DEPLOIEMENT D'UN MODELE DE CHURN PREDICTION AVEC PYCARET, STREAMLIT ET FASTAPI

REALISE PAR

- René Lothaire BAZIE

PLAN

INTRODUCTION

Dans ce didacticiel, nous allons former un pipeline d'apprentissage automatique à l'aide de PyCaret et créer une application Web à l'aide de 2 Frameworks open source, Streamlit et Fastapi. Ces deux applications Web seront des interfaces simples permettant aux utilisateurs professionnels de générer des prédictions sur un nouvel ensemble de données à l'aide d'un pipeline d'apprentissage automatique entraîné.

I.) PRESENTATION DES BIBLIOTHEQUES UTILISEES

I.1) PYCARET

PyCaret est une bibliothèque d'apprentissage automatique à code source libre et un outil de gestion de modèle de bout en bout intégré à Python pour automatiser les flux de travail d'apprentissage automatique. Il est incroyablement populaire pour sa facilité d'utilisation, sa simplicité et sa capacité à créer et déployer rapidement et efficacement des prototypes ML de bout en bout.

PyCaret est une bibliothèque alternative low-code qui peut remplacer des centaines de lignes de code par quelques lignes seulement. Cela rend le cycle d'expérience exponentiellement rapide et efficace.

PyCaret est simple et facile à utiliser. Toutes les opérations effectuées dans PyCaret sont séquentiellement stockées dans un Pipeline entièrement automatisé pour le déploiement.Qu'il s'agisse d'imputer des valeurs manquantes, d'encoder à chaud, de transformer des données catégorielles, d'ingénierie de caractéristiques ou même de réglage d'hyperparamètres, PyCaret automatise tout cela.

PyCaret peut être installé facilement à l'aide de pip : pip install pycaret

Pour en savoir plus sur PyCaret, consultez leur GitHub

→https://www.github.com/pycaret/pycaret.

I.2) FASTAPI

FastAPI est un framework Web moderne et rapide (haute performance) pour la création d'API avec Python 3.6+ basé sur des conseils de type Python standard. Les principales caractéristiques sont :

Rapide : Très performant, à égalité avec NodeJS et Go (merci à Starlette et Pydantic). L'un des frameworks Python les plus rapides disponibles.

Rapide à coder : augmentez la vitesse de développement des fonctionnalités d'environ 200 % à 300 %.

Facile : Conçu pour être facile à utiliser et à apprendre. Moins de temps à lire des documents.

Pour en savoir plus sur FastAPI, consultez leur GitHub

➔ https://github.com/tiangolo/fastapi

Installation de PyCaret

L'installation de PyCaret est très simple et ne prend que quelques minutes. Nous vous recommandons fortement d'utiliser un environnement virtuel pour éviter les conflits potentiels avec d'autres bibliothèques.

L'installation par défaut de PyCaret est une version simplifiée de pycaret qui installe uniquement les dépendances matérielles répertoriées ici

➔ https://github.com/pycaret/pycaret/blob/master/requirements.txt.

Lorsque vous installez la version complète de pycaret, toutes les dépendances facultatives répertoriées ici sont également installées.

➔ https://github.com/pycaret/pycaret/blob/master/requirements-optional.txt

```
# installez la version mince (par défaut)
pip install pycaret
# installer la version complète
pip install pycaret[full]
```

I.3) STREAMLIT

Streamlit est une bibliothèque Python open source qui facilite la création de belles applications Web personnalisées pour l'apprentissage automatique et la science des données. Streamlit peut être installé facilement à l'aide de pip.

pip install streamlit

I.4) UVICORN

Uvicorn est une implémentation de serveur Web ASGI pour Python.

Jusqu'à récemment, Python manquait d'une interface serveur/application minimale de bas niveau pour les frameworks asynchrones. La spécification ASGI comble cette lacune et signifie que nous sommes désormais en mesure de commencer à créer un ensemble commun d'outils utilisables dans tous les frameworks asynchrones.

Uvicorn prend actuellement en charge HTTP/1.1 et WebSockets.

II.) DESCRIPTION DU JEU DE DONNEES

Les opérateurs de télécommunications doivent prévoir avec précision le taux de désabonnement des clients pour survivre sur le marché des télécommunications. Il existe un volume énorme de dossiers clients tels que les appels, les SMS et l'utilisation d'Internet. Ces données contiennent des informations riches et précieuses sur le comportement du client et son mode de consommation. L'apprentissage automatique est un outil puissant pour l'extraction d'informations sur les clients qui peut être utile pour la prédiction de désabonnement. Bien que plusieurs chercheurs aient étudié certains types de méthodes d'apprentissage automatique, il n'existe aucun travail qui évalue différentes méthodes de différents points de vue. L'objectif de ce travail est d'évaluer les performances d'un large éventail de méthodes d'apprentissage automatique pour la prédiction du churn sous la forme d'une étude comparative.

Chaque ligne représente un client ; chaque colonne contient les attributs du client. Les jeux de données ont les attributs ou fonctionnalités suivants :

- État : chaîne
- Longueur du compte : entier
- Indicatif régional : entier

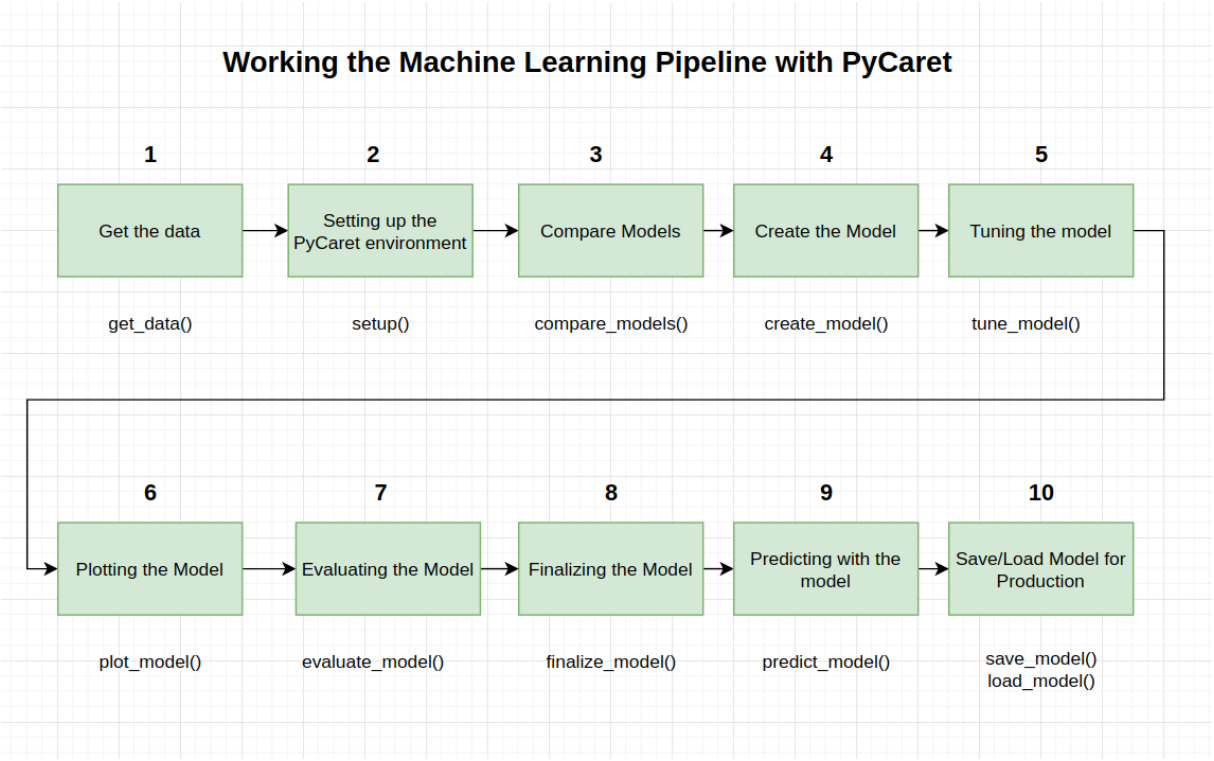
- Numéro de téléphone : entier
- Plan international : chaîne
- Forfait de messagerie vocale : chaîne
- Nombre de messages vmail : nombre entier
- Minutes totales de la journée : double
- Nombre total d'appels quotidiens : nombre entier
- Charge journalière totale : double
- Total des minutes de la veille : double
- Nombre total d'appels à la veille : nombre entier
- Charge totale de la veille : double
- Total des minutes de nuit : double
- Nombre total d'appels de nuit : nombre entier
- Tarif nuit totale : double
- Minutes internationales totales : double
- Nombre total d'appels internationaux : nombre entier
- Charge totale intl : double
- Appels au service client : nombre entier
- churn : chaîne

```
1 df.info()
2
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   state                   3333 non-null   object 
 1   account length          3333 non-null   int64  
 2   area code               3333 non-null   int64  
 3   phone number            3333 non-null   object 
 4   international plan      3333 non-null   object 
 5   voice mail plan         3333 non-null   object 
 6   number vmail messages   3333 non-null   int64  
 7   total day minutes       3333 non-null   float64
 8   total day calls         3333 non-null   int64  
 9   total day charge        3333 non-null   float64
 10  total eve minutes       3333 non-null   float64
 11  total eve calls         3333 non-null   int64  
 12  total eve charge        3333 non-null   float64
 13  total night minutes     3333 non-null   float64
 14  total night calls       3333 non-null   int64  
 15  total night charge      3333 non-null   float64
 16  total intl minutes      3333 non-null   float64
 17  total intl calls        3333 non-null   int64  
 18  total intl charge       3333 non-null   float64
 19  customer service calls  3333 non-null   int64  
 20  churn                   3333 non-null   bool   
dtypes: bool(1), float64(8), int64(8), object(4)
memory usage: 524.2+ KB
```

III.) PROCEDURES DE CREATION DU PIPELINE DE MACHINE LEARNING (CHURN PREDICTION)

Cette partie couvre l'ensemble du processus ML, depuis l'ingestion des données, le prétraitement, la formation du modèle, l'ajustement des hyperparamètres, la prédiction et le stockage du modèle pour une utilisation ultérieure.



Recréer l'intégralité de l'expérience sans PyCaret nécessite plus de 100 lignes de code dans la plupart des bibliothèques. La bibliothèque vous permet également de faire des choses plus avancées, telles que le prétraitement avancé, l'assemblage, l'empilement généralisé et d'autres techniques qui vous permettent de personnaliser entièrement le pipeline ML et sont indispensables pour tout scientifique des données.

PyCaret est une bibliothèque open source de bas niveau pour ML avec Python qui vous permet de passer de la préparation de vos données au déploiement de votre modèle en quelques minutes. Permet aux scientifiques et aux analystes de données d'effectuer efficacement des expériences itératives de science des données du début à la fin et leur permet de tirer des conclusions plus rapidement car beaucoup moins de temps est consacré à la programmation.

Lorsque Nous travaillons sur un projet de science des données, il faut généralement beaucoup de temps pour comprendre les données (EDA et feature

engineering). Alors, et si nous pouvions réduire de moitié le temps que nous passons sur la partie modélisation du projet ?

Voyons comment…

Ici, les documents de la bibliothèque pycaret. → https://pycaret.org/

❖ Module de classement

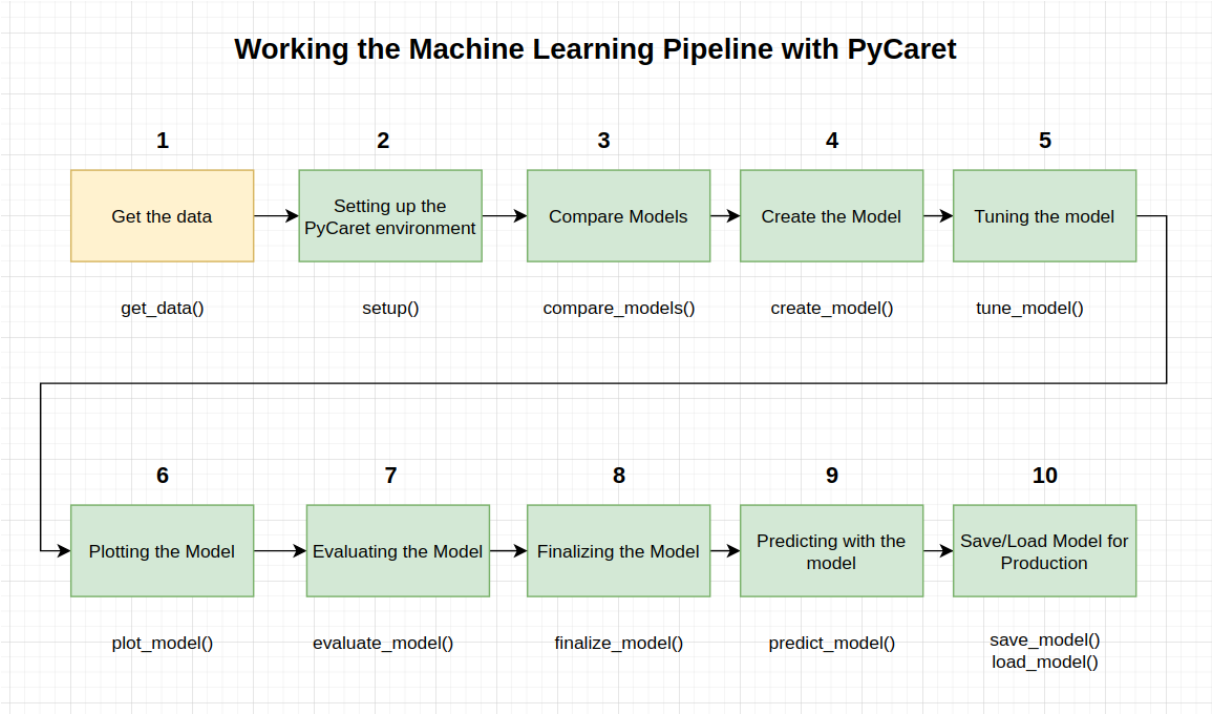
Le module de classification PyCaret (pycaret.classification) est un module d'apprentissage automatique supervisé utilisé pour classer des éléments dans un groupe binaire basé sur diverses techniques et algorithmes. Certaines utilisations courantes des problèmes de classification comprennent la prédiction du défaut du client (oui ou non), l'abandon du client (le client partira ou restera), la maladie rencontrée (positive ou négative) et ainsi de suite.

Le module de classification PyCaret peut être utilisé pour des problèmes de classification binaire ou multi-classes. Il dispose de plus de 18 algorithmes et de 14 tracés pour analyser les performances du modèle. Qu'il s'agisse de réglage d'hyperparamètres, d'assemblage ou de techniques avancées telles que l'empilement, le module de classification de PyCaret a tout pour plaire.

	Name	Reference	Turbo
ID			
lr	Logistic Regression	sklearn.linear_model.LogisticRegression	True
knn	K Neighbors Classifier	sklearn.neighbors.KNeighborsClassifier	True
nb	Naive Bayes	sklearn.naive_bayes.GaussianNB	True
dt	Decision Tree Classifier	sklearn.tree.DecisionTreeClassifier	True
svm	SVM - Linear Kernel	sklearn.linear_model.SGDClassifier	True
rbfsvm	SVM - Radial Kernel	sklearn.svm.SVC	False
gpc	Gaussian Process Classifier	sklearn.gaussian_process.GPC	False
mlp	MLP Classifier	sklearn.neural_network.MLPClassifier	False
ridge	Ridge Classifier	sklearn.linear_model.RidgeClassifier	True
rf	Random Forest Classifier	sklearn.ensemble.RandomForestClassifier	True
qda	Quadratic Discriminant Analysis	sklearn.discriminant_analysis.QDA	True
ada	Ada Boost Classifier	sklearn.ensemble.AdaBoostClassifier	True
gbc	Gradient Boosting Classifier	sklearn.ensemble.GradientBoostingClassifier	True
lda	Linear Discriminant Analysis	sklearn.discriminant_analysis.LDA	True
et	Extra Trees Classifier	sklearn.ensemble.ExtraTreesClassifier	True
xgboost	Extreme Gradient Boosting	xgboost.readthedocs.io	True
lightgbm	Light Gradient Boosting Machine	github.com/microsoft/LightGBM	True
catboost	CatBoost Classifier	catboost.ai	True

III.1) Obtenir les données

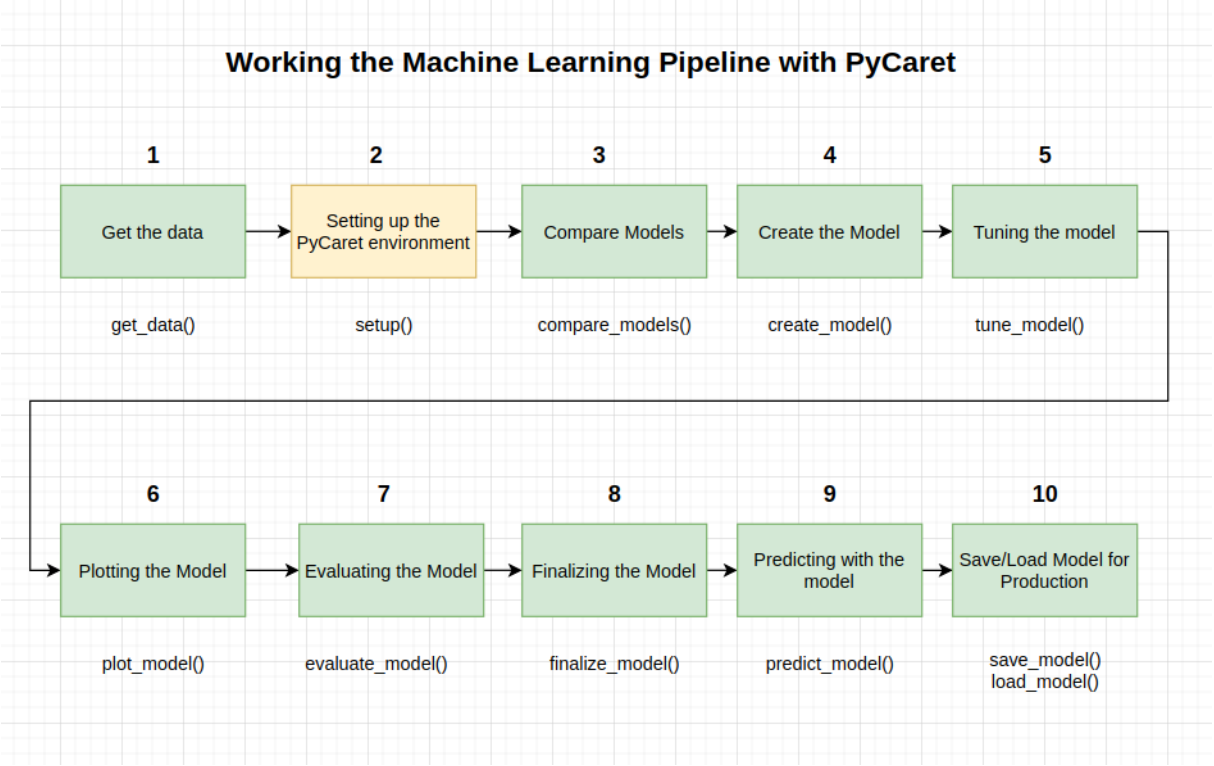
Notre jeu de données est composé de 3333 lignes et 21 colonnes.




```
df = pd.read_csv('bigml_59c28831336c6604c800002a.csv')
#print first 10 rows
df.head()
```

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34

III.2) Mise en place de l'environnement PyCaret



Configurons maintenant l'environnement Pycaret. La setup()fonction initialise l'environnement dans pycaret et crée le pipeline de transformation pour préparer les données pour la modélisation et le déploiement. setup()doit être appelé avant d'exécuter toute autre fonction dans pycaret. Il prend deux paramètres obligatoires : une dataframe pandas et le nom de la colonne cible. La majeure partie de cette partie de la configuration est effectuée automatiquement, mais certains paramètres peuvent être définis manuellement. Par exemple:

- La validation croisée K-fold est définie sur 10 par défaut ;
- " session_id" est notre classique " random_state".

```
# init setup
from pycaret.classification import *
s = setup(df, target = "churn",session_id=123, categorical_features=
          ["state","area_code", "international_plan",
           "voice_mail_plan", "customer_service_calls"],
          fix_imbalance = True )
```

Remarque : Après avoir exécuté la commande suivante, vous devez appuyer sur Entrée pour terminer le processus. Nous allons vous expliquer comment ils procèdent. Le processus de configuration peut prendre un certain temps.

Lorsque nous exécutons setup(), l'algorithme d'inférence de PyCaret déduira automatiquement les types de données de toutes les fonctionnalités en fonction de certaines propriétés. Le type de données doit être déduit correctement, mais ce n'est pas toujours le cas. Pour en tenir compte, PyCaret affiche un tableau contenant les fonctionnalités et leurs types de données déduits après setup()-exécution. Si tous les types de données sont correctement identifiés, nous pouvons appuyer sur Entrée pour continuer ou Quitter pour terminer l'expérience. Nous appuyons sur Entrée et nous devrions obtenir le même résultat que celui que nous avons obtenu ci-dessus.

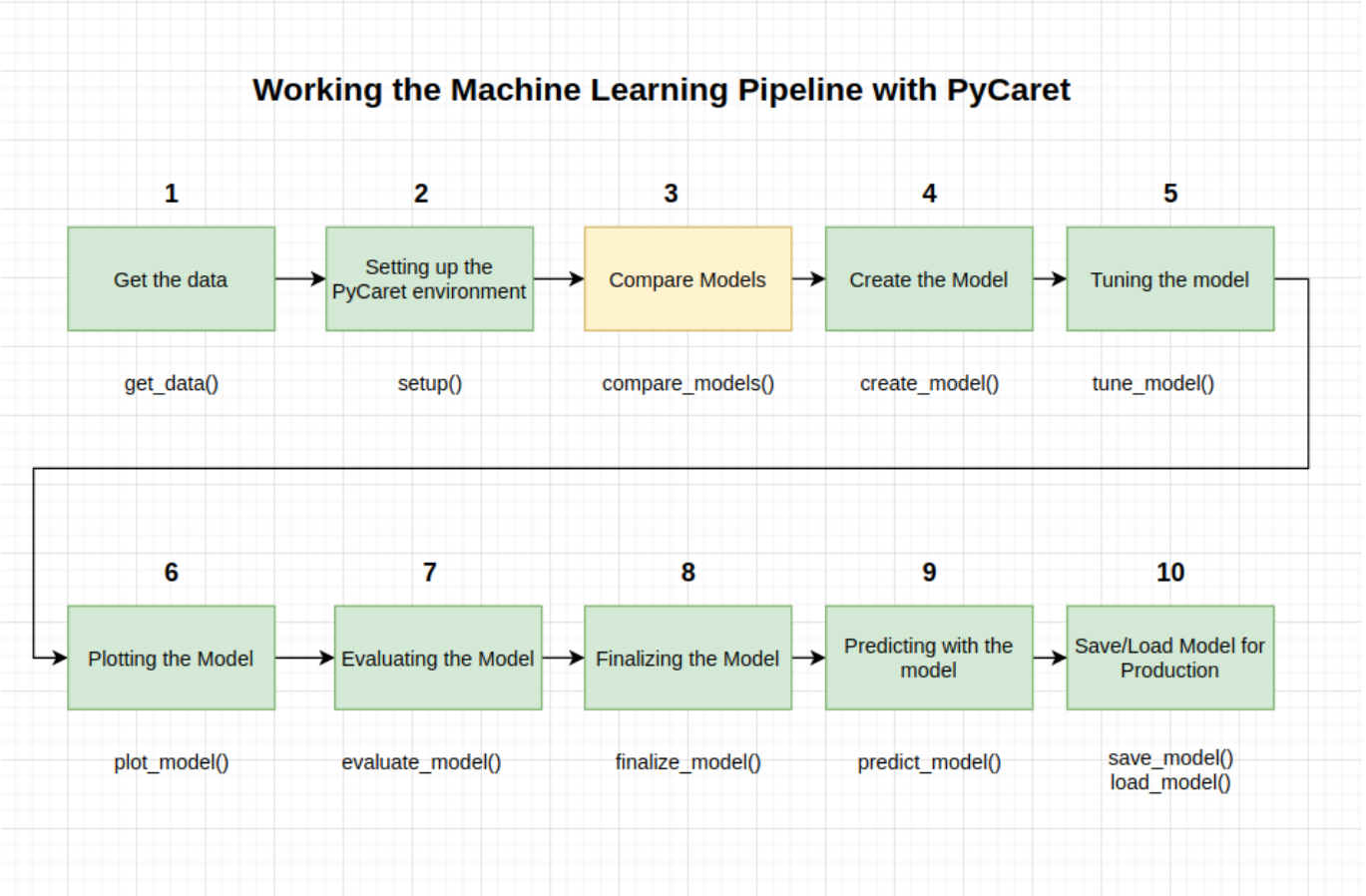
S'assurer que les types de données sont corrects est essentiel dans PyCaret, car il effectue automatiquement certaines tâches de prétraitement qui sont essentielles à toute expérience ML. Ces tâches sont effectuées différemment pour chaque type de données, ce qui signifie qu'il est très important qu'elles soient correctement configurées.

Nous pourrions écraser le type de données déduit de PyCaret en utilisant les paramètres numeric_featureset categorical_featuresdans setup(). Une fois le setup exécuté avec succès, la grille d'information contenant plusieurs informations importantes est imprimée. La plupart des informations sont liées au pipeline de prétraitement créé lorsque nous exécutons setup().

Notons comment certaines tâches impératives pour effectuer la modélisation sont gérées automatiquement, telles que l'imputation des valeurs manquantes (dans ce cas, il n'y a pas de valeurs manquantes dans les données d'apprentissage, mais nous avons toujours besoin d'imputer pour les données invisibles), l'encodage catégoriel, etc...

La plupart des setup() paramètres sont facultatifs et sont utilisés pour personnaliser le pipeline de prétraitement.

III.3) Comparer les modèles



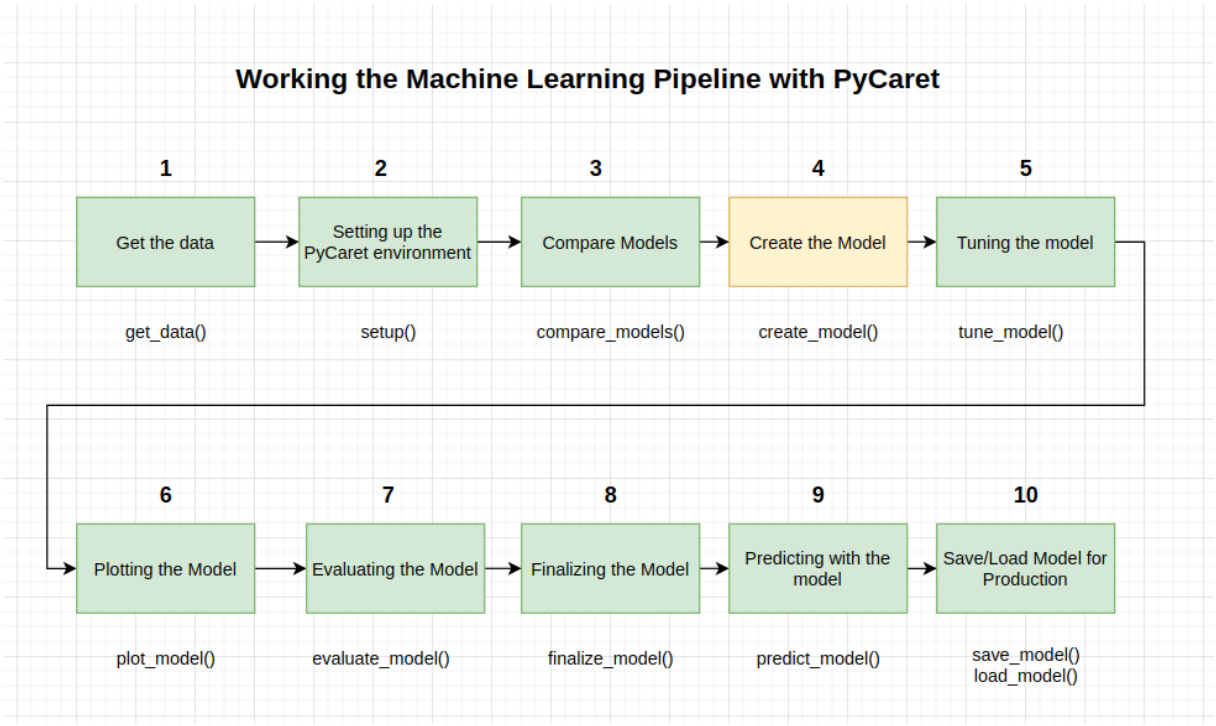
Une solution consiste à effectuer une validation croisée N-Fold. L'idée centrale ici est que nous allons effectuer tout ce processus N fois et ensuite faire la moyenne de la précision. Par exemple, dans une validation croisée 10 fois, nous ferons en sorte que l'ensemble de test représente les premiers 10% des données et nous calculerons l'exactitude, la précision, le rappel et le score F1.

Ensuite, nous ferons en sorte que la validation croisée établisse les seconds 10% des données et nous calculerons à nouveau ces statistiques. Nous pouvons effectuer ce processus 10 fois, et à chaque fois l'ensemble de test sera un élément de données différent. Ensuite, nous faisons la moyenne de toutes les précisions, et nous aurons une meilleure idée de la façon dont notre modèle fonctionne en moyenne.

```
# compare all models
best_model = compare_models()
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
xgboost	Extreme Gradient Boosting	0.9421	0.9055	0.7103	0.8825	0.7849	0.7520	0.7592	1.2240
lightgbm	Light Gradient Boosting Machine	0.9421	0.9129	0.6988	0.8913	0.7805	0.7480	0.7570	0.5770
gbc	Gradient Boosting Classifier	0.9237	0.9020	0.6931	0.7753	0.7290	0.6850	0.6882	1.0720
rf	Random Forest Classifier	0.9061	0.8954	0.4607	0.8567	0.5860	0.5399	0.5786	0.4100
dt	Decision Tree Classifier	0.8975	0.8202	0.7100	0.6432	0.6720	0.6117	0.6146	0.0770
et	Extra Trees Classifier	0.8877	0.8825	0.3953	0.7400	0.5107	0.4542	0.4850	0.3910
ada	Ada Boost Classifier	0.8602	0.8336	0.4868	0.5429	0.5113	0.4303	0.4323	0.3560
dummy	Dummy Classifier	0.8504	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0330
lr	Logistic Regression	0.8229	0.8412	0.7621	0.4483	0.5631	0.4616	0.4884	2.6690

III.4) Créer le modèle



create_modelest la fonction la plus granulaire de PyCaret et constitue souvent la base de la plupart des fonctionnalités de PyCaret. Comme son nom l'indique,

cette fonction entraîne et évalue un modèle à l'aide d'une validation croisée paramétrable avec le paramètre fold. La sortie imprime un tableau de notation indiquant par Fold the Precision, AUC, Recall, F1, Kappa et MCC.

Pour le reste de ce didacticiel, nous travaillerons avec le modèle XGBOOST car ayant été le plus performant.

```
1 xgboost_model = create_model('xgboost')
```

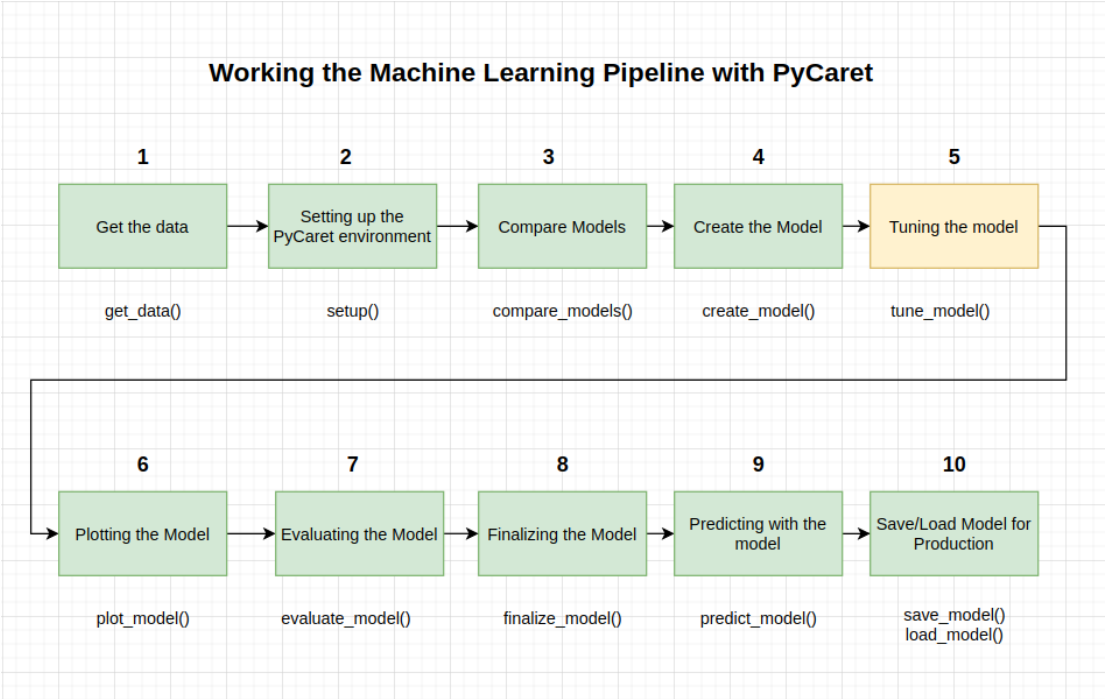
Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.9573	0.9407	0.7143	1.0000	0.8333	0.8096	0.8247
1	0.9444	0.9202	0.7143	0.8929	0.7937	0.7620	0.7683
2	0.9573	0.9881	0.7714	0.9310	0.8438	0.8192	0.8241
3	0.9356	0.8698	0.6176	0.9130	0.7368	0.7017	0.7191
4	0.9442	0.9081	0.7143	0.8929	0.7937	0.7619	0.7682
5	0.9227	0.8284	0.6571	0.7931	0.7188	0.6744	0.6784
6	0.9356	0.9045	0.7143	0.8333	0.7692	0.7321	0.7350
7	0.9528	0.8550	0.7714	0.9000	0.8308	0.8035	0.8067
8	0.9227	0.9082	0.6000	0.8400	0.7000	0.6571	0.6693
9	0.9485	0.9320	0.8286	0.8286	0.8286	0.7983	0.7983
Mean	0.9421	0.9055	0.7103	0.8825	0.7849	0.7520	0.7592
Std	0.0121	0.0433	0.0672	0.0572	0.0491	0.0555	0.0541

Il y a 18 classificateurs disponibles dans la bibliothèque de modèles PyCaret. Pour voir une liste de tous les classificateurs, consultez la documentation ou utilisez la models() fonction pour afficher la bibliothèque.

```
1 models()
```

ID	Name	Reference	Turbo
lr	Logistic Regression	sklearn.linear_model._logistic.LogisticRegression	True
knn	K Neighbors Classifier	sklearn.neighbors._classification.KNeighborsCl...	True
nb	Naive Bayes	sklearn.naive_bayes.GaussianNB	True
dt	Decision Tree Classifier	sklearn.tree._classes.DecisionTreeClassifier	True
svm	SVM - Linear Kernel	sklearn.linear_model._stochastic_gradient.SGDC...	True
rbfsvm	SVM - Radial Kernel	sklearn.svm._classes.SVC	False
gpc	Gaussian Process Classifier	sklearn.gaussian_process._gpc.GaussianProcessC...	False
mlp	MLP Classifier	sklearn.neural_network._multilayer_perceptron....	False
ridge	Ridge Classifier	sklearn.linear_model._ridge.RidgeClassifier	True
rf	Random Forest Classifier	sklearn.ensemble._forest.RandomForestClassifier	True
qda	Quadratic Discriminant Analysis	sklearn.discriminant_analysis.QuadraticDiscrim...	True
ada	Ada Boost Classifier	sklearn.ensemble._weight_boosting.AdaBoostClas...	True
gbc	Gradient Boosting Classifier	sklearn.ensemble._gb.GradientBoostingClassifier	True
lda	Linear Discriminant Analysis	sklearn.discriminant_analysis.LinearDiscrimina...	True
et	Extra Trees Classifier	sklearn.ensemble._forest.ExtraTreesClassifier	True
xgboost	Extreme Gradient Boosting	xgboost.sklearn.XGBClassifier	True

III.5) Réglage du modèle



Lors de la création d'un modèle à l'aide de la create_model()fonction, les hyperparamètres par défaut sont utilisés pour entraîner le modèle. Pour régler les hyperparamètres, la tune_model()fonction est utilisée. Cette fonction ajuste automatiquement les hyperparamètres d'un modèle à l'aide de la recherche de grille aléatoire dans un espace de recherche prédéfini.

La sortie imprime une grille de score indiquant la précision, l'AUC, le rappel, la précision, la F1, le Kappa et le MCC par pli pour le meilleur modèle. Pour utiliser une grille de recherche personnalisée, nous pouvons passer le custom_grid paramètre dans la tune_model fonction.

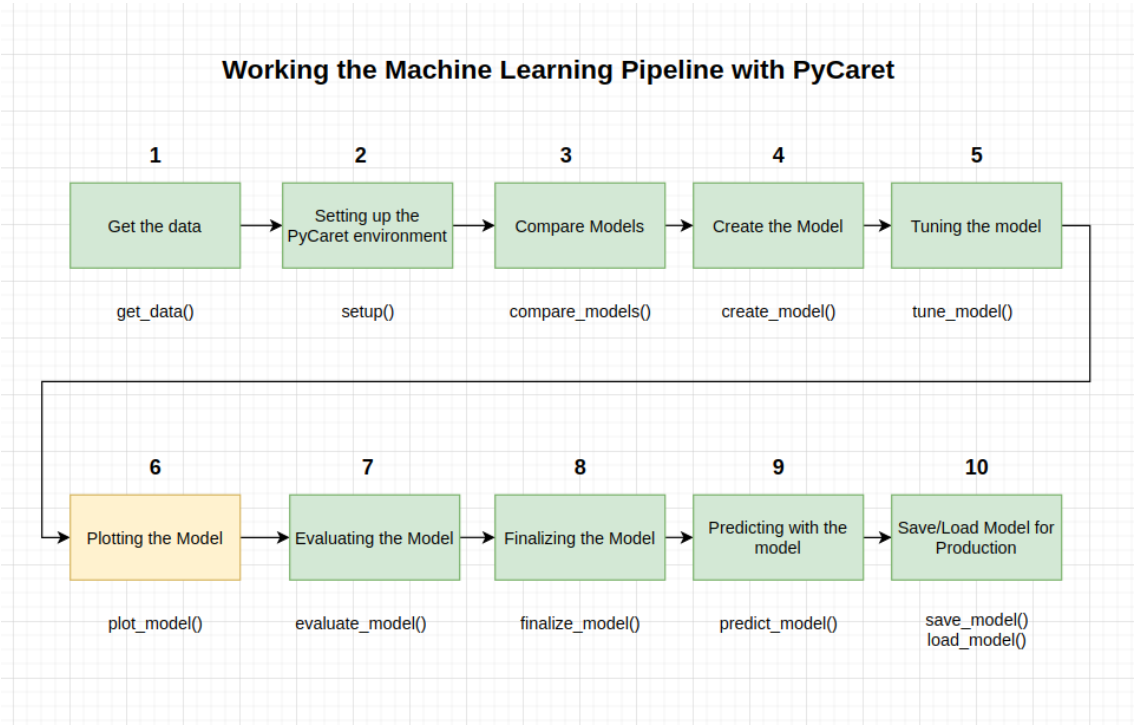
```
1 tuned_xgboost_model = tune_model(xgboost_model)
```

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.9530	0.9393	0.8571	0.8333	0.8451	0.8174	0.8175
1	0.9274	0.8702	0.7429	0.7647	0.7536	0.7110	0.7111
2	0.9402	0.9757	0.9429	0.7333	0.8250	0.7896	0.7987
3	0.9013	0.8490	0.6471	0.6667	0.6567	0.5991	0.5992
4	0.9485	0.9501	0.8857	0.7949	0.8378	0.8073	0.8090
5	0.8884	0.8423	0.6857	0.6154	0.6486	0.5826	0.5838
6	0.9185	0.9014	0.8571	0.6818	0.7595	0.7112	0.7179
7	0.9356	0.8587	0.8000	0.7778	0.7887	0.7508	0.7509
8	0.9056	0.8924	0.8000	0.6512	0.7179	0.6620	0.6670
9	0.9056	0.9180	0.8571	0.6383	0.7317	0.6759	0.6867
Mean	0.9224	0.8997	0.8076	0.7157	0.7565	0.7107	0.7142
Std	0.0208	0.0432	0.0876	0.0710	0.0663	0.0781	0.0783

Par défaut, tune_modeloptimise la précision, mais cela peut être modifié à l'aide du optimize paramètre.

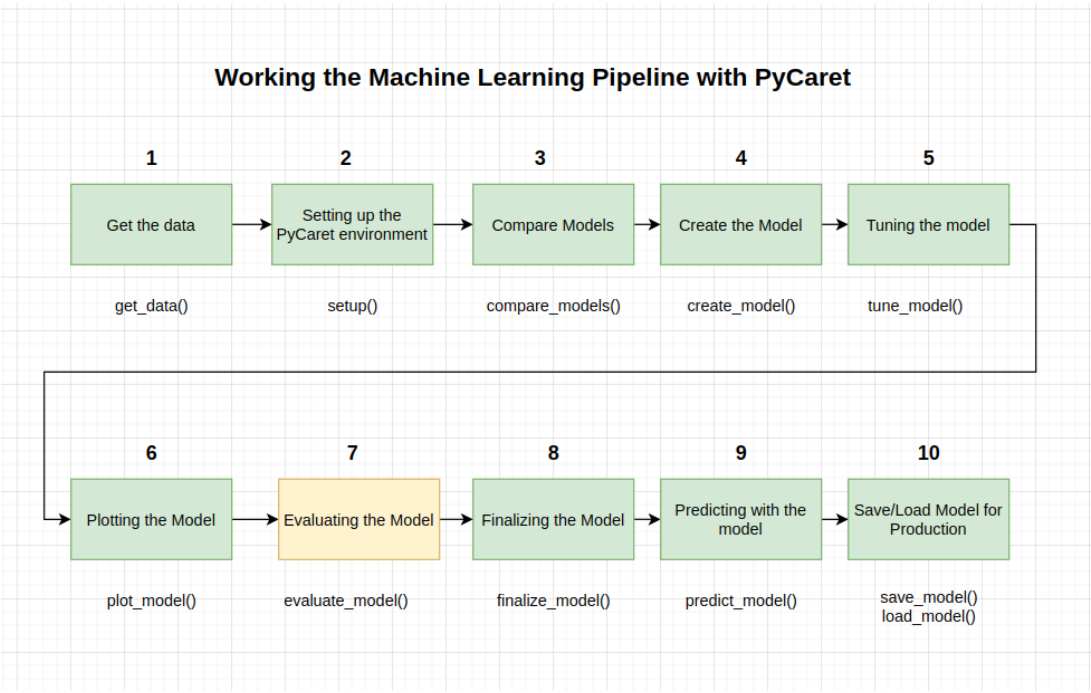
Par exemple : tune_model(dt, optimize='AUC')recherchera les hyperparamètres d'un classificateur d'arbre de décision qui donne l'AUC la plus élevée au lieu de la précision.

III.6) Tracer le modèle



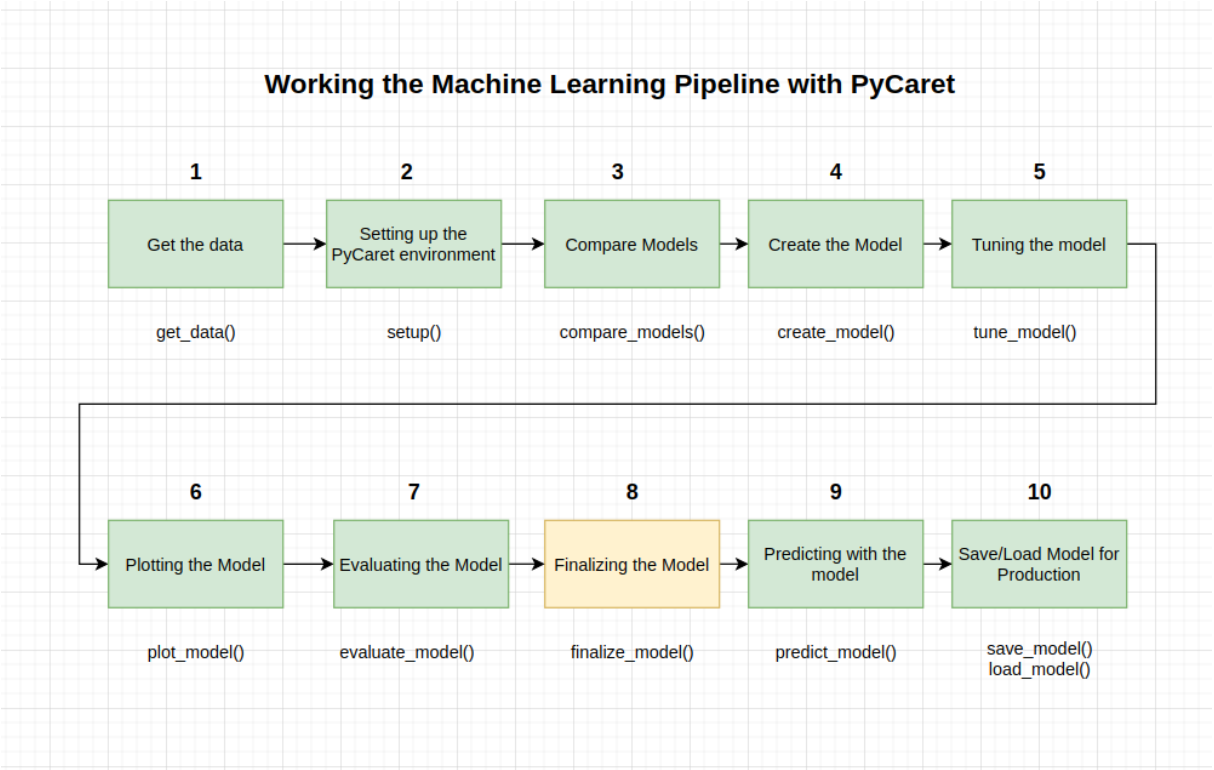
Avant de finaliser le modèle (étape 8), la plot_model() fonction peut être utilisée pour analyser les performances à travers différents aspects tels que AUC, confusion_matrix, limite de décision, etc. Cette fonction prend un objet modèle formé et renvoie un graphique.

III.7) Évaluation du modèle



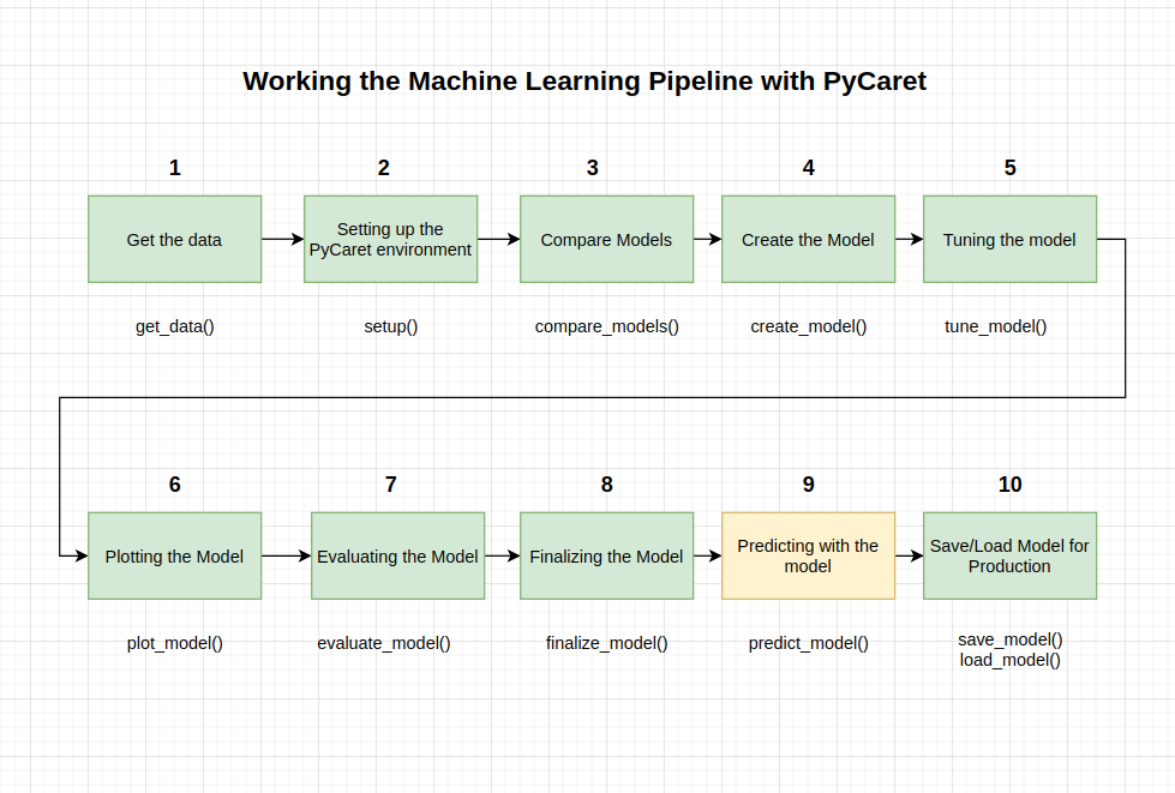
Une autre façon d'analyser les performances du modèle consiste à utiliser la evaluate_model() fonction qui affiche une interface utilisateur pour tous les graphiques disponibles pour un modèle donné. En interne, il utilise la plot_model() fonction.

III.8) Finalisation du Modèle



Ce flux de travail vous mènera éventuellement au meilleur modèle à utiliser pour faire des prédictions sur des données nouvelles et inédites. La finalize_model() fonction ajuste le modèle à l'ensemble de données complet. Le but de cette fonction est de former le modèle sur l'ensemble de données complet avant qu'il ne soit déployé en production.

III.9) Prédire avec le modèle



Avant de finaliser le modèle, il est conseillé d'effectuer une dernière vérification en prédisant les données de sortie et en passant en revue les métriques d'évaluation. Le label est la prédiction et le score est la probabilité de la prédiction.

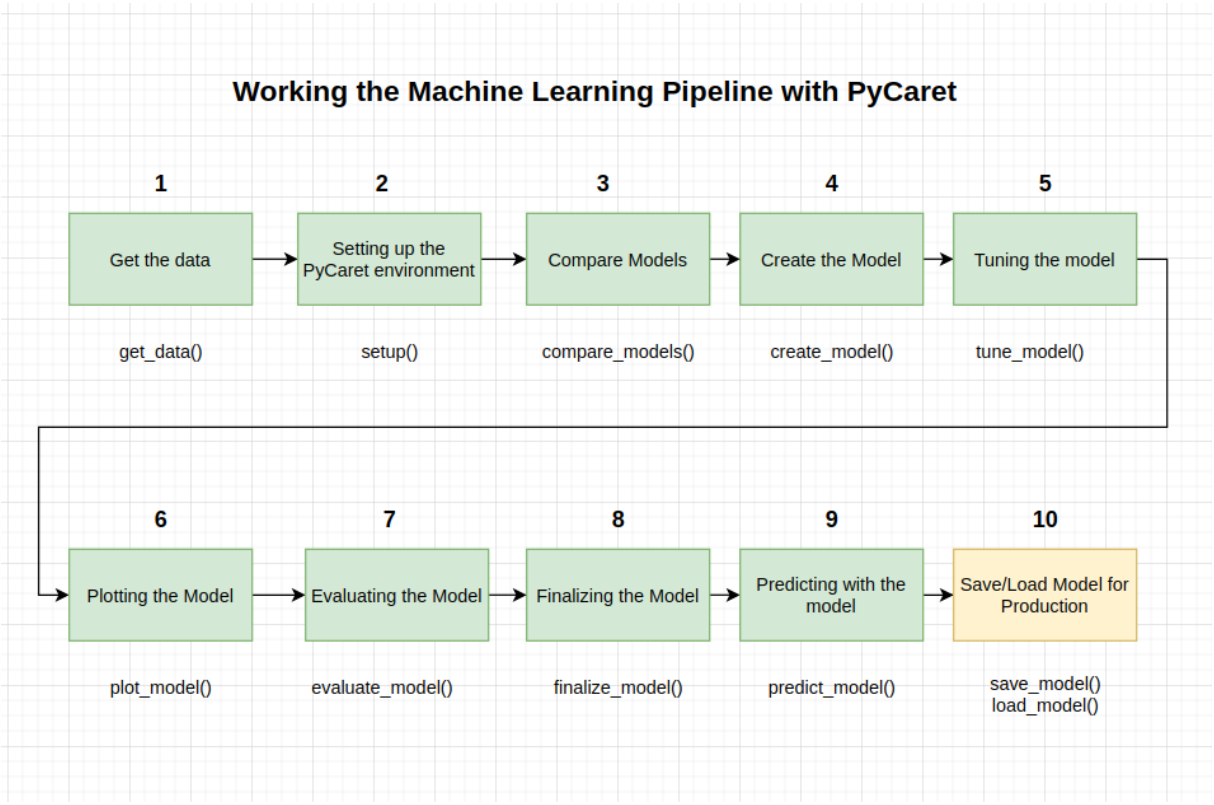
```
# from joblib import load
from pycaret.classification import load_model, predict_model
```

```
cols = ['state', 'account_length', 'area_code','international_plan',

records = [[16,12,5,0,1,25,26.1,10,5.07,7.4,9,1.78,2.7,9,11.01,10.0,

data = pd.DataFrame(data = records, columns=cols)

# predict_model(model, data)
pred = predict_model(estimator= final_tuned_xgboost_model, data = da
pred
```

	state	account_length	area_code	international_plan	voice_mail_plan	number_vmail_me
0	16	12	5	0	1	

1 rows × 21 columns

III.10) Enregistrer/Charger le modèle pour la production



Pour charger un modèle enregistré à une date future dans le même environnement ou dans un environnement alternatif, nous utiliserions la load_model()fonction de PyCaret, puis appliquerions facilement le modèle enregistré à de nouvelles données invisibles pour la prédiction.

❖ Avantages et inconvénients

Comme pour toute nouvelle bibliothèque, il y a encore place à l'amélioration. Nous énumérerons certains des avantages et des inconvénients que nous avons trouvés lors de l'utilisation de la bibliothèque.

- Avantages :

- Cela rend la partie modélisation de votre projet beaucoup plus facile.
- Vous pouvez créer de nombreuses analyses différentes avec une seule ligne de code.
- Oubliez la transmission d'une liste de paramètres lors de l'ajustement du modèle. PyCaret le fait automatiquement pour vous.
- Vous avez de nombreuses options différentes pour évaluer le modèle, encore une fois, avec une seule ligne de code puisqu'il est construit sur

des bibliothèques ML célèbres, vous pouvez facilement le comparer avec votre méthode traditionnelle

- Les inconvénients :

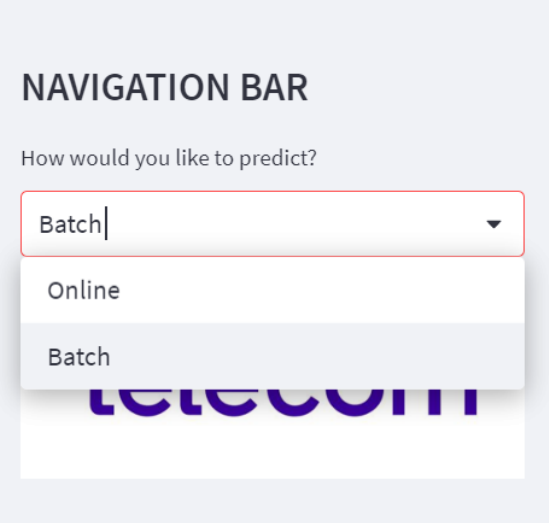
- La bibliothèque n'en est qu'à ses premières versions, elle n'est donc pas assez mature et est sensible aux bogues. Pas grave pour être honnête
- Comme toutes les bibliothèques Auto ML, c'est une boîte noire, vous ne pouvez donc pas vraiment voir ce qui se passe à l'intérieur. Par conséquent, je ne le recommanderais pas aux débutants.
- Cela pourrait rendre le processus d'apprentissage un peu superficiel.

IV.) PROCEDURES DE CREATION D'UNE APPLICATION WEB A PARTIR DU PIPELINE DE MACHINE LEARNING (CHURN PREDICTION) AVEC STREAMLIT

Maintenant que notre pipeline et notre modèle d'apprentissage automatique sont prêts, nous allons commencer à créer une application Web frontale capable de générer des prédictions sur de nouveaux points de données. Cette application prendra en charge les prédictions « en ligne » ainsi que « par lots » via un téléchargement de fichier csv. Décomposons le code de l'application en deux parties principales :

- **En-tête / Mise en page**

Cette section importe des bibliothèques, charge le modèle formé et crée une mise en page de base avec un logo en haut, une image jpg et un menu déroulant sur la barre latérale pour basculer entre la prédiction "En ligne" et "Batch".



- **Prédictions en ligne**

Cette section traite de la première fonctionnalité de l'application, à savoir la prédiction en ligne (un par un). Nous utilisons des widgets éclairés tels que la saisie de nombres, la saisie de texte, le menu déroulant et la case à cocher pour collecter les points de données utilisés pour former le modèle

- **Prédictions par lots**

Cette partie traite de la seconde fonctionnalité à savoir la prédiction par batch. Nous avons utilisé le widget file_uploader de streamlit pour télécharger un fichier csv, puis appelé la fonction native predict_model() de PyCaret pour générer des prédictions qui sont affichées avec la fonction write() de streamlit.

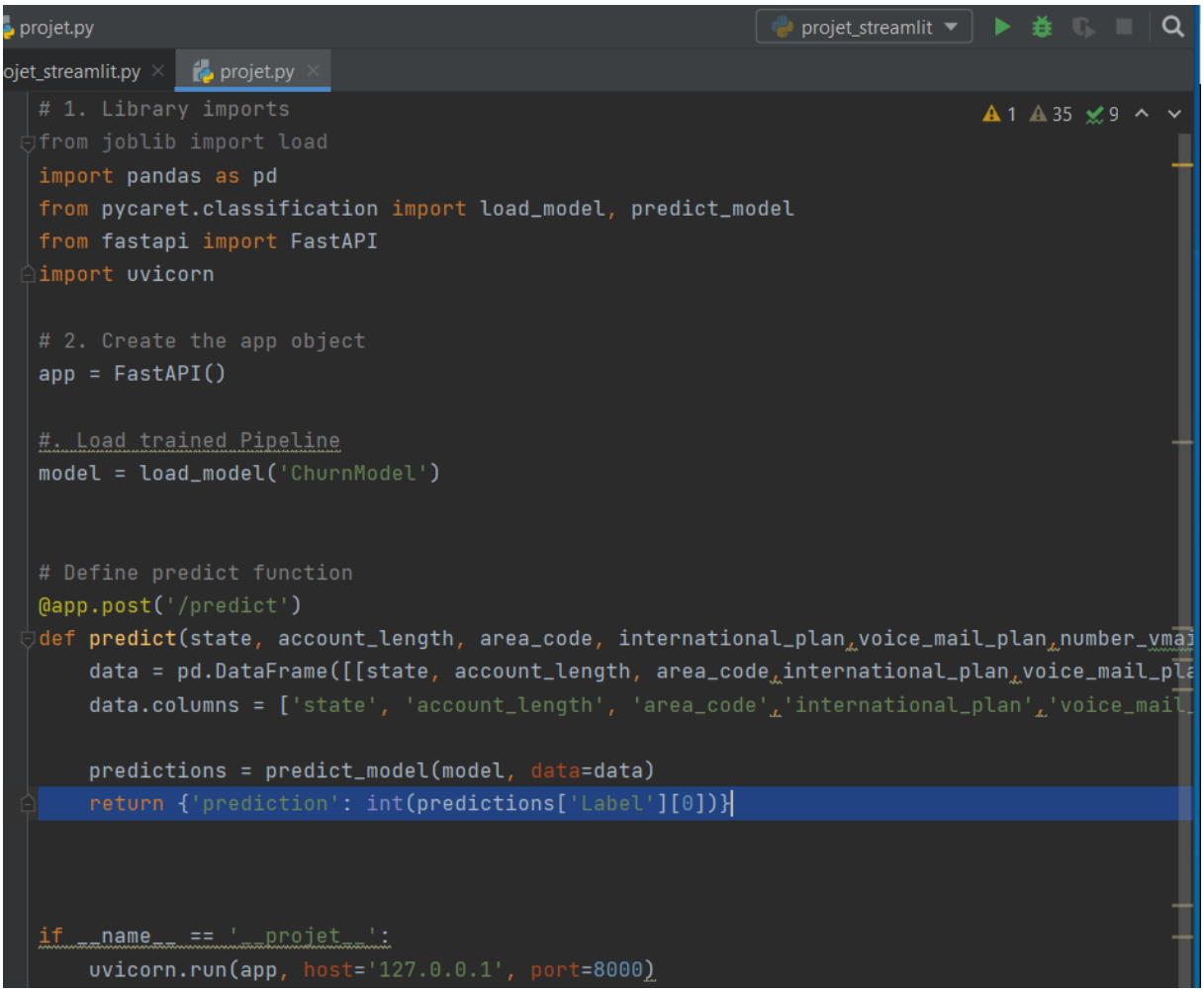
- Test de l'application

Ouvrez l'invite Anaconda, accédez au dossier de votre projet et exécutez le code suivant :

streamlit exécuter nomfichier.py

V.) PROCEDURES DE DEPLOIEMENT DU PIPELINE DE MACHINE LEARNING (CHURN PREDICTION) AVEC FASTAPI

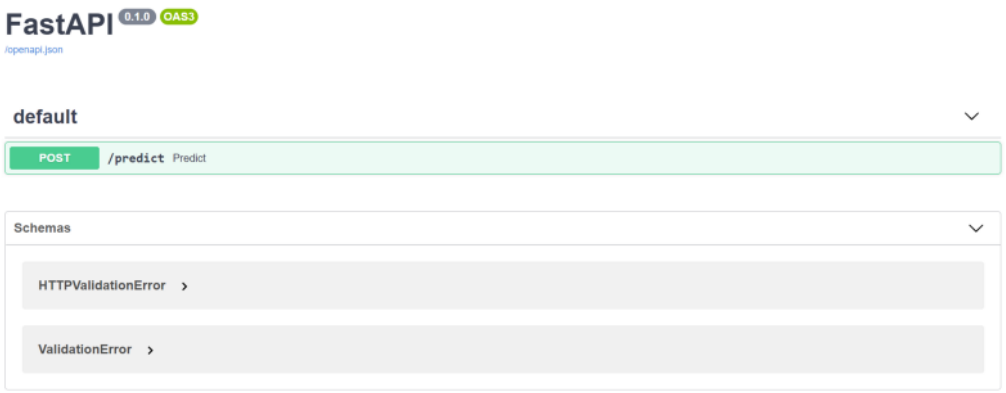
Le déploiement de modèles d'apprentissage automatique est le processus de mise à disposition de modèles en production où les applications Web, les logiciels d'entreprise et les API peuvent utiliser le modèle formé en fournissant de nouveaux points de données et en générant des prédictions. Normalement, les modèles d'apprentissage automatique sont construits de manière à pouvoir être utilisés pour prédire un résultat (valeur binaire, c'est-à-dire 1 ou 0 pour la classification, valeurs continues pour la régression, étiquettes pour le clustering, etc. Il existe deux grandes façons de générer des prédictions (i) prédire par batch ; et (ii) prédire en temps réel. Cette partie montrera comment nous pouvons déployer nos modèles de machine Learning en tant qu'API pour prédire en temps réel.



Nous pouvons ensuite exécuter ce script en exécutant la commande suivante dans notre invite de commande. Nous devons être dans le même répertoire que le script python et le fichier pickle du modèle avant d'exécuter cette commande.

uvicorn main:app --reload

Cela initialisera un service API sur votre hôte local. Sur notre navigateur, tapons http://localhost:8000/docs et cela devrait afficher quelque chose comme ceci :



CONCLUSION

Ce projet a couvert l'ensemble du processus ML, depuis l'ingestion des données, le prétraitement, la formation du modèle, l'ajustement des hyperparamètres, la prédiction et le stockage du modèle pour une utilisation ultérieure. Nous avons réalisé toutes ces étapes en moins de 10 commandes naturellement construites et très intuitives à retenir, telles que create_model(), tune_model(), compare_models(). Recréer toute l'expérience sans PyCaret aurait nécessité plus de 100 lignes de code dans la plupart des bibliothèques.

La bibliothèque vous permet également de faire des choses plus avancées, telles que le prétraitement avancé, l'assemblage, l'empilement généralisé et d'autres techniques qui vous permettent de personnaliser entièrement le pipeline ML et sont indispensables pour tout scientifique des données.

FIN