Spaceship Titanic: Proyecto de Aprendizaje Automático

Tomás Ausensi

Diciembre 2023

Table of contents

Introducción

Spaceship Titanic

El desafío es una versión modificada del Titanic: se situa en un contexto espacial, donde, por ciertas anomalías espacio temporales, parte de la tripulación fue transportada del barco a otra dimensión.

Introducción

Así, La idea es utilizar herramientas del aprendizaje automático para predecir, dadas ciertas características de cada tripulante, si fue transportado, o no.

Spaceship Titanic

Resumen

- Datasets
- Porcentaje de valores nulos
- Valores de cada columna
- Ingeniería de atributos

Datasets

¿Cuál es el contenido de los datasets?

- train set
- test set

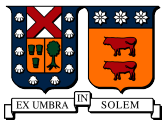
Datasets

El train set posee las siguientes columnas:

- HomePlanet
- CryoSleep
- Cabin
- Destination
- Age
- Roomservice
- FoodCourt
- ShoppingMall
- Spa
- VRDeck
- Name
- Transported

Datasets

El test set posee las mismas columnas, pero sin el target (Transported)



Valores Nulos

```
train_set.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8693 entries, 0 to 8692
Data columns (total 14 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   PassengerId   8693 non-null   object
 1   HomePlanet    8492 non-null   object
 2   CryoSleep     8476 non-null   object
 3   Cabin         8494 non-null   object
 4   Destination   8511 non-null   object
 5   Age           8514 non-null   float64
 6   VIP           8490 non-null   object
 7   RoomService   8512 non-null   float64
 8   FoodCourt     8510 non-null   float64
 9   ShoppingMall  8485 non-null   float64
 10  Spa           8510 non-null   float64
 11  VRDeck        8505 non-null   float64
 12  Name          8493 non-null   object
 13  Transported   8693 non-null   bool
dtypes: bool(1), float64(6), object(7)
memory usage: 891.5+ KB
```

train set info

```
test_set.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4277 entries, 0 to 4276
Data columns (total 13 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   PassengerId   4277 non-null   object
 1   HomePlanet    4190 non-null   object
 2   CryoSleep     4184 non-null   object
 3   Cabin         4177 non-null   object
 4   Destination   4185 non-null   object
 5   Age           4186 non-null   float64
 6   VIP           4184 non-null   object
 7   RoomService   4195 non-null   float64
 8   FoodCourt     4171 non-null   float64
 9   ShoppingMall  4179 non-null   float64
 10  Spa           4176 non-null   float64
 11  VRDeck        4197 non-null   float64
 12  Name          4183 non-null   object
dtypes: float64(6), object(7)
memory usage: 434.5+ KB
```

test set info

Valores Nulos

```
PassengerId     0.000000
HomePlanet      2.312205
CryoSleep       2.496261
Cabin           2.289198
Destination     2.093639
Age             2.059128
VIP             2.335212
RoomService     2.082135
FoodCourt       2.105142
ShoppingMall    2.392730
Spa             2.105142
VRDeck          2.162660
Name            2.300702
Transported     0.000000
dtype: float64
```

% valores nulos train

```
PassengerId     0.000000
HomePlanet      2.034136
CryoSleep       2.174421
Cabin           2.338087
Destination     2.151040
Age             2.127660
VIP             2.174421
RoomService     1.917232
FoodCourt       2.478373
ShoppingMall    2.291326
Spa             2.361468
VRDeck          1.870470
Name            2.197802
dtype: float64
```

% valores nulos test

Valores por columna

```
Unique values for the column PassengerId: 8693
Unique values for the column HomePlanet: 3
Unique values for the column CryoSleep: 2
Unique values for the column Cabin: 6560
Unique values for the column Destination: 3
Unique values for the column Age: 80
Unique values for the column VIP: 2
Unique values for the column RoomService: 1273
Unique values for the column FoodCourt: 1507
Unique values for the column ShoppingMall: 1115
Unique values for the column Spa: 1327
Unique values for the column VRDeck: 1306
Unique values for the column Name: 8473
Unique values for the column Transported: 2
```

Valores únicos por columna

Valores por columna

```
Unique values for the column PassengerId: ['0001_01' '0002_01' '0003_01' ... '9279_01' '9280_01' '9280_02']
Unique values for the column HomePlanet: ['Europa' 'Earth' 'Mars' nan]
Unique values for the column CryoSleep: [False True nan]
Unique values for the column Cabin: ['B/0/P' 'F/0/S' 'A/0/S' ... 'G/1499/S' 'G/1500/S' 'E/608/S']
Unique values for the column Destination: ['TRAPPIST-1e' 'PSO J318.5-22' '55 Cancri e' nan]
Unique values for the column Age: [39. 24. 58. 33. 16. 44. 26. 28. 35. 14. 34. 45. 32. 48. 31. 27.  0.  1.
 49. 29. 10.  7. 21. 62. 15. 43. 47.  2. 20. 23. 30. 17. 55.  4. 19. 56.
 nan 25. 38. 36. 22. 18. 42. 37. 13.  8. 40.  3. 54.  9.  6. 64. 67. 61.
 50. 41. 57. 11. 52. 51. 46. 60. 63. 59.  5. 79. 68. 74. 12. 53. 65. 71.
 75. 70. 76. 78. 73. 66. 69. 72. 77.]
Unique values for the column VIP: [False True nan]
Unique values for the column RoomService: [   0.  109.   43. ... 1569. 8586.  745.]
Unique values for the column FoodCourt: [   0.    9. 3576. ... 3208. 6819. 4688.]
Unique values for the column ShoppingMall: [   0.   25.  371. ... 1085.  510. 1872.]
Unique values for the column Spa: [   0.  549. 6715. ... 2868. 1107. 1643.]
Unique values for the column VRDeck: [   0.   44.   49. ... 1164.  971. 3235.]
Unique values for the column Name: ['Maham Ofracculy' 'Juanna Vines' 'Altark Susent' ... 'Fayey Connon'
 'Celeon Hontichre' 'Propsh Hontichre']
Unique values for the column Transported: [False  True]
```

Valores únicos por columna

Ingeniería de atributos

Según el desafío en Kaggle, la columna *Cabin* consiste de tres valores: Uno corresponde a la cubierta, otro al número de cabina y el último corresponde a si la cabina se ubica en babor o en estribor.

Así, es posible separar la columna en tres distintas. La ventaja de esto es reducir la cardinalidad (de alrededor de 6500) a 8, 1817 y 2, respectivamente.

Ingeniería de atributos

Es posible realizar lo mismo para la columna *PassengerId*, pero no se realizó en esta ocasión

Datos Categóricos

Lo primero es dividir las columnas por sus categorías:

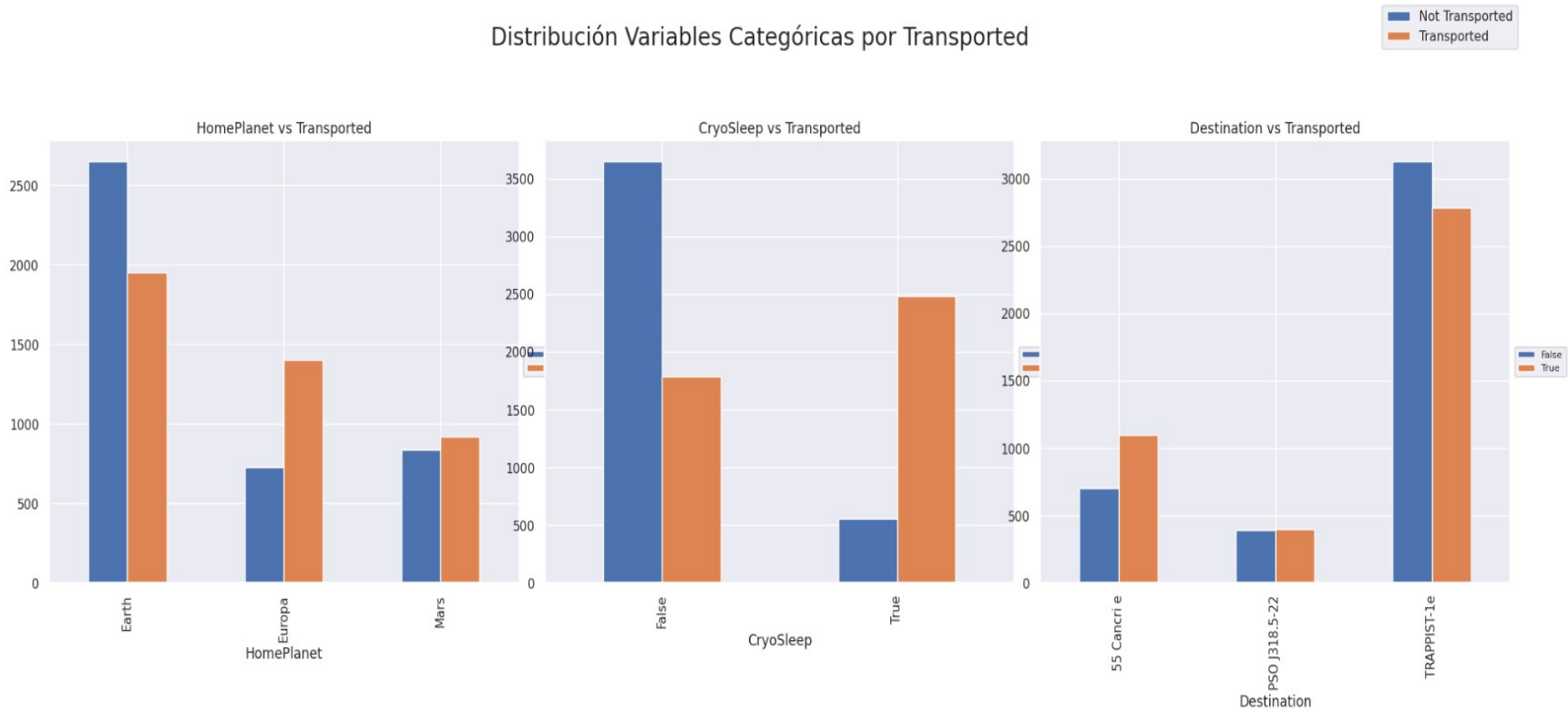
```
categorical_cols = [col for col in train_set2.columns
                    if train_set2[col].dtype == 'object']

numerical_cols = [col for col in train_set2.columns
                  if train_set2[col].dtype in ['int64', 'float64']
                  and col != 'Transported']

target = train_set2['Transported']
```

```
print(categorical_cols)
print(numerical_cols)
```

```
['HomePlanet', 'CryoSleep', 'Destination', 'VIP', 'Deck', 'Side']
['Age', 'RoomService', 'FoodCourt', 'ShoppingMall', 'Spa', 'VRDeck']
```

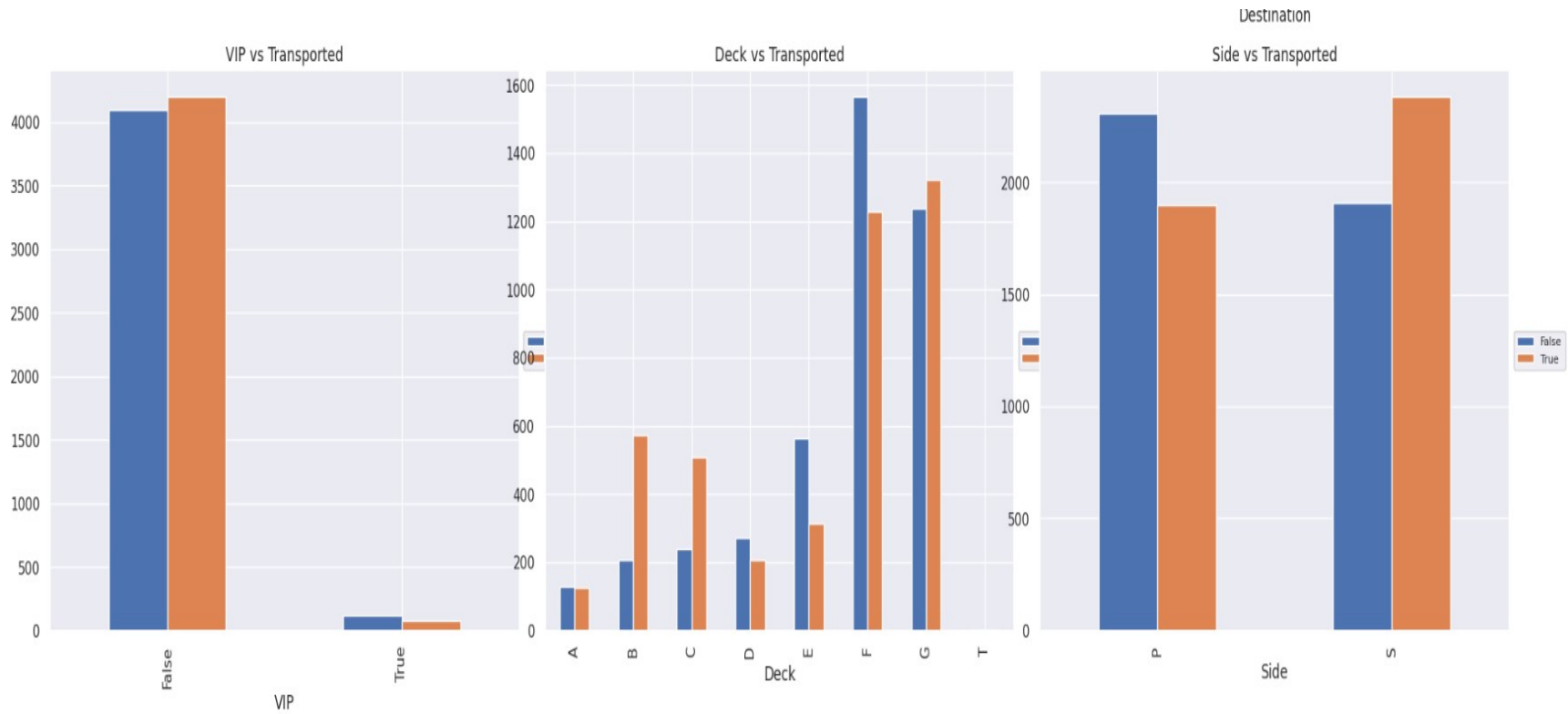
Columnas según sus tipos

Datos Categóricos



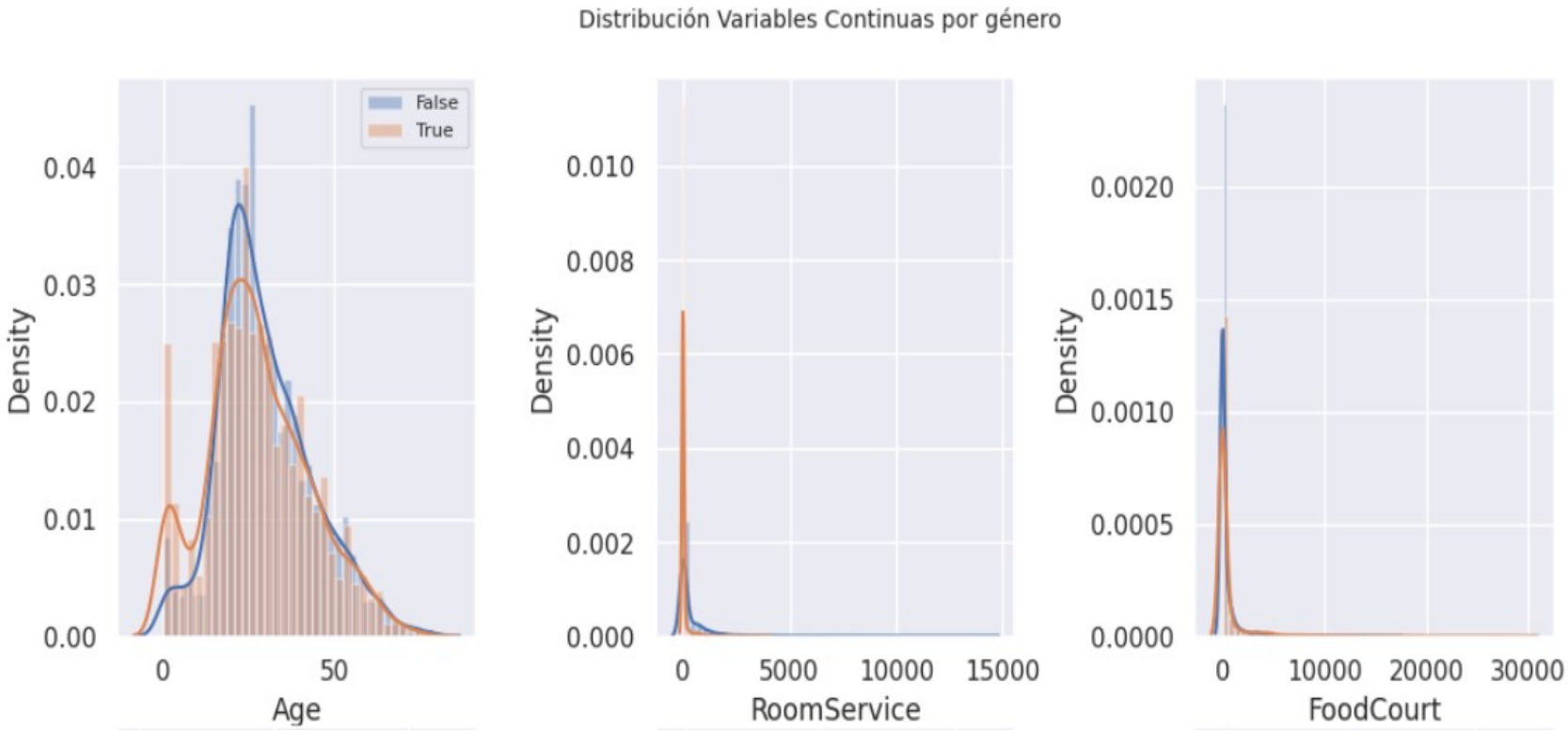
Gráficos datos categóricos

Datos Categóricos



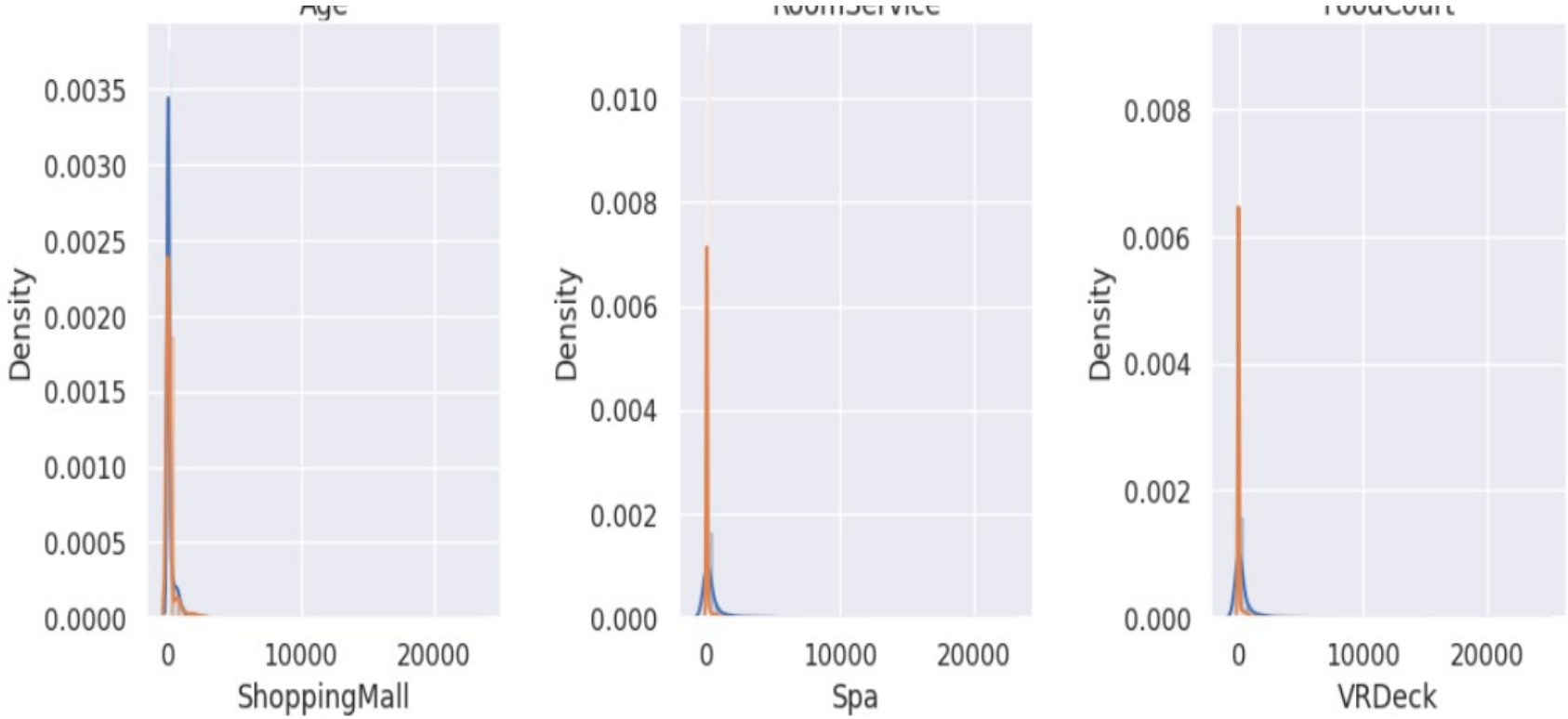
Gráficos datos categóricos

Datos numéricos



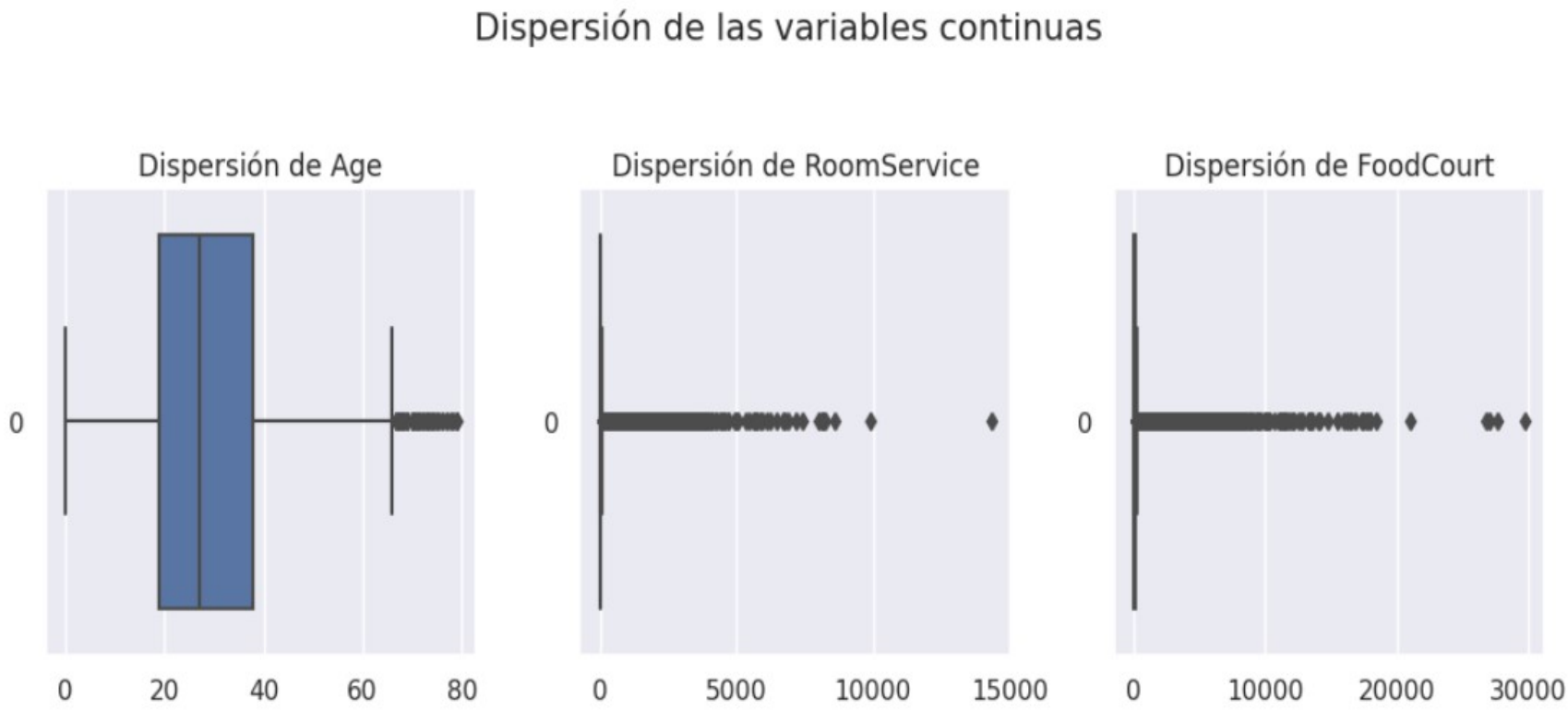
Gráficos datos continuos

Datos numéricos



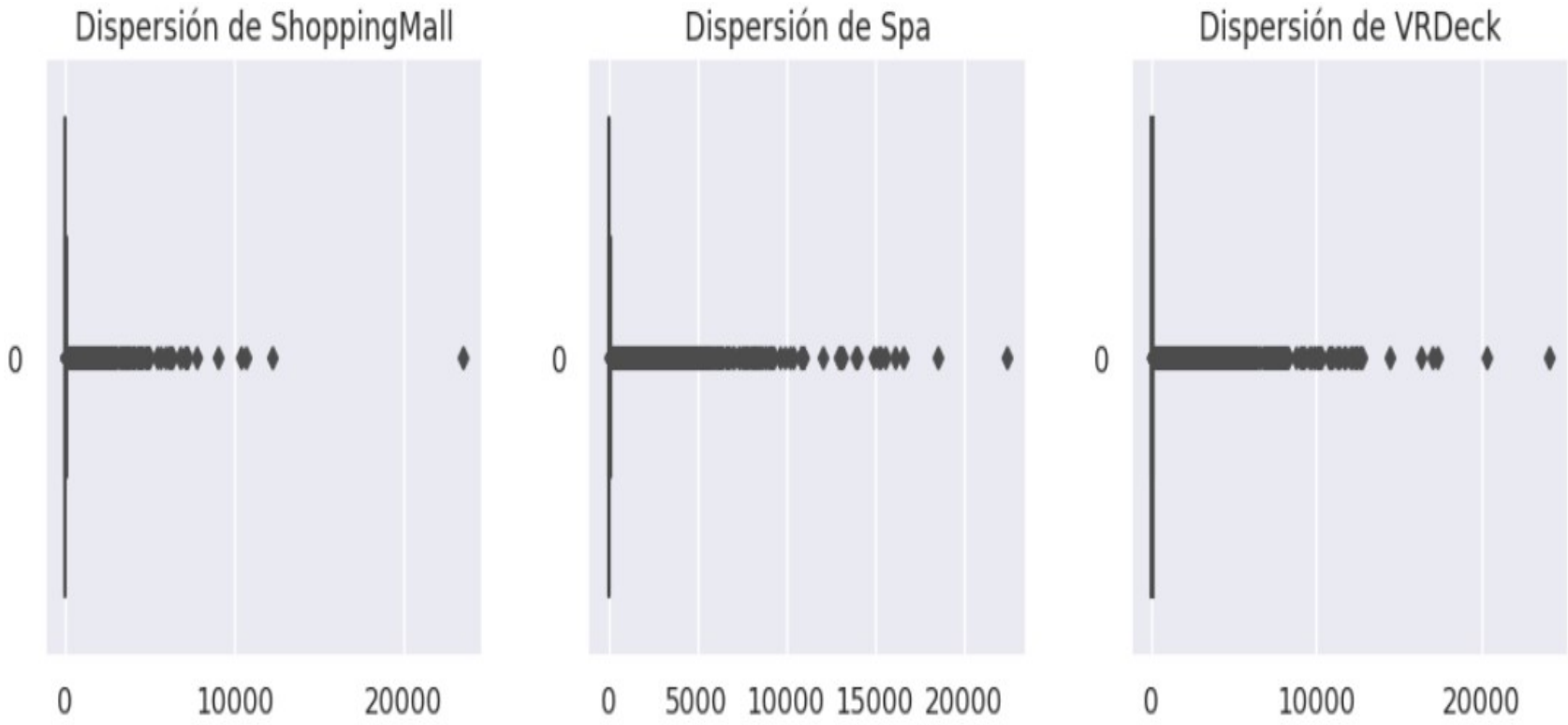
Gráficos datos continuos

Datos numéricos



Dispersión datos continuos

Datos numéricos



Dispersión datos continuos

Imputers, Encoders y Transformers

En resumen, el preprocesamiento, por un lado, será incorporado a la pipeline de los modelos. Por otro lado, se compone de distintas subpipelines que se encargan de cada tipo de columna en específico:

- **SimpleImputer** para los datos nulos
- **KNNImputer** para datos nulos continuos
- **OneHotEncoder** para columnas categóricas
- **OrdinalEncoder** para columna *Deck*
- **StandardScaler** para columnas continuas
- **LabelEncoder** para la columna Target

Imputers, Encoders y Transformers

```
numeric_transformer = Pipeline(steps=[
    ('imputer', KNNImputer(n_neighbors = 10, weights = 'distance')),
    ('scaler', StandardScaler())
])

categorical_transformer_ohe = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy = 'most_frequent')),
    ('encoder', OneHotEncoder(drop = 'if_binary'))
])

categorical_transformer_oe = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy = 'most_frequent')),
     ('encoder', OrdinalEncoder()),
])

label_encoder = LabelEncoder()

preprocessor = ColumnTransformer(
    transformers=[
        ('numerical', numeric_transformer, numerical_cols),
        ('categorical_ohe', categorical_transformer_ohe, categorical_cols_ohe),
        ('categorical_te', categorical_transformer_oe, categorical_cols_oe)
    ],
    remainder = 'passthrough', verbose_feature_names_out = False
)
```

Preprocesador

Modelos y grillas de parámetros

Los modelos a seleccionar son:

- **LogisticRegression**
- **KNeighborsClassifier**
- **RandomForestClassifier**
- **GradientBoostingClassifier**
- **XGBClassifier**
- **AdaBoostClassifier**

Modelos y grillas de parámetros

Las grillas de parámetros son extensas, pero, para dar un ejemplo:

```
    'RandomForest' : {
    'model__n_estimators': [50, 100, 200],
    'model__max_depth': [None, 10, 20, 30],
    'model__min_samples_split': [2, 5, 10],
    'model__min_samples_leaf': [1, 2, 4]
},
```

Grilla de RandomForest

Modelos y grillas de parámetros

Posteriormente, se utiliza RandomizedSearchCV para la búsqueda de parámetros óptimos para cada modelo. Después de ello, se evalua el performance utilizando cross-validation, para obtener las distintas métricas

Métricas

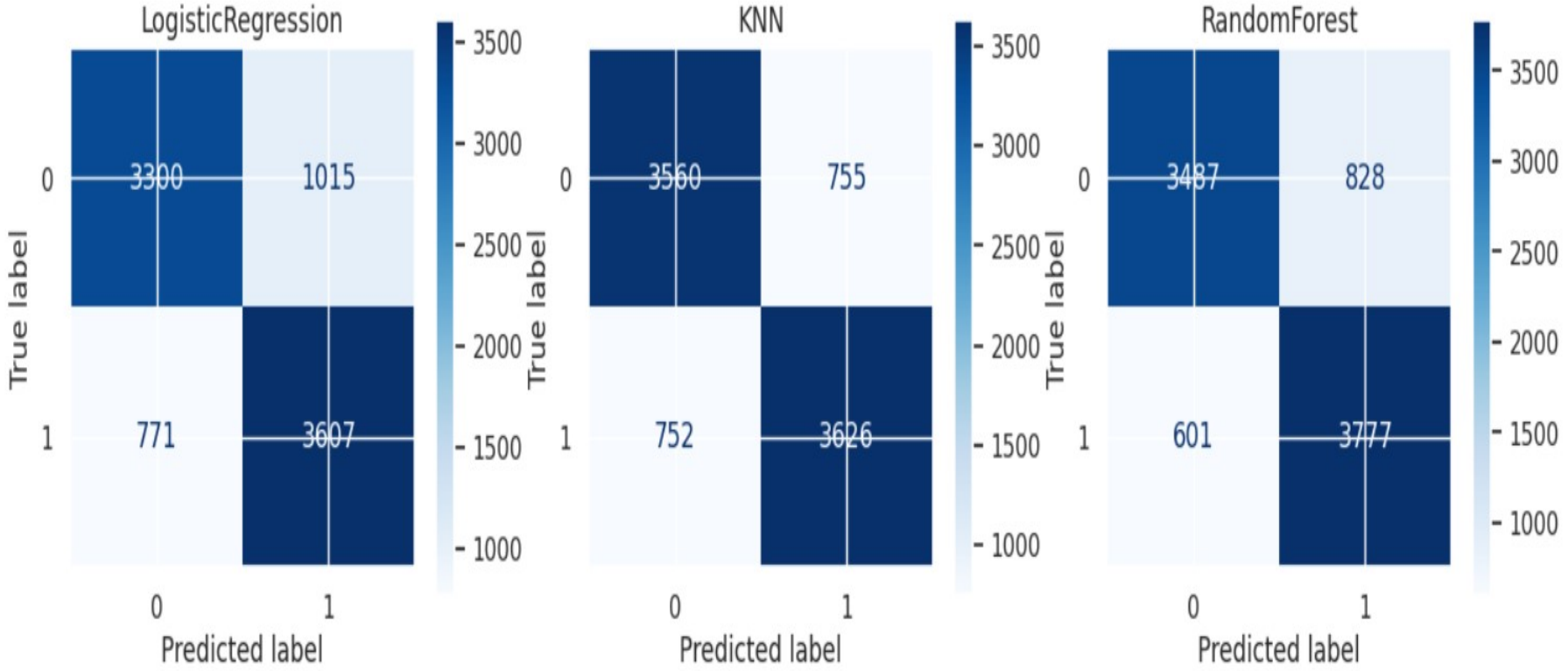
	Model	accuracy	precision	recall	f1
0	LogisticRegression	0.790178	0.791651	0.790178	0.789862
1	KNN	0.777639	0.778905	0.777639	0.777396
2	RandomForest	0.798116	0.799285	0.798116	0.797867
3	GradientBoosting	0.799957	0.801859	0.799957	0.799568
4	XGBoost	0.802257	0.804080	0.802257	0.801884
5	AdaBoost	0.751296	0.752121	0.751296	0.751038

Resultados de cross-validation

Métricas

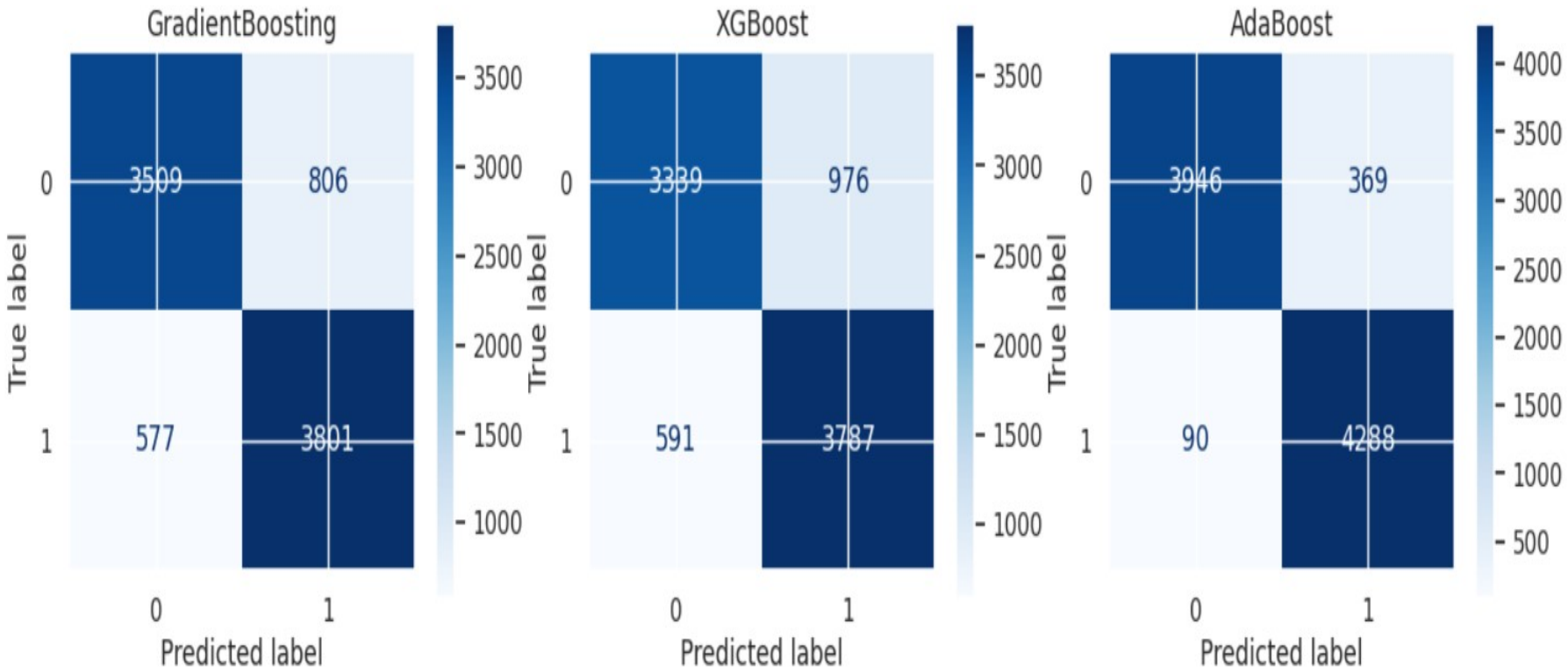
Es importante notar la importancia del uso de algun método de validación del performance para obtener una estimación del error de predicción. El uso de CV permitió obtener una estimación robusta del error de predicción. En este caso, el modelo escogido es XGBoost, por ser **accuracy** la métrica de la competencia

Matrices de confusión



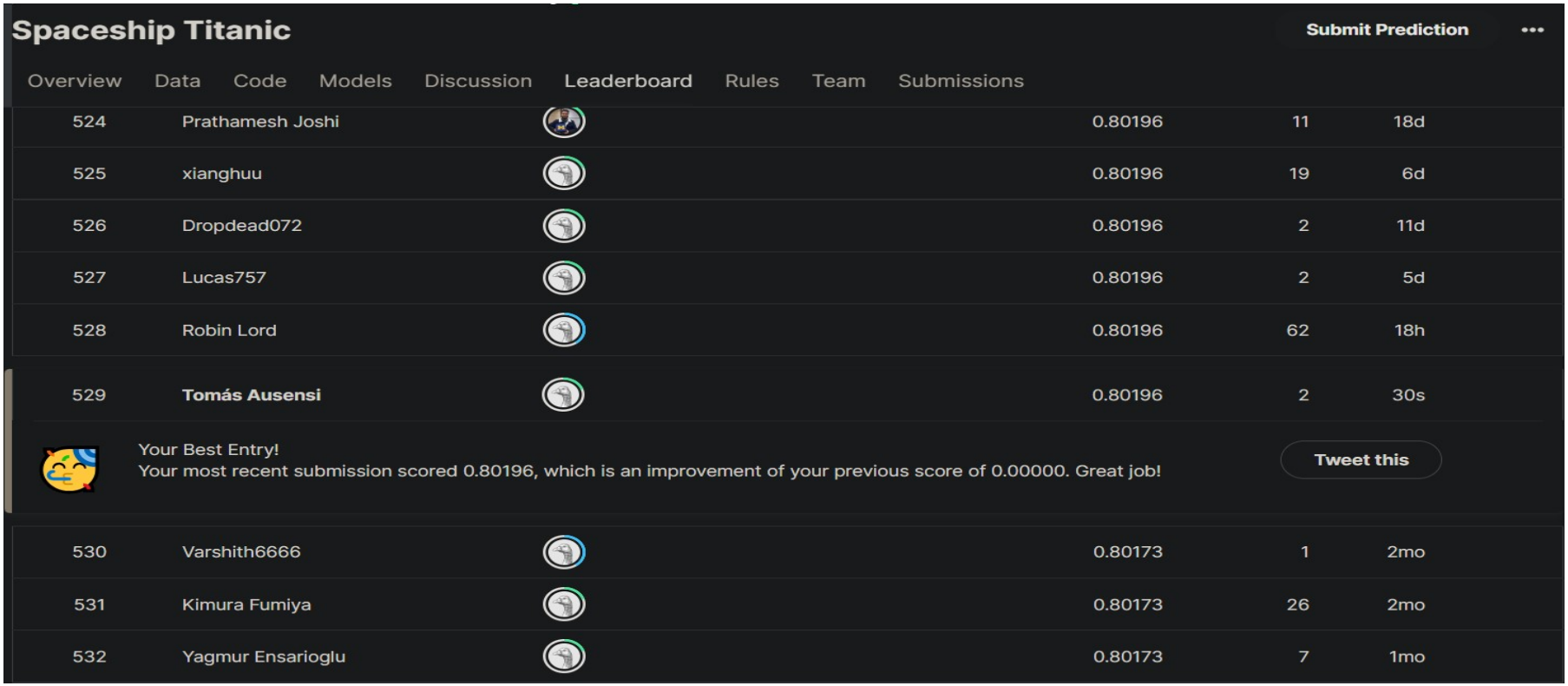
Matrices de confusión

Matrices de confusión



Matrices de confusión

Conclusiones



Spaceship Titanic — Submit Prediction

Overview | Data | Code | Models | Discussion | **Leaderboard** | Rules | Team | Submissions

524	Prathamesh Joshi	0.80196	11	18d
525	xianghuu	0.80196	19	6d
526	Dropdead072	0.80196	2	11d
527	Lucas757	0.80196	2	5d
528	Robin Lord	0.80196	62	18h
529	**Tomás Ausensi**	0.80196	2	30s
530	Varshith6666	0.80173	1	2mo
531	Kimura Fumiya	0.80173	26	2mo
532	Yagmur Ensarioglu	0.80173	7	1mo

Your Best Entry!
Your most recent submission scored 0.80196, which is an improvement of your previous score of 0.00000. Great job!

Tweet this

Resultados en la competencia

Conclusiones

- El análisis de los datos nos permite detectar ciertas regulradidades/irregularidades.
- Es importante notar las columnas y las posibilidades de imputación/codificación/transformación.
- La estimación del error de predicción mediante alguna técnica de validación es sumamente importante.

Conclusiones

Sobre los aspectos a mejorar:

- Se debe profundizar en las técnicas de imputación
- Un EDA más profundo puede detectar otro tipo de correspondencias que se pueden explotar para mejorar el poder predictivo de los modelos
- La ingeniería de atributos puede ser ampliamente fortalecida
- Utilizar GridSearchCV en vez de RandomizedSearchCV

References

[1] Ryan Holbrook Addison Howard Ashley Chow. *Spaceship Titanic*. 2022. URL: https://kaggle.com/competitions/spaceship-titanic.

[2] Fabian Pedregosa et al. "Scikit-learn: Machine learning in Python". In: *Journal of machine learning research* 12.Oct (2011), pp. 2825–2830.