Koushik Sahu
118CS0597
Machine Learning Lab – 1

Code:
Problem 1:

```
from pathlib import Path
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

class Config:
    iris_datapath = Path('../data/iris.data')

iris_dataframe = pd.read_csv(Config.iris_datapath,
                             names=['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class'])

iris_dataframe.head()

classes = list(iris_dataframe['class'].unique())
print(f'Classes in iris dataset: {classes}')

iris_class = classes[2]
print(f'Class picked for the problem: {iris_class}')

iris_virginica_df = iris_dataframe[iris_dataframe['class'] == iris_class].reset_index(drop=True)
iris_virginica_df.head()

# lets pick the attribute petal_length for this problem
iris_attr = iris_virginica_df.columns[2]
petal_length_column = iris_dataframe[iris_attr]
print(f'Attribute picked for this problem: {iris_attr}')

def evaluate_mean(vals):
    count = vals.shape[0]
    summation = 0

    for i in vals:
        summation += i

    mean = summation / count
    return mean

def evaluate_std(vals):
    count = vals.shape[0]
    mean = evaluate_mean(vals)
    variance = 0
```

```
    for i in vals:
        variance += (i-mean)**2

    variance /= count
    std = np.sqrt(variance)
    return std

# calculating mean
petal_length_mean = evaluate_mean(petal_length_column)
print(f'Mean of petal length: {petal_length_mean}')

# calculating standard deviation
petal_length_std = evaluate_std(petal_length_column)
print(f'Standard deviation of petal length: {petal_length_std}')

def normal_distribution(x, mean, std):
    return 1/(std*np.sqrt(np.pi)) * np.power(np.e, (-1/2)*(((x-mean)/std)**2))

# ploting normal distribution
low_x = -5
high_x = 15
plot_pt_count = int(1e3)
delta = (high_x-low_x) / plot_pt_count

norm_distr_values = list()
plot_pts = list()
while low_x <= high_x:
    plot_pts.append(low_x)
    norm_distr_values.append(normal_distribution(low_x, petal_length_mean,
petal_length_std))
    low_x += delta

plt.plot(plot_pts, norm_distr_values)
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.show()

# finding minimum and maximum
min_petal_length = min(petal_length_column)
max_petal_length = max(petal_length_column)

def uniform_distribution(x, min_val, max_val):
    if min_val < x and x < max_val:
        return 1 / (max_val-min_val)
    return 0

# ploting normal distribution
low_x = -5
```

```
high_x = 15
plot_pt_count = int(1e3)
delta = (high_x-low_x) / plot_pt_count

uniform_distr_values = list()
plot_pts = list()
while low_x <= high_x:
    plot_pts.append(low_x)
    uniform_distr_values.append(uniform_distribution(low_x, min_petal_length,
max_petal_length))
    low_x += delta

plt.plot(plot_pts, uniform_distr_values)
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.show()

def exponential_distribution(x, λ):
    if x >= 0:
        return λ * np.power(np.e, -λ*x)
    return 0

# ploting exponential distribution
low_x = -5
high_x = 30
plot_pt_count = int(1e3)
delta = (high_x-low_x) / plot_pt_count
λ = 1/petal_length_mean

expo_distr_values = list()
plot_pts = list()
while low_x <= high_x:
    plot_pts.append(low_x)
    expo_distr_values.append(exponential_distribution(low_x, λ))
    low_x += delta

plt.plot(plot_pts, expo_distr_values)
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.show()

def factorial(x):
    if x==1 or x==0:
        return 1
    return x*factorial(x-1)

def poisson_distribution(x, λ):
    return (np.power(np.e, -λ)*np.power(λ, x)) / factorial(np.floor(x))
```

```
# ploting exponential distribution
low_x = 0
high_x = 30
plot_pt_count = int(1e3)
delta = (high_x-low_x) / plot_pt_count
λ = 1/petal_length_mean

poisson_distr_values = list()
plot_pts = list()
while low_x <= high_x:
    plot_pts.append(low_x)
    poisson_distr_values.append(poisson_distribution(low_x, λ))
    low_x += delta

plt.plot(plot_pts, poisson_distr_values)
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.show()

# using function written in problem 1
petal_length_mean = evaluate_mean(petal_length_column)
print(f'Mean petal length: {petal_length_mean}')
```

Problem 2:

```
# Median
def evaluate_median(vals):
    n = vals.shape[0]
    if n%2==1:
        return vals.iloc[n//2]
    return (vals.iloc[n//2] + vals[(n+1)//2]) / 2

petal_length_median = evaluate_median(petal_length_column)
print(f'Median petal length: {petal_length_median}')

petal_length_std = evaluate_std(petal_length_column)
print(f'Standard deviation petal length: {petal_length_std}')

petal_length_variance = petal_length_std**2
print(f'petal length: {petal_length_variance}')

petal_length_skewness = (3*(petal_length_median-petal_length_mean)) / petal_length_std
print(f'Skewness petal length: {petal_length_skewness}')

count = petal_length_column.shape[0]
kurtosis = 0
```

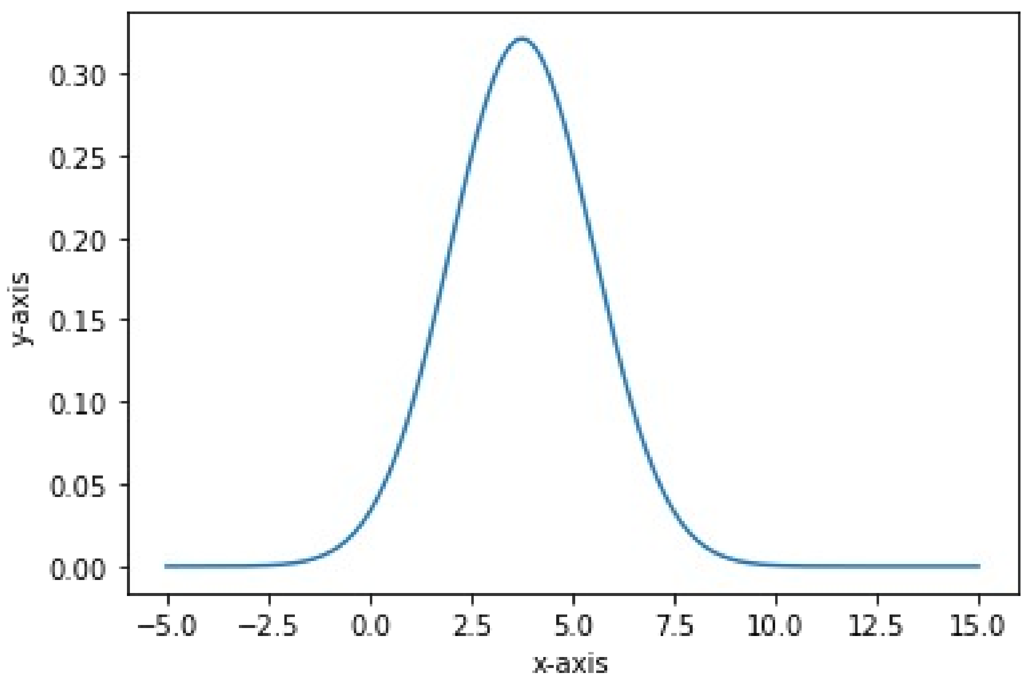
```
for i in petal_length_column:
    kurtosis += ((i-petal_length_mean)/petal_length_std)**4

kurtosis /= count

print(f'Kurtosis petal length: {kurtosis}')
```

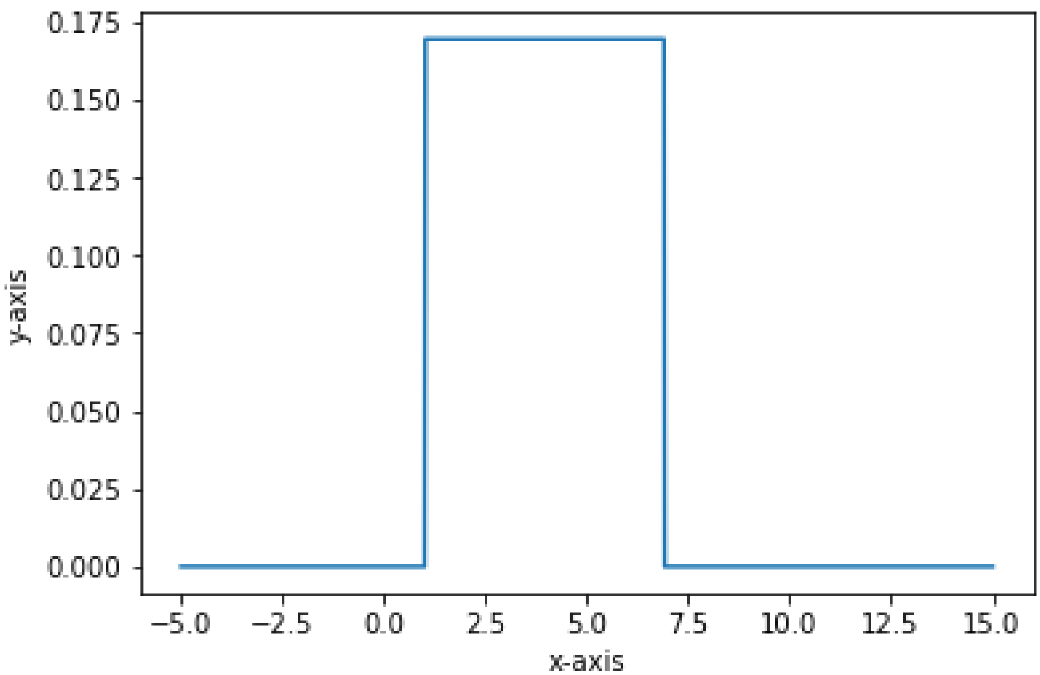
Problem 3:

```
mean, variance = 0, 1

data = np.random.normal(mean, variance, 1000)
count, bins, ignored = plt.hist(data, 20, density=1)
plt.plot(bins,
    1/(variance * np.sqrt(2 * np.pi)) * np.exp( - (bins - mean)**2 / (2 * variance**2) ),
    linewidth=2,
    color='r')

plt.title('Normal Distribution')
plt.show()
```

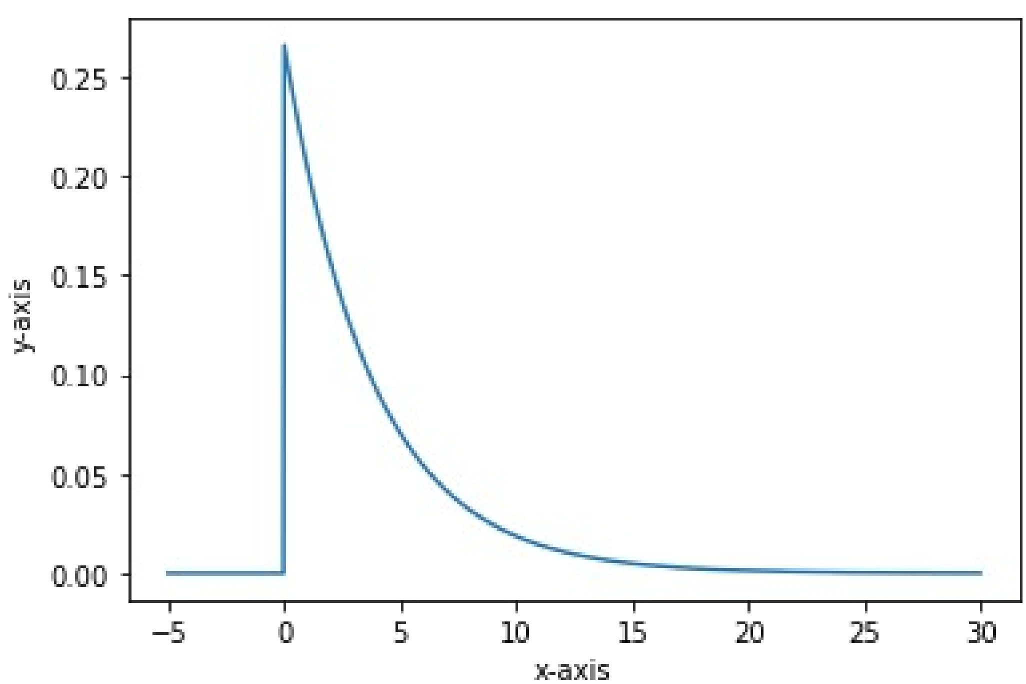
Output:

Problem 1:

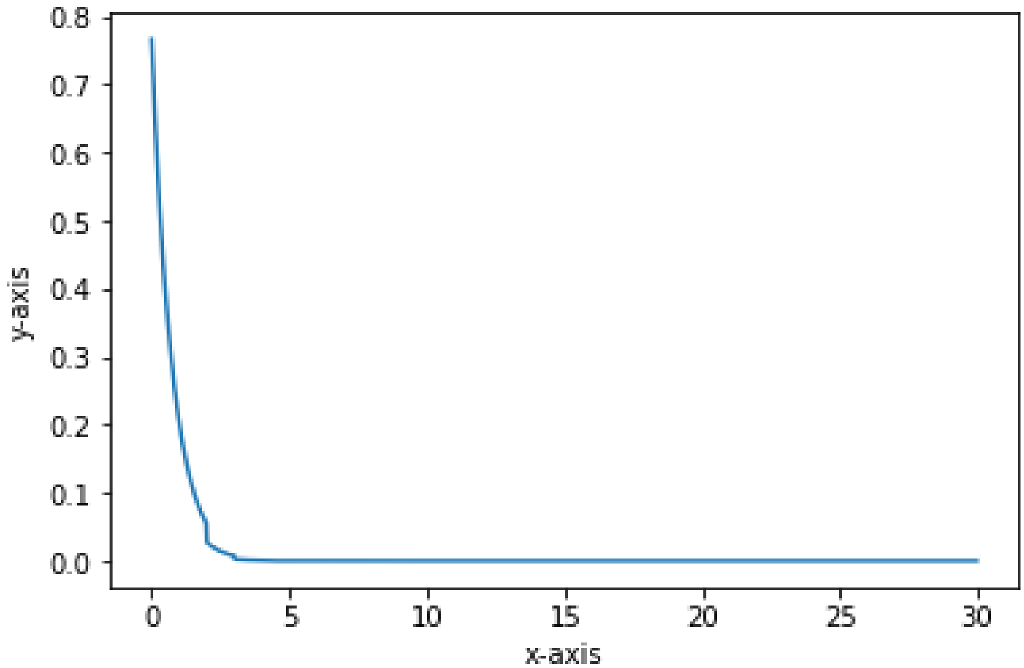




0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
y-axis
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
x-axis



0.25
0.20
0.15
0.10
0.05
0.00
y-axis
−5
0
5
10
15
20
25
30
x-axis



Mean petal length: 3.7586666666666693
Median petal length: 4.4
Standard deviation petal length: 1.7585291834055201
petal length variance: 3.0924248888888854
Skewness petal length: 1.0940961447532118
Kurtosis petal length: 1.6046406978602903