GREENPLUM SUMMIT

by Pivotal

AT POSTGRESCONF | NEW YORK | #SCALEMATTERS

Pivotal

Agile Data Science on Greenplum Using Airflow

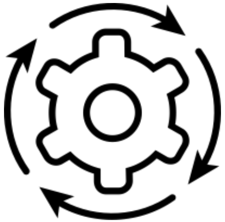
Ambarish Joshi - Data Scientist

Aditya Padhye - Data Engineer

“**Agile software development** refers to a group of **software development** methodologies based on **iterative development**, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams.”

Data Science Phases

Discovery Phase



Operationalization (O16n) Phase

Data Science Phases

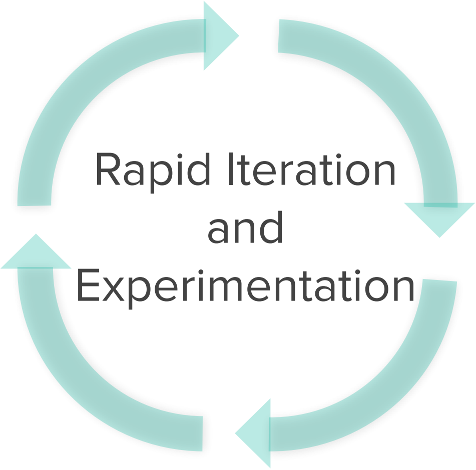
Discovery Phase

- ✓ Data exploration & cleaning
- ✓ Feature engineering
- ✓ Model Building
- ✓ Model Evaluation

Data Science Phases - Agility

Discovery Phase

- ✓ Data exploration & cleaning
- ✓ Feature engineering
- ✓ Model Building
- ✓ Model Evaluation

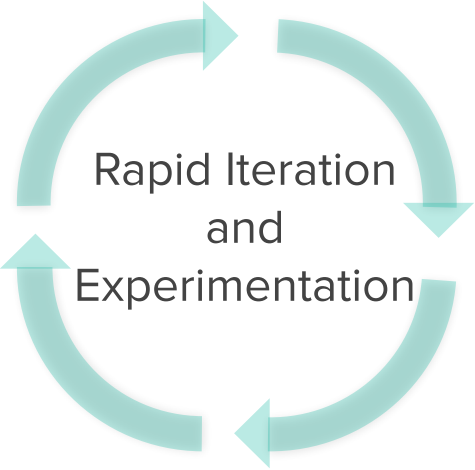


Data Science Phases - Agility

Discovery Phase

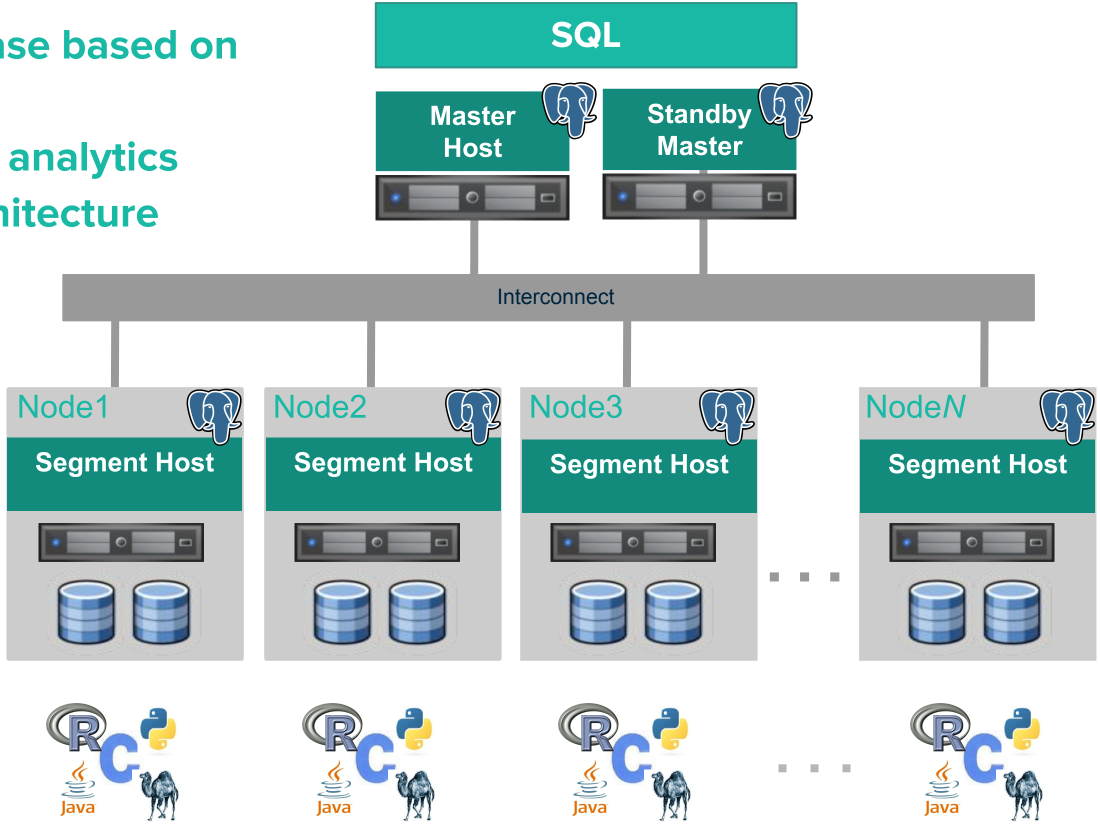
- ✓ Data exploration & cleaning
- ✓ Feature engineering
- ✓ Model Building
- ✓ Model Evaluation

jupyter Pivotal Greenplum

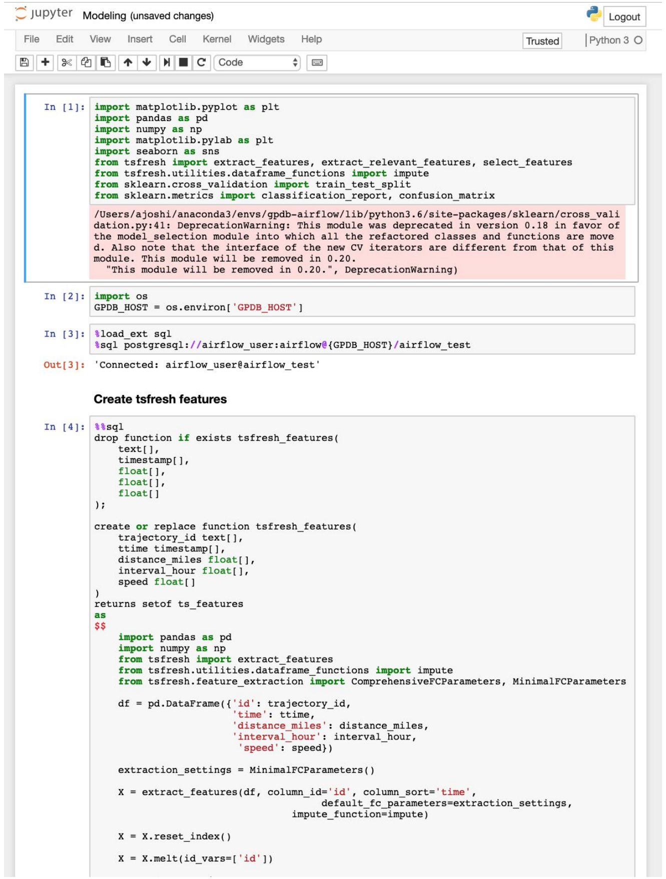


Greenplum Database

- MPP database based on Postgres
- In database analytics
- Parallel architecture

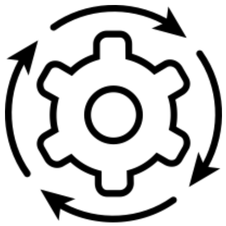


Jupyter Notebooks



Data Science Phases

Discovery Phase



Operationalization (O16n) Phase

Data Science Phases

Operationalization (O16n) Phase

- ✓ Data Pipelines
- ✓ Testing
- ✓ Monitoring
- APIs to consume model output

Data Science Phases - Agility

✓ Automated manageable pipelines

✓ Testing with CI

✓ Monitoring to react to Failures

Madlib Flow Talk by Frank and Sridhar

Operationalization (O16n) Phase

✓ Data Pipelines

✓ Testing

✓ Monitoring

APIs to consume model output

Data Science Phases - Agility

Pivotal Greenplum

Operationalization (O16n) Phase

- ✓ Automated manageable pipelines
- ✓ Testing with CI
- ✓ Monitoring to react to Failures

Madlib Flow Talk by Frank and Sridhar

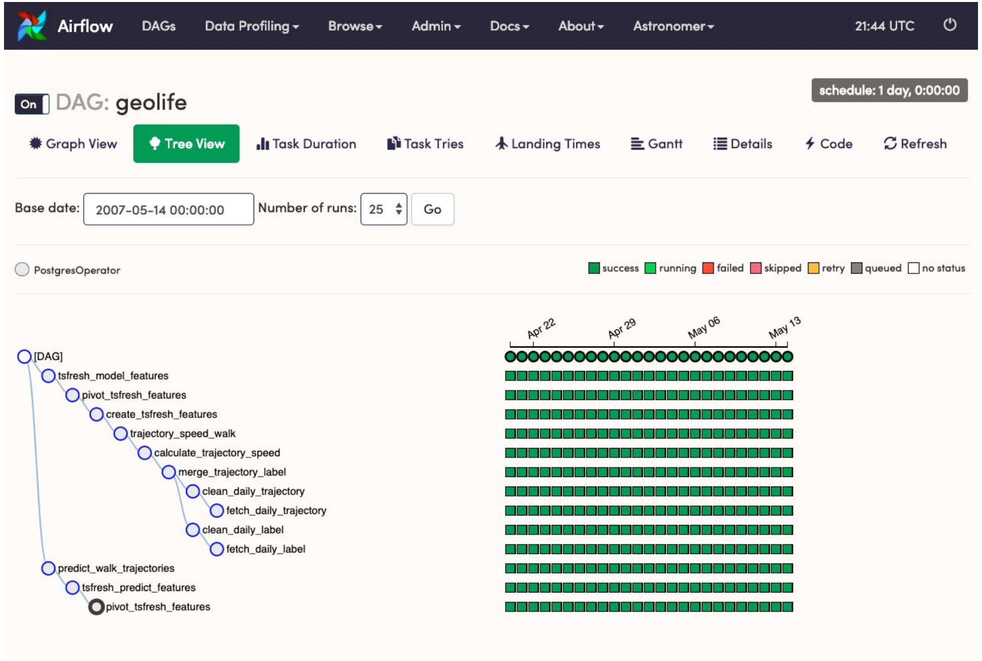
- ✓ Data Pipelines
- ✓ Testing
- ✓ Monitoring

APIs to consume model output

Airflow

- Apache Project spun out of Airbnb

- “Airflow is a platform to programmatically author, schedule and monitor workflows.”



Data Science Use-Case

- **The Data**
  - **Time-series trajectories** with **latitude** and **longitude** of location.
  - Subset of trajectories are labeled as **walk** / **not walk**

- **Our Model**
  - Build **Classification model** using labelled data to identify if new unlabeled trajectories are walk or not walk

`

Example trajectories

Example data

We have mode labels of walk and not walk only for subset of incoming daily trajectories

Example trajectory data

uid	latitude	longitude	tdate	ttime
020	39.9744533333333	116.302163333333	2011-08-25	14:38:25
020	39.97445	116.302165	2011-08-25	14:38:26
020	39.9746016666667	116.302073333333	2011-08-25	14:39:22
020	39.97473	116.302066666667	2011-08-25	14:39:23
020	39.9748116666667	116.30207	2011-08-25	14:39:24

Example label data

uid	start_date	start_time	end_date	end_time	mode
020	2011-08-27	06:13:01	2011-08-27	08:01:37	walk
020	2011-08-27	09:34:43	2011-08-27	14:50:30	walk
020	2011-08-27	14:50:31	2011-08-27	15:01:58	bus
020	2011-08-27	15:01:59	2011-08-27	15:31:43	walk
020	2011-08-28	04:33:31	2011-08-28	04:44:25	walk

Discovery phase → Operationalization phase

After every model iteration we check if the model is viable

- Check the quantitative metrics of the model like AUC, ROC curve, accuracy etc
- Check the qualitative results of the model and if it make sense to a subject matter expert

Once we are convinced that the model is both quantitatively and qualitatively viable we can move to the Operationalization phase

Discovery phase → Operationalization phase

Example of code from the discovery phase which is converted into a task script

Jupyter Geolife Airflow (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

Code

calculate speed

In [53]:

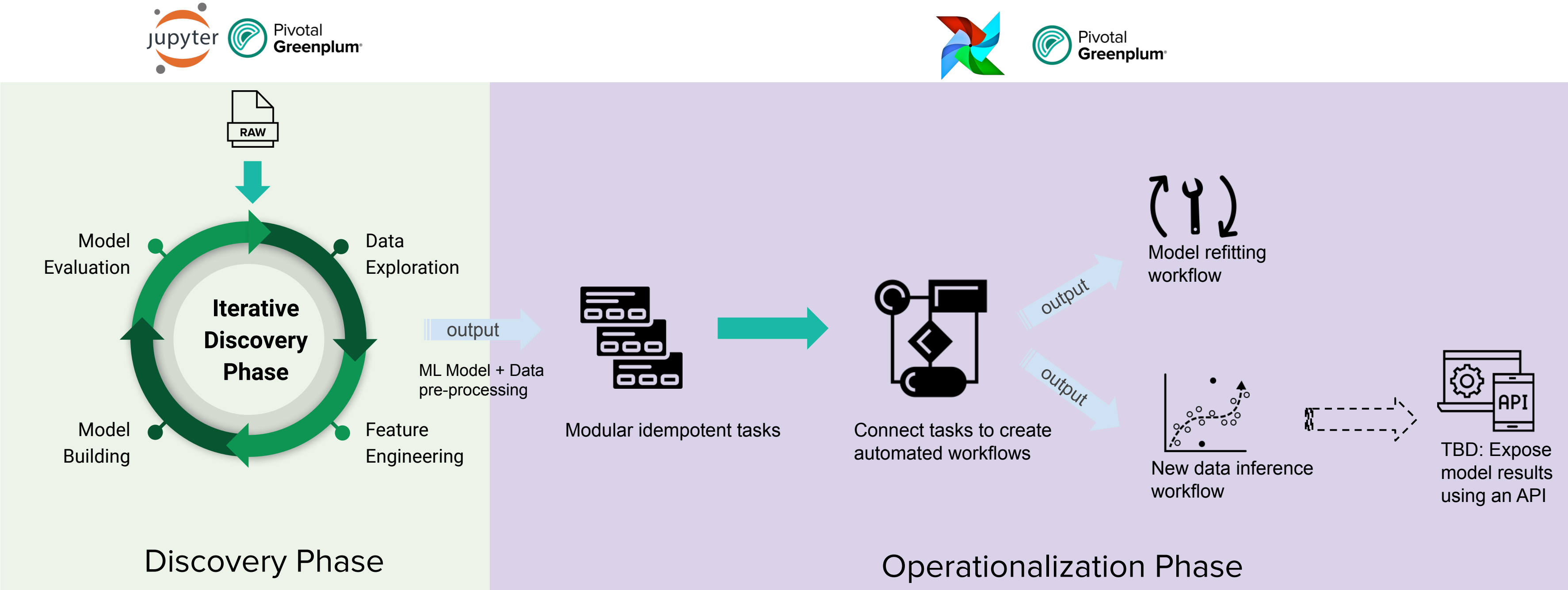
```
%%sql
drop table if exists exp.trajectory_label_speed;
create table exp.trajectory_label_speed
as
with lead_trajectory as (
select *,
    lead(latitude) over(partition by trajectory_id order by ttimestamp) as lead_lat,
    lead(longitude) over(partition by trajectory_id order by ttimestamp) as lead_long,
    lead(ttimestamp) over(partition by trajectory_id order by ttimestamp) as lead_ttimestar
from exp.trajectory_label
),
t2 as (
select uid,
    trajectory_id,
    mode,
    pt,
    ST_SetSRID(st_point(lead_long, lead_lat),4326) as lead_pt,
    tdate,
    ttime,
    ttimestamp,
    lead_ttimestamp
from lead_trajectory
),
t3 as (
    select *,
        st_distance(st_transform(pt, 2163) , st_transform(lead_pt, 2163)) / 1609.34 as dis
        EXTRACT(EPOCH FROM (lead_ttimestamp - ttimestamp)) / 3600.0 as interval_hour
    from t2
    where lead_ttimestamp != ttimestamp --removing divide by zero error
)
select *,
    distance_miles / interval_hour as speed
from t3
distributed by (uid, trajectory_id)
```

```
 * postgresql://airflow_user:***@172.16.223.128/airflow_test
Done.
35177 rows affected.
```

calculate_trajectory_speed.sql

```
alter table geolife.geolife_trajectory_label_speed drop partition if exists p{{ ds_nodash }};
alter table geolife.geolife_trajectory_label_speed add partition p{{ ds_nodash }}
values (date '{{ ds }}');

-- Calculate distance, interval and speed between 2 consecutive gps location points
insert into geolife.geolife_trajectory_label_speed
with lead_trajectory as (
select *,
    lead(latitude) over(partition by trajectory_id order by ttimestamp) as lead_lat,
    lead(longitude) over(partition by trajectory_id order by ttimestamp) as lead_long,
    lead(ttimestamp) over(partition by trajectory_id order by ttimestamp) as lead_ttimestamp
from geolife.geolife_trajectory_label_clean
where tdate = '{{ ds }}'
),
t2 as (
select uid,
    trajectory_id,
    mode,
    pt,
    ST_SetSRID(st_point(lead_long, lead_lat),4326) as lead_pt,
    tdate,
    ttime,
    ttimestamp,
    lead_ttimestamp
from lead_trajectory
),
t3 as (
    select *,
        st_distance(st_transform(pt, 2163) , st_transform(lead_pt, 2163)) / 1609.34 as distance_miles,
        EXTRACT(EPOCH FROM (lead_ttimestamp - ttimestamp)) / 3600.0 as interval_hour
    from t2
    where lead_ttimestamp != ttimestamp --removing divide by zero error
)
select *,
    distance_miles / interval_hour as speed
from t3;
```

Architecture overview



Data Science Phases - Agility

Operationalization (O16n) Phase

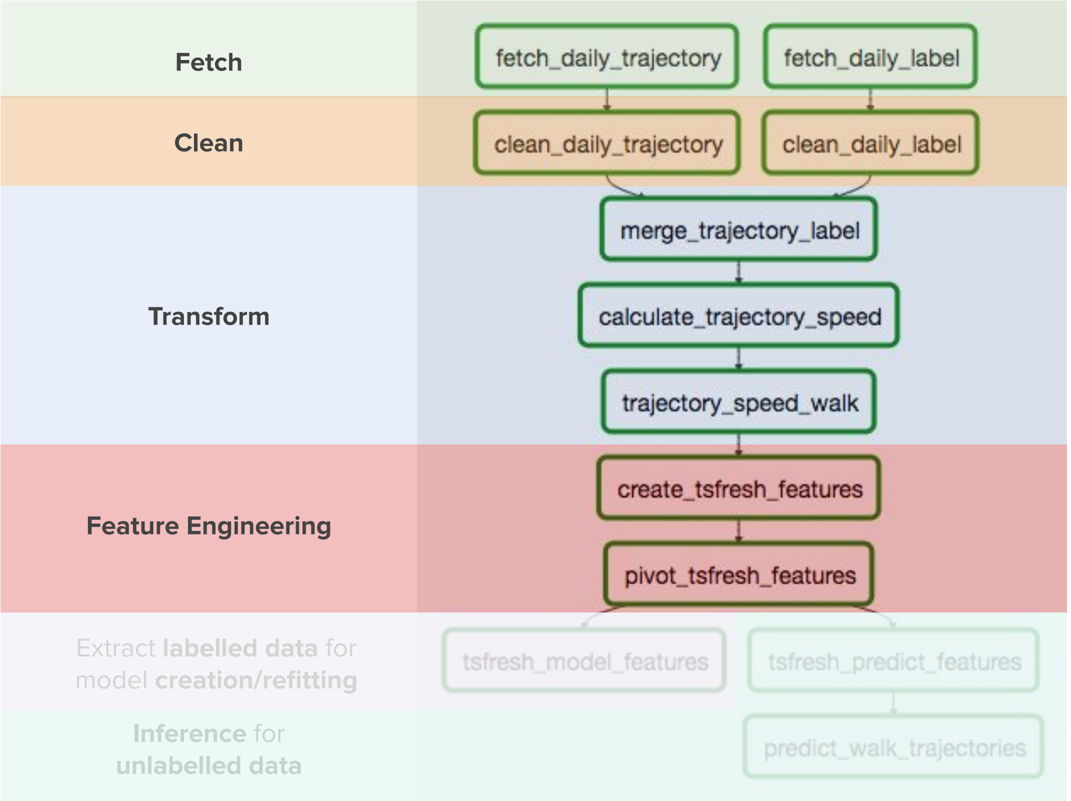
- ✓ Automated manageable Pipelines
- ✓ Testing with CI/CD
- ✓ Monitoring to React to Failures

Madlib Flow Talk by Frank and Sridhar

- ✓ Pipelines
- ✓ Testing
- ✓ Monitoring

APIs to consume model output

Data Prep and Feature Engineering

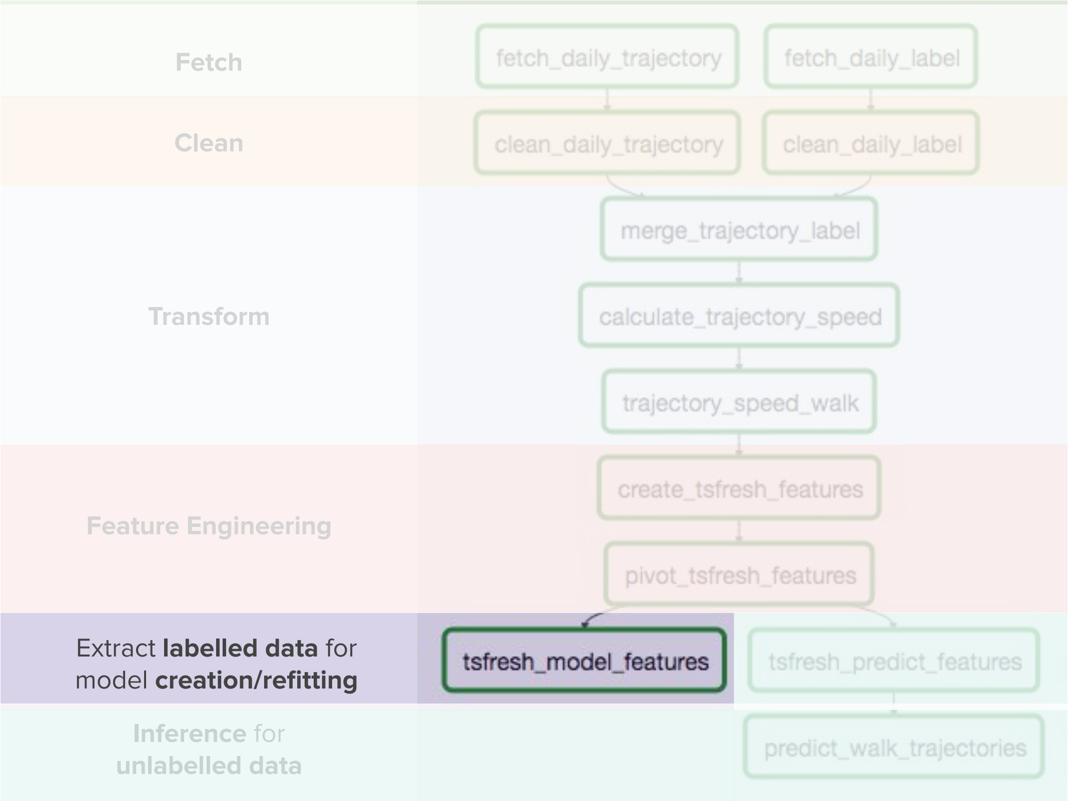


Pivotal

Data Prep and Feature Engineering - Demo

Pivotal

Model Training



Model Training

- This DAG has a single task for model training
- In this task we split the data into train and test samples, train the model, evaluate the model and capture the accuracy, auc and model tables.
- We want all of the above to run at the same time

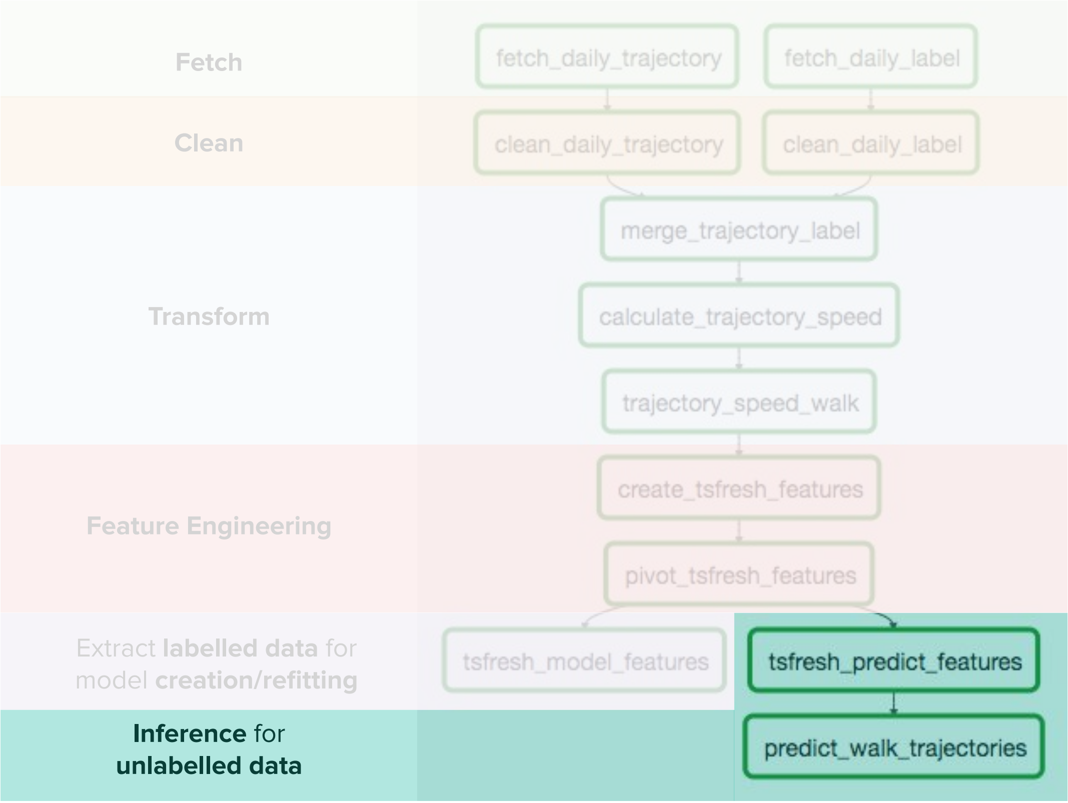
train_model

```
SELECT madlib.train_test_split(
                                'geolife.tsfresh_model_features',    -- Source table
                                'geolife.features_walk_{{ds_nodash}}',    -- Output table
                                0.8,        -- Sample proportion
                                0.2,        -- Sample proportion
                                NULL, -- Strata definition
                                NULL, -- Columns to output
                                FALSE,      -- Sample without replacement
                                TRUE);     -- Separate output tables

-- build a random forest model using madlib

DROP TABLE IF EXISTS geolife.rf_walk_{{ds_nodash}}_output, geolife.rf_walk_{{ds_nodash}}_outp
SELECT madlib.forest_train('geolife.features_walk_{{ds_nodash}}_train',          -- source tab
                           'geolife.rf_walk_{{ds_nodash}}_output',    -- output model table
                           'id',                -- id column
                           'label',             -- response
                           '*',     -- features
                           'tdate',                -- exclude columns
                           NULL,                -- grouping columns
                           20::integer,         -- number of trees
                           2::integer,          -- number of random features
                           TRUE::boolean,       -- variable importance
                           1::integer,          -- num_permutations
                           8::integer,          -- max depth
                           3::integer,          -- min split
                           1::integer,          -- min bucket
                           10::integer          -- number of splits per continuous variable
                           );

-- Evalute the built model

DROP TABLE IF EXISTS geolife.rf_walk_{{ds_nodash}}_results;
SELECT madlib.forest_predict('geolife.rf_walk_{{ds_nodash}}_output',        -- tree model
                             'geolife.features_walk_{{ds_nodash}}_test',          -- new d
                             'geolife.rf_walk_{{ds_nodash}}_results') ;--,   -- output table
                             --'prob');


-- Capture model results

drop table if exists geolife.walk_{{ds_nodash}}_result;
create table geolife.walk_{{ds_nodash}}_result
as
with t as (
select id,
    case when label = True then 1.0 else 0.0 end as obs
from geolife.features_walk_{{ds_nodash}}_test
```

Model Training - Demo

Pivotal

Model Scoring



Pivotal

Model Scoring

- The unlabeled data which is extracted from the features table is scored in this DAG
- We first check if any model has been built
- If there is a model so we score the data (inference)

```
DO
$do$
DECLARE
tabname   character varying(255);
BEGIN
IF (select count(*) from
  (select 1 from geolife.tsfresh_predict_features{{ds_nodash}}  limit 1) as t) > 0
  and
  (select count(*) from (select 1 from geolife.models_metadata  limit 1) as p) > 0
THEN
    tabname := (select model_tabname
    from geolife.models_metadata
    order by mdate
    limit 1);

    DROP TABLE IF EXISTS prediction_results;
    PERFORM madlib.forest_predict(tabname,
                                 'geolife.tsfresh_predict_features{{ds_nodash}}',
                                 'geolife.walk_prediction_results{{ds_nodash}}',
                                 'response');

    insert into geolife.walk_prediction_results
    select *,
        regexp_replace(id, '^.*([0-9-]{10})_.*$', E'\\1')::date as tdate
    from geolife.walk_prediction_results{{ds_nodash}};

END IF;
END
$do$;

drop table if exists geolife.tsfresh_predict_features{{ds_nodash}};
drop table if exists geolife.walk_prediction_results{{ds_nodash}};
```

Model Scoring - Demo

Model Re-Training

- Daily we get some more labeled data, once we have accumulated enough labeled data we can retrain the model for better accuracy
- We have scheduled model re-training monthly

Model Re-Training - Demo

Data Science Phases - Agility

Operationalization (O16n) Phase

- ✓ Automated manageable Pipelines
- ✓ Testing with CI/CD
- ✓ Monitoring to React to Failures
- Madlib Flow Talk by Frank and Jarrod

- ✓ Pipelines
- ✓ Testing
- ✓ Monitoring
- APIs to consume model output

Testing with CI/CD

- Testing Data Pipelines is hard
- Test Coverage (Test Tasks vs Test DAGs)
- Testing as part of the CI/CD

Testing with CI/CD - Demo

Pivotal

Data Science Phases - Agility

Pivotal Greenplum

- ✓ Automated manageable Pipelines
- ✓ Testing with CI/CD
- ✓ Monitoring to React to Failures

Madlib Flow Talk by Frank and Sridhar

Operationalization (O16n) Phase

- ✓ Pipelines
- ✓ Testing
- ✓ Monitoring

APIs to consume model output

Monitoring and Error Fixing

- Monitoring and error fixing is big part of responsive data pipelines
- Ability to quickly identify what is failing, why it is failing and fixing it with minimum lead time is crucial
- In this demo we will showcase an error fixing case

Monitoring and Error Fixing - Demo

Pivotal

Data Science Phases - Agility

Pivotal Greenplum

✓ Automated manageable Pipelines

✓ Testing with CI/CD

✓ Monitoring to React to Failures

Madlib Flow Talk by Frank and Sridhar

Operationalization (O16n) Phase

✓ Pipelines

✓ Testing

✓ Monitoring

APIs to consume model output

Conclusion

- ✓ Greenplum and Jupyter notebooks provides a set of tools to do Agile Data Science during discovery phase

- ✓ Greenplum along with Airflow and Circle CI is very effective to do Agile Data Science during the operationalization phase

Questions

"We partner to help you compete, grow, and transform."

Pivotal