Ateliers

Formation Machine Learning d'Elastic Stack

Pré-requis :
Poste développeur avec accès réseau Internet libre
Linux (Recommandé) ou Windows 10
Minimum 16 Go RAM
Pré-installation de :

- Docker
- Git
- Éditeur YAML (VSCode, ...)

Table des matières

Ateliers 1 : Mise en place

1.1 Installation et démarrage

Ouvrir une ligne de commande dans le répertoire :

```
$TP_DATA/1_MiseEnPlace
```

Exécuter la commande qui initialise les utilisateurs systèmes de l'installation et démarrer un cluster 3 nœuds ElasticSearch ainsi que Kibana.

```
docker compose up -d
```

Accéder à ElasticSearch http://localhost:9200

Accéder ensuite à Kibana à http://localhost:5601

Utiliser l'utilisateur : ***elastic/secret***

Retrouver la *Dev Console*.

Exécuter les requêtes suivantes :

```
GET /_cluster/health
GET /_cat/nodes
GET /_cat/indexes
GET /_search
```

1.2 Activation License

Connectez-vous à *localhost:5601* et *Management → Stack Management → License management*

Activer la licence d'évaluation

1.3 Beats

Dans le répertoire ***$MyWork/metricbeat-<version>***, démarrer metricbeat afin qu'il alimente le cluster avec des données de monitoring :

```
sudo ./metricbeat -e
```

Ateliers 2 : Jobs Single et Multi-metric

2.1 Chargement et prise en main des données

Les données représente les réponses d'un service. Les données sont déjà agrégées. On a :

- ***timestamp***
- ***accept*** : Le nombre de requêtes acceptées
- ***deny*** : le nombre de requêtes refusées
- ***total*** : le nombre total de requêtes
- ***response*** : le temps de réponse moyen
- ***host*** : le hôte
- ***service*** : le nom du service

Récupérer l'archive fournie $TP_DATA/2_Tutorial/files.zip et la décompresser dans un répertoire de travail.

Après l'avoir rendu exécutable, Utiliser le script fourni pour importer les données.

```
./upload_server_metrics.sh
```

Définir ensuite dans Kibana un index pattern ***server-metrics****

Utiliser le Data Visualizer pour visualiser les données.
Sur quelle période porte les données ?

Visualiser la distribution des valeurs des champs, en particulier le champ ***total***

2.2 Single Metric Job

2.2.1 Création du job

Dans Kibana :
Machine Learning → Create new job

Sélectionner le data view et Single Metric Job puis :

- Toute la plage de données
- Le champ Sum de total
- Un bucket span de 15mn

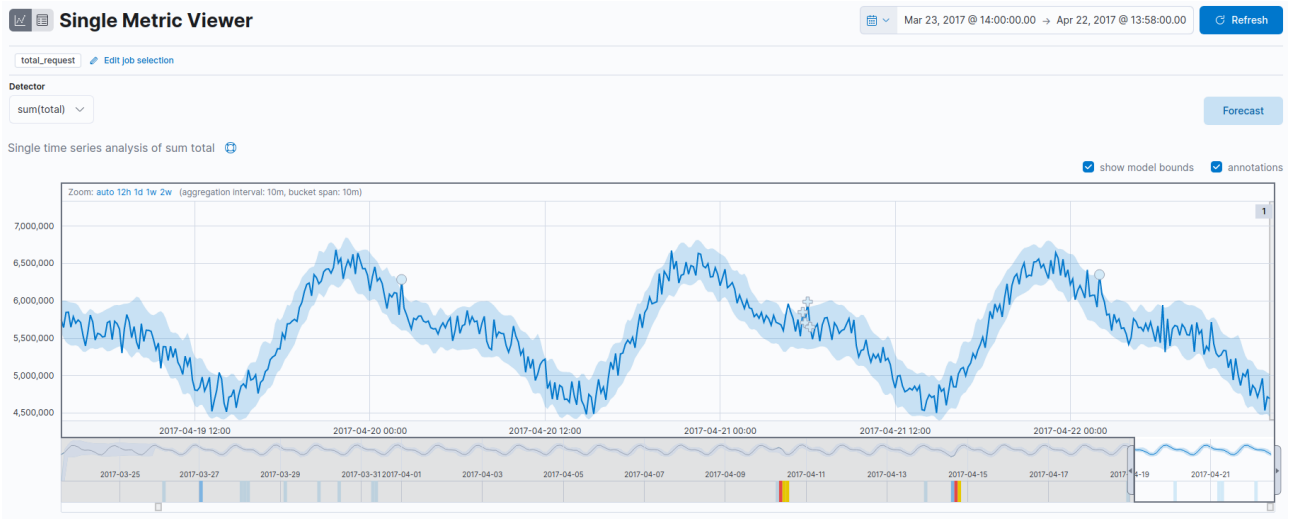
Utilisez toutes les données disponibles pour l'analyse

Donner un nom au job : ***total_request*** , un groupe : ***formation***

Créer et démarrer le job

2.2.2 Visualisation des résultats

Visualiser les résultats dans le ***Single Metric Viewer***



Les valeurs réelles sont indiquées par une ligne bleue. Une zone bleue ombrée représente les limites des valeurs attendues.

Si une valeur est en dehors de la zone ombragée, elle est anormale.

Multi-bucket impact :

Il se peut que des événements anormaux n'existent pas au sein d'une seule plage mais se produisent sur une gamme de plages contigus. Pour prendre en compte cette situation, EL-ML effectue une analyse multi-bucket en prenant en compte les buckets contigus

Dans le visualiseur, les anomalies avec des impacts multi_bucket élevé sont indiquées avec des croix plutôt que des points

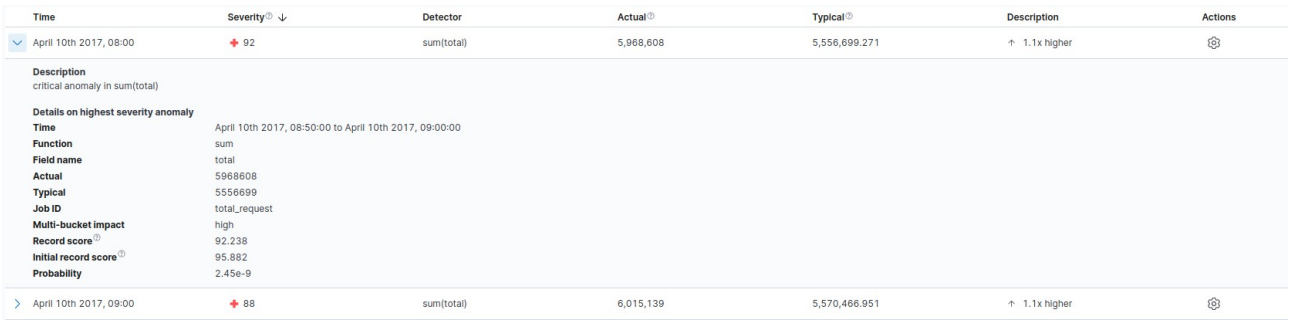
Un score d’anomalie est calculé pour le bucket et donne une couleur au point

Les détails des anomalies est visible dans la partie inférieure

Il est possible de faire glisser le sélecteur de temps.

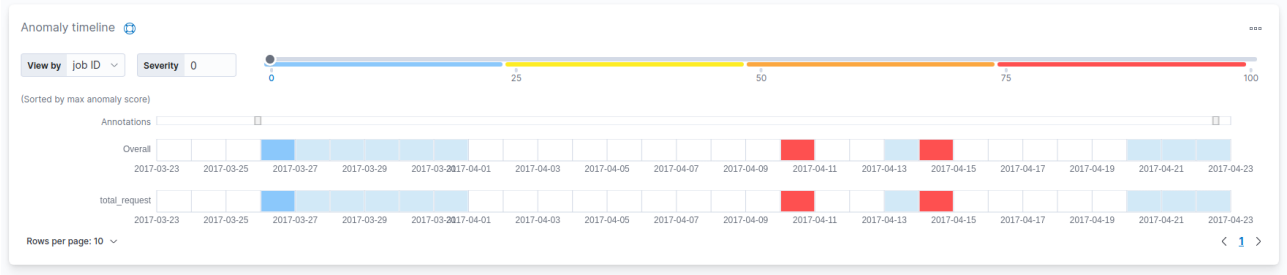
Faire glisser le sélecteur de temps pour sélectionner une section contenant une anomalie critique.

Visualiser les informations liées à l’anomalie, dans le tableau en bas



Visualiser ensuite les résultats avec l’ ***Anomaly Explorer***

Sélectionner une anomalie critique



2.2.3 Prévisions

Effectuer une prévision d’1 semaine

2.2 Jobs multi-métriques

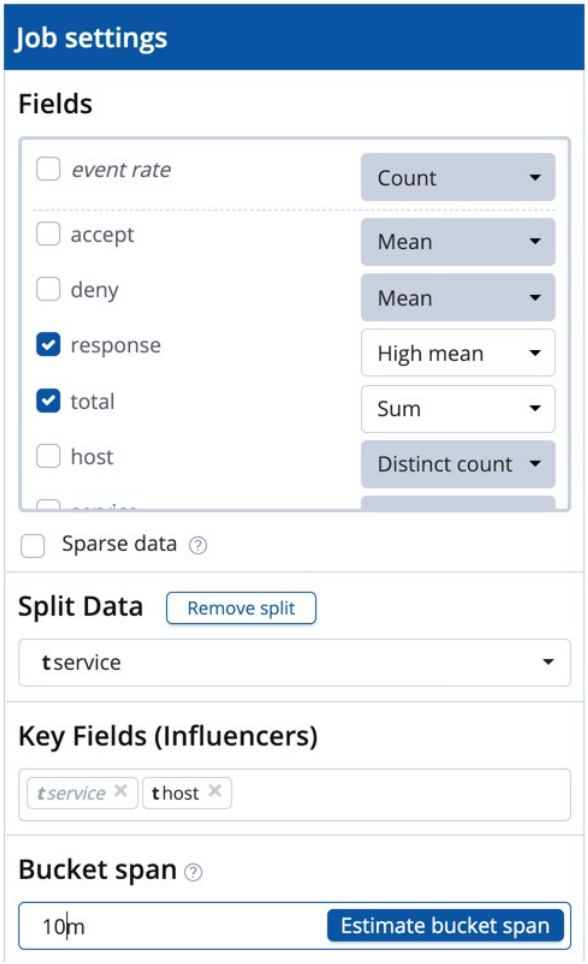
2.2.1 Création du job

Démarrer l’assistant pour créer un job multi-métriques et le configurer comme suit :

La configuration consiste en :

- 2 détecteurs
- 1 bucket span de 10 minutes
- Une catégorisation par le champ service
- Un influenceur supplémentaire ***host***

Lui donner le nom de *response_request_by_service*, en utilisant le groupe *formation*



2.2.2 Visualisation des résultats

Visualiser les résultats dans l'explorateur d'anomalies

Identifier les services et hosts les plus responsables des anomalies

Changer la fenêtre temporelle pour zoomer sur les anomalies

Ateliers 3 : Autres jobs

3.1 Jobs de population

3.1.1 Importation des données avec logstash

Commencer par créer un gabarit d'index permettant d'identifier un champ geo_point

```
PUT _index_template/logstash-apache-template
{
  "index_patterns": ["logstash-apache-*"],
  "template": {
    "mappings": {
      "properties": {
        "client": {
          "properties": {
            "geo": {
              "properties": {
                "location": {
                  "type": "geo_point"
                }
              }
            }
          }
        }
      }
    }
  },
  "priority": 100
}
```

Installer logstash et regarder les fichiers de configuration fournis permettant de démarrer 2 pipelines qui traite des fichiers de logs :

- Logs d'accès d'un site web
- Logs applicatifs d'une application métier

Pour démarrer correctement *logstash*, vous devez :

- Placer le fichier *pipelines.yml* dans le répertoire config de *logstash* et l'adapter à votre environnement (chemins des autres fichiers de configuration)
- Éditer les 2 fichiers de configuration *web/config_apache.conf* et *appli/logstash-grok-spring-boot.conf* et modifier les chemins indiqués

Ensuite :

```
${LOGSTASH_HOME}/bin/logstash -r
```

3.1.2 Jobs de population

A partir des logs d'accès d'un site web, nous recherchons des adresses IP agissant comme des robots automatisés.

- Par exemple, les Ips effectuant de nombreuses requêtes.
- Des IPS effectuant du scan d'URL

3.2 Jobs de catégorisation de message

Créer une dataview pour le datastream ***logs-springboot-app-springboot***

Sur cette dataview créer 2 jobs de caractérisation travaillant sur le champ ***message*** et un bucket span de 1h:

- ***cat-spring-count*** utilise la fonction count
- ***cat-spring-rare*** utilise la fonction rare

Pour les 2 jobs, éditer la requête du datafeed afin de filtrer tous les messages générés par un scheduler :

```
{
  "bool": {
    "must_not": [
     {
      "match_phrase": {
        "thread": " scheduling-1"
      }
    }
  ]
  }
}
```

Démarrer les job et visualiser les anomalies

3.3 Rareté

Sur la dataview Apache, créer 3 jobs rare :

- ***apache-rare-status*** : Détecter les codes réponses rares
- ***apache-rare-status-by-ip*** : Détecter les codes réponses rares pour un client ip particulier
- ***apache-freq_rare-status-by-ip*** : Détecter les codes réponses rares pour une ip cliente qui arrive fréquemment

Comparer les résultats

3.4 Géo job

Sur la dataview Apache, créer 1 jobs géo :

- ***apache-geo-rare*** : Détecte les emplacements inhabituels par continent.

Ateliers 4. API et détecteurs

4.1 Comparaison *by_field_name* et *partition_name*

Par l'API ou l'assistant de projet avancé, créer 2 jobs ayant chacun 1 détecteur qui partitionnent les données d'accès web, pour un bucket de 1j par pays :

- Soit en utilisant *by_field_name*
- Soit en utilisant *partition_name*

Les résultats sont stockés dans un index particulier

Un influenceur sur le champ ***method***

Comparer les résultats obtenus

4.2 Création de job multi-détecteurs

Via l'API, créer un job qui réunit les 3 détecteurs de l'atelier 3.3 en 1 seul

4.3 Règle personnalisée

Utiliser une règle personnalisée pour filtrer les pays comme Monaco, Seychelles, Panama, Luxembourg

5. Analyse de cause et corrélation des jobs

Les données disponibles sont :

- Le volume de transactions (KPI) sommé par minutes
- Logs applicatifs (messages textuels semi-structurés) du moteur de traitement des transactions
- Mesures de performance d'utilisation du réseau

5.1 Importer les données

KPI :
Créer l'index avec le bon mapping

```
PUT /it_ops_kpi
{
  "settings" : {
    "number_of_shards" : 1,
    "number_of_replicas" : 1
  },
  "mappings": {
      "properties" : {
        "@timestamp" : {
          "type" :  "date",
          "format": "epoch_millis"
        }
} } }
```

Puis importer les données:

```
curl -X POST -H "Content-Type: application/json"
http://localhost:9200/it_ops_kpi/_bulk --data-binary "@it_ops_kpi.json"
```

Vérifier avec :

```
GET /it_ops_kpi/_count
```

Traces du moteur

```
PUT /it_ops_app
{
  "settings" : {
    "number_of_shards" : 1,
    "number_of_replicas" : 1
  },
  "mappings": {
```

```
      "properties" : {
        "@timestamp" : {
          "type" :  "date",
          "format": "epoch_millis"
        }
} } }
```

Importer les données avec :

```
curl -X POST -H "Content-Type: application/json"
http://localhost:9200/it_ops_app/_bulk --data-binary "@it_ops_app_logs.json"
```

Vérifier avec :

```
GET /it_ops_app/_count
```

Réseau

Créer l'index *pour les données réseau :*

```
PUT /it_ops_network
{
  "settings" : {
    "number_of_shards" : 1,
    "number_of_replicas" : 1
  },
  "mappings": {
      "properties" : {
        "@timestamp" : {
          "type" :  "date",
          "format": "epoch_millis"
        }
} } }
```

Puis :

```
curl -X POST -H "Content-Type: application/json"
http://localhost:9200/it_ops_network/_bulk --data-binary "@it_ops_network.json"
```

Vérifier avec :

```
GET /it_ops_network/_count
```

5.2 Jobs ML

Créer 3 jobs :

- Détection d'anomalie sur des nombre de transactions faibles
- Anomalie sur le décompte des messages de traces catégorisés
- Anomalies sur les performances réseau (moyennes sur les métriques)

Utiliser un bucket span de 15m et des influenceurs partagés (*hostname*, *physical host*)

5.3 Corrélation des analyses

Identifier les influenceurs responsable de l’anomalie

6. Prévisions

6.1 Flux de données continu

Créer une dataview sur le datastream metricbeat

Créer un job simple-métrique surveillant la valeur moyenne de la mémoire utilisée

6.3 Prévisions

Essayer de prévoir via l'api de forecast quand le nombre de document de l'index ***.monitoring-es**** dépassera une certaine valeur

Ateliers 7 : Analyse de trame de données

7.1 Détection de valeurs anormales

Ajouter les données exemple ***Kibana Sample Data Logs***

Créer une transformation :

```
POST _transform/_preview
{
  "source": {
    "index": [
      "kibana_sample_data_logs"
    ]
  },
  "pivot": {
    "group_by": {
      "clientip": {
        "terms": {
          "field": "clientip"
        }
      }
    },
    "aggregations": {
      "@timestamp.value_count": {
        "value_count": {
          "field": "@timestamp"
        }
      },
      "bytes.max": {
        "max": {
          "field": "bytes"
        }
      },
      "bytes.sum": {
        "sum": {
          "field": "bytes"
        }
      },
      "request.value_count": {
        "value_count": {
          "field": "request.keyword"
```

```
        }
      }
    }
  }
}

PUT _transform/logs-by-clientip
{
  "source": {
    "index": [
      "kibana_sample_data_logs"
    ]
  },
  "pivot": {
    "group_by": {
      "clientip": {
        "terms": {
          "field": "clientip"
        }
      }
    },
    "aggregations": {
      "@timestamp.value_count": {
        "value_count": {
          "field": "@timestamp"
        }
      },
      "bytes.max": {
        "max": {
          "field": "bytes"
        }
      },
      "bytes.sum": {
        "sum": {
          "field": "bytes"
        }
      },
      "request.value_count": {
        "value_count": {
          "field": "request.keyword"
        }
      }
    }
  },
  "description": "Web logs by client IP",
  "dest": {
```

```
    "index": "weblog-clientip"
  }
}
```

La démarrer :

```
POST _transform/logs-by-clientip/_start
```

Vous pouvez voir le résultat dans ***Stack Management → Transform***

Créer un job d'analyse de type « Outlier detection»

Par l'assistant ou via l'API :

```
PUT _ml/data_frame/analytics/weblog-outliers
{
  "source": {
    "index": "weblog-clientip"
  },
  "dest": {
    "index": "weblog-outliers"
  },
  "analysis": {
    "outlier_detection": {
    }
  },
  "analyzed_fields" : {
    "includes" :
["@timestamp.value_count","bytes.max","bytes.sum","request.value_c
ount"]
  }
}
```

Démarrer l'analyse

```
POST _ml/data_frame/analytics/ecommerce/_start
```

Le score ***ml.outlier*** est une valeur comprise entre 0 et 1. Plus la valeur est élevée, plus il est probable qu'il s'agisse d'une valeur aberrante
Chaque document est également annoté avec les valeurs d'influence de caractéristiques pour chaque champ.
Ces valeurs totalisent 1 et indiquent quels champs sont les plus importants pour décider si une entité est une valeur aberrante ou pas
Kibana fournit également une matrice de nuages de points . Les valeurs aberrantes avec un score qui dépasse le seuil sont mises en évidence

7.2. Régression

Ajouter les données exemple ***Flight Data***

On veut prévoir le champ numérique ***FlightDelayMins*** représentant le retard en minutes d'un vol

Les données sont déjà sous une forme tabulaire et sont donc prêtes pour une analyse, nous n'avons pas besoin de transformations

Le jeu de données contient des informations telles que les conditions météorologiques, les destinations et les origines des vols, les distances de vol, les transporteurs et le nombre de minutes de retard de chaque vol.

Exemple de document :


```
"_source": {
"FlightNum": "9HY9SWR",
"DestCountry": "AU",
"OriginWeather": "Sunny",
"OriginCityName": "Frankfurt am Main",
"AvgTicketPrice": 841.2656419677076,
"DistanceMiles": 10247.856675613455,
"FlightDelay": false,
"DestWeather": "Rain",
"Dest": "Sydney Kingsford Smith International Airport",
"FlightDelayType": "No Delay",
"OriginCountry": "DE",
"dayOfWeek": 0,
"DistanceKilometers": 16492.32665375846,
"timestamp": "2024-12-30T00:00:00",
"DestLocation": {
"lat": "-33.94609833",
"lon": "151.177002"
},
"DestAirportID": "SYD",
"Carrier": "Kibana Airlines",
"Cancelled": false,
"FlightTimeMin": 1030.7704158599038,
"Origin": "Frankfurt am Main Airport",
"OriginLocation": {
"lat": "50.033333",
"lon": "8.570556"
},
"DestRegion": "SE-BD",
"OriginAirportID": "FRA",
"OriginRegion": "DE-HE",
```

```
"DestCityName": "Sydney",
"FlightTimeHour": 17.179506930998397,
"FlightDelayMin": 0
}
```

Créer une analyse de type régression sur ce jeu de données :

- Améliorez éventuellement la qualité de l'analyse en ajoutant une requête qui supprime les données erronées. Dans ce cas, nous omettons les vols d'une distance de 0 kilomètre ou moins.
- Choisir ***FlightDelayMin*** comme variable dépendante,
- Ajoutez ***Cancelled***, ***FlightDelay*** et ***FlightDelayType*** à la liste des champs exclus. Ces champs seront exclus de l'analyse. Il est recommandé d'exclure les champs qui contiennent des données erronées ou qui décrivent la variable dépendante.

Spécifier une valeur de 5 dans le champ *feature importance*
Un modèle mémoire limité à 50 Mo
Un job ID : ***model-flight-delay-regression***

API :

```
PUT _ml/data_frame/analytics/model-flight-delays-regression
{
  "source": {
    "index": [
      "kibana_sample_data_flights"
    ],
    "query": {
      "range": {
        "DistanceKilometers": {
          "gt": 0
        }
      }
    }
  },
  "dest": {
    "index": "model-flight-delays-regression"
  },
  "analysis": {
    "regression": {
      "dependent_variable": "FlightDelayMin",
      "training_percent": 90,
      "num_top_feature_importance_values": 5,
      "randomize_seed": 1000
    }
  },
  "model_memory_limit": "50mb",
  "analyzed_fields": {
```

```
    "includes": [],
    "excludes": [
      "Cancelled",
      "FlightDelay",
      "FlightDelayType"
    ]
  }
}
```

Démarrage du job :

```
POST _ml/data_frame/analytics/model-flight-delays-regression/_start
```

Suivre l'évolution du job et accéder aux résultats lorsqu'il est terminé.

```
GET _ml/data_frame/analytics/model-flight-delays-regression/_stats
```

Visualisation des résultats

Les résultats affichent :

- Le contenu de l'index de destination dans un format tabulaire.
- les métriques d'évaluation du modèle,
- les valeurs des Feature importance
- et une matrice de nuage de points

Le tableau de résultat montre une colonne pour la variable dépendante (*FlightDelayMin*), qui contient les véritables valeurs de vérité terrain et les valeurs de prédiction (***ml.FlightDelayMin_prediction***) et une colonne. Il indique également si le document a été utilisé dans l'ensemble d'apprentissage (***ml.is_training***).

Le panneau *Feature Importance* indique comment chaque champ affecte une prédiction particulière. Seules les 5 valeurs les plus significatives sont stockées dans l'index. Cependant, les métadonnées du modèle entraîné contiennent également l'amplitude moyenne des valeurs d'importance des caractéristiques pour chaque champ sur toutes les données d'entraînement.

Les valeurs des *feature importance* pour chaque prédiction peuvent être visualisées sous forme de graphique

Kibana affiche également une matrice de nuage dans le résultat

Évaluation des résultats
Kibana fournit des métriques **d'erreur de l'entraînement**, qui représentent les performances du modèle sur l'ensemble de données d'entraînement.
Il fournit également des mesures **d'erreur de généralisation**, qui représentent les performances du modèle sur les données de test.

Une erreur quadratique moyenne (MSE) de zéro signifie que les modèles prédisent la variable dépendante avec une précision parfaite
De même, une valeur R-squared de 1 indique que toute la variance de la variable dépendante peut être expliquée par les variables caractéristiques.

7.3. Classification

Sur le même jeu de données, on veut prédire si un vol sera retardé ou pas (valeur booléenne)

Créer une analyse de type Classification.

- Choisir FlightDelay comme variable dépendante, qui est le champ que nous voulons prédire avec l'analyse de classification.
- Ajoutez Cancelled, FlightDelayMin et FlightDelayType à la liste des champs exclus.
- Choisir un pourcentage d'entraînement de 10
- Spécifier une valeur pour le nombre de feature importance
- Une limite mémoire
- Un job id : model-flight-delays-classification

Démarrer le job

En API, cela donne :

```
PUT _ml/data_frame/analytics/model-flight-delays-classification
{
  "source": {
    "index": [
      "kibana_sample_data_flights"
    ]
  },
  "dest": {
    "index": "model-flight-delays-classification",
    "results_field": "ml"
  },
  "analysis": {
    "classification": {
      "dependent_variable": "FlightDelay",
      "training_percent": 10,
      "num_top_feature_importance_values": 10
    }
  },
  "analyzed_fields": {
    "includes": [],
    "excludes": [
      "Cancelled",
      "FlightDelayMin",
      "FlightDelayType"
```

```
    ]
  }
}
```

Démarrer le job

```
POST _ml/data_frame/analytics/model-flight-delays-classification/_start
```

Une fois le job terminé, visualisez les résultats

Évaluation des résultats :

Kibana fournit une matrice de confusion normalisée qui contient le pourcentage d'occurrences où l'analyse a classé correctement les points de données avec leur classe réelle et le pourcentage d'occurrences où elle les a mal classées

Kibana fournit également la courbe ROC. Le graphique compare le taux de vrais positifs (axe des y) au taux de faux positifs (axe des x) pour chaque classe ; (dans ce cas vrai et faux). A partir de ce tracé, la valeur de l'aire sous la courbe (AUC) est calculée. C'est un nombre compris entre 0 et 1. Plus l'AUC est élevée, meilleur est le modèle pour prédire correctement les classes