Gaussian Copula

Consider the example where we have the marginal distribution X and Y which are Standard Normal Distributions but have an element of dependence (Called a Bivariate Normal). The relationship between X and Y is important in order to determine the joint distribution.

$$X, Y \sim N(0, 1)$$
$$\rho = \begin{bmatrix} 1 & \rho_{XY} \\ \rho_{XY} & 1 \end{bmatrix}$$

Suppose X and Y are dependent Standard Normal distributions with a pearson correlation of $\rho_{XY} = 0.8$ (i.e. they are 80% linearly correlated). Assuming we have a random sample $x_1 = 2$ from $X \sim N(0, 1)$.

In [2]:
```
plt.style.available
```

Out[2]:
```
['Solarize_Light2',
 '_classic_test_patch',
 '_mpl-gallery',
 '_mpl-gallery-nogrid',
 'bmh',
 'classic',
 'dark_background',
 'fast',
 'fivethirtyeight',
 'ggplot',
 'grayscale',
 'seaborn-v0_8',
 'seaborn-v0_8-bright',
 'seaborn-v0_8-colorblind',
 'seaborn-v0_8-dark',
 'seaborn-v0_8-dark-palette',
 'seaborn-v0_8-darkgrid',
 'seaborn-v0_8-deep',
 'seaborn-v0_8-muted',
 'seaborn-v0_8-notebook',
 'seaborn-v0_8-paper',
 'seaborn-v0_8-pastel',
 'seaborn-v0_8-poster',
 'seaborn-v0_8-talk',
 'seaborn-v0_8-ticks',
 'seaborn-v0_8-white',
 'seaborn-v0_8-whitegrid',
 'tableau-colorblind10']
```

In [4]:

```
### Import the required packages
import scipy as sp
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib.patches import Ellipse
import matplotlib.transforms as transforms
import seaborn as sns
import math
import pandas as pd

### Set theme
plt.style.use("seaborn-v0_8-whitegrid")
# plt.style.use('seaborn')
sns.set_style("darkgrid")

### Define the colour scheme
c1 = "#173f5f"
c2 = "#20639b"
c3 = "#3caea3"
c4 = "#f6d55c"
c5 = "#ed553b"

print("Imported the required packages successfully!")

def confidence_ellipse(x, y, ax, n_std=3.0, facecolor='none', **kwargs):
    if x.size != y.size:
        raise ValueError("x and y must be the same size")
    cov = np.cov(x, y)
    pearson = cov[0, 1]/np.sqrt(cov[0, 0] * cov[1, 1])
    ell_radius_x = np.sqrt(1 + pearson)
    ell_radius_y = np.sqrt(1 - pearson)
    ellipse = Ellipse((0, 0), width=ell_radius_x * 2, height=ell_radius_y
    scale_x = np.sqrt(cov[0, 0]) * n_std
    mean_x = np.mean(x)
    scale_y = np.sqrt(cov[1, 1]) * n_std
    mean_y = np.mean(y)
    transf = transforms.Affine2D().rotate_deg(45).scale(scale_x, scale_y).
    ellipse.set_transform(transf + ax.transData)
    return ax.add_patch(ellipse)
```

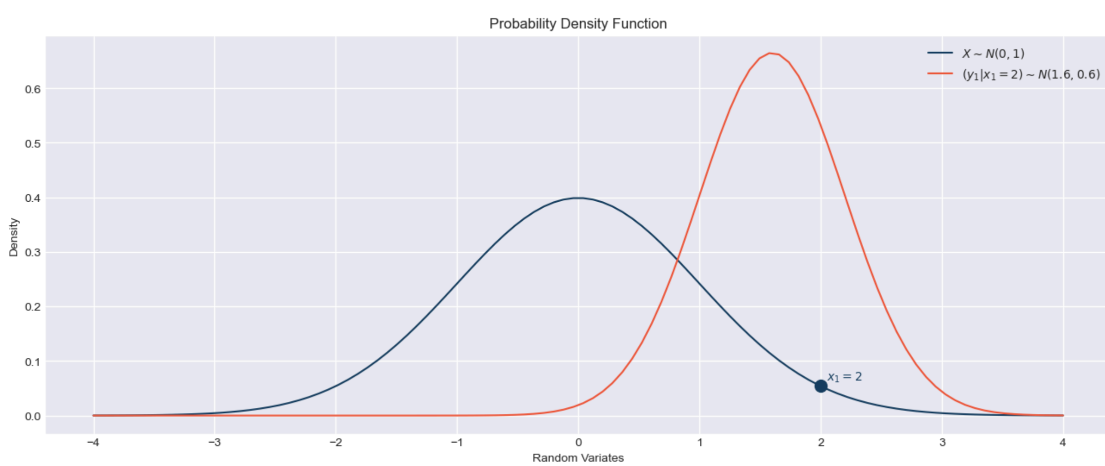
```
Imported the required packages successfully!
```

In [7]:

```
n = 1000

X = sp.stats.norm.rvs(loc=0, scale=1, size=n, random_state=123)
Y = sp.stats.norm.rvs(loc=0, scale=1, size=n, random_state=321)
```

In [8]:

```
### Variable Parameters
correlation = 0.8
x1 = 2

### Fixed Parameters
mu_X = 0
std_X = 1
mu_Y = 0
std_Y = 1

### Theoretical x1 PDF
x1_pdf_x = np.linspace(-4, 4, 100)
x1_pdf_y = sp.stats.norm.pdf(x=x1_pdf_x, loc=0, scale=1)

### Emperical x1 PDF
x1_pdf = sp.stats.norm.pdf(x=x1, loc=0, scale=1)

### Expected Value and Standard Deviation of y1
E_y1 = mu_Y + (correlation * std_Y * ( (x1 - mu_X)/std_X) )
std_y1 = std_Y * math.sqrt(1 - correlation**2)

### Theoretical y1 PDF
y1_pdf_x = np.linspace(-4, 4, 100)
y1_pdf_y = sp.stats.norm.pdf(x=y1_pdf_x, loc=E_y1, scale=std_y1)

### Plot the sampled PDF and CDF against the theoretical distribution
fig, ax = plt.subplots(figsize=(16,6))

ax.scatter(x1, x1_pdf, color=c1, s=100, label="_nolabel_")
ax.plot(x1_pdf_x, x1_pdf_y, color=c1, label=r"$X\sim N(0,1)$")
ax.plot(y1_pdf_x, y1_pdf_y, color=c5, label=r"$(y_1|x_1={}) \sim N({:,.1f}
ax.annotate(r'$x_1={}$'.format(x1), xy=(x1+0.05, x1_pdf+0.01), color=c1)

ax.set(title="Probability Density Function", xlabel="Random Variates", yla
ax.legend()
plt.show()
```

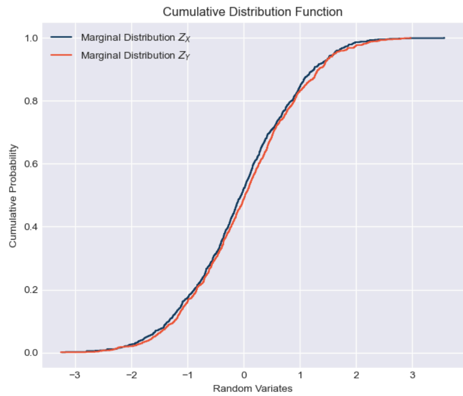


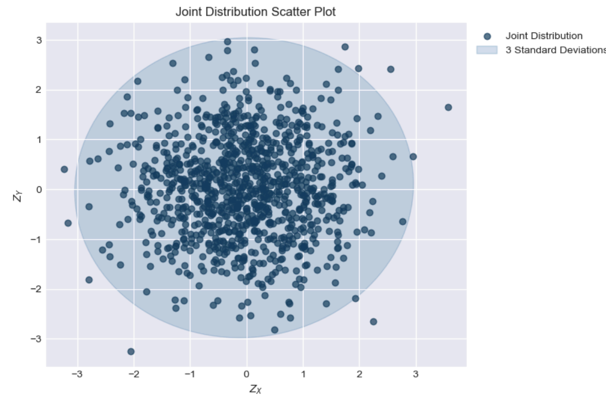
Obtain random variates

In [9]:

```
n = 1000

Z_x = sp.stats.norm.rvs(loc=0, scale=1, size=n, random_state=123)
Z_y = sp.stats.norm.rvs(loc=0, scale=1, size=n, random_state=321)

### Emperical CDF
cdf_y = np.arange(1, n+1) / n
X_cdf_x = np.sort(Z_x)
Y_cdf_x = np.sort(Z_y)

### Correlation
correlation_sample = np.corrcoef(X, Y)

### Expected Value
X_mean = np.mean(Z_x)
Y_mean = np.mean(Z_y)

X2 = np.square(Z_x)
Y2 = np.square(Z_y)

### Variance
var = (np.mean(X2)*np.mean(Y2)) - ((np.mean(X)**2)*(np.mean(Y)**2))

print("Sample Expected Value: ({:,.2f},{:,.2f})".format(X_mean, Y_mean))
print("Sample Variance: {:,.2f}".format(var))

### Plot the sampled PDF and CDF against the theoretical distribution
fig, ax = plt.subplots(figsize=(16,6), nrows=1, ncols=2)

ax[0].step(X_cdf_x, cdf_y, where='post', color=c1, label=r"Marginal Distri
ax[0].step(Y_cdf_x, cdf_y, where='post', color=c5, label=r"Marginal Distri
ax[0].set(title="Cumulative Distribution Function", xlabel="Random Variate
ax[0].legend()

ax[1].scatter(Z_x, Z_y, color=c1, alpha=0.7, label="Joint Distribution")
confidence_ellipse(Z_x, Z_y, ax=ax[1],alpha=0.2, facecolor=c2, edgecolor=c
ax[1].set(title="Joint Distribution Scatter Plot", xlabel=r"$Z_X$", ylabel
ax[1].legend(bbox_to_anchor=(1,1), loc="upper left")
plt.show()
```

```
Sample Expected Value: (-0.04,0.03)
Sample Variance: 1.01
```


Cumulative Distribution Function
Marginal Distribution Z_X
Marginal Distribution Z_Y
Cumulative Probability
Random Variates

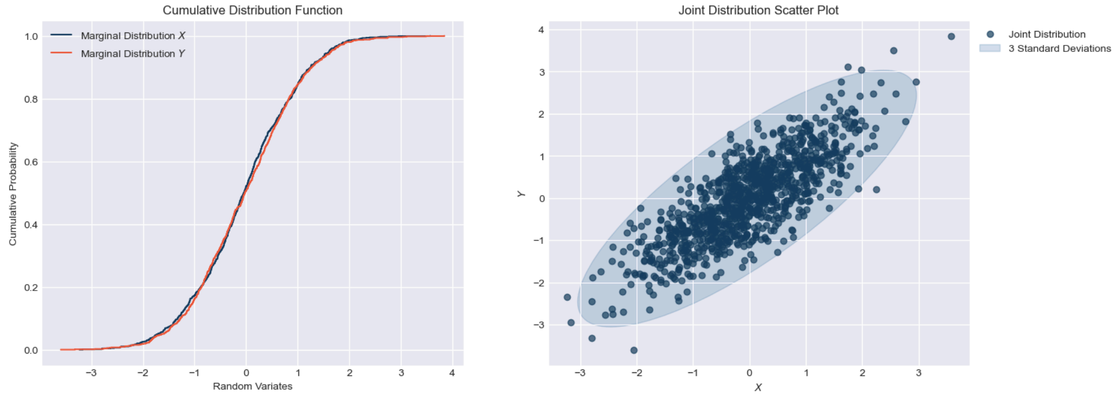

Joint Distribution Scatter Plot
Joint Distribution
3 Standard Deviations
Z_Y
Z_X

Apply Cholesky Decomposition and Determine X and Y

In [10]:

```
correlation = 0.8
n = 1000
Z_x = sp.stats.norm.rvs(loc=0, scale=1, size=n, random_state=123)
Z_y = sp.stats.norm.rvs(loc=0, scale=1, size=n, random_state=321)

# Construct the correlation matrix and Cholesky Decomposition
rho = np.matrix([[1, correlation], [correlation, 1]])
cholesky = np.linalg.cholesky(rho)
Z = np.matrix([Z_x, Z_y])
Z_XY = cholesky * Z
X = np.array(Z_XY[0,:]).flatten()
Y = np.array(Z_XY[1,:]).flatten()

### Emperical CDF
cdf_y = np.arange(1, n+1) / n
X_cdf_x = np.sort(X)
Y_cdf_x = np.sort(Y)

### Correlation
correlation_sample = np.corrcoef(X, Y)

### Expected Value
X_mean = np.mean(X)
Y_mean = np.mean(Y)
X2 = np.square(X)
Y2 = np.square(Y)
### Variance
var = (np.mean(X2)*np.mean(Y2)) - ((np.mean(X)**2)*(np.mean(Y)**2))
print("Sample Expected Value: ({:,.2f},{:,.2f})".format(X_mean, Y_mean))
print("Sample Variance: {:,.2f}".format(var))
### Plot the sampled PDF and CDF against the theoretical distribution
fig, ax = plt.subplots(figsize=(16,6), nrows=1, ncols=2)
ax[0].step(X_cdf_x, cdf_y, where='post', color=c1, label=r"Marginal Distri
ax[0].step(Y_cdf_x, cdf_y, where='post', color=c5, label=r"Marginal Distri
ax[0].set(title="Cumulative Distribution Function", xlabel="Random Variate
ax[0].legend()
ax[1].scatter(X, Y, color=c1, alpha=0.7, label="Joint Distribution")
confidence_ellipse(X, Y, ax=ax[1],alpha=0.2, facecolor=c2, edgecolor=c2, z
ax[1].set(title="Joint Distribution Scatter Plot", xlabel=r"$X$", ylabel=r
ax[1].legend(bbox_to_anchor=(1,1), loc="upper left")
plt.show()
```

```
Sample Expected Value: (-0.04,-0.01)
Sample Variance: 1.03
```


Cumulative Distribution Function
Marginal Distribution X
Marginal Distribution Y
Cumulative Probability
Random Variates
Joint Distribution Scatter Plot
Joint Distribution
3 Standard Deviations
X
Y

In [ ]: