

Pandas Data Wrangling Project – Phase 5

Introduction

Bruno is convinced there's even more we can learn by creating new *features* in our dataset that capture customer behavior patterns in different ways.

(Again, feel free to continue building on your notebook from the previous phase of the project, OR export the end-state DataFrame from that phase as a CSV file and use it as a starting point here.)

Task 1: Create Tenure Category Column

To kick off this phase, we'll use the powerful *map* and *apply* methods to transform existing data and create new columns.

Bruno's first request: "I'd like to categorize customers by their tenure into meaningful segments. Can you create a 'tenure_category' column that groups customers as 'New' (0-12 months), 'Established' (13-36 months), or 'Loyal' (37+ months)?"

HINT: Define a custom function that categorizes tenure values, then use the map method to apply it to the tenure column.

After creating the column, analyze churn rates by tenure category to see what patterns emerge.

Task 2: Create Has Family Indicator

Bruno has a theory that customers with family ties – either a partner or dependents – might behave differently. Create a 'has_family' indicator that's 1 if the customer has a partner OR dependents, and 0 otherwise.

HINT: When you need to check multiple columns per row, use apply with axis=1. This passes each row to your function as a Series.

After creating the indicator, compare churn rates for customers with and without family to see if Bruno's theory holds up.

Task 3: Build a Predictive Model – Stage 1 (Baseline)

Finally, we arrive at the (even more) exciting stuff!

Bruno has been reading up on predictive analytics and wants to explore whether we can predict which customers are likely to churn.

To accomplish this, we'll build a predictive model using scikit-learn and compare performance across three stages:

1. **Baseline:** Using only the original numeric columns
2. **With `get_dummies`:** Adding categorical columns converted to numeric format
3. **With engineered features:** Adding a custom feature to (hopefully) improve model performance

This will demonstrate the cumulative value of each technique.

Start by importing the necessary scikit-learn components, then build a baseline model using only the original numeric columns (`tenure`, `monthly_charges`, `total_charges`, and `senior_citizen`).

Also, make sure to create a new `DataFrame` with just those original columns, as opposed to overwriting the original `customers_df` `DataFrame`.

Task 4: Build a Predictive Model – Stage 2 (with `get_dummies`)

Now use `get_dummies` to incorporate categorical columns into the model. But before you do, think carefully about *which* columns to include.

Some columns should be excluded from the model entirely:

- **`customer_id`:** This is just a unique identifier with no predictive value – including it would cause the model to memorize individual customers rather than learn general patterns.
- **`signup_date`:** Raw dates aren't meaningful to a model. While we *could* extract useful features from dates (like year or month), the raw date string itself won't help.

Also hold off on including the derived features created earlier in the project (like `tenure_category`, `has_family`, etc.). Adding too many features without careful consideration can actually *hurt* model performance – a phenomenon sometimes called the "curse of dimensionality." For now, see how much improvement you get from simply converting the categorical columns to dummy variables.

Once again, make sure to create a new DataFrame including just the columns you want to train the model with at this stage, as opposed to overwriting the original customers_df DataFrame.

Task 5: Build a Predictive Model – Stage 3 (with Engineered Features)

Now add a custom engineered feature that captures information not directly available in the original columns or from simple dummy encoding, that will *also* potentially improve the predictive performance of the model.

Specifically, your task in this stage is to create a new feature called `is_high_value` that flags customers in the top 25% of monthly charges – these are customers who might warrant special retention attention. Then incorporate this feature into the model – alongside the original numeric columns AND the "dummy" columns (and again via a **new** DataFrame, to avoid overwriting the original customers_df DataFrame) – and see if it improves performance.

HINT: You can either use the describe method to determine the 75th percentile of monthly charges, or – if you're feeling adventurous – try using the Pandas quantile method to find the threshold. Then, regardless of the approach you take, simply employ a boolean comparison and convert the results to integer.

WARNING: long-winded disclaimers below!

Now you may be wondering why I'm "giving" you this particular feature rather than asking you to invent one from scratch.

Well, the reality is that feature engineering for machine learning is as much art as science (and there's a lot of science too!). It requires both intuition and domain knowledge to do effectively, and even experienced data scientists often experiment with many features before finding ones that work.

So why is `is_high_value` *potentially* capable of improving our model?

Generally speaking, because it captures information that isn't directly encoded in the existing features. For example, while `monthly_charges` tells us the raw amount, `is_high_value` explicitly flags a meaningful business segment - customers whose revenue makes them particularly important to retain.

But more specifically, this kind of threshold-based feature often helps models by highlighting *non-linear* relationships in the data - the kinds of relationships we should be trying to help the model "understand" with new features.

And if you found any of the above remotely interesting, I **absolutely** encourage you to experiment with creating your own features! Just keep in mind that not every feature you dream up will improve performance - and that's perfectly normal.

Task 6: Compare Model Performance

Display a summary comparison of all three model stages (baseline, dummies, and full model with engineered features) to see the cumulative improvement in accuracy.

Task 7: Export to CSV

Finally, export the `core customers_df` DataFrame to a CSV file – we'll need it later!