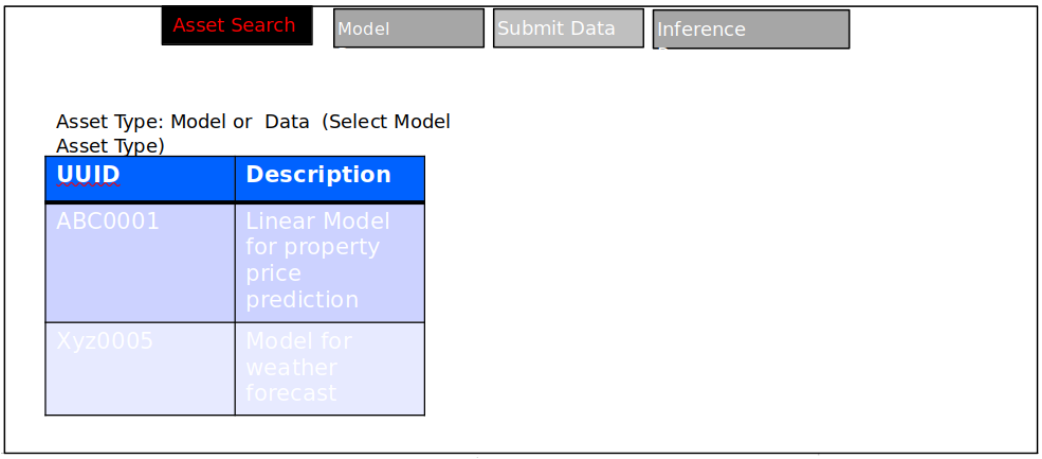
1 Asset Search: AI Assistant



The above screen allows `AI Assistant` to search for *model* or *data* assets, although here we consider only model asset. The user specifies that she wants to view model assets, and the table displays all assets of type `Linear-Model` from the blockchain. The box below contains file location for a sample asset, with the description of relevant keys.

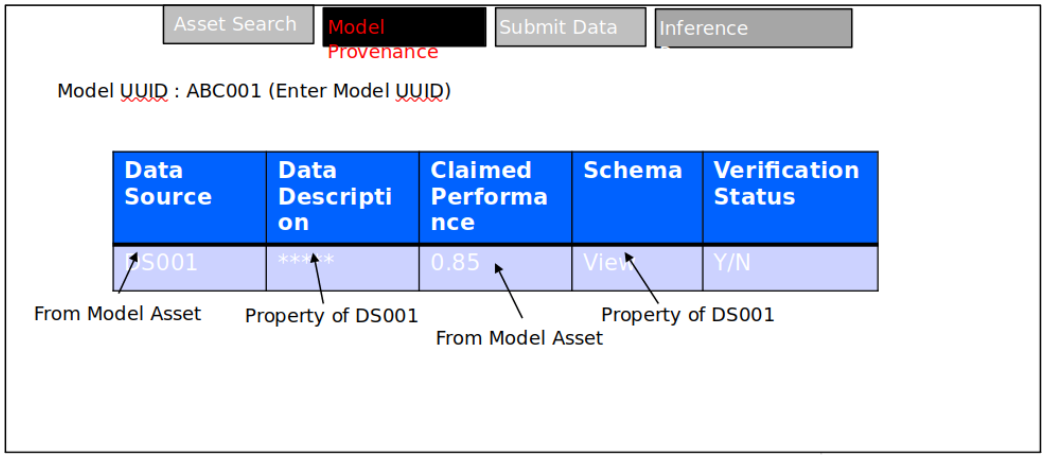
```
assets/housing_model_asset.json
.assetUUID :- key containing asset UUID.
.lineageInfo.transformationInfo.description :- key containing asset
   description.
```

1.1 Action

The following actions need to be performed to generate the page:

```
1. Query all assets with assetType == Linear_Model.
2. Display values of above two keys for each such asset.
```

2 Model Provenance: AI Assistant



The above screen allows an `AI Assistant` to verify provenance of one of the model assets from previous screen, by entering the UUID of the model asset. The table contents are populated from two kinds of assets:

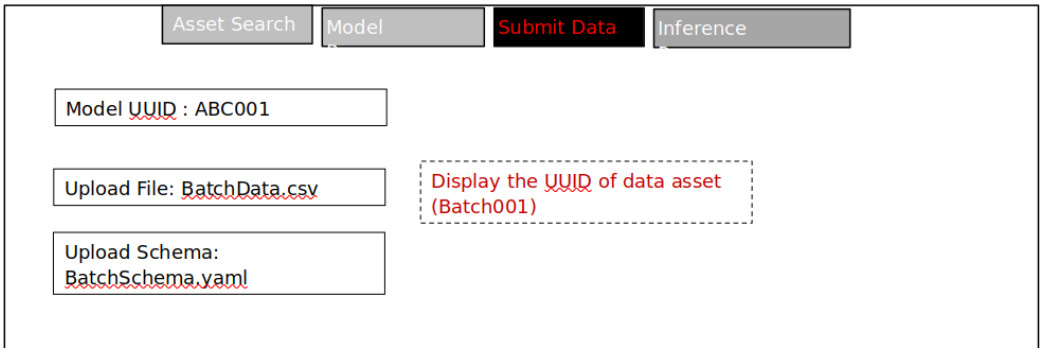
```
1. Linear-Model asset (assets/housing_model_asset.json).
    Relevant Keys:
    .lineageInfo.sourceAssets[0] :- contains UUID of associated data
   asset.
    .lineageInfo.transformationInfo.MetricR2 :- performance
2. Data asset (assets/housing_data_asset.json).
    Relevant Keys:
    ..lineageInfo.transformationInfo.description :- Data description
    .plainTextContext.assetSchema :- Data schema
```

2.1 Action

When user specifies the `modelUUID` of the model asset, following actions need to be performed:

1. Fetch asset LM, by querying asset with assetUUID=modelUUID. The asset LM should be of type Linear-Model.
2. Read the UUID of associated data asset (dataUUID) by reading the key lineageInfo.sourceAssets[0] of LM.
3. Fetch the data asset D by querying asset with assetUUID=dataUUID.
4. Data source and Performance columns are populated using asset LM using the keys in above box.
5. Description and Schema columns are populated using asset D.
6. Verify provenance of the linear model by using the script
   --- ./verify_model_asset.sh <path to fetched model asset>

3 Submit Data: AI Assitant



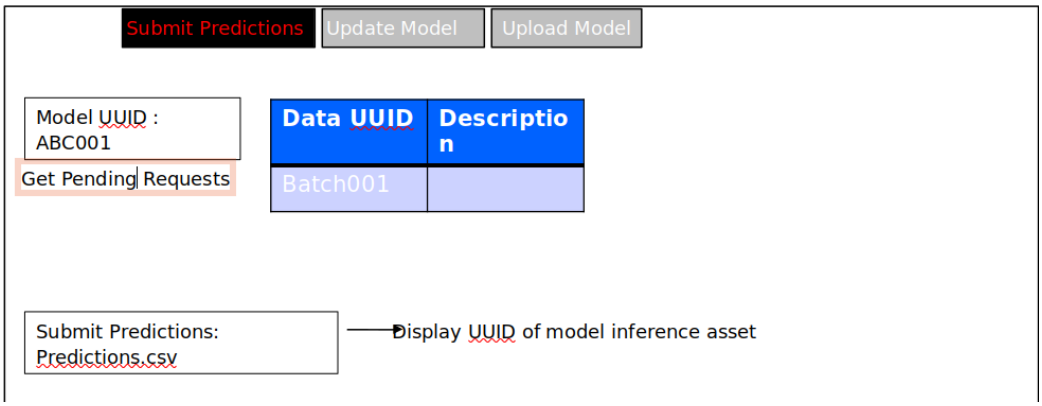
This screen allows an `AI Assitant` to get predictions on its data, based on the model of its choice.

3.1 Action

This page involves creating an asset of type data (public) on blockchain.

```
1. Let modelUUID be input by user in the text input box.
2. Upload a file containing the batch of records.
   -- use the file data/TestData.csv in the above.
3. Upload a file containing schema of the batch of records.
   -- use the file data/test_data_schema.yaml
4. To submit the asset to blockchain use the script (present in
asset-management-scripts):
  ./submit_batch_asset.sh ../data/TestData.csv
    ../data/test_data_schema.yaml modelUUID
    ../assets/batch_data_asset.json "Batch data for scoring"
5. Display the UUID by reading assetUUID key of the received asset.
```

4 Submit Predictions: Model Provider



This screen allows a model provider to look at requests for prediction from a given model, and allows him to submit the predictions from the model on the blockchain. The assets used for this page are:

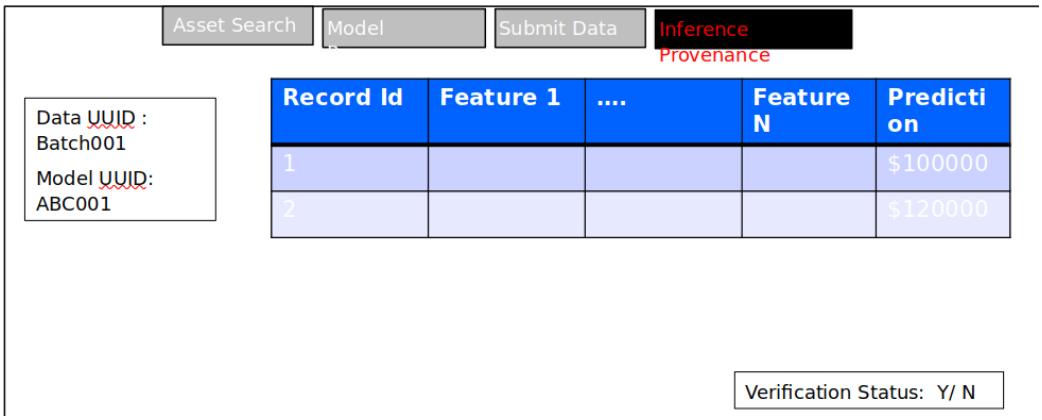
```
1. Batch data asset (assets/batch_data_asset.json)
    Relevant Keys:
    .assetUUID :- for showing UUID in the table.
    .lineageInfo.transformationInfo.description :- for showing
   description in the table
2. Model Inference asset (assets/scores_batch_asset0.json)
    A model inference asset will be created and submitted to
   blockchain (see Action subsection).
```

4.1 Action

Model provider provides the `modelUUID` and gets a table of pending requests for that model. Then for one of the data batches, it generates predictions (offline, with proof) and adds the `Model Inference Asset` to blockchain.

```
1. Query the blockchain for '.assetType==data and
    .plainTextContext.requestedModel==modelUUID'.
2. Display the description for the assets in the table
(.lineageInfo.transformationInfo.description key).
3. The model provider uploads predictions (generated offline) with
    proof.
     -- upload the file ../data/predictions.yaml
4. After uploading, on "Submit" execute the following script to
    create Model Inference asset on blockchain:
     -- ./submit_model_inference_asset.sh ../data/predictions.yaml
<ABC-001-model-asset>  <Batch-001-data-asset>
    <Scores-001-model-inference-asset> "Scores on batch data"
5. Display the assetUUID from the recevied asset (Scores-001-*).
```

5 Inference Provenance: AI Assistant



This screen allows `AI Assistant` to fetch predictions on previously submitted data batch (as in Section 3) and verify that the predictions were made from the specified model. The AI assistant specifies `modelUUID` and `dataUUID`. The `Model Inference` asset is fetched from blockchain and verified. The assets involved are:

```
Model Inference Asset(assets/scores_data_asset.json).
Relevant Keys:
   - .plainTextContext.assetPlainText :- key containing predictions.

Batch Data Asset(assets/batch_data_asset.json)
Relevant Keys:
   - .plainTextContext.assetPlainText :- key containing batch
    records.
```

5.1 Action

The following needs to be executed to populate verification status.

```
1. Fetch asset with assetType == Model-Inference,
    .lineageInfo.sourceAssets[0] == modelUUID,
    .lineageInfo.sourceAssets[1] == dataUUID.
2. Fetch the data asset with assetUUID == dataUUID.
3. Display the records from data asset into the table using the key
    .plainTextContext.assetPlainText.
4. Display predictions from model inference asset into the table
    using the key .plainTextContext.assetPlainText.
5. Verification status can be computed by running the script:
  --- ./verify_model_inference_asset.sh <path to the fetched
    inference asset>
```