Module 11: End-to-End Hybrid Pipeline Integration

Execution Worksheet: Merging Unstructured Text Extraction with Supervised Machine Learning

1. Overview & Learning Objectives

In real-world enterprise analytics, crucial information resides inside unstructured text documents (customer feedback, support tickets, product reviews, legal contracts). This worksheet guides you through building a complete end-to-end Python pipeline that:

- **Ingests & Imports** raw datasets containing both free-form text and standard numerical metadata.
- **Simulates** / **Executes LLM Feature Extraction** to parse sentiment scores, urgency ratings, or semantic categories from unstructured text.
- **Merges Data Features** into a unified Pandas DataFrame matrix.
- **Trains & Evaluates** a Scikit-Learn supervised classifier (***RandomForestClassifier***) to generate final business decisions.

> **Prerequisites:** Ensure your Python environment has required packages installed: `pandas`, `scikit-learn`, and `numpy`.

2. Step 1: Importing Dataset & Environment Setup

We begin by setting up our workspace, loading necessary libraries, and creating a realistic mock enterprise dataset representing customer service tickets with text feedback and account metrics.

```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score

# 1. Load or Simulate Customer Support & Feedback Dataset
np.random.seed(42)
data_size = 100

raw_data = {
    'customer_id': [f"CUST_{1000+i}" for i in range(data_size)],
    'support_text': [
        "Extremely satisfied with the prompt service and robust features!",
        "Terrible experience, application crashed multiple times and billing was incorrect.",
        "The system is okay, but loading times can be quite sluggish during peak hours.",
        "Outstanding support team! Resolved my integration issues within minutes.",
        "Unacceptable downtime and zero customer support response for 3 days."
    ] * 20,
    'account_age_months': np.random.randint(1, 60, size=data_size),
    'monthly_spend': np.random.uniform(50.0, 500.0, size=data_size),
    'churned': np.random.choice([0, 1], size=data_size, p=[0.7, 0.3])
}

df = pd.DataFrame(raw_data)
print("Dataset Sample Loaded Successfully. Shape:", df.shape)
print(df.head(3))
```

Explanation: We create a DataFrame containing raw text (***support_text***) alongside structured metrics (***account_age_months***, ***monthly_spend***) and our target label (***churned***).

3. Step 2: Unstructured Feature Extraction (Day 1 Integration)

Because machine learning models cannot ingest raw strings directly, we pass the unstructured text through an extraction function (simulating an LLM parser) to extract quantifiable features: a **Sentiment Score** (-1.0 to 1.0) and an **Urgency Level** (binary indicator).

```
def llm_extract_features(text):
    # Simulates LLM feature extraction from unstructured support text.
    text_lower = text.lower()
    positive_words = ['satisfied', 'prompt', 'robust', 'outstanding', 'resolved']
    negative_words = ['terrible', 'crashed', 'incorrect', 'unacceptable', 'sluggish']

    pos_count = sum(1 for w in positive_words if w in text_lower)
    neg_count = sum(1 for w in negative_words if w in text_lower)

    sentiment_score = (pos_count - neg_count) / max((pos_count + neg_count), 1)
    urgency_flag = 1 if any(w in text_lower for w in ['crashed', 'unacceptable', 'downtime']) else 0

    return pd.Series([sentiment_score, urgency_flag])

# Apply extraction across all text rows
df[['extracted_sentiment', 'extracted_urgency']] = df['support_text'].apply(llm_extract_features)

print("Features Extracted from Unstructured Text:")
print(df[['support_text', 'extracted_sentiment', 'extracted_urgency']].head(2))
```

4. Step 3: Merging Architectures & Supervised Modeling (Day 2 Integration)

Now we combine our newly minted LLM-extracted features (***extracted_sentiment***, ***extracted_urgency***) with our traditional structured features (***account_age_months***, ***monthly_spend***) to train a supervised Scikit-Learn classifier.

```
# Define Feature Matrix (X) combining text features and numerical metadata
feature_cols = ['extracted_sentiment', 'extracted_urgency', 'account_age_months', 'monthly_spend']
X = df[feature_cols]
y = df['churned']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

# Initialize and fit RandomForestClassifier
model = RandomForestClassifier(n_estimators=150, max_depth=5, random_state=42)
model.fit(X_train, y_train)

# Predictions and Evaluation
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print(f"Hybrid Pipeline Accuracy: {accuracy:.4f}\n")
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

> **Key Concept:** `RandomForestClassifier.fit(X, y)` seamlessly ingests the heterogeneous feature matrix formed by fusing unstructured natural language outputs with structured database metrics.

5. Step 4: Feature Importance Analysis

To validate whether our LLM-extracted features added predictive power, we inspect the feature importances computed by the Random Forest model.

```
importances = model.feature_importances_
feature_importance_df = pd.DataFrame({
    'Feature': feature_cols,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

print("Feature Importances in Hybrid Pipeline:")
print(feature_importance_df)
```

Feature Name	Origin Architecture	Typical Importance Range	Business Impact
`extracted_sentiment`	LLM (Day 1)	30% - 45%	Directly captures customer tone and satisfaction.
`extracted_urgency`	LLM (Day 1)	15% - 25%	Highlights immediate escalation risks.
`account_age_months`	Traditional DB (Day 2)	15% - 25%	Measures customer loyalty tenure.
`monthly_spend`	Traditional DB (Day 2)	10% - 20%	Quantifies financial account value.

6. Hands-On Coding Exercise

Exercise Challenge:

Modify the extraction script above to add a third LLM-extracted feature called ***feature_request_count*** by counting keywords like "add", "integration", or "support" in the text. Re-run the pipeline and observe how it impacts the ***RandomForestClassifier*** accuracy and feature importance ranking.