# Regression

*Supervised* machine learning techniques involve training a model to operate on a set of *features* and predict a *label* using a dataset that includes some already-known label values. The training process *fits* the features to the known labels to define a general function that can be applied to new features for which the labels are unknown, and predict them. You can think of this function like this, in which ***y*** represents the label we want to predict and ***x*** represents the features the model uses to predict it.

𝑦=𝑓(𝑥)y=f(x)

In most cases, *x* is actually a *vector* that consists of multiple feature values, so to be a little more precise, the function could be expressed like this:

𝑦=𝑓([𝑥1,𝑥2,𝑥3,...])y=f([x1,x2,x3,...])

The goal of training the model is to find a function that performs some kind of calculation to the *x* values that produces the result *y*. We do this by applying a machine learning *algorithm* that tries to fit the *x* values to a calculation that produces *y* reasonably accurately for all of the cases in the training dataset.

There are lots of machine learning algorithms for supervised learning, and they can be broadly divided into two types:

* ***Regression* algorithms**: Algorithms that predict a *y* value that is a numeric value, such as the price of a house or the number of sales transactions.
* ***Classification* algorithms**: Algorithms that predict to which category, or *class*, an observation belongs. The *y* value in a classification model is a vector of probability values between 0 and 1, one for each class, indicating the probability of the observation belonging to each class.

In this notebook, we'll focus on *regression*, using an example based on a real study in which data for a bicycle sharing scheme was collected and used to predict the number of rentals based on seasonality and weather conditions. We'll use a simplified version of the dataset from that study.

**Citation**: The data used in this exercise is derived from [Capital Bikeshare](https://www.capitalbikeshare.com/system-data) and is used in accordance with the published [license agreement](https://www.capitalbikeshare.com/data-license-agreement).

**Explore the Data**

The first step in any machine learning project is to explore the data that you will use to train a model. The goal of this exploration is to try to understand the relationships between its attributes; in particular, any apparent correlation between the *features* and the *label* your model will try to predict. This may require some work to detect and fix issues in the data (such as dealing with missing values, errors, or outlier values), deriving new feature columns by transforming or combining existing features (a process known as *feature engineering*), *normalizing* numeric features (values you can measure or count) so they're on a similar scale, and *encoding* categorical features (values that represent discrete categories) as numeric indicators.

Let's start by loading the bicycle sharing data as a **Pandas** DataFrame and viewing the first few rows.

In [1]:



**import** pandas **as** pd

​

*# load the training dataset*

bike\_data **=** pd.read\_csv('data/daily-bike-share.csv')

bike\_data.head()

Out[1]:

|  | **instant** | **dteday** | **season** | **yr** | **mnth** | **holiday** | **weekday** | **workingday** | **weathersit** | **temp** | **atemp** | **hum** | **windspeed** | **rentals** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1 | 1/1/2011 | 1 | 0 | 1 | 0 | 6 | 0 | 2 | 0.344167 | 0.363625 | 0.805833 | 0.160446 | 331 |
| **1** | 2 | 1/2/2011 | 1 | 0 | 1 | 0 | 0 | 0 | 2 | 0.363478 | 0.353739 | 0.696087 | 0.248539 | 131 |
| **2** | 3 | 1/3/2011 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0.196364 | 0.189405 | 0.437273 | 0.248309 | 120 |
| **3** | 4 | 1/4/2011 | 1 | 0 | 1 | 0 | 2 | 1 | 1 | 0.200000 | 0.212122 | 0.590435 | 0.160296 | 108 |
| **4** | 5 | 1/5/2011 | 1 | 0 | 1 | 0 | 3 | 1 | 1 | 0.226957 | 0.229270 | 0.436957 | 0.186900 | 82 |

The data consists of the following columns:

* **instant**: A unique row identifier
* **dteday**: The date on which the data was observed - in this case, the data was collected daily; so there's one row per date.
* **season**: A numerically encoded value indicating the season (1:spring, 2:summer, 3:fall, 4:winter)
* **yr**: The year of the study in which the observation was made (the study took place over two years - year 0 represents 2011, and year 1 represents 2012)
* **mnth**: The calendar month in which the observation was made (1:January ... 12:December)
* **holiday**: A binary value indicating whether or not the observation was made on a public holiday)
* **weekday**: The day of the week on which the observation was made (0:Sunday ... 6:Saturday)
* **workingday**: A binary value indicating whether or not the day is a working day (not a weekend or holiday)
* **weathersit**: A categorical value indicating the weather situation (1:clear, 2:mist/cloud, 3:light rain/snow, 4:heavy rain/hail/snow/fog)
* **temp**: The temperature in celsius (normalized)
* **atemp**: The apparent ("feels-like") temperature in celsius (normalized)
* **hum**: The humidity level (normalized)
* **windspeed**: The windspeed (normalized)
* **rentals**: The number of bicycle rentals recorded.

In this dataset, **rentals** represents the label (the *y* value) our model must be trained to predict. The other columns are potential features (*x* values).

As mentioned previously, you can perform some *feature engineering* to combine or derive new features. For example, let's add a new column named **day** to the dataframe by extracting the day component from the existing **dteday** column. The new column represents the day of the month from 1 to 31.

In [2]:



bike\_data['day'] **=** pd.DatetimeIndex(bike\_data['dteday']).day

bike\_data.head(32)

Out[2]:

|  | **instant** | **dteday** | **season** | **yr** | **mnth** | **holiday** | **weekday** | **workingday** | **weathersit** | **temp** | **atemp** | **hum** | **windspeed** | **rentals** | **day** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1 | 1/1/2011 | 1 | 0 | 1 | 0 | 6 | 0 | 2 | 0.344167 | 0.363625 | 0.805833 | 0.160446 | 331 | 1 |
| **1** | 2 | 1/2/2011 | 1 | 0 | 1 | 0 | 0 | 0 | 2 | 0.363478 | 0.353739 | 0.696087 | 0.248539 | 131 | 2 |
| **2** | 3 | 1/3/2011 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0.196364 | 0.189405 | 0.437273 | 0.248309 | 120 | 3 |
| **3** | 4 | 1/4/2011 | 1 | 0 | 1 | 0 | 2 | 1 | 1 | 0.200000 | 0.212122 | 0.590435 | 0.160296 | 108 | 4 |
| **4** | 5 | 1/5/2011 | 1 | 0 | 1 | 0 | 3 | 1 | 1 | 0.226957 | 0.229270 | 0.436957 | 0.186900 | 82 | 5 |
| **5** | 6 | 1/6/2011 | 1 | 0 | 1 | 0 | 4 | 1 | 1 | 0.204348 | 0.233209 | 0.518261 | 0.089565 | 88 | 6 |
| **6** | 7 | 1/7/2011 | 1 | 0 | 1 | 0 | 5 | 1 | 2 | 0.196522 | 0.208839 | 0.498696 | 0.168726 | 148 | 7 |
| **7** | 8 | 1/8/2011 | 1 | 0 | 1 | 0 | 6 | 0 | 2 | 0.165000 | 0.162254 | 0.535833 | 0.266804 | 68 | 8 |
| **8** | 9 | 1/9/2011 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0.138333 | 0.116175 | 0.434167 | 0.361950 | 54 | 9 |
| **9** | 10 | 1/10/2011 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0.150833 | 0.150888 | 0.482917 | 0.223267 | 41 | 10 |
| **10** | 11 | 1/11/2011 | 1 | 0 | 1 | 0 | 2 | 1 | 2 | 0.169091 | 0.191464 | 0.686364 | 0.122132 | 43 | 11 |
| **11** | 12 | 1/12/2011 | 1 | 0 | 1 | 0 | 3 | 1 | 1 | 0.172727 | 0.160473 | 0.599545 | 0.304627 | 25 | 12 |
| **12** | 13 | 1/13/2011 | 1 | 0 | 1 | 0 | 4 | 1 | 1 | 0.165000 | 0.150883 | 0.470417 | 0.301000 | 38 | 13 |
| **13** | 14 | 1/14/2011 | 1 | 0 | 1 | 0 | 5 | 1 | 1 | 0.160870 | 0.188413 | 0.537826 | 0.126548 | 54 | 14 |
| **14** | 15 | 1/15/2011 | 1 | 0 | 1 | 0 | 6 | 0 | 2 | 0.233333 | 0.248112 | 0.498750 | 0.157963 | 222 | 15 |
| **15** | 16 | 1/16/2011 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0.231667 | 0.234217 | 0.483750 | 0.188433 | 251 | 16 |
| **16** | 17 | 1/17/2011 | 1 | 0 | 1 | 1 | 1 | 0 | 2 | 0.175833 | 0.176771 | 0.537500 | 0.194017 | 117 | 17 |
| **17** | 18 | 1/18/2011 | 1 | 0 | 1 | 0 | 2 | 1 | 2 | 0.216667 | 0.232333 | 0.861667 | 0.146775 | 9 | 18 |
| **18** | 19 | 1/19/2011 | 1 | 0 | 1 | 0 | 3 | 1 | 2 | 0.292174 | 0.298422 | 0.741739 | 0.208317 | 78 | 19 |
| **19** | 20 | 1/20/2011 | 1 | 0 | 1 | 0 | 4 | 1 | 2 | 0.261667 | 0.255050 | 0.538333 | 0.195904 | 83 | 20 |
| **20** | 21 | 1/21/2011 | 1 | 0 | 1 | 0 | 5 | 1 | 1 | 0.177500 | 0.157833 | 0.457083 | 0.353242 | 75 | 21 |
| **21** | 22 | 1/22/2011 | 1 | 0 | 1 | 0 | 6 | 0 | 1 | 0.059130 | 0.079070 | 0.400000 | 0.171970 | 93 | 22 |
| **22** | 23 | 1/23/2011 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0.096522 | 0.098839 | 0.436522 | 0.246600 | 150 | 23 |
| **23** | 24 | 1/24/2011 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0.097391 | 0.117930 | 0.491739 | 0.158330 | 86 | 24 |
| **24** | 25 | 1/25/2011 | 1 | 0 | 1 | 0 | 2 | 1 | 2 | 0.223478 | 0.234526 | 0.616957 | 0.129796 | 186 | 25 |
| **25** | 26 | 1/26/2011 | 1 | 0 | 1 | 0 | 3 | 1 | 3 | 0.217500 | 0.203600 | 0.862500 | 0.293850 | 34 | 26 |
| **26** | 27 | 1/27/2011 | 1 | 0 | 1 | 0 | 4 | 1 | 1 | 0.195000 | 0.219700 | 0.687500 | 0.113837 | 15 | 27 |
| **27** | 28 | 1/28/2011 | 1 | 0 | 1 | 0 | 5 | 1 | 2 | 0.203478 | 0.223317 | 0.793043 | 0.123300 | 38 | 28 |
| **28** | 29 | 1/29/2011 | 1 | 0 | 1 | 0 | 6 | 0 | 1 | 0.196522 | 0.212126 | 0.651739 | 0.145365 | 123 | 29 |
| **29** | 30 | 1/30/2011 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0.216522 | 0.250322 | 0.722174 | 0.073983 | 140 | 30 |
| **30** | 31 | 1/31/2011 | 1 | 0 | 1 | 0 | 1 | 1 | 2 | 0.180833 | 0.186250 | 0.603750 | 0.187192 | 42 | 31 |
| **31** | 32 | 2/1/2011 | 1 | 0 | 2 | 0 | 2 | 1 | 2 | 0.192174 | 0.234530 | 0.829565 | 0.053213 | 47 | 1 |

OK, let's start our analysis of the data by examining a few key descriptive statistics. We can use the dataframe's **describe** method to generate these for the numeric features as well as the **rentals** label column.

In [3]:



numeric\_features **=** ['temp', 'atemp', 'hum', 'windspeed']

bike\_data[numeric\_features **+** ['rentals']].describe()

Out[3]:

|  | **temp** | **atemp** | **hum** | **windspeed** | **rentals** |
| --- | --- | --- | --- | --- | --- |
| **count** | 731.000000 | 731.000000 | 731.000000 | 731.000000 | 731.000000 |
| **mean** | 0.495385 | 0.474354 | 0.627894 | 0.190486 | 848.176471 |
| **std** | 0.183051 | 0.162961 | 0.142429 | 0.077498 | 686.622488 |
| **min** | 0.059130 | 0.079070 | 0.000000 | 0.022392 | 2.000000 |
| **25%** | 0.337083 | 0.337842 | 0.520000 | 0.134950 | 315.500000 |
| **50%** | 0.498333 | 0.486733 | 0.626667 | 0.180975 | 713.000000 |
| **75%** | 0.655417 | 0.608602 | 0.730209 | 0.233214 | 1096.000000 |
| **max** | 0.861667 | 0.840896 | 0.972500 | 0.507463 | 3410.000000 |

The statistics reveal some information about the distribution of the data in each of the numeric fields, including the number of observations (there are 731 records), the mean, standard deviation, minimum and maximum values, and the quartile values (the threshold values for 25%, 50% - which is also the median, and 75% of the data). From this, we can see that the mean number of daily rentals is around 848; but there's a comparatively large standard deviation, indicating a lot of variance in the number of rentals per day.

We might get a clearer idea of the distribution of rentals values by visualizing the data. Common plot types for visualizing numeric data distributions are *histograms* and *box plots*, so let's use Python's **matplotlib** library to create one of each of these for the **rentals** column.

In [4]:



**import** pandas **as** pd

**import** matplotlib.pyplot **as** plt

​

*# This ensures plots are displayed inline in the Jupyter notebook*

**%**matplotlib inline

​

*# Get the label column*

label **=** bike\_data['rentals']

​

​

*# Create a figure for 2 subplots (2 rows, 1 column)*

fig, ax **=** plt.subplots(2, 1, figsize **=** (9,12))

​

*# Plot the histogram*

ax[0].hist(label, bins**=**100)

ax[0].set\_ylabel('Frequency')

​

*# Add lines for the mean, median, and mode*

ax[0].axvline(label.mean(), color**=**'magenta', linestyle**=**'dashed', linewidth**=**2)

ax[0].axvline(label.median(), color**=**'cyan', linestyle**=**'dashed', linewidth**=**2)

​

*# Plot the boxplot*

ax[1].boxplot(label, vert**=False**)

ax[1].set\_xlabel('Rentals')

​

*# Add a title to the Figure*

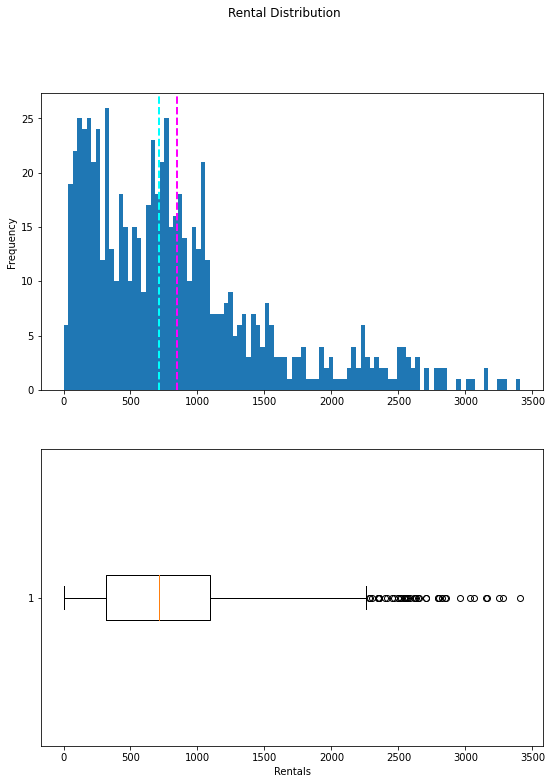
fig.suptitle('Rental Distribution')

​

*# Show the figure*

fig.show()

​



The plots show that the number of daily rentals ranges from 0 to just over 3,400. However, the mean (and median) number of daily rentals is closer to the low end of that range, with most of the data between 0 and around 2,200 rentals. The few values above this are shown in the box plot as small circles, indicating that they are *outliers* - in other words, unusually high or low values beyond the typical range of most of the data.

We can do the same kind of visual exploration of the numeric features. Let's create a histogram for each of these.

In [5]:



*# Plot a histogram for each numeric feature*

**for** col **in** numeric\_features:

fig **=** plt.figure(figsize**=**(9, 6))

ax **=** fig.gca()

feature **=** bike\_data[col]

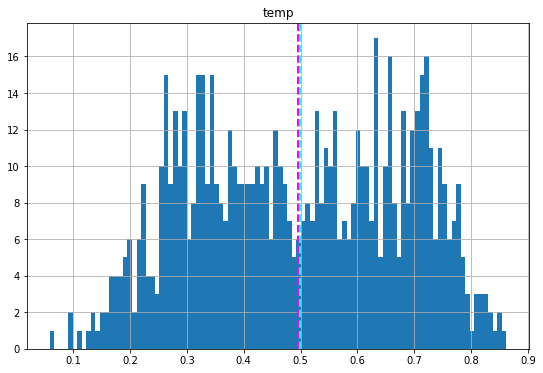
feature.hist(bins**=**100, ax **=** ax)

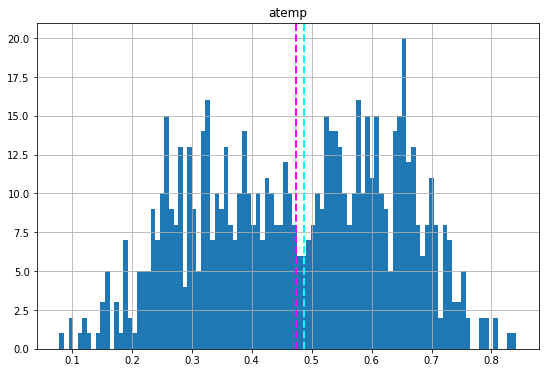
ax.axvline(feature.mean(), color**=**'magenta', linestyle**=**'dashed', linewidth**=**2)

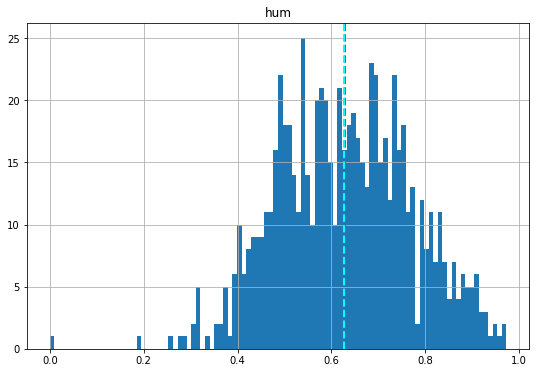
ax.axvline(feature.median(), color**=**'cyan', linestyle**=**'dashed', linewidth**=**2)

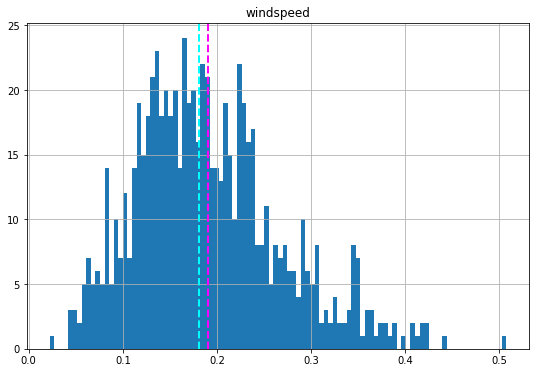
ax.set\_title(col)

plt.show()









The numeric features seem to be more *normally* distributed, with the mean and median nearer the middle of the range of values, coinciding with where the most commonly occurring values are.

**Note**: The distributions are not truly *normal* in the statistical sense, which would result in a smooth, symmetric "bell-curve" histogram with the mean and mode (the most common value) in the center; but they do generally indicate that most of the observations have a value somewhere near the middle.

We've explored the distribution of the numeric values in the dataset, but what about the categorical features? These aren't continuous numbers on a scale, so we can't use histograms; but we can plot a bar chart showing the count of each discrete value for each category.

In [6]:



**import** numpy **as** np

​

*# plot a bar plot for each categorical feature count*

categorical\_features **=** ['season','mnth','holiday','weekday','workingday','weathersit', 'day']

​

**for** col **in** categorical\_features:

counts **=** bike\_data[col].value\_counts().sort\_index()

fig **=** plt.figure(figsize**=**(9, 6))

ax **=** fig.gca()

counts.plot.bar(ax **=** ax, color**=**'steelblue')

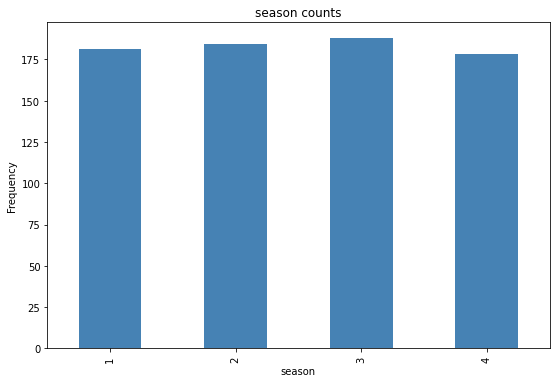
ax.set\_title(col **+** ' counts')

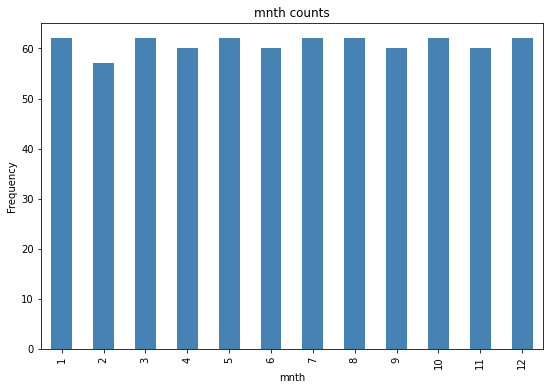
ax.set\_xlabel(col)

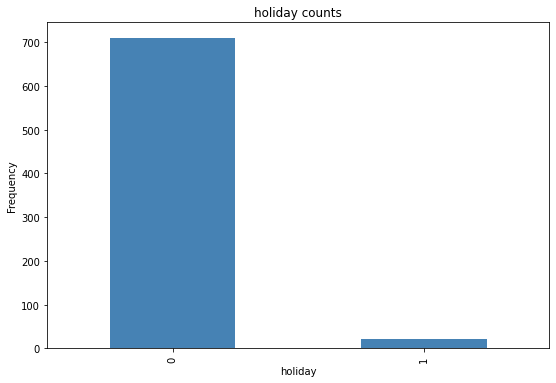
ax.set\_ylabel("Frequency")

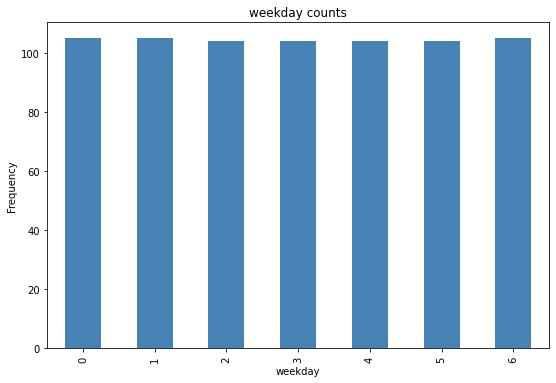
plt.show()

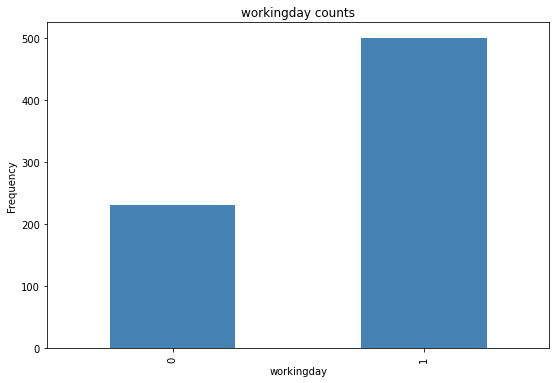
​

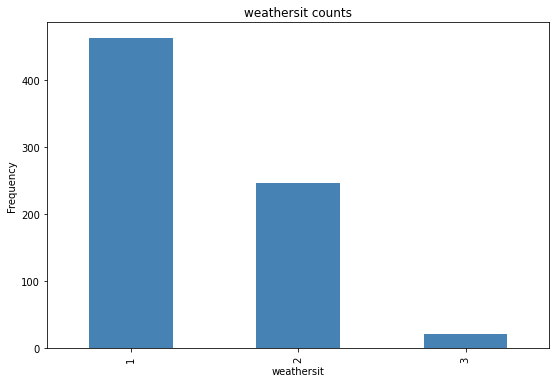


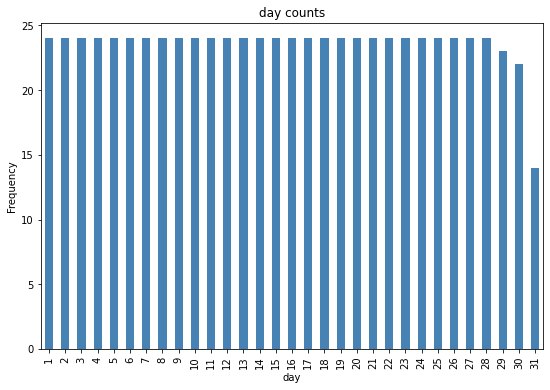












Many of the categorical features show a more or less *uniform* distribution (meaning there's roughly the same number of rows for each category). Exceptions to this include:

* **holiday**: There are many fewer days that are holidays than days that aren't.
* **workingday**: There are more working days than non-working days.
* **weathersit**: Most days are category *1* (clear), with category *2* (mist and cloud) the next most common. There are comparatively few category *3* (light rain or snow) days, and no category *4* (heavy rain, hail, or fog) days at all.

Now that we know something about the distribution of the data in our columns, we can start to look for relationships between the features and the **rentals** label we want to be able to predict.

For the numeric features, we can create scatter plots that show the intersection of feature and label values. We can also calculate the *correlation* statistic to quantify the apparent relationship..

In [7]:



**for** col **in** numeric\_features:

fig **=** plt.figure(figsize**=**(9, 6))

ax **=** fig.gca()

feature **=** bike\_data[col]

label **=** bike\_data['rentals']

correlation **=** feature.corr(label)

plt.scatter(x**=**feature, y**=**label)

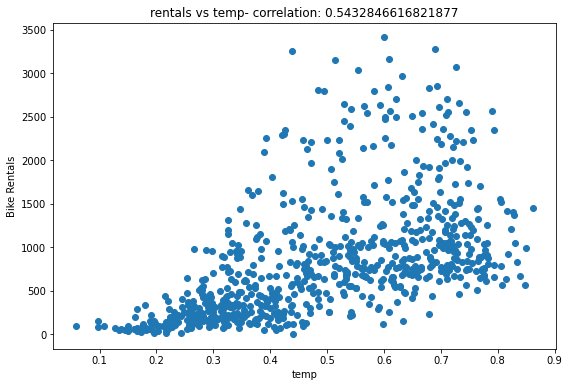
plt.xlabel(col)

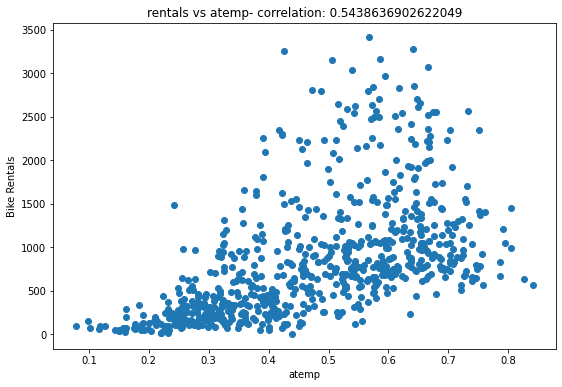
plt.ylabel('Bike Rentals')

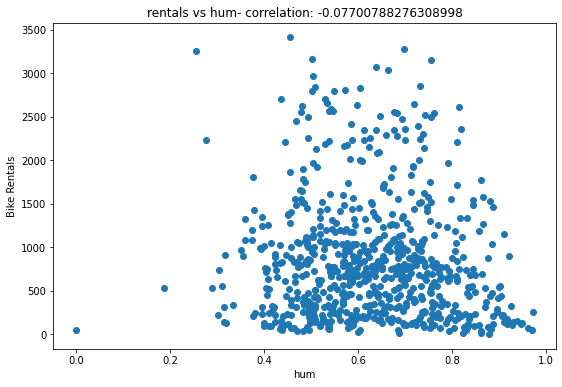
ax.set\_title('rentals vs ' **+** col **+** '- correlation: ' **+** str(correlation))

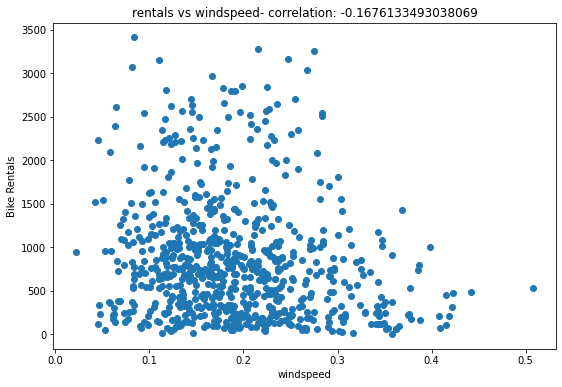
plt.show()

​









The results aren't conclusive, but if you look closely at the scatter plots for **temp** and **atemp**, you can see a vague diagonal trend showing that higher rental counts tend to coincide with higher temperatures; and a correlation value of just over 0.5 for both of these features supports this observation. Conversely, the plots for **hum** and **windspeed** show a slightly negative correlation, indicating that there are fewer rentals on days with high humidity or windspeed.

Now let's compare the categorical features to the label. We'll do this by creating box plots that show the distribution of rental counts for each category.

In [8]:



*# plot a boxplot for the label by each categorical feature*

**for** col **in** categorical\_features:

fig **=** plt.figure(figsize**=**(9, 6))

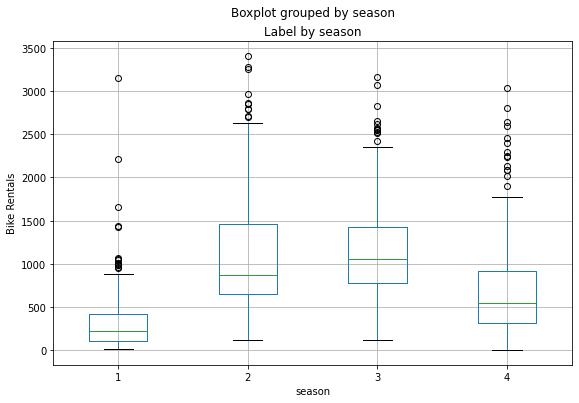
ax **=** fig.gca()

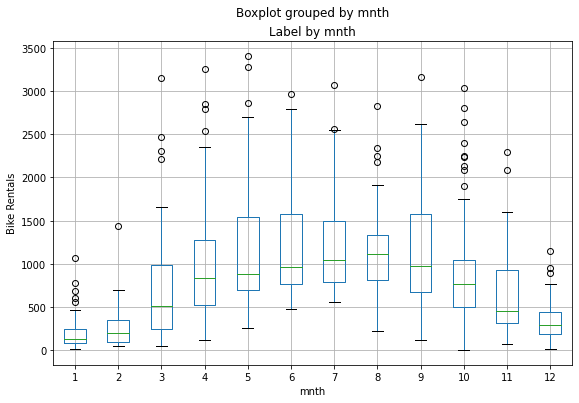
bike\_data.boxplot(column **=** 'rentals', by **=** col, ax **=** ax)

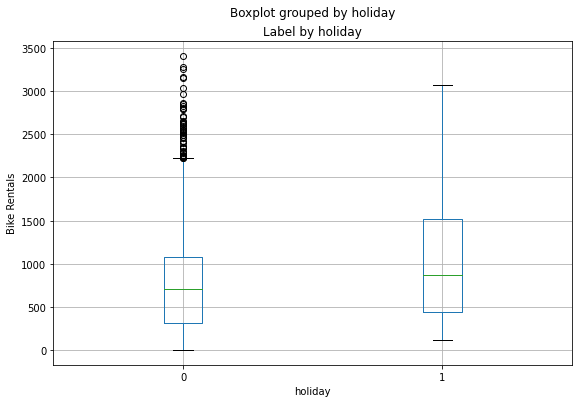
ax.set\_title('Label by ' **+** col)

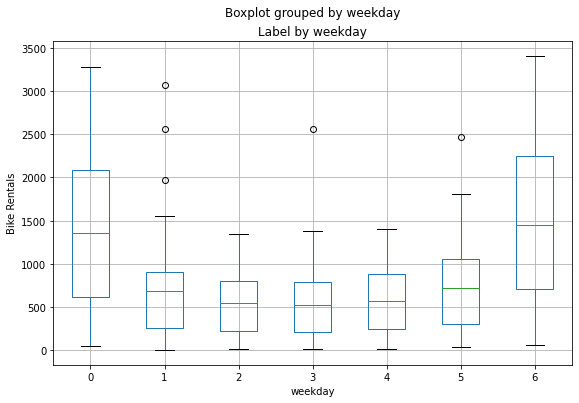
ax.set\_ylabel("Bike Rentals")

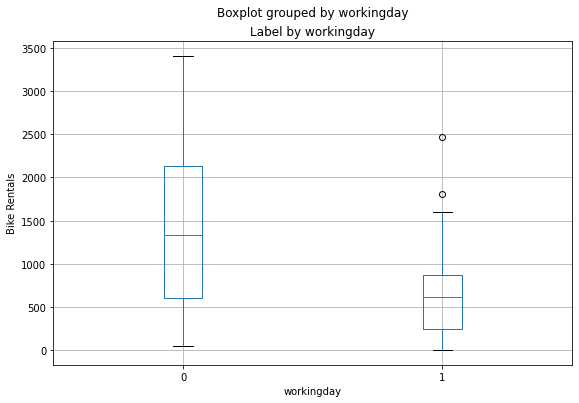
plt.show()

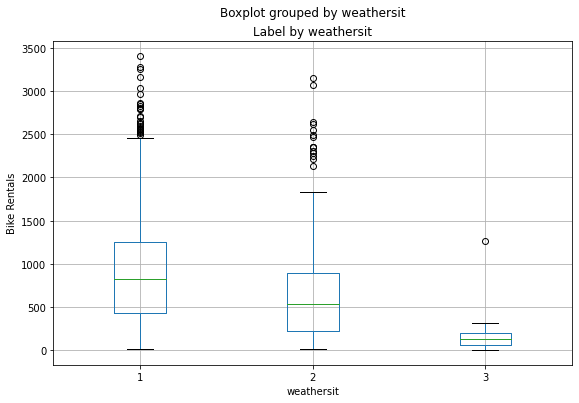


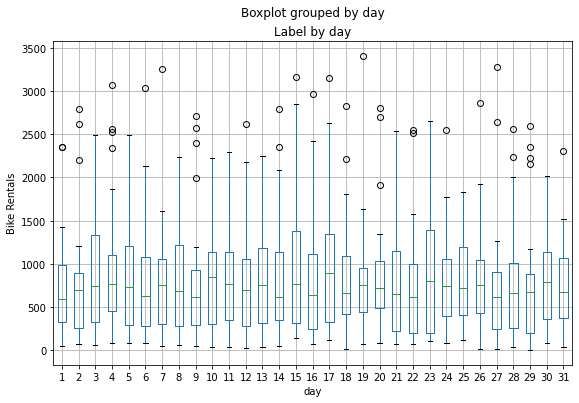












The plots show some variance in the relationship between some category values and rentals. For example, there's a clear difference in the distribution of rentals on weekends (**weekday** 0 or 6) and those during the working week (**weekday** 1 to 5). Similarly, there are notable differences for **holiday** and **workingday** categories. There's a noticeable trend that shows different rental distributions in summer and fall months compared to spring and winter months. The **weathersit** category also seems to make a difference in rental distribution. The **day** feature we created for the day of the month shows little variation, indicating that it's probably not predictive of the number of rentals.

**Train a Regression Model**

Now that we've explored the data, it's time to use it to train a regression model that uses the features we've identified as potentially predictive to predict the **rentals** label. The first thing we need to do is to separate the features we want to use to train the model from the label we want it to predict.

In [9]:



*# Separate features and labels*

X, y **=** bike\_data[['season','mnth', 'holiday','weekday','workingday','weathersit','temp', 'atemp', 'hum', 'windspeed']].values, bike\_data['rentals'].values

print('Features:',X[:10], '\nLabels:', y[:10], sep**=**'\n')

Features:

[[1. 1. 0. 6. 0. 2. 0.344167

0.363625 0.805833 0.160446 ]

[1. 1. 0. 0. 0. 2. 0.363478

0.353739 0.696087 0.248539 ]

[1. 1. 0. 1. 1. 1. 0.196364

0.189405 0.437273 0.248309 ]

[1. 1. 0. 2. 1. 1. 0.2

0.212122 0.590435 0.160296 ]

[1. 1. 0. 3. 1. 1. 0.226957

0.22927 0.436957 0.1869 ]

[1. 1. 0. 4. 1. 1. 0.204348

0.233209 0.518261 0.0895652]

[1. 1. 0. 5. 1. 2. 0.196522

0.208839 0.498696 0.168726 ]

[1. 1. 0. 6. 0. 2. 0.165

0.162254 0.535833 0.266804 ]

[1. 1. 0. 0. 0. 1. 0.138333

0.116175 0.434167 0.36195 ]

[1. 1. 0. 1. 1. 1. 0.150833

0.150888 0.482917 0.223267 ]]

Labels:

[331 131 120 108 82 88 148 68 54 41]

After separating the dataset, we now have numpy arrays named **X** containing the features, and **y** containing the labels.

We *could* train a model using all of the data; but it's common practice in supervised learning to split the data into two subsets; a (typically larger) set with which to train the model, and a smaller "hold-back" set with which to validate the trained model. This enables us to evaluate how well the model performs when used with the validation dataset by comparing the predicted labels to the known labels. It's important to split the data *randomly* (rather than say, taking the first 70% of the data for training and keeping the rest for validation). This helps ensure that the two subsets of data are statistically comparable (so we validate the model with data that has a similar statistical distribution to the data on which it was trained).

To randomly split the data, we'll use the **train\_test\_split** function in the **scikit-learn** library. This library is one of the most widely used machine learning packages for Python.

In [10]:



**from** sklearn.model\_selection **import** train\_test\_split

​

*# Split data 70%-30% into training set and test set*

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size**=**0.30, random\_state**=**0)

​

print ('Training Set: %d rows\nTest Set: %d rows' **%** (X\_train.shape[0], X\_test.shape[0]))

Training Set: 511 rows

Test Set: 220 rows

Now we have the following four datasets:

* **X\_train**: The feature values we'll use to train the model
* **y\_train**: The corresponding labels we'll use to train the model
* **X\_test**: The feature values we'll use to validate the model
* **y\_test**: The corresponding labels we'll use to validate the model

Now we're ready to train a model by fitting a suitable regression algorithm to the training data. We'll use a *linear regression* algorithm, a common starting point for regression that works by trying to find a linear relationship between the *X* values and the *y* label. The resulting model is a function that conceptually defines a line where every possible X and y value combination intersect.

In Scikit-Learn, training algorithms are encapsulated in *estimators*, and in this case we'll use the **LinearRegression** estimator to train a linear regression model.

In [11]:



*# Train the model*

**from** sklearn.linear\_model **import** LinearRegression

​

*# Fit a linear regression model on the training set*

model **=** LinearRegression().fit(X\_train, y\_train)

print (model)

LinearRegression(copy\_X=True, fit\_intercept=True, n\_jobs=None, normalize=False)

**Evaluate the Trained Model**

Now that we've trained the model, we can use it to predict rental counts for the features we held back in our validation dataset. Then we can compare these predictions to the actual label values to evaluate how well (or not!) the model is working.

In [12]:



**import** numpy **as** np

​

predictions **=** model.predict(X\_test)

np.set\_printoptions(suppress**=True**)

print('Predicted labels: ', np.round(predictions)[:10])

print('Actual labels : ' ,y\_test[:10])

Predicted labels: [1896. 1184. 1007. -28. 314. 385. 475. 590. 1476. -22.]

Actual labels : [2418 754 222 47 244 145 240 555 3252 38]

Comparing each prediction with its corresponding "ground truth" actual value isn't a very efficient way to determine how well the model is predicting. Let's see if we can get a better indication by visualizing a scatter plot that compares the predictions to the actual labels. We'll also overlay a trend line to get a general sense for how well the predicted labels align with the true labels.

In [13]:



**import** matplotlib.pyplot **as** plt

​

**%**matplotlib inline

​

plt.scatter(y\_test, predictions)

plt.xlabel('Actual Labels')

plt.ylabel('Predicted Labels')

plt.title('Daily Bike Share Predictions')

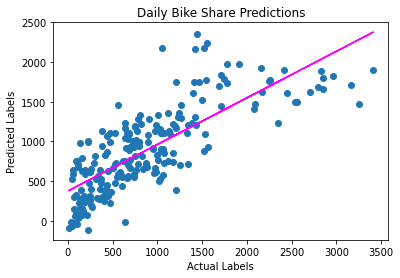
*# overlay the regression line*

z **=** np.polyfit(y\_test, predictions, 1)

p **=** np.poly1d(z)

plt.plot(y\_test,p(y\_test), color**=**'magenta')

plt.show()



There's a definite diagonal trend, and the intersections of the predicted and actual values are generally following the path of the trend line; but there's a fair amount of difference between the ideal function represented by the line and the results. This variance represents the *residuals* of the model - in other words, the difference between the label predicted when the model applies the coefficients it learned during training to the validation data, and the actual value of the validation label. These residuals when evaluated from the validation data indicate the expected level of *error* when the model is used with new data for which the label is unknown.

You can quantify the residuals by calculating a number of commonly used evaluation metrics. We'll focus on the following three:

* **Mean Square Error (MSE)**: The mean of the squared differences between predicted and actual values. This yields a relative metric in which the smaller the value, the better the fit of the model
* **Root Mean Square Error (RMSE)**: The square root of the MSE. This yields an absolute metric in the same unit as the label (in this case, numbers of rentals). The smaller the value, the better the model (in a simplistic sense, it represents the average number of rentals by which the predictions are wrong!)
* **Coefficient of Determination (usually known as *R-squared* or R2**: A relative metric in which the higher the value, the better the fit of the model. In essence, this metric represents how much of the variance between predicted and actual label values the model is able to explain.

**Note**: You can find out more about these and other metrics for evaluating regression models in the [Scikit-Learn documentation](https://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics)

Let's use Scikit-Learn to calculate these metrics for our model, based on the predictions it generated for the validation data.

In [14]:



**from** sklearn.metrics **import** mean\_squared\_error, r2\_score

​

mse **=** mean\_squared\_error(y\_test, predictions)

print("MSE:", mse)

​

rmse **=** np.sqrt(mse)

print("RMSE:", rmse)

​

r2 **=** r2\_score(y\_test, predictions)

print("R2:", r2)

MSE: 201972.55947035592

RMSE: 449.4135728595165

R2: 0.6040454736919191

So now we've quantified the ability of our model to predict the number of rentals. It definitely has *some* predictive power, but we can probably do better!

**Experiment with Algorithms**

The linear regression algorithm we used to train the model has some predictive capability, but there are many kinds of regression algorithm we could try, including:

* **Linear algorithms**: Not just the Linear Regression algorithm we used above (which is technically an *Ordinary Least Squares* algorithm), but other variants such as *Lasso* and *Ridge*.
* **Tree-based algorithms**: Algorithms that build a decision tree to reach a prediction.
* **Ensemble algorithms**: Algorithms that combine the outputs of multiple base algorithms to improve generalizability.

**Note**: For a full list of Scikit-Learn estimators that encapsulate algorithms for supervised machine learning, see the [Scikit-Learn documentation](https://scikit-learn.org/stable/supervised_learning.html). There are many algorithms to choose from, but for most real-world scenarios, the [Scikit-Learn estimator cheat sheet](https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html) can help you find a suitable starting point.

**Try Another Linear Algorithm**

Let's try training our regression model by using a **Lasso** algorithm. We can do this by just changing the estimator in the training code.

In [15]:



**from** sklearn.linear\_model **import** Lasso

​

*# Fit a lasso model on the training set*

model **=** Lasso().fit(X\_train, y\_train)

print (model, "\n")

​

*# Evaluate the model using the test data*

predictions **=** model.predict(X\_test)

mse **=** mean\_squared\_error(y\_test, predictions)

print("MSE:", mse)

rmse **=** np.sqrt(mse)

print("RMSE:", rmse)

r2 **=** r2\_score(y\_test, predictions)

print("R2:", r2)

​

*# Plot predicted vs actual*

plt.scatter(y\_test, predictions)

plt.xlabel('Actual Labels')

plt.ylabel('Predicted Labels')

plt.title('Daily Bike Share Predictions')

*# overlay the regression line*

z **=** np.polyfit(y\_test, predictions, 1)

p **=** np.poly1d(z)

plt.plot(y\_test,p(y\_test), color**=**'magenta')

plt.show()

Lasso(alpha=1.0, copy\_X=True, fit\_intercept=True, max\_iter=1000,

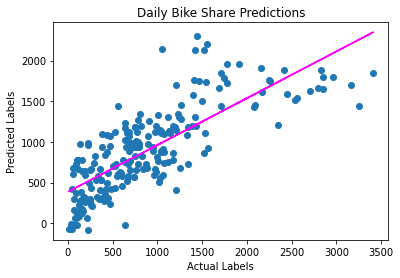
normalize=False, positive=False, precompute=False, random\_state=None,

selection='cyclic', tol=0.0001, warm\_start=False)

MSE: 201155.70593338404

RMSE: 448.5038527519959

R2: 0.6056468637824488



**Try a Decision Tree Algorithm**

As an alternative to a linear model, there's a category of algorithms for machine learning that uses a tree-based approach in which the features in the dataset are examined in a series of evaluations, each of which results in a *branch* in a *decision tree* based on the feature value. At the end of each series of branches are leaf-nodes with the predicted label value based on the feature values.

It's easiest to see how this works with an example. Let's train a Decision Tree regression model using the bike rental data. After training the model, the code below will print the model definition and a text representation of the tree it uses to predict label values.

In [16]:



**from** sklearn.tree **import** DecisionTreeRegressor

**from** sklearn.tree **import** export\_text

​

*# Train the model*

model **=** DecisionTreeRegressor().fit(X\_train, y\_train)

print (model, "\n")

​

*# Visualize the model tree*

tree **=** export\_text(model)

print(tree)

DecisionTreeRegressor(ccp\_alpha=0.0, criterion='mse', max\_depth=None,

max\_features=None, max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, presort='deprecated',

random\_state=None, splitter='best')

|--- feature\_6 <= 0.45

| |--- feature\_4 <= 0.50

| | |--- feature\_7 <= 0.32

| | | |--- feature\_8 <= 0.41

| | | | |--- feature\_1 <= 2.50

| | | | | |--- feature\_3 <= 3.00

| | | | | | |--- feature\_8 <= 0.36

| | | | | | | |--- value: [558.00]

| | | | | | |--- feature\_8 > 0.36

| | | | | | | |--- value: [515.00]

| | | | | |--- feature\_3 > 3.00

| | | | | | |--- value: [317.00]

| | | | |--- feature\_1 > 2.50

| | | | | |--- feature\_9 <= 0.28

| | | | | | |--- feature\_8 <= 0.37

| | | | | | | |--- value: [968.00]

| | | | | | |--- feature\_8 > 0.37

| | | | | | | |--- value: [981.00]

| | | | | |--- feature\_9 > 0.28

| | | | | | |--- feature\_1 <= 7.00

| | | | | | | |--- value: [710.00]

| | | | | | |--- feature\_1 > 7.00

| | | | | | | |--- value: [532.00]

| | | |--- feature\_8 > 0.41

| | | | |--- feature\_7 <= 0.25

| | | | | |--- feature\_7 <= 0.18

| | | | | | |--- feature\_8 <= 0.43

| | | | | | | |--- value: [284.00]

| | | | | | |--- feature\_8 > 0.43

| | | | | | | |--- feature\_6 <= 0.11

| | | | | | | | |--- value: [150.00]

| | | | | | | |--- feature\_6 > 0.11

| | | | | | | | |--- feature\_7 <= 0.18

| | | | | | | | | |--- feature\_5 <= 1.50

| | | | | | | | | | |--- value: [73.00]

| | | | | | | | | |--- feature\_5 > 1.50

| | | | | | | | | | |--- feature\_6 <= 0.17

| | | | | | | | | | | |--- value: [68.00]

| | | | | | | | | | |--- feature\_6 > 0.17

| | | | | | | | | | | |--- value: [67.00]

| | | | | | | | |--- feature\_7 > 0.18

| | | | | | | | | |--- value: [117.00]

| | | | | |--- feature\_7 > 0.18

| | | | | | |--- feature\_9 <= 0.17

| | | | | | | |--- feature\_9 <= 0.11

| | | | | | | | |--- value: [140.00]

| | | | | | | |--- feature\_9 > 0.11

| | | | | | | | |--- value: [123.00]

| | | | | | |--- feature\_9 > 0.17

| | | | | | | |--- feature\_9 <= 0.19

| | | | | | | | |--- value: [333.00]

| | | | | | | |--- feature\_9 > 0.19

| | | | | | | | |--- feature\_8 <= 0.53

| | | | | | | | | |--- feature\_3 <= 0.50

| | | | | | | | | | |--- value: [251.00]

| | | | | | | | | |--- feature\_3 > 0.50

| | | | | | | | | | |--- feature\_2 <= 0.50

| | | | | | | | | | | |--- value: [205.00]

| | | | | | | | | | |--- feature\_2 > 0.50

| | | | | | | | | | | |--- value: [217.00]

| | | | | | | | |--- feature\_8 > 0.53

| | | | | | | | | |--- feature\_1 <= 7.00

| | | | | | | | | | |--- value: [288.00]

| | | | | | | | | |--- feature\_1 > 7.00

| | | | | | | | | | |--- value: [275.00]

| | | | |--- feature\_7 > 0.25

| | | | | |--- feature\_9 <= 0.11

| | | | | | |--- value: [706.00]

| | | | | |--- feature\_9 > 0.11

| | | | | | |--- feature\_8 <= 0.54

| | | | | | | |--- feature\_5 <= 1.50

| | | | | | | | |--- feature\_7 <= 0.26

| | | | | | | | | |--- value: [309.00]

| | | | | | | | |--- feature\_7 > 0.26

| | | | | | | | | |--- feature\_0 <= 2.50

| | | | | | | | | | |--- feature\_6 <= 0.26

| | | | | | | | | | | |--- value: [408.00]

| | | | | | | | | | |--- feature\_6 > 0.26

| | | | | | | | | | | |--- truncated branch of depth 2

| | | | | | | | | |--- feature\_0 > 2.50

| | | | | | | | | | |--- feature\_1 <= 11.50

| | | | | | | | | | | |--- value: [440.00]

| | | | | | | | | | |--- feature\_1 > 11.50

| | | | | | | | | | | |--- value: [502.00]

| | | | | | | |--- feature\_5 > 1.50

| | | | | | | | |--- value: [618.00]

| | | | | | |--- feature\_8 > 0.54

| | | | | | | |--- feature\_6 <= 0.29

| | | | | | | | |--- feature\_6 <= 0.28

| | | | | | | | | |--- value: [318.00]

| | | | | | | | |--- feature\_6 > 0.28

| | | | | | | | | |--- value: [354.00]

| | | | | | | |--- feature\_6 > 0.29

| | | | | | | | |--- feature\_7 <= 0.29

| | | | | | | | | |--- value: [195.00]

| | | | | | | | |--- feature\_7 > 0.29

| | | | | | | | | |--- value: [155.00]

| | |--- feature\_7 > 0.32

| | | |--- feature\_9 <= 0.25

| | | | |--- feature\_6 <= 0.37

| | | | | |--- feature\_7 <= 0.36

| | | | | | |--- feature\_6 <= 0.36

| | | | | | | |--- feature\_1 <= 10.50

| | | | | | | | |--- feature\_6 <= 0.34

| | | | | | | | | |--- feature\_9 <= 0.21

| | | | | | | | | | |--- feature\_9 <= 0.20

| | | | | | | | | | | |--- truncated branch of depth 2

| | | | | | | | | | |--- feature\_9 > 0.20

| | | | | | | | | | | |--- value: [1047.00]

| | | | | | | | | |--- feature\_9 > 0.21

| | | | | | | | | | |--- value: [724.00]

| | | | | | | | |--- feature\_6 > 0.34

| | | | | | | | | |--- feature\_8 <= 0.83

| | | | | | | | | | |--- feature\_9 <= 0.16

| | | | | | | | | | | |--- value: [694.00]

| | | | | | | | | | |--- feature\_9 > 0.16

| | | | | | | | | | | |--- truncated branch of depth 2

| | | | | | | | | |--- feature\_8 > 0.83

| | | | | | | | | | |--- value: [879.00]

| | | | | | | |--- feature\_1 > 10.50

| | | | | | | | |--- feature\_8 <= 0.57

| | | | | | | | | |--- feature\_8 <= 0.51

| | | | | | | | | | |--- value: [943.00]

| | | | | | | | | |--- feature\_8 > 0.51

| | | | | | | | | | |--- feature\_8 <= 0.53

| | | | | | | | | | | |--- value: [1156.00]

| | | | | | | | | | |--- feature\_8 > 0.53

| | | | | | | | | | | |--- truncated branch of depth 2

| | | | | | | | |--- feature\_8 > 0.57

| | | | | | | | | |--- feature\_9 <= 0.10

| | | | | | | | | | |--- feature\_9 <= 0.06

| | | | | | | | | | | |--- value: [955.00]

| | | | | | | | | | |--- feature\_9 > 0.06

| | | | | | | | | | | |--- truncated branch of depth 2

| | | | | | | | | |--- feature\_9 > 0.10

| | | | | | | | | | |--- feature\_3 <= 3.00

| | | | | | | | | | | |--- value: [922.00]

| | | | | | | | | | |--- feature\_3 > 3.00

| | | | | | | | | | | |--- value: [767.00]

| | | | | | |--- feature\_6 > 0.36

| | | | | | | |--- value: [1658.00]

| | | | | |--- feature\_7 > 0.36

| | | | | | |--- feature\_0 <= 2.50

| | | | | | | |--- value: [331.00]

| | | | | | |--- feature\_0 > 2.50

| | | | | | | |--- feature\_1 <= 11.50

| | | | | | | | |--- value: [560.00]

| | | | | | | |--- feature\_1 > 11.50

| | | | | | | | |--- value: [538.00]

| | | | |--- feature\_6 > 0.37

| | | | | |--- feature\_9 <= 0.24

| | | | | | |--- feature\_9 <= 0.15

| | | | | | | |--- feature\_8 <= 0.67

| | | | | | | | |--- feature\_7 <= 0.41

| | | | | | | | | |--- value: [2252.00]

| | | | | | | | |--- feature\_7 > 0.41

| | | | | | | | | |--- value: [2290.00]

| | | | | | | |--- feature\_8 > 0.67

| | | | | | | | |--- feature\_7 <= 0.41

| | | | | | | | | |--- feature\_9 <= 0.08

| | | | | | | | | | |--- value: [1249.00]

| | | | | | | | | |--- feature\_9 > 0.08

| | | | | | | | | | |--- value: [1153.00]

| | | | | | | | |--- feature\_7 > 0.41

| | | | | | | | | |--- value: [1619.00]

| | | | | | |--- feature\_9 > 0.15

| | | | | | | |--- feature\_7 <= 0.38

| | | | | | | | |--- value: [1651.00]

| | | | | | | |--- feature\_7 > 0.38

| | | | | | | | |--- feature\_9 <= 0.20

| | | | | | | | | |--- feature\_8 <= 0.78

| | | | | | | | | | |--- feature\_1 <= 2.00

| | | | | | | | | | | |--- value: [1070.00]

| | | | | | | | | | |--- feature\_1 > 2.00

| | | | | | | | | | | |--- truncated branch of depth 2

| | | | | | | | | |--- feature\_8 > 0.78

| | | | | | | | | | |--- value: [1188.00]

| | | | | | | | |--- feature\_9 > 0.20

| | | | | | | | | |--- feature\_7 <= 0.43

| | | | | | | | | | |--- value: [665.00]

| | | | | | | | | |--- feature\_7 > 0.43

| | | | | | | | | | |--- value: [642.00]

| | | | | |--- feature\_9 > 0.24

| | | | | | |--- value: [2301.00]

| | | |--- feature\_9 > 0.25

| | | | |--- feature\_5 <= 2.50

| | | | | |--- feature\_8 <= 0.49

| | | | | | |--- value: [397.00]

| | | | | |--- feature\_8 > 0.49

| | | | | | |--- feature\_5 <= 1.50

| | | | | | | |--- value: [982.00]

| | | | | | |--- feature\_5 > 1.50

| | | | | | | |--- feature\_1 <= 6.50

| | | | | | | | |--- value: [640.00]

| | | | | | | |--- feature\_1 > 6.50

| | | | | | | | |--- value: [480.00]

| | | | |--- feature\_5 > 2.50

| | | | | |--- feature\_3 <= 3.00

| | | | | | |--- value: [120.00]

| | | | | |--- feature\_3 > 3.00

| | | | | | |--- value: [121.00]

| |--- feature\_4 > 0.50

| | |--- feature\_6 <= 0.34

| | | |--- feature\_1 <= 2.50

| | | | |--- feature\_7 <= 0.29

| | | | | |--- feature\_6 <= 0.19

| | | | | | |--- feature\_7 <= 0.14

| | | | | | | |--- feature\_8 <= 0.43

| | | | | | | | |--- value: [95.00]

| | | | | | | |--- feature\_8 > 0.43

| | | | | | | | |--- feature\_9 <= 0.26

| | | | | | | | | |--- value: [86.00]

| | | | | | | | |--- feature\_9 > 0.26

| | | | | | | | | |--- value: [89.00]

| | | | | | |--- feature\_7 > 0.14

| | | | | | | |--- feature\_8 <= 0.46

| | | | | | | | |--- feature\_7 <= 0.17

| | | | | | | | | |--- value: [75.00]

| | | | | | | | |--- feature\_7 > 0.17

| | | | | | | | | |--- value: [61.00]

| | | | | | | |--- feature\_8 > 0.46

| | | | | | | | |--- feature\_9 <= 0.30

| | | | | | | | | |--- feature\_8 <= 0.49

| | | | | | | | | | |--- feature\_8 <= 0.48

| | | | | | | | | | | |--- value: [38.00]

| | | | | | | | | | |--- feature\_8 > 0.48

| | | | | | | | | | | |--- value: [41.00]

| | | | | | | | | |--- feature\_8 > 0.49

| | | | | | | | | | |--- feature\_3 <= 2.00

| | | | | | | | | | | |--- value: [42.00]

| | | | | | | | | | |--- feature\_3 > 2.00

| | | | | | | | | | | |--- truncated branch of depth 2

| | | | | | | | |--- feature\_9 > 0.30

| | | | | | | | | |--- value: [25.00]

| | | | | |--- feature\_6 > 0.19

| | | | | | |--- feature\_3 <= 4.50

| | | | | | | |--- feature\_9 <= 0.26

| | | | | | | | |--- feature\_3 <= 2.50

| | | | | | | | | |--- feature\_8 <= 0.60

| | | | | | | | | | |--- feature\_9 <= 0.19

| | | | | | | | | | | |--- truncated branch of depth 2

| | | | | | | | | | |--- feature\_9 > 0.19

| | | | | | | | | | | |--- truncated branch of depth 2

| | | | | | | | | |--- feature\_8 > 0.60

| | | | | | | | | | |--- feature\_3 <= 1.50

| | | | | | | | | | | |--- truncated branch of depth 2

| | | | | | | | | | |--- feature\_3 > 1.50

| | | | | | | | | | | |--- value: [186.00]

| | | | | | | | |--- feature\_3 > 2.50

| | | | | | | | | |--- feature\_8 <= 0.51

| | | | | | | | | | |--- feature\_6 <= 0.22

| | | | | | | | | | | |--- truncated branch of depth 2

| | | | | | | | | | |--- feature\_6 > 0.22

| | | | | | | | | | | |--- value: [82.00]

| | | | | | | | | |--- feature\_8 > 0.51

| | | | | | | | | | |--- feature\_7 <= 0.23

| | | | | | | | | | | |--- value: [15.00]

| | | | | | | | | | |--- feature\_7 > 0.23

| | | | | | | | | | | |--- truncated branch of depth 4

| | | | | | | |--- feature\_9 > 0.26

| | | | | | | | |--- feature\_1 <= 1.50

| | | | | | | | | |--- value: [34.00]

| | | | | | | | |--- feature\_1 > 1.50

| | | | | | | | | |--- feature\_7 <= 0.23

| | | | | | | | | | |--- value: [64.00]

| | | | | | | | | |--- feature\_7 > 0.23

| | | | | | | | | | |--- value: [72.00]

| | | | | | |--- feature\_3 > 4.50

| | | | | | | |--- feature\_8 <= 0.47

| | | | | | | | |--- value: [115.00]

| | | | | | | |--- feature\_8 > 0.47

| | | | | | | | |--- feature\_8 <= 0.51

| | | | | | | | | |--- feature\_7 <= 0.21

| | | | | | | | | | |--- value: [148.00]

| | | | | | | | | |--- feature\_7 > 0.21

| | | | | | | | | | |--- value: [149.00]

| | | | | | | | |--- feature\_8 > 0.51

| | | | | | | | | |--- value: [174.00]

| | | | |--- feature\_7 > 0.29

| | | | | |--- feature\_3 <= 4.50

| | | | | | |--- feature\_5 <= 1.50

| | | | | | | |--- feature\_9 <= 0.23

| | | | | | | | |--- feature\_3 <= 1.50

| | | | | | | | | |--- value: [206.00]

| | | | | | | | |--- feature\_3 > 1.50

| | | | | | | | | |--- feature\_8 <= 0.62

| | | | | | | | | | |--- value: [163.00]

| | | | | | | | | |--- feature\_8 > 0.62

| | | | | | | | | | |--- value: [173.00]

| | | | | | | |--- feature\_9 > 0.23

| | | | | | | | |--- value: [218.00]

| | | | | | |--- feature\_5 > 1.50

| | | | | | | |--- feature\_9 <= 0.12

| | | | | | | | |--- value: [74.00]

| | | | | | | |--- feature\_9 > 0.12

| | | | | | | | |--- value: [135.00]

| | | | | |--- feature\_3 > 4.50

| | | | | | |--- feature\_7 <= 0.30

| | | | | | | |--- value: [227.00]

| | | | | | |--- feature\_7 > 0.30

| | | | | | | |--- feature\_6 <= 0.32

| | | | | | | | |--- value: [310.00]

| | | | | | | |--- feature\_6 > 0.32

| | | | | | | | |--- value: [307.00]

| | | |--- feature\_1 > 2.50

| | | | |--- feature\_9 <= 0.20

| | | | | |--- feature\_9 <= 0.12

| | | | | | |--- feature\_9 <= 0.06

| | | | | | | |--- feature\_9 <= 0.05

| | | | | | | | |--- value: [337.00]

| | | | | | | |--- feature\_9 > 0.05

| | | | | | | | |--- value: [362.00]

| | | | | | |--- feature\_9 > 0.06

| | | | | | | |--- feature\_3 <= 3.50

| | | | | | | | |--- feature\_5 <= 1.50

| | | | | | | | | |--- value: [143.00]

| | | | | | | | |--- feature\_5 > 1.50

| | | | | | | | | |--- feature\_0 <= 2.50

| | | | | | | | | | |--- value: [174.00]

| | | | | | | | | |--- feature\_0 > 2.50

| | | | | | | | | | |--- value: [178.00]

| | | | | | | |--- feature\_3 > 3.50

| | | | | | | | |--- feature\_7 <= 0.31

| | | | | | | | | |--- feature\_9 <= 0.12

| | | | | | | | | | |--- value: [243.00]

| | | | | | | | | |--- feature\_9 > 0.12

| | | | | | | | | | |--- value: [254.00]

| | | | | | | | |--- feature\_7 > 0.31

| | | | | | | | | |--- feature\_7 <= 0.32

| | | | | | | | | | |--- value: [261.00]

| | | | | | | | | |--- feature\_7 > 0.32

| | | | | | | | | | |--- value: [268.00]

| | | | | |--- feature\_9 > 0.12

| | | | | | |--- feature\_8 <= 0.64

| | | | | | | |--- feature\_8 <= 0.45

| | | | | | | | |--- feature\_1 <= 7.00

| | | | | | | | | |--- value: [316.00]

| | | | | | | | |--- feature\_1 > 7.00

| | | | | | | | | |--- value: [245.00]

| | | | | | | |--- feature\_8 > 0.45

| | | | | | | | |--- feature\_3 <= 4.50

| | | | | | | | | |--- feature\_7 <= 0.30

| | | | | | | | | | |--- feature\_9 <= 0.17

| | | | | | | | | | | |--- truncated branch of depth 3

| | | | | | | | | | |--- feature\_9 > 0.17

| | | | | | | | | | | |--- truncated branch of depth 2

| | | | | | | | | |--- feature\_7 > 0.30

| | | | | | | | | | |--- feature\_1 <= 7.00

| | | | | | | | | | | |--- value: [359.00]

| | | | | | | | | | |--- feature\_1 > 7.00

| | | | | | | | | | | |--- truncated branch of depth 3

| | | | | | | | |--- feature\_3 > 4.50

| | | | | | | | | |--- feature\_0 <= 2.50

| | | | | | | | | | |--- value: [491.00]

| | | | | | | | | |--- feature\_0 > 2.50

| | | | | | | | | | |--- value: [429.00]

| | | | | | |--- feature\_8 > 0.64

| | | | | | | |--- feature\_7 <= 0.31

| | | | | | | | |--- value: [168.00]

| | | | | | | |--- feature\_7 > 0.31

| | | | | | | | |--- feature\_9 <= 0.13

| | | | | | | | | |--- value: [349.00]

| | | | | | | | |--- feature\_9 > 0.13

| | | | | | | | | |--- feature\_6 <= 0.32

| | | | | | | | | | |--- value: [289.00]

| | | | | | | | | |--- feature\_6 > 0.32

| | | | | | | | | | |--- value: [314.00]

| | | | |--- feature\_9 > 0.20

| | | | | |--- feature\_8 <= 0.78

| | | | | | |--- feature\_9 <= 0.23

| | | | | | | |--- feature\_9 <= 0.21

| | | | | | | | |--- feature\_1 <= 7.00

| | | | | | | | | |--- feature\_8 <= 0.38

| | | | | | | | | | |--- value: [222.00]

| | | | | | | | | |--- feature\_8 > 0.38

| | | | | | | | | | |--- value: [221.00]

| | | | | | | | |--- feature\_1 > 7.00

| | | | | | | | | |--- value: [198.00]

| | | | | | | |--- feature\_9 > 0.21

| | | | | | | | |--- feature\_5 <= 1.50

| | | | | | | | | |--- feature\_3 <= 3.00

| | | | | | | | | | |--- value: [137.00]

| | | | | | | | | |--- feature\_3 > 3.00

| | | | | | | | | | |--- value: [123.00]

| | | | | | | | |--- feature\_5 > 1.50

| | | | | | | | | |--- feature\_1 <= 3.50

| | | | | | | | | | |--- value: [191.00]

| | | | | | | | | |--- feature\_1 > 3.50

| | | | | | | | | | |--- value: [177.00]

| | | | | | |--- feature\_9 > 0.23

| | | | | | | |--- feature\_3 <= 4.50

| | | | | | | | |--- feature\_8 <= 0.56

| | | | | | | | | |--- feature\_0 <= 1.50

| | | | | | | | | | |--- feature\_9 <= 0.26

| | | | | | | | | | | |--- value: [203.00]

| | | | | | | | | | |--- feature\_9 > 0.26

| | | | | | | | | | | |--- truncated branch of depth 3

| | | | | | | | | |--- feature\_0 > 1.50

| | | | | | | | | | |--- feature\_7 <= 0.28

| | | | | | | | | | | |--- value: [326.00]

| | | | | | | | | | |--- feature\_7 > 0.28

| | | | | | | | | | | |--- value: [317.00]

| | | | | | | | |--- feature\_8 > 0.56

| | | | | | | | | |--- feature\_0 <= 3.00

| | | | | | | | | | |--- feature\_1 <= 8.00

| | | | | | | | | | | |--- value: [195.00]

| | | | | | | | | | |--- feature\_1 > 8.00

| | | | | | | | | | | |--- value: [247.00]

| | | | | | | | | |--- feature\_0 > 3.00

| | | | | | | | | | |--- feature\_7 <= 0.29

| | | | | | | | | | | |--- value: [150.00]

| | | | | | | | | | |--- feature\_7 > 0.29

| | | | | | | | | | | |--- value: [139.00]

| | | | | | | |--- feature\_3 > 4.50

| | | | | | | | |--- feature\_9 <= 0.23

| | | | | | | | | |--- value: [456.00]

| | | | | | | | |--- feature\_9 > 0.23

| | | | | | | | | |--- feature\_0 <= 1.50

| | | | | | | | | | |--- value: [247.00]

| | | | | | | | | |--- feature\_0 > 1.50

| | | | | | | | | | |--- feature\_6 <= 0.28

| | | | | | | | | | | |--- value: [300.00]

| | | | | | | | | | |--- feature\_6 > 0.28

| | | | | | | | | | | |--- value: [307.00]

| | | | | |--- feature\_8 > 0.78

| | | | | | |--- feature\_9 <= 0.28

| | | | | | | |--- feature\_1 <= 6.50

| | | | | | | | |--- feature\_7 <= 0.26

| | | | | | | | | |--- value: [179.00]

| | | | | | | | |--- feature\_7 > 0.26

| | | | | | | | | |--- value: [166.00]

| | | | | | | |--- feature\_1 > 6.50

| | | | | | | | |--- feature\_8 <= 0.81

| | | | | | | | | |--- value: [123.00]

| | | | | | | | |--- feature\_8 > 0.81

| | | | | | | | | |--- value: [87.00]

| | | | | | |--- feature\_9 > 0.28

| | | | | | | |--- value: [9.00]

| | |--- feature\_6 > 0.34

| | | |--- feature\_3 <= 4.50

| | | | |--- feature\_8 <= 0.48

| | | | | |--- feature\_6 <= 0.42

| | | | | | |--- feature\_8 <= 0.44

| | | | | | | |--- feature\_6 <= 0.40

| | | | | | | | |--- feature\_9 <= 0.23

| | | | | | | | | |--- value: [229.00]

| | | | | | | | |--- feature\_9 > 0.23

| | | | | | | | | |--- feature\_1 <= 6.00

| | | | | | | | | | |--- value: [324.00]

| | | | | | | | | |--- feature\_1 > 6.00

| | | | | | | | | | |--- value: [340.00]

| | | | | | | |--- feature\_6 > 0.40

| | | | | | | | |--- value: [208.00]

| | | | | | |--- feature\_8 > 0.44

| | | | | | | |--- feature\_3 <= 3.50

| | | | | | | | |--- feature\_8 <= 0.47

| | | | | | | | | |--- feature\_8 <= 0.46

| | | | | | | | | | |--- value: [518.00]

| | | | | | | | | |--- feature\_8 > 0.46

| | | | | | | | | | |--- value: [482.00]

| | | | | | | | |--- feature\_8 > 0.47

| | | | | | | | | |--- value: [413.00]

| | | | | | | |--- feature\_3 > 3.50

| | | | | | | | |--- value: [663.00]

| | | | | |--- feature\_6 > 0.42

| | | | | | |--- feature\_7 <= 0.43

| | | | | | | |--- value: [1192.00]

| | | | | | |--- feature\_7 > 0.43

| | | | | | | |--- feature\_9 <= 0.32

| | | | | | | | |--- feature\_0 <= 3.00

| | | | | | | | | |--- value: [819.00]

| | | | | | | | |--- feature\_0 > 3.00

| | | | | | | | | |--- value: [834.00]

| | | | | | | |--- feature\_9 > 0.32

| | | | | | | | |--- value: [795.00]

| | | | |--- feature\_8 > 0.48

| | | | | |--- feature\_5 <= 1.50

| | | | | | |--- feature\_8 <= 0.55

| | | | | | | |--- feature\_9 <= 0.30

| | | | | | | | |--- feature\_9 <= 0.21

| | | | | | | | | |--- feature\_7 <= 0.36

| | | | | | | | | | |--- value: [141.00]

| | | | | | | | | |--- feature\_7 > 0.36

| | | | | | | | | | |--- value: [199.00]

| | | | | | | | |--- feature\_9 > 0.21

| | | | | | | | | |--- feature\_6 <= 0.40

| | | | | | | | | | |--- value: [253.00]

| | | | | | | | | |--- feature\_6 > 0.40

| | | | | | | | | | |--- value: [259.00]

| | | | | | | |--- feature\_9 > 0.30

| | | | | | | | |--- feature\_7 <= 0.41

| | | | | | | | | |--- value: [432.00]

| | | | | | | | |--- feature\_7 > 0.41

| | | | | | | | | |--- value: [331.00]

| | | | | | |--- feature\_8 > 0.55

| | | | | | | |--- feature\_8 <= 0.65

| | | | | | | | |--- feature\_9 <= 0.19

| | | | | | | | | |--- feature\_6 <= 0.40

| | | | | | | | | | |--- value: [615.00]

| | | | | | | | | |--- feature\_6 > 0.40

| | | | | | | | | | |--- value: [571.00]

| | | | | | | | |--- feature\_9 > 0.19

| | | | | | | | | |--- feature\_9 <= 0.23

| | | | | | | | | | |--- value: [460.00]

| | | | | | | | | |--- feature\_9 > 0.23

| | | | | | | | | | |--- value: [394.00]

| | | | | | | |--- feature\_8 > 0.65

| | | | | | | | |--- feature\_9 <= 0.10

| | | | | | | | | |--- feature\_8 <= 0.74

| | | | | | | | | | |--- feature\_9 <= 0.08

| | | | | | | | | | | |--- value: [376.00]

| | | | | | | | | | |--- feature\_9 > 0.08

| | | | | | | | | | | |--- value: [370.00]

| | | | | | | | | |--- feature\_8 > 0.74

| | | | | | | | | | |--- value: [305.00]

| | | | | | | | |--- feature\_9 > 0.10

| | | | | | | | | |--- feature\_8 <= 0.76

| | | | | | | | | | |--- feature\_8 <= 0.67

| | | | | | | | | | | |--- value: [433.00]

| | | | | | | | | | |--- feature\_8 > 0.67

| | | | | | | | | | | |--- value: [410.00]

| | | | | | | | | |--- feature\_8 > 0.76

| | | | | | | | | | |--- value: [439.00]

| | | | | |--- feature\_5 > 1.50

| | | | | | |--- feature\_6 <= 0.38

| | | | | | | |--- feature\_9 <= 0.17

| | | | | | | | |--- feature\_3 <= 1.50

| | | | | | | | | |--- value: [330.00]

| | | | | | | | |--- feature\_3 > 1.50

| | | | | | | | | |--- feature\_8 <= 0.63

| | | | | | | | | | |--- value: [466.00]

| | | | | | | | | |--- feature\_8 > 0.63

| | | | | | | | | | |--- value: [534.00]

| | | | | | | |--- feature\_9 > 0.17

| | | | | | | | |--- feature\_9 <= 0.35

| | | | | | | | | |--- feature\_8 <= 0.82

| | | | | | | | | | |--- feature\_6 <= 0.37

| | | | | | | | | | | |--- truncated branch of depth 3

| | | | | | | | | | |--- feature\_6 > 0.37

| | | | | | | | | | | |--- value: [269.00]

| | | | | | | | | |--- feature\_8 > 0.82

| | | | | | | | | | |--- value: [203.00]

| | | | | | | | |--- feature\_9 > 0.35

| | | | | | | | | |--- value: [127.00]

| | | | | | |--- feature\_6 > 0.38

| | | | | | | |--- feature\_9 <= 0.26

| | | | | | | | |--- feature\_1 <= 3.00

| | | | | | | | | |--- value: [190.00]

| | | | | | | | |--- feature\_1 > 3.00

| | | | | | | | | |--- feature\_7 <= 0.40

| | | | | | | | | | |--- value: [233.00]

| | | | | | | | | |--- feature\_7 > 0.40

| | | | | | | | | | |--- feature\_6 <= 0.42

| | | | | | | | | | | |--- truncated branch of depth 3

| | | | | | | | | | |--- feature\_6 > 0.42

| | | | | | | | | | | |--- truncated branch of depth 2

| | | | | | | |--- feature\_9 > 0.26

| | | | | | | | |--- feature\_8 <= 0.75

| | | | | | | | | |--- feature\_8 <= 0.69

| | | | | | | | | | |--- feature\_0 <= 3.00

| | | | | | | | | | | |--- value: [167.00]

| | | | | | | | | | |--- feature\_0 > 3.00

| | | | | | | | | | | |--- value: [181.00]

| | | | | | | | | |--- feature\_8 > 0.69

| | | | | | | | | | |--- value: [255.00]

| | | | | | | | |--- feature\_8 > 0.75

| | | | | | | | | |--- feature\_5 <= 2.50

| | | | | | | | | | |--- feature\_3 <= 2.00

| | | | | | | | | | | |--- value: [81.00]

| | | | | | | | | | |--- feature\_3 > 2.00

| | | | | | | | | | | |--- value: [112.00]

| | | | | | | | | |--- feature\_5 > 2.50

| | | | | | | | | | |--- feature\_8 <= 0.93

| | | | | | | | | | | |--- value: [2.00]

| | | | | | | | | | |--- feature\_8 > 0.93

| | | | | | | | | | | |--- value: [50.00]

| | | |--- feature\_3 > 4.50

| | | | |--- feature\_8 <= 0.39

| | | | | |--- value: [1807.00]

| | | | |--- feature\_8 > 0.39

| | | | | |--- feature\_5 <= 1.50

| | | | | | |--- feature\_7 <= 0.37

| | | | | | | |--- feature\_7 <= 0.35

| | | | | | | | |--- value: [484.00]

| | | | | | | |--- feature\_7 > 0.35

| | | | | | | | |--- feature\_6 <= 0.36

| | | | | | | | | |--- value: [618.00]

| | | | | | | | |--- feature\_6 > 0.36

| | | | | | | | | |--- value: [709.00]

| | | | | | |--- feature\_7 > 0.37

| | | | | | | |--- feature\_8 <= 0.61

| | | | | | | | |--- value: [1603.00]

| | | | | | | |--- feature\_8 > 0.61

| | | | | | | | |--- value: [1095.00]

| | | | | |--- feature\_5 > 1.50

| | | | | | |--- feature\_7 <= 0.37

| | | | | | | |--- feature\_8 <= 0.58

| | | | | | | | |--- value: [178.00]

| | | | | | | |--- feature\_8 > 0.58

| | | | | | | | |--- value: [246.00]

| | | | | | |--- feature\_7 > 0.37

| | | | | | | |--- feature\_6 <= 0.39

| | | | | | | | |--- value: [796.00]

| | | | | | | |--- feature\_6 > 0.39

| | | | | | | | |--- feature\_8 <= 0.73

| | | | | | | | | |--- feature\_0 <= 2.50

| | | | | | | | | | |--- value: [447.00]

| | | | | | | | | |--- feature\_0 > 2.50

| | | | | | | | | | |--- value: [470.00]

| | | | | | | | |--- feature\_8 > 0.73

| | | | | | | | | |--- value: [548.00]

|--- feature\_6 > 0.45

| |--- feature\_4 <= 0.50

| | |--- feature\_8 <= 0.83

| | | |--- feature\_1 <= 10.50

| | | | |--- feature\_9 <= 0.28

| | | | | |--- feature\_6 <= 0.81

| | | | | | |--- feature\_8 <= 0.67

| | | | | | | |--- feature\_9 <= 0.09

| | | | | | | | |--- value: [3065.00]

| | | | | | | |--- feature\_9 > 0.09

| | | | | | | | |--- feature\_8 <= 0.66

| | | | | | | | | |--- feature\_9 <= 0.15

| | | | | | | | | | |--- feature\_9 <= 0.14

| | | | | | | | | | | |--- truncated branch of depth 8

| | | | | | | | | | |--- feature\_9 > 0.14

| | | | | | | | | | | |--- truncated branch of depth 3

| | | | | | | | | |--- feature\_9 > 0.15

| | | | | | | | | | |--- feature\_9 <= 0.16

| | | | | | | | | | | |--- truncated branch of depth 4

| | | | | | | | | | |--- feature\_9 > 0.16

| | | | | | | | | | | |--- truncated branch of depth 10

| | | | | | | | |--- feature\_8 > 0.66

| | | | | | | | | |--- value: [3031.00]

| | | | | | |--- feature\_8 > 0.67

| | | | | | | |--- feature\_7 <= 0.67

| | | | | | | | |--- feature\_6 <= 0.51

| | | | | | | | | |--- feature\_8 <= 0.79

| | | | | | | | | | |--- feature\_7 <= 0.46

| | | | | | | | | | | |--- value: [1138.00]

| | | | | | | | | | |--- feature\_7 > 0.46

| | | | | | | | | | | |--- truncated branch of depth 2

| | | | | | | | | |--- feature\_8 > 0.79

| | | | | | | | | | |--- value: [2207.00]

| | | | | | | | |--- feature\_6 > 0.51

| | | | | | | | | |--- feature\_1 <= 4.00

| | | | | | | | | | |--- value: [3155.00]

| | | | | | | | | |--- feature\_1 > 4.00

| | | | | | | | | | |--- feature\_6 <= 0.69

| | | | | | | | | | | |--- truncated branch of depth 7

| | | | | | | | | | |--- feature\_6 > 0.69

| | | | | | | | | | | |--- truncated branch of depth 6

| | | | | | | |--- feature\_7 > 0.67

| | | | | | | | |--- feature\_6 <= 0.74

| | | | | | | | | |--- feature\_3 <= 0.50

| | | | | | | | | | |--- feature\_7 <= 0.70

| | | | | | | | | | | |--- value: [1249.00]

| | | | | | | | | | |--- feature\_7 > 0.70

| | | | | | | | | | | |--- value: [1298.00]

| | | | | | | | | |--- feature\_3 > 0.50

| | | | | | | | | | |--- feature\_1 <= 6.50

| | | | | | | | | | | |--- value: [1549.00]

| | | | | | | | | | |--- feature\_1 > 6.50

| | | | | | | | | | | |--- value: [1521.00]

| | | | | | | | |--- feature\_6 > 0.74

| | | | | | | | | |--- value: [1920.00]

| | | | | |--- feature\_6 > 0.81

| | | | | | |--- feature\_9 <= 0.13

| | | | | | | |--- value: [1203.00]

| | | | | | |--- feature\_9 > 0.13

| | | | | | | |--- value: [987.00]

| | | | |--- feature\_9 > 0.28

| | | | | |--- feature\_8 <= 0.52

| | | | | | |--- value: [1558.00]

| | | | | |--- feature\_8 > 0.52

| | | | | | |--- feature\_7 <= 0.54

| | | | | | | |--- value: [998.00]

| | | | | | |--- feature\_7 > 0.54

| | | | | | | |--- value: [1198.00]

| | | |--- feature\_1 > 10.50

| | | | |--- feature\_3 <= 0.50

| | | | | |--- value: [787.00]

| | | | |--- feature\_3 > 0.50

| | | | | |--- value: [1097.00]

| | |--- feature\_8 > 0.83

| | | |--- feature\_6 <= 0.67

| | | | |--- feature\_8 <= 0.88

| | | | | |--- feature\_7 <= 0.57

| | | | | | |--- value: [1582.00]

| | | | | |--- feature\_7 > 0.57

| | | | | | |--- value: [1483.00]

| | | | |--- feature\_8 > 0.88

| | | | | |--- feature\_6 <= 0.48

| | | | | | |--- value: [1462.00]

| | | | | |--- feature\_6 > 0.48

| | | | | | |--- feature\_7 <= 0.49

| | | | | | | |--- value: [1033.00]

| | | | | | |--- feature\_7 > 0.49

| | | | | | | |--- value: [902.00]

| | | |--- feature\_6 > 0.67

| | | | |--- value: [226.00]

| |--- feature\_4 > 0.50

| | |--- feature\_8 <= 0.83

| | | |--- feature\_3 <= 4.50

| | | | |--- feature\_7 <= 0.53

| | | | | |--- feature\_9 <= 0.10

| | | | | | |--- feature\_6 <= 0.48

| | | | | | | |--- value: [1348.00]

| | | | | | |--- feature\_6 > 0.48

| | | | | | | |--- feature\_6 <= 0.53

| | | | | | | | |--- feature\_9 <= 0.09

| | | | | | | | | |--- feature\_3 <= 2.50

| | | | | | | | | | |--- value: [830.00]

| | | | | | | | | |--- feature\_3 > 2.50

| | | | | | | | | | |--- value: [846.00]

| | | | | | | | |--- feature\_9 > 0.09

| | | | | | | | | |--- value: [763.00]

| | | | | | | |--- feature\_6 > 0.53

| | | | | | | | |--- value: [1122.00]

| | | | | |--- feature\_9 > 0.10

| | | | | | |--- feature\_9 <= 0.23

| | | | | | | |--- feature\_3 <= 2.50

| | | | | | | | |--- feature\_8 <= 0.64

| | | | | | | | | |--- feature\_9 <= 0.18

| | | | | | | | | | |--- feature\_1 <= 7.50

| | | | | | | | | | | |--- truncated branch of depth 3

| | | | | | | | | | |--- feature\_1 > 7.50

| | | | | | | | | | | |--- truncated branch of depth 2

| | | | | | | | | |--- feature\_9 > 0.18

| | | | | | | | | | |--- feature\_8 <= 0.52

| | | | | | | | | | | |--- value: [838.00]

| | | | | | | | | | |--- feature\_8 > 0.52

| | | | | | | | | | | |--- value: [922.00]

| | | | | | | | |--- feature\_8 > 0.64

| | | | | | | | | |--- feature\_9 <= 0.14

| | | | | | | | | | |--- feature\_5 <= 1.50

| | | | | | | | | | | |--- value: [699.00]

| | | | | | | | | | |--- feature\_5 > 1.50

| | | | | | | | | | | |--- value: [637.00]

| | | | | | | | | |--- feature\_9 > 0.14

| | | | | | | | | | |--- feature\_9 <= 0.18

| | | | | | | | | | | |--- value: [409.00]

| | | | | | | | | | |--- feature\_9 > 0.18

| | | | | | | | | | | |--- value: [486.00]

| | | | | | | |--- feature\_3 > 2.50

| | | | | | | | |--- feature\_6 <= 0.51

| | | | | | | | | |--- feature\_9 <= 0.13

| | | | | | | | | | |--- value: [655.00]

| | | | | | | | | |--- feature\_9 > 0.13

| | | | | | | | | | |--- feature\_7 <= 0.45

| | | | | | | | | | | |--- value: [516.00]

| | | | | | | | | | |--- feature\_7 > 0.45

| | | | | | | | | | | |--- truncated branch of depth 4

| | | | | | | | |--- feature\_6 > 0.51

| | | | | | | | | |--- feature\_7 <= 0.53

| | | | | | | | | | |--- feature\_9 <= 0.15

| | | | | | | | | | | |--- value: [735.00]

| | | | | | | | | | |--- feature\_9 > 0.15

| | | | | | | | | | | |--- value: [695.00]

| | | | | | | | | |--- feature\_7 > 0.53

| | | | | | | | | | |--- feature\_9 <= 0.15

| | | | | | | | | | | |--- value: [550.00]

| | | | | | | | | | |--- feature\_9 > 0.15

| | | | | | | | | | | |--- value: [559.00]

| | | | | | |--- feature\_9 > 0.23

| | | | | | | |--- feature\_7 <= 0.47

| | | | | | | | |--- feature\_9 <= 0.37

| | | | | | | | | |--- feature\_9 <= 0.31

| | | | | | | | | | |--- value: [614.00]

| | | | | | | | | |--- feature\_9 > 0.31

| | | | | | | | | | |--- value: [745.00]

| | | | | | | | |--- feature\_9 > 0.37

| | | | | | | | | |--- value: [471.00]

| | | | | | | |--- feature\_7 > 0.47

| | | | | | | | |--- feature\_9 <= 0.28

| | | | | | | | | |--- value: [1008.00]

| | | | | | | | |--- feature\_9 > 0.28

| | | | | | | | | |--- feature\_3 <= 2.50

| | | | | | | | | | |--- value: [905.00]

| | | | | | | | | |--- feature\_3 > 2.50

| | | | | | | | | | |--- value: [834.00]

| | | | |--- feature\_7 > 0.53

| | | | | |--- feature\_9 <= 0.13

| | | | | | |--- feature\_7 <= 0.72

| | | | | | | |--- feature\_6 <= 0.68

| | | | | | | | |--- feature\_8 <= 0.82

| | | | | | | | | |--- feature\_9 <= 0.09

| | | | | | | | | | |--- feature\_7 <= 0.59

| | | | | | | | | | | |--- truncated branch of depth 2

| | | | | | | | | | |--- feature\_7 > 0.59

| | | | | | | | | | | |--- truncated branch of depth 2

| | | | | | | | | |--- feature\_9 > 0.09

| | | | | | | | | | |--- feature\_6 <= 0.64

| | | | | | | | | | | |--- truncated branch of depth 6

| | | | | | | | | | |--- feature\_6 > 0.64

| | | | | | | | | | | |--- truncated branch of depth 3

| | | | | | | | |--- feature\_8 > 0.82

| | | | | | | | | |--- value: [1334.00]

| | | | | | | |--- feature\_6 > 0.68

| | | | | | | | |--- feature\_9 <= 0.12

| | | | | | | | | |--- feature\_6 <= 0.69

| | | | | | | | | | |--- value: [1177.00]

| | | | | | | | | |--- feature\_6 > 0.69

| | | | | | | | | | |--- feature\_9 <= 0.10

| | | | | | | | | | | |--- value: [1363.00]

| | | | | | | | | | |--- feature\_9 > 0.10

| | | | | | | | | | | |--- truncated branch of depth 2

| | | | | | | | |--- feature\_9 > 0.12

| | | | | | | | | |--- feature\_9 <= 0.13

| | | | | | | | | | |--- feature\_6 <= 0.73

| | | | | | | | | | | |--- value: [989.00]

| | | | | | | | | | |--- feature\_6 > 0.73

| | | | | | | | | | | |--- truncated branch of depth 2

| | | | | | | | | |--- feature\_9 > 0.13

| | | | | | | | | | |--- value: [1233.00]

| | | | | | |--- feature\_7 > 0.72

| | | | | | | |--- feature\_8 <= 0.61

| | | | | | | | |--- feature\_3 <= 3.50

| | | | | | | | | |--- feature\_8 <= 0.55

| | | | | | | | | | |--- value: [921.00]

| | | | | | | | | |--- feature\_8 > 0.55

| | | | | | | | | | |--- value: [872.00]

| | | | | | | | |--- feature\_3 > 3.50

| | | | | | | | | |--- value: [778.00]

| | | | | | | |--- feature\_8 > 0.61

| | | | | | | | |--- feature\_6 <= 0.76

| | | | | | | | | |--- value: [568.00]

| | | | | | | | |--- feature\_6 > 0.76

| | | | | | | | | |--- value: [673.00]

| | | | | |--- feature\_9 > 0.13

| | | | | | |--- feature\_8 <= 0.53

| | | | | | | |--- feature\_6 <= 0.81

| | | | | | | | |--- feature\_9 <= 0.15

| | | | | | | | | |--- value: [1281.00]

| | | | | | | | |--- feature\_9 > 0.15

| | | | | | | | | |--- feature\_8 <= 0.51

| | | | | | | | | | |--- feature\_1 <= 6.50

| | | | | | | | | | | |--- truncated branch of depth 6

| | | | | | | | | | |--- feature\_1 > 6.50

| | | | | | | | | | | |--- truncated branch of depth 4

| | | | | | | | | |--- feature\_8 > 0.51

| | | | | | | | | | |--- feature\_1 <= 6.00

| | | | | | | | | | | |--- value: [1242.00]

| | | | | | | | | | |--- feature\_1 > 6.00

| | | | | | | | | | | |--- value: [1032.00]

| | | | | | | |--- feature\_6 > 0.81

| | | | | | | | |--- value: [1405.00]

| | | | | | |--- feature\_8 > 0.53

| | | | | | | |--- feature\_5 <= 1.50

| | | | | | | | |--- feature\_9 <= 0.17

| | | | | | | | | |--- feature\_9 <= 0.17

| | | | | | | | | | |--- feature\_3 <= 3.50

| | | | | | | | | | | |--- truncated branch of depth 7

| | | | | | | | | | |--- feature\_3 > 3.50

| | | | | | | | | | | |--- truncated branch of depth 2

| | | | | | | | | |--- feature\_9 > 0.17

| | | | | | | | | | |--- feature\_9 <= 0.17

| | | | | | | | | | | |--- value: [1128.00]

| | | | | | | | | | |--- feature\_9 > 0.17

| | | | | | | | | | | |--- value: [1198.00]

| | | | | | | | |--- feature\_9 > 0.17

| | | | | | | | | |--- feature\_7 <= 0.73

| | | | | | | | | | |--- feature\_7 <= 0.66

| | | | | | | | | | | |--- truncated branch of depth 7

| | | | | | | | | | |--- feature\_7 > 0.66

| | | | | | | | | | | |--- truncated branch of depth 5

| | | | | | | | | |--- feature\_7 > 0.73

| | | | | | | | | | |--- feature\_7 <= 0.74

| | | | | | | | | | | |--- value: [662.00]

| | | | | | | | | | |--- feature\_7 > 0.74

| | | | | | | | | | | |--- value: [606.00]

| | | | | | | |--- feature\_5 > 1.50

| | | | | | | | |--- feature\_3 <= 1.50

| | | | | | | | | |--- feature\_8 <= 0.76

| | | | | | | | | | |--- feature\_7 <= 0.55

| | | | | | | | | | | |--- truncated branch of depth 2

| | | | | | | | | | |--- feature\_7 > 0.55

| | | | | | | | | | | |--- truncated branch of depth 4

| | | | | | | | | |--- feature\_8 > 0.76

| | | | | | | | | | |--- feature\_6 <= 0.58

| | | | | | | | | | | |--- value: [653.00]

| | | | | | | | | | |--- feature\_6 > 0.58

| | | | | | | | | | | |--- value: [630.00]

| | | | | | | | |--- feature\_3 > 1.50

| | | | | | | | | |--- feature\_9 <= 0.19

| | | | | | | | | | |--- feature\_9 <= 0.15

| | | | | | | | | | | |--- truncated branch of depth 5

| | | | | | | | | | |--- feature\_9 > 0.15

| | | | | | | | | | | |--- truncated branch of depth 3

| | | | | | | | | |--- feature\_9 > 0.19

| | | | | | | | | | |--- feature\_1 <= 8.00

| | | | | | | | | | | |--- truncated branch of depth 5

| | | | | | | | | | |--- feature\_1 > 8.00

| | | | | | | | | | | |--- value: [428.00]

| | | |--- feature\_3 > 4.50

| | | | |--- feature\_1 <= 4.00

| | | | | |--- value: [2469.00]

| | | | |--- feature\_1 > 4.00

| | | | | |--- feature\_8 <= 0.72

| | | | | | |--- feature\_9 <= 0.12

| | | | | | | |--- feature\_9 <= 0.08

| | | | | | | | |--- value: [1325.00]

| | | | | | | |--- feature\_9 > 0.08

| | | | | | | | |--- feature\_8 <= 0.65

| | | | | | | | | |--- feature\_0 <= 3.50

| | | | | | | | | | |--- value: [1511.00]

| | | | | | | | | |--- feature\_0 > 3.50

| | | | | | | | | | |--- value: [1516.00]

| | | | | | | | |--- feature\_8 > 0.65

| | | | | | | | | |--- value: [1379.00]

| | | | | | |--- feature\_9 > 0.12

| | | | | | | |--- feature\_7 <= 0.77

| | | | | | | | |--- feature\_1 <= 6.50

| | | | | | | | | |--- feature\_8 <= 0.52

| | | | | | | | | | |--- feature\_8 <= 0.36

| | | | | | | | | | | |--- value: [898.00]

| | | | | | | | | | |--- feature\_8 > 0.36

| | | | | | | | | | | |--- truncated branch of depth 2

| | | | | | | | | |--- feature\_8 > 0.52

| | | | | | | | | | |--- feature\_0 <= 2.50

| | | | | | | | | | | |--- truncated branch of depth 3

| | | | | | | | | | |--- feature\_0 > 2.50

| | | | | | | | | | | |--- truncated branch of depth 2

| | | | | | | | |--- feature\_1 > 6.50

| | | | | | | | | |--- feature\_6 <= 0.72

| | | | | | | | | | |--- feature\_6 <= 0.69

| | | | | | | | | | | |--- truncated branch of depth 2

| | | | | | | | | | |--- feature\_6 > 0.69

| | | | | | | | | | | |--- truncated branch of depth 3

| | | | | | | | | |--- feature\_6 > 0.72

| | | | | | | | | | |--- feature\_9 <= 0.16

| | | | | | | | | | | |--- value: [1366.00]

| | | | | | | | | | |--- feature\_9 > 0.16

| | | | | | | | | | | |--- truncated branch of depth 2

| | | | | | | |--- feature\_7 > 0.77

| | | | | | | | |--- feature\_6 <= 0.84

| | | | | | | | | |--- value: [829.00]

| | | | | | | | |--- feature\_6 > 0.84

| | | | | | | | | |--- value: [670.00]

| | | | | |--- feature\_8 > 0.72

| | | | | | |--- feature\_9 <= 0.20

| | | | | | | |--- feature\_1 <= 5.50

| | | | | | | | |--- feature\_6 <= 0.61

| | | | | | | | | |--- value: [909.00]

| | | | | | | | |--- feature\_6 > 0.61

| | | | | | | | | |--- value: [1417.00]

| | | | | | | |--- feature\_1 > 5.50

| | | | | | | | |--- feature\_6 <= 0.55

| | | | | | | | | |--- value: [1182.00]

| | | | | | | | |--- feature\_6 > 0.55

| | | | | | | | | |--- feature\_9 <= 0.16

| | | | | | | | | | |--- feature\_8 <= 0.75

| | | | | | | | | | | |--- truncated branch of depth 2

| | | | | | | | | | |--- feature\_8 > 0.75

| | | | | | | | | | | |--- truncated branch of depth 2

| | | | | | | | | |--- feature\_9 > 0.16

| | | | | | | | | | |--- value: [1045.00]

| | | | | | |--- feature\_9 > 0.20

| | | | | | | |--- feature\_1 <= 8.00

| | | | | | | | |--- value: [533.00]

| | | | | | | |--- feature\_1 > 8.00

| | | | | | | | |--- value: [529.00]

| | |--- feature\_8 > 0.83

| | | |--- feature\_5 <= 2.50

| | | | |--- feature\_8 <= 0.94

| | | | | |--- feature\_1 <= 5.50

| | | | | | |--- feature\_9 <= 0.19

| | | | | | | |--- value: [692.00]

| | | | | | |--- feature\_9 > 0.19

| | | | | | | |--- feature\_3 <= 2.50

| | | | | | | | |--- value: [678.00]

| | | | | | | |--- feature\_3 > 2.50

| | | | | | | | |--- feature\_7 <= 0.55

| | | | | | | | | |--- value: [536.00]

| | | | | | | | |--- feature\_7 > 0.55

| | | | | | | | | |--- value: [547.00]

| | | | | |--- feature\_1 > 5.50

| | | | | | |--- feature\_3 <= 3.50

| | | | | | | |--- feature\_6 <= 0.60

| | | | | | | | |--- value: [438.00]

| | | | | | | |--- feature\_6 > 0.60

| | | | | | | | |--- feature\_3 <= 2.50

| | | | | | | | | |--- value: [477.00]

| | | | | | | | |--- feature\_3 > 2.50

| | | | | | | | | |--- value: [480.00]

| | | | | | |--- feature\_3 > 3.50

| | | | | | | |--- value: [555.00]

| | | | |--- feature\_8 > 0.94

| | | | | |--- value: [258.00]

| | | |--- feature\_5 > 2.50

| | | | |--- feature\_1 <= 11.00

| | | | | |--- feature\_7 <= 0.53

| | | | | | |--- feature\_9 <= 0.25

| | | | | | | |--- value: [254.00]

| | | | | | |--- feature\_9 > 0.25

| | | | | | | |--- feature\_1 <= 9.50

| | | | | | | | |--- value: [204.00]

| | | | | | | |--- feature\_1 > 9.50

| | | | | | | | |--- value: [217.00]

| | | | | |--- feature\_7 > 0.53

| | | | | | |--- value: [315.00]

| | | | |--- feature\_1 > 11.00

| | | | | |--- value: [126.00]

So now we have a tree-based model; but is it any good? Let's evaluate it with the test data.

In [17]:



*# Evaluate the model using the test data*

predictions **=** model.predict(X\_test)

mse **=** mean\_squared\_error(y\_test, predictions)

print("MSE:", mse)

rmse **=** np.sqrt(mse)

print("RMSE:", rmse)

r2 **=** r2\_score(y\_test, predictions)

print("R2:", r2)

​

*# Plot predicted vs actual*

plt.scatter(y\_test, predictions)

plt.xlabel('Actual Labels')

plt.ylabel('Predicted Labels')

plt.title('Daily Bike Share Predictions')

*# overlay the regression line*

z **=** np.polyfit(y\_test, predictions, 1)

p **=** np.poly1d(z)

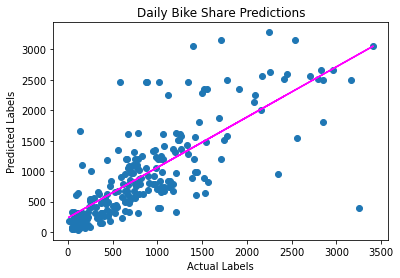
plt.plot(y\_test,p(y\_test), color**=**'magenta')

plt.show()

MSE: 256377.46363636362

RMSE: 506.33730223672404

R2: 0.4973880737244255



The tree-based model doesn't seem to have improved over the linear model, so what else could we try?

**Try an Ensemble Algorithm**

Ensemble algorithms work by combining multiple base estimators to produce an optimal model, either by applying an aggregate function to a collection of base models (sometimes referred to a *bagging*) or by building a sequence of models that build on one another to improve predictive performance (referred to as *boosting*).

For example, let's try a Random Forest model, which applies an averaging function to multiple Decision Tree models for a better overall model.

In [18]:



**from** sklearn.ensemble **import** RandomForestRegressor

​

*# Train the model*

model **=** RandomForestRegressor().fit(X\_train, y\_train)

print (model, "\n")

​

*# Evaluate the model using the test data*

predictions **=** model.predict(X\_test)

mse **=** mean\_squared\_error(y\_test, predictions)

print("MSE:", mse)

rmse **=** np.sqrt(mse)

print("RMSE:", rmse)

r2 **=** r2\_score(y\_test, predictions)

print("R2:", r2)

​

*# Plot predicted vs actual*

plt.scatter(y\_test, predictions)

plt.xlabel('Actual Labels')

plt.ylabel('Predicted Labels')

plt.title('Daily Bike Share Predictions')

*# overlay the regression line*

z **=** np.polyfit(y\_test, predictions, 1)

p **=** np.poly1d(z)

plt.plot(y\_test,p(y\_test), color**=**'magenta')

plt.show()

RandomForestRegressor(bootstrap=True, ccp\_alpha=0.0, criterion='mse',

max\_depth=None, max\_features='auto', max\_leaf\_nodes=None,

max\_samples=None, min\_impurity\_decrease=0.0,

min\_impurity\_split=None, min\_samples\_leaf=1,

min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0,

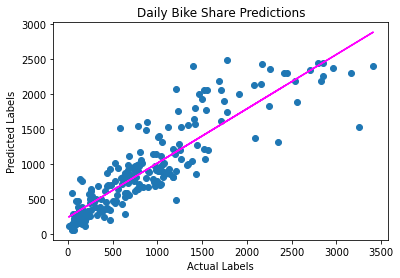
n\_estimators=100, n\_jobs=None, oob\_score=False,

random\_state=None, verbose=0, warm\_start=False)

MSE: 110217.67247636366

RMSE: 331.99047046016796

R2: 0.7839251707727003



For good measure, let's also try a *boosting* ensemble algorithm. We'll use a Gradient Boosting estimator, which like a Random Forest algorithm builds multiple trees, but instead of building them all independently and taking the average result, each tree is built on the outputs of the previous one in an attempt to incrementally reduce the *loss* (error) in the model.

In [19]:



*# Train the model*

**from** sklearn.ensemble **import** GradientBoostingRegressor

​

*# Fit a lasso model on the training set*

model **=** GradientBoostingRegressor().fit(X\_train, y\_train)

print (model, "\n")

​

*# Evaluate the model using the test data*

predictions **=** model.predict(X\_test)

mse **=** mean\_squared\_error(y\_test, predictions)

print("MSE:", mse)

rmse **=** np.sqrt(mse)

print("RMSE:", rmse)

r2 **=** r2\_score(y\_test, predictions)

print("R2:", r2)

​

*# Plot predicted vs actual*

plt.scatter(y\_test, predictions)

plt.xlabel('Actual Labels')

plt.ylabel('Predicted Labels')

plt.title('Daily Bike Share Predictions')

*# overlay the regression line*

z **=** np.polyfit(y\_test, predictions, 1)

p **=** np.poly1d(z)

plt.plot(y\_test,p(y\_test), color**=**'magenta')

plt.show()

GradientBoostingRegressor(alpha=0.9, ccp\_alpha=0.0, criterion='friedman\_mse',

init=None, learning\_rate=0.1, loss='ls', max\_depth=3,

max\_features=None, max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, n\_estimators=100,

n\_iter\_no\_change=None, presort='deprecated',

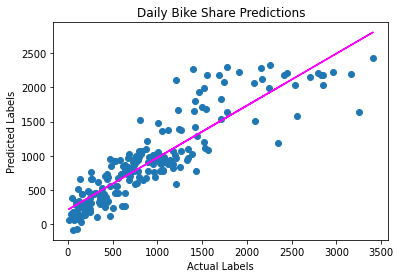
random\_state=None, subsample=1.0, tol=0.0001,

validation\_fraction=0.1, verbose=0, warm\_start=False)

MSE: 104221.67073819469

RMSE: 322.83381287931206

R2: 0.79567995584947



**Optimize Hyperparameters**

Take a look at the **GradientBoostingRegressor** estimator definition in the output above, and note that it, like the other estimators we tried previously, includes a large number of parameters that control the way the model is trained. In machine learning, the term *parameters* refers to values that can be determined from data; values that you specify to affect the behavior of a training algorithm are more correctly referred to as *hyperparameters*.

The specific hyperparameters for an estimator vary based on the algorithm that the estimator encapsulates. In the case of the **GradientBoostingRegressor** estimator, the algorithm is an ensemble that combines multiple decision trees to create an overall predictive model. You can learn about the hyperparameters for this estimator in the [Scikit-Learn documentation](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html).

We won't go into the details of each hyperparameter here, but they work together to affect the way the algorithm trains a model. In many cases, the default values provided by Scikit-Learn will work well; but there may be some advantage in modifying hyperparameters to get better predictive performance or reduce training time.

So how do you know what hyperparameter values you should use? Well, in the absence of a deep understanding of how the underlying algorithm works, you'll need to experiment. Fortunately, SciKit-Learn provides a way to *tune* hyperparameters by trying multiple combinations and finding the best result for a given performance metric.

Let's try using a *grid search* approach to try combinations from a grid of possible values for the **learning\_rate** and **n\_estimators** hyperparameters of the **GradientBoostingRegressor** estimator.

In [20]:



**from** sklearn.model\_selection **import** GridSearchCV

**from** sklearn.metrics **import** make\_scorer, r2\_score

​

*# Use a Gradient Boosting algorithm*

alg **=** GradientBoostingRegressor()

​

*# Try these hyperparameter values*

params **=** {

'learning\_rate': [0.1, 0.5, 1.0],

'n\_estimators' : [50, 100, 150]

}

​

*# Find the best hyperparameter combination to optimize the R2 metric*

score **=** make\_scorer(r2\_score)

gridsearch **=** GridSearchCV(alg, params, scoring**=**score, cv**=**3, return\_train\_score**=True**)

gridsearch.fit(X\_train, y\_train)

print("Best parameter combination:", gridsearch.best\_params\_, "\n")

​

*# Get the best model*

model**=**gridsearch.best\_estimator\_

print(model, "\n")

​

*# Evaluate the model using the test data*

predictions **=** model.predict(X\_test)

mse **=** mean\_squared\_error(y\_test, predictions)

print("MSE:", mse)

rmse **=** np.sqrt(mse)

print("RMSE:", rmse)

r2 **=** r2\_score(y\_test, predictions)

print("R2:", r2)

​

*# Plot predicted vs actual*

plt.scatter(y\_test, predictions)

plt.xlabel('Actual Labels')

plt.ylabel('Predicted Labels')

plt.title('Daily Bike Share Predictions')

*# overlay the regression line*

z **=** np.polyfit(y\_test, predictions, 1)

p **=** np.poly1d(z)

plt.plot(y\_test,p(y\_test), color**=**'magenta')

plt.show()

Best parameter combination: {'learning\_rate': 0.1, 'n\_estimators': 150}

GradientBoostingRegressor(alpha=0.9, ccp\_alpha=0.0, criterion='friedman\_mse',

init=None, learning\_rate=0.1, loss='ls', max\_depth=3,

max\_features=None, max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, n\_estimators=150,

n\_iter\_no\_change=None, presort='deprecated',

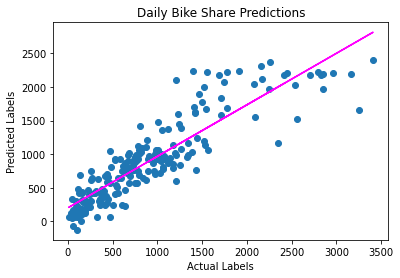
random\_state=None, subsample=1.0, tol=0.0001,

validation\_fraction=0.1, verbose=0, warm\_start=False)

MSE: 104454.95124129216

RMSE: 323.19491215254635

R2: 0.7952226240646816



**Note**: The use of random values in the Gradient Boosting algorithm results in slightly different metrics each time. In this case, the best model produced by hyperparameter tuning is unlikely to be significantly better than one trained with the default hyperparameter values; but it's still useful to know about the hyperparameter tuning technique!

**Preprocess the Data**

We trained a model with data that was loaded straight from a source file, with only moderately successful results.

In practice, it's common to perform some preprocessing of the data to make it easier for the algorithm to fit a model to it. There's a huge range of preprocessing transformations you can perform to get your data ready for modeling, but we'll limit ourselves to a few common techniques:

**Scaling numeric features**

Normalizing numeric features so they're on the same scale prevents features with large values from producing coefficients that disproportionately affect the predictions. For example, suppose your data includes the following numeric features:

| **A** | **B** | **C** |
| --- | --- | --- |
| 3 | 480 | 65 |

Normalizing these features to the same scale may result in the following values (assuming A contains values from 0 to 10, B contains values from 0 to 1000, and C contains values from 0 to 100):

| **A** | **B** | **C** |
| --- | --- | --- |
| 0.3 | 0.48 | 0.65 |

There are multiple ways you can scale numeric data, such as calculating the minimum and maximum values for each column and assigning a proportional value between 0 and 1, or by using the mean and standard deviation of a normally distributed variable to maintain the same *spread* of values on a different scale.

**Encoding categorical variables**

Machine learning models work best with numeric features rather than text values, so you generally need to convert categorical features into numeric representations. For example, suppose your data includes the following categorical feature.

| **Size** |
| --- |
| S |
| M |
| L |

You can apply *ordinal encoding* to substitute a unique integer value for each category, like this:

| **Size** |
| --- |
| 0 |
| 1 |
| 2 |

Another common technique is to use *one hot encoding* to create individual binary (0 or 1) features for each possible category value. For example, you could use one-hot encoding to translate the possible categories into binary columns like this:

| **Size\_S** | **Size\_M** | **Size\_L** |
| --- | --- | --- |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |

To apply these preprocessing transformations to the bike rental, we'll make use of a Scikit-Learn feature named *pipelines*. These enable us to define a set of preprocessing steps that end with an algorithm. You can then fit the entire pipeline to the data, so that the model encapsulates all of the preprocessing steps as well as the regression algorithm. This is useful, because when we want to use the model to predict values from new data, we need to apply the same transformations (based on the same statistical distributions and category encodings used with the training data).

**Note**: The term *pipeline* is used extensively in machine learning, often to mean very different things! In this context, we're using it to refer to pipeline objects in Scikit-Learn, but you may see it used elsewhere to mean something else.

In [21]:



*# Train the model*

**from** sklearn.compose **import** ColumnTransformer

**from** sklearn.pipeline **import** Pipeline

**from** sklearn.impute **import** SimpleImputer

**from** sklearn.preprocessing **import** StandardScaler, OneHotEncoder

**from** sklearn.linear\_model **import** LinearRegression

**import** numpy **as** np

​

*# Define preprocessing for numeric columns (scale them)*

numeric\_features **=** [6,7,8,9]

numeric\_transformer **=** Pipeline(steps**=**[

('scaler', StandardScaler())])

​

*# Define preprocessing for categorical features (encode them)*

categorical\_features **=** [0,1,2,3,4,5]

categorical\_transformer **=** Pipeline(steps**=**[

('onehot', OneHotEncoder(handle\_unknown**=**'ignore'))])

​

*# Combine preprocessing steps*

preprocessor **=** ColumnTransformer(

transformers**=**[

('num', numeric\_transformer, numeric\_features),

('cat', categorical\_transformer, categorical\_features)])

​

*# Create preprocessing and training pipeline*

pipeline **=** Pipeline(steps**=**[('preprocessor', preprocessor),

('regressor', GradientBoostingRegressor())])

​

​

*# fit the pipeline to train a linear regression model on the training set*

model **=** pipeline.fit(X\_train, (y\_train))

print (model)

Pipeline(memory=None,

steps=[('preprocessor',

ColumnTransformer(n\_jobs=None, remainder='drop',

sparse\_threshold=0.3,

transformer\_weights=None,

transformers=[('num',

Pipeline(memory=None,

steps=[('scaler',

StandardScaler(copy=True,

with\_mean=True,

with\_std=True))],

verbose=False),

[6, 7, 8, 9]),

('cat',

Pipeline(memory=None,

steps=[('onehot',

OneHotEncoder(categories='...

learning\_rate=0.1, loss='ls',

max\_depth=3, max\_features=None,

max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0,

min\_impurity\_split=None,

min\_samples\_leaf=1,

min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0,

n\_estimators=100,

n\_iter\_no\_change=None,

presort='deprecated',

random\_state=None, subsample=1.0,

tol=0.0001, validation\_fraction=0.1,

verbose=0, warm\_start=False))],

verbose=False)

OK, the model is trained, including the preprocessing steps. Let's see how it performs with the validation data.

In [22]:



*# Get predictions*

predictions **=** model.predict(X\_test)

​

*# Display metrics*

mse **=** mean\_squared\_error(y\_test, predictions)

print("MSE:", mse)

rmse **=** np.sqrt(mse)

print("RMSE:", rmse)

r2 **=** r2\_score(y\_test, predictions)

print("R2:", r2)

​

*# Plot predicted vs actual*

plt.scatter(y\_test, predictions)

plt.xlabel('Actual Labels')

plt.ylabel('Predicted Labels')

plt.title('Daily Bike Share Predictions')

z **=** np.polyfit(y\_test, predictions, 1)

p **=** np.poly1d(z)

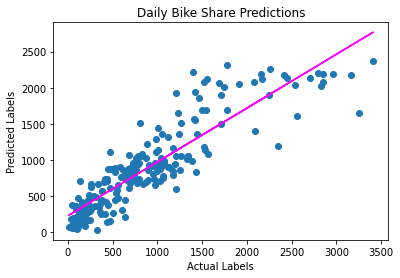
plt.plot(y\_test,p(y\_test), color**=**'magenta')

plt.show()

MSE: 105659.89424126886

RMSE: 325.0536790151265

R2: 0.7928604089398391



The pipeline is composed of the transformations and the algorithm used to train the model. To try an alternative algorithm you can just change that step to a different kind of estimator.

In [23]:



*# Use a different estimator in the pipeline*

pipeline **=** Pipeline(steps**=**[('preprocessor', preprocessor),

('regressor', RandomForestRegressor())])

​

​

*# fit the pipeline to train a linear regression model on the training set*

model **=** pipeline.fit(X\_train, (y\_train))

print (model, "\n")

​

*# Get predictions*

predictions **=** model.predict(X\_test)

​

*# Display metrics*

mse **=** mean\_squared\_error(y\_test, predictions)

print("MSE:", mse)

rmse **=** np.sqrt(mse)

print("RMSE:", rmse)

r2 **=** r2\_score(y\_test, predictions)

print("R2:", r2)

​

*# Plot predicted vs actual*

plt.scatter(y\_test, predictions)

plt.xlabel('Actual Labels')

plt.ylabel('Predicted Labels')

plt.title('Daily Bike Share Predictions - Preprocessed')

z **=** np.polyfit(y\_test, predictions, 1)

p **=** np.poly1d(z)

plt.plot(y\_test,p(y\_test), color**=**'magenta')

plt.show()

Pipeline(memory=None,

steps=[('preprocessor',

ColumnTransformer(n\_jobs=None, remainder='drop',

sparse\_threshold=0.3,

transformer\_weights=None,

transformers=[('num',

Pipeline(memory=None,

steps=[('scaler',

StandardScaler(copy=True,

with\_mean=True,

with\_std=True))],

verbose=False),

[6, 7, 8, 9]),

('cat',

Pipeline(memory=None,

steps=[('onehot',

OneHotEncoder(categories='...

RandomForestRegressor(bootstrap=True, ccp\_alpha=0.0,

criterion='mse', max\_depth=None,

max\_features='auto', max\_leaf\_nodes=None,

max\_samples=None,

min\_impurity\_decrease=0.0,

min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0,

n\_estimators=100, n\_jobs=None,

oob\_score=False, random\_state=None,

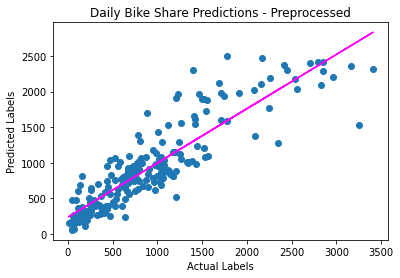
verbose=0, warm\_start=False))],

verbose=False)

MSE: 104542.41260363636

RMSE: 323.33019129619856

R2: 0.7950511615531994



We've now seen a number of common techniques used to train predictive models for regression. In a real project, you'd likely try a few more algorithms, hyperparameters, and preprocessing transformations; but by now you should have got the general idea. Let's explore how you can use the trained model with new data.

**Use the Trained Model**

First, let's save the model.

In [24]:



**import** joblib

​

*# Save the model as a pickle file*

filename **=** './models/bike-share.pkl'

joblib.dump(model, filename)

Out[24]:

['./models/bike-share.pkl']

Now, we can load it whenever we need it, and use it to predict labels for new data. This is often called *scoring* or *inferencing*.

In [25]:



*# Load the model from the file*

loaded\_model **=** joblib.load(filename)

​

*# Create a numpy array containing a new observation (for example tomorrow's seasonal and weather forecast information)*

X\_new **=** np.array([[1,1,0,3,1,1,0.226957,0.22927,0.436957,0.1869]]).astype('float64')

print ('New sample: {}'.format(list(X\_new[0])))

​

*# Use the model to predict tomorrow's rentals*

result **=** loaded\_model.predict(X\_new)

print('Prediction: {:.0f} rentals'.format(np.round(result[0])))

New sample: [1.0, 1.0, 0.0, 3.0, 1.0, 1.0, 0.226957, 0.22927, 0.436957, 0.1869]

Prediction: 95 rentals

The model's **predict** method accepts an array of observations, so you can use it to generate multiple predictions as a batch. For example, suppose you have a weather forecast for the next five days; you could use the model to predict bike rentals for each day based on the expected weather conditions.

In [26]:



*# An array of features based on five-day weather forecast*

X\_new **=** np.array([[0,1,1,0,0,1,0.344167,0.363625,0.805833,0.160446],

[0,1,0,1,0,1,0.363478,0.353739,0.696087,0.248539],

[0,1,0,2,0,1,0.196364,0.189405,0.437273,0.248309],

[0,1,0,3,0,1,0.2,0.212122,0.590435,0.160296],

[0,1,0,4,0,1,0.226957,0.22927,0.436957,0.1869]])

​

*# Use the model to predict rentals*

results **=** loaded\_model.predict(X\_new)

print('5-day rental predictions:')

**for** prediction **in** results:

print(np.round(prediction))

5-day rental predictions:

584.0

713.0

252.0

232.0

292.0

**Further Reading**

To learn more about Scikit-Learn, see the [Scikit-Learn documentation](https://scikit-learn.org/stable/user_guide.html).

**Challenge: Predict Real Estate Prices**

Think you're ready to create your own regression model? Try the challenge of predicting real estate property prices in the [/challenges/02 - Real Estate Regression Challenge.ipynb](https://newcomputeinst.eastus2.instances.azureml.ms/notebooks/Users/ml-basics/challenges/02%20-%20Real%20Estate%20Regression%20Challenge.ipynb) notebook!

**Note**: The time to complete this optional challenge is not included in the estimated time for this exercise - you can spend as little or as much time on it as you like!