Développement

1) NA
2)
Cas de tests :
Cf. fichier CasPratiques.cpp
Methode TestAbreger

Développement, algorithmes

1) NA
2)
Cas de tests :

TestCase 1: Assert No error for mondays, tuesdays and sundays
TestCase 2: Assert Data on full week except for one day
TestCase 3: Assert Data on full week
TestCase 4: Assert Data on two weeks
TestCase 5: Assert Data on several years with obvious expected outcome (data repeated over weeks)

La complexité algorithmique de la méthode getMostFrequentDates est O(n*log(n)) pour les tris qui sont effectués:

- la méthode getMostFrequentDates
    - fait un appel à la méthode getTotalUseByDates qui est appellée une fois
    - fait un nouveau tri
    - parcourt les k premières données
- la méthode getTotalUseByDates
    - parcourt toutes les entrées
    - fait un tri
    - reparcourt les entrées

3) Cf. projet
(Je n'ai pas compris si la méthode getNextCount pouvait retourner plusieurs fois des valeurs avec la même date mais j'ai imaginé que oui.)
Ainsi, pour "fusionner" les données qui ont la même date, il est plus efficace de trier les données par date pour ne pas avoir à chercher dans toute la liste si on a déjà rencontré cette nouvelle date pour chaque nouvelle donnée

Développement, architecture

(PS : Je n'ai jamais fait de deep learning, pour l'instant, tout ce que je sais est issu de ma culture générale.)

1)

- La banque de données doit être très grande pour que l'algorithme d'apprentissage ait beaucoup de matière pour s'entrainer.
- Chaque image doit être labelisée : pour guider l'apprentissage, l'algorithme a besoin de savoir si un bagage abandonné figure sur l'image qu'il inspecte.

2)

Je ne sais pas vraiment. J'imagine qu'il faut commencer par chercher sur internet si une telle banque de données existante est accessible, à défaut une proche.
A défaut, il faudra trouver des images de lieux publiques, ce qui n'est pas compliqué et manuellement en labeliser le plus possible.

3)

Pour augmenter artificiellement la taille d'une banque d'images, on peut traiter celles existantes pour en produire de nouvelles en gardant le label d'origine.
Les transformations d'images envisageables sont

- la création d'images miroir
- les rotations
- les translations
- les réductions, agrandissements
- l'altération parcellaire aléatoire de l'image

4)

Cf. repo DataBankAugmenter

Dans mon code, j'ai mis une classe qui permet d'effectuer des transformations mirroir, et une autre pour les rotations.

PS : J'ai considéré dans l'ossature de mon code (dans la classe dataAugmentationUtils) que les images étaient stockées dans un système de fichier, mais utiliser une base de données spécialisée pour les images est probablement une meilleure idée.

DevOps, intégration continue

1)

La démarche DevOps cherche à unifier les pratiques de développement logiciel pour que tous les projets soient conduits le proprement en termes de suivi/gestion, de développement, de testing, de livraison etc…

L'intérêt est qu'on s'assure ainsi de bien faire les choses et qu'on ait des réflexes de bonnes pratiques qui ne changent pas d'un projet un autre.

L'inconvénient de la démarche DevOps est qu'elle ne produit pas de valeur pour le client, elle permet uniquement de bien gérer un projet et réduire les facteurs de risque. De plus, comme elle est peu scalable selon la taille du projet et demande beaucoup de ressources humaines ou matérielles, il peut être difficile à mettre en place ces pratiques pour des petits projets, surtout dans les premières phases.

2)
Une base de code est saine et maintenable si elle ne produit pas trop d'erreurs et de fuites mémoire, qu'elle est régulièrement testée (avec des tests à jour et une couverture de code élevée). De plus, il faut que son architecture soit cohérente et bien découpée en petites parties indépendantes.
Pour s'assurer de la qualité du code, il faut utiliser des outils d'analyse contraignant le formatage du code (nommage, indentation, commentaires, taille des fonctions, taille des fichiers, indice de complexité cyclomatique…), et le taux de couverture.

3)
(PS : je ne suis pas sûr d'avoir bien compris la question)
Pour créer un pipeline d'intégration continue :
- définir les méthodes de travail par tâche et par version à déployer (branche de code, tests, développement, condition de merge).
- avoir une plateforme de tests proche de l'environnement de production qui s'assure continuellement que le code actuel tourne de manière nominale
- surveiller les indicateurs clés et conserver une architecture de code claire et cohérente

Déploiement d'un client lourd

(PS : Je n'ai pour l'instant pas beaucoup été confronté directement aux problématiques de déploiement d'application et je ne suis pas sûr d'avoir compris si la question fait référence au processus de livraison/déploiement ou au choix d'architecture système ou aux deux.)

Pour déployer la solution, on peut créer un environnement portable et autonome contenant tout ce qu'il faut pour lancer le client lourd, mais dans le cas décrit dans cette question où l'on maîtrise complètement l'environnement dans lequel la solution sera déployée, cela n'est pas nécessaire.

Une bonne pratique consiste à décorréler les différents systèmes, donc il ne faut pas que la procédure de déploiement du client lourd gère aussi le déploiement de l'API ou dépende de celle-ci.
Le client lourd, pour chacune de ses versions doit définir (à titre informatif) la version minimale dans laquelle l'API doit être lancée pour qu'elle puisse tourner sans problème, mais ce n'est pas son rôle de le vérifier ; cependant, on peut imaginer qu'en cas d'erreur lors d'une requête et que le serveur Rest tourne, le client tente de récupérer le numéro de version avant de remonter l'erreur.

Les seules différences à ce que l'entreprise gère l'API Rest sont que :

- l'entreprise peut coordonner comme elle le veut les deux déploiements
- il est plus facile de faire reconnaître et autoriser le client lourd pour l'API
- en cas de comportement inattendu lors d'une requête, l'entreprise peut investiguer l'API en la considérant comme une boîte blanche.

Si l'on veut s'assurer de la disponibilité du service, il faut redonder les applications (à chaud ou à froid) pour que le service reste disponible si l'on veut monter de version ou si un serveur tombe. Il faudra aussi redéployer la version redondée lorsque l'instance principale sera disponible.

Bonus)
Pour inscrire le déploiement dans une démarche DevOps, il faut automatiser le processus de livraison / déploiement, le maintenir (que les différents scripts utilisés au cours de la procédure soient versionnés). Il faut également tâcher de rendre ce processus le plus générique possible et distinguer ce qui est propre au projet, ce qui est dépendant de l'architecture mise en place, de l'environnement (plateforme…), afin de factoriser ce qui peut être réutilisable dans d'autres projets.