Session 3.3: Practical Machine Learning with Scikit-Learn: Model Training and Validation

THE BUSINESS SITUATION

- A commercial property valuation firm prices office buildings by hand, which takes an expert several days per property
- It has the sale records of 14 recent deals: size, number of rooms, distance to the city center, tax rate, and price
- The valuation manager asks: can a model estimate a property's sale price from its size and location, and how far off will it be? Today, you train your first machine learning model and measure its error honestly

Today's flow: 60 min lecture, then 120 min hands-on lab

EXPECTED OUTCOMES

By the end of today, you will be able to:

1. ***"Can a model learn prices from past deals?"***
   Train a linear regression model with Scikit-Learn's `.fit()` and read its coefficients (how much the price changes for one more unit of each feature)
2. ***"How far off are the predictions?"***
   Measure the error with RMSE (Root Mean Squared Error) and MAE (Mean Absolute Error), and check stability with cross-validation (testing on several different splits)
3. ***"What will this new building sell for?"***
   Predict the price of one new property, and fix the `ValueError` a one-row prediction often raises

Session Map: Follow the Numbers

Today's question: *Can a model estimate a property's sale price from its size and location, and how far off will it be?*

1 Why it matters
Lecture, part 1

1. **Session Purpose**
   answer today's question
2. **Learning Outcomes**
   3 skills you will show today
3. **Business Context**
   When a Price Model Is Worth Using
4. **Data Basics**
   How Linear Regression Works

→

2 Learn the tools
Lecture, part 2

5. **Lab Dataset**
   1 CSV
6. **Keywords**
   6 terms
7. **Core Foundations**
   5 ideas, one at a time
8. **IDE Setup**
   `pip install scikit-learn`
9. **Code Examples**
   3 demos + Quick Checks
10. **References**
    2 official docs

→

3 Practice
Lab

11. **Exercise 1: Guided**
    Property Valuation Engine
12. **Exercise 2: Applied**
    Error per Property and a One-Row Prediction
13. **Exercise 3: Advanced**
    Does a Bigger Model Price Better?

When a Price Model Is Worth Using

KEY POINTS

- An estimate is useful if its typical error is small compared with the price.
- A model learns from past deals only.
- Regression vs classification:
  - **Regression** predicts a number, such as a price. Today's model, linear regression, is one example.
  - **Classification** predicts a category, such as "fraud" or "not fraud". Logistic regression, used in Session 3.4, is one example.

How Linear Regression Works

KEY POINTS

- Linear regression finds the best straight-line formula: `price = intercept + coefficient_1 x feature_1 + coefficient_2 x feature_2 + ...`. "Best" means the smallest total squared error on the training rows.
- Reading a coefficient: the change in price for one more unit of that feature, **while the other features stay the same**.
- Coefficients can surprise you.
- Three error measures, in plain words:

AT A GLANCE

Measure	Meaning	Lower or higher is better?
MAE (Mean Absolute Error)	the average size of the errors, in dollars	lower
RMSE (Root Mean Squared Error)	like MAE, but large errors count extra	lower
R-squared	the share of the price differences the model explains, from 0 to 1	higher

The Data for Today

FILES

- `data/session_03_03_dataset.csv`

Audited 14-row commercial property sales dataset (fictional teaching data), one row per sale:

COLUMNS

`Property_ID`	Sale code (e.g., `PROP-401`).
`Square_Feet_Units`	Usable floor area in square feet.
`Bedrooms_Count`	Number of separate rooms or office suites.
`Distance_To_CBD_Miles`	Distance to the CBD (Central Business District, the city's main business center), in miles.
`Local_Tax_Rate_Pct`	Local property tax rate, in percent.
`Sale_Price_USD`	The price the property sold for, in U.S. Dollars. This is the target.

6 Terms You Need Today

TERM 1

`Estimator` (`.fit()` / `.predict()`)

What: any Scikit-Learn model; `.fit(X, y)` learns from examples, `.predict(X)` makes predictions

Why: every model in the library uses the same two methods

TERM 2

`Linear Regression`

What: a model that predicts a number with a straight-line formula

Why: simple, fast, and its coefficients are easy to explain to a manager

TERM 3

`Logistic Regression`

What: a model that predicts the probability of a yes/no outcome

Why: the classification partner of linear regression (Session 3.4)

TERM 4

`RMSE` and `MAE`

What: the typical prediction error, in the target's own unit (here, dollars)

Why: tells the manager how far off an estimate can be

TERM 5

`Cross-Validation`

What: splitting the data into K parts and testing K times, each time on a different part

Why: one lucky split can make a model look better than it is

TERM 6

`Overfitting`

What: a model that memorizes its training rows and does worse on new data

Why: the main risk when data is small; testing on unseen rows reveals it

How It Works

7.1

The estimator pattern

Remember: create, fit, predict; learned values end with `_`.

7.2

Reading the price formula

Remember: a coefficient is the effect of one feature with the others held fixed.

7.3

Measuring error

Remember: MAE is the plain average error; RMSE punishes large errors; test on unseen rows.

7.4

One split is not enough

Remember: report the average over the folds, and look at the worst fold too.

7.5

One row is still a table

Remember: one row = a one-row DataFrame with the same column names.

7.1 The estimator pattern: On Real Data

SEE IT ON REAL DATA

Step	Code	What happens
Create	`model = LinearRegression()`	an empty model, nothing learned yet
Fit	`model.fit(X_train, y_train)`	learns the price formula from 10 sales
Predict	`model.predict(X_test)`	4 estimated prices for the 4 test sales
Score	`model.score(X_test, y_test)`	`0.996`, the share of price differences explained

Remember: create, fit, predict; learned values end with `_`.

Train a Price Model

YourName_S33_demo_1.py — Input → Process → Output

```python
# 1. Input: load the sales and split them
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
df = pd.read_csv("data/session_03_03_dataset.csv")
features = ["Square_Feet_Units", "Bedrooms_Count", "Distance_To_CBD_Miles"]
X = df[features]
y = df["Sale_Price_USD"]
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42
)

# 2. Process: learn the price formula from the training rows
model = LinearRegression()
model.fit(X_train, y_train)

# 3. Output: the learned formula
print("Intercept:", round(model.intercept_, 1))
for name, coef in zip(features, model.coef_):
    print(f"{name}: {coef:,.1f}")
```

HOW IT WORKS

- First time only, in the Terminal: `pip install scikit-learn`
- `features = [...]` lists the 3 input columns; `X = df[features]` and `y = df["Sale_Price_USD"]` separate inputs and answer (Session 3.2).
- `train_test_split(..., test_size=0.25, random_state=42)` keeps 4 of the 14 sales aside for testing.

READ THE OUTPUT

```
Intercept: 123568.1
Square_Feet_Units: 180.1
Bedrooms_Count: 14,703.6
Distance_To_CBD_Miles: -1,915.8
```

- Each extra square foot adds about `$180`, each extra room about `$14,700`, and each extra mile from the center subtracts about `$1,900`, with the other features held fixed (section 7.2).

Quick Check 1: `.score()`

NEW TOOL

`.score()`

`model.score(X_test, y_test)` returns R-squared on the test rows in one call. Why: a quick first check of how well the model explains prices it has not seen.

YOUR TASK

Fill in the Blank

Add one line to part **3. Output**: `print("Test R-squared:", round(model._____(X_test, y_test), 3))`

Check: the Terminal shows `Test R-squared: 0.996`.

Predict the Test Sales and Measure the Error

YourName_S33_demo_2.py — Input → Process → Output

```
# 1. Input: load, split, and train (same as Micro-Example 1)
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
df = pd.read_csv("data/session_03_03_dataset.csv")
features = ["Square_Feet_Units", "Bedrooms_Count", "Distance_To_CBD_Miles"]
X_train, X_test, y_train, y_test = train_test_split(
    df[features], df["Sale_Price_USD"], test_size=0.25, random_state=42
)
model = LinearRegression().fit(X_train, y_train)

# 2. Process: predict the 4 test sales and compute the RMSE
y_pred = model.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
comparison = pd.DataFrame({
    "Property_ID": df.loc[X_test.index, "Property_ID"],
    "Actual_USD": y_test,
    "Predicted_USD": y_pred.round(0),
})

# 3. Output: actual vs predicted, then the typical error
print(comparison)
print(f"Test RMSE: ${rmse:,.0f}")
```

HOW IT WORKS

- Part 1 loads, splits, and trains, as in Micro-Example 1.
- `model.predict(X_test)` returns 4 estimated prices.
- `mean_squared_error()` averages the squared errors; `np.sqrt()` (square root, from NumPy) brings the result back to dollars: the RMSE (section 7.3).

READ THE OUTPUT

```
   Property_ID  Actual_USD  Predicted_USD
9     PROP-409    465000.0       460268.0
11    PROP-411    330000.0       345735.0
...
```

- The model is close on the large building (`PROP-412`: off by `$2,141`) and on `PROP-409`.

Quick Check 2: `mean_absolute_error()`

NEW TOOL

`mean_absolute_error()`

`mean_absolute_error(y_test, y_pred)` returns the average error size in dollars. Why: the easiest error number to explain: "on average, we are off by this much".

YOUR TASK

Fill in the Blank

Change the import line to `from sklearn.metrics import mean_squared_error, mean_absolute_error`, then add one line to part **3. Output**: `print(f"Test MAE: ${____________________(y_test, y_pred):,.0f}")`

Check: the Terminal shows `Test MAE: $9,668`, lower than the RMSE because RMSE weighs the two larger errors (about $16,000 each) more heavily.

Five Splits Instead of One

YourName_S33_demo_3.py — Input → Process → Output

```
# 1. Input: load the sales
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
df = pd.read_csv("data/session_03_03_dataset.csv")
features = ["Square_Feet_Units", "Bedrooms_Count", "Distance_To_CBD_Miles"]

# 2. Process: train and test 5 times, each time on a different part
scores = cross_val_score(LinearRegression(), df[features],
                         df["Sale_Price_USD"], cv=5, scoring="r2")

# 3. Output: the 5 scores and their average
print("R-squared per fold:", scores.round(3).tolist())
print(f"Average R-squared: {scores.mean():.3f}")
```

HOW IT WORKS

- `cross_val_score(LinearRegression(), X, y, cv=5, scoring="r2")` does all the work of section 7.4: it cuts the 14 sales into 5 folds, trains 5 fresh models, and tests each one on its own fold.
- It returns 5 scores, one per fold.
- `scores.mean()` averages them.

READ THE OUTPUT

- Every fold scores above `0.99`: the model does well no matter which sales are held out, not just on one lucky split.

Quick Check 3: RMSE per fold

NEW TOOL

RMSE per fold

`scoring="neg_root_mean_squared_error"` makes `cross_val_score()` return the RMSE of each fold as a negative number (Scikit-Learn treats higher scores as better), so put a minus sign in front to read it. Why: shows the error in dollars for every fold.

YOUR TASK

Fill in the Blank

Add two lines to part **3. Output**: `rmse = -cross_val_score(LinearRegression(), df[features], df["Sale_Price_USD"], cv=5, scoring="_____")` and `print(rmse.round(0).tolist())`

Check: `[7862.0, 15224.0, 4004.0, 8201.0, 3374.0]`. The error ranges from about $3,400 to $15,200 depending on which properties are tested.

Where to Look Things Up

REFERENCE

Scikit-Learn: Linear Models (LinearRegression Documentation)

https://scikit-learn.org/stable/modules/linear_model.html

REFERENCE

Scikit-Learn: Cross-Validation Metrics and Scoring

https://scikit-learn.org/stable/modules/cross_validation.html

REFERENCE

Scikit-Learn: Regression Metrics (MAE, RMSE, R-squared)

https://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics

REFERENCE

Gareth James et al.: An Introduction to Statistical Learning (ISLR with Applications in Python)

https://www.statlearning.com/

Three Exercises

EXERCISE 1: GUIDED

Property Valuation Engine

Goal: Train, predict, and evaluate a price model

File:

`YourName_S33_regression.py`

Check: `Test RMSE: $11,539 | R-squared: 0.996`

EXERCISE 2: APPLIED

Error per Property and a One-Row Prediction

Goal: Show each property's error and price one new building

File:

`YourName_S33_practice.py`

Check: MAPE `2.68`%

EXERCISE 3: ADVANCED

Does a Bigger Model Price Better?

Goal: Do more features give a better price model?

File:

`YourName_S33_challenge.py`

Check: model C has the lowest average RMSE (`$7,733`)

Full steps for all three exercises: sections 11-13 of the course page.