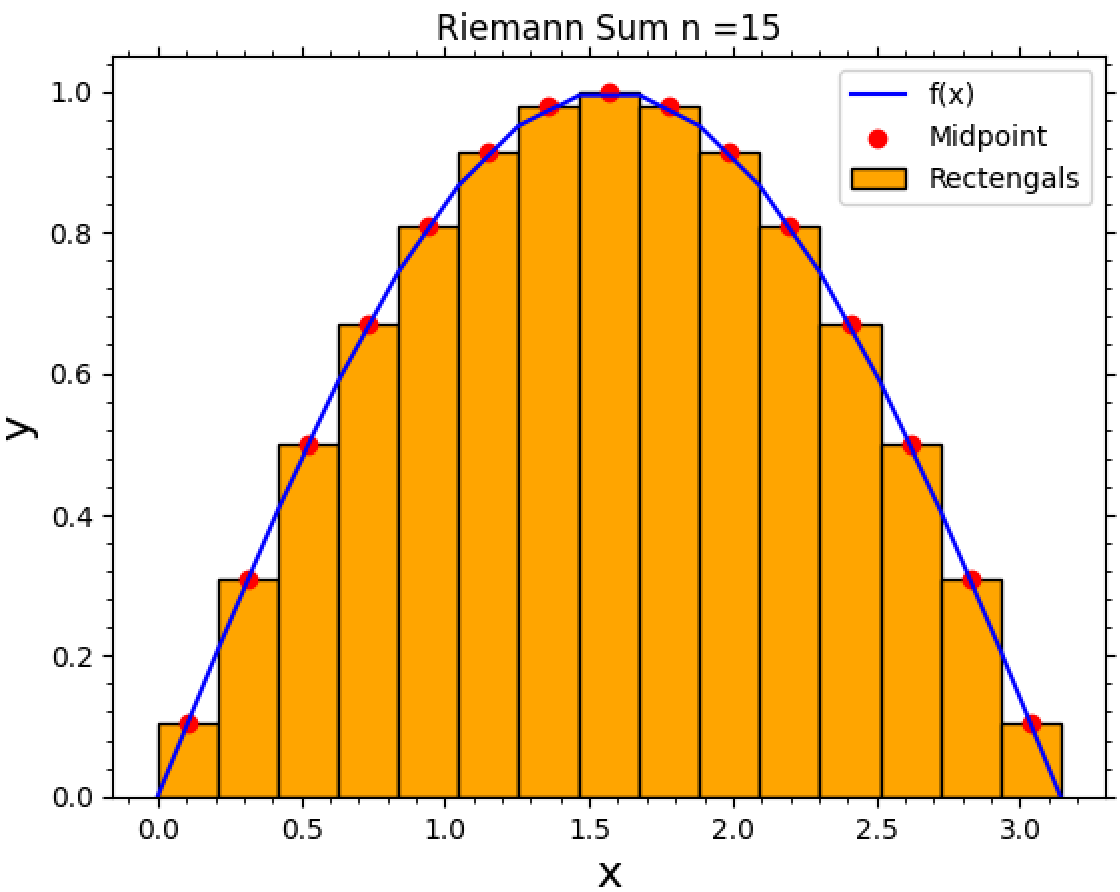
In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import AutoMinorLocator
import pandas as pd
import matplotlib.ticker as ticker
# import the lib we need later
```

In [ ]:
```
# Problem 1
# a)
def riemann_sum(func, a, b, n=15, plot=True):
    """
    Compute the Riemann sum of func(x) over the interval [a, b] with n subinterv

    Parameters
    ----------
    func : callable
        The function to integrate, with calling sequence func(x).
    a : float
        The lower limit of integration.
    b : float
        The upper limit of integration.
    n : int, optional
        The number of subintervals to use. Default is 15.
    plot : bool, optional
        If True, plot the function func(x) over the interval [a, b] (using a fin
        used to compute the Riemann sum and the midpoints.  Default is True.

    Returns
    -------
    answer : float
        The estimate of the integral of func(x) over the interval [a, b].

    """
    x = np.linspace(a,b,n+1)
    # this is the total amout of points we need to get from the range.
    dx = (b-a)/n
    # dx this the width of the riemann sum
    x_mid = (x[:-1]+x[1:])/2
    # find the mid point we need by add the x[n] and x[n+1] for all n belong to
    y = func(x_mid)
    # find all the y at each x we need
    if (plot==True):
        # this program will run if plot is true
        plt.bar(x_mid,y,dx,edgecolor = 'k',facecolor = 'orange',label = 'Recteng
        plt.plot(x,func(x),c = 'blue',label = 'f(x)')
        plt.scatter(x_mid,y,c='red', label = 'Midpoint')
        plt.gca().xaxis.set_minor_locator(AutoMinorLocator(5))
        plt.gca().yaxis.set_minor_locator(AutoMinorLocator(5))
        plt.xlabel('x', fontsize=16)
        plt.ylabel('y', fontsize=16)
        plt.title('Riemann Sum n ={}'.format(n))
        plt.gca().tick_params(which='both', top=True, right=True)
        plt.legend(loc='best')
        # use the plot lib to plot the graph we need, coped from lecture notes
    return sum(y)*dx
    # return the integral, since for each dx, the integral is y*dx, due to
```

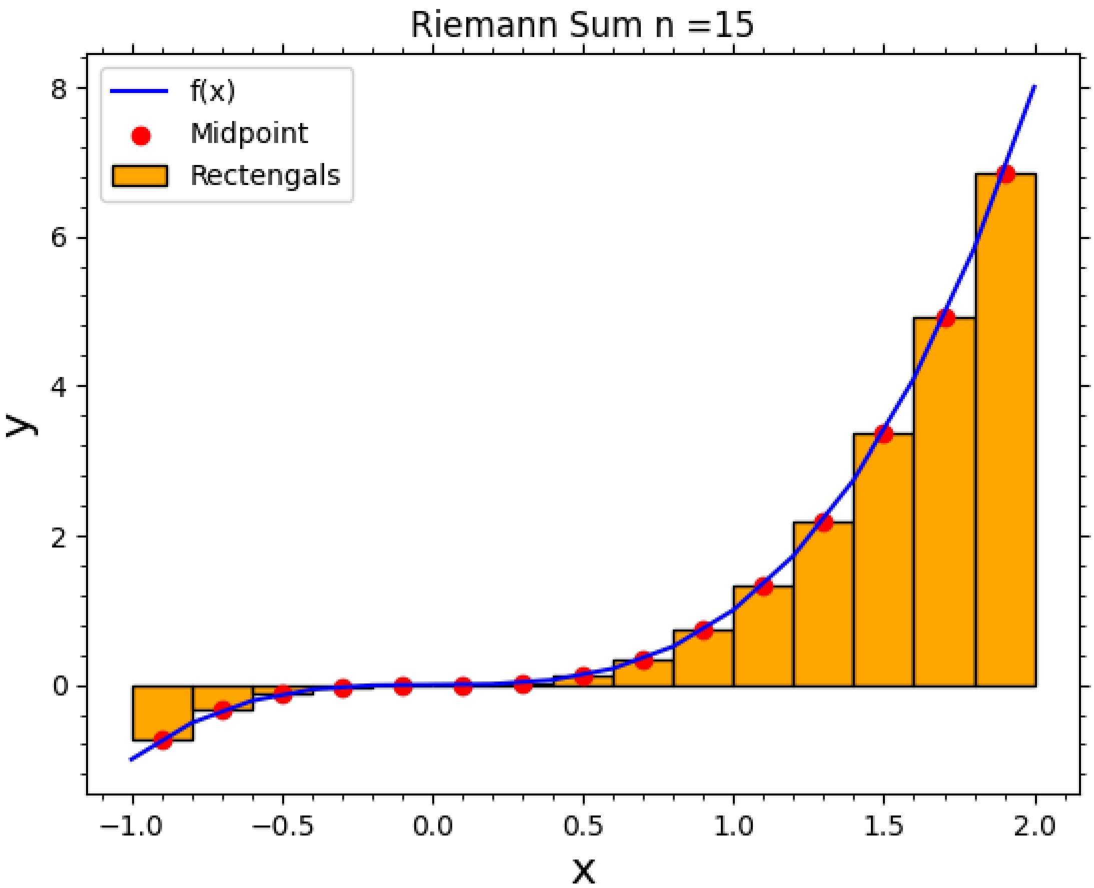
In [ ]:
```
# b)
n_result = riemann_sum(np.sin,0,np.pi)
# this is the numberic result for the function
e_result = 2
# this is the anaylical resulr from the problem
error = abs(n_result - e_result)
# the error is defined as the defined as the abs difference between numric resul
print(n_result,e_result,error)
# prin the result to the screen in the order of numerical result, analytical res
```

2.0036600911565396 2 0.0036600911565396466



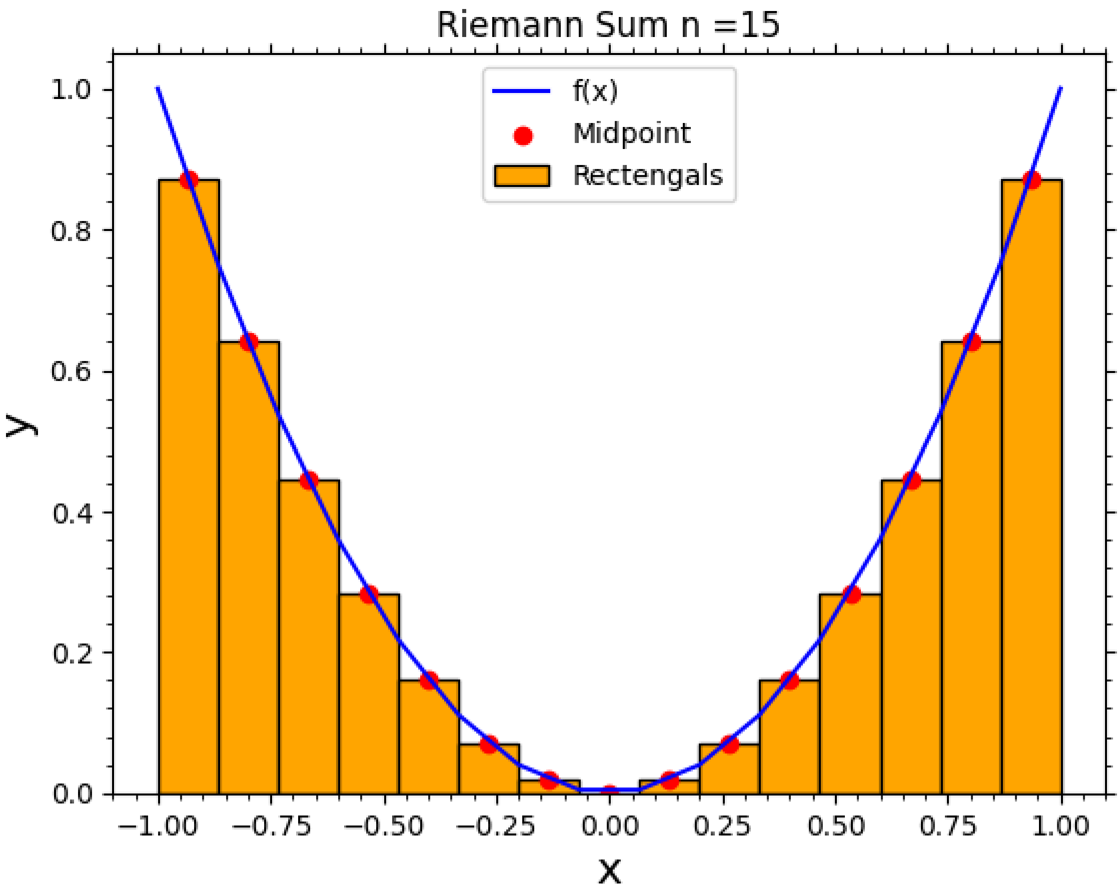
In [ ]:
```
def cube(x):
    return x**3
# define the function we need for this problem, which is y=x^3
n_result = riemann_sum(cube,-1,2)
# use the function fo calculate the numeric result
e_result = 15/4
# the analytical result from the problem
error = abs(n_result - e_result)
# error calculate
print(n_result,e_result,error)
# print the result to the sreen.
```

3.7350000000000017 3.75 0.014999999999998348



In [ ]:
```
def squre(x):
    return x**2
# define the the squre function as asked
riemann_sum(squre,-1,1)
# plot the graph as asked.
```

Out[ ]: 0.6637037037037038

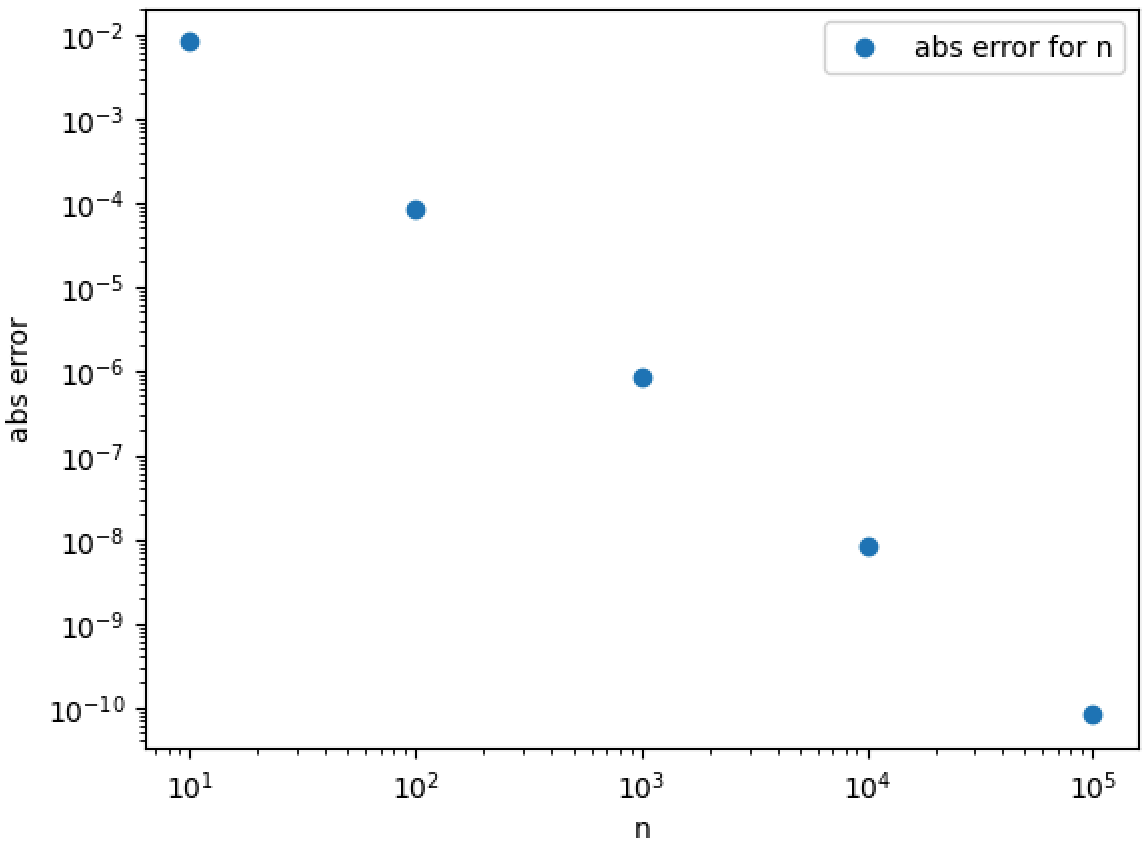


In [ ]:

```
# c)
def abs_error(func,a_result,a,b,n=np.array([10,100,1000,10000,100000],int)):
    # create a function for calculate the error
    answer = np.zeros(len(n))
    # create an array to store the answer of each n
    for i in range(len(n)):
        # loop over n
        answer[i] = abs(a_result-riemann_sum(func,a,b,n[i],False))
        # for each n, we calculate the abs error with the function written above
    return answer

plt.scatter([10,100,1000,10000,100000],abs_error(np.sin,2,0,np.pi),label = 'abs
# plot the scatter graph
plt.xscale('log')
plt.yscale('log')
# change the scale to log
plt.xlabel('n')
plt.ylabel('abs error')
plt.legend()
# show name of axis and label
```

Out[ ]:

```
<matplotlib.legend.Legend at 0x14ef9a6a410>
```



from the diagram we can find out that the slope in log scale is -2.which means that in the form of eaquation given in the prblem, A = -2

Problem 2

a)

For function $F(\nu) = \frac{2\pi h\nu^3}{c^2}\frac{1}{\exp(h\nu/k_\mathrm{B}T)-1}$, we can use taylor expansion for $\exp(h\nu/k_\mathrm{B}T)$ centered at 0 is apprixmately $1+\frac{h\nu}{k_\mathrm{B}T}+\cdots$ and we know that $h\nu \ll k_\mathrm{B}T$, $\exp(h\nu/k_\mathrm{B}T) \approx 1+\frac{h\nu}{k_\mathrm{B}T}$. The origin function can be rewrite as $F(\nu) \approx \frac{2\pi h\nu^3}{c^2}\frac{1}{1+h\nu/k_\mathrm{B}T-1} = \frac{2\pi h\nu^3}{c^2}k_\mathrm{B}T/h\nu = \frac{2\pi\nu^2 k_\mathrm{B}T}{c^2}$. From the function we can find out that $F(\nu) \sim \nu^2$

In [ ]:
```
# problem 2
# b)
def F_function(frequency,temperature):
    # the function of the blackbody radiation
    c = 299792458
    kb = 1.380649e-23
    h = 6.62607015e-34
    # set constant
    return 2*np.pi*h*(frequency**3)/(c**2)*1/(np.exp(h*frequency/(kb*temperature
    # return the fucntion

def Approximate_F(frequency,temperature):
    # the function of the approximation of blackbody raidtion
    c = 299792458
    kb = 1.380649e-23
    # set constant
```

```
    return 2*np.pi*(frequency**2)*kb*temperature/(c**2)
    # return the value of the function

def flux_F(func,temperature,frequency_low=1e3,frequency_high=1e18,n=5000):
    # the function to calculate the flux of function put into with given tempera
    # the strating point and end point of frequcy is chosen by testing
    dv = np.logspace(np.log10(frequency_low),np.log10(frequency_high),n)
    # generate all the points which is even distributed on a log graph
    flux = np.zeros(n-1)
    # set the array to store the flux
    for i in range(n-1):
        # gonging over the series to calculate the flux
        flux[i] = (func(dv[i],temperature) + func(dv[i+1],temperature))*(dv[i+1]
        # since the flux is denfied as an integral of the function, thus, I am u
    return flux,dv
    # return the flux and the point of x
```

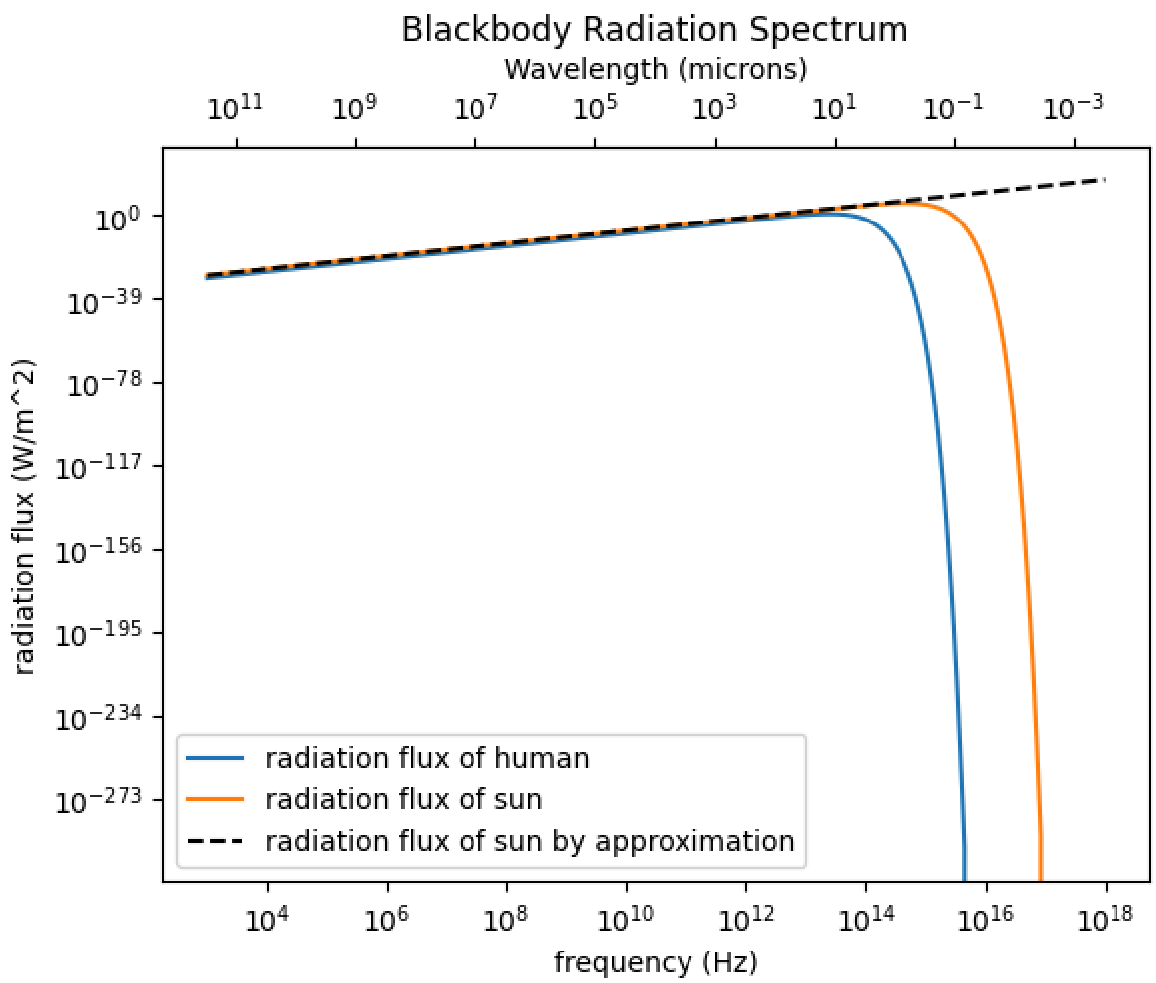
In [ ]:

```
# c)
line_human,x = flux_F(F_function,310.15)
line_sun,x = flux_F(F_function,5778)
line_sun_approxiamtion,x = flux_F(Approximate_F,5778)
# get the x,and y of the plot, but since we are using the same range and points

plt.plot(x[:-1],line_human,label = 'radiation flux of human')
plt.plot(x[:-1],line_sun, label = 'radiation flux of sun')
plt.plot(x[:-1],line_sun_approxiamtion,'k--',label = 'radiation flux of sun by a
# plot the line as asked

plt.xscale('log')
plt.yscale('log')
# change x and y to the log scale

plt.xlabel('frequency (Hz)')
plt.ylabel('radiation flux (W/m^2)')
plt.title('Blackbody Radiation Spectrum')
plt.gca().secondary_xaxis('top', functions=(lambda nu: 299792458 / nu / 1e-6, la
# show the notes asked to show

plt.legend()
# print the legend
```

```
C:\Users\botao\AppData\Local\Temp\ipykernel_3840\2146089864.py:9: RuntimeWarning:
overflow encountered in exp
  return 2*np.pi*h*(frequency**3)/(c**2)*1/(np.exp(h*frequency/(kb*temperature))-
1)
```

Out[ ]: <matplotlib.legend.Legend at 0x14efa30a110>

```
C:\Users\botao\AppData\Local\Temp\ipykernel_3840\3804959965.py:19: RuntimeWarnin
g: divide by zero encountered in divide
  plt.gca().secondary_xaxis('top', functions=(lambda nu: 299792458 / nu / 1e-6, l
ambda lam: 299792458 / (lam * 1e-6))).set_xlabel('Wavelength (microns)')
```



d) From the graph, it is easy to find out that the approximation is good for given range. it is also cleat that the radiation fulx of human is always smaller than the sun at any given frequency

In [ ]:
```
# problem 3
# a)
df = pd.read_csv('data\census_income_data_2022.csv')
# use pandas to read the file
df.describe()
# use describe as asked in the problem
```

Out[ ]:

	WGTP	NPF	HINCP	Unnamed: 3
count	1.611650e+06	1.611650e+06	1.611650e+06	0.0
mean	8.920851e+01	2.053273e+00	7.552088e+04	NaN
std	8.848104e+01	1.344968e+00	1.237117e+05	NaN
min	0.000000e+00	1.000000e+00	-6.000000e+04	NaN
25%	3.700000e+01	1.000000e+00	1.470000e+04	NaN
50%	6.700000e+01	2.000000e+00	5.700000e+04	NaN
75%	1.120000e+02	3.000000e+00	1.142000e+05	NaN
max	2.339000e+03	1.800000e+01	2.481200e+06	NaN

In [ ]:
```
# b)
df_WDGP = np.array(df['WGTP'])
df_NPF = np.array(df['NPF'])
df_HINCP = np.array(df['HINCP'])
# use three arrays to store the data

N_households = sum(df_WDGP)
print(N_households)
# calculate the total number of households

N_population = sum(df_WDGP*df_NPF)
print(N_population)
# calculate the total number of population
```

```
143772895
314293387
```

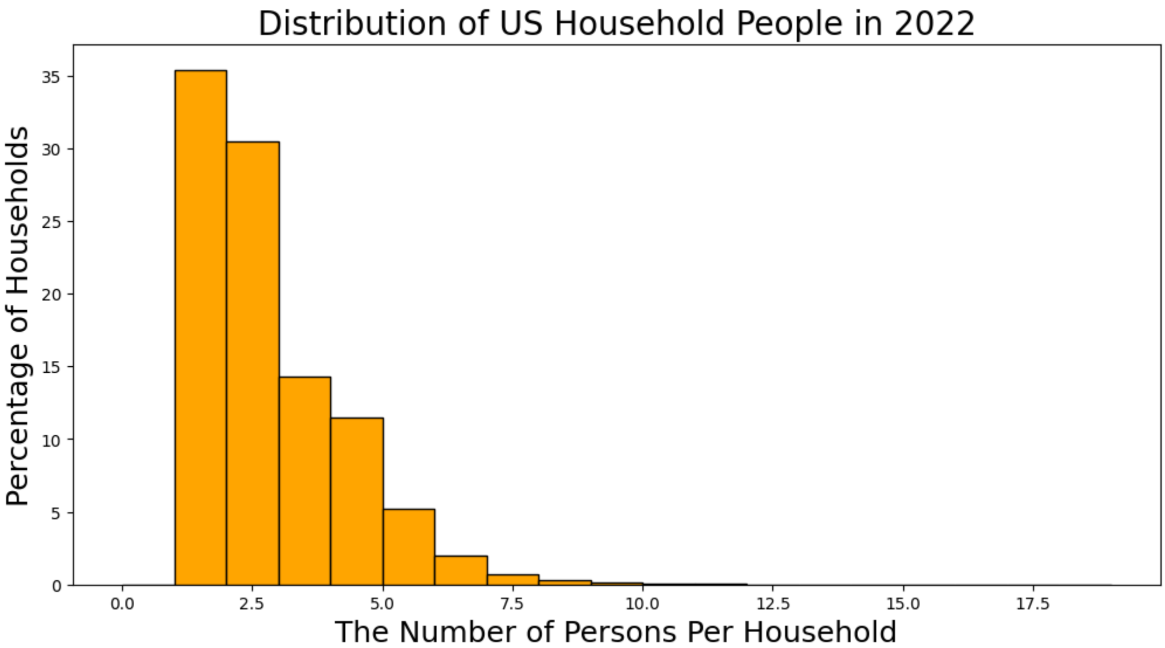
The data looks good, since households looks like 2 to 3 times smaller than total population.

In [ ]:
```
# c)
df_modified = df[:][df_HINCP>0]
# use boolean index to find all the value with HINCP greater than 0
N_households = sum(df_modified['WGTP'])
N_population = sum(df_modified['WGTP']*df_modified['NPF'])
print(N_households)
print(N_population)
# calculate the households and population and print them to the screen
```

```
127970381
297334150
```

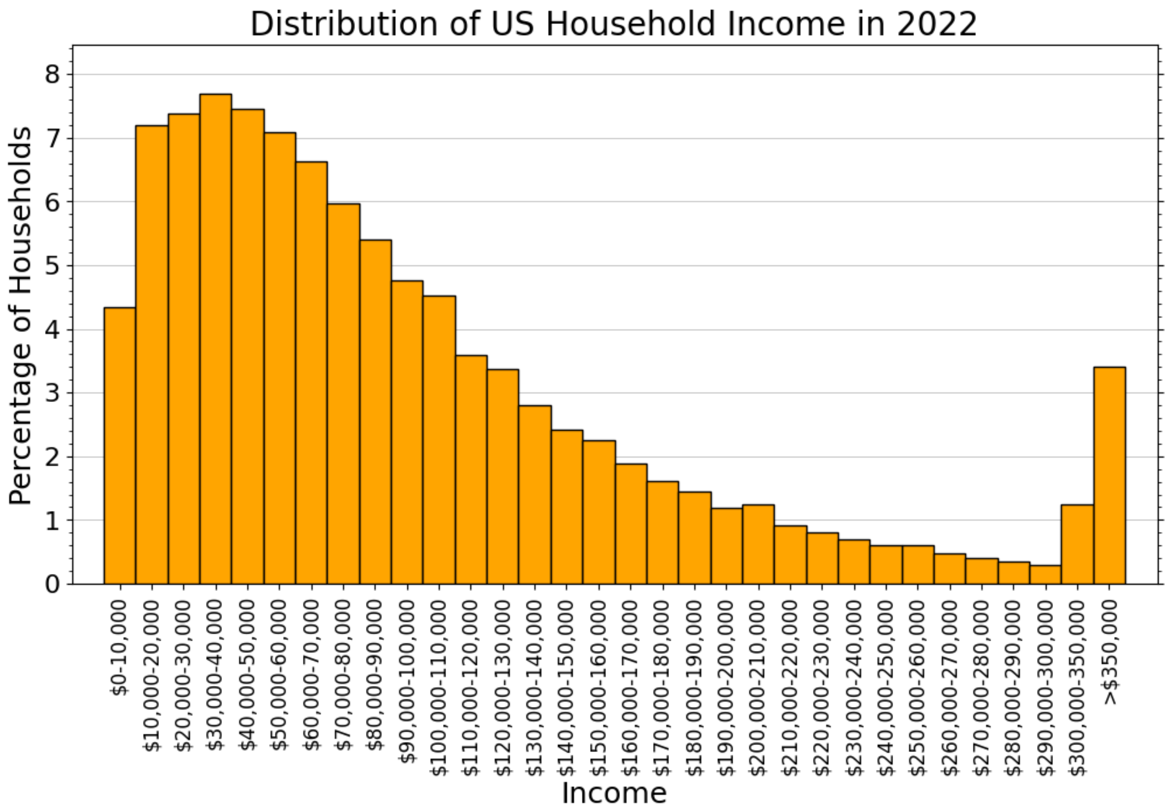
In [ ]:
```
# d)
# Filter out all people with incomes less than or equal to zero
filtered_df = df[df['HINCP'] > 0]

# Construct the bins
bins = np.array(range(int(filtered_df['NPF'].values.min())-1,int(filtered_df['NP

bins_for_plot = np.array(range(int(filtered_df['NPF'].min())-1,int(filtered_df['

# Compute the histogram using np.histogram
hist, bin_edges = np.histogram(filtered_df['NPF'], bins=bins, weights=filtered_d

# Normalize the histogram by the total weight
total_weight = np.sum(hist)
hist_normalized = (hist / total_weight)*100.0


# Create a figure and axes
figsize = (12, 6)
fig, ax = plt.subplots(figsize=figsize)

# Plot the histogram as a bar chart
ax.bar(bins_for_plot[:-1], hist_normalized, width=np.diff(bins_for_plot), align=

# Set labels and title
ax.set_xlabel('The Number of Persons Per Household', fontsize=18)
ax.set_ylabel('Percentage of Households', fontsize=18)
```

```
ax.set_title('Distribution of US Household People in 2022', fontsize=20)


# Force the y-axis major ticks to be five units apart
ax.yaxis.set_major_locator(ticker.MultipleLocator(5))
```



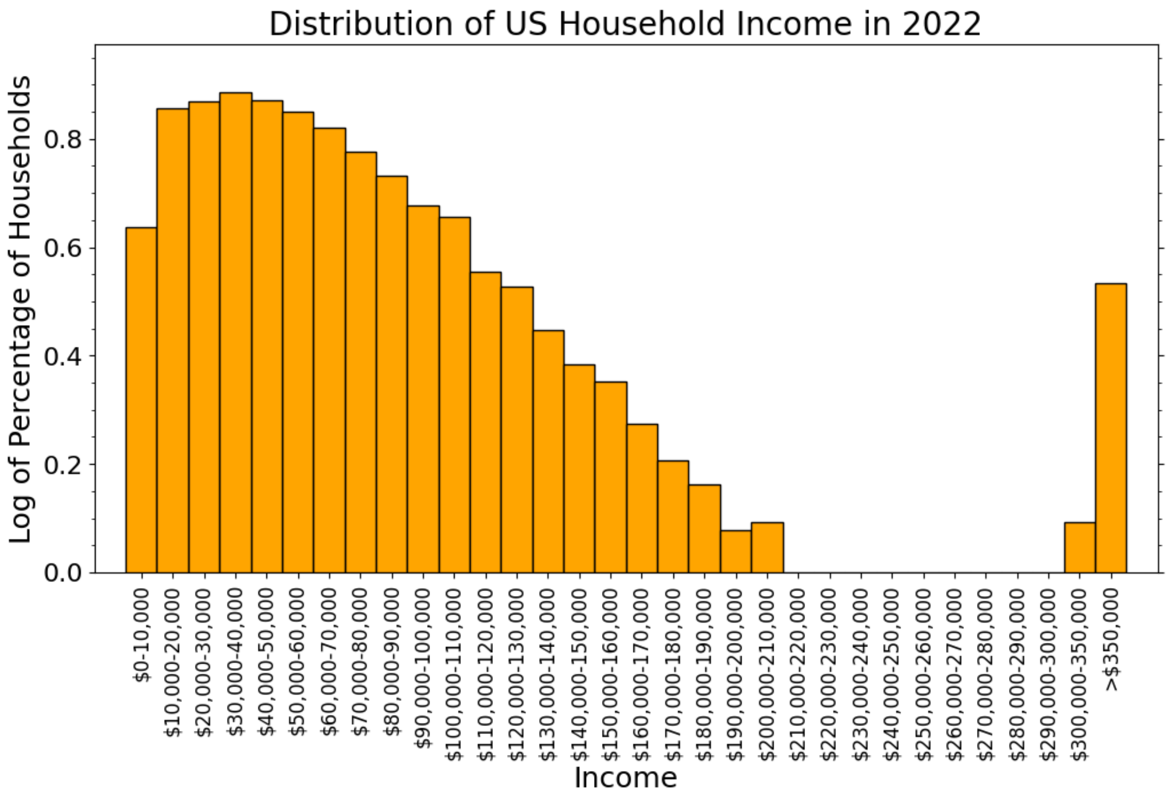
In [ ]:
```
# e)

# Construct the bins
bin_width = 10000
bins = np.array([i for i in range(0, 300000-bin_width + 1, bin_width)] + [300000

bins_for_plot = np.arange(0, 300000 + 2*bin_width+1, bin_width)

# Compute the histogram using np.histogram
hist, bin_edges = np.histogram(filtered_df['HINCP'], bins=bins, weights=filtered

# Normalize the histogram by the total weight
total_weight = np.sum(hist)
hist_normalized = (hist / total_weight)*100.0

# Create a figure and axes
figsize = (12, 6)
fig, ax = plt.subplots(figsize=figsize)

# Plot the histogram as a bar chart
ax.bar(bins_for_plot[:-1], hist_normalized, width=np.diff(bins_for_plot), align=

# Set labels and title
ax.set_xlabel('Income', fontsize=18)
ax.set_ylabel('Percentage of Households', fontsize=18)
ax.set_title('Distribution of US Household Income in 2022', fontsize=20)

# Create labels for the bins
labels = ['${:,}-{:,}'.format(int(bin_edges[i]), int(bin_edges[i+1])) for i in r

bins_for_plot_center = (bins_for_plot[1:] + bins_for_plot[:-1]) / 2
# Set the x-ticks and x-tick labels
ax.set_xticks(bins_for_plot_center)
ax.set_xticklabels(labels, rotation='vertical')
```

```
# Force the y-axis major ticks to be one unit apart
ax.yaxis.set_major_locator(ticker.MultipleLocator(1))

# Add minor ticks to the y-axis
ax.yaxis.set_minor_locator(ticker.AutoMinorLocator())
# Set ticks on both sides of the y-axis
ax.yaxis.set_ticks_position('both')
# Hide the labels of the second y-axis
ax.yaxis.set_tick_params(labelleft=True, labelright=False)
ax.set_xlim((-10000, 300000 + 3*bin_width))
ax.set_ylim((0,1.1*max(hist_normalized)))

# Manually draw the grid lines
for y in ax.get_yticks():
    ax.hlines(y, xmin=ax.get_xlim()[0], xmax=ax.get_xlim()[1], colors='black', l

# Set the x-ticks and x-tick labels
ax.set_xticks(bins_for_plot_center)
ax.set_xticklabels(labels, rotation='vertical')

# Increase the font size of the x-tick labels
ax.tick_params(axis='x', labelsize=12)
# Increase the font size of the y-tick labels
ax.tick_params(axis='y', labelsize=16)
```



In [ ]:
```
# f)

# Construct the bins
bin_width = 10000
bins = np.array([i for i in range(0, 300000-bin_width + 1, bin_width)] + [300000

bins_for_plot = np.arange(0, 300000 + 2*bin_width+1, bin_width)

# Compute the histogram using np.histogram
```

```
hist, bin_edges = np.histogram(filtered_df['HINCP'], bins=bins, weights=filtered

# Normalize the histogram by the total weight
total_weight = np.sum(hist)
hist_normalized = (hist / total_weight)*100.0


# Create a figure and axes
figsize = (12, 6)
fig, ax = plt.subplots(figsize=figsize)

# Plot the histogram as a bar chart
ax.bar(bins_for_plot[:-1], np.log10(hist_normalized), width=np.diff(bins_for_plo


# Set labels and title
ax.set_xlabel('Income', fontsize=18)
ax.set_ylabel('Log of Percentage of Households', fontsize=18)
ax.set_title('Distribution of US Household Income in 2022', fontsize=20)

# Create labels for the bins
labels = ['${:,}-{:,}'.format(int(bin_edges[i]), int(bin_edges[i+1])) for i in r

bins_for_plot_center = (bins_for_plot[1:] + bins_for_plot[:-1]) / 2
# Set the x-ticks and x-tick labels
ax.set_xticks(bins_for_plot_center)
ax.set_xticklabels(labels, rotation='vertical')


# Add minor ticks to the y-axis
ax.yaxis.set_minor_locator(ticker.AutoMinorLocator())
# Set ticks on both sides of the y-axis
ax.yaxis.set_ticks_position('both')
# Hide the labels of the second y-axis
ax.yaxis.set_tick_params(labelleft=True, labelright=False)
ax.set_xlim((-10000, 300000 + 3*bin_width))
ax.set_ylim((0,1.1*max(np.log10(hist_normalized))))


# Set the x-ticks and x-tick labels
ax.set_xticks(bins_for_plot_center)
ax.set_xticklabels(labels, rotation='vertical')

# Increase the font size of the x-tick labels
ax.tick_params(axis='x', labelsize=12)
# Increase the font size of the y-tick labels
ax.tick_params(axis='y', labelsize=16)
```



From e) and f) the graph looks similar, but f) makes the difference greater. Thus, though they both represent a similair result, the normal dispaly of y is better than use the log y.

In [ ]:
```
# g)
for i in range(len(bins_for_plot[:-1])):
    # going through all bins
    print(i,bins_for_plot[i],bins_for_plot[i+1],hist[i])
    # print the index, the left, the right, and the number of house included in
```

```
0 0 10000 5550111
1 10000 20000 9204494
2 20000 30000 9452366
3 30000 40000 9840860
4 40000 50000 9531309
5 50000 60000 9067130
6 60000 70000 8475399
7 70000 80000 7645462
8 80000 90000 6913074
9 90000 100000 6080467
10 100000 110000 5789354
11 110000 120000 4590876
12 120000 130000 4310239
13 130000 140000 3577436
14 140000 150000 3096173
15 150000 160000 2879292
16 160000 170000 2410918
17 170000 180000 2062878
18 180000 190000 1856233
19 190000 200000 1529964
20 200000 210000 1587479
21 210000 220000 1162189
22 220000 230000 1038762
23 230000 240000 887054
24 240000 250000 764497
25 250000 260000 774517
26 260000 270000 598254
27 270000 280000 517059
28 280000 290000 452530
29 290000 300000 376443
30 300000 310000 1583782
31 310000 320000 4363780
```

In [ ]:
```
# f)
print(sum(hist_normalized[bins_for_plot[:-1]>500000]))
# print the value asked in the problem use boolean index
```

```
0
```

In [ ]:
```
# g)

print(len(df_modified[df_modified['HINCP']>500000])/len(df_modified)*100)
# find the precentage using the original data
```

```
1.8774098782055204
```

There is actually 1.8% of households have a income over 500000. However, when we use the data from histogram, since tere is only value for greater than 350000, 500000 is over the discription of the data. Thus it is reasonable to get the invalid number 0.