Lab 13 Template

Proving that the SNe data is consistent with the BenchMark Cosmology.

In [25]:

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

from scipy.integrate import simps
import astropy.units as u
from astropy.constants import c
```

In [5]:

```

# Define the benchmark cosmology at z =0
# Planck 2016 results. XIII. Cosmological parameters
# Planck Collaboration+2016 A&A 594 13  Table 4, column 2

OmegaM0_planck = 0.308   # Matter Density Parameter
OmegaR0_planck = 8.24e-5  # Radiation Density Parameter
OmegaL0_planck = 0.692  # Dark Energy Density Parameter
h_planck = 0.6781   # Hubble Constant  100 h km/s/Mpc
```

In [2]:
```
class CosmologicalTools:
    # Define a class that provides functions to compute various cosmologic
    # for a given cosmology

    def __init__(self, OmegaM0, OmegaR0, OmegaL0, h):
        """ initialize the instance of the class - for any given Cosmology

        PARAMETERS
        ----------
        OmegaM0: `float`
            the Matter density parameter at z=0

        OmegaR0: `float`
            the radiation density parameter at z=0

        OmegaL0: `float`
            The dark energy density parameter at z=0

        h: `float`
            the Normalization for the hubble parameter at z=0
        """

        # initialize the cosmology at z=0
        self.OmegaM0 = OmegaM0    ### Matter Density Parameter
        self.OmegaR0 = OmegaR0    ### Radiation Density Parameter
        self.OmegaL0 = OmegaL0    ### Dark Energy Density Parameter
        self.OmegaK0 = 1 - (OmegaM0 + OmegaR0 + OmegaL0)    #### Curvature

        self.h = h   # Normalization of Hubble Parameter
        self.Ho = self.h*100*u.km/u.s/u.Mpc #  Hubble Constant at z=0  10


    def HubbleParameter(self, z):
        """  Method that defines the Hubble Parameter as a function of red
            H(z)^2 = H_o^2 [ OmegaM0(1+z)^3 + OmegaR0(1+z)^4 + OmegaLambda

        PARAMETERS
        ----------
            z: `float`
                redshift

        RETURNS
        -------
            Hz: `float`
                Hubble Parameter as a function of z, in units of km/s/Mpc
        """

        # FILL THIS IN
        OmegaM = self.OmegaM0*(1+z)**3 # OmegaM
        OmegaR = self.OmegaR0*(1+z)**4
        OmegaL = self.OmegaL0
        OmegaK = self.OmegaK0*(1+z)**2

        Hz = self.Ho*np.sqrt(OmegaM+OmegaR+OmegaL+OmegaK)

        return  Hz
```

```
    def OmegaM_Z(self,z):
        """ Method that defines the matter density parameter as a functio
        OmegaM0*(1+z)**3*(Ho/Hz)^2

        PARAMETERS
        ----------
        z  `float or np.ndarray`
            Redshift

        RETURNS
        -------
        OmegaM: `float or np.ndarray`
            Matter Density Parameter at the given redshift.
        """

        OmegaM = self.OmegaM0*(1+z)**3*self.Ho**2/self.HubbleParameter(z)

        return OmegaM

    def OmegaR_Z(self,z):
        """ Method that defines the radiation density parameter as a funct
         OmegaR0*(1+z)**4*(Ho/Hz)^2

        PARAMETERS
        ----------
        z  `float or np.ndarray`
            Redshift

        RETURNS
        -------
        OmegaR: `float or np.ndarray`
            Radiation Density Parameter at the given redshift.
        """

        OmegaR =   self.OmegaR0*(1+z)**4*self.Ho**2/self.HubbleParameter(


    def OmegaL_Z(self,z):
        """ Method that defines the dark energy density parameter as a fu
        OmegaL0*(Ho/Hz)^2

        PARAMETERS
        ----------
        z  `float or np.ndarray`
            Redshift

        RETURNS
        -------
        OmegaL: `float or np.ndarray`
            Dark Energy Density Parameter at the given redshift.
        """

        OmegaL = self.OmegaL0*self.Ho**2/self.HubbleParameter(z)**2
```

```
        return OmegaL



    ###########################
    # Lab 11 Starts Here
    ###########################




    def LookBackTime(self, ze):
        """ Method that computes the Look Back Time at a given redshift
         i.e. the difference in time from when a photon was emitted to whe

         Integrand: 1/H(z)/(1+z)

        PARAMETERS
        ----------
        ze : `float`
            Redshift emitted (ze). This cannot be an array

        RETURNS
        -------
        time: `float`
            Time in units of Gyr ago (relative to present day)
        """

        # Observed redshift - today
        zo = 0

        # define an array with redshifts, spaced in intervals of 0.001 fro
        zrange = np.arange(zo, ze, 1e-3)


        # Integrand
        # y = (1/H(zrange)).to(GYR)  /  (1+zrange) --> this conversion als
        # But need to correct units of 1/H to be Gyr rather than seconds
        # use the astropy.units functionality .to(units)
        # FILL THIS IN
        y = (1.0/self.HubbleParameter(zrange)).to(u.Gyr)/(1+zrange)


        # Integrate y numerically over zrange and return in units of Gyr
        # FILL THIS IN
        # for whatever reason simps gets rid of the units so you have to a
        time = simps(y,zrange)*u.Gyr

        return time



    def ComovingDistance(self, zo, ze):
        """ Method that computes the Comoving Radial Distance to an object
        i.e, Distance to a galaxy that is moving with the Hubble Flow (exp
            Dc = c*Int_z0^ze 1/H(z)
```

```
        PARAMETERS
        ----------
        zo: `float`
            Redshift of the observer

        ze: `float`
            Redshift of the object emitting the photon

        RETURNS
        -------
        DC: `float`
            Comoving Radial Distance (Mpc)
        """

        # define an array with redshifts, spaced  in intervals of 0.001
        # Note that if you want redshifts smaller than 0.001 you'll need t
        zrange = np.arange(zo, ze, 1e-3)


        # Integrand
        # 1/H(zrange)*speed of light
        # Speed of light is loaded in modules from astropy, but in units o
        # FILL THIS IN
        y = c.to(u.km/u.s)*(1.0/self.HubbleParameter(zrange))


        # Integrate y numerically over zrange and return in units of Mpc
        # FILL THIS IN
        DC = simps(y,zrange)*u.Mpc

        return DC


    def ProperDistance(self, zo, ze):
        """ Method that returns the Proper Distance to an object at some r
            to an observer a given redshift (the distance measured by a ru
             R(tobs)*DC = DC/(1+zobs)

        PARAMETERS
        ---------

        zo: `float`
            Redshift of the observer

        ze: `float`
            Redshift of the object emitting the photon

        RETURNS
        -------
        DH: `float`
            Proper Distance (Mpc)
        """

        # Comoving Distance[ independent of time] x the scale factor at th
        DH = self.ComovingDistance(zo,ze)/(1+zo) # if zo=0 then this is ju

        return DH
```

```
    def LuminosityDistance(self, ze):
        """ Method that computes the Luminosity Distance to an object at s
               DL = DC*(1+z_emitted)

        PARAMETERS
        ---------
        ze: `float`
            Redshift of the object emitting the photons

        RETURNS
        -------
        DL: `float`
            Luminosity Distance (Mpc)
        """

        zo = 0 # This is a quantity computed by an observer at z=0

        DL = self.ComovingDistance(zo,ze)*(1+ze)

        return DL


    def AngularDiameterDistance(self, ze):
        """ Method that computes the Angular Diameter Distance to an objec
               DA = DC/(1+z_emitted)
            DA is the distance to the source,  such that it subtends the s
            it would have in Euclidean Space


        PARAMETERS
        ---------
        ze: `float`
            Redshift of the object emitting the photons

        RETURNS
        -------
        DA: `float`
            Angular Diameter Distance (Mpc)
        """

        # this is an observable so
        zo = 0

        # # FILL THIS IN
        DA = self.ComovingDistance(zo,ze)/(1+ze)

        return DA


    def Size(self, ze, angle):
        """ Method to compute the physical distance corresponding to a giv
            angular separation at a given redshift
            S = DA*angle

        PARAMETERS
```

```
        ---------
        ze: `float`
            Redshift of the object emitting the photons

        angle: `float`
            Angular separation or size (arcsec)

        RETURNS
        -------
        size: `float`
            Physical Size of the Object or the Separation between objects
        """

        # convert angle from arcsec to radians
        angleRad = (angle*u.arcsec).to(u.rad)

        #   DA*angleRad
        size = self.AngularDiameterDistance(ze).to(u.kpc)*angleRad.value
        # recall angular diameter distance is in Mpc

        return size


    ################
    ## Lab 12  Starts here
    ################

    #  Part 1: Question 1
    def Temperature(self, z):
        """ Method that computes the temperature of the universe as a func
            T = To(1+z)

        PARAMETERS
        ----------
        z: `float`
            Redshift of interest

        RETURNS
        -------
        T: `float`
            Temperature at that redshift in K

        """

        # Temperature of the universe today

        # Fill this in

        return


    # Part 3: Question 1
    ## Fill this in
    def SoundHorizon(self, ze, zlarge):
        """ Method that computes the maximal distance that sound can trave
        until the given epoch.
            ProperDistance/sqrt(3)    (cs = c/sqrt(3))
```

```
        PARAMETERS
        ----------
        ze:  `float`
            Redshift of interest (usually redshift of recombination = 110(

        zlarge: `float`
            A large redshift to denote early age of the universe

        RETURNS
        -------
        SH: `float`
            Sound Horizon in Mpc

        """

        # Fill this in

        SH = self.ProperDistance(ze, zlarge) / np.sqrt(3)

        return SH
```

In [6]:
```
# Define the Einstein-DeSitter cosmology (Matter Dominated)
OmegaMD = 1
OmegaRD = 0
OmegaLD = 0
# h is the same = h_planck
```

In [7]:
```
BenchMark = CosmologicalTools(OmegaM0_planck,OmegaR0_planck,OmegaL0_plancl
EinsteinDeSitter = CosmologicalTools(OmegaMD,OmegaRD,OmegaLD,h_planck)
```

In this exercise we will use data from the Supernova Cosmology project, one of the two teams which first found that the expansion rate of the Universe is accelerating in 1999. A simple introduction to the methods and findings of the SCP group can be found at [https://newscenter.lbl.gov/2009/10/27/evolving-dark-energy/ (https://newscenter.lbl.gov/2009/10/27/evolving-dark-energy/)](https://newscenter.lbl.gov/2009/10/27/evolving-dark-energy/). The original paper is Perlmutter *et al.* 1999, "Measurement of Ω and Λ from 42 High Redshift Supernovae", The Astrophysical Journal, Vol. 517, page 565.

The data set we will be using is a more recent sample, containing observations of 580 supernovae, known as the Union 2.1 sample from the paper Suzuki *et al*. 2012, "THE *HUBBLE SPACE TELESCOPE* CLUSTER SUPERNOVA SURVEY. V. IMPROVING THE DARK-ENERGY CONSTRAINTS ABOVE $z > 1$ AND BUILDING AN EARLY-TYPE-HOSTED SUPERNOVA SAMPLE", The Astrophysical Journal, vol. 746, page 85.

The data are in the file SNeData.txt.

Take a look at the file using the `with` statement.

One should always close files when finished using them. The `with` statement makes this automatic; using it is a good habit to form.

Lets simply open the file and print out the first 10 lines to see how the file is formatted:

In [8]:
```
with open('SNeData.txt', 'r') as infile:
    for i in range(10):
        line = infile.readline()
        line = line.rstrip("\n")
        print(line)
```

```
# Supernova Cosmology Project Union2.1 Data
#  Suzuki et al.
#Name z DistMod DistModErr ProbLowMassHost
1993ah 0.028488        35.3465833928 0.223905932998  0.128418942246
1993ag 0.050043        36.6823679154 0.166828851413  0.128418942246
1993o  0.052926        36.8176912545 0.1557559148    0.128418942246
1993b  0.070086        37.4467365424 0.158466934433  0.128418942246
1992bs 0.062668        37.4834093505 0.156099434739  0.128418942246
1992br 0.087589        38.2290570494 0.187745679272  0.128418942246
1992bp 0.078577        37.4881622607 0.155635656185  0.128418942246
```

The top of any good data file intended for sharing with others contains a "header" -- some lines at the top which describe the contents of the file.

Here we see that the file contains the SCP Union2.1 data, and that the columns are:

- the name of the supernova
- the redshift measured from its spectrum
- its distance modulus
- an estimate of the measurement error in the distance modulus
- the probability the supernova occurred in a low-mass host galaxy

For this exercise, we won't care what a supernova's name is, and we won't get to the last column until the end of the exercise.

Part A

The difference between the absolute magnitude M and the apparent magnitude m, a number called the *distance modulus* which depends only upon the distance to the source

$$\begin{aligned} m - M &= -2.5 \log_{10}\left(\frac{1}{F_0}\frac{L}{4\pi d^2}\right) + 2.5\log_{10}\left(\frac{1}{F_0}\frac{L}{4\pi(10\ \mathrm{pc})^2}\right) \\ &= 5\log_{10}\left(\frac{d}{10\ \mathrm{pc}}\right) \end{aligned}$$

Because M and m are logarithmic functions, their difference is proportional to the *ratio* of the distance d to 10 pc.

This is the distance measurement given in the data file for the distance to the supernovae. The measured LUMINOSITY distance is then

$$d_L = 10^{(m-M)/5+1}\mathrm{pc}$$

In [9]:

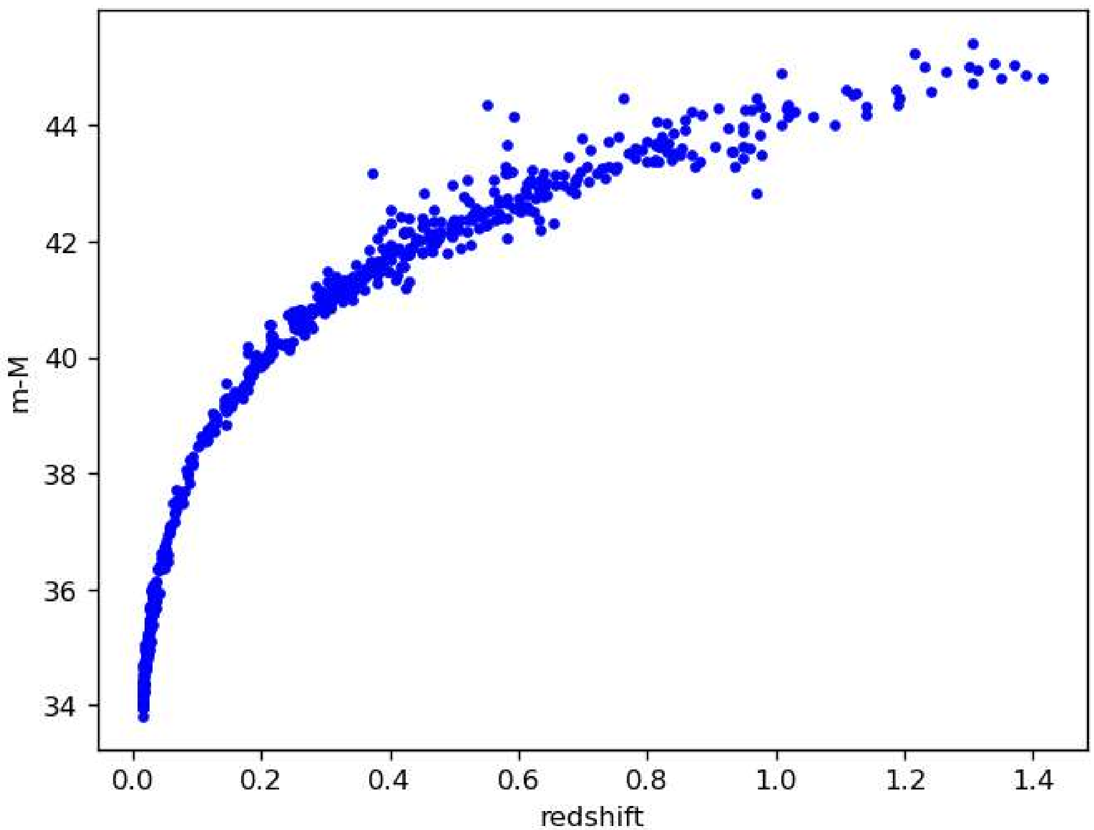
```
def Distance_fromMod(mod):
    """ Function to compute the distance using the distance modulus
    PARAMETERS
    ----------
    mod: `float`
        distance modulus (apparent magnitude - absolute magnitude)

    OUTPUTS
    -------
    DL: `float`
        luminiosity distance in Mpc
    """

    a = mod/5 + 1
    dist = 10**a *u.pc.to(u.Mpc)
    return dist
```

In [10]:

```
# Read in the file "SNeData.txt" using `npgenfromtxt`

data = np.genfromtxt('SNeData.txt', names=True, skip_header=2)
```

In [11]:
```
# Create a plot of Distance Modulus Vs. Redshift

plt.rcParams["figure.dpi"] = 120


plt.plot(data['z'] , data['DistMod'], 'b.')
plt.xlabel('redshift')
plt.ylabel('m-M')
```

Out[11]: Text(0, 0.5, 'm-M')



Part B

Now let's form an actual distance in mega-parsecs (Mpc) from the distance modulus and a velocity in km/second from the redshifts

In [12]:
```
# 1) Distance
LD = Distance_fromMod(data['DistMod'])
print(LD[0])
```

```
117.30504178865594
```

In [13]:
```
# 2) velocity
VR = data['z']*c.to(u.km/u.s)
```

Part C

plot distance versus velocity just for the "nearby" supernovae, those within 200 Mpc of Earth. We can select the set of indices of the nearby supernovae using the `numpy.where` function

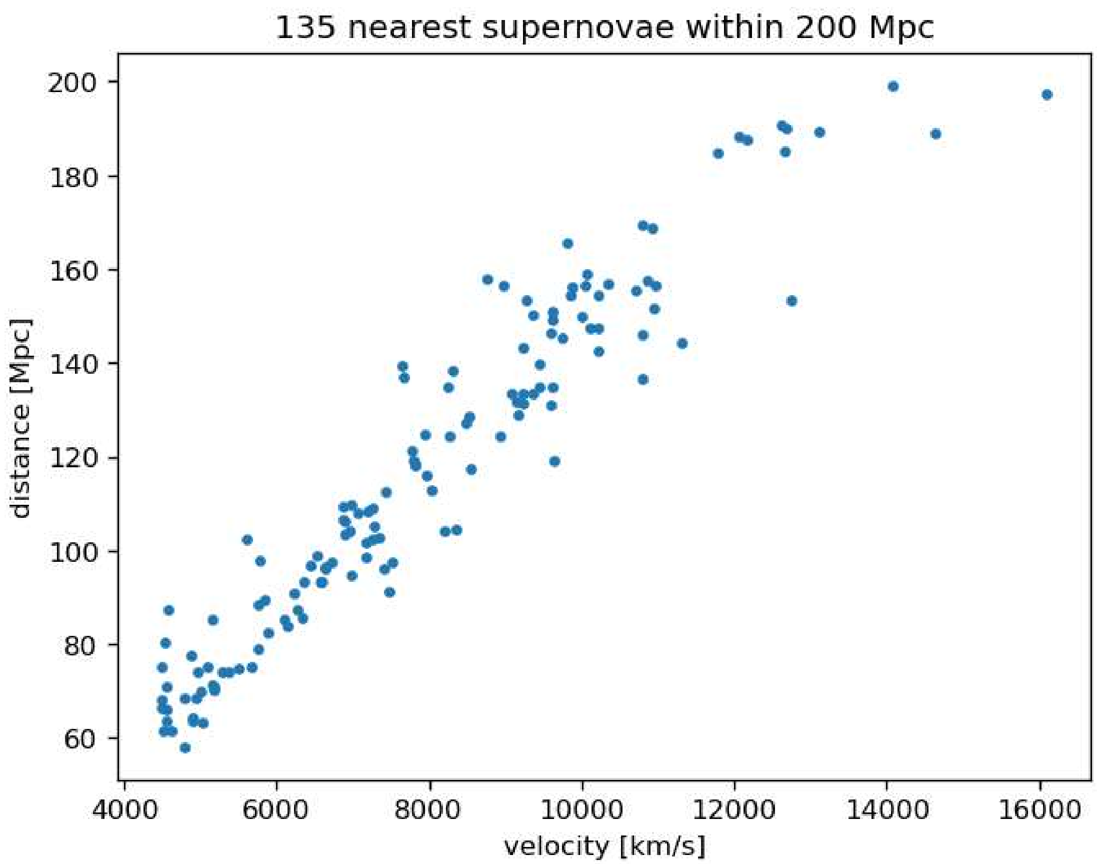
In [14]:
```
# Create an index for the nearby supernovae
near = np.where(LD < 200)
```

In [15]:
```
# get the number of nearby supernovae
nnear = len(near[0])
print(nnear)
```

```
135
```

In [18]:
```
# Plot the Luminosity Distance vs. Recessional Speed for all nearby Superr

plt.rcParams["figure.dpi"] = 120

# Fill this in
plt.plot(VR[near] , LD[near], '.')

plt.xlabel('velocity [km/s]')
plt.ylabel('distance [Mpc]')

# Fill this in :   Add a relevant title
plt.title(f"{nnear} nearest supernovae within 200 Mpc")
```

Out[18]:
```
Text(0.5, 1.0, '135 nearest supernovae within 200 Mpc')
```



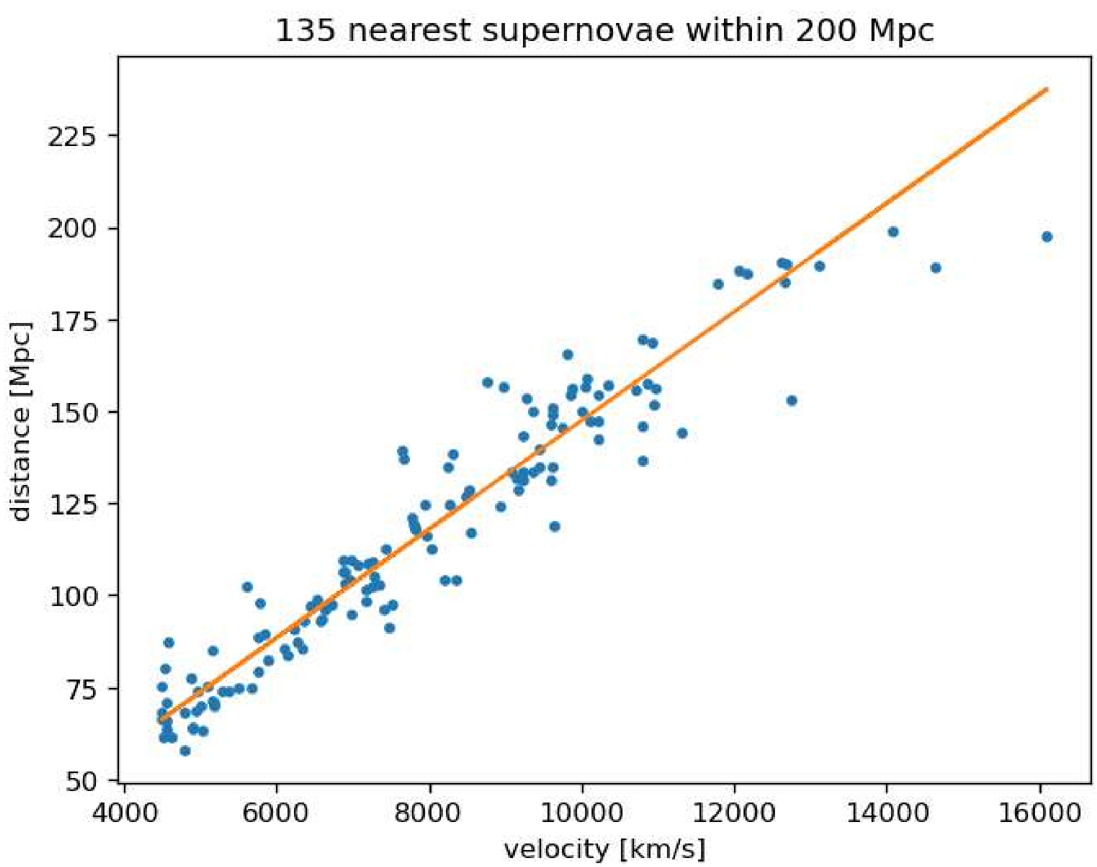
Part D

Plot a linear relationship atop the data

In [19]:
```python
# Create a linear model
# V = H*R --> R = V/ H

# 1/Ho ~ Age of the universe
# this line is equivalently = t_age * VR[near] --> constant expansion over

modelLD = VR/BenchMark.Ho
```

In [20]:
```python
# Recreate the plot, now including the linear model
# FILL THIS IN

plt.rcParams["figure.dpi"] = 120

# Fill this in
plt.plot(VR[near] , LD[near], '.')
plt.plot(VR[near] , modelLD[near], '-')

plt.xlabel('velocity [km/s]')
plt.ylabel('distance [Mpc]')

# Fill this in :   Add a relevant title
plt.title(f"{nnear} nearest supernovae within 200 Mpc")
```

Out[20]: Text(0.5, 1.0, '135 nearest supernovae within 200 Mpc')

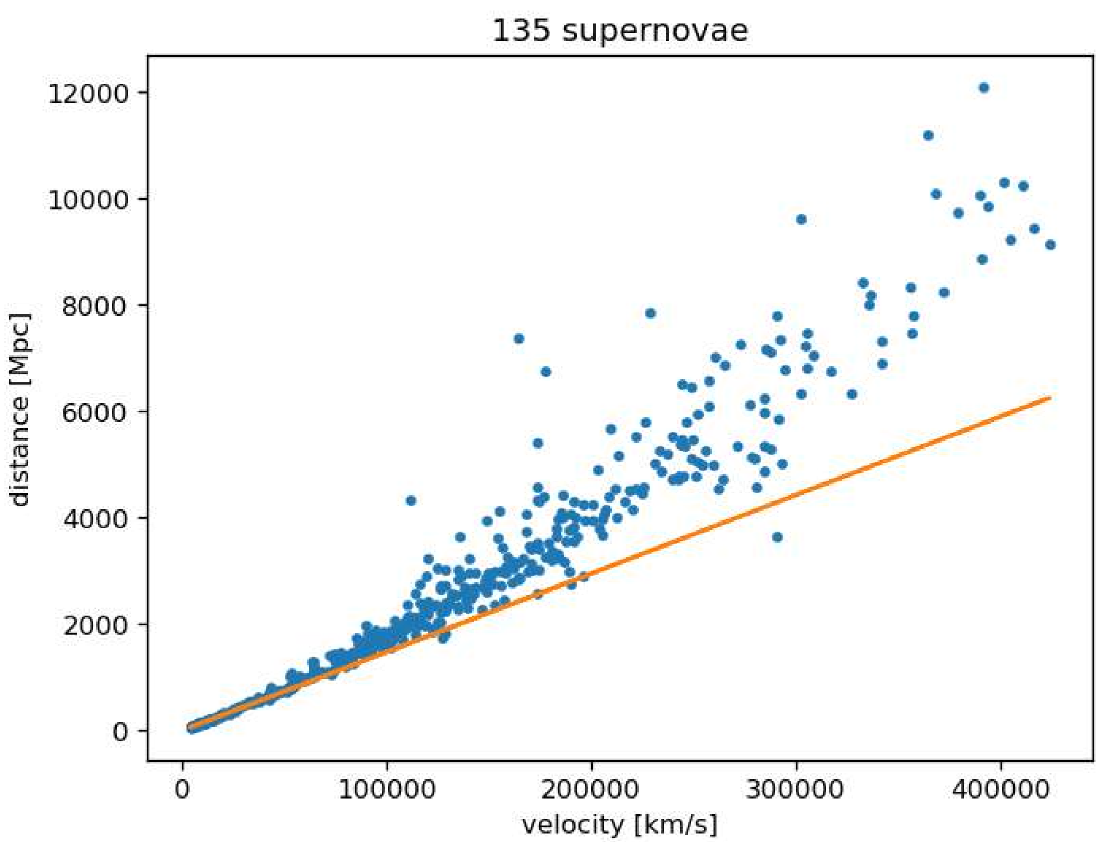


Part E

Let's now try plotting the whole dataset, which extends to distances far beyond what Hubble

In [21]:
```
# Plot the whole data set. Not just the nearby Sne.
# FILL THIS IN

# Recreate the plot, now including the linear model
# FILL THIS IN

plt.rcParams["figure.dpi"] = 120

# Fill this in
plt.plot(VR , LD, '.')
plt.plot(VR , modelLD, '-')

plt.xlabel('velocity [km/s]')
plt.ylabel('distance [Mpc]')

# Fill this in :   Add a relevant title
plt.title(f"{nnear} supernovae")
```

Out[21]: `Text(0.5, 1.0, '135 supernovae')`



Part F

Instead of assuming a linear model, let's use our code to compute a model for the luminosity distance to objects moving with the Hubble flow.

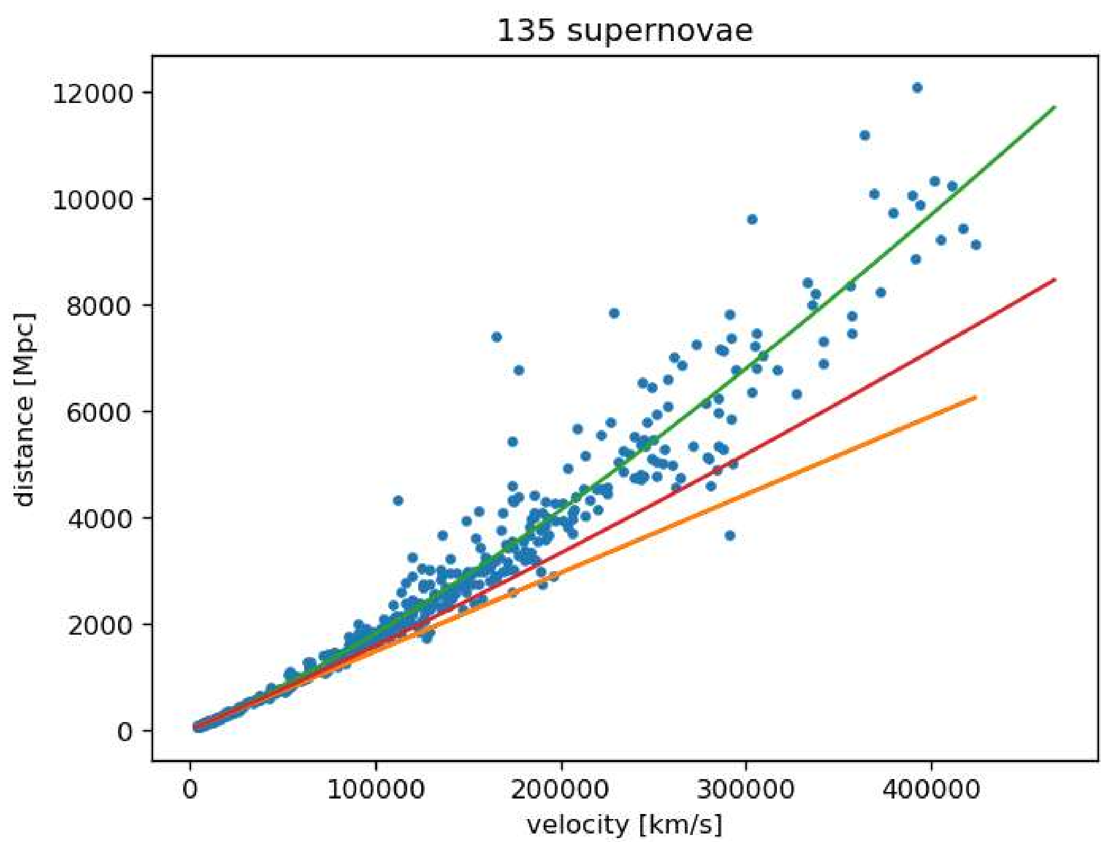
In [22]:
```
# define a vector of redshifts to compute the luminosity distance
zvec = np.linspace(0.01, 1.1*max(data['z']), 100)
```

In [23]:
```
## Compute the corresponding recessional velocities
vr_vec = zvec * c.to(u.km/u.s)
```

In [27]:
```
## Compute the Luminosity Distance at each redshift  in the BenchMark and
import scipy
modelBenchmark = [BenchMark.LuminosityDistance(i).value for i in zvec]
modelEinsteinDeSitter = [EinsteinDeSitter.LuminosityDistance(i).value for
```

In [28]:
```python
## Plot the New models on top of the data.
## FILL THIS IN

# Plot the whole data set. Not just the nearby Sne.
# FILL THIS IN

# Recreate the plot, now including the linear model
# FILL THIS IN

plt.rcParams["figure.dpi"] = 120

# Fill this in
plt.plot(VR , LD, '.')
plt.plot(VR , modelLD, '-')
plt.plot(vr_vec , modelBenchmark, '')
plt.plot(vr_vec , modelEinsteinDeSitter, '')

plt.xlabel('velocity [km/s]')
plt.ylabel('distance [Mpc]')

# Fill this in :   Add a relevant title
plt.title(f"{nnear} supernovae")
```

Out[28]: `Text(0.5, 1.0, '135 supernovae')`



Part G

We can characterize how well the model fits the data by computing the "χ^2" of the model with respect to the data

$$\chi = \sqrt{\frac{\sum_i (\text{model}(z_i) - r_i)}{N - 1}}$$

Let's write a function to do this:

In [ ]:
```
def chi(model, data):
    """ Function to compute the deviation between a model and data
            Chi = sqrt (Sum (deviation between model and data) / N-1)

    PARAMETERS
    ---------
    model: `np.darray`
        Array containing the model to be compared to the data

    data: `np.array`
        Array containing the data to be compared to the model

    OUTPUTS
    -------
    Chi: `float`
        Describes how well the model fits the data

    """

    return
```

The χ of our linear model is then

In [ ]:

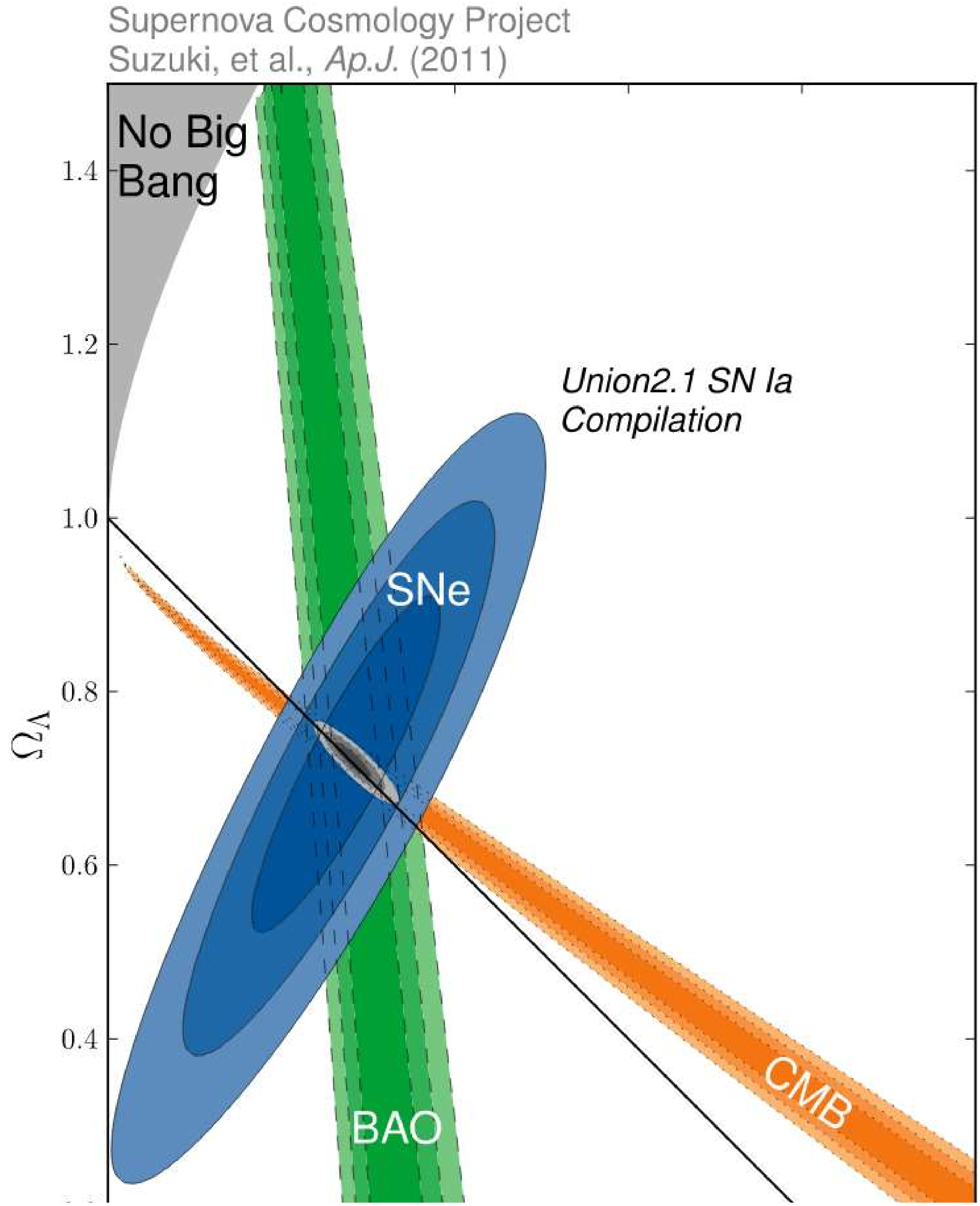
The χ of our Einstein-DeSitter Luminosity Distance model is then

In [ ]:

The χ of our BenchMark model is then

In [ ]:

To test this more rigorously we could run through different values of Omega_M and Omega_L and generate probability contours (rather than χ)


Supernova Cosmology Project
Suzuki, et al., Ap.J. (2011)
No Big Bang
Union2.1 SN Ia Compilation
SNe
BAO
CMB
1.4
1.2
1.0
0.8
0.6
0.4
Ω_Λ

In [ ]: