Here are the data for 2.31 and 2.32 inputted as numpy arrays: each list is presented as its own array (rather than arrays with multiple lists in them).

Note: this could also be accomplished by data entry into a Google Sheet, followed by exporting as a .csv file and then importing using standard Python importing commands (which we might use in other locations). This method of manual entry just ends up being the easiest thing to do here because we are working with textual data that came from a table in a book.

As is typical, we need to start with packages that we will use here: numpy for managing the data and matplotlib for making plots.

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import math
from scipy import constants
import sympy as sy
```

Now here are the data: there are four columns so I will make four separately named arrays.

In [ ]:
```
Pressures = np.array([0.500,2.00,10.00,20.00,40.00,60.00,80.00,100.0,120.0,160.0,200.0,
Vmolarethane = np.array([83.076,20.723,4.105,2.028,0.9907,0.6461,0.4750,0.3734,0.3068,0
Vmolarargon = np.array([40.506,10.106,1.999,0.9857,0.4795,0.3114,0.2279,0.1785,0.1462,0
```

Now let's check that we didn't lose any points by checking the length of each list/array

In [ ]:
```
np.size(Pressures)
np.size(Vmolarethane)
np.size(Vmolarargon)
```

Out[ ]:
```
19
```

Ok so that works; we could go on tweaking limits of axes, etc. to make things look nicer but the plots are working and the data are intact.

Now we need to manipulate the existing data to make a plot of either Z vs the reduced molar volume (V/Vc) or Z vs the reduced pressure (P/Pc). So we need to make new arrays from the existing ones by doing relatively simple math manipulations to yield Z, Vreduced, and Preduced for each gas. For this to work we need to define Z and also look up the Vc and Pc for each gas.

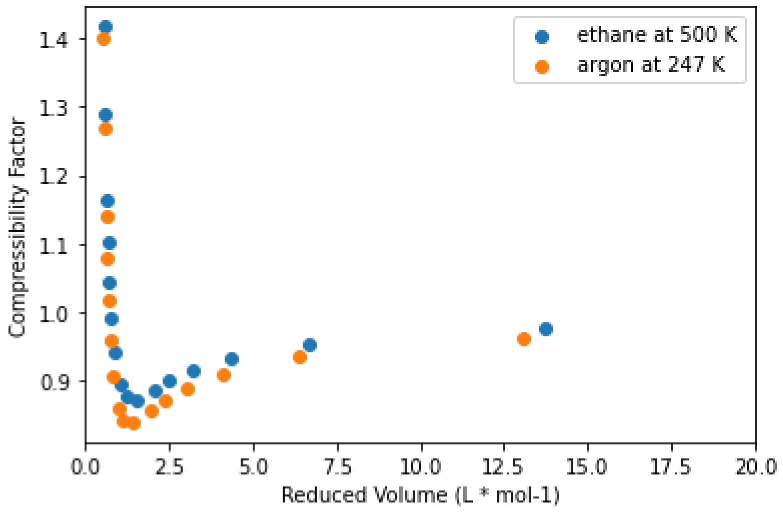
In [ ]:
```
# Define necessary constants first
R = 8.314e-2 #L bar /mol K
Vc_ethane = 0.1480 #L/mol
Pc_ethane = 48.714 #bar
Vc_argon = 0.07530 #L/mol
Pc_argon = 48.643 #bar

# Now define Z, as before, for any P and Vbar and T
def Z(P,Vbar,T): # This is the definition of Z based on P and Vbar
  return P*Vbar/(R*T)
```

In [ ]:
```
#now make arrays of P/Pc and V/Vc, and Z for each gas
Zethane = Z(Pressures,Vmolarethane,500)
Zargon = Z(Pressures,Vmolarargon,247)
Preduced_argon = Pressures/Pc_argon
Preduced_ethane = Pressures/Pc_ethane
Vreduced_argon = Vmolarargon/Vc_argon
Vreduced_ethane = Vmolarethane/Vc_ethane
```

Solution of 2-31 is found below where we graph Compressibility Factor versus Volume Reduced and we get two gases that seem to behave relatively the same at their critical temperatures.

In [ ]:
```
plt.scatter(Vreduced_ethane,Zethane,label='ethane at 500 K')
plt.scatter(Vreduced_argon,Zargon,label='argon at 247 K')
# plt.figure(figsize=(16,9))
plt.xlim(0,20)
plt.xlabel('Reduced Volume (L * mol-1)')
plt.ylabel('Compressibility Factor')
plt.legend()
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x1e5cbd62b20>
```



So why does this show a "law of corresponding states"? That's something that you will need to discuss amongst yourselves and figure out for your own notes. The plot here is an essential part of the solution to problem 2-31. Now do 2-32, and then try 2-33 and/or 2-34 where you compare data to an empirical function.

Answer: This shows a "law of corresponding states" because of their similar natures at the critical temperature. As Reduced Volume and Pressure increases, both of these gases at their critical temperatures perform the same.

Note that we are not doing any fitting or anything here at this point, but that is also something that could be assessed with all of these arrays working successfully.

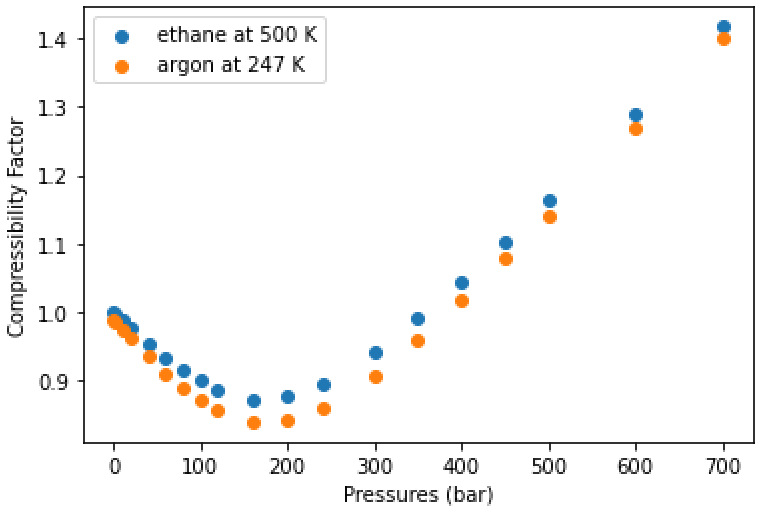
Solution for 2-32 can be found below. We plot Compressibility Factor versus Pressure in this question.

In [ ]:

```
plt.scatter(Pressures,Zethane,label='ethane at 500 K')
plt.scatter(Pressures,Zargon,label='argon at 247 K')
# plt.figure(figsize=(16,9))
# plt.xlim(0,20)
plt.legend()
plt.xlabel('Pressures (bar)')
plt.ylabel('Compressibility Factor')
```

Out[ ]:

```
Text(0, 0.5, 'Compressibility Factor')
```



In [ ]:

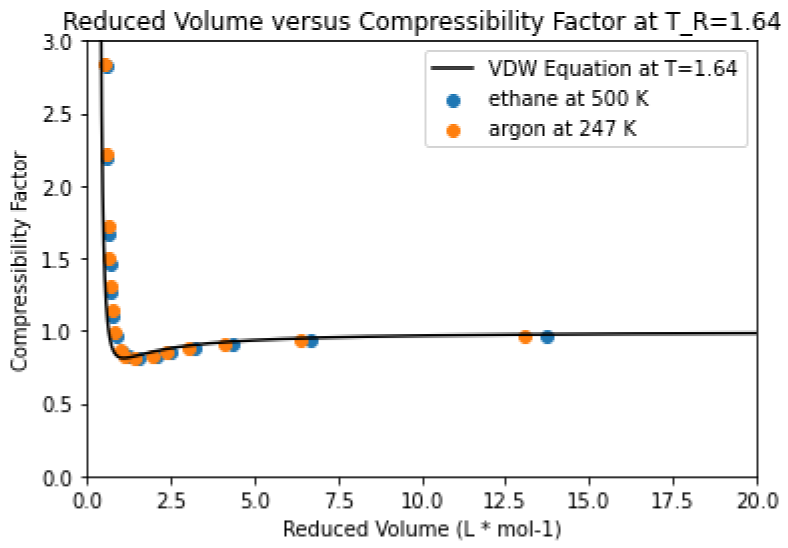
```
"""
Below, we define a bunch of functions that we calculate the compressibility factor and
the Van Der Waal equation as one function with various amounts of inputs depending on t

Continuing on though, we define various constants along with four functions:
1) P_vdw_ethane outputs Pressure for Ethane given Vbar and Temperature
2) P_vdw_argon outputs Pressure for Ethane given Vbar and Temperature
3) Z_vdw_bad outputs Compressibility Factor through calculate of Pressure in terms of V
4) Z_vdw_reduced outputs Compressibility Factor given a certain critical point temperat
"""

# Constants for Argon and Ethane for the Van Der Waal Equation
a_argon = 1.3483
a_ethane = 5.5818
b_argon = 0.031830
b_ethane = 0.065144

V_reduced_any = np.linspace(0.401,100,100000) # Defining an almost continous amount of

def P_vdw_ethane(Vbar,T): # Van der Waal equations for Pressure and Z
    return ((R*T)/(Vbar-b_ethane))-(a_ethane/(Vbar**2))

def P_vdw_argon(Vbar,T): # Van der Waal equations for Pressure and Z
    return ((R*T)/(Vbar-b_argon))-(a_argon/(Vbar**2))

def Z_vdw_bad(P,Vbar,T): # Van Der Waal Equation where we input Pressure which is in te
    return (P*(Vbar))/(R*T)

def Z_vdw_reduced(Vbar,T): # Van Der Waal Equation where we consider Compressibility Fa
    return Vbar/(Vbar-1/3)-9/(8*Vbar*T)
```

Solution for 2-33:

We are graphing Compressibility Factor of Ethane and Argon versus their Reduced Volume. We then continue upon showing their similar behavior be graphing the Compressibility Factor of any gas at a critical temperature of $T_R = 1.64$

In [ ]:
```
plt.scatter(Vreduced_ethane,Z_vdw_bad(P_vdw_ethane(Vmolarethane,500),Vmolarethane,500),
plt.scatter(Vreduced_argon,Z_vdw_bad(P_vdw_argon(Vmolarargon,247),Vmolarargon,247),labe
plt.plot(V_reduced_any,Z_vdw_reduced(V_reduced_any,1.64),label='VDW Equation at T=1.64'
# plt.figure(figsize=(16,9))
plt.xlim(0,20)
plt.ylim(0,3)
plt.xlabel('Reduced Volume (L * mol-1)')
plt.ylabel('Compressibility Factor')
plt.title('Reduced Volume versus Compressibility Factor at T_R=1.64')
plt.legend()
```

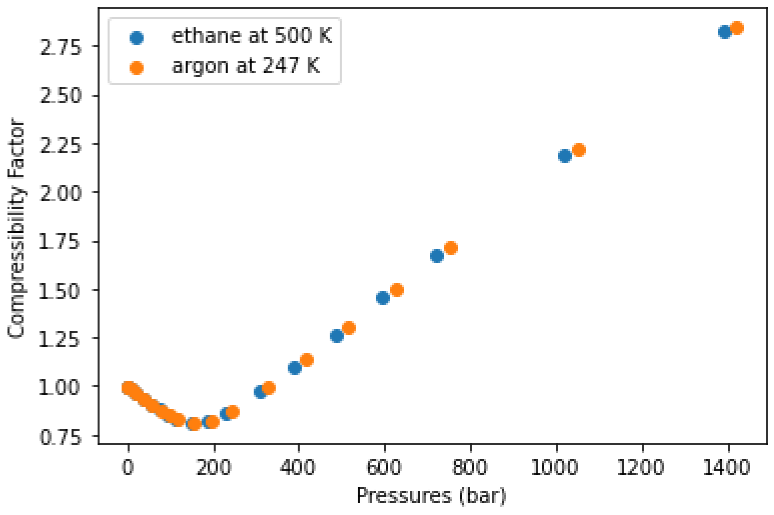
Out[ ]:
```
<matplotlib.legend.Legend at 0x1e5ccec2ee0>
```



The below graph is unneccessary, but I like the fact that I could do it haha but this is unneccesary for my solutions.

In [ ]:
```
"""
Below Information is not part of the PS, but I did this in class showing how Compressib
the Ideal Gas Law.
"""

Pressure_any = np.linspace(0.05,1500,100000)
plt.scatter(P_vdw_ethane(Vmolarethane,500),Z_vdw_bad(P_vdw_ethane(Vmolarethane,500),Vmo
plt.scatter(P_vdw_argon(Vmolarargon,247),Z_vdw_bad(P_vdw_argon(Vmolarargon,247),Vmolara
# plt.figure(figsize=(16,9))
# plt.xlim(0,20)
plt.legend()
plt.xlabel('Pressures (bar)')
plt.ylabel('Compressibility Factor')
```

Out[ ]:
```
Text(0, 0.5, 'Compressibility Factor')
```



Solution for 2-34:

Below, we complete two tasks: define functions for the Redlich Equation and plot the Redlich calculations for Compressibility Factor vs. Reduced Volume. The functions do the following:

1) P_Redlich takes in four inputs: Vbar, Temperature, and the Redlich Constants found in the book. It ouputs the Pressure calculated in terms of Vbar for the Redlich Equation

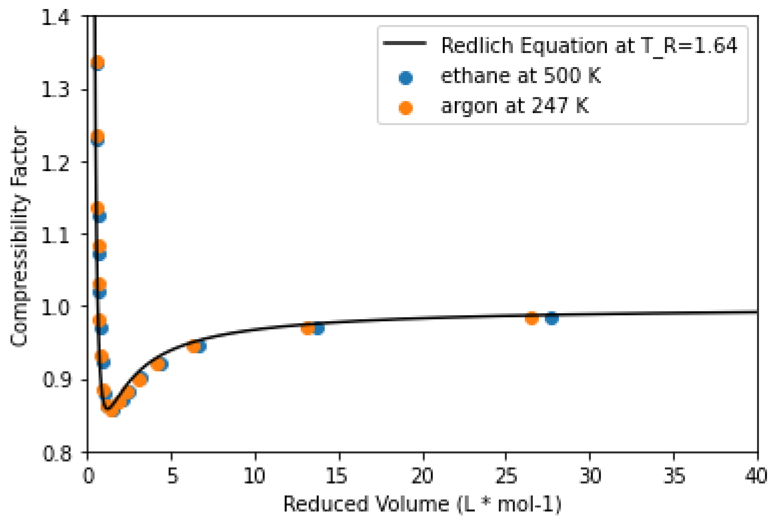
2) Z_Redlich is just the Ideal Gas Law equivalent where it assumes that it's input Pressure is in terms of the Redlich parameters.

3) Z_redlich_reduced calculates the Compressibility Factor of arbitrary gases given a particularly critical temperature.

We then use these functions to graph Reduced Volume versus Compressibility Factors calculated through the Redlich Equation. On top of those scatter plots, we plot a continuous graph that shows the Compressibility Factor of the Redlich Equation given at $T_R = 1.64$. This verifies the Redlich Equation because this shows a "law of corresponding states" because of their similar natures at the critical temperature. As Reduced Volume and Pressure increases, both of these gases at their critical temperatures perform the same.

In [ ]:
```
def P_redlich(Vbar,T,A,B): # Defining pressures in terms of Vbar for the Redlich Equati
    return ((R*T)/(Vbar-B))-(A/(T**(1/2)*Vbar*(Vbar+B)))
def Z_redlich(P,Vbar,T): # Compressibility Factor funciton
    return (P*(Vbar))/(R*T)
def Z_redlich_reduced(Vbar,T): # Compressibility Factor function for particularly at th
    return Vbar/(Vbar-0.25992)-1.2824/(T**(3/2)*(Vbar+0.25992))

plt.scatter(Vreduced_ethane,Z_redlich(P_redlich(Vmolarethane,500,98.831,0.045153),Vmola
plt.scatter(Vreduced_argon,Z_redlich(P_redlich(Vmolarargon,247,16.786,0.022062),Vmolara
plt.plot(V_reduced_any,Z_redlich_reduced(V_reduced_any,1.64),label='Redlich Equation at
plt.xlim(0,40)
plt.ylim(0.8,1.4)
plt.legend()
plt.xlabel('Reduced Volume (L * mol-1)')
plt.ylabel('Compressibility Factor')
```

```
Out[ ]: Text(0, 0.5, 'Compressibility Factor')
```



Calculating the vibration and rotational states of H_2 with the vibrational and rotational constants being $\hbar\omega$. The two codes of block below complete that task. The first block uses Casey's defined functions for translational energy, rotational energy, and vibrational energy, to complete the task of calculating our energies. The second block goes through calculating those values and printing the values out.

```
In [ ]:
def transenergy(n,m,L): #translational energy, particle in a box; mass in kg, box lengt
  return (n**2 * constants.Planck**2)/(8*m*L**2)

def rotenergy(J,mu,bondlength): #rotational energy in terms of reduced mass (kg) and ef
  return J*(J+1)*constants.hbar**2/(2*mu*bondlength**2)

def vibenergy(v,mu,k): #vibrational energy in terms of force constant (N/m or kg sec^2)
  return (v+0.5)*constants.Planck*(1/(2*math.pi))*sy.sqrt(k/mu)
```

Some things to note about the information provided to us (this will also be explained definitely in my HW Writeup):

C-1:

For our Vibrational Energy Function, we have the inputs k, μ, and v which is energy level. We can calculate the reduced mass μ through looking up the masses of Hydrogen, however, we must create an equation to find k. This deviation can be found in my writeup. We also have a similar problem where we must find H_2's bondlength where I write this derivation in our writeup.

```
In [ ]:
# Below I calculate the first five rotational states of H2!

mass_hydrogen = 1.6735575e-27 # kg
h2_I = constants.Planck/(8*math.pi**2 * 5832 * constants.c)

J = np.array([0,1,2,3,4]) # first five rotational states


h2_bondlength_book = 7.42e-11 # book value for bond length
```

```
h2_reduced = (mass_hydrogen * mass_hydrogen)/(mass_hydrogen + mass_hydrogen) # reduced
h2_bondlength = math.sqrt(h2_I/h2_reduced) # Calculating the bondlength given iternia a

rot_energies = rotenergy(J,h2_reduced,h2_bondlength) # calculate the first five rotatio

# Print out the first five rotational energies using Book's provided bondlength
print("Here are the first five rotational energies of H2 using the book provided bondle
i = 0
for energy in rot_energies:
    print(i,") ", energy, "J")
    i += 1

    rot_energies = rotenergy(J,h2_reduced,h2_bondlength_book)

# Print out the first five rotational energies using my calculated booklength
print("\n Here are the first five rotational energies of H2 using my calculated bondlen
i = 0
# print(rot_energies)
for energy in rot_energies:
    print(i,") ", energy, "J")
    i += 1
```

```
Here are the first five rotational energies of H2 using the book provided bondlength:
0 )  0.0 J
1 )  2.316990447778511e-21 J
2 )  6.950971343335532e-21 J
3 )  1.3901942686671065e-20 J
4 )  2.316990447778511e-20 J

 Here are the first five rotational energies of H2 using my calculated bondlength:
0 )  0.0 J
1 )  2.4139811184769626e-21 J
2 )  7.241943355430888e-21 J
3 )  1.4483886710861776e-20 J
4 )  2.4139811184769625e-20 J
```

In [ ]:

```
"""
Below are is the caclulations for the vibrational energies!
"""

v = np.array([0,1,2,3,4])
# v_obs = 440100/ constants.Planck
k = ((2*math.pi*440100*constants.c)**2)*h2_reduced
vib_energies = vibenergy(v,h2_reduced,k)

print("Here are the first five vibrational energies of H2 in Joules:")
i = 0
for energy in vib_energies:
    print(i,") ", energy, "J")
    i += 1
```

```
Here are the first five vibrational energies of H2 in Joules:
0 )  4.37117410865622e-20 J
1 )  1.31135223259687e-19 J
2 )  2.18558705432811e-19 J
3 )  3.05982187605935e-19 J
4 )  3.93405669779060e-19 J
```

C-2:

We are already using functions provided by casey to compute the rotational and vibrational levels/energies of various molecules and atoms.

However, we still need to computer the difference between $H^{35}Cl$ at $v = 0, J = 1$ and $v = 1, J = 0$.

In [ ]:

```
# Below I calculate the rotational states of H^35Cl!

mass_hydrogen = 1.6735575e-27 # kg
mass_chlorine = 5.81189e-26 # kg
molecule_I = constants.Planck/(8*math.pi**2 * 1044 * constants.c)

bondlength = 1.275e-10 #

molecule_reduced = (mass_hydrogen*mass_chlorine)/(mass_hydrogen+mass_chlorine)
J = np.array([0,1]) # the two rotational states that we care about

molecule_bondlength = math.sqrt(molecule_I/molecule_reduced)

rot_energies_molecule = rotenergy(J,molecule_reduced,molecule_bondlength)

print("Here are the two rotational energies of H35Cl that we desire:")
i=0
for energy in rot_energies_molecule:
    print(i,") ", energy, "J")
    i += 1
i = 0
```

```
Here are the two rotational energies of H35Cl that we desire:
0 )  0.0 J
1 )  4.1476989497269637e-22 J
```

In [ ]:

```
# Below I calculate the rotational states of H^35Cl!
v = np.array([0,1])
# v_obs = 440100/ constants.Planck
k = ((2*math.pi*288600*constants.c)**2)*molecule_reduced
vib_energies_molecule = vibenergy(v,molecule_reduced,k)

print("Here are the first five vibrational energies of H2 in Joules:")
i = 0
for energy in vib_energies_molecule:
    print(i,") ", energy, "J")
    i += 1
```

```
Here are the first five vibrational energies of H2 in Joules:
0 )  2.86644137186590e-20 J
1 )  8.59932411559771e-20 J
```

In [ ]:

```
# Below, we compare the energies of v=0,J=1 and v=1,J=0

energy_total_1 = vib_energies_molecule[0] + rot_energies_molecule[1] # v=0,J=1
energy_total_2 = vib_energies_molecule[1] + rot_energies_molecule[0] # v=1,J=0

energy_difference = energy_total_1 - energy_total_2

print("The energy difference between E_(v=0,J=1)-E_(v=1,J=0)=", energy_difference)
print("\n Where the energy of E_(v=1,J=0)>E_(v=0,J=1)")
```

```
The energy difference between E_(v=0,J=1)-E_(v=1,J=0)= -5.69140575423454e-20

 Where the energy of E_(v=1,J=0)>E_(v=0,J=1)
```

C-3:

We are resquired to solve for K for H_2 and $H^{35}Cl$. However, I have already completed that task above for our molecules. However, for convenience, I have copied and pasted those sections down below.

```
In [ ]:
k_molecule = ((2*math.pi*288600*constants.c)**2)*molecule_reduced
k_h2 = ((2*math.pi*440100*constants.c)**2)*h2_reduced

print("H2's k value is ", k_h2, " kg * s^{-2}")
print("H35Cl's k value is ", k_molecule, " kg * s^{-2}")

answer = (constants.hbar*26.7522e7*5/constants.Boltzmann)
answer2 = constants.e**(-(answer/298))
```

```
H2's k value is  575.0618040437768  kg * s^{-2}
H35Cl's k value is  480.7346399828114  kg * s^{-2}
```