Statistics & Modeling Interview (60 mins):

- The Stats & Modeling interview is designed to evaluate your ability to reason through data-driven problems using statistical thinking and modeling techniques.
- This interview will focus on how you apply quantitative concepts to real-world situations, especially those relevant to data science and analytics roles.

Toss Die - Sum is multiple of 6

A fair die toss it 6 times, each time you write down the number on the top side. What is the probability that the sum of these 6 numbers is a multiple of 6?
—-----------------------------

Each die outcome can be written as the number mod 6. Since the die has faces 1–6, the possible outcome mod 6 are {1, 2, 3, 4, 5, 0} — all equally likely. So every roll contributes a random remainder between 0 and 5.

Everything is symmetrical here. When we roll the die six times, the sum mod 6 can be anything from 0 to 5. Because the die is fair and each remainder is equally likely, all six possible remainders of the total sum (0, 1, 2, 3, 4, 5) are equally probable.

A “multiple of 6” means the total remainder is 0 (mod 6). Since all 6 remainders are equally likely,
P(sum is multiple of 6)=⅙(Because the sum modulo 6 is equally likely to be 0, 1, 2, 3, 4, or 5 by symmetry, the chance it’s a multiple of 6 is 1/6.)

Toss Die - 投出所有面

一个dice投出所有面，需要投多少次

Statistically

When you’ve already seen k unique sides, there are n−k sides left unseen
The probability that a new roll gives you a *new* side is (n−k)/n
Therefore, the expected number of rolls to see one new side is n/(n-k)
So, if we sum this over all k=0,1,...,6

$$E[T_6] = 6 \times (1 + 1/2 + ... + 1/6) \approx 14.7$$

Monte Carlo Simulation
we can simulate the experiment many times.
Start with an empty set seen to record which faces have appeared.

Then, use a while loop. Each time generate a random number from 1 to 6. Add the number to the set seen. We should count how many rolls we make, and stop when all faces appear in the set.
Does this sounds good?

```
def exp():
  seen = set()
  count = 0
  while len(seen) < 6:
     face = random.randint(1, 6) #random number from 1 to 6
     seen.add(face)
     count += 1
  return count

N = 100_000
results = [exp() for _ in range(N)]

mean = np.mean(results)
median = np.median(results)
std = np.std(results)
```

Toss Coin

What is the expected number of streaks if we toss the coin 1000 times?

答案是500.5
follow up：如果这个不是一个fair coin，e.g. head with probability 5/7

A streak means a run of the same result — like a few heads in a row or a few tails in a row.
A new streak starts at the first toss, or every time the result changes from the previous toss.

We can model this using indicators.
Let's say $I_i = 1$ if toss i is different from toss $i-1$, and 0 otherwise.
Then the total number of streaks equals one plus the summation of all indicators that mark a change in result:

$$X = 1 + \sum_{i=2}^{n} I_i.$$

For a fair coin, the chance that two consecutive tosses are different is one-half.
So the expected number of streaks is

$$E[X] = 1 + (n-1) \times \tfrac{1}{2}.$$

If we toss 1000 times, that's $1 + 999/2 = 500.5$.

Follow-up:
If the coin is biased, say heads has probability $5/7$,
then the chance of switching is not one-half anymore.
It can switch in two ways — head to tail or tail to head —
so the probability is

$$p(1-p) + (1-p)p = 2p(1-p).$$

Then

$$E[X] = 1 + (n-1) \times 2p(1-p).$$

Plugging $p = 5/7$ and $n = 1000$, we get around 409 streaks.

Intuitively, for a fair coin we switch about half the time,
but when the coin is biased, we stay on the same side longer,
so we get fewer streaks overall.

Linear Regression

X regress on Y

y=x+e, 做OLS, regress x on y，问y的系数？

For OLS： $$\beta_{X|Y} = \frac{\text{Cov}(X,Y)}{\text{Var}(Y)}$$

1. **Model & assumptions**
   $Y = X + e, \quad X \perp e, \quad \mathbb{E}[X] = \mu_X,\ \mathbb{E}[e] = \mu_e,\ \mathrm{Var}(X) = \sigma_X^2,\ \mathrm{Var}(e) = \sigma_e^2.$
2. **Moments you need**

$$\mathrm{Cov}(X,Y) = \mathrm{Cov}(X, X+e) = \sigma_X^2 + 0 = \sigma_X^2,$$

$$\mathrm{Var}(Y) = \mathrm{Var}(X+e) = \sigma_X^2 + \sigma_e^2.$$

3. **Slope (the coefficient on Y in $X \sim Y$)**

$$\beta_{X|Y} = \frac{\mathrm{Cov}(X,Y)}{\mathrm{Var}(Y)} = \frac{\sigma_X^2}{\sigma_X^2 + \sigma_e^2}.$$

   If "standard normal" for both ($\sigma_X^2 = \sigma_e^2 = 1$), then $\beta_{X|Y} = 1/2$.
4. **Intercept (optional but neat)**

$$\alpha_{X|Y} = \mathbb{E}[X] - \beta_{X|Y}\,\mathbb{E}[Y] = \mu_X - \beta_{X|Y}(\mu_X + \mu_e).$$

   With zero means, $\alpha_{X|Y} = 0$.

simulation

```
import numpy as np
def ols_slope(y, x):
    x_centered = x - x.mean()
    y_centered = y - y.mean()
    return np.mean(x_centered * y_centered) / np.mean(x_centered**2)
```

$$\hat{\beta} = \frac{\mathrm{Cov}(Y,X)}{\mathrm{Var}(X)} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}$$

```
n = 100_000
rng = np.random.default_rng(42)

sigma_x = 1.0
sigma_e = 1.0
X = rng.normal(0, sigma_x, size=n)
e = rng.normal(0, sigma_e, size=n)
Y = X + e

# Regression slopes
beta_Y_on_X = ols_slope(Y, X)   # slope for Y ~ X
beta_X_on_Y = ols_slope(X, Y)   # slope for X ~ Y
```

求E(a), E(b)

Let X and Y be random variables with E(X) = 4, E(Y) = 6, VAR(X) = 5, VAR(Y) = 3, COV(X, Y) = 0. Linear regression, Y_i = a + bX_i + e_i. What's your best estimate of E[a_hat] and E[b_hat] ?

Slope (b): $\frac{\mathrm{Cov}(X,Y)}{\mathrm{Var}(X)}$

intercept (a): $E[Y] - b^{\backslash *} E[X]$

—---

$$\min_{a,b} \frac{1}{n} \sum_{i=1}^{n} (Y_i - a - bX_i)^2.$$

For OLS, we minimize the **sample** mean squared error
if we had infinite data drawn, then becomes minimizing the *expected* squared error

$$(a^*, b^*) = \arg\min_{a,b} E[(Y - a - bX)^2].$$

To find the minimum, we can differentiate and set to zero.

$$\frac{\partial}{\partial a} E[(Y - aX - b)^2] = -2\, E[X(Y - aX - b)] = 0,$$
$$\frac{\partial}{\partial b} E[(Y - aX - b)^2] = -2\, E[(Y - aX - b)] = 0.$$

From first one: $a = E[Y] - bE[X]$

Plug into second one, $E[X(Y - (E[Y] - bE[X]) - bX)] = 0$

$$E[XY] - E[X]E[Y] - b(E[X^2] - E[X]^2) = 0.$$

$$E[XY] - E[X]E[Y] = \mathrm{Cov}(X,Y), \quad E[X^2] - E[X]^2 = \mathrm{Var}(X).$$

Under standard OLS assumptions (i.i.d. sampling, Var(X)>0, and exogeneity E[e∣X]=0): E[a] = a, E[b] = b.
(exogeneity, E[e∣X]=0, means that the regressor Xi is *uncorrelated* with the error term ei)

X, Y, Z; R^2

We have random samples for 3 variables X,Y and Z, where X and Z's sample correlation coefficient is 0. We fit a sample linear regression of Y on X with intercept, and the coefficient of the determination R^2 = 0

R^2 = 0

$R^2 = (\mathrm{corr}(X,Y))^2.$ So if R^2 = 0, corr(X,Y)=0

In multiple regression, R^2 measures how much of Y’s variance is explained by the linear combination of X and Z.
The variable X alone explains no variation in Y. So adding X doesn’t improve predictive power by itself.
So R^2 will increase from 0 to (corr(Y,Z))^2

without R^2 = 0
If all we know is that **corr(X, Z)=0**, then adding Z to a regression of Y on X increases R^2 by exactly the *unique* contribution of Z

$$R^2(Y \sim X, Z) \;=\; R^2(Y \sim X) \;+\; R^2(Y \sim Z) \;=\; R^2(Y \sim X) \;+\; \big(\mathrm{corr}(Y, Z)\big)^2$$

LR 系数not SS, Performance好

一个regression model里面系数都不显著，但是performance很好是什么原因。

Usually points to multicollinearity. Each variable’s individual effect isn’t significant, but together they predict well. I’d check the correlation matrix, then consider dimensionality reduction, or using regularization like LASSO

(a) Multicollinearity
The predictors are highly correlated with each other, so individually each coefficient appears statistically insignificant, even though together they explain the target variable well.
→ The model’s R^2 or accuracy looks good, but t-tests fail because of overlapping explanatory power.

(b) Large model complexity / overfitting
The model might be too flexible (many parameters, high-order terms, etc.), fitting noise rather than true signal.
→ On training data, it performs well, but coefficients lose interpretability and significance.

Diagnose:
(a) Check correlation matrix or VIF (Variance Inflation Factor) for multicollinearity.
(b) Compare train vs test performance to detect overfitting.

How to Fix:
Reduce multicollinearity:

- Drop or combine correlated variables.
- Use regularization (e.g. LASSO or Ridge regression).

LASSO (L1) helps select features and shrink unimportant coefficients to zero.
Ridge (L2) reduces coefficient variance, improving stability.

孩子Simulation

一个村庄的家庭只可能有1，2，3个小孩，随机simulate 100个，50%说来自1kid家庭，30%来自2kid，20%来自3kid。问这个村庄1kid家庭占比是多少
—---
Larger families contribute more kids. And we randomly selected kids, not families.

So the observed kid-level probabilities are the kid-weighted probabilities (a randomly chosen kid comes from a 1/2/3-child family)

- F_1, F_2, F_3 = number of families with 1, 2, 3 children.
- The total number of families = $F = F_1 + F_2 + F_3$.
- The total number of children = $1F_1 + 2F_2 + 3F_3$.

And we want to calculate $P(\text{1-child family}) = \frac{F_1}{F_1 + F_2 + F_3}$

Given we know that,

$$F_1 : 2F_2 : 3F_3 = 5 : 2 : 3$$

(The ratio of F1, 2F2, and 3F3 is **5 to 2 to 3**.)
So if we say the number of families with 2 children is x, then families with 3 children are also x, then families with 1 child is 5x.
The total families = 7x

$$P(\text{1-child family}) = \frac{F_1}{F_1 + F_2 + F_3} = \frac{5x}{7x} = \frac{5}{7}$$

5/7 = 71.43%

Confidence Interval

We can model this with a **multinomial distribution. 100 student responses follow a multinomial distribution with three categories — families with one, two, or three children.**

I think an easier way is to use the bootstrap method to simulate this. We can also try to approximate it statistically.

(
Statistically
So the 100 student responses follow a multinomial distribution with three categories. And for the share of students from one-child families, can be viewed marginally as a binomial variable, since each student either belongs to the one-child families or not. So we can approximate its standard error using the binomial formula, $\sqrt{p(1-p)/n}$, with n = 100 and p ≈ 0.5. (In the data, **50 out of 100** students said they were from 1-child families.). So SE = 0.05.

Then we should transform this child-level proportion into a family-level proportion. The 1-child family portion depends on all three categories, not only on the probability of students belonging to one-child families. The SE should decrease slightly.

(To calculate the SE exactly, I think we need to take **derivatives** (the delta method))

Or we can just, use a very very rough approximate standard error. Such as shrinking it from 0.05 to 0.04. Based on this, giving a 95% confidence interval around 0.71 ± 1.96 × 0.04 ≈ [0.63, 0.79]
)

Bootstrap/Simulation

Yeah I think we can use bootstrap for this. With the current observed sample, we can repeatedly resample with replacement to generate many “new” datasets of 100 kids. For each resample, we can compute a 1-family portion. Then, from results of many bootstrap iterations, we can get an estimate of sampling variance.

```
import numpy as np
import math

# 100 children: 50 say 1-child family, 20 say 2-child, 30 say 3-child
kid_counts = np.array([50, 20, 30])
# p1, p2, p3 = kid_counts / kid_counts.sum() # child-level proportions
# p1_family = p1 / (p1 + p2/2 + p3/3)

def estimate_family_1_from_childs(c):
    """
    Given counts [c1, c2, c3], compute the estimator:
        π1 = p1 / (p1 + p2/2 + p3/3)
    """
    p1, p2, p3 = c / c.sum()
    total_families = p1 + p2/2 + p3/3
    return p1 / total_families


# Nonparametric bootstrap (resample the 100 student labels directly)
# Expand the 100 student labels explicitly: 50 '1's, 20 '2's, 30 '3's
iterations = 50000                 # number of bootstrap replications
rn_g = np.random.default_rng(123) # set a random seed reproducible randomness

labels = np.repeat([1, 2, 3], kid_counts)
p_family_1_array = np.empty(iterations) # similar to np.zeros, but not
assigning each value to 0 for performance
n = 100
for i in range(iterations):
    # 1) Resample 100 labels with replacement from the observed 100 labels
    resampled = rn_g.choice(labels, size=n, replace=True)
    # 2) Turn labels back into counts [c1, c2, c3]
    c1 = np.sum(resampled == 1)
    c2 = np.sum(resampled == 2)
    c3 = n - c1 - c2
    # 3) Recompute π1 from the resampled counts
    p_family_1_array[i] = estimate_family_1_from_childs(np.array([c1, c2, c3]))

# 4) Summarize the bootstrap distribution
se = p_family_1_array.std()
ci_lo, ci_hi = np.percentile(p_family_1_array, [2.5, 97.5]) #95% percentile CI
```

```
print(f"  SE: {se:.6f}")
print(f"  95% percentile CI: [{ci_lo:.3f}, {ci_hi:.3f}]")

----------------------------
# we can further improve the loop to vectorized numpy operations to make it
faster. Numbers in vector will be stored in memory and can batch operations
idx = rn_g.integers(0, n, size=(iterations, n))  # all bootstrap draws at once
samples = labels[idx] # shape (iterations, n), each row = one bootstrap sample
c1 = (samples == 1).sum(axis=1)
c2 = (samples == 2).sum(axis=1)
c3 = n - c1 - c2
p1 = c1 / n
p2 = c2 / n
p3 = c3 / n
p_family_1_array = p1 / (p1 + p2/2 + p3/3)
----------------------------

# Optional: visualize bootstrap distribution
import matplotlib.pyplot as plt
plt.hist(boot, bins=60, density=True)
plt.title("Bootstrap distribution of p1")
plt.xlabel("p1")
plt.ylabel("Density")
plt.show()

# ------------------------------
# Parametric bootstrap (Assume Multinomial)
# ------------------------------
pi1_param = np.empty(iterations)      # to store π1 estimates
for i in range(iterations):
    # 1) Generate a synthetic "survey" of size n from Multinomial(n, phat)
    sample = rn_g.multinomial(n=100, pvals=phat)
    # 2) Recompute π1 from this synthetic sample
    pi1_param[i] = estimate_family_1_from_childs(sample)

# 3) Summarize the bootstrap distribution
se = pi1_param.std()
ci_lo, ci_hi = np.percentile(pi1_param, [2.5, 97.5])
```

Why it works

we want to know how uncertain the estimation is (e.g. standard error, confidence interval, or sampling distribution)

Bootstrapping works because the observed sample is itself a good approximation of the population. By resampling with replacement from that sample, we simulate the process of drawing new samples from the true population. Then, by recomputing the values on each resample, the variation across those bootstrap replicates approximates the sampling variability you'd see in reality. As sample size grows, this approximation becomes mathematically consistent.

When bootstrap *doesn't* work well? Very small sample (n < ~10); Highly skewed or heavy-tailed data (Resampling may underrepresent rare events); time series data;

实际问题

发放贷款Model - Prime / Near Prime

[RS] https://www.1point3acres.com/bbs/thread-1132130-1-1.html
最后问了一个实际操作中的问题，公司只会给信用分大于650以上的发放贷款，所有数据里的客户都是信用分大于650，现在公司想拓展到650分一下的客户，如何利用当前的model
我提了用synthetic control 等等，还有利用linear model的extrapolate，另外也想到了可以缓慢拓展边界，645分的人应该和650分的人在各个维度非常相似，可以假定他们的信用分是一样的，差异来自于noise，这个问题比较接近casual的问题。

https://www.1point3acres.com/bbs/thread-1030660-1-1.html
2023.11

2a. 如果已有一个random forest 模型建立在prime population上，那么如果要将模型延展到nearprime是否有问题，如果有可能会有什么问题，要求和模型本身相结合。
没答出来

2b 如果只有这个模型/数据的基础上，如何处理near prime的问题？
没答好，我认为通过A/B testing 留取数据，并分析模型的效率，以及留存数据已备新模型

2c 该留取多少数据？.
没答好，我说经验上是10K 到100K

答案

是否有问题 **- yes.**
可能会有什么问题？

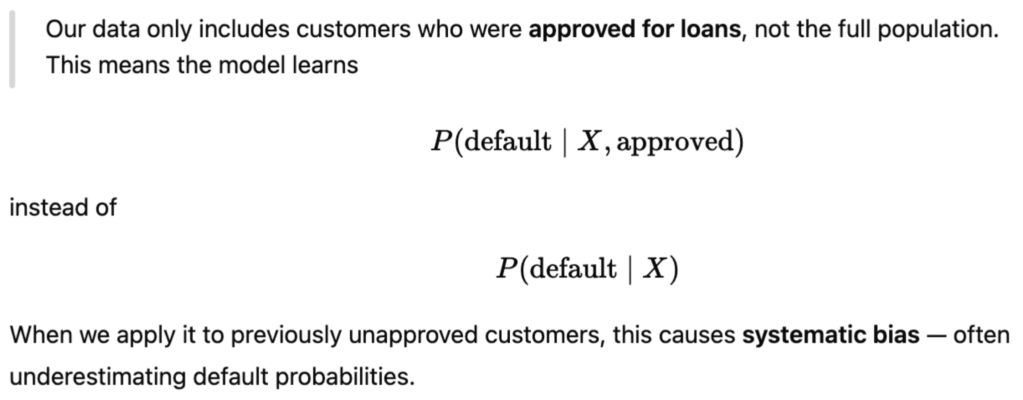
So from our data, we only see customers with score ≥650 (prime) because the business never lent below that. Therefore the existing model we trained only learns relationships **within the supported region** of the feature space.

We should not directly applying the model. I think the model isn't just "less accurate", but also **not valid** in for those users.

The model has only learned relationships within the range of 650 and above. When we apply it to lower scores, we're **extrapolating outside the data support**.
For users with score below 650, the feature distribution and outcome distribution may be fundamentally different, so predictions can be very wrong .

(Example:
Above 650, default risk might increase linearly with debt-to-income ratio;
below 650, it could increase exponentially.
→ The model would likely **underestimate risk**.)

Besides,



> Our data only includes customers who were **approved for loans**, not the full population.
> This means the model learns

$$P(\text{default} \mid X, \text{approved})$$

instead of

$$P(\text{default} \mid X)$$

When we apply it to previously unapproved customers, this causes **systematic bias** — often underestimating default probabilities.

(If the model includes nonlinear transformations: tree splits or binned variables)
Also, since it is a tree-model, then it is possible that values below 650 fall into **unseen bins**. So if we apply it to new samples with scores below 650, the trees will simply **map them to the nearest existing leaf**, often treating them as if they were 650+. That means the model won't increase predicted risk — it'll **produce misleadingly optimistic predictions**.

如何利用当前的**model**？如果只有这个模型/数据的基础上，如何处理**near prime**的问题？

So the main challenge is that our current credit model was only trained on customers with scores above 650. If we apply it directly to lower-score customers, the model has never seen this population, so predictions could be biased or unreliable.

I'd frame the solution into two stages: a short-term pragmatic approach and a long-term data-driven strategy.

<u>Short-term: Leverage current model with controlled assumptions</u>

In the short term, we can cautiously extend the model's use by assuming local similarity — that customers just below 650 (say, 640–650) behave similarly to those just above it.
We can test this assumption using simple or interpretable models like linear regression or monotonic gradient boosting, which can extrapolate more predictably than black-box models.

We can also apply uncertainty calibration or add conservative buffers to reduce risk when lending near that boundary. (manually add risk adjustments to predicted probabilities to reflect uncertainty in the unobserved region.)

This helps the business start exploring new segments without fully retraining from scratch.

<u>Long-term: collect data, re-train model with new data</u>

In the long run, the real solution is not to stretch the model, but to expand the data. We need to intentionally collect new data for <650 customers.

One way to do this is to launch a small controlled pilot, lending to a limited sample below 650 with strict limits. That allows us to observe real repayment outcomes, reduce selection bias, and retrain a more general model that truly captures the lower-score population.

(Over time, we can integrate this into a causal or policy learning framework, similar to uplift modeling or contextual bandits, to safely explore new customer segments.)

<u>Summary</u>
So in summary, short term we can extrapolate near the boundary under strict assumptions; long term, we should treat this as a data-generation and causal problem — collect real outcomes below 650 and rebuild the model on an expanded, unbiased dataset.

该留取多少数据？
We want enough data to get a statistically meaningful signal, but not so much that we take excessive business risk.

One approach is to start with a small randomized pilot test. After monitoring the guardrail metrics and validate the assumption of similarity, we can gradually expand if performance is stable.

The actual sample size depends on the **expected variance of default rate** and the **minimum detectable difference** we care about.

We can perform a power analysis based on this. We can compute how many observations are needed to detect, say, a 2% increase in default rate at 95% confidence.

房地产数据

2025 - [RS] https://www.1point3acres.com/bbs/thread-1132130-1-1.html
加州房地产数据，但是income最低的那个category(e.g. <$25000)的地区是没有在数据中的，现在我们想做一个ML model，用这个缺失数据train，但是仍然要能在全局数据中有好的表现。我就说了缓慢拓展边界的这个方法，最低income area (<$25000)应该和倒数第二低($25000-$26000)的地区非常相似，特别是这是zip code level数据，本身就average了，而且最低income实际可能就是$20000+，不太可能去到$10000的level。面试官也欣然同意了这个方法，然后就开始code了。但是总共就是可能只有50分钟不到，慢慢弄超时了，model没有完全跑完，他们的这个系统甚至连xgboost 都没有装，报错之后我还要pip install，感觉这样一个面试确实有点奇怪，其他家一般这种问题都是让做take home assignment。

So we're missing the `lowest` income bucket in data, so naïve training would force the model to extrapolate at the low end. Since the missing bucket is *close* to the lowest observed one ($25k–$26k), we can assume local continuity of the mapping between income and other features.

Especially at the **zip code level**, each observation is already an average, so noise dominates the small difference between $24k and $25k. Thus, it's reasonable to treat the missing group as approximately similar to the next-lowest one and **gradually expand the decision boundary**.

So I think a potential approach:
- Avoid unsafe extrapolation by *flooring income at $25k* (convert extrapolation
  into interpolation).
- Preserve information by adding a binary flag "below_floor" indicating the
  original income was < $25k.

(LightGBM: adding a monotonic constraint so predictions move smoothly with income. (in LightGBM)
I then stress-test the boundary by pushing $25–26k rows slightly below $25k and verifying predictions are stable.)

"I start with standard data preprocessing. I'll define the target, and also split features into numeric vs categorical to clean and process data before training"

Dataset Interview Cheat Sheet- Python/Pandas Regression/ML

```
import numpy as np
import pandas as pd

from typing import Dict, List, Optional, Tuple
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
```

```
import lightgbm as lgb

# ============
# CONFIG
# ============
TARGET = "y"                          # e.g., median_home_value
INCOME = "median_household_income"    # numeric column name for income
CAT_COLS = ["state", "urbanicity"]    # replace with your categorical cols
NUM_COLS = ["median_age", "crime_rate", "edu_bachelor_pct", INCOME]  # all
numeric incl. income

FLOOR_DOLLARS = 25_000
```

Load data

```
from sklearn.model_selection import train_test_split
train_df, test_df = train_test_split(df, test_size=0.2, random_state=42)
```

“Because the <$25k bucket is missing, I convert extrapolation into interpolation by flooring income to $25k and adding a binary flag. I preprocess train and test consistently using the same medians, modes, and one-hot schema. Then I train a RandomForest; optionally a LightGBM with a +1 monotone constraint on income to keep predictions smooth. Finally, I stress-test by nudging $25–26k rows slightly below $25k and measuring prediction drift — small drift shows stable behavior at the boundary.”

Preprocess the train data

```
X_train = train_df.drop(columns=[TARGET]).copy()
y_train = train_df[TARGET].astype(float).values

# numeric impute
for c in NUM_COLS:
    X_train[c] = pd.to_numeric(X_train[c], errors="coerce")
    X_train[c] = X_train[c].fillna(X_train[c].median())

# categorical impute
for c in CAT_COLS:
    X_train[c] = X_train[c].astype("object")
    X_train[c] = X_train[c].fillna(X_train[c].mode())

# boundary treatment: floor + flag
X_train[INCOME + "_below_floor_flag"] = (X_train[INCOME] < FLOOR).astype(int)
X_train[INCOME] = np.maximum(X_train[INCOME].values, FLOOR)

# one-hot encode categoricals
X_train_ohe = pd.get_dummies(X_train[CAT_COLS], prefix=CAT_COLS,
drop_first=False)
train_dummy_cols = list(X_train_ohe.columns)

# combine numeric + flag + OHE
```

```
X_train_proc = pd.concat(
    [X_train[NUM_COLS + [INCOME + "_below_floor_flag"]].reset_index(drop=True),
     X_train_ohe.reset_index(drop=True)],
    axis=1
)
final_cols = list(X_train_proc.columns)
```

Preprocess the test data

```
X_test = test_df.drop(columns=[TARGET]).copy()
y_test = test_df[TARGET].astype(float).values

# impute
for c in NUM_COLS:
    X_test[c] = pd.to_numeric(X_test[c],
errors="coerce").fillna(X_test[c].median())
for c in CAT_COLS:
    X_test[c] = X_test[c].astype("object").fillna(X_test[c].mode())

# boundary treatment
X_test[INCOME + "_below_floor_flag"] = (X_test[INCOME] < FLOOR).astype(int)
X_test[INCOME] = np.maximum(X_test[INCOME].values, FLOOR)

# one-hot using training schema
X_test_ohe = pd.get_dummies(X_test[CAT_COLS], prefix=CAT_COLS,
drop_first=False)
for col in train_dummy_cols:
    if col not in X_test_ohe.columns:
        X_test_ohe[col] = 0
X_test_ohe = X_test_ohe[train_dummy_cols]

X_test_proc = pd.concat(
    [X_test[NUM_COLS + [INCOME + "_below_floor_flag"]].reset_index(drop=True),
     X_test_ohe.reset_index(drop=True)],
    axis=1
)
X_test_proc = X_test_proc[final_cols]
```

Random Forest

```
rf = RandomForestRegressor(n_estimators=400, n_jobs=-1, random_state=42)
rf.fit(X_train_proc.values, y_train)

y_pred = rf.predict(X_test_proc.values)

rf_r2   = r2_score(y_test, y_pred)
rf_mae  = mean_absolute_error(y_test, y_pred)
rf_rmse = np.sqrt(mean_squared_error(y_test, y_pred))

print(f"[RF] R2={rf_r2:.4f}, MAE={rf_mae:,.0f}, RMSE={rf_rmse:,.0f}")
```

LGBM (with monotone)

```
income_idx = final_cols.index(INCOME)
    monotone = [0] * len(final_cols)
    monotone[income_idx] = +1

    lgbm = lgb.LGBMRegressor(
        n_estimators=1200, learning_rate=0.05, num_leaves=63,
        subsample=0.8, colsample_bytree=0.8, random_state=42,
        monotone_constraints=monotone
    )
    lgbm.fit(X_train_proc.values, y_train)

    y_pred_lgbm = lgbm.predict(X_test_proc.values)
    lgbm_r2   = r2_score(y_test, y_pred_lgbm)
    lgbm_mae  = mean_absolute_error(y_test, y_pred_lgbm)
    lgbm_rmse = np.sqrt(mean_squared_error(y_test, y_pred_lgbm))
    print(f"[LGBM] R2={lgbm_r2:.4f}, MAE={lgbm_mae:,.0f},
RMSE={lgbm_rmse:,.0f}")
```

CTR / Testing

还有一题很奇怪的至今都没想懂他想问什么 : 两个design下的CTR 分别是20%和30%，问是否应该launch 第二个design以及launch后真实的CTR是多少？完全不知道怎么approach..

We have two variants with observed CTRs: A = 20%, B = 30%.

1. **Should we launch B?**

Clarify

Before I analyze the results, I'd like to confirm that,

- Is this a controlled A/B test? Like both designs were tested in parallel under same conditions and random assignment?
- Do we know the sample sizes or impressions per variant?

Verify experiment

So first, I'd verify if the experiment is valid. We can quickly run some checks to make sure there is no overlap between variants, exposures are balanced, and there's stable traffic quality.

Statistical Significance Test

Then, I'd formally test if the observed 30% vs 20% CTR difference is statistically significant.
I'd use a two-sample z-test for proportions, comparing the CTRs of the two groups.
The **null hypothesis** is that both designs have the same true CTR, and the alternative is that B's CTR is higher.

I'd compute the pooled standard error and the z-score, then check the p-value or 95% confidence interval.
We also need to know the minimum detectable effect for this business goal.
So if the p-value is below 0.05 and the lift exceeds our **minimum detectable effect**, that's strong statistical evidence to prefer B.

Check Practical Significance

Even if it's statistically significant, I'd also check its practical business significance. We should know whether the CTR increase translates to meaningful business impact.
For example, if each click drives revenue, then we'll expect the CTR increase could also raise total revenue. In this part, we can sanity-check the trade-offs — maybe CTR increases but conversion rate worsens

Guardrail Metrics Checks
Besides, we need to review the guardrail metrics: such as latency, retention, or user satisfaction.
If we have more context about the CTR related to some specific segmentation, then we can also check the results across the segments. Such as geography, user groups — to ensure the improvement is consistent.

2. **If we launch, what's the expected "true CTR" post-launch?**

For the expected CTR post-launch, I would estimate the launch impact use variant B's observed CTR as the point estimate, and report its uncertainty using a confidence interval.

The 95% CI is roughly $p_B \pm 1.96 \times \sqrt{p_B(1-p_B)/n_B}$.

In practice, the observed CTR may regress a bit after launch due to novelty effects or slightly different traffic. We need to verify if the experiment run enough time to capture this, or use a long-term holdout to monitor it post-launch

Also if we are doing a gradual roll-out, the actual CTR also depends on how much traffic we have for variants.

Risks and Mitigation
Novelty effects that fade over time
Seasonality
To mitigate these, I'd monitor long-term metrics, confirm the correct unit type (e.g. randomize by user not session)

Other A/B Test Knowledge

"We use a **two-sample z-test** instead of a t-test because our metric, CTR, is a **proportion**, not a continuous mean.

When sample sizes are large, the sampling distribution of a proportion is approximately **normal** (by the Central Limit Theorem), so we can use a z-test with a known standard error derived from the binomial variance p(1−p)/np(1-p)/np(1−p)/n.
A t-test, on the other hand, is used when we're comparing **means of continuous variables** and we need to estimate the standard deviation from data."

Bias-variance trade off

https://www.1point3acres.com/bbs/thread-1031593-1-1.html
手推Bias-variance trade off；然后MAP 和MLE 的区别；l1 和L2 regularization中coefficient分别是什么分布，为什么？
1) Bias-variance trade off: break down the error term into bias, variance and irreducible error; not only definition but also derive them from scratch.

Derive bias-variance trade off
（break down the error term into bias, variance and irreducible error）

High-level
In supervised learning, we try to approximate the true relationship for Y of some function of X using a model. When fitting the model, our goal usually is to minimize the expected squared prediction error. This total error can be decomposed into three parts: bias, variance, and Irreducible error

Bias is the difference between the average prediction of our model and the correct value which we are trying to predict. Model with high bias oversimplifies the model. It always leads to high error on training and test data. -> underfitting

$$E[\hat{f}(x)] - f(x)$$

Variance is the variability of model prediction for a given data point. Model with high variance pays too much attention to training data and does not generalize on the data. So, models with high variance perform very well on training data but have high error rates on test data. -> overfitting

$$E\left[\left(\hat{f}(x) - E[\hat{f}(x)]\right)^2\right]$$

Trade-off: If our model is too simple and unable to capture the underlying pattern, then it may have high bias and low variance. On the other hand, if our model has large number of parameters then it's going to have high variance and low bias. The complex model may capture the noise along with the underlying pattern in data. So we need to find a good balance without overfitting and underfitting the data.

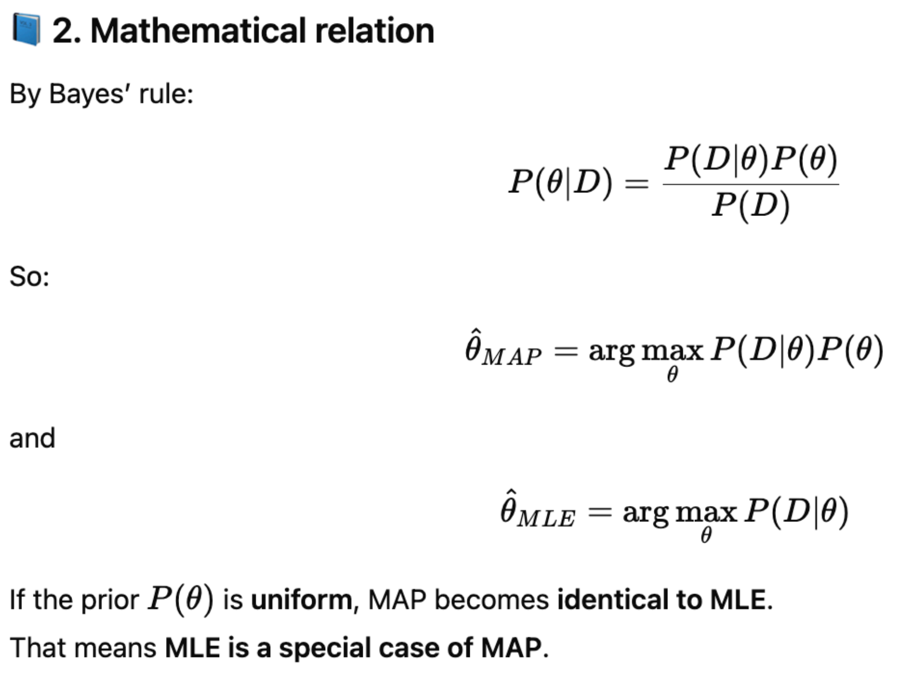
手推过程：https://chatgpt.com/c/691c0532-cd30-8327-aa74-56d3e5b86637

MAP 和MLE 的区别

MLE — or Maximum Likelihood Estimation — finds the parameter that makes the observed data most likely. It only looks at the likelihood term P(D∣θ)and doesn’t use any prior knowledge.

MAP — or Maximum A Posteriori Estimation — extends this idea by including a prior belief about the parameter. It maximizes the posterior probability P(θ∣D), which equals the likelihood times the prior.

Intuitively, we can think of MAP as MLE plus a regularization term — the prior acts as a penalty that pulls parameters toward what we already believe is likely.



📘 2. Mathematical relation

By Bayes′ rule:

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)}$$

So:

$$\hat{\theta}_{MAP} = \arg\max_{\theta} P(D|\theta)P(\theta)$$

and

$$\hat{\theta}_{MLE} = \arg\max_{\theta} P(D|\theta)$$

If the prior $P(\theta)$ is **uniform**, MAP becomes **identical to MLE**.
That means **MLE is a special case of MAP**.

⚙️ 4. Connection to regularization

Take the log of the posterior:

$$\log P(\theta|D) = \log P(D|\theta) + \log P(\theta) + const$$

- MLE → maximize log-likelihood → minimize loss = $-\log P(D|\theta)$
- MAP → same but add $-\log P(\theta)$
  → equivalent to **adding a regularization term**

Examples:

- Gaussian prior → $L2$ regularization (Ridge)
- Laplace prior → $L1$ regularization (Lasso)

L1 和L2 regularization中coefficient分别是什么分布，为什么

L1 - **Laplace** distribution (sharp peak at zero and heavier tails), encourages coefficients to be exactly zero
L2 - **Gaussian** centered at zero, gives smooth shrinkage

From a Bayesian perspective, we can interpret regularization as placing a prior on the model's coefficients.

*For **L2 regularization**, the penalty is the sum of squared weights. That corresponds to assuming a **Gaussian prior** centered at zero. Intuitively, it means we expect the parameters to be small, but not exactly zero — the Gaussian curve penalizes large weights smoothly. As a result, L2 tends to make all coefficients smaller but rarely eliminates them completely.*

*For **L1 regularization**, the penalty is the sum of absolute weights, which corresponds to a **Laplace prior**. The Laplace distribution has a sharp peak at zero and heavier tails, meaning it strongly encourages coefficients to be exactly zero while still allowing a few large values. That's why L1 tends to produce **sparse models** and performs implicit feature selection.*

In short, L2 assumes weights come from a Gaussian prior and gives smooth shrinkage, while L1 assumes a Laplace prior and encourages sparsity.