Universidad San Francisco de Quito
Data Mining

Modalidad del proyecto

- **Individual**
  - Trabajas con **1 activo** (acciones, índices u otros activos de inversión disponibles en la API de Yahoo Finance).
  - **No** haces la parte de **Twitter** (solo datos de mercado).
  - **Una** entrega = Proyecto 6 **+** Proyecto Final
- **Grupal (4-5 personas)**
  - Trabajan con **2 activos**.
  - Incluyen **Twitter** (ingesta + features de sentimiento - NLP de texto).
  - **Dos** entregas: Proyecto 6 **y** Proyecto Final

En ambos casos, los datos de mercado deben tener **mínimo 3 años de historia** por activo.

En ambos casos habrá presentación de los resultados de su proyecto: datos de entrenamiento, evaluación de modelos, selección del mejor modelo y simulación.

Proyecto #6

Pipeline de Datos para Trading Algorítmico con Mercado y (opcional) Twitter

---

1) Resumen

Levantarás un **Docker Compose** con:

- `jupyter-notebook`: entorno de trabajo (Python).

- `postgres`: base de datos relacional.
- `feature-builder`: servicio “worker” que corre tu script CLI para construir la tabla de features diarias.

Tú:

- Escogerás un **portafolio de activos** (tickers) desde la API de Yahoo Finance:
  - Acciones, índices u otros activos de inversión.
  - **Individual**: 3 activos.
  - **Grupal**: 5 activos.
- Ingerirás a Postgres (esquema `raw`) datos de:
  - **Precios diarios (OHLCV)** de cada activo (mínimo 3 años de historia).
  - **Tweets relacionados** (solo grupal) con esos activos (cashtags, nombre, @cuenta, hashtags).
- Ejecutarás el servicio `feature-builder` que correrá tu script CLI para crear/actualizar una tabla tipo “One Big Table”:
  - `analytics.daily_features` (1 fila = 1 día bursátil por activo).

Esta tabla será el **insumo principal** para el Proyecto Final.

---

2) Objetivos de aprendizaje

- Diseñar un **pipeline reproducible** de ingesta de datos de mercado (y Twitter si aplica).
- Trabajar con esquemas `raw` y `analytics` en Postgres.
- Construir una tabla de **features diarias lista para ML**.
- Usar **variables de ambiente** y un **script CLI** ejecutable desde Docker (`feature-builder`).
- Documentar cómo reconstruir el pipeline con **un solo comando**.

---

3) Alcance y restricciones

Fuente de datos

- **Mercado** (obligatorio):
  - Precios diarios OHLCV de **3 (individual)** o **5 (grupal)** activos.
  - Activos permitidos: **acciones, índices u otros activos de inversión** que se puedan extraer desde la API de **Yahoo Finance**.
  - Mínimo **3 años de historia** por activo.
- **Twitter** (solo grupal):
  - Tweets que mencionen cashtags, nombre de empresa, @cuenta, hashtags relacionados a cada activo.

Destino: Postgres

Esquemas/tablas mínimas:

- `raw`
  - `raw.prices_daily`
  - `raw.tweets` (solo grupal)
  - `raw.company_info` (opcional)
- `analytics`
  - `analytics.daily_features`
    - 1 fila = 1 día bursátil por activo.

Procesamiento

- **Ingesta**: notebooks en `jupyter-notebook` + scripts Python.
- **Construcción de features**: servicio `feature-builder` que corre un script CLI (ej: `build_features.py`) leyendo `raw.*` y escribiendo `analytics.daily_features`.

Restricciones

- Todo acceso a Postgres y parámetros (tickers, fechas, etc.) via **variables de ambiente**.
- Prohibido hardcodear credenciales o parámetros clave.
- Proyecto 06 = **solo data engineering** (no entrenar modelos aún).

---

4) Arquitectura esperada (alto nivel)

- `jupyter-notebook`
  - Notebooks de ingesta de precios (y tweets, si aplica) hacia `raw.*`.
- `postgres`
  - Guarda `raw.*` y `analytics.daily_features`.
- `feature-builder`
  - Contenedor worker que corre tu script CLI para crear/actualizar `analytics.daily_features`.
- `pgadmin` (opcional)
  - UI web para inspeccionar tablas.

---

5) Seguridad y variables de ambiente (obligatorio)

Entregar:

- `.env.example` (sin secretos, solo nombres de variables).
- Usar `.env` local (no subirlo al repo).

Variables mínimas:

Postgres

- `PG_HOST`
- `PG_PORT`
- `PG_DB`
- `PG_USER`
- `PG_PASSWORD`
- `PG_SCHEMA_RAW=raw`
- `PG_SCHEMA_ANALYTICS=analytics`

Ingesta mercado

- `TICKERS` (lista: ej. `AAPL,MSFT,SPY`)
- `START_DATE`, `END_DATE`
- `DATA_PROVIDER` (ej: `yfinance`)
- `RUN_ID`

Ingesta Twitter (solo grupal)

- `TWEET_QUERY` / cashtags por ticker
- `TWEET_LANG` (ej: `en`, `es`)
- Rango de fechas para tweets

feature-builder

- Parámetros para modo de ejecución (rango de fechas, ticker, run_id, etc.).

---

6) Dataset y tabla de features (mínimos)

`raw.prices_daily`

- `date`
- `ticker`
- `open, high, low, close, adj_close, volume`

- Metadatos: `run_id`, `ingested_at_utc`, `source_name`.

`raw.tweets` (solo grupal)

- `tweet_id`
- `created_at`
- `ticker` o `symbol_detected`
- `text`
- `author_id` (opcional)
- Metadatos: `run_id`, `ingested_at_utc`, `source`, `lang`.

`analytics.daily_features`

Identificación de día

- `date`, `ticker`, `year`, `month`, `day_of_week`

Mercado (agregado diario)

- `open`, `close`, `high`, `low`, `volume`
- `return_close_open = (close - open) / open`
- `return_prev_close = close / close_lag1 - 1`
- `volatility_n_days` (std de retornos últimos N días)

Actividad Twitter (agregada al día, solo grupal)

- `tweet_count`
- `sentiment_avg`, `sentiment_std`
- `sentiment_pos_share`, `sentiment_neg_share`

Derivadas simples

- `tweet_count_lag1`, `sentiment_avg_lag1`
- Flags: `is_monday`, `is_friday`, `is_earnings_day` (opcional)

Metadatos

- `run_id`
- `ingested_at_utc`

> La definición de sentimiento se detalla en el Proyecto Final, pero las columnas deben quedar listas aquí.

---

7) ¿Qué es la tabla de features diarias?

Tu “One Big Table” por día y por activo:

- Mercado + (Twitter si aplica) + derivadas + metadatos en una sola tabla.

Ventajas:

- Lista para usar en notebooks de ML.
- Menos JOINs y menos errores.

Costo:

- Algo de duplicación de datos, pero pipeline más claro.

---

8) Infra con Docker Compose y servicio feature-builder

8.1 Servicios

- `jupyter-notebook`: puerto expuesto, volumen de trabajo.
- `postgres`: BD con esquemas `raw` y `analytics`.
- `feature-builder`: ejecuta tu script CLI para `analytics.daily_features`.

Entrada:

- Credenciales y parámetros por variables de ambiente.
- Args de CLI: `ticker`, rango de fechas, `run_id`, modo, etc.

Salida:

- `analytics.daily_features` creada/actualizada.
- Logs con conteos, fechas mín/máx, tiempos.

Requisito:

docker compose run feature-builder --mode full --ticker AAPL

Debe construir `analytics.daily_features` end-to-end desde `raw.*`.

8.2 Especificación CLI (build_features.py, sin código)

Argumentos mínimos:

- `--mode {full, by-date-range}`
- `--ticker <string>`
- `--start-date YYYY-MM-DD`
- `--end-date YYYY-MM-DD`
- `--run-id <string>`
- `--overwrite {true,false}`

Comportamiento:

- `full`: (re)crea la tabla completa para el ticker/rango.
- `by-date-range`: solo procesa un subconjunto de fechas.
- Idempotente: sin duplicar filas (sobrescribir por `(ticker, date)` o borrar/reinsertar).

Logs:

- Filas creadas/actualizadas.
- Fecha mín y máx procesada.
- Duración.

---

9) Notebooks obligatorios

9.1 Ingesta precios → `raw.prices_daily`

Archivo: `01_ingesta_prices_raw.ipynb`

- Leer parámetros desde env (`TICKERS`, `START_DATE`, `END_DATE`).
- Descargar precios diarios (Yahoo Finance).
- Estandarizar tipos y nombres.
- Cargar a `raw.prices_daily` con `run_id` y `ingested_at_utc`.
- Mostrar conteos y fechas mín/máx.

9.2 Ingesta Twitter → `raw.tweets` (solo grupal)

Archivo: `02_ingesta_tweets_raw.ipynb`

- Leer parámetros (query, rango fechas, idioma).
- Extraer tweets por ticker.
- Limpiar campos mínimos.

- Cargar a `raw.tweets` con metadatos.
- Log de número de tweets por día.

---

10) Métricas y evaluación (Proyecto 06)

- Cobertura de fechas entre `raw.prices_daily` y `analytics.daily_features`.
- Días bursátiles sin datos (deben ser pocos y explicados).
- Twitter (grupal): tweets promedio por día, días sin señal.
- Idempotencia del `feature-builder`.

---

11) Entregables (GitHub, Proyecto 06)

- `docker-compose.yml` con `jupyter-notebook`, `postgres`, `feature-builder`.
- Script CLI `build_features.py`.
- Notebooks:
  - `01_ingesta_prices_raw.ipynb`
  - `02_ingesta_tweets_raw.ipynb` (solo grupal)
- `.env.example`
- `README.md` con:
  - Cómo levantar Compose.
  - Comandos de ingesta.
  - Comando para construir `analytics.daily_features`.
  - Breve explicación de las columnas principales.

---

12) Rúbrica (Proyecto 06, 100 pts)

- **A. Pipeline RAW en Postgres (30 pts)**
  - 15: `raw.prices_daily` completo y consistente.
  - 15: `raw.tweets` (si aplica) con metadatos y conteos razonables.
- **B. Tabla `analytics.daily_features` (30 pts)**
  - 15: estructura correcta.
  - 10: features agregadas y derivadas coherentes.
  - 5: metadatos completos.
- **C. feature-builder y CLI (20 pts)**
  - 10: `build_features.py` con `--mode full` y `--mode by-date-range`.

    - 5: idempotencia.
    - 5: logs claros.
- **D. Reproducibilidad & documentación (20 pts)**
    - 10: `.env.example`, seeds, comandos claros.
    - 10: README completo y probado.

---

13) Checklist de aceptación (Proyecto 06)

- `raw.prices_daily` poblado para todos los activos.
- `raw.tweets` poblado (si es grupal).
- `analytics.daily_features` creada por `feature-builder`.
- CLI ejecutable con los argumentos definidos.
- README explica cómo reproducir todo.

Proyecto Final

Modelo de Clasificación, Simulación de Inversión y API

(Usa como insumo `analytics.daily_features` del Proyecto 06.)

---

1) Resumen

Trabajarás con `analytics.daily_features` (1 fila = 1 día bursátil por activo) para:

1. Definir una variable objetivo binaria:
    - `target_up = 1` si `close > open`
    - `target_up = 0` en caso contrario.
2. Diseñar un pipeline de ML que use:
    - Features de mercado (obligatorio).
    - Features de Twitter (si es grupal).
3. Entrenar, **tunear** y comparar **al menos 7 modelos de clasificación** de distintos tipos:
    - Lineales, árbol, bosque, boosting.
4. Seleccionar el mejor modelo (según validación), evaluarlo en test y analizar errores.

5. Con el **mejor modelo tuneado**, **simular una inversión de USD 10,000 durante 2025** y comparar:
    - Retorno de la estrategia vs.
    - Métricas de validación y test.
6. Desplegar el modelo ganador como una API REST (Flask o FastAPI) empaquetada en Docker.

---

2) Objetivos de aprendizaje

- Formular un problema de clasificación binaria en trading.
- Construir un pipeline reproducible:
    - carga datos → split temporal → preprocesamiento → modelado → evaluación.
- Aplicar **GridSearch** (o RandomizedSearch) para tuning de hiperparámetros.
- Integrar el modelo ganador en una **API REST**.
- Conectar métricas de ML con resultados de una **simulación de inversión realista**.

---

3) Alcance y restricciones

Fuente obligatoria

- `analytics.daily_features` construido en Proyecto 06 (no se puede saltar el pipeline).

Problema a resolver

- Clasificar si, para un día dado, el activo **cerrará arriba o abajo** respecto al precio de apertura.

Restricciones importantes

Sin leakage:

- Solo usar información disponible **antes** o al momento de la apertura del día que quieres predecir.
- Puedes usar:
    - Lags de precios y retornos.
    - Lags de sentimiento y tweets agregados hasta el día anterior.
- No puedes usar `close` del día que predices para ese mismo día.

Split temporal (no aleatorio):

- Train: años más antiguos.
- Validación: período intermedio.
- Test: años más recientes.
- Datos mínimos: **≥3 años para Train+Val** y **un año reciente (ej. 2025)** para Test/Simulación.

---

4) Arquitectura esperada

- `jupyter-notebook`: notebook de ML (`ml_trading_classifier.ipynb`).
- `postgres`: provee `analytics.daily_features` (o CSV exportado).
- `model-api`: servicio Docker con la API REST del modelo ganador.

---

5) Seguridad y variables de ambiente (API)

- `.env.example` (Postgres + API).
- `.env` local (no subir).

Variables mínimas:

Base de datos

- Reutilizar `PG_HOST`, `PG_PORT`, `PG_DB`, etc.

Modelo y API

- `MODEL_PATH` (archivo `.pkl` o `.joblib`).
- `API_PORT`
- `API_ENV` (dev, prod, etc.)

---

6) Dataset y target (recordatorio)

Features numéricas (ejemplos):

- Mercado: `open`, `high`, `low`, `close_prev` (lag), `volume`, `return_prev_close`, `volatility_n_days`.

- Twitter (solo grupal): `tweet_count`, `sentiment_avg`, `sentiment_pos_share`, `sentiment_neg_share`.

- Derivadas: lags de retornos, lags de sentimiento, rolling features, etc.

Features categóricas (opcionales):

- `day_of_week`
- Flags: `is_monday`, `is_friday`, `is_earnings_day`.

Target:

- `target_up = 1` si `close > open`, 0 si no.

---

7) Problema de clasificación (síntesis)

- No predices el precio exacto.
- Solo la **dirección diaria** (sube o no).
- El modelo sirve como insumo para estrategias de trading (no se evalúa diseño de estrategia compleja, solo la simulación pedida).

---

8) Infra con Docker Compose y servicio model-api

8.1 Servicios requeridos

- `jupyter-notebook`
- `postgres`
- `model-api`:
    - Carga el modelo entrenado desde `MODEL_PATH`.
    - Expone endpoint REST `/predict`.

---

9) Notebook obligatorio de ML

Archivo: `ml_trading_classifier.ipynb`

Estructura mínima:

1. **Definición del problema**
    - Explicar `target_up` y la decisión que habilita.
2. **Carga de datos**
    - Leer desde Postgres o CSV exportado.
3. **EDA breve**
    - Balance de clases (días up/down).
    - Distribución de retornos.
    - Correlaciones simples (incluyendo sentimiento si aplica).
4. **Features y target**
    - Seleccionar columnas.
    - Justificar lags y ventanas.
5. **Split temporal (Train / Val / Test)**
    - Explicar fechas elegidas y años para 2025 como test.
6. **Preprocesamiento**
    - Imputación de nulos.
    - Escalado (StandardScaler/MinMax) para modelos sensibles.
    - One-hot encoding si usas categóricas.
    - Guardar pipeline de preprocesamiento.
7. **Modelado – mínimo 7 modelos distintos**

    Debes incluir **al menos 7 modelos**, cubriendo:
    - **Lineales (≥1)**
        - Ej: `LogisticRegression`, `LinearSVC`.
    - **Árbol de decisión (≥1)**
    - **Random Forest (≥1)**
    - **Boosting (≥2)**
        - Ej: `GradientBoostingClassifier`, `XGBoost`, `LightGBM`, `CatBoost`.

8. Para cada modelo:
    - Definir hiperparámetros a tunear.
    - Usar **GridSearchCV** o **RandomizedSearchCV** (puedes usar TimeSeriesSplit o validación simple temporal).
    - Registrar métricas en **Train y Validación**.
9. **Evaluación**
    - Métricas: Accuracy, Precision, Recall, F1, ROC-AUC.
    - Matriz de confusión.
    - Comparación tabular entre los 7 modelos.
10. **Selección del modelo ganador**
    - Criterio: por ejemplo, mayor F1 en validación + simplicidad.
    - Reentrenar en **Train+Val**.
    - Evaluar en **Test** (años más recientes, incluyendo antes de 2025).
11. **Análisis de errores**
    - ¿En qué situaciones falla más?
    - Días con alta volatilidad, earnings, etc.

12. **Exportar modelo**
    - Guardar pipeline completo (preprocesamiento + modelo) en `MODEL_PATH`.

---

10) Métricas, modelos y baseline

Modelos (obligatorio):

- ≥ **7 modelos** distintos, cubriendo:
  - lineal, árbol, bosque, boosting, y otros.

Métricas primarias:

- F1 (macro o para clase positiva).
- ROC-AUC.

Métricas secundarias:

- Accuracy.
- Precision, Recall.

Baseline:

- Modelo trivial: siempre predice la clase mayoritaria.
- Tu mejor modelo debe mejorar claramente al baseline.

---

11) Simulación de inversión con USD 10,000 en 2025

Con el **mejor modelo tuneado y elegido**:

1. **Período de simulación:**
   - Año **2025** (o el último año disponible, claro en el notebook).
   - Los datos de 2025 deben ser **solo Test** (nunca usados para entrenar ni validar).
2. **Regla simple de trading (sugerida):**
   Para cada día bursátil de 2025, por cada activo:
   - Si `prediction` (o `prob_up ≥ threshold`) = 1:
     - Considera que entras **largo** en el activo ese día (por ejemplo, comprar al open y vender al close).
   - Si `prediction = 0`:
     - Te quedas en efectivo ese día.

3. Puedes:
    - Simular por un solo activo principal **o**
    - Hacer un portafolio **equally-weighted** entre tus activos.
4. Explica claramente tu regla.
5. **Capital inicial:**
    - `capital_inicial = USD 10,000`.
6. **Suposiciones:**
    - Sin costos de transacción ni fricciones (a menos que quieras modelarlos).
    - No se permite apalancamiento (capital nunca negativo).
7. **Salidas de la simulación:**
    - Valor final del portafolio al cierre del período.
    - **Retorno total (%)**.
    - **Retorno anualizado** (si aplica).
    - Número de trades ejecutados.
    - Curva de equity (capital vs. tiempo).
8. **Comparación con métricas de ML:**
    - Relacionar:
        - F1 / ROC-AUC en Validación y Test.
        - VS retorno y drawdowns de la estrategia en 2025.
    - Comentar:
        - ¿Un buen F1 significa necesariamente buen retorno?
        - ¿En qué casos no?

---

12) Entregables (GitHub, Proyecto Final)

- Notebook: `ml_trading_classifier.ipynb`.
- Código de la API:
    - `app.py` (Flask o FastAPI).
    - `Dockerfile` para `model-api`.
- `docker-compose.yml` actualizado con `model-api`.
- Archivo de modelo entrenado (`.pkl` / `.joblib`) o instrucciones claras para regenerarlo.
- `README.md` con:
    - Cómo entrenar el modelo.
    - Cómo guardar el modelo.
    - Cómo levantar la API y probar `/predict` (ejemplo con curl/Postman).
    - Cómo correr la **simulación de inversión de USD 10,000 en 2025**.

---

13) Rúbrica (Proyecto Final, 100 pts)

- **A. Pipeline de ML (25 pts)**
  - 10: EDA y definición clara del problema.
  - 10: Features bien justificadas, sin leakage.
  - 5: Split temporal correcto.
- **B. Modelos y tuning (30 pts)**
  - 15: ≥ **7 modelos** distintos correctamente entrenados (lineal, árbol, bosque, boosting, otros).
  - 15: Tuning razonable (grids claras, comparación justa, uso de GridSearch/RandomizedSearch).
- **C. Evaluación, selección y simulación (25 pts)**
  - 10: Métricas completas en Train/Val/Test + baseline.
  - 10: Selección bien justificada del modelo ganador.
  - 5: Simulación de inversión con USD 10,000 en 2025, análisis y comparación con métricas de validación.
- **D. API REST (15 pts)**
  - 10: `model-api` funcional con `/predict`
  - 5: Respuestas JSON claras y manejo básico de errores.
- **E. Reproducibilidad & documentación (10 pts)**
  - 5: `.env.example`, rutas, seeds.
  - 5: README con ejemplos de uso (incluyendo simulación).

---

14) Checklist de aceptación (Proyecto Final)

- Se usa `analytics.daily_features` como base.
- `target_up` definido sin leakage.
- ≥ 7 modelos entrenados, tuneados y comparados.
- Baseline implementado.
- Modelo ganador reentrenado en Train+Val y evaluado en Test.
- Simulación de inversión con USD 10,000 en 2025 realizada y documentada.
- Modelo serializado y cargado por la API.
- `model-api` responde correctamente a `/predict`.