ML Enablement Series 【ML-Dark-08】

Amazon SageMaker モニタリング Part2

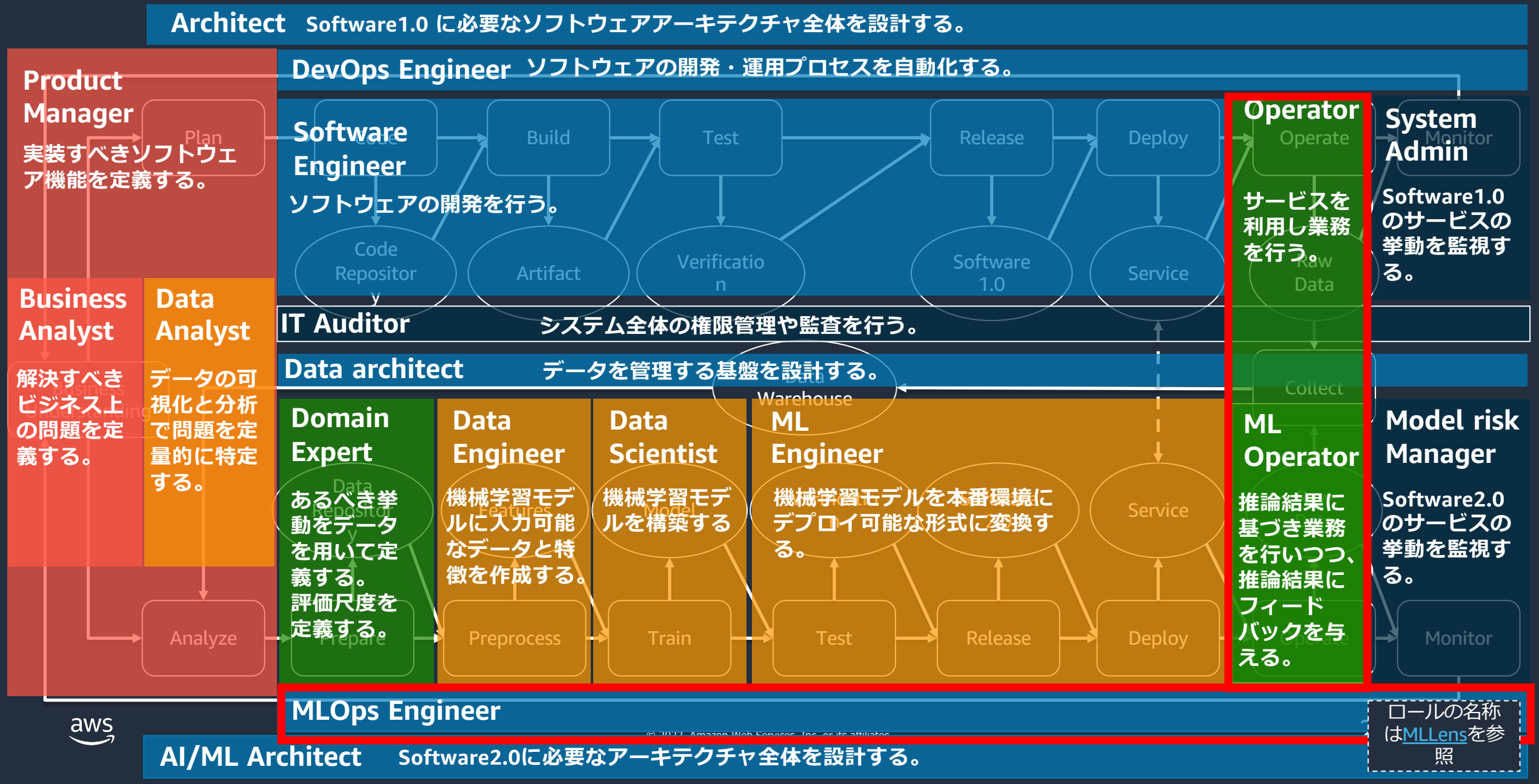
データと推論結果の変化に気づく

Masafumi Otsuki

Professional Services

2022/12

モニタリングしフィードバックしたり、される人向け動画です



SageMaker Model Monitorに関する動画公開予定

1. Amazon SageMaker Model Monitorを理解しよう

   公開済み https://www.youtube.com/watch?v=Q-vTO1_QjMs

2. Amazon SageMaker Model Monitor で
   データと推論結果の変化に気づく

   本動画

3. Amazon SageMaker Model Monitor で
   モデルバイアス、Feature Attributionをモニタしよう(仮)

この動画のゴール

2. Amazon SageMaker Model Monitor で
データと推論結果の変化に気づく

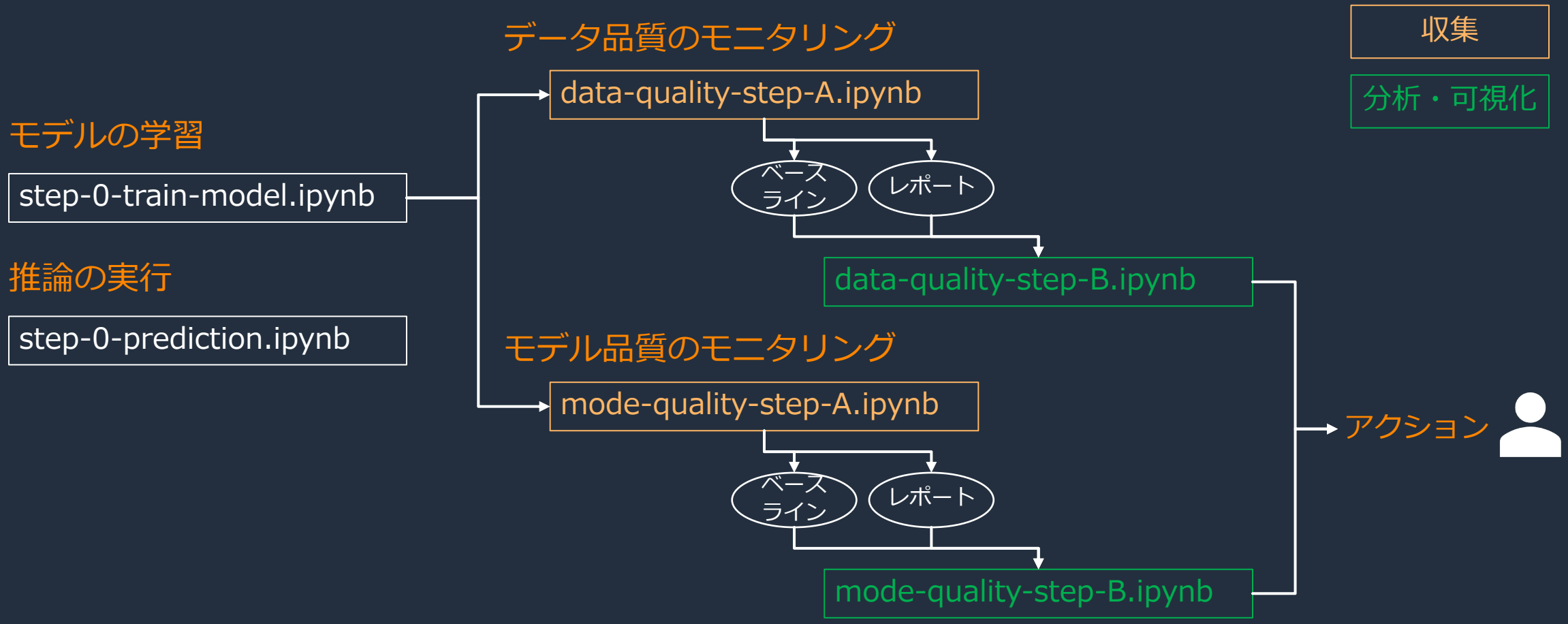
Amazon SageMaker を利用したデータ品質やモデル品質のモニタリング手順を理解し、セッション終了後にご自身の環境に適用できるようになること

解説すること

1. Amazon SageMakerによるデータ品質のモニタリング
2. Amazon SageMakerによるモデル品質のモニタリング

この動画に記載のコードはGitHubにて公開

https://github.com/aws-samples/aws-ml-jp/tree/main/sagemaker/sagemaker-model-monitor/black-belt-part2



この動画で利用するデータ

https://aws.amazon.com/marketplace/pp/prodview-okyonroqg5b2u

Description
- PARQUET files containing NYC TLC trip data.

Resource type
- S3 Bucket

Amazon Resource Name (ARN)
- arn:aws:s3:::nyc-tlc

AWS Region
- us-east-1

AWS CLI Access
- aws s3 ls s3://nyc-tlc/

この動画で利用するデータ

New York City（マンハッタン島）で営業するYellow Taxiの1日あたりの乗車数推移

参考｜Amazon SageMaker Model Monitor を活用したデータドリフト検知の解説
https://aws.amazon.com/jp/blogs/news/detect-data-drift-with-amazon-sagemaker-model-monitor/

モデルをモニタリングする前に

0. モニタリングの対象を作る

0-1. MLモデルで解く問題を決める
ビジネス課題の解決に寄与するタスクを設定

0-2. データを集める
データの発生元を特定し、収集経路と
手段、データの処理方式を決める

0-3. モデルを作成する
データを整形し、問題に適した
アルゴリズムを選定し、トレーニングを行う

この動画では

（たとえば急な需要増への対応のために）
NYC Taxiの3時間後の乗降数を予測する

Open Data on AWSを利用
（収集されたデータがS3に配置されると仮定して、明細から15分間隔のサマリーデータを生成）

XGBoostを使用した予測モデルを作成
（15分間隔のサマリーから特徴量を生成し、12レコード後の乗降数を予測する回帰問題）

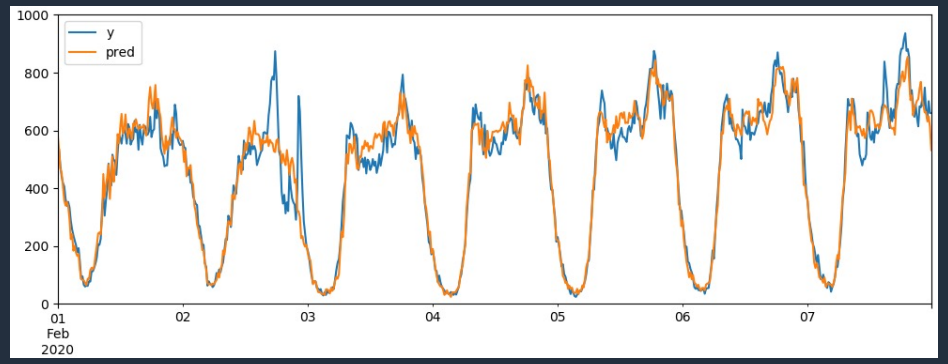
コードは step-0-train-model.ipynb にあります

MLモデルをトレーニングする

1. NYC Taxiのデータ公開用S3 Bucket（s3://nyc-tlc）からParquetファイルを取得（データ量が多いため20%にサンプリング）
2. 乗降の明細データから15分単位のサマリーに変換し、同時に特徴量を抽出
3. データを学習（2019/7～12）、検証（2020/1）、テスト（2020/2）に分割してモデルをトレーニング、optunaでハイパーパラメーターチューニングを実施
4. モデルをSageMaker Endpointにデプロイして動作を確認

テストデータ期間の一部を抜粋：

RMSE: 53.0093
MAE: 38.2950
R2: 0.949919



サンプルMLモデルで行った特徴量抽出

コードは step-0-train-model.ipynb にあります

元データ（Yellow Taxi Trip Data）

項目名	データ例	内容
VendorID	2	会社コード
tpep_pickup_datetime	2019-11-06 19:32:32	乗車した時刻
tpep_dropoff_datetime	2019-11-06 19:58:58	降車した時刻
passenger_count	1.0	乗客数
trip_distance	14.32	移動距離
RatecodeID	1.0	料金コード
store_and_fwd_flag	N	データ送出タイミング
PULocationID	132	乗車した区画ID
DOLocationID	64	降車した区画ID
payment_type	2	支払い手段
fare_amount	39.5	乗車料金
extra	1.0	追加料金
mta_tax	0.5	MTA Tax金額
tip_amount	0.0	チップ額
tolls_amount	0.0	有料道路料金
improvement_surcharge	0.3	
total_amount	41.3	合計金額
congestion_surcharge	0.0	渋滞追加料金
airport_fee	None	空港乗車の追加料金

明細から15分間隔の特徴量に変換

モデルの学習用データ

項目名	データ例	内容
pickup_count	522.0	当日の乗車数（ラベル）
history_12slots	690.0	3時間前（15分x12）の乗車数
history_16slots	432.0	4時間前（15分x16）の乗車数
history_20slots	208.0	5時間前（15分x20）の乗車数
…		
history_2016slots	502.0	約21日前の乗車数
history_2018slots	543.0	約21日前の乗車数
passenger_count_mean_12slot〜200slot	1.37〜1.56	12slot〜200slot前の乗客数平均
trip_distance_mean_12slot〜200slot	3.584	12slot〜200slot前の移動距離平均
fare_amount_mean_12slot〜200slot	13.693	12slot〜200slot前の乗車料金平均
extra_mean_12slot〜200slot	0.256	12slot〜200slot前の追加料金平均
tip_amount_mean_12slot〜200slot	1.765	12slot〜200slot前のチップ額平均
tolls_amount_mean_12slot〜200slot	0.418	12slot〜200slot前の有料道路料金平均
time_slot	0〜95	24時間を15分で刻んだ時刻スロット

Amazon SageMakerによるデータ品質のモニタリング

データ品質モニタリングで行う作業

A. モニタリング開始前にやること

A1. エンドポイントのデプロイ
データキャプチャの開始

A2. ベースラインの作成
データセットの統計量や制約、
SHAP値、バイアス指標、精度の基準生成

A3. モニタリングのスケジュール
データ品質、モデル精度、バイアス、
Feature Attributionのモニタリング設定

B. モニタリング中にやること

B1. Ground Truthの収集
Ground Truthの作成、登録、予測とのマージ

B2. モニタリング結果の分析
メトリクスの可視化、レポートの分析

B3. CloudWatchのアラート
モデル再学習・データ更新のトリガー

収集

A1. エンドポイントのデプロイ

前後を含めた完全なコードは
data-quality-step-A.ipynb にあります

モニタリングするエンドポイントのEndpoint Configにデータキャプチャの設定を追加する

```
# コンフィグの作成
create_endpoint_config_response = sm_client.create_endpoint_config(
    EndpointConfigName=endpoint_config_name,
    ProductionVariants=[{
        'InstanceType': 'ml.t2.medium',
        'InitialVariantWeight': 1,
        'InitialInstanceCount': 1,
        'ModelName': model_name,
        'VariantName': 'AllTraffic'}],
    # Set data capture config
    DataCaptureConfig={
        'EnableCapture': True,
        'InitialSamplingPercentage': 100,
        'DestinationS3Uri': f's3://{bucket}/model_monitor/endpoint-data-capture',
        'CaptureOptions': [{'CaptureMode': 'Input'}, {'CaptureMode': 'Output'}],
        'CaptureContentTypeHeader': {
            'CsvContentTypes': ['text/csv'],
            'JsonContentTypes': ['application/json']
        }
    }
)

# エンドポイントの作成
create_endpoint_response = sm_client.create_endpoint(
    EndpointName=endpoint_name,
    EndpointConfigName=endpoint_config_name
)
```

通常の推論エンドポイント設定

通常の設定についてはML-Dark-04で紹介しています！

モニタリング用の追加設定

キャプチャーするデータの格納先

ここではリクエストのINとOUTを両方取得

CSV, JSONそれぞれのcontent typeを指定する。推論リクエストのパラメーターと一致させる

既存のエンドポイントを更新する場合

前後を含めた完全なコードは
data-quality-step-A.ipynb にあります

既存のエンドポイントに追加でデータキャプチャの設定をする場合はSageMaker SDKが便利

```
from sagemaker.model_monitor import DataCaptureConfig
import sagemaker

data_capture_config = DataCaptureConfig(
    enable_capture = True,
    sampling_percentage=100,
    destination_s3_uri=f's3://{bucket}/model_monitor/endpoint-data-capture',
    capture_options=["REQUEST", "RESPONSE"],
    csv_content_types=["text/csv"],
    json_content_types=["application/json"]
)

predictor = sagemaker.Predictor(endpoint_name=endpoint_name)
predictor.update_data_capture_config(data_capture_config=data_capture_config)
```

DataCaptureConfigクラスを使ってConfigを作成する。内容は前ページと同じ

既存のエンドポイント名でpredictorクラスを作成する

data_capture_configのみを渡せば更新してくれる

収集

A2. ベースラインの作成

前後を含めた完全なコードは
data-quality-step-A.ipynb にあります

モデルモニターが比較対象とするためのベースラインを作成する

```
my_default_monitor = model_monitor.DefaultModelMonitor(
    role=sagemaker.get_execution_role(),
    instance_count=1,
    instance_type='ml.t3.large',
    volume_size_in_gb=100,
    max_runtime_in_seconds=3600,
)

my_default_monitor.suggest_baseline(
    baseline_dataset=f's3://{bucket}/model_monitor/baseline_input/',
    dataset_format=DatasetFormat.csv(header=True, output_columns_position='START'),
    output_s3_uri=f's3://{bucket}/model_monitor/baseline/',
    wait=True
)
```

データ品質のモニタリングでは DefaultModelMonitorを利用する。カスタムコンテナを使用したい場合はModelMonitorクラスを使用

ベースラインデータにはラベルを含める。ラベルデータは行の先頭か末尾に配置する。ここでは行の先頭にラベルを配置した場合のオプション指定

実行するとSageMaker Processing Jobを利用したSparkの処理が起動し、ベースラインが計算される

```
Job Name: baseline-suggestion-job-2022-12-03-12-25-33-448 Inputs: [{'InputName': 'baseline_dataset_input', 'AppManaged':
False, 'S3Input': {'S3Uri': 's3://sagemaker-ap-northeast-1-370828233696/model_monitor/baseline_input/', 'LocalPath':
'/opt/ml/processing/input/baseline_dataset_input', 'S3DataType': 'S3Prefix', 'S3InputMode': 'File', 'S3DataDistributionType':
'FullyReplicated', 'S3CompressionType': 'None'}}] Outputs: [{'Outp
{'S3Uri': 's3://sagemaker-ap-northeast-1-370828233696/model_r
'S3UploadMode': 'EndOfJob'}}] .....20
Generating new fontManager, this may take some time...
```

output_s3_uriに2つのjsonファイルが出力される

名前	タイプ	最終更新日時	サイズ
constraints.json	json	2022/12/03 07:21:39 PM JST	5.7 KB
statistics.json	json	2022/12/03 07:21:39 PM JST	176.2 KB

データ品質のベースラインで何が計算されるか

statitstics.jsonから抜粋（数値データ）


```
{
  "version" : 0.0,
  "dataset" : {"item_count" : 23424},
  "features" : [ {
    "name" : "pickup_count",
    "inferred_type" : "Integral",
    "numerical_statistics" : {
      "common" : {
        "num_present" : 23424,
        "num_missing" : 0
      },
      "mean" : 446.163806352459,
      "sum" : 1.0450941E7,
      "std_dev" : 229.86398761822053,
      "min" : 17.0,
      "max" : 1029.0,
      "distribution" : {
        "kll" : {
          "buckets" : [ {
            "lower_bound" : 17.0,
            "upper_bound" : 118.2,
            "count" : 3443.0
          }, {
            "lower_bound" : 118.2,
            "upper_bound" : 219.4,
            "count" : 2102.0
          }, {
            "lower_bound" : 219.4,
            "upper_bound" : 320.6,
            "count" : 1517.0
...
      "sketch" : {
          "parameters" : {
            "c" : 0.64,
            "k" : 2048.0
          },
          "data" : [ [806.0, 838.0, 743.0, 736.0, … ] ]
        }
```


入力ファイルに含まれるFeatureごとに一連の統計情報が出力される

statitstics.jsonから抜粋（文字データ）


```
  }, {
    "name" : "department",
    "inferred_type" : "String",
    "string_statistics" : {
      "common" : {
        "num_present" : 18014,
        "num_missing" : 640
      },
      "distinct_count" : 155.0
    }
```


Feature名やデータタイプなどメタデータ

データの基礎統計量

KLL Sketchによって算出したデータ分布

KLL Sketchのパラメータとサンプリングされた値

GitHub Deeq:
https://github.com/awslabs/deequ

収集

データ品質のベースラインで何が計算されるか

constraints.jsonから抜粋

```
{
  "version" : 0.0,
  "features" : [ {
    "name" : "pickup_count",
    "inferred_type" : "Integral",
    "completeness" : 1.0,
    "num_constraints" : {
      "is_non_negative" : true
    }
  }, {
    "name" : "history_1days",
    "inferred_type" : "Fractional",
    "completeness" : 1.0,
    "num_constraints" : {
      "is_non_negative" : true
    }
…
  }, {
    "name" : "department",
    "inferred_type" : "String",
    "completeness" : 0.9656910046102712
    }
  } ],
  "monitoring_config" : {
    "evaluate_constraints" : "Enabled",
    "emit_metrics" : "Enabled",
    "datatype_check_threshold" : 1.0,
    "domain_content_threshold" : 1.0,
    "distribution_constraints" : {
      "perform_comparison" : "Enabled",
      "comparison_threshold" : 0.1,
      "comparison_method" : "Robust"
    }
  }
}
```

入力データから推測した各Featureの値が従っているルールの提案。**実際のデータの特性と異なる場合は更新**してからモニタリングジョブで利用する。

例：”completeness: 1.0”は、欠損値が存在しないことを示し、欠損値が一つでもあるとcompleteness_checkが違反となる

違反のしきい値など
共通する設定項目

詳細は開発者ガイドを参照
https://docs.aws.amazon.com/ja_jp/sagemaker/latest/dg/model-monitor-byoc-constraints.html

収集

A3. スケジュールの作成

前後を含めた完全なコードは
data-quality-step-A.ipynb にあります

モデルモニターが比較対象とするためのベースラインを作成する

```
my_default_monitor.create_monitoring_schedule(
    monitor_schedule_name=f'{endpoint_name}-schedule',
    endpoint_input=endpoint_name,
    output_s3_uri=f's3://{bucket}/model_monitor/monitoring_report',
    statistics=my_default_monitor.baseline_statistics(),
    constraints=my_default_monitor.suggested_constraints(),
    schedule_cron_expression=model_monitor.CronExpressionGenerator.daily(),
    enable_cloudwatch_metrics=True,
)
```

A2. ベースライン作成に続いて実行する場合は、A2で作成したModelMonitorインスタンスからstatisticsとconstraintsが取得可能

モデルモニターがサポートするモニタリング周期

周期	SageMaker Model Monitor SDKによる指定	cron式
1時間ごと	model_monitor.CronExpressionGenerator.hourly()	cron(0 * ? * * *)
日次	model_monitor.CronExpressionGenerator.daily()	cron(0 0 ? * * *)
一定時間ごと	model_monitor.CronExpressionGenerator.daily_every_x_hours(6)	cron(0 0/6 ? * * *)
一定時間ごと（開始時刻指定）	model_monitor.CronExpressionGenerator.daily_every_x_hours(6, starting_hour=2)	cron(0 2/6 ? * * *)

カスタマイズした制約でモニタリングする場合

カスタマイズした制約を入力とする場合は、カスタマイズ後のConstraints.jsonをS3などから取得する

```
statistics_from_s3 = model_monitor.Statistics.from_s3_uri(f's3://{bucket}/model_monitor/baseline/statistics.json',)
constraints_from_s3 = model_monitor.Constraints.from_s3_uri(f's3://{bucket}/model_monitor/baseline/constraints.json',)

my_default_monitor = model_monitor.DefaultModelMonitor(
    role=sagemaker.get_execution_role(),
    instance_count=1,
    instance_type='ml.t3.large',
    volume_size_in_gb=100,
    max_runtime_in_seconds=3600,
)

my_default_monitor.create_monitoring_schedule(
    monitor_schedule_name=f'{endpoint_name}-schedule',
    endpoint_input=endpoint_name,
    output_s3_uri=f's3://{bucket}/model_monitor/monitoring_report',
    statistics=statistics_from_s3,
    constraints=constraints_from_s3,
    schedule_cron_expression=model_monitor.CronExpressionGenerator.daily(),
    enable_cloudwatch_metrics=True,
)
```

StatisticsおよびConstraintsをS3から取得する（ローカルファイルからの読み込みも可能）

DefaultModelMonitorをインスタンス化する

取得したstatisticsとconstrainsを与えてスケジュールを作成する

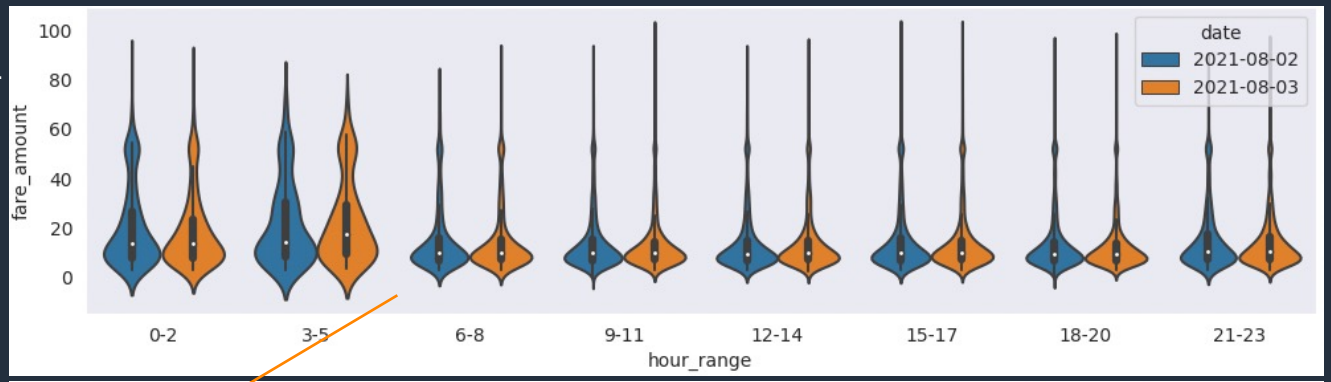
前後を含めた完全なコードは
data-quality-step-A.ipynb にあります

データ収集のため推論してみる場合は
step-0-prediction.ipynb が利用可能です

スケジュール設定の考え方

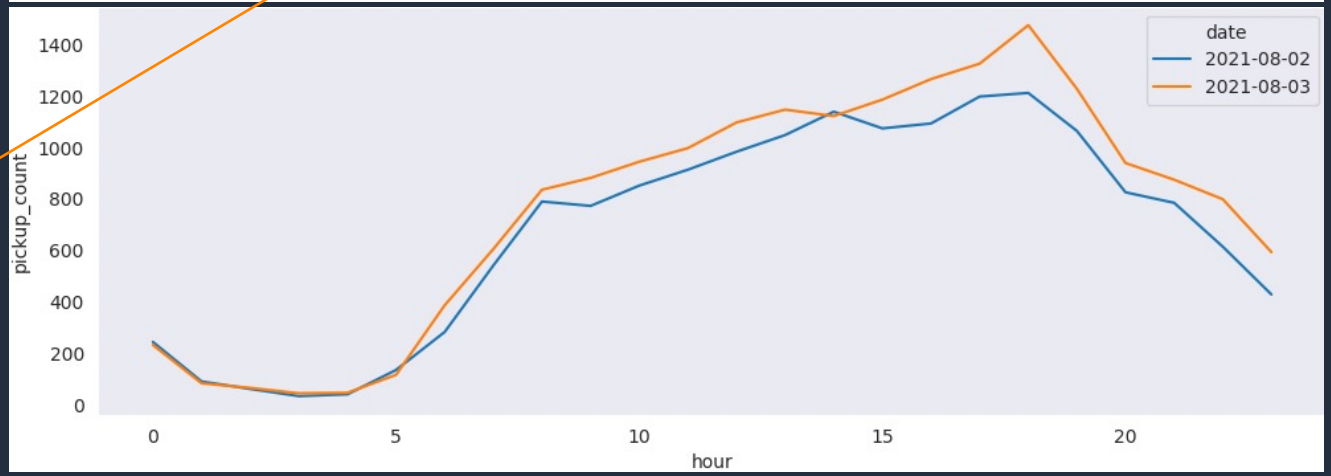
どんな周期でモニタリングすれば適切に傾向を把握できるのか

時間帯ごとの
乗車料金の分布
（2日間）



乗車数の推移
（2日間）

深夜と日中帯には
明確な分布の差異
が見られる



- 今回のタスク（3時間後の乗降数予測）では、**時間帯ごとの変動をまとめて傾向を見るために24時間間隔**が適切
- 時間帯に分けて取り扱う場合は、0-5時台と6-23時台で**データの性質が異なる可能性**がある
- 今回のデータでは、1時間などの**短い間隔ではベースラインと母集団が異なってしまう**

分析・可視化

B2. モニタリング結果の分析

まずconstraint_violations.jsonの内容をチェックする

オレンジのケースは推論の失敗につながり得るので特に注意が必要

違反チェックタイプ	内容	メッセージ例	考えられるリスク
data_type_check	ベースラインと収集データでデータ型が異なる	Data type match requirement is not met. Expected data type: String, Expected match: 100.0%. Observed: Only 0.0% of data is String.	数値データが文字データに変わるなどモデルが処理できないデータが推論リクエストで渡されている
completeness_check	欠損値の割合がしきい値を超えてベースラインを上回った		欠損を許容しないアルゴリズムでは推論に失敗する
baseline_drift_check	ベースラインと収集データでデータの分布が異なり、KS検定の差異がしきい値を上回った	Baseline drift distance: 0.19258531843403345 exceeds threshold: 0.1	モデル作成時の特徴量とラベルの関係性が変化して精度が低下する
missing_column_check	収集データに含まれる特徴量の数がベースラインよりも少ない		特徴量が足りなくて推論が失敗する
extra_column_check	収集データに含まれる特徴量の数がベースラインよりも多い	There are extra columns in current dataset. Number of columns in current dataset: 30, Number of columns in baseline constraints: 28	csv形式のリクエストではヘッダーがないため、新しい特徴量が途中に差し込まれると列がずれ、意味のない推論になる
categorical_values_check	カテゴリ変数に含まれる未知の数がしきい値を上回った		モデル作成時の特徴量とラベルの関係性が変化して精度が低下する

ベースライン・ドリフトとは

ベースライン・ドリフト発生のメッセージ例（constraint_violations.jsonから抜粋）


```
{
  "feature_name" : "trip_distance_mean",
  "constraint_check_type" : "baseline_drift_check",
  "description" : "Baseline drift distance: 0.16661310139662427 exceeds threshold: 0.1"
 }
```


ベースライン（statistics.json）


```
{
    "name" : "trip_distance_mean_12slot",
…
      "distribution" : {
        "kll" : {
          "buckets" : [ {
            "lower_bound" : 2.590930979431202,
            "upper_bound" : 2.7188091863246053,
            "count" : 15.0
          }, {
            "lower_bound" : 2.7188091863246053,
            "upper_bound" : 2.8466873932180086,
            "count" : 51.0
          }, {
            "lower_bound" : 2.8466873932180086,
            "upper_bound" : 2.9745656001114122,
            "count" : 98.0
          }, {
            "lower_bound" : 2.9745656001114122,
            "upper_bound" : 3.1024438070048155,
            "count" : 108.0
          }, {
            "lower_bound" : 3.1024438070048155,
            "upper_bound" : 3.2303220138982187,
            "count" : 54.0
          }, {
```

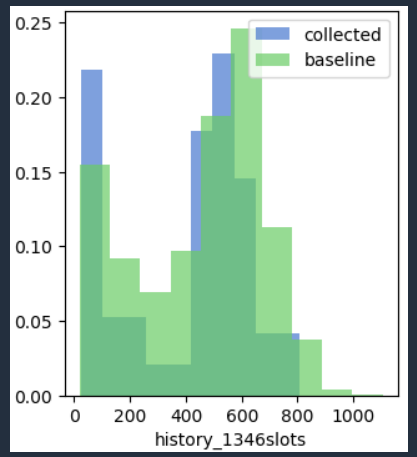

ベースラインとレポートに含まれる分布を比較し、コルモゴロフ–スミルノフ検定（K-S検定）等を用いた指標化を行う

モニタリング・レポート（statistics.json）

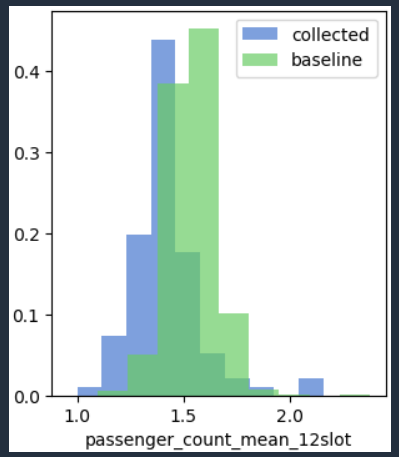

```
{
    "name" : "trip_distance_mean_12slot",
…
      "distribution" : {
        "kll" : {
          "buckets" : [ {
            "lower_bound" : 2.625352008364555,
            "upper_bound" : 2.758025546300555,
            "count" : 10.0
          }, {
            "lower_bound" : 2.758025546300555,
            "upper_bound" : 2.8906990842365543,
            "count" : 10.0
          }, {
            "lower_bound" : 2.8906990842365543,
            "upper_bound" : 3.023372622172554,
            "count" : 1.0
          }, {
            "lower_bound" : 3.023372622172554,
            "upper_bound" : 3.156046160108554,
            "count" : 5.0
          }, {
            "lower_bound" : 3.156046160108554,
            "upper_bound" : 3.288719698044553,
            "count" : 1.0
          }, {
```

K-S検定値0.xとは、たとえばどの程度の分布ずれなのか

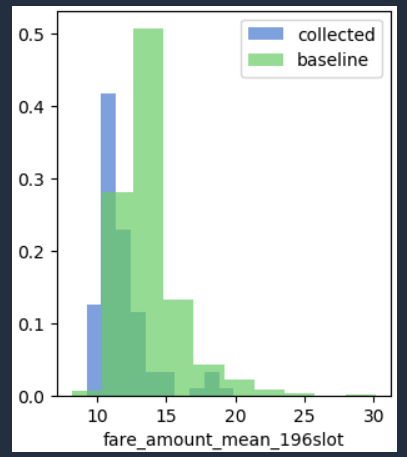
K-S検定値: 0.1



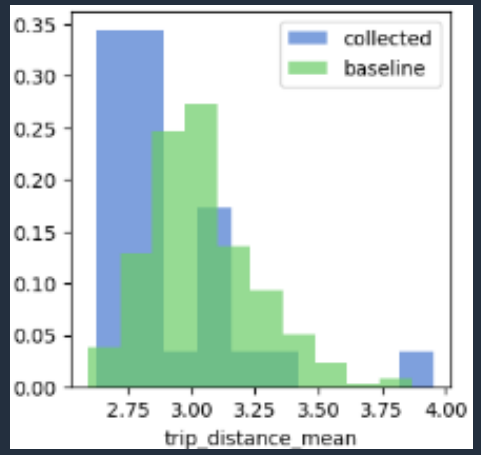
K-S検定値: 0.2



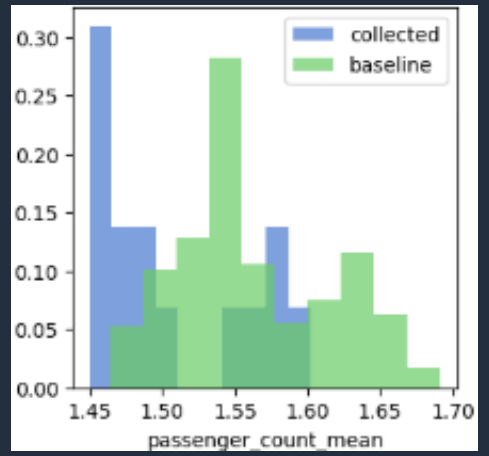
K-S検定値: 0.3



K-S検定値: 0.5



K-S検定値: 0.66



分析・可視化

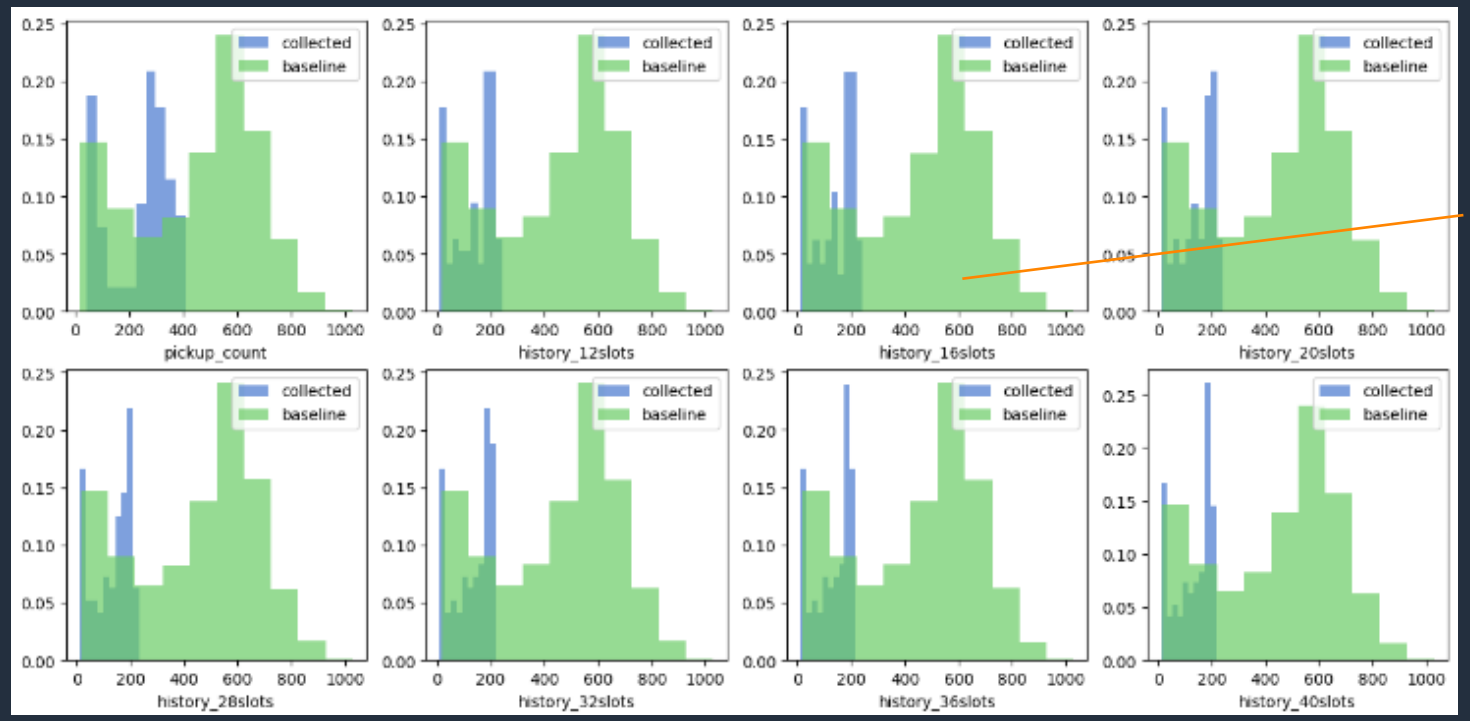
レポートを深掘りして調査する

前後を含めた完全なコードは
data-quality-step-B.ipynb にあります

可視化ライブラリなどを活用して変化を深掘りすることも可能

```
# Get baselines
baseline_statistics = model_monitor.Statistics.from_s3_uri('s3://bucket/model_monitor/baseline/statistics.json').body_dict
statistics = model_monitor.Statistics.from_s3_uri(f's3://bucket/model_monitor/report/statistics.json').body_dict

# Parse statistics
features = mu.get_features(statistics)
feature_baselines = mu.get_features(baseline_statistics)

# Visualizep
mu.show_distributions(features, feature_baselines)
```

2019年7月～12月データのベースラインと2020年3月16日のレポートを比較



乗車数が伸びる時間枠（15分間隔）が明らかに少なくなっている

分析・可視化

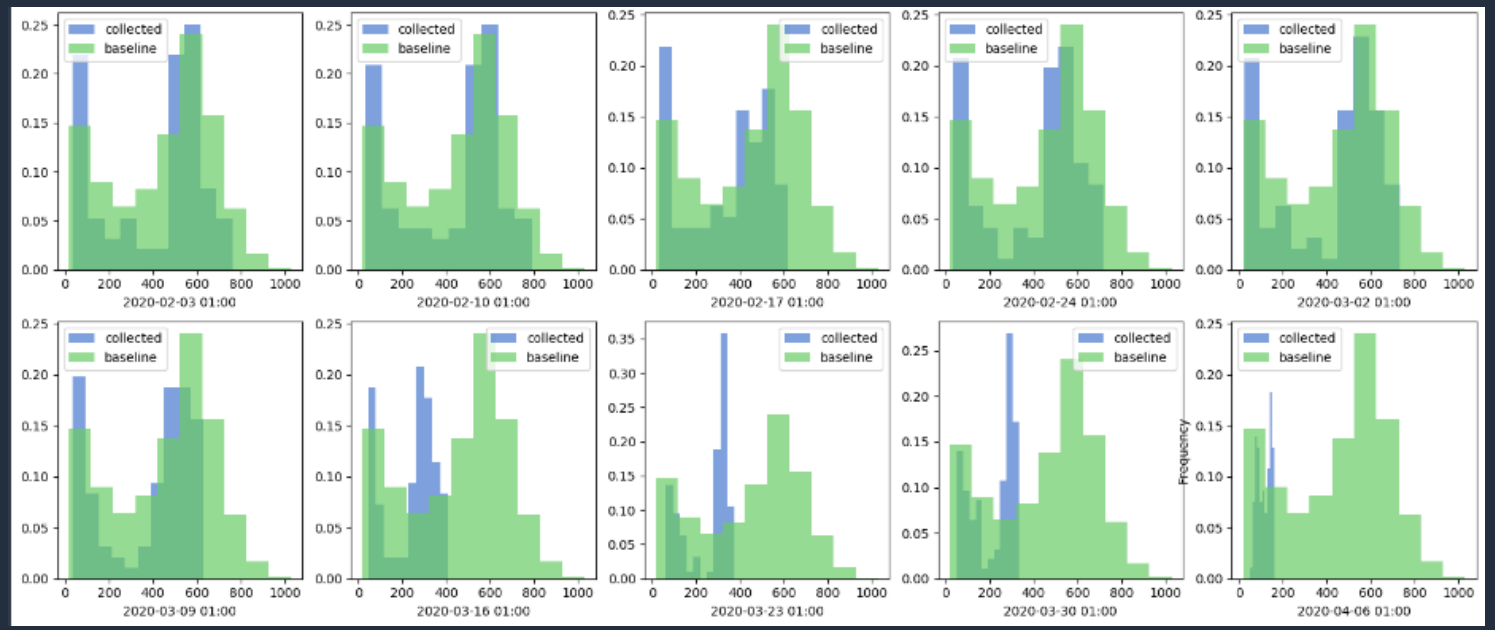
複数のモニタリング期間にまたがる傾向を把握する

単一のレポートのみからは把握しづらい傾向は、複数のレポートにまたがって可視化することで理解しやすくなる

```
# レポートを格納するS3 Bucketをセットする
report_bucket = sagemaker.Session().default_bucket()

# レポートを格納するprefixをセットする
report_prefix = 'model_monitor/monitoring_report/nyctaxi-xgboost-endpoint-x/'

# statisticsの複数期間にまたがるレポートを取得する
statistics_reports = get_reports(report_bucket, report_prefix, 'statistics')

# 特定の特徴量に着目して時系列での変化を可視化する
mu.show_distribution_trend(statistics_reports, 'history_12slots', feature_baselines)
```

前後を含めた完全なコードは data-quality-step-B.ipynb にあります

1週間ごとにどれぐらい分布がずれていったか



2月から4月にかけての月曜日を対象に、15分間あたりの乗車数の分布推移を可視化

B3. CloudWatchのアラート

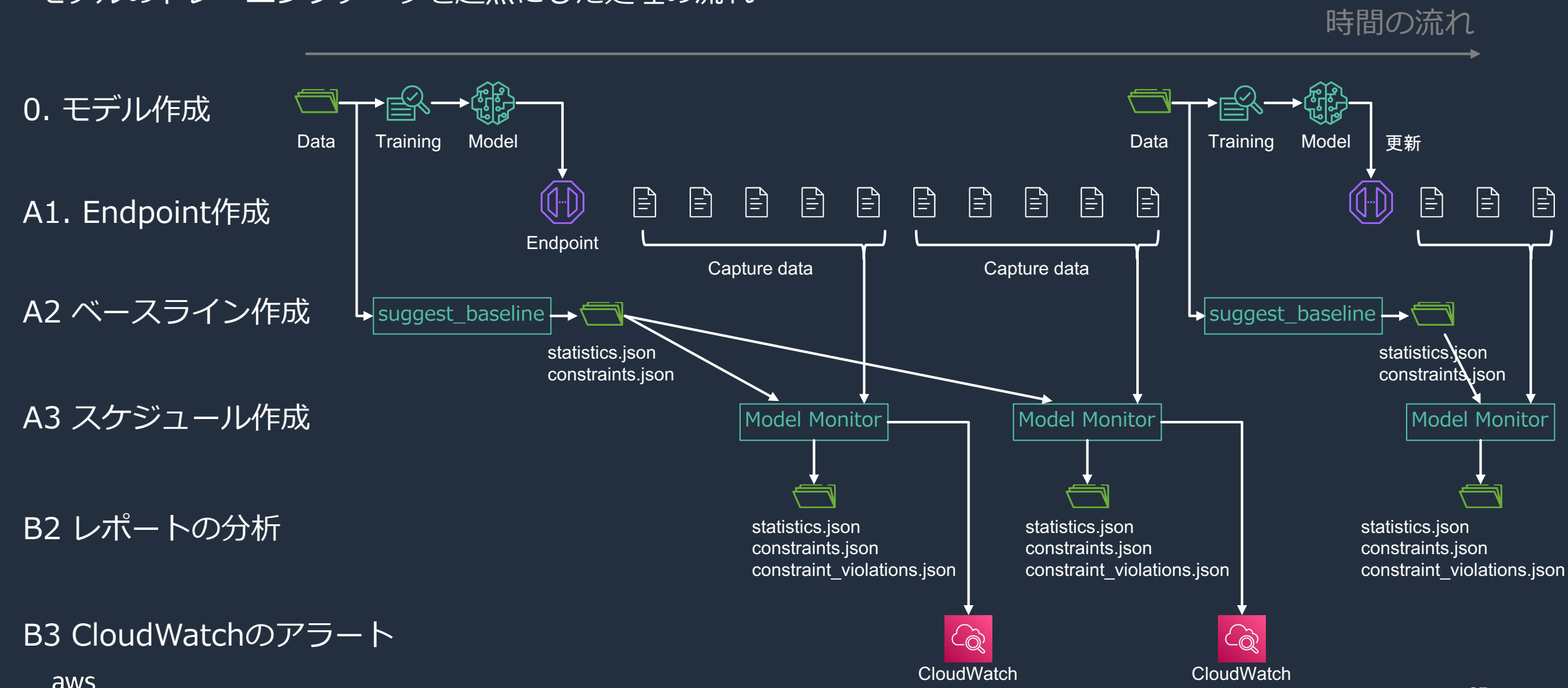
メトリクスの違反をトリガーにした通知とアクション

メトリクス	対象	内容	値の範囲	アラート設定の考え方（例）
total_number_of_violations	モニタリング設定ごとに1つ	モニタリング時に検出された違反の数	0以上	
feature_data_<特徴量名>	数値の特徴量	値の最大、最小、平均値など	特徴量の値の範囲	その特徴量が取り得ない値など
feature_non_null_<特徴量名>	数値、文字の特徴量	値が欠損していない比率	0～100	必須の特徴量である場合は100未満（欠損の発生）を監視する
feature_baseline_drift_<特徴量名>	数値、文字の特徴量	分布の差異（K-S検定値）	0～1	複数のモニタリング周期にまたがって増加傾向など

シンプルなしきい値違反の他に、標準偏差に対するズレや複数回にまたがっての違反などをトリガーとして設定可能

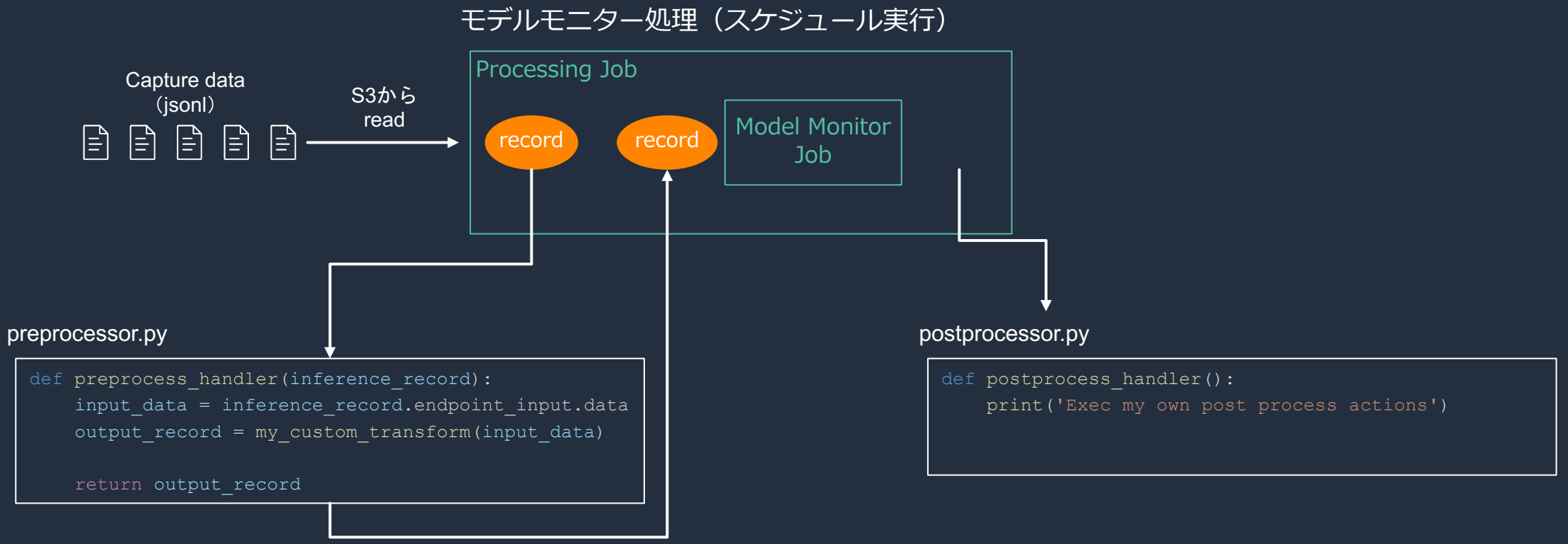
データ品質のモニタリングを流れで理解する

モデルのトレーニングデータを起点にした処理の流れ



Tips｜モニタリングジョブの前後に処理を追加する

モデルモニターで処理するデータの加工や、終了後にカスタムの処理を記述したい場合に使える



レコードを受け取ってレコードを返すので、入力データの変換に使える

後処理はMonitoring Jobとは特段の入出力なし

Tips｜モニタリングの前後に処理を追加する

前後処理を設定するときは、S3に配置したスクリプトのURIをスケジュール作成時に渡す

```
pre_processor_script_s3uri = sagemaker.s3.S3Uploader.upload('./preprocess.py', f's3://{bucket}/model_monitor/script')
postprocess_script_s3uri = sagemaker.s3.S3Uploader.upload('./postprocess.py', f's3://{bucket}/model_monitor/script')

my_default_monitor = model_monitor.DefaultModelMonitor(
    role=sagemaker.get_execution_role(),
    instance_count=1,
    instance_type='ml.t3.large',
    volume_size_in_gb=100,
    max_runtime_in_seconds=3600,
)

my_default_monitor.create_monitoring_schedule(
    monitor_schedule_name=f'{endpoint_name}-schedule',
    endpoint_input=endpoint_name,
    output_s3_uri=f's3://{bucket}/model_monitor/data_quality_monitoring_report',
    statistics=baseline_statistics,
    constraints=suggested_constraints,
    schedule_cron_expression=CronExpressionGenerator.hourly(),
    enable_cloudwatch_metrics=True,
    record_preprocessor_script=pre_processor_script_s3uri,
    post_analytics_processor_script=postprocess_script_s3uri,
)
```

S3Uploaderは、アップロード先のS3URIを戻り値として返却する

返却されたS3URIを、record_preprocessor_scriptもしくはpost_analytics_processor_scriptに渡す

Tips｜ Batch Transformでのモニタリング

2022年10月New！

Batch Transformではデータキャプチャとスケジュールの設定方法が変わるだけで、あとは同じ

A1. データキャプチャ設定のオプション付きでTransformを実行

```
transfomer.transform(
    "s3://{}/transform-input".format(bucket),
    content_type="text/csv",
    split_type="Line",
    batch_data_capture_config=BatchDataCaptureConfig(
        destination_s3_uri=f's3://{bucket}/model_monitor/transform_data_capture',
    ),
)
```

キャプチャの出力先を
スケジュール作成時にセット

A2. ベースライン作成（変更なし）

A3 スケジュール作成

```
my_default_monitor.create_monitoring_schedule(
    monitor_schedule_name=f'{endpoint_name}-schedule',
    batch_transform_input=BatchTransformInput(
        data_captured_destination_s3_uri=f's3://{bucket}/model_monitor/transform_data_capture',
        destination="/opt/ml/processing/input",
        dataset_format=MonitoringDatasetFormat.csv(header=False),
    ),
    output_s3_uri=f's3://{bucket}/model_monitor/data_quality_monitoring_report',
    statistics=statistics_from_s3,
    constraints=constraints_from_s3,
    schedule_cron_expression=model_monitor.CronExpressionGenerator.daily(),
    enable_cloudwatch_metrics=True,
)
```

Amazon SageMakerによるモデル品質のモニタリング

モデル品質モニタリングで行う作業

A. モニタリング開始前にやること

A1. エンドポイントのデプロイ
データキャプチャの開始

A2. ベースラインの作成
データセットの統計量や制約、
SHAP値、バイアス指標、精度の基準生成

A3. モニタリングのスケジュール
データ品質、モデル精度、バイアス、
Feature Attributionのモニタリング設定

B. モニタリング中にやること

B1. Ground Truthの収集
Ground Truthの作成、登録、予測とのマージ

B2. モニタリング結果の分析
メトリクスの可視化、レポートの分析

B3. CloudWatchのアラート
モデル再学習・データ更新のトリガー

収集

A1. エンドポイントのデプロイ

前後を含めた完全なコードは
model-quality-step-A.ipynb にあります

モニタリングするエンドポイントのEndpoint Configにデータキャプチャの設定を追加する

```
# コンフィグの作成
create_endpoint_config_response = sm_client.create_endpoint_config(
    EndpointConfigName=endpoint_config_name,
    ProductionVariants=[{
        'InstanceType': 'ml.t2.medium',
        'InitialVariantWeight': 1,
        'InitialInstanceCount': 1,
        'ModelName': model_name,
        'VariantName': 'AllTraffic'}],
    # Set data capture config
    DataCaptureConfig={
        'EnableCapture': True,
        'InitialSamplingPercentage': 100,
        'DestinationS3Uri': f's3://{bucket}/model_monitor/endpoint-data-capture',
        'CaptureOptions': [{'CaptureMode': 'Input'}, {'CaptureMode': 'Output'}],
        'CaptureContentTypeHeader': {
            'CsvContentTypes': ['text/csv'],
            'JsonContentTypes': ['application/json']
        }
    }
)

# エンドポイントの作成
create_endpoint_response = sm_client.create_endpoint(
    EndpointName=endpoint_name,
    EndpointConfigName=endpoint_config_name
)
```

データ品質用にデータキャプチャ設定がされていれば改めて追加する必要はない

モデル品質のモニタリングではOutputの指定が必須

収集

A2. ベースラインの作成

前後を含めた完全なコードは
model-quality-step-A.ipynb にあります

モデルモニターが比較対象とするためのベースラインを作成する

```
model_quality_monitor = ModelQualityMonitor(
    role=role,
    instance_count=1,
    instance_type='ml.m5.xlarge',
    volume_size_in_gb=20,
    max_runtime_in_seconds=1800,
)

job = model_quality_monitor.suggest_baseline(
    job_name=baseline_job_name,
    baseline_dataset=f's3://{bucket}/model_monitor/model_quality_baseline_input/',
    dataset_format=DatasetFormat.csv(header=True),
    output_s3_uri = f's3://{bucket}/model_monitor/model_quality_baseline/',
    problem_type='Regression',
    inference_attribute= "pred",
    ground_truth_attribute= "pickup_count"
)
job.wait(logs=False)
```

ベースラインに与えたデータセットの予測精度がモニタリング時の基準となるため、**テストデータのみ**を与える

回帰問題では
ground_truth_attributeに**Ground Truthのカラム名**を、
inference_attributeに**学習時の予測結果を格納したカラム名**を指定する

分類問題で与える引数は開発者ガイドに例が示されている

問題のタイプを指定する。
Regression, BinaryClassification, MulticlassClassificationの3種類

データ品質と同様にoutput_s3_uriに2つのjsonファイルが出力される

名前	タイプ	最終更新日時	サイズ
constraints.json	json	2022/12/03 07:21:39 PM JST	5.7 KB
statistics.json	json	2022/12/03 07:21:39 PM JST	176.2 KB

収集

モデル品質のベースラインで何が計算されるか

回帰問題（Regression）の場合のベースライン

statitstics.json


```
{'version': 0.0,
 'dataset': {
    'item_count': 2688,
    'evaluation_time': '2022-12-11T07:15:20.74Z'},
 'regression_metrics': {
    'mae': {'value': 38.29501488095238,
            'standard_deviation': 0.24146914041470052},
    'mse': {'value': 2809.991443452381,
            'standard_deviation': 41.9839494650476},
    'rmse': {'value': 53.00935241495015,
             'standard_deviation': 0.39589778065391784},
    'r2': {'value': 0.9499197226669486,
           'standard_deviation': 0.0010604830245174754}}
}
```


constraints.json


```
{'version': 0.0,
 'regression_constraints': {
    'mae': {'threshold': 38.29501488095238,
            'comparison_operator': 'GreaterThanThreshold'},
    'mse': {'threshold': 2809.991443452381,
            'comparison_operator': 'GreaterThanThreshold'},
    'rmse': {'threshold': 53.00935241495015,
             'comparison_operator': 'GreaterThanThreshold'},
    'r2':   {'threshold': 0.9499197226669486,
             'comparison_operator': 'LessThanThreshold'}}
}
```


モデル学習時の評価に用いたメトリクスをConstraintsとして使用する

分類問題（Binary Classification/Multiclass Classification）の場合のベースライン

https://docs.aws.amazon.com/sagemaker/latest/dg/model-monitor-model-quality-metrics.html

収集

A3. スケジュールの作成

前後を含めた完全なコードは
model-quality-step-A.ipynb にあります

モデル品質のモニタリングジョブをスケジュールする

```
endpont_input = EndpointInput(
    endpoint_name=endpoint_name,
    destination="/opt/ml/processing/input/endpoint",
    start_time_offset="-PT3H",
    end_time_offset="-PT1H",
    inference_attribute="0",
)

model_quality_monitor.create_monitoring_schedule(
    monitor_schedule_name=f'{endpoint_name}-schedule',
    output_s3_uri=f's3://{bucket}/model_monitor/model_quality_monitoring_report',
    constraints=model_constraints,
    schedule_cron_expression=model_monitor.CronExpressionGenerator.hourly(),
    enable_cloudwatch_metrics=True,
    endpoint_input=endpont_input,
    ground_truth_input=f's3://{bucket}/model_monitor/model_quality_ground_truth',
    problem_type='Regression',
)
```

Ground Truthが付与できるタイミングの幅を指定する。くわしくはのちほど

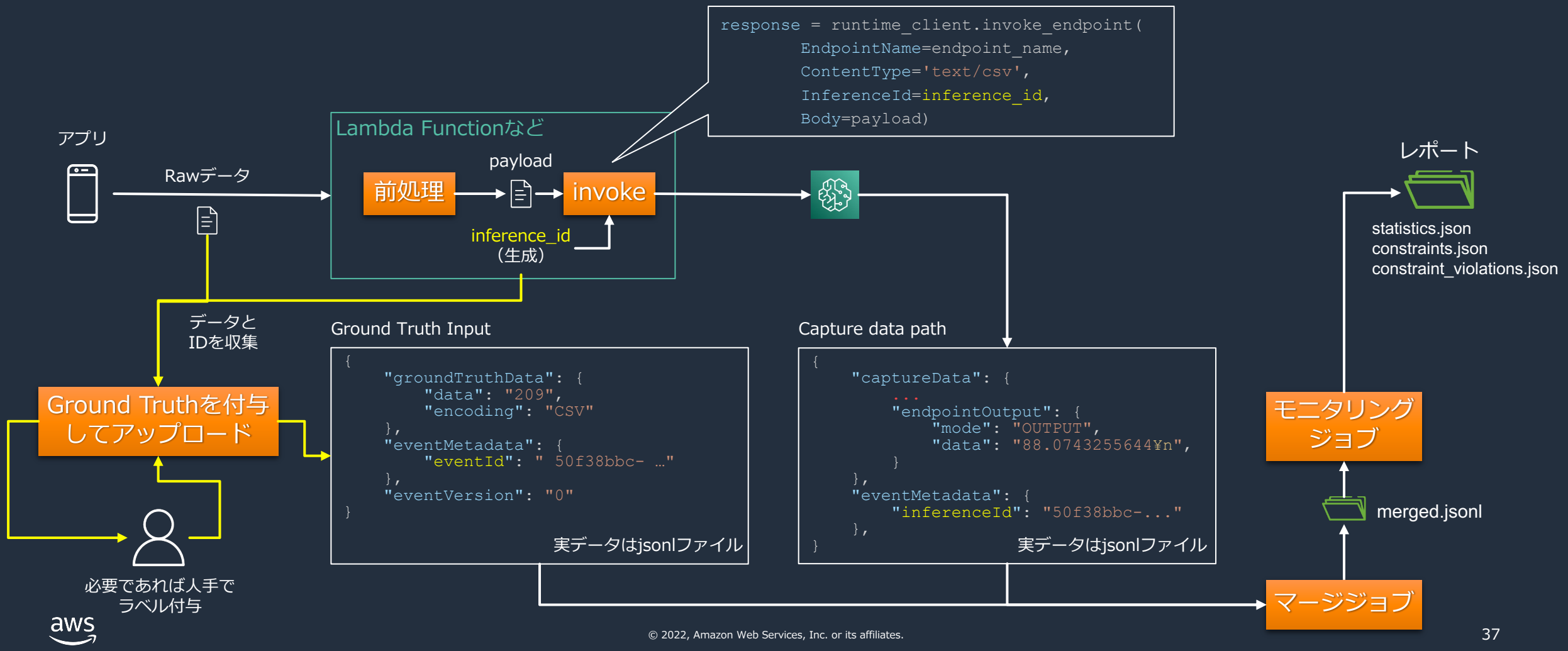
EndpointのOutputの何列目に推論結果がセットされるかを指定する。回帰問題では0固定。

推論後に収集するGround Truthをアップロードするパス

B1. Ground Truthの収集

前後を含めた完全なコードは
model-quality-step-B.ipynb にあります

モデル品質のモニタリングでは、Ground Truthを収集し、紐付けのためのキー（event_idもしくはinference_id）とともにモニタリングジョブの入力として与える



参考｜Ground Truthマージ前後のフォーマット

エンドポイントによるキャプチャーデータ


```
{
    "eventVersion": "0",
    "captureData": {
        "endpointInput": {
            "data": "378.0,335.0,402.0,788.0,…
            "encoding": "CSV",
            "mode": "INPUT",
            "observedContentType": "text/csv"
        },
        "endpointOutput": {
            "data": "165.10098266601562¥n",
            "encoding": "CSV",
            "mode": "OUTPUT",
            "observedContentType": "text/csv; charset=utf-8"
        }
    },
    "eventMetadata": {
        "eventId": "4a92f099-31ec-439b-86db-330ed127eb07",
        "inferenceId": "9a12b376-23ad-4c48-be0e-084501c180a3",
        "inferenceTime": "2022-12-11T10:24:49Z"
    }
}
```


追加で作成するGround Truthデータ


```
{
    "groundTruthData": {
        "data": "174",
        "encoding": "CSV"
    },
    "eventMetadata": {
        "eventId": "9a12b376-23ad-4c48-be0e-084501c180a3"
    },
    "eventVersion": "0"
}
```


マージ
ジョブ

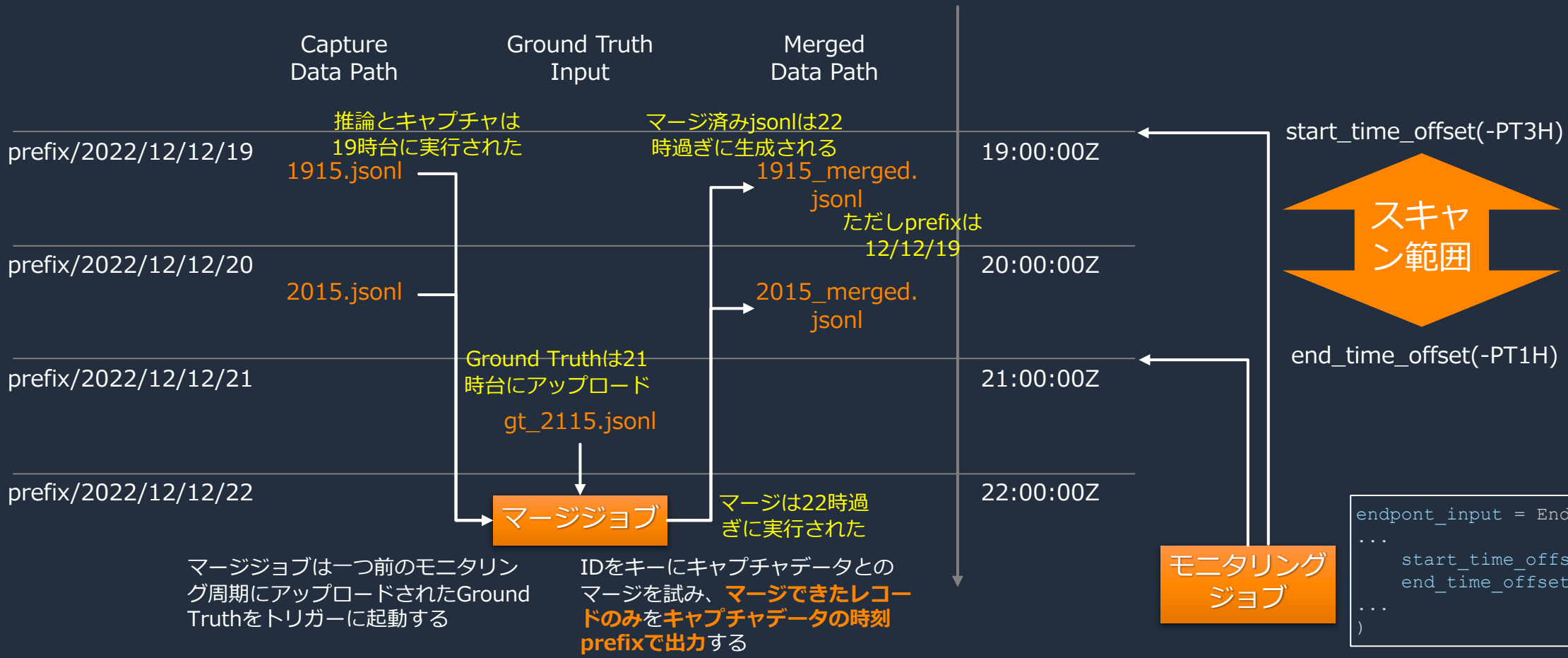
マージ済みレコード


```
{
    "eventVersion": "0",
    "groundTruthData": {
        "data": "174",
        "encoding": "CSV"
    },
    "captureData": {
        "endpointInput": {
            "data": "378.0,335.0,402.0,788.0,…
            "encoding": "CSV",
            "mode": "INPUT",
            "observedContentType": "text/csv"
        },
        "endpointOutput": {
            "data": "165.10098266601562¥n",
            "encoding": "CSV",
            "mode": "OUTPUT",
            "observedContentType": "text/csv; charset=utf-8"
        }
    },
    "eventMetadata": {
        "eventId": "4a92f099-31ec-439b-86db-330ed127eb07",
        "inferenceId": "9a12b376-23ad-4c48-be0e-084501c180a3",
        "inferenceTime": "2022-12-11T10:24:49Z"
    }
}
```


実データはすべてjsonlファイル

Ground Truth収集までの時間差を設定する

start_time_offsetとend_time_offsetは、Ground Truthが付与済みのファイルがアップロードされる時間の範囲で設定する



マージジョブは一つ前のモニタリング周期にアップロードされたGround Truthをトリガーに起動する

IDをキーにキャプチャデータとのマージを試み、**マージできたレコードのみを**キャプチャデータの時刻**prefixで出力**する

```
endpont_input = EndpointInput(
...
    start_time_offset="-PT3H",
    end_time_offset="-PT1H",
...
)
```

モニタリングジョブは、起動時間の0分から遡ってstart_timeとend_timeの範囲にあるmergedデータを入力として参照する

分析・可視化

B2. モニタリング結果の分析

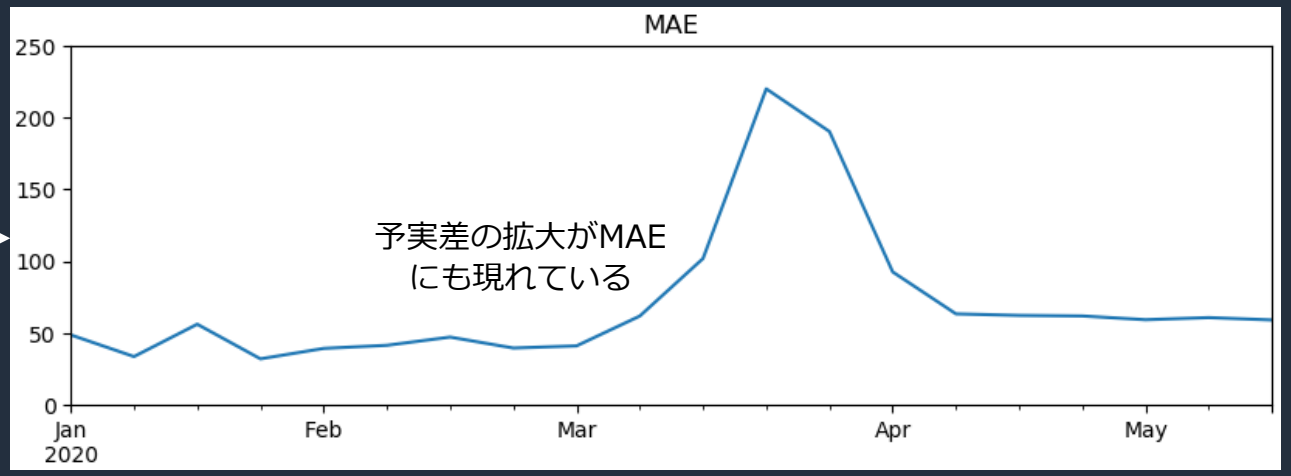
グラフ化のコードは
model-quality-step-B.ipynb にあります

レポートを時系列で可視化することで傾向を把握できる。ここでは日次のレポートを7日間ごとに取得して可視化

レポート

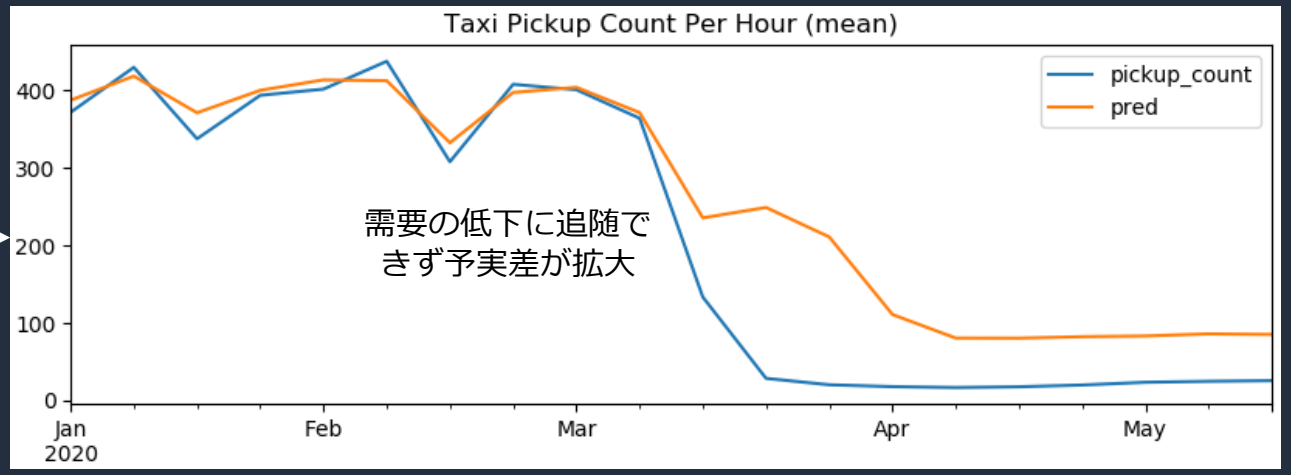
時系列の推移を
グラフ化

statistics.json
constraints.json
constraint_violations.json



Ground Truth

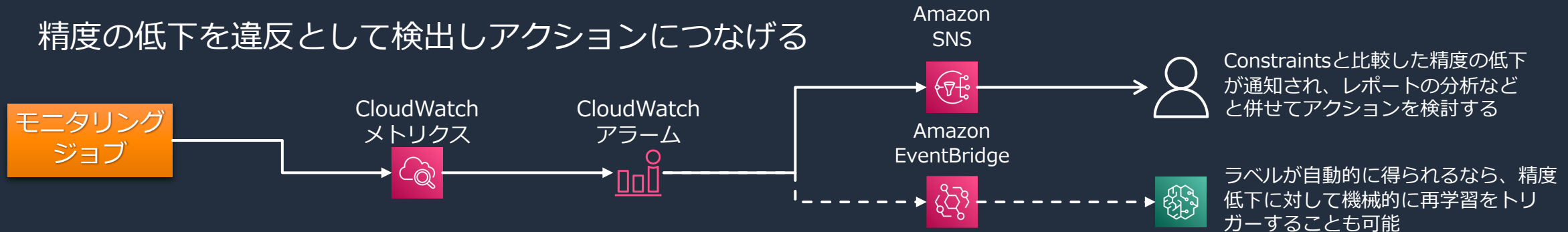
Ground Truthと
予測結果をグラフ化



- 長期間の蓄積が可能な場合は**精度メトリクス自体の分布を取る**ことも有用
- ある精度が**実用に耐えるかは業務にも依存**するため、**LOBへの定期的な連携**も検討する
- 多くの関係者に展開する場合は**QuickSightによるダッシュボード公開**も効果的

B3. CloudWatchのアラート

精度の低下を違反として検出しアクションにつなげる



分類問題	二項分類	多項分類
mae	confusion_matrix	confusion_matrix
mse	recall	accuracy
rmse	precision	weighted_recall
r2	accuracy	weighted_precision
	recall_best_constant_classifier	weighted_f0_5
	precision_best_constant_classifier	weighted_f1
	accuracy_best_constant_classifier	weighted_f2
	true_positive_rate	accuracy_best_constant_classifier
	true_negative_rate	weighted_recall_best_constant_classifier
	false_positive_rate	weighted_precision_best_constant_classifier
	false_negative_rate	weighted_f0_5_best_constant_classifier
	receiver_operating_characteristic_curve	weighted_f1_best_constant_classifier
	precision_recall_curve	weighted_f2_best_constant_classifier
	auc	
	f0_5, f1, f2	
	f0_5_best_constant_classifier	
	f1_best_constant_classifier	
	f2_best_constant_classifier	

Tips｜ Batch Transformでのモニタリング

2022年10月New！

Batch Transformでのモデル品質モニタリング

A1. データキャプチャ設定

```
transfomer.transform(
    "s3://{}/transform-input".format(bucket),
    content_type="text/csv",
    split_type="Line",
    batch_data_capture_config=BatchDataCaptureConfig(
        destination_s3_uri=f's3://{bucket}/model_monitor/transform_data_capture',
        generate_inference_id= True,
    ),
)
```

Batch Transformのモデル品質モニタリングでは、inference id生成の指定をTrueにしておく

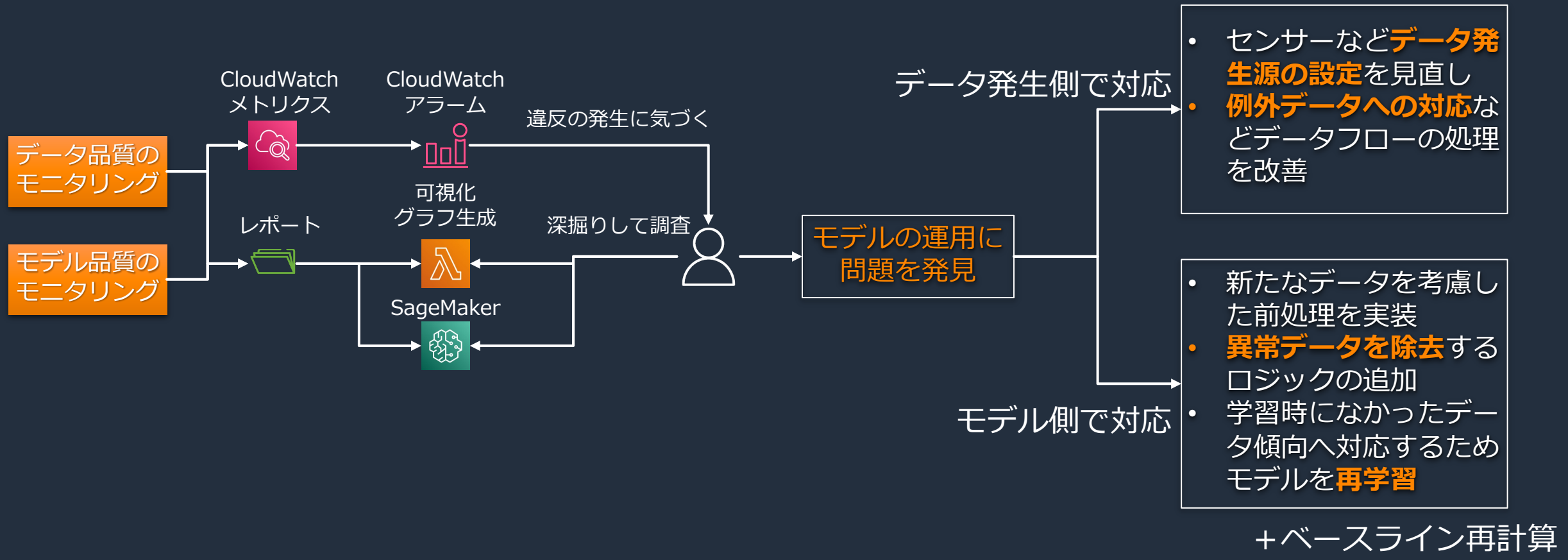
A3. スケジュール作成

```
model_quality_monitor.create_monitoring_schedule(
    monitor_schedule_name=f'{endpoint_name}-schedule',
    batch_transform_input=BatchTransformInput(
        data_captured_destination_s3_uri=f's3://{bucket}/model_monitor/transform_data_capture',
        destination="/opt/ml/processing/input",
        dataset_format=MonitoringDatasetFormat.csv(header=False),
        probability_attribute="0",
        probability_threshold_attribute=0.5,
        start_time_offset="-PT6H",
        end_time_offset="-PT0H",
    ),
    ground_truth_input=f's3://{bucket}/model_monitor/model_quality_ground_truth',
    problem_type='BinaryClassification',
    output_s3_uri=f's3://{bucket}/model_monitor/model_quality_monitoring_report',
    constraints=constraints_from_s3,
    schedule_cron_expression=model_monitor.CronExpressionGenerator.daily(),
    enable_cloudwatch_metrics=True,
)
```

スケジュール作成側ではGround Truth関連の設定が追加になる

モニタリングの結果からアクションを起こす

モニタリングの設定はデータ品質とモデル品質で個別に行うが、問題の発生時は相互に関連するため、いずれかの異常をトリガーとして相互に調査してアクションを決める



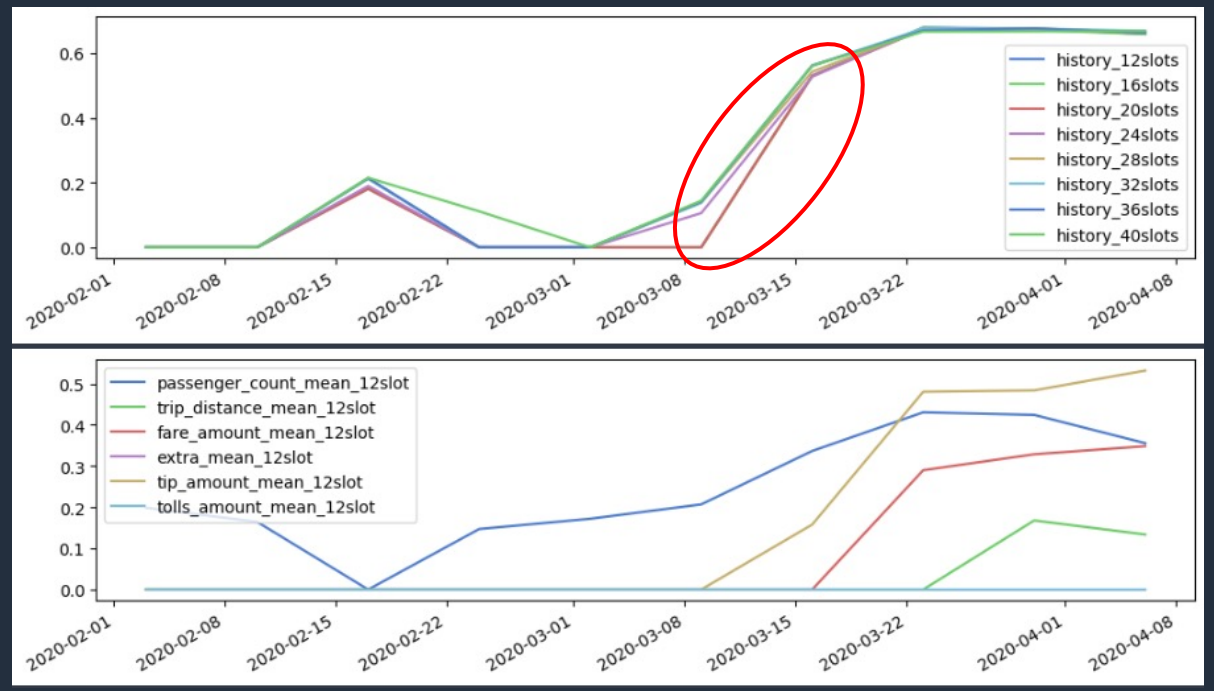
今回のタスクでモニタリングを通じて何がわかったか

やったこと

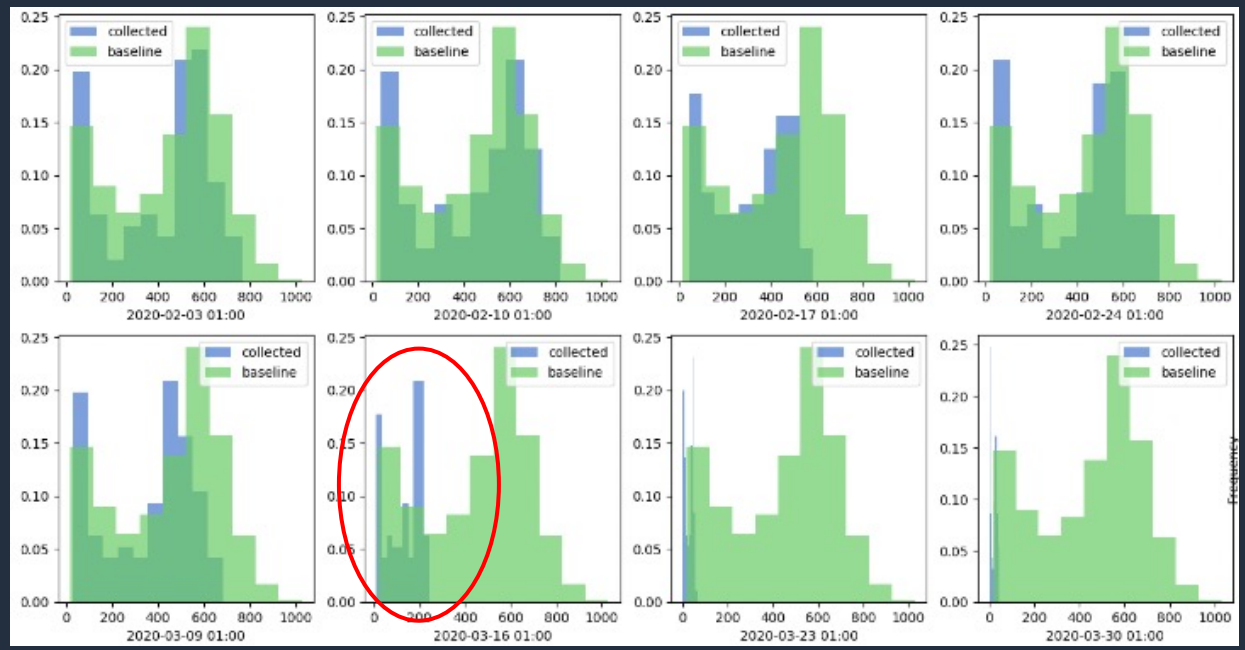
- 1日分96回の推論リクエストをキャプチャーし、1日分を1周期としてモニタリングを実行
- 長期の傾向を見るために、2020年前半の毎週月曜日をピックアップしてレポートの推移を可視化

データ品質のモニタリングでは。。。

baseline_driftは3月中旬から急激に上昇し高止まり



特徴量（history_12slots）の分布も3/16から左側に大きくシフト



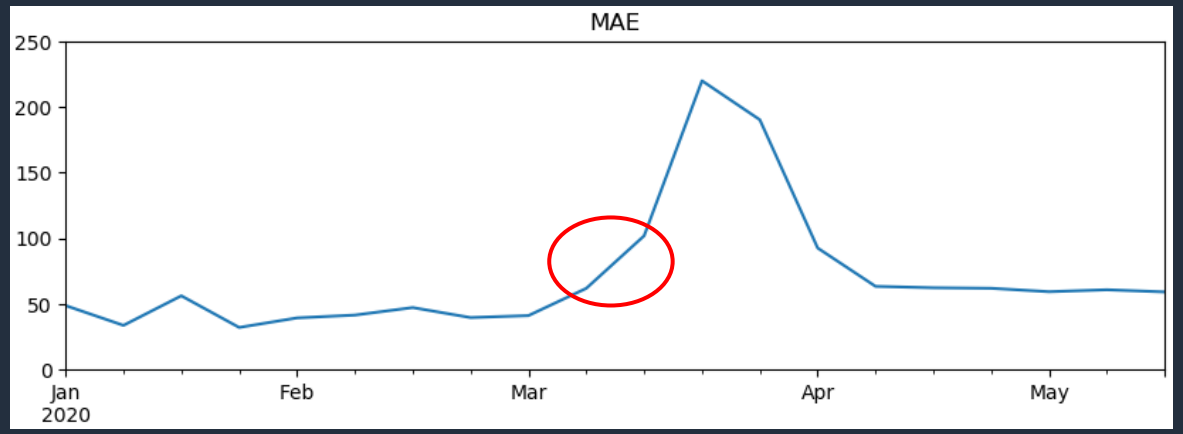
今回のタスクでモニタリングを通じて何がわかったか

やったこと

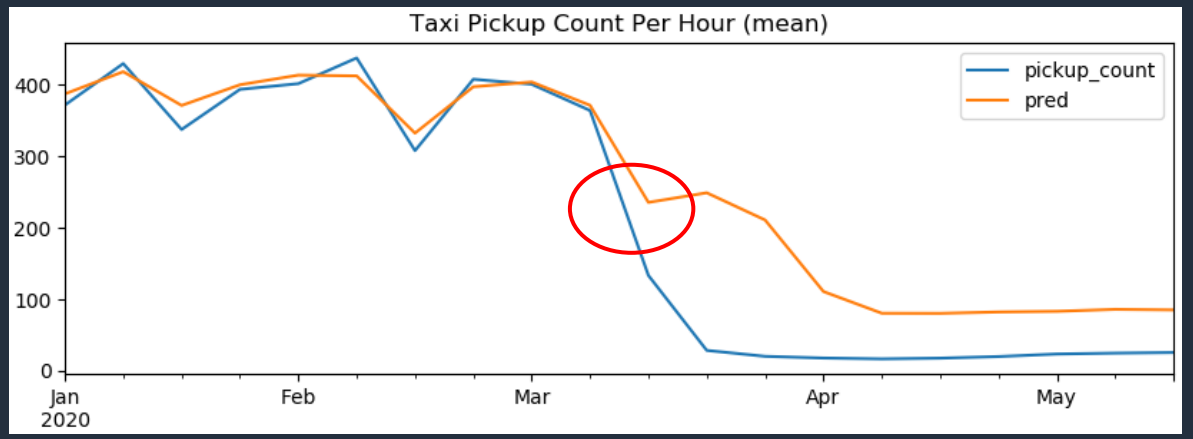
- 1日分96回の推論リクエストをキャプチャーし、1日分を1周期としてモニタリングを実行
- 長期の傾向を見るために、2020年前半の毎週月曜日をピックアップしてレポートの推移を可視化

モデル品質のモニタリングでは。。。

MAE（平均絶対誤差）も3月中盤から上昇



Ground Truthから得た予実差も増加



3月の前半ごろに気づいてモデルを再学習

＋ベースライン再計算

まとめ

- オープンデータを利用した回帰問題を例にデータ品質モニタリングとモデル品質モニタリングの実践を解説
- データ品質モニタリングでは、収集と分析・可視化に分けてコードサンプルと結果の例を具体的に説明
- モデル品質モニタリングでは、追加で必要となるGround Truthの収集と利用の流れを詳しく説明
- 紹介したサンプルコードはGitHubで公開

ML Enablement Seriesの動画

機械学習モデルをビジネス価値につなげる方法を全力で解説！

Light Part

製品やサービスに機械学習を導入するプロジェクトの進め方

https://bit.ly/3M1F9as

Step Up!!

Dark Part

機械学習モデルの開発や運用をマネージドサービスで効率的に行う方法

https://bit.ly/3927PCN

資料集・お問合せ・Special Thanks

AWSの日本語資料の場所: 「AWS 資料」で検索

AWSのハンズオン資料の場所: 「AWS ハンズオン」で検索

お問合せ

技術的なお問合せ

料金のお問合せ

個別相談会のお申込み

Thank you!