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

PEDAGOGICAL AGENDA

1:2 Delivery Rhythm

Session Learning Architecture

Structured 3-hour applied computing session designed for business students with zero prior coding experience.

- **Phase 1 (60 Min Lecture):** Computational foundations, business vocabulary, and syntactic mechanics.
- **Phase 2 (120 Min Lab):** Guided hands-on implementation, error diagnosis, and AI co-pilot resolution.
- **Zero Fluff Mandate:** Focus directly on executable Python code, tabular data wrangling, and commercial intelligence.

STRATEGIC OUTCOME

Operational Autonomy

Business Analytical Competency

Transforming passive spreadsheet users into autonomous programmatic decision makers.

- **Speed & Scale:** Processing datasets exceeding spreadsheet limits with instantaneous vectorized execution.
- **Repeatable Pipelines:** Replacing manual copy-paste workflows with fully automated Python scripts.
- **AI Debugging Literacy:** Leveraging modern IDEs to diagnose and resolve errors independently.

Core Computational Keywords & Business Meaning

CONCEPT 01

Core Keyword

Unified Estimator API

The standard three-stage Scikit-Learn interface: instantiate model, fit on training data (`.fit()`), predict on new data (`.predict()`), and evaluate accuracy (`.score()`).

CONCEPT 02

Core Keyword

Linear Regression

A fundamental supervised regression algorithm that models the linear relationship between continuous independent variables and a target business metric.

CONCEPT 03

Core Keyword

Logistic Regression

A foundational supervised classification algorithm that models the probability of a binary categorical event using the logistic sigmoid function.

CONCEPT 04

Core Keyword

K-Fold Cross-Validation

Splitting training data into K subsets to iteratively train and validate the model K times, providing an unbiased estimate of generalization performance.

Computational Foundations & Business Operations

PILLAR 01 Architecture

**Core Programming & Computational Foundations

**

PILLAR 02 Architecture

The Scikit-Learn Estimator Interface

Every model in Scikit-Learn (whether simple linear regression or advanced random forests) implements the identical class interface. Learning `.fit()` and `.predict()` unlocks hundreds of advanced machine learning algorithms.

PILLAR 03 Architecture

Linear Regression for Continuous Business Forecasting

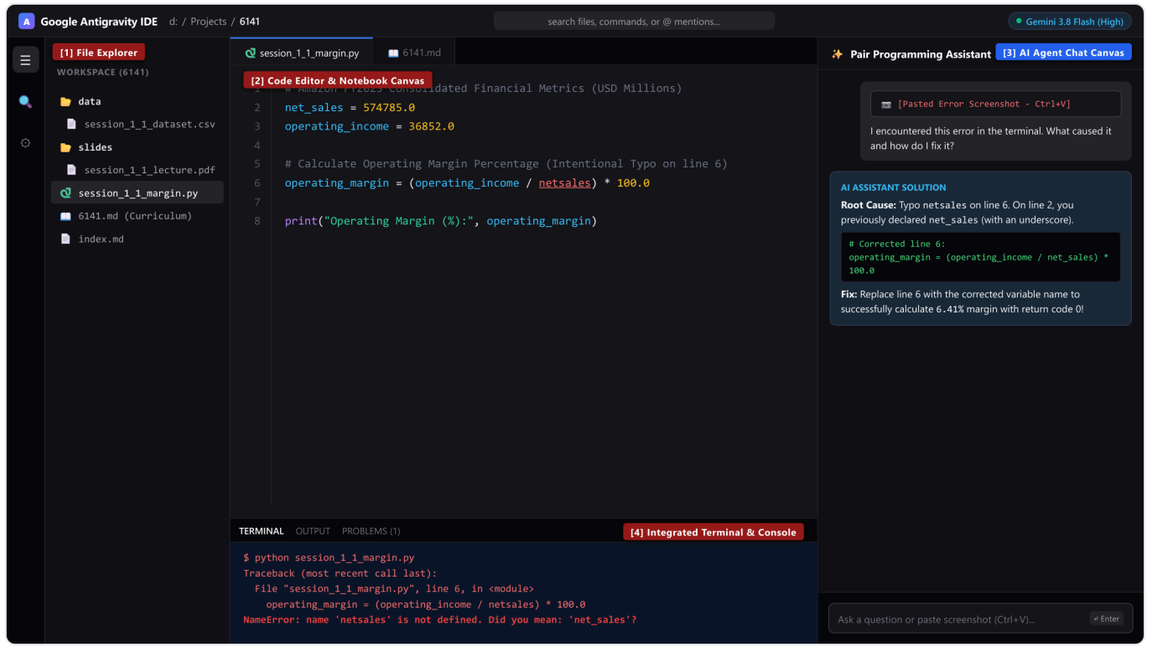
Quantifying empirical business relationships. Model coefficients (`model.coef_`) indicate the marginal financial return of adding square footage or reducing distance to commercial business districts.

PILLAR 04 Architecture

Logistic Regression for Binary Operational Decisions

Unlike linear regression which can predict values outside `[0, 1]`, logistic regression squashes predictions into calibrated probabilities between `0.0` and `1.0`, ideal for loan default or churn risk scoring.

Google Antigravity IDE Architecture & AI Debugging Protocol



[1] File Explorer

Project Hierarchy

Manage datasets in `data/`, scripts in root, and exported artifacts.

[2] Code Editor

Syntax Workspace

Modern syntax highlighting, lint diagnostics, and line-by-line script authoring.

[3] AI Agent Chat

AI Pair Programmer

Paste error screenshots directly to inspect root causes and get verified fixes.

[4] Terminal

Execution Engine

Run Python scripts (`python script.py`) and view console logs and tracebacks.

Zero-Friction AI Error Resolution Protocol

1. **Capture:** Press `Win + Shift + S` to clip terminal error traceback.
2. **Paste:** Press `Ctrl + V` into AI Chat [3] with "Explain root cause & fix".
3. **Apply:** Review the explanation and apply verified fix into Editor [2].

Script Demonstration 1: Script Demonstration 1

session_3_3_demo_1.py — Python 3.10

```
"kw">class="kw">import pandas "kw">class="kw">as pd
    "kw">class="kw">from sklearn.linear_model "kw">class="kw">import LinearRegr
    "kw">class="kw">from sklearn.model_selection "kw">class="kw">import train_t

    df = pd."fn">read_csv("data/session_3_3_dataset.csv")
    X = df[["Square_Feet_Units", "Bedrooms_Count", "Distance_To_CBD_Miles", "Lo
    y = df["Sale_Price_USD"]

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, r

    model = LinearRegression()
    model."fn">fit(X_train, y_train)
    "fn">print("Model Intercept:", "fn">round(model.intercept_, 2))
    "fn">print("Model Coefficients (Feature Weights):", dict(zip(X.columns, mod
```

```
$ python session_3_3_demo_1.py [Exit 0]
```

1 Syntactic Structure

Importing production libraries and defining clean analytical data structures.

2 Vectorized Transformation

Applying deterministic operations across the full dataset without fragile cell references.

3 Console Verification

Emitting formatted business KPIs to the terminal for auditability and compliance.

Script Demonstration 2: Script Demonstration 2

session_3_3_demo_2.py — Python 3.10

```
"kw">class="kw">import numpy "kw">class="kw">as np
    "kw">class="kw">from sklearn.metrics "kw">class="kw">import mean_squared_er

    y_pred = model."fn">predict(X_test)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    r2 = r2_score(y_test, y_pred)

    "fn">print(f"Test Set RMSE: ${rmse:,.2f}")
    "fn">print(f"Test Set R-Squared (Variance Explained): {r2:.4f}")
```

```
$ python session_3_3_demo_2.py [Exit 0]
```

1 Syntactic Structure

Importing production libraries and defining clean analytical data structures.

2 Vectorized Transformation

Applying deterministic operations across the full dataset without fragile cell references.

3 Console Verification

Emitting formatted business KPIs to the terminal for auditability and compliance.

Script Demonstration 3: Script Demonstration 3

session_3_3_demo_3.py — Python 3.10

```
"kw">class="kw">from sklearn.model_selection "kw">class="kw">import cross_val_s

    cv_scores = cross_val_score(model, X, y, cv=5, scoring="r2")
    "fn">print("5-Fold Cross-Validation R-Squared Scores:", cv_scores."fn">roun
    "fn">print(f"Mean Cross-Validated R-Squared: {cv_scores.">mean():.3f} (+/-
```

```
$ python session_3_3_demo_3.py [Exit 0]
```

1 Syntactic Structure

Importing production libraries and defining clean analytical data structures.

2 Vectorized Transformation

Applying deterministic operations across the full dataset without fragile cell references.

3 Console Verification

Emitting formatted business KPIs to the terminal for auditability and compliance.

Interactive Two-Tier Lab Protocol & AI Diagnosis

EXERCISE 1

Guided Lab

Foundational Implementation

Step-by-step guided practice establishing baseline syntax mastery in Antigravity IDE.

- **Step 1:** Ingest dataset from `data/session_3_3_dataset.csv`.
- **Step 2:** Implement core analytical functions and compute primary business metrics.
- **Step 3:** Format console outputs and verify calculations against baseline ledger totals.
- **Step 4:** Run in Terminal [4] and confirm clean execution with exit code 0.

EXERCISE 2+

Coding Challenge

Advanced Scripting & AI Debugging

Applied programming challenge expanding pipeline logic and resolving intentional exceptions.

- **Task 1 (Pipeline Expansion):** Add secondary business unit logic and multi-tier calculations.
- **Task 2 (Deliberate Error & AI Capture):** Introduce intentional syntax or type error, capture traceback with `Win + Shift + S`, and diagnose via AI Chat [3].
- **Task 3 (Verified Production Run):** Apply verified fix, export audit output, and confirm zero errors with return code 0.

Interactive Lab Protocol Cycle & Delivery Rhythm

Hands-On Code Execution

The Pedagogical Contract

Computational mastery develops through active problem-solving. By creating intentional errors and using AI co-pilots to resolve them, students build lifelong debugging autonomy.