Python for data analysis

LIN-SI Marina

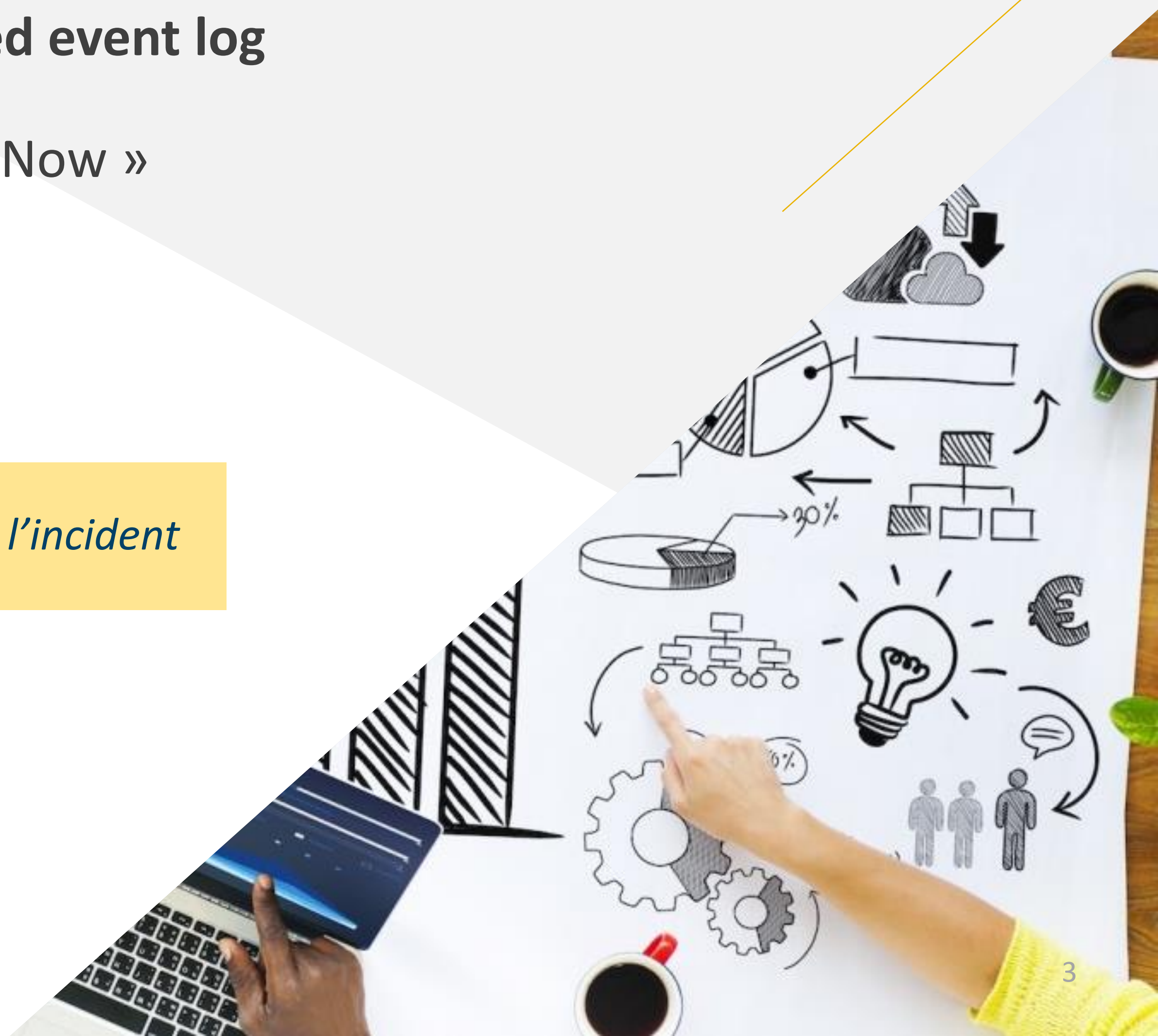
Sommaire

1. Objectif

Dataset : Incident management process enriched event log Data Set

- « Event log » extrait de la plateforme « ServiceNow »
- Caractéristiques :
    - Nombre de lignes : 141712 (24918 incidents)
    - Nombre d'attributs : 36

Objectif : *Prédire le temps restant avant résolution de l'incident*



2. Constitution du dataset

A. Phases préliminaires

- Les attributs retenus pour la constitution du dataset :

Attribut	Description
incident state	eight levels controlling the incident management process transitions from opening until closing the case
reassignment_count	number of times the incident has the group or the support analysts changed
reopen_count	number of times the incident resolution was rejected by the caller
sys_mod_count	number of incident updates until that moment
made_sla	boolean attribute that shows whether the incident exceeded the target SLA
sys_updated_at	incident system update date and time
contact_type	categorical attribute that shows by what means the incident was reported
category	first-level description of the affected service
subcategory	second-level description of the affected service (related to the first level description, i.e., to category)
u_symptom	description of the user perception about service availability
impact	description of the impact caused by the incident (values: 1-High; 2-Medium; 3-Low)
urgency	description of the urgency informed by the user for the incident resolution (values: 1-High; 2-Medium; 3-Low)
priority	calculated by the system based on 'impact' and 'urgency'
knowledge	boolean attribute that shows whether a knowledge base document was used to resolve the incident
u_priority_confirmation	boolean attribute that shows whether the priority field has been double-checked
notify	categorical attribute that shows whether notifications were generated for the incident
Closed_at	incident user close date and time (dependent variable)

2. Constitution du dataset

B. Ajout de variable

- « **remaining_time** »
  - Créée en faisant la différence entre les dates « closed_at » et « updated_at »
  - Indique le temps restant avant la complétion de l'incident (en heures)

```
# Ajout de la colonne "remaining_time" qui correspond au temps restant avant résolution de l'incident
# (en heures)
datetimeFormat = '%d/%m/%Y %H:%M'
datas['remaining_time'] = datas.apply(lambda row: (datetime.datetime.strptime(row.closed_at, datetimeFormat)\
    - datetime.datetime.strptime(row.sys_updated_at, datetimeFormat)).total_seconds()/3600, axis=1)

# Supression de la colonne "closed_at"
# (plus d'utilité après le calcul de "remaining_time")
del datas['closed_at']
```

2. Constitution du dataset

C. Nettoyage du dataset

- Après avoir nettoyé les données et traité les données catégoriques, on obtient le dataset suivant :

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 141712 entries, 0 to 141711
Data columns (total 38 columns):
reassignment_count            141712 non-null int64
reopen_count                  141712 non-null int64
sys_mod_count                 141712 non-null int64
made_sla                      141712 non-null bool
category                      141712 non-null int64
subcategory                   141712 non-null int64
u_symptom                     141712 non-null int64
impact                        141712 non-null int64
urgency                       141712 non-null int64
priority                      141712 non-null int64
knowledge                     141712 non-null bool
u_priority_confirmation       141712 non-null bool
notify                        141712 non-null int64
remaining_time                141712 non-null float64
state_Active                  141712 non-null uint8
state_Awaiting_Evidence       141712 non-null uint8
state_Awaiting_Problem        141712 non-null uint8
state_Awaiting_User_Info      141712 non-null uint8
state_Awaiting_Vendor         141712 non-null uint8
state_Closed                  141712 non-null uint8
state_New                     141712 non-null uint8
state_Resolved                141712 non-null uint8
state_Unknown                 141712 non-null uint8
update_month                  141712 non-null int64
update_day                    141712 non-null int64
update_hour                   141712 non-null int64
update_weekday_0              141712 non-null uint8
update_weekday_1              141712 non-null uint8
update_weekday_2              141712 non-null uint8
update_weekday_3              141712 non-null uint8
update_weekday_4              141712 non-null uint8
update_weekday_5              141712 non-null uint8
update_weekday_6              141712 non-null uint8
contact_Direct opening        141712 non-null uint8
contact_Email                 141712 non-null uint8
contact_IVR                   141712 non-null uint8
contact_Phone                 141712 non-null uint8
contact_Self service          141712 non-null uint8
dtypes: bool(3), float64(1), int64(13), uint8(21)
memory usage: 18.4 MB
```

3. Modélisation

A. X et y

- **y :** « remaining_time »
- **X :** toutes les autres colonnes du dataset

```
X_all = datas.drop(['remaining_time'], axis=1)
y_all = datas['remaining_time']
```

3. Modélisation

B. Training set et Testing set

- **Trainaing set :** 75% du dataset
- **Testing set :** 25% du dataset

```
from sklearn.model_selection import train_test_split

num_test = 0.25
X_train, X_test, y_train, y_test = train_test_split(X_all, y_all, test_size=num_test, random_state=23)
```

3. Modélisation

C. RandomForestRegressor

- Après avoir essayé 5, 10, 20, 50, 100 et 300 n_estimators, j'ai décidé de garder 50 car c'est celui qui a une précision et temps d'exécution optimals

```
Entrée [69]: algo = RandomForestRegressor(n_estimators=5)
             model = algo.fit(X_train, y_train)
             score = model.score(X_test,y_test)
             score

   Out[69]: 0.34092053637741104
```

```
Entrée [70]: algo = RandomForestRegressor(n_estimators=10)
             model = algo.fit(X_train, y_train)
             score = model.score(X_test,y_test)
             score

   Out[70]: 0.4010802447233974
```

```
Entrée [71]: algo = RandomForestRegressor(n_estimators=20)
             model = algo.fit(X_train, y_train)
             score = model.score(X_test,y_test)
             score

   Out[71]: 0.42774239350334664
```

```
Entrée [72]: algo = RandomForestRegressor(n_estimators=50)
             model = algo.fit(X_train, y_train)
             score = model.score(X_test,y_test)
             score

   Out[72]: 0.45022172186264975
```

```
Entrée [73]: algo = RandomForestRegressor(n_estimators=100)
             model = algo.fit(X_train, y_train)
             score = model.score(X_test,y_test)
             score

   Out[73]: 0.4500607978030912
```

```
Entrée [74]: algo = RandomForestRegressor(n_estimators=300)
             model = algo.fit(X_train, y_train)
             score = model.score(X_test,y_test)
             score

   Out[74]: 0.45568815764234205
```

3. API

- J'ai créé une API Django contenant une requête :
  - On entre les paramètres d'un event log pour lequel on veut prédire le temps restant avant complétion
  - L'API affiche dans une vue le résultat de la prédiction
- Ci-contre, un exemple d'utilisation sur Postman