Linear_Regression_with_TensorFlow

September 26, 2021

Copyright 2020 Google LLC.

```
[ ]: # Licensed under the Apache License, Version 2.0 (the "License");
     # you may not use this file except in compliance with the License.
     # You may obtain a copy of the License at
     #
     # https://www.apache.org/licenses/LICENSE-2.0
     #
     # Unless required by applicable law or agreed to in writing, software
     # distributed under the License is distributed on an "AS IS" BASIS,
     # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
     # See the License for the specific language governing permissions and
     # limitations under the License.
```

1 Linear Regression with TensorFlow

In previous units we learned about regression and about how to build and apply a regression model using scikit-learn. For many regression cases, `scikit-learn` is more than adequate. However, there are times when more powerful tools are needed. TensorFlow is one of those tools. It is a computational toolkit built to perform machine learning and data science tasks at scale.

1.1 Problem Framing

Machine learning is one of a variety of solutions that might work for solving a problem. It is always important to understand the problem space before diving in and starting to clean data and code.

In this lab we would like to be able to **predict the price of a house**.

Questions we should ask ourselves might include the following:

- Predict the price when? Now? In the past? In the future? For what range?
- Where are we predicting prices for? One market? One state? One country?
- What is our tolerance for being wrong?
- Are we okay with a few huge outliers if the overall model is better?
- What metrics are we using to define success and what are the acceptable values?
- Is there an non-ML way to solve this problem?
- What data is available to solve the problem?

The list of questions is boundless. Eventually you'll need to move on, but understanding the problem and the solution space is vital.

For this problem we'll further define the problem by saying:

> We want to create a system that predicts the prices of houses in California in 1990. We have census data from 1990 available to build and test the system. We will accept a system with a root mean squared error of 200,000 or better.

Since this is a contrived example, we'll shortcut around the question of choosing a technique and say that our analysis has led us to believe that we want to use a linear regression model to serve as our prediction system.

1.2 Exploratory Data Analysis

The dataset we'll use for this Colab contains California housing data taken from the 1990 census data. This is a popular dataset for experimenting with machine learning models.

As with any data science project, it is a good idea to take some time and review the data schema and description. Ask yourself the following:

- What data is available? What are the columns?
- What do those columns mean?
- What data types are those columns?
- What is the granularity of the data? In this particular case, what is a "block"?
- How many rows of data are there?
- Roughly how big is the data? Kilobytes? Megabytes? Gigabytes? Terabytes? More?
- Are any of the columns highly correlated?
- What bias is contained in the data?

1.2.1 Load the Data

Now that we have an understanding of the data that we are going to use in our model, let's load it into this Colab and examine it more closely.

Since the data is hosted on Kaggle, you'll need to upload your `kaggle.json` file to this lab and then run the code block below.

[ ]:
```
! chmod 600 kaggle.json && (ls ~/.kaggle 2>/dev/null || mkdir ~/.kaggle) && mv␣
↪kaggle.json ~/.kaggle/ && echo 'Done'
```

```
chmod: kaggle.json: No such file or directory
```

Once you are done, use the `kaggle` command to download the file into the lab.

[ ]:
```
!kaggle datasets download camnugent/california-housing-prices
!ls
```

```
california-housing-prices.zip: Skipping, found more recently modified local copy
(use --force to force download)
Linear_Regression_with_TensorFlow.ipynb slides.md
california-housing-prices.zip           slides.pptx
colab-key.zip
```

We now have a file called `california-housing-prices.zip` that we can load into a `DataFrame`.

[ ]:
```python
import pandas as pd

housing_df = pd.read_csv('california-housing-prices.zip')

housing_df
```

[ ]:
```
       longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0        -122.23     37.88                41.0        880.0           129.0
1        -122.22     37.86                21.0       7099.0          1106.0
2        -122.24     37.85                52.0       1467.0           190.0
3        -122.25     37.85                52.0       1274.0           235.0
4        -122.25     37.85                52.0       1627.0           280.0
…            …         …                  …            …              …
20635    -121.09     39.48                25.0       1665.0           374.0
20636    -121.21     39.49                18.0        697.0           150.0
20637    -121.22     39.43                17.0       2254.0           485.0
20638    -121.32     39.43                18.0       1860.0           409.0
20639    -121.24     39.37                16.0       2785.0           616.0

       population  households  median_income  median_house_value  \
0           322.0       126.0         8.3252            452600.0
1          2401.0      1138.0         8.3014            358500.0
2           496.0       177.0         7.2574            352100.0
3           558.0       219.0         5.6431            341300.0
4           565.0       259.0         3.8462            342200.0
…             …           …              …                  …
20635       845.0       330.0         1.5603             78100.0
20636       356.0       114.0         2.5568             77100.0
20637      1007.0       433.0         1.7000             92300.0
20638       741.0       349.0         1.8672             84700.0
20639      1387.0       530.0         2.3886             89400.0

      ocean_proximity
0            NEAR BAY
1            NEAR BAY
2            NEAR BAY
3            NEAR BAY
4            NEAR BAY
…                …
20635          INLAND
20636          INLAND
20637          INLAND
20638          INLAND
20639          INLAND
```

```
[20640 rows x 10 columns]
```

1.2.2 Exploration

You should always look at your data and statistics about that data before you begin modelling it. First, let's see the columns and data types that we have available.

```
[ ]: housing_df.dtypes
```

```
[ ]: longitude             float64
     latitude              float64
     housing_median_age    float64
     total_rooms           float64
     total_bedrooms        float64
     population            float64
     households            float64
     median_income         float64
     median_house_value    float64
     ocean_proximity        object
     dtype: object
```

Eight floating point features, one object features, and a floating point target, `median_house_value`. This is what we expect based on the data documentation.

Statistics It is a good idea to also describe the dataset. We use the `include='all'` argument to ensure our object column is also described.

```
[ ]: housing_df.describe(include='all')
```

```
[ ]:          longitude      latitude  housing_median_age   total_rooms  \
     count  20640.000000  20640.000000        20640.000000  20640.000000
     unique          NaN           NaN                 NaN           NaN
     top             NaN           NaN                 NaN           NaN
     freq            NaN           NaN                 NaN           NaN
     mean    -119.569704     35.631861           28.639486   2635.763081
     std        2.003532      2.135952           12.585558   2181.615252
     min     -124.350000     32.540000            1.000000      2.000000
     25%     -121.800000     33.930000           18.000000   1447.750000
     50%     -118.490000     34.260000           29.000000   2127.000000
     75%     -118.010000     37.710000           37.000000   3148.000000
     max     -114.310000     41.950000           52.000000  39320.000000

            total_bedrooms    population    households  median_income  \
     count    20433.000000  20640.000000  20640.000000   20640.000000
     unique            NaN           NaN           NaN            NaN
     top               NaN           NaN           NaN            NaN
     freq              NaN           NaN           NaN            NaN
     mean       537.870553   1425.476744    499.539680       3.870671
```

```
std          421.385070    1132.462122     382.329753        1.899822
min            1.000000       3.000000       1.000000        0.499900
25%          296.000000     787.000000     280.000000        2.563400
50%          435.000000    1166.000000     409.000000        3.534800
75%          647.000000    1725.000000     605.000000        4.743250
max         6445.000000   35682.000000    6082.000000       15.000100

        median_house_value ocean_proximity
count         20640.000000           20640
unique                 NaN               5
top                    NaN       <1H OCEAN
freq                   NaN            9136
mean         206855.816909             NaN
std          115395.615874             NaN
min           14999.000000             NaN
25%          119600.000000             NaN
50%          179700.000000             NaN
75%          264725.000000             NaN
max          500001.000000             NaN
```

In this case we can see that all of the column counts are the same except for `total_bedrooms`, which seems to be missing some data.

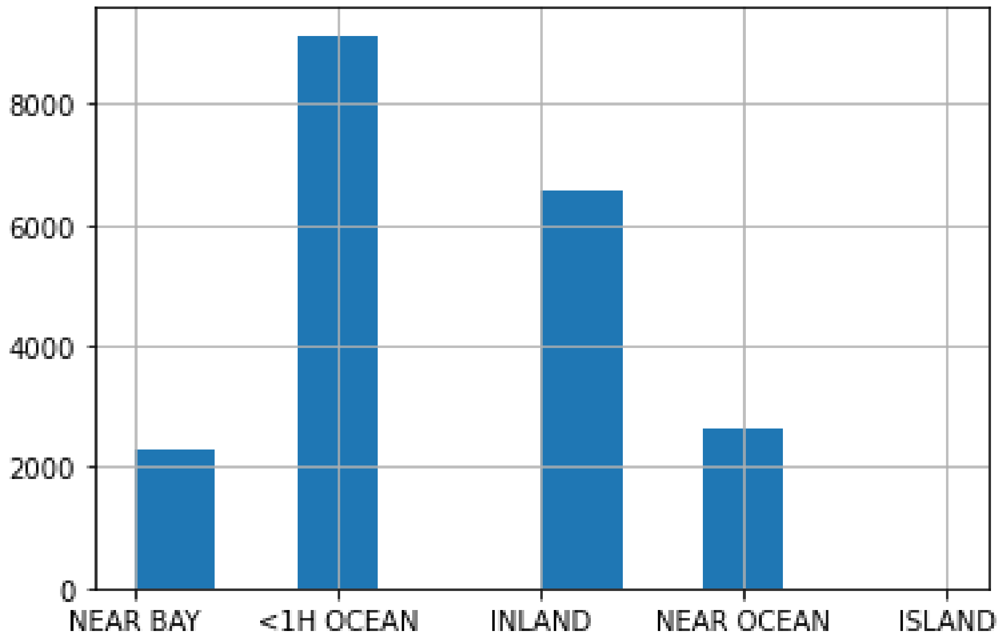
Looking at the min and max can be helpful, too. Does a 2.0 value for a minimum number of rooms for a block match your mental model of what a block is? What about that max of 39,320 rooms? In cases like this, it can be useful to research your topic area.

In this particular case, those numbers might be okay as long as the dense block is in an urban area with very dense and tall buildings on the block. As you probe a dataset, you should ask yourself questions like this. When something doesn't look right, investigate it.

Also notice the ocean proximity column. It has five unique values. Let's see what they are.

[ ]:
```
housing_df['ocean_proximity'].hist()
```

[ ]:
```
<AxesSubplot:>
```



We used `.hist()` so that we can also see the distribution. We can see the five values and can see that the largest group is `<1H OCEAN`. We don't seem to have any values missing.

Exercise 1: Sanity Check Use Pandas to find the row of data that contains the census block with the largest number of rooms. Search for the latitude and longitude for that location and answer the questions below.

Student Solution

[ ]:
```python
# Your Code Goes Here
import pandas as pd

beds = housing_df['total_rooms']

max_value = beds.max()


maxRoom = housing_df[housing_df['total_rooms'] == max_value]
maxRoom
```

[ ]:
```
       longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
13139    -121.44     38.43                 3.0      39320.0          6210.0

       population  households  median_income  median_house_value  \
13139     16305.0      5358.0         4.9516            153700.0
```

```
      ocean_proximity
13139          INLAND
```

1. What city is the block located in? > *Elk Grove California
2. Are 39320.0 total rooms reasonable? Why? > It's reasonable

---

Sampling It is also a good idea to take a look at the actual data. We can use Panda's `head()`, `tail()`, and/or `sample()` methods to do this.

[ ]:
```
housing_df.head(10)
```

[ ]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms
0	-122.23	37.88	41.0	880.0	129.0
1	-122.22	37.86	21.0	7099.0	1106.0
2	-122.24	37.85	52.0	1467.0	190.0
3	-122.25	37.85	52.0	1274.0	235.0
4	-122.25	37.85	52.0	1627.0	280.0
5	-122.25	37.85	52.0	919.0	213.0
6	-122.25	37.84	52.0	2535.0	489.0
7	-122.25	37.84	52.0	3104.0	687.0
8	-122.26	37.84	42.0	2555.0	665.0
9	-122.25	37.84	52.0	3549.0	707.0

	population	households	median_income	median_house_value	ocean_proximity
0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	496.0	177.0	7.2574	352100.0	NEAR BAY
3	558.0	219.0	5.6431	341300.0	NEAR BAY
4	565.0	259.0	3.8462	342200.0	NEAR BAY
5	413.0	193.0	4.0368	269700.0	NEAR BAY
6	1094.0	514.0	3.6591	299200.0	NEAR BAY
7	1157.0	647.0	3.1200	241400.0	NEAR BAY
8	1206.0	595.0	2.0804	226700.0	NEAR BAY
9	1551.0	714.0	3.6912	261100.0	NEAR BAY

Did you gain any insight from peeking at the actual data? Is the data sorted in a manner that might lead to a bad model?

In this case the data seems to be sorted ascending by longitude and possibly secondarily descending by latitude. We need to consider this when sampling or splitting the data.

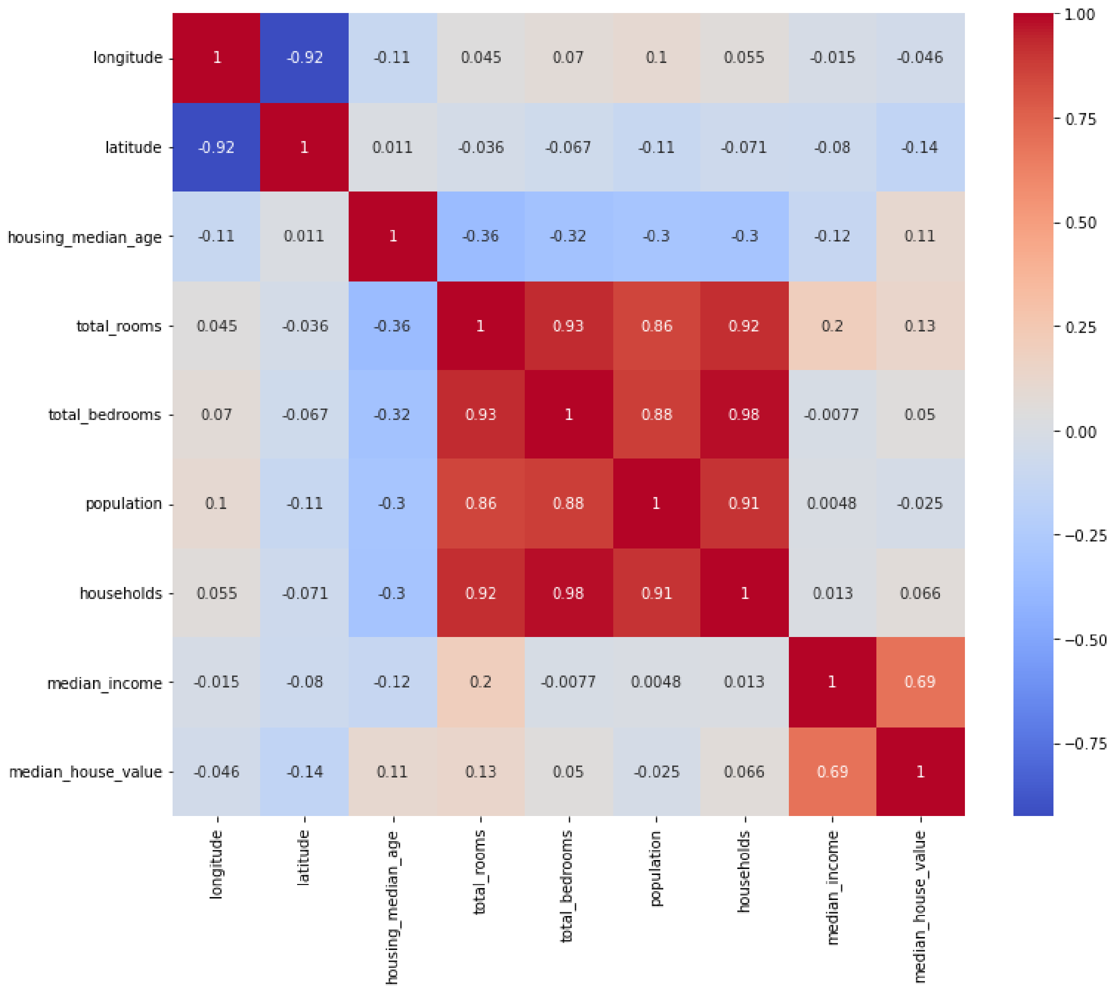
Also, there are some duplicate latitude and longitude values. This seems odd, but will be difficult to troubleshoot. For now, we'll just accept that the values are okay.

Correlation It is important to understand how columns relate to one another. Every feature that you add to your training set increases the amount of work that must be done to train your model. If you can find columns with a high degree of correlation, you can potentially not use one of the columns in your training and still get a model that performs well.

Let's create a correlation matrix heatmap for our data set.

```
[ ]: import matplotlib.pyplot as plt
     import seaborn as sns

     plt.figure(figsize=(12,10))
     _ = sns.heatmap(housing_df.corr(), cmap='coolwarm', annot=True)
```



Exercise 2: Correlated Columns Answer the following questions about the correlation between columns in our dataset.

Student Solution

1. Which columns are the most highly correlated? > Households and total bedrooms
2. Which column is most strongly correlated with `median_house_value`? > median house values and median inccome
3. Which columns have the strongest negative correlation? > Longitude and Latitude

1.2.3 Data Preprocessing

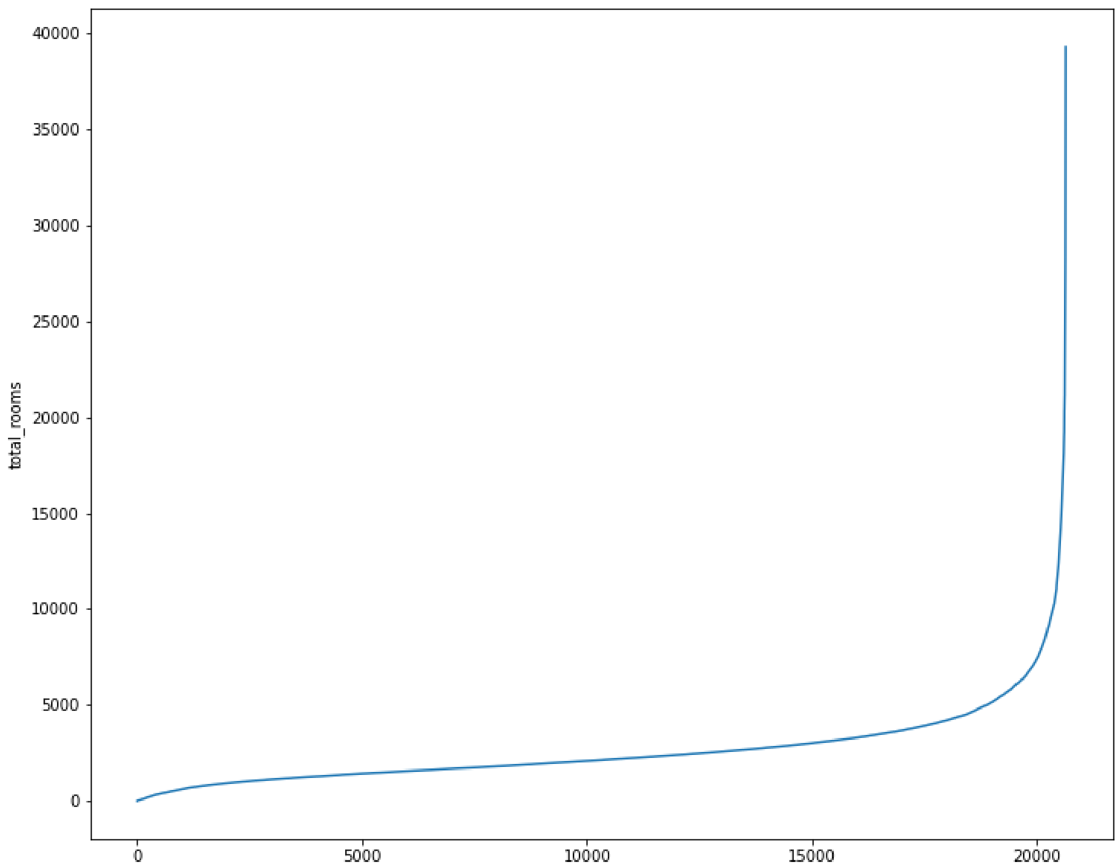
Now is the stage where we would perform model-independent preprocessing to the data to repair any missing data. Since there isn't very much missing data, we don't have much pre-processing to do.

Let's look at those room counts again, though. The values seem a little odd.

First we'll plot the room counts in ascending order.

[ ]:
```
rooms = housing_df['total_rooms'].sort_values().reset_index(drop=True)

plt.figure(figsize=(12,10))
_ = sns.lineplot(x=rooms.index.values, y=rooms)
```



That's quite a spike there at the end!

Looking at the chart, let's pick a point where the number of rooms really starts to extremely slope upward, say 10,000. If we chose to drop the rows with really large values, what would that do to our data?

[ ]:
```
many_rooms = rooms[rooms > 10000].size

percent = (many_rooms / rooms.size) * 100

print(f'{many_rooms} blocks have more than 10000 rooms ' +
      f'which is {percent:0.2f}% of our data')
```

```
287 blocks have more than 10000 rooms which is 1.39% of our data
```

So we'd knock out over 1% of our data by trying to remove what we think are outliers. That's not horrible, but it's probably not something we would want to do on a hunch.

We do need to fix our missing total bedrooms data. There are a few strategies that we could use:

- Fill in the values with the mean of the `total_bedrooms` data in the dataset.
- Fill in the values with zero.
- Find the closest lat/long values and use the mean of them.
- Find the ratio of `total_bedrooms` to `total_rooms` and multiply it against the `total_rooms` values that correspond with the missing `total_bedrooms` values.

Which is best?

Each method has pros and cons. For instance, using the dataset-wide mean values might lead to some unrealistic values if the blocks happen to be in extremely dense or extremely rural areas.

Filling in with zeros in this case works, but seems like a lazy approach that we can be pretty sure is not accurate.

Finding the closest lat/long that has a value and using its value - or the mean of a few of the closest lat/longs - is tempting since density probably changes slowly. However, this might be difficult to do, even with the data being sorted by latitude and longitude.

Finding the ratio of total rooms to bedrooms seems like a reasonable compromise since we have `total_rooms` data for every row, and the two columns are highly correlated. Using these values, we can derive a reasonable guess for the number of bedrooms.

The code to find the ratio sums the values for total bedrooms and total rooms across all fully-populated rows in the dataset.

[ ]:
```
has_all_data = housing_df[~housing_df['total_bedrooms'].isna()]

sums = has_all_data[['total_bedrooms', 'total_rooms']].sum().tolist()

bedrooms_to_total_rooms_ratio = sums[0] / sums[1]

bedrooms_to_total_rooms_ratio
```

[ ]: 0.20400898497877112

If we think that the outliers might throw off the ratio, we can check the median.

[ ]:
```
has_all_data = housing_df[~housing_df['total_bedrooms'].isna()]

sums = has_all_data[['total_bedrooms', 'total_rooms']].median().tolist()

bedrooms_to_total_rooms_ratio = sums[0] / sums[1]

bedrooms_to_total_rooms_ratio
```

[ ]:
```
0.20451339915373765
```

It seems to match the mean pretty closely.

Now we just need to patch the data.

[ ]:
```
missing_total_bedrooms_idx = housing_df['total_bedrooms'].isna()

housing_df.loc[missing_total_bedrooms_idx, 'total_bedrooms'] = housing_df[
    missing_total_bedrooms_idx]['total_rooms'] * bedrooms_to_total_rooms_ratio

housing_df.describe()
```

[ ]:
```
          longitude      latitude  housing_median_age   total_rooms  \
count  20640.000000  20640.000000        20640.000000  20640.000000
mean    -119.569704     35.631861           28.639486   2635.763081
std        2.003532      2.135952           12.585558   2181.615252
min     -124.350000     32.540000            1.000000      2.000000
25%     -121.800000     33.930000           18.000000   1447.750000
50%     -118.490000     34.260000           29.000000   2127.000000
75%     -118.010000     37.710000           37.000000   3148.000000
max     -114.310000     41.950000           52.000000  39320.000000

       total_bedrooms    population    households  median_income  \
count    20640.000000  20640.000000  20640.000000   20640.000000
mean       537.732315   1425.476744    499.539680       3.870671
std        420.856140   1132.462122    382.329753       1.899822
min          1.000000      3.000000      1.000000       0.499900
25%        295.000000    787.000000    280.000000       2.563400
50%        435.000000   1166.000000    409.000000       3.534800
75%        647.000000   1725.000000    605.000000       4.743250
max       6445.000000  35682.000000   6082.000000      15.000100

       median_house_value
count        20640.000000
mean        206855.816909
std         115395.615874
min          14999.000000
25%         119600.000000
50%         179700.000000
```

```
75%          264725.000000
max          500001.000000
```

1.3 Modeling

It is time to actually build our model. In this case, we know we are going to build a linear regression model using TensorFlow. We could build the model by hand, but luckily we don't have too. TensorFlow provides many pre-built models in its estimator library. We are going to use the `tensorflow.estimator.LinearRegressor` model.

The `Estimator` class is the base class for TensorFlow estimators. Its methods define the API for estimators. In the remainder of this lab, we will create an instance of `LinearRegressor` and use the `Estimator` API to train the model and make test predictions.

1.3.1 Prepare the Data

Earlier we considered preprocessing the data. That preprocessing was intended to be more generic preprocessing that needed to be done to correct errors with the data set.

Now that we have chosen a model, we need to do specific preprocessing related to the type of model that we'll be building and how we are going to test and train the model.

Initially we'll be using the `LinearRegressor` with default options. For measuring model quality, we will perform hold-out testing with 20% of the data being held out for test.

Normalization The scale and range of data in each column of our dataset varies widely. In many models larger values will be over-considered in training. In order to combat this we can *normalize* our data.

Note that we only want to normalize the feature data so let's first create variables to hold our feature and target column names.

[ ]:
```
target_column = 'median_house_value'
feature_columns = [c for c in housing_df.columns if c != target_column]



#features versus median_house_value
target_column, feature_columns
```

[ ]:
```
('median_house_value',
 ['longitude',
  'latitude',
  'housing_median_age',
  'total_rooms',
  'total_bedrooms',
  'population',
  'households',
  'median_income',
  'ocean_proximity'])
```

Also remember that `ocean_proximity` contains string values, so let's separate our features even more.

[ ]:
```
numeric_feature_columns = [c for c in feature_columns if c != 'ocean_proximity']

numeric_feature_columns
```

[ ]:
```
['longitude',
 'latitude',
 'housing_median_age',
 'total_rooms',
 'total_bedrooms',
 'population',
 'households',
 'median_income']
```

To normalize, we subtract the minimum value from each column and then divide by the delta between the min and max. This should make all of our feature values fall into the range of 0.0 to 1.0. You can see in the `describe()` output that we now have a min values of 0.0 and max values of 1.0.

[ ]:
```
housing_df.loc[:, numeric_feature_columns] = (
    housing_df[numeric_feature_columns] -
      housing_df[numeric_feature_columns].min()) / (
          housing_df[numeric_feature_columns].max() -
            housing_df[numeric_feature_columns].min())

housing_df[numeric_feature_columns].describe()
```

[ ]:
```
          longitude      latitude  housing_median_age   total_rooms  \
count  20640.000000  20640.000000        20640.000000  20640.000000
mean       0.476125      0.328572            0.541951      0.066986
std        0.199555      0.226988            0.246776      0.055486
min        0.000000      0.000000            0.000000      0.000000
25%        0.253984      0.147715            0.333333      0.036771
50%        0.583665      0.182784            0.549020      0.054046
75%        0.631474      0.549416            0.705882      0.080014
max        1.000000      1.000000            1.000000      1.000000

       total_bedrooms    population    households  median_income
count    20640.000000  20640.000000  20640.000000   20640.000000
mean         0.083292      0.039869      0.081983       0.232464
std          0.065310      0.031740      0.062873       0.131020
min          0.000000      0.000000      0.000000       0.000000
25%          0.045624      0.021974      0.045881       0.142308
50%          0.067349      0.032596      0.067094       0.209301
75%          0.100248      0.048264      0.099326       0.292641
max          1.000000      1.000000      1.000000       1.000000
```

Another option would be to *standardize* the data. Standardization is the process of subtracting the mean from each column and then dividing by the standard deviation. We chose not to do that in this case because that creates negative values, which don't work well with this model.

Should we modify the target in any way?

Let's take a look at the values again.

```
[ ]: housing_df[target_column].describe()
```

```
[ ]: count     20640.000000
     mean     206855.816909
     std      115395.615874
     min       14999.000000
     25%      119600.000000
     50%      179700.000000
     75%      264725.000000
     max      500001.000000
     Name: median_house_value, dtype: float64
```

Those are some pretty big values. It does look like there is a ceiling of 500,001 applied to the data and a minimum value of 14,999.

Given enough time, our model could train to predict values this large. However, we are going to be using a pretty small learning rate by default with the `Ftrl` optimizer: 0.0001. In order to speed things up, we can shrink the values in the target column by some constant.

```
[ ]: TARGET_FACTOR = 100000

     housing_df[target_column] = housing_df[target_column] / TARGET_FACTOR

     housing_df[target_column].describe()
```

```
[ ]: count    20640.000000
     mean         2.068558
     std          1.153956
     min          0.149990
     25%          1.196000
     50%          1.797000
     75%          2.647250
     max          5.000010
     Name: median_house_value, dtype: float64
```

We've reduced the values from the range of 14,999-500,001 to 0.14999-5.0. This should allow the model to converge faster. Of course, now our predictions will need to be multiplied by 100,000 in order to reflect real dollar values.

Train/Test Split We want to go ahead and divide our data into testing and training splits. For this example we'll hold out 20% of the data for testing.

One easy way to do that is just to slice the data. Our data is sorted by latitude and longitude, however, so we need to shuffle it first so that we aren't testing with data from just one location in California.

[ ]:
```
# Shuffle
housing_df = housing_df.sample(frac=1)

# Calculate test set size
test_set_size = int(len(housing_df) * 0.2)

# Split the data
testing_df = housing_df[:test_set_size]
training_df = housing_df[test_set_size:]

print(f'Holding out {len(testing_df)} records for testing. ')
print(f'Using {len(training_df)} records for training.')
```

```
Holding out 4128 records for testing.
Using 16512 records for training.
```

1.3.2 Load TensorFlow

Next, we'll load the TensorFlow library.

TensorFlow released version 2.0 in late 2019. As of the writing of the lab, Colab supports both versions 1 and 2, but it defaults to version 1. In order to tell Colab to use TensorFlow 2, you need to run the magic in the cell below.

[ ]:
```
%tensorflow_version 2.x
```

```
UsageError: Line magic function `%tensorflow_version` not found.
```

Next, we'll load TensorFlow and check to make sure that we are running version 2.

[ ]:
```
import tensorflow as tf
tf.__version__
```

[ ]:
```
'2.4.1'
```

Finally, we can set some global settings for TensorFlow. In this case we want to ensure that any time there is a question about the size of a floating point value that it is processed as a 64-bit number.

[ ]:
```
tf.keras.backend.set_floatx('float64')
```

1.3.3 TensorFlow Data Set

`DataFrame` is a container for a dataset in Pandas. To process the data with TensorFlow we need to get the data in the `DataFrame` into a TensorFlow Dataset.

Since our housing data fits in memory, we can use the `from_tensor_slices` class method to create our `Dataset`. There are a few different data formats that we could pass the method, but our model expects a feature map and a list of labels.

A feature map is a Python dictionary with feature names for keys and an iterable of column values as the value. Labels are just an iterable of our target values.

Below, we create the test and training `DataSet` objects.

[ ]:
```
testing_ds = tf.data.Dataset.from_tensor_slices((
    {c: testing_df[c] for c in feature_columns},  # feature map
    testing_df[target_column]                     # labels
))

training_ds = tf.data.Dataset.from_tensor_slices((
    {c: training_df[c] for c in feature_columns},  # feature map
    training_df[target_column]                     # labels
))

testing_ds, training_ds
```

```
2021-09-26 12:51:59.782711: I tensorflow/core/platform/cpu_feature_guard.cc:142]
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical
operations:  SSE4.1 SSE4.2
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
```

[ ]:
```
(<TensorSliceDataset shapes: ({longitude: (), latitude: (), housing_median_age:
(), total_rooms: (), total_bedrooms: (), population: (), households: (),
median_income: (), ocean_proximity: ()}, ()), types: ({longitude: tf.float64,
latitude: tf.float64, housing_median_age: tf.float64, total_rooms: tf.float64,
total_bedrooms: tf.float64, population: tf.float64, households: tf.float64,
median_income: tf.float64, ocean_proximity: tf.string}, tf.float64)>,
 <TensorSliceDataset shapes: ({longitude: (), latitude: (), housing_median_age:
(), total_rooms: (), total_bedrooms: (), population: (), households: (),
median_income: (), ocean_proximity: ()}, ()), types: ({longitude: tf.float64,
latitude: tf.float64, housing_median_age: tf.float64, total_rooms: tf.float64,
total_bedrooms: tf.float64, population: tf.float64, households: tf.float64,
median_income: tf.float64, ocean_proximity: tf.string}, tf.float64)>)
```

The code above runs and displays two `TensorSliceDataset` objects that seem to have the correct columns. However, we can't tell how many rows of data each contains.

Intuitively you'd think this would be as simple as asking for the length of the data sets from Python:

```
len(testing_ds)
len(training_ds)
```

This won't work, though. TensorFlow Dataset objects can represent in-memory data, like what we have now. They can also represent data in multiple sources stored in different locations. They can even represent a stream of data that is never-ending. For this reason having a standard `len` is impossible.

Because of this], we'll need to do a little more work to get a count of the data in a TensorFlow dataset. To get a count, we'll use the `reduce` operation. This operation takes an initial value, in our case 0, and then performs some function over and over for each row in the dataset. In this case we just add one for each value. The reduction returns values for each row and feeds it to the next. The final row simply returns the value to the runtime.

We can see below that the `reduce` operation counts the number of rows for the testing and training dataset and they both match the values we saw above in the Colab.

[ ]:
```
import numpy as np

testing_ds_count = testing_ds.reduce(np.int64(0), lambda x, _: x + 1)
training_ds_count = training_ds.reduce(np.int64(0), lambda x, _: x + 1)

print(testing_ds_count.numpy())
print(training_ds_count.numpy())
```

```
2021-09-26 12:51:59.876385: I
tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:116] None of the MLIR
optimization passes are enabled (registered 2)

4128
16512
```

1.3.4 LinearRegressor

The model that we'll use is the LinearRegressor. This class complies with the TensorFlow Estimator API. This API takes care of a lot of the low-level model plumbing, and exposes convenient methods for performing model training, evaluation, and inference.

Though the `LinearRegressor` has many configuration options, only feature columns have to be specified when the regressor is created.

We provide the regressor feature columns as a list of columns that we'd like the model to use for training and prediction. For now that will be every one of our features. Most of these columns are all floating point numbers so we use a list expansion to create a list of `float64` `numeric_column` objects.

For the `ocean_proximity` column we create a categorical column. This converts the values in the column into numbers matching their index in the vocabulary list.

A warning will be issued if you don't specify a `model_dir`. For now that's fine since we don't plan on saving our model and plan to train it completely now. If we do specify a model directory, state will be saved, which can cause issues as you iterate on the design of the model.

[ ]:
```
housing_features = [
    tf.feature_column.numeric_column(c, dtype=tf.dtypes.float64)
      for c in numeric_feature_columns
]
```

```
housing_features.append(
    tf.feature_column.categorical_column_with_vocabulary_list(
        key='ocean_proximity',
        vocabulary_list=sorted(housing_df['ocean_proximity'].unique()))
)

linear_regressor = tf.estimator.LinearRegressor(
    feature_columns=housing_features,
)

linear_regressor
```

```
INFO:tensorflow:Using default config.
WARNING:tensorflow:Using temporary folder as model directory:
/var/folders/q7/wrxzkb515gqcskvhd38dwx6h0000gn/T/tmpz3v1h5qc
INFO:tensorflow:Using config: {'_model_dir':
'/var/folders/q7/wrxzkb515gqcskvhd38dwx6h0000gn/T/tmpz3v1h5qc',
'_tf_random_seed': None, '_save_summary_steps': 100, '_save_checkpoints_steps':
None, '_save_checkpoints_secs': 600, '_session_config': allow_soft_placement:
true
graph_options {
  rewrite_options {
    meta_optimizer_iterations: ONE
  }
}
, '_keep_checkpoint_max': 5, '_keep_checkpoint_every_n_hours': 10000,
'_log_step_count_steps': 100, '_train_distribute': None, '_device_fn': None,
'_protocol': None, '_eval_distribute': None, '_experimental_distribute': None,
'_experimental_max_worker_delay_secs': None, '_session_creation_timeout_secs':
7200, '_checkpoint_save_graph_def': True, '_service': None, '_cluster_spec':
ClusterSpec({}), '_task_type': 'worker', '_task_id': 0, '_global_id_in_cluster':
0, '_master': '', '_evaluation_master': '', '_is_chief': True,
'_num_ps_replicas': 0, '_num_worker_replicas': 1}
```

[ ]:
```
<tensorflow_estimator.python.estimator.canned.linear.LinearRegressorV2 at
0x7fc16944cbb0>
```

If we had multiple workers, we could distribute the training and evaluation of the model by using a distribution strategy. In the example below, you can see that we are using a `MirroredStrategy` to spread out the work.

More information on distributing `Estimator` work can be found in the TensorFlow documentation.

[ ]:
```
housing_features = [
    tf.feature_column.numeric_column(c, dtype=tf.dtypes.float64)
      for c in numeric_feature_columns
]
```

```python
housing_features.append(
    tf.feature_column.categorical_column_with_vocabulary_list(
        key='ocean_proximity',
        vocabulary_list=sorted(housing_df['ocean_proximity'].unique()))
)

mirrored_strategy = tf.distribute.MirroredStrategy()
config = tf.estimator.RunConfig(
    train_distribute=mirrored_strategy,
    eval_distribute=mirrored_strategy,
)

linear_regressor = tf.estimator.LinearRegressor(
    feature_columns=housing_features,
    config=config,
)

linear_regressor
```

```
WARNING:tensorflow:There are non-GPU devices in `tf.distribute.Strategy`, not
using nccl allreduce.
INFO:tensorflow:Using MirroredStrategy with devices
('/job:localhost/replica:0/task:0/device:CPU:0',)
INFO:tensorflow:Initializing RunConfig with distribution strategies.
INFO:tensorflow:Not using Distribute Coordinator.
WARNING:tensorflow:Using temporary folder as model directory:
/var/folders/q7/wrxzkb515gqcskvhd38dwx6h0000gn/T/tmpype0sfb1
INFO:tensorflow:Using config: {'_model_dir':
'/var/folders/q7/wrxzkb515gqcskvhd38dwx6h0000gn/T/tmpype0sfb1',
'_tf_random_seed': None, '_save_summary_steps': 100, '_save_checkpoints_steps':
None, '_save_checkpoints_secs': 600, '_session_config': allow_soft_placement:
true
graph_options {
  rewrite_options {
    meta_optimizer_iterations: ONE
  }
}
, '_keep_checkpoint_max': 5, '_keep_checkpoint_every_n_hours': 10000,
'_log_step_count_steps': 100, '_train_distribute':
<tensorflow.python.distribute.mirrored_strategy.MirroredStrategy object at
0x7fc15dcfc4f0>, '_device_fn': None, '_protocol': None, '_eval_distribute':
<tensorflow.python.distribute.mirrored_strategy.MirroredStrategy object at
0x7fc15dcfc4f0>, '_experimental_distribute': None,
'_experimental_max_worker_delay_secs': None, '_session_creation_timeout_secs':
7200, '_checkpoint_save_graph_def': True, '_service': None, '_cluster_spec':
ClusterSpec({}), '_task_type': 'worker', '_task_id': 0, '_global_id_in_cluster':
0, '_master': '', '_evaluation_master': '', '_is_chief': True,
'_num_ps_replicas': 0, '_num_worker_replicas': 1,
```

```
'_distribute_coordinator_mode': None}
```

[ ]:
```
<tensorflow_estimator.python.estimator.canned.linear.LinearRegressorV2 at
0x7fc169517b50>
```

1.3.5 Training Input Function

The LinearRegressor that we just created is still not trained. To train the model we need to call the train method and pass it an input function that provides a `Dataset` to extract data from.

We saw how to create a `Dataset` earlier. It would be nice if we could reuse that `Dataset`, but TensorFlow requires that you create the `Dataset` in your function, so we'll use the same `Dataset` creation code from above.

We also need to change a few attributes of the dataset. Our training data only has 13600 records, which isn't a lot of data. We can choose to repeat the data so that it is fed to the model multiple times. In this case we chose to repeat it 10 times. Hopefully this will give the optimizer enough data to find a good solution.

Since we are repeating the same data over and over, we also are going to shuffle it in between repeats. This will add some variability to the training data.

Finally, we choose to process the data in batches of 100. These mini batches of 100 are used for a single optimization step.

[ ]:
```python
def training_input():
  ds = tf.data.Dataset.from_tensor_slices((
    {c: training_df[c] for c in feature_columns},  # feature map
    training_df[target_column]                     # labels
  ))
  ds = ds.repeat(100)
  ds = ds.shuffle(buffer_size=10000)
  ds = ds.batch(100)
  return ds
```

1.3.6 Training

We can now call the `train` method on the regressor, passing it the input function that we defined.

[ ]:
```python
linear_regressor.train(input_fn=training_input)
```

```
INFO:tensorflow:Calling model_fn.
```

```
/Users/josemartinez/opt/anaconda3/envs/data/lib/python3.9/site-
packages/tensorflow/python/keras/engine/base_layer_v1.py:1727: UserWarning:
`layer.add_variable` is deprecated and will be removed in a future version.
Please use `layer.add_weight` method instead.
  warnings.warn('`layer.add_variable` is deprecated and '
```

```
WARNING:tensorflow:From
/Users/josemartinez/opt/anaconda3/envs/data/lib/python3.9/site-
packages/tensorflow/python/keras/optimizer_v2/ftrl.py:133: calling
```

```
Constant.__init__ (from tensorflow.python.ops.init_ops) with dtype is deprecated
and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the
constructor
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Create CheckpointSaverHook.
WARNING:tensorflow:From
/Users/josemartinez/opt/anaconda3/envs/data/lib/python3.9/site-
packages/tensorflow_estimator/python/estimator/util.py:96:
DistributedIteratorV1.initialize (from tensorflow.python.distribute.input_lib)
is deprecated and will be removed in a future version.
Instructions for updating:
Use the iterator's `initializer` property instead.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
```

```
2021-09-26 12:52:02.908637: I
tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:196] None of the MLIR
optimization passes are enabled (registered 0 passes)
```

```
INFO:tensorflow:Calling checkpoint listeners before saving checkpoint 0…
INFO:tensorflow:Saving checkpoints for 0 into
/var/folders/q7/wrxzkb515gqcskvhd38dwx6h0000gn/T/tmpype0sfb1/model.ckpt.
INFO:tensorflow:Calling checkpoint listeners after saving checkpoint 0…
INFO:tensorflow:loss = 5.096572, step = 0
INFO:tensorflow:global_step/sec: 636.771
INFO:tensorflow:loss = 0.802818, step = 100 (0.157 sec)
INFO:tensorflow:global_step/sec: 1263.89
INFO:tensorflow:loss = 0.6617569, step = 200 (0.079 sec)
INFO:tensorflow:global_step/sec: 1353.07
INFO:tensorflow:loss = 0.80071974, step = 300 (0.074 sec)
INFO:tensorflow:global_step/sec: 1407.58
INFO:tensorflow:loss = 0.6671283, step = 400 (0.071 sec)
INFO:tensorflow:global_step/sec: 1390.74
INFO:tensorflow:loss = 0.58316916, step = 500 (0.072 sec)
INFO:tensorflow:global_step/sec: 1234.05
INFO:tensorflow:loss = 0.6678221, step = 600 (0.081 sec)
INFO:tensorflow:global_step/sec: 822.206
INFO:tensorflow:loss = 0.58931464, step = 700 (0.122 sec)
INFO:tensorflow:global_step/sec: 1157.74
INFO:tensorflow:loss = 0.64264405, step = 800 (0.087 sec)
INFO:tensorflow:global_step/sec: 1032.33
INFO:tensorflow:loss = 0.5650078, step = 900 (0.096 sec)
INFO:tensorflow:global_step/sec: 1120.05
INFO:tensorflow:loss = 0.36732674, step = 1000 (0.090 sec)
INFO:tensorflow:global_step/sec: 1348.72
INFO:tensorflow:loss = 0.48410568, step = 1100 (0.074 sec)
```

```
INFO:tensorflow:global_step/sec: 1318.18
INFO:tensorflow:loss = 0.3134191, step = 1200 (0.076 sec)
INFO:tensorflow:global_step/sec: 1283.19
INFO:tensorflow:loss = 0.6016485, step = 1300 (0.078 sec)
INFO:tensorflow:global_step/sec: 1408.8
INFO:tensorflow:loss = 0.41829845, step = 1400 (0.071 sec)
INFO:tensorflow:global_step/sec: 1443.25
INFO:tensorflow:loss = 0.3260663, step = 1500 (0.069 sec)
INFO:tensorflow:global_step/sec: 1417.68
INFO:tensorflow:loss = 0.5439154, step = 1600 (0.071 sec)
INFO:tensorflow:global_step/sec: 1446.02
INFO:tensorflow:loss = 0.40847456, step = 1700 (0.069 sec)
INFO:tensorflow:global_step/sec: 1346.57
INFO:tensorflow:loss = 0.57888246, step = 1800 (0.074 sec)
INFO:tensorflow:global_step/sec: 1361.23
INFO:tensorflow:loss = 0.6066516, step = 1900 (0.073 sec)
INFO:tensorflow:global_step/sec: 1463.32
INFO:tensorflow:loss = 0.50504905, step = 2000 (0.068 sec)
INFO:tensorflow:global_step/sec: 1461.58
INFO:tensorflow:loss = 0.50332785, step = 2100 (0.068 sec)
INFO:tensorflow:global_step/sec: 1410.66
INFO:tensorflow:loss = 0.57374483, step = 2200 (0.071 sec)
INFO:tensorflow:global_step/sec: 1340.41
INFO:tensorflow:loss = 0.468124, step = 2300 (0.075 sec)
INFO:tensorflow:global_step/sec: 1351.66
INFO:tensorflow:loss = 0.61617297, step = 2400 (0.074 sec)
INFO:tensorflow:global_step/sec: 1274.19
INFO:tensorflow:loss = 0.3654571, step = 2500 (0.078 sec)
INFO:tensorflow:global_step/sec: 1309.35
INFO:tensorflow:loss = 0.52632797, step = 2600 (0.076 sec)
INFO:tensorflow:global_step/sec: 1271.97
INFO:tensorflow:loss = 0.67714846, step = 2700 (0.079 sec)
INFO:tensorflow:global_step/sec: 1431.23
INFO:tensorflow:loss = 0.7357325, step = 2800 (0.070 sec)
INFO:tensorflow:global_step/sec: 1458.89
INFO:tensorflow:loss = 0.304365, step = 2900 (0.069 sec)
INFO:tensorflow:global_step/sec: 1434.55
INFO:tensorflow:loss = 0.47581983, step = 3000 (0.070 sec)
INFO:tensorflow:global_step/sec: 1310.92
INFO:tensorflow:loss = 0.47287706, step = 3100 (0.076 sec)
INFO:tensorflow:global_step/sec: 1304.31
INFO:tensorflow:loss = 0.49958584, step = 3200 (0.077 sec)
INFO:tensorflow:global_step/sec: 1411.65
INFO:tensorflow:loss = 0.64111376, step = 3300 (0.071 sec)
INFO:tensorflow:global_step/sec: 1432.13
INFO:tensorflow:loss = 0.5021939, step = 3400 (0.070 sec)
INFO:tensorflow:global_step/sec: 1416.13
INFO:tensorflow:loss = 0.41288993, step = 3500 (0.071 sec)
```

```
INFO:tensorflow:global_step/sec: 1161.14
INFO:tensorflow:loss = 0.39826813, step = 3600 (0.087 sec)
INFO:tensorflow:global_step/sec: 1009.03
INFO:tensorflow:loss = 0.5612414, step = 3700 (0.099 sec)
INFO:tensorflow:global_step/sec: 1035.05
INFO:tensorflow:loss = 0.52257067, step = 3800 (0.096 sec)
INFO:tensorflow:global_step/sec: 1241.52
INFO:tensorflow:loss = 0.5417849, step = 3900 (0.080 sec)
INFO:tensorflow:global_step/sec: 1329.03
INFO:tensorflow:loss = 0.62726736, step = 4000 (0.075 sec)
INFO:tensorflow:global_step/sec: 1336.54
INFO:tensorflow:loss = 0.5036971, step = 4100 (0.075 sec)
INFO:tensorflow:global_step/sec: 1335.14
INFO:tensorflow:loss = 0.44286492, step = 4200 (0.075 sec)
INFO:tensorflow:global_step/sec: 1355.75
INFO:tensorflow:loss = 0.6634143, step = 4300 (0.074 sec)
INFO:tensorflow:global_step/sec: 1300.66
INFO:tensorflow:loss = 0.44959012, step = 4400 (0.077 sec)
INFO:tensorflow:global_step/sec: 1033.09
INFO:tensorflow:loss = 0.679279, step = 4500 (0.099 sec)
INFO:tensorflow:global_step/sec: 801.872
INFO:tensorflow:loss = 0.4729192, step = 4600 (0.123 sec)
INFO:tensorflow:global_step/sec: 1050.17
INFO:tensorflow:loss = 0.43559426, step = 4700 (0.095 sec)
INFO:tensorflow:global_step/sec: 1172.54
INFO:tensorflow:loss = 0.39200056, step = 4800 (0.085 sec)
INFO:tensorflow:global_step/sec: 1108.34
INFO:tensorflow:loss = 0.58325124, step = 4900 (0.090 sec)
INFO:tensorflow:global_step/sec: 1418.25
INFO:tensorflow:loss = 0.6018573, step = 5000 (0.070 sec)
INFO:tensorflow:global_step/sec: 1368.27
INFO:tensorflow:loss = 0.6361645, step = 5100 (0.073 sec)
INFO:tensorflow:global_step/sec: 1331.29
INFO:tensorflow:loss = 0.4429782, step = 5200 (0.075 sec)
INFO:tensorflow:global_step/sec: 994.559
INFO:tensorflow:loss = 0.55307174, step = 5300 (0.101 sec)
INFO:tensorflow:global_step/sec: 862.708
INFO:tensorflow:loss = 0.48209354, step = 5400 (0.116 sec)
INFO:tensorflow:global_step/sec: 945.645
INFO:tensorflow:loss = 0.47383013, step = 5500 (0.104 sec)
INFO:tensorflow:global_step/sec: 1267.23
INFO:tensorflow:loss = 0.61423945, step = 5600 (0.079 sec)
INFO:tensorflow:global_step/sec: 1243.38
INFO:tensorflow:loss = 0.4152697, step = 5700 (0.080 sec)
INFO:tensorflow:global_step/sec: 1330.99
INFO:tensorflow:loss = 0.58131295, step = 5800 (0.075 sec)
INFO:tensorflow:global_step/sec: 1377.6
INFO:tensorflow:loss = 0.5316272, step = 5900 (0.073 sec)
```

```
INFO:tensorflow:global_step/sec: 1210.26
INFO:tensorflow:loss = 0.42011055, step = 6000 (0.083 sec)
INFO:tensorflow:global_step/sec: 1297.72
INFO:tensorflow:loss = 0.57136667, step = 6100 (0.077 sec)
INFO:tensorflow:global_step/sec: 1364.35
INFO:tensorflow:loss = 0.47671548, step = 6200 (0.073 sec)
INFO:tensorflow:global_step/sec: 1458.6
INFO:tensorflow:loss = 0.41010702, step = 6300 (0.069 sec)
INFO:tensorflow:global_step/sec: 1215.07
INFO:tensorflow:loss = 0.45463526, step = 6400 (0.083 sec)
INFO:tensorflow:global_step/sec: 872.583
INFO:tensorflow:loss = 0.5546354, step = 6500 (0.115 sec)
INFO:tensorflow:global_step/sec: 986.262
INFO:tensorflow:loss = 0.6503165, step = 6600 (0.101 sec)
INFO:tensorflow:global_step/sec: 1194.26
INFO:tensorflow:loss = 0.34377038, step = 6700 (0.085 sec)
INFO:tensorflow:global_step/sec: 1186.75
INFO:tensorflow:loss = 0.4526175, step = 6800 (0.083 sec)
INFO:tensorflow:global_step/sec: 1344.43
INFO:tensorflow:loss = 0.53024805, step = 6900 (0.074 sec)
INFO:tensorflow:global_step/sec: 1335.54
INFO:tensorflow:loss = 0.7777574, step = 7000 (0.075 sec)
INFO:tensorflow:global_step/sec: 1371.97
INFO:tensorflow:loss = 0.56572205, step = 7100 (0.073 sec)
INFO:tensorflow:global_step/sec: 1368.65
INFO:tensorflow:loss = 0.47675264, step = 7200 (0.073 sec)
INFO:tensorflow:global_step/sec: 1405.3
INFO:tensorflow:loss = 0.38141036, step = 7300 (0.071 sec)
INFO:tensorflow:global_step/sec: 1410.14
INFO:tensorflow:loss = 0.53225785, step = 7400 (0.071 sec)
INFO:tensorflow:global_step/sec: 1342.57
INFO:tensorflow:loss = 0.41957858, step = 7500 (0.075 sec)
INFO:tensorflow:global_step/sec: 1412.67
INFO:tensorflow:loss = 0.60250765, step = 7600 (0.071 sec)
INFO:tensorflow:global_step/sec: 1387.97
INFO:tensorflow:loss = 0.4198337, step = 7700 (0.072 sec)
INFO:tensorflow:global_step/sec: 1035.16
INFO:tensorflow:loss = 0.52355736, step = 7800 (0.097 sec)
INFO:tensorflow:global_step/sec: 938.365
INFO:tensorflow:loss = 0.4291622, step = 7900 (0.107 sec)
INFO:tensorflow:global_step/sec: 1059.55
INFO:tensorflow:loss = 0.6154276, step = 8000 (0.095 sec)
INFO:tensorflow:global_step/sec: 1330.57
INFO:tensorflow:loss = 0.49184364, step = 8100 (0.074 sec)
INFO:tensorflow:global_step/sec: 1464.6
INFO:tensorflow:loss = 0.26218092, step = 8200 (0.068 sec)
INFO:tensorflow:global_step/sec: 1460.86
INFO:tensorflow:loss = 0.3825206, step = 8300 (0.068 sec)
```

```
INFO:tensorflow:global_step/sec: 1457.21
INFO:tensorflow:loss = 0.47099578, step = 8400 (0.069 sec)
INFO:tensorflow:global_step/sec: 1470.98
INFO:tensorflow:loss = 0.5708747, step = 8500 (0.068 sec)
INFO:tensorflow:global_step/sec: 1430.69
INFO:tensorflow:loss = 0.6418402, step = 8600 (0.070 sec)
INFO:tensorflow:global_step/sec: 1406.49
INFO:tensorflow:loss = 0.51387244, step = 8700 (0.071 sec)
INFO:tensorflow:global_step/sec: 1356.52
INFO:tensorflow:loss = 0.2901835, step = 8800 (0.074 sec)
INFO:tensorflow:global_step/sec: 1299.29
INFO:tensorflow:loss = 0.5380206, step = 8900 (0.077 sec)
INFO:tensorflow:global_step/sec: 1310
INFO:tensorflow:loss = 0.5678397, step = 9000 (0.076 sec)
INFO:tensorflow:global_step/sec: 1362.08
INFO:tensorflow:loss = 0.5914903, step = 9100 (0.073 sec)
INFO:tensorflow:global_step/sec: 1156.43
INFO:tensorflow:loss = 0.48358634, step = 9200 (0.087 sec)
INFO:tensorflow:global_step/sec: 1055.69
INFO:tensorflow:loss = 0.44032001, step = 9300 (0.095 sec)
INFO:tensorflow:global_step/sec: 1160.98
INFO:tensorflow:loss = 0.7849304, step = 9400 (0.085 sec)
INFO:tensorflow:global_step/sec: 1355.86
INFO:tensorflow:loss = 0.5005672, step = 9500 (0.074 sec)
INFO:tensorflow:global_step/sec: 1418.72
INFO:tensorflow:loss = 0.7860344, step = 9600 (0.070 sec)
INFO:tensorflow:global_step/sec: 1359.51
INFO:tensorflow:loss = 0.3466546, step = 9700 (0.074 sec)
INFO:tensorflow:global_step/sec: 1425.23
INFO:tensorflow:loss = 0.5313357, step = 9800 (0.070 sec)
INFO:tensorflow:global_step/sec: 1448.41
INFO:tensorflow:loss = 0.54238474, step = 9900 (0.069 sec)
INFO:tensorflow:global_step/sec: 1454.93
INFO:tensorflow:loss = 0.5991773, step = 10000 (0.069 sec)
INFO:tensorflow:global_step/sec: 1411.59
INFO:tensorflow:loss = 0.5747124, step = 10100 (0.071 sec)
INFO:tensorflow:global_step/sec: 1321.02
INFO:tensorflow:loss = 0.6004567, step = 10200 (0.076 sec)
INFO:tensorflow:global_step/sec: 1457.2
INFO:tensorflow:loss = 0.40501237, step = 10300 (0.069 sec)
INFO:tensorflow:global_step/sec: 1452.24
INFO:tensorflow:loss = 0.5759165, step = 10400 (0.069 sec)
INFO:tensorflow:global_step/sec: 1445.07
INFO:tensorflow:loss = 0.4586322, step = 10500 (0.069 sec)
INFO:tensorflow:global_step/sec: 1471.58
INFO:tensorflow:loss = 0.3866516, step = 10600 (0.068 sec)
INFO:tensorflow:global_step/sec: 1451.53
INFO:tensorflow:loss = 0.44326347, step = 10700 (0.069 sec)
```

```
INFO:tensorflow:global_step/sec: 1458.47
INFO:tensorflow:loss = 0.6011107, step = 10800 (0.069 sec)
INFO:tensorflow:global_step/sec: 1453.79
INFO:tensorflow:loss = 0.55156976, step = 10900 (0.069 sec)
INFO:tensorflow:global_step/sec: 1348.67
INFO:tensorflow:loss = 0.4153244, step = 11000 (0.074 sec)
INFO:tensorflow:global_step/sec: 1270.58
INFO:tensorflow:loss = 0.7866856, step = 11100 (0.079 sec)
INFO:tensorflow:global_step/sec: 1142.01
INFO:tensorflow:loss = 0.3286642, step = 11200 (0.088 sec)
INFO:tensorflow:global_step/sec: 876.269
INFO:tensorflow:loss = 0.4195396, step = 11300 (0.115 sec)
INFO:tensorflow:global_step/sec: 1024.36
INFO:tensorflow:loss = 0.4847543, step = 11400 (0.096 sec)
INFO:tensorflow:global_step/sec: 1046.89
INFO:tensorflow:loss = 0.39103016, step = 11500 (0.096 sec)
INFO:tensorflow:global_step/sec: 1134.37
INFO:tensorflow:loss = 0.4965997, step = 11600 (0.088 sec)
INFO:tensorflow:global_step/sec: 1174.69
INFO:tensorflow:loss = 0.50906664, step = 11700 (0.085 sec)
INFO:tensorflow:global_step/sec: 1273.43
INFO:tensorflow:loss = 0.3639187, step = 11800 (0.079 sec)
INFO:tensorflow:global_step/sec: 1297.29
INFO:tensorflow:loss = 0.5398876, step = 11900 (0.077 sec)
INFO:tensorflow:global_step/sec: 1319.31
INFO:tensorflow:loss = 0.4078012, step = 12000 (0.076 sec)
INFO:tensorflow:global_step/sec: 1338.25
INFO:tensorflow:loss = 0.5520755, step = 12100 (0.075 sec)
INFO:tensorflow:global_step/sec: 1169.49
INFO:tensorflow:loss = 0.4823429, step = 12200 (0.086 sec)
INFO:tensorflow:global_step/sec: 1125.55
INFO:tensorflow:loss = 0.5240508, step = 12300 (0.088 sec)
INFO:tensorflow:global_step/sec: 1149.34
INFO:tensorflow:loss = 0.673871, step = 12400 (0.087 sec)
INFO:tensorflow:global_step/sec: 1282.31
INFO:tensorflow:loss = 0.3692883, step = 12500 (0.078 sec)
INFO:tensorflow:global_step/sec: 1381.05
INFO:tensorflow:loss = 0.72282714, step = 12600 (0.072 sec)
INFO:tensorflow:global_step/sec: 1403.94
INFO:tensorflow:loss = 0.54437006, step = 12700 (0.071 sec)
INFO:tensorflow:global_step/sec: 1417.11
INFO:tensorflow:loss = 0.3657521, step = 12800 (0.071 sec)
INFO:tensorflow:global_step/sec: 1434.7
INFO:tensorflow:loss = 0.67300516, step = 12900 (0.070 sec)
INFO:tensorflow:global_step/sec: 1294.97
INFO:tensorflow:loss = 0.37097654, step = 13000 (0.077 sec)
INFO:tensorflow:global_step/sec: 1344.25
INFO:tensorflow:loss = 0.4266548, step = 13100 (0.074 sec)
```

```
INFO:tensorflow:global_step/sec: 1387.46
INFO:tensorflow:loss = 0.6192177, step = 13200 (0.072 sec)
INFO:tensorflow:global_step/sec: 1412.88
INFO:tensorflow:loss = 0.57256794, step = 13300 (0.071 sec)
INFO:tensorflow:global_step/sec: 1339.84
INFO:tensorflow:loss = 0.5990072, step = 13400 (0.075 sec)
INFO:tensorflow:global_step/sec: 1062.85
INFO:tensorflow:loss = 0.38251373, step = 13500 (0.094 sec)
INFO:tensorflow:global_step/sec: 1125.67
INFO:tensorflow:loss = 0.48259208, step = 13600 (0.089 sec)
INFO:tensorflow:global_step/sec: 1340.4
INFO:tensorflow:loss = 0.710937, step = 13700 (0.075 sec)
INFO:tensorflow:global_step/sec: 1426.19
INFO:tensorflow:loss = 0.4152833, step = 13800 (0.070 sec)
INFO:tensorflow:global_step/sec: 1354.68
INFO:tensorflow:loss = 0.38348728, step = 13900 (0.074 sec)
INFO:tensorflow:global_step/sec: 1409.9
INFO:tensorflow:loss = 0.34869397, step = 14000 (0.071 sec)
INFO:tensorflow:global_step/sec: 1489.05
INFO:tensorflow:loss = 0.5180343, step = 14100 (0.067 sec)
INFO:tensorflow:global_step/sec: 916.776
INFO:tensorflow:loss = 0.5000782, step = 14200 (0.109 sec)
INFO:tensorflow:global_step/sec: 1029.52
INFO:tensorflow:loss = 0.35638642, step = 14300 (0.097 sec)
INFO:tensorflow:global_step/sec: 1198.62
INFO:tensorflow:loss = 0.5720459, step = 14400 (0.083 sec)
INFO:tensorflow:global_step/sec: 1273.74
INFO:tensorflow:loss = 0.5627103, step = 14500 (0.079 sec)
INFO:tensorflow:global_step/sec: 1309.46
INFO:tensorflow:loss = 0.4588607, step = 14600 (0.076 sec)
INFO:tensorflow:global_step/sec: 1298.13
INFO:tensorflow:loss = 0.50618684, step = 14700 (0.077 sec)
INFO:tensorflow:global_step/sec: 1335.12
INFO:tensorflow:loss = 0.49894485, step = 14800 (0.075 sec)
INFO:tensorflow:global_step/sec: 1037.36
INFO:tensorflow:loss = 0.54750115, step = 14900 (0.096 sec)
INFO:tensorflow:global_step/sec: 1054.21
INFO:tensorflow:loss = 0.48693535, step = 15000 (0.095 sec)
INFO:tensorflow:global_step/sec: 1217.14
INFO:tensorflow:loss = 0.49131447, step = 15100 (0.082 sec)
INFO:tensorflow:global_step/sec: 1242.53
INFO:tensorflow:loss = 0.46073857, step = 15200 (0.080 sec)
INFO:tensorflow:global_step/sec: 1160.46
INFO:tensorflow:loss = 0.60986245, step = 15300 (0.086 sec)
INFO:tensorflow:global_step/sec: 1161.98
INFO:tensorflow:loss = 0.45043084, step = 15400 (0.086 sec)
INFO:tensorflow:global_step/sec: 1173.43
INFO:tensorflow:loss = 0.40055215, step = 15500 (0.085 sec)
```

```
INFO:tensorflow:global_step/sec: 1288.56
INFO:tensorflow:loss = 0.3849408, step = 15600 (0.078 sec)
INFO:tensorflow:global_step/sec: 1327.76
INFO:tensorflow:loss = 0.4701432, step = 15700 (0.075 sec)
INFO:tensorflow:global_step/sec: 1272.94
INFO:tensorflow:loss = 0.28493366, step = 15800 (0.079 sec)
INFO:tensorflow:global_step/sec: 1196.86
INFO:tensorflow:loss = 0.40219772, step = 15900 (0.084 sec)
INFO:tensorflow:global_step/sec: 1193.8
INFO:tensorflow:loss = 0.4141207, step = 16000 (0.084 sec)
INFO:tensorflow:global_step/sec: 1234.45
INFO:tensorflow:loss = 0.4889355, step = 16100 (0.081 sec)
INFO:tensorflow:global_step/sec: 1230.6
INFO:tensorflow:loss = 0.34818667, step = 16200 (0.081 sec)
INFO:tensorflow:global_step/sec: 1240.14
INFO:tensorflow:loss = 0.3806489, step = 16300 (0.081 sec)
INFO:tensorflow:global_step/sec: 1293.66
INFO:tensorflow:loss = 0.5710654, step = 16400 (0.077 sec)
INFO:tensorflow:global_step/sec: 1784.28
INFO:tensorflow:loss = 0.45997825, step = 16500 (0.056 sec)
INFO:tensorflow:Calling checkpoint listeners before saving checkpoint 16512…
INFO:tensorflow:Saving checkpoints for 16512 into
/var/folders/q7/wrxzkb515gqcskvhd38dwx6h0000gn/T/tmpype0sfb1/model.ckpt.
INFO:tensorflow:Calling checkpoint listeners after saving checkpoint 16512…
INFO:tensorflow:Loss for final step: 0.5940797.
```

[ ]:
```
<tensorflow_estimator.python.estimator.canned.linear.LinearRegressorV2 at
0x7fc169517b50>
```

We can see in the above output how TensorFlow's LinearRegressor will tell us, while it's training, what the loss is as the model improves. This output can be useful when, later on, we'll tweak the learning rate.

1.3.7 Testing Input Function

In order to evaluate the quality of our model, we need to make predictions and see how close they are to reality. To do this we rely on the `predict()` method.

Similar to `train`, this method expects an input function. We'll create one similar to the one we created for train, only we won't repeat or shuffle the data and will process the data in batches of 1.

Exercise 3: Create a Testing Input Function Create a testing input function called `testing_input`. The function should accept no arguments and should return a `Dataset`. The `Dataset` should not repeat, nor shuffle, and should have batches of size 1. Also, target/label values aren't needed for testing input.

[ ]:
```
def testing_input():
  ds = tf.data.Dataset.from_tensor_slices((
    {c: testing_df[c] for c in feature_columns},  # feature map
```

```
    testing_df[target_column]                    # labels
  ))
  ds = ds.batch(1)
  return ds
```

1.3.8 Make Predictions

Now we need to make predictions using our test features. To do that we pass our testing input function to the `predict` method on our trained linear regressor.

[ ]:

```

```

[ ]:

```

```

[ ]:

```
predictions = linear_regressor.predict(input_fn=testing_input)
```

That runs pretty fast… almost suspiciously fast. The reason is that the model isn't actually making predictions at this point. We have just built the graph to make predictions. TensorFlow is using lazy execution. The predictions won't be made until we ask for them.

Let's go ahead and get the predictions and put them in a NumPy array so that we can calculate our error.

[ ]:

```
predicted_median_values = [item['predictions'][0] for item in predictions]
print("Our predictions: ", predicted_median_values)
```

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from
/var/folders/q7/wrxzkb515gqcskvhd38dwx6h0000gn/T/tmpype0sfb1/model.ckpt-16512
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
Our predictions:  [2.896512, 3.3325655, 3.309964, 2.4740586, 2.2142208,
2.8938513, 1.0999267, 2.090536, 1.7355188, 2.508802, 1.8608006, 0.7614353,
1.4228094, 3.1078875, 0.91481197, 2.050161, 1.9105798, 2.4933681, 1.9505873,
2.3273716, 2.6405427, 3.2925398, 2.373021, 0.9260584, 1.559162, 1.7040212,
1.6611798, 2.3783598, 1.2895219, 1.0449965, 2.784998, 0.8649989, 1.7460887,
2.341486, 0.69396806, 1.150087, 2.7967327, 4.415779, 3.2405474, 2.0205498,
2.0642443, 2.20498, 2.7791386, 2.4565072, 2.8610094, 1.838804, 1.3816988,
3.0028663, 1.1213497, 1.763861, 1.8206351, 2.619825, 3.747479, 1.1529521,
0.5179055, 1.389236, 4.13529, 2.4800653, 2.0839252, 2.0262852, 1.4030399,
2.2795765, 2.400907, 1.5761889, 2.6429358, 1.4200342, 2.4097962, 1.3623465,
2.8140826, 0.87751925, 2.7696872, 0.89182866, 2.1312227, 2.4762988, 1.5795969,
0.9743769, 1.8898935, 1.1757333, 0.57357097, 1.3878936, 2.7118578, 2.7432985,
1.9505401, 2.530203, 2.6601365, 1.6235964, 1.7982743, 2.5281672, 2.4651089,
0.7785151, 0.63051516, 1.234196, 0.5408995, 3.3116865, 1.6546823, 3.256574,
2.4835248, 2.0618808, 0.782117, 2.9552665, 0.9548508, 2.9272122, 1.4890069,
```

2.2107406, 0.7950691, 1.8864036, 1.1078552, 2.469498, 1.726963, 1.5412121,
1.5121514, 0.71565896, 1.714715, 1.517605, 2.1407404, 1.7428815, 5.361265,
1.7863536, 1.8858626, 2.0479262, 1.2987726, 3.117819, 1.6721355, 2.3468242,
2.195273, 2.4429183, 1.282897, 3.0128508, 2.473184, 5.9726954, 2.7820387,
0.9346142, 3.5201678, 2.347354, 1.1093837, 2.4648576, 1.8567175, 1.6842945,
2.319633, 2.7399435, 2.5752676, 2.1240516, 2.823837, 0.9820926, 1.8786812,
2.7071924, 2.559392, 3.6549103, 2.8006907, 1.1899062, 1.9160659, 0.83543396,
1.0191948, 2.7677307, 1.3549895, 1.1649623, 1.3758836, 2.8953528, 2.2342017,
1.8761628, 1.6065294, 3.2622151, 3.8725784, 1.6046747, 1.1639031, 1.7824634,
1.7545576, 2.6336122, 1.3673022, 3.2954926, 1.9040186, 2.649672, 2.3523545,
2.1458251, 2.649656, 1.6978946, 2.6611795, 2.5959592, 3.785234, 2.2715044,
2.5092583, 2.1075768, 2.1837654, 1.1629965, 2.2492056, 3.0659456, 2.547949,
1.4971324, 1.5288295, 3.8986275, 3.5899055, 2.3486936, 2.345014, 5.048996,
0.7747197, 2.784348, 2.728683, 2.7887952, 1.4302082, 1.8616408, 0.85133684,
0.975052, 1.38562, 0.8318896, 2.443771, 1.5613823, 1.8826392, 0.83218414,
0.839254, 2.375174, 0.97491175, 1.0656478, 3.626924, 2.4811337, 2.2816143,
2.799902, 1.2929957, 2.4650433, 2.6827266, 3.0429678, 3.0561454, 1.9363189,
1.7235267, 2.6975927, 1.4287992, 2.9856064, 1.5318028, 1.9412607, 1.298657,
1.9830046, 2.4732485, 1.6148233, 2.477652, 0.96169764, 2.0054507, 1.5756257,
2.282258, 3.0158534, 1.9859551, 2.3838677, 2.2197213, 3.1086652, 1.7371624,
0.6587391, 2.4645014, 1.5689316, 1.2810998, 1.7450137, 1.5598786, 1.7041936,
1.776999, 1.988729, 0.7954665, 2.5206232, 1.0397149, 2.4852571, 1.3778837,
4.776605, 2.8984761, 1.462414, 2.2855942, 1.7196918, 2.0499377, 1.2949042,
1.6387756, 1.236395, 2.1137466, 2.2595503, 1.4301482, 2.1334095, 2.3737187,
1.5978281, 2.829174, 2.148003, 1.5225084, 2.4146762, 1.5562321, 2.4569254,
2.6651917, 2.6383686, 1.6425743, 1.6869962, 2.271993, 2.5411859, 2.0540175,
2.099074, 1.3335758, 1.3390813, 2.4485364, 1.6120108, 1.4927925, 2.178256,
1.235564, 2.3921685, 3.8952067, 0.7449653, 2.089885, 4.4305153, 2.4007528,
2.7700365, 3.4877656, 1.8653854, 1.9322984, 1.0335555, 1.6845539, 2.507427,
2.2942352, 2.4254944, 2.706634, 1.5496252, 1.9740014, 0.98938155, 2.055882,
1.8118538, 2.2479978, 2.411798, 0.89561844, 2.7758706, 1.8146086, 1.8992366,
6.718791, 2.127987, 2.137591, 6.870056, 2.5824952, 2.3681967, 1.1396072,
1.04632, 0.6572284, 1.0500901, 2.3865042, 2.5182652, 2.3383741, 2.4952252,
2.9357872, 1.627516, 2.5165148, 1.826283, 0.9886942, 0.8040603, 2.8103037,
1.3669422, 2.8024247, 1.5064883, 2.2828283, 1.8612802, 2.6736965, 0.58843136,
1.7018871, 3.286809, 0.47697008, 3.0409386, 4.5323706, 1.554832, 0.99182284,
1.8610396, 3.2391455, 1.5113395, 2.0185275, 1.6784118, 2.2310839, 2.2018194,
2.3024993, 2.605579, 1.3396335, 1.4623916, 2.9305577, 1.494951, 1.8357987,
3.0539258, 1.6449229, 2.127534, 1.3354928, 2.8999684, 0.89592206, 2.153425,
0.89314914, 2.786346, 2.5750308, 3.3853252, 2.406549, 2.2236905, 0.6482719,
2.2238975, 1.5603226, 2.7355847, 4.3178663, 1.7674489, 3.327584, 0.91922927,
1.6995273, 0.8210427, 2.3116496, 2.2762995, 1.5544832, 1.014577, 0.6883904,
2.971593, 2.5666056, 1.7879248, 1.0672777, 2.6831353, 2.6424913, 3.0350227,
0.9487352, 2.2594686, 2.007494, 6.6658974, 0.44391048, 1.9687113, 2.0158622,
1.642055, 1.8151073, 2.1762533, 2.5065725, 1.5318079, 2.965499, 1.6519194,
1.6210154, 2.1462493, 0.6102963, 2.5200686, 1.0303481, 3.2755935, 2.952724,
1.2814134, 1.699172, 1.0663891, 2.4594202, 2.0989408, 3.1496544, 2.6441038,
5.293066, 1.8624582, 1.2578568, 3.9914904, 1.6177793, 2.6783013, 2.2709877,

3.509928, 2.4379382, 2.4862003, 2.2584121, 1.1176786, 1.9904475, 1.1923616,
1.8800105, 1.1817113, 2.6876774, 2.8036675, 0.76245284, 1.8506093, 1.6395359,
1.4329642, 2.3981142, 2.481264, 3.1492817, 1.5626264, 2.2495832, 1.6336296,
2.124456, 2.7568207, 0.41903448, 3.5964751, 1.0367109, 1.6132214, 1.929229,
3.6728485, 1.3310032, 1.8744258, 1.4809196, 2.2395124, 2.4952435, 1.3112272,
1.3204955, 2.6186423, 1.8181642, 1.9172316, 1.4633765, 1.4294232, 2.5884786,
1.8195698, 2.5842633, 1.4294553, 1.1339571, 3.361314, 3.4822447, 1.6420603,
2.3697436, 1.8616841, 2.431652, 3.0167665, 1.586909, 1.7270322, 2.31159,
1.538653, 2.2757921, 1.4001245, 1.454082, 1.8599923, 3.4644263, 2.6771555,
1.4318151, 1.2943665, 1.8750159, 1.8830496, 1.617635, 2.8702626, 1.9125407,
0.67406845, 3.2475183, 2.3100605, 1.0739127, 2.6456695, 2.4072423, 1.9855287,
2.2044609, 1.6984866, 0.9312652, 3.5407312, 1.5129151, 0.7393677, 3.989917,
1.8446336, 1.7361407, 2.5703979, 1.4782472, 2.8410466, 4.2460313, 2.4709258,
2.7090707, 2.8215406, 2.1917257, 2.6019106, 3.160894, 1.5861896, 1.6169796,
1.9528967, 1.5792389, 2.5045285, 6.670128, 4.2641983, 1.3970568, 0.939481,
4.867298, 0.70867926, 2.2537699, 2.4717178, 1.6791637, 2.8869472, 2.736115,
1.6069096, 0.4496168, 2.0904121, 2.018642, 2.6726096, 1.7824652, 2.6191425,
4.3782873, 2.1919823, 3.9517968, 0.20980406, 2.2715888, 2.6618485, 0.5697324,
2.7386818, 2.4408333, 1.8659072, 1.980155, 2.403416, 2.409948, 1.1254065,
0.82492995, 1.763409, 1.7531756, 1.6871066, 2.5989194, 2.3597784, 2.626741,
1.8244326, 2.2237391, 1.6976995, 2.7383065, 1.8482833, 2.8755026, 2.4006062,
2.418446, 1.9701848, 1.0233774, 1.165831, 1.0707803, 2.8065183, 1.8438137,
1.9671012, 2.22612, 1.5705144, 3.6161547, 3.0050774, 2.3533688, 5.526408,
1.6937467, 1.1330615, 1.4150782, 1.0720854, 1.1977451, 0.9552423, 2.9097407,
1.8736349, 1.8530712, 2.360518, 1.6284287, 2.1277723, 2.357018, 1.8683332,
1.9554403, 1.517878, 2.4713936, 1.6979649, 0.5793518, 2.044029, 0.4403038,
2.1218202, 3.0094967, 2.8549867, 2.48344, 1.1277242, 2.4158597, 1.4306686,
1.6078568, 2.1066399, 1.676784, 1.2822846, 0.96503514, 2.0271554, 6.0579205,
1.914808, 1.8263173, 1.2194395, 2.7138433, 2.6136942, 2.81673, 0.7472911,
0.9816768, 1.0869788, 1.8425331, 1.4417, 2.05913, 2.895083, 2.6272135,
1.4534732, 1.7063146, 2.2038646, 2.604342, 2.8224604, 2.139461, 1.2897764,
1.2849404, 1.5201712, 2.3816056, 2.000392, 2.9516945, 2.0381112, 1.4029797,
1.5933143, 2.7913582, 3.5567951, 1.9887247, 0.93668544, 3.4350548, 2.7171965,
1.4858198, 3.412129, 0.5093751, 2.4330244, 1.3536663, 1.5072002, 1.3288527,
1.5588629, 1.0985609, 0.93807715, 2.5113192, 1.3399507, 1.6229663, 2.745173,
3.4541328, 2.3914247, 1.6130195, 0.43644315, 1.3133183, 0.68178904, 1.8265042,
1.4460938, 0.5792377, 1.3056867, 2.8885293, 0.84753525, 2.410371, 1.8920412,
2.2598746, 0.84132314, 3.1809335, 1.3425418, 1.0900505, 2.451007, 3.0013452,
1.4620857, 0.85770845, 1.1310077, 2.5519047, 1.4699708, 1.9024038, 1.8095505,
0.92724323, 2.9034226, 2.4385965, 2.8940415, 1.9056432, 2.336903, 1.6648595,
1.5057093, 1.3927441, 2.535252, 1.2218517, 1.7036041, 1.6962008, 0.9638429,
2.255466, 2.7262688, 4.2861633, 1.9809453, 1.9254951, 1.5068114, 3.0921519,
2.3544874, 2.2857363, 0.25360644, 3.9237497, 2.8846278, 1.1288913, 0.82381755,
1.6508825, 1.6451564, 1.728884, 2.8955164, 1.4633194, 4.5359297, 0.48373747,
0.38420993, 2.8442855, 1.248435, 1.0524869, 3.122943, 2.7084527, 2.177258,
0.9047394, 2.524553, 2.5888536, 2.2268548, 1.6594948, 1.7570133, 3.2999375,
2.1839342, 2.3744354, 1.9735129, 1.1370451, 1.4500214, 1.9088607, 1.8664,
3.8333087, 2.6554189, 0.81337845, 1.9562948, 2.603468, 1.5296221, 0.49401742,

1.6634316, 1.1401317, 0.73563933, 2.7685928, 2.8860304, 1.4306656, 1.3371074,
2.6402872, 2.7732148, 1.1510748, 2.888455, 2.2592168, 1.0130606, 0.8717824,
3.8271134, 0.90666735, 1.3465507, 1.5784678, 2.659433, 0.6480954, 1.3945215,
1.5930548, 2.3718352, 1.9514394, 2.42009, 1.4393497, 2.7266786, 2.2550778,
2.4362817, 2.3694925, 2.010367, 1.3510773, 1.0760643, 1.6959338, 4.0614414,
1.5893154, 0.89259166, 0.6160337, 2.4277306, 2.3511715, 0.6414274, 4.6857996,
1.8569332, 2.2903867, 2.7430744, 1.6509051, 0.59807134, 2.438754, 1.6163268,
2.3942323, 2.044985, 2.4183211, 0.7523644, 2.6131005, 1.2044365, 1.7734714,
1.7828305, 2.76116, 0.9573047, 0.586252, 3.405127, 3.599685, 2.5464902,
3.313723, 1.8020289, 1.1480289, 2.3061836, 2.8711736, 1.584592, 0.9138008,
2.7571769, 1.4299599, 2.5603871, 2.9889355, 0.8234936, 1.4152877, 0.7839614,
1.4558201, 2.2622569, 5.114889, 1.5142148, 2.6464489, 2.0020812, 2.1113515,
2.2399225, 1.229853, 0.99349296, 2.7259398, 3.027663, 5.8293104, 1.8344047,
1.3267641, 1.6902583, 1.0242712, 1.7793012, 2.5610495, 1.1197063, 1.1789693,
6.7730646, 1.320455, 2.90943, 2.2343118, 1.6910735, 1.922327, 2.3894386,
1.8240981, 0.40054476, 1.7707238, 2.5995915, 2.8761668, 2.9510524, 5.5919485,
1.9595368, 2.5522282, 2.1503944, 1.1829009, 3.0622134, 1.5107017, 3.6188276,
2.0827923, 4.635482, 2.8542256, 2.872918, 0.9987569, 0.47910124, 0.7883281,
2.5263088, 1.5795326, 2.2472823, 4.4170685, 2.1368093, 3.4209082, 2.014951,
2.3017507, 1.8193319, 1.2649206, 1.9184887, 0.45353842, 1.991543, 2.0188508,
2.6254559, 2.0351038, 2.0751464, 1.7407779, 0.9679135, 2.412356, 2.787763,
1.9915718, 2.6624868, 0.54867554, 1.1510291, 2.3683786, 0.7681186, 2.4670796,
2.1866062, 0.81467724, 2.938914, 2.3023102, 2.7219744, 3.6275628, 3.2669282,
0.9581718, 3.3465629, 2.8902528, 3.4431381, 3.3880897, 1.9335985, 2.4415445,
1.684805, 2.748685, 0.54018044, 2.645393, 2.405342, 2.5160518, 1.15622,
1.1550211, 2.2948134, 1.1772811, 2.174098, 2.6524506, 2.47678, 3.9944422,
1.1421834, 2.6609333, 1.9490478, 0.5981657, 2.1000779, 4.40601, 2.0300355,
3.3671796, 2.403114, 3.9703405, 2.6637924, 2.076107, 2.0325348, 2.0701175,
1.5617759, 0.8910545, 2.2351394, 1.3409543, 0.9723745, 1.9912459, 1.1199785,
1.4644457, 1.8934546, 1.3845847, 2.4023883, 1.5331347, 0.5511035, 1.0471177,
1.0638031, 2.873239, 1.2918549, 1.1217546, 1.039489, 2.5602899, 1.0089213,
2.166029, 2.5150523, 1.470108, 2.360962, 2.1614032, 2.8286846, 2.1704154,
3.644534, 2.9087598, 0.9259283, 1.1330097, 0.6275435, 2.0186057, 2.7427392,
0.90630233, 4.6091275, 1.0289731, 1.2836039, 1.7418766, 1.7448803, 4.7891197,
1.582015, 2.1797707, 1.1253933, 1.070622, 2.4748492, 3.049927, 3.12573,
2.035954, 2.9014618, 2.8273802, 1.5409677, 2.2497272, 2.472249, 0.6417655,
1.9187276, 2.600859, 1.3483204, 0.85885125, 2.3956008, 3.9261112, 1.582956,
2.7280931, 0.97793746, 2.1341932, 2.0840402, 1.6414461, 3.0517333, 1.2438477,
1.148869, 2.5273843, 2.1447926, 2.104634, 1.143863, 1.2620372, 1.9586933,
2.355991, 2.7074382, 2.3646326, 1.941467, 1.163233, 2.2522812, 1.8334495,
2.4979181, 3.3641236, 3.0995247, 2.4813259, 1.6458206, 2.0978532, 2.2208276,
2.0069466, 1.4584428, 2.2545047, 1.8209689, 4.817853, 1.3197646, 1.0605977,
2.386864, 1.9758928, 2.47056, 2.6576762, 2.1929073, 2.8115997, 2.5809422,
2.743846, 1.8319142, 3.6294081, 0.85938686, 1.8016486, 3.772367, 2.520909,
2.3645692, 2.883154, 2.3235044, 1.0169839, 1.5859104, 1.5191619, 1.7043394,
3.3916464, 1.1051184, 1.6506271, 2.0070271, 1.2898674, 1.0507762, 2.0091348,
2.831086, 2.5023313, 1.6079619, 1.3400433, 1.6342165, 2.8613014, 1.351974,
1.1471726, 1.6304162, 1.3086473, 1.3661487, 1.7570156, 0.55729055, 2.4265823,

4.880514, 0.46872312, 1.4040631, 1.6003125, 1.2104018, 3.382064, 1.4589688,
1.7015874, 1.7673566, 0.6008439, 1.0926309, 0.7367976, 0.88280606, 0.41440356,
0.7566402, 1.4337294, 1.442229, 2.6252139, 2.402481, 1.9748937, 2.146605,
2.4228582, 2.0409555, 2.4472451, 2.4578712, 0.66692287, 2.0312552, 2.2142467,
3.3321297, 1.6746714, 3.0206728, 2.89814, 1.7092143, 3.3533988, 1.0454597,
2.918531, 1.7698662, 3.2306502, 1.806029, 2.0471008, 1.8614297, 1.2806649,
2.011882, 2.2197034, 2.6618228, 3.2579284, 2.3979063, 0.8260542, 1.916585,
2.147027, 1.9842052, 1.51666, 4.2472363, 2.059669, 1.0736926, 0.35854328,
2.221436, 2.5854921, 4.358782, 2.2760925, 2.0117826, 1.7934787, 2.4398012,
2.8702016, 1.4960959, 3.7615426, 1.3822141, 1.3710481, 1.5995927, 1.5269194,
3.4916873, 5.3507166, 1.3735179, 2.642768, 3.09366, 1.4749324, 2.525679,
0.62898326, 0.82411635, 3.049267, 1.4125881, 2.796269, 1.6576447, 2.9002147,
0.47598815, 1.3190418, 2.770008, 2.2782502, 2.28408, 3.1164129, 2.4260855,
0.9782043, 2.42672, 2.2649446, 2.842931, 1.8008964, 2.0224361, 2.0103087,
2.8390226, 1.4800322, 1.2393361, 3.6927319, 1.7167764, 1.9999044, 2.018785,
1.7202957, 1.0821894, 2.8733625, 2.2412066, 1.9699224, 3.194168, 1.623651,
1.1647347, 1.4546504, 3.7768316, 2.2595131, 2.9102645, 2.5036843, 2.464744,
2.4584768, 3.9796882, 2.37092, 2.3065414, 2.3051481, 2.9100664, 1.7647076,
0.9826329, 1.2031425, 1.7054762, 2.7492886, 1.0173182, 1.0597479, 2.13995,
1.3058741, 2.2520757, 2.060086, 2.0034385, 1.1232009, 0.49139, 3.412614,
2.6550121, 1.0506968, 2.1380477, 1.9886627, 2.6925845, 2.7843196, 3.752195,
1.9685942, 1.5855672, 1.4737588, 1.7980318, 1.580534, 0.7849831, 1.6243986,
1.5953584, 2.004249, 1.5815628, 5.6422405, 2.2665749, 2.2585964, 2.4142976,
2.4095469, 1.7556698, 2.0825205, 3.3545141, 2.071703, 1.6513157, 2.7828596,
2.7269993, 2.4371192, 1.1356548, 1.9789515, 1.8323009, 0.87476647, 4.1985593,
2.1556585, 1.4691386, 2.6593919, 0.75999904, 2.0802286, 1.1000552, 1.7411169,
2.7481403, 0.7773999, 1.5574327, 3.3781886, 2.8865776, 1.1580824, 0.93147576,
2.7813058, 0.8116467, 2.0076854, 1.9171699, 0.78323305, 3.380915, 1.0792108,
3.1485486, 1.5827374, 1.6012075, 3.0878317, 2.1463182, 2.125589, 0.62091213,
2.9142358, 1.1874002, 1.6162047, 1.6484337, 2.6277387, 0.6047312, 0.6391282,
2.3638077, 1.5050309, 1.5753982, 3.1809595, 2.6491988, 2.4207802, 1.7100861,
1.1996204, 0.60960585, 2.328527, 1.2951459, 1.9588752, 2.5943723, 0.85729,
1.3596172, 1.4062994, 2.2619376, 0.72731847, 2.7046316, 1.4703753, 3.1267803,
1.6822774, 2.4779482, 2.6087337, 2.924409, 3.3369052, 2.2546525, 1.2911118,
2.4084072, 6.7078714, 2.9304066, 0.81484926, 2.026476, 1.5120194, 1.0241194,
2.161078, 0.9842403, 2.2754383, 2.4736042, 2.5815635, 0.8268225, 2.6323195,
2.369474, 2.0292547, 0.6940607, 3.3645492, 2.1464503, 1.2710681, 2.0159016,
1.7038896, 2.342019, 1.0622025, 1.7348734, 2.3374527, 2.081553, 0.9558448,
0.90388656, 2.8164845, 1.375827, 0.6481036, 0.32657093, 2.4849586, 0.88002455,
0.67669016, 1.1611958, 1.5734022, 2.3420804, 2.1576777, 0.7195318, 1.8203062,
1.6774266, 1.051692, 0.6155844, 1.4731622, 1.2335373, 3.7914457, 3.0055583,
1.6174963, 1.6013229, 2.6405869, 1.0967876, 1.4896684, 1.5748972, 2.3462644,
0.91047925, 2.055101, 1.5337844, 2.8265913, 2.8896842, 2.3697562, 1.040745,
0.9391496, 1.3639435, 0.5011242, 0.576458, 2.626358, 2.8320482, 2.8409386,
1.8687714, 1.368417, 1.1586463, 2.8818817, 1.9644108, 1.7689829, 1.556054,
4.702467, 1.146006, 2.466498, 0.481597, 1.8762509, 2.4065943, 0.7444768,
0.92663246, 1.4486047, 0.73944795, 1.7483987, 1.7610786, 2.1398458, 2.4293094,
1.688655, 2.8890562, 1.2922013, 1.3612821, 0.6494509, 1.9611835, 1.9641598,

2.2036066, 1.2558366, 1.1976372, 1.3373177, 4.85528, 2.6043417, 2.278924,
2.0822787, 1.2585063, 1.0474653, 4.3479557, 2.6113222, 2.2843657, 0.9579579,
2.0725894, 2.4133263, 2.1847925, 2.484837, 1.3644314, 1.6021737, 2.257624,
1.9897399, 1.3682933, 2.4289236, 0.61890024, 2.4568744, 1.5870335, 2.1915221,
2.305719, 1.7224779, 0.8676187, 3.0024643, 1.5168607, 1.4284245, 2.7134967,
1.364095, 2.9261198, 1.6498761, 2.6271513, 1.8285463, 2.7427204, 1.7652434,
1.7813301, 2.3932133, 2.8058705, 3.1025085, 1.4107559, 1.3994249, 2.5593734,
0.8632613, 0.5647905, 2.2706165, 1.3381885, 2.4225721, 2.584838, 2.4418726,
2.3086762, 3.1861365, 2.821134, 1.1476822, 2.6364484, 1.4480121, 2.0128162,
1.1725279, 1.264775, 1.8756174, 3.235069, 3.0020478, 2.4340487, 1.5335273,
2.0085945, 1.3137784, 1.3211961, 2.6365824, 2.9789214, 3.414183, 0.576933,
1.4291584, 1.0104297, 1.4290622, 2.656171, 3.2383075, 2.6694126, 1.7872893,
2.9932775, 2.3222919, 1.472348, 1.3437042, 1.5317657, 3.097672, 0.478958,
1.0139697, 3.1439672, 0.5375943, 1.2782371, 2.4412808, 1.4680822, 2.7065363,
1.6281083, 0.99289894, 2.6512399, 3.4880767, 2.5412307, 2.708806, 1.7174702,
1.08036, 1.193082, 2.379986, 2.2287838, 3.7962246, 0.9087361, 0.70868903,
3.3649492, 2.4260912, 2.0305772, 2.347331, 0.8079791, 3.9368641, 1.0020082,
1.6572202, 1.6703259, 2.7357445, 4.193507, 2.3686504, 2.096467, 2.2443702,
1.8150841, 3.7778804, 1.5366707, 0.22832686, 2.012279, 1.563411, 2.574175,
2.062101, 2.5258493, 1.3536079, 2.670529, 2.0583, 0.8807314, 2.13209, 1.1229998,
2.8049026, 2.489529, 1.0277913, 2.5454457, 0.6756612, 1.7650576, 1.9749827,
0.72782636, 1.551143, 2.1246634, 1.9791434, 3.3017213, 1.683074, 3.0437891,
1.5231024, 2.1009407, 0.89638305, 2.4691117, 0.8864453, 2.0568748, 2.0079517,
1.4466629, 1.5404658, 0.85171175, 2.6288915, 1.6500885, 1.0593193, 3.329224,
1.8719254, 1.318688, 2.8306134, 2.9904861, 2.1574135, 1.3969617, 3.1188047,
2.3410182, 0.7133793, 0.66710275, 0.9657763, 1.6378006, 3.0722687, 1.998191,
2.523095, 2.1953812, 0.83460945, 2.4083166, 2.25901, 3.3069732, 0.36752576,
0.2608006, 2.1945395, 3.2409852, 2.0438204, 2.1652112, 2.9618263, 0.682053,
2.6933596, 3.2999725, 1.3241733, 0.9122885, 2.643673, 2.3919706, 2.0443425,
0.71666944, 1.1940098, 0.40284514, 2.7733822, 2.224287, 0.7498734, 1.9780962,
2.295229, 0.8736284, 2.799529, 0.9522314, 2.2380238, 3.49928, 1.9457972,
1.5124117, 2.9576159, 2.7889433, 2.3542898, 2.1876554, 2.9332833, 1.0637385,
1.5030689, 2.610937, 1.961911, 3.423083, 1.6653128, 2.2131476, 2.781402,
1.9962457, 2.7522132, 2.9905038, 2.4781466, 0.43986213, 1.4406526, 2.2592874,
0.7205286, 1.775897, 2.1632967, 3.7532887, 0.4759571, 3.4854276, 2.614252,
1.0129517, 0.78695863, 0.9833883, 2.8079963, 2.0267222, 2.06524, 1.4296167,
1.6064801, 1.520895, 1.5438443, 1.7198465, 2.9358358, 0.71597207, 2.7508588,
3.8481505, 1.6724, 1.0954535, 0.52067137, 1.9877801, 1.6316488, 1.3324213,
0.92475003, 2.5380144, 2.8599014, 1.4325069, 2.2428837, 1.8755815, 1.9187546,
2.3014731, 2.280623, 2.0094793, 2.146225, 1.8300536, 1.934084, 0.5107892,
3.435595, 1.678537, 1.3500313, 0.7691281, 1.9244146, 2.0160878, 1.1487073,
2.5668828, 2.2113347, 1.6779256, 0.7738047, 2.4914165, 2.7302694, 0.91201663,
1.0380982, 1.1749281, 2.641322, 0.8635923, 1.9884186, 3.1271665, 1.0264739,
2.449257, 3.644556, 2.205144, 2.4803514, 5.343422, 3.1822047, 2.6310816,
3.1280186, 1.3906449, 0.78734875, 1.5025119, 2.8240743, 3.1214864, 3.2612023,
4.094284, 0.79649013, 1.9415727, 2.8338485, 0.97090596, 2.1005683, 1.9525592,
1.1554143, 1.2915332, 2.3490741, 3.336094, 1.3636315, 1.5109138, 2.767167,
2.1176972, 1.087327, 2.0104585, 3.9338768, 1.6371924, 1.6864097, 2.4606657,

0.46373403, 2.1810904, 1.4813504, 1.2629918, 2.4070032, 1.4934728, 2.5816083,
2.1288395, 1.5188522, 1.44486, 1.7413421, 2.2265282, 2.3373735, 2.4652975,
2.0520642, 1.2133799, 2.6368015, 2.7331367, 1.453155, 1.2675555, 3.1145475,
2.7039242, 2.2597675, -0.014301419, 1.3993778, 1.8140185, 3.0993543, 0.7552196,
0.9151422, 1.3295795, 3.9367113, 1.0559164, 3.3426738, 2.7086196, 1.8041632,
0.758821, 1.8994029, 2.185491, 1.1625248, 1.14087, 1.9874421, 2.76943,
0.6115028, 2.8902109, 2.45874, 2.3812432, 0.88623124, 2.653558, 2.5120578,
1.3437719, 0.8652897, 2.0395412, 0.97202307, 2.3279777, 3.2263384, 3.2653143,
0.8601865, 0.9944474, 2.7739148, 2.3837738, 3.2996986, 2.7359066, 1.2402194,
1.5992293, 1.5072004, 2.3917997, 1.3760324, 2.924707, 2.9379597, 0.30510032,
3.6024249, 2.5507398, 1.722994, 1.1041694, 2.0347545, 2.0583067, 2.0696518,
2.5519938, 1.7679782, 2.6107438, 1.3893852, 1.4700834, 1.5673785, 1.6783497,
1.3460459, 2.0597675, 2.4756222, 2.219411, 2.0937095, 0.55144274, 3.0635698,
3.3867004, 3.0665262, 3.1180084, 2.1514783, 3.3253748, 2.025144, 2.4561372,
1.8070533, 2.0552852, 3.0611196, 2.3494558, 2.7989178, 4.0352077, 3.4527898,
3.2766294, 1.9965951, 1.1966988, 6.6081305, 1.3877968, 1.6315491, 2.661563,
4.105152, 2.0652065, 3.5282238, 2.6320229, 1.7546101, 1.1178652, 1.4600961,
1.578484, 1.7577434, 1.7000012, 2.5101936, 2.5005894, 1.4899888, 4.3117003,
1.0746894, 1.364111, 0.35613877, 0.85448724, 0.88139623, 0.9062738, 2.0961611,
5.439275, 2.3479676, 2.9117265, 1.6317203, 2.9128659, 2.8555856, 1.5798287,
1.7859604, 1.0393796, 1.2613463, 2.0184135, 2.262544, 2.269498, 1.8007252,
2.5564535, 1.9664364, 0.60671633, 1.438472, 1.3694307, 2.6086473, 1.839025,
1.1721349, 2.277553, 1.3997014, 2.7155766, 1.6153697, 1.3748392, 0.8685709,
1.1491688, 1.8549628, 2.4480677, 2.6083302, 2.6200323, 1.1593524, 2.1243773,
2.4631574, 2.5697856, 3.5623357, 1.1877744, 0.752248, 2.4520502, 0.55417454,
4.043964, 1.1847818, 2.913899, 2.516264, 1.3874487, 0.6393952, 1.1658101,
4.345667, 1.8277464, 2.7323194, 1.2289195, 0.76410586, 1.4940083, 2.2889009,
1.0213813, 1.8459374, 2.0746126, 2.089754, 2.5449674, 2.1063836, 1.2458751,
0.8910636, 2.324592, 1.2127675, 2.2527192, 1.8380702, 2.0707684, 1.5274923,
2.2726197, 2.345954, 1.700741, 4.318884, 1.4064933, 2.6482987, 1.4929123,
2.081058, 2.5449367, 2.0649972, 1.3147413, 1.7324052, 2.1578755, 1.4849429,
3.4260805, 2.8695996, 2.228584, 1.6677914, 2.1861105, 1.581066, 0.6173291,
0.6177168, 3.7654767, 1.4386824, 2.9122396, 1.1390268, 4.3922596, 3.5160193,
1.9364861, 2.045705, 1.5643075, 2.7783628, 1.5648267, 1.6355785, 2.1332545,
2.6593957, 2.2370281, 2.4382174, 1.4813554, 1.3390539, 3.2560778, 2.35149,
1.5606911, 1.367836, 1.2783208, 2.638973, 0.98133576, 3.4861474, 2.290164,
0.67620707, 2.1763258, 1.7341602, 1.3683851, 1.1174331, 2.0256526, 2.3253536,
3.8189185, 2.8063712, 1.8513762, 1.7800663, 3.7438579, 2.324212, 1.2294586,
2.3609447, 2.6747808, 1.5138068, 0.8108522, 2.6327815, 1.7732842, 2.1614053,
1.5514089, 1.7977321, 2.1247108, 2.987301, 1.578646, 3.2403812, 3.0534773,
2.6941118, 1.0892767, 1.2281721, 1.9808614, 1.8943024, 1.727948, 1.5096858,
3.806996, 1.387897, 1.8264027, 1.7569873, 3.2554288, 2.7039075, 2.1537824,
2.2649217, 2.2333708, 2.2187803, 1.6176672, 2.5666704, 3.4746852, 1.8619735,
1.8850807, 3.4559982, 1.6725385, 3.394926, 3.7087417, 0.74829507, 2.785851,
5.8294296, 2.6575968, 1.4635217, 2.15864, 0.28290087, 2.936801, 2.8420753,
6.6918793, 3.1676843, 2.5715122, 1.9872589, 1.5555309, 2.7338777, 1.095816,
1.6320031, 1.6647326, 3.3942544, 1.2383251, 1.5640824, 2.8331485, 3.1076381,
2.5948133, 1.1574435, 1.6897806, 2.0800805, 1.6008515, 1.5225654, 1.8677983,

2.3253927, 1.5360694, 2.0758345, 2.450784, 2.522284, 1.9080114, 0.8097133,
2.1607866, 1.0541934, 2.8848171, 0.32526124, 2.1642022, 2.9800014, 1.3456731,
0.48517382, 0.71767575, 1.6795462, 1.0114335, 2.0347743, 1.6403965, 1.8343391,
2.4790165, 0.3014536, 2.5104094, 1.7780656, 1.8512354, 3.4712505, 2.1666708,
2.1276321, 1.979356, 1.2850887, 1.4034244, 3.832577, 1.6355591, 0.70759857,
0.421444, 3.7890565, 0.56043077, 0.81359625, 1.7901728, 0.9529112, 1.0746223,
1.097248, 2.6638074, 1.0226324, 1.1548682, 1.5722079, 1.695138, 3.131634,
1.9417175, 1.71841, 2.065265, 2.1372128, 2.1627638, 1.9030623, 1.9886961,
3.3475473, 1.5564355, 1.5481548, 1.7627381, 2.3732972, 1.82493, 2.972047,
1.752173, 1.4202902, 2.8358445, 0.8722767, 1.786479, 1.6439661, 2.570117,
3.349989, 0.91567504, 1.6655605, 2.4026196, 1.4428992, 3.1593626, 2.4499426,
1.3575528, 2.2845178, 2.465837, 2.1692643, 0.34531683, 2.3246045, 1.6368383,
2.3795488, 1.4178452, 2.222399, 1.3124154, 2.2891655, 3.862389, 1.8196543,
2.7658343, 1.9837794, 5.414667, 2.6360283, 2.0390077, 1.4140685, 3.2929616,
2.437172, 2.2504802, 1.7106843, 1.4438262, 2.6866388, 1.5631294, 0.627192,
3.5736182, 3.621744, 2.4868047, 3.9934745, 2.0674186, 1.8249114, 1.5192215,
2.7991276, 0.7596554, 2.7070422, 2.184621, 1.7599188, 1.9756021, 2.179628,
1.9924802, 1.9524485, 1.6594828, 2.6218605, 0.5757929, 1.6565686, 2.4693785,
6.4978485, 3.120516, 1.3143555, 0.93479353, 0.8982362, 2.7630458, 2.3169374,
0.9945536, 2.4961557, 2.9968083, 2.2553566, 1.4288485, 2.675128, 1.4306483,
1.4789468, 3.017266, 0.68007004, 1.5801349, 2.8387396, 3.61815, 2.4937177,
1.072994, 2.8328657, 1.5482416, 2.2520053, 0.5896665, 2.9854226, 2.3108928,
3.3118894, 2.611247, 2.4462872, 1.6839775, 1.1553926, 2.158387, 1.533654,
1.2010561, 2.106749, 2.4965935, 3.1616364, 1.8951263, 1.7703358, 1.5098057,
0.88092697, 2.3299181, 2.1693668, 2.688698, 2.784411, 1.0062914, 2.2013755,
0.88123167, 1.6258563, 2.1565993, 1.9257479, 2.7067194, 2.3938549, 3.7530448,
1.5796599, 2.0446715, 1.8339522, 2.3523397, 2.902071, 2.129957, 1.4650953,
5.394662, 2.5770438, 2.2914875, 1.5767151, 1.6555734, 1.432461, 2.678148,
2.1838462, 1.8507156, 2.5777416, 2.333299, 1.687844, 2.6027796, 2.383405,
2.2040124, 3.8586795, 2.336818, 1.3995559, 1.5332103, 2.3523715, 1.8857346,
2.3162746, 1.4589972, 2.1819286, 1.0413275, 1.5431828, 2.5544188, 2.595261,
0.9894398, 0.8586521, 1.9768665, 1.5337317, 3.0291343, 3.434483, 0.41427672,
2.959281, 2.9052844, 3.132837, 2.616747, 1.1312762, 0.6194138, 2.0648239,
2.2178416, 2.3731132, 1.8225563, 1.7179387, 1.0503062, 2.2552388, 1.6232202,
0.36250675, 3.014674, 2.8538818, 1.8457974, 1.3054705, 2.9878192, 0.45808142,
3.1226902, 2.242302, 1.9838678, 2.5927513, 2.525559, 0.3894707, 2.4083557,
1.0291669, 1.5516076, 2.7709298, 1.9105201, 0.6591377, 0.82334787, 2.4801574,
2.365138, 0.86658496, 3.344652, 2.531079, 2.4922552, 2.471424, 3.536315,
1.6707703, 1.7299283, 0.99995935, 2.5831614, 2.4717236, 2.239201, 1.9192356,
1.0377275, 1.6084751, 2.7890332, 2.7823157, 2.8256266, 1.8795539, 1.6405061,
1.8728669, 0.64258313, 1.2975049, 2.7347412, 1.9747602, 2.0938547, 1.8634841,
1.4485981, 1.2237, 1.1691387, 1.1850374, 1.4594561, 1.472052, 2.5601006,
1.8576674, 2.2290359, 1.8232343, 2.2172456, 0.8472641, 2.7908704, 2.7669537,
1.7275047, 2.4300227, 1.2090333, 2.0327008, 2.2590246, 2.4076104, 4.3587313,
1.5879714, 0.9475082, 3.1533687, 2.0292883, 0.69573283, 1.2292244, 1.4356526,
5.0676155, 3.006452, 1.8688546, 3.5911007, 2.0544164, 1.7374315, 2.2755961,
2.4403079, 2.2602282, 2.2439747, 2.3442135, 1.4971428, 2.8340101, 1.2216152,
3.1396484, 1.9984121, 1.437942, 1.2476842, 0.7576814, 0.688195, 2.6634033,

4.558322, 2.2994409, 3.4298196, 2.2624478, 0.96138877, 2.6558185, 0.90769434,
1.4701842, 1.0133206, 1.3575872, 4.1623673, 1.0213381, -0.16425502, 1.5820436,
2.6323724, 0.37906766, 1.9854004, 2.8454146, 2.4177308, 2.1306434, 2.0691886,
0.67281413, 2.6132956, 2.3323467, 2.3451161, 1.1878362, 2.7413857, 0.3708523,
2.2859483, 2.705707, 2.3200378, 2.4679239, 5.0394135, 2.8431082, 1.919657,
2.0427263, 0.7048226, 1.5141263, 1.4654465, 1.7121468, 2.1790657, 1.5590777,
3.141472, 1.0200487, 2.2957315, 3.0156844, 1.6369133, 2.5922332, 2.0489244,
1.7966099, 1.6542807, 1.6883233, 1.6868457, 1.6107928, 0.87962866, 1.9335858,
2.4998655, 2.9094841, 3.410619, 1.9089141, 2.8169014, 1.176867, 1.7950351,
2.1363451, 1.7817794, 2.2021852, 0.3829527, 1.6837504, 1.1151673, 1.0739269,
1.2912171, 2.258997, 2.6576595, 2.1063313, 3.545848, 1.9837009, 2.830559,
2.0260904, 2.0163846, 3.1040967, 1.2842276, 2.306172, 1.7343255, 2.7873707,
3.4808793, 2.4076233, 0.3651322, 2.7701879, 2.1192398, 1.9145803, 2.0619287,
2.3568785, 2.2086802, 0.9765823, 0.4876575, 2.2320833, 1.9473653, 2.5228567,
0.72658515, 2.596894, 2.1576223, 2.7380495, 2.5820975, 3.4402392, 2.3699162,
3.537905, 2.3162465, 0.9262736, 0.9265113, 1.8932996, 2.1497207, 3.7301784,
2.2351818, 2.3390825, 2.7658007, 1.0971798, 1.8213315, 2.1536484, 1.1083862,
1.4167075, 2.2023215, 1.8096524, 2.9975145, 0.977622, 0.7740417, 3.8198338,
1.659487, 1.7109499, 1.8564935, 0.83885074, 2.0282688, 1.2965759, 4.545182,
6.4937887, 1.1400476, 6.0989285, 1.582449, 0.9865901, 1.5474725, 2.0352936,
2.3681598, 1.4678364, 1.1095016, 0.9502987, 2.4354677, 1.7899182, 1.4556519,
2.408769, 1.2880914, 4.455206, 1.3983742, 1.5263894, 1.440604, 3.096271,
1.2273235, 1.8992972, 2.6404524, 2.4824462, 2.0923896, 1.3243269, 1.9541999,
3.1714969, 2.8719935, 2.1255355, 1.4496933, 2.550601, 2.6856942, 2.0150547,
2.091553, 1.6381252, 2.3172958, 2.021328, 2.7478213, 1.7489216, 1.5857123,
3.398544, 2.1144507, 1.9907988, 0.815657, 2.1514616, 1.2902586, 1.2575192,
1.7412989, 2.958302, 1.8778555, 1.7597141, 1.1930346, 2.5827456, 1.6087236,
2.7137775, 1.7373811, 3.6001487, 1.2869825, 0.6329211, 2.484237, 2.081861,
2.4899635, 1.4390247, 2.173996, 2.4908037, 0.66925275, 2.095701, 2.944511,
5.037956, 1.6121033, 1.1024437, 1.1303407, 2.6472707, 2.0417967, 2.56989,
2.2139907, 2.28437, 1.2603061, 1.4446386, 0.7088835, 1.0795496, 3.0398805,
2.2813363, 3.3158438, 1.3010762, 2.63084, 2.5981696, 2.3262525, 1.1573855,
3.6336036, 1.8705859, 1.0747962, 1.0462513, 1.8907034, 1.8589337, 0.6305908,
2.127997, 2.1149087, 0.74330926, 1.1350967, 2.5174909, 2.133687, 2.9583373,
3.9506803, 2.192054, 1.2012514, 2.917632, 2.7054944, 3.0043273, 4.721367,
1.2043378, 1.4742934, 2.3563848, 0.62190783, 2.7102103, 1.7276592, 2.0498667,
0.950005, 1.567734, 3.0224745, 2.33281, 3.7264693, 0.72077084, 0.71861696,
3.4334013, 1.3458301, 1.3589904, 2.8523855, 1.4370742, 2.3057523, 1.3926105,
3.4457762, 2.7791605, 0.23114878, 1.4685259, 1.0028636, 3.2874637, 3.2883482,
2.4159253, 0.5405748, 2.595709, 1.8503807, 0.86063385, 2.8055756, 0.5729292,
3.0228248, 2.9846592, 1.9497259, 2.1585636, 3.5523174, 1.8865768, 1.6746821,
1.8397803, 1.7289672, 1.5305542, 2.105289, 1.5021765, 1.7532665, 1.7510449,
1.5480895, 1.9665401, 2.5466678, 2.128471, 2.2536461, 2.489623, 2.3261886,
3.0096219, 1.2924033, 1.632499, 2.8745298, 1.2066631, 1.3562297, 2.8613863,
1.643219, 1.5409862, 3.0229194, 2.2391114, 1.0688968, 1.8549966, 3.1654632,
1.8881259, 0.9157139, 2.7796917, 1.3624507, 2.0158346, 1.105554, 1.2014011,
2.8819885, 1.5628548, 1.1598967, 1.1838439, 2.6850336, 1.5522953, 1.4279041,
2.8237557, 2.9651089, 3.2392163, 2.4651208, 2.333727, 3.4585323, 1.6206179,

1.6601504, 2.050119, 2.158924, 2.8369458, 0.8912978, 0.937905, 1.5440059,
2.3462822, 2.9180455, 3.8730967, 2.3939295, 1.8010483, 2.4035554, 3.0113585,
4.4646645, 2.436034, 2.0493329, 2.13655, 1.8267982, 3.1085062, 2.7082763,
3.6627576, 2.520423, 2.6630332, 3.1236181, 1.9760562, 2.9186387, 4.6393833,
1.3854942, 2.0803547, 1.6278216, 2.3515973, 4.6713085, 2.7054753, 1.8770204,
3.8507867, 1.8861616, 1.975539, 2.4174447, 1.427659, 0.8419541, 1.9698919,
2.197443, 2.205711, 3.3745282, 1.1623569, 0.35644382, 2.855449, 1.764394,
2.8680305, 0.5246126, 2.5015779, 1.86726, 1.7651417, 2.3482637, 3.0169423,
1.2320257, 2.6322658, 2.565444, 4.1851196, 2.4957218, 1.5736706, 0.8735952,
2.2729163, 1.5951129, 2.3129642, 3.4604867, 2.0296512, 2.36939, 1.843338,
1.7991042, 2.97337, 2.4191945, 2.4621754, 0.7784594, 2.286162, 1.1049219,
0.65563846, 1.7788701, 2.9696136, 0.53936297, 2.9886103, 2.5912986, 2.335022,
2.175992, 1.2284155, 1.6709944, 2.3580866, 3.1973708, 2.277217, 1.2682277,
2.5157394, 1.6411922, 2.1468587, 2.1551654, 2.1679692, 1.9463359, 2.5186167,
3.0767362, 1.277091, 2.7773163, 0.7085781, 1.4575081, 1.837162, 2.1968288,
0.6604443, 2.288286, 1.103253, 0.10342562, 1.2345517, 1.4186109, 2.1309702,
2.7209845, 2.4267797, 2.3888447, 1.6093187, 1.2309868, 1.1792195, 2.665735,
1.5238539, 1.5906281, 2.3652694, 0.7005427, 3.2896488, 1.4612226, 2.9220028,
1.6409453, 2.5172338, 1.0570321, 3.2096093, 0.6295742, 1.0897429, 2.3557615,
2.0005345, 1.6024473, 1.6357, 1.9016466, 0.8541361, 2.4275613, 1.7189887,
0.8992839, 2.3888338, 2.7549853, 1.2277533, 1.3574009, 5.0112433, 0.8497683,
1.5354998, 1.6892061, 2.7702212, 2.8905983, 2.618546, 1.732549, 2.8644338,
2.7868161, 2.3735776, 2.4489172, 1.2367166, 0.8900746, 2.5474672, 1.1936549,
1.2749572, 3.0635123, 2.0294151, 0.9006346, 1.0164889, 4.997998, 2.1522765,
1.1348306, 1.0533286, 2.1794806, 2.8729455, 2.9085393, 2.6377265, 1.0976076,
1.3380857, 0.8484568, 0.91884947, 3.8183632, 1.0589969, 0.8701611, 2.462185,
3.006429, 1.99508, 2.5787477, 1.5803943, 1.6175997, 1.8098303, 1.2614763,
2.5518205, 1.6737193, 0.74471223, 2.2327387, 2.1143823, 3.4099634, 0.78429496,
2.410005, 2.4795241, 2.1799853, 2.217856, 1.3534262, 2.319926, 1.8665783,
2.4624758, 2.394954, 2.831987, 2.5677068, 2.6295688, 2.2483187, 1.4530718,
1.0687401, 1.237719, 2.6523619, 1.4044886, 3.321553, 2.543067, 1.894872,
1.3015351, 2.183961, 1.0744811, 1.3354058, 1.0765558, 3.4752185, 1.9609976,
0.8528582, 2.038066, 2.382288, 1.5199242, 2.6377637, 2.716037, 2.1356251,
0.87376755, 0.7848717, 2.0235717, 1.4965951, 1.7619373, 1.9954674, 1.5942318,
3.6774375, 0.08912015, 1.4316123, 0.66874456, 1.4287789, 2.3799217, 2.848597,
2.4521685, 1.4717989, 3.0066383, 2.27524, 1.9263699, 1.965646, 3.0927434,
1.7462527, 1.1284469, 3.2722855, 0.6833433, 1.5567484, 1.375026, 1.7092006,
2.7996483, 2.0603611, 1.5883791, 1.6477156, 1.3904152, 3.407138, 2.7993302,
2.3154979, 1.251627, 1.4825662, 0.7919916, 2.3092303, 3.980184, 0.96607196,
0.9016928, 2.621974, 2.7498488, 2.0562582, 3.7934847, 2.370739, 1.9146934,
3.2434907, 4.062347, 1.5553806, 2.2002635, 2.4338622, 3.17645, 1.763813,
1.2953633, 2.7763467, 2.325243, 1.5771266, 2.0605326, 1.14308, 0.92048275,
3.3330095, 3.1808574, 0.91463, 2.5705137, 2.1149433, 1.8232671, 0.44528836,
2.145521, 1.3819435, 2.3832064, 2.022441, 3.5028553, 1.9363037, 3.6897395,
1.878963, 2.9297004, 1.9071226, 2.161615, 2.834694, 0.6667819, 1.5770046,
1.535234, 1.7145827, 1.3898818, 1.5994573, 1.4207348, 1.4385977, 3.5765028,
0.9388084, 2.111147, 2.6810975, 2.469967, 1.9779761, 2.892637, 2.7466555,
1.1189836, 3.0659378, 2.434482, 2.3126793, 2.099525, 1.8249705, 3.126338,

2.0358412, 2.870573, 2.1896548, 2.6013505, 2.9470594, 1.7666242, 2.1015897,
2.8293514, 2.2971368, 1.1307677, 1.3855637, 3.0994294, 3.5937705, 1.9953985,
1.6654773, 1.5822947, 1.3635402, 1.0802512, 3.0230153, 0.74982667, 2.2035687,
3.3540142, 1.8618543, 2.5281878, 0.69682455, 2.1057248, 2.2044258, 2.2930067,
2.9979315, 3.882776, 2.4478798, 2.3549957, 1.740727, 2.766346, 1.4044378,
2.0601726, 2.7810493, 2.2639909, 2.6281357, 1.1687076, 0.8907152, 1.390211,
2.728701, 2.2362494, 6.352539, 1.7122858, 1.1855527, 1.6370394, 1.678456,
1.5301368, 2.9959893, 1.8578825, 2.535406, 0.6115953, 1.0797064, 2.3566713,
2.9977446, 1.9507768, 2.5357494, 2.5743303, 2.3320937, 2.0319536, 1.7672472,
2.3949366, 1.9479232, 0.91510093, 2.1368196, 1.3407159, 3.0566847, 2.3584058,
3.7063863, 2.7225552, 1.6393539, 5.323141, 1.8758943, 2.2997925, 1.0004752,
2.9594984, 2.1525416, 3.039569, 1.1964208, 1.7996941, 0.7320026, 2.8647087,
1.0513602, 1.6438287, 2.0780838, 0.8188912, 2.1557121, 2.1204684, 1.4819441,
1.4036922, 2.108715, 3.0937235, 2.389667, 2.9676642, 2.99544, 1.7057891,
6.8173513, 1.9966161, 2.5042424, 0.6585379, 2.7332542, 1.8118554, 0.39061737,
1.843046, 2.3677747, 3.3257618, 2.2578602, 1.7303314, 4.050285, 2.5272117,
0.44459385, 0.8401478, 3.4732113, 2.7526917, 1.8248754, 2.481262, 1.8465726,
1.8295532, 2.2110977, 5.517272, 1.6415219, 2.3125458, 1.745663, 1.6826192,
2.1004932, 2.0830333, 2.073357, 3.4417608, 1.5827868, 1.041304, 2.639734,
2.3503642, 1.5843269, 4.2018948, 1.1464942, 2.266243, 2.0419853, 2.8288774,
2.0330877, 4.7160225, 1.0909411, 2.1840148, 2.3345737, 2.3201966, 0.8931649,
1.6179895, 1.4492778, 2.7186375, 2.6893606, 2.3416932, 1.679007, 4.5704603,
1.5728419, 0.94782555, 1.0272825, 2.7874303, 2.1398692, 2.3872638, 0.5433458,
1.1002915, 2.139215, 3.0830343, 1.0670464, 1.8784677, 3.7223277, 1.9805534,
1.4709779, 0.7151474, 1.7898958, 2.682846, 2.3388472, 2.5343595, 1.7847717,
1.5311947, 1.5179013, 2.548312, 1.9392302, 1.9897332, 3.3691864, 0.775111,
0.96621805, 2.1838076, 3.4997368, 1.2266052, 3.505635, 3.526775, 1.1454499,
2.0397997, 2.6063232, 3.1000595, 2.7326057, 1.0972251, 1.7145727, 1.3196061,
1.1150031, 2.3189254, 3.5369635, 2.5209908, 2.2128823, 1.5196376, 1.6517683,
1.2867905, 2.0962007, 0.63266826, 1.3088613, 2.0123796, 2.9095945, 1.9311597,
1.2104871, 1.0430313, 2.4214664, 0.873478, 2.5276356, 2.663938, 1.015916,
1.5320246, 3.1482267, 2.8536224, 3.299944, 1.0346552, 1.6772399, 1.909209,
1.6328406, 2.9239988, 0.45997643, 1.2939171, 2.3210464, 6.7448425, 2.7133017,
1.077732, 2.5645573, 1.8297724, 2.1514673, 1.6052046, 2.4176483, 2.262927,
1.0414342, 0.7526009, 3.1987503, 1.1472023, 1.7448146, 0.99751854, 1.5178804,
2.1263967, 2.9722888, 1.6615605, 2.411767, 2.8251936, 2.0904832, 1.6970036,
1.936671, 1.2769198, 1.8047049, 1.0309371, 2.778438, 3.0958877, 1.6150947,
1.7352004, 0.8719716, 1.4176166, 3.9215412, 3.053881, 3.6983662, 2.6131835,
2.2273304, 4.262268, 2.1249523, 2.5935912, 2.164531, 1.9331232, 2.2628722,
2.4782853, 1.8720379, 2.4574308, 2.5837677, 3.3185182, 2.1433163, 2.9105544,
0.72232455, 1.9689667, 2.0425854, 1.7119012, 2.5359564, 1.0990252, 2.81862,
1.8948914, 1.467145, 1.995999, 2.0275035, 2.3484526, 2.3842373, 2.162292,
1.4688251, 3.007269, 1.8071173, 5.4977093, 0.74154496, 1.9753418, 1.2327013,
2.314538, 1.9554636, 2.134164, 1.0368654, 1.4932653, 1.4113543, 2.2937922,
1.4542071, 1.3194017, 1.323469, 0.8290416, 0.8085568, 0.92358124, 1.0036446,
2.6732812, 0.3568794, 2.644081, 1.2465023, 0.9657847, 3.0693684, 3.2645564,
0.48990053, 2.4613779, 1.7831073, 2.7488313, 1.4987909, 1.2966049, 3.1596677,
2.5343242, 1.4526737, 1.9480525, 1.3468665, 2.2331154, 1.0273525, 2.061911,

1.2148778, 2.3709807, 1.3530347, 1.2559923, 1.9408681, 1.7885368, 2.4473796,
2.5466611, 0.6766649, 1.4155054, 1.4558684, 2.5920196, 1.7359926, 0.82395387,
2.0780606, 1.5739588, 2.148093, 1.8098736, 2.4890165, 3.3093786, 1.9677993,
2.4786453, 2.3344107, 2.5603805, 2.379902, 1.837713, 2.099197, 1.2838485,
0.840453, 1.9988828, 0.65279704, 1.6010168, 2.7036781, 1.5485464, 2.456367,
2.667967, 1.1282078, 1.5811033, 1.2650204, 1.6457641, 2.509304, 1.8632166,
1.4692457, 1.8390698, 2.0676346, 2.1256633, 0.9143683, 0.8626294, 0.99648845,
2.3051224, 1.6996121, 4.94091, 0.43571633, 1.490959, 2.0635529, 0.742332,
1.4401133, 2.5113175, 3.6781187, 1.18829, 2.7417598, 1.2366185, 2.443326,
2.1838818, 1.206465, 2.575751, 2.56989, 2.609236, 2.043407, 1.6633666,
2.2602072, 0.78854, 3.480335, 2.8457522, 1.4445925, 2.8468537, 0.9593288,
1.6617643, 1.4959567, 1.4259806, 1.8269026, 1.4531503, 3.1738203, 0.59700453,
3.8889744, 2.6803324, 1.5735328, 0.8662783, 3.8056757, 3.4013374, 0.96412766,
1.8534772, 2.3106394, 1.4912114, 2.3283744, 2.9595075, 2.3857002, 1.147581,
1.3240529, 3.3960598, 0.94611204, 2.4544094, 3.3930068, 1.5849907, 2.9829829,
3.1080627, 2.532328, 3.8534932, 2.4704676, 1.3678932, 2.1924906, 0.65828323,
2.381267, 2.6790109, 1.7169745, 4.92097, 2.9427547, 2.9730172, 2.7300186,
3.9420686, 3.8173742, 4.1186566, 1.379612, 2.8687758, 2.6797981, 1.4633102,
1.0038414, 2.3266766, 2.6039865, 1.1509316, 2.1200025, 1.2328382, 2.0349076,
2.5858188, 2.89641, 2.4402933, 2.0486212, 2.76714, 1.4450535, 3.3100905,
0.8631218, 0.8024701, 1.2144283, 3.4722524, 1.4158452, 2.0796895, 2.3985846,
1.6016402, 1.7726154, 0.797843, 3.3995378, 3.5124884, 2.7009213, 0.9572847,
3.522322, 2.8040433, 2.5985372, 1.3364617, 1.3876579, 2.9518645, 1.8593516,
2.0901055, 2.2081883, 2.0832534, 2.1275313, 1.2981346, 3.1212933, 2.8450851,
0.7733109, 2.045426, 2.2364833, 1.492343, 2.5300007, 2.7306194, 0.9052267,
3.0589411, 5.525116, 0.6242388, 2.1088846, 2.3606262, 2.2683415, 2.5882635,
2.3669446, 1.3468401, 2.3122349, 2.5888085, 2.3127213, 1.3641751, 1.151647,
1.3771766, 1.2980189, 1.8069756, 0.734427, 2.445116, 2.2233863, 2.043786,
2.0451775, 0.90983945, 1.4571056, 2.2522743, 0.9289102, 1.8091311, 3.2208636,
3.075812, 1.6633112, 3.115486, 1.5417066, 2.6344972, 2.0034726, 0.62678653,
1.6850517, 0.8845375, 2.5235155, 2.4231632, 4.208503, 3.2697117, 2.7014785,
3.2285013, 2.645646, 2.1631155, 3.0889034, 2.4203224, 2.6105487, 2.2185874,
1.9706769, 1.1681839, 1.5330722, 2.605154, 1.7792165, 2.3239841, 2.3013353,
4.079622, 1.4781744, 2.720171, 2.4411104, 1.8764998, 1.7461603, 2.2178757,
2.7074656, 3.4804642, 2.1297913, 3.5520363, 1.7107104, 1.2970939, 1.1979373,
2.5274057, 1.3454995, 2.150007, 1.4590434, 1.5746102, 1.4717782, 5.1556845,
1.4799137, 1.7751489, 1.0152644, 1.3391212, 3.3043587, 0.75023293, 1.5015858,
1.7166817, 1.7866018, 0.98277533, 1.8203033, 0.9747681, 2.5926533, 2.3822465,
0.93708205, 2.4774508, 3.7149959, 1.6236076, 1.4087927, 1.2821937, 2.1660008,
0.5639436, 1.553494, 2.052226, 2.8614295, 3.037568, 2.7770996, 2.816753,
3.1424932, 1.4493291, 1.8869283, 2.5279276, 2.385326, 1.2053517, 2.618237,
1.5659943, 1.1567982, 2.6003191, 1.7160454, 1.6727504, 3.2336943, 1.9780098,
2.6367114, 1.8971245, 2.3197632, 1.0408682, 1.418751, 2.7229023, 2.36368,
2.3609676, 1.1761888, 1.2761755, 1.8294296, 2.855733, 0.769855, 2.7936795,
1.6071016, 1.6381316, 2.046625, 0.19497567, 3.8536532, 2.6620507, 3.3066432,
1.5927213, 3.1685236, 2.3988438, 1.738423, 1.6425583, 0.78224945, 3.0517967,
1.4288638, 2.7469015, 2.6824756, 1.7136298, 2.1935432, 2.0643682, 2.1957426,
2.3351884, 0.9401052, 2.8198557, 3.0330718, 0.9299974, 3.5497081, 2.037836,

1.8739961, 2.3009715, 3.06655, 3.2570045, 1.3706474, 0.9091214, 2.4708955,
2.3519173, 2.038397, 1.5002127, 1.1096873, 1.0492735, 2.1573381, 2.3455405,
3.27553, 1.631994, 2.3039455, 2.3533206, 1.495307, 3.2447028, 2.6573923,
1.4307187, 2.979393, 2.3384886, 0.9411809, 2.748363, 2.9676986, 2.3292804,
1.4465886, -0.07712984, 1.9197521, 0.6149585, 2.8120313, 1.6691194, 1.30808,
1.8874705, 1.3713768, 1.8236015, 2.8992822, 1.9147706, 2.2665324, 4.6619987,
0.9993953, 2.282126, 1.445409, 2.5615509, 4.3453107, 2.2666323, 2.969769,
1.2113419, 0.8324924, 1.3047483, 2.9535189, 1.5690799, 3.4831884, 1.6752725,
1.2009263, 1.9844043, 2.0461462, 0.7447983, 2.5012636, 1.4366913, 2.9872518,
3.122786, 0.9063157, 0.6020107, 2.9443731, 2.03363, 1.9673834, 1.7531157,
0.90579367, 0.7258876, 1.775829, 2.7869163, 1.1384482, 1.8705282, 3.0183253,
1.422044, 0.3677215, 3.466259, 1.6414263, 2.6548884, 1.6859798, 2.2901952,
2.1334267, 0.837826, 1.9024282, 2.6497855, 2.0758276, 0.8361368, 1.8229998,
1.5209775, 2.7515574, 2.685192, 1.1581146, 4.4712534, 2.1072845, 1.036072,
5.1048384, 1.8129997, 2.3343744, 2.1582332, 2.0439768, 2.7377906, 1.9853934,
2.8391097, 2.3188043, 0.7835326, 2.1543708, 1.0296288, 1.4884794, 1.6183759,
2.6956592, 1.8556659, 1.1630023, 2.252492, 2.8100305, 1.261531, 1.6404479,
2.1182823, 3.2101822, 1.1748697, 1.6000881, 6.755994, 0.973543, 1.7246234,
1.6119931, 2.566164, 0.84823936, 2.4130373, 0.68265617, 2.6965752, 2.1287274,
2.7278752, 2.0353932, 2.14606, 2.7209418, 2.4154415, 2.3403368, 1.4944389,
1.7867169, 2.2868435, 1.7742908, 3.332398, 2.4931946, 1.424496, 1.2590429,
2.040131, 2.3273144, 1.1907077, 2.2047296, 2.5181885, 3.74154, 1.1112094,
2.1415281, 4.19674, 1.5684795, 2.89943, 1.8347797, 3.5143456, 3.2513165,
1.2923832, 1.0241119, 1.5107825, 1.7921805, 2.0974815, 1.3678219, 2.6606064,
2.1476579, 2.345335, 1.4233384, 1.3319722, 3.3459754, 2.3625927, 3.0385807,
3.0160306, 2.5641518, 4.792306, 2.4859128, 2.2935588, 2.0404425, 3.3197281,
1.730794, 1.697653, 2.3770149, 0.92742413, 3.9866002, 1.9731244, 2.6825638,
2.1669576, 3.1973493, 2.2456791, 0.52336085, 2.6006153, 1.5921152, 1.7518858,
1.6093464, 1.6685017, 2.4301329, 2.0397854, 1.9104408, 2.5177183, 2.7919402,
1.5007824, 1.4697492, 2.8252888, 2.6329384, 2.492169, 1.1975076, 0.77655053,
2.0081549, 2.1841063, 2.561262, 1.343324, 2.5924058, 2.003411, 1.7492883,
1.8478545, 1.6800508, 2.4050612, 1.3088504, 2.6857524, 2.5808, 1.0415248,
2.14578, 3.482493, 3.546237, 1.947895, 1.912622, 2.174475, 1.8668277, 1.7735825,
2.479575, 2.4648495, 1.7543521, 1.2961116, 1.4858669, 2.123374, 1.8222156,
1.7078757, 0.8930624, 2.175773, 2.6957822, 2.5350857, 2.4001193, 1.9202149,
2.4388494, 2.9415674, 2.495235, 1.5957873, 1.0665202, 1.4560673, 2.180534,
2.9984348, 2.9224463, 1.9603637, 1.0076325, 0.77902955, 2.9296422, 2.4230638,
0.6531013, 2.914185, 2.3211293, 1.5304011, 3.16889, 1.3605701, 1.6655861,
3.2311306, 2.762546, 2.037523, 2.2209795, 1.3947875, 0.70936155, 1.3663146,
2.551239, 1.9974939, 2.3341136, 0.7651106, 6.5873203, 2.0323148, 1.5391418,
2.0613592, 0.9394916, 0.9481921, 1.1957743, 1.9386406, 2.2422957, 0.5934086,
1.5702922, 2.1851122, 1.5220976, 2.037297, 0.86742806, 1.1775014, 1.0029757,
2.3039737, 1.1290543, 2.5768478, 2.6155124, 2.128623, 0.71955574, 1.7860667,
3.2088213, 2.59835, 0.97762036, 3.1401443, 0.87349117, 2.828392, 2.4873579,
0.6240799, 2.335403, 2.8100388, 1.7899749, 1.5687444, 3.0414188, 2.0697918,
1.1461213, 2.1721034, 3.2421677, 3.0535207, 2.764039, 2.2468023, 1.2077665,
0.6257949, 2.807015, 1.4560475, 2.3527608, 1.6790509]

1.3.9 Evaluate Model

Now that we have predictions, we can compare them to our actual values and evaluate the quality of our model.

[ ]:
```
import math

from sklearn import metrics

mean_squared_error = metrics.mean_squared_error(
    np.array(predicted_median_values) * TARGET_FACTOR,
    testing_df[target_column] * TARGET_FACTOR
)
print("Mean Squared Error (on training data): %0.3f" % mean_squared_error)

root_mean_squared_error = math.sqrt(mean_squared_error)
print("Root Mean Squared Error (on training data): %0.3f" %␣
 ↪root_mean_squared_error)
```

```
Mean Squared Error (on training data): 4692367165.939
Root Mean Squared Error (on training data): 68500.855
```

What is this telling us? The mean square error is somewhat hard to think about. However, whenever you take the root you get the units of the target column. In our test run, we were `68700.557` dollars off on our predictions. (Your numbers might be slightly different because we randomly shuffled the data before splitting it into training and testing datasets.)

Is that good?

Let's see what the mean price is in our test data.

[ ]:
```
testing_df[target_column].mean() * TARGET_FACTOR
```

[ ]: 207075.28730620118

About 206,700 dollars. 68,700 is about 33% of 206,700 so our model is off by a mean of 33% of the actual price. I probably wouldn't make many bets using this model.

1.4 Exercise 4: Hyperparameters

There are a few hyperparameters that we can adjust in order to try to improve our model. In the code cell below, you'll find most of the code that we've used so far in this lab. There are three `TODO` markers in the code. Find them and:

1. Have the model use the Adam Optimizer
2. Configure the training `Dataset`. Experiment with different batch sizes. Leave the batch size that performs the best in the code.
3. Configure the testing `Dataset`.

Student Solution

[ ]:
```python
import math
import numpy as np
import pandas as pd
import tensorflow as tf

from sklearn import metrics

tf.keras.backend.set_floatx('float64')

# Load the data
url = 'california-housing-prices.zip'

housing_df = pd.read_csv(url)

# Repair data
has_all_data = housing_df[~housing_df['total_bedrooms'].isna()]
sums = has_all_data[['total_bedrooms', 'total_rooms']].median().tolist()
bedrooms_to_total_rooms_ratio = sums[0] / sums[1]
missing_total_bedrooms_idx = housing_df['total_bedrooms'].isna()
housing_df.loc[missing_total_bedrooms_idx, 'total_bedrooms'] = housing_df[
    missing_total_bedrooms_idx]['total_rooms'] * bedrooms_to_total_rooms_ratio

# Create lists of column names
target_column = 'median_house_value'
feature_columns = [c for c in housing_df.columns if c != target_column]
numeric_feature_columns = [c for c in feature_columns if c != 'ocean_proximity']

# Normalize the feature columns
housing_df.loc[:, numeric_feature_columns] = (
    housing_df[numeric_feature_columns] -
      housing_df[numeric_feature_columns].min()) / (
          housing_df[numeric_feature_columns].max() -
            housing_df[numeric_feature_columns].min())

# Scale the target column
TARGET_FACTOR = 100000
housing_df[target_column] = housing_df[target_column] / TARGET_FACTOR

# Test/Train split
housing_df = housing_df.sample(frac=1)
test_set_size = int(len(housing_df) * 0.2)
testing_df = housing_df[:test_set_size]
training_df = housing_df[test_set_size:]

# Create TensorFlow features
housing_features = [
    tf.feature_column.numeric_column(c, dtype=tf.dtypes.float64)
```

```
      for c in numeric_feature_columns
]
housing_features.append(
    tf.feature_column.categorical_column_with_vocabulary_list(
        key='ocean_proximity',
        vocabulary_list=sorted(housing_df['ocean_proximity'].unique()))
)

# Create model
linear_regressor = tf.estimator.LinearRegressor(
    feature_columns=housing_features,
    optimizer = 'Adam'
    # TODO: Set Optimizer


)

# Train the model
def training_input():
  ds = tf.data.Dataset.from_tensor_slices((
    {c: training_df[c] for c in feature_columns},  # feature map
    training_df[target_column]                     # labels
  ))
  ds = ds.repeat(100)
  ds = ds.shuffle(buffer_size=10000)
  ds = ds.batch(100)
  # TODO: Configure Dataset
  return ds

linear_regressor.train(
 input_fn=training_input
)

# Make predictions
def testing_input():
  ds = tf.data.Dataset.from_tensor_slices((
    {c: testing_df[c] for c in feature_columns},  # feature map
    testing_df[target_column]                     # labels
  ))
  # TODO: Configure Dataset
  ds = ds.batch(1)
  return ds

predictions_node = linear_regressor.predict(
  input_fn=testing_input,
)
```

```python
# Convert the predictions to a NumPy array
predicted_median_values = np.array(
    [item['predictions'][0] for item in predictions_node])

# Find the RMSE
root_mean_squared_error = math.sqrt(
    metrics.mean_squared_error(
      predicted_median_values * TARGET_FACTOR,
      testing_df[target_column] * TARGET_FACTOR
))

print("Root Mean Squared Error (on training data): %0.3f" %
root_mean_squared_error)
```

```
INFO:tensorflow:Using default config.
WARNING:tensorflow:Using temporary folder as model directory:
/var/folders/q7/wrxzkb515gqcskvhd38dwx6h0000gn/T/tmpoo22j73g
INFO:tensorflow:Using config: {'_model_dir':
'/var/folders/q7/wrxzkb515gqcskvhd38dwx6h0000gn/T/tmpoo22j73g',
'_tf_random_seed': None, '_save_summary_steps': 100, '_save_checkpoints_steps':
None, '_save_checkpoints_secs': 600, '_session_config': allow_soft_placement:
true
graph_options {
  rewrite_options {
    meta_optimizer_iterations: ONE
  }
}
, '_keep_checkpoint_max': 5, '_keep_checkpoint_every_n_hours': 10000,
'_log_step_count_steps': 100, '_train_distribute': None, '_device_fn': None,
'_protocol': None, '_eval_distribute': None, '_experimental_distribute': None,
'_experimental_max_worker_delay_secs': None, '_session_creation_timeout_secs':
7200, '_checkpoint_save_graph_def': True, '_service': None, '_cluster_spec':
ClusterSpec({}), '_task_type': 'worker', '_task_id': 0, '_global_id_in_cluster':
0, '_master': '', '_evaluation_master': '', '_is_chief': True,
'_num_ps_replicas': 0, '_num_worker_replicas': 1}
WARNING:tensorflow:From
/Users/josemartinez/opt/anaconda3/envs/data/lib/python3.9/site-
packages/tensorflow/python/training/training_util.py:235:
Variable.initialized_value (from tensorflow.python.ops.variables) is deprecated
and will be removed in a future version.
Instructions for updating:
Use Variable.read_value. Variables in 2.X are initialized automatically both in
eager and graph (inside tf.defun) contexts.
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Create CheckpointSaverHook.
```

```
/Users/josemartinez/opt/anaconda3/envs/data/lib/python3.9/site-
```

```
packages/tensorflow/python/keras/engine/base_layer_v1.py:1727: UserWarning:
`layer.add_variable` is deprecated and will be removed in a future version.
Please use `layer.add_weight` method instead.
  warnings.warn('`layer.add_variable` is deprecated and '

INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Calling checkpoint listeners before saving checkpoint 0…
INFO:tensorflow:Saving checkpoints for 0 into
/var/folders/q7/wrxzkb515gqcskvhd38dwx6h0000gn/T/tmpoo22j73g/model.ckpt.
INFO:tensorflow:Calling checkpoint listeners after saving checkpoint 0…
INFO:tensorflow:loss = 6.4933977, step = 0
INFO:tensorflow:global_step/sec: 466.755
INFO:tensorflow:loss = 0.42602116, step = 100 (0.215 sec)
INFO:tensorflow:global_step/sec: 833.453
INFO:tensorflow:loss = 0.51236886, step = 200 (0.120 sec)
INFO:tensorflow:global_step/sec: 866.536
INFO:tensorflow:loss = 0.44917774, step = 300 (0.115 sec)
INFO:tensorflow:global_step/sec: 863.498
INFO:tensorflow:loss = 0.41552353, step = 400 (0.116 sec)
INFO:tensorflow:global_step/sec: 853.474
INFO:tensorflow:loss = 0.43652564, step = 500 (0.117 sec)
INFO:tensorflow:global_step/sec: 625.74
INFO:tensorflow:loss = 0.5293705, step = 600 (0.160 sec)
INFO:tensorflow:global_step/sec: 761.377
INFO:tensorflow:loss = 0.5547879, step = 700 (0.131 sec)
INFO:tensorflow:global_step/sec: 756.679
INFO:tensorflow:loss = 0.46528324, step = 800 (0.132 sec)
INFO:tensorflow:global_step/sec: 780.099
INFO:tensorflow:loss = 0.54360473, step = 900 (0.128 sec)
INFO:tensorflow:global_step/sec: 784.814
INFO:tensorflow:loss = 0.39498952, step = 1000 (0.127 sec)
INFO:tensorflow:global_step/sec: 798.958
INFO:tensorflow:loss = 0.5511298, step = 1100 (0.125 sec)
INFO:tensorflow:global_step/sec: 790.5
INFO:tensorflow:loss = 0.40782532, step = 1200 (0.127 sec)
INFO:tensorflow:global_step/sec: 808.185
INFO:tensorflow:loss = 0.45543817, step = 1300 (0.124 sec)
INFO:tensorflow:global_step/sec: 817.007
INFO:tensorflow:loss = 0.45976555, step = 1400 (0.122 sec)
INFO:tensorflow:global_step/sec: 845.266
INFO:tensorflow:loss = 0.5487977, step = 1500 (0.118 sec)
INFO:tensorflow:global_step/sec: 822.918
INFO:tensorflow:loss = 0.6536463, step = 1600 (0.122 sec)
INFO:tensorflow:global_step/sec: 828.15
INFO:tensorflow:loss = 0.7104588, step = 1700 (0.121 sec)
INFO:tensorflow:global_step/sec: 792.952
```

```
INFO:tensorflow:loss = 0.5597425, step = 1800 (0.126 sec)
INFO:tensorflow:global_step/sec: 805.977
INFO:tensorflow:loss = 0.63266826, step = 1900 (0.124 sec)
INFO:tensorflow:global_step/sec: 823.601
INFO:tensorflow:loss = 0.39875472, step = 2000 (0.121 sec)
INFO:tensorflow:global_step/sec: 819.33
INFO:tensorflow:loss = 0.31715116, step = 2100 (0.122 sec)
INFO:tensorflow:global_step/sec: 816.826
INFO:tensorflow:loss = 0.560731, step = 2200 (0.122 sec)
INFO:tensorflow:global_step/sec: 820.541
INFO:tensorflow:loss = 0.51537585, step = 2300 (0.122 sec)
INFO:tensorflow:global_step/sec: 820.869
INFO:tensorflow:loss = 0.47113526, step = 2400 (0.122 sec)
INFO:tensorflow:global_step/sec: 820.742
INFO:tensorflow:loss = 0.5986018, step = 2500 (0.122 sec)
INFO:tensorflow:global_step/sec: 817.514
INFO:tensorflow:loss = 0.46997944, step = 2600 (0.122 sec)
INFO:tensorflow:global_step/sec: 812.314
INFO:tensorflow:loss = 0.41589805, step = 2700 (0.123 sec)
INFO:tensorflow:global_step/sec: 820.932
INFO:tensorflow:loss = 0.47074744, step = 2800 (0.122 sec)
INFO:tensorflow:global_step/sec: 817.42
INFO:tensorflow:loss = 0.57115287, step = 2900 (0.122 sec)
INFO:tensorflow:global_step/sec: 715.466
INFO:tensorflow:loss = 0.48840794, step = 3000 (0.140 sec)
INFO:tensorflow:global_step/sec: 769.586
INFO:tensorflow:loss = 0.47934693, step = 3100 (0.130 sec)
INFO:tensorflow:global_step/sec: 660.219
INFO:tensorflow:loss = 0.46150094, step = 3200 (0.151 sec)
INFO:tensorflow:global_step/sec: 772.393
INFO:tensorflow:loss = 0.38242698, step = 3300 (0.130 sec)
INFO:tensorflow:global_step/sec: 782.105
INFO:tensorflow:loss = 0.42785683, step = 3400 (0.128 sec)
INFO:tensorflow:global_step/sec: 827.888
INFO:tensorflow:loss = 0.6441854, step = 3500 (0.121 sec)
INFO:tensorflow:global_step/sec: 821.559
INFO:tensorflow:loss = 0.662757, step = 3600 (0.122 sec)
INFO:tensorflow:global_step/sec: 745.946
INFO:tensorflow:loss = 0.5232695, step = 3700 (0.134 sec)
INFO:tensorflow:global_step/sec: 801.624
INFO:tensorflow:loss = 0.37153855, step = 3800 (0.125 sec)
INFO:tensorflow:global_step/sec: 691.481
INFO:tensorflow:loss = 0.3151463, step = 3900 (0.144 sec)
INFO:tensorflow:global_step/sec: 809.703
INFO:tensorflow:loss = 0.5622544, step = 4000 (0.124 sec)
INFO:tensorflow:global_step/sec: 811.219
INFO:tensorflow:loss = 0.54935503, step = 4100 (0.123 sec)
INFO:tensorflow:global_step/sec: 814.817
```

```
INFO:tensorflow:loss = 0.37875146, step = 4200 (0.123 sec)
INFO:tensorflow:global_step/sec: 823.412
INFO:tensorflow:loss = 0.39219254, step = 4300 (0.121 sec)
INFO:tensorflow:global_step/sec: 827.356
INFO:tensorflow:loss = 0.43305162, step = 4400 (0.121 sec)
INFO:tensorflow:global_step/sec: 821.591
INFO:tensorflow:loss = 0.44197917, step = 4500 (0.122 sec)
INFO:tensorflow:global_step/sec: 815.46
INFO:tensorflow:loss = 0.6147056, step = 4600 (0.123 sec)
INFO:tensorflow:global_step/sec: 832.023
INFO:tensorflow:loss = 0.47087196, step = 4700 (0.120 sec)
INFO:tensorflow:global_step/sec: 815.508
INFO:tensorflow:loss = 0.3699667, step = 4800 (0.123 sec)
INFO:tensorflow:global_step/sec: 832.383
INFO:tensorflow:loss = 0.36661842, step = 4900 (0.120 sec)
INFO:tensorflow:global_step/sec: 816.813
INFO:tensorflow:loss = 0.3498941, step = 5000 (0.122 sec)
INFO:tensorflow:global_step/sec: 776.934
INFO:tensorflow:loss = 0.68245476, step = 5100 (0.129 sec)
INFO:tensorflow:global_step/sec: 646.633
INFO:tensorflow:loss = 0.48614365, step = 5200 (0.154 sec)
INFO:tensorflow:global_step/sec: 790.127
INFO:tensorflow:loss = 0.52082026, step = 5300 (0.127 sec)
INFO:tensorflow:global_step/sec: 809.33
INFO:tensorflow:loss = 0.45185712, step = 5400 (0.123 sec)
INFO:tensorflow:global_step/sec: 841.326
INFO:tensorflow:loss = 0.38673362, step = 5500 (0.119 sec)
INFO:tensorflow:global_step/sec: 805.497
INFO:tensorflow:loss = 0.43696678, step = 5600 (0.124 sec)
INFO:tensorflow:global_step/sec: 619.241
INFO:tensorflow:loss = 0.59276366, step = 5700 (0.161 sec)
INFO:tensorflow:global_step/sec: 736.133
INFO:tensorflow:loss = 0.5090029, step = 5800 (0.136 sec)
INFO:tensorflow:global_step/sec: 821.882
INFO:tensorflow:loss = 0.3942646, step = 5900 (0.122 sec)
INFO:tensorflow:global_step/sec: 833.763
INFO:tensorflow:loss = 0.5061787, step = 6000 (0.120 sec)
INFO:tensorflow:global_step/sec: 715.278
INFO:tensorflow:loss = 0.52209544, step = 6100 (0.140 sec)
INFO:tensorflow:global_step/sec: 736.12
INFO:tensorflow:loss = 0.5574725, step = 6200 (0.136 sec)
INFO:tensorflow:global_step/sec: 825.602
INFO:tensorflow:loss = 0.34081325, step = 6300 (0.121 sec)
INFO:tensorflow:global_step/sec: 688.986
INFO:tensorflow:loss = 0.39102668, step = 6400 (0.145 sec)
INFO:tensorflow:global_step/sec: 784.01
INFO:tensorflow:loss = 0.42713442, step = 6500 (0.127 sec)
INFO:tensorflow:global_step/sec: 821.107
```

```
INFO:tensorflow:loss = 0.7163536, step = 6600 (0.122 sec)
INFO:tensorflow:global_step/sec: 703.043
INFO:tensorflow:loss = 0.6676507, step = 6700 (0.143 sec)
INFO:tensorflow:global_step/sec: 766.872
INFO:tensorflow:loss = 0.53622645, step = 6800 (0.130 sec)
INFO:tensorflow:global_step/sec: 800.461
INFO:tensorflow:loss = 0.52648747, step = 6900 (0.125 sec)
INFO:tensorflow:global_step/sec: 829.606
INFO:tensorflow:loss = 0.53031015, step = 7000 (0.121 sec)
INFO:tensorflow:global_step/sec: 826.905
INFO:tensorflow:loss = 0.6168415, step = 7100 (0.121 sec)
INFO:tensorflow:global_step/sec: 822.393
INFO:tensorflow:loss = 0.70456946, step = 7200 (0.122 sec)
INFO:tensorflow:global_step/sec: 827.261
INFO:tensorflow:loss = 0.52576345, step = 7300 (0.121 sec)
INFO:tensorflow:global_step/sec: 746.53
INFO:tensorflow:loss = 0.6192044, step = 7400 (0.134 sec)
INFO:tensorflow:global_step/sec: 614.716
INFO:tensorflow:loss = 0.8814814, step = 7500 (0.163 sec)
INFO:tensorflow:global_step/sec: 648.058
INFO:tensorflow:loss = 0.31381485, step = 7600 (0.154 sec)
INFO:tensorflow:global_step/sec: 826.536
INFO:tensorflow:loss = 0.601199, step = 7700 (0.121 sec)
INFO:tensorflow:global_step/sec: 819.41
INFO:tensorflow:loss = 0.293849, step = 7800 (0.122 sec)
INFO:tensorflow:global_step/sec: 817.828
INFO:tensorflow:loss = 0.56127954, step = 7900 (0.122 sec)
INFO:tensorflow:global_step/sec: 823.011
INFO:tensorflow:loss = 0.70381105, step = 8000 (0.121 sec)
INFO:tensorflow:global_step/sec: 826.845
INFO:tensorflow:loss = 0.45825756, step = 8100 (0.121 sec)
INFO:tensorflow:global_step/sec: 824.388
INFO:tensorflow:loss = 0.37536147, step = 8200 (0.121 sec)
INFO:tensorflow:global_step/sec: 815.662
INFO:tensorflow:loss = 0.39761907, step = 8300 (0.123 sec)
INFO:tensorflow:global_step/sec: 824.279
INFO:tensorflow:loss = 0.5797437, step = 8400 (0.121 sec)
INFO:tensorflow:global_step/sec: 701.911
INFO:tensorflow:loss = 0.34954754, step = 8500 (0.142 sec)
INFO:tensorflow:global_step/sec: 696.544
INFO:tensorflow:loss = 0.41454393, step = 8600 (0.144 sec)
INFO:tensorflow:global_step/sec: 684.926
INFO:tensorflow:loss = 0.57426614, step = 8700 (0.146 sec)
INFO:tensorflow:global_step/sec: 797.927
INFO:tensorflow:loss = 0.6365391, step = 8800 (0.125 sec)
INFO:tensorflow:global_step/sec: 709.124
INFO:tensorflow:loss = 0.4363665, step = 8900 (0.141 sec)
INFO:tensorflow:global_step/sec: 750.869
```

```
INFO:tensorflow:loss = 0.52885354, step = 9000 (0.133 sec)
INFO:tensorflow:global_step/sec: 832.985
INFO:tensorflow:loss = 0.39617223, step = 9100 (0.120 sec)
INFO:tensorflow:global_step/sec: 650.546
INFO:tensorflow:loss = 0.5118426, step = 9200 (0.154 sec)
INFO:tensorflow:global_step/sec: 741.246
INFO:tensorflow:loss = 0.48267907, step = 9300 (0.135 sec)
INFO:tensorflow:global_step/sec: 822.032
INFO:tensorflow:loss = 0.3924963, step = 9400 (0.122 sec)
INFO:tensorflow:global_step/sec: 838.447
INFO:tensorflow:loss = 0.3332514, step = 9500 (0.119 sec)
INFO:tensorflow:global_step/sec: 688.085
INFO:tensorflow:loss = 0.58856046, step = 9600 (0.145 sec)
INFO:tensorflow:global_step/sec: 634.059
INFO:tensorflow:loss = 0.6562984, step = 9700 (0.158 sec)
INFO:tensorflow:global_step/sec: 782.693
INFO:tensorflow:loss = 0.44418252, step = 9800 (0.127 sec)
INFO:tensorflow:global_step/sec: 824.655
INFO:tensorflow:loss = 0.5847775, step = 9900 (0.122 sec)
INFO:tensorflow:global_step/sec: 836.804
INFO:tensorflow:loss = 0.42686737, step = 10000 (0.119 sec)
INFO:tensorflow:global_step/sec: 748.257
INFO:tensorflow:loss = 0.32733628, step = 10100 (0.134 sec)
INFO:tensorflow:global_step/sec: 660.672
INFO:tensorflow:loss = 0.56632674, step = 10200 (0.151 sec)
INFO:tensorflow:global_step/sec: 761.488
INFO:tensorflow:loss = 0.49848384, step = 10300 (0.131 sec)
INFO:tensorflow:global_step/sec: 793.865
INFO:tensorflow:loss = 0.68426013, step = 10400 (0.126 sec)
INFO:tensorflow:global_step/sec: 855.565
INFO:tensorflow:loss = 0.4996505, step = 10500 (0.117 sec)
INFO:tensorflow:global_step/sec: 852.66
INFO:tensorflow:loss = 0.4297498, step = 10600 (0.117 sec)
INFO:tensorflow:global_step/sec: 847.456
INFO:tensorflow:loss = 0.4665046, step = 10700 (0.118 sec)
INFO:tensorflow:global_step/sec: 772.631
INFO:tensorflow:loss = 0.50975925, step = 10800 (0.129 sec)
INFO:tensorflow:global_step/sec: 630.017
INFO:tensorflow:loss = 0.6955234, step = 10900 (0.159 sec)
INFO:tensorflow:global_step/sec: 635.861
INFO:tensorflow:loss = 0.94600385, step = 11000 (0.157 sec)
INFO:tensorflow:global_step/sec: 503.728
INFO:tensorflow:loss = 0.44106033, step = 11100 (0.198 sec)
INFO:tensorflow:global_step/sec: 686.093
INFO:tensorflow:loss = 0.4218407, step = 11200 (0.146 sec)
INFO:tensorflow:global_step/sec: 798.084
INFO:tensorflow:loss = 0.6690831, step = 11300 (0.125 sec)
INFO:tensorflow:global_step/sec: 803.039
```

```
INFO:tensorflow:loss = 0.45150742, step = 11400 (0.125 sec)
INFO:tensorflow:global_step/sec: 691.617
INFO:tensorflow:loss = 0.57715, step = 11500 (0.145 sec)
INFO:tensorflow:global_step/sec: 691.142
INFO:tensorflow:loss = 0.5889075, step = 11600 (0.145 sec)
INFO:tensorflow:global_step/sec: 799.788
INFO:tensorflow:loss = 0.39457008, step = 11700 (0.125 sec)
INFO:tensorflow:global_step/sec: 811.866
INFO:tensorflow:loss = 0.6509364, step = 11800 (0.123 sec)
INFO:tensorflow:global_step/sec: 821.411
INFO:tensorflow:loss = 0.36408097, step = 11900 (0.122 sec)
INFO:tensorflow:global_step/sec: 826.308
INFO:tensorflow:loss = 0.83028567, step = 12000 (0.121 sec)
INFO:tensorflow:global_step/sec: 813.584
INFO:tensorflow:loss = 0.3804531, step = 12100 (0.123 sec)
INFO:tensorflow:global_step/sec: 820.848
INFO:tensorflow:loss = 0.5118382, step = 12200 (0.122 sec)
INFO:tensorflow:global_step/sec: 819.398
INFO:tensorflow:loss = 0.47960237, step = 12300 (0.122 sec)
INFO:tensorflow:global_step/sec: 818.439
INFO:tensorflow:loss = 0.62231886, step = 12400 (0.122 sec)
INFO:tensorflow:global_step/sec: 817.579
INFO:tensorflow:loss = 0.783374, step = 12500 (0.122 sec)
INFO:tensorflow:global_step/sec: 814.599
INFO:tensorflow:loss = 0.59800744, step = 12600 (0.123 sec)
INFO:tensorflow:global_step/sec: 821.051
INFO:tensorflow:loss = 0.28488696, step = 12700 (0.122 sec)
INFO:tensorflow:global_step/sec: 819.956
INFO:tensorflow:loss = 0.4663884, step = 12800 (0.122 sec)
INFO:tensorflow:global_step/sec: 750.131
INFO:tensorflow:loss = 0.77658737, step = 12900 (0.133 sec)
INFO:tensorflow:global_step/sec: 693.24
INFO:tensorflow:loss = 0.6638115, step = 13000 (0.144 sec)
INFO:tensorflow:global_step/sec: 772.493
INFO:tensorflow:loss = 0.79575884, step = 13100 (0.129 sec)
INFO:tensorflow:global_step/sec: 829.552
INFO:tensorflow:loss = 0.45058846, step = 13200 (0.121 sec)
INFO:tensorflow:global_step/sec: 814.034
INFO:tensorflow:loss = 0.4012457, step = 13300 (0.123 sec)
INFO:tensorflow:global_step/sec: 804.009
INFO:tensorflow:loss = 0.52596235, step = 13400 (0.124 sec)
INFO:tensorflow:global_step/sec: 825.421
INFO:tensorflow:loss = 0.749497, step = 13500 (0.121 sec)
INFO:tensorflow:global_step/sec: 833.974
INFO:tensorflow:loss = 0.45175767, step = 13600 (0.120 sec)
INFO:tensorflow:global_step/sec: 826.474
INFO:tensorflow:loss = 0.4162001, step = 13700 (0.121 sec)
INFO:tensorflow:global_step/sec: 821.018
```

```
INFO:tensorflow:loss = 0.51919204, step = 13800 (0.122 sec)
INFO:tensorflow:global_step/sec: 824.539
INFO:tensorflow:loss = 0.4319097, step = 13900 (0.121 sec)
INFO:tensorflow:global_step/sec: 826.884
INFO:tensorflow:loss = 0.5157019, step = 14000 (0.121 sec)
INFO:tensorflow:global_step/sec: 825.082
INFO:tensorflow:loss = 0.45635998, step = 14100 (0.121 sec)
INFO:tensorflow:global_step/sec: 824.694
INFO:tensorflow:loss = 0.46995133, step = 14200 (0.121 sec)
INFO:tensorflow:global_step/sec: 821.105
INFO:tensorflow:loss = 0.8607532, step = 14300 (0.122 sec)
INFO:tensorflow:global_step/sec: 818.712
INFO:tensorflow:loss = 0.666293, step = 14400 (0.122 sec)
INFO:tensorflow:global_step/sec: 817.903
INFO:tensorflow:loss = 0.51491946, step = 14500 (0.122 sec)
INFO:tensorflow:global_step/sec: 822.646
INFO:tensorflow:loss = 0.26570186, step = 14600 (0.122 sec)
INFO:tensorflow:global_step/sec: 820.432
INFO:tensorflow:loss = 0.5641014, step = 14700 (0.122 sec)
INFO:tensorflow:global_step/sec: 823.153
INFO:tensorflow:loss = 0.52831537, step = 14800 (0.122 sec)
INFO:tensorflow:global_step/sec: 818.618
INFO:tensorflow:loss = 0.5613891, step = 14900 (0.122 sec)
INFO:tensorflow:global_step/sec: 814.942
INFO:tensorflow:loss = 0.38660866, step = 15000 (0.123 sec)
INFO:tensorflow:global_step/sec: 816.235
INFO:tensorflow:loss = 0.33790398, step = 15100 (0.123 sec)
INFO:tensorflow:global_step/sec: 825.878
INFO:tensorflow:loss = 0.41856375, step = 15200 (0.121 sec)
INFO:tensorflow:global_step/sec: 828.501
INFO:tensorflow:loss = 0.41428265, step = 15300 (0.121 sec)
INFO:tensorflow:global_step/sec: 822.674
INFO:tensorflow:loss = 0.52018285, step = 15400 (0.122 sec)
INFO:tensorflow:global_step/sec: 819.154
INFO:tensorflow:loss = 0.55119056, step = 15500 (0.122 sec)
INFO:tensorflow:global_step/sec: 826.439
INFO:tensorflow:loss = 0.4601345, step = 15600 (0.121 sec)
INFO:tensorflow:global_step/sec: 823.676
INFO:tensorflow:loss = 0.36738414, step = 15700 (0.121 sec)
INFO:tensorflow:global_step/sec: 824.336
INFO:tensorflow:loss = 0.4010563, step = 15800 (0.121 sec)
INFO:tensorflow:global_step/sec: 823.912
INFO:tensorflow:loss = 0.61679846, step = 15900 (0.121 sec)
INFO:tensorflow:global_step/sec: 816.066
INFO:tensorflow:loss = 0.5595817, step = 16000 (0.123 sec)
INFO:tensorflow:global_step/sec: 825.248
INFO:tensorflow:loss = 0.4606649, step = 16100 (0.121 sec)
INFO:tensorflow:global_step/sec: 815.034
```

```
INFO:tensorflow:loss = 0.38051102, step = 16200 (0.123 sec)
INFO:tensorflow:global_step/sec: 824.738
INFO:tensorflow:loss = 0.40295073, step = 16300 (0.121 sec)
INFO:tensorflow:global_step/sec: 834.256
INFO:tensorflow:loss = 0.4555317, step = 16400 (0.120 sec)
INFO:tensorflow:global_step/sec: 1160.85
INFO:tensorflow:loss = 0.37167668, step = 16500 (0.086 sec)
INFO:tensorflow:Calling checkpoint listeners before saving checkpoint 16512…
INFO:tensorflow:Saving checkpoints for 16512 into
/var/folders/q7/wrxzkb515gqcskvhd38dwx6h0000gn/T/tmpoo22j73g/model.ckpt.
INFO:tensorflow:Calling checkpoint listeners after saving checkpoint 16512…
INFO:tensorflow:Loss for final step: 0.7988467.
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from
/var/folders/q7/wrxzkb515gqcskvhd38dwx6h0000gn/T/tmpoo22j73g/model.ckpt-16512
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
Root Mean Squared Error (on training data): 69938.430
```

1.5 Exercise 5: Weights

The `LinearRegressor` builds a linear model with weights for each feature. Use the `get_variable_names` and `get_variable_value` methods to find the weights. Print the weights in a format similar to that shown below:

```
bias_weights 3.170546
population -12.792054
median_income 5.906482
total_bedrooms 5.3723865
households 4.3297663
longitude -3.7551448
latitude -3.533678
total_rooms -2.850763
housing_median_age 0.66154426
```

The columns are sorted by the relative impact to the formula (absolute value). Notice the `bias_weights` in the list. This is the constant bias and should go first in the list.

Student Solution

[ ]:
```
variable_names = linear_regressor.get_variable_names()
values = []



bias_weights = 'bias_weights'
#print(variable_names)
```

```python
variable_names_adjusted =[]




values = []
for i in variable_names:
  if i[20:-8] in numeric_feature_columns:
    variable_names_adjusted.append(i[20:-8])
    values.append(str(linear_regressor.get_variable_value(i)).replace(']',"").
 ↪replace('[',""))






values.insert(0,str(linear_regressor.get_variable_value('linear/linear_model/
 ↪bias_weights')).replace(']',"").replace('[',""))
variable_names_adjusted.insert(0,bias_weights)



for i in range(len(variable_names_adjusted)):
  print(variable_names_adjusted[i],values[i])





#print(variable_names_adjusted)
#string_values = []


#for i in values:
  #string_values.append(str(i).replace('[','').replace(']',''))

#print(string_values)


#print(values)
#print(values[0])

#print(dictionary)
```

```
bias_weights 1.0160959
households 2.5996149
housing_median_age 0.586927
latitude -1.4667833
```

```
longitude -1.6874348
median_income 5.6382713
population -9.6550255
total_bedrooms 4.092218
total_rooms -1.1015552
```