

```
from google.colab import drive
drive.mount('/content/gdrive')
```

Mounted at /content/gdrive

```
root_path = "gdrive/MyDrive/ML/"
```

```
!pip install --quiet optuna
```

308 kB	35.1 MB/s
209 kB	70.0 MB/s
81 kB	10.7 MB/s
78 kB	6.9 MB/s
147 kB	68.4 MB/s
49 kB	6.7 MB/s
112 kB	73.6 MB/s

Building wheel for pyperclip (setup.py) ... done

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import matplotlib
import seaborn as sns
```

```
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import KNNImputer, IterativeImputer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.feature_selection import mutual_info_classif
from sklearn.mixture import GaussianMixture
from sklearn.cluster import KMeans, DBSCAN
from yellowbrick.cluster import KElbowVisualizer
from sklearn.svm import SVC
import optuna
```

## ▼ Data Loading

```
df = pd.read_csv(root_path + "train.csv", index_col='PassengerId').reset_index(drop=True)
df_test = pd.read_csv(root_path + "test.csv", index_col='PassengerId').reset_index(drop=True)

df.head()
```

	HomePlanet	CryoSleep	Cabin	Destination	Age	VIP	RoomService	FoodCourt	Sho
0	Europa	False	B/0/P	TRAPPIST-1e	39.0	False	0.0	0.0	
1	Earth	False	F/0/S	TRAPPIST-1e	24.0	False	109.0	9.0	

```
print("Shape ")
print()
print("Train: ", df.shape)
print("Test: ", df_test.shape)
```

Shape

Train: (8693, 13)  
Test: (4277, 12)

```
print("NaN values ")
print()
print("Train: \n", df.isna().sum(), "\n")
print("Test: \n", df_test.isna().sum())
```

NaN values

Train:

HomePlanet	201
CryoSleep	217
Cabin	199
Destination	182
Age	179
VIP	203
RoomService	181
FoodCourt	183
ShoppingMall	208
Spa	183
VRDeck	188
Name	200
Transported	0

dtype: int64

Test:

HomePlanet	87
CryoSleep	93
Cabin	100
Destination	92
Age	91
VIP	93
RoomService	82
FoodCourt	106
ShoppingMall	98
Spa	101
VRDeck	80
Name	94

dtype: int64

```
df.head()
```

	HomePlanet	CryoSleep	Cabin	Destination	Age	VIP	RoomService	FoodCourt	ShoppingM
0	Europa	False	B/0/P	TRAPPIST-1e	39.0	False	0.0	0.0	
1	Earth	False	F/0/S	TRAPPIST-1e	24.0	False	109.0	9.0	
2	Europa	False	A/0/S	TRAPPIST-1e	58.0	True	43.0	3576.0	

```
df.Transported.value_counts()
```

```
1    4378
0    4315
Name: Transported, dtype: int64
```

```
cabin_splited = df["Cabin"].str.split("/", expand=True)
cabin_splited.columns = ["Cabin_deck", "Cabin_num", "Cabin_side"]
cabin_splited.head()
```

	Cabin_deck	Cabin_num	Cabin_side
0	B	0	P
1	F	0	S
2	A	0	S
3	A	0	S
4	F	1	S

```
df1 = pd.concat([df, cabin_splited], axis=1).drop("Cabin", axis=1)
df1.head()
```

	HomePlanet	CryoSleep	Destination	Age	VIP	RoomService	FoodCourt	ShoppingM
0	Europa	False	TRAPPIST-1e	39.0	False	0.0	0.0	
1	Earth	False	TRAPPIST-1e	24.0	False	109.0	9.0	2
2	Europa	False	TRAPPIST-1e	58.0	True	43.0	3576.0	

```
cabin_splited = df_test["Cabin"].str.split("/", expand=True)
cabin_splited.columns = ["Cabin_deck", "Cabin_num", "Cabin_side"]
df_test_1 = pd.concat([df_test, cabin_splited], axis=1).drop("Cabin", axis=1)
df_test_1.head()
```

	HomePlanet	CryoSleep	Destination	Age	VIP	RoomService	FoodCourt	ShoppingM
0	Earth	True	TRAPPIST-1e	27.0	False	0.0	0.0	
1	Earth	False	TRAPPIST-1e	19.0	False	0.0	9.0	
2	Europa	True	55 Cancri e	31.0	False	0.0	0.0	

```
object_cols = [i for i in df1.columns if df1[i].dtype == "O"]
```

```
for i in object_cols:
    print(i, ": ", df1[i].unique())
```

```
HomePlanet : 3
CryoSleep : 2
Destination : 3
VIP : 2
Name : 8473
Cabin_deck : 8
Cabin_num : 1817
Cabin_side : 2
```

Since Cabin\_num hasn't any relation ship is a passenger Transported, we'll remove it. With Name column the same thing

```
df2 = df1.drop(["Name", "Cabin_num"], axis=1)
df_test_2 = df_test_1.drop(["Name", "Cabin_num"], axis=1)
df2.head()
```

	HomePlanet	CryoSleep	Destination	Age	VIP	RoomService	FoodCourt	ShoppingM
0	Europa	False	TRAPPIST-1e	39.0	False	0.0	0.0	
1	Earth	False	TRAPPIST-1e	24.0	False	109.0	9.0	2
2	Europa	False	TRAPPIST-1e	58.0	True	43.0	3576.0	

## ► Handle NaN values

[ ] ↳ 9 cells hidden

## ► Encoding values

[ ] ↳ 13 cells hidden

## ▼ Model Creation and Training

```

model = RandomForestClassifier(n_estimators=100, random_state=0)

scores = cross_val_score(model, X, y, cv=10)
scores

array([0.79770115, 0.75402299, 0.78390805, 0.79171461, 0.80437284,
       0.81127733, 0.80552359, 0.79056387, 0.80782509, 0.79516686])

def run(trial):
    criterion = trial.suggest_categorical('criterion', ['gini', 'entropy'])
    bootstrap = trial.suggest_categorical('bootstrap', ['True', 'False'])
    max_depth = trial.suggest_int('max_depth', 2, 50)
    max_features = trial.suggest_categorical('max_features', ['auto', 'sqrt', 'log2'])
    max_leaf_nodes = trial.suggest_int('max_leaf_nodes', 2, 100)
    n_estimators = trial.suggest_int('n_estimators', 100, 500)

    X_train, X_val, y_train, y_val = train_test_split(X, y)
    regr = RandomForestClassifier(bootstrap = bootstrap, criterion = criterion,
                                max_depth = max_depth, max_features = max_features,
                                max_leaf_nodes = max_leaf_nodes, n_estimators = n_estimators)
    regr.fit(X_train, y_train)

    return regr.score(X_val, y_val)

study = optuna.create_study(direction='maximize')
study.optimize(run, n_trials=200)

print(study.best_params)
bootstrap : False , max_depth : 41, max_features : log2 , max_leaf_nodes : 7
'bootstrap': 'False', 'max_depth': 43, 'max_features': 'log2', 'max_leaf_nodes': 7
'bootstrap': 'False', 'max_depth': 41, 'max_features': 'log2', 'max_leaf_nodes': 7
'bootstrap': 'False', 'max_depth': 25, 'max_features': 'log2', 'max_leaf_nodes': 81
'bootstrap': 'False', 'max_depth': 6, 'max_features': 'log2', 'max_leaf_nodes': 84,
bootstrap': 'False', 'max_depth': 45, 'max_features': 'log2', 'max_leaf_nodes': 79,
bootstrap': 'False', 'max_depth': 40, 'max_features': 'log2', 'max_leaf_nodes': 92,
, 'bootstrap': 'False', 'max_depth': 37, 'max_features': 'sqrt', 'max_leaf_nodes':
'bootstrap': 'True', 'max_depth': 9, 'max_features': 'auto', 'max_leaf_nodes': 100,
, 'bootstrap': 'False', 'max_depth': 39, 'max_features': 'auto', 'max_leaf_nodes':
'bootstrap': 'False', 'max_depth': 35, 'max_features': 'sqrt', 'max_leaf_nodes': 7
'bootstrap': 'True', 'max_depth': 8, 'max_features': 'auto', 'max_leaf_nodes': 97,
'bootstrap': 'False', 'max_depth': 40, 'max_features': 'log2', 'max_leaf_nodes': 87
'bootstrap': 'True', 'max_depth': 12, 'max_features': 'auto', 'max_leaf_nodes': 94,
'bootstrap': 'True', 'max_depth': 15, 'max_features': 'auto', 'max_leaf_nodes': 91,
bootstrap': 'True', 'max_depth': 10, 'max_features': 'auto', 'max_leaf_nodes': 7,
, 'bootstrap': 'False', 'max_depth': 38, 'max_features': 'sqrt', 'max_leaf_nodes':
'bootstrap': 'True', 'max_depth': 30, 'max_features': 'auto', 'max_leaf_nodes': 95,
'bootstrap': 'True', 'max_depth': 42, 'max_features': 'auto', 'max_leaf_nodes': 98,
bootstrap': 'False', 'max_depth': 7, 'max_features': 'auto', 'max_leaf_nodes': 61,

```

```

'bootstrap': 'True', 'max_depth': 11, 'max_features': 'auto', 'max_leaf_nodes': 98,
'bootstrap': 'False', 'max_depth': 36, 'max_features': 'auto', 'max_leaf_nodes': 65,
, 'bootstrap': 'True', 'max_depth': 20, 'max_features': 'auto', 'max_leaf_nodes': 9
'bootstrap': 'True', 'max_depth': 13, 'max_features': 'sqrt', 'max_leaf_nodes': 76,
'bootstrap': 'False', 'max_depth': 32, 'max_features': 'auto', 'max_leaf_nodes': 100
'bootstrap': 'False', 'max_depth': 30, 'max_features': 'auto', 'max_leaf_nodes': 90
, 'bootstrap': 'False', 'max_depth': 32, 'max_features': 'log2', 'max_leaf_nodes':
, 'bootstrap': 'False', 'max_depth': 32, 'max_features': 'log2', 'max_leaf_nodes':
'bootstrap': 'False', 'max_depth': 32, 'max_features': 'log2', 'max_leaf_nodes': 100
'bootstrap': 'False', 'max_depth': 34, 'max_features': 'log2', 'max_leaf_nodes': 100
'bootstrap': 'False', 'max_depth': 29, 'max_features': 'auto', 'max_leaf_nodes': 98
'bootstrap': 'False', 'max_depth': 29, 'max_features': 'log2', 'max_leaf_nodes': 100
'bootstrap': 'False', 'max_depth': 28, 'max_features': 'auto', 'max_leaf_nodes': 94
'bootstrap': 'False', 'max_depth': 31, 'max_features': 'auto', 'max_leaf_nodes': 97
'bootstrap': 'False', 'max_depth': 27, 'max_features': 'auto', 'max_leaf_nodes': 91
'bootstrap': 'False', 'max_depth': 26, 'max_features': 'log2', 'max_leaf_nodes': 95,
'bootstrap': 'False', 'max_depth': 31, 'max_features': 'auto', 'max_leaf_nodes': 93
'bootstrap': 'False', 'max_depth': 30, 'max_features': 'auto', 'max_leaf_nodes': 93
'bootstrap': 'False', 'max_depth': 33, 'max_features': 'auto', 'max_leaf_nodes': 89
'bootstrap': 'False', 'max_depth': 32, 'max_features': 'auto', 'max_leaf_nodes': 89
'bootstrap': 'False', 'max_depth': 33, 'max_features': 'auto', 'max_leaf_nodes': 98
'bootstrap': 'False', 'max_depth': 31, 'max_features': 'auto', 'max_leaf_nodes': 92
'bootstrap': 'False', 'max_depth': 29, 'max_features': 'auto', 'max_leaf_nodes': 88
'bootstrap': 'False', 'max_depth': 34, 'max_features': 'auto', 'max_leaf_nodes': 96,
, 'bootstrap': 'False', 'max_depth': 33, 'max_features': 'log2', 'max_leaf_nodes':
'bootstrap': 'False', 'max_depth': 28, 'max_features': 'auto', 'max_leaf_nodes': 94
'bootstrap': 'True', 'max_depth': 32, 'max_features': 'auto', 'max_leaf_nodes': 98,
'bootstrap': 'False', 'max_depth': 30, 'max_features': 'auto', 'max_leaf_nodes': 100
'bootstrap': 'False', 'max_depth': 31, 'max_features': 'auto', 'max_leaf_nodes': 86
'bootstrap': 'False', 'max_depth': 33, 'max_features': 'auto', 'max_leaf_nodes': 91,
'bootstrap': 'False', 'max_depth': 30, 'max_features': 'auto', 'max_leaf_nodes': 90
'bootstrap': 'False', 'max_depth': 9, 'max_features': 'auto', 'max_leaf_nodes': 81,
'bootstrap': 'False', 'max_depth': 34, 'max_features': 'log2', 'max_leaf_nodes': 53
'bootstrap': 'True', 'max_depth': 27, 'max_features': 'auto', 'max_leaf_nodes': 78,
, 'bootstrap': 'False', 'max_depth': 35, 'max_features': 'auto', 'max_leaf_nodes':
'bootstrap': 'True', 'max_depth': 29, 'max_features': 'auto', 'max_leaf_nodes': 98,
tors': 465}

```

```

rf_params = study.best_params
rf_params

```

```

{'criterion': 'gini',
 'bootstrap': 'True',
 'max_depth': 37,
 'max_features': 'auto',
 'max_leaf_nodes': 100,
 'n_estimators': 465}

```

```
model = RandomForestClassifier(**rf_params, n_jobs=2, random_state=0)
```

```

model.fit(X, y)
print("Done!")


```

Done!

```
preds = model.predict(df_test_6).round()
preds
```

```
array([1, 0, 1, ..., 1, 1, 1], dtype=int8)
```

```
ss = pd.read_csv(root_path + "sample_submission.csv")
ss.Transported = preds.astype(bool)
ss
```

	PassengerId	Transported	
0	0013_01	True	
1	0018_01	False	
2	0019_01	True	
3	0021_01	True	
4	0023_01	True	
...	...	...	
4272	9266_02	True	
4273	9269_01	False	
4274	9271_01	True	
4275	9273_01	True	
4276	9277_01	True	

4277 rows × 2 columns

```
ss.to_csv(root_path + "submission.csv", index=False)
```

```
X_train, X_val, y_train, y_val = train_test_split(X, y)
```

```
from scipy.sparse.sputils import matrix
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
model = LogisticRegression(max_iter = 1000)
model.fit(X_train, y_train)
prediction = model.predict(X_val)
print('The accuracy of the Logistic Regression is', accuracy_score(prediction, y_val))
```

📄 The accuracy of the Logistic Regression is 0.7805887764489421  
 /usr/local/lib/python3.7/dist-packages/sklearn/linear\_model/\_logistic.py:818: Converge  
 STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max\_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

[https://scikit-learn.org/stable/modules/linear\\_model.html#logistic-regression](https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

extra\_warning\_msg=\_LOGISTIC\_SOLVER\_CONVERGENCE\_MSG,



✓ 2s completed at 8:47 PM

