Klion_Lab2

October 26, 2022

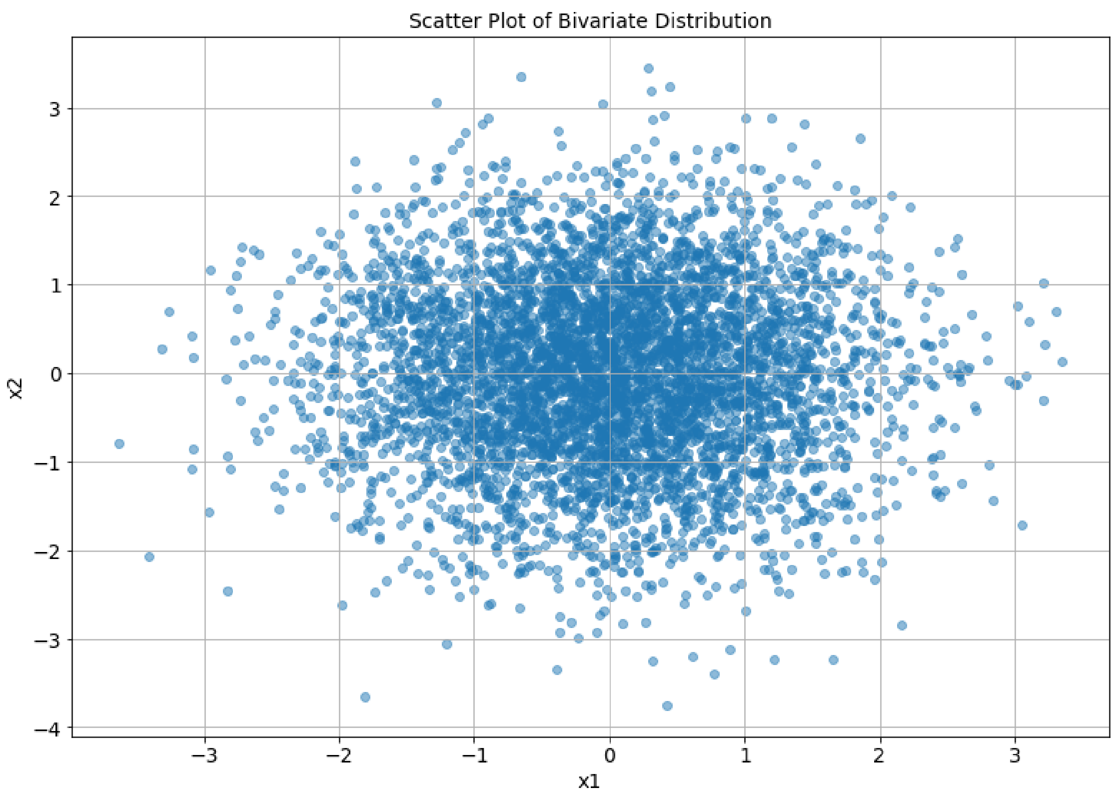
[1]:
```
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline
plt.rcParams['figure.figsize'] = (14,8)
```

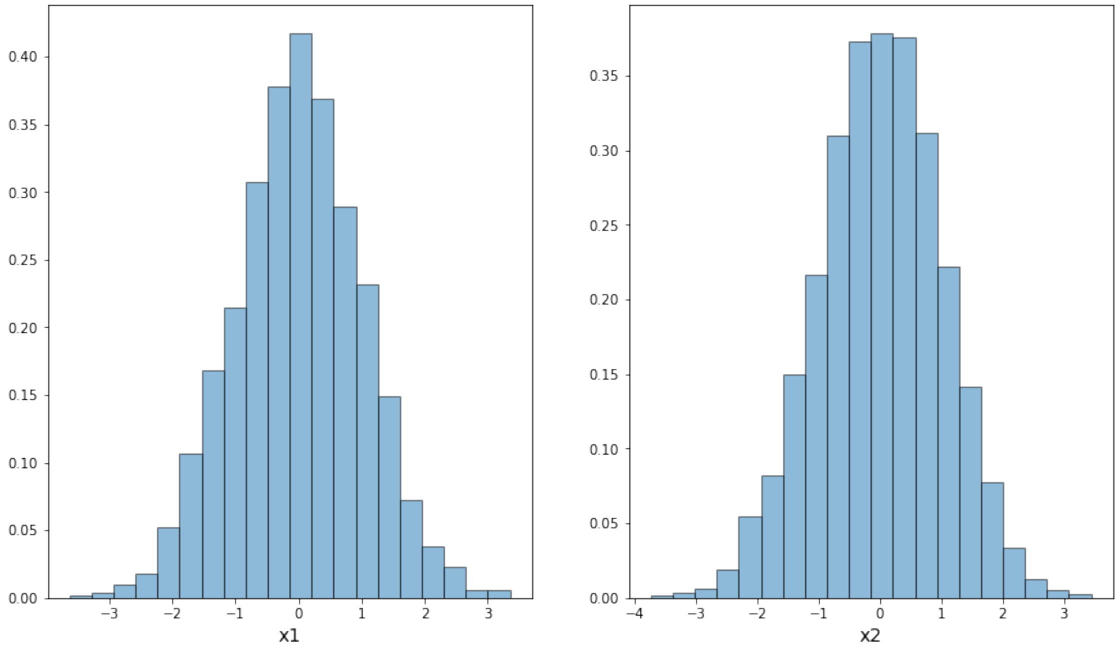
1 E1: Generate N = 5000 points from this bivariate distribution and visualize them using a scatter plot. Plot histograms to show the marginal distributions of x_1 and x_2.

[22]:
```
def bivariateDist(npts: int, mean: np.ndarray = [0, 0], cov: np.ndarray = [[1,␣
↪0], [0, 1]], isPlot: bool = False):
    '''
    Create a bivariate normal distribution with given mean and variance

    Params:
        npts: number of points to sample from distribution
        mean: 1-D array of the mean of each variable (default: [0, 0])
        cov: 2-D array analagous to variance of 1d normal dist (default: [[1,␣
↪0], [0, 1]])
        isPlot: bool

    Returns:
        out: The samples drawn of shape (npts, )

    '''
    rng = np.random.default_rng()
    pts = rng.multivariate_normal(mean, cov, npts)
    x1, x2 = pts[:, 0], pts[:, 1]
    if isPlot:
        plt.figure(figsize=(13,9))
        plt.tick_params(labelsize=14)
        plt.scatter(x1, x2, alpha = 0.5)
        plt.title("Scatter Plot of Bivariate Distribution", fontsize = 14)
        plt.xlabel("x1", fontsize = 14)
        plt.ylabel("x2", fontsize = 14)
```

```
        plt.grid()
        plt.show()
    return x1, x2
```

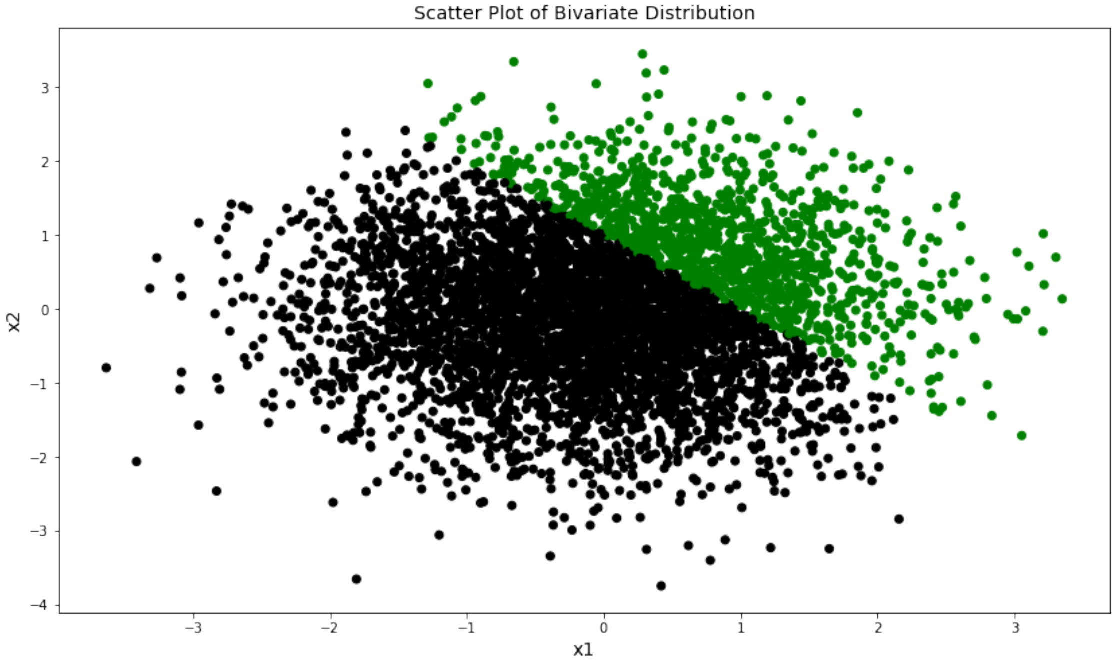
[3]:
```
#draw 5000 samples from the bivariate distribution
x1, x2 = bivariateDist(5000, isPlot=True)

#histogram of x1
plt.subplot(121)
plt.hist(x1, 20, alpha = 0.5, density = True, label = 'x1', edgecolor = 'k')
plt.xlabel('x1', fontsize = 14)

#histogram of x2
plt.subplot(122)
plt.hist(x2, 20, alpha = 0.5, density = True, label = 'x2', edgecolor = 'k')
plt.xlabel('x2', fontsize = 14)
plt.show()
#plt.hist([x1, x2], 20, density = True, label = ['x1', 'x2'])
```





2 E2: For the samples generated in E1, show/highlight only the points $x_1 + x_2 \geq 1$.

[4]:
```
import matplotlib as mpl
cond1 = (x1 + x2 >= 1) #highlighting condition
my_cmap = mpl.colors.ListedColormap(['k', 'green']) #green if cond met,␣
 ↪otherwise black
plt.scatter(x1, x2, c = cond1, cmap = my_cmap)
plt.title("Scatter Plot of Bivariate Distribution", fontsize = 14)
plt.xlabel("x1", fontsize = 14)
plt.ylabel("x2", fontsize = 14)
plt.show()
```



[5]:
```
#Compute the Pearson correlation coefficient for x1 and x2
rho1 = np.corrcoef(x1[cond1], x2[cond1])
print(rho1)
```

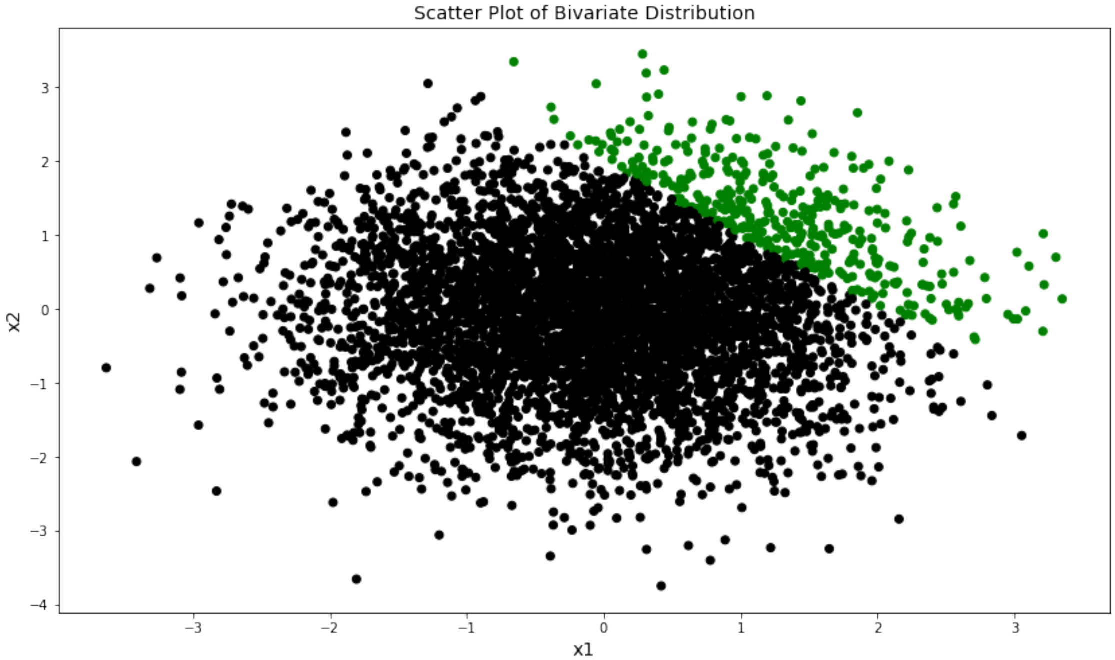
```
[[ 1.         -0.62975027]
 [-0.62975027  1.        ]]
```

[6]:
```
print('Var[x1] =', np.var(x1)) #Variance in skill for original
print('Var[x1[cond]]', np.var(x1[cond1])) #Variance in skill for successful
```

```
Var[x1] = 1.0080561475355654
Var[x1[cond]] 0.6237341659362142
```

3 E3: Repeat E2 for a more stringent selection criteria: $x_1+x_2 \geq 2$.

[7]:
```
cond2 = (x1 + x2 >= 2) #highlighting condition
plt.scatter(x1, x2, c = cond2, cmap = my_cmap)
plt.title("Scatter Plot of Bivariate Distribution", fontsize = 14)
plt.xlabel("x1", fontsize = 14)
plt.ylabel("x2", fontsize = 14)
plt.show()
```




```
[8]: #Compute the Pearson correlation coefficient for x1 and x2
rho2 = np.corrcoef(x1[cond2], x2[cond2])
print(rho2)
```

```
[[ 1.         -0.75144256]
 [-0.75144256  1.        ]]
```

```
[9]: print('Var[x1] =', np.var(x1)) #Variance in skill for original
print('Var[x1[cond]]', np.var(x1[cond2])) #Variance in skill for successful
```

```
Var[x1] = 1.0080561475355654
Var[x1[cond]] 0.5516015311448879
```

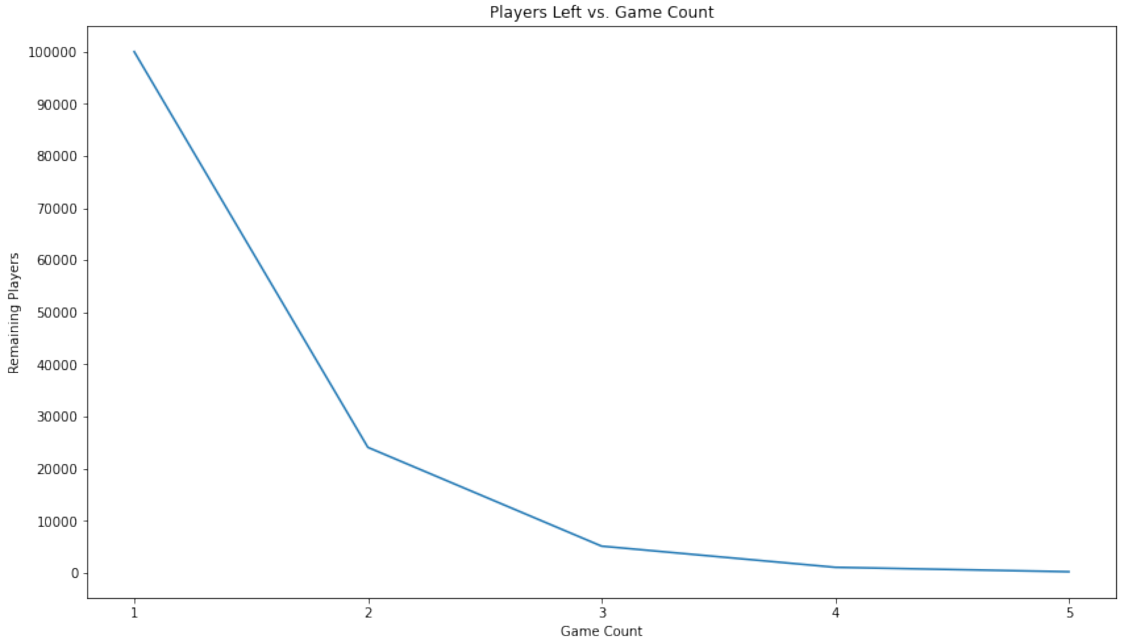
The more stringent condition makes the correlation between skill and luck that much more negative while also lowering the variance slightly.

4 E4: Write a program to model a dynamic game that shows the paradox of skill

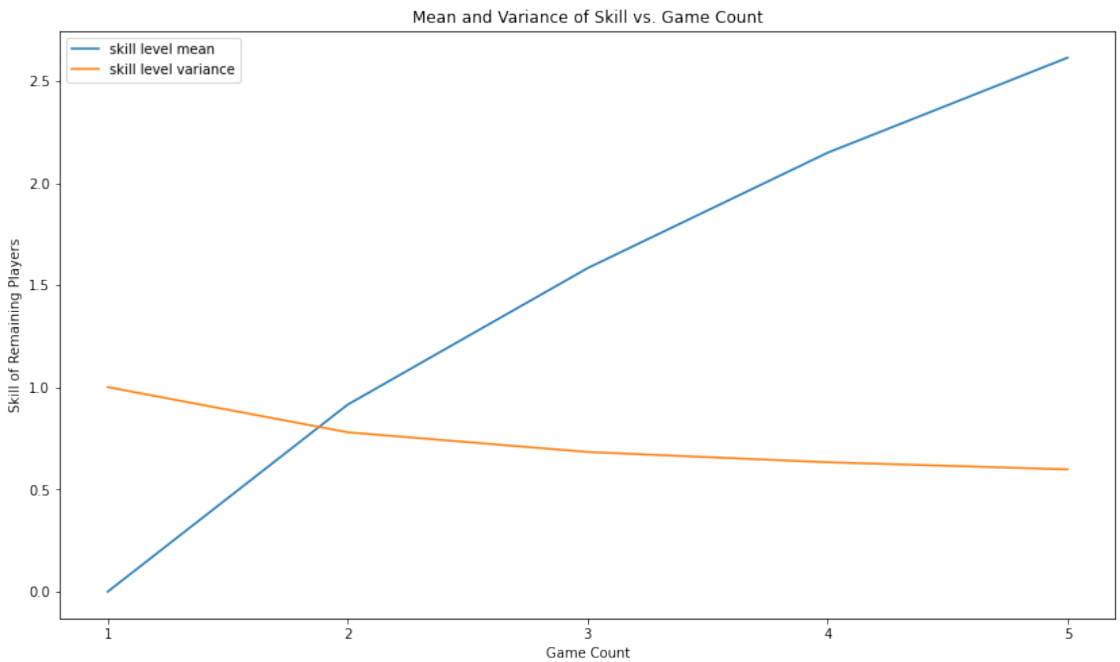
```
[13]: def gameModel(numPlayers: int, cutoff: float, mean: float, std: float):
    #create an instance of the default RNG generator
    rng = np.random.default_rng()
    #fix skill level at game start
    skillLvls = rng.normal(mean, std, numPlayers) #x1

    #start current player skills equal to starting number of player skills
```

```
    #currSkills = skillLvls

    #create an array to store surviving players and their skills per game number
    survivors, survSkillMean, survSkillVar = [], [], []

    #game ends when less than cutoff of original players left
    ''' while len(currSkills) > (cutoff * numPlayers):
        survivors.append(len(currSkills))
        #draw fresh luck for each player each game
        luckLvls = rng.normal(mean, std, len(currSkills))
        #calculate the mean skill level for threshold
        mu = np.mean(currSkills)
        threshold = currSkills + luckLvls >= 1 + mu
        currSkills = currSkills[threshold]'''
    while len(skillLvls) > (cutoff * numPlayers):
        survivors.append(len(skillLvls))
        #draw fresh luck for each player each game
        luckLvls = rng.normal(mean, std, len(skillLvls))
        #calculate the mean skill level for threshold
        mu = np.mean(skillLvls)
        #calculate the variance of the skill level
        var = np.std(skillLvls)

        #append post values to an array for future plotting
        survSkillMean.append(mu)
        survSkillVar.append(var)

        #threshold of success for continuation in game
        threshold = skillLvls + luckLvls >= 1 + mu
        #apply threshold
        skillLvls = skillLvls[threshold]
    return survivors, survSkillMean, survSkillVar

totPlayers = 100000
playerCts, meanSkill, varSkill = gameModel(totPlayers, 0.001, 0, 1)
```

[18]:
```
gameCount = range(1, len(playerCts) + 1)
ystep = totPlayers / 10 #get 10 yticks for better idea on number players␣
 ↪remaining

plt.plot(gameCount, playerCts)
plt.xticks(gameCount)
plt.yticks(np.arange(0, totPlayers + ystep, ystep))
plt.xlabel('Game Count')
plt.ylabel('Remaining Players')
plt.title("Players Left vs. Game Count")
plt.show()
```



[21]:
```
plt.plot(gameCount, meanSkill, label = 'skill level mean')
plt.plot(gameCount, varSkill, label = 'skill level variance')
plt.xticks(gameCount)
plt.xlabel('Game Count')
plt.ylabel('Skill of Remaining Players')
plt.title("Mean and Variance of Skill vs. Game Count")
plt.legend(loc = 'upper left')
plt.show()
```



Mean and Variance of Skill vs. Game Count
skill level mean
skill level variance
Skill of Remaining Players
Game Count
0.0
0.5
1.0
1.5
2.0
2.5
1
2
3
4
5