Projet Encadré
"Analyse et prédiction des prix des billets d'avion"

Rapport détaillé

Préparé par :
MAAMA Fatima
DINE Ikram

Encadrant : BENYACOUB Badreddine

Année Universitaire 2024-2025

Introduction

Dans un contexte où le transport aérien connaît une croissance soutenue, la prévision des prix des billets d'avion constitue un enjeu majeur pour les compagnies comme pour les voyageurs. Les tarifs peuvent varier en fonction de multiples facteurs, rendant la prédiction des prix complexe mais précieuse.

Pour cette étude, nous avons utilisé le dataset *Flight Price Prediction* provenant de la plateforme Kaggle. Ce jeu de données, riche et varié, nous a permis d'explorer les relations entre différentes caractéristiques des vols et leurs prix.

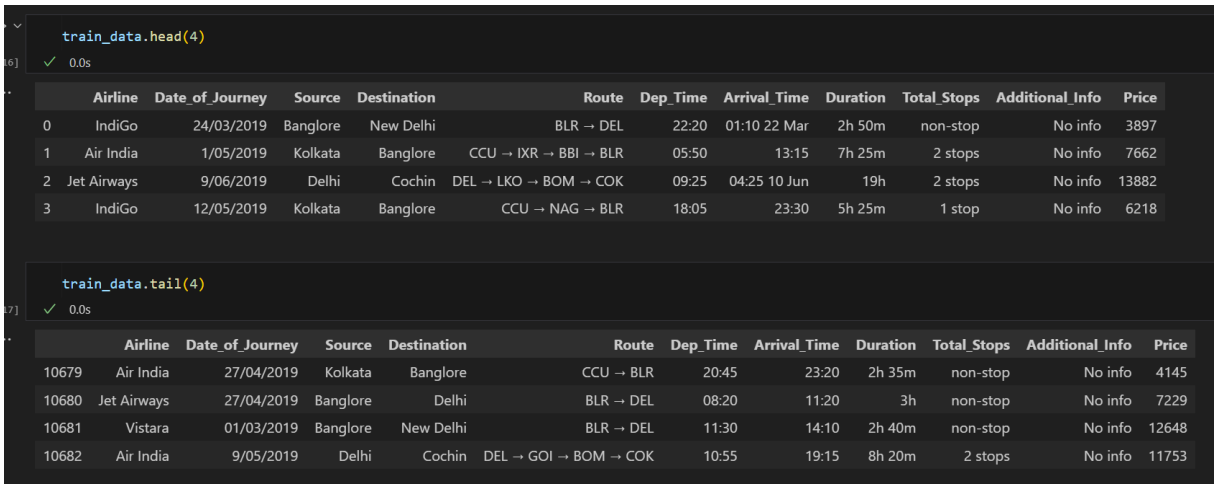
L'objectif principal de ce projet est de développer un modèle performant capable d'estimer le prix des billets d'avion, en s'appuyant sur des méthodes d'analyse de données et d'apprentissage automatique. Cette démarche vise à fournir des outils d'aide à la décision pour mieux anticiper les fluctuations tarifaires.

Les données utilisées comportent plusieurs variables pertinentes telles que la compagnie aérienne, la ville de départ, la destination, la date du voyage, la durée du vol ou encore le nombre d'escales. Ces informations serviront de base pour construire un modèle robuste de prédiction des prix, en s'appuyant sur des techniques d'apprentissage automatique.

1. Prétraitement de données

Chargement du dataset

FIGURE 1 – Importations



	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non-stop	No info	3897
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2 stops	No info	7662
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2 stops	No info	13882
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1 stop	No info	6218

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
10679	Air India	27/04/2019	Kolkata	Banglore	CCU → BLR	20:45	23:20	2h 35m	non-stop	No info	4145
10680	Jet Airways	27/04/2019	Banglore	Delhi	BLR → DEL	08:20	11:20	3h	non-stop	No info	7229
10681	Vistara	01/03/2019	Banglore	New Delhi	BLR → DEL	11:30	14:10	2h 40m	non-stop	No info	12648
10682	Air India	9/05/2019	Delhi	Cochin	DEL → GOI → BOM → COK	10:55	19:15	8h 20m	2 stops	No info	11753

FIGURE 2 – Aperçu de notre base de données

Nom de la colonne	Description
`Airline`	Nom de la compagnie aérienne
`Date_of_Journey`	Date du vol
`Source`	Ville de départ
`Destination`	Ville d'arrivée
`Route`	Itinéraire du vol, incluant les escales éventuelles
`Dep_Time`	Heure de départ du vol
`Arrival_Time`	Heure d'arrivée prévue à destination
`Duration`	Durée totale du vol
`Total_Stops`	Nombre d'escales
`Additional_Info`	Informations complémentaires sur le vol
`Price`	Prix du billet

TABLE 1 – Description des colonnes du dataset

Après avoir chargé la base de données nous procédons au traitement des valeurs manquantes

Traitement des valeurs manquantes

```
train_data.isnull().sum()
```

```
Airline            0
Date_of_Journey    0
Source             0
Destination        0
Route              1
Dep_Time           0
Arrival_Time       0
Duration           0
Total_Stops        1
Additional_Info    0
Price              0
dtype: int64
```

```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10683 entries, 0 to 10682
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Airline          10683 non-null  object
 1   Date_of_Journey  10683 non-null  object
 2   Source           10683 non-null  object
 3   Destination      10683 non-null  object
 4   Route            10682 non-null  object
 5   Dep_Time         10683 non-null  object
 6   Arrival_Time     10683 non-null  object
 7   Duration         10683 non-null  object
 8   Total_Stops      10682 non-null  object
 9   Additional_Info  10683 non-null  object
 10  Price            10683 non-null  int64
dtypes: int64(1), object(10)
memory usage: 918.2+ KB
```

FIGURE 3 – Détection des valeurs manquantes

On remarque que le nombre des valeurs manquantes est très faible par rapport à la dimension de notre base de données donc on préfère supprimer les valeurs manquantes au lieu de les imputer.

```
train_data.dropna(inplace=True)
```

```
train_data.isnull().sum()
```

```
Airline            0
Date_of_Journey    0
Source             0
Destination        0
Route              0
Dep_Time           0
Arrival_Time       0
Duration           0
Total_Stops        0
Additional_Info    0
Price              0
dtype: int64
```

FIGURE 4 – Suppression des valeurs manquantes

Data type casting

```
data.dtypes
```

```
Airline            object
Date_of_Journey    object
Source             object
Destination        object
Route              object
Dep_Time           object
Arrival_Time       object
Duration           object
Total_Stops        object
Additional_Info    object
Price               int64
dtype: object
```

FIGURE 5 – Types des colonnes

On constate que les variables Date_of_Journey, Dep_Time et Arrival_Time sont de type object. Par conséquent, il est nécessaire de les convertir en type datetime afin de pouvoir les utiliser correctement dans la phase de prédiction.

```
def change_into_Datetime(col):
    data[col] = pd.to_datetime(data[col])
```

```
import warnings
from warnings import filterwarnings
filterwarnings("ignore")
```

```
for feature in ['Dep_Time', 'Arrival_Time' , 'Date_of_Journey']:
    change_into_Datetime(feature)
```

FIGURE 6 – Type casting

Prétraitement des colonnes

Dans le cadre du prétraitement, nous avons nettoyé les colonnes `Date_of_Journey`, `Dep_Time` et `Arrival_Time`, puis extrait des attributs dérivés à partir de ces dernières.

À partir de la colonne `Date_of_Journey`, nous avons extrait trois nouvelles variables représentant respectivement le jour, le mois et l'année du vol. Concernant les colonnes `Dep_Time` et `Arrival_Time`, nous avons extrait les heures et les minutes de manière structurée, afin de faciliter l'analyse temporelle et l'utilisation dans les modèles de prédiction.

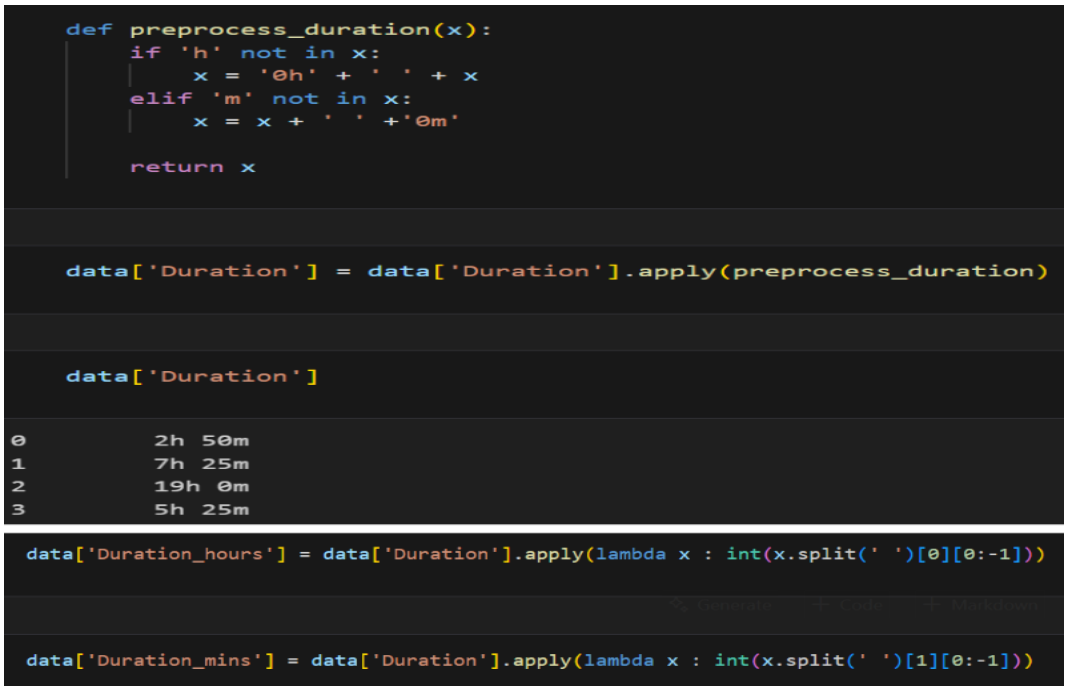
FIGURE 7 – Extraction des attributs dérivés

Et après ce traitement on supprime les colonnes `Date_of_Journey`, `Dep_Time` et `Arrival_Time` qui sont maintenant inutiles.

La colonne `Duration` contient la durée totale de chaque vol, exprimée sous la forme d'une chaîne de caractères combinant les heures et les minutes.

Afin d'exploiter efficacement cette variable, nous avons procédé à son nettoyage et à son découpage en deux nouvelles colonnes distinctes :

— `Duration_hours` : pour représenter le nombre d'heures ;
— `Duration_minutes` : pour représenter le nombre de minutes.



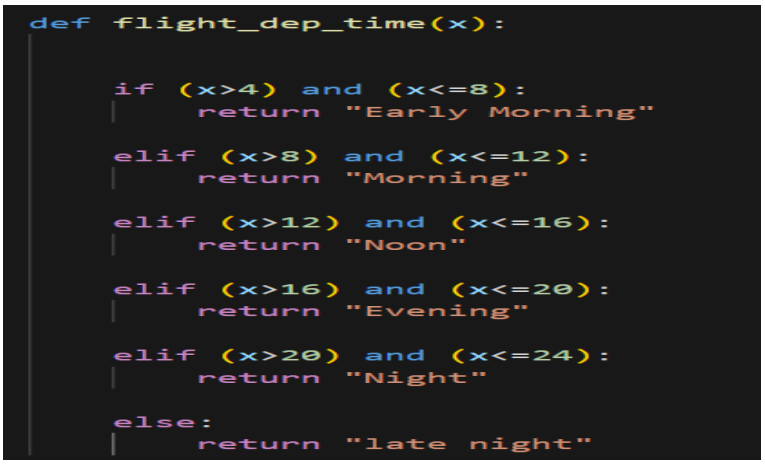
```
def preprocess_duration(x):
    if 'h' not in x:
        x = '0h' + ' ' + x
    elif 'm' not in x:
        x = x + ' ' +'0m'

    return x


data['Duration'] = data['Duration'].apply(preprocess_duration)


data['Duration']

0         2h 50m
1         7h 25m
2         19h 0m
3         5h 25m

data['Duration_hours'] = data['Duration'].apply(lambda x : int(x.split(' ')[0][0:-1]))

data['Duration_mins'] = data['Duration'].apply(lambda x : int(x.split(' ')[1][0:-1]))
```

FIGURE 8 – Prétraitement de la colonne `Duration`

2. Visualisation des données

Analyse des départs des vols

Afin d'analyser les horaires de départ des vols, nous avons procédé à une classification des valeurs de la colonne `Dep_Time` en différentes plages horaires : *Early Morning*, *Morning*, *Noon*, *Evening*, *Night* et *Late Night*. Cette transformation permet de regrouper les départs selon des périodes significatives de la journée, facilitant ainsi l'analyse temporelle et la modélisation.



```
def flight_dep_time(x):

    if (x>4) and (x<=8):
        return "Early Morning"

    elif (x>8) and (x<=12):
        return "Morning"

    elif (x>12) and (x<=16):
        return "Noon"

    elif (x>16) and (x<=20):
        return "Evening"

    elif (x>20) and (x<=24):
        return "Night"

    else:
        return "late night"
```

FIGURE 9 – Fonction pour extraire les périodes de la journée

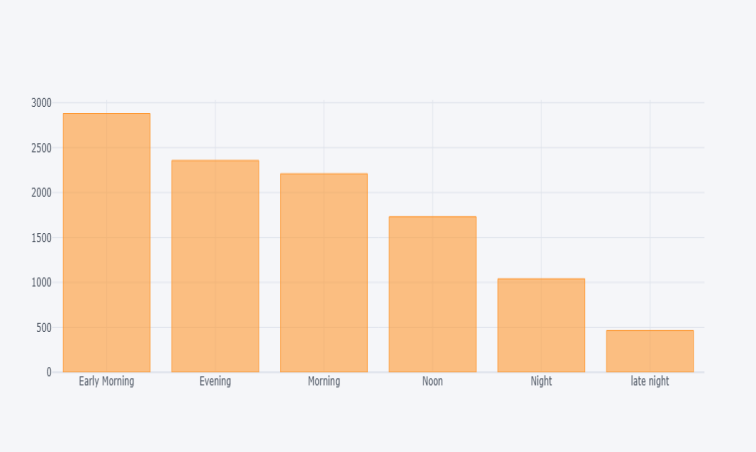


FIGURE 10 – Répartition des vols selon les plages horaires de départ

On observe que certaines périodes, comme très tôt le matin ou le soir, concentrent un nombre élevé de départs, ce qui peut refléter les préférences des voyageurs ou les politiques de programmation des compagnies aériennes. À l'inverse, les vols programmés durant la nuit et très tard la nuit sont moins fréquents.

Analyse de la relation entre la durée des vols et leurs prix

Pour analyser cette relation on extrait d'abord la colonne contenant le nombre total des minutes dans la durée corresponsante du vol et on visualise ensuite cette variable par rapport au prix du vol.

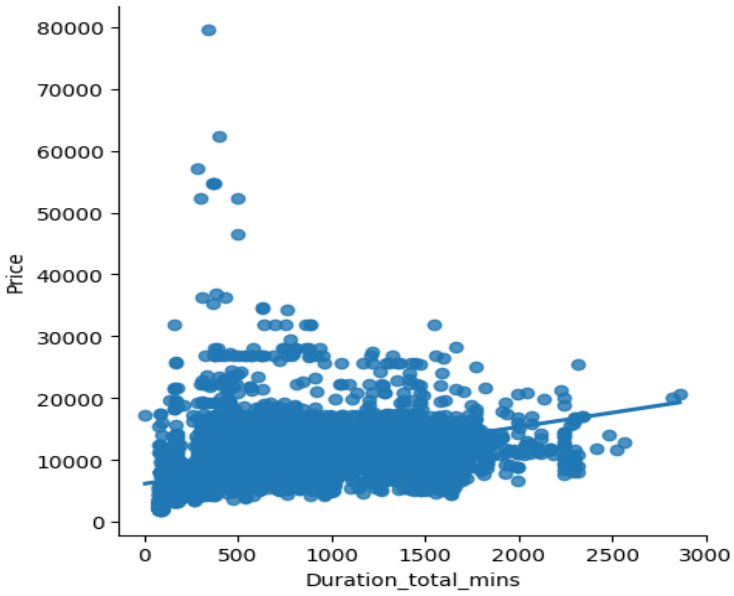


FIGURE 11 – Variation des prix selon la durée totale du vol

Il apparaît de manière assez claire que le prix des vols tend à augmenter avec la durée (en minutes) du trajet. Cette relation suggère une corrélation positive entre le temps de vol et le tarif appliqué.

Analyse de la relation entre le nombre des escales et les prix

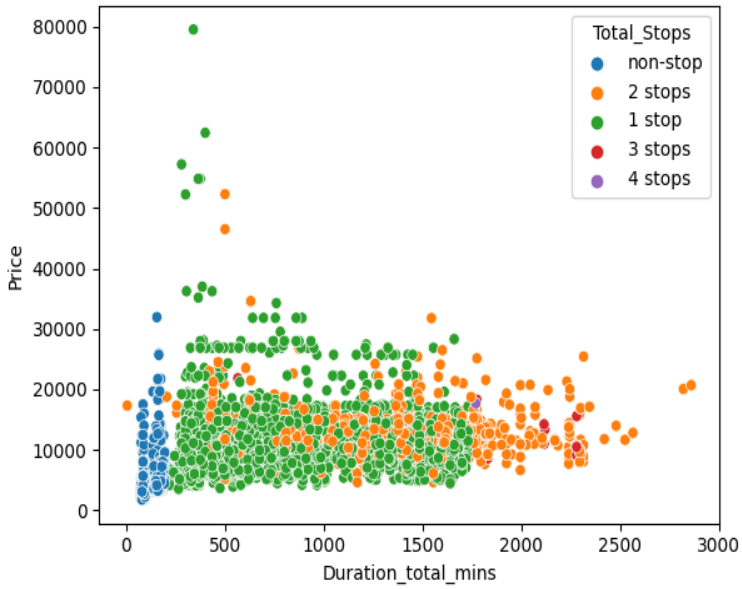


FIGURE 12 – Variation des prix selon la durée totale du vol et le nombre d'escales

On remarque que les vols directs présentent une durée plus courte et un tarif plus faible ; à mesure que le nombre d'escales augmente, la durée du trajet s'allonge et le prix tend également à croître.

Analyse de la relation entre la compagine aérienne et les prix

Pour analyser la relation entre la compagnie aérienne (`Airline`) et les prix, nous avons réalisé un boxplot permettant de visualiser la distribution des prix pour chaque compagnie. Cette représentation graphique facilite la comparaison des tarifs pratiqués et met en évidence la variabilité des prix au sein de chaque compagnie.

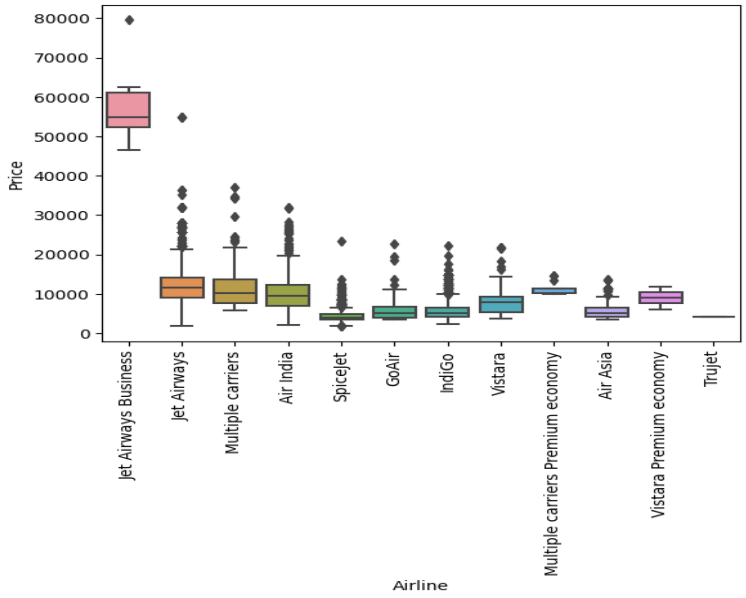


FIGURE 13 – Variation des prix selon la compagnie aérienne

D'après le graphique, on observe que la compagnie *Jet Airways Business* affiche les prix les plus élevés. Mis à part cette compagnie, la majorité des autres présentent des médianes de prix relativement similaires.

Traitement des valeurs aberrantes

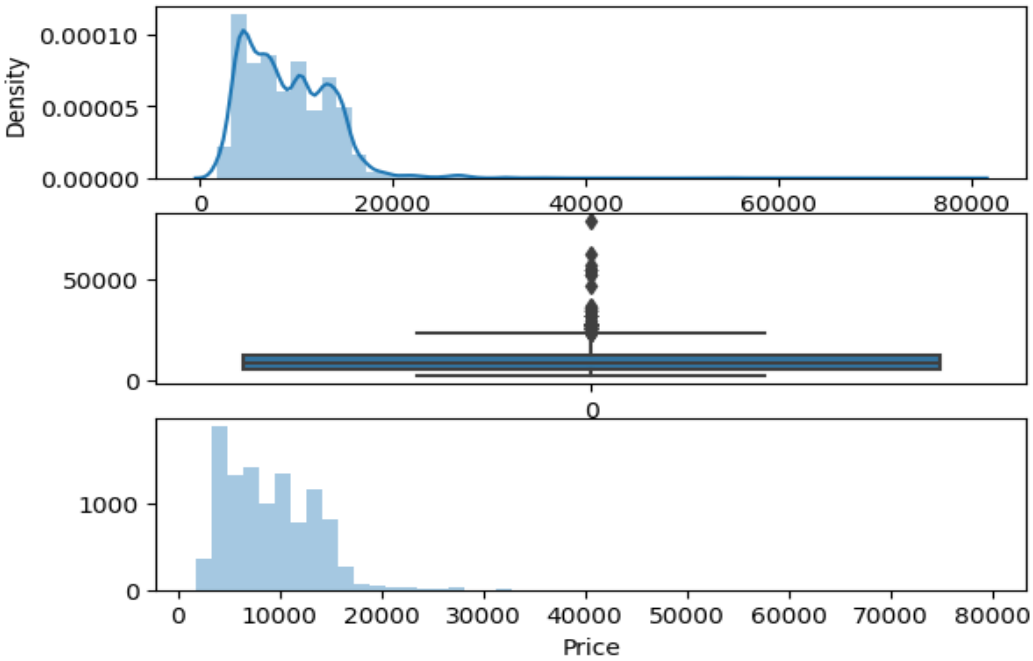


FIGURE 14 – Détection des valeurs aberrantes

Pour détecter les valeurs aberrantes dans les prix des vols, nous avons utilisé une fonction de visualisation personnalisée, `plot(df, col)`, qui génère trois représentations complémentaires pour la variable `Price` :

— Un **histogramme avec courbe de densité** à l'aide de `sns.distplot`, permettant de visualiser la distribution globale et les éventuelles asymétries ;
— Un **boxplot** avec `sns.boxplot`, qui met en évidence les valeurs extrêmes se situant en dehors de l'intervalle interquartile. Ces points sont considérés comme des valeurs aberrantes selon la règle classique :

$$[Q1 - 1.5 \times IQR,\ Q3 + 1.5 \times IQR]$$

où $Q1$ et $Q3$ désignent respectivement le premier et le troisième quartile, et $IQR = Q3 - Q1$.
— Un **histogramme sans courbe KDE**, afin d'observer la fréquence brute des valeurs sans lissage.

Pour la variable `Price`, nous avons identifié les valeurs aberrantes comme étant celles supérieures à

$$Q_3 + 1.5 \times IQR$$

ou inférieures à

$$Q_1 - 1.5 \times IQR,$$

où Q_1 et Q_3 représentent respectivement le premier et le troisième quartile, et $IQR = Q_3 - Q_1$.

Nous avons également observé que les valeurs supérieures à 35 000 constituent des outliers extrêmes susceptibles de fausser les analyses. Afin de limiter leur impact, ces valeurs ont été remplacées par la médiane de la variable `Price`.

```
data['Price'] = np.where(data['Price']>=35000 , data['Price'].median() , data['Price'])
```

FIGURE 15 – Traitement des valeurs aberrantes

Cette méthode permet de préserver la distribution générale des données tout en atténuant l'influence des outliers extrêmes.

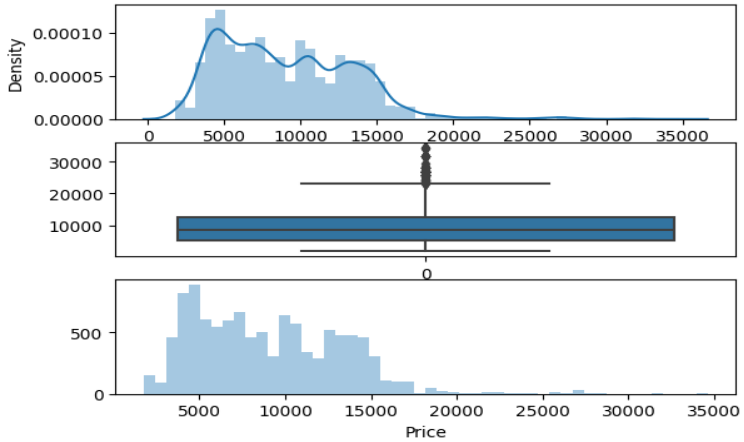


FIGURE 16 – Distribution de la variable *Price* après traitement des valeurs aberrantes

3. Feature engineering

Transformation de la variable `Source`

La variable `Source`, qui représente la ville ou l'aéroport de départ, a été transformée par un encodage one-hot afin de convertir ses catégories textuelles en variables numériques binaires. Cette technique permet de représenter chaque modalité de la variable par une nouvelle colonne indicatrice.

```
for sub_category in data['Source'].unique():
    data['Source_'+sub_category] = data['Source'].apply(lambda x : 1 if x==sub_category else 0)
```

FIGURE 17 – One hot encoding appliquée à la variable `Source`

Transformation des variables `Airline` et `Destination`

Pour les variables catégorielles `Airline` et `Destination`, nous avons procédé à un encodage ordinal guidé par la variable cible (*Target Guided Ordinal Encoding*). Les étapes principales de cette méthode sont les suivantes :

1. Calculer la moyenne de la variable cible `Price` pour chaque catégorie de la variable.
2. Trier les catégories selon ces moyennes, de la plus faible à la plus élevée.
3. Assigner à chaque catégorie un rang ordinal basé sur cet ordre.
4. Remplacer les catégories originales par ces rangs dans le jeu de données.

Cette technique permet de conserver une relation ordinale entre les catégories en fonction de leur influence sur la variable cible, tout en réduisant la dimensionnalité vue que la variable `Airline` contient 12 modalités et la variable `Destination` contient 5 modalités .

```
data.groupby(['Airline'])['Price'].mean().sort_values()
```

```
Airline
Trujet                                4140.000000
SpiceJet                              4338.284841
Air Asia                              5590.260188
IndiGo                                5673.682903
GoAir                                 5861.056701
Vistara                               7796.348643
Vistara Premium economy               8962.333333
Air India                             9612.427756
Multiple carriers                    10902.678094
Multiple carriers Premium economy    11418.846154
Jet Airways                          11643.923357
Jet Airways Business                 58358.666667
Name: Price, dtype: float64
```

```
airlines = data.groupby(['Airline'])['Price'].mean().sort_values().index
```

```
airlines
```

```
Index(['Trujet', 'SpiceJet', 'Air Asia', 'IndiGo', 'GoAir', 'Vistara',
       'Vistara Premium economy', 'Air India', 'Multiple carriers',
       'Multiple carriers Premium economy', 'Jet Airways',
       'Jet Airways Business'],
      dtype='object', name='Airline')
```

```
dict_airlines = {key:index for index , key in enumerate(airlines , 0)}
```

FIGURE 18 – Procédure de *Target Guided Ordinal Encoding* pour la variable `Airline`

```
{'Trujet': 0,
 'SpiceJet': 1,
 'Air Asia': 2,
 'IndiGo': 3,
 'GoAir': 4,
 'Vistara': 5,
 'Vistara Premium economy': 6,
 'Air India': 7,
 'Multiple carriers': 8,
 'Multiple carriers Premium economy': 9,
 'Jet Airways': 10,
 'Jet Airways Business': 11}
```

FIGURE 19 – Résultat de *Target Guided Ordinal Encoding* pour la variable `Airline`

Une procédure identique a été appliquée pour la variable `Destination`.

Transformation de la variable `Total_stops`

La variable `Total_Stops` est de nature *catégorique ordinale*, car le nombre d'escales (*non-stop*, *1 stop*, *2 stops*, etc.) possède un ordre logique croissant. Nous avons donc choisi de réaliser un **label encoding** : chaque modalité est associée à une clé entière reflétant cet ordre naturel. Concrètement, la valeur *non-stop* a été codée par **0**, *1 stop* par **1**, *2 stops* par **2**, et ainsi de suite. Cette transformation préserve l'information ordinale tout en fournissant une représentation numérique compacte et directement exploitable par les algorithmes de modélisation.

```
{'Trujet': 0,
 'SpiceJet': 1,
 'Air Asia': 2,
 'IndiGo': 3,
 'GoAir': 4,
 'Vistara': 5,
 'Vistara Premium economy': 6,
 'Air India': 7,
 'Multiple carriers': 8,
 'Multiple carriers Premium economy': 9,
 'Jet Airways': 10,
 'Jet Airways Business': 11}
```

FIGURE 20 – Encodage de la variable `Total_stops`

Suppression des colonnes non nécessaires

Certaines variables du jeu de données ont été supprimées car elles n'apportaient pas d'information utile pour la modélisation ou leur contenu était redondant. Voici les principales justifications :

- `Additional_Info` : cette variable contient près de 80 % de la valeur *no_info*, ce qui la rend peu informative.
- `Date_of_Journey` : les informations pertinentes de cette variable ont déjà été extraites dans les colonnes `Journey_day`, `Journey_month` et `Journey_hour`.
- `Duration_total_mins` : cette variable a été décomposée en deux colonnes plus expressives : `Duration_hours` et `Duration_mins`.
- `Source` : cette variable a déjà été encodée sous une autre forme dans une étape précédente.
- `Journey_year` : cette variable est constante (valeur unique 2019) pour l'ensemble du jeu de données, elle n'apporte donc aucune variabilité utile à la modélisation.
- `Duration` : cette variable a été décomposée en deux nouvelles colonnes plus explicites : `Duration_hours` et `Duration_mins`.
- `Route` : cette colonne est fortement corrélée à `Total_Stops`. Conserver les deux introduirait une redondance inutile dans le modèle.

La suppression de ces variables permet d'alléger le jeu de données tout en conservant les informations pertinentes nécessaires à la prédiction du prix.

Sélection des variables

Nous avons utilisé la fonction `mutual_info_regression` du module `sklearn.feature_selection` pour estimer l'information mutuelle entre chaque variable explicative et la variable cible `Price`. Cette méthode permet d'évaluer le degré de dépendance entre les variables, même lorsque cette relation est non linéaire.

La sortie de cette fonction est un vecteur de scores : un score élevé indique qu'une variable contient beaucoup d'information utile pour prédire le prix, tandis qu'un score proche de zéro suggère peu ou pas de relation avec la cible.

	importance
Airline	1.324624
Arrival_Time_hour	1.137299
Duration_hours	1.123044
Destination	1.070478
Dep_Time_hour	0.929271
Arrival_Time_minute	0.902881
Total_Stops	0.788275
Dep_Time_minute	0.759634
Duration_mins	0.675380
Journey_month	0.625884
Source_Delhi	0.527107
Source_Kolkata	0.453649
Source_Banglore	0.396003
Journey_day	0.377369
Source_Mumbai	0.203103
Source_Chennai	0.139058

FIGURE 21 – Information mutuelle de chaque variables par ordre d'importance

Cette évaluation nous a permis de prioriser les variables les plus pertinentes pour l'entraînement de notre modèle. Nous avons ainsi décidé de supprimer les variables les moins informatives, notamment `Source_Mumbai` et `Source_Chennai`, afin de réduire la dimensionnalité du jeu de données et d'optimiser les performances du modèle.

3. Construction des modèles

Avant la construction du modèle on a d'abord réparti les données en variable cible et variables caractéristiques et puis on a diviser les données entre échantillon de train et de test. Nous avons réservé 25 % des données pour le test et 75 % pour l'entraînement . L'utilisation de `random_state=42` permet de garantir la reproductibilité de la répartition.

```
X = data.drop(['Price'] , axis=1)
✓ 0.0s

y = data['Price']
```

```
from sklearn.model_selection import train_test_split
✓ 0.0s

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42)
✓ 0.0s
```

FIGURE 22 – Séparation et division des données

Modèle de régression linéaire

Dans un premier temps, nous avons débuté avec un modèle de base : la régression linéaire. Pour cela, nous avons importé la classe `LinearRegression` de la bibliothèque `scikit-learn`, puis entraîné le modèle sur les données d'entraînement.

Une fois le modèle ajusté, nous avons réalisé des prédictions sur les ensembles d'entraînement et de test, puis évalué ses performances à l'aide de différentes métriques.

Les résultats obtenus sont les suivants :

— Coefficient de détermination R^2 sur le train : 0,5951
— Coefficient de détermination R^2 sur le test : 0,5686
— Erreur absolue moyenne (MAE) : 2021,98
— Racine de l'erreur quadratique moyenne (RMSE) : 2898,04

```
# Affichage
print("Résultat de la régression linéaire")
print(f"R² sur le train : {r2_train:.4f}")
print(f"R² sur le test  : {r2_test:.4f}")
print(f"MAE             : {mae:.2f}")
print(f"RMSE            : {rmse:.2f}")
```

✓ 0.0s

```
Résultat de la régression linéaire
R² sur le train : 0.5951
R² sur le test  : 0.5686
MAE             : 2021.98
RMSE            : 2898.04
```

FIGURE 23 – Résultats de la régression linéaire

Ces résultats montrent que le modèle capte une part modérée de la variance du prix, ce qui constitue une base de comparaison pour les modèles plus complexes que nous allons explorer par la suite.

Modèle de K_plus proches voisins

Suite aux performances modérées du modèle de régression linéaire, nous avons opté pour un modèle non linéaire : celui des k plus proches voisins. Ce modèle repose sur la similarité entre les observations : la prédiction pour une instance se base sur la moyenne des valeurs cibles des k instances les plus proches dans l'espace des variables explicatives.

Nous avons commencé par importer la classe `KNeighborsRegressor` et instancié un modèle avec $k = 5$ voisins. Le modèle a été entraîné sur les données d'entraînement, puis utilisé pour prédire les valeurs cibles sur les ensembles d'entraînement et de test. Pour chaque ensemble, nous avons calculé le coefficient de détermination R^2. De plus, sur l'ensemble de test, nous avons évalué les performances à l'aide de l'erreur absolue moyenne (MAE) et de la racine de l'erreur quadratique moyenne (RMSE). Les performances du modèle KNN sont les suivantes :

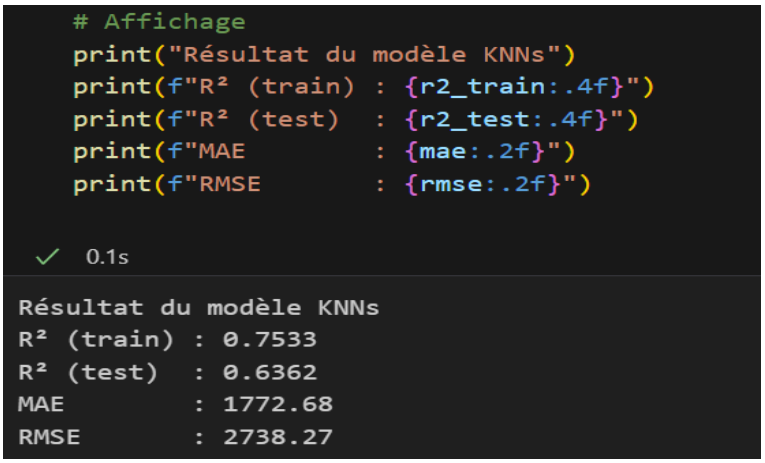


FIGURE 24 – Résultats du modèle KNNs

Cette diminution de Coefficient de détermination R^2 entre train et test est normale et indique que le modèle généralise relativement bien, sans surapprentissage excessif.

Les erreurs moyennes (MAE = 1772,68 et RMSE = 2738,27) indiquent que, en moyenne, la prédiction s'écarte d'environ 1773 unités monétaires du prix réel, avec une dispersion des erreurs modérée (RMSE). En gros, le modèle KNNs présente des performances assez bonnes.

Afin d'améliorer davantage les performances obtenues avec le modèle KNN, nous avons opté pour un modèle plus complexe et puissant : Random Forest.

Modèle de Random Forest

Ce modèle, basé sur l'ensemble de plusieurs arbres de décision, permet de mieux capturer les relations non linéaires et les interactions entre variables, tout en limitant le risque de surapprentissage grâce à son mécanisme d'agrégation.

Nous avons entraîné un modèle de Random Forest Regressor sur les données d'entraînement. Après ajustement, nous avons effectué des prédictions sur les ensembles d'entraînement et de test. Nous avons ensuite évalué les performances du modèle en calculant le coefficient de détermination R^2 sur les deux ensembles, ainsi que l'erreur absolue moyenne (MAE) et la racine de l'erreur quadratique moyenne (RMSE) sur l'ensemble de test.

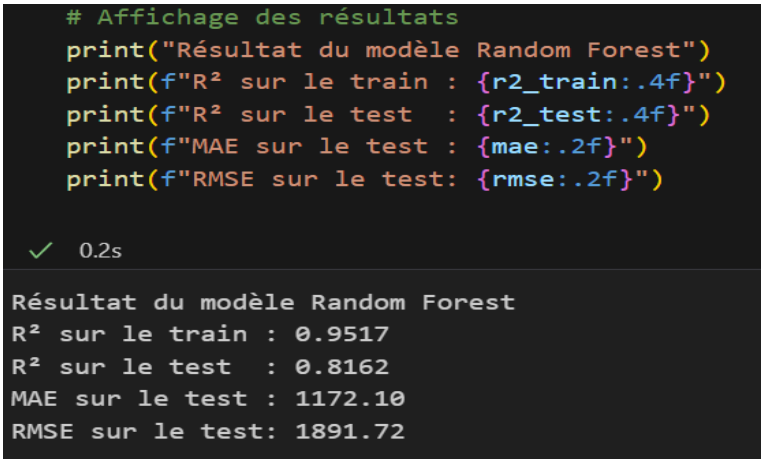


FIGURE 25 – Résultats du modèle Random Forest

Le modèle Random Forest a montré de très bonnes performances. Le coefficient de détermination atteint $R^2 = 0{,}95$ sur les données d'entraînement, indiquant un excellent ajustement du modèle. Sur les données de test, le R^2 reste élevé à 0,81, ce qui traduit une bonne capacité de généralisation. L'erreur absolue moyenne est de 1172,10, et la racine de l'erreur quadratique moyenne est de 1891,72. Ces résultats confirment que le modèle Random Forest est plus performant que les précédents, à la fois en précision et en stabilité.

Optimisation des hyperparamètres avec RandomizedSearchCV

L'optimisation des hyperparamètres constitue une étape cruciale dans le processus de modélisation afin d'améliorer la performance prédictive du modèle. Dans cette étude, nous avons utilisé la méthode `RandomizedSearchCV` de la bibliothèque `scikit-learn` pour ajuster les paramètres du modèle `RandomForestRegressor`.

Contrairement à `GridSearchCV` qui évalue systématiquement toutes les combinaisons possibles d'hyperparamètres, `RandomizedSearchCV` explore un sous-ensemble aléatoire de ces combinaisons, ce qui permet un gain de temps considérable lorsque l'espace de recherche est vaste. Dans notre cas, nous avons défini trois hyperparamètres à optimiser : `n_estimators` (nombre d'arbres), `max_depth` (profondeur maximale de chaque arbre) et `min_samples_split` (nombre minimal d'échantillons requis pour diviser un nœud).

FIGURE 26 – Hyperparamètres à optimiser pour le modèle Random Forest

Chaque hyperparamètre a été associé à un ensemble de valeurs possibles, générant ainsi 36 combinaisons potentielles. La recherche aléatoire a été limitée à 10 itérations, avec une validation croisée à 3 plis (`cv=3`) et une fonction de score basée sur le coefficient de détermination R^2.

```python
from sklearn.model_selection import RandomizedSearchCV

search = RandomizedSearchCV(ml_model, param_distributions=param_dist,
                            n_iter=10, cv=3, scoring='r2',
                            random_state=42, n_jobs=-1)

search.fit(X_train, y_train)
```

FIGURE 27 – Initialisation du RandomizedSearch

Après entraînement du modèle sur les données d'apprentissage, la meilleure combinaison trouvée est la suivante : `n_estimators=300`, `max_depth=None`, et `min_samples_split=10`.

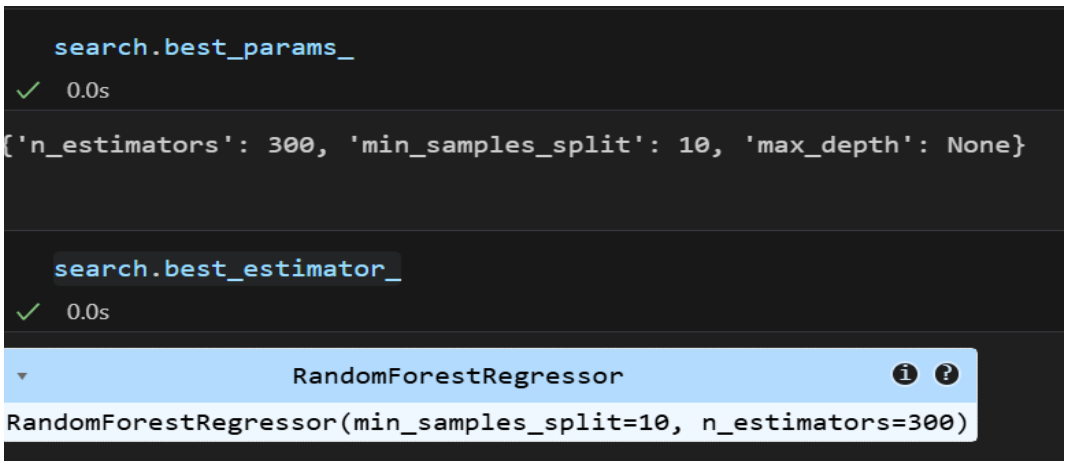


FIGURE 28 – Hyperparamètres optimaux du modèle Random Forest

Le modèle associé à cette configuration a atteint un score moyen R^2 de **0,8146** sur les trois plis de la validation croisée.

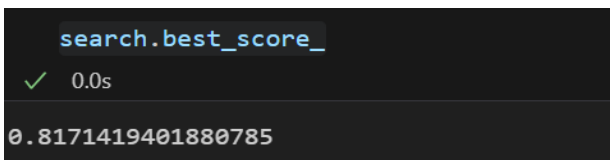


FIGURE 29 – R^2 du modèle Random Forest optimisé

Ce résultat indique que le modèle ainsi optimisé explique environ 82 % de la variance des données, ce qui témoigne d'une performance satisfaisante du modèle.

```
best_model = search.best_estimator_
y_pred = best_model.predict(X_test)

# Évaluation finale
print("R² sur les données de test :", r2_score(y_test, y_pred))

✓ 0.1s

R² sur les données de test : 0.8337022324230574
```

FIGURE 30 – R^2 du modèle Random Forest optimisé sur les données du test

En résumé, l'utilisation de `RandomizedSearchCV` a permis d'améliorer de manière significative la précision du modèle tout en limitant le coût computationnel par rapport à une recherche exhaustive.

Conclusion

Ce projet d'analyse et de prédiction des prix des billets d'avion a permis de mettre en œuvre une démarche complète alliant exploration des données, traitement des valeurs aberrantes, et modélisation prédictive avancée.

Après un prétraitement rigoureux des données, notamment la gestion des outliers et la sélection des variables pertinentes, nous avons construit un modèle de régression basé sur le `RandomForestRegressor`. L'optimisation des hyperparamètres a été réalisée via un `RandomizedSearchCV` associée à une validation croisée, garantissant ainsi la robustesse et la généralisation du modèle.

Les résultats obtenus, avec un coefficient de détermination R^2 supérieur à 0,83 sur les données de test, démontrent une capacité satisfaisante du modèle à expliquer la variabilité des prix et à fournir des prédictions fiables. Cette performance souligne la pertinence des variables utilisées et l'efficacité des méthodes d'optimisation employées.

En conclusion, ce projet illustre l'importance d'une approche méthodique combinant exploration, nettoyage et modélisation pour répondre à un enjeu réel d'analyse de données, et ouvre la voie à des améliorations futures, notamment par l'intégration de variables additionnelles ou l'exploration de modèles alternatifs.