Week 6: Tree Ensembles

for Portfolio Optimization

Simple Explanations Edition

MSc Banking and Finance – FinTech Course

October 14, 2025

What You'll Learn:

- How decision trees make predictions (like a game of 20 questions!)
- Why combining many trees is better than one tree
- How Random Forest works (wisdom of the crowd)
- What Gradient Boosting does (learning from mistakes)
- Why XGBoost wins competitions
- How to explain your model's decisions

Course: Big Data & Statistical Learning
for Finance

Contents

1 Introduction: Why Should You Care About Trees?

1.1 The Problem with Lines

In Week 5, we learned about Ridge and Lasso regression. These models draw **straight lines** through your data to make predictions.

Simple Example

Imagine you're trying to predict if you should bring an umbrella:

- **Linear model:** "If cloudiness is above 50%, bring umbrella"
- **Tree model:** "If it's cloudy AND it's morning AND the humidity is high AND it's fall, bring umbrella"

Real life is complicated! Sometimes the relationship between things isn't a straight line.

Think of it Like This...

Think about deciding what to wear:

- If temperature $>$ 25°C: wear shorts
- BUT if it's raining: wear pants even if warm
- AND if it's a business meeting: wear formal clothes regardless

A straight line can't capture all these "if this, then that" rules. But a tree can!

1.2 What Makes Finance Special?

In financial markets:

- Markets behave differently when volatility is high vs. low (regime changes)
- Good returns with low risk is great, but good returns with high risk is scary (interactions)
- What works in bull markets fails in bear markets (non-linear relationships)

Trees are perfect for this! They naturally learn these complex rules.

2 Decision Trees: Like Playing 20 Questions

2.1 How a Decision Tree Works

Think of it Like This...

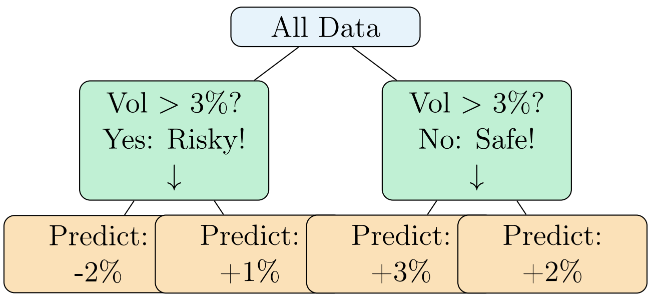
Remember the game 20 Questions? You think of an animal, and your friend asks yes/no questions to guess it:

- "Does it live in water?" – Yes
- "Is it a mammal?" – Yes
- "Is it bigger than a car?" – Yes
- "Is it a whale?" – YES!

A decision tree works **exactly like this** but with numbers!

2.2 A Real Example: Predicting Stock Returns

Let's say we want to predict if Bitcoin will go up tomorrow:



How to read this tree:

1. Start at the top (root)
2. Answer the question (Is volatility $> 3\%$?)
3. Go left for YES, right for NO
4. Keep going until you reach the bottom (leaf)
5. That's your prediction!

2.3 How Does the Tree Know What Questions to Ask?

The tree tries different questions and picks the one that best separates the data.

Simple Example

Imagine sorting coins by size and color:

- **Bad question:** "Is it shiny?" (All coins are shiny!)
- **Good question:** "Is it bigger than 2cm?" (Separates quarters from pennies)

The tree picks questions that create the most "pure" groups – groups where most things are similar.

2.4 Why Single Trees Are Problematic

Watch Out!

The Problem: A single tree is like asking only ONE person for advice.

- If your data changes a little, the tree can change a LOT
- One outlier can mess up the whole tree
- The tree might memorize the training data instead of learning patterns

This is called **high variance** – the tree is too sensitive!

Solution: Ask MANY trees for their opinion and average the answers!

3 Bias-Variance: The Goldilocks Principle

3.1 What Are Bias and Variance?

Think of it Like This...

Imagine you're throwing darts at a dartboard:
High Bias (Systematic Error):

- All your darts land in the same spot... but it's far from the bullseye
- You're **consistently wrong**
- Like always predicting "stock will go up 1%" regardless of conditions

High Variance (Random Error):

- Your darts are all over the board
- Sometimes you hit the bullseye, sometimes you miss completely
- You're **unpredictable**
- Like making wild predictions that change with tiny data changes

Just Right (Low Bias, Low Variance):

- Your darts cluster around the bullseye
- This is what we want!

3.2 The Tradeoff

Key Idea

There's usually a tradeoff:

- **Simple models:** High bias (too simple to capture reality) but low variance (stable)
- **Complex models:** Low bias (can fit anything) but high variance (too sensitive)
- **Goal:** Find the sweet spot in the middle!

3.3 Where Trees Fit In

- **Single Tree:** Low bias (can fit complex patterns) BUT high variance (unstable)
- **Ensemble of Trees:** Low bias (still complex) AND low variance (averaging helps!)

This is why we use **ensembles** – many trees working together!

4 Random Forest: Wisdom of the Crowd

4.1 The Main Idea

Think of it Like This...

Scenario: You want to estimate how many jellybeans are in a jar.
Bad Strategy: Ask one person. They might be way off.
Good Strategy: Ask 100 people and average their guesses. Studies show the average is usually very close to the true number!
This is called **wisdom of the crowd** – many imperfect guesses average out to a good answer.

4.2 How Random Forest Works

Step-by-step:

1. **Create diversity:** Make each tree different by:
   - Giving each tree a different random sample of data (with replacement – some data points appear multiple times)
   - At each split, only let the tree see a random subset of features
2. **Train many trees:** Create 100+ different trees (called a "forest")
3. **Average their predictions:** When you want to make a prediction, ask all trees and average their answers

Simple Example

Predicting Bitcoin return tomorrow:

- Tree 1 sees data from Jan-Jun, predicts: +2%
- Tree 2 sees data from Mar-Aug, predicts: +1.5%
- Tree 3 sees data from May-Oct, predicts: +2.5%
- ...
- Tree 100 predicts: +1.8%

Final prediction: Average = +2.1%

4.3 Why Random Forest Works Better

Key Idea

The Magic: When you average many trees that make different mistakes, the mistakes cancel out!
Think of it like this:

- Tree 1 might overestimate by +0.5%
- Tree 2 might underestimate by -0.3%
- Tree 3 might be spot on
- Average: Pretty close to truth!

Mathematical fact: Averaging 100 independent predictions reduces error by about 10x!

4.4 Important Settings (Hyperparameters)

Think of these as the "knobs" you can turn:

- **Number of trees:** More trees = better predictions (but slower)
  - Start with 100 trees
  - 500 trees is often enough
  - More than 1000 rarely helps
- **Tree depth:** How many questions each tree can ask
  - Too shallow: Can't capture complexity
  - Too deep: Trees memorize data
  - Sweet spot: 10-20 levels deep
- **Features per split:** How many features each tree considers
  - Common choice: square root of total features
  - Example: If you have 16 features, each tree sees 4 random features at each split

4.5 Free Validation Trick

Remember This!

Cool Feature: About 1/3 of your data isn't used by each tree (called "out-of-bag" data). You can use this leftover data to test your tree for free! No need for a separate test set. It's like having a built-in validation set!

5 Gradient Boosting: Learning from Mistakes

5.1 The Core Difference from Random Forest

Key Idea

Random Forest: All trees are created at the same time (parallel), independent of each other. This independence actually means, that the trees are trained separately and that the two randomization techniques (row bagging and feature sampling) intentionally de-correlate the individual trees to reduce the model's overall variance, which is the primary goal of the RF ensemble method.
Gradient Boosting: Trees are created one at a time (sequential), each trying to fix the previous tree's mistakes

5.2 How It Works

Think of it Like This...

Imagine you're learning to shoot basketball free throws:
Attempt 1: You shoot and miss – ball hits the front of the rim
Lesson learned: Shoot with more power
Attempt 2: You shoot harder but now it's too far – ball hits the back
Lesson learned: Reduce power slightly
Attempt 3: You adjust... and it gets closer!
Each attempt learns from the previous mistake. That's exactly how Gradient Boosting works!

5.3 The Process Step-by-Step

1. **Start simple:** Begin with a basic prediction (like the average)
2. **Find mistakes:** Look at where your prediction was wrong
   - If you predicted +2% but actual was +5%, your mistake is +3%
   - If you predicted +2% but actual was +0%, your mistake is -2%
3. **Train a tree to predict mistakes:** Build a small tree that learns to predict these errors
4. **Add this tree to your model:** Your new prediction = old prediction + (learning rate $\times$ tree's correction)
5. **Repeat:** Go back to step 2 with your new predictions, and keep improving!

Simple Example

Iteration 1:

- Prediction: +2% (just the average)
- Actual: +5%
- Mistake: +3% (too low!)

Iteration 2:

- Build tree to predict the +3% mistake
- New prediction: +2% + 0.1×(+3%) = +2.3%
- (0.1 is the "learning rate" – we don't trust each tree completely)

Iteration 3:

- Current prediction: +2.3%
- Remaining mistake: +2.7%
- Build another tree...

After 100 iterations, you get very close to the true answer!

5.4 Important Settings

- **Learning rate:** How much to trust each new tree
  - Small (0.01): Very careful, takes many trees but accurate
  - Large (0.3): More aggressive, fewer trees but might overshoot
  - **Rule of thumb:** Lower learning rate + more trees = better results
- **Number of trees:** How many correction steps to make
  - Typical: 100-1000 trees
  - Stop early if performance stops improving
- **Tree depth:** Usually keep trees small (3-7 levels)
  - **Why?** Each tree only needs to fix a small part of the problem
  - Shallow trees = "weak learners" that combine to become strong!

5.5 Why It's Powerful

Key Idea

Gradient Boosting is like having a team where:

- Member 1 makes an initial attempt
- Member 2 specifically focuses on fixing Member 1's weaknesses
- Member 3 fixes what Members 1 and 2 still get wrong
- And so on...

Each member becomes an expert at a specific type of mistake!

6 XGBoost: The Competition Winner

6.1 What Makes XGBoost Special?

XGBoost = "eXtreme Gradient Boosting"

Key Idea

XGBoost is like Gradient Boosting but with three superpowers:

1. **Built-in regularization:** Prevents overfitting automatically
2. **Optimized for speed:** Uses computer tricks to run fast
3. **Smart handling of missing data:** Doesn't break if some data is missing

6.2 Regularization: Keeping Trees Simple

Think of it Like This...

Imagine writing an essay:
No regularization: You write 50 pages with every tiny detail you know. Your teacher gets bored and you lose points for being too complex.
With regularization: You're penalized for each extra page, so you focus on the most important points. Result: A clear, concise essay that gets an A!
XGBoost penalizes trees for being too complex (too many leaves, too deep). This forces it to learn only the important patterns.

6.3 How Regularization Works

XGBoost adds penalties:

- **Penalty for number of leaves:** Too many splits? Pay a cost!
- **Penalty for large predictions:** Extreme predictions? Pay a cost!

Simple Example

Without regularization:

- Tree learns: "If it's exactly 3:47pm on a Tuesday when humidity is 47.3%, predict +8.2%"
- This is **overfitting** – memorizing random noise!

With regularization:

- Tree learns: "If volatility is high and momentum is positive, predict +2%"
- This is **generalizing** – learning real patterns!

6.4 Important XGBoost Settings

All the Gradient Boosting settings PLUS:

- **Alpha (L1 regularization):** Makes some features completely ignored
  - Like Lasso from Week 5
  - Good for feature selection
- **Lambda (L2 regularization):** Shrinks all predictions toward zero
  - Like Ridge from Week 5
  - Makes model more conservative
- **Gamma:** Extra penalty for adding new splits
  - Higher gamma = simpler trees

6.5 Why XGBoost Dominates Competitions

Remember This!

XGBoost is the "Swiss Army knife" of machine learning:

- Combines Random Forest's column sampling (randomness)
- Gradient Boosting's sequential learning (learning from mistakes)
- Ridge/Lasso regularization (staying simple)
- Plus computational tricks (runs on GPUs, parallel processing)

Result: Wins most Kaggle competitions on tabular data!

7 Understanding What Your Model Learned

7.1 Why This Matters

Watch Out!

Just because your model works doesn't mean you should trust it blindly!
You need to understand:

- Which features actually matter?
- Is the model using real patterns or random noise?
- Can you explain decisions to your boss/client?

7.2 Method 1: Counting Splits

The simplest approach: Count how often each feature is used in the trees.

Simple Example

After training 100 trees:

- "Volatility" was used in 87 trees → Very important!
- "Day of week" was used in 3 trees → Not important
- "Momentum" was used in 54 trees → Somewhat important

Problem: This can be misleading! Just because a feature is used doesn't mean it helps much.

7.3 Method 2: Permutation Importance (More Reliable)

Better approach: Shuffle a feature and see if performance drops.

Think of it Like This...

Imagine you're baking cookies, and you want to know if sugar is important:

1. Bake cookies normally → They taste great (90/100)
2. Randomly replace sugar amounts in your recipe → Cookies are terrible (20/100)
3. Conclusion: Sugar is VERY important (70 point drop!)

Now try the same with vanilla extract:

1. Normal cookies: 90/100
2. Random vanilla amounts: 85/100
3. Conclusion: Vanilla helps but isn't critical (5 point drop)

Key Idea

For each feature:

1. Measure current model performance
2. Randomly shuffle that feature's values
3. Measure new performance
4. Importance = How much performance dropped

Bigger drop = More important feature!

8 SHAP: Explaining Individual Predictions

8.1 The Problem with Regular Feature Importance

Feature importance tells you what's important **on average**, but doesn't explain:

- Why did the model predict +5% for Bitcoin **today specifically**?
- Which features pushed the prediction up vs. down?

SHAP solves this!

8.2 What is SHAP?

SHAP = **SH**apley **A**dditive ex**P**lanations

Think of it Like This...

Imagine a team project with 4 members that scores 80/100:

- Alice is brilliant at research: contributed +25 points
- Bob is great at writing: contributed +15 points
- Carol is good at editing: contributed +10 points
- Dave slacked off: contributed -5 points

Base score (if nobody worked): 35/100
Alice's contribution: +25 → Final includes her effort
Total: 35 + 25 + 15 + 10 - 5 = 80
SHAP does this for features! It fairly divides credit among all features.

8.3 How SHAP Works (Simple Version)

For each prediction, SHAP calculates:
Base value: What the model would predict on average (like "35 points if nobody worked")
Feature contributions: How much each feature pushes the prediction up or down

Simple Example

Predicting Bitcoin return:

- Base prediction: +1.5% (average across all days)
- Volatility is high today: -0.8% (pushes down)
- Momentum is positive: +1.2% (pushes up)
- Bitcoin-Ethereum spread: +0.3% (pushes up)
- Other features: +0.1% total

Final prediction: 1.5 - 0.8 + 1.2 + 0.3 + 0.1 = +2.3%
Now you know **why** the model predicted +2.3%!

8.4 Reading SHAP Plots

Summary Plot:

- Features listed from most to least important
- Each dot = one prediction
- **Red dots:** High feature value
- **Blue dots:** Low feature value
- **Position:** How much it pushed prediction left (negative) or right (positive)

Remember This!

Example Interpretation:
If you see red dots on the right for "Momentum":

- High momentum (red) → Pushes prediction higher (right)
- Makes sense: Positive momentum = higher returns!

If you see red dots on the left for "Volatility":

- High volatility (red) → Pushes prediction lower (left)
- Makes sense: High risk = lower expected returns!

8.5 Why SHAP is Powerful for Finance

Key Idea

In finance, explainability is crucial:

- **Regulators:** Want to know why you made a trading decision
- **Clients:** Need to understand recommendations
- **Risk management:** Must identify what drives predictions
- **Debugging:** Find if model learned something wrong

SHAP lets you say: "The model recommended buying Bitcoin because momentum is strong (+1.2%), despite higher volatility (-0.8%)"

9 Applying Trees to Portfolio Optimization

9.1 The Journey So Far

Let's recap how we got here:

1. **Week 4:** Used ARIMA and GARCH to forecast returns and volatility
2. **Week 5:** Created features like Sharpe ratios and momentum indicators
3. **Week 5:** Used Ridge/Lasso to make linear predictions
4. **Week 6:** Use trees to capture non-linear patterns

9.2 Why Trees Are Perfect for Portfolios

Key Idea

Financial markets have complex rules that trees naturally learn:
Market Regimes:

- Bull market (prices rising): Invest aggressively
- Bear market (prices falling): Be defensive
- Sideways market: Wait and see

A tree learns: "If momentum > 0 AND volatility $< 2\%$, allocate 60% to risky assets"
Risk-Return Balance:

- High return + low volatility = Great! Invest more
- High return + high volatility = Risky! Invest less
- Low return + low volatility = Safe but boring

Trees learn these interactions automatically!

9.3 The Process

1. **Train tree model:** Predict which assets will perform well
2. **Calculate risk-adjusted returns:** For each asset (Bitcoin, Ethereum, Dogecoin)
   - Higher predicted return = Good
   - Lower predicted volatility = Good
   - Combine these for "risk-adjusted score"
3. **Optimize weights:** Allocate more money to assets with better scores
4. **Constraints:**
   - All weights must sum to 100% (invest all your money)
   - No short selling (no negative weights)

Simple Example

Tree Predictions for Tomorrow:

- Bitcoin: +2% return, 3% volatility → Score: 0.67
- Ethereum: +1.5% return, 2% volatility → Score: 0.75
- Dogecoin: +3% return, 5% volatility → Score: 0.60

Portfolio Weights:

- Ethereum gets highest weight: 40% (best risk-adjusted score)
- Bitcoin: 35%
- Dogecoin: 25% (highest return but too risky)

10 Finding the Best Settings (Hyperparameter Tuning)

10.1 What Are Hyperparameters?

Think of it Like This...

Think of training a model like baking a cake:

- **Ingredients:** Your data (can't change)
- **Recipe:** Your algorithm (Random Forest, XGBoost, etc.)
- **Oven settings:** Your hyperparameters (temperature, time)

Just like 350°F for 30 minutes might be perfect, while 500°F for 10 minutes burns everything, the right hyperparameters make a huge difference!

10.2 Grid Search: Try Everything

Strategy: Create a checklist of settings to try, test them all, pick the best.

Simple Example

Hyperparameters to test:

- Number of trees: [50, 100, 200]
- Tree depth: [5, 10, 15]
- Learning rate: [0.01, 0.1, 0.5]

Total combinations: $3 \times 3 \times 3 = 27$ different models to test
Process:

1. Train model with trees=50, depth=5, lr=0.01
2. Test its performance
3. Train model with trees=50, depth=5, lr=0.1
4. Test its performance
5. ... (25 more times)
6. Pick the combination that performed best!

Pros: Guaranteed to find the best combination you tested
Cons: Slow if you test many parameters

10.3 Random Search: Try Random Combinations

Strategy: Instead of testing ALL combinations, randomly try 50-100 combinations.

Key Idea

Surprising result: Random search often works as well as grid search but much faster! Why? Usually only 2-3 hyperparameters really matter. Random search still explores these well.

10.4 Practical Tuning Tips

Remember This!

Smart Tuning Strategy:
Round 1 - Coarse Search:

- Try wide range: depths [3, 10, 20], trees [50, 200, 500]
- Find general area: "Ah, depth around 10 seems best"

Round 2 - Fine Search:

- Narrow down: depths [7, 10, 13], trees [150, 200, 250]
- Find exact sweet spot

Key Rules:

1. Always use cross-validation (test on multiple data splits)
2. Start with default values, then adjust
3. Don't spend hours tuning – diminishing returns!
4. More trees = almost always better (until you run out of time)

11 How Do You Know Which Model is Better?

11.1 Don't Trust Training Scores!

Watch Out!

Common Mistake:

- Train model on data from January-October
- Test on same January-October data
- Get 95% accuracy!
- **Problem:** Model memorized the answers!

This is like giving students the exam questions beforehand – of course they do well!

11.2 Cross-Validation: The Fair Test

Think of it Like This...

Imagine testing students:

Bad way:

- Give them practice problems
- Test them on the SAME problems
- They all get 100%!
- But did they really learn?

Good way:

- Give them practice problems
- Test them on DIFFERENT problems
- See if they can apply what they learned
- This shows true understanding

Cross-Validation Process:

1. Split data into 5 equal parts
2. Train on parts 1,2,3,4 → Test on part 5
3. Train on parts 1,2,3,5 → Test on part 4
4. Train on parts 1,2,4,5 → Test on part 3
5. Train on parts 1,3,4,5 → Test on part 2
6. Train on parts 2,3,4,5 → Test on part 1
7. Average all 5 test scores → That's your true performance!

11.3 Comparing Random Forest vs XGBoost

Simple Example

Results from 5-fold cross-validation:
Random Forest:

- Fold 1: 82%
- Fold 2: 85%
- Fold 3: 83%
- Fold 4: 84%
- Fold 5: 81%
- **Average: 83%** $\pm$ 1.5%

XGBoost:

- Fold 1: 87%
- Fold 2: 88%
- Fold 3: 86%
- Fold 4: 87%
- Fold 5: 87%
- **Average: 87%** $\pm$ 0.7%

Winner: XGBoost (higher score AND more consistent!)

11.4 Statistical Testing

Key Idea

Question: Is XGBoost **really** better, or did we just get lucky?
Use a statistical test (t-test) to check:

- If p-value < 0.05: Difference is **real** (statistically significant)
- If p-value > 0.05: Difference might be **luck** (not significant)

In our example: p-value $= 0.002 \rightarrow$ XGBoost is genuinely better!

12 Key Takeaways: What You Must Remember

The Big Ideas

1. Why Trees?

Real life isn't linear. Trees capture "if-then" rules naturally.

2. Single Tree Problem

One tree is like asking one person – too unstable (high variance).

3. Random Forest Solution

Ask 100+ trees, average their answers. Wisdom of the crowd works!

4. Gradient Boosting Approach

Trees learn sequentially, each fixing previous mistakes. Like iterative improvement.

5. XGBoost Wins

Combines Random Forest randomness + Gradient Boosting learning + Ridge/Lasso regularization. Competition champion!

6. No Scaling Needed

Trees don't care about feature scales. Only relative ordering matters.

7. Explain with SHAP

Don't use black boxes! SHAP tells you exactly why a prediction was made.

8. Tune Your Model

Default settings rarely optimal. Try different combinations (grid search).

9. Use Cross-Validation

Test on unseen data to ensure your model really learned patterns, not memorized answers.

10. Trees for Finance

Perfect for capturing market regimes, risk-return tradeoffs, and complex interactions.

13 Practical Advice

13.1 Start Simple, Then Improve

Key Idea

Day 1: Train a single decision tree

- See how it works
- Visualize the tree
- Understand its logic

Day 2: Add Random Forest

- Compare to single tree
- Should be much better!

Day 3: Try Gradient Boosting and XGBoost

- Find the best model
- Tune hyperparameters

Day 4: Analyze with SHAP

- Understand what model learned
- Create explanations

13.2 Warning Signs to Watch For

Watch Out!

Red Flag 1: Perfect Training Score
If your model gets 100% on training data → It memorized everything!
Solution: Make trees shallower or add regularization

Red Flag 2: Big Gap Between Train and Test
Train: 95%, Test: 70% → Overfitting!
Solution: Reduce complexity (fewer trees, less depth)

Red Flag 3: Weird Feature Importance
If "day of week" is your 1 feature → Model found spurious correlation!
Solution: Check with SHAP, remove nonsense features

Red Flag 4: Unstable Predictions
Small data change → Huge prediction change → Too sensitive!
Solution: Use more trees, cross-validation

13.3 Quick Debugging Checklist

1. ☐ Used cross-validation (not just train/test split)?
2. ☐ Checked for overfitting (train vs test gap)?

3. ☐ Tuned hyperparameters (tried different settings)?
4. ☐ Analyzed feature importance (makes sense)?
5. ☐ Created SHAP plots (understand decisions)?
6. ☐ Compared multiple models (which is best)?
7. ☐ Tested on completely unseen data (real performance)?

14 What's Next?

14.1 Week 7 Preview

Coming Next Week: Dimensionality Reduction
The Problem: What if you have 100+ features? Trees can get confused!
The Solution:

- **PCA:** Compress 100 features into 10 "super-features"
- **Clustering:** Group similar assets together
- **Factor Models:** Find hidden patterns in markets

Think of it like: Instead of memorizing 100 details about each person, remember the key traits: "sporty", "academic", "artistic"

14.2 For Your Assignment

Remember This!

Assignment Tips:

1. Start with provided code examples
2. Modify them step by step
3. Document what you tried and why
4. If something doesn't work, explain what you learned
5. Compare at least 3 models
6. Create SHAP plots for your best model
7. Write explanations a non-technical person would understand

You now understand tree ensembles at an intuitive level!
Remember:

- Trees = Smart decision-making
- Ensembles = Wisdom of the crowd
- SHAP = Explain your decisions
- Cross-validation = Honest evaluation

You're ready to build powerful ML models for finance!