Unified MLOps:
Feature Stores and Model Deployment

Monte Zweben

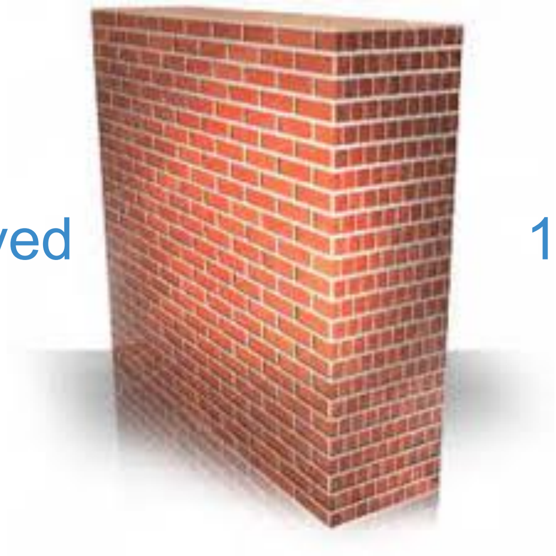
Co-Founder & CEO

QCon 2021

splice MACHINE

Why Do You Need a Feature Store?

1-5 models deployed



100’s models deployed

Why Is This So Hard?

Productivity

Feature engineering consumes everyone

Predictive Accuracy

Training and serving pipelines are often inconsistent and wrong

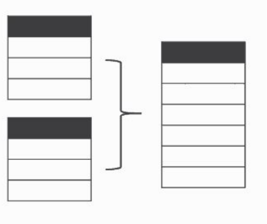
Model Governance

Why did the model do that?

How do you use a Feature Store?

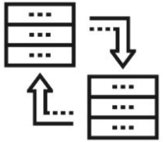
Feature Reuse

Create Training Sets



Serve data to models and dashboards

End-to-end lineage



Feature Reuse

- Feature Reuse
- Create Training Sets
- Serve data to models and dashboards
- End-to-end lineage

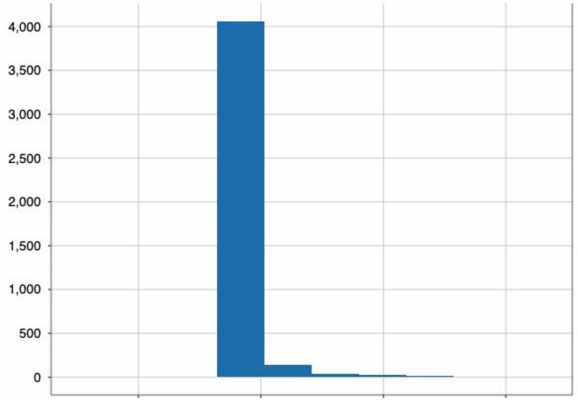
Without a Feature Store	With a Feature Store
Data Scientists define features in Python, engineers then translate into robust SQL or Spark pipelines	Data Scientists easily define production ready features
Duplicate code and infrastructure for redundant feature engineering pipelines	Single line of Feature Store API to reuse features

Feature Search

index	NAME	FEATURE_TYPE	DESCRIPTION	SCHEMA_NAME	TABLE_NAME	SETDESCRIPTION
0	CUSTOMER_RFM_DELICATESSEN_RATE_1W	C	Last weeks units purchased count in the Deli category.	RETAIL_FS	CUSTOMER_RFM_BY_CATEGORY	Describes customer by aggregating their purchases by category over multiple time windo
1	CUSTOMER_RFM_DELICATESSEN_RATE_2W	C	Last 2 weeks units purchased count in the Deli category.	RETAIL_FS	CUSTOMER_RFM_BY_CATEGORY	Describes customer by aggregating their purchases by category over multiple time windo
2	CUSTOMER_RFM_DELICATESSEN_RATE_4W	C	Last 4 weeks units purchased count in the Deli category.	RETAIL_FS	CUSTOMER_RFM_BY_CATEGORY	Describes customer by aggregating their purchases by category over multiple time windo
3	CUSTOMER_RFM_DELICATESSEN_RATE_8W	C	Last 8 weeks units purchased count in the Deli category.	RETAIL_FS	CUSTOMER_RFM_BY_CATEGORY	Describes customer by aggregating their purchases by category over multiple time windo
4	CUSTOMER_RFM_DELICATESSEN_RATE_16W	C	Last 16 weeks units purchased count in the Deli category.	RETAIL_FS	CUSTOMER_RFM_BY_CATEGORY	Describes customer by aggregating their purchases by category over multiple time windo
5	CUSTOMER_RFM_DELICATESSEN_RATE_32W	C	Last 32 weeks units purchased count in the Deli category.	RETAIL_FS	CUSTOMER_RFM_BY_CATEGORY	Describes customer by aggregating their purchases by category over multiple time windo

```
:quiring feature data...
ean:   1.62    std:    19.57
edian: 0.0
ange: -38.0 to 1080.0
```

Feature - RETAIL_FS.CUSTOMER_RFM_BY_CATEGORY - CUSTOMER_RFM_DELICATESSEN_RATE_1W Distribution



Automatically Generate Common Feature Transformations

```
##################################

start_time = '2020-12-28T00:00:00'
schedule_interval = AggWindow.get_window(7,AggWindow.DAY)


backfill_start = datetime.strptime('2019-04-01 00:00:00', '%Y-%m-%d %H:%M:%S')
backfill_interval = schedule_interval

###################################

fs.create_aggregation_feature_set_from_source(
    source_name, 'ecommerce_fs', 'customer_purchases', start_time=start_time,
    schedule_interval=schedule_interval, backfill_start_time=backfill_start,
    backfill_interval=backfill_interval,
    aggregations = [
        FeatureAggregation(feature_name_prefix = 'PRODUCE_REVENUE',column_name = 'PRODUCE_REVENUE',agg_functions=['sum'], agg_windows=['1d','7d', '30d', '365d'],agg_default_
        FeatureAggregation(feature_name_prefix = 'DELI_REVENUE',column_name = 'DELI_REVENUE',agg_functions=['sum'], agg_windows=['1d','7d', '30d', '365d'],agg_default_value
        FeatureAggregation(feature_name_prefix = 'DAIRY_REVENUE',column_name = 'DAIRY_REVENUE',agg_functions=['sum'], agg_windows=['1d','7d', '30d', '365d'],agg_default_valu

        FeatureAggregation(feature_name_prefix = 'CARROTS_REVENUE',column_name = 'CARROTS_REVENUE',agg_functions=['sum'], agg_windows=['1d','7d', '30d', '365d'],agg_default_
        FeatureAggregation(feature_name_prefix = 'APPLES_REVENUE',column_name = 'APPLES_REVENUE',agg_functions=['sum'], agg_windows=['1d','7d', '30d', '365d'],agg_default_va
        FeatureAggregation(feature_name_prefix = 'BANANAS_REVENUE',column_name = 'BANANAS_REVENUE',agg_functions=['sum'], agg_windows=['1d','7d', '30d', '365d'],agg_default_
        FeatureAggregation(feature_name_prefix = 'GRAPES_REVENUE',column_name = 'GRAPES_REVENUE',agg_functions=['sum'], agg_windows=['1d','7d', '30d', '365d'],agg_default_va
        FeatureAggregation(feature_name_prefix = 'TURKEY_REVENUE',column_name = 'TURKEY_REVENUE',agg_functions=['sum'], agg_windows=['1d','7d', '30d', '365d'],agg_default_va
        FeatureAggregation(feature_name_prefix = 'CHICKEN_REVENUE',column_name = 'CHICKEN_REVENUE',agg_functions=['sum'], agg_windows=['1d','7d', '30d', '365d'],agg_default_
        FeatureAggregation(feature_name_prefix = 'BEEF_REVENUE',column_name = 'BEEF_REVENUE',agg_functions=['sum'], agg_windows=['1d','7d', '30d', '365d'],agg_default_value
        FeatureAggregation(feature_name_prefix = 'SKIM_MILK_REVENUE',column_name = 'SKIM_MILK_REVENUE',agg_functions=['sum'], agg_windows=['1d','7d', '30d', '365d'],agg_defa
        FeatureAggregation(feature_name_prefix = 'SOY_MILK_REVENUE',column_name = 'SOY_MILK_REVENUE',agg_functions=['sum'], agg_windows=['1d','7d', '30d', '365d'],agg_defaul
        FeatureAggregation(feature_name_prefix = 'CHEESE_REVENUE',column_name = 'CHEESE_REVENUE',agg_functions=['sum'], agg_windows=['1d','7d', '30d', '365d'],agg_default_va
    ]
)
```

Create Training Set

Feature Reuse	Create Training Sets
Serve data to models and dashboards	End-to-end lineage

Without a Feature Store	With a Feature Store
Hundreds of lines of complex and error prone SQL to join features and labels correctly	Simply specify a training label and join keys to automatically create training sets
Feature values of the past may be lost, making it nearly impossible to build training sets in the future	Feature values are automatically versioned for use in future models

Training Set Creation

```
sql = """
SELECT ltv.CUSTOMERID,
       ((w.WEEK_END_DATE - ltv.CUSTOMER_START_DATE)/ 7) CUSTOMERWEEK,
       CAST(w.WEEK_END_DATE as TIMESTAMP) CUSTOMER_TS,
       ltv.CUSTOMER_LIFETIME_VALUE as CUSTOMER_LTV
FROM retail_rfm.weeks w --splice-properties useSpark=True
INNER JOIN
    retail_fs.customer_lifetime ltv
    ON w.WEEK_END_DATE > ltv.CUSTOMER_START_DATE AND w.WEEK_END_DATE <= ltv.CUSTOMER_START_DATE + 28 --only first 4 weeks
"""

pks = ['CUSTOMERID','CUSTOMERWEEK'] # Each unique training row is identified by the customer and their week of spending activity
join_keys = ['CUSTOMERID'] # This is the primary key of the Feature Sets that we want to join to

fs.create_training_view(
    'Customer_Lifetime_Value',
    sql=sql,
    primary_keys=pks,
    join_keys=join_keys,
    ts_col = 'CUSTOMER_TS', # How we join each unique row with our eventual Features
    label_col='CUSTOMER_LTV', # The thing we want to predict
    desc = 'The current (as of queried) lifetime value of each customer per week of being a customer'
)
```

```
Registering Training View Customer_Lifetime_Value in the Feature Store
```

Easily extract all features

Every time this code is re-run you have access to the most up-to-date features

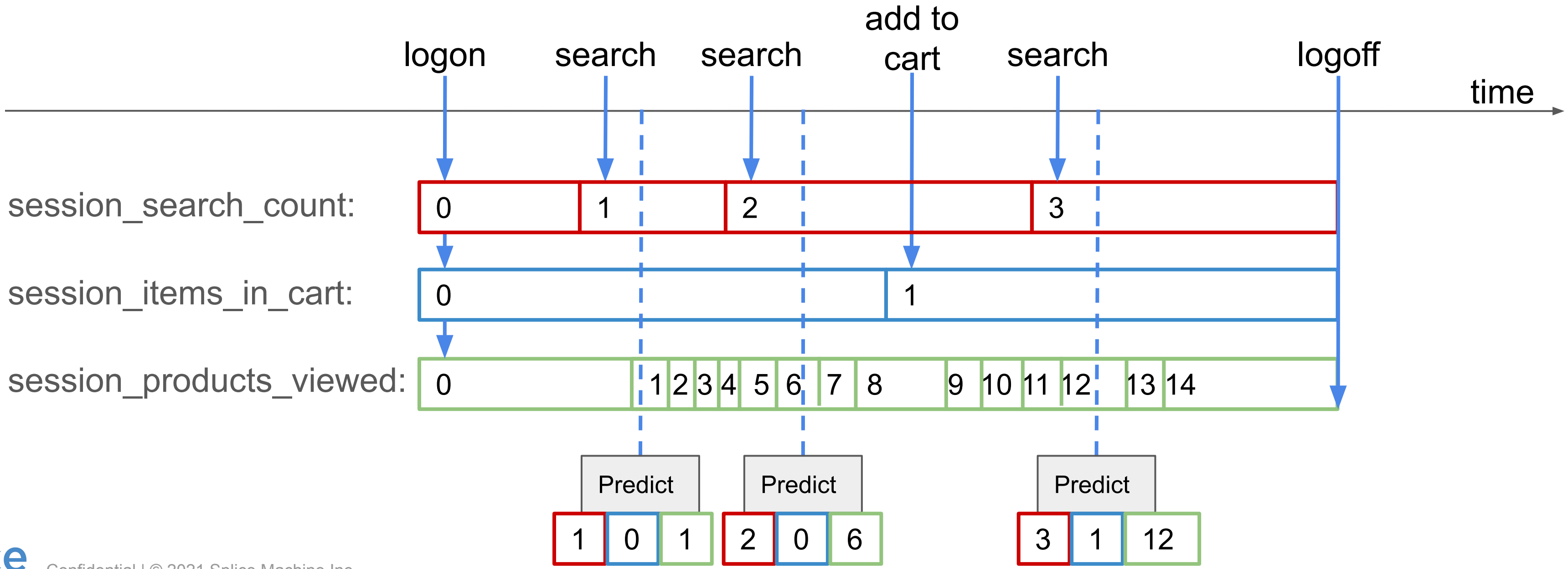
```
#Spark Dataframe
all_features = fs.get_training_set_from_view('Customer_Lifetime_Value')
all_features.limit(8).toPandas()
```

CUSTOMER_LIFETIME_ITEMS_PER_ACTIVE_DAY	CUSTOMER_LIFETIME_REVENUE_PER_ACTIVE_DAY	CUSTOMER_LIFETIME_DAYS	CUSTOMER_DAYS_SINCE_PURCHASE	CUSTOMER_LIFETIME_VALUE	CUSTOMER_START_DATE	CUSTOMER_L
137.0	219.350	522	522	219.35	2019-11-04	219.35
137.0	219.350	522	522	219.35	2019-11-04	219.35
137.0	219.350	522	522	219.35	2019-11-04	219.35
137.0	219.350	522	522	219.35	2019-11-04	219.35

Point-in-Time Correctness

Training should be done with the features available at model run time.

The problem: features change over time.



Serving Features

	Without a Feature Store	With a Feature Store
	Create bespoke pipelines that feed a key-value store	Serve the same features used to train models to deployed models
	Inevitable inconsistency between two separate databases means the features you need aren't always available or consistent with training data	ACID compliant triggers ensures that data used for training is always available for serving

- Feature Reuse
- Create Training Sets
- Serve data to models and dashboards
- End-to-end lineage

Feature Serving

```
feature_vector_sql = fs.get_feature_vector(features=features_list, return_sql=True, join_key_values={'customerid':'14235'})
print(feature_vector_sql)
```

```
SELECT fset2.customerid, fset2.customer_rfm_home_revn_rate_4w, fset2.customer_rfm_toys_rate_2w, fset2.customer_rfm_jewelry_rate_1w, fset2.customer_rfm_total_revn_rate_4w, fset2.customer_rfm_delicatessen_r
ate_4w, fset2.customer_rfm_home_revn_rate_2w, fset2.customer_rfm_school_supplies_rate_2w, fset3.customer_lifetime_days, fset2.customer_rfm_kitchen_revn_rate_4w, fset2.customer_rfm_total_rate_4w
FROM retail_fs.customer_rfm_by_category AS fset2, retail_fs.customer_lifetime AS fset3
WHERE fset2.customerid = '14235' AND fset3.customerid = '14235'
```

```
%%time
%%sql
SELECT CUSTOMER_RFM_HOME_DECOR_RATE_4W,CUSTOMER_RFM_KITCHEN_RATE_1W,CUSTOMER_RFM_TOTAL_RATE_4W,CUSTOMER_RFM_DELICATESSEN_REVN_RATE_2W,CUSTOMER_RFM_HOME_REVN_RATE_1W,CUSTOMER_RFM_HOME_REVN_RATE_4W,CUSTOM
FROM retail_fs.CUSTOMER_RFM_BY_CATEGORY fset1,
     retail_fs.CUSTOMER_LIFETIME fset10
WHERE fset1.CUSTOMERID = 14235 AND fset10.CUSTOMERID = 14235
```

index	Key	Value
0	CUSTOMER_RFM_HOME_DECOR_RATE_4W	0
1	CUSTOMER_RFM_KITCHEN_RATE_1W	0
2	CUSTOMER_RFM_TOTAL_RATE_4W	0
3	CUSTOMER_RFM_DELICATESSEN_REVN_RATE_2W	0
4	CUSTOMER_RFM_HOME_REVN_RATE_1W	0
5	CUSTOMER_RFM_HOME_REVN_RATE_4W	0
6	CUSTOMER_RFM_HOME_DECOR_REVN_RATE_4W	0
7	CUSTOMER_RFM_KITCHEN_REVN_RATE_4W	0
8	CUSTOMER_RFM_TOTAL_REVN_RATE_4W	0
9	CUSTOMER_LIFETIME_DAYS	612

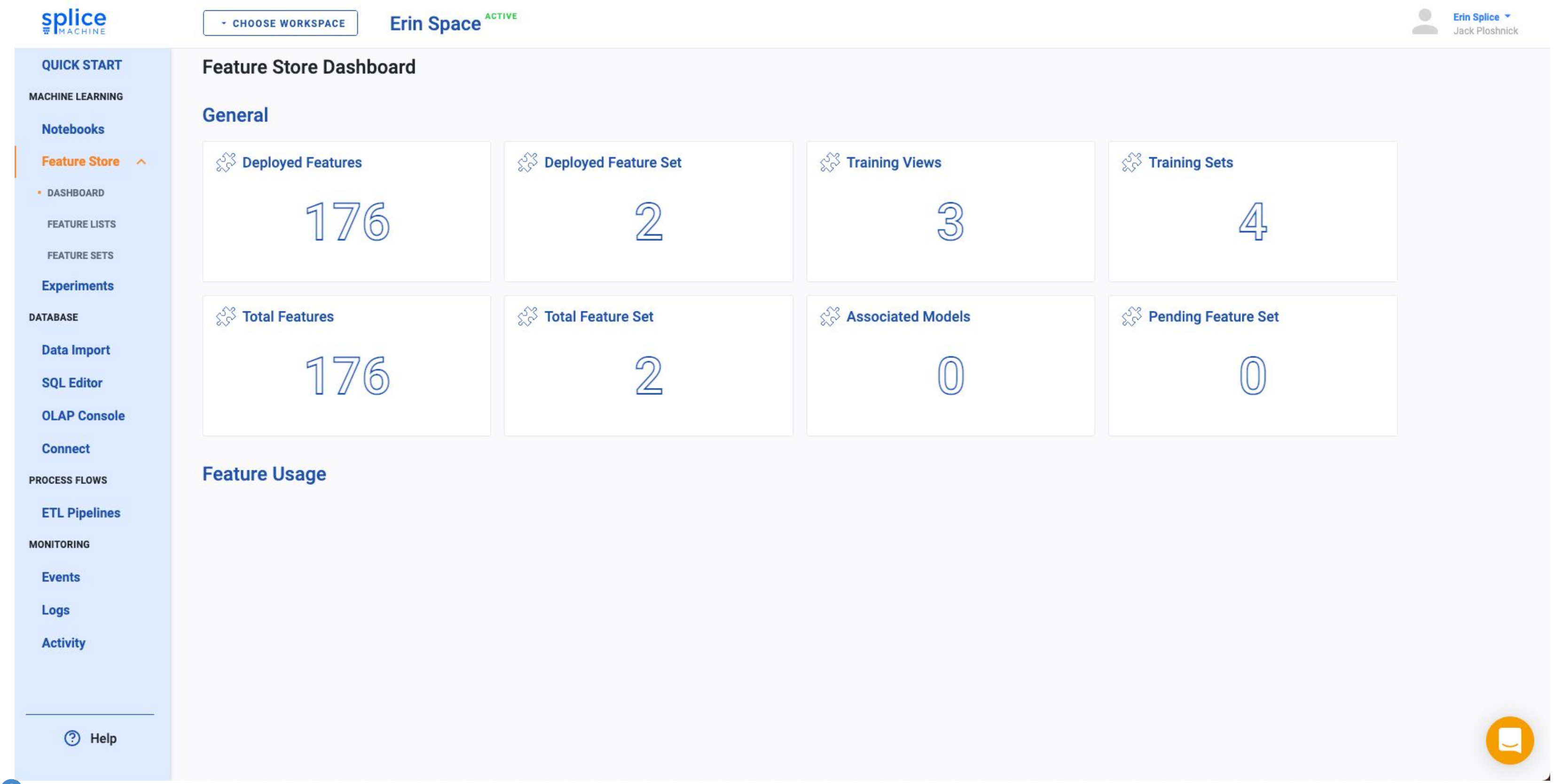
```
CPU times: user 6.69 ms, sys: 1.13 ms, total: 7.82 ms
```

End-to-End Governance

Feature Reuse	Create Training Sets
Serve data to models and dashboards	End-to-end lineage

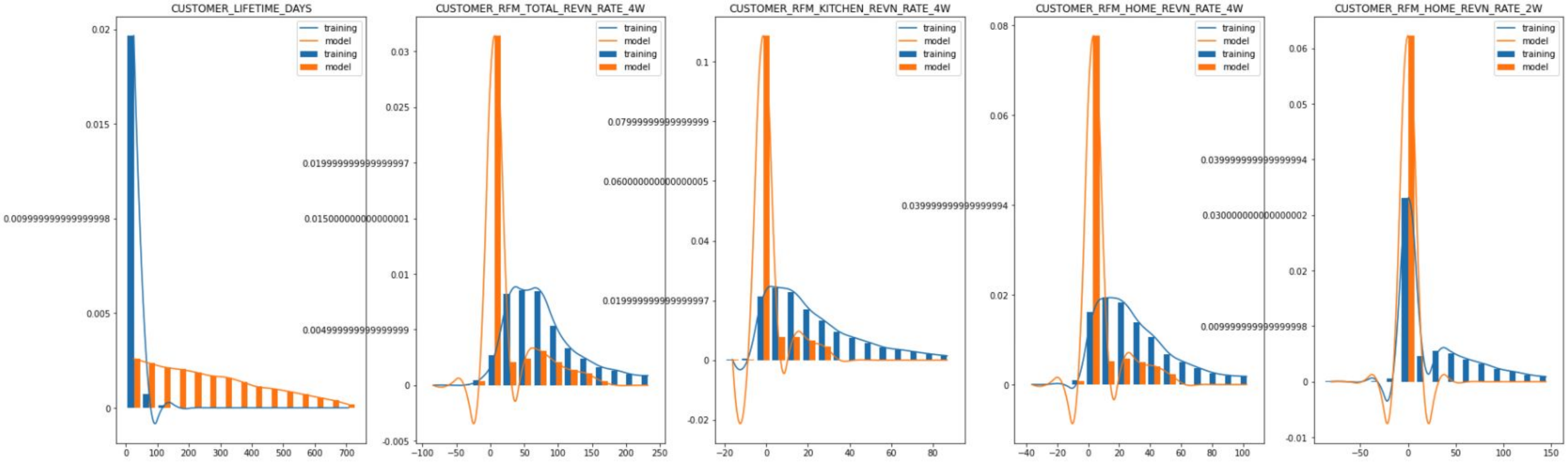
Without a Feature Store	With a Feature Store
Impossible or prohibitively expensive to find the training set used for a model	Instantly return the exact training set used for a model without having to persist the dataset
Search through API logs to determine what features were used and what predictions were made	Identify the features served to deployed models easily with a built in evaluation store

End-to-End Governance



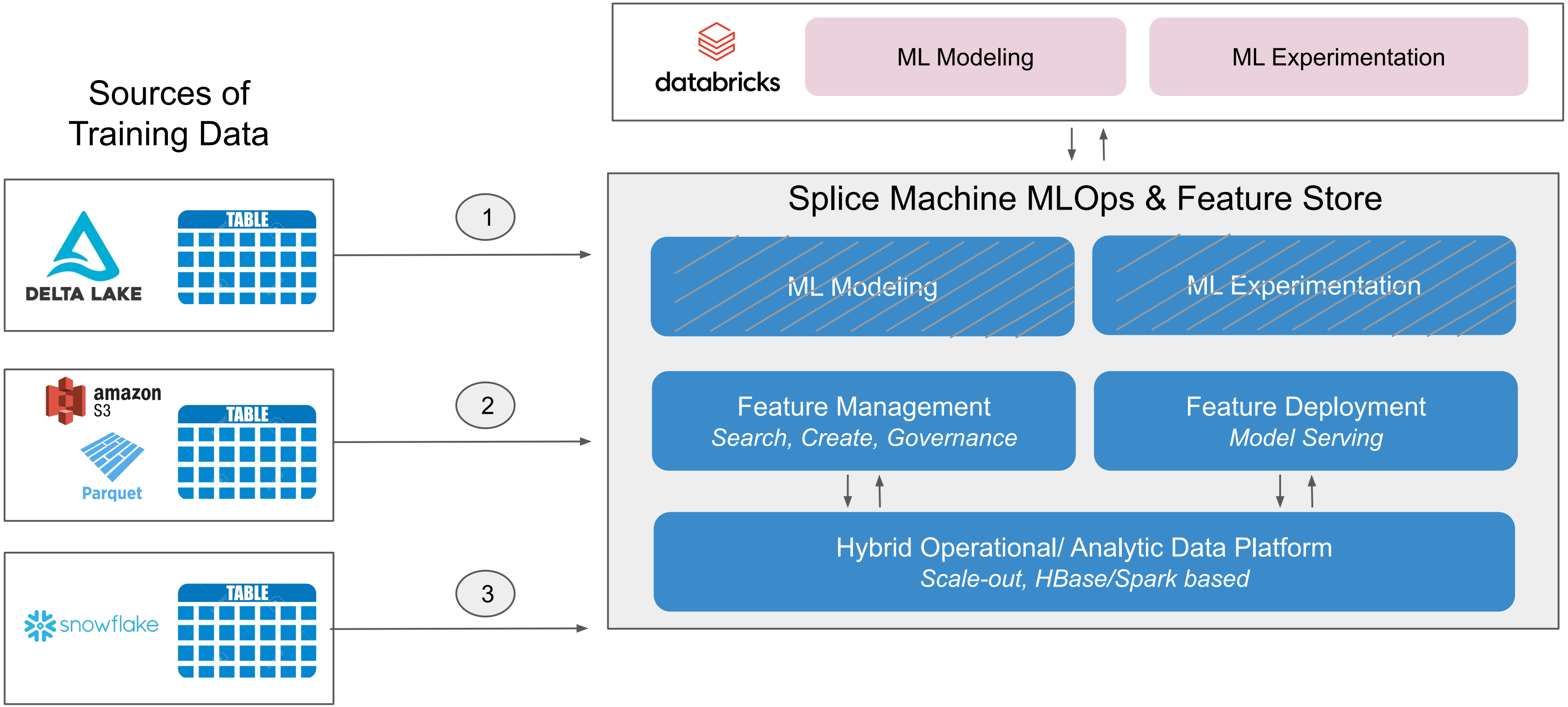
End-to-End Governance

```
fs.display_model_feature_drift('deployed_models','retail_regression')
```

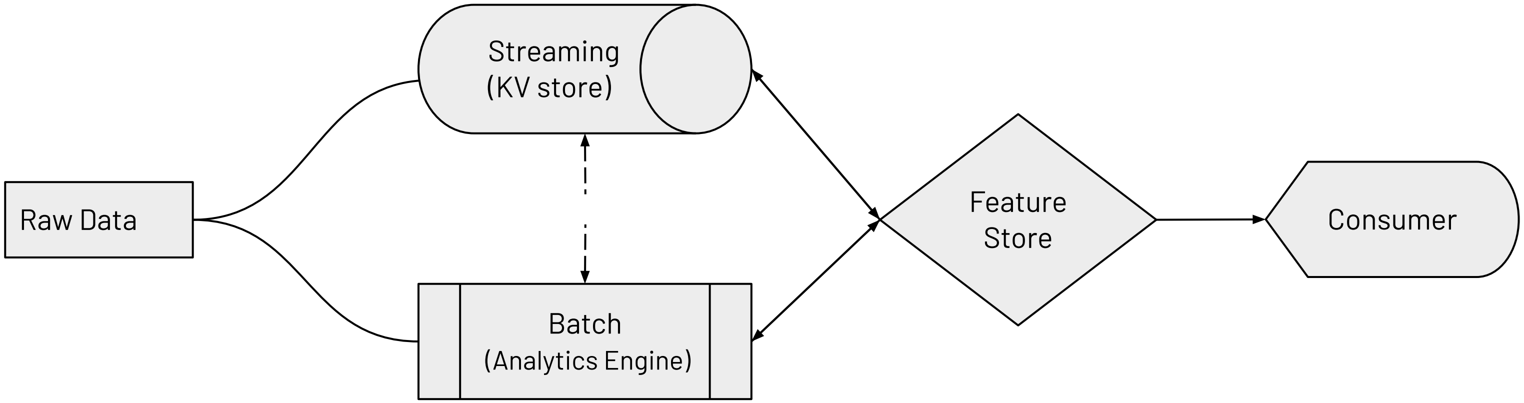


How It Works

How does this fit into the ML stack?

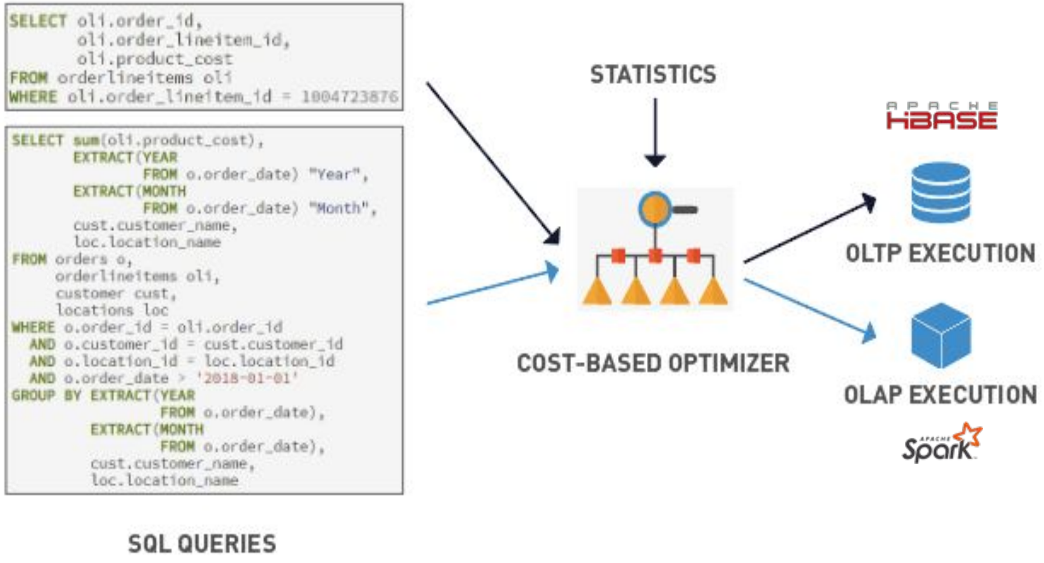


Problem: Disconnected Compute Engines

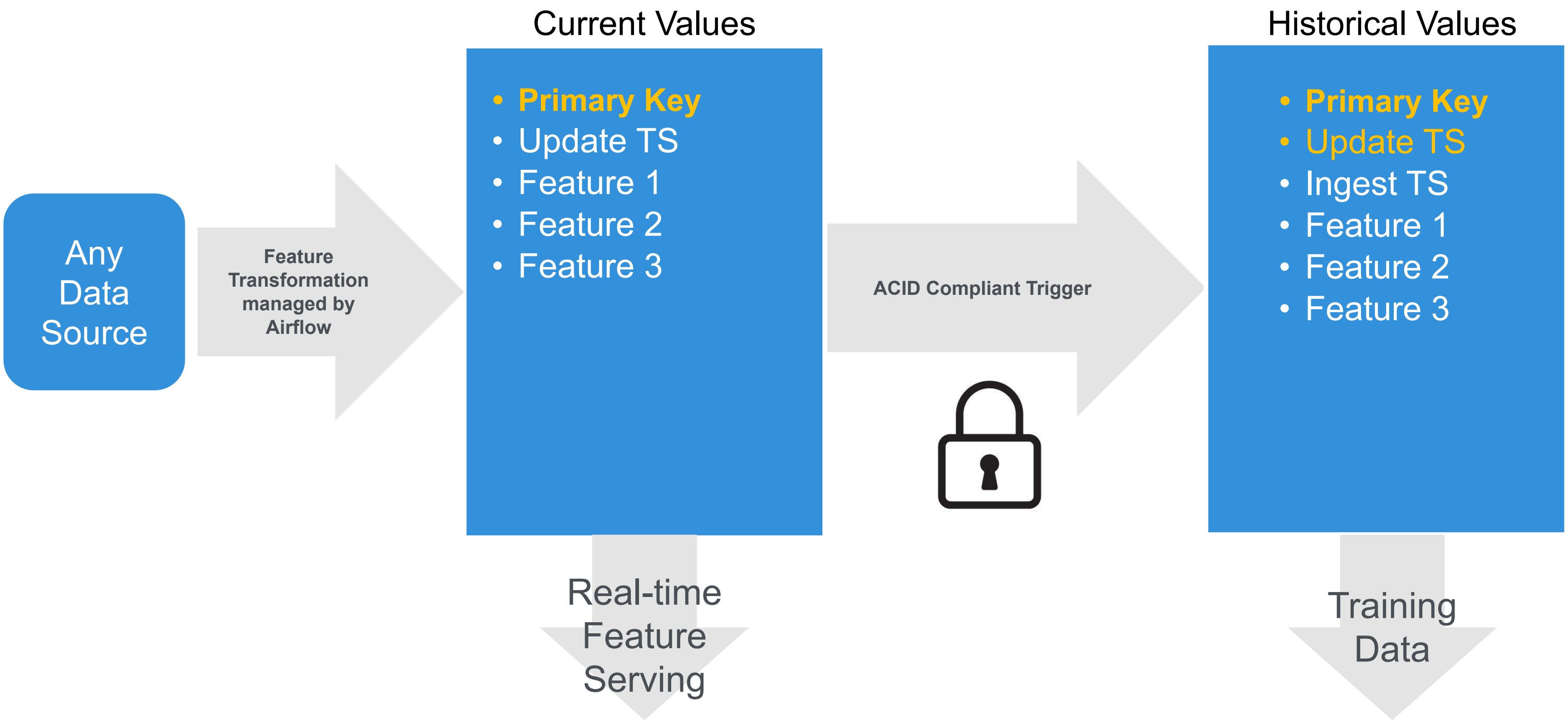


Solution: Hybrid Operational/ Analytical RDBMS

- Scale-out
- Any Cloud/On-Prem
- Indexes and Triggers
- Full ACID Compliance

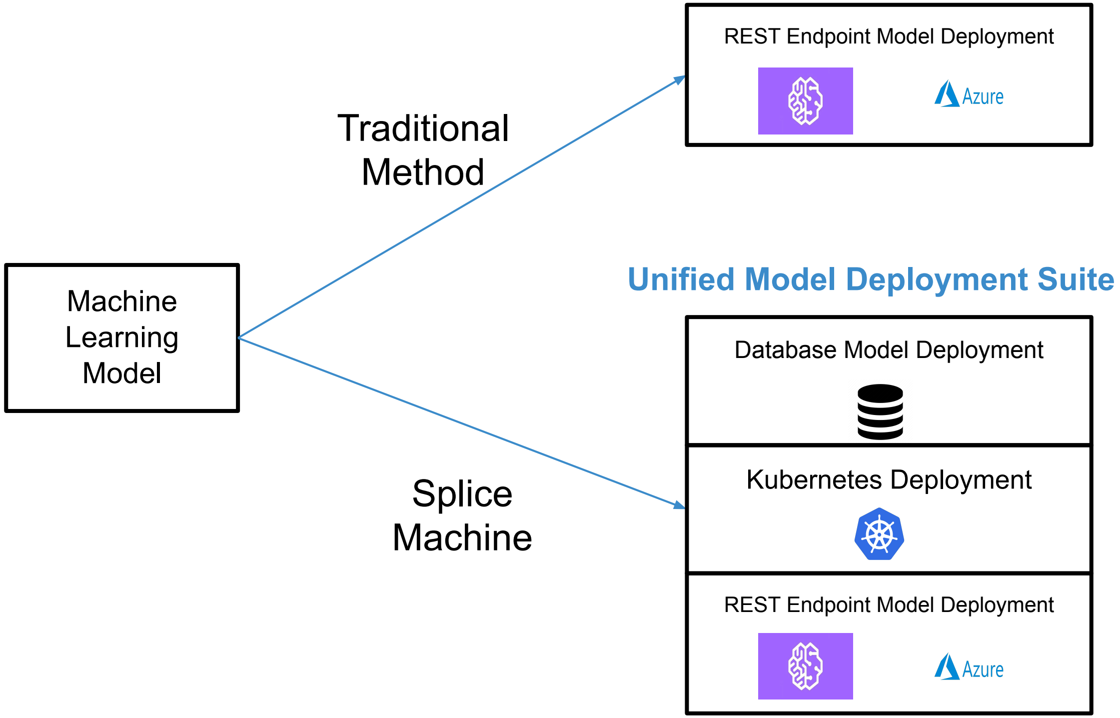


Feature Set- Two Tables in a Single Database



MLOps

- 1-line deployment
- Deploy models to database
- Features and prediction are memorialized



Database Deployment

- New records automatically trigger predictions
- Single click - low code deployment
- Easy model governance and traceability via SQL in an “evaluation store”
- Sub-millisecond predictions and scalable
- No extra endpoint programming

User	TS	RunID	F1	F2	F3	Pred
Jack	2020-09-01	4400	5.2	-12	15.4	Fraud
John	2020-09-01	4401	9.4	4	2.3	No Fraud
Sarah	2020-09-25	4402	2.3	7	1.1	Fraud
Steve	2020-10-12	4403	1.3	0	3.4	...

Prediction made and populated at sub-millisecond speed

Benefits

Data Preparation | Feature Engineering | Experimentation | Deployment

Data Preparation

Automated Aggregation Pipelines

ffs.get_feature_set_from _source

Feature Engineering

Feature Reuse

ffs.serach_feature_store

Experimentation

Automated Training Set

fs.get_training_set_from_view

Deployment

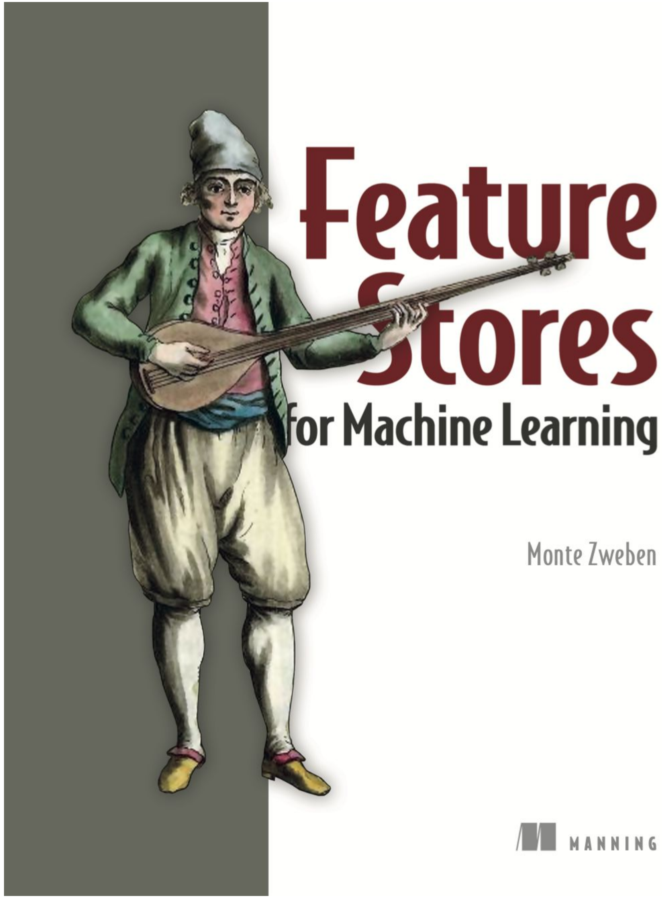
Database Deployment

ml.deploy_model

100X Faster

Coming in 2021

- Comprehensive how-to text
- Hands-on exercises
- Best practices



Summary

Improve Productivity

Feature engineering is streamlined and automated

Predictive Accuracy

Training and serving pipelines are consistent and repeatable

Model Governance

Full transparency

Next Steps

Get Demo: https://www.splicemachine.com

Free Trial: https://cloud.splicemachine.io/login

We are hiring!

Q&A