

Databricks on Google Cloud を活用した データエンジニアリング、 データサイエンス、分析

サンプルコード、Notebook を含むユースケース付き



目次

はじめに	3
本 eBook の概要	4
対象とする読者	4
データエンジニアリングのユースケース：ローンデータの精査と分析	5
Databricks on Google Cloud の設定	5
データセット	6
Delta Lake on Google Cloud Storage を使った ETL パイプラインの構築	7
Databricks でデータを分析する	7
BigQuery でデータを読み込み、Looker で視覚化する	8
Notebook	9
データサイエンスのユースケース：自転車での移動時間の予測	17
Databricks on Google Cloud の設定	17
データセット	18
マネージド ML フローによる実験追跡の自動化	19
MLflow モデルのデプロイメント	19
Notebook	20

はじめに

企業がデータや AI を活用したイノベーションに苦戦しているのは、アーキテクチャ、ツール、インフラが複雑すぎるためです。多様な種類のデータや、データサイエンス、機械学習、分析のさまざまなユースケースによって、複数のバラバラなシステムができあがり、メンテナンスや増え続けるビジネス要件への対応が困難になっています。

Databricksは、全てのデータ、分析、AI のワークロードを統合する単一のレイクハウスプラットフォームを提供することで、これらの問題を解決します。レイクハウスプラットフォームは、データレイクとデータウェアハウスの最良の要素を組み合わせたもので、データウェアハウスで一般的に見られるデータ管理とパフォーマンスを、データレイクで提供される低コストで柔軟なオブジェクトストアで実現します。この統合されたプラットフォームは、従来、分析、データサイエンス、機械学習を分断していたデータサイロを排除することで、データアーキテクチャを簡素化します。また、オープンソースとオープンスタンダードに基づいて構築されているため、柔軟性を最大限に高めることができます。

Google Cloud と連携することで、Databricks は Google Cloud のグローバルな規模でプラットフォームを提供できます。Databricks on Google Cloud は、データエンジニアリング、データサイエンス、分析をオープンなレイクハウスアーキテクチャで統一することで、組織のデータと AI を簡素化します。Google Kubernetes Engine (GKE) を活用することで、Databricks on Google Cloud は、完全にコンテナ化されたクラウド環境で Databricks を展開することを初めて可能にしました。Google Cloud Storage、BigQuery、および Google Cloud AI Platform との緊密な統合により、Databricks は Google Cloud 上のデータおよび AI サービスでシームレスに動作します。

本 eBook の概要

この eBook では、Databricks のコラボレーティブ Notebook を使ったデータエンジニアリングとデータサイエンスの 2 つの実践的なシナリオを取り上げています。Databricks Notebook は、Python、R、SQL、Scala をネイティブにサポートするインタラクティブなワークスペースであり、ユーザーは任意の言語やライブラリを使って共同作業を行い、知見の発見、視覚化、共有を行うことができます。Google Cloud 上の Databricks を実際に体験していただくために、コードサンプルを掲載しています。

対象とする読者

この eBook は、Databricks と Google Cloud との緊密な連携を利用したデータパイプラインの構築や機械学習モデルの展開に関心のあるデータエンジニアやデータサイエンティストを主な対象として書かれています。

データエンジニアリングのユースケース： ローンデータの精査と分析

このユースケースでは、データレイクに信頼性、セキュリティ、パフォーマンスを提供する Delta Lake を使って、オープンなレイクハウスを構築します。私たちの目標は、Databricks でデータを精査・分析し、それを即座にクエリとして利用できるようにして知見を導き出すことです。この例では、米国の州ごとにローンのステータスを集約して生成し、国内のローンの現在の分布を特定します。

Databricks on Google Cloud の設定

Databricks on Google Cloud の設定が済んでいない場合は、以下のドキュメントを参照して設定してください。

[Databricks on Google Cloud のアカウントの設定](#)

[Databricks で Google Cloud Storage \(GCS\) のテーブルにアクセスする](#)

[Databricks から BigQuery への接続](#)

[Databricks での Looker の使用](#)

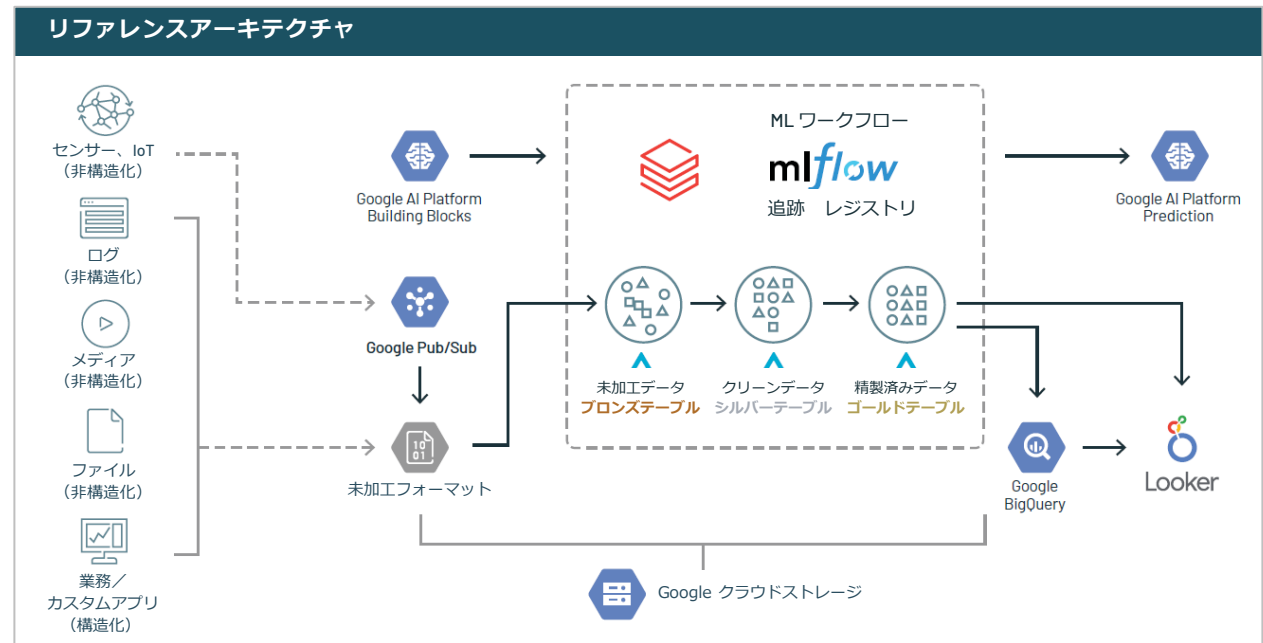
データセット

LendingClub から公開されているローンデータセットを使用します。このデータには、申込者のローン情報と、完済、遅延、デフォルトなどのローン状況が含まれています。このデータセットを Delta Lake で精査し、ローンのステータスに変更があった場合には更新し、州とローンのステータスごとに集計して、どの州で最もローンの記録が多いかを確認し、最後にダッシュボードを作成して組織全体のビジネスユーザーと共有したいと考えています。

公開データセット：www.kaggle.com/wordsforthewise/lending-club

Databricks のワークスペースでも提供されています。

dbfs:/databricks-datasets/samples/lending_club/parquet/



Delta Lake on Google Cloud Storage を使った ETL パイプラインの構築

このデータエンジニアリングのユースケースでは、Google Cloud Storage 上の Delta Lake を使って ETL パイプラインを構築します。Delta Lake は、オープンフォーマットのストレージレイヤーで、データレイクの信頼性、セキュリティ、パフォーマンスを提供します。ストリーミングやバッチ処理を行うことができます。私たちは、データエンジニアリングのパイプラインにおいて、異なる品質レベルに対応するテーブルを使用し、データに構造を徐々に追加していく共通のアーキテクチャを採用しています。「ブロンズテーブル」はデータ取り込み用、「シルバーテーブル」は変換/特徴量エンジニアリング用、「ゴールドテーブル」は機械学習のトレーニングや予測、あるいはビジネスレベルのデータを集約したものです。このアーキテクチャにより、データエンジニアは、未加工データを「単一のデータソース」として、ダウンストリームに流れるパイプラインを構築できます。

Delta Lake の詳細については、[ドキュメント](#)をご覧ください。

Databricks でデータを分析する

シルバーとゴールドのテーブルを使って、Databricks Notebook の中でデータ分析を行うことができます。Delta Lake は、UPDATE、DELETE、MERGE を含むデータ操作言語 (DML) コマンドをサポートしています。UPDATE 操作は、フィルタリング条件に合致する行を選択的に更新するために使用します。この例では、UPDATE を使用して、選択された数のローンの遅延支払額のみを更新します。

SQL クエリを使用して Delta Lake のデータを分析し、Redash ダッシュボードを即座に構築してデータの傾向を把握できます。連結されたゴールドテーブルの基礎となるソースが Delta Lake テーブルであるため、表示されるビューはリアルタイムで更新されます。Spark SQL を使用しているため、このデータに対して大規模な集計クエリを実行できます。

BigQuery でデータを読み込み、Looker で視覚化する

ユーザーは、Delta テーブルから BigQuery のデータを読み込んで、BI や視覚化に活用できます。Databricks には、Google BigQuery との最適化されたコネクタがあり、BigQuery のデータに Storage API を介して直接アクセスし、高性能なクエリを実行できます。このコネクタは、追加の述語プッシュダウン、名前付きテーブルやビューへのクエリ、BigQuery 上での直接の SQL 実行と Apache Spark™ の DataFrame への結果のロードをサポートしています。

その後、Looker を使用して Databricks と BigQuery の両方からデータを取り出し、BI やレポート作成を行います。Looker は Databricks と統合されており、ユーザーは、新しい視覚化エクスペリエンスによって、データレイクに対して直接クエリを実行できます。

Notebook

- ローンデータセットを Google Cloud Storage から Delta ブロンズテーブルに読み込む
- ブロンズテーブルを精製し、Delta シルバーテーブルに書き込む
- データを集約して Delta ゴールドテーブルに書き込み、BigQuery にプッシュ
- 州ごとのローンの分布を BigQuery で分析
- Delta テーブルと BigQuery テーブルを結合し、Looker でデータを視覚化

Databricks の[セットアップスクリプト](#)で [Notebook](#) をお試しください。

STEP 1 : Python の設定スクリプトを実行する

Python の設定スクリプトを実行します。これらのスクリプト*は、ローン状況の Parquet データセットから未加工の CSV ファイルをキュレートし、Delta Lake パイプラインの構築に必要な一時テーブルを作成します。

*gcp_launch_event という名前の GCS インスタンスがあることを前提としています。

```
from pyspark.sql.functions import *
from pyspark.sql.types import StructType, StructField, StringType, IntegerType

# Grab the parquet dataset that comes in the sample dataset path of a Databrick workspace
lspq_path = "dbfs:/databricks-datasets/samples/lending_club/parquet/"

# Read loanstats_2012_2017.parquet
data = spark.read.parquet(lspq_path)

# Reduce the amount of data (to run on Community Edition)
(loan_stats, loan_stats_rest) = data.randomSplit([0.10, 0.90], seed=123)

#Update with your own GCS path
loan_stats.write.format("csv").save("gs://gcp_launch_event/raw/", header='true')

csv_path = dbutils.fs.ls("gs://gcp_launch_event/raw/")[5].path
df = spark.read.option("header", True).option("inferSchema", True).csv(csv_path)
csv_schema = df.schema

df_evolution = df.filter("addr_state == 'VT']").limit(5)
df_evolution.select("addr_state", "loan_status", "grade").createOrReplaceTempView("new_records")

data = [("CA", "Late (16-30 days)", 79),
        ("CA", "Fully Paid", 7992)]
]
```

```

schema = StructType([ \
    StructField("addr_state",StringType(),True), \
    StructField("loan_status",StringType(),True), \
    StructField("count",IntegerType(),True)
])
df_merge = spark.createDataFrame(data=data,schema=schema).createOrReplaceTempView("merge_records")

# Optionally insert your own database name below
sql("""create database gcp_launch_event""")
sql("""use gcp_launch_event""")

```

STEP 2 : ローンの状態ステータスを取り込んで Delta ブロンズテーブルに書き込む

```

df_csv = spark.read.schema(csv_schema).option("header", True).csv("gs://gcp_launch_event/raw/")
df_csv.write.format("delta").saveAsTable("loanstats_delta_bronze")
display(table("loanstats_delta_bronze"))

```

Delta ブロンズテーブルスニペットは以下のようになり、ローン申請者の情報が表示されているはずですが。

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_grade
1	null	null	1000	1000	1000	36 months	5.32%	30.12	A	A1
2	null	null	1000	1000	1000	36 months	5.32%	30.12	A	A1
3	null	null	1000	1000	1000	36 months	5.32%	30.12	A	A1
4	null	null	1000	1000	1000	36 months	6.00%	30.42	D	D1
5	null	null	1000	1000	1000	36 months	6.49%	30.65	A	A2
6	null	null	1000	1000	1000	36 months	6.49%	30.65	A	A2

STEP 3 : 新しい Delta シルバーテーブルを作成する

NULL 値や、不要な列がたくさんあります。"addr_state" と "loan_status" の 2 つの列以外を削除して、SQL を使って新しい Delta シルバーテーブルを作成してみましょう。Python のセルを SQL のセルに変換するには、セルの中に %sql [マジックコマンド](#)を追加する必要があります。

※ 以下はいつでも PySpark で実行できます。

%sql

```
CREATE TABLE loanstats_delta_silver
USING delta
AS
  SELECT addr_state, loan_status
  FROM loanstats_delta_bronze
  WHERE addr_state is not null;

SELECT * FROM loanstats_delta_silver;
```

簡略化したシルバーテーブルは以下のようになります。

	addr_state ▲	loan_status ▲
1	MN	Late (31-120 days)
2	CA	Current
3	NJ	Current
4	TX	Current
5	IL	Charged Off
6	MI	Current

STEP 4 : データにグレードをつけてローン申請者のリスクを判断する

ローン審査担当者がこのデータに「グレード」をつけて、ローン申請者のリスクを判断してみます。テーブルに「グレード」という列を追加して、Python でシルバーテーブルのスキーマを進化させます。

```
(table("new_records")
  .write
  .format("delta")
  .option("mergeSchema", True) #Add merge schema option
  .mode("append")
  .saveAsTable("loanstats_delta_silver"))

display(table("loanstats_delta_silver").filter("addr_state == 'VT'").
  orderBy(col("grade").desc()).limit(20))
```

最終的なシルバーテーブルは以下のようになるはずです。

	addr_state ▲	loan_status ▲	grade ▲
1	VT	Fully Paid	D
2	VT	Late (31-120 days)	C
3	VT	Current	C
4	VT	Fully Paid	C
5	VT	Fully Paid	B
6	VT	Charged Off	null

グレードの列がなかったレコードには、その列に NULL 値が入っていることがわかります。

※ Delta Lake はデフォルトで **スキーマを適用** します。mergeSchema オプションを追加しないと、スキーマの不一致のために追加操作が失敗します。

STEP 5 : SQL を使ってローン状況タイプの分布を見る

ローン状況タイプの州別分布を見てみましょう。最終的にゴールドテーブルを作成し、州とローン状況でレコードをグループ化して集計し、SQL を使ってカウントを実行します。

```
CREATE TABLE loanstats_delta_gold
USING delta
AS
SELECT addr_state, loan_status, count(*) AS count FROM loanstats_delta_silver
GROUP BY addr_state, loan_status;

SELECT * FROM loanstats_delta_gold WHERE addr_state = 'CA' ORDER BY loan_status;
```

カリフォルニア州のレコードだけを見てみましょう。addr_state = CA でフィルタリングされたゴールドテーブルは、以下のようになります。

	addr_state ▲	loan_status ▲	count ▲
1	CA	Charged Off	1896
2	CA	Current	10046
3	CA	Default	1
4	CA	Fully Paid	7990
5	CA	In Grace Period	165
6	CA	Late (16-30 days)	81

16~30 日支払いが遅れた 2 つのローンを含む複数のローンが完済されたという通知を受け取ったため、Delta テーブルを UPDATE する必要があります。次の SQL コマンドを実行して、カウントを更新します。

```
MERGE INTO loanstats_delta_gold g
USING merge_records m
ON g.addr_state = m.addr_state AND g.loan_status = m.loan_status
WHEN MATCHED THEN UPDATE SET *;

SELECT * FROM loanstats_delta_gold WHERE addr_state = 'CA';
```

これでゴールドテーブルが新しい数で更新されるはずですが。

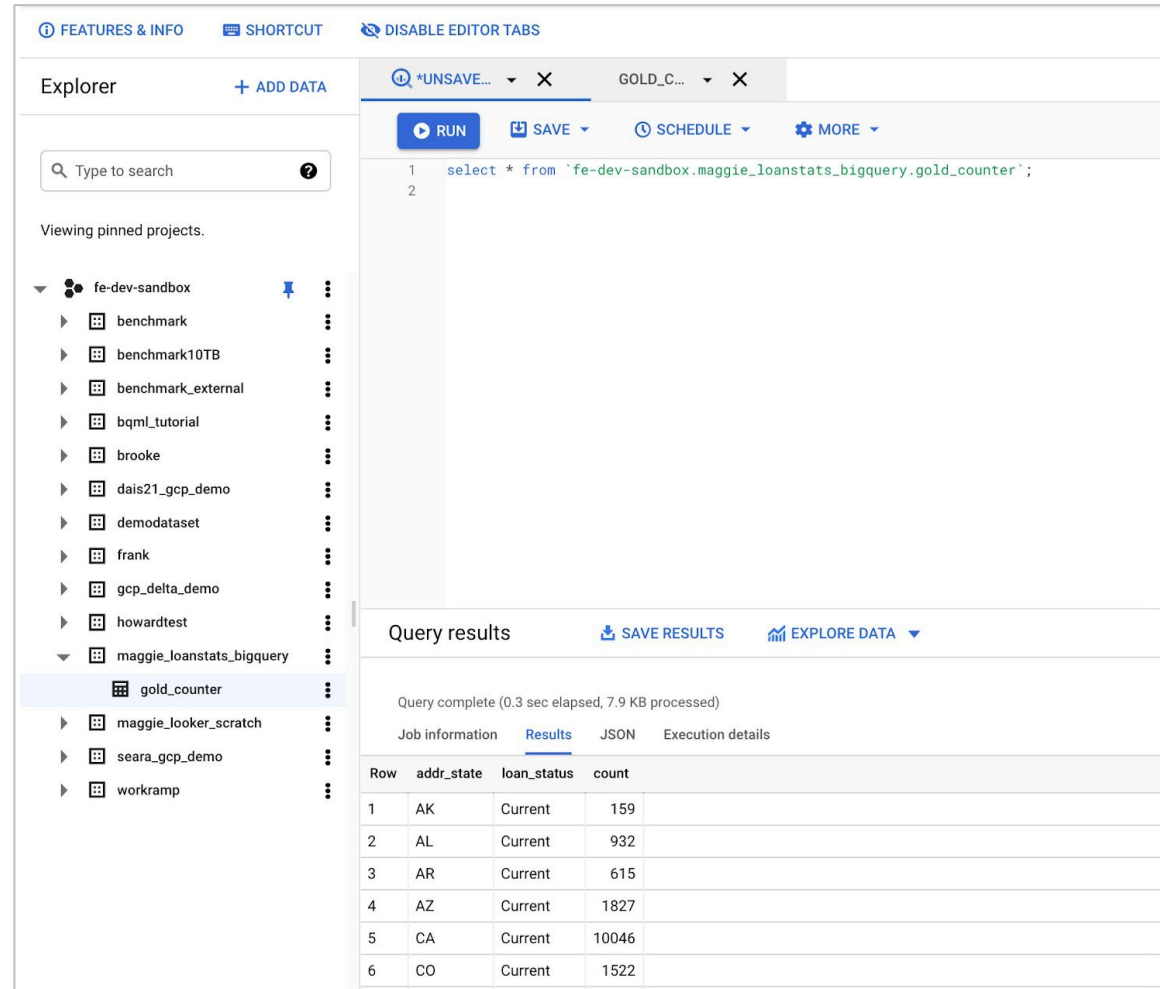
	addr_state ▲	loan_status ▲	count ▲
1	CA	Charged Off	1896
2	CA	Current	10046
3	CA	Default	1
4	CA	Fully Paid	7992
5	CA	In Grace Period	165
6	CA	Late (16-30 days)	79

STEP 6 : BigQuery コネクタを使ってゴールドテーブルを書き込む

BigQuery をデータソースとして使用しているアナリストとゴールドテーブルを共有することもできます。ビルトインの BigQuery コネクタを使って、Python で `Fe-dev-sandbox.loanstats_bigquery.gold_counter` という名前の BigQuery テーブルにゴールドテーブルを書き込んでみましょう。

```
(table("loanstats_delta_gold").write
  .format("bigquery")
  .mode("overwrite")
  .option("temporaryGcsBucket", "gcp_launch_event") #Create a temporary GCS bucket
  .option("table", "fe-dev-sandbox.loanstats_bigquery.gold_counter") #Write into a BigQuery table
  .save())
```

BigQuery テーブル `fe-dev-sandbox.loanstats_bigquery.gold_counter` は以下のようになります。



The screenshot displays the Databricks workspace interface. On the left, the 'Explorer' sidebar shows a tree view of projects under 'fe-dev-sandbox', with 'gold_counter' selected under 'maggie_loanstats_bigquery'. The main editor area contains a SQL query: `select * from `fe-dev-sandbox.maggie_loanstats_bigquery.gold_counter`;`. Below the query, the 'Query results' section shows a table with 6 rows and 4 columns: 'addr_state', 'loan_status', and 'count'. The table data is as follows:

Row	addr_state	loan_status	count
1	AK	Current	159
2	AL	Current	932
3	AR	Current	615
4	AZ	Current	1827
5	CA	Current	10046
6	CO	Current	1522

STEP 7 : BigQuery テーブルを Looker ダッシュボードに取り込んでデータを視覚化する

アナリストはビジネスレベルのレポートも作成し、その知見を共有する必要があるでしょう。この BigQuery テーブルを Looker ダッシュボードに取り込み、データを視覚化してみます。ボーナス : [Looker を Delta テーブルに直接接続](#)して、シルバーまたはブロンズレイヤーを視覚化します。

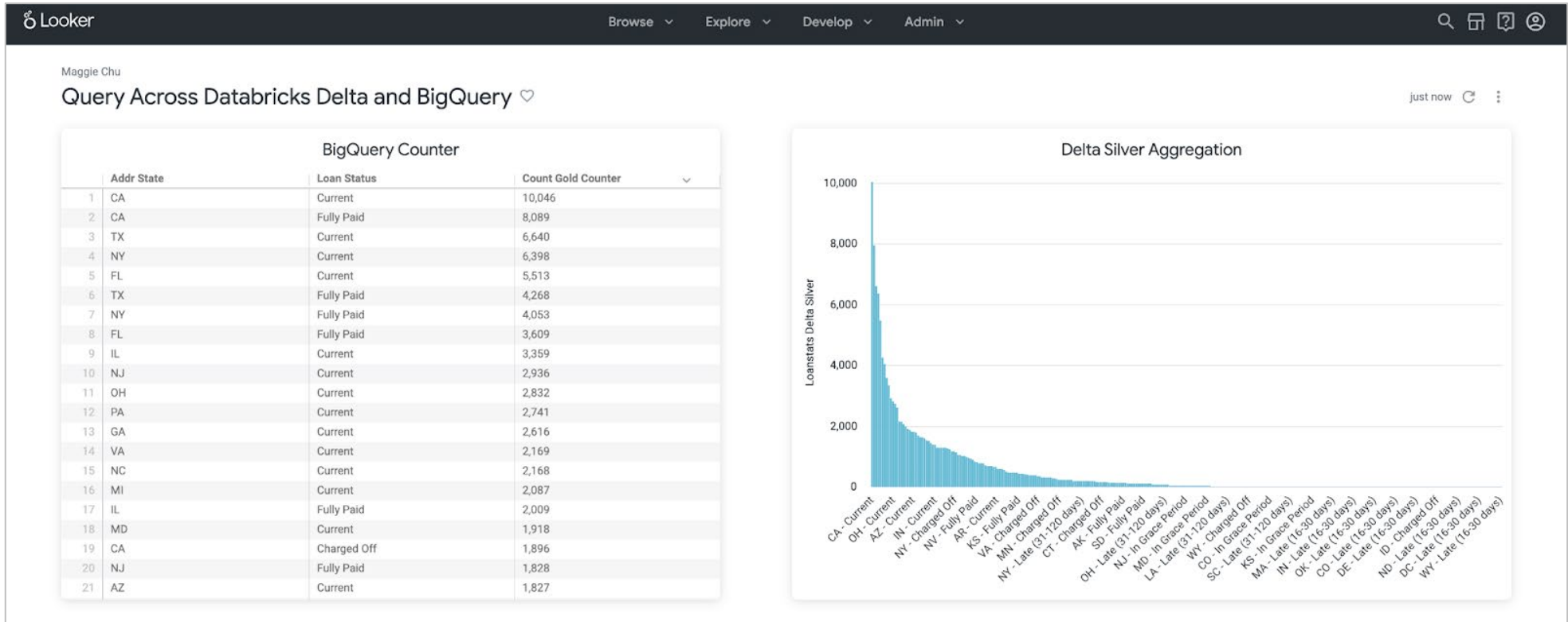


図 1: Looker のダッシュボード

データサイエンスのユースケース：自転車での移動時間の予測

Databricks Machine Learning は、実験の追跡、モデルのトレーニング、特徴量の開発と管理、および特徴量とモデルの提供のためのマネージドサービスを組み込んだ、統合されたエンドツーエンドの機械学習環境です。このユースケースでは、特徴量テーブルを作成し、モデルのトレーニングと推論のためにアクセスし、モデルレジストリを使用してモデルを共有、管理、提供し、最後にモデルを展開して予測を生成します。

この例では、機械学習を利用して、サブスクリプションタイプや時間帯などの特徴を考慮して、2つのステーション間の自転車旅行にかかる時間を予測します。

Databricks on Google Cloud の設定

Databricks on Google Cloud の設定が済んでいない場合は、以下のドキュメントを参照して設定してください。

[Databricks on Google Cloud のアカウントの設定](#)

[MLflow で機械学習に Databricks Runtime を使用する](#)

[Databricks から BigQuery への接続](#)

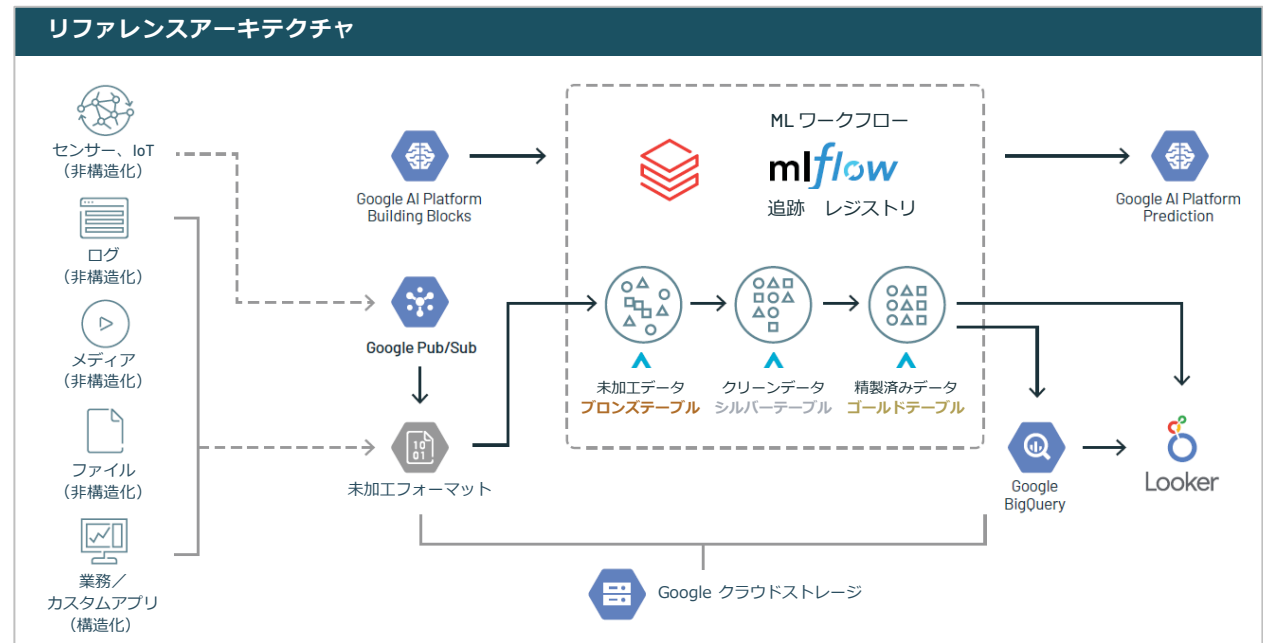
[モデルレジストリへの MLflow モデルの追加](#)

データセット

テキサス州オースティン市で公開されている自転車シェアリングのデータセットを使用して、同市の自転車利用率を予測します。このデータセットには、サブスクリプションタイプ、トリップ ID、スタートとエンドのステーションの位置、スタートとエンドの乗車時間などの情報が含まれています。しかし、ステーションの中には閉鎖されたものもあります。アクティブなステーションの最新情報を得るためには、これを bikeshare_stations のデータセットと結合する必要があります。これらのデータセットは、以下のリンクから入手できます。

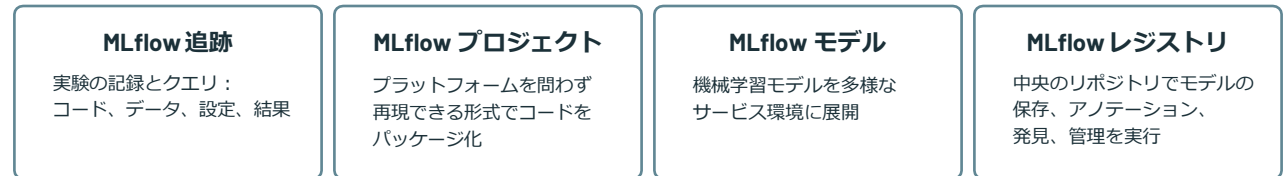
公開データセット：www.kaggle.com/jboysen/austin-bike

[Databricks Notebook](#) でも提供しています。



マネージド ML フローによる実験追跡の自動化

Databricks のデータサイエンスと機械学習の連携機能を利用して、実験から本番までの ML ライフサイクルを管理しています。特に、機械学習のライフサイクルを管理するために、Databricks が開発したオープンソースツール [MLflow](#) を活用しています。Databricks は、MLflow のマネージドサービスを提供しており、MLflow には、以下の機能が組み込まれています。Databricks ワークスペースとの統合で、機械学習実験をセキュアに共有、バージョン、管理、再現できます。



MLflow モデルのデプロイメント

Databricks は、MLflow の `spark_udf` を使って、ML モデルをバッチまたはストリーミングでデプロイするワークフローを簡素化します。リアルタイムに提供するには、Databricks MLflow モデルサービングを使用すると、モデルのデプロイと更新を簡単に行うことができます。Databricks の MLflow モデルサービングは、MLflow モデルレジストリと直接連携し、モデルの新バージョンを自動的にデプロイし、リクエストをルーティングします。モデルレジストリは、全ての機械学習ライブラリ（TensorFlow、scikit-learn など）のモデルを格納することができ、モデルの複数のバージョンを格納し、それらをレビューし、ステージングやプロダクションなどの異なるライフサイクルステージに昇格させることができます。MLflow モデルレジストリでモデルを構築してバージョン管理を行った後は、推論を BigQuery テーブルに書き込むことも可能です。

Notebook

- BigQuery からのデータの読み込み
 - Apache Spark を特徴量の前処理に使用
- sklearn モデルを構築し、MLflow でパラメータ、メトリクス、アーティファクトなどを自動的に追跡する
 - MLflow モデルレジストリによるモデルのバージョン管理
- モデルのデプロイ
 - BigQuery テーブルに推論を書き込む

Databricks の [Notebook](#) をお試しください。

STEP 1 : 特徴量の前処理に Apache Spark を使用する

まず、Apache Spark を使って、2 台のバイクを共有する DataFrames の結合、カラムの選択、レコードのフィルタリング、NULL レコードの削除など、機能の前処理を分散して行います。

この例では、Databricks に組み込まれた視覚化により、オースティンでの自転車乗車の開始時間などの傾向を見ることができます。

以下のコードスニペットを見ると、2 つの DataFrames を結合してからフィルタリングしていることがわかります。フィルタリングしてから結合した方が速いのでは？と疑問に思うかもしれませんが、これこそが Spark Catalyst Optimizer の機能です。Spark の UI を見てみると、フィルタを結合の前にプッシュしていることがわかります。フィルタリングしてから結合するようにコードを変更することはベストプラクティスではありませんが、Catalyst Optimizer は最終結果を変えないことを保証したうえで、クエリの実行順序を変更できます。

データセットの結合と視覚化

```
stations_df = spark.read.format("bigquery").option("table",
"bigquery-public-data.austin_bikeshare.bikeshare_stations").load()
spark_df = (trip_df.
    .join(stations_df, on=trips_df.start_station_name==stations_df.name)
    .selectExpr("subscriber_type", "dayofweek(start_time) as day_of_week",
"hour(start_time) as start_hour", "start_station_id", "end_station_id", "duration_minutes")
    .filter("duration_minutes > 0 AND duration_minutes < 180")
    .filter("status = 'active'")
    .dropna())
display(spark_df)
```

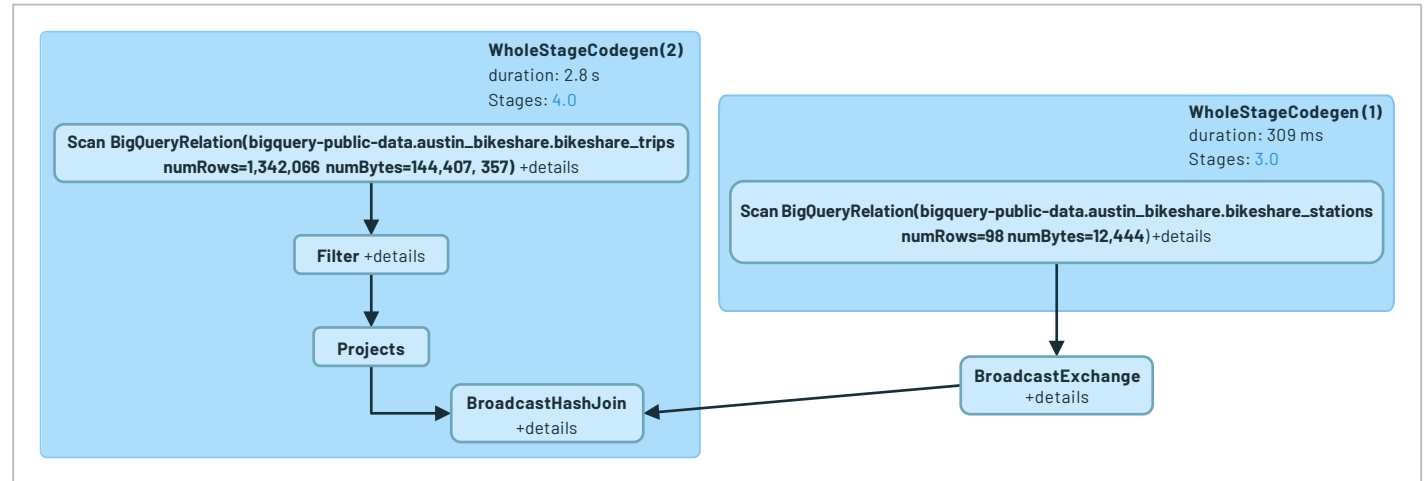


図2 : Spark UI のクエリプラン

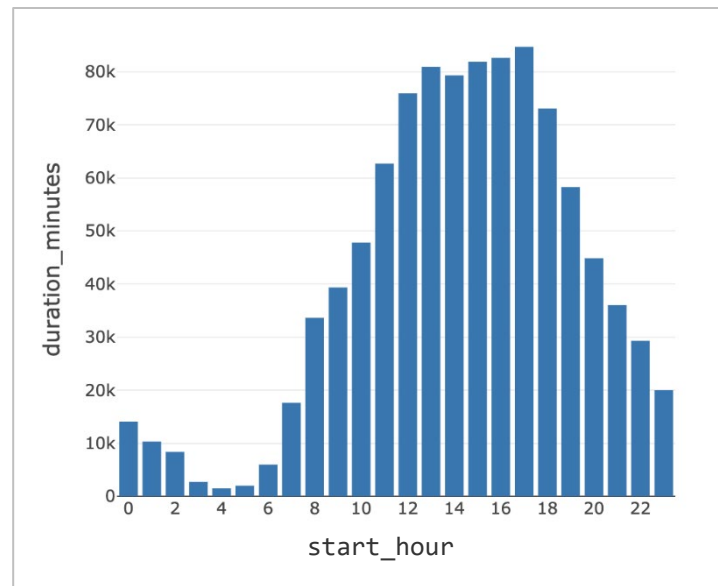


図3 : 自転車での移動開始時間の分布

STEP 2 : Spark DataFrame を pandas DataFrame に変換し、処理したデータセットで sklearn モデルを構築

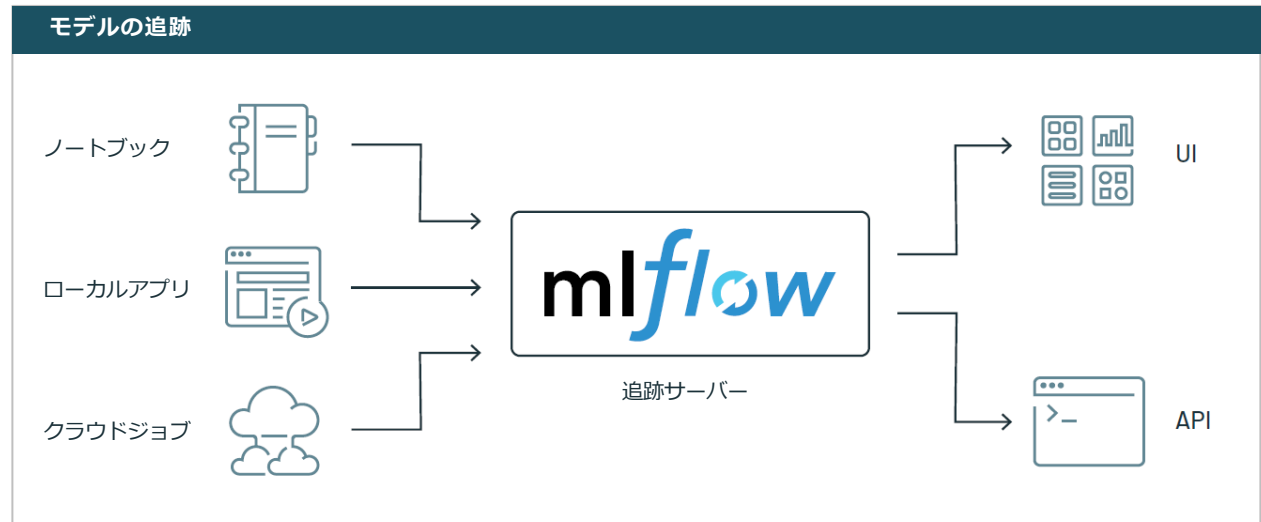
今回の例では、特徴量から旅行の所要時間（分）を予測する線形回帰モデルを構築します。カテゴリカルな特徴については、ホットエンコードを行います。しかし、モデルをデータセットにフィットさせる前に、調整を行うことにします。MLflow をインポートし、sklearn のオートロギング機能を有効にして、実験を MLflow の追跡サーバーに自動的に追跡します。その後、UI や API を介して MLflow 追跡サーバーから実験結果をクエリできます。

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
import mlflow
df = spark_df.toPandas()
X = df.drop(columns=["duration_minutes"])
y = df.duration_minutes

cat_features = ["subscriber_type", "day_of_week", "start_station_id", "end_station_id"]

ohe = OneHotEncoder(handle_unknown="ignore")
preprocessor = ColumnTransformer(transformers=[("ohe", ohe, cat_features)])
lr = LinearRegression()

pipeline = Pipeline([
    ("preprocessor", preprocessor),
    ("lr", lr)
])
mlflow.sklearn.autolog()
pipeline.fit(X,y)
```



STEP3 : MLflow モデルを中央のモデルレジストリに登録する

MLflow の追跡 API を使って、この Notebook で開発した sklearn モデルのパラメータ、メトリクス、モデルなどを自動的に追跡します。そして、このモデルをモデルレジストリに登録して、バージョン管理を行い、発見しやすくします。MLflow のモデルレジストリでは、保留中のリクエストやモデルに対して行われたアクティビティの記録が可能のため、モデルの全行程を把握できます。sklearn モデルなのか TensorFlow モデルなのかを指定する必要はなく、MLflow が自動的に追跡してくれます。

```
logged_model = "models:/Austin_Bike_Share/Staging"  
loaded_model = mlflow.pyfunc.spark_udf(spark, model_uri=logged_model)
```

STEP 4 : モデルをバッチ推論用にデプロイし、予測値を BigQuery に書き込む

モデルをステージングに移動させたら、Spark DataFrame に適用してみましょう。

```
features = spark_df.drop("duration_minutes").columns
pred_df = spark_df.withColumn("prediction", loaded_model(*features))
display(pred_df)
```

予測は以下のようなになるでしょう。

	subscriber_type	day_of_week	start_hour	start_station_id	end_station_id	duration_minutes	prediction
1	Founding Member (Austin B-cycle)	6	11	2823	2823	16	22.286556168692155
2	Annual Membership (Austin B-cycle)	6	13	2823	2823	23	22.530732982991914
3	Annual Membership (Austin B-cycle)	6	15	2823	2823	4	22.530732982991914
4	Annual Membership (Austin B-cycle)	6	16	2823	2536	10	10.751374364991559
5	Weekender	5	19	2568	2536	3	14.562198063999087
6	Weekender	5	8	2568	2536	11	14.562198063999087

結果が出たら、予測結果を BigQuery に書き込むという方法もあります。データエンジニアリングのユースケースで述べたように、Databricks には Google BigQuery との最適化されたコネクタがあり、Storage API を介して BigQuery のデータに直接容易にアクセスできます。

```
(pred_df
 .write
 .format("bigquery")
 .mode("overwrite")
 .option("temporaryGcsBucket", "brooke-gcp-demo")
 .option("table", "fe-dev-sandbox.brooke.austin_bike_pred_df")
 .save())
```


オプション : Google Cloud AI Platform に展開してリアルタイムで推論

Spark を活用したバッチまたはストリーミング推論に加えて、Google Cloud AI Platform に展開してリアルタイム推論を行うことも可能です。Google Cloud の AI Platform は、データサイエンスと機械学習のための完全に管理されたエンドツーエンドのプラットフォームであり、AutoML を使ったポイント&クリック型のデータサイエンスや、高度なモデル最適化のためのツールを提供します。AI Platform は、データサイエンティストや ML エンジニアが、大規模なモデルのアイデア出し、実験、デプロイメント、管理を通じて生産性を高めることができます。また、モデルのデプロイとモデルのモニタリングのために、低遅延で自動スケーリング可能な Kubernetes ベースのサービングインフラを提供します。

AI プラットフォームにモデルを展開したら、サンプルレコードを渡して予測値を生成できます。ここでは、契約タイプ、曜日、時間帯、開始駅 ID、終了駅 ID を設定しています。早速、予測値を生成してみましょう。

```
{
  "instances": [
    ["Weekender", 7, 13, 2568, 2536],
    ["24 Hour Walk Up", 4, 9, 3293, 2576],
    ["Annual", 5, 11, 2823, 2823]
  ]
}
```

結果 :

```
{
  "predictions": [
    23.360899363705965,
    20.397530185653874,
    21.56391079297561
  ]
}
```

もし、週末のみの利用者に年間サブスクリプションを提案できたらどうなるでしょうか？彼らはより頻繁に乗車するようになり、同じ移動でも予想移動時間を5分短縮できるでしょう。AIプラットフォームであなたの予測モデルを試してみてください。

```
{
  "instances": [
    ["Annual", 7, 13, 2568, 2536],
    ["24 Hour Walk Up", 4, 9, 3293, 2576],
    ["Annual", 5, 11, 2823, 2823]
  ]
}
```

結果

```
{
  "predictions": [
    18.1635271025103,
    20.397530185653874,
    21.56391079297561
  ]
}
```



データブリックスについて

データブリックスは、米国サンフランシスコに本社を置き、世界中に拠点を持つデータとAIの企業です。Apache Spark、Delta Lake、MLflowのオリジナル開発メンバーによる創業以来、データブリックスは、データの活用によって難題解決に挑む組織の支援に取り組んでいます。データブリックスのレイクハウスプラットフォームは、コムキャスト、コンデナスト、H&Mをはじめ、フォーチュン500企業の40%を含むさまざまな業界の5,000社以上におけるデータ、分析、AIの取り組みに活用されています。

[Twitter](#)、[LinkedIn](#)、[Facebook](#)での情報発信も行っております。ぜひご覧ください。

[Google Cloud on Databricksの無料トライアル](#)をご利用いただけます。