Milestone 3: Group Model Performance Evaluation

The coding portion of this final milestone will be fairly short and sweet. This exercise is designed to give you (brief) experience with evaluating a model's performance after it has been deployed, given new data that has accrued since the deployment date. However, it is not intended to be a heavy or time consuming obligation for your group because you have other deliverables to put together in preparation for your final deliverable / presentation. So I'll walk you through the experience so that you can *have* the experience, but it shouldn't take long.

The "Future" Dataset

Of course, we haven't actually deployed our models, so we'll be simulating the accrual of new data using the "holdout" sets of data that I've prepared for you. These datasets have been listed in a separate section of the "Group Project Datasets" page on Learning Suite, and you can read them in from their Dropbox links as usual. Just do me a favor and at least *pretend* that you're accessing the new data via another business system/API several months down the road. :)

Milestone Requirements

Your task for this milestone will be to read in your "future" data along with most of the things you saved out to your disk at the end of your last group milestone, create predictions for that future data, and finally compare performance of this new data with the performance you observed (in the test set) during your model training effort. Sounds fun, right? Well it's going to be.

Read in All the Things

Before we read everything in, just note that you're going to need to copy the objects below from wherever you stashed them at the end of your last group milestone into this repository folder on your computer. Then you can use the `setwd()` command provided in the script to make sure you are working in the correct location. **Please make sure you do this as it will allow us to grade this portion of the milestone in a consistent way across all of the groups.**

After copying the files into this directory, you can read each of the objects below into your environment, assigning each to the object name indicated:

- `test_set_results` : the tibble of test set results, read in from the `test_set_results.csv` file you saved out in the last milestone.
- `trained_workflow` : the final, trained workflow, read in from the `trained_workflow.rds` file you saved out in the last milestone.
- `future_data_raw` : use this to read in your "future" dataset from the Dropbox link on Learning Suite. (If you have multiple files, you don't really need to worry about this one because you can read each into its own respective `_raw` file. The grader won't care.)

1. After doing the above, you should have added `test_set_results` and `trained_workflow` to your environment. (And `future_data_raw` for most of you, but no grading will be done on that.)

Light Prep Work on Future Data and the Test Set

Depending on what you ended up doing in your model building script *prior* to creating your split object, you may need to add those same operations to your `future_data_raw` data. Specifically, If you applied any

mutations, renaming, joins, or other alterations to the data BEFORE you used it with `initial_split()`, you'll need to apply those to `future_data_raw`, and then save the result as `future_data`.

> *Note: if you attempt to get predictions with your future data in the next step and your data isn't structured the same or if there are missing columns or if the columns aren't named the same as they were when they went into your `initial_split()` function on the last deliverable, you're likely to get errors. This should **hopefully** be as easy as copying and pasting in any custom logic you used before the `initial_split()` function in your prior script.*

Similarly, if your model is a classification model, there will be two columns in your `test_set_results` data that came in as regular character/numeric columns that need to be converted. This is because you likely converted your outcome column to a factor data type prior to modeling, so you'll want to do the same `as.factor()` conversion to (a) your outcome column (which will vary from group to group) and (b) the `.pred_class` column (which will be the same for all groups with classification models). I would just take care of these factor mutations as the data is read in from `test_set_results.csv`, which you've already saved as `test_set_results` above.

2. After doing the above, you should now have `future_data` ready in your environment.

Score your Future Data

Now you can use your `trained_workflow` object to make predictions with your future data. Bind these predictions to the `future_data` tibble (or, to a subset of the columns from `future_data` including at least your outcome variable) so you can do some evaluation of how your model performed on this holdout dataset. Save the result as `future_results`.

> *Note: If your model is a classification model, you'll need to bind **both** a `.pred_class` column as well as the probability columns that have the format `.pred_[outcome_value]`. This will involve two separate `predict()` operations.*

3. After doing the above, you should now have `future_results` in your environment.

Workflow Objects

Recall the set of custom evaluation metrics that you used in your last deliverable (which you saved to the variable called `custom_metrics`). We are going to use each of those metrics to compare the performance you saw in your test set during training with the performance of the predictions you just created for your future data. To do this, you'll need to run both your `test_set_results` and your `future_results` through a series of performance evaluation functions, eventually producing a nice side-by-side comparison tibble (named `performance_comparison_tibble`) that compares the performance of the test set and future data predictions as you see below. (This will likely involve some row binding and pivoting.) Note that you may have more or fewer (and possibly different) metrics from the example I'm showing below, but the *columns* in your comparison tibble should match mine.

(A table like the one below will likely end up somewhere in your final presentation and/or summary report.)

```
# A tibble: 4 × 3
  .metric  test_set_results future_results
  <chr>               <dbl>          <dbl>
1 sens                0.620          0.607
2 spec                0.883          0.841
```

```
3 accuracy            0.787          0.755
4 roc_auc             0.835          0.815
```

> *Note: If your model is a classification model, remember that the* `roc_curve()` *uses a predicted* ***probability*** *column for the* `estimate` *parameter rather than the* `.pred_class` *column like the rest of the classification performance metric functions.*

4. After doing the above, you should now have `performance_comparison_tibble` in your environment.

Performance Comparison Plot

Now you'll need to formulate a single, clear, well-labeled plot that summarizes your performance comparison at a glance. This will be a plot very similar to the one you created toward the end of your previous deliverable, except that you'll be showing performance of your `test_set_results` together with that of your `future_results`, probably using color, fill, etc., to visually compare the two and show whether your model generalized well to your (simulated) future data. I'm not going to be overly prescriptive here, but your plot needs to be compelling and clear. You'll be including this plot as a slide in your presentation as well as in your written summary report as a part of this milestone. **You should know what your plot means and be able to defend it when questioned during your presentation.**

To give you an idea of what type of plot would work for this "single plot summary", I've created two example plots (one for a classification model and the other for a regression model), which you can find in the `/plots` directory.

Save your plot as `performance_comparison_plot`. Once you are happy with your plot, you can again use the code snippet below to save your plot out as a `.png` file:

```
performance_comparison_plot %>%
  ggsave('perf_comparison_classification.png', plot = .,
         device = 'png', width = 14, height = 9)
```

5. After doing the above, you should now have `performance_comparison_plot` in your environment.

Read this before submission

In order for us to grade your submission properly, we're going to need access to the same `test_set_results.csv` and `trained_workflow.rds` files you read in at the beginning of this script. So make sure those two files get added to the Box folder that you share with us as a part of your final submission. (Details of that Box folder sharing are found in the main "Milestone 3" document that describes the requirements for the presentation and summary report.) As with the last milestone, the `test_set_results.csv` and `trained_workflow.rds` files will be excluded from GitHub when you commit and push your submission.

Final Run

Please do the following to make sure your code runs without errors and passes the naming checks by doing the following:

1. Restart your R session (Session >> Restart R).
2. Run your entire script from the beginning, watching for any errors along the way.

3. After your script is finished, make sure that you pass all tests in the code in the "Naming Checks" section at the bottom of your script.

Housekeeping

Once you have completed the steps in the prior section, please check all of the following and make adjustments as needed. (Failure to do the housekeeping steps below will likely cause an error in our grading process, and that will make it hard to give you the right credit for your hard work.)

1. Please COMMENT OUT the `setwd()` command near the beginning of the script before committing and pushing your code.
2. Please also comment out any code where you are either using the `View()` or `glimpse()` functions. These cause issues with our grading procedures.
3. If you installed any new packages as you completed the assignment (using `install.packages()`), please comment those installation commands out as well.
4. Lastly, please ensure that you have **NO** absolute references in your code that are not commented out. (An absolute reference is something like: `/Users/YourName/Documents/GitHub/...` or `C:/GitHubProjects/...` .) These also throw errors and make it hard for us to grade your work.

Save, Commit, Push

You're now ready to do all three of the following:

1. Save your script.
2. Do a final commit with your git tool to stage your work for submission.
3. Push your changes up to your repository. And you're done!