```
1.66308583e-02,   1.66076861e-02,   1.82402545e-02,
1.63458761e-02,   3.26497987e-01,   6.04365775e-02,
1.13055290e-01,   7.79324766e-02,   1.12166442e-02,
1.53344918e-01,   8.41308969e-05,   2.68483884e-03,
3.46681181e-03])
```

Let's display these importance scores next to their corresponding attribute names:

```
>>> extra_attribs = ["rooms_per_hhold", "pop_per_hhold", "bedrooms_per_room"]
>>> cat_one_hot_attribs = list(encoder.classes_)
>>> attributes = num_attribs + extra_attribs + cat_one_hot_attribs
>>> sorted(zip(feature_importances, attributes), reverse=True)
[(0.32649798665134971, 'median_income'),
 (0.15334491760305854, 'INLAND'),
 (0.11305529021187399, 'pop_per_hhold'),
 (0.07793247662544775, 'bedrooms_per_room'),
 (0.071415642259275158, 'longitude'),
 (0.067613918945568688, 'latitude'),
 (0.060436577499703222, 'rooms_per_hhold'),
 (0.04442608939578685, 'housing_median_age'),
 (0.018240254462909437, 'population'),
 (0.01663085833886218, 'total_rooms'),
 (0.016607686091288865, 'total_bedrooms'),
 (0.016345876147580776, 'households'),
 (0.011216644219017424, '<1H OCEAN'),
 (0.0034668118081117387, 'NEAR OCEAN'),
 (0.0026848388432755429, 'NEAR BAY'),
 (8.4130896890070617e-05, 'ISLAND')]
```

With this information, you may want to try dropping some of the less useful features (e.g., apparently only one `ocean_proximity` category is really useful, so you could try dropping the others).

You should also look at the specific errors that your system makes, then try to understand why it makes them and what could fix the problem (adding extra features or, on the contrary, getting rid of uninformative ones, cleaning up outliers, etc.).

Evaluate Your System on the Test Set

After tweaking your models for a while, you eventually have a system that performs sufficiently well. Now is the time to evaluate the final model on the test set. There is nothing special about this process; just get the predictors and the labels from your test set, run your `full_pipeline` to transform the data (call `transform()`, *not* `fit_transform()`!), and evaluate the final model on the test set:

```
final_model = grid_search.best_estimator_

X_test = strat_test_set.drop("median_house_value", axis=1)
y_test = strat_test_set["median_house_value"].copy()

X_test_prepared = full_pipeline.transform(X_test)
```

```
final_predictions = final_model.predict(X_test_prepared)

final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)   # => evaluates to 48,209.6
```

The performance will usually be slightly worse than what you measured using cross-validation if you did a lot of hyperparameter tuning (because your system ends up fine-tuned to perform well on the validation data, and will likely not perform as well on unknown datasets). It is not the case in this example, but when this happens you must resist the temptation to tweak the hyperparameters to make the numbers look good on the test set; the improvements would be unlikely to generalize to new data.

Now comes the project prelaunch phase: you need to present your solution (highlighting what you have learned, what worked and what did not, what assumptions were made, and what your system's limitations are), document everything, and create nice presentations with clear visualizations and easy-to-remember statements (e.g., "the median income is the number one predictor of housing prices").

Launch, Monitor, and Maintain Your System

Perfect, you got approval to launch! You need to get your solution ready for production, in particular by plugging the production input data sources into your system and writing tests.

You also need to write monitoring code to check your system's live performance at regular intervals and trigger alerts when it drops. This is important to catch not only sudden breakage, but also performance degradation. This is quite common because models tend to "rot" as data evolves over time, unless the models are regularly trained on fresh data.

Evaluating your system's performance will require sampling the system's predictions and evaluating them. This will generally require a human analysis. These analysts may be field experts, or workers on a crowdsourcing platform (such as Amazon Mechanical Turk or CrowdFlower). Either way, you need to plug the human evaluation pipeline into your system.

You should also make sure you evaluate the system's input data quality. Sometimes performance will degrade slightly because of a poor quality signal (e.g., a malfunctioning sensor sending random values, or another team's output becoming stale), but it may take a while before your system's performance degrades enough to trigger an alert. If you monitor your system's inputs, you may catch this earlier. Monitoring the inputs is particularly important for online learning systems.

Finally, you will generally want to train your models on a regular basis using fresh data. You should automate this process as much as possible. If you don't, you are very