Une image contenant Police, texte, logo, Graphique

Le contenu généré par l’IA peut être incorrect.

CESI – Ecole d’ingénieur

Promotion : FISE A3 - 24/25 - Science du numérique.

Rapport Technique

Par Abdisamad Abdourahman Abdillahi

Table des matières

[Introduction 3](#_Toc203031424)

[1. Contexte et Outils 4](#_Toc203031425)

[1.1. Spaceship Titanic : prédire les passagers transportés 4](#_Toc203031426)

[1.2. Outils utilisés 4](#_Toc203031427)

[2. Modèle IA 6](#_Toc203031428)

[2.1. Description des données 6](#_Toc203031429)

[2.2. Préparation des données 7](#_Toc203031430)

[2.2.1. Valeurs manquantes 7](#_Toc203031431)

[2.2.2. Transformation des colonnes 8](#_Toc203031432)

[2.2.3. Encodage et normalisation 10](#_Toc203031433)

[2.3. Modèles de classification 11](#_Toc203031434)

[2.3.1. Séparation des données 11](#_Toc203031435)

[2.3.2. Choix et évaluation des modèles de classification 12](#_Toc203031436)

[2.3.3. Modèle Final 13](#_Toc203031437)

[3. Interface avec Streamlit et Docker 13](#_Toc203031438)

[3.1. Présentation de Streamlit 13](#_Toc203031439)

[3.2. Fonctionnalités demandées 13](#_Toc203031440)

[3.3. Interface développée 14](#_Toc203031441)

[3.3.1. Prédiction individuelle 14](#_Toc203031442)

[3.3.2. Prédiction par lot (fichier CSV) 14](#_Toc203031443)

[3.6. Déploiement dockers 15](#_Toc203031444)

[4. Bilan et retour d’expériences 16](#_Toc203031445)

[Conclusion 16](#_Toc203031446)

[Annexe : 17](#_Toc203031447)

[Annexe 1 : 17](#_Toc203031448)

[Annexe 2 18](#_Toc203031449)

[Annexe 3 : 18](#_Toc203031450)

[Annexe 4 : 19](#_Toc203031451)

# Introduction

Ce projet se situe dans un monde de science-fiction où le vaisseau spatial Titanic navigue vers une destination éloignée. Il a été confronté à un problème spatio-temporelle qui a entraîné la disparition de certains passagers. L'objectif est de concevoir un modèle d'intelligence artificielle qui peut déterminer si un passager a été déplacé ou non, en se basant sur des informations incomplètes recueillies à bord.Ce projet intègre l'étude de données, la création de modèles statistiques et l'élaboration d'une application intuitive en utilisant des outils modernes comme Python, Scikit-learn, Streamlit et Docker.

# Contexte et Outils

## Spaceship Titanic : prédire les passagers transportés

Dans le cadre de ce projet, nous avons été plongés dans un monde de science-fiction se situant en l'an 2912. Le Titanic, un vaisseau interstellaire transportant plus de 13 000 passagers, percute une anomalie spatio-temporelle qui orbite autour de 55 Cancri E. Bien que le vaisseau ne soit pas totalement détruit, près de la moitié des passagers ont disparu de manière énigmatique, ayant été projetés dans une autre dimension.  
Le défi consiste à concevoir une solution d'intelligence artificielle capable de reconnaître, à partir des données partiellement extraites du système informatique du vaisseau, les passagers qui ont été transportés. Ces informations englobent une variété de détails concernant les passagers, comme leur âge, leur monde natal, leurs dépenses effectuées à bord ou encore leur statut de VIP.  
La problématique posée peut être formalisée comme une problème de classification binaire supervisée, où l'on cherche à prédire la variable cible Transported, en se basant sur des variables explicatives.

## Outils utilisés

Le développement de cette solution a nécessité la mobilisation d’un ensemble d’outils et de technologies adaptés au cycle complet d’un projet d’intelligence artificielle, depuis la phase d’exploration des données jusqu’au déploiement d’une application fonctionnelle.

**Environnement de développement**

* **Python** : langage principal utilisé pour le traitement des données, la modélisation et l’intégration des composants de l’application.
* **Jupyter Notebook / IDE Python** : utilisés pour l’exploration initiale, la visualisation et la construction du pipeline de données.

**Traitement et modélisation des données**

* **Pandas** : bibliothèque incontournable pour la manipulation des données structurées (fichiers .csv, jointures, nettoyage, transformations).
* **NumPy** : pour les calculs numériques, vecteurs et matrices.
* **Scikit-learn** : pour le développement du modèle de machine learning (classification), incluant la gestion des pipelines de transformation, l'entraînement, l’évaluation et la sérialisation.

**Interface utilisateur**

* **Streamlit** : outil de développement d’interfaces web interactives en Python, utilisé pour créer une application intuitive, accessible via un simple navigateur, sans développement HTML/CSS.

**Déploiement**

* **Docker** : utilisé pour conteneuriser l'application Streamlit, facilitant son exécution sur n’importe quelle machine, quel que soit le système d’exploitation, avec toutes les dépendances embarquées.
* **Dockerfile** : script de construction de l’image Docker, décrivant l’environnement nécessaire pour exécuter l’application.

**Sérialisation et stockage**

* **Pickle / Joblib** : bibliothèques permettant de sauvegarder le modèle entraîné pour une réutilisation directe dans l’application web, sans avoir à le réentraîner.

**Versionnement et livrables**

* **Git** : gestion des versions du code tout au long du projet.
* **GitHub** : dépôt distant hébergeant le code source, les ressources du projet et la documentation, facilitant le partage et la revue de code.

## Objectifs du projet

Les objectifs de ce projet sont définis selon la méthode SMART :

1. Élaborer un modèle prédictif capable de détecter, avec un F1-score supérieur à 0,80, si un passager a été transporté dans une autre dimension.   
2. Créer une application web interactive à l'aide de Streamlit qui comprend un formulaire de saisie, un bouton pour des prédictions instantanées, et qui intègre aussi une fonctionnalité pour importer des fichiers CSV afin de réaliser des prédictions par lots.   
3. Garantir le déploiement via Docker en empaquetant l’intégralité de l'application pour permettre un déploiement rapide en moins de 5 minutes sur toute machine Linux équipée de Docker.

# Modèle IA

## Description des données

Le jeu de données dans le cadre du projet *Spaceship Titanic* est constitué de fichiers CSV représentant les informations personnelles des passagers à bord du vaisseau. Ces données sont divisées en deux partie : données d’entraînement (**train.csv**) et données de test (**test.csv**).

* **Données d’entrainement : train.csv**

Ce fichier contient les informations personnelles de près de 2/3 des passagers (environ 8 700). Il est utilisé pour entraîner le modèle de machine learning.

Les colonnes disponibles sont les suivantes :

* **PassengerId** : Identifiant unique du passager. Il suit le format gggg\_pp, où gggg indique le groupe de voyage (souvent familial) et pp représente la position du passager dans ce groupe.
* **HomePlanet** : Planète d’origine du passager, généralement son lieu de résidence permanent.
* **CryoSleep** : Indique si le passager a choisi d’être placé en animation suspendue pendant le voyage. Les passagers en cryo-sommeil sont confinés à leur cabine.
* **Cabin** : Numéro de cabine, au format pont/numéro/côté, où le côté est soit P (Port ou bâbord) soit S (Starboard ou tribord).
* **Destination** : Planète de destination du passager.
* **Age** : Âge du passager.
* **VIP** : Indique si le passager a bénéficié d’un service VIP durant le voyage.
* **RoomService**, **FoodCourt**, **ShoppingMall**, **Spa**, **VRDeck** : Montants dépensés par le passager dans les différents services à bord du vaisseau.
* **Name** : Prénom et nom du passager.
* Une image contenant texte, capture d’écran, Police, noir

  Le contenu généré par l’IA peut être incorrect.**Transported** : **Variable cible** (booléenne), indiquant si le passager a été transporté dans une autre dimension à la suite de l'incident.

Figure 1 - Apercu du fichier train.csv

* **Données de test : test.csv**

Ce fichier contient les données des 1/3 des passagers, sans indication de la variable cible **Transported**. Il est destiné à être utilisé pour générer des prédictions à l’aide du modèle entraîné.

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.Sa structure est identique à celle de train.csv, mais sans de la colonne **Transported**.

Figure 2 Aperçu du fichier test.csv

## Préparation des données

### 2.2.1. Valeurs manquantes

Avant de débuter le traitement des valeurs absentes, je supprime la colonne « Name » car elle n'est pas nécessaire pour l'entrainement d'un modèle de machine learning.

Les colonnes catégorielles comme HomePlanet, Destination et VIP ont été remplies avec la valeur la plus courante de chacune. Cette sélection s'appuie sur l'hypothèse que les données manquantes représentent les profils les plus fréquents, ce qui favorise la cohérence générale.

Code utilisé :

for col in ['HomePlanet', 'Destination', 'VIP']:

    data[col] = data[col].fillna(data[col].mode()[0])

Concernant la colonne *Age*, la médiane a été utilisée pour remplir les valeurs manquantes.

Code utilisé :

data['Age'] = data['Age'].fillna(data['Age'].median())

Pour la colonne *CryoSleep*, si un passager avait une ou plusieurs dépenses strictement supérieures à zéro (dans les colonnes *RoomService*, *FoodCourt*, *ShoppingMall*, *Spa*, *VRDeck*), cela signifiait qu’il n’était pas en cryosommeil.

Code utilisé :

cols\_depenses = ['RoomService', 'FoodCourt', 'ShoppingMall', 'Spa', 'VRDeck']

data.loc[data['CryoSleep'].isna() & (data[cols\_depenses].sum(axis=1) > 0), 'CryoSleep'] = False

Les différentes colonnes de dépenses ont quant à elles été complétées par la médiane de chaque colonne.

Code utilisé :

for col in cols\_depenses:

    mediane = data[col].median()

    data[col] = data[col].fillna(mediane)

Enfin, les lignes contenant encore des valeurs manquantes dans les colonnes *CryoSleep* et *Cabin* ont été supprimées. Cette opération permet de limité les pertes d’information.

Code utilisé :

colonnes\_cibles = ["CryoSleep", "Cabin"]

data\_cleaned = data.dropna(subset=colonnes\_cibles)

### 2.2.2. Transformation des colonnes

Une fois les valeurs manquantes traitées, j’ai commencé la transformation des données pour les rendre plus lisible par les algorithmes de machine learning.

Tout d’abord, la colonne *CryoSleep* a été convertie en type booléen.

Code utilisé :

data\_cleaned['CryoSleep'] = data\_cleaned['CryoSleep'].astype(bool)

À partir de *PassengerId*, j’ai extrait un identifiant de groupe, permettant de calculer la taille du groupe de voyageurs (*GroupSize*) auquel appartenait chaque passager. Après extraction, les colonnes *GroupId* et *PassengerId* ont été supprimées, car elles ne sont plus utile pour l’algorithme.

Code utilisé :

group\_sizes = data\_cleaned['GroupId'].value\_counts()

data\_cleaned['GroupSize'] = data\_cleaned['GroupId'].map(group\_sizes)

data\_cleaned.drop(["GroupId", "PassengerId"], axis=1, inplace=True)

La colonne *Cabin* a été décomposée en trois nouvelles variables : *Deck*, *Cabin\_Num* et *Side*. Ces éléments fournissent des indications plus précises sur l’emplacement des passagers dans le vaisseau spatial. La colonne initiale *Cabin* a ensuite été supprimée car elle est inutile pour notre modèle de prédiction.

Code utilisé :

data\_cleaned[['Deck','Cabin\_Num','Side']]=data\_cleaned['Cabin'].str.split("/",expand = True)

data\_cleaned.drop(['Cabin'], axis=1, inplace=True)

Une variable synthétique appelée *TotalSpend* a également été créée. Elle correspond à la somme des dépenses réalisées par chaque passager dans les différents services proposés à bord.

Code utilisé :

cols\_depenses = ['RoomService','FoodCourt','Spa','VRDeck']

data\_cleaned['TotalSpend'] = data\_cleaned[cols\_depenses].sum(axis =1)

Enfin, les colonnes *CryoSleep*, *VIP* et *Transported* ont été converties en entiers ( 0 ou 1).

Ensuit je transforme les colonnes *Cabin\_Num* et *GroupSize* en décimal.

Code utilisé :

colonne =  ["CryoSleep","VIP","Transported","Cabin\_Num"]

data\_cleaned[colonne] = data\_cleaned[colonne].astype(int)

colonne =  ["Cabin\_Num","GroupSize"]

data\_cleaned[colonne] = data\_cleaned[colonne].astype(float)

### 2.2.3. Encodage et normalisation

Pour pouvoir exploiter les colonnes catégorielles dans un modèle de machine learning, j’ai appliqué un encodage en variables binaires (one-hot encoding) sur les colonnes *HomePlanet*, *Destination*, *Deck* et *Side*.

Code utilisé :

categorical\_cols = ['HomePlanet', 'Destination', 'Deck', 'Side']

data\_encoded = pd.get\_dummies(data\_cleaned, columns=categorical\_cols, drop\_first=False)

Certaines colonnes étaient encore de type booléen après traitement. Elles ont été converties en entiers (0 ou 1).

Code utilisé :

bool\_columns = data\_encoded.select\_dtypes(include='bool').columns

data\_encoded[bool\_columns] = data\_encoded[bool\_columns].astype(int)

En ce qui concerne les variables numériques, j’ai appliqué une normalisation à l’aide de la méthode *StandardScaler*. Ce traitement ramène chaque variable à une moyenne de 0 et un écart-type de 1. Les colonnes concernées par cette normalisation sont : *Age*, *RoomService*, *FoodCourt*, *ShoppingMall*, *Spa*, *VRDeck*, *Cabin\_Num*, *GroupSize* et *TotalSpend*.

Code utilisé :

numeric\_cols = [

    'Age', 'RoomService', 'FoodCourt', 'ShoppingMall', 'Spa', 'VRDeck', 'Cabin\_Num',

'GroupSize', 'TotalSpend']

scaler = StandardScaler()

data\_encoded[numeric\_cols] = scaler.fit\_transform(data\_encoded[numeric\_cols])

Pour assurer une cohérence totale entre le traitement effectué ici et celui utilisé dans l’application Streamlit destinée à l’inférence, j’ai sauvegardé deux éléments essentiels. Le premier est la liste des colonnes dans leur ordre final (*colonnes\_modele*), indispensable pour structurer correctement les nouvelles données au moment de la prédiction.

Le second est le *StandardScaler* utilisé pour la normalisation, afin de garantir que les futures données seront traitées selon les mêmes paramètres statistiques que ceux appris lors de l'entraînement.

Code utilisé :

colonnes\_modele = data\_encoded.drop(columns=['Transported'], axis=1).columns.tolist()

joblib.dump(colonnes\_modele, "colonnes\_modele.joblib")

joblib.dump(scaler , "scarler2.joblib")

## Modèles de classification

### 2.3.1. Séparation des données

Avant de construire un modèle de machine learning, j’ai commencé par séparer les variables caractéristiques de la variable cible. La variable *Transported* représente la classe à prédire (0 ou 1) et a donc été isolée dans un objet nommé cible. Toutes les autres colonnes ont été conservées dans l’objet carac, qui contient les caractéristiques décrivant les passagers.

Code utilisé :

cible = data\_encoded['Transported']

carac = data\_encoded.drop(columns=['Transported'], axis=1)

Ensuite, j’ai divisé le jeu de données en deux ensembles : un jeu d’entraînement et un jeu de test. J’ai opté pour une répartition de 70 % pour l’entraînement et 30 % pour le test, en fixant un random\_state à 42.

Code utilisé :

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(carac, cible, test\_size=0.3, random\_state=42)

### 2.3.2. Choix et évaluation des modèles de classification

**a. Régression logistique**

Initialement, j'ai expérimenté avec un modèle de régression logistique, qui est une méthode fondamentale pour les problèmes de classification binaire. Ce modèle a été formé sur les données d'entraînement, puis il a été examiné en utilisant la matrice de confusion, le rapport de classification (précision, rappel, score F1) et le score AUC-ROC.   
Pour un modèle simple, les résultats étaient convaincants : une précision de 0.79 pour la classe 0 et de 0.81 pour la classe 1, avec un rappel de 0.80 pour les deux classes. La moyenne générale se situe à 80 % et l'AUC-ROC était de 0,799. Ces résultats sont près de l'objectif que je visais, mais ils restent légèrement en dessous du seuil de plus de 80 % que j'avais établi.

**b. Arbre de décision**

Par la suite, j'ai essayé d'utiliser un arbre de décision, qui est un modèle plus souple et non linéaire. Bien qu'il soit capable de saisir des interactions complexes entre les variables, ses performances se sont avérées moins bonnes comparées à celles de la régression logistique. La précision a chuté à 76 %, tandis que l'AUC-ROC est tombée à 0.757. Le rappel et la précision ont diminué pour les deux catégories, se stabilisant aux alentours de 0.75 à 0.77, tandis que le score F1 reste constant à 0.76.   
Ce modèle a surapprit sur les données d'entraînement (overfitting), ce qui compromet sa capacité à se généraliser sur des données inédites. Donc, il a été mis de côté.

**c. Support Vector Machine (SVM)**

Par la suite, j'ai utilisé un modèle SVM (Classificateur à Vecteurs de Support), en permettant le calcul des probabilités pour l'AUC-ROC. Ce modèle a produit les performances les plus élevées parmi tous les tests effectués jusqu'à présent. Pour la classe 1, la précision est de 0.83 tandis que pour la classe 0, elle est de 0.80. Le rappel était aussi harmonieusement distribué : 0.83 pour la classe 0 et 0.80 pour la classe 1. La valeur AUC-ROC a été fixée à 0.81.   
Ce modèle satisfaisait entièrement les exigences initiales : des taux de précision et de rappel d'au moins 80 % pour chaque catégorie, ainsi qu'une solide fiabilité dans les prévisions. Ainsi, c'est le modèle choisi pour la suite du projet.

**d. Autres modèles testés**

Pour une approche complète, j'ai également testé d'autres algorithmes tels que le Perceptron, Naïve Bayes et Random Forest. Néanmoins, aucun de ces modèles n'a réussi à faire mieux que les performances du SVM. Dans l'ensemble, leurs performances étaient inférieures, en particulier en ce qui concerne la précision et le rappel, et parfois variables selon les paramètres utilisés. Ainsi, ils n'ont pas été sélectionnés.

### 2.3.3. Modèle Final

Étant donné que le modèle SVM a produit les meilleurs résultats, j'ai utilisé la fonction dump() de la bibliothèque joblib pour l'exporter.

Code utilisé :

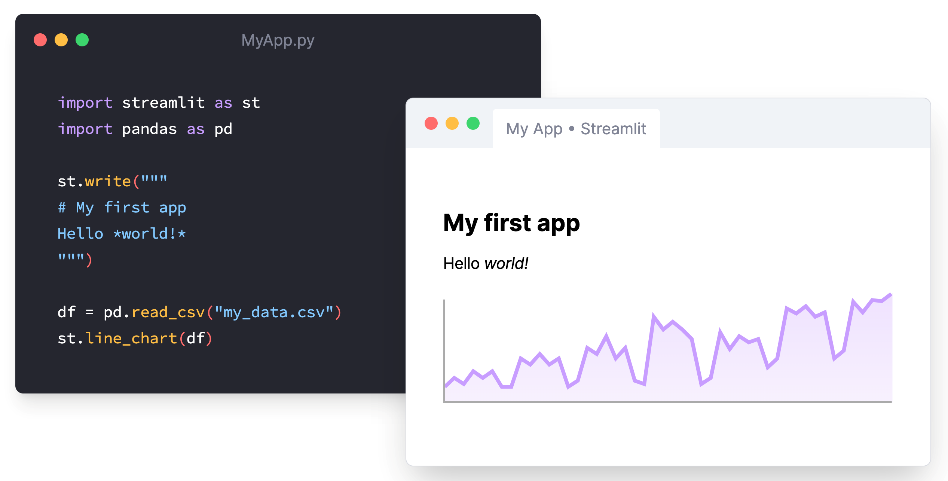
joblib.dump(svm , "modele\_svm.joblib")

Ce fichier (modele\_svm.joblib) pourra être importé dans notre application Streamlit.

# Interface avec Streamlit et Docker

## 3.1. Présentation de Streamlit

Streamlit est une librairie open source Python créée en 2018.  L’open source qui est devenu un véritable mouvement dans le développement logiciel, encourage la production collaboratrice et permet, ainsi, d’améliorer la qualité des logiciels.

Streamlit permet de créer rapidement des interfaces web interactives pour la visualisation et l'exploration de données. Il permet de transformer un simple script Python en application web sans avoir besoin de connaissances en développement front-end.

Il s’intègre naturellement avec des bibliothèques python comme :Pandas, NumPy , Matplotlib,

Figure 3 – Exemple d'application Streamlit simple

Seaborn, Plotly ,Altair et Scikit-learn

Streamlit offre une variété de composantes interactives telles que des boutons, menus déroulants, cases à cocher, curseurs, champs de saisie, sélecteurs de date, de fichiers ou de couleur. Il permet également le chargement de fichiers CSV via une interface simple, puis leur visualisation sous forme de tableau ou de graphique.

## 3.2. Fonctionnalités demandées

Dans notre contexte, l’interface développée avec Streamlit doit répondre aux besoins suivants afin d’être utilisée efficacement.

* Saisie manuelle des attributs d’un passager

L’interface propose un formulaire permettant de renseigner les attributs d’un passager pour effectuer une prédiction individuelle. Seules les caractéristiques jugées pertinentes pour le modèle prédictif sont demandées à l’utilisateur (ex. : âge, numéro de cabine, montant des dépenses, etc.).  
Un champ **nom du passager** peut être ajouté pour l’affichage sur l’interface, sans influence sur la prédiction.

* Bouton "Prédire"

Après avoir saisi les informations, un bouton **"Prédire"** permet de lancer le processus de prédiction. Ce bouton active le modèle final en s’assurant que toutes les étapes de prétraitement nécessaires sont appliquées aux données saisies (encodage, normalisation, gestion des valeurs manquantes, etc.). Le résultat s’affiche, indiquant si le passager est susceptible d’avoir été transporté dans une dimension parallèle.

* Prédictions en lot via fichier CSV

Dans un autre onglet, l’application propose une option de prédiction en lot. L’utilisateur peut importer un fichier au format CSV contenant les données de plusieurs passagers. Ce fichier doit respecter une structure conforme aux attributs attendus par le modèle. Après chargement, l’interface affiche les prédictions pour chaque passager sous forme de tableau interactif, permettant une consultation rapide des cas critiques.

## 3.3. Interface développée

### 3.3.1. Prédiction individuelle

### Un formulaire interactif est à la disposition de l'utilisateur pour entrer les détails concernant un passager. Cela inclut des informations personnelles telles que l'âge, le statut VIP ou la mise en cryosommeil, ainsi que des données relatives à la localisation (planète d'origine, destination, pont de la cabine, etc.) et les dépenses engagées à bord (services de chambre, centre commercial, spa, etc.). Un champ facultatif permet aussi de spécifier un nom, qui sera utilisé uniquement pour la présentation du résultat.

Après avoir entré les données, l'utilisateur appuie sur un bouton pour déclencher la prédiction. Le modèle fonctionne en tâche de fond et présente instantanément le verdict : il précise si, au vu des informations fournies, le passager a été transporté ou pas dans une autre dimension. [**Voir exemple en annexe 1.**](#_Annexe_1_:)

### 3.3.2. Prédiction par lot (fichier CSV)

L'interface permet aussi d'importer un fichier CSV qui contient les informations de plusieurs passagers. Cette approche offre la possibilité d'évaluer le modèle sur un groupe d'individus. L'utilisateur introduit son fichier à travers l'interface, après quoi il initie le processus de traitement en appuyant sur un bouton de prédiction.  
Avant d'introduire les données dans le modèle, l'application réalise d'abord plusieurs opérations automatiquement : elle extrait des informations du champ cabine (pour déterminer le pont, le numéro et la direction), elle calcule les dépenses totales à bord et enfin, elle détermine la taille du groupe en fonction de l'identifiant du passager. Le programme est prévu pour s'ajuster automatiquement au format requis, en rectifiant ou en indiquant les potentielles erreurs de saisie.

Après avoir produit les prévisions, les résultats apparaissent en format de tableau. Un indicateur visuel accompagne chaque passager pour indiquer s'il a été transporté ou non. Si des données sont absentes, un message explicite informe l'utilisateur pour rectifier le fichier d'origine.

[**Voir l’exemple en annexe 2**](#_Annexe_2)

## 3.6. Déploiement dockers

Afin de simplifier le déploiement et garantir la portabilité de l'application Streamlit, nous avons procédé à sa containerisation en utilisant Docker.   
Le répertoire de déploiement comprend :

* Dockerfile : fichier de configuration qui définit l'image Docker. ([**annexe 4**](#_Annexe_4_:))
* interface.py : le script principal de l'application Streamlit.
* requirement.txt : énumération des bibliothèques Python dependantes. ([**annexe 3**](#_Annexe_3_:_1))
* modele\_svm.joblib, scaler.joblib, colonne\_modele.joblib : fichiers contenant les objets requis pour effectuer des prédictions.

Le **Dockerfile** utilise une image Python officielle comme base. Il intègre tous les fichiers du répertoire dans l'image, gère l'installation des dépendances grâce à pip en se basant sur le fichier requirement.txt, puis établit la commande par défaut pour démarrer l'application avec Streamlit :

Streamlit run interface.py --server.port 8501 --server.address 0.0.0.0

La construction de l’image Docker s’effectue avec :

docker build -t spaceship-titanic-app .

et le lancement du conteneur avec :

docker run -p 8501:8501 spaceship-titanic-app

# Bilan du projet

Tous les objectifs fixés ont été atteints, avec une solution complète, performante et facilement déployable:

1. **Développer un modèle prédictif performant**  
   L'intention était d'obtenir un modèle dont le score AUC dépasserait 0,80. Le modèle SVM élaboré a obtenu un score AUC qui répond à ce critère, prouvant ainsi une capacité appréciable à différencier les passagers transportés de ceux qui ne le sont pas dans une autre dimension.
2. **Créer une application web interactive pour la prédiction**  
   L'application Streamlit mise en place répond parfaitement aux exigences : elle comprend un formulaire de saisie manuelle des données, autorise la prédiction immédiate et propose l'option d'importer un fichier CSV pour effectuer des prédictions à grande échelle.
3. **Assurer un déploiement rapide et portable grâce à Docker**  
   Le projet comprend une image Docker stable ainsi qu'un Dockerfile explicite, facilitant un déploiement aisé et rapide sur n'importe quelle machine compatible en moins de cinq minutes. Cela assure la portabilité et simplifie l'adaptation à divers environnements.

# Conclusion

Ce projet a impliqué l'analyse de données dans le but de prévoir quels passagers ont été déplacés vers une autre dimension. Suite à un nettoyage et un prétraitement des données, plusieurs modèles d'apprentissage automatique ont été évalués pour déterminer celui qui présente la meilleure précision. L'application interactive développée avec Streamlit intègre le modèle SVM, choisi pour sa précision et sa solidité. Cette application offre à l'utilisateur la possibilité de réaliser des prédictions soit en entrant des données manuellement, soit en important des fichiers CSV. Finalement, l'utilisation de Docker pour la conteneurisation simplifie le déploiement et la portabilité de la solution.

**Bibliographie de mémoire technique**

* GitHub. (n.d.). Dépôt Spaceship Titanic [en ligne]. GitHub. Disponible sur : <https://github.com/Abdisamad06/Spaceship-Titanic.git> . Consulté le 11/06/2025.
* Monday. (n.d.). Plateforme Monday [en ligne]. Disponible sur : <https://view.monday.com> . Consulté le 11/06/2025.
* Innovatiana. (n.d.). Data Preparation for AI : Guide complet sur les étapes de préparation des données [en ligne]. Disponible sur : <https://www.innovatiana.com/post/data-preparation-for-ai> . Consulté le 15/06/2025.
* Kaggle. (n.d.). Spaceship Titanic – Descriptions des données [en ligne]. Disponible sur : <https://www.kaggle.com/competitions/spaceship-titanic/data> . Consulté le 11/06/2025.
* Streamlit. (n.d.). Documentation officielle Streamlit [en ligne]. Disponible sur : <https://streamlit.io/components?category=all> . Consulté le 15/06/2025.
* Docker. (n.d.). Documentation officielle Docker [en ligne]. Disponible sur : <https://docs.docker.com/> . Consulté le 07/07/2025.

# Annexe :

## Annexe 1 :

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Le contenu généré par l’IA peut être incorrect.

Figure 4 - Exemple de la prédiction individuelle de l’interface Streamlit.

## Annexe 2

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Le contenu généré par l’IA peut être incorrect.

Figure 5 - Exemple de la prédiction par lot de l’interface Streamlit.

## 

## Annexe 3 :

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

Figure 6 - Contenu du fichier « requirements.txt ».

## Annexe 4 :

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Le contenu généré par l’IA peut être incorrect.

Figure 7 - Contenu du Dockerfile