lab13

May 31, 2022

Name: Allan Gongora

Section: 0131

1 Lab 13: Regression Inference

Welcome to Lab 13!

Sometimes, the primary purpose of regression analysis is to learn something about the slope or intercept of the best-fitting line. When we use a sample of data to estimate the slope or intercept, our estimate is subject to random error, just as in the simpler case of the mean of a random sample.

In this lab, we'll use regression to get an accurate estimate for the age of the universe, using pictures of exploding stars. Our estimate will come from a sample of all exploding stars. We'll compute a confidence interval to quantify the error caused by sampling.

Please complete this notebook by filling in the cells provided. Before you begin, execute the following cell to load the provided tests.

[1]:
```
pip install gofer-grader
```

```
Requirement already satisfied: gofer-grader in /opt/conda/lib/python3.7/site-
packages (1.1.0)
Requirement already satisfied: jinja2 in /opt/conda/lib/python3.7/site-packages
(from gofer-grader) (3.0.3)
Requirement already satisfied: tornado in /opt/conda/lib/python3.7/site-packages
(from gofer-grader) (6.1)
Requirement already satisfied: pygments in /opt/conda/lib/python3.7/site-
packages (from gofer-grader) (2.11.2)
Requirement already satisfied: MarkupSafe>=2.0 in /opt/conda/lib/python3.7/site-
packages (from jinja2->gofer-grader) (2.0.1)
Note: you may need to restart the kernel to use updated packages.
```

[2]:
```
# Run this cell, but please don't change it.

# These lines import the Numpy and Datascience modules.
import numpy as np
from datascience import *

# These lines do some fancy plotting magic
```

```
import matplotlib
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
import warnings
warnings.simplefilter('ignore', FutureWarning)
warnings.simplefilter('ignore', UserWarning)
from matplotlib import patches
from ipywidgets import interact, interactive, fixed
import ipywidgets as widgets

# These lines load the tests.
from gofer.ok import check
```

Recommended Reading: * Prediction

1) For all problems that you must write explanations and sentences for, you **must** provide your answer in the designated space. This can include:
   A) Sentence reponses to questions that ask for an explanation
   B) Numeric responses to multiple choice questions
   C) Programming code
2) Moreover, throughout this lab and all future ones, please be sure to not re-assign variables throughout the notebook! For example, if you use `max_temperature` in your answer to one question, do not reassign it later on. Otherwise, you will fail tests that you thought you were passing previously!

Once you're finished, select "Save and Checkpoint" in the File menu. Your name and course section number should be in the first and last cell of the assignment. Be sure you have run all cells with code and that the output from that is showing. Then click "Print Preview" in the File menu. Print a copy from there in pdf format. (This means you right click and choose print and choose "save as pdf" from your printer options.) You will need to submit the pdf in Canvas by the deadline.

The gopher grader output and/or output from your coding are essential to helping your instructor grade your work correctly and in a timely manner.

Files submitted that are missing the required output will lose some to all points so double check your pdf before submitting.

1.1 1. The Age of the Universe

1.1.1 The Actual Big Bang Theory

In the early 20th century, the most popular cosmological theory suggested that the universe had always existed at a fixed size. Today, the Big Bang theory prevails: Our universe started out very small and is still expanding.

A consequence of this is Hubble's Law, which states that every celestial object that's reasonably far away from Earth (for example, another galaxy) is moving away from us at a constant speed. If we extrapolate that motion backwards to the time when everything in the universe was in the same place, that time is (roughly) the beginning of the universe!

Scientists have used this fact, along with measurements of the current *location* and *movement speed* of other celestial objects, to estimate when the universe started.

The cell below simulates a universe in which our Sun is the center and every other star is moving away from us. Each star starts at the same place as the Sun, then moves away from it over time. Different stars have different directions *and speeds*; the arrows indicate the direction and speed of travel.

Run the cell, then move the slider to see how things change over time.

[3]:
```
# Just run this cell.  (The simulation is actually not
# that complicated; it just takes a lot of code to draw
# everything.  So you don't need to read this unless you
# have time and are curious about more advanced plotting.)

num_locations = 15
example_velocities = Table().with_columns(
    "x", np.random.normal(size=num_locations),
    "y", np.random.normal(size=num_locations))
start_of_time = -2

def scatter_after_time(t, start_of_time, end_of_time, velocities, center_name,␣
 ↪other_point_name, make_title):
    max_location = 1.1*(end_of_time-start_of_time)*max(max(abs(velocities.
 ↪column("x"))), max(abs(velocities.column("y"))))
    new_locations = velocities.with_columns(
            "x", (t-start_of_time)*velocities.column("x"),
            "y", (t-start_of_time)*velocities.column("y"))
    plt.scatter(make_array(0), make_array(0), label=center_name, s=100,␣
 ↪c="yellow")
    plt.scatter(new_locations.column("x"), new_locations.column("y"),␣
 ↪label=other_point_name)
    for i in np.arange(new_locations.num_rows):
        plt.arrow(
            new_locations.column("x").item(i),
            new_locations.column("y").item(i),
            velocities.column("x").item(i),
            velocities.column("y").item(i),
            fc='black',
            ec='black',
            head_width=0.025*max_location,
            lw=.15)
    plt.xlim(-max_location, max_location)
    plt.ylim(-max_location, max_location)
    plt.gca().set_aspect('equal', adjustable='box')
    plt.gca().set_position(make_array(0, 0, 1, 1))
    plt.legend(bbox_to_anchor=(1.6, .7))
    plt.title(make_title(t))
```

```
    plt.show()

interact(
    scatter_after_time,
    t=widgets.FloatSlider(min=start_of_time, max=5, step=.05, value=0,␣
 ↪msg_throttle=1),
    start_of_time=fixed(start_of_time),
    end_of_time=fixed(5),
    velocities=fixed(example_velocities),
    center_name=fixed("our sun"),
    other_point_name=fixed("other star"),
    make_title=fixed(lambda t: "The world {:01g} year{} in the {}".
 ↪format(abs(t), "" if abs(t) == 1 else "s", "past" if t < 0 else "future")));
```

```
interactive(children=(FloatSlider(value=0.0, description='t', max=5.0, min=-2.0,␣
 ↪step=0.05), Output()), _dom_c…
```

1.1.2 Analogy: Driving

Here's an analogy to illustrate how scientists use information about stars to estimate the age of the universe.

Suppose that at some point in the past, our friend Mei started driving in a car going at a steady speed of 60 miles per hour straight east. We're still standing where she started.

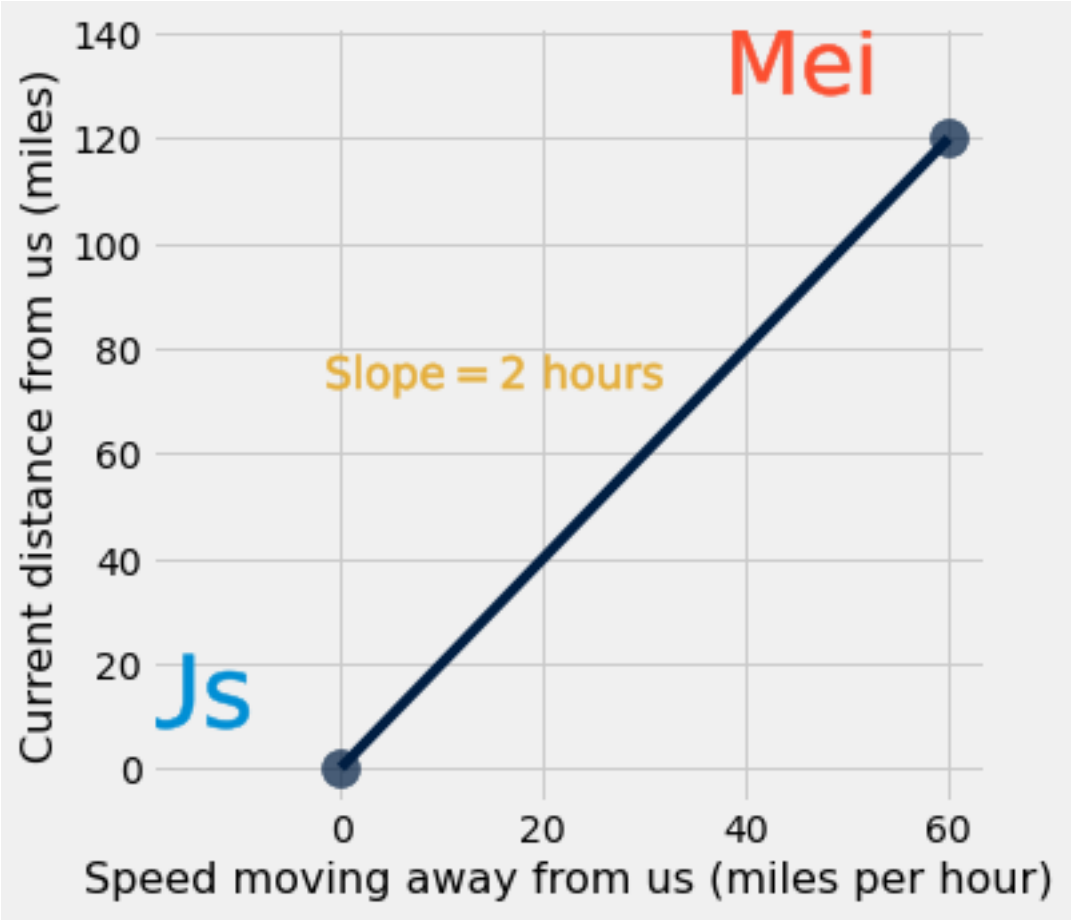
[4]:
```
# Run this cell to see a picture of Mei's locations over time.

mei_velocity = Table().with_columns("x", make_array(60), "y", make_array(0))
interact(
    scatter_after_time,
    t=widgets.FloatSlider(min=-2, max=1, step=.05, value=0, msg_throttle=1),
    start_of_time=fixed(-2),
    end_of_time=fixed(1),
    velocities=fixed(mei_velocity),
    center_name=fixed("Us"),
    other_point_name=fixed("Mei"),
    make_title=fixed(lambda t: "Mei's position {:01g} hour{} in the {}".
 ↪format(abs(t), "" if abs(t) == 1 else "s", "past" if t < 0 else "future")));
```

```
interactive(children=(FloatSlider(value=0.0, description='t', max=1.0, min=-2.0,␣
 ↪step=0.05), Output()), _dom_c…
```

We want to know how long she's been driving, but we forgot to record the time when she left. If we find out that she's 120 miles away, and she's been going 60 miles per hour the whole time, we can infer that she left 2 hours ago.

One way we can compute that number is by fitting a line to a scatter plot of our locations and speeds. It turns out that the *slope* of that line is the amount of time that has passed. Run the next cell to see a picture:

[5]:
```
# Just run this cell.
small_driving_example = Table().with_columns(
        "Name",                                    make_array("Us", "Mei"),
        "Speed moving away from us (miles per hour)", make_array(0,    60),
        "Current distance from us (miles)",        make_array(0,    120))

small_driving_example.scatter(1, 2, s=200, fit_line=True)

# Fancy magic to draw each person's name with their dot.
with_slope_indicator = small_driving_example.with_row(
    ["Slope = 2\ hours", small_driving_example.column(1).mean(),␣
 ↪small_driving_example.column(2).mean()])
for i in range(with_slope_indicator.num_rows):
    name = with_slope_indicator.column(0).item(i)
    x = with_slope_indicator.column(1).item(i)
    y = with_slope_indicator.column(2).item(i)
    plt.scatter(make_array(x - 15), make_array(y + 15), s=1000*len(name),␣
 ↪marker="$\mathrm{" + name + "}$")
```

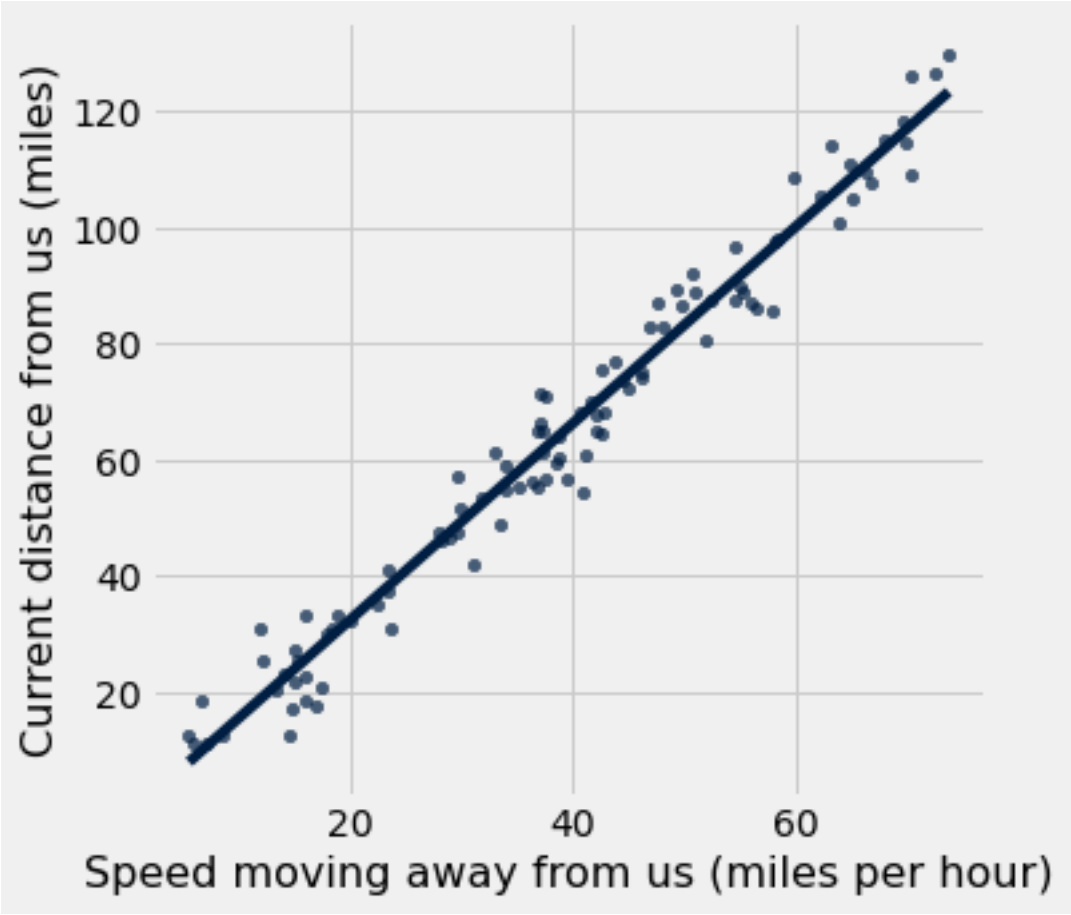


The slope of the line is 2 hours. (The units are vertical-axis units divided by horizontal-axis units, which are $\frac{\text{miles}}{\text{miles/hour}}$, or hours.) So that's our answer.

Imagine that you don't know Mei's exact distance or speed, only rough estimates. Then, if you drew this line, you'd get a slightly bad estimate of the time since she left. But if you measured the distance and speed of hundreds of people who left you at the same time going different speeds, and drew a line through them, the slope of that line would be a pretty good estimate of the time they left, even if the individual measurements weren't exactly right.

The `drivers.csv` dataset contains the speeds and distances-from-start of 100 drivers. They all left the same starting location at the same time, driving at a fixed speed on a straight line away from the start. The measurements aren't exact, so they don't fit exactly on a line. We've created a scatter plot and drawn a line through the data.

[6]:
```
# Just run this cell.
Table.read_table("drivers.csv").scatter(0, 1, fit_line=True)
```



Question 1.1 By looking at the fit line, estimate how long ago (in hours) Mei left.

[7]:
```
# Fill in the start time you infer from the above line.
driving_start_time_hours = (100 - 68) / (60 - 40)
driving_start_time_hours
```

[7]:
```
1.6
```

[8]:
```
check('tests/q1_1.py')
```

[8]:
```
<gofer.ok.OKTestsResult at 0x7f63e35395d0>
```

1.1.3 Back to Cosmology

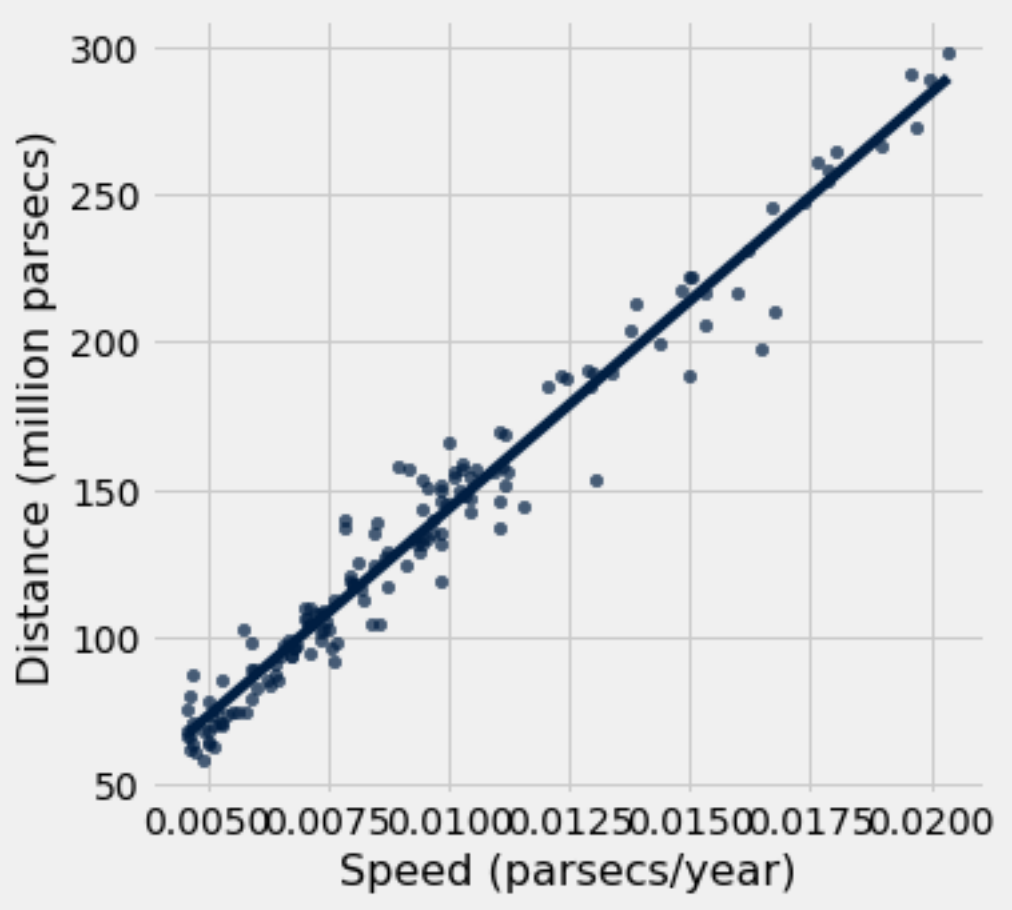
To do the same thing for the universe, we need to know the distance-from-Earth and speed-away-from-Earth of many celestial objects. Using pictures taken by very accurate telescopes and a lot of physics, astronomers have been able to estimate both. It turns out that *nearby supernovae* – stars that have recently died and exploded – are among the best sources of this data, because they are very easy to see. This picture taken by the Hubble telescope shows an entire galaxy, with a single supernova - as bright by itself as billions of stars - at the bottom left.

Our astronomical data for today will come from the Supernova Cosmology Project at Lawrence Berkeley Lab. The original dataset is here, with (brief) documentation here. Each row in the table corresponds to a supernova near Earth that was observed by astronomers. From pictures like the one above, the astronomers deduced how far away each supernova was from Earth and how fast it was moving away from Earth. Their deductions were good, but not perfect.

Run the cell below to load the data into a table called `close_novas` and make a scatter plot. (If you prefer, you can also use the name `close_novae`; both are correct.)

[9]:
```
# Just run this cell.
close_novas = Table.read_table("close_novas.csv")
close_novae = close_novas

close_novas.scatter(0, 1, fit_line=True)
close_novas
```

[9]:
```
Speed (parsecs/year) | Distance (million parsecs)
0.00873361           | 117.305
0.0153418            | 217.007
0.0162256            | 230.961
0.00528131           | 85.2853
0.0129474            | 185.051
0.0138862            | 212.841
0.0111837            | 151.728
0.0060085            | 82.6121
0.00838228           | 104.029
0.00812078           | 124.778
… (146 rows omitted)
```



Question 1.2 Looking at this plot, make a guess at the age of the universe.

Note: Make sure you get the units right! In case you need to know what a parsec is, it's a big unit of distance, equivalent to 30.86 trillion kilometers.

[10]:
```
# Fill this in manually by examining the line above.
first_guess_universe_age_years = 10*1e9

# This just shows your guess as a nice string, in billions of years.
"{:,} billion years".format(round(first_guess_universe_age_years / 1e9, 2))
```

[10]:
```
'10.0 billion years'
```

[11]:
```
check('tests/q1_2.py')
```

[11]:
```
<gofer.ok.OKTestsResult at 0x7f63e2d39990>
```

1.1.4 Fitting the Line Yourself

`fit_line=True` is convenient, but we need to be able to calculate the slope as a number. Recall that the least-squares regression line for our supernova data is: * the line * with the smallest average (over all the supernovae we observe) * error * squared * where the error is

the supernova's actual distance from Earth − the height of the line at that supernova's speed.

Question 1.3 Define a function called `errors`. It should take three arguments: 1. A table like `close_novas` (with the same column names and meanings, but not necessarily the same data) 2. The slope of a line (a number) 3. The intercept of a line (a number)

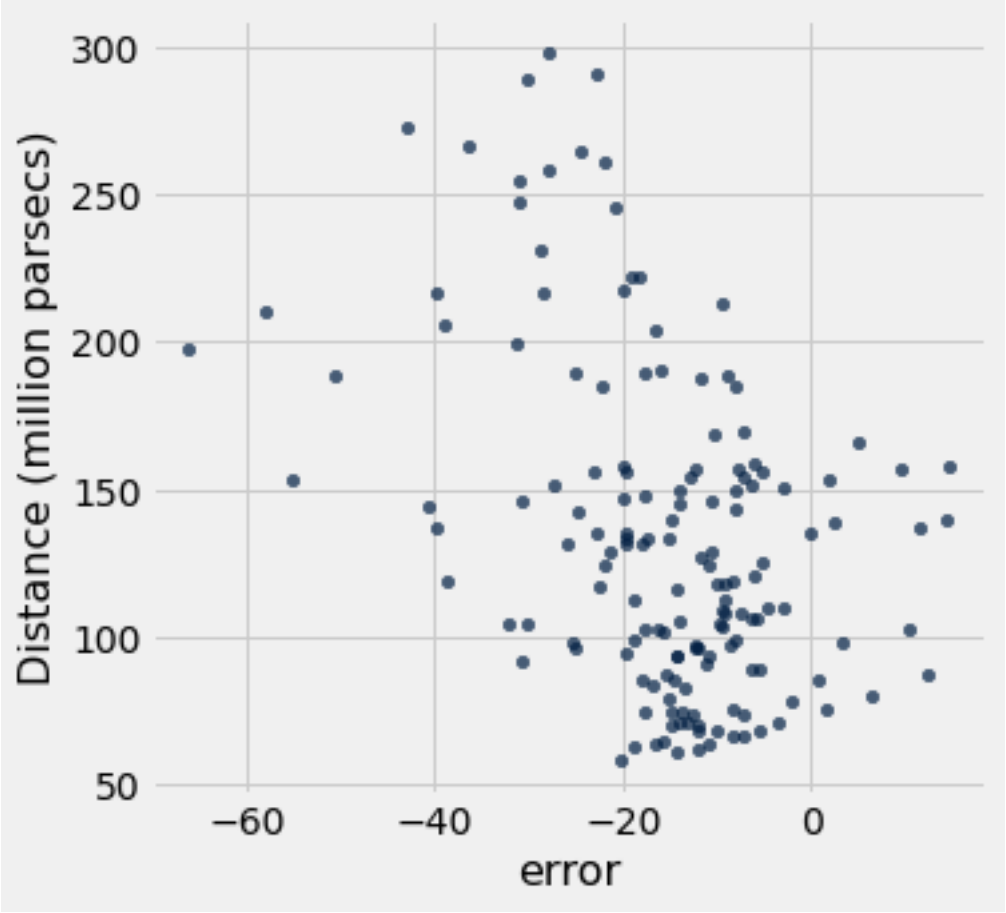
It should return an array of the errors made when a line with that slope and intercept is used to predict distance from speed for each supernova in the given table. (The error is the actual distance minus the predicted distance.)

[12]:
```
def errors(t, slope, intercept):
    return t[1] - (t[0]*slope + intercept)
```

Question 1.4 Using `errors`, compute the errors for the line with slope `16000` and intercept `0` on the `close_novas` dataset. Name that array `example_errors`. Then, make a scatter plot of the errors.

Hint: To make a scatter plot of the errors, plot the error for each supernova in the dataset. Put the actual speed on the horizontal axis and the error on the vertical axis.

[13]:
```
example_errors = errors(close_novas, 16000, 0)
close_novas.with_column("error", example_errors).scatter("error", 1)
```



[14]:
```
check('tests/q1_4.py')
```

[14]:
```
<gofer.ok.OKTestsResult at 0x7f63e09eafd0>
```

You should find that the errors are almost all negative. That means our line is a little bit too steep. Let's find a better one.

Question 1.5 Define a function called `fit_line`. It should take a table like `close_novas` (with the same column names and meanings) as its argument. It should return an array containing the slope (as item 0) and intercept (as item 1) of the least-squares regression line predicting distance from speed for that table.

Note: If you haven't tried to use the `minimize` function yet, now is a great time to practice. Here's an example from the textbook.

[15]:
```
def fit_line(tbl):
    # Your code may need more than 1 line below here.
    def mse(m, b):
        return np.mean(errors(tbl, m, b)**2)
    return minimize(mse)
```

```
# Here is an example call to your function.  To test your function,
# figure out the right slope and intercept by hand.
example_table = Table().with_columns(
    "Speed (parsecs/year)", make_array(0, 1),
    "Distance (million parsecs)", make_array(1, 3))
fit_line(example_table)
```

[15]:
```
array([2., 1.])
```

[16]:
```
check('tests/q1_5.py')
```

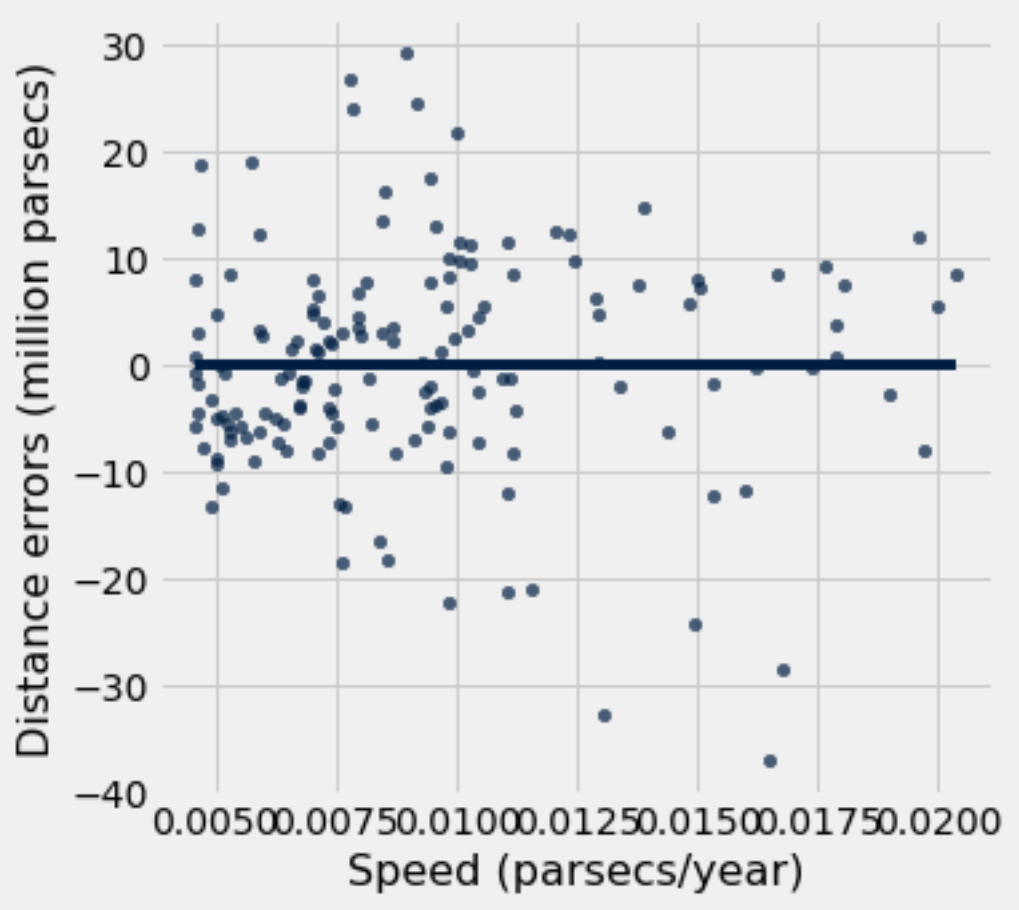
[16]:
```
<gofer.ok.OKTestsResult at 0x7f63e2cde290>
```

Question 1.6 Use your function to fit a line to `close_novas`. Then, set `new_errors` equal to the errors that we get calling `errors` with our new line. The cell below will graph the corresponding residual plot with a best fit line. Make sure that the residual plot makes sense.

Hint: What qualities should the best fit line of a residual plot have?

[17]:
```
best_line_slope, best_line_intercept = fit_line(close_novas)

new_errors = errors(close_novas, best_line_slope, best_line_intercept)

# This code displays the residual plot, given your values for the␣
 ↪best_line_slope and best_line_intercept
Table().with_columns("Speed (parsecs/year)",
                     close_novas.column("Speed (parsecs/year)"),
                     "Distance errors (million parsecs)",
                     new_errors
                    ).scatter(0, 1, fit_line=True)

# This just shows your answer as a nice string, in billions of years.
"Slope: {:g} (corresponding to an estimated age of {:,} billion years)".
 ↪format(best_line_slope, round(best_line_slope/1000, 4))
```

[17]:
```
'Slope: 14094.5 (corresponding to an estimated age of 14.0945 billion years)'
```



That slope (multiplied by 1 million) is an estimate of the age of the universe. The current best estimate of the age of the universe (using slightly more sophisticated techniques) is 13.799 billion years. Did we get close?

One reason our answer might be a little off is that we are using a sample of only some of the supernovae in the universe. Our sample isn't exactly random, since astronomers presumably chose the novae that were easiest to measure (or used some other nonrandom criteria). But let's assume it is. How can we produce a confidence interval for the age of the universe?

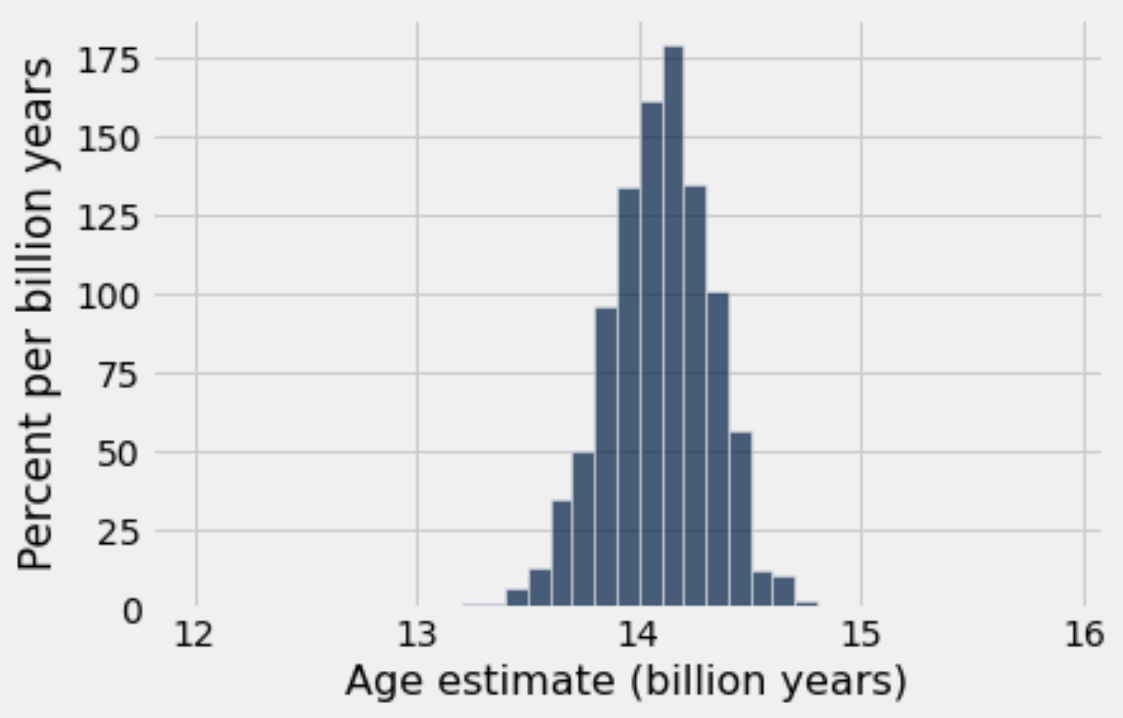
Question 1.7 It's time to bootstrap so that we can quantify the variability in our estimate! Simulate 1000 resamples from `close_novas`. For each resample, compute the slope of the least-squares regression line, and multiply it by 1 million to compute an estimate of the age of the universe. Store these ages in an array called `bootstrap_ages`, and then use them to compute a 95% confidence interval for the age of the universe.

Note: This might take up to a minute, and more repetitions will take even longer.

[18]:
```
bootstrap_ages = make_array()
for i in np.arange(1000):
```

```
    bootstrap_ages = np.append(bootstrap_ages, fit_line(close_novas.
  ↪sample())[0]*1e6)

lower_end = np.percentile(bootstrap_ages, 2.5)
upper_end = np.percentile(bootstrap_ages, 97.5)
Table().with_column("Age estimate", bootstrap_ages*1e-9).hist(bins=np.
  ↪arange(12, 16, .1), unit="billion years")
print("95% confidence interval for the age of the universe: [{:g}, {:g}]␣
  ↪billion years".format(lower_end*1e-9, upper_end*1e-9))
```

```
95% confidence interval for the age of the universe: [13.5939, 14.5046] billion
years
```



[19]:
```
check('tests/q1_7.py')
```

[19]:
```
<gofer.ok.OKTestsResult at 0x7f63e2c55910>
```

Nice work, data astronomer! You can compare your result to the Planck Project 2015 results, which estimated the age of the universe to be 13.799±0.021 billion years.

1.2 2. Submission

Once you're finished, select "Save and Checkpoint" in the File menu. Your name and course section number should be in the first and last cell of the assignment. Be sure you have run all cells with code and that the output from that is showing.

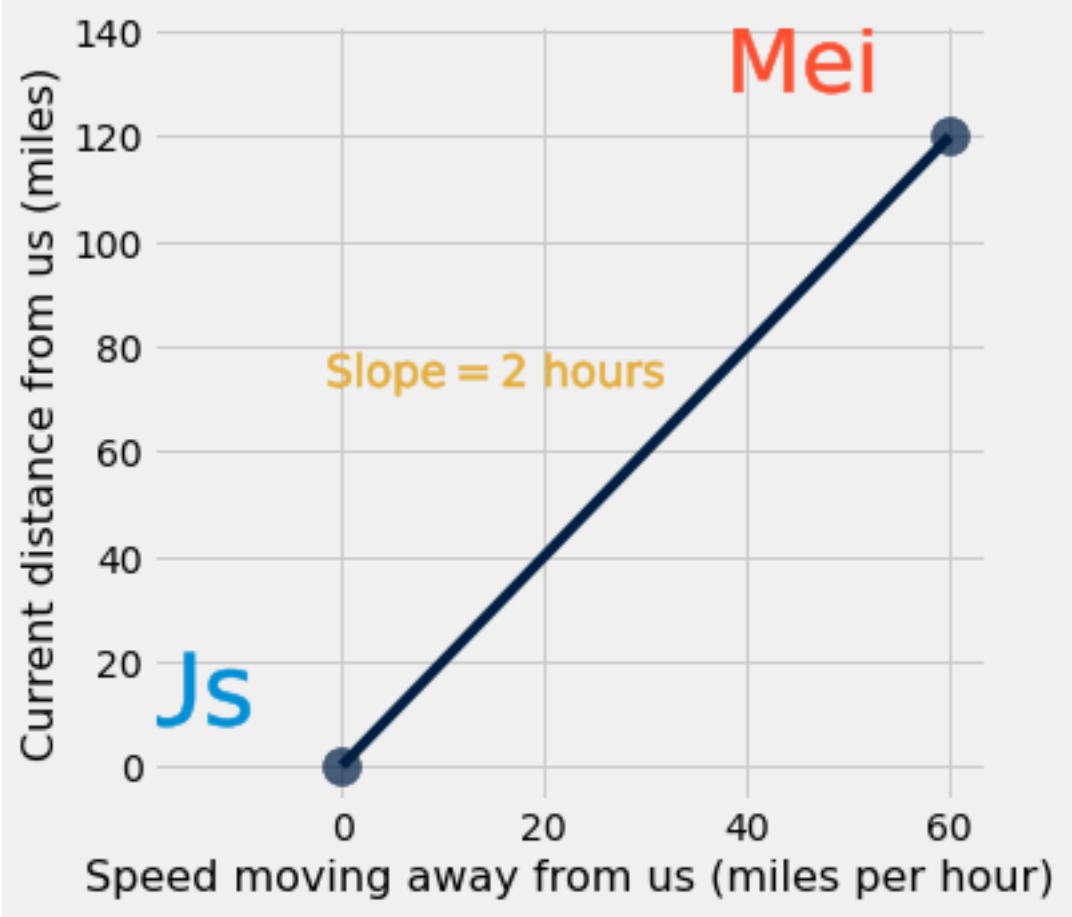
Double check that you have completed all of the free response questions as the autograder does NOT check that and YOU are responsible for knowing those questions are there and completing them as part of the grade for this lab. When ready, click "Print Preview" in the File menu. Print a copy from there in pdf format. (This means you right click and choose print and choose "save as pdf" from your printer options.) You will need to submit the pdf in Canvas by the deadline.

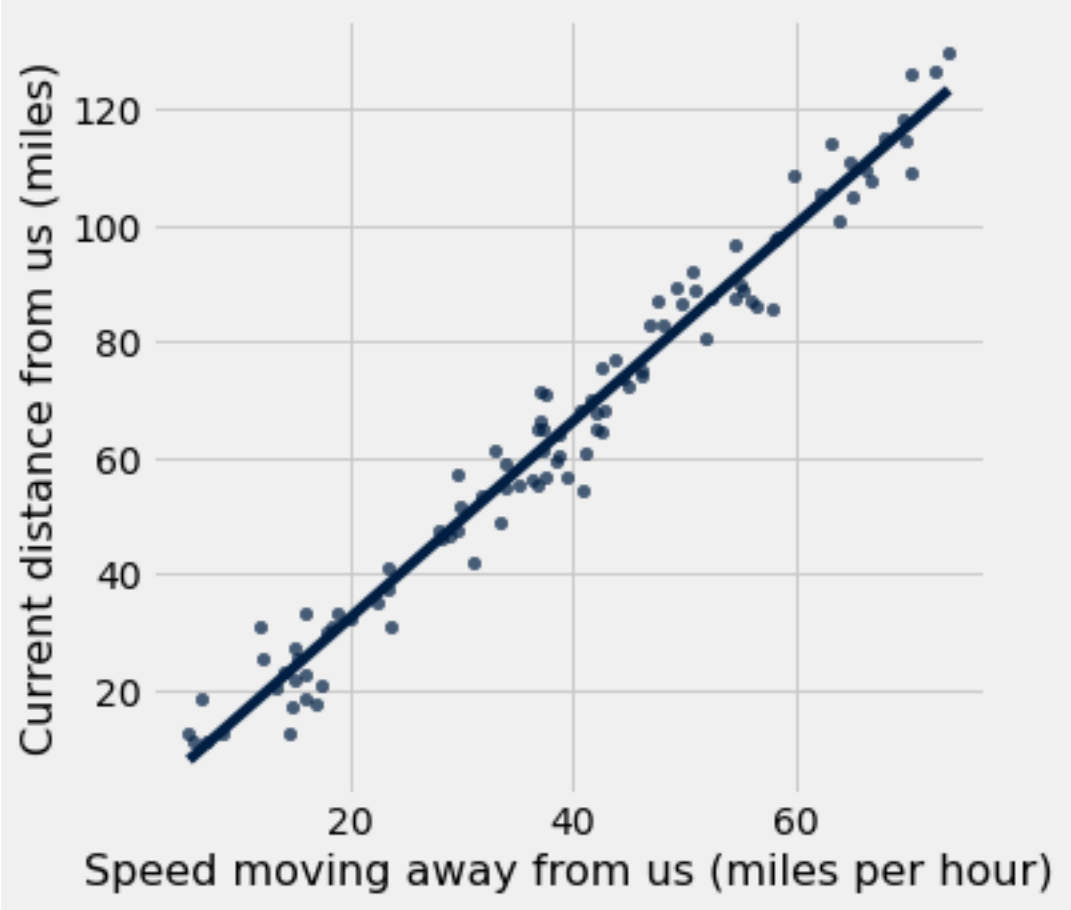
The gopher grader output and/or output from your coding are essential to helping your instructor grade your work correctly and in a timely manner.

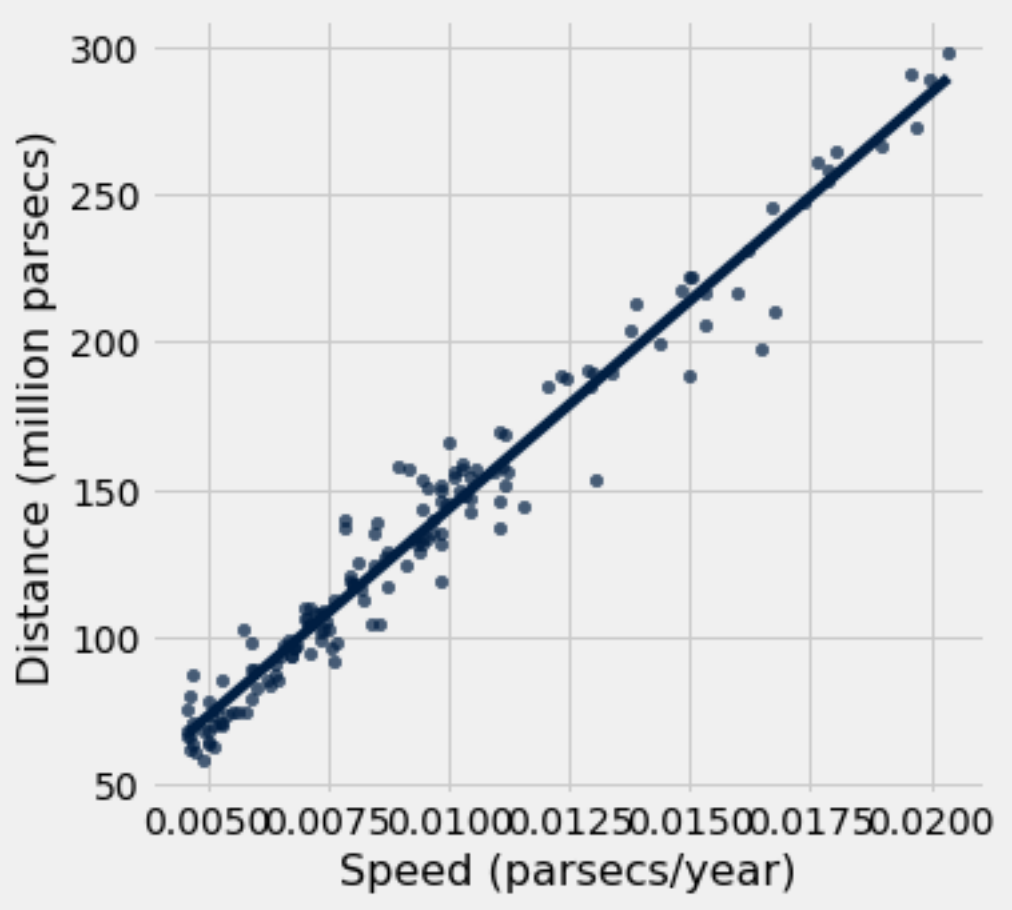
Files submitted that are missing the required output will lose some to all points so double check your pdf before submitting.

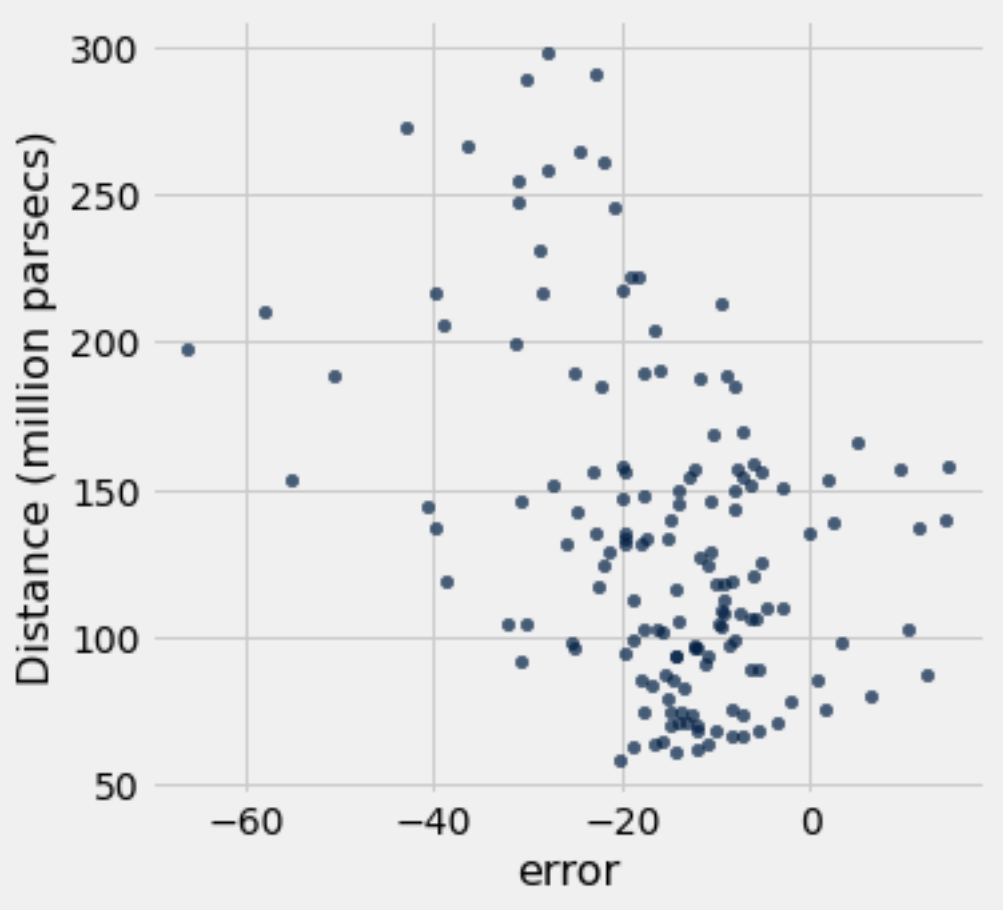
[20]:
```
# For your convenience, you can run this cell to run all the tests at once!
import glob
from gofer.ok import grade_notebook
if not globals().get('__GOFER_GRADER__', False):
    display(grade_notebook('lab13.ipynb', sorted(glob.glob('tests/q*.py'))))
```

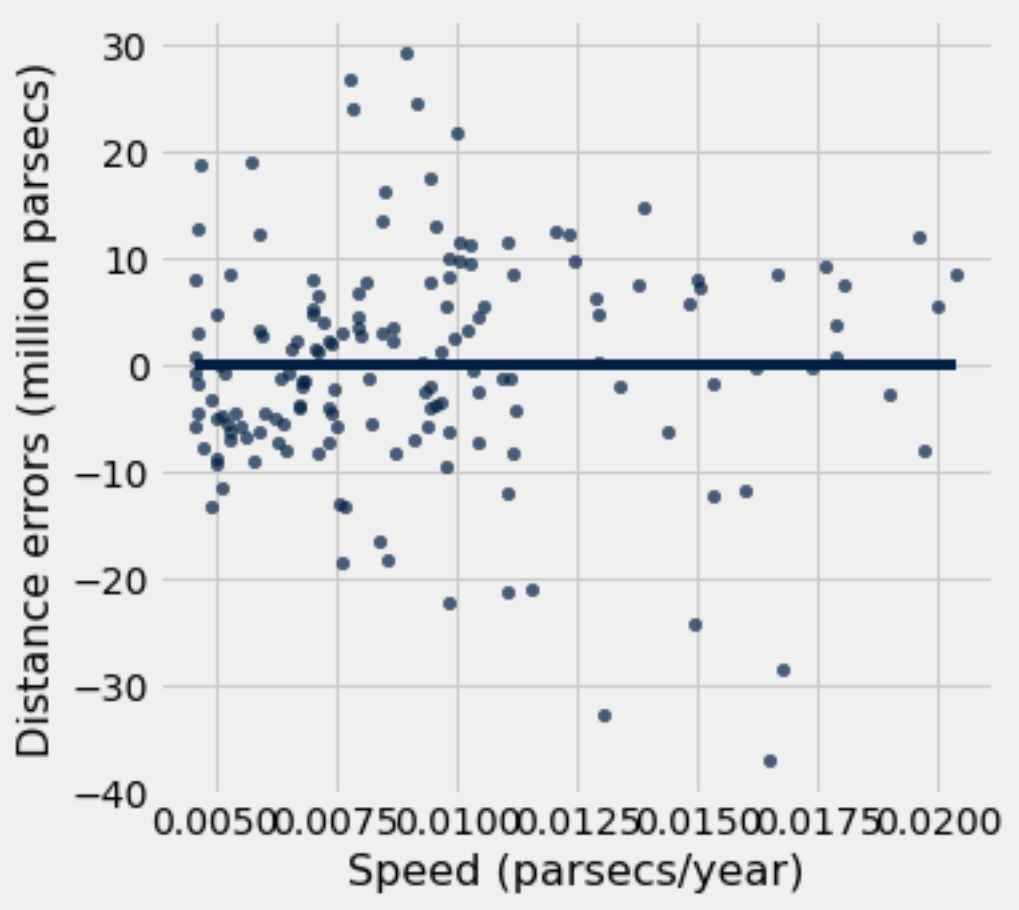
```
95% confidence interval for the age of the universe: [13.6545, 14.5041] billion
years
['tests/q1_1.py', 'tests/q1_2.py', 'tests/q1_4.py', 'tests/q1_5.py',
'tests/q1_7.py']
Question 1:

<gofer.ok.OKTestsResult at 0x7f63e04d5850>

Question 2:

<gofer.ok.OKTestsResult at 0x7f63e055a290>

Question 3:

<gofer.ok.OKTestsResult at 0x7f63e03b3ad0>

Question 4:

<gofer.ok.OKTestsResult at 0x7f63e038b9d0>

Question 5:

<gofer.ok.OKTestsResult at 0x7f63e03a96d0>

1.0
```

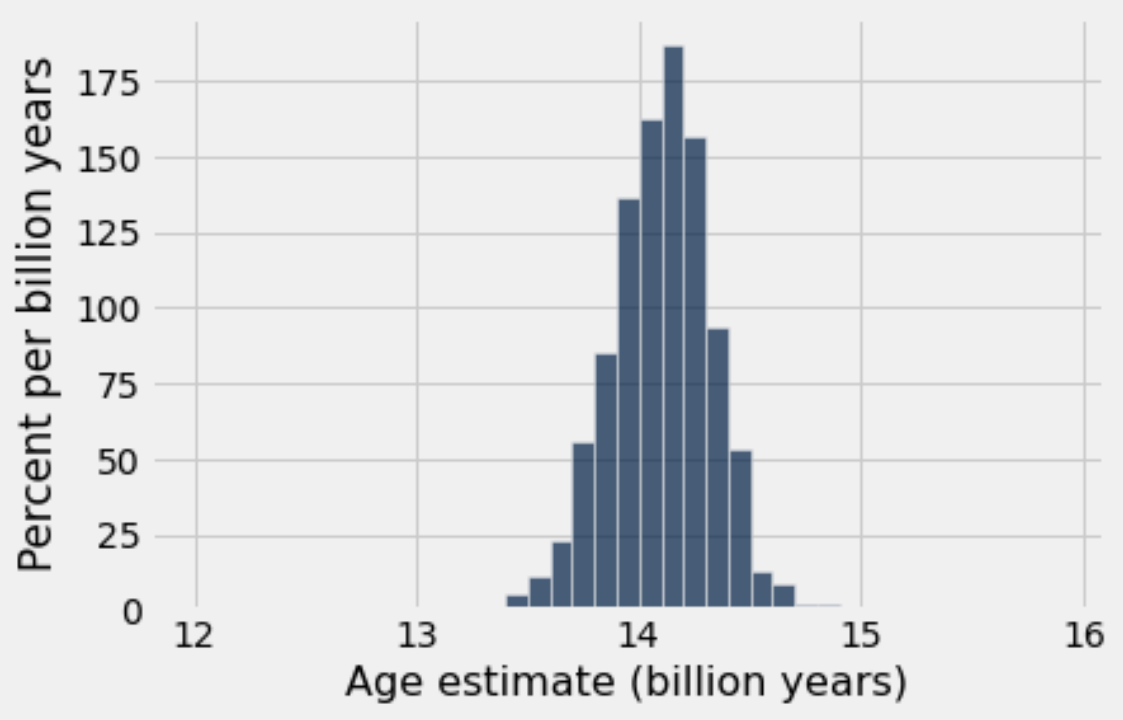

Mei
Slope = 2 hours
Js
Current distance from us (miles)
140
120
100
80
60
40
20
0
0
20
40
60
Speed moving away from us (miles per hour)

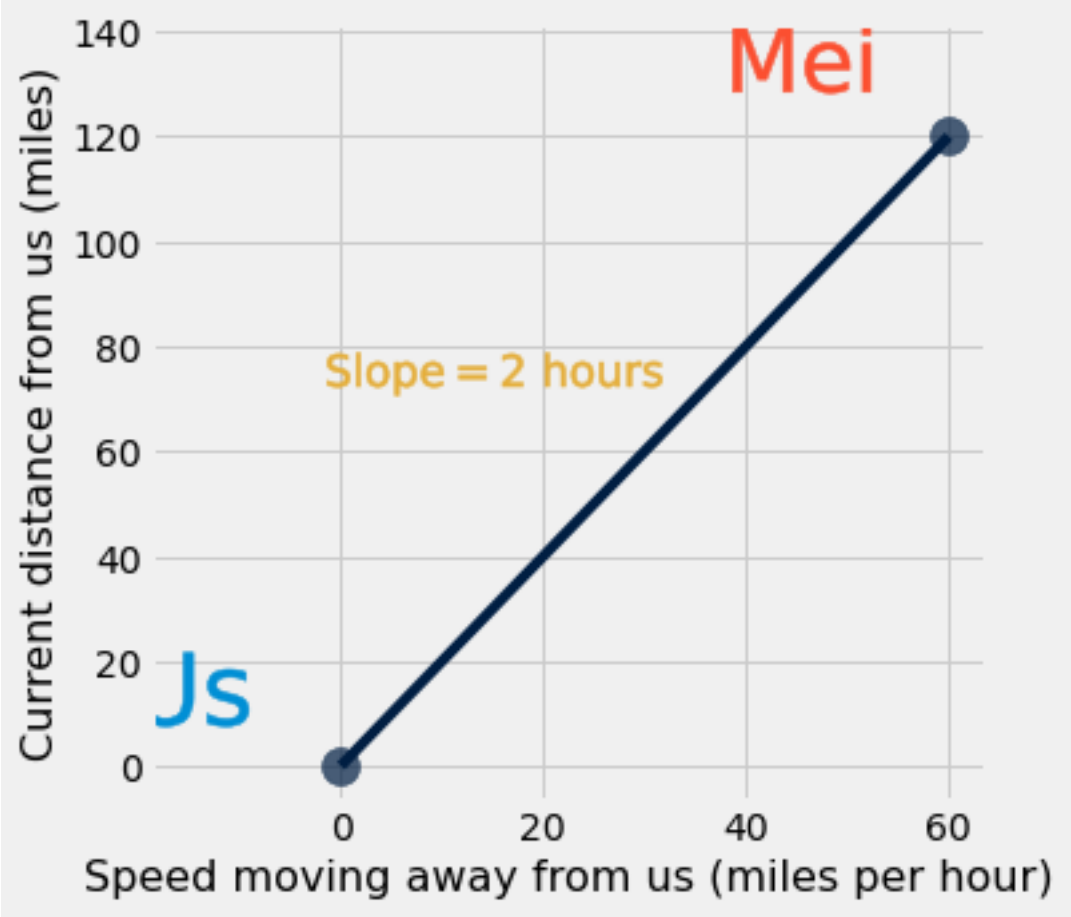

Current distance from us (miles)
120
100
80
60
40
20
20
40
60
Speed moving away from us (miles per hour)

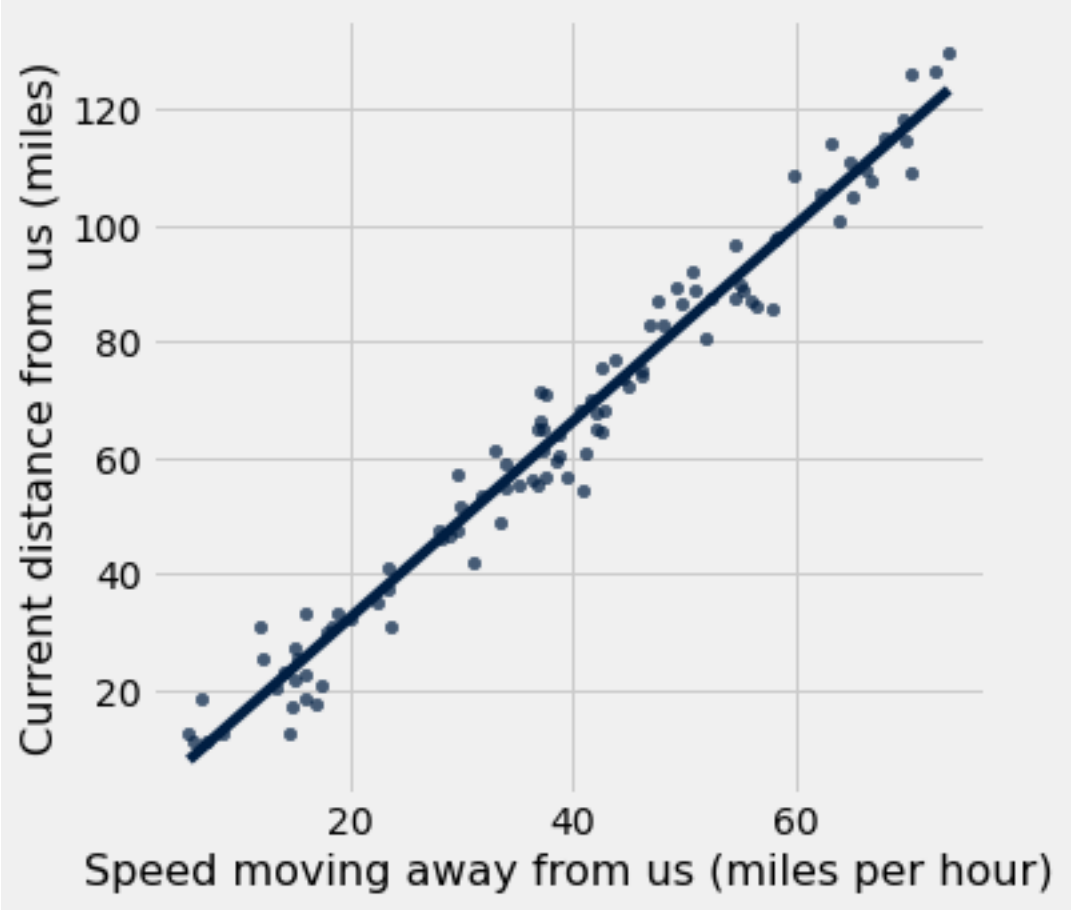

Distance (million parsecs)
300
250
200
150
100
50
0.0050
0.0075
0.0100
0.0125
0.0150
0.0175
0.0200
Speed (parsecs/year)

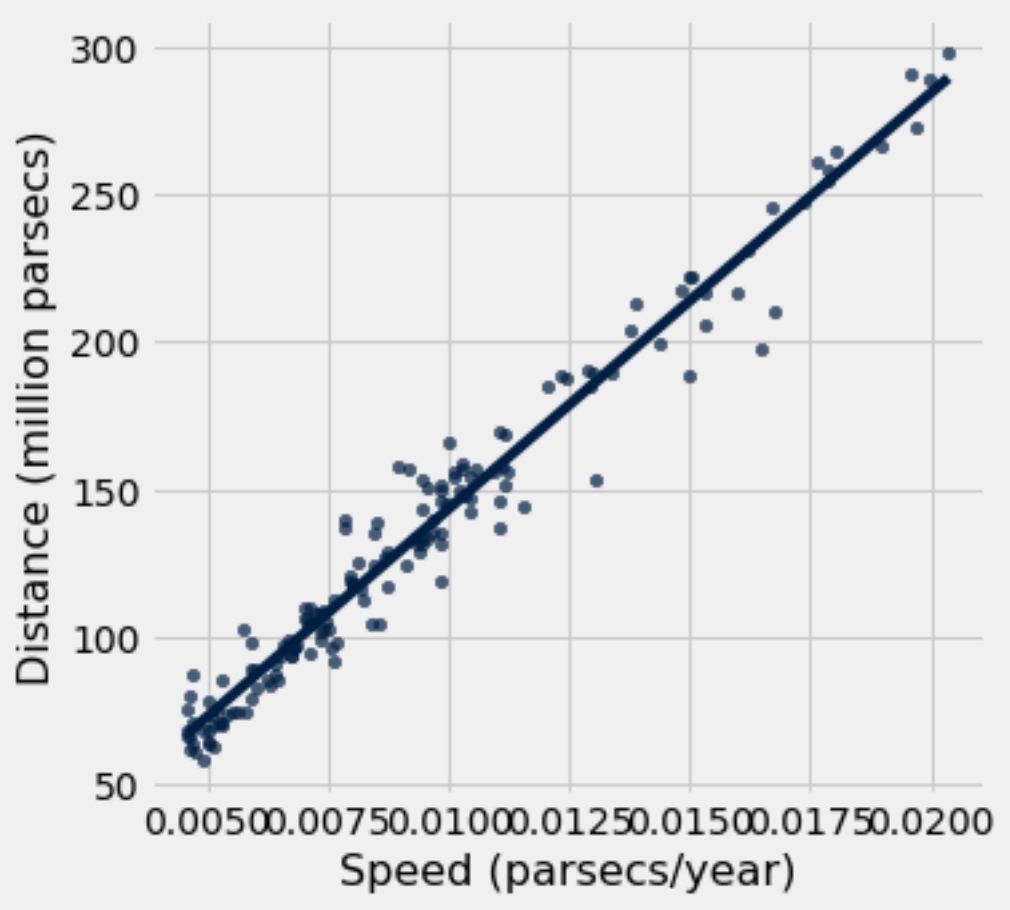

300
250
200
150
100
50
Distance (million parsecs)
−60
−40
−20
0
error

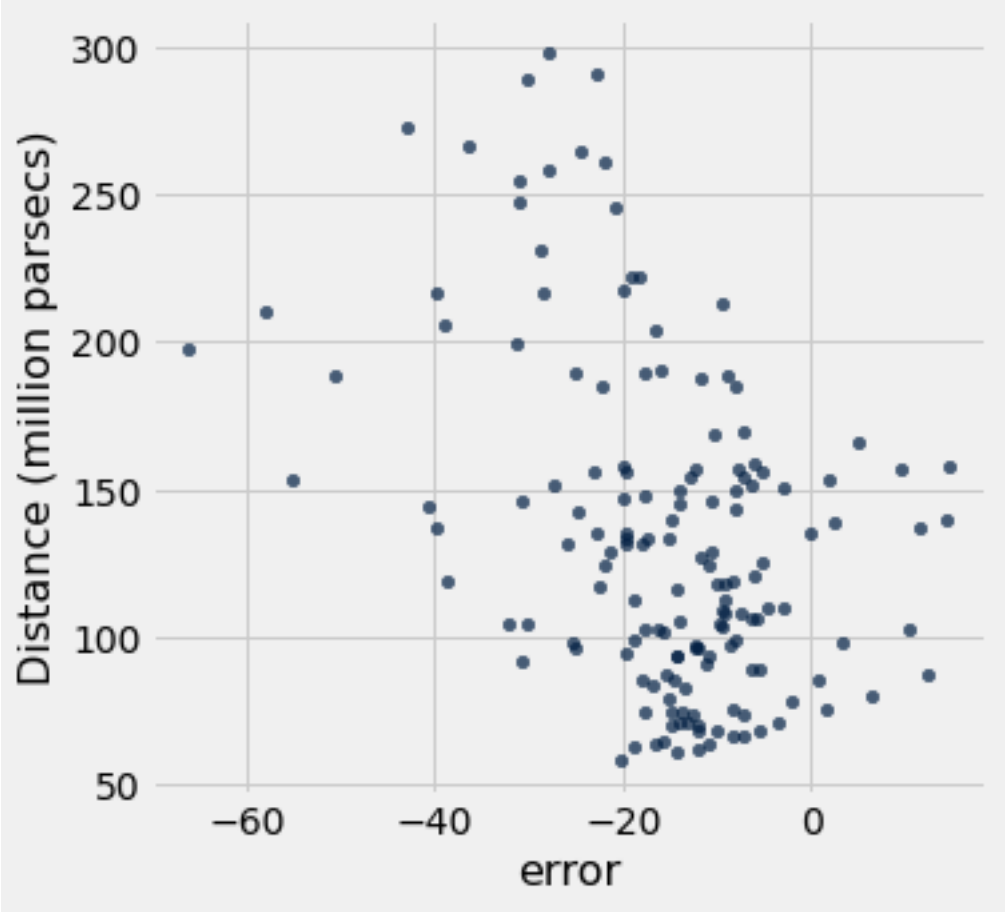

Distance errors (million parsecs)
30
20
10
0
−10
−20
−30
−40
0.0050
0.0075
0.0100
0.0125
0.0150
0.0175
0.0200
Speed (parsecs/year)

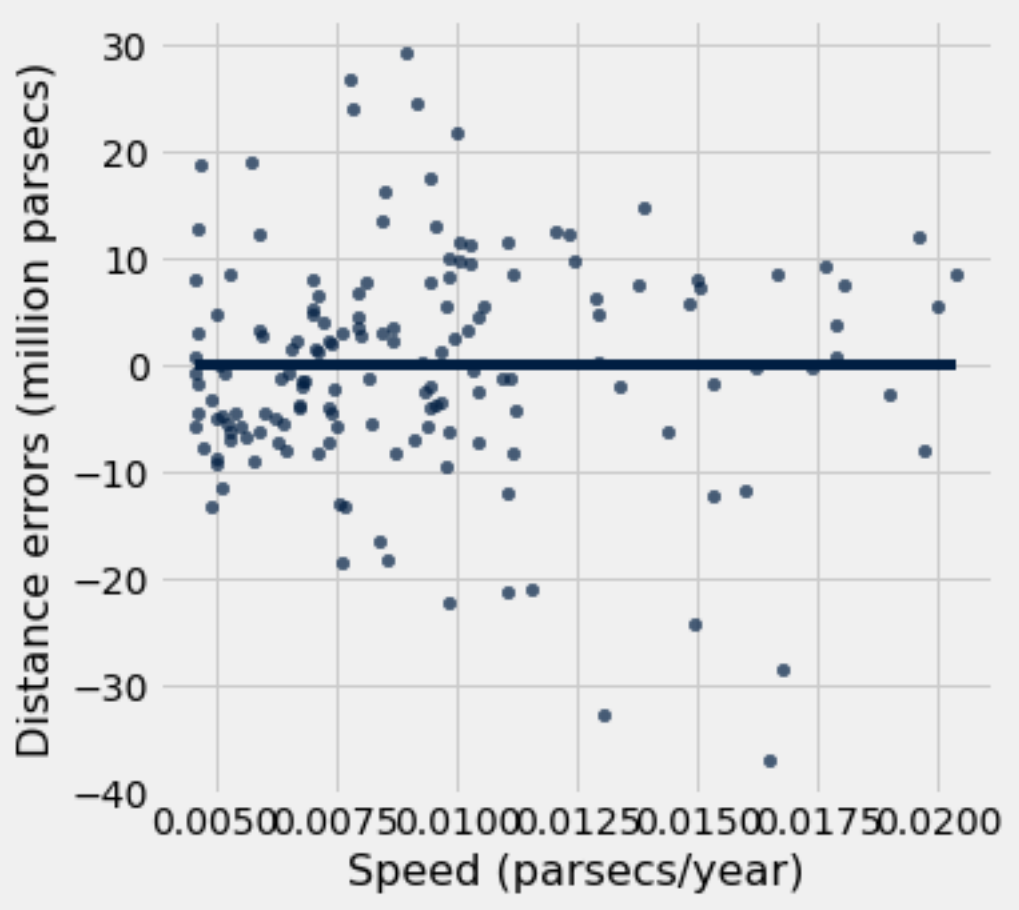

Percent per billion years
175
150
125
100
75
50
25
0
12
13
14
15
16
Age estimate (billion years)

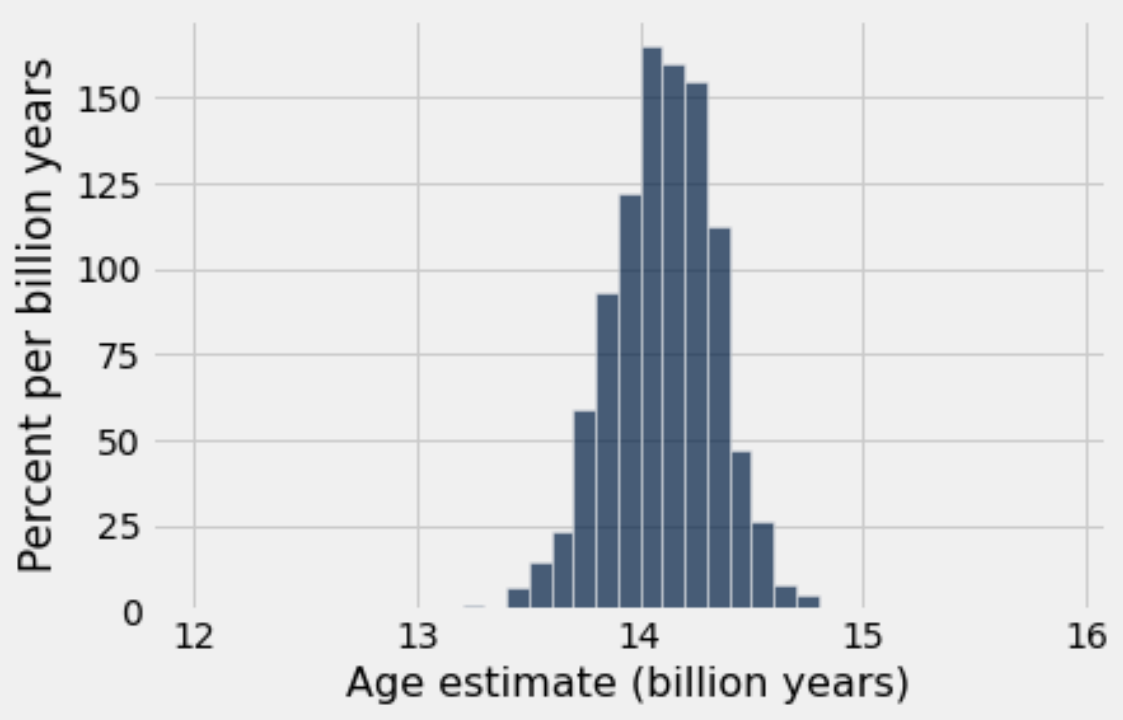

Mei
Slope = 2 hours
Js
Current distance from us (miles)
140
120
100
80
60
40
20
0
0
20
40
60
Speed moving away from us (miles per hour)


Current distance from us (miles)
120
100
80
60
40
20
20
40
60
Speed moving away from us (miles per hour)


Distance (million parsecs)
300
250
200
150
100
50
0.0050
0.0075
0.0100
0.0125
0.0150
0.0175
0.0200
Speed (parsecs/year)


300
250
200
150
100
50
Distance (million parsecs)
−60
−40
−20
0
error


Distance errors (million parsecs)
30
20
10
0
−10
−20
−30
−40
0.0050
0.0075
0.0100
0.0125
0.0150
0.0175
0.0200
Speed (parsecs/year)



Name: Allan Gongora

Section: 0131