

# Titanic Survivability

---

Di Tanna Alessandro - Favale Enrico - Di Canio Emanuele - Palella Piergaetano

# Dataset e preprocessing

---

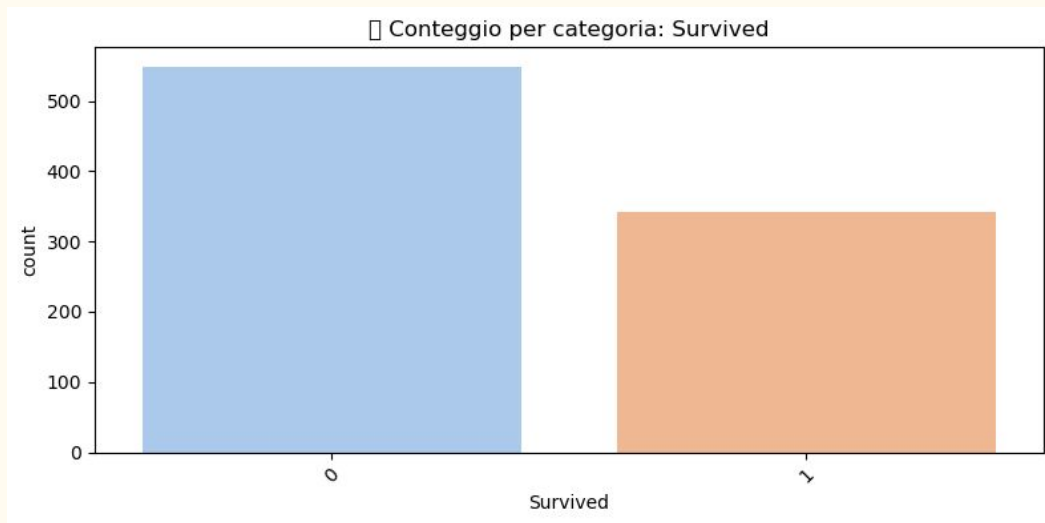
# Dataset

Istanze: **891**

Feature: **12**

- PassengerId : **int**
  - Survived : [ **Yes, No** ]
  - Pclass : [ **1, 2, 3** ]
  - Name : **str**
  - Sex : [ **male, female** ]
  - Age : **float**
  - SibSp : **int**
  - Parch : **int**
  - Ticket : **str**
  - Fare : **float**
  - Cabin : **str**
  - Embarked : [ **C, Q, S** ]
-

# Suddivisione delle classi



Classi

1	Sopravvissuto - 342
0	Non Sopravvissuto - 549

# Preprocessing

1. Rimozione di colonne inutili
  2. Gestione dei valori mancanti
  3. Aggiunta di feature riassuntive
  4. Normalizzazione dei dati
-

# — Rimozione delle colonne inutili —

---

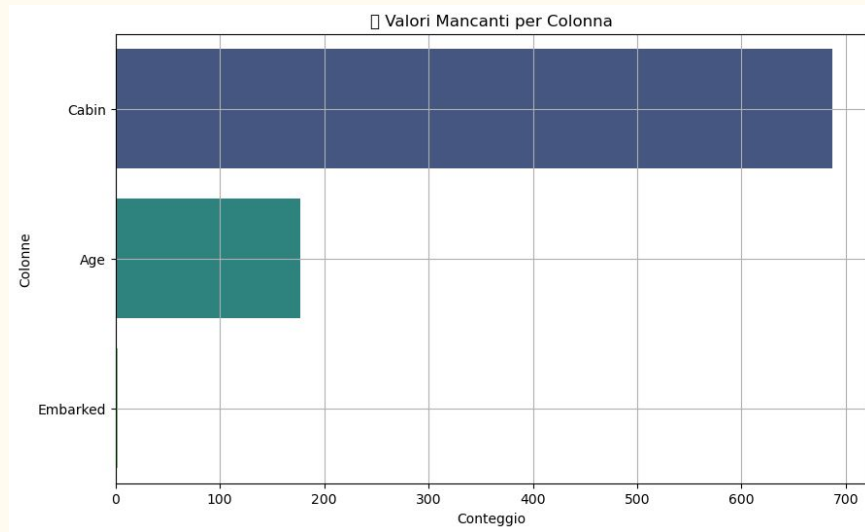
Colonne eliminate:

1. PassengerId
2. Name
3. Ticket
4. Cabin

Queste colonne contengono **dati univoci** che non apportano valore predittivo al modello. La loro rimozione ha aiutato a **ridurre l'overfitting**, evitando che il modello impari a memoria **informazioni irrilevanti**.

# — Gestione dei valori mancanti

- **Cabin** - creazione di una feature binaria '**HasCabin**' per determinare se la cabina è presente
- **Age** - gestione dei valori mancanti con la mediana
- **Embarked** - gestione dei valori nulli con la **moda** (valore più frequente).



La distribuzione dei **valori mancanti** per colonna.

# — Aggiunta di feature riassuntive —

---

**Title** - estrazione dei titoli [ ' Rare ', ' Miss ', ' Mr ' ] dal nome dei passeggeri. Utile nella definizione della classe sociale del passeggero insieme ad altre feature.

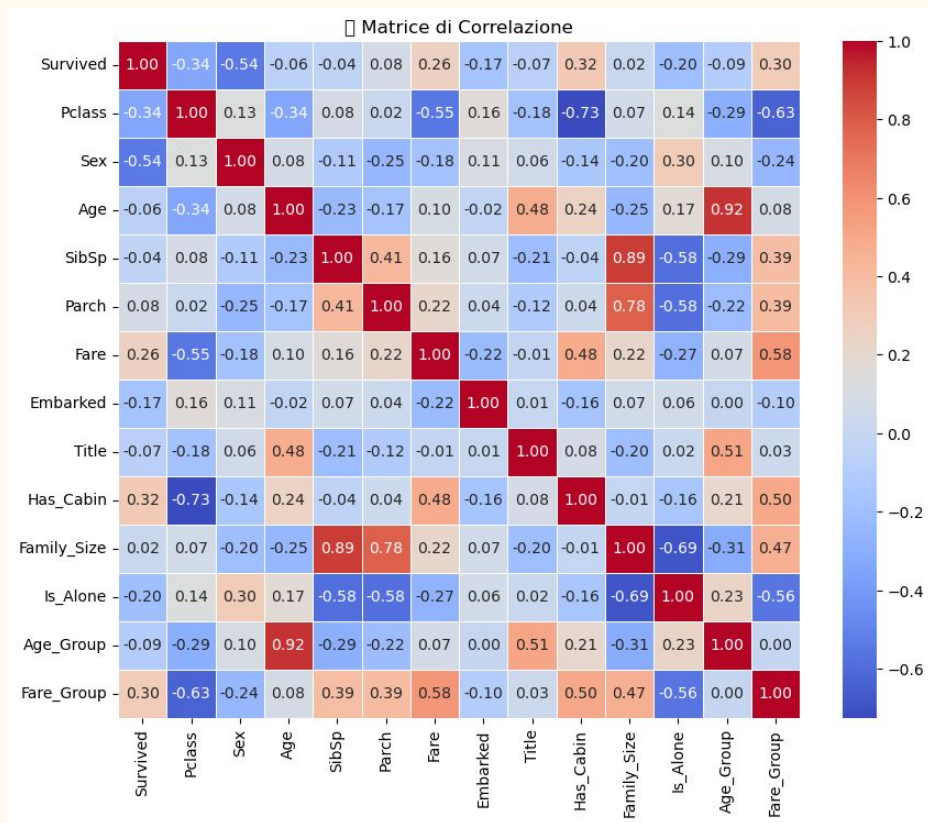
**Family\_Size, Is\_Alone** - feature complementari derivanti da **SibSp** e **Parch**.

**Age\_Group** - creazione di 5 fasce di età con soglie: [0, 12, 18, 35, 60, 100].

**Fare\_Group** - creazione di 4 fasce di tariffa.



# Analisi della correlazione tra feature



Alcune variabili mostrano forti correlazioni con la probabilità di sopravvivenza:

→ Sex

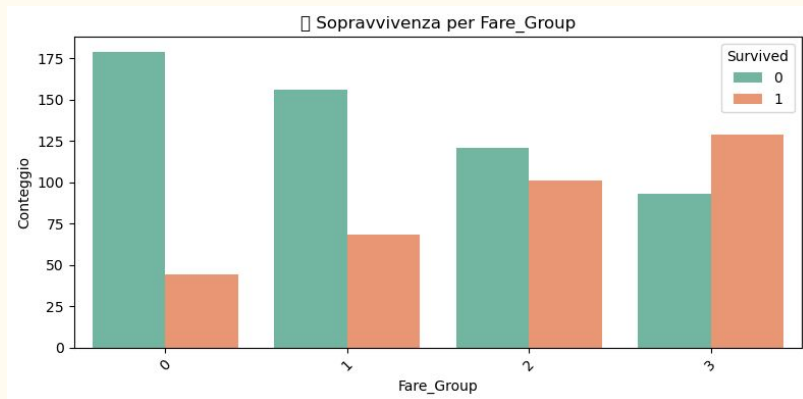
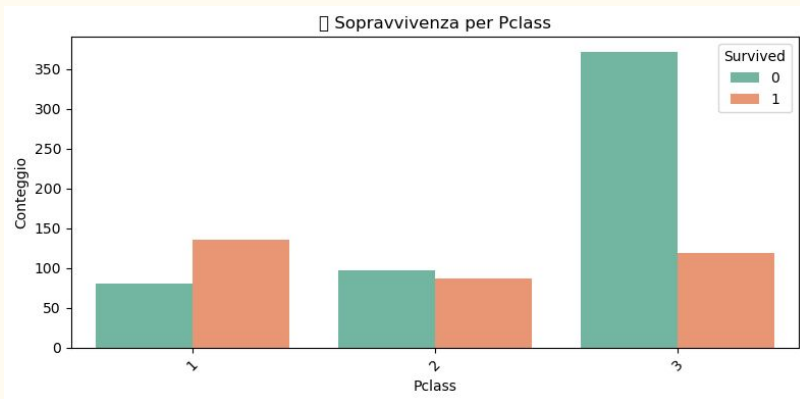
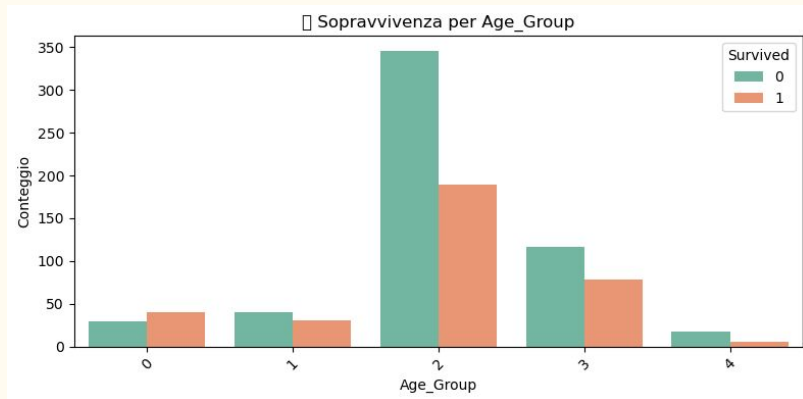
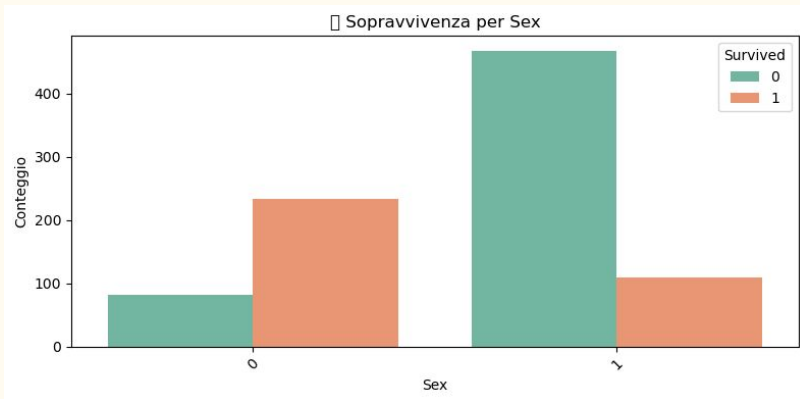
→ Fare / Fare\_Group

→ Has\_Cabin

→ Pclass

Queste feature sono di particolare interesse nella fase di selezione del modello, poiché potrebbero contribuire significativamente alla predizione dell'output (Survived).

# Feature di maggiore rilevanza



# — Normalizzazione dei dati —

**Obiettivo:** portare le feature su una scala comune per migliorare le prestazioni degli algoritmi.

→ viene istanziato uno **scaler** tramite `StandardScaler()`.

→ in seguito viene applicato il metodo `.fit_transform()` sui dati di training.

**Risultato:**

Media = <b>0</b>
Deviazione standard = <b>1</b>

# Le metriche

1. **Accuracy:** quanto spesso il modello predice correttamente (su tutti i campioni).
  2. **Precision:** quanti tra i positivi predetti sono davvero positivi.
  3. **F1-score:** media armonica tra precision e recall; utile in caso di classi sbilanciate.
  4. **Loss:** misura dell'errore complessivo del modello durante addestramento e validazione.
-

# Struttura della rete

---

# — Rete Fully Connected —

---

- Layer di input con **160 neuroni**, funzione di attivazione **ReLU** e **Dropout** al **10%**.
- **3 hidden layer** da **104**, **48** e **112** neuroni, con **ReLU** e **Dropout** rispettivamente al **40%**, **40%** e **20%**.
- Output layer con **1 unità** e funzione di attivazione **Sigmoid**.

Ogni layer è seguito da una **BatchNormalization** per migliorare la stabilità dell'allenamento.

# Training e Tuning

---

# Training e Tuning

1. Numero di epoche: **37**
  2. Batch Size: **64**
  3. Training set: **70%**
  4. Validation set: **20%**
  5. Test set: **10%**
  6. Learning rate: **5e-4**
  7. Optimizer: ***Adam***
  8. Loss: ***BinaryCrossentropy***
-



# — Inizializzazione del modello

```
1 model = TitanicSurvivalModel(dataset_path = 'data/dataset.csv',
2                               epochs = 37,
3                               batch_size = 64,
4                               test_split = 0.1,
5                               validation_split = 0.2,
6                               learning_rate = 5e-4,
7                               )
```

La **configurazione ottimale** del modello è stata individuata tramite **auto-tuning** con **Keras Tuner**, per determinare:

- il numero di layer,
- i neuroni per layer,
- il tasso di dropout,
- il learning rate.

```
1 self.model = tf.keras.models.Sequential([
2     # Input layer
3     tf.keras.layers.Dense(160, activation='relu', input_shape=(self.X_train.shape[1])),
4     tf.keras.layers.BatchNormalization(),
5     tf.keras.layers.Dropout(0.1),
6
7     # Hidden layers
8     tf.keras.layers.Dense(104, activation='relu'),
9     tf.keras.layers.BatchNormalization(),
10    tf.keras.layers.Dropout(0.4),
11
12    tf.keras.layers.Dense(48, activation='relu'),
13    tf.keras.layers.BatchNormalization(),
14    tf.keras.layers.Dropout(0.4),
15
16    tf.keras.layers.Dense(112, activation='relu'),
17    tf.keras.layers.BatchNormalization(),
18    tf.keras.layers.Dropout(0.2),
19
20    # Output layer
21    tf.keras.layers.Dense(1, activation='sigmoid')
22 ])
23
24 optimizer = tf.keras.optimizers.Adam(learning_rate=self.learning_rate)
25 loss = tf.keras.losses.BinaryCrossentropy()
26
27 self.model.compile(
28     optimizer=optimizer,
29     loss=loss,
30     metrics=[
31         tf.keras.metrics.Accuracy(name="accuracy"),
32         tf.keras.metrics.Precision(name="precision"),
33         tf.keras.metrics.F1Score(name="f1_score")
34     ]
35 )
```

# Autotuning

**RandomSearch** è un algoritmo che esplora **combinazioni casuali** di iperparametri per trovare la configurazione ottimale del modello.

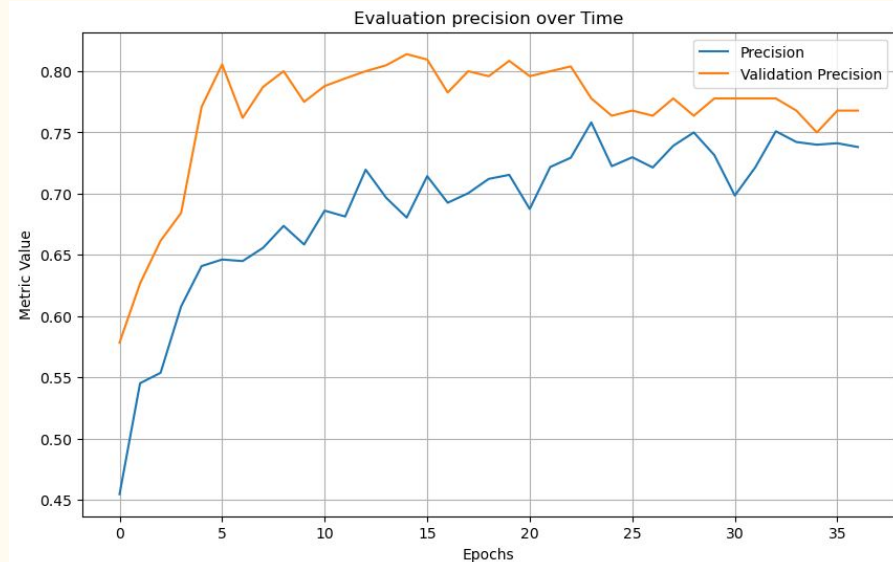
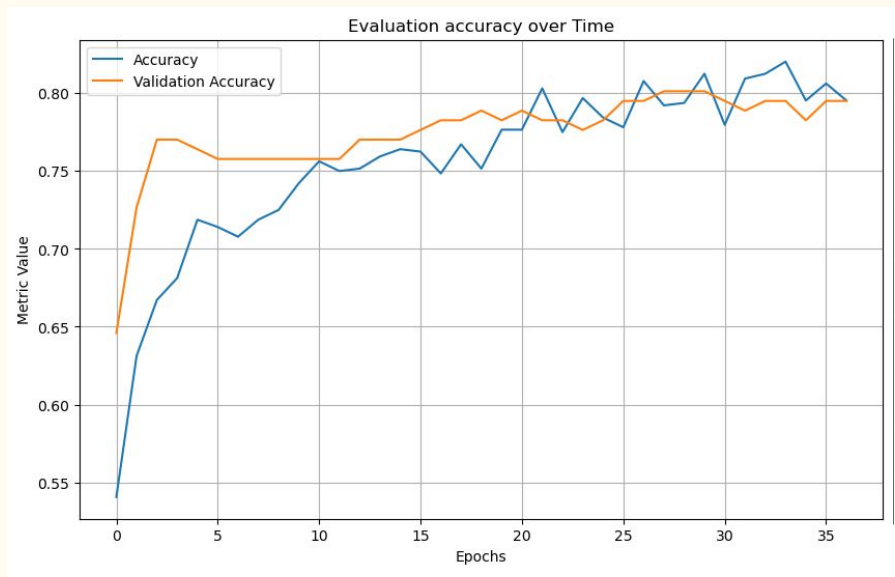
La metrica scelta da ottimizzare è la **validation accuracy**, con l'obiettivo di *massimizzarla*.

Al termine della ricerca,

il tuner restituisce i **migliori iperparametri**, che verranno utilizzati per addestrare il modello finale.

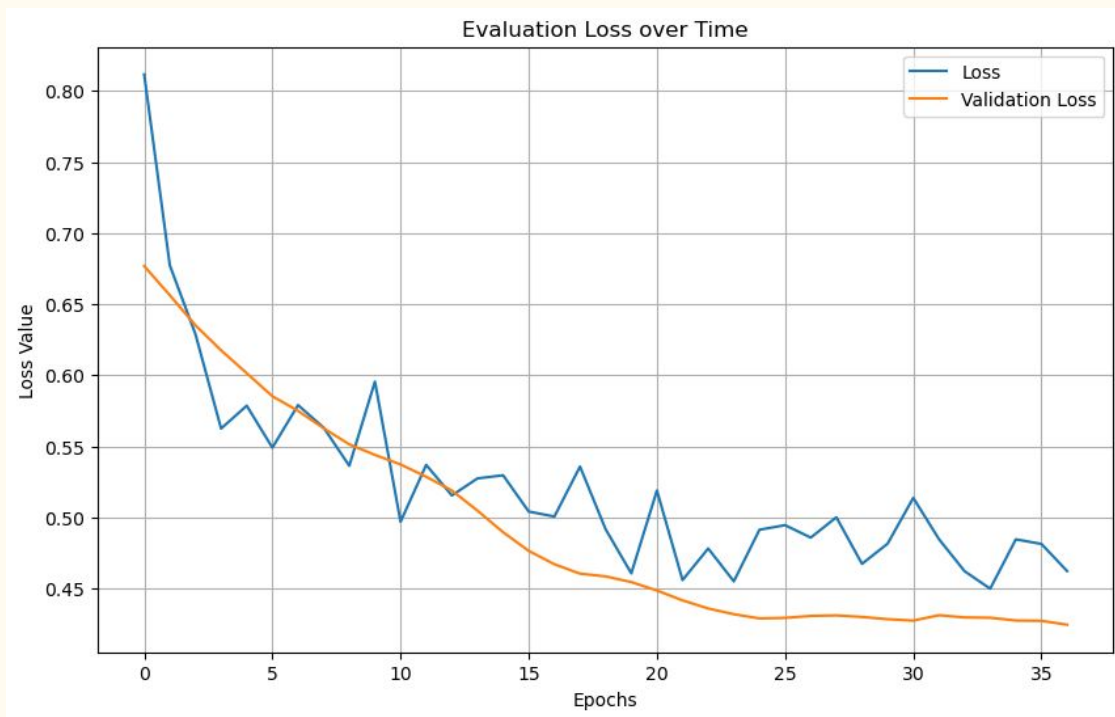
```
1 tuner = kt.RandomSearch(  
2     self.model_builder,  
3     objective=kt.Objective('val_accuracy', direction='max'),  
4     max_trials=self.max_trials, # 10  
5     executions_per_trial=2,  
6     directory='keras_tuner',  
7     project_name='titanic_tuning'  
8 )  
9  
10 tuner.search(  
11     self.X_train,  
12     self.y_train,  
13     epochs=self.epochs, # 100  
14     validation_split=self.validation_split, # 0.2  
15     callbacks=[tf.keras.callbacks.EarlyStopping(patience=10)]  
16 )
```

# Accuracy e Precision



Sia **Accuracy** che **Precision** migliorano progressivamente durante le epoche, indicando che il modello **apprende** i dati in modo efficace.

# — Loss - BinaryCrossentropy



La **BinaryCrossentropy** quantifica l'errore tra probabilità predette (un valore che varia tra 0 e 1) e valori reali (0 o 1), penalizzando le predizioni **sicure ma sbagliate**.

# Risultati

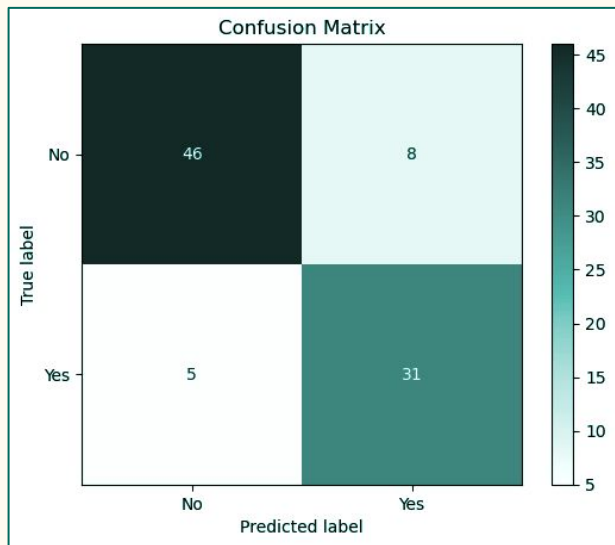
---

## — Metriche di test

---

<b>Accuracy</b> <b>84%</b>	<b>Precision</b> <b>87%</b>	<b>F1-Score</b> <b>57%</b>	<b>Loss</b> <b>38%</b>
-------------------------------	--------------------------------	-------------------------------	---------------------------

# Matrice di confusione



- **True Negative:** Predetti non sopravvissuti, realmente non sopravvissuti.
- **False Negative:** Predetti non sopravvissuti, in realtà sopravvissuti.
- **True Positive:** Predetti sopravvissuti, realmente sopravvissuti.
- **False Positive:** Predetti sopravvissuti, in realtà non sopravvissuti.