# Titanic - Machine Learning from Disaster

#### Michael Berger, Son Levi

## Overview

The data has been split into two groups:

* training set (train.csv)
* test set (test.csv)

The training set should be used to build your machine learning models. For the training set, we provide the outcome (also known as the “ground truth”) for each passenger. Your model will be based on “features” like passengers’ gender and class. You can also use feature engineering to create new features.

The test set should be used to see how well your model performs on unseen data. For the test set, we do not provide the ground truth for each passenger. It is your job to predict these outcomes. For each passenger in the test set, use the model you trained to predict whether or not they survived the sinking of the Titanic.

We also include gender\_submission.csv, a set of predictions that assume all and only female passengers survive, as an example of what a submission file should look like.

**Data Dictionary**

| Variable | Definition | Key |
| --- | --- | --- |
| survival | Survival | 0 = No, 1 = Yes |
| pclass | Ticket class | 1 = 1st, 2 = 2nd, 3 = 3rd |
| sex | Sex |  |
| Age | Age in years |  |
| sibsp | # of siblings / spouses aboard the Titanic |  |
| parch | # of parents / children aboard the Titanic |  |
| ticket | Ticket number |  |
| fare | Passenger fare |  |
| cabin | Cabin number |  |
| embarked | Port of Embarkation | C = Cherbourg, Q = Queenstown, S = Southampton |

**Variable Notes**

**pclass**: A proxy for socio-economic status (SES)
1st = Upper
2nd = Middle
3rd = Lower

**age**: Age is fractional if less than 1. If the age is estimated, is it in the form of xx.5

**sibsp**: The dataset defines family relations in this way...
Sibling = brother, sister, stepbrother, stepsister
Spouse = husband, wife (mistresses and fiancés were ignored)

**parch**: The dataset defines family relations in this way...
Parent = mother, father
Child = daughter, son, stepdaughter, stepson
Some children travelled only with a nanny, therefore parch=0 for them.

**Note**

**This submission was made with MyST**  
MyST (Markedly Structured Text) is designed to create publication-quality documents
written entirely in Markdown. The markup and publishing build system is fantastic,
MyST seamlessly exports to any PDF template, while collecting metadata to make your
writing process as easy as possible.

**We split the work so Michael will be mainly responsible for the data cleansing and Preprocessing and Son will be mainly responsible for the model training and Evaluating. We both show our individual Data Exploration and merge if necessary.**

## Libraries

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

import re

import seaborn as sns

import plotly.express as px

import plotly.subplots as subplots

from plotly.subplots import make\_subplots

import plotly.io as pio

import plotly.graph\_objects as go

# sklearn imports

from sklearn import metrics

from sklearn import pipeline

from sklearn import linear\_model

from sklearn import preprocessing

from sklearn import model\_selection

from sklearn.model\_selection import ( train\_test\_split, cross\_val\_predict,

GridSearchCV, cross\_val\_score)

from sklearn.linear\_model import Lasso, Ridge, LogisticRegression

from sklearn.svm import SVC

from sklearn.ensemble import (GradientBoostingClassifier, RandomForestClassifier,

RandomForestRegressor, AdaBoostRegressor,

GradientBoostingRegressor

from sklearn.metrics import (accuracy\_score, confusion\_matrix, precision\_score,

recall\_score, f1\_score)

from sklearn.tree import DecisionTreeClassifier

import pandasql as ps

import os

from datetime import datetime

import json

import scipy.stats as st

Cell In[161], line 23

from sklearn.ensemble import (GradientBoostingClassifier, RandomForestClassifier,

^

SyntaxError: '(' was never closed

## Reading Train Data

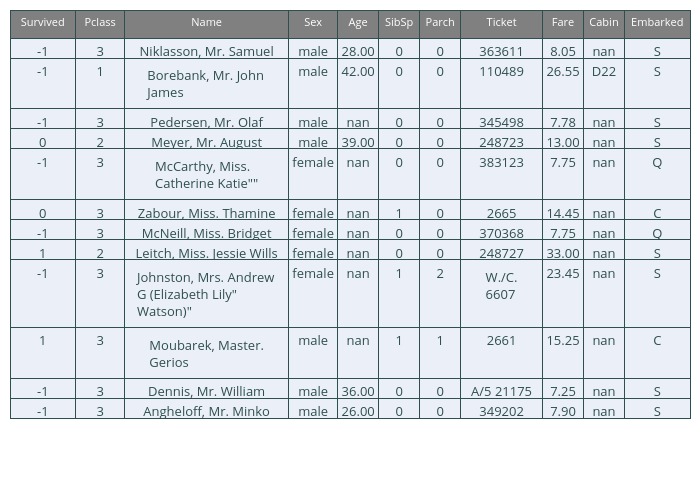
passenger\_df\_train = pd.read\_csv(pref+"train.csv", index\_col="PassengerId")

passenger\_df\_test = pd.read\_csv(pref+"test.csv", index\_col="PassengerId")

passenger\_df\_test["Survived"] = -1

passenger\_df = pd.concat([passenger\_df\_train, passenger\_df\_test])

passenger\_df.vu(12)



**Which features are categorical?**

These values classify the samples into sets of similar samples. Within categorical features are the values nominal, ordinal, ratio, or interval based? Among other things this helps us select the appropriate plots for visualization.

* Categorical: Survived, Sex, and Embarked. Ordinal: Pclass.

**Which features are numerical?**

Which features are numerical? These values change from sample to sample. Within numerical features are the values discrete, continuous, or timeseries based? Among other things this helps us select the appropriate plots for visualization.

* Continous: Age, Fare. Discrete: SibSp, Parch.

**Which features may contain errors or typos?**

* Name feature may contain errors or typos as there are several ways used to describe a name including titles, round brackets, and quotes used for alternative or short names.

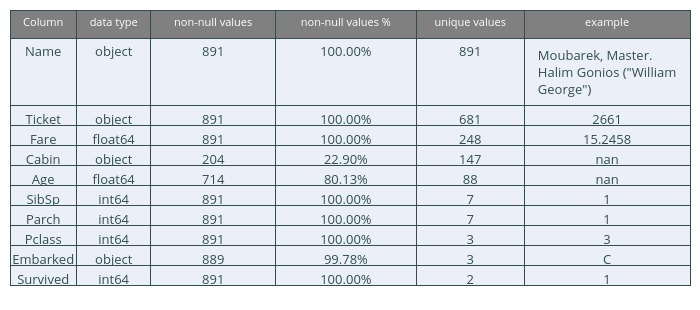
## Explatory Data Analysis (EDA) and Data Visualization

### Part 1 - Data Visualization

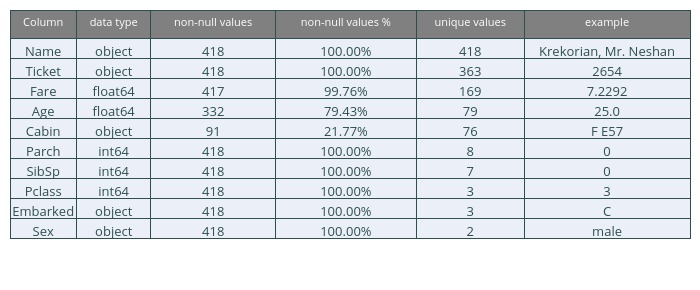
#### Describe Data

**Check if there any null values**

summary(passenger\_df\_train).vu(r=0)



summary(passenger\_df\_test).vu(r=0)



passenger\_df\_train.describe()

**Which features contain blank, null or empty values?**

These will require correcting.

* Cabin > Age > Embarked features contain a number of null values in that order for the training dataset.
* Cabin > Age are incomplete in case of test dataset.

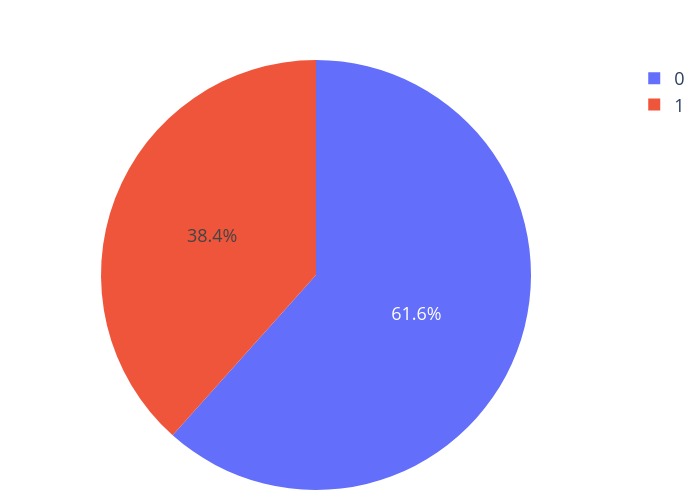
**What are the data types for various features?**

Helping us during converting goal.

* Seven features are integer or floats. Six in case of test dataset.
* Five features are strings (object).

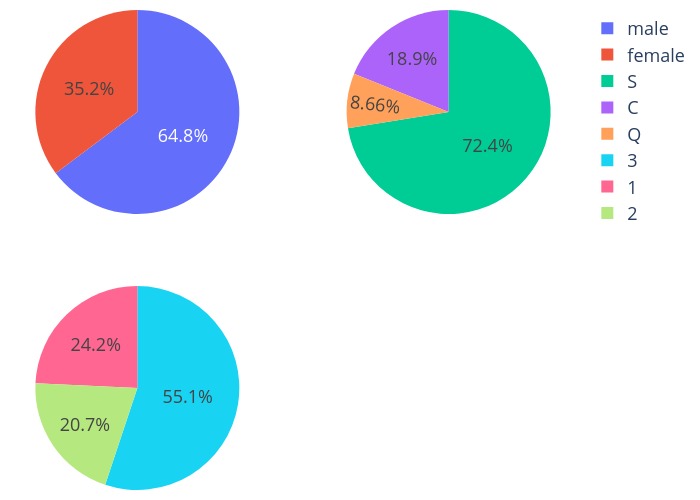
#### Amount of Survivors

create\_pie\_chart\_of\_count(passenger\_df\_train, 'Survived')



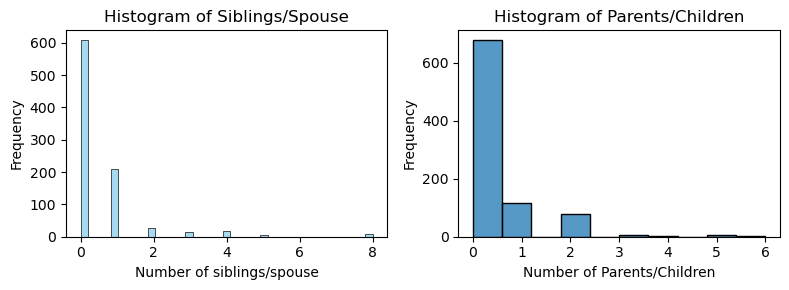
#### Pie Charts for Embark, Sex and Pclass

create\_pie\_chart\_subplot\_of\_count(passenger\_df\_train, ['Sex', 'Embarked', 'Pclass'])



#### Histograms for Siblings/Spouse and Parents/Children

fig



### Observations in a Nutshell for all features separately:

**passengers**:

1. There were 891 passengers in the data, with 681 unique tickets and 148 Cabins
2. Most passengers did not stay at a Cabin.

**sex**:

1. 65% of passengers are male and the rest female

**survived**:

1. 38% of passengers survived the disaster

**embarked**:

1. The majority of the passengers embarked from Southampton (makes sense because assumed higher population)
2. small amount of passengers have an unknown embarkment

**pclass**:

1. Most of the passengers are 3rd Class

**age**:

1. There are 177 passengers that have an unknown age
2. The average age is 23 and most of the passengers were in their 20’s

**sibsp**:

1. 600+ passengers were without siblings/spouses
2. 1 Outlier of 8 siblings/spouse (probably the family members as each index)

**parch**:

1. The big majority of the passengers are without parents/children
2. No big outlier (max=6)
3. Mainly between 0-2

### Assumptions based on the data

***Correlating***

We want to know how well does each feature correlate with Survival.

***Completing***

1. We may want to complete Age feature as it is definitely correlated to survival.
2. We may want to complete the Embarked feature as it may also correlate with survival or another important feature.

***Filtering***

1. Ticket feature may be dropped from our analysis as it contains high ratio of duplicates (22%) and there may not be a correlation between Ticket and survival.
2. Cabin feature may be dropped as it is highly incomplete or contains many null values both in training and test dataset.
3. PassengerId may be dropped from training dataset as it does not contribute to survival.
4. Name feature is relatively non-standard, may not contribute directly to survival, so maybe dropped.

***Engineering***

1. We may want to create a new feature called Family based on Parch and SibSp to get total count of family members on board.
2. We may want to engineer the Name feature to extract Title as a new feature.
3. We may want to create new feature for Age bands. This turns a continous numerical feature into an ordinal categorical feature.
4. We may also want to create a Fare range feature if it helps our analysis.
5. We may want to divide the Cabin into Letter and number of cabin instead of filtering the feature completely to get further information.

***Classifying***

We may also add to our assumptions based on the problem description noted earlier.

1. Women (Sex=female) were more likely to have survived.
2. Children (Age<?) were more likely to have survived.
3. The upper-class passengers (Pclass=1) were more likely to have survived.

### Data Exploration

To confirm some of our observations and assumptions, we can quickly analyze our feature correlations by pivoting features against each other. We can only do so at this stage for features which don’t have empty values. It also makes sense doing so only for feature types which are categorical (Sex), ordinal (Pclass) or discrete (SibSp, Parch).

* **Pclass** We observe significant correlation (>0.5) among Pclass=1 and Survived (classifying #3). We decide to include this feature in our model.
* **Sex** We confirm the observation during problem definition that Sex=female had very high survival rate at 74% (classifying #1).
* **SibSp and Parch** These features have zero correlation for certain values. It may be best to derive a feature or a set of features from these individual features (engineering #1).

#### Comparing non-null features to survived

#### Based on the Age vs Survived Histograms:

***Observations***

* Infants (Age <=4) had high survival rate.
* Large number of 15-25 year olds did not survive.
* Most passengers are in 15-35 age range.

***Decisions***

* We should consider Age (classifying #2) in our model training.
* Complete the Age feature for null values (completing #1).
* We should band age groups (engineering #3).

#### Based on the Pclass vs Survived Histograms:

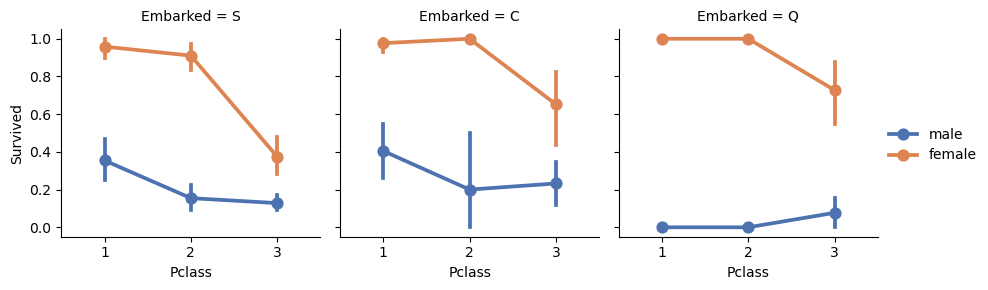
***Observations***

* Pclass=3 had most passengers, however most did not survive. Confirms our classifying assumption #2.
* Oldest passengers (Age = 80) survived.
* Infant passengers in Pclass=2 and Pclass=3 mostly survived. Further qualifies our classifying assumption #2.
* Most passengers in Pclass=1 survived. Confirms our classifying assumption #3.
* Pclass varies in terms of Age distribution of passengers.

***Decisions***

* Consider Pclass for model training.

grid.fig



#### Based on the Pclass vs Survived vs Sex based on Embarked pointplots:

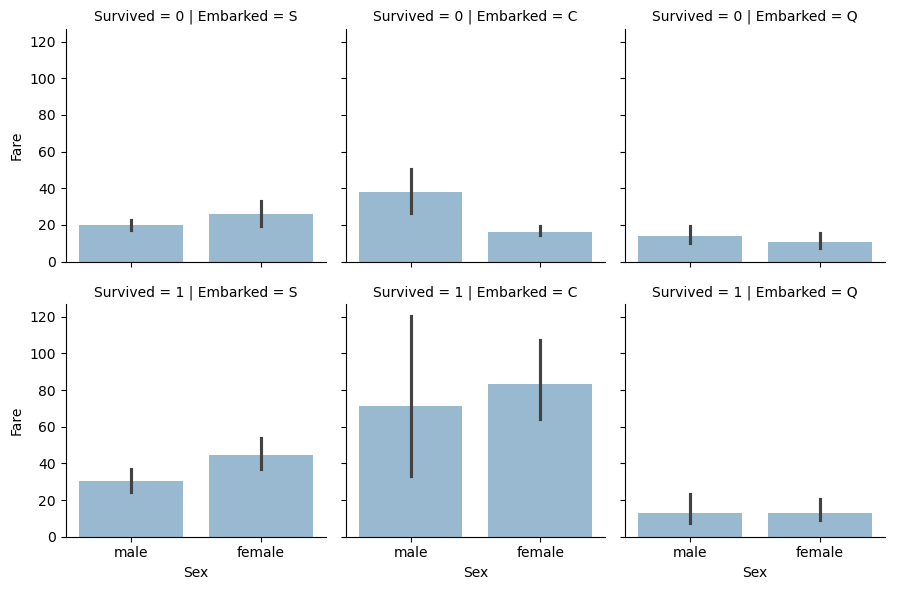
***Observations***

* Female passengers had much better survival rate than males. Confirms classifying (#1).
* Exception in Embarked=C where males had higher survival rate. This could be a correlation between Pclass and Embarked and in turn Pclass and Survived, not necessarily direct correlation between Embarked and Survived.
* Males had better survival rate in Pclass=3 when compared with Pclass=2 for C and Q ports. Completing (#2).
* Ports of embarkation have varying survival rates for Pclass=3 and among male passengers. Correlating (#1).

***Decisions***

* Add Sex feature to model training.
* Complete and add Embarked feature to model training.

grid.fig



#### Based on the Sex vs Fare vs Embarked vs Survived Barplots:

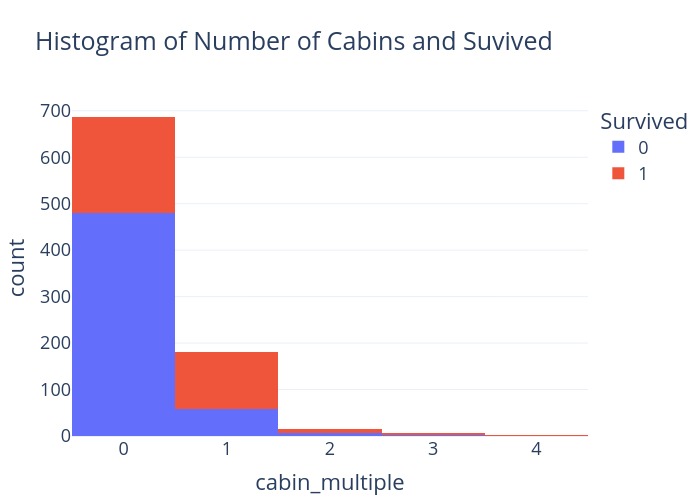
***Observations***

* Higher fare paying passengers had better survival. Confirms our assumption for creating (#4) fare ranges.
* Port of embarkation correlates with survival rates. Confirms correlating (#1) and completing (#2).

***Decisions***

* Consider banding Fare feature.

px.histogram(data\_frame= cabin\_divide, x="cabin\_multiple", color="Survived",title='Histogram of Number of Cabins and Suvived')

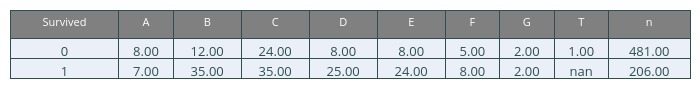


Create categories based on the cabin letter (n stands for null). In this case we will treat null values like it’s own category

pd.pivot\_table(cabin\_divide,index='Survived',

columns='cabin\_deck', values = 'Name',

aggfunc='count').vu(r=0, i=1)



#### Based on the Cabins Pivot Tables:

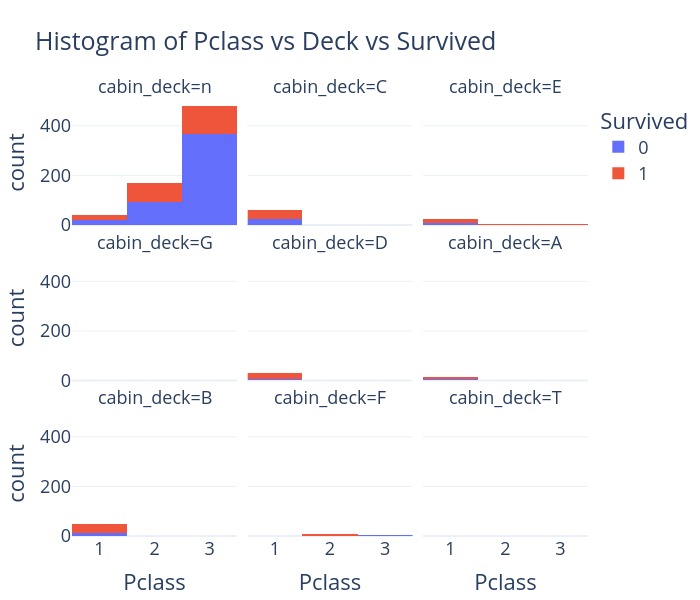
***Observations***

* Passengers with at least a Cabin listed to there ticket have a higher chance of surviving. Confirms engineering (#5)
* Cabin titles B,C,D,E and F have a higher chance of survival. Confirms engineering (#5) and debunks Filtering (#2)

***Decisions***

* Consider Seperating the cabin feature into only cabin letters.
* Consider creating a number of Cabins feature.

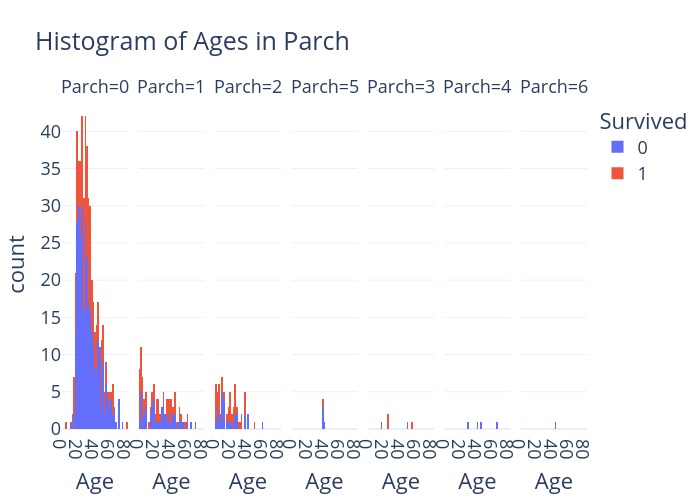
fig



**Ages vs ParCh**

px.histogram(data\_frame= passenger\_df\_train, facet\_col="Parch",

x="Age", color="Survived",title='Histogram of Ages in Parch')

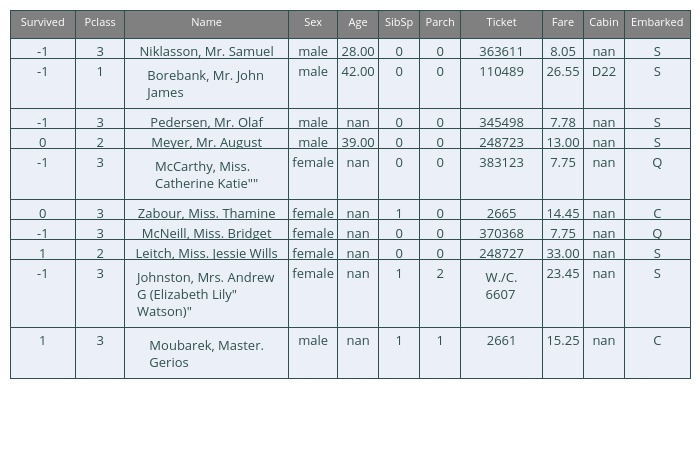


### Exploration with no regard to Survival

For the purpose of this exploration and feature engineering we will unite training and testing data. The advantage of this is that we can perform same transformations on both datasets at the same time.
Since test set has all NaNs in Survived, we will mark it with “-1”. This will later allow for splitting them back easily.  
During this exploration we will not touch “Survived” feature.

passenger\_df.loc[passenger\_df.Survived.isna(),"Survived"] = -1

passenger\_df[original].vu(10)



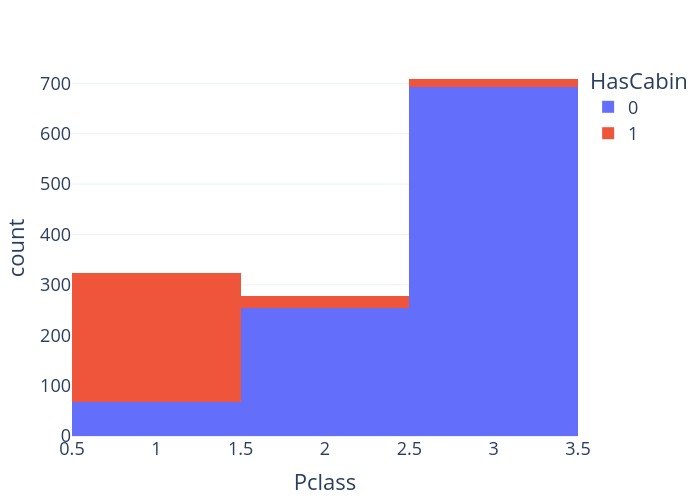
#### Cabin

passenger\_df["HasCabin"] = ~passenger\_df.Cabin.isnull() \*1

print("Have Cabin:" , int( passenger\_df.HasCabin.sum() / passenger\_df.shape[0] \*100), "%" )

px.histogram(passenger\_df, x = "Pclass", color="HasCabin")

Have Cabin: 22 %



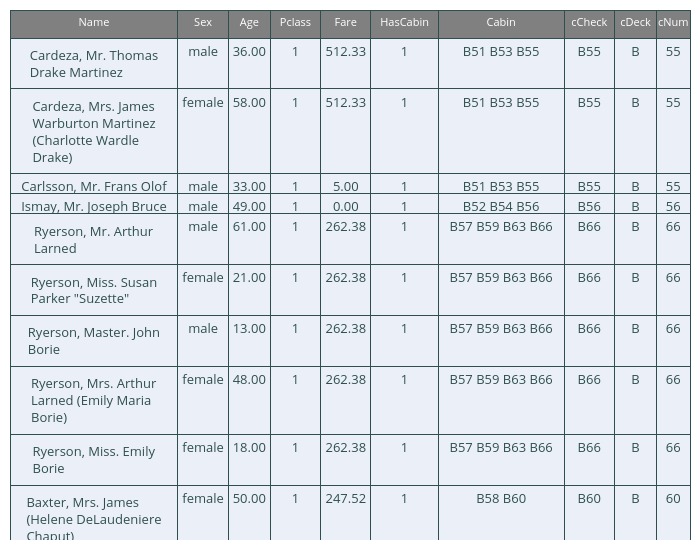
We observe that the cabin column is the emptiest. Only 22% of
the passengers have it, with the majority being in first class.
Upon examining the Titanic deck plans, we see that all the living space was represented by cabins. We can conclude that the emptiness in data is not because those passengers did not have a cabin, but rather because this information was just not written down. In the chaos of a sinking ship, such inaccuracy is perfectly understandable.

This sparsity of data usually disqualifies the column from a statistical model. However, some researchers (reference here)[] claim that including the column (while imputing the missing data) has allowed them to significantly increase the score of the model. Their approach to imputing the missing data was straightforward: replace it with mean value of cabin.  
While this approach is sound, our opinion is that it can be further improved.

For this purpose we need to split the Cabin column into it’s components:

* Letter (A through G + T) representing ship’s deck
* Cabin number
  A fairly simple regular expression can be used for this.

check.loc[check["HasCabin"]==1, columns\_of\_interest ].vu(r=0)



Some cabins seem to not have splitted correctly. Mostly, those having several cabins listed per person. However, upon examining these cabins we can conclude that these are families occupying several (1st class) cabins close to each other. Our splitting method takes one of the cabins from each such group, essentially “squeezing” the whole family in one cabin. Hence, we deem our splitting method as good enough for this case.

#### Ticket/placement

Explore what we can find from ticket/ placement data.  
Columns involved:  
Fare, Cabin, Pclass, Embarked, Ticket

passenger\_df.drop("tPref tNum".split(" "),axis=1, inplace=True, errors='ignore')

rx = r'(?P<tPref>[A-Za-z/.\d]+\s(?:[A-Za-z.\d]+\s)?)?(?P<tNum>\d+)$'

tspl = passenger\_df.Ticket.str.extract(rx)

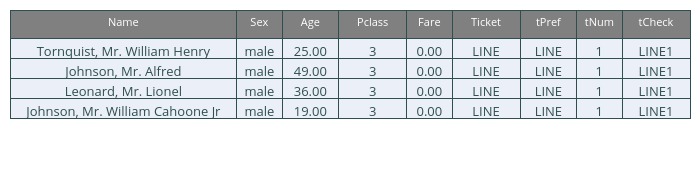
passenger\_df = passenger\_df.join(tspl)

Validate: all tickets got split correctly?

passenger\_df["tCheck"] = (passenger\_df['tPref']).fillna('') + "" +passenger\_df['tNum'].astype(str)

columns\_of\_interest = "Name Sex Age Pclass Fare Ticket tPref tNum tCheck".split(" ")

passenger\_df.loc[passenger\_df['Ticket'] != passenger\_df["tCheck"], columns\_of\_interest].vu(4)



Only 4 tickets have splitted incorrectly. But they have no ticket number in the first place, so it does not matter.

##### Analyzing ticket prefixes

It seemed that ticket prefixes could contain additional information encoded in them. Our theory was that somehow it could be indicative of placement on the ship.

q = """

Select tPref, count(Ticket) as tickets

from passenger\_df

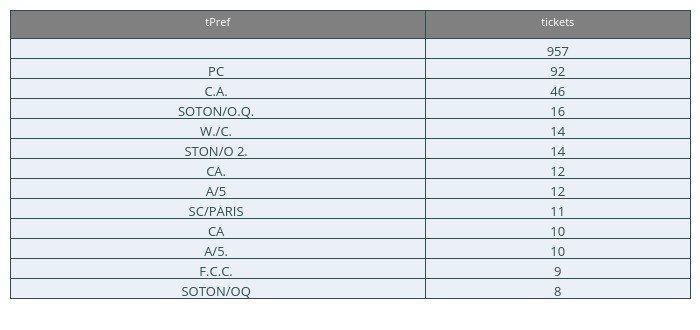
group by tPref

order by tickets desc

limit 13

"""

ps.sqldf(q).vu(13,r=False)



Having extracted and summarized the prefixes, we observed that a lot of these prefixes were identical among some tickets.  
Also, by eliminating special characters, some different prefixes
could be merged into one, e.g  
(C.A. , CA. , CA) => CA

By performing this transform, we’ve reduced the amount of unique prefixes from 52 to 30

q = """

select Pclass, tPrefTr, count(\*) as count

from passenger\_df

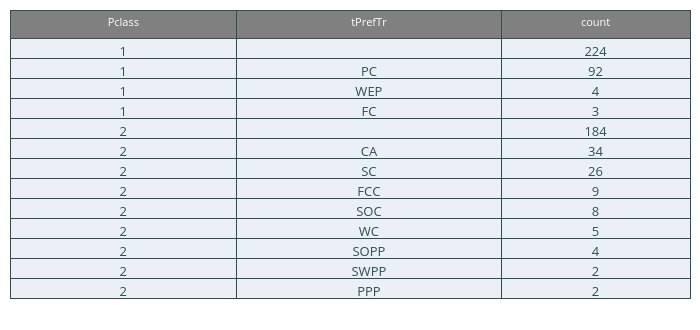
group by Pclass, tPrefTr

order by Pclass, count desc

limit 13

"""

ps.sqldf(q).vu(13, r=False)



According to forums dedicated to Titanic, PC, FC mean Private Class and First CLass.  
Unfortunately, we could not guess what most of the rest mean, and no clues were found on the Internet.

##### Hypothesis: ticket number has meaning

During exploration We had another hipothesis, that ticket number could somehow contain an encoding to placement of the passenger on the ship.
To explore this hypothesis, we created various plots of features that might be involved, such as:

* Ticket number
* Ticket prefix
* Fare
* Class
* Deck
* Cabin number

Ticket numbers seem to be concentrated into several groups. Encyclopedia Titanica Forum suggest that these groups come from individual stores from which the tickets were purchased. The order inside group is probably the order in which the tickets were purchased, so probably not very relevant to current research.

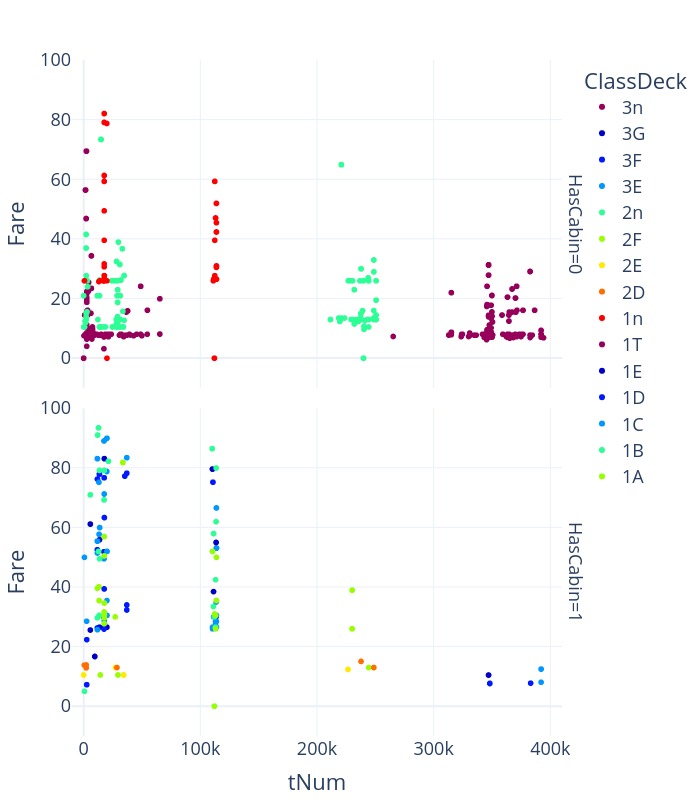
Zooming down into distributions of passengers with cabins vs. those who had no cabin registered, we were wondering if deck and/or cabin number could somehow be “encoded” in Fare and ticket number. To check this out we’ve built the following plot:

fig = px.scatter(passenger\_df, x = "tNum", y = "Fare", color="ClassDeck", facet\_row="HasCabin", \

color\_discrete\_sequence= px.colors.sequential.Rainbow, category\_orders=co ,

height=800, range\_x= [-10000, 4.1e5], range\_y=[-10,100])

fig



At first glance, the fares and ticket numbers are clearly distributed into a number of distinct clusters, which appear to have some relation with composite variable ClassDeck.
However the available data for classes 2 and 3 turned out to be too sparse to provide a reliable basis for an estimator for missing data.  
Hence, we decided to abandon the Cabin variable in the end.

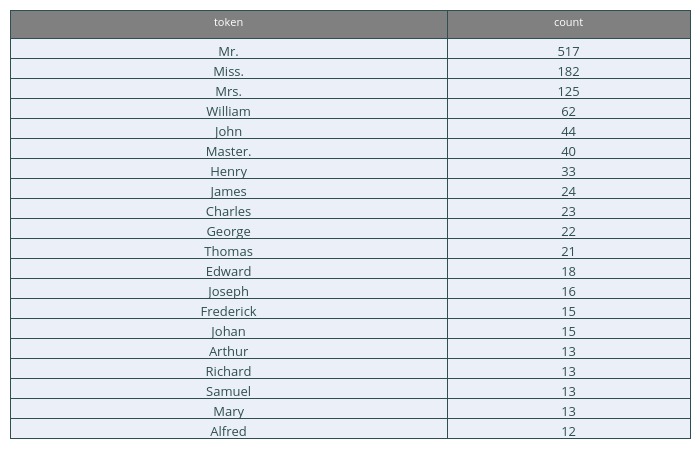
#### Age

We observe that about 20% of passengers have no age registered. We think that it could be estimated from other features, such as Title, amount of children/siblings, etc

#### Name

There appears to be a lot of information contained in passengers names. Let us check, what can be extracted from it.  
First, let’s see what tokens beside names we can expect to see in this column

tokens.vu(20,0)

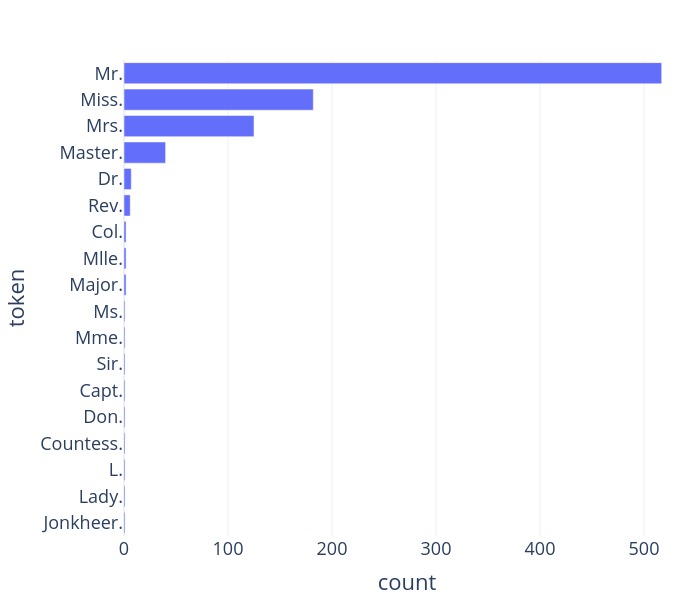


24% of tokens in in the whole Name column are the following tokens

print(perc)

fig

24% of tokens in in the whole Name column are represented by the following tokens:



We see that people’s titles have high token frequencies, suggesting that lots of people have them. Moreover, the Name column appears to have a very consistent structure:  
Last\_name, Title. First\_Name (Second\_Name)  
This allows to split the Name column into its components using a relatively simple regular expression :) .

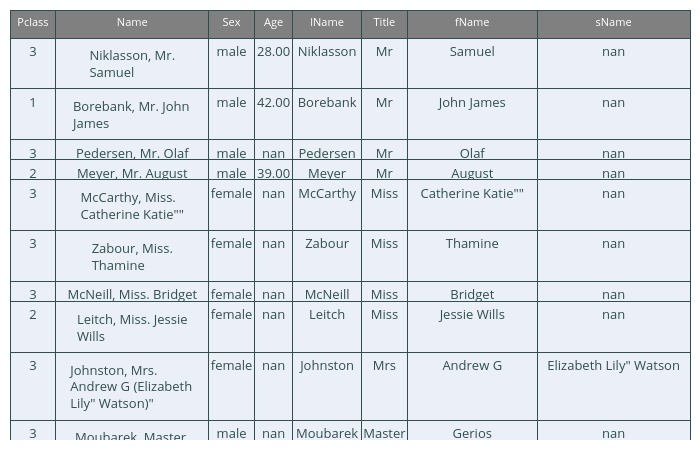
rx = r"^(?P<lName>[A-Za-z\s'-]+),\s(?P<Title>[A-Za-z\s]+)"

rx+= r"\.(?:\s(?P<fName>[A-Za-z\s\/\"]+))?(?:\s\((?P<sName>[A-Za-z\s\"'\.-]+)\).\*)?$"

nspl = passenger\_df.Name.str.extract(rx)

passenger\_df = passenger\_df.join(nspl)

passenger\_df[coi].vu(10)



##### Title

Let us further explore the title column.

q = """

select Title, count(\*) as cnt

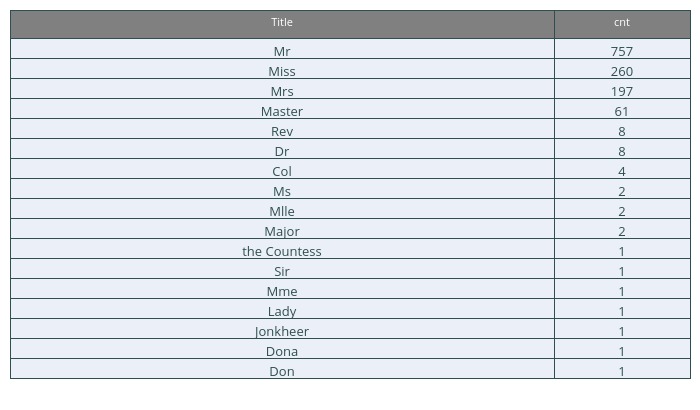
from passenger\_df

group by Title

order by cnt desc

"""

ps.sqldf(q).vu(17,0)



We have 17 titles in total, most of which are common: Mr, Mrs, Miss and Master, with the rest being rare:

Several military titles, as well as other relarted to person’s occupation. These can be joined into a single category Rare:

* Col, Major, Jonkeer, Capt.
* Rev is Reverend - a member of clergy
* Dr is Doctor  
  Some titles are the equivalents of common titles in other languages or alternative spelling:
* Ms, Mlle = Miss
* Mme = Mrs
  Several people have a noble title. But since they are few, they can be joined into Mr, Mrs category.
* the Countess, Lady, Dona = Mrs
* Don, Sir = Mr

After the replacing we have just 5 categories in title:

q = """

select Title, count(\*) as count

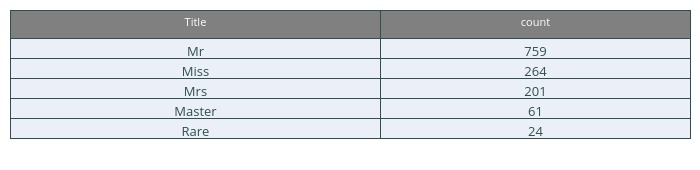
from passenger\_df

group by Title

order by count desc

"""

ps.sqldf(q).vu(5,0)



We think that this may be used to estimate Age where it’s unknown

##### Second name

Let’s explore the second name (sName column):

coi = "Fare Sex Age Pclass Title fName lName sName".split(" ")

passenger\_df.loc[~passenger\_df.sName.isna(),coi].vu(10)



We perform a similar token analysis with these as with the full name before

names = passenger\_df.sName.astype(str).apply(func=spl).values.tolist()

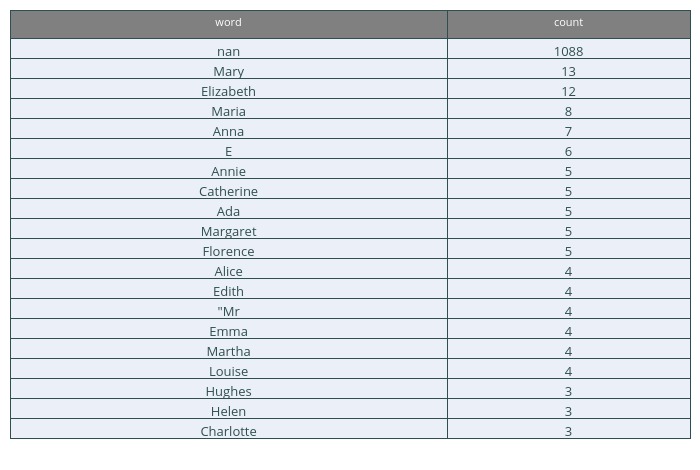
words = sum(names,[])

unique, counts = np.unique(words, return\_counts=True)

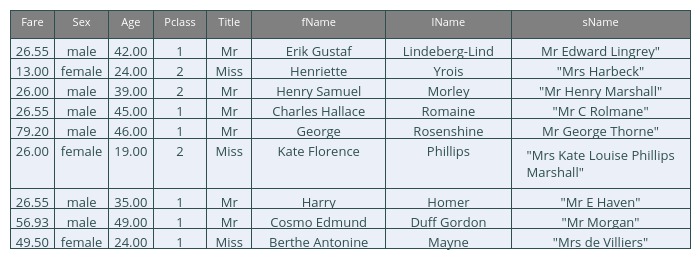
wc = pd.DataFrame({"word":unique, "count": counts})

tokens = wc.sort\_values("count",ascending=False)

tokens.vu(20,0)



passenger\_df.loc[passenger\_df.sName.astype(str).str.contains("Mr"), coi].vu(9)



Seems like majority of these contain full/maiden names of women travelling with ticket under their husbands’ names.  
Maybe this property could be used to separate SibSp column into Siblings and Spouses and ultimately, for Age estimation.

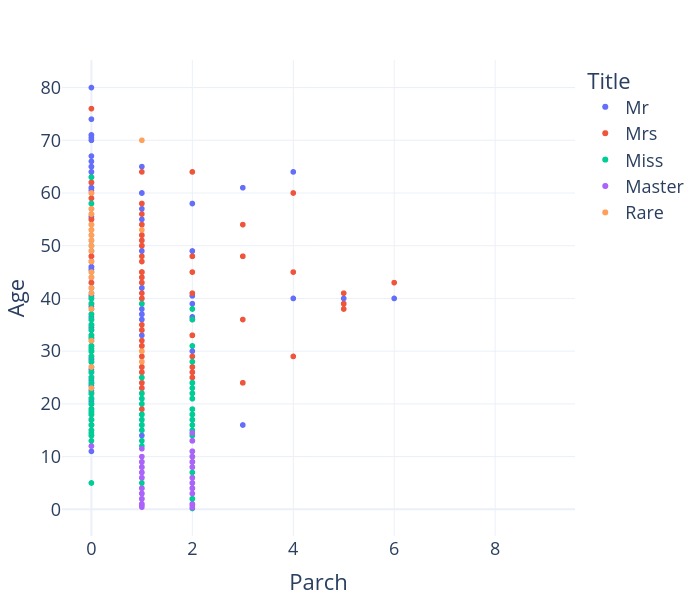
#### Family composition data

Upon acqiring these new columns, we can plot them on a graph and discover an interesting pattern:

* The title Master, meaning underage male, is concentrated in lower age range,
* The title Miss, meaning underage or unmarried female, is concentrated higher
* The Mrs and Mr are grownups.

px.scatter(passenger\_df, y = "Age", x = "Parch", color="Title", hover\_data=["Fare", "Name"], \

category\_orders=co, height= 600 )



We think that the titles, along with the amount of parents and siblings, can serve as a good indicator for person’s age.

### Part 2 - Data Engineering + Encoding Categorical Values

### Data Imputation

As discussed in Exploration section, about 20% of passengers have no Are registered. We would like to impute the null values of Age with an estimate based on other variables.

**Prepare dataset for training and imputation**

Convert categorical variables into dummy variables using one-hot encoding

Cx = ["Fare", "Sex", "SibSp", "Parch", "Pclass", "Title"]

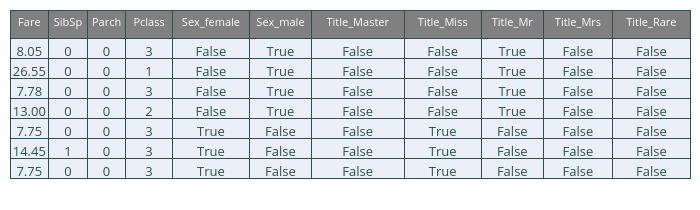
Cy = "Age"

categorical\_columns = ["Sex", "Title"]

X = pd.get\_dummies(passenger\_df[Cx], columns=categorical\_columns)

y = passenger\_df[Cy]

X.vu(7)

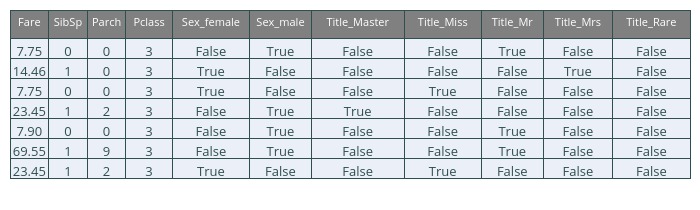


Select rows with missing values for ‘Age’. Those will be imputed

Ximp = X[y.isna()]

yimp = y[y.isna()]

Ximp.vu(7)

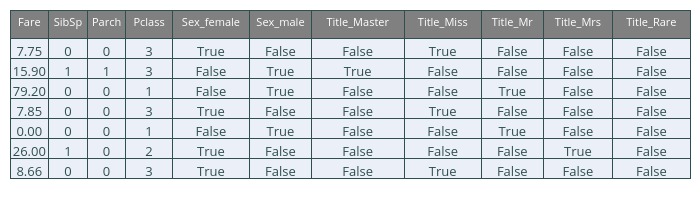


Select rows with existing values for ‘Age’ in target. Those will be used to learn the pattern for imputation

X = X[~y.isna()]

y = y[~y.isna()]

X.vu(7)



And split the Dataset into Train and Validation

#### Cross-validate ensemble models

We need to train a model that will predict Age of a person best.  
At this stage we will concentrate on ensemble family of models
This convenience function will be used for training, evaluation and summarization of various ML models.

**Random Forest**

rfr = RandomForestRegressor()

param\_grid ={'max\_depth': st.randint(6, 20),

'n\_estimators': st.randint(10, 500),

'max\_features': np.arange(5, 12),

'max\_leaf\_nodes': st.randint(6, 30)}

grid = model\_selection.RandomizedSearchCV(rfr,

param\_grid, cv=10,

verbose=1, n\_iter=iterations, n\_jobs=16, )

Run\_and\_Report(grid, X, y)

Fitting 10 folds for each of 600 candidates, totalling 6000 fits

Elapsed Time: 00:02:19

====================

Best Score: 0.441

Best Parameters: {'max\_depth': 10, 'max\_features': 5, 'max\_leaf\_nodes': 29, 'n\_estimators': 145}

**AdaBoost**

abr = AdaBoostRegressor()

param\_grid ={

'learning\_rate': st.randint(1, 10),

'n\_estimators': st.randint(10, 500),

}

grid = model\_selection.RandomizedSearchCV(abr,

param\_grid, cv=10,

verbose=1, n\_iter=iterations, n\_jobs=16 )

Run\_and\_Report(grid, X, y)

Fitting 10 folds for each of 600 candidates, totalling 6000 fits

Elapsed Time: 00:01:18

====================

Best Score: 0.395

Best Parameters: {'learning\_rate': 1, 'n\_estimators': 329}

**GradientBoosting**

gbr = GradientBoostingRegressor()

param\_grid ={'max\_depth': st.randint(6, 20),

'n\_estimators': st.randint(10, 500),

'max\_features': np.arange(5,12),

'max\_leaf\_nodes': st.randint(6, 30)}

grid = model\_selection.RandomizedSearchCV(gbr,

param\_grid, cv=10,

verbose=1, n\_iter=iterations, n\_jobs=16 )

Run\_and\_Report(grid, X, y)

Fitting 10 folds for each of 600 candidates, totalling 6000 fits

Elapsed Time: 00:01:49

====================

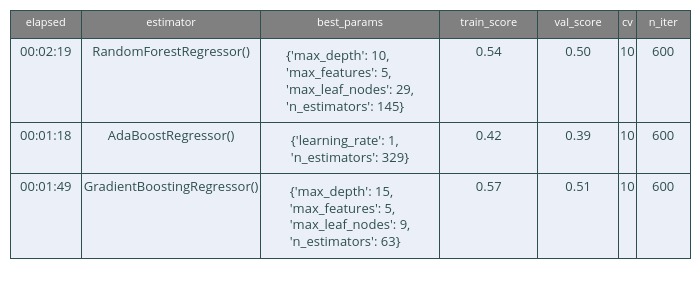
Best Score: 0.446

Best Parameters: {'max\_depth': 15, 'max\_features': 5, 'max\_leaf\_nodes': 9, 'n\_estimators': 63}

CV\_df = pd.DataFrame(CV\_Runs)

CV\_df[['elapsed', 'estimator', 'best\_params', 'train\_score',

'val\_score', 'cv', 'n\_iter']].vu(r=0)



#### Estimate missing ages

Based on the benchmarking results above, we decided to choose model 3 (GradientBoostingRegressor)

best\_params = {'max\_depth': 13, 'max\_features': 5, 'max\_leaf\_nodes': 29, 'n\_estimators': 435}

rfc = GradientBoostingRegressor(\*\*best\_params)

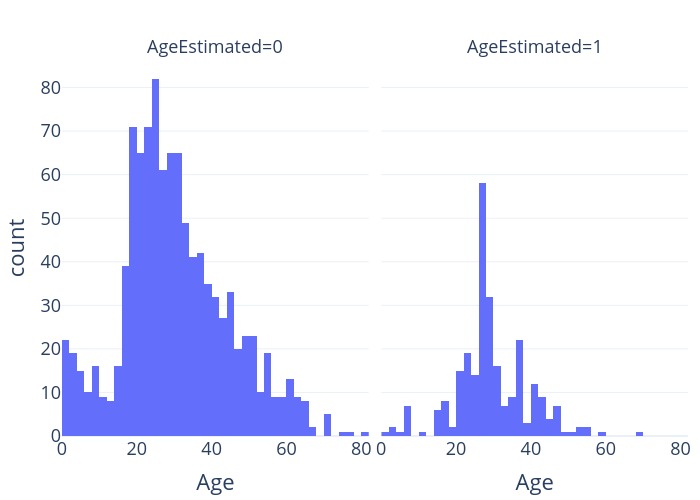
rfc.fit(X,y)

y\_hat = rfc.predict(Ximp)

y\_hat = pd.Series(rfc.predict(Ximp), index=Ximp.index)

Impute the new predicted age values into original dataset and visually compare distributions of existing and estimated ages

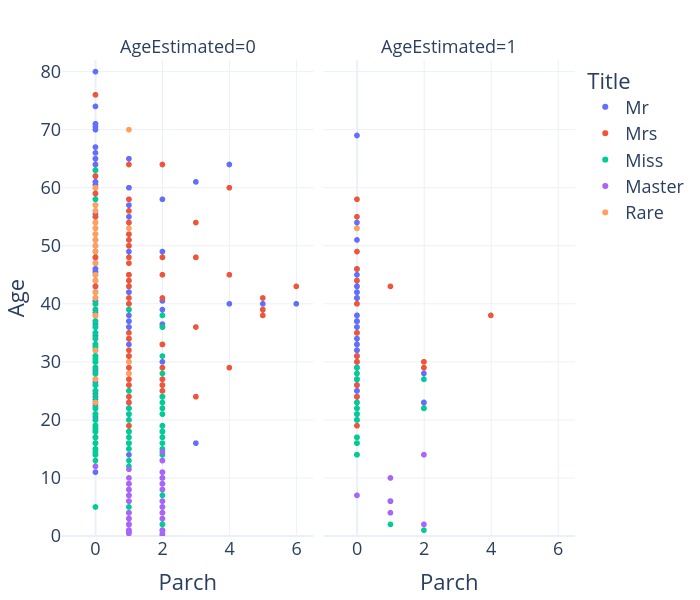
px.histogram(passenger\_df, x="Age", facet\_col = "AgeEstimated")



px.scatter(passenger\_df, y = "Age", x = "Parch", color="Title", facet\_col= "AgeEstimated",

hover\_data=["SibSp", "Fare", "Name"], range\_x=[-1,6.5], range\_y = [0,82],

category\_orders=co, height= 600)



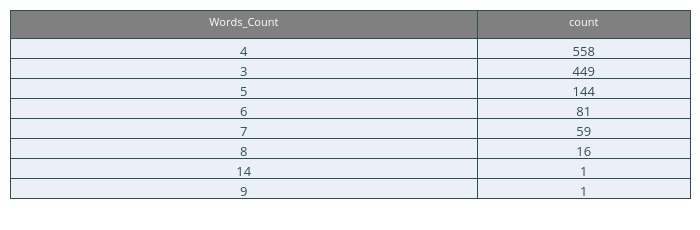
It seems that imputation went quite well.

### Construct More features

The length of the Name

passenger\_df['Words\_Count'] = passenger\_df['Name'].apply(lambda x: len(x.split()))

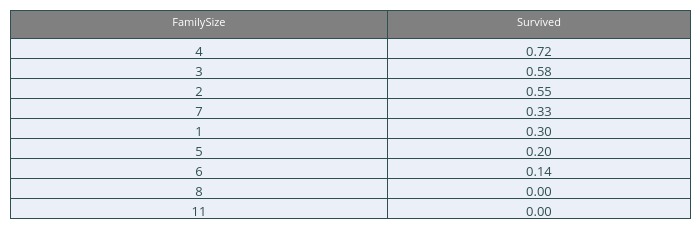
pd.DataFrame(passenger\_df.Words\_Count.value\_counts()).vu(r=0,i=1)



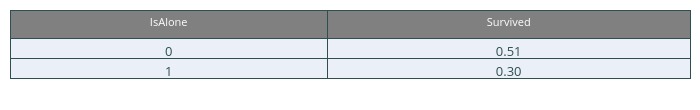
Creating new features:

* cabin\_multiple: number of cabins each passenger had.
* cabin\_deck: the ship’s deck where cabnin is located
* FamilySize as a combination of SibSp and Parch
* IsAlone from FamilySize

df1.vu(r=0)



df2.vu(r=0)

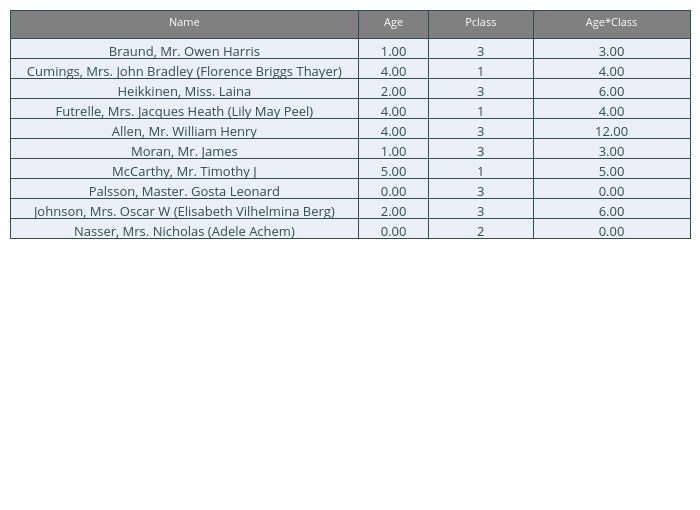


Remove all NULLS in the Fare column and Create new feature CategoricalFare

Create a New feature CategoricalAge

### Mapping Categorical and High Ordinal Features

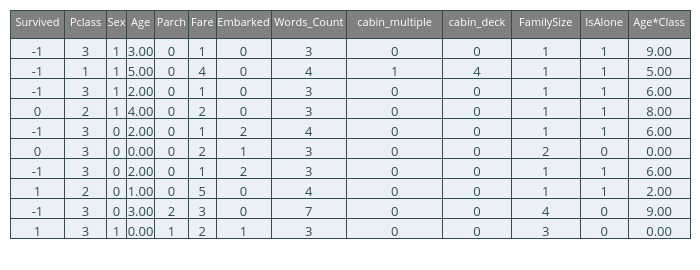
passenger\_df.loc[:, ['Name', 'Age', 'Pclass', 'Age\*Class']].vu(10,0)



### Feature Selection

In order to fit the “trained data”, we turn the indexes in the combined dataset to whoever has a null in survived (meaning they’re from the test dataset) into -1 value, therefore -1 is test and 0/1 is trained.

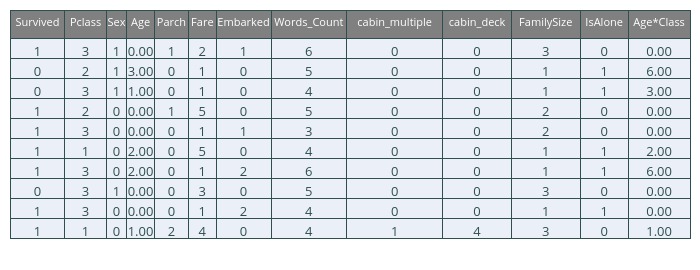
passenger\_df.vu(10)



Splitting the dataset back into training and testing

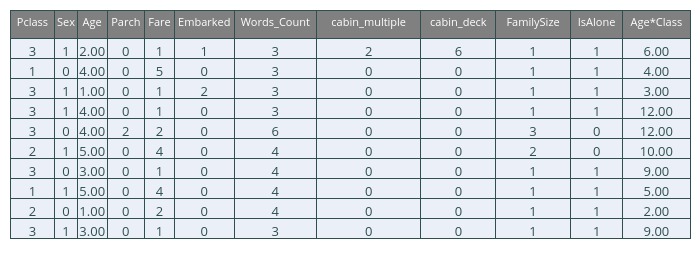
passenger\_df\_train = passenger\_df[passenger\_df.Survived != -1]

passenger\_df\_train.vu(10)

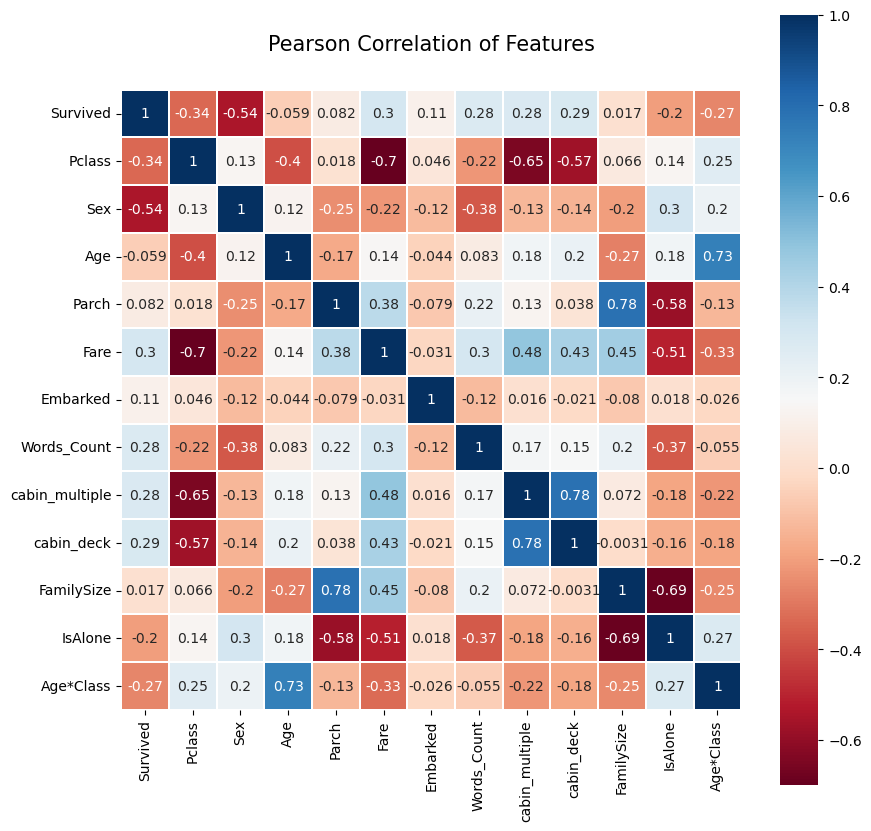


passenger\_df\_test = passenger\_df[passenger\_df.Survived == -1].drop("Survived", axis=1)

passenger\_df\_test.vu(10)



fig



### Takeaway from the Heatmap

There aren’t many features strongly correlated with one another (highest is 0.78 between Parch and FamilySize and between the two cabin features. We’ll still leave both features.) This is good from a point of view of feeding these features into your learning model because there isn’t much redundant or superfluous data in our training set and we accept that each feature carries data with some unique information.

## Model Learning

### Splitting the passenger data 80/20

We split the data into training data and validation data randomly. so that 80 percent will be for training and 20 percent for validation the performance of the model and comparing the two models.
The splitting is done randomly so that 20 percent of samples are chosen randomly so that there is no connection between them so that the testing and comparison between the models are reliable.

X = passenger\_df\_train.drop('Survived', axis=1)

y = passenger\_df\_train['Survived']

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

### Model Functions

#### Train and Evaluate models

#### Visualize Results

#### Confusion Matrix for Best Model

### Classic models

Initialize models with Hyperparameters

# Define models

models = {

'Logistic Regression': LogisticRegression(),

'Random Forest': RandomForestClassifier(),

'SVM': SVC(),

'Gradient Boosting': GradientBoostingClassifier(),

'Decision Tree': DecisionTreeClassifier()

}

# Define hyperparameter grids for each model

param\_grids = {

'Logistic Regression': {'C': [0.001, 0.01, 0.1, 1, 10, 100]},

'Random Forest': {'n\_estimators': [10, 50, 100, 200, 500],

'max\_depth': [None, 10, 20, 30, 50]},

'SVM': {'C': [0.01, 0.1, 1, 10, 100], 'gamma': [0.01, 0.1, 1, 10, 100]},

'Gradient Boosting': {'n\_estimators': [10, 50, 100, 200, 500],

'learning\_rate': [0.001, 0.01, 0.1, 1]},

'Decision Tree': {'max\_depth': [None, 10, 20, 30, 50, 100]}

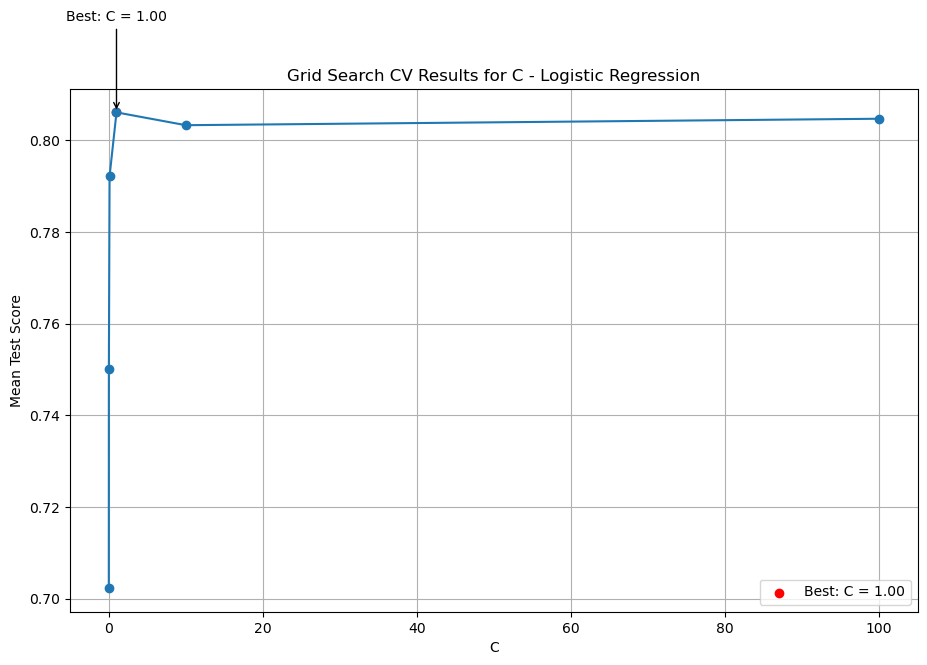
}

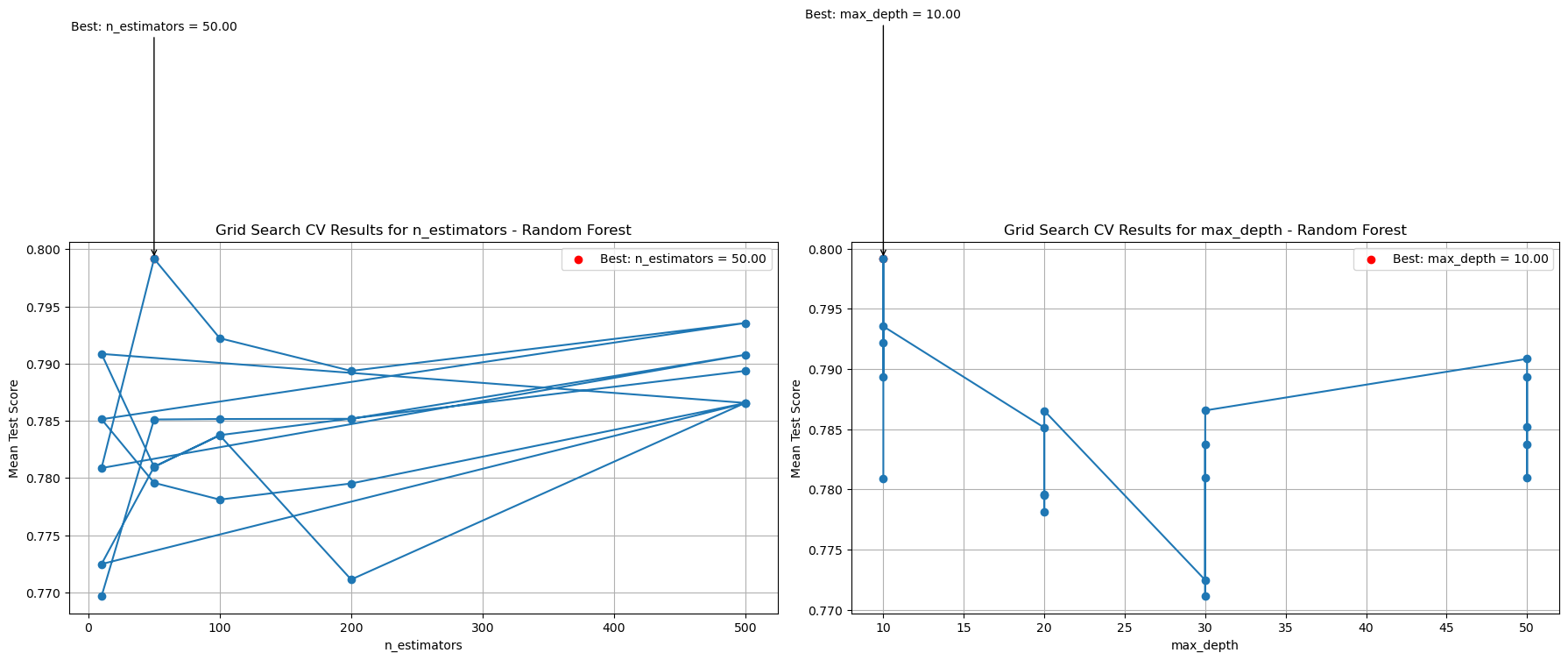
##### Training and Evaluating the models

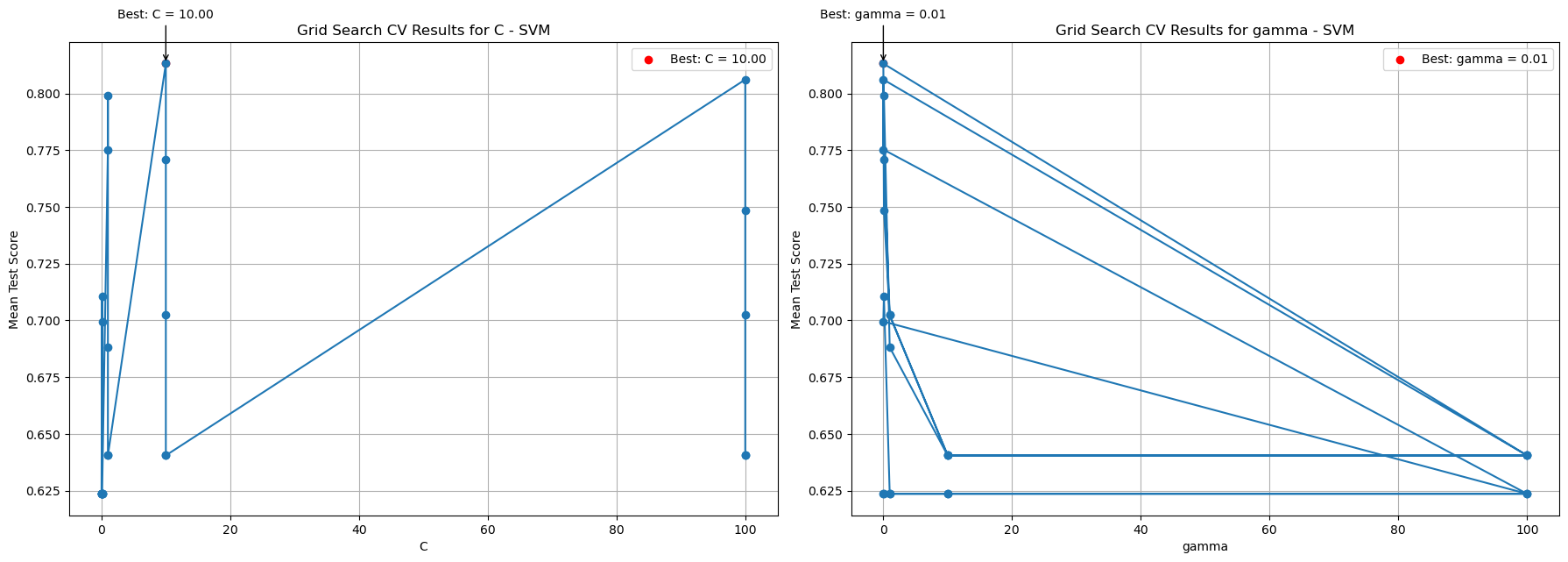
Showing the process of Cross Validation using each and every combination of parameters for each model, so we can show in the final dataframe their best parameter combination (highest model mean test score)

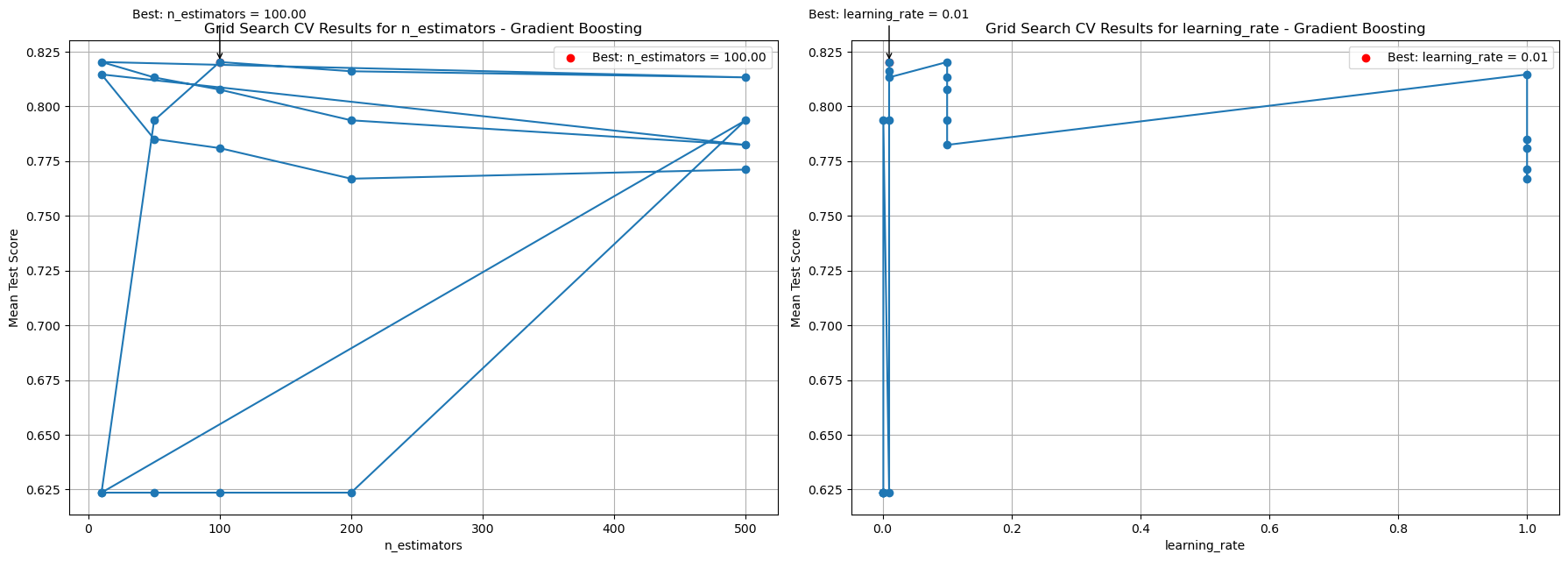
results, evaluation\_df = train\_and\_evaluate\_models(models, X\_train, y\_train, X\_val, y\_val,

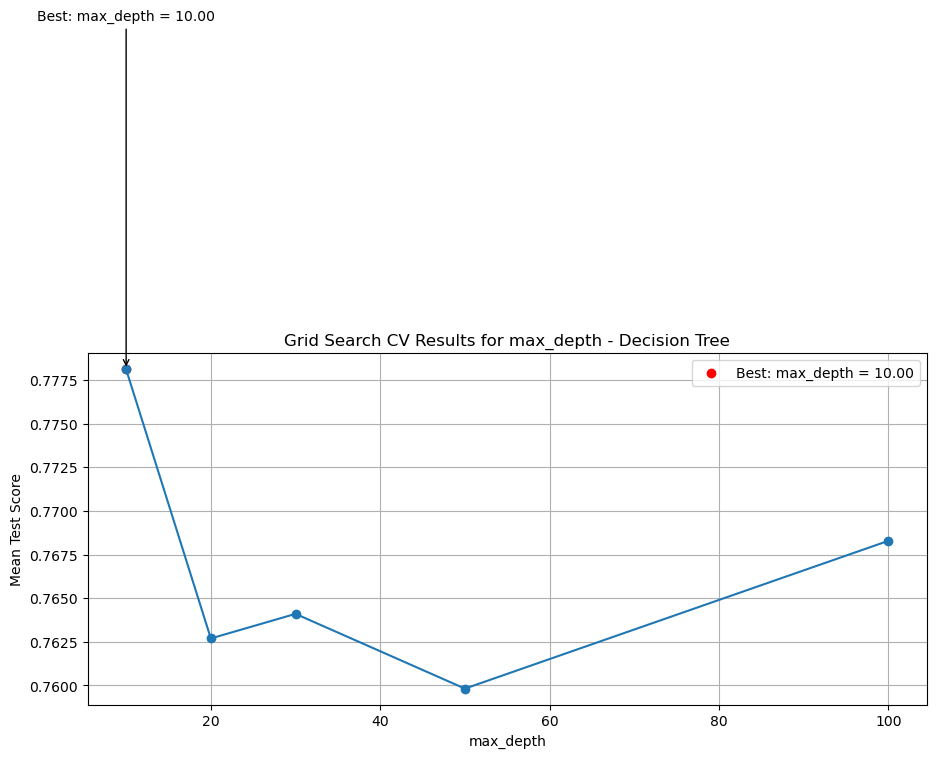
param\_grids, scoring='accuracy', cv=10)



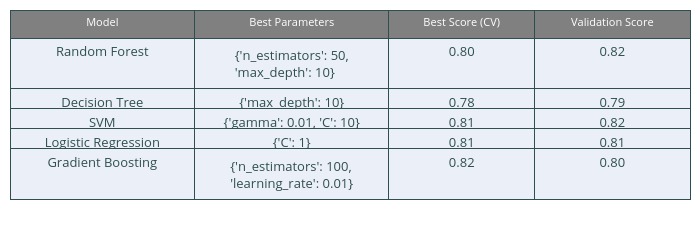








evaluation\_df.vu(5)



We can see that Random Forest gives us the best Validation score, meaning Random Forest works best with new Data.

print(best)

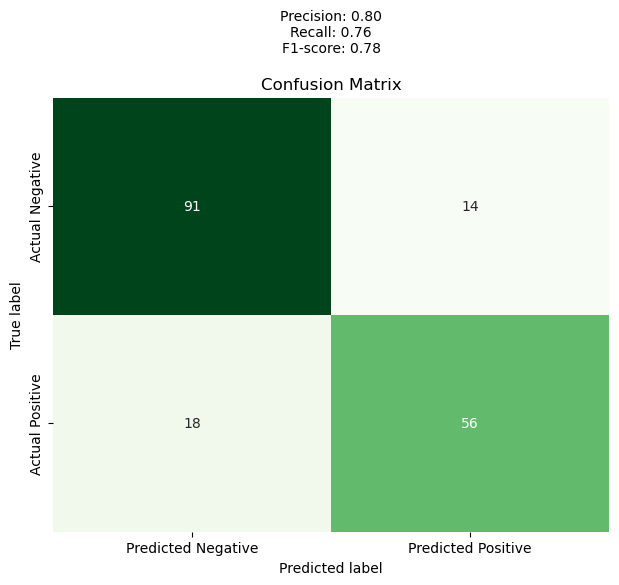
Best Model: Random Forest

Best Parameters: {'n\_estimators': 50, 'max\_depth': 10}

Best Validation Score: 0.8212

Short summary of the best model:

sc = evaluate\_best\_model(best\_model, X\_train, y\_train, X\_val, y\_val)



Train Accuracy: 0.9242

Validation Accuracy: 0.8212

#### Results

Final Model that was running on the validation data created:

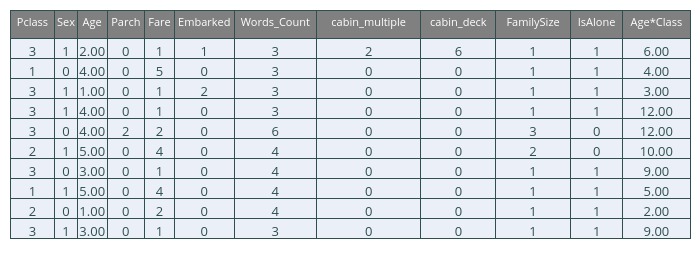
* 83.24% Accuracy for Survival

\*we added more variety of values in the parameters, but it took longer with no impact to the accuracy, so we stayed with these values.

## Test input data for submission

passenger\_df\_test = passenger\_df[passenger\_df.Survived == -1].drop("Survived", axis=1)

passenger\_df\_test.vu(10)



predictions = best\_model.predict(passenger\_df\_test)

predictions = predictions.astype(int)

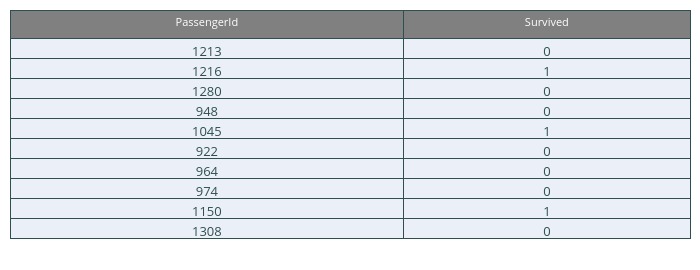
output = pd.DataFrame({'PassengerId': passenger\_df\_test.index, 'Survived': predictions})

output.to\_csv('submission.csv', index=False)

print("Your submission was successfully saved!")

Your submission was successfully saved!

output.vu(10)



### Discussion

Our Hypothesis between each other was:

* Should we classify and engineer missing values in the data or just randomly write values based on the mean and standard deviation of the data?
* Based on the preprocessed data, which is the best model to learn new data?

The Hypothesis is answered in multiple categories:

#### Preprocessing

Based on the Inferred data, some features that have barely any contribution to classifing Survival like Ticket, Name and even in some cases Cabin. Therefore we either need to completely filter them or engineer them to further correlate with the data. Important Features like Age, Family members and Sex need to fully encode for them to get learned by the model. Age specifically has some null values, therefore the easiest process to fill the missing values is to randomly assign values based on mean and standard deviation, but it’s less accurate. We decided that the most accurate method to fill missing values in Age is to estimate based on learning the other features in the data and using regression models.

#### Inference

This Step takes the prepocessed data from the previous step, and the objective is to classify if the passenger survives or not, therefore we need a classification model. we used 5 model types: Logistic Regression, Decision Trees, SVM, Random Forest and Gradient Boosting. Each with numerous parameter values added into a 10-fold Cross Validation function, which picks the combination that estimates the trained data with the highest accuracy score(Visualized in GridScores for each parameter). We considered using regularization models Lasso and Ridge to introduce cost functions fitting regression models and rounding up there output values ( submitted to Kaggle in addition to the actual notebook) and concluded that the main focus of this task is classification, therefore we put the regression models asside.

#### Postprocessing

After model training different models, we pick the model based on its accuracy metrics with new data, in othe words, validation data that hasn’t been trained in the models. We assumed the best model to fit the data is based on decision trees (Boosting,Random Forest or Regular Trees). After comparing model training, Random Forest gives the best validation accuracy score, therefore we’ll use this model for estimating test data.