Chapter 5: Automating and Optimizing Engagement Scores

~2,200 WORDS

An Engagement Score is a valuable thing. If it withers on the vine, then the time, resources, and passion that went into creating it is lost, and the morale of staff who built it will sour. While I like to build my Scoring rubric with a combination of staff collaboration and historical data in Excel, it is critical that the Score is updated, easy to use, and strategically implemented.



This chapter provides an in-depth guide to building and automating Engagement Scoring systems in Slate, focusing on query construction, automating procedures, and making the Score itself easy and intuitive to use.

Queries

Automating Engagement Scoring in Slate requires building Queries that aggregate data points and apply Scoring rubrics to student behaviors. These Queries then update automatically on your schedule as well as any additional batch updates as needed such as admission committees or large events. This provides admission teams with current insights into student engagement. Later, you will use translation codes to create intuitive and time-specific Score bands that use your current data.

The first step in automating Engagement Scoring is constructing queries that capture the key behaviors indicative of student interest. These behaviors might include event attendance, email interactions, portal logins, application progress, etc. By creating a query that tracks these interactions, you can calculate a comprehensive Engagement Score for each student. Queries will be needed to build your Scoring rubric, calculating live student Scores, and building any required bonuses.

While Configurable Joins has been live since 2021, many institutions have yet to meaningfully switch to them. And even fewer are leveraging them in powerful ways. Even if you haven't tackled them yet, Engagement Scoring will be an excellent field to learn on. And the feature engineering process (and chapter) of Predictive Modeling will get you across the finish line. You'll go pro in no time. And I think I have exhausted the football metaphors (for now).

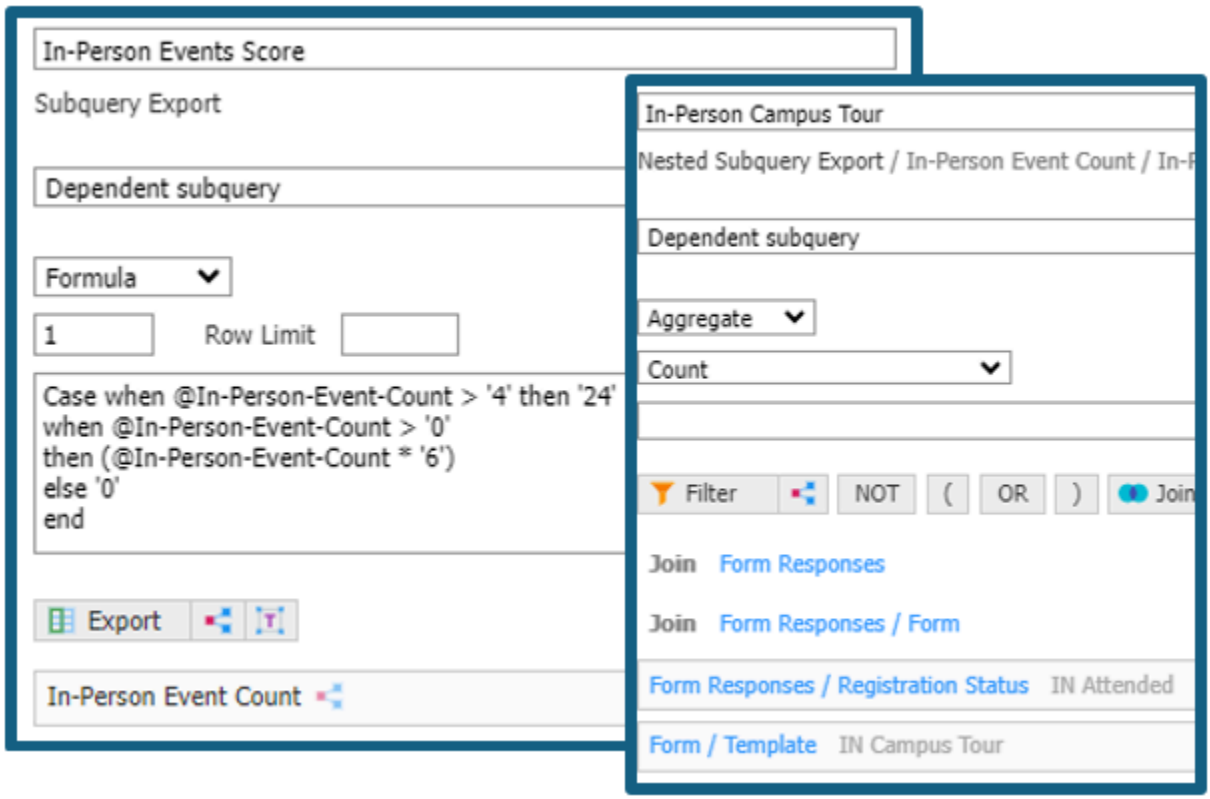


Much like this book is not a statistics textbook, it's also not a Configurable Joins 101 guide. There are wonderful resources in the community and by Partners that will teach you a lot about starting and using CJs. This book will show several screenshots though and walk through the logic.

I like to think of Configurable Joins as a drag and drop interface for SQL. It's a no-code/low-code way to connect different datatables, aggregate behaviors, and engineer features. The vision for my next book is Adaptive Enrollment Management. While the third act of this book applies the AEM lens to using Scoring and Modeling, there are a lot of really fun CJs you will build with a full AEM implementation.

As a general recommendation, create nested subquery exports to clearly define each point of the filters and formulas. This makes understanding the factors easier and updates faster.

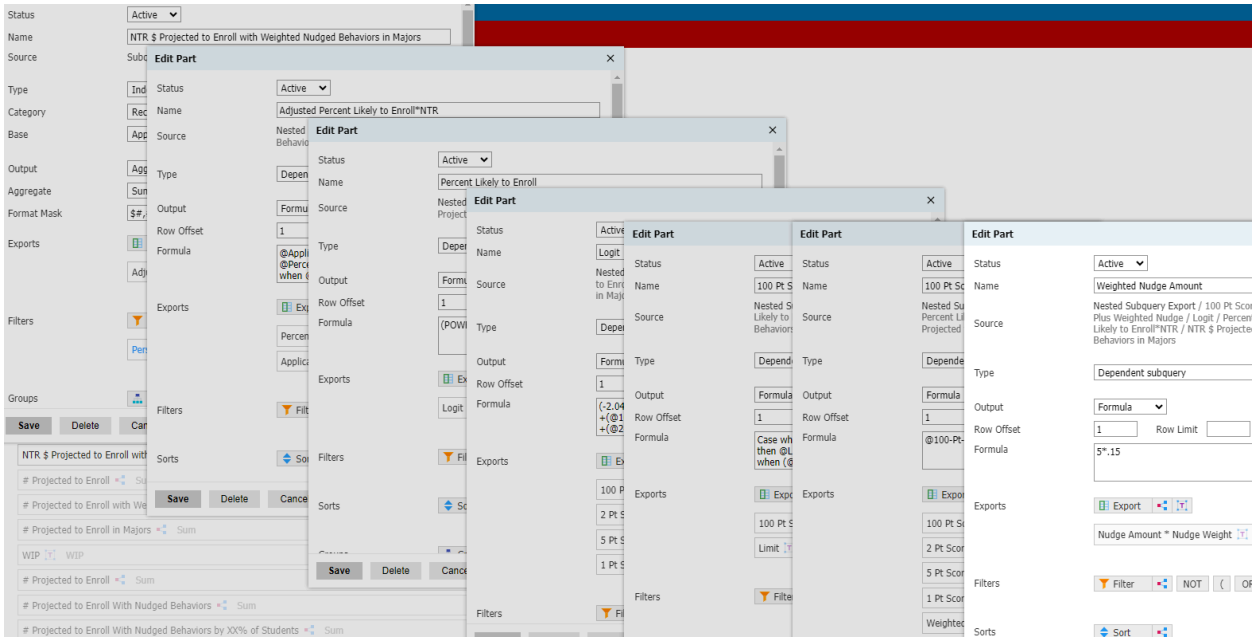
Whether you only update once a year or a different staff ends up updating the query, this at-a-glance spelling out will help whoever is working with the query in the future (or even troubleshooting). I typically do the same with Excel and my other work.



This process may cause you to recreate a few exports as you find you need everything to be nested one layer deeper to put the final touches on the export with a Formula, Concatenation, or Translation code.

This becomes especially true when you get into complicated subqueries. When dealing with time-anchored behaviors (e.g. behaviors happening before the application deadline or that happen in a narrow window after Decisions/Transfer Evaluation/Financial Aid Award is received), this can become even more troublesome but are more in the domain of feature engineering with Predictive Modeling. Most important of all is clearly building your reports. With the different slicing of data, increased number of staff in the report and likely longer use, unclear report building can incapacitate a project.

When I first started building the reports to project Net Tuition Revenue with incorporated nudges (e.g. what is the NTR if 20% of students attend Admitted Student Day), I repeatedly rebuilt this export which had several nested exports, going 7 layers deep.



And don't even get me started on the number of times I rebuilt the required exports and filters to create the emails for my Course Insights Engine. That took the outstanding courses for the student, what was being offered, and success indicators to not only recommend classes for students to register for, but also flagged the courses they were more likely to be successful in. Well, maybe. As I write this, AEM and that tool are both books I'd like to write.

Another STRONG recommendation for Scoring and Modeling is to create Slate Fields for your behavior. While you may be able to run these calculations in a query,

depending on the number of exports, the size of your population(s), factors like bonus, frequency of the automated updates, and other considerations, dedicated fields will not only make life much easier, they will make troubleshooting a much better experience.

While you may run the query for all relevant records, having static fields does allow you to compare the field's existing value with the current math for needed updates. Not to mention the need to replicate rubric changes wherever you need to do these calculations.



Engagement Scores can be something of a gateway tool, so the flashing marquee recommendation is to be extremely clear and exact with your naming convention. You have one Score now, but what about the MarComm subscore that aggregates a few behaviors. Or the separate rubric used for International students? Or rubrics for Inquiries, Applicants, Admits, etc. The separate Score for current students? Alumnae? Or fields for Application Modeling? Enrollment Modeling? Retention or Philanthropy Modeling? And those models will likely need additional models to account for the differences in predictors for your different populations.

That is all to say, don't name your field "Event Attendance" and call it a day.



Automation

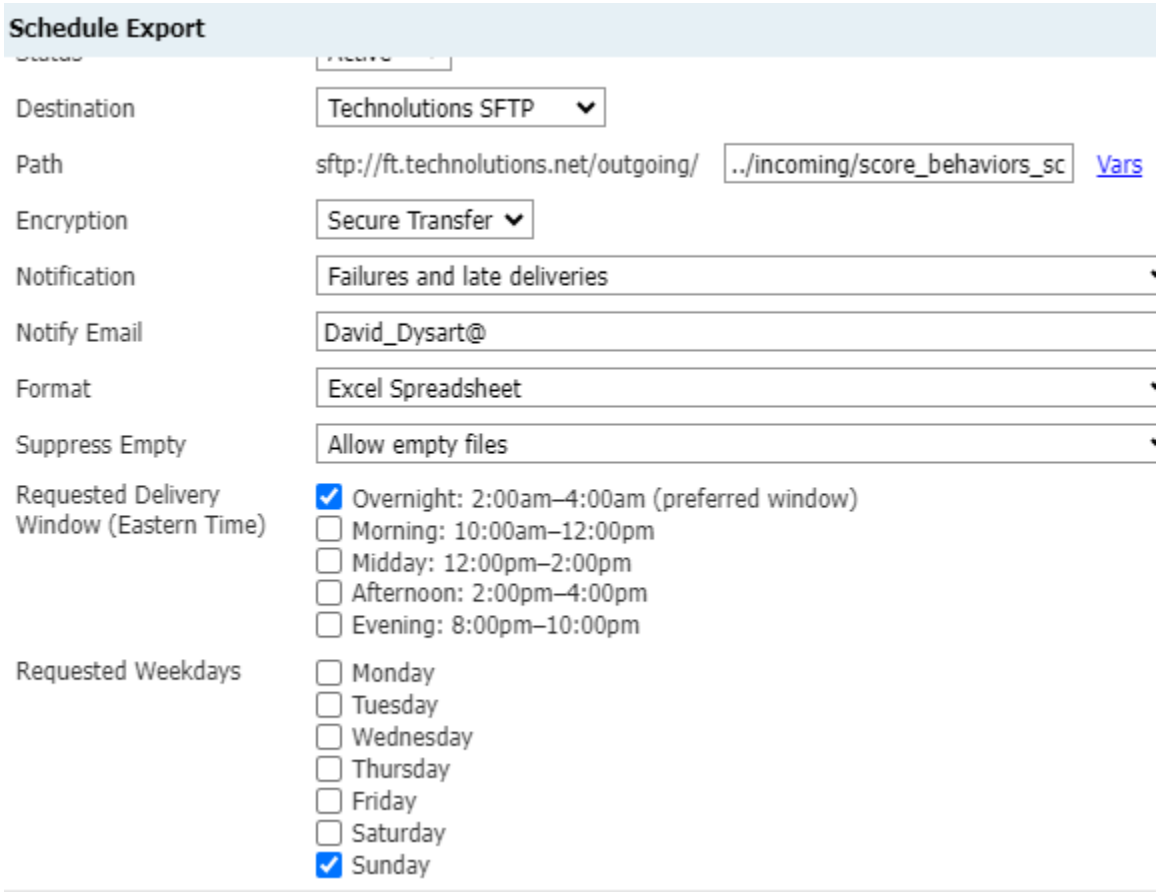
To ensure that Engagement Scores remain current, you can leverage Slate's automation features to schedule regular updates. By automating Score calculations, your team will focus on interpreting the data and making informed decisions, rather than manually updating Scores.

1. **Schedule Query Runs**: Use Slate's Schedule Export to automate query execution at regular intervals. This updates Engagement Scores on the chosen schedule, reflecting interactions available at each successful refresh.

a. Note that the path needs "../incoming/" to place the query results in the "Incoming" folder which is where the Source Format will need them to be to pick it up and import it.

b. Set up an email to receive notification for at least Failures and late deliveries.

c. Schedule the more intensive queries for slower days/times if possible. If there are multiple queries that need to run, staggering the windows will be better error-proofing than trying to ingest them in the appropriate order in the same window.



2. **Automate Data Imports**: Automate the import of data with a Source Format

   a. There are a few notes on the General Tab -

      i. Set Status to Active

      ii. Give it a descriptive Name

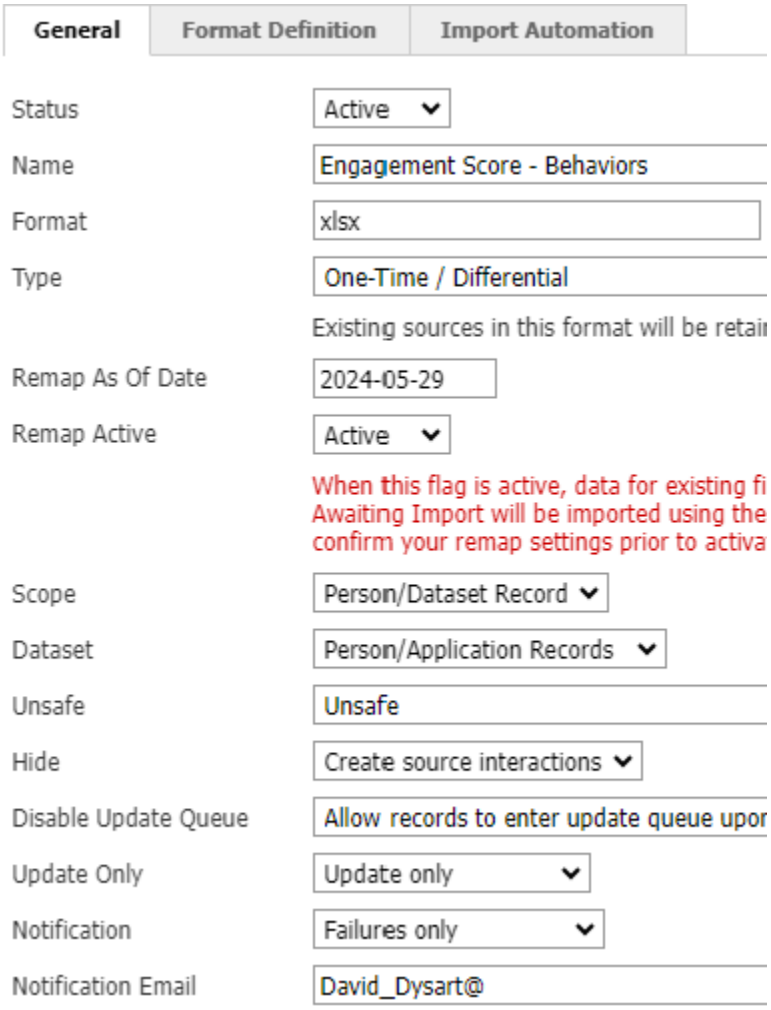
iii. Remap As Of to capture the earliest file and set Remap Active to Active

iv. Set to Unsafe so current Applicants can have fields updated (even if you think you set your new fields to Unsafe. No one wants to spend 3 hours troubleshooting the Score just to find out in-person events weren't updating because they were Safe)

v. Set Update Only

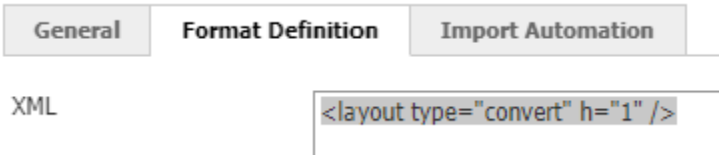
vi. Set Notifications to at least email relevant staff on Failures

vii. Adjust other settings as needed.



b. Under the Format Definition Tab, this code in the XML box tells Slate that the first row in your file is the Header with Column Names

i. <layout type="convert" h="1" />



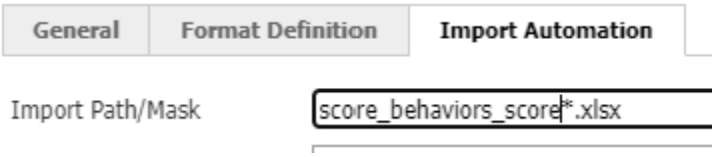
c. The last important piece to update is in the Import Automation Tab. You must add the name specified in the Query's Scheduled Export. This tells Slate what file to pick up and ingest. This picture example uses an asterisk wildcard to look for the file name and any additional text where the "*" is at. This is important if you are adding a timestamp.

d. If you have multiple Scheduled Exports that may be picked up, you will need to revise your naming convention. "score_behaviors_score*.xlsx" would pick up all three of these files:

   i. score_behaviors_score.xlsx

   ii. score_behaviors_score_09082024.xlsx

   iii. Score_behaviors_score_pt2_bonuses.xlsx



Some of the automation and calculations can be offloaded to Rules. In my experience, I prefer running everything through the Query and Source Format process. But you can experiment with including Rules into your process to make your own decision. Ultimately, the processes and build are similar.

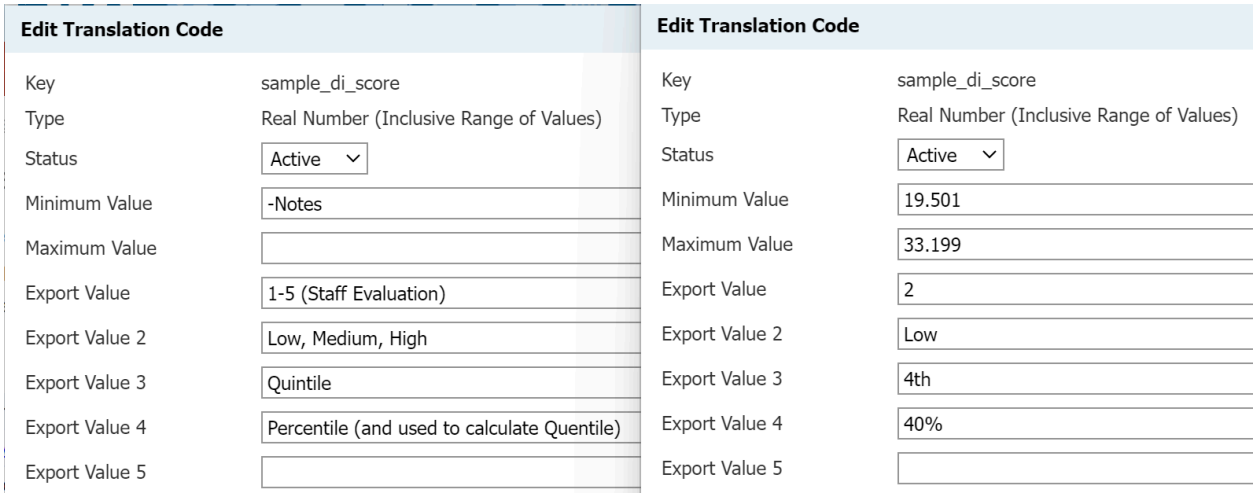
Data Transformation and Integration

In the earlier chapter on creating a Scoring rubric, you created a Score where 100 points was the maximum possible Score. And truly, I stand by that recommendation generally. It helps create a balanced Score as it provides an architecture and enforced scarcity. You have no option other than to be mindful and judicious with the behaviors and rubric.

The other side of that coin is it always provides a point of reference for interpreting the Score of an individual or group. Given the encompassing nature of different forms of engagement and progressive nature of the recruitment cycle. Students simply aren't going to score a 100.

Scoring an 85 isn't a B. A student Scoring 20 at the beginning of the cycle might be one of the highest Scores. By the time of Decision Release, maybe that 20 is on the lower side. And while it may be easier to fit behaviors into a 100-pt score, your staff may be socialized to a 1-5, A-F, or 1-10.

A user-friendly framework comes with transforming the Engagement Score with a Translation Code.



Translation Codes give several options. Any Score between 10 and 30 can become one of the following in a query, Dashboard, Report, Portal, Voyager, or as a Merge Field.

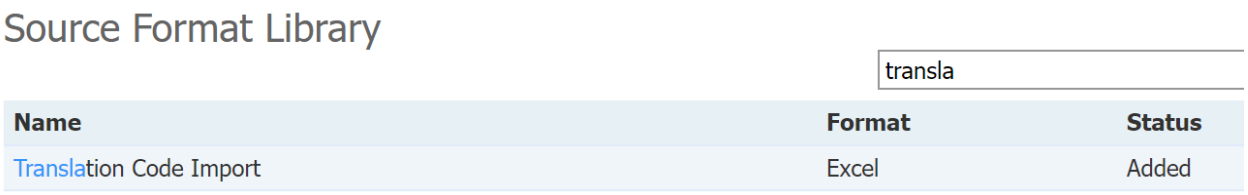
The first Export Value pictured is where their Score lands on a 1-5 scale. We'll detail how to set this Score band shortly.

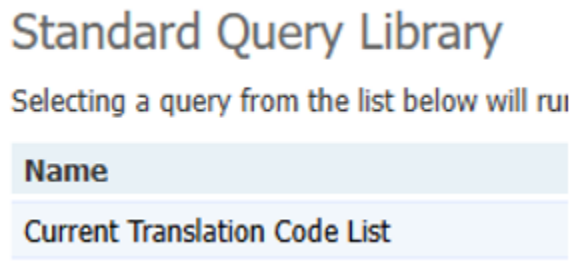
Export Value 2 uses a description like "Low", "Medium", and "High". The third and fourth Export Values are more descriptive of the data in your institution/subgroups.

The Translation Code will not inherently scale to your instance's data (and I would not recommend doing it internally in Slate). Translation Codes can be created and updated with an Excel file through Upload Dataset.

You may need to activate the Source Format from the Library. If you were not using this process to create and update Translation Code values, this may truly unlock the tool at

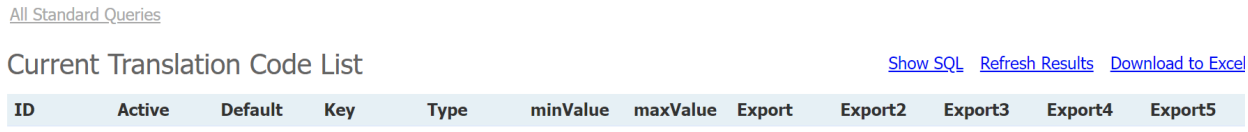
your institution. There are so many ways Translation Codes can transform your processes, and being able to batch them through Excel makes it so much faster and easier to manage than manually interacting with Slate's user interface.





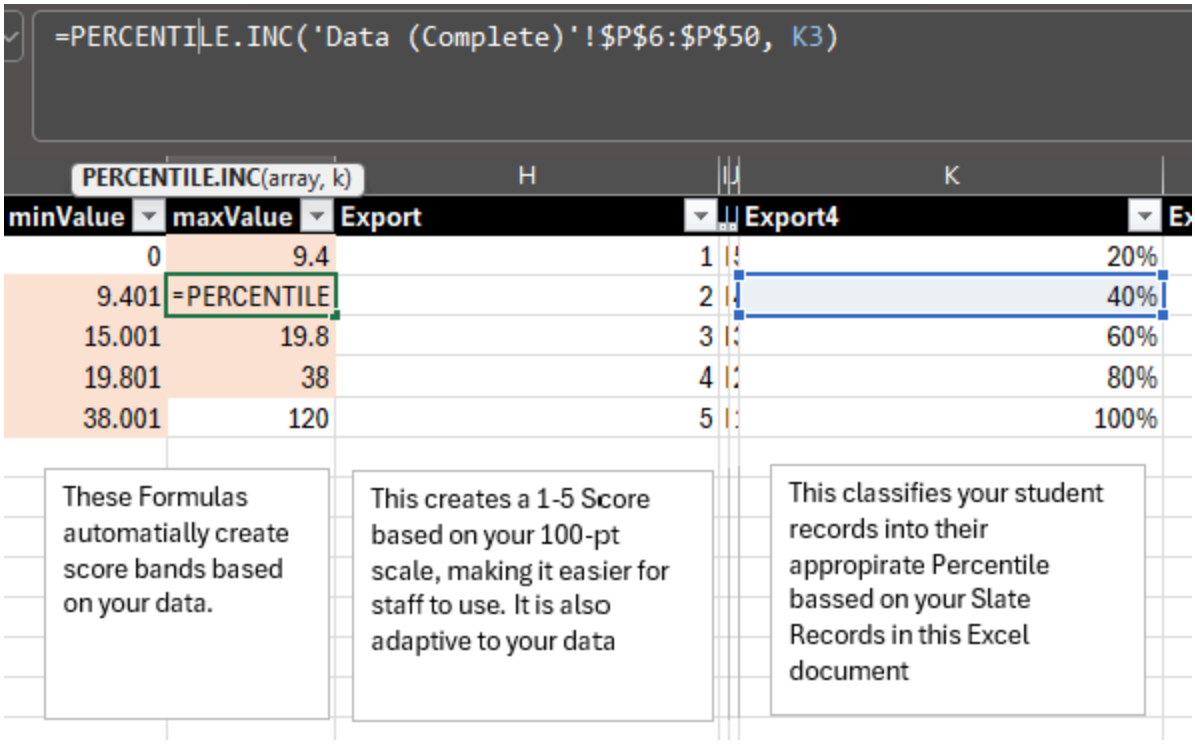
After activating the Source Format, you will need to download the current list/template spreadsheet from your Standard Query Library.

You do not need to upload all values every time to upload the spreadsheet. Just the ones you want to update or create. If you are updating a value, you will need the ID from the file, otherwise, Slate will create a new key instead of updating what you wanted to update. Not the funnest time.



What I would recommend is building a query with your relevant group of students that you need behavior data on. Export this data and add it as a tab in your Translation Code Upload Excel file.

The Score bands are created by referencing the Quintiles in your Translation Code Uploads tab, making the bands specific to your current pool of students you just downloaded. The Translation Codes must be the left-most tab in your file when uploading to Slate's Upload Dataset tool.



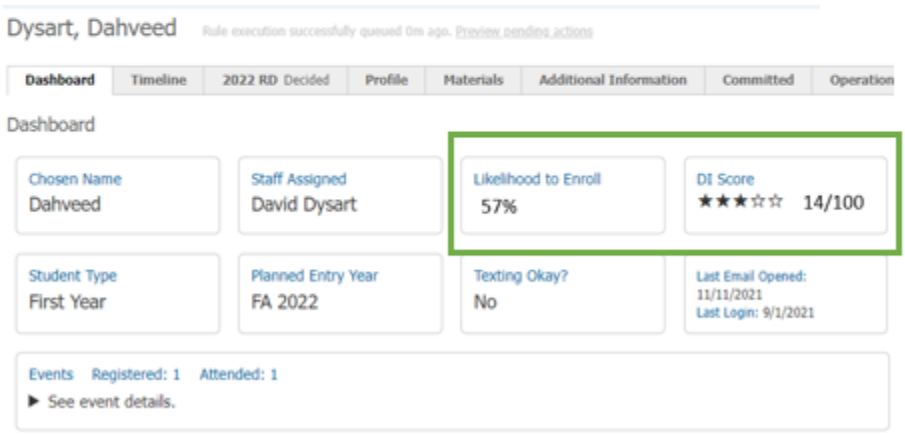
When you want to refresh your Translation Code bands, you can either use the live sync function with your Query or Download the updated data from Slate and simply upload the file again since the formulas are all live, making this a quick process that can be completed as often as needed through the cycle.

So not only does the Translation Code provide context to what a specific Score means, it is specific to the current time in the cycle.

Later chapters will cover several uses for this Score, but I want to make one more recommendation that makes the Score itself more utilizable on a day-to-day basis.

Embedding the Score (and Predictive Modeling values) into a Dashboard on the Slate Record provides additional context for the student as people interact with them. This example comes from a wonderful former colleague - Paloma Barragan who was redesigning the Dashboard at the time and integrated the new Scores while doing it.

Depending on which step of the funnel the student was in, it flipped between predictive models - Likelihood to Apply and Likelihood to Enroll. It also uses the raw Engagement Score as well as translating it to 5 stars.



Conclusion

Automating Engagement Scores gives you a powerful tool for optimizing recruitment and admissions strategies. By building queries that aggregate data points and applying Scoring logic, you can create dynamic Scoring models that provide valuable insights into student engagement.

With Slate automating the update process (with additional manual updates easy to do in Batch Updates), the Score is always an up-to-date reflection of the student's engagement with extremely low maintenance. And the quick process to update Translation Codes makes the context of the Score also timely and relevant.

As the lights dim on Act I, the bright promise of Act II will explore the role of Predictive Modeling in higher education.



Chapter 9: Implementing and Automating the Model

~2,300 words

Now that you have a better grasp on the data and considerations around Predictive Modeling, you have a few forks in the road ahead of you. With over a dozen statistical models and machine learning algorithms, you will need to find one that fits the outcomes you need, maps well to your data and staffing expertise, and fits within your technical limitations. From there, you will need systems to not only build but run your Model. While your Model may inform these decisions, if your systems are a limitation, then finding the right model that can be run, implemented, and used may be a decision for the end of this road.

For the purposes of this book, we will be using SPSS to build a Binary Logistic Regression Model. The model will then be replicated in Slate to run and apply to your records automatically in a similar process to Engagement Scoring in Act I. This process includes transforming your model results into a 0-100% likelihood Score.

Choosing Between SPSS and R (Or Other Systems)



One of the first major decisions when building a Predictive Model is selecting the right system for Modeling and execution. Building the model requires statistics software like SPSS, Stata, SAS, or R. Many institutions have licenses for SPSS, and staff are more likely to have been trained on SPSS. R is a great, open source, and free alternative that is far more robust in its total possible functionality, especially with machine learning algorithms. The best analogy for Slate users would be that:

- SPSS is the Standard Template Library. Much easier to stand up and run with. Preset user interface and functionality.

- R is writing your query in SQL (not to be confused with the fact that the programming language for R is extremely similar to Python). The sky is the limit… if you already know how to fly.

- RStudio is a combination of Configurable Joins and Suitcases. Better interface than a blank screen. Open-source packages help with the lift. Learning curve is still steep compared to SPSS.

R is a powerful, open-source language widely used for statistical computing and machine learning. Commonly photographed in the hands of data scientists because it offers a vast number of packages that support sophisticated algorithms and statistical models. However, R comes with a steeper learning curve, particularly for users who may not have a programming background. R's syntax-heavy interface, while flexible and scalable, can be intimidating.

Alternatively, SPSS is a robust, user-friendly statistical analysis tool that many professionals in higher education are already familiar with. One of the primary advantages of SPSS is its intuitive graphical user interface (GUI), which allows users to perform complex analyses without needing to code. In SPSS, you can select your data, run models, and produce results through a series of clicks. Paste-ing these operations or learning the syntax behind the operations does offset some of the advantages of R, but only partially at that.

For the purposes of this book, we will focus on using **SPSS** to build our Predictive Models, given that many participants already have experience with this software. However, for those interested in learning **R**, keep in mind that it's an excellent tool for expanding your skills in machine learning and advanced statistical Modeling.

SPSS: Interacting with the Software

SPSS is designed to be as user-friendly as possible, with a few key window types that you'll interact with:

Data View: This is similar to a spreadsheet where you input and manage your dataset. Before you can begin building a model, you need to ensure your data is structured properly.

Variable View: This is where you define the characteristics of each variable in your dataset. You can label variables, define variable types, and set value labels.

Output View: This is where SPSS shows the results of your analysis. Whenever you run a model, SPSS will generate tables, charts, and statistics here.

Syntax View: This is the hidden gem of SPSS. While the GUI is a fantastic tool for those who prefer a click-based interface, syntax allows users to write or paste code that centralizes repetitive, slow tasks and reduces errors.

Integrating Predictive Models with Slate CRM

Predictive Models are only as impactful as their ability to be deployed. After you've built your model in SPSS and generated the unstandardized regression coefficients, you'll want to program the math into Slate to automate predictions for current and future students. Slate offers seamless capabilities for importing and exporting data as well as running models internally, making it an ideal platform for higher education Predictive Modeling.

If you are using a separate system or CRM but still want to run the model in-system, ensure it can export/import data and perform calculations in a formula like SQL.

For Slate, this involves using Configurable Joins to aggregate behaviors and data for Predictive Modeling and to export the training data.

Much of the Slate-side of this build uses the same process of automated Engagement Scoring by exporting the data in Slate Queries and re-importing them automatically every weekend with a Source Format.

Because Slate has to replicate not only the behavior counts but also the math of the Binary Logistic Regression, this will take a couple more rounds of Queries/Source Formats. Not to worry, This just involves a little copy and pasting from your SPSS results into Slate formulas.

After running the binary logistic regression in SPSS. You will need the unstandardized regression coefficients. These represent the change in the log-odds of the outcome for a one-unit increase in the predictor variable, holding other variables constant. These coefficients show how each predictor influences the likelihood of the binary event occurring without being scaled. You will get a table like the one pictured.

Variables in the Equation

		B	S.E.	Wald	df	Sig.	Exp(B)	95% C.I.for EXP(B) Lower	Upper
Step 1[a]	IV100Pt	4.96779	640.271	.000	1	.994	143.708	.000	.
	IV5Pt	-7.79237	2381.102	.000	1	.997	.000	.000	.
	IV2Pt(1)	-32.40123	8896.086	.000	1	.997	.000	.000	.
	IV1Pt(1)	-24.24786	7946.572	.000	1	.998	.000	.000	.
	Constant	-184.53242	25496.883	.000	1	.994	.000		

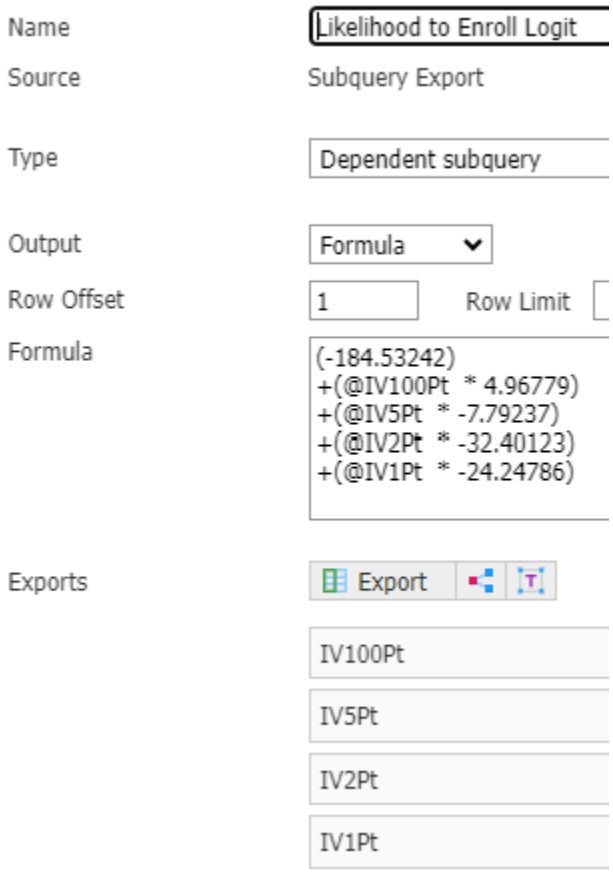
a. Variable(s) entered on step 1: IV100Pt, IV5Pt, IV2Pt, IV1Pt.

With standalone Slate fields (meticulously named as described in the Scoring Chapter) created for each of your behavior, your first query calculates the number of each behavior or the appropriate datapoints. Examples of this can be seen in the Feature Engineering Chapter.

The second query calculates the logit: the natural logarithm of the odds of an event occurring, log(p / (1 - p)), for probabilities strictly between 0 and 1. Binary logistic regression models this log-odds value as a linear combination of predictors. The next step converts it to a probability for staff to use.



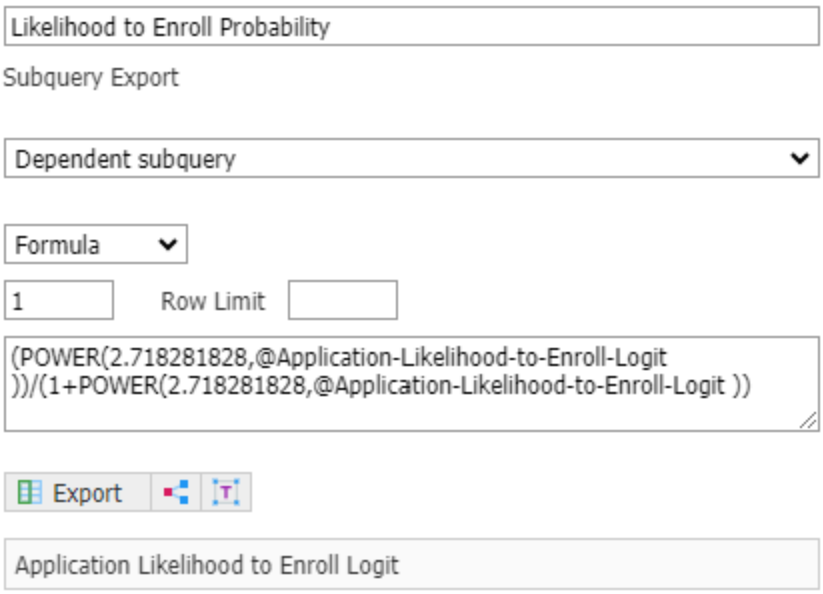
This is where you need your SPSS values. In a Slate Formula Export, add your constant value, essentially the default value of the student before any behaviors are added. Then you will sum the parenthetical value of the students' datapoints multiplied by the unstandardized regression weight - the capital B in the SPSS table. Do this for each of the behaviors in your model.



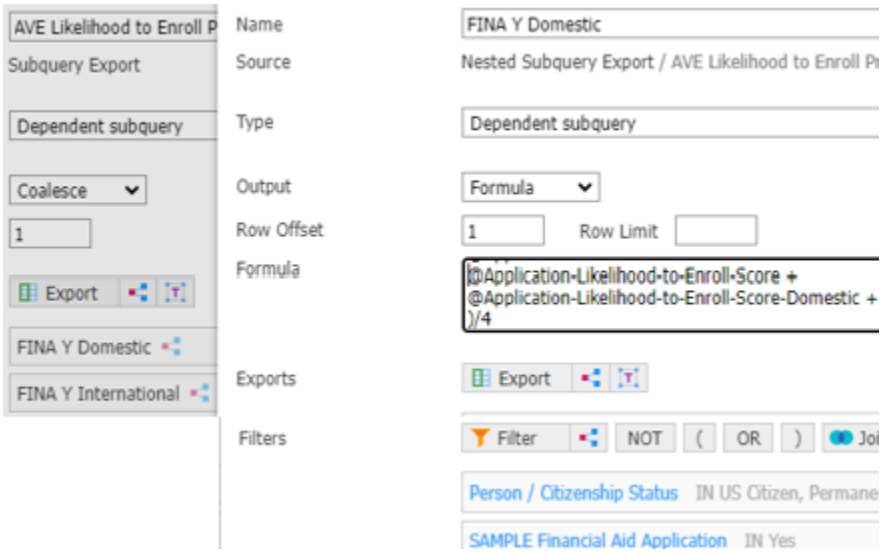
If you are running multiple models - one set of variables and weights for domestic students, another set for students on financial aid, etc, create Exports in your Query for each of these models to run them all at the same time.

The third Query in the sequence creates your 0-100% probability. You have to take Euler's Number (2.718281828) and exponentiate it to your Logit. Divide that by 1 + the same thing.

Again, do this in the same Query for each of your enrollment models to cut down on the number of Queries running.



If there are multiple models, then one additional Query is needed to aggregate the models to create a central probability of each student to enroll. Coalesce these model aggregates



Each aggregate averages the relevant models. Adding Filters allows the single Coalesce Export to write the appropriate probability for each student regardless of which populations they fall into.

In that short sequence of Queries, you went from raw behaviors to an estimated 0-100% likelihood written to each student's Slate fields. The same math can run in a Report. Changing inputs can support scenario exploration, but a predictive model alone does not estimate the causal effect of a nudge or intervention.

Automating Model Updates

While presenting this process at Slate Summit, AACRAO SEM, and the Empowering Admission Workshop series, I found that automation was often overshadowed. Slate can refresh predictions as recorded behaviors change. An illustrative model might show a four-percentage-point increase after a campus visit or 1.2 points after another email. These are hypothetical, model-specific changes in estimates, not guaranteed outcomes or causal effects of those actions. Automation can keep running, but the model still needs monitoring, validation, and periodic updating.

Automate the refresh; keep reviewing the model.



Before I start sounding like a snake oil salesman, Models do need to be updated. Binary Logistic Regression requires accurate records and behaviors, and the changing landscape of higher education and institution practice and policy can make older records less valuable in Modeling future behaviors. These older records engage differently with the institution (and may not have engaged at all in new practices coded into the model). There may be other factors that are not in Slate that will reduce the value in the model.

Not only will these older records affect the accuracy of your Predictive Modeling, but also the consistency of your student populations. When building this predictive model at a highly selective, small, liberal arts college, the school's popularity was actually a hindrance to the Modeling. Students showing very low demonstrated interest as captured in Slate enrolled are remarkably high levels. Even stealth applicants whose only interaction with the institution was applying and submitting required materials. Even moderators struggled to ferret out an effective way to identify this group of low-engagement/high-enrollment students.

While those are two things more out of the Model builder's control. Selecting the "right" behaviors and other datapoints being added are critically important. As is the proper coding of the behaviors. The model will functionally work regardless of the quality or number of variables. If there is only one variable - a domestic/international flag, and domestic students historically enroll 45% of the time and international students historically enroll 40% of the time, then Slate will look at if they are domestic or not and apply those %s to them. Could be valuable, but there's so much more context you could add (and you also don't need Binary Logistic Modeling for this). On the other hand, adding 30 variables, some being significant predictors and some not, may not be optimal at predicting future student behavior.

There is significant nuance to the process. If your Slate team does not have the required knowledge, an IR Office or faculty may be able to assist with this side of the process.



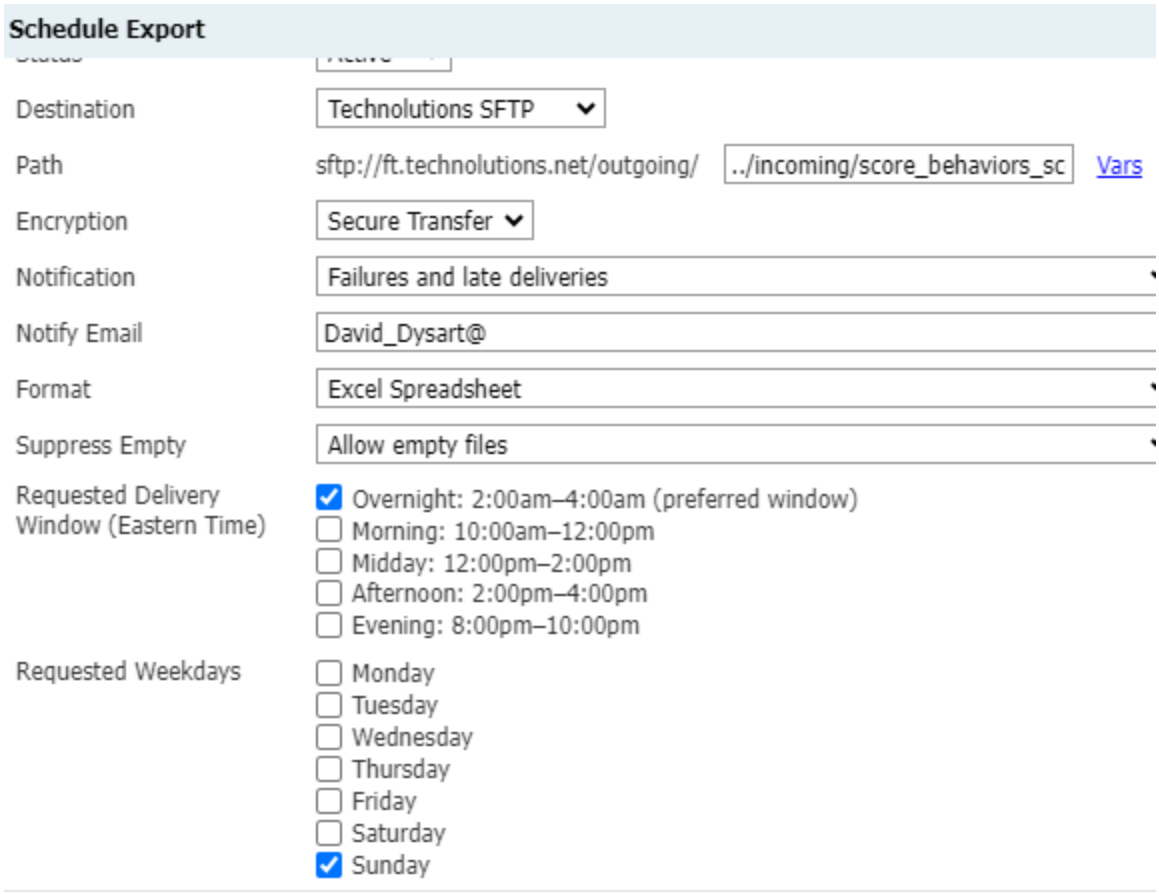
Note: Data is a great element to add to your business process, but avoiding technical debt in the process by the way of automation can drive better, next generation outcomes.

To ensure that predictions remain updated to recent behaviors, you can schedule each of the above Queries to run to your SFTP server. Use Slate's Schedule Export feature in Queries to automate execution at regular intervals. This ensures that probabilities are updated, reflecting the latest student interactions. Some recommendations:

Note that the path needs “../incoming/” to place the query results in the “Incoming” folder which is where the Source Format will need them to be to pick the file up and import it.

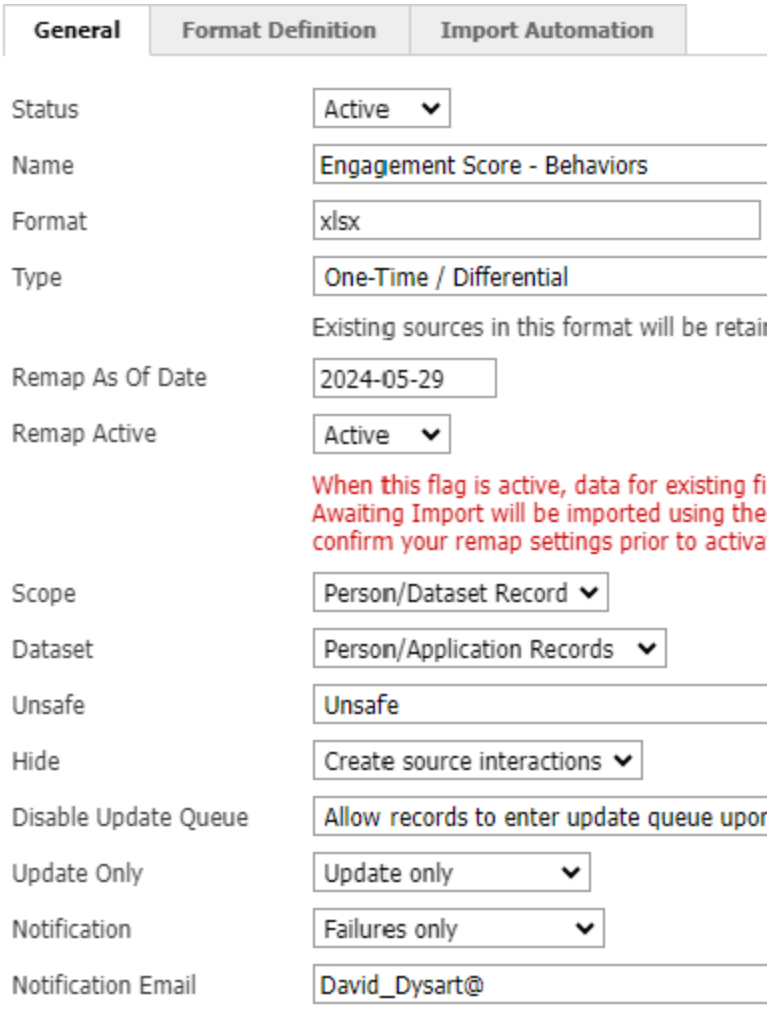
Set up an email to receive notification for at least Failures and late deliveries.

Schedule the more intensive queries for slower days/times if possible. If there are multiple queries that need to run, staggering the windows will help error-proof your sequence rather than trying to ingest them in the appropriate order in the same window.



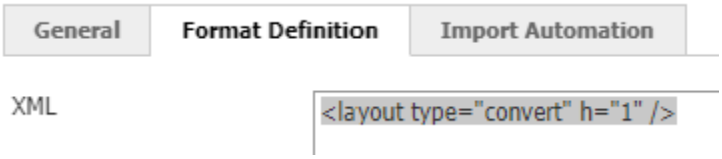
You can then automate the import of data with a Source Format. There are a few notes:

On the General Tab - Set Status to Active, give it a descriptive Name, Remap As Of to capture the earliest file and set Remap Active to Active, Set to Unsafe so current Applicants can have fields updated (even if you think you set your new fields to Unsafe), set Update Only, and set Notifications to at least email relevant staff on Failures. Adjust other settings as needed.



Under the Format Definition Tab, this code in the XML box tells Slate that the first row in your file is the Header with Column Names

<layout type="convert" h="1" />



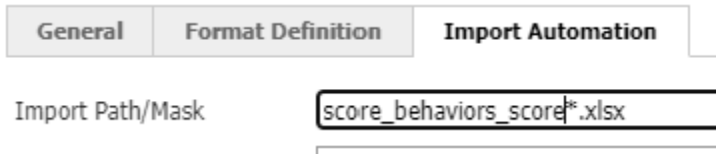
The last important piece to update is in the Import Automation Tab. You must add the name specified in the Query's Scheduled Export. This tells Slate what file to pick up and ingest. This example uses an asterisk wildcard to look for the file name and any additional text where the "*" is at. This is important if you are adding a timestamp.

If you have multiple Scheduled Exports that may be picked up, you will need to revise your naming convention. “model_enroll*.xlsx” would pick up all three of these files:

model_enroll_behaviors.xlsx

model_enroll_logit.xlsx

model_enroll_probability.xlsx



By automating model calculations, admissions teams can focus on interpreting the data and making informed decisions, rather than manually updating Scores to upkeep external systems and integrations.

Conclusion

The process of creating a binary logistic regression Model in SPSS as well as transforming it in Slate is probably the lowest hanging branch to get into Predictive Modeling for higher education. Because of its interpretability and lower computation cost, it’s also extremely flexible and adaptable for a whole suite of initiatives and strategies we will discuss in Act III.

While advanced algorithms offer significant potential, starting with accessible tools ensures immediate impact and scalability. As the landscape evolves, institutions equipped with these foundational models will be well-positioned to adopt more sophisticated approaches when ready and needed.

Chapter 10: Time-Specific Anchors and Models

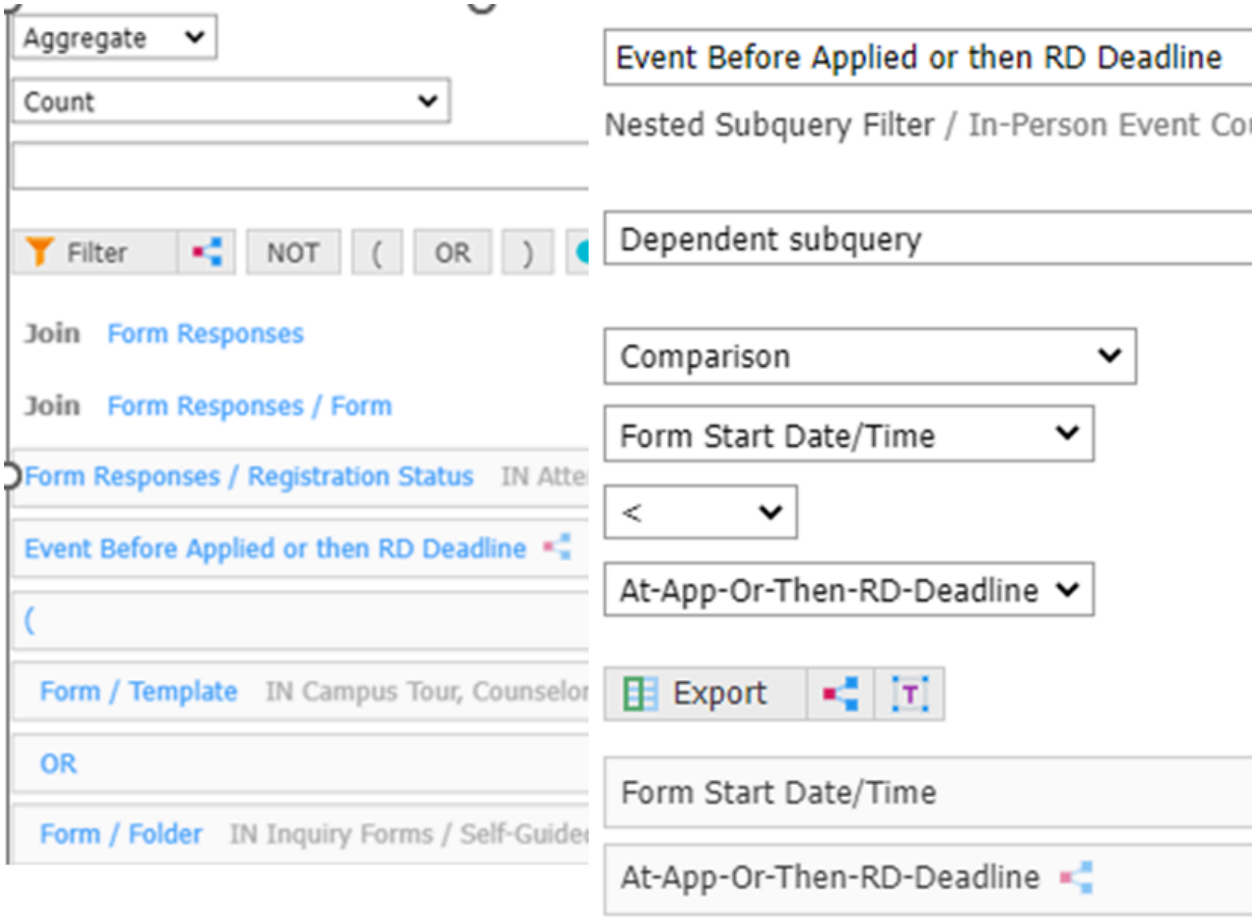
~1,100 words

In the context of higher education enrollment, time-anchored data and timestamping behaviors is essential for creating accurate and reliable predictions. By focusing on the precise timing of student interactions relative to outcomes like application submission or enrollment, institutions can avoid weaker predictors and underperforming models. We'll explore strategies for handling timestamped behaviors and how both multiple models and multiple versions of a model can improve outcomes for your students and institution.

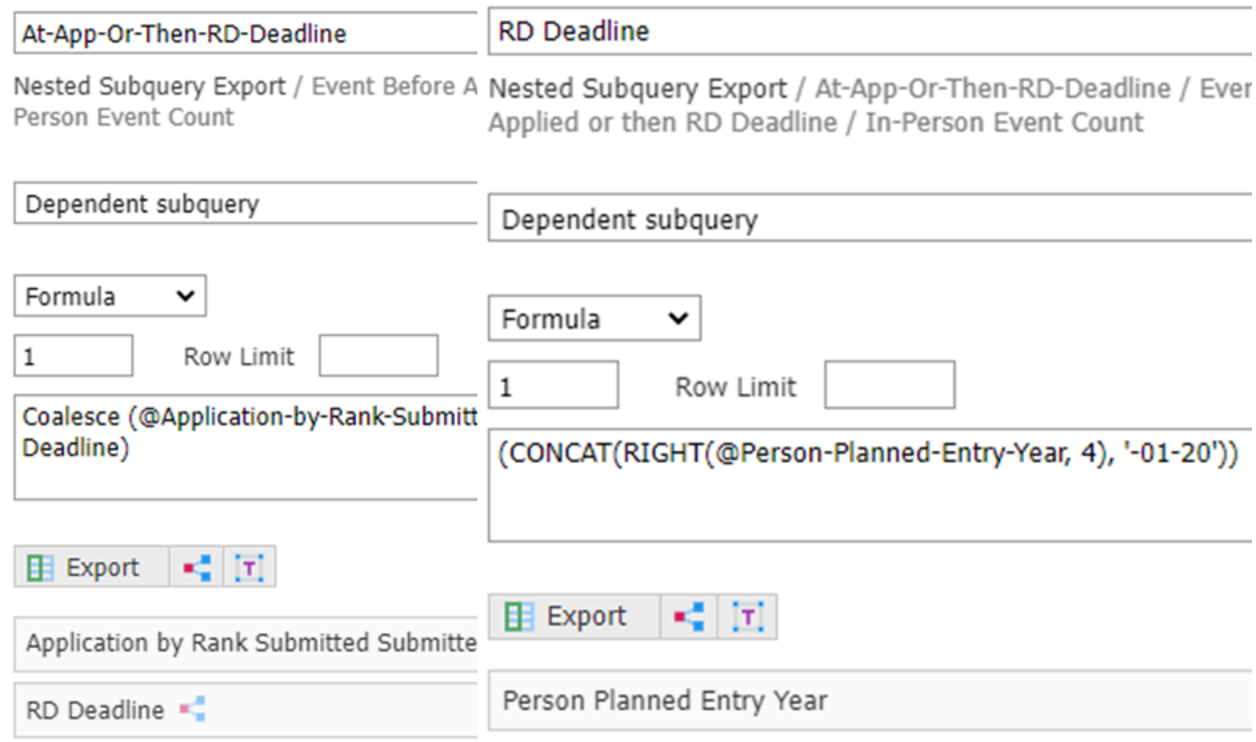
Depending on the use of your model, you may need to timestamp your data and behaviors to accurately train your model. If you are Modeling on how likely a prospective student is to apply, you don't want to train a model based on behaviors like sending emails AFTER they have already applied. Similarly, if you are trying to predict the enrollment likelihood of admitted students at time of admission, then you don't want to include that Admitted Student Day (or other events) in your training data.

Unsurprisingly, matriculating students tend to eventually engage in more behaviors than just applicants. Applicants will also inherently have higher behavior counts than prospective students as they have more reason, time, opportunities, and programming to do so.

But the flexibility of Configurable Joins Queries can accommodate this. Compare timestamps of the behaviors relative to the outcome in question. For applicant Modeling, pull variables at time of application (or application deadline for non-applicants). Depending on your use, the app deadline for both groups may be more appropriate. This screenshot example shows the Subquery Export's Filter.



Coalescing the date filter (if Application date is blank, move to the next criteria such as the application deadline) pulls these apps and non-apps in the same query run with the appropriate behavior counts, so it is not much more complicated* to use both application timestamp and application deadline than applying the application deadline for all records.





Note: The Joins Juggler understands that when David says "not much more complicated" that it is relative. Running additional nested exports that include Coalesces and written formulas can be substantially different than other Configurable Joins being used. David can just be as short-sighted as his glasses prescription.

Similarly, for enrollment Modeling that I typically reference in this book, Slate uses behaviors as of decision release by the institution. While matriculation predictions could be more accurate if they used behaviors at time of committing, deadline to commit, or even census, this use case is particularly interested in using the probability to enroll at time of decision committee to inform census headcount and net tuition revenue projections. That way, it was tailored to admitting the correct number of applicants to achieve the needed incoming class.

The above pictures are a good guide to start building the query. It is a bit blunt and may miss records with wandering entry terms (changing their planned entry term, multiple applications, etc.), so if that is a substantial population for the institution, a more refined process of pulling historical data per cycle to build the model may be needed.

You will need to be thoughtful of what the main use cases are for these models and what decisions will… yield the most impactful data. Multiple versions of the models can be created with an incremental additional lift (less than the 9-model solution above). The variables for an enrollment model at time of admission committee and an enrollment model at time of census could have heavy overlap.



In Elaboratione (Latin for "In Elaboration" where I will expand and offer some additional guidance instead of finding a better place for the topic to live in the book.)

On the topic of Multiple Models and Multiple Versions

It may be helpful to distinguish multiple models and multiple versions. I advocate for multiple models regardless. This could be averaging across several models relevant to a student. If they are an International STEM Student on Financial Aid who submitted a Stealth Application (applied without engaging with the institution), each of these factors may have a separate Model with their own predictors and weights -

- STEM Model: 80% likely to enroll.
- International Model: 70% likely to enroll.
- Financial Aid Model: 65% likely to enroll.
- Stealth Applicant Model: 85% likely to enroll.
- Aggregated Predicted: 75% likely to enroll.

This is an ensemble formed by averaging estimates from subgroup models, not a random forest, which combines decision trees. Averaging does not automatically improve accuracy or calibration. Compare the combined result with simpler alternatives on held-out data and review performance across groups. Extra segments need enough data; a weak component model can reduce the quality of the combined prediction.

You may find that the additional predictive power is not worth the time to build and maintain multiple Models. You may also be able to account for some differences with standalone predictors and interaction/moderator effects. Ultimately, when it's your flying monkeys, it's also your circus (or in this case, twisted, non-Euclidean castle set high atop the cliffs).



It may also serve your institution to have multiple VERSIONS of your models. Your model predicting enrollment at time of decision release may be an important element of your practices, but a month after decisions, that model does not distinguish how a student's behaviors since decisions will influence the likelihood of enrolling.

Creating an additional Model(s) that includes additional/different factors may help guide yield efforts and Waitlist offers, so depending on resources, needs, and goals, it may be important for your institution to create these additional Models. It could be prudent to pilot the first Model(s) and then each cycle, add Models as needed to address the gaps at your institution. This could also be a good opportunity to experiment with different types of Models/Algorithms to find the best versions for your institution or specific use cases.

This second version of enrollment Modeling would require a second query with the proper timestamp comparison and different model values that get plugged into Slate queries (remember when I said naming your fields would be VERY important). Feature-engineering additional behaviors around institutional actions like decision/transfer evaluation, aid awards would likely pose the largest tasks as nothing particularly similar would have been created for a At Decision Model.

Terminus: In Elaboratione (Latin for "End: In Elaboration")

Conclusion

Effective Predictive Modeling in higher education requires a precise understanding of time-anchored data and student behaviors, allowing institutions to tailor models accurately to various outcomes. By creating models and versions that adapt to unique timestamps and student actions, enrollment managers can make data-informed decisions, enhancing outreach, admissions strategies, and yield management. Thoughtful model design and multiple iterations enable institutions to support students more effectively while meeting enrollment goals with greater precision.

Chapter 11: Feature Engineering

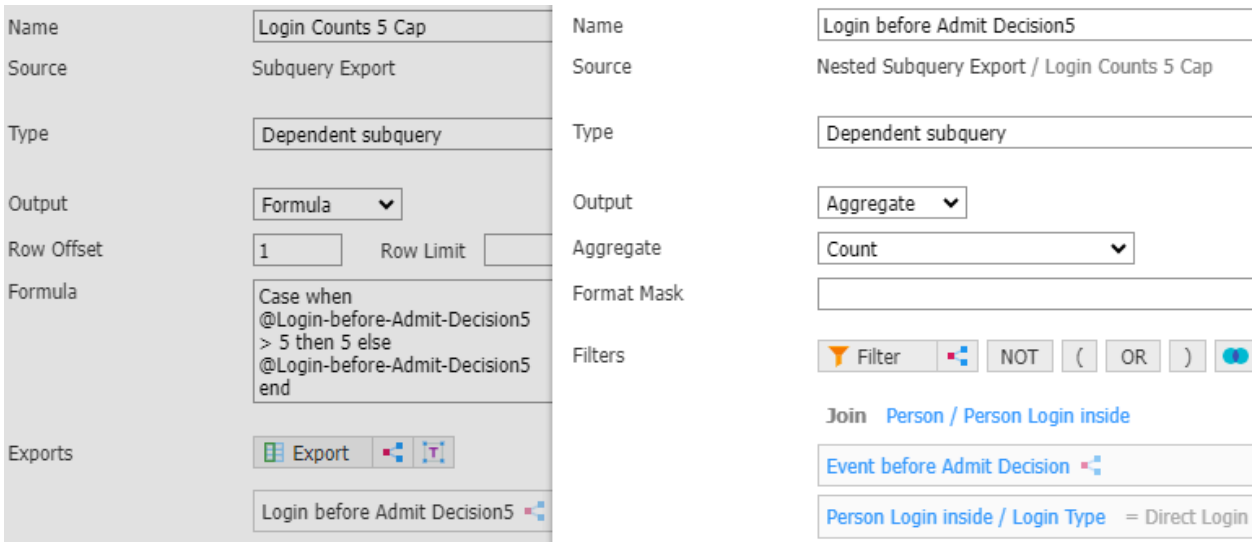
~2,000 words

The data available to you for Predictive Modeling grows exponentially the longer you are on Slate and the more integrated it is with your student experience. Feature engineering plays a critical role in enhancing the performance of Predictive Models though. Utilizing Configurable Joins and building robust models that reflect the realities of student engagement over time and around important moments that matter, your model and actions can become exponentially more impactful.

I will use the term “feature engineering” relatively broadly in the book. In general, it involves transforming raw data into meaningful predictors that improve model performance. This process includes creating, modifying, or selecting variables that capture important patterns, relationships, or trends in the data, ultimately helping the model better predict outcomes by enhancing its ability to recognize key insights.

The Predictive Modeling I’m covering in this book is designed to be replicated and automated in Slate, so I will be limiting our examples to what can be done in Configurable Joins and in Export Formulas (essentially SQL with only a smaller selection of functions). While there are much more extensive transformations that can be done on your data with the final value being imported into Slate, that’s another topic for another book. Let’s explore a few different ways to manipulate your Slate data. Some of these variables may lend themselves more to Engagement Scoring or other flags to drive institutional action.

Capping Behaviors (Logins)



In this Export, you are Aggregating the Count (summing the number of times a behavior

occurred). In this case, the number of times a student logged into the portal, though this could be anything from attending events, sending emails, time on the website with Ping, etc.

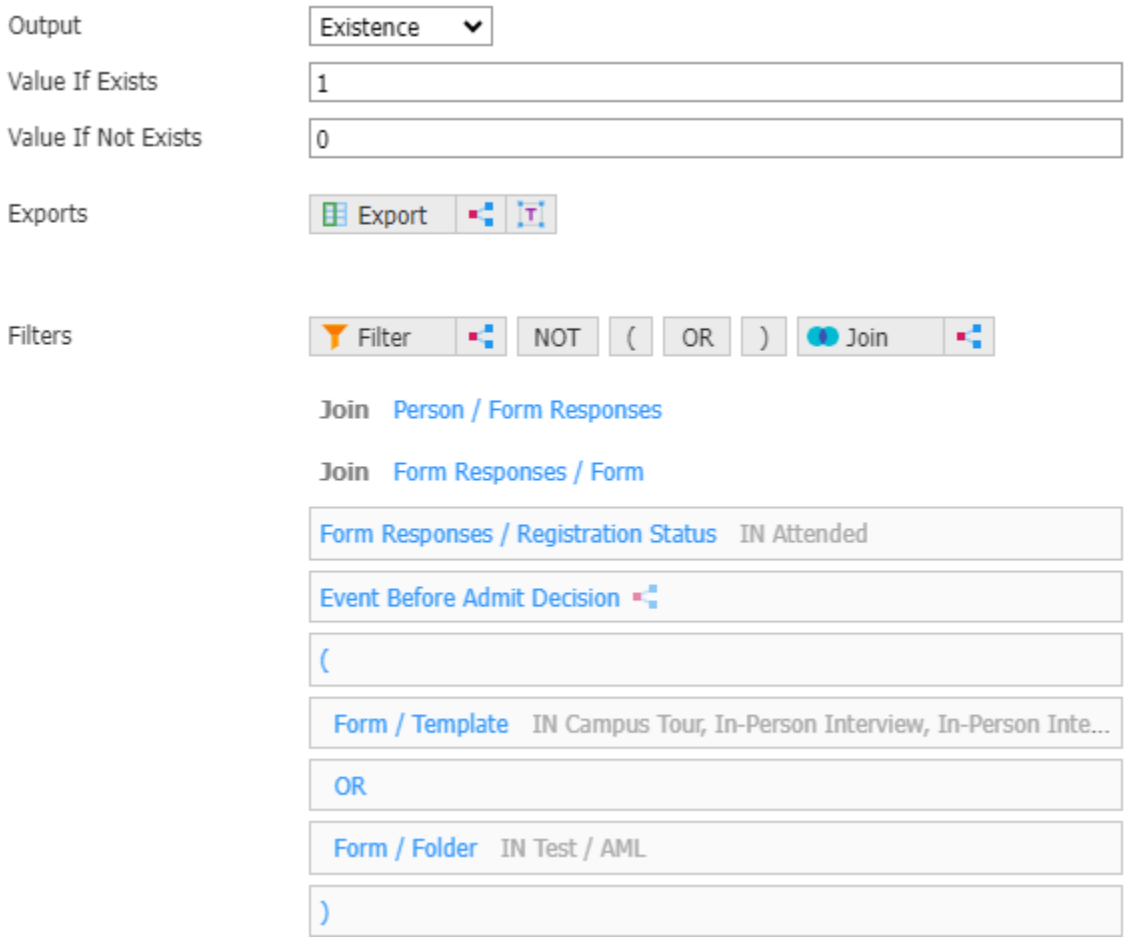
Since you are counting the number of portal logins, you also need to restrict it to Direct Logins to exclude any staff impersonations. And again, we are looking specifically at the number of times before the Admit decision was released.

After exploring the data, you may choose to cap the login count at 5. In binary logistic regression, a numeric predictor has a constant coefficient on the log-odds scale, holding other predictors fixed. Its effect on predicted probability is not constant: it depends on the starting probability and other inputs. An uncapped count may impose a poor functional form if very high counts do not carry proportionate information. Test a cap or other transformation against alternatives using held-out data rather than assuming it improves the model. If you only need a raw event count, remove the Formula layer and use the Aggregate.



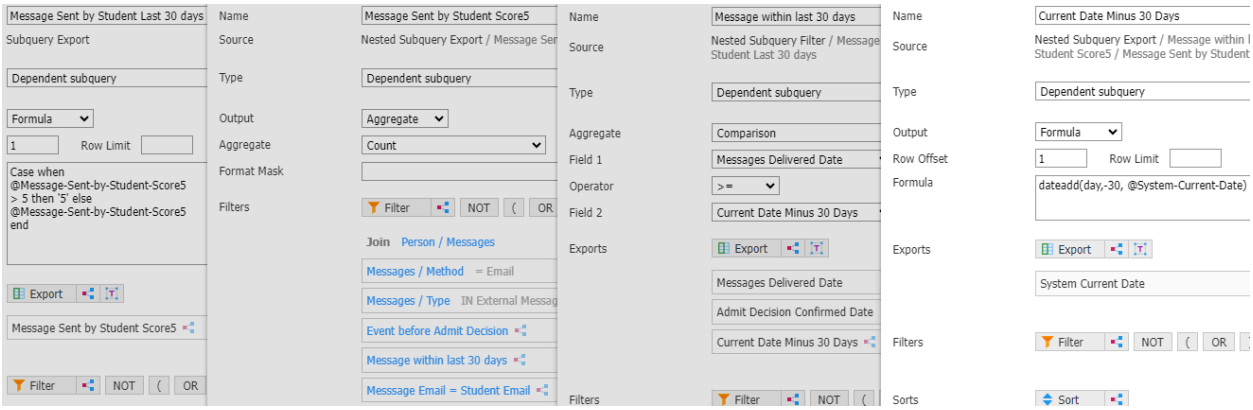
Note from our resident Field of Statistics Mouse: While in general, it's not usually advisable to limit, dummy code, restrict, etc. data in statistics, a bit of engineering will likely increase the impact of your model. This must be done with great care to prevent overfitting, cherry picking, and dilution of quality. I like to do extensive data exploration to get a feel for variables and their relationship with the outcome to inform transformations.

Flags and Dummy Coding (Event Attendance)



Perhaps the number of events (even restricted) is not as good of an indicator as the existence of an event (or other behavior) at all. The previous formula could be used, but this would be more computationally efficient. It exports a 0 if the student has not attended an event and a 1 if they have. This can be especially helpful when there is a very non-linear relationship between the raw count and the logit of the outcome.

Recent Behaviors (Emails Sent by Student)



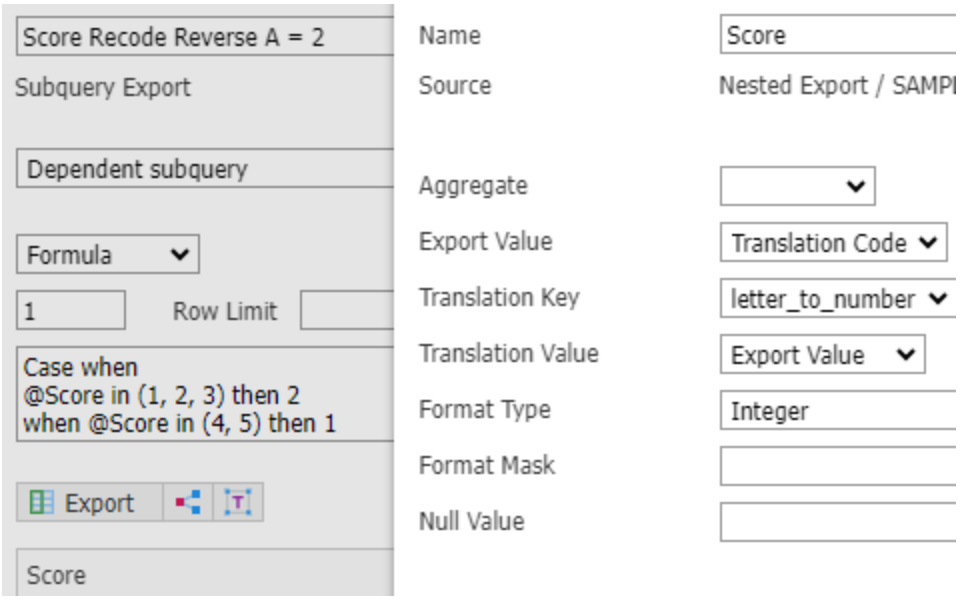
This example counts the number of times a behavior like emails sent by the student occurred within the last 30 days. We'll calculate time-relative behaviors differently in another example. The second pop-up has several relevant filters for querying emails sent.

The Method allows you to specify "Email" as opposed to other data on that table like SMS, Letters, Mail Merges, Voice, etc.

The Type also specifies we want External Messages coming into Slate as opposed to Deliver Mailings, Form Communications, and Ad Hoc Messages all sent from Slate.

Lastly, you will want to specify the email address sending the email is the student's email as emails merely attached to the record from parents, counselors, vendors providing material, etc. will also be included without this filter which are not going to be as predictive as actual messages from the student.

Recoding Values (Score Transformations)



This example uses multiple methods and reasons to transform your data. A formula gives you the ability to reverse code your Score. In this case, a 1 in the Score variable is a “better” Score. Perhaps to make explanation easier, you are using the formula to change the “better” Score to be the “higher” numeric value, so as the student Scores better, their value in this datapoint is higher. It’s a more intuitive way to describe the model.

The formula is also recoding the Score to group the values. If the student Scored a “1”, “2”, or “3”, then their new value is a “2”. This is likely because those three values have a similar relationship to the outcome while a “4” and “5” share a similar relationship. You would also want to be mindful of size discrepancies between the groups.

Lastly, you are using a Translation Code to transform the raw values of letters into numeric values. In theory, all three of these transformations could be done with a single Translation Code, but that may become too blunt and hyperspecific of a use case for an office-available Translation Code.



The current Translation Code is used to convert the letters to numbers which may have several business practices associated with it. If you use a 1 to indicate the best Score as a general practice, you probably don't want to clutter your Translation Code with an Export Value that contradicts how staff will interpret Scores in case they use this Translation Code instead of another that keeps Scoring consistent.

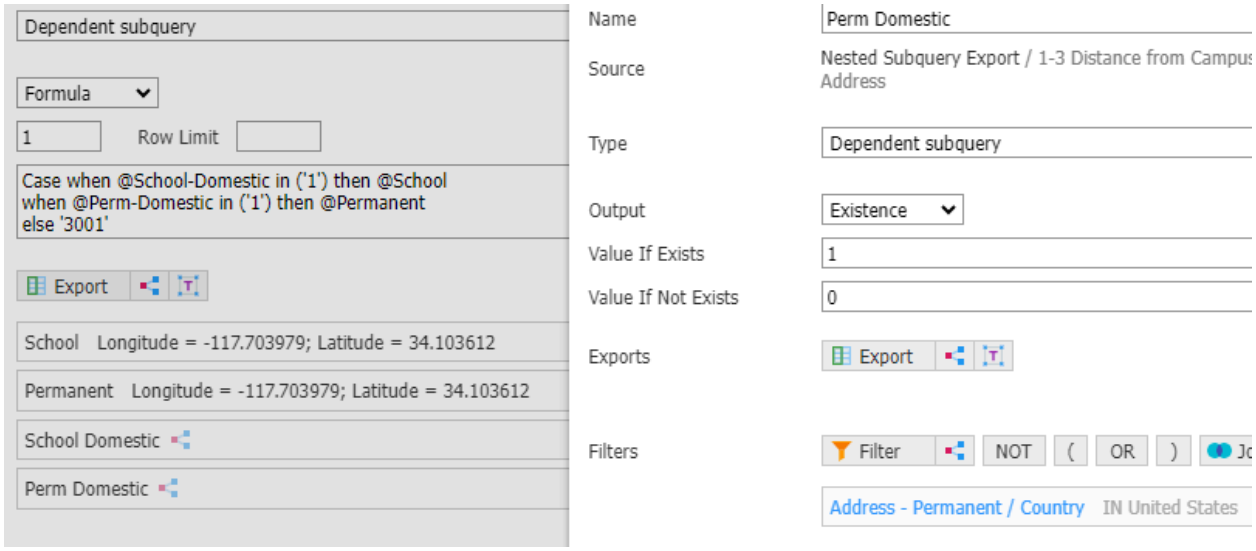
Similarly, if only one model requires this particular translation of "1", "2", and "3" = "2", then you may require several similar Translation Codes and may accidentally update a Report or Query's calculations when updating this Translation Code. So I would recommend controlling your Translations more granularly and clearly.

Translation Codes

Key	letter_to_number
Type	String (Discrete Values)
Default	No default exports have been configured for this key.

Value	Export Value
New Code	
A	1
B	2
C	3
D	4
F	5

Distance Recoding

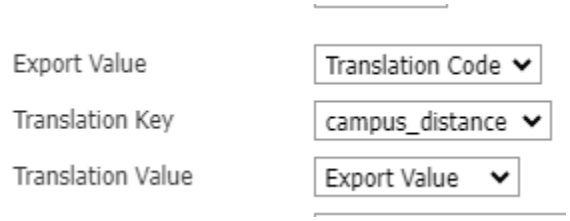


A student's proximity to campus can impact their probability to apply and enroll in several avenues. Most students attend college in their state. There is some inferential data from vendors that can indicate in/out of state college-going desire and trends. International/domestic is another way to examine probability.

In this example, you are using a formula to examine a couple data elements based on the students distance from the institution. If the school they attend is in the country, then it returns the number of miles the student's school is from the institution. If the school is international or you don't have the address information, then it examines the student's

permanent address. If neither are domestic, then the formula just returns a static number of miles.

A Translation Code then categorizes the raw number of miles into three buckets that can be entered into the model as a predictor, interactions effect, or augment a feature. More customization can go into a variable like this depending on how distance-related variables are related to your outcome.



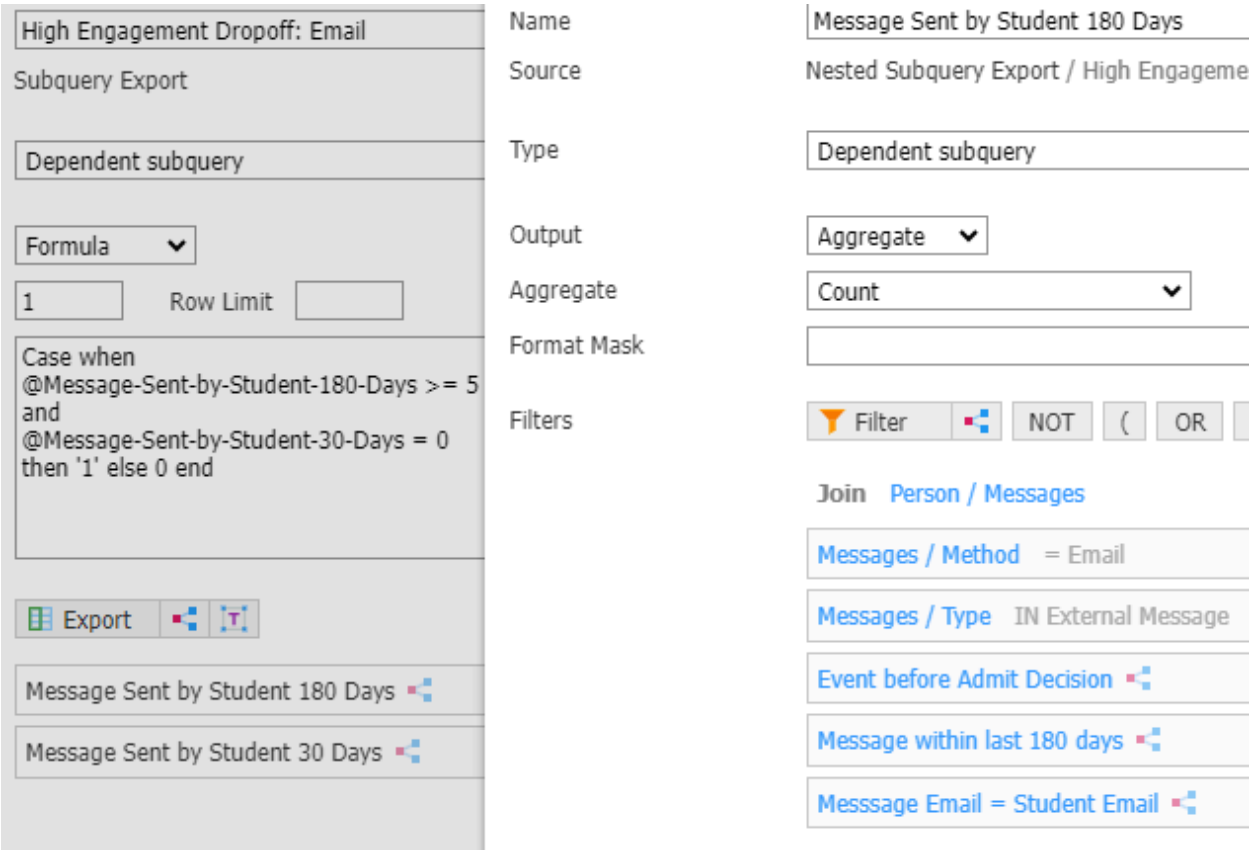
Translation Codes

Key	campus_distance
Type	Real Number (Inclusive Range of Values)
Default	No default exports have been configured for this key.

Sea

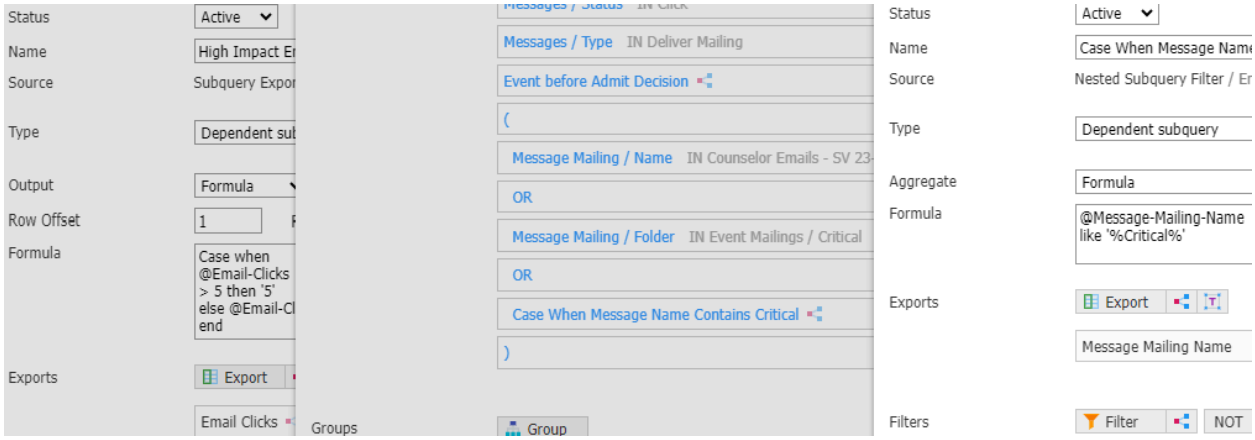
Min Value	Max Value	Export Value
New Code		
0	1000	1
1000.001	3000	2
3000.001	100000	3

Behavioral Changes



A drastic change in engagement behaviors should set off alarm bells in your office. In this example, the student was highly engaged, sending at least five emails in the last 180 days. But they sent zero emails in the last 30 days. These criteria could also use other anchors like Engagement Score greater than thirty but zero engagements (emails, Ping, events) after Decision Received, Campus tour, Financial Aid Meeting, etc. This could be strongly indicative of a change in enrollment probability. Re-engaging the student could be needed to keep them in the funnel.

Categorizing Behaviors



Institutions send a variety of emails and call-to-actions. A student ignoring an email about how good your weather is or not clicking on an event registration link in it probably isn't the end of the world or their journey with your institution.

But what about those emails and SMSs about missing application materials or onboarding actions? There's a broad spectrum of importance and topics, so being more granular in examining engagements could provide more information on the student's intentions to enroll at your institution.

Unfortunately, there's not currently a way to add meta data or associate a form to a mailing. And mirroring mailings with a dataset would be difficult and time-consuming to do. So some other options include

Filter: Maintain a list of these categorized emails to manually add as a filter

Folder: Create subfolders where these emails live to add as a filter

Name: Include a keyword like "Critical" in name of the emails that can have a formula filter written against them

Each of these require ongoing diligence to keep your mailings categorized and contributing to your model properly. A regular review and checklist for staff will help keep these requirements synced though.



Note from our Messenger Pigeon: Tailoring Calls to Actions to the student may increase the likelihood of the student engaging with your communications. A highly engaged local student is more likely to attend a Campus Tour than a full-need international student who may be more likely to attend a virtual event or Portal experience. Additional models to predict engagement topics can further refine your MarComms strategy.

Moments that Matter Behaviors: Application Anchors

Days from Event to App

Subquery Export

Dependent subquery

Formula

1 Row Limit

```
datediff(day,
@Form-Start-Date-Time,
@Application-Submitted-Date)
```

Export

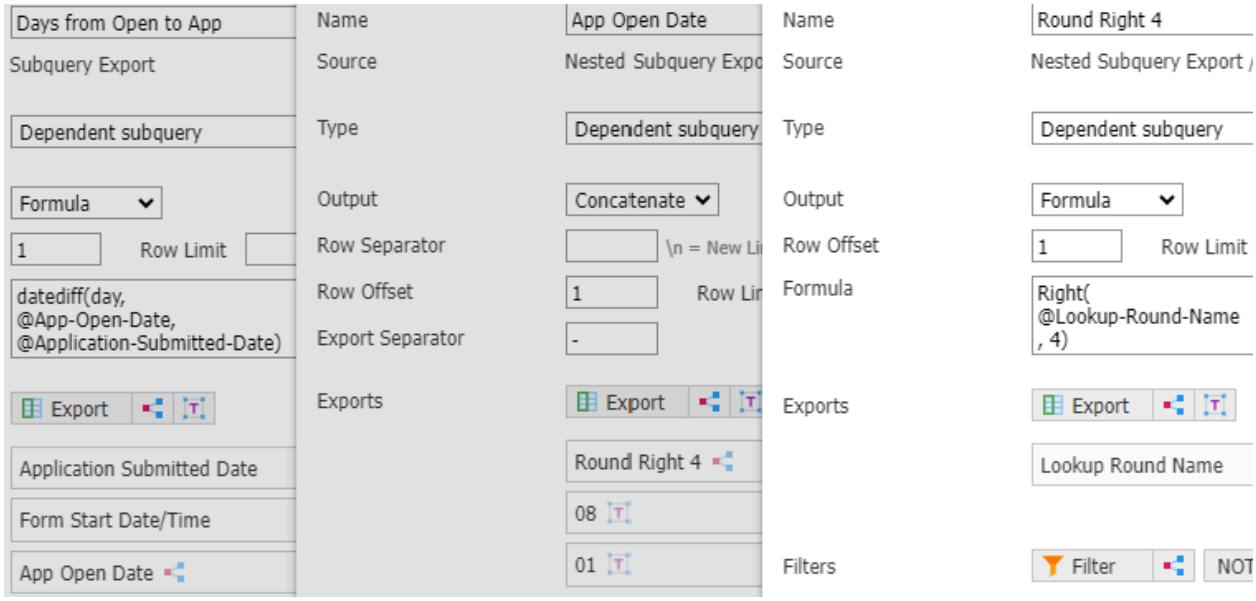
Application Submitted Date

Form Start Date/Time

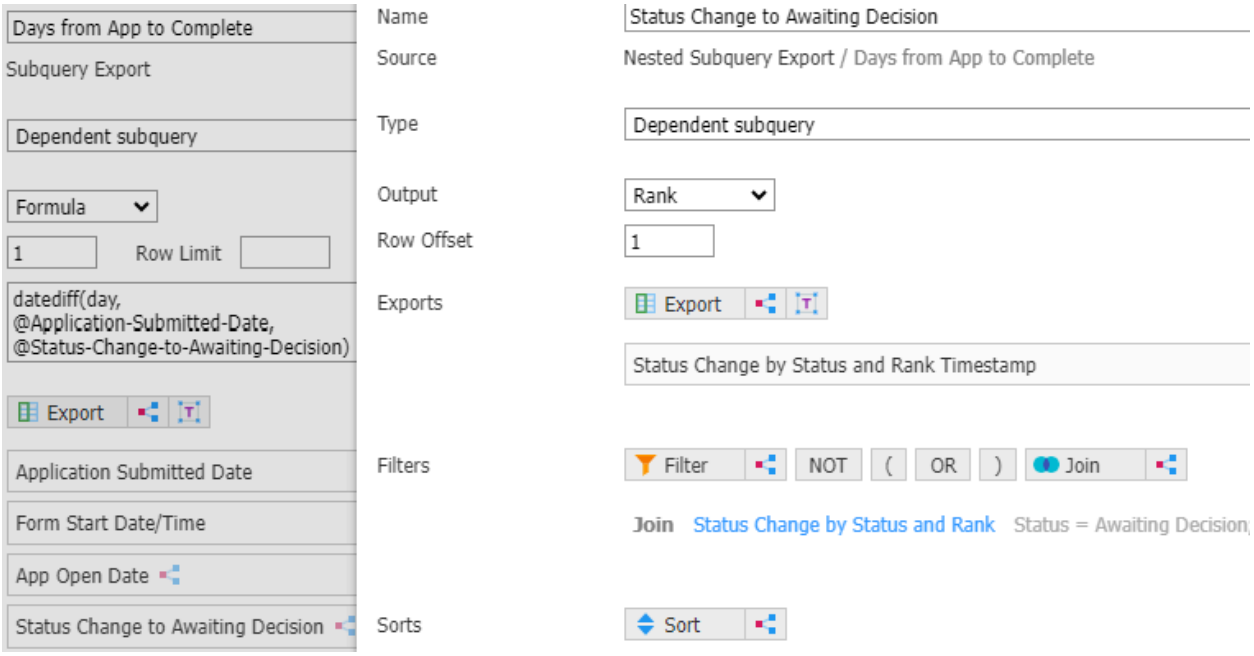
How students behave and engage around critical moments and milestones can have a substantial impact on their enrollment likelihood.

This example counts the number of days between an event and when the student submitted the application. This might pair better with assigning a value based on a close timeframe like within a week of the event.

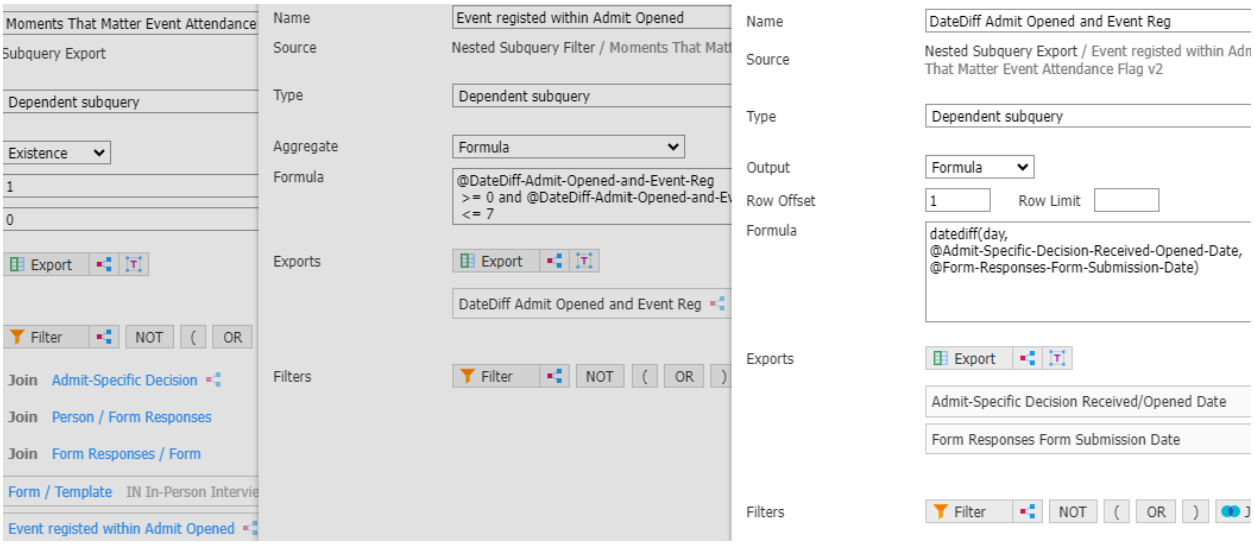
Another indicator could be how long from when the application opened to when it was submitted. This starts to get into shakier territory as students applying for Early Decision/Action Rounds, or with better support/counseling, or certain demographic trends can introduce more noise than insights.



Another variable to explore would be how long from application submitted to completion. Again, there is likely to be considerable noise from populations that have additional requirements or constraints.



Moments that Matter Behaviors: Decision Anchors



A critical glimpse into your student behaviors are how and if students engage around your outreach like admission decisions, financial aid awards, and transfer evaluations. Are they registering for events or reaching out to staff? Are they going radio silent? Are they filing appeals?

In this example, you calculate the time from the admission offer being opened to registering for an event. If that happens sometime between the day of opening and a week from opening, then that student gets a flag of ‘1’. Others get a ‘0’. While I harp on

pre-decision behaviors a lot in this book, adding decision-anchored behaviors should be explored if your model incorporates that timeline.

Configurable Joins is such a robust, powerful, and flexible platform for feature engineering. The above examples are just a brief walk along some of the things that are possible when building these more complex and relational variables.

Conclusion

Effective feature engineering using time-anchored data enables institutions to build more accurate predictive models. By thoughtfully organizing and timestamping behaviors, institutions can capture the nuances of student engagement, leading to more informed decisions and better resource allocation. Through strategies like coalescing data, leveraging configurable joins, and creating multiple Models, we underscore the importance of tailoring Predictive Models to specific institutional goals, ultimately enhancing the student recruitment and enrollment process.

With your Engagement Scores and Predictive Models built, the stage has been set for the third act of our time together. Enter Adaptive Enrollment Management -

