Big Data in Finance

Lecture 1: Foundations of Machine Learning

Dr Daniele Bianchi

Spring 2026

Contents

Overview

This lecture sets the stage for everything that follows by introducing the foundational concepts of machine learning as applied to financial problems.

At its core, this module is about a deceptively simple question: *Can we make better use of data to improve predictions about financial outcomes?* The question sounds straightforward, but as we will discover, answering it rigorously is surprisingly difficult. Financial markets are noisy, economic fundamentals shift over time, and the history of quantitative finance is littered with strategies that looked brilliant in backtests but failed spectacularly in practice.

The lecture is divided into two parts. In Sections 1 to 4, we develop the conceptual framework: what does it mean to "learn" from data, what can go wrong, and how do we evaluate whether we have actually learned something useful? In Sections 5 to 7, we turn to practical concerns: the specific pitfalls that plague financial applications, the tools we will use throughout the course, and a preview of the two empirical problems—stock return prediction and credit risk assessment—that will serve as our running examples throughout the module.

1 Introduction: What Are We Trying to Do?

Let us begin with the big picture. Imagine you are a portfolio manager at an asset management firm. Every month, you need to decide which stocks to buy, which to sell, and in what quantities. You have access to vast amounts of data: company financials, market prices, analyst forecasts, macroeconomic indicators, perhaps even social media sentiment. The question is: how do you use all of this information to make better investment decisions?

This is fundamentally a prediction problem. You want to forecast which stocks will outperform over the coming month, and then tilt your portfolio toward those stocks. The same logic applies to many other financial problems. A bank deciding whether to approve a loan is trying to predict whether the borrower will default. An insurance company pricing a policy is predicting the likelihood of a claim. A trading firm deciding when to enter a position is making a short-term price-movement prediction.

Machine learning (ML) offers a systematic approach to these prediction problems. Rather than relying on intuition or simple rules of thumb, ML methods learn patterns from historical data and apply them to predict future outcomes. The promise is that by leveraging large datasets and flexible algorithms, we can uncover relationships that would be invisible to traditional analysis.

But there is a catch. Financial markets are extraordinarily noisy. The signal-to-noise ratio in stock returns is tiny compared to, say, image recognition or language translation, where ML has achieved remarkable success. Moreover, financial relationships are not stable; what worked in the past may not work in the future. And perhaps most importantly, the very act of discovering a profitable pattern can cause it to disappear, as other investors exploit the same opportunity.

These challenges do not mean ML is useless in finance—far from it. But they do mean we need to be extremely careful about how we apply these methods and how we evaluate their performance. A strategy that looks impressive in a backtest may be worthless or even dangerous in practice. Developing the judgment to distinguish genuine predictive power from statistical illusions is one of the central goals of this course.

1.1 Two Motivating Applications

To make these ideas concrete, we will focus on two applications throughout the module:

Market Timing. Can we forecast the return on the aggregate stock market—the equity premium—from one month to the next? Using macroeconomic and valuation indicators such as the dividend-price ratio, the term spread, and inflation, we ask whether the market's expected return is predictable over time. This is a classic and contested question in finance, connecting to a vast academic literature on "return predictability" and the out-of-sample performance of predictive regressions.

Credit Risk Assessment. Given information about a borrower—their income, employment history, credit score, the loan amount they are requesting—can we predict whether they will default? This is the problem facing banks, fintech lenders, and anyone else in the business of extending credit.

These two applications are complementary in several ways. Market timing is a regression problem (we want to predict a continuous outcome: next month's market return), while credit risk is a classification problem (we want to predict a binary outcome: default or no default). Market timing operates in an environment with very low signal-to-noise ratios, while credit risk typically has stronger predictive signals. By studying both, we will develop a more complete understanding of how ML methods perform across different settings.

1.2 Machine Learning vs. Traditional Econometrics

Before diving into the technical details, it is worth taking a step back and considering how machine learning relates to the econometric methods you may have encountered in other courses. Both approaches work with data, both involve fitting models, and both use statistical techniques. So what is the difference?

The distinction lies primarily in the goal. Traditional econometrics is usually concerned with *understanding*—specifically, with identifying causal relationships. When an economist estimates a regression of wages on education, the goal is typically to understand the causal effect of an additional year of schooling on earnings. This requires careful attention to issues like omitted variable bias, endogeneity, and identification.

Machine learning, by contrast, is primarily concerned with *prediction*. When we build an ML model to forecast stock returns, we are not necessarily claiming that our features *cause* higher returns; we are simply trying to predict which stocks will do well. This shift in focus has important implications for how we build and evaluate models.

	Econometrics	**Machine Learning**
Primary goal	Understand *why*	Predict *what*
Model structure	Often theory-driven	Mostly data-driven
Variables	Few, interpretable	Many, algorithm selects
Evaluation focus	Statistical significance	Out-of-sample accuracy

This table presents a stylized contrast, and in practice, the boundaries are blurry. Good econometricians care about predictive accuracy, and good ML practitioners care about interpretability. But the distinction is useful for understanding why ML approaches look different from what you may have seen in econometrics courses.

Perhaps more importantly, ML methods tend to be more agnostic about model structure. In econometrics, you typically start with a theory that suggests a particular functional form and a particular set of variables. In ML, you are more likely to start with a large set of candidate features and let the algorithm figure out which ones are useful and how they should be combined. This flexibility is both a strength (the algorithm can discover patterns you would not have thought to look for) and a weakness (without discipline, it is easy to "discover" spurious patterns that do not hold up out-of-sample).

A second implication is that ML places much greater emphasis on out-of-sample evaluation. In econometrics, you often evaluate a model by examining in-sample fit statistics and the significance of the coefficients. In ML, we take it for granted that a model can fit the training data well; the real test is whether it performs well on new data it has never seen. This distinction is crucial, and we will return to it repeatedly throughout the course.

2 The Prediction Problem

Let us now formalize what we mean by "prediction" and establish notation that we will use throughout the course.

2.1 Setup and Notation

We observe data on a set of **features** (also called predictors, covariates, or independent variables) and a **target** (also called the outcome, response, or dependent variable). For a single observation, we denote the features as $x = (x_1, x_2, \ldots, x_p)$, where p is the number of features, and the target as y.

In the context of return prediction, features might include firm size (market capitalization), valuation ratios (price-to-earnings, book-to-market), momentum (past returns), profitability measures, and more. The target is typically the stock's return over the next month, or an indicator of whether the stock outperforms the market.

In the credit risk context, features might include the borrower's income, employment tenure, credit score, debt-to-income ratio, and loan characteristics (amount, interest rate, purpose). The target is whether the borrower eventually defaults.

We assume we have n observations, each consisting of a feature vector and an outcome:

$$(x^{(1)}, y^{(1)}),\ (x^{(2)}, y^{(2)}),\ \ldots,\ (x^{(n)}, y^{(n)})$$

Our goal is to learn a function f that predicts y from x:

$$\widehat{y} = \widehat{f}(x)$$

The hat notation $\widehat{f}$ indicates that this is an estimate—our best guess at the true relationship, based on the data we have observed. Different ML methods correspond to different assumptions about what kind of function f might be:

- **Linear models** assume $f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p$. The predicted outcome is a weighted sum of the features.
- **Decision trees** partition the feature space into regions and predict a constant value within each region. The structure is like a flowchart: "If size is large and momentum is positive, predict high returns."
- **Neural networks** compose simple nonlinear functions into complex hierarchical representations. They can approximate virtually any function, given enough data.

We will study each of these model families in detail in subsequent lectures. For now, the key point is that we are searching for a function — without knowing in advance what form it should take — that maps features to predictions.

2.2 The Irreducible Noise

A crucial insight is that even with the best possible prediction function, we cannot predict y exactly. There is always some unpredictable component. Conceptually, we can think of the data as generated by:

$$y = f(x) + \epsilon$$

Here $f(x)$ is the "true" predictable part—the systematic relationship between features and outcomes—and ϵ is the irreducible noise. The noise captures everything that affects y but is not contained in our features x. In financial contexts, this includes:

- Information we do not have access to (e.g., private order flow, insider knowledge)
- Events that occur after we make our prediction (e.g., surprise earnings announcements)
- Measurement error in reported data
- Pure randomness in how markets process information

This decomposition is important because it sets a limit on how well any prediction method can perform. The function $f(x)$ is what we can hope to learn—it represents the signal in the data. The error ϵ is **irreducible**; no matter how sophisticated our algorithm, we cannot predict it.

In finance, noise is typically much larger than the signal. If you think about it, this makes sense. Financial markets are highly competitive, with thousands of sophisticated participants all trying to exploit predictable patterns. Any obvious pattern is quickly traded away. What remains is a world where signal-to-noise ratios are low and genuine predictability is hard to find.

2.3 Measuring Prediction Quality

How do we know if our predictions are any good? We need a way to quantify prediction error. The most common measure for continuous outcomes is the **Mean Squared Error (MSE)**:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - \widehat{f}(x^{(i)}))^2$$

This is the average of the squared differences between actual outcomes and predictions. Squaring serves two purposes: it ensures that positive and negative errors do not cancel out, and it penalizes large errors more heavily than small ones. A prediction that is off by 10% contributes 100 times more to the MSE than a prediction that is off by 1%.

Lower MSE means more accurate predictions. But there is a subtlety here that turns out to be critical: *which* data should we use to compute the MSE? If we compute it on the same data we used to fit the model (the training data), we get the **training error**. If we compute it on new data the model has never seen, we get the **test error**.

As we will see, these two quantities can differ substantially, and confusing them is one of the most common mistakes in applied ML.

3 The Bias-Variance Tradeoff

We now turn to one of the most important ideas in machine learning: the bias-variance trade-off. Understanding this trade-off is essential for building models that perform well in practice, rather than merely fitting the training data.

3.1 The Core Question

Suppose we have fit a model $\widehat{f}$ to our training data and achieved a low training error. Can we conclude that the model will make good predictions on new data?

The answer, unfortunately, is no—at least not without additional information. A model can achieve zero training error and still be completely useless for prediction. This phenomenon is called **overfitting**, and it is the central challenge in machine learning.

To understand why this happens, consider a simple thought experiment. Suppose your training data consists of 100 stock-months, and you want to predict returns using 200 characteristics. You could construct a model complex enough to essentially "memorize" all 100 training examples, achieving perfect training accuracy. But this model has not learned anything about the true relationship between characteristics and returns; it has just memorized the particular noise realizations in your training sample. When you apply it to new data, it will perform terribly.

This example is extreme, but milder versions of the same problem occur constantly in practice. Any time you use training data both to fit a model and to evaluate its performance, you risk overfitting.

3.2 Bias and Variance

To understand overfitting more precisely, it helps to decompose prediction error into two components: bias and variance.

Bias measures how far off our predictions are on average. A model has high bias if it systematically misses the true relationship — for example, if we try to fit a linear model to data with a curved pattern. High-bias models are too simple; they "underfit" the data and miss real structure.

Variance measures how much our predictions would change if we trained the model on a different sample of data. A model has high variance if it is highly sensitive to specific training samples — small changes in the data lead to large changes in the fitted model. High-variance models are too complex; they "overfit" the training data and treat noise as signal.

Mathematically, the expected prediction error decomposes as:

$$\mathbb{E}\left[(y - \widehat{f}(x))^2\right] = \text{Bias}^2 + \text{Variance} + \sigma^2$$

The first term is the squared bias—how far off are we on average? The second is the variance — how much does our prediction vary across different training samples? The third term, σ^2, is the variance of the irreducible error we discussed earlier.

3.3 The Tradeoff

Here is the key insight: bias and variance are in tension with each other. As we make our model more complex, bias tends to decrease (we can capture more of the true pattern), but variance tends to increase (we become more sensitive to the particular training sample). Conversely, as we make our model simpler, variance decreases but bias increases.

	Simple Model	**Complex Model**
Bias	High	Low
Variance	Low	High
Training Error	High	Low
Test Error	High (initially)	High (eventually)

The implications for test error are crucial. If the model is too simple (high bias), the test error is high because we are missing real patterns. If the model is too complex (high variance), the test error is again high because we are fitting noise. Somewhere in between is a "sweet spot" where the model is complex enough to capture the true signal but simple enough to avoid fitting noise.

This is why training error is a misleading guide to model quality. Training error always decreases as we make the model more complex — by adding more flexibility, we can always fit the training data better. But test error follows a U-shape: it initially decreases as we reduce bias, then increases as variance takes over. The minimum of this U-curve is where we want to be.

3.4 Why Finance Is Especially Challenging

The bias-variance tradeoff exists in every application of ML, but it is particularly severe in finance for several reasons:

Low signal-to-noise ratio. Financial markets are highly competitive. Obvious patterns get arbitraged away quickly, leaving only subtle signals buried in noise. This means that even small amounts of overfitting can swamp the true signal.

Many candidate predictors. Academic research has documented over 300 characteristics that purportedly predict stock returns. With so many potential features, the temptation to overfit is enormous. If you try enough combinations, something will appear to work by chance.

Limited data. Monthly stock returns give you only 12 observations per year. Even with decades of data, you may have only a few hundred truly independent observations. Compare this to image recognition, where you might have millions of training examples.

Non-stationarity. Financial relationships change over time. A pattern that held in the 1990s may not hold today. This means that even a model that worked well in the past may fail in the future — not because of overfitting, but because the world has changed.

These challenges do not make ML impossible in finance, but they do mean we need to be much more careful than in other domains. A healthy skepticism is warranted whenever you see impressive backtest results.

4 Model Validation

If training error is misleading and we cannot observe the true test error (because we lack access to future data), how can we evaluate our models? The answer lies in carefully designed validation procedures that simulate out-of-sample performance using the available data.

4.1 The Holdout Principle

The fundamental principle is simple: the data you use to evaluate a model must be different from the data you used to build it. If you fit a model on some data and then evaluate it on the same data, you are measuring training error, not test error.

The most straightforward implementation is to split your data into separate pieces:

- **Training set**: Used to fit the model parameters.
- **Validation set**: Used to compare different models or select hyperparameters.
- **Test set**: Used for final evaluation of the chosen model.

A typical split might be 60% training, 20% validation, and 20% test. The training set is what the model "sees" during fitting. The validation set is used to make modeling choices — what

features to include, what value of the regularization parameter to use, whether to use a linear model or a tree, and so on. The test set is held in reserve for the final evaluation.

Why do we need both a validation set and a test set? Because the validation set is "used up" as we make modeling decisions. Every time we compare two models on the validation set and choose the better one, we are implicitly fitting to the validation data. After many such choices, our validation set performance becomes optimistic, just like training performance. The test set provides an unbiased final assessment because we never made decisions based on it.

4.2 Cross-Validation

The holdout approach has a significant drawback: we are wasting data. Holding out 40% of our observations for validation and testing means we can only train on 60% of the data.

Cross-validation offers a more efficient alternative. The idea is to use each portion of the data for both training and validation, but not simultaneously. In K-fold cross-validation:

1. Split the data into K equal parts ("folds").
2. For each fold $k = 1, \ldots, K$:
    - Train the model on all data except fold k.
    - Evaluate the model on fold k and record the error.
3. Average the K validation errors to get an estimate of test error.

Common choices are $K = 5$ or $K = 10$. With 5-fold CV, each observation appears in the validation set exactly once, and we use 80% of the data for training in each iteration.

Cross-validation provides a more reliable estimate of test error by averaging over multiple train-validation splits. It also makes efficient use of limited data. The cost is computational: we have to fit the model K times instead of once.

4.3 Time-Series Cross-Validation

Here is where finance diverges from standard ML practice. The cross-validation procedure described above assumes that observations are exchangeable — any observation could equally well be in the training or validation set. But financial data is ordered in time, and this ordering matters.

If we randomly split stock returns into training and validation sets, we might end up training on January 2020 data and "validating" on December 2019 data. But this is nonsensical for a prediction problem! In December 2019, we could not possibly have known what would happen in January 2020. By using future data to predict the past, we introduce **look-ahead bias**, and our validation error becomes wildly optimistic.

The solution is **time-series cross-validation**, also called walk-forward validation. The key principle is simple: only use past data to predict future data. A typical implementation:

1. Start with an initial training period (e.g., 10 years of data).
2. Fit the model on the training period.
3. Make predictions for the next period (e.g., the next month).
4. Record the prediction error.
5. Add the new period to the training data and repeat.

This mimics how the model would actually be used in practice. At any given time, we know only what has happened up to that point, and we use that information to predict what will happen next. The accumulated prediction errors give us a realistic assessment of out-of-sample performance.

There are two common variants. In an *expanding window*, we use all available past data at each step, so the training set grows over time. In a *rolling window*, we use a fixed amount of past data, dropping older observations as new ones arrive. The rolling window approach is useful when you believe older data may be less relevant due to structural changes in the market.

4.4 The Golden Rule

We conclude this section with a rule that is simple to state but often violated in practice:

> **The Golden Rule:** Once you look at test set results, the test set is contaminated. You get one shot at honest out-of-sample evaluation.

It is tempting, after seeing disappointing test results, to go back and "fix" the model — try different features, tweak the hyperparameters, use a different algorithm — and then evaluate again on the test set. But every time you do this, you are fitting to the test set. After enough iterations, you will find something that works on the test set, but it will be pure overfitting.

Disciplined practice requires that you:

1. Make all modeling decisions using only training and validation data.
2. Lock in your final model before ever looking at the test set.
3. Evaluate once on the test set and report the result, even if it is disappointing.

This discipline is hard to maintain, especially when results are not what you hoped. But it is essential for producing research that replicates and strategies that work in the real world.

5 Common Pitfalls in Financial Machine Learning

Beyond the general challenges of overfitting and model selection, financial applications face several specific pitfalls that can make backtested performance look far better than what you would achieve in practice. Being aware of these issues — and knowing how to avoid them — is essential for anyone applying ML to financial problems.

5.1 Look-Ahead Bias

We touched on look-ahead bias in the context of time-series validation, but the problem is pervasive and warrants further examination. **Look-ahead bias** occurs whenever you use information that would not have been available at the time the prediction was made.

The most obvious approach is to use future data directly, which time-series CV prevents. But subtler forms are everywhere:

Timing of data releases. Company earnings for Q4 (ending December 31) are typically not announced until late January or February. If you use Q4 earnings to predict January returns, you are using information that investors could not have known. The same issue arises with many accounting variables.

Data revisions. Economic statistics like GDP and employment are frequently revised after their initial release. If your data source shows the revised values (which most do by default), you

are using information that was not available in real time. For example, initial GDP estimates are often revised substantially in subsequent months.

Index membership. If you are studying "S&P 500 stocks" and you use the current index composition, you are implicitly conditioning on firms having survived and grown large enough to be in the index today. The S&P 500 of 1990 looked very different from today's index.

Restated financials. Companies sometimes restate historical financial results due to accounting errors or fraud. If your data shows the restated numbers, you are using information that was not available when the original numbers were reported.

How do you avoid look-ahead bias? The gold standard is to use *point-in-time* databases that record exactly what information was available at each historical date. These exist for some data (e.g., CRSP for stock returns, IBES for analyst forecasts), but not for others. When point-in-time data is unavailable, apply conservative lags. A common rule of thumb: assume quarterly accounting data is available 45 days after quarter-end, and annual data is available 90 days after fiscal-year-end. When in doubt, add more lag. Being conservative costs you some predictive power, but using future information costs you much more — it renders your entire analysis worthless.

5.2 Survivorship Bias

Survivorship bias arises when your analysis includes only entities that "survived" to some point, excluding those that failed, were acquired, delisted, or otherwise disappeared.

Consider studying mutual fund performance. If you analyze existing funds and look at their historical returns, you are missing the funds that performed poorly and subsequently closed. The surviving funds look better on average than the full population of funds that existed at each historical date.

The same issue arises with stocks (firms that have been delisted due to bankruptcy or poor performance are excluded), hedge funds (many databases only include funds that voluntarily report, and poorly performing funds often stop reporting before closing), and corporate bonds (defaulted bonds disappear from typical datasets).

The bias can be substantial. Studies of mutual funds suggest that survivorship bias can inflate average returns by 1-2% per year. For hedge funds, which have higher attrition rates, the bias may be even larger.

To avoid survivorship bias, use databases that explicitly track all historical entities, including those that subsequently disappeared. CRSP, for example, includes all stocks that were ever listed, along with delisting returns that capture the final price movement when a stock leaves the exchange. When you cannot find survivorship-free data, be aware of the direction of the bias (usually, it makes things look better than they were) and interpret results accordingly.

5.3 Data Snooping and Multiple Testing

Data snooping is perhaps the most insidious pitfall because it happens implicitly. It occurs when you test multiple hypotheses, models, or strategies on the same data and selectively report only those that perform well.

Suppose you test 100 randomly generated trading strategies, each with no true predictive power. At a 5% significance level, you would expect about 5 of these strategies to appear "significant" purely by chance. If you then write a paper about these 5 strategies—ignoring the 95 that failed—you have created a false impression that you have discovered something real.

The problem is not limited to the explicit testing of many strategies. It also arises from researchers' degrees of freedom: choices about which variables to include, how to handle outliers, which sample period to use, which stocks to analyze, and countless other decisions. Each choice is an implicit test, and selective reporting of successful choices biases results upward.

Harvey, Liu, and Zhu (2016) argued that much of the published literature on stock return predictability suffers from exactly this problem. They document more than 300 factors reported as significant predictors of returns. But given the extent of data mining across researchers over several decades, most of these are likely false discoveries. Their proposed solution: require much higher statistical thresholds (t-statistics above 3, rather than the traditional 2) for new factor discoveries.

What can you do about data snooping? First, adjust for multiple testing. If you try n strategies, divide your significance level by n (Bonferroni correction) or use more sophisticated procedures like Benjamini-Hochberg for false discovery rate control. Second, pre-register your analysis plan before examining the data, committing to a specific methodology you will follow regardless of the results. Third, genuinely hold out test data that you never use for any modeling decisions. Fourth, insist on economic plausibility: a pattern should make sense, not just be statistically significant. Pure data-mined patterns without economic intuition are almost certainly spurious.

5.4 A Checklist for Honest Backtests

Before trusting any ML strategy in finance—whether your own or someone else's—run through this checklist:

- ✓ **No look-ahead bias.** All features use only information that was available at the time the prediction was made. This includes proper lags for accounting data and economic releases.
- ✓ **No survivorship bias.** The analysis includes failed, delisted, and closed entities, not just survivors.
- ✓ **Time-series validation.** Model selection used walk-forward validation, not random splits.
- ✓ **Clean holdout.** The final test set was never used for any modeling decisions.
- ✓ **Multiple testing adjustment.** The reported results account for all strategies and specifications that were tried, not just the winners.
- ✓ **Transaction costs.** Performance is evaluated net of realistic trading costs, including bid-ask spreads, market impact, and any borrowing costs for short positions.
- ✓ **Economic sensibility.** The results have a plausible economic interpretation. Why would this pattern exist, and why would it persist?

If a study—including your own—does not address these issues, treat the results with serious skepticism.

6 Python for Machine Learning

Having established the conceptual framework and the pitfalls to avoid, let us turn briefly to the practical tools we will use throughout the course. Python has emerged as the dominant language for machine learning in finance, and for good reason: it offers a rich ecosystem of libraries, strong data manipulation capabilities, and widespread industry adoption.

6.1 The Ecosystem

The core libraries we will use are:

NumPy provides the fundamental data structure for numerical computing: the n-dimensional array. Almost everything else is built on top of NumPy arrays.

Pandas extends NumPy with DataFrames—tabular data structures with labeled rows and columns, similar to a spreadsheet or SQL table. Pandas is essential for data cleaning, manipulation, and time-series operations.

Scikit-learn is the workhorse ML library. It provides consistent interfaces for dozens of algorithms, from linear regression to random forests to support vector machines. It also includes tools for preprocessing, model selection, and evaluation.

Matplotlib and **Seaborn** handle visualization. Matplotlib is the low-level foundation; Seaborn provides higher-level statistical graphics.

For more specialized needs, we will also use **XGBoost** for gradient-boosting (the current state of the art for tabular data), and **SHAP** for model interpretability.

6.2 The Scikit-Learn Pattern

One of the beauties of scikit-learn is its consistent API. Almost every model follows the same pattern:

```
from sklearn.linear_model import Ridge

# 1. Create model with hyperparameters
model = Ridge(alpha=1.0)

# 2. Fit to training data
model.fit(X_train, y_train)

# 3. Make predictions
predictions = model.predict(X_test)

# 4. Evaluate
score = model.score(X_test, y_test)
```

Whether you are using linear regression, random forests, or neural networks, the interface is the same: create the model, call `fit()`, call `predict()`. This consistency makes it trivial to swap one algorithm for another and compare performance.

6.3 Pipelines and Time-Series CV

Real ML workflows involve multiple steps: handling missing values, scaling features, engineering new variables, and finally fitting the model. Scikit-learn's `Pipeline` class bundles these steps together:

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer

pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler()),
    ('model', Ridge(alpha=1.0))
])
```

```
pipeline.fit(X_train, y_train)
```

Pipelines can introduce a subtle form of data leakage: if you scale your features using statistics from the entire dataset (including validation and test data), you implicitly use future information. By placing scaling within the pipeline, scikit-learn ensures that statistics are computed only on the training data.

For time-series validation, scikit-learn provides `TimeSeriesSplit`:

```
from sklearn.model_selection import TimeSeriesSplit, cross_val_score

tscv = TimeSeriesSplit(n_splits=5)
scores = cross_val_score(model, X, y, cv=tscv)
```

This ensures that validation always uses future data relative to the training set, preserving the temporal ordering of financial observations. Note that your data must be sorted by time before applying `TimeSeriesSplit`.

7 Applications Preview

Let us conclude by previewing the two empirical applications that will accompany us through the remainder of the course.

7.1 Application 1: Market Timing

The first application asks: Can we time our investment in the equity market? That is, can we forecast the return on the aggregate market—the equity premium—from one period to the next? This is a classic and contested question in finance, with a vast academic literature on "return predictability".

We will work with monthly U.S. equity-market returns and a set of macroeconomic and valuation predictors. These include valuation ratios (dividend-price, earnings-price, book-to-market), interest-rate variables (the Treasury-bill rate, the term spread, the default spread), and other macro indicators (inflation, stock-market variance). Academic research has documented in-sample predictive power for many of these variables, though, as we have discussed, the out-of-sample evidence is far weaker and data snooping is a serious concern.

A key insight we will develop is that statistical prediction accuracy and economic performance are distinct. A model with a positive out-of-sample R^2—even a small one—is genuinely predicting returns. But whether that translates into a profitable timing strategy depends on the signal's magnitude, trading costs, and the strategy's capacity.

7.2 Application 2: Credit Risk Assessment

The second application asks: Can we predict which borrowers will default? This is a classification problem (default vs. no default) rather than a regression problem, which introduces new techniques and evaluation metrics.

We will work with data from LendingClub, a peer-to-peer lending platform. The features include borrower characteristics (income, employment length, home ownership), credit history (FICO score, number of delinquencies, credit utilization), and loan characteristics (amount, interest rate, purpose, term). The target is whether the borrower eventually defaults on the loan.

Credit risk introduces several challenges we did not face in return prediction:

Class imbalance. Defaults are relatively rare events—perhaps 10-20% of loans, depending on the product. This means that a naive model predicting "no default" for everyone would be 80-90% accurate but completely useless.

Interpretability and fairness. Credit decisions are regulated. Lenders must be able to explain why they denied a loan, and they cannot discriminate based on protected characteristics like race or gender. These requirements constrain the kinds of models we can use in practice.

Despite these complications, ML methods have proven highly effective for credit risk. Khandani, Kim, and Lo (2010) showed that ML approaches significantly outperform traditional credit scorecards, and the industry has increasingly adopted these techniques.

8 Summary and Looking Ahead

This lecture has covered a lot of ground. Let us recap the key takeaways:

Machine learning is fundamentally about prediction. We learn a function $\widehat{f}$ from historical data and use it to predict outcomes for new observations. This distinguishes ML from econometrics, which focuses primarily on causal inference.

The bias-variance tradeoff is central. Models that are too simple miss real patterns (high bias); models that are too complex fit noise (high variance). Minimizing test error requires finding the right balance.

Validation must respect time. For financial applications, random train-test splits create look-ahead bias. Time-series cross-validation — using only past data to predict future data — is essential.

Finance has specific pitfalls. Beyond general overfitting, be alert to look-ahead bias (using information not available at prediction time), survivorship bias (analyzing only entities that survived), and data snooping (selectively reporting successful strategies).

Honest evaluation requires discipline. Hold out test data you never touch during model development. Report results even when disappointing. Account for all strategies tried, not just the winners.

In the next lecture, we begin our journey through specific ML methods, starting with regularized linear regression: Ridge, Lasso, and Elastic Net. These methods will be our first tools for handling the high-dimensional settings common in financial prediction, where we have many potential predictors and need to control overfitting.

Readings

Required:

- James, G., Witten, D., Hastie, T., Tibshirani, R., & Taylor, J. (2023). *An Introduction to Statistical Learning with Applications in Python*, Chapters 2 and 5.

Supplementary:

- Harvey, C. R., Liu, Y., & Zhu, H. (2016). "... and the Cross-Section of Expected Returns." *The Review of Financial Studies*, 29(1), 5–68. This paper documents the multiple testing problem in factor research and proposes higher statistical thresholds for new discoveries.
- Chordia, T., Goyal, A., & Saretto, A. (2020). "Anomalies and False Rejections." *The Review of Financial Studies*, 33(5), 2134–2179. This paper shows that accounting for correlations among anomaly returns changes conclusions about which factors are robust.

- Gu, S., Kelly, B., & Xiu, D. (2020). "Empirical Asset Pricing via Machine Learning." *The Review of Financial Studies*, 33(5), 2223–2273. A comprehensive study of ML methods for return prediction. Read the Introduction and Section 2 for background.