VISTAAR user guide

VISTAAR is a local Python library for a binary customer outcome. You give it a table and a target. It profiles the columns, finds readable micro-segments, ranks customers with out-of-fold scores, and writes a report you can hand to someone who will not open a notebook.

Version 0.3 is the campaign autopilot. On a large book it draws a stratified sample, checks that the sample still looks like the book, and scores the full book afterwards. It compares a short list of models on capture in the top 20%, not only on AUC. It ranks segments into a short non-overlapping portfolio for the score cut. Every automatic choice is written down with the reason.

Version 0.2 adds five things on top of that ranking:

1. Who sits inside a score decile, in plain segments.
2. Uplift models for a randomised contact test.
3. A profit cut from cost and value.
4. Calibration, and a monitor for next month's file.
5. A champion/challenger registry with a paired bootstrap.

Nothing in the library is a causal effect unless you are in the uplift section **and** the treatment was randomised (or you accept the unconfoundedness assumption, which the table cannot prove). Lift is a rate ratio. A scenario is arithmetic.

The worked notebook is `examples/VISTAAR_end_to_end.ipynb`. It runs top to bottom on Telco churn and the Hillstrom e-mail experiment.

Install

Python 3.10 or newer.

```
pip install vistaar
pip install "vistaar[all]"    # CatBoost, XGBoost, LightGBM, SHAP, HDBSCAN, PySpark
```

From a wheel you built or were sent:

```
python -m build
pip install "dist/vistaar-0.2.0-py3-none-any.whl[all]"
```

Colab, after you upload the wheel:

```
%pip install "vistaar-0.2.0-py3-none-any.whl[all]"
```

Databricks, as a cluster library, or from a notebook. Restart Python after `%pip` before the import.

```
%pip install "/Volumes/<catalog>/<schema>/<volume>/vistaar-0.2.0-py3-none-any.whl"
%pip install "/dbfs/FileStore/jars/vistaar-0.2.0-py3-none-any.whl"
```

```
import vistaar as vt
print(vt.__version__)   # 0.2.0
```

Core dependencies are pandas, NumPy, SciPy, scikit-learn, statsmodels, Plotly, openpyxl, and tqdm. Uplift, profit, calibration, monitoring, and the registry do not need an extra package. CatBoost needs `vistaar[ml]`.

Quickstart

```
df = vt.datasets.load_conversion()
engine = vt.Analytics(df, target="converted", id_column="customer_id", theme="hdfc")
results = engine.run()
results.summary()
results.deciles()
results.campaign(strategy="mass", top_deciles=2)
results.campaign(strategy="precision", target_precision=0.75, min_customers=100)
results.export("analytics_report.html")
```

`run()` prints a stage line and a timing summary. `progress=False` silences it. In a notebook or a TTY the stages use tqdm. Otherwise they are one plain line each, which is what you want on Databricks.

`theme="hdfc"` is the default: navy `#004C8F`, red `#ED232A`, white, and the Wikimedia public-domain HDFC Bank wordmark embedded in the HTML. The name and logo are trademarks of HDFC Bank. The theme is report styling, not a bank endorsement. `theme="neutral"` drops the logo.

Your own table

Required: a binary target. Yes/No, True/False, and 1/0 are inferred. A target whose name contains "churn" is described as churn, and a higher rate is the worse direction.

```
engine = vt.Analytics(
    df,
    target="Churn",
    id_column="customerID",          # kept out of the model
    value_column="MonthlyCharges",   # known before the outcome
    treatment_column=None,           # set this only for uplift
```

```
    positive_label=None,            # set it if the positive class is not obvious
    model_names=("logreg", "catboost"),
    random_state=42,
    cv_folds=5,
    theme="hdfc",
    progress=True,
)
```

Amounts that arrive as text are read as numbers when at least 90% of the non-blank values parse. Blanks, including a field of spaces, become missing. The Telco file has 11 blank `TotalCharges` values. They stay missing. The column stays in the model. It is roughly tenure times the monthly charge, not a label leak.

Do not pass a column that is filled in only when the customer converts. Revenue that is zero exactly on non-converters is leakage, and VISTAAR will drop it from segments and models.

Public files, not bundled:

```
telco = vt.load_telco()                 # IBM mirror of the Kaggle churn file
telco = vt.load_telco("Telco-Customer-Churn.csv")
hill = vt.load_hillstrom()              # MineThatData, 64,000 customers
hill = vt.load_hillstrom("hillstrom.csv")
```

Profiling and micro-segments

```
profile = engine.profile()
segments = engine.segment()             # or results.segments() after run()
results.opportunities()
results.what_if(target_rate=0.03, definition_contains="Month-to-month")
```

A segment row:

Column	Meaning
`n`, `conversion_rate`	Full-sample description
`baseline_rate`	Overall event rate in the same table
`relative_lift`	Segment rate / baseline
`absolute_lift`	Segment rate − baseline
`wilson_low`, `wilson_high`	Wilson interval for the segment rate
`p_value_adjusted`, `significant`	Holdout test against everyone else, then Benjamini–Hochberg
`replicates`	Holdout lift has the same sign, with enough holdout rows

`significant` is not "this campaign will work". It is "this group's rate differs from everyone else's on the holdout slice".

On Telco, month-to-month is 3,875 customers at 42.7% churn (1.61x the 26.5% baseline, Wilson 41.2–44.3%). Two-year is 1,695 customers at 2.8%. Fiber optic is 41.9%. Electronic check is 45.3%.

Models and deciles

Scores used for deciles, campaigns, profit, and calibration are out-of-fold. Each row was scored by a model that did not train on that row. A model is also refit on all rows so you can score a new file. That refit is not an honest test score.

```
results.models()          # roc_auc_mean, roc_auc_std, and the other CV metrics
results.deciles()
results.score(new_df, strategy="mass")
results.export_contacts("contacts.csv", new_df, strategy="mass")
```

Decile 1 is the highest scores. The Telco CatBoost deciles (7,043 customers, 1,869 churn events, baseline 26.5%):

Decile	Customers	Events	Hit rate	Cumulative capture	Cumulative lift
1	704	537	76.3%	28.7%	2.87x
2	704	421	59.8%	51.3%	2.56x
3	704	290	41.2%	66.8%	2.23x
4	705	238	33.8%	79.5%	1.99x
5	704	153	21.7%	87.7%	1.75x
10	705	8	1.1%	100%	1.00x

KS on those scores is 0.54. CatBoost ROC-AUC is 0.849 (fold sd 0.011). Logistic regression is 0.845 (fold sd 0.015).

`vistaar_selected` on a scored file uses the out-of-fold threshold from the campaign. Deciles on that file are ranks within the file.

Campaign strategies

```
results.campaign(strategy="mass", top_deciles=2)
results.campaign(strategy="precision", target_precision=0.75, min_customers=100)
results.campaign(strategy="compare")
```

Mass / top 2 deciles. About 20% of the book. Use it when you need to catch a large share of the events and you can contact that many people. On Telco: 1,408 customers, 958 churn events, hit rate 68.0% (Wilson 65.6–70.4%), capture 51.3% (Wilson 49.0–53.5%), lift 2.56x.

Precision / 75 per 100. The largest audience whose point hit rate is at least 75%, with at least `min_customers` rows (default `max(50, 2% of n)`). On Telco, with a floor of 100: 764 customers (10.8%), 573 events, hit rate 75.0% (Wilson 71.8–77.9%), capture 30.7%. The point estimate meets 75%. The lower bound is 71.8%, so the interval does not sit entirely above 75%.

If the target is out of reach, `achievable` is false and `max_precision_at_min_customers` shows the best hit rate at the floor. On the bundled sample, 75% is not achievable at 100 customers. The best there is 68% (Wilson 58.3–76.3%).

Wilson intervals describe the out-of-fold slice the cut was read from. They are not a second, untouched confirmation sample.

Segments inside the deciles

```
inside = results.decile_segments(top_deciles=2)
inside["narrative"]
inside["segments"]
```

For each discovered rule that has at least 30 people inside the cut:

Column	Meaning
`customers_in_cut`, `share_of_cut`	How much of the contacted list this rule covers
`hit_rate_in_cut`	Event rate of those people
`lift_vs_cut`	That rate / the cut's rate
`lift_vs_baseline`	That rate / the book rate

Telco, top 2 deciles (1,408 customers, 68.0% churn versus 26.5%):

Who	Share of the cut	Hit rate in the cut	vs the cut	vs baseline
Month-to-month	100% (1,408)	68.0%	1.00x	2.56x
Month-to-month and no online security	94% (1,329)	68.4%	1.01x	2.58x
Month-to-month and no tech support	92% (1,293)	68.8%	1.01x	2.59x
Month-to-month and fiber	85% (1,191)	69.2%	1.02x	2.61x

The message for that list is month-to-month customers, and within them the ones without support add-ons or on fiber are slightly hotter. That sentence is for the creative. It is not a reason a retention offer will work.

The same block is in the HTML report under "Who is in the score cut", after you have called `decile_segments`.

Profit cut-off

```
profit = results.profit_curve(
    cost_per_contact=50,
    value_per_conversion=400,          # one value for every row
    # or: value_column="MonthlyCharges", value_multiplier=12,
    fixed_cost=0,
    budget=None,                       # stop when spend would exceed this
    capacity=None,                     # stop at this many contacts
    use_uplift=False,                  # True after results.uplift()
)
results.optimal_cut(cost_per_contact=50, value_column="MonthlyCharges", value_multiplier
```

Customers are ordered by descending score. Expected profit of a contact is `score × value - cost`. Fixed cost is subtracted once. Budget and capacity truncate the list before the maximum is taken.

The result:

Field	Meaning
`optimal["customers"]`	Audience size with the highest expected profit
`optimal["threshold_score"]`	Score of the last customer in that list
`optimal["expected_profit"]`	Expected profit at that size
`optimal["spend"]`	Contact cost plus fixed cost
`optimal["break_even_customers"]`	Last size where expected profit is still ≥ 0
`optimal["realized_profit"]`	Same ranking, using the actual 0/1 outcome instead of the score
`comparison["mass_profit"]`	Expected profit at the top-2-decile headcount
`comparison["precision_profit"]`	Expected profit at the precision-campaign headcount

Telco, cost 50, value = MonthlyCharges × 12, CatBoost out-of-fold scores:

Cut	Customers	Expected profit
Profit-maximising	5,040 (score about 0.059)	1,377,421
Top 2 deciles	1,408	817,867
75% precision list	764	507,852
Break-even	7,043 (the whole book)	1,311,511

Realised profit at the optimal cut, using actual churn, is about 1,386,147. It is close to the expected number, which is what a calibrated score should do.

Read the assumption before you quote the money. On a churn target, profit assumes the contact **retains** a customer who would otherwise have left. A year of charges is much larger than a 50 contact, so anyone with a churn probability above a few percent looks profitable, and the break-even point is the whole book. The

maximum is still interior (5,040, not 7,043) because the bottom of the list has a negative marginal value. This is the value of a perfect save. It is not evidence that a call saves anyone.

For an uplift model, pass `use_uplift=True`. Profit then uses incremental probability, which can be negative. A do-not-disturb customer reduces expected profit.

The HTML report has a "Profit curve" section and a profit chart after `profit_curve` has been called.

Calibration and monitoring

```
calibration = results.calibration()
calibration["brier"]          # lower is better
calibration["ece"]            # expected calibration error, lower is better
calibration["table"]          # equal-width bins: mean predicted vs observed rate
calibration["by_decile"]      # same check inside score deciles
results.recalibrate("isotonic")   # or "platt", cross-fit
```

Telco CatBoost: Brier 0.134, ECE 0.0096. Decile 1 predicts 0.756 and observes 0.763. Decile 10 predicts 0.012 and observes 0.011. Cross-fit isotonic moves Brier from 0.134 to 0.134 and ECE from 0.0096 to 0.0126, so the original scores stay.

Thresholds for monitoring:

PSI	Band	What to do
< 0.10	green	Keep scoring
0.10–0.25	amber	Review the column or the score
> 0.25	red	Retrain if it is the score, or if three or more features are red

CSI is the PSI of the score. KS is the distance between the training score distribution and the new one. When you pass actuals, decile stability checks that a higher decile number (worse rank) still has a lower event rate. Spearman below −0.3 counts as holding.

A retrain is recommended when the score is red, when three or more features are red, or when the rank order breaks. Otherwise one red feature is a review, and an all-green file is keep.

```
path = results.save_model("telco_model.joblib")
loaded = vt.load_model(path)
scored = loaded.score(new_month)                # vistaar_score, vistaar_decile, vistaa
report = loaded.monitor(new_month, actuals=y)    # actuals optional, aligned to new_mont
# or, while the results object is still in memory:
results.monitor(new_month, actuals=y)
```

The bundle holds the refit pipeline, the feature lists, a sample of the training columns (up to 5,000 rows), and the out-of-fold scores used as the reference. Numeric amounts that arrive as text on the new file are coerced the same way as in training.

Telco, simulated "next month" = the 2,303 customers with tenure of 48 months or more:

- Score PSI 1.28, red. KS 0.32.
- Tenure (PSI about 4.9), TotalCharges (about 2.8), and Contract (about 0.81) are red. The slice was defined on tenure, and long-tenure customers have larger totals and longer contracts.
- Recommendation: **retrain**. The population is not the training population.
- Rank order of actual churn still holds (Spearman about −0.997). The scores still sort the outcome inside that slice. The retrain flag is the shift, not a broken ranking.

A random 1,500-row slice of the same file keeps a green score PSI (about 0.01). That is the check that the monitor is not shouting on a sample from the same book.

The HTML report has a Monitoring section after `monitor` has been called.

Champion versus challenger

```
registry = vt.Registry("registry")          # a folder you choose
log_id = registry.register(results, name="logreg", model="logreg")
cat_id = registry.register(results, name="catboost", model="catboost")
registry.promote(log_id)                    # empty seat: this becomes champion
decision = registry.promote(cat_id, champion=log_id, n_bootstrap=400)
registry.compare(log_id, cat_id)
registry.list_runs()
```

Each run records: timestamp, VISTAAR version, data fingerprint (row count, target, baseline, feature list), model name, random seed and fold count, and out-of-fold metrics (AUC, KS, top-2 capture, top-2 lift, precision at the 75% target, Brier, ECE). Scores and the outcome vector are stored beside the JSON so the comparison is paired.

`model=` selects which fitted model's out-of-fold scores to store. One `run()` can register logistic regression and CatBoost separately. Both must have scored the same customers. If the outcome vectors differ, the comparison refuses.

The rule: draw the same customers many times. Compute challenger minus champion for AUC and for capture in the top 20%. Promote when at least one of those mean differences is positive and fewer than 5% of draws are non-positive, and the other metric is not significantly worse on the same rule. The decision is appended to `registry/decisions.jsonl`. The current champion id is `registry/champion.json`.

Telco, 300 bootstrap draws, CatBoost challenging logistic regression:

- AUC difference +0.0037, one-sided share of non-positive draws 0.007.
- Top-2 capture difference +0.0038, one-sided share 0.28 (not significant on its own).
- Decision: **promote**. Capture is not worse. AUC is.

Logistic regression challenging CatBoost is refused. The first model you promote into an empty registry becomes champion without a comparison.

Uplift

Set `treatment_column` to a 0/1 column. 1 is contacted, 0 is not.

```
hill = vt.load_hillstrom("hillstrom.csv")
hill["email"] = vt.hillstrom_treatment(hill, arm="any_email")
# arm="mens" or "womens" keeps that creative versus the no-email control
# and marks the other creative as missing. Drop those rows, or let uplift() drop them.
model = hill.drop(columns=["segment", "spend", "conversion"])
engine = vt.Analytics(model, target="visit", treatment_column="email", model_names=("log
results = engine.run()
uplift = results.uplift(n_splits=2)
uplift["models"]                # AUUC by learner
uplift["groups"]
uplift["deciles"]
uplift["propensity_auc"]
uplift["propensity_warning"]
uplift["assumptions"]
results.uplift_contacts("uplift_contacts.csv")
results.profit_curve(cost_per_contact=0.10, value_per_conversion=5, use_uplift=True)
```

`segment` in the Hillstrom file **is** the assignment (Mens E-Mail, Womens E-Mail, No E-Mail). Leave it out of the features. `spend` is purchases in the fortnight after the mail. Leave it out, and leave out the outcome you are not modelling.

Assumptions, printed on the result:

- Randomised assignment, or unconfoundedness given the covariates. Unconfoundedness is not testable.
- Overlap: both arms occur across the covariate range.
- SUTVA: one customer's mail does not change another's outcome.
- The outcome models are main-effects logistic regressions.
- Cross-fitting stops a row from scoring itself. It does not create a randomised experiment.

If the propensity model's AUC is above 0.60, the result warns that treatment is predictable from the covariates and does not look randomised.

Learners:

Learner	What it fits
T-learner	One outcome model for the treated rows, one for the controls. Uplift is the difference.
S-learner	One outcome model with treatment as a feature. Uplift is the prediction with treatment on minus treatment off.
X-learner	A second-stage Ridge model of the imputed individual effect.

Groups, from the T-learner scores:

Group	Rule of thumb
Persuadable	Top quartile of predicted uplift. These are the people the contact is expected to move.
Sure thing	Untreated score already at or above the base rate, and the uplift is small. They respond anyway.
Lost cause	Everyone else with a non-negative uplift.
Do-not-disturb	Predicted uplift below −1 percentage point. Leave them alone.

The contact file is sorted by predicted uplift, not by the group name. Mail the top of that list. Skip the do-not-disturb rows.

Qini at the top k is incremental outcomes versus a random mix of treated and control inside that top k. AUUC is the area under the curve. Compare it with `auuc_random`. An AUUC below the random line means the ranking is not finding incremental outcomes.

Hillstrom, visit as the outcome, any e-mail versus no e-mail (64,000 customers, visit rate 14.7%, conversion rate 0.90%):

Segment	Visit rate
Mens E-Mail	18.3%
Womens E-Mail	15.1%
No E-Mail	10.6%

Overall treated-minus-control gap on visit: 6.1 percentage points. Propensity AUC 0.50. No warning. The assignment looks randomised once `segment` is not used as a feature.

Learner	AUUC	Random AUUC
T-learner	1,489	1,300
S-learner	1,405	1,300
X-learner	1,224	1,299

The T-learner is the one to use on this file. The X-learner does not beat the random line here. Decile 1 of the T-learner has an observed uplift of 9.3 percentage points (predicted 11.3). Decile 10 has 5.2 points (predicted 1.9). The top is higher than the bottom. The middle deciles wobble, so do not over-read a single middle row.

Group	Customers	Observed uplift (treated − control)
Persuadable	16,000 (25%)	+8.1 points
Sure thing	8,296 (13%)	+5.3 points
Lost cause	39,437 (62%)	+5.4 points
Do-not-disturb	267 (0.4%)	+7.5 points

The persuadable group is where the mail moves visits the most among the large groups. The do-not-disturb tag is a **prediction** of harm. With only 267 people the observed gap is noisy and, in this file, not negative. Do not claim the mail hurt them. Do not put them at the top of the contact list either.

Example unit economics, not Hillstrom's margin: value of an incremental visit 5, cost of an e-mail 0.10. Expected incremental profit is highest at about 61,500 customers (about 13,200). The top 2 deciles are worth about 5,000 on the same inputs. A 75% precision campaign is not available, because a 15% visit rate cannot produce a 75% hit rate.

Conversion is rare (0.90%, ATE about 0.5 points). A T-learner on conversion has AUUC about 99 against a random line of about 106. It does not beat random. Do not target conversion with this ranking.

`uplift(outcome="conversion")` uses another 0/1 column on the same table when you kept it. Drop it from the feature list if it is an outcome rather than a covariate.

Reports and exports

```
results.decile_segments(top_deciles=2)
results.profit_curve(cost_per_contact=50, value_column="MonthlyCharges", value_multiplie
results.calibration()
results.monitor(new_df, actuals=y)          # optional
results.uplift()                            # optional
results.export("report.html")               # also .xlsx, .csv, .json, .md
results.export_contacts("contacts.csv", df, strategy="mass")
results.plot("gains")                       # also decile_lift, precision, profit, calib
```

The HTML report includes a section only after the matching method has been called. Charts for profit, calibration, and the Qini curve appear when those results are on the object.

Call order that fills the report: `run`, then `decile_segments`, `profit_curve`, `calibration`, `monitor`, `uplift`, then `export`.

Command cheat-sheet

```
import vistaar as vt

df = vt.load_telco("Telco-Customer-Churn.csv")
engine = vt.Analytics(df, target="Churn", id_column="customerID", value_column="MonthlyC
results = engine.run()

results.summary()
results.segments()
results.deciles()
results.campaign("mass", top_deciles=2)
results.campaign("precision", target_precision=0.75, min_customers=100)
results.decile_segments(top_deciles=2)
results.profit_curve(cost_per_contact=50, value_column="MonthlyCharges", value_multiplie
results.optimal_cut(cost_per_contact=50, value_column="MonthlyCharges", value_multiplier
results.calibration()
results.recalibrate("isotonic")
results.save_model("model.joblib")
vt.load_model("model.joblib").monitor(new_df, actuals=y)
vt.load_model("model.joblib").score(new_df)

registry = vt.Registry("registry")
registry.register(results, model="catboost")
registry.compare(champion_id, challenger_id)
registry.promote(challenger_id, champion=champion_id)

hill = vt.load_hillstrom("hillstrom.csv")
hill["email"] = vt.hillstrom_treatment(hill, arm="any_email")
up = vt.Analytics(hill.drop(columns=["segment", "spend", "conversion"]), target="visit",
up_results = up.run()
up_results.uplift()
up_results.uplift_contacts("uplift.csv")
up_results.profit_curve(cost_per_contact=0.10, value_per_conversion=5, use_uplift=True)

results.export("report.html")
results.export_contacts("contacts.csv", df, strategy="mass")
```

Sampling a large book

```
engine = vt.Analytics(
    df,
    target="subscribed",
    id_column="customer_id",
    sample="auto",                  # True, False, or "auto"
    sample_row_threshold=500_000,  # auto starts at this row count
    sample_memory_mb=512,
    sample_size=400_000,           # or sample_frac=0.1
    random_state=42,
)
```

```
results = engine.run()
results.representativeness()       # status, text, feature table
results.decisions()                # one line per automatic choice
```

`auto` samples when the table has at least 500,000 rows or the rough size is at least 512 MB. The draw is stratified on the target, up to two low-cardinality categoricals, and one binned numeric, with proportional allocation. Rare strata are folded into the target so a tiny cell is not over-sampled. The sample's positive count is adjusted to the population rate.

The representativeness card compares the sample with the population: target rate, PSI, a Kolmogorov–Smirnov statistic, standardized mean difference, and category share gaps. PASS means the target-rate gap is within `max(0.2 percentage points, 1/sqrt(n))`, no feature has PSI above 0.25, at most one feature has PSI above 0.10, no numeric mean difference is above 0.10, and no category share moves by more than 3 points. These are conventions. If the card fails, VISTAAR enlarges the sample once and stratifies on the target only. A second failure stays FAIL.

Search and model fitting run on the sample. Deciles and campaign counts are then recomputed by scoring the full book in chunks with the refit model. The report says so. AUC, Gini, KS, and capture used to pick the model stay out-of-fold on the sample. Do not quote the full-book hit rate as an untouched test.

On a Spark DataFrame larger than `spark_sample_limit`, the first pull uses `sampleBy` on the target so each class is kept with the same probability.

Choosing the model

```
engine = vt.Analytics(
    df,
    target="y",
    model_selection="auto",          # default when you do not pass model_names
    model_objective="capture_at_20", # also precision_at_target, lift_at_decile1, ks, gi
    time_budget_seconds=0,           # set this to turn tuning on
    balance_classes=None,            # None: on only when the event rate is outside 15-8
    allow_ensemble=True,
)
results.models()   # oof_roc_auc, oof_gini, oof_ks, oof_pr_auc, oof_capture_at_20, oof_l
```

Auto compares a scorecard logistic (`logreg_woe`), an ordinary logistic, and histogram gradient boosting. One of CatBoost, LightGBM, or XGBoost is added when the table has at least 20,000 rows or you set a time budget, and only if that library is installed. Passing `model_names=` locks the list and keeps AUC as the sort unless you also pass `model_selection="auto"`.

Class weights are used when the event is rare. Campaign scores are then cross-fit isotonic, so a 75-per-100 target is read on the frequency scale rather than on a reweighted score.

Tuning runs only when `time_budget_seconds` is positive. Optuna is used if it is installed; otherwise a short random search. The chosen hyperparameters are frozen and scored with a fresh out-of-fold pass. That pass is honest for those hyperparameters. It is slightly optimistic about the search, because the search saw the same rows. If the tuned model does not beat the default on that pass, the default stays.

A blend of the top two models is kept only when a paired bootstrap says it beats the leader by more than noise (mean gain above 0.005 and fewer than 5% of draws non-positive). Otherwise the single model stays. Simpler wins ties.

The leaderboard reports AUC, Gini (`2*AUC-1`), KS, PR-AUC, capture in the top 20%, and lift in the top decile. Fold spread above 0.05 on AUC, or decile rates that are not ordered (Spearman of decile number with the rate not below −0.3), is called out in the decisions.

`duration`, `call_duration`, `talk_time`, and `contact_duration` are excluded as post-contact fields. Call length is known only after the conversation, so it cannot choose whom to call. Override with `leakage_keep_columns=("duration",)` if the name is not a call length.

If `date_column` parses as dates, the holdout is the latest slice and cross-validation trains only on earlier rows.

Reason codes on a contact list:

```
results.reason_codes(df.head(200), limit=200)
results.export_contacts("contacts.csv", df, strategy="mass", reasons=True)
```

Each code is the feature that moves that row's score the most when it is replaced by a typical value, or a SHAP value when a tree model and the shap package are both available. It is a local sensitivity, not a cause. The default limit is 400 rows.

Segment portfolio

```
results.portfolio()["portfolio"]     # non-overlapping cover of the top deciles
results.portfolio()["high_lift"]
results.portfolio()["high_volume"]
```

Rules, trees, and a beam search (weighted relative accuracy: share times the rate gap) are ranked together. The composite score uses lift, a holdout significance flag, replication, size, share of events, and a shorter definition. Rules with Jaccard overlap of 0.75 or more are dropped. The portfolio then greedily covers the top score deciles. Each row has a suggested message. The message describes who is in the slice. It does not say an offer will create the event.

`forbidden_columns` and `allowed_columns` limit what the search is allowed to use.
`max_segment_conditions` caps the length of a rule (default 3).

Numbers from the 0.3 check

Telco churn, 7,043 customers, 3-fold out-of-fold, up to 12 seconds of tuning, objective capture in the top 20%. CatBoost stayed the leader. Tuned CatBoost did not beat the default hyperparameters on the fresh pass, so the default stayed. A blend did not clear the noise bar.

Model	ROC-AUC	Gini	KS	PR-AUC	Capture @ 20%	Lift, top decile
CatBoost	0.848	0.695	0.540	0.667	51.2%	2.90x
Logistic	0.845	0.690	0.529	0.655	50.6%	2.86x
Scorecard (WoE logistic)	0.843	0.686	0.531	0.648	50.3%	2.79x
HistGradientBoosting	0.839	0.678	0.524	0.652	50.1%	2.88x

Against the 0.2 CatBoost check (AUC 0.849, top-2 capture 51.3%), the 0.3 capture is 51.2% and the top-2 hit rate is 67.9% on 1,408 customers (lift 2.56x). The 75-per-100 list is 782 customers, hit rate 75.1% (Wilson 71.9–78.0%), capture 31.4%. The interval still does not sit entirely above 75%. The segment portfolio is one rule, month-to-month (3,875 customers, 42.7% churn, 1.61x, significant, replicates), and it covers the whole top-2 cut. The run took about 20 seconds.

Bank marketing (`bank-additional-full`, 41,188 rows, subscribe rate 11.3%). `duration` was excluded as a post-contact field before any model saw it. Class weights were on, then isotonic calibration moved the Brier score from 0.158 to 0.078. `emp.var.rate` was dropped for correlation 0.97 with another macro, and `euribor3m` for VIF 47.

Model	ROC-AUC	Gini	KS	Capture @ 20%	Lift, top decile
CatBoost	0.797	0.594	0.494	63.5%	4.41x
HistGradientBoosting	0.793	0.585	0.491	63.4%	4.43x
Logistic	0.777	0.553	0.466	60.8%	4.25x
Scorecard (WoE logistic)	0.773	0.547	0.454	58.1%	3.89x

Top 2 deciles: 8,237 customers, hit rate 35.6%, capture 63.3%, lift 3.16x. A 75-per-100 list is achievable for 820 customers (2.0% of the book), 615 subscriptions, hit rate 75.0% (Wilson 71.9–77.8%), capture 13.3%. The portfolio covers 97% of the top 2 deciles with two segments. The lead one is `nr.employed ≤ 5076 AND poutcome in {failure, nonexistent} AND contact = cellular`: 3,237 customers, subscription rate 39.1% (Wilson 37.5–40.8%), lift 3.47x, significant, and it replicates on the holdout.

Hillstrom visit uplift is unchanged from 0.2: T-learner AUUC 1,489 against a random line of 1,300, propensity AUC 0.50.

Synthetic campaign, 2,000,000 rows, positive rate 12.041%. Auto sampling drew 400,000 stratified on target, channel, product, and age. Representativeness PASS, and the sample rate matched the population rate to the printed precision. Scoring the full 2,000,000 with the refit logistic took the sampled pipeline to about

33 seconds. The same pipeline with sampling off, on all 2,000,000 rows, took about 110 seconds. Full-book top 2 deciles: 400,000 customers, hit rate 17.1%, capture 28.3%, lift 1.42x. A 75% hit rate is not achievable. The best hit rate at 100 customers is 16%. The signal in that synthetic book is mild, and the card says so by not inventing a precision list.

What was left off, and why

- Naive oversampling (SMOTE and copies of rare rows) is not used. It leaks across folds and distorts the hit rate. Class weights plus isotonic calibration are the substitute.
- Boruta and a full SHAP prune are not on by default. They are slow, and a correlation / VIF screen plus the existing permutation importance already drops the dead columns.
- A fully nested hyperparameter search (retune inside every outer fold) is not the default. It multiplies the budget by the number of folds. The fresh out-of-fold pass after the search is the check you get, and the decision text says the search itself saw those rows.
- Every installed booster is not fit. One is enough. Fitting CatBoost, LightGBM, and XGBoost on the same table rarely changes a campaign cut and always changes the runtime.
- Target encoding of high-cardinality fields is inside the weight-of-evidence scorecard (levels with fewer than 20 rows get weight 0), not a second encoder stacked on the one-hot models.

Troubleshooting

The import fails. Install the wheel in the same interpreter the notebook is using. On Databricks, restart Python after `%pip`.

CatBoost is missing. `pip install "vistaar[ml]"`, or set `model_names=("logreg",)`. The rest of the library runs without it.

TotalCharges became a category or was dropped. You are on a build from before the numeric-text coercion. 0.2 reads it as a float. Eleven blanks stay missing.

75% precision says it is not achievable. The score never reaches that hit rate at your minimum audience. Read `max_precision_at_min_customers`. Lower the target or the floor, or accept the mass list.

Profit says to contact almost everyone. The value per save is large relative to the contact cost, and the churn assumption treats every predicted churner as retained. Raise `cost_per_contact`, lower the value, or set `capacity`. Read the assumption string on the result before you act.

Monitoring says retrain on a file I built by filtering. If you filter on tenure, tenure's PSI will be red and the recommendation will follow. That is the monitor doing its job. A random slice of the training book should stay green.

The registry will not compare two runs. The out-of-fold outcome vectors differ. Register both models from the same `results` object with `model=`, or refit both on the same rows and the same fold seed.

Uplift warns that treatment is predictable. A covariate is standing in for the assignment. On Hillstrom that covariate is `segment`. Drop it. If the warning remains on your own test, the assignment was not random and the uplift scores are not incremental effects.

Propensity AUC is fine but AUUC is below the random line. The ranking is not finding incremental outcomes. Do not mail by that score. The Hillstrom conversion model is in this situation. The visit T-learner is not.

The HTML report is missing profit, calibration, or uplift. Those sections are written only after you call the method. Export last.

SHAP is empty. Install `vistaar[shap]`, or leave `enable_shap=True` and read `results.shap().reason`. Disabling it does not affect deciles or profit.

The progress bar is noisy. `progress=False`. On a non-TTY log you get one line per stage, not a redraw.

A word of caution on the report. The insight text refuses causal wording on purpose. If you add your own sentences, keep the same habit: rate, count, interval, and the assumption. Not "this will cause" and not "guaranteed".