# 02-credit-risk-model-feature-registration

September 22, 2025

## 1 Feature Registartion

The goal is the **Features in Feature Registry**.

### 1.1 This Notebook (02-feature-registration)

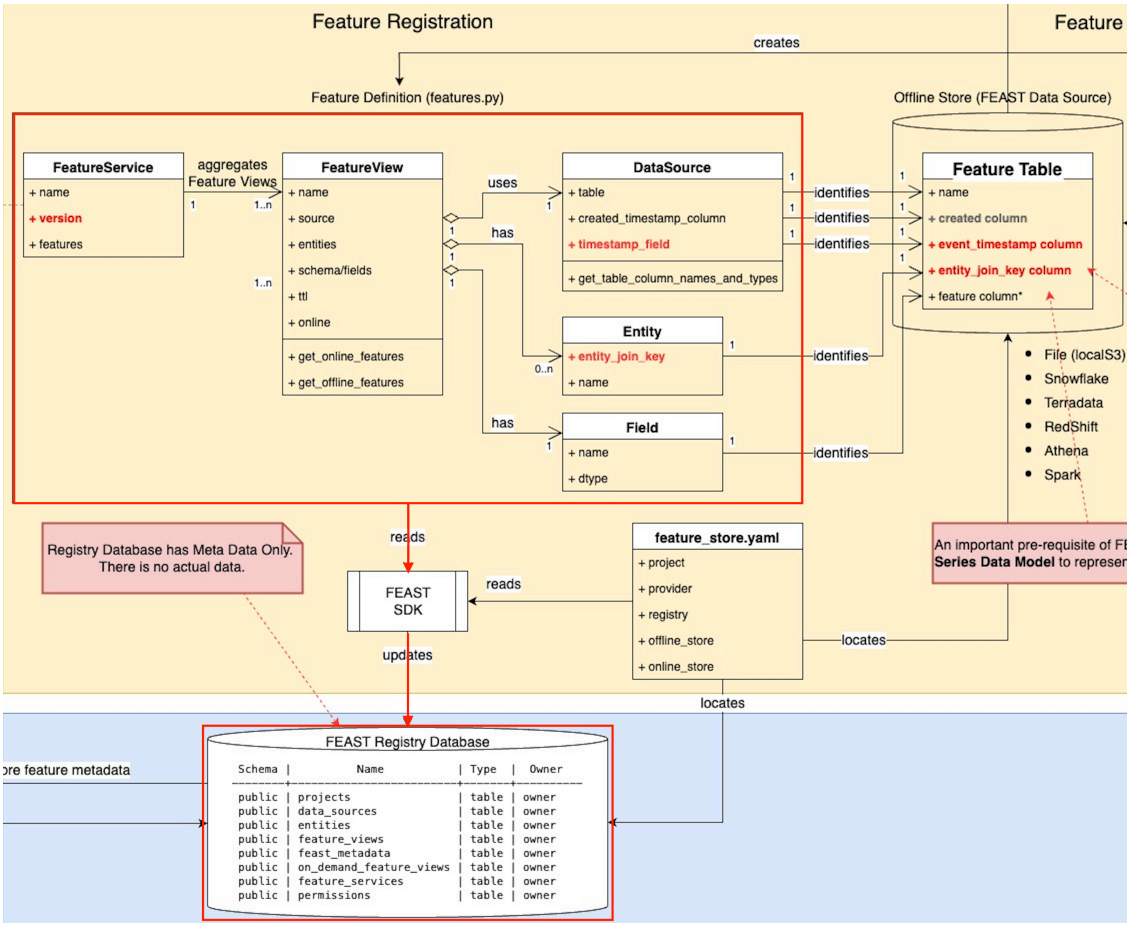
This notebook goes through the feature registration - **registering the metadata of the features** into the **FEAST Feature Registry Database (Feature Registry)**.

#### 1.1.1 Steps

2. Create a **Feature Store Project** (FEAST Project) via `feature_store.yaml` (or equivalent `RepoConfig`) that defines **Where the data is and How to access**.
3. Define the **Metadata of Features** (schema and data types) in `features.py` file in the FEAST project.
4. Register the **Metadata of Features** into the **Feature Registry Database** using FEAST SDK.

#### 1.1.2 Key Points

1. **Only the metadata** is in the Registry Database. It has **no actual features**.
2. **Offline Store is the source of truth**. It has the actual feature data.
3. **No-validation by FEAST** if the definitions in features.py are correct (consistent with what actually exist in the Online Store). You are responsible to make sure.




## 2 Setup

```
<IPython.core.display.HTML object>
```

---

## 3 FEAST Project

Each data science project creates a FEAST project to manage its ML Features via `feature_store.yaml` (or equivalent `RepoConfig`). A FEAST project creates a namespace that is a unit of isolation where resources of the project are managed. When there are multiple data science projects, each project is segregated by the namespace.

- FEAST Project

  Projects provide complete isolation of feature stores at the infrastructure level. This is accomplished through **resource namespacing, e.g., prefixing table names with the associated project**. Each project should be considered a completely separate universe of entities and features.

The FEAST project encompases the objects which we explain later. * Feature View * Entity * Feature Field * Data Source

## 3.1 feature_store.yaml

`feature_store.yaml` or RepoConfig class is the **configuration of a FEAST project** that defines a project, where the data is, and how to access them.

- FEAST Document - Registry

  Users can specify the registry through a feature_store.yaml config file, or programmatically.

  Option 1: programmatically specifying the registry “‘ repo_config = RepoConfig( registry=RegistryConfig(path=“gs://feast-test-gcs-bucket/registry.pb”), project=“feast_demo_gcp”, provider=“gcp”, offline_store=“file”, # Could also be the OfflineStoreConfig e.g. FileOfflineStoreConfig online_store=“null”, # Could also be the OnlineStoreConfig e.g. RedisOnlineStoreConfig )

  Option 2: specifying the registry in the project’s feature_store.yaml file

The following are the top-level configuration items.

| Item | Description | Value |
|---|---|---|
| project | a namespace for the entire feature store. | YOUR_PROJECT_NAME |
| provider | provider is an implementation of a feature store, like Terraform provider. | local or aws or gcp |
| registry | Where the Feature Registry is | e.g. `path: s3://feast-bucket/registry.pb` |
| offline_store | Where the Feature is | e.g. `type: redshift` |
| online_store | Where the low latency feature server. | e.g. `type: dynamodb` |

### 3.1.1 Example

```
project: customer_credit_risk
provider: local
registry:
    registry_type: sql
    path: postgresql+psycopg2://${PG_ADMIN_USER}@${PG_FEAST_HOST}:${PG_FEAST_PORT}/${PG_FEAST_

offline_store:
  type: postgres
  host: ${PG_OFFLINE_HOST}
  port: ${PG_OFFLINE_PORT}
  database: ${PG_OFFLINE_DB}
  db_schema: ${PG_OFFLINE_SCHEMA}
  user: ${PG_OFFLINE_USER}
  password: ${POSTGRES_PASSWORD}

online_store:
```

```
  type: sqlite
  path: data/online_store.db
```

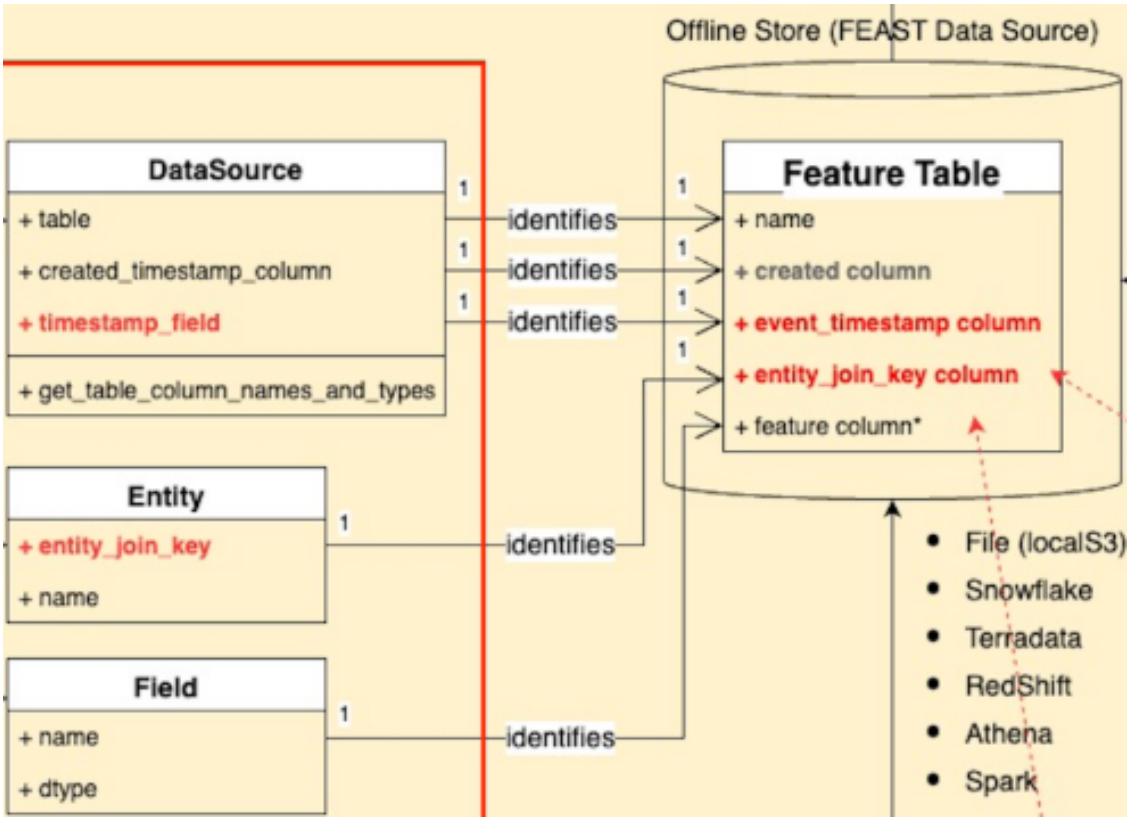
### 3.2 Create a FEAST Project

We create a project configuration using `RepoConfig` (we can use feature_store.yaml as well)

```
# %%bash -s "$FEATURE_REPO_DIR"
# (cd $1 && feast apply)
```

---

## 4 Feature Definitions (features.py)




### 4.1 Datasource

Datasource is an interface to the Offline Store to load the Features. The `offline_store` configuration parameters defines the physical Offline Store instance, and the `DataSource` in `features.py` automatically ties the DataSource to the physical Offline Store.

```
offline_store:
  type: postgres
  host: ${PG_OFFLINE_HOST}
  port: ${PG_OFFLINE_PORT}
  database: ${PG_OFFLINE_DB}
```

```
db_schema: ${PG_OFFLINE_SCHEMA}
user: ${PG_OFFLINE_USER}
password: ${POSTGRES_PASSWORD}
```

- Datasource - PostgreSQL

  PostgreSQL data sources are PostgreSQL tables or views. These can be specified either by a table reference or a SQL query.

## 4.2 Entity

Entity a column or field that identifies a record e.g. customer or product. STAR Schema equivalent would be Dimension.

Suppose there is a class `Customer`.

```
class Customer:
    customer_id: int
    name: str
    age: int
    gender: string
```

Each instance of the class is an entity (or a record) in a database table and the `customer_id` field is the key of the entity.

### 4.2.1 Note

`join_keys` currently only support one column. FEAST can use the `join_key` to do join operations, but not get into details here.

```
/var/folders/_y/676ck7wn74q07wgfpd7v2sxh0000gp/T/ipykernel_27910/4111618432.py:1
: DeprecationWarning: Entity value_type will be mandatory in the next release.
Please specify a value_type for entity 'customer'.
  customer = Entity(
```

## 4.3 FeatureView

FeatureView and Offline Store table has one to one relation. It defines which columns in the Offline Store table to use as the Features.

- Feature view

  In the offline setting, Feature View is a stateless collection of features that are created when the get_historical_features method is called.

## 4.4 FeatureService

FeatureService is a mechanism to aggregate multiple FeatureViews. Suppose `FeatureView S` refers to a Snowflake table and `FeatureView T` refers to a Terradata table. We can use combine features from `S` and `T` as one group of features.

Also, FeatureService can be the unit of versioning. One version of a FeatureService will corresponds with a Model version.

## 5 Feature Registration

Once the FEAST Project has been setup, we regisers the Features into the **FEAST Feature Registry Database**. The Registry Database only manages the **Metadata** of the Features, and there is no actual data there.

Using the apply method or CLI apply command, the Features get registred to the Registry Database.

> NOTE: FEAST `apply` **does not verify** if there are corresponding columns in the offline store for the Fields defined in the FeatureView.

### 5.1 Verify the Registration

Once the registraiton is done, verify the FEAST objects registered.

```
{
  "spec": {
    "name": "customer_credit_risk",
    "description": "A project for customer credit risk"
  },
  "meta": {
    "createdTimestamp": "2025-09-18T02:27:25.053090Z",
    "lastUpdatedTimestamp": "2025-09-18T02:27:25.053090Z"
  }
}
FeatureView: customer_credit_risk_feature_view
  Feature: risk (Float32)
  Feature: purpose_business (Float32)
  Feature: purpose_car (Float32)
  Feature: purpose_domestic_appliances (Float32)
  Feature: purpose_education (Float32)
  Feature: purpose_furniture_equipment (Float32)
  Feature: purpose_radio_tv (Float32)
  Feature: purpose_repairs (Float32)
  Feature: purpose_vacation_others (Float32)
  Feature: gender_female (Float32)
  Feature: gender_male (Float32)
  Feature: housing_free (Float32)
  Feature: housing_own (Float32)
  Feature: housing_rent (Float32)
  Feature: saving_accounts_little (Float32)
  Feature: saving_accounts_moderate (Float32)
  Feature: saving_accounts_no_inf (Float32)
  Feature: saving_accounts_quite_rich (Float32)
  Feature: saving_accounts_rich (Float32)
  Feature: checking_account_little (Float32)
```

```
  Feature: checking_account_moderate (Float32)
  Feature: checking_account_no_inf (Float32)
  Feature: checking_account_rich (Float32)
  Feature: generation_student (Float32)
  Feature: generation_young (Float32)
  Feature: generation_adult (Float32)
  Feature: generation_senior (Float32)
  Feature: job_0 (Float32)
  Feature: job_1 (Float32)
  Feature: job_2 (Float32)
  Feature: job_3 (Float32)
  Feature: amount_0 (Float32)
  Feature: amount_1 (Float32)
  Feature: amount_2 (Float32)
  Feature: amount_3 (Float32)
  Feature: amount_4 (Float32)

FeatureService: customer_credit_risk_feature_service
  FeatureProjection: customer_credit_risk_feature_view
    Feature: risk (Float32)
    Feature: purpose_business (Float32)
    Feature: purpose_car (Float32)
    Feature: purpose_domestic_appliances (Float32)
    Feature: purpose_education (Float32)
    Feature: purpose_furniture_equipment (Float32)
    Feature: purpose_radio_tv (Float32)
    Feature: purpose_repairs (Float32)
    Feature: purpose_vacation_others (Float32)
    Feature: gender_female (Float32)
    Feature: gender_male (Float32)
    Feature: housing_free (Float32)
    Feature: housing_own (Float32)
    Feature: housing_rent (Float32)
    Feature: saving_accounts_little (Float32)
    Feature: saving_accounts_moderate (Float32)
    Feature: saving_accounts_no_inf (Float32)
    Feature: saving_accounts_quite_rich (Float32)
    Feature: saving_accounts_rich (Float32)
    Feature: checking_account_little (Float32)
    Feature: checking_account_moderate (Float32)
    Feature: checking_account_no_inf (Float32)
    Feature: checking_account_rich (Float32)
    Feature: generation_student (Float32)
    Feature: generation_young (Float32)
    Feature: generation_adult (Float32)
    Feature: generation_senior (Float32)
    Feature: job_0 (Float32)
    Feature: job_1 (Float32)
```

```
Feature: job_2 (Float32)
Feature: job_3 (Float32)
Feature: amount_0 (Float32)
Feature: amount_1 (Float32)
Feature: amount_2 (Float32)
Feature: amount_3 (Float32)
Feature: amount_4 (Float32)
```