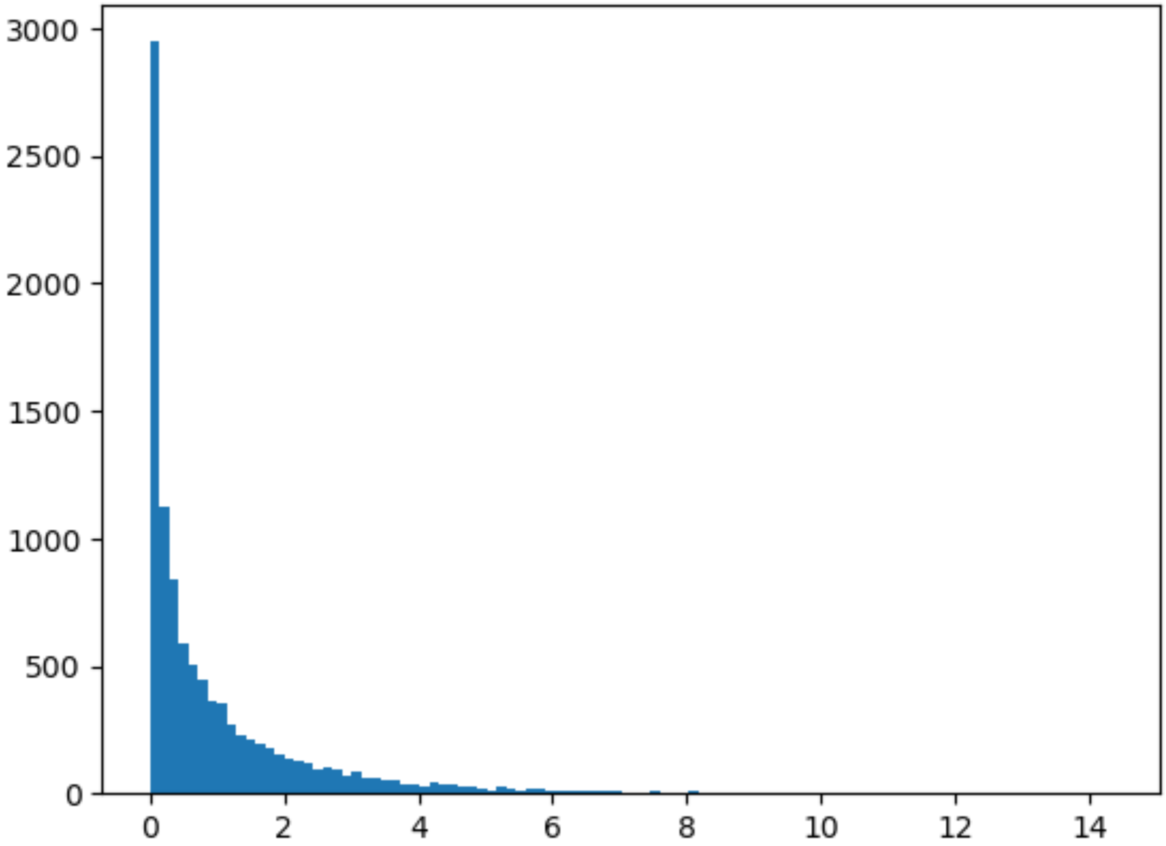


In [130…

```
b = a ** 2
```

In [130…

```
plt.hist(b, bins=100);
```

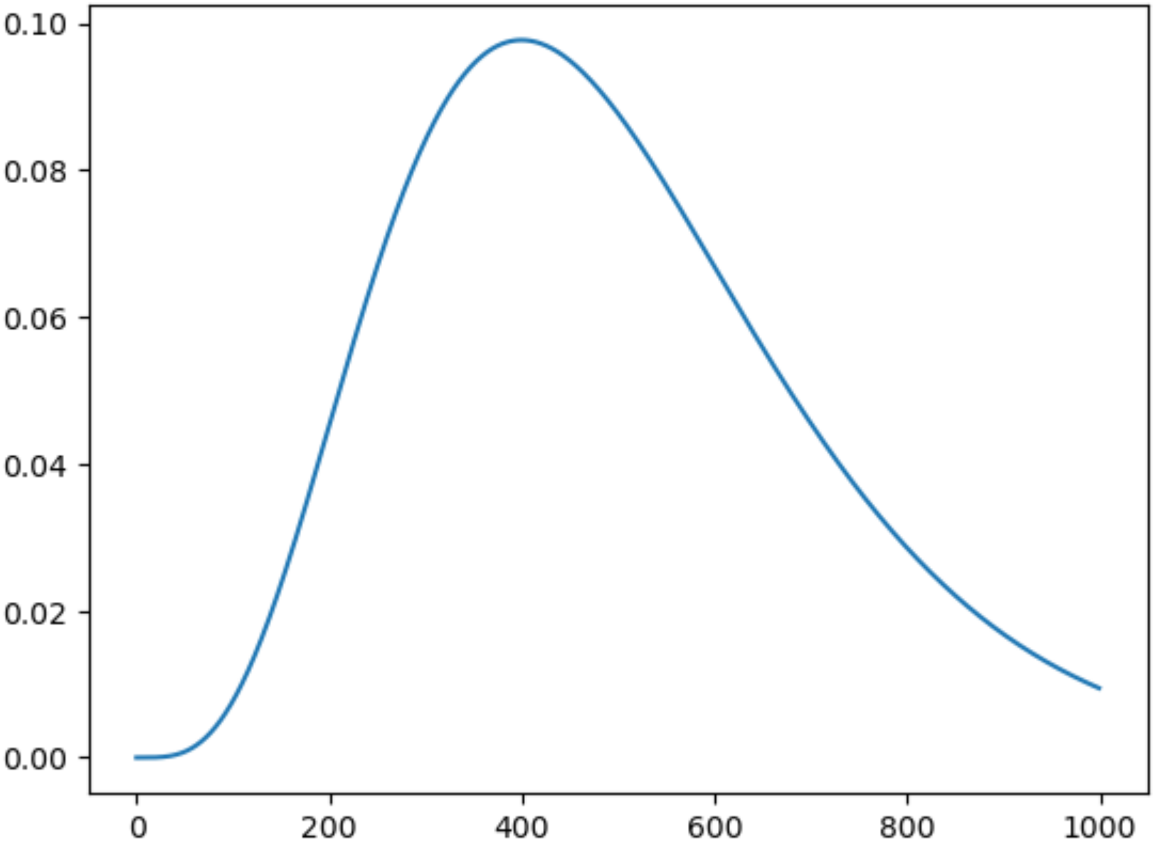


In [130…

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import chi2
```

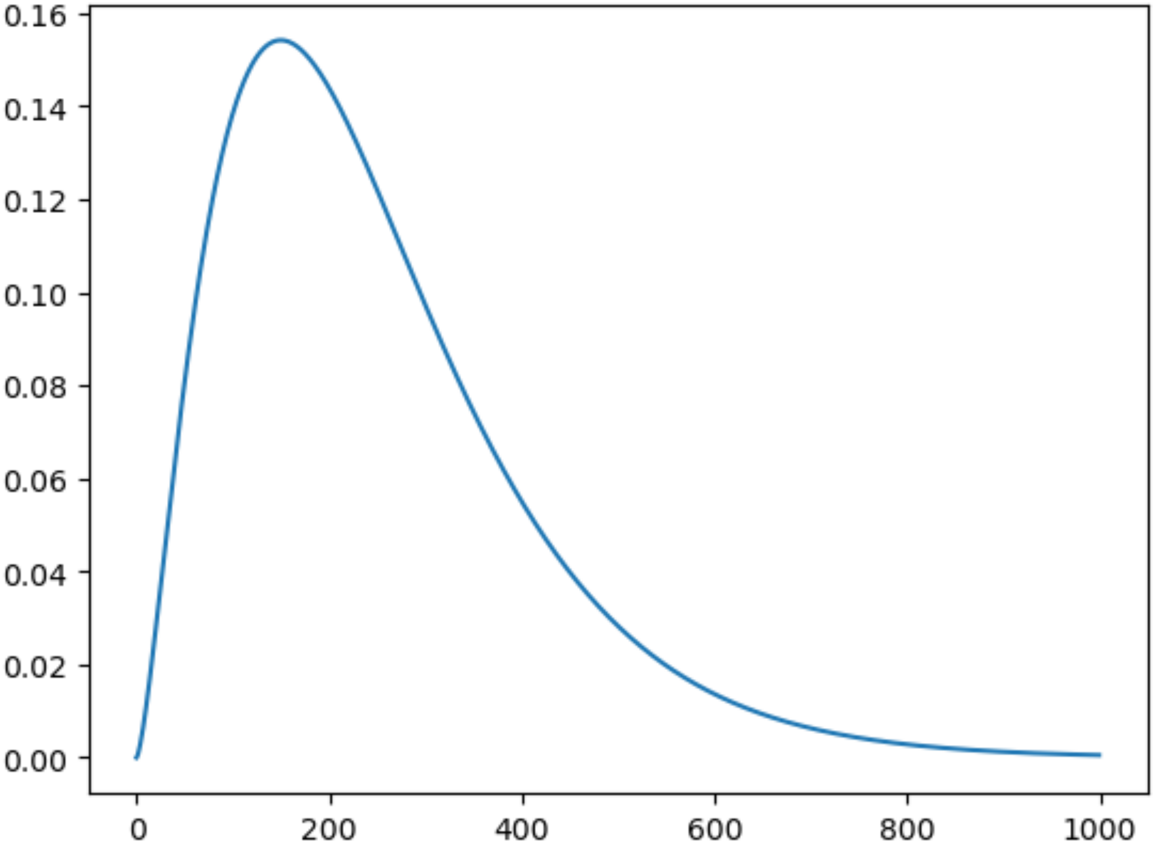
In [131…

```
df = 10
x = np.linspace(0,20,1000)
pdf = chi2.pdf(x, df)
plt.plot(pdf);
```



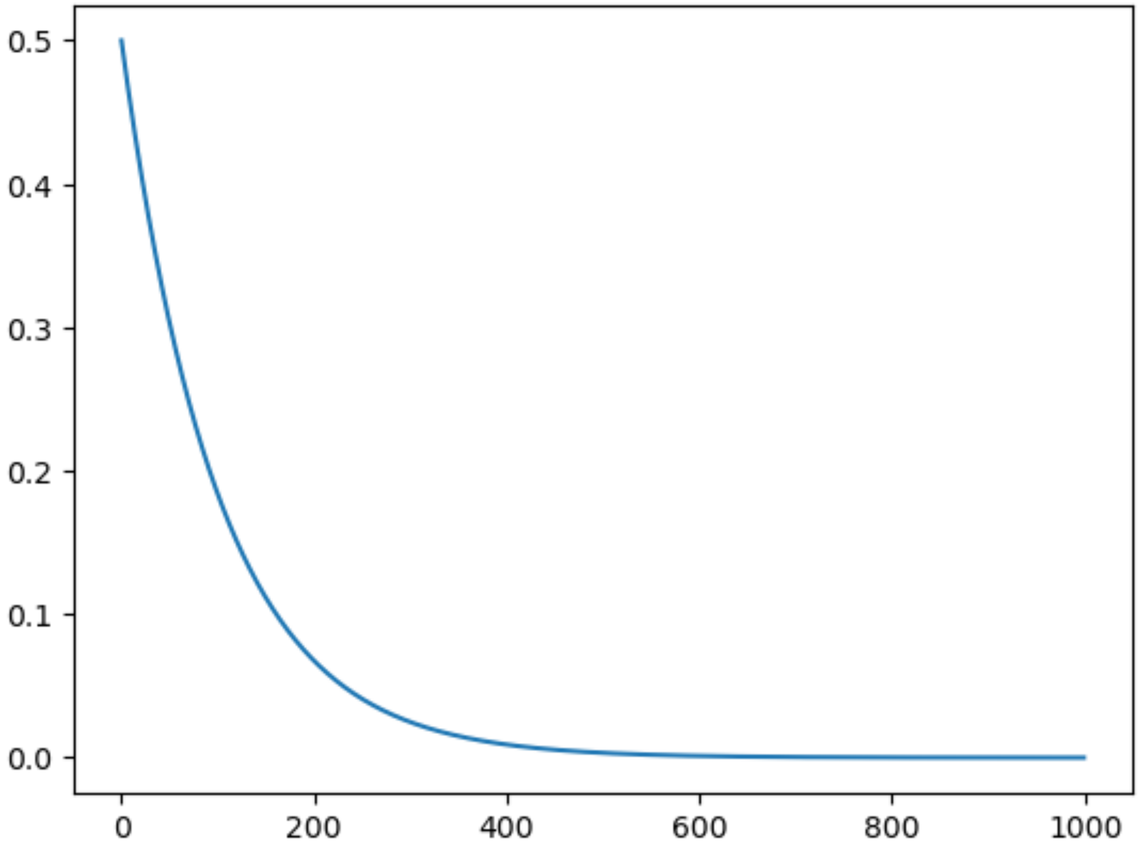
In [131…

```
df = 5
x = np.linspace(0,20,1000)
pdf = chi2.pdf(x, df)
plt.plot(pdf);
```



In [131…

```
df = 2
x = np.linspace(0,20,1000)
pdf = chi2.pdf(x, df)
plt.plot(pdf);
```



Train/Test Splits

In [131…

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold, cross_val_score
```

In [132…

```
df = pd.read_csv('https://tinyurl.com/y6r7qjrp', sep=',')

x = df.values[:, :-1]
y = df.values[:, -1]

kfold = KFold(n_splits=3, random_state=7, shuffle=True)
model = LogisticRegression(penalty=None)
result = cross_val_score(model, x, y, cv=kfold)
result.mean() # Accuracy
```

Out[132…

```
0.6111111111111112
```

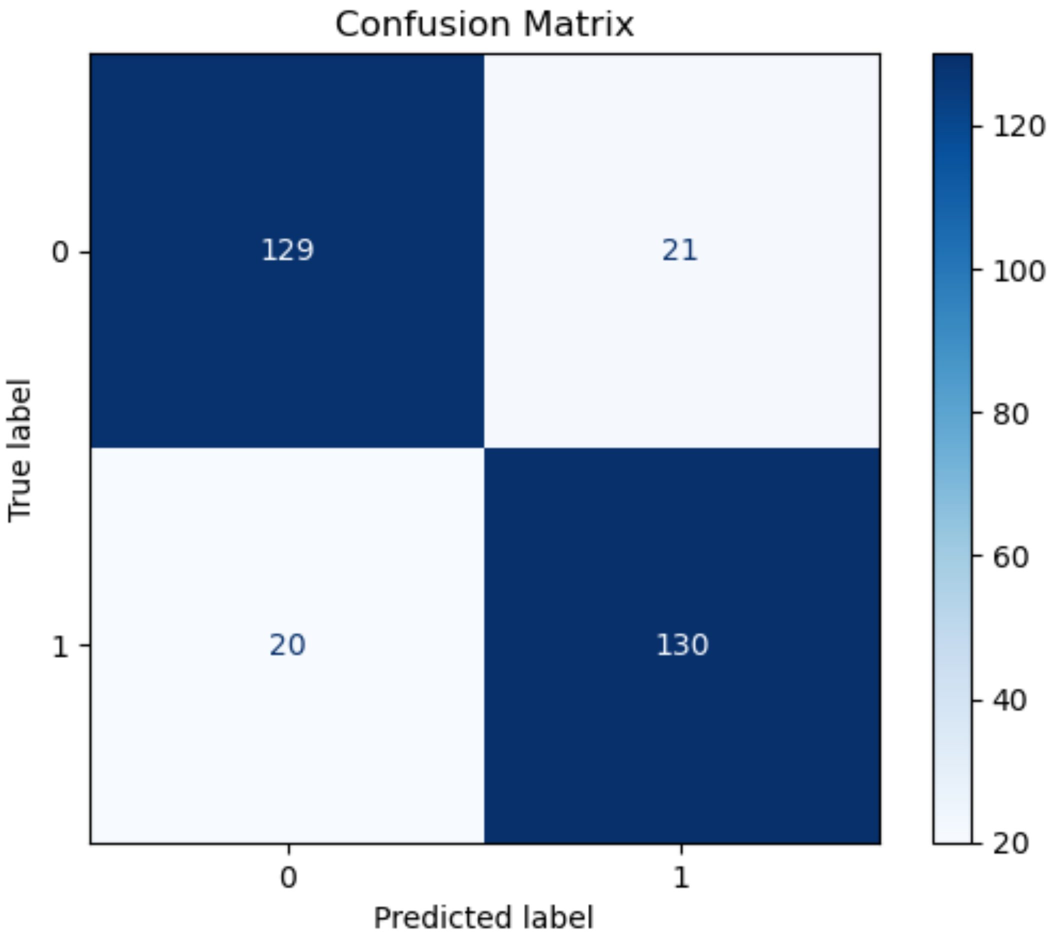
Confusion Matrices

In [138…

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, classi

# Generate synthetic binary classification data
X, y = make_classification(n_samples=1000, n_features=20, random_state=42)

# Split data into training and test sets, using stratify to solve the imbala
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, ran

# Fit a logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)
```

In [138…

```
# Compute the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Display the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=model.clas
disp.plot(cmap='Blues')
plt.title('Confusion Matrix')
plt.show()
```



In [138…
```
model.classes_
```

Out[138…
```
array([0, 1])
```

In [138…
```
cm
```

Out[138…
```
array([[129,  21],
       [ 20, 130]])
```

In [138…
```
cm.sum() # Total variables
```

Out[138…
```
300
```

In [138…
```
cm.sum(axis=1) # Get the supports
```

Out[138…
```
array([150, 150])
```

In [138…
```
# Print classification report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.86      0.86       150
           1       0.86      0.87      0.86       150

    accuracy                           0.86       300
   macro avg       0.86      0.86      0.86       300
weighted avg       0.86      0.86      0.86       300
```

In [138…

```
TP = cm[1,1]
TN = cm[0,0]
FP = cm[0,1]
FN = cm[1,0]
```

In [139…

```
# This is also be the precision for 0
Negative_predict_value = TN / (TN + FN)
print('{0:.2f}'.format(Negative_predict_value))
```

```
0.87
```

In [139…

```
Precision = TP / (TP + FP)
print('{0:.2f}'.format(Precision))
```

```
0.86
```

In [139…

```
Recall = TP / (TP + FN) # Also called Sensitivity
print('{0:.2f}'.format(Sensitivity))
```

```
0.83
```

In [139…

```
# This is also be the recall for 0
Specificity = TN / (TN + FP)
print('{0:.2f}'.format(Specificity))
```

```
0.86
```

In [139…

```
f1_score = 2 * Precision * Recall / (Precision + Recall)
print('{0:.2f}'.format(f1_score))
```

```
0.86
```

In [139…

```
Accuracy = (TP + TN) / (TP + TN + FP + FN)
print('{0:.2f}'.format(Accuracy))
```

```
0.86
```

ROC (Receiver Operator Characters) & AUC (Area Under Curve)

In [139…

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

from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, roc_auc_score, RocCurveDisplay
```

In [140…

```
x, y = make_classification(n_samples=1000, n_features=20, n_classes=2, rando
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=1/3, rar
model = LogisticRegression().fit(x_train, y_train)
```

In [141…

```
y_pred = model.predict(x_test)
y_prob = model.predict_proba(x_test) # returns an array where each column re
y_prob = y_prob[:,1] #only retrive the probability for 1 (since the model.cl
```

In [141…

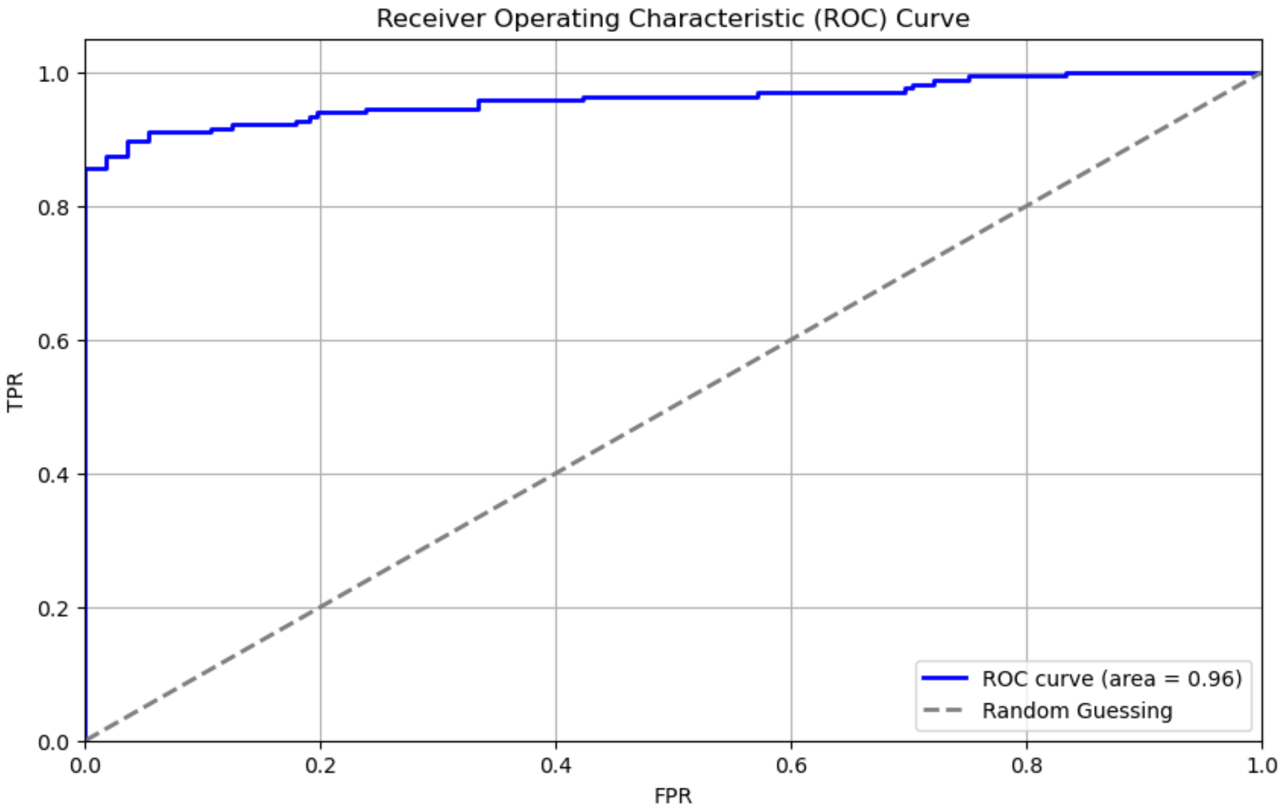
```
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
# fpr = 1 - precision
# tpr = recall
# thresholds: the predicted probability is converted to a binary classificat
# examine how the True Positive Rate(TPR) and false Positive Rate(FPR) chang
```

In [141…

```
roc_auc = roc_auc_score(y_test, y_prob)
```

In [142…

```
plt.figure(figsize=(10, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (area = {roc_auc:.2
plt.plot([0,1],[0,1], color='gray', lw=2, linestyle='--', label='Random Gues
plt.xlim([0,1])
plt.ylim([0,1.05])
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.grid()
plt.show();
```


Receiver Operating Characteristic (ROC) Curve
TPR
1.0
0.8
0.6
0.4
0.2
0.0
FPR
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.96)
Random Guessing