Titanic - Machine Learning from Disaster

Assignment 1 submission

Michael Berger[1]

Son Levi

Wednesday 15th May, 2024

Abstract

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.

Keywords

1 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

[1] Correspondence to: michael.berger.e@gmail.com

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.

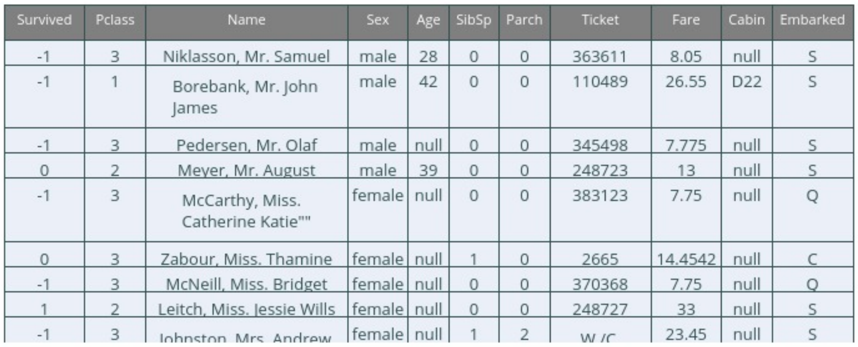
We split the work so Michael will be mainly responsible for the data cleansing and Preprocessing and Son will be mainly responsible for the model training and Evaluating. We both show our individual Data Exploration and merge if necessary.

2 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,
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
```

3 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)
```



Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
-1	3	Niklasson, Mr. Samuel	male	28	0	0	363611	8.05	null	S
-1	1	Borebank, Mr. John James	male	42	0	0	110489	26.55	D22	S
-1	3	Pedersen, Mr. Olaf	male	null	0	0	345498	7.775	null	S
0	2	Meyer, Mr. August	male	39	0	0	248723	13	null	S
-1	3	McCarthy, Miss. Catherine Katie""	female	null	0	0	383123	7.75	null	Q
0	3	Zabour, Miss. Thamine	female	null	1	0	2665	14.4542	null	C
-1	3	McNeill, Miss. Bridget	female	null	0	0	370368	7.75	null	Q
1	2	Leitch, Miss. Jessie Wills	female	null	0	0	248727	33	null	S
-1	3	Johnston, Mrs. Andrew	female	null	1	2	W./C.	23.45	null	S

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.

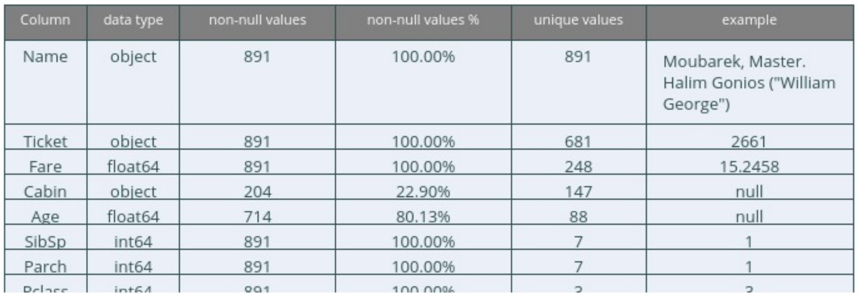
4 Explatory Data Analysis (EDA) and Data Visualization

4.1 Part 1 - Data Visualization

4.1.1 Describe Data

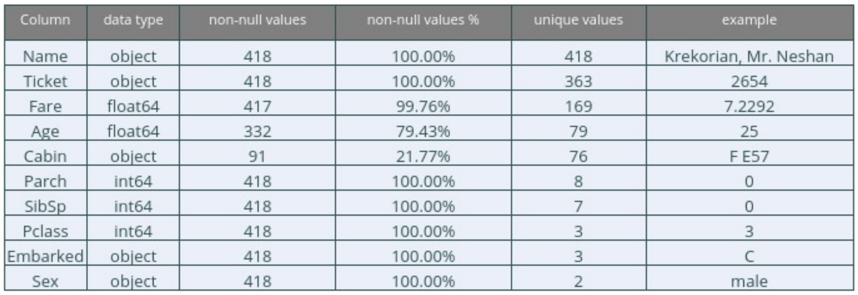
Check if there any null values

```
summary(passenger_df_train).vu(r=0)
```



Column	data type	non-null values	non-null values %	unique values	example
Name	object	891	100.00%	891	Moubarek, Master. Halim Gonios ("William George")
Ticket	object	891	100.00%	681	2661
Fare	float64	891	100.00%	248	15.2458
Cabin	object	204	22.90%	147	null
Age	float64	714	80.13%	88	null
SibSp	int64	891	100.00%	7	1
Parch	int64	891	100.00%	7	1
Pclass	int64	891	100.00%	3	3

```
summary(passenger_df_test).vu(r=0)
```



Column	data type	non-null values	non-null values %	unique values	example
Name	object	418	100.00%	418	Krekorian, Mr. Neshan
Ticket	object	418	100.00%	363	2654
Fare	float64	417	99.76%	169	7.2292
Age	float64	332	79.43%	79	25
Cabin	object	91	21.77%	76	F E57
Parch	int64	418	100.00%	8	0
SibSp	int64	418	100.00%	7	0
Pclass	int64	418	100.00%	3	3
Embarked	object	418	100.00%	3	C
Sex	object	418	100.00%	2	male

```
passenger_df_train.describe()
```

	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.00	891.00	714.00	891.00	891.00	891.00
mean	0.38	2.31	29.70	0.52	0.38	32.20
std	0.49	0.84	14.53	1.10	0.81	49.69
min	0.00	1.00	0.42	0.00	0.00	0.00
25%	0.00	2.00	20.12	0.00	0.00	7.91
50%	0.00	3.00	28.00	0.00	0.00	14.45
75%	1.00	3.00	38.00	1.00	0.00	31.00
max	1.00	3.00	80.00	8.00	6.00	512.33

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.

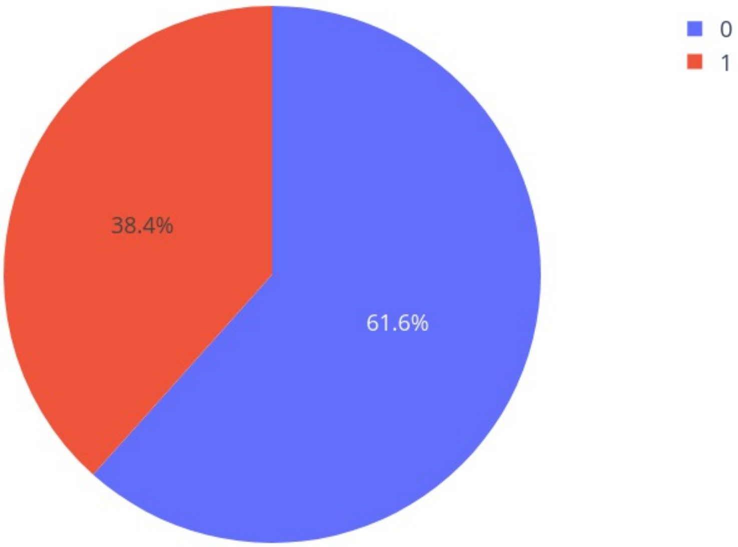
What are the data types for various features?

Helping us during converting goal.

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

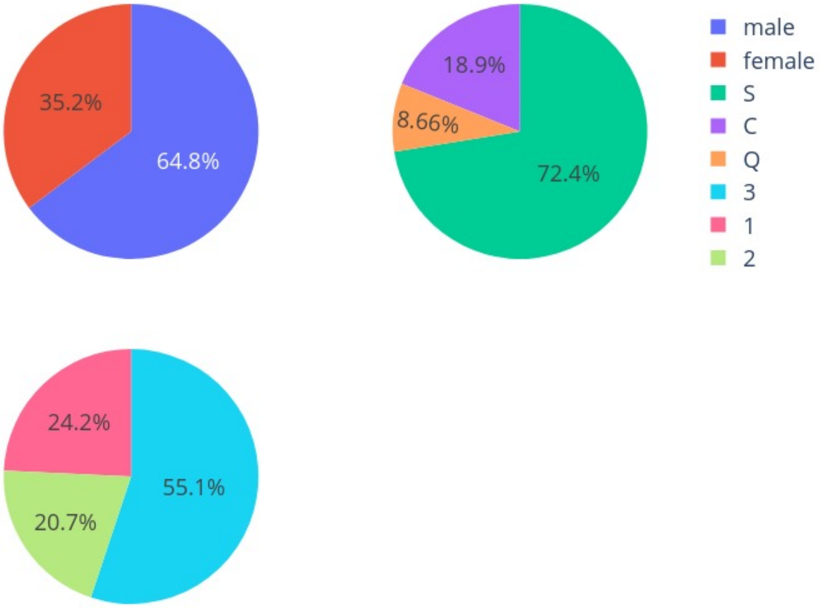
4.1.2 Amount of Survivors

```
create_pie_chart_of_count(passenger_df_train, 'Survived')
```



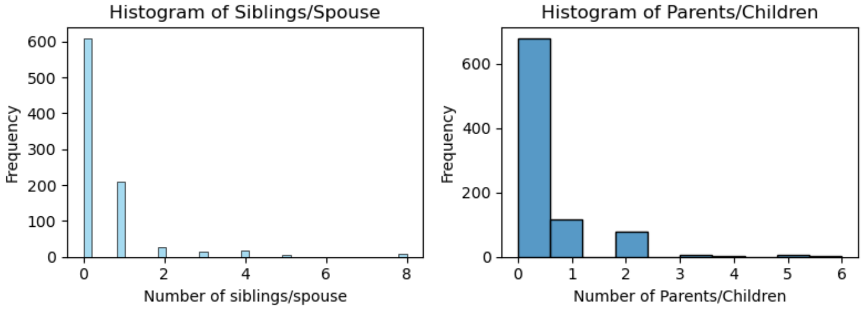
4.1.3 Pie Charts for Embark, Sex and Pclass

```
create_pie_chart_subplot_of_count(passenger_df_train, ['Sex', 'Embarked', 'Pclass'])
```



4.1.4 Histograms for Siblings/Spouse and Parents/Children

```
fig
```



4.2 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

4.3 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.

4.4 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).

4.4.1 Comparing non-null features to survived

4.4.2 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).

4.4.3 Based on the Pclass vs Survived Histograms:

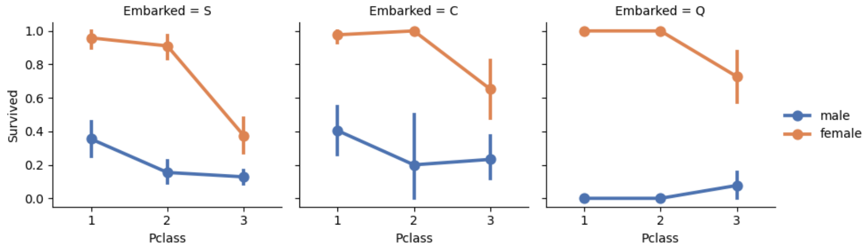
Observations

- 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
```



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

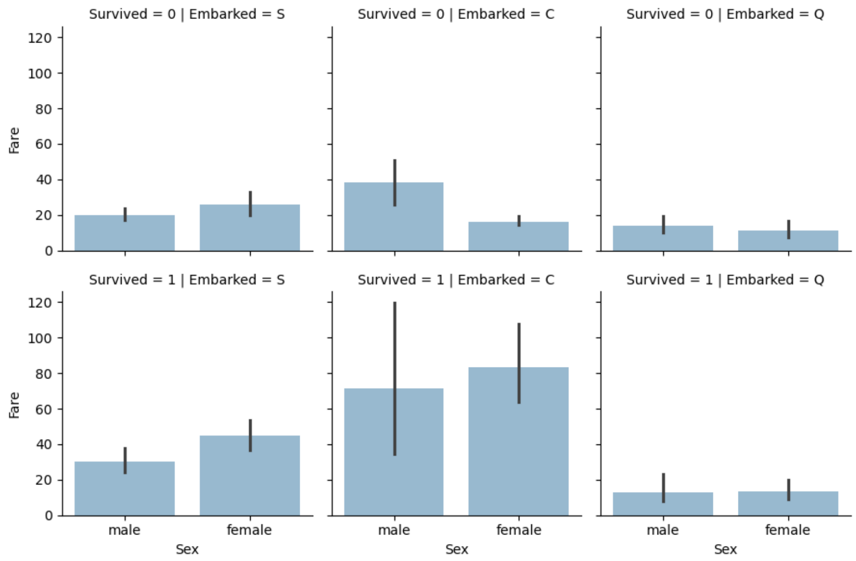
Observations

- 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
```



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

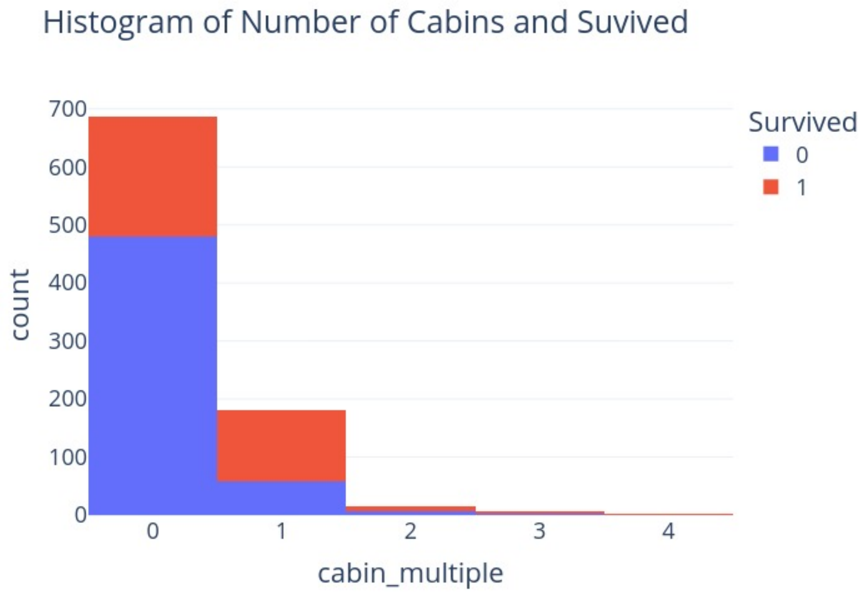
Observations

- 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

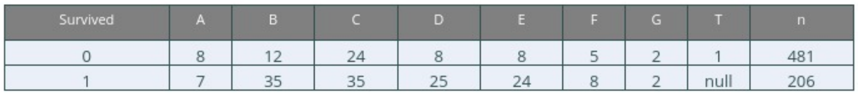
- Consider banding Fare feature.

```
px.histogram(data_frame= cabin_divide, x="cabin_multiple", color="Survived",title='Histogram of Number
```



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)
```



Survived	A	B	C	D	E	F	G	T	n
0	8	12	24	8	8	5	2	1	481
1	7	35	35	25	24	8	2	null	206

4.4.6 Based on the Cabins Pivot Tables:

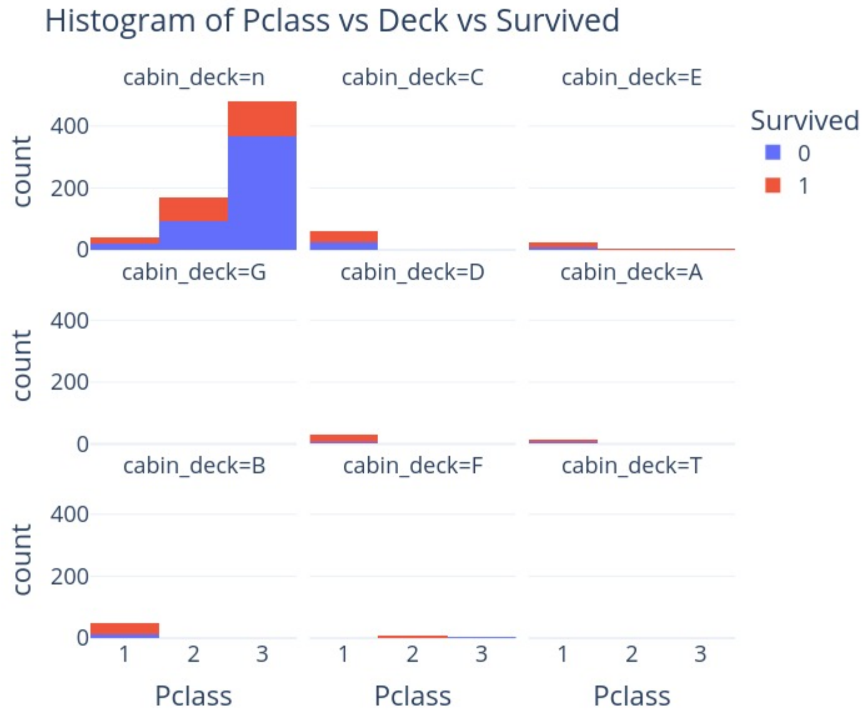
Observations

- 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

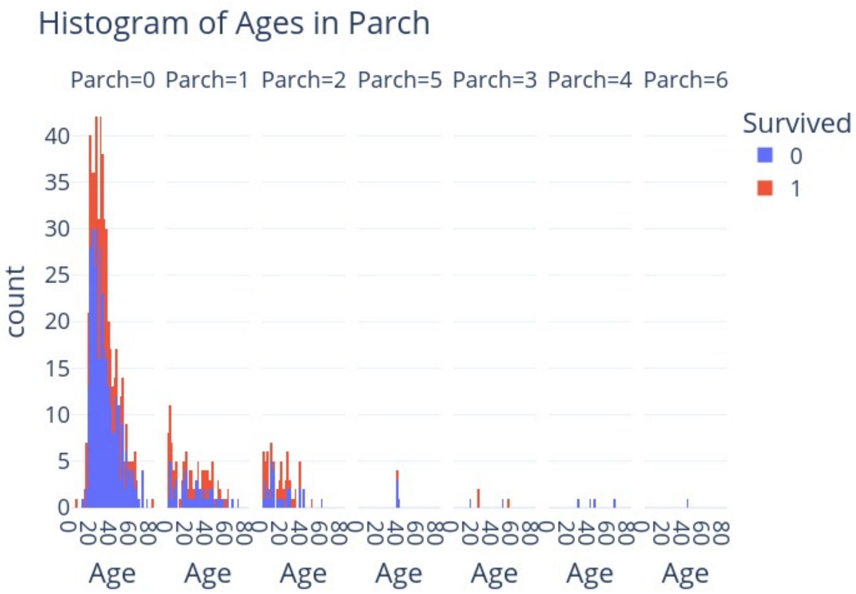
- Consider Seperating the cabin feature into only cabin letters.
- Consider creating a number of Cabins feature.

```
fig
```



Ages vs ParCh

```
px.histogram(data_frame= passenger_df_train, facet_col="Parch",
             x="Age", color="Survived",title='Histogram of Ages in Parch')
```

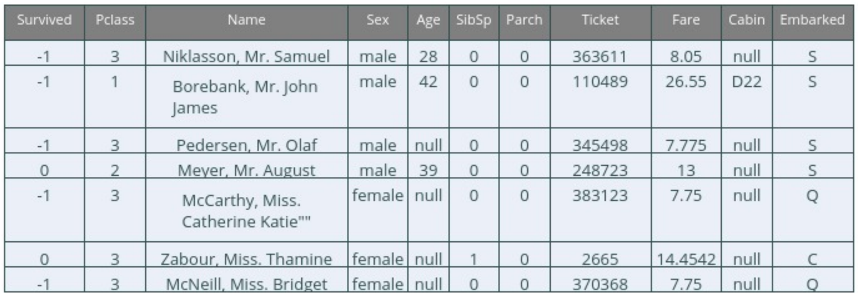


/newpage

4.5 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)
```

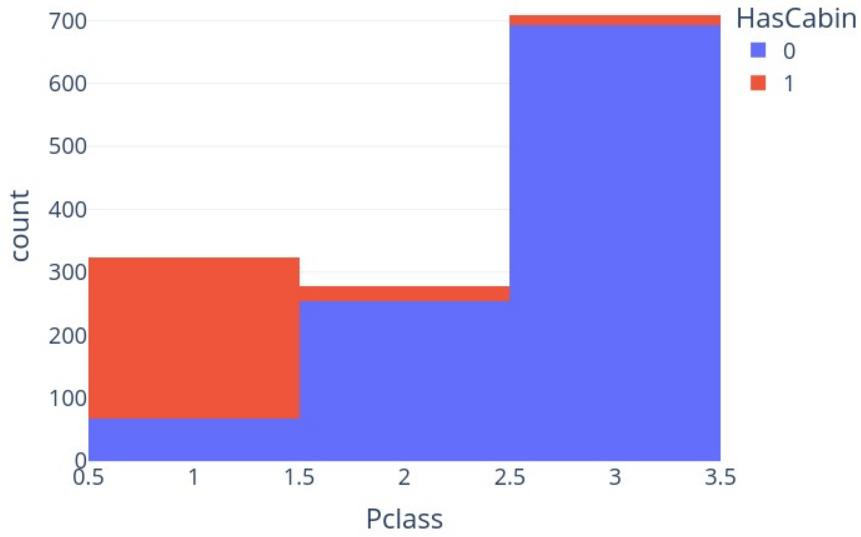


Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
-1	3	Niklasson, Mr. Samuel	male	28	0	0	363611	8.05	null	S
-1	1	Borebank, Mr. John James	male	42	0	0	110489	26.55	D22	S
-1	3	Pedersen, Mr. Olaf	male	null	0	0	345498	7.775	null	S
0	2	Meyer, Mr. August	male	39	0	0	248723	13	null	S
-1	3	McCarthy, Miss. Catherine Katie""	female	null	0	0	383123	7.75	null	Q
0	3	Zabour, Miss. Thamine	female	null	1	0	2665	14.4542	null	C
-1	3	McNeill, Miss. Bridget	female	null	0	0	370368	7.75	null	Q

4.5.1 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 %
```



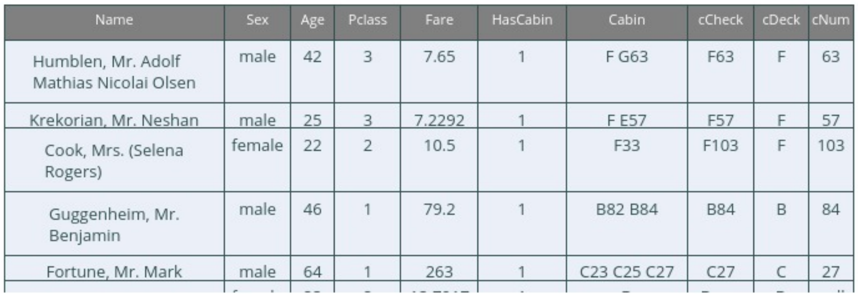
We observe that the cabin column is the emptiest. Only 22% of the passengers have it, with the majority being in first class. Upon examining the Titanic deck plans, we see that all the living space was represented by cabins. We can conclude that the emptiness in data is not because those passengers did 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()
```



Name	Sex	Age	Pclass	Fare	HasCabin	Cabin	cCheck	cDeck	cNum
Humblen, Mr. Adolf Mathias Nicolai Olsen	male	42	3	7.65	1	F G63	F63	F	63
Krekorian, Mr. Neshan	male	25	3	7.2292	1	F E57	F57	F	57
Cook, Mrs. (Selena Rogers)	female	22	2	10.5	1	F33	F103	F	103
Guggenheim, Mr. Benjamin	male	46	1	79.2	1	B82 B84	B84	B	84
Fortune, Mr. Mark	male	64	1	263	1	C23 C25 C27	C27	C	27

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.

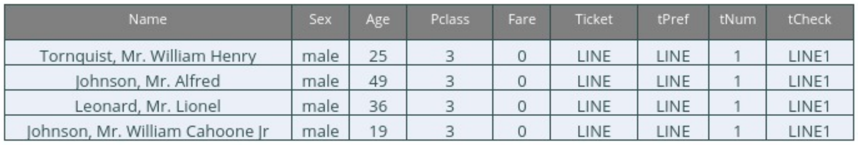
4.5.2 Ticket/placement

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)
```

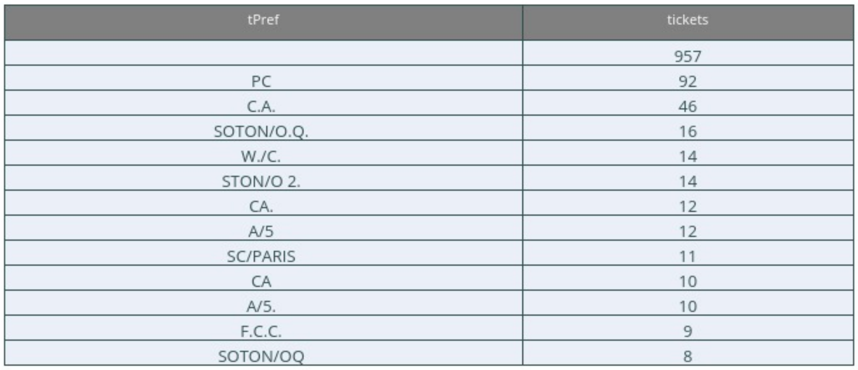


Name	Sex	Age	Pclass	Fare	Ticket	tPref	tNum	tCheck
Tornquist, Mr. William Henry	male	25	3	0	LINE	LINE	1	LINE1
Johnson, Mr. Alfred	male	49	3	0	LINE	LINE	1	LINE1
Leonard, Mr. Lionel	male	36	3	0	LINE	LINE	1	LINE1
Johnson, Mr. William Cahoone Jr	male	19	3	0	LINE	LINE	1	LINE1

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)
```

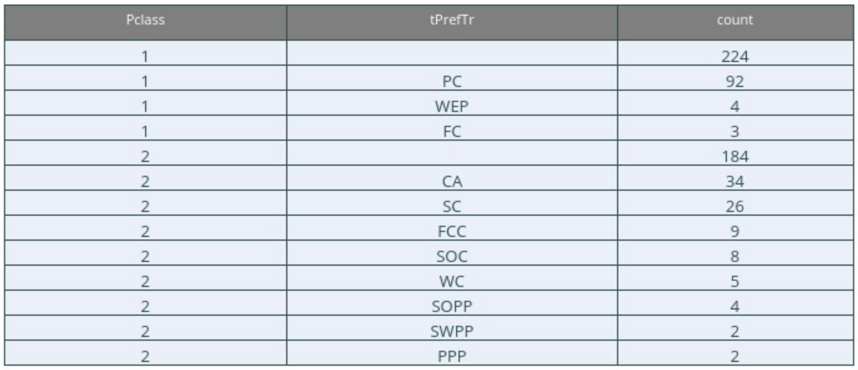


tPref	tickets
	957
PC	92
C.A.	46
SOTON/O.Q.	16
W./C.	14
STON/O 2.	14
CA.	12
A/5	12
SC/PARIS	11
CA	10
A/5.	10
F.C.C.	9
SOTON/OQ	8

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)
```



Pclass	tPrefTr	count
1		224
1	PC	92
1	WEP	4
1	FC	3
2		184
2	CA	34
2	SC	26
2	FCC	9
2	SOC	8
2	WC	5
2	SOPP	4
2	SWPP	2
2	PPP	2

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

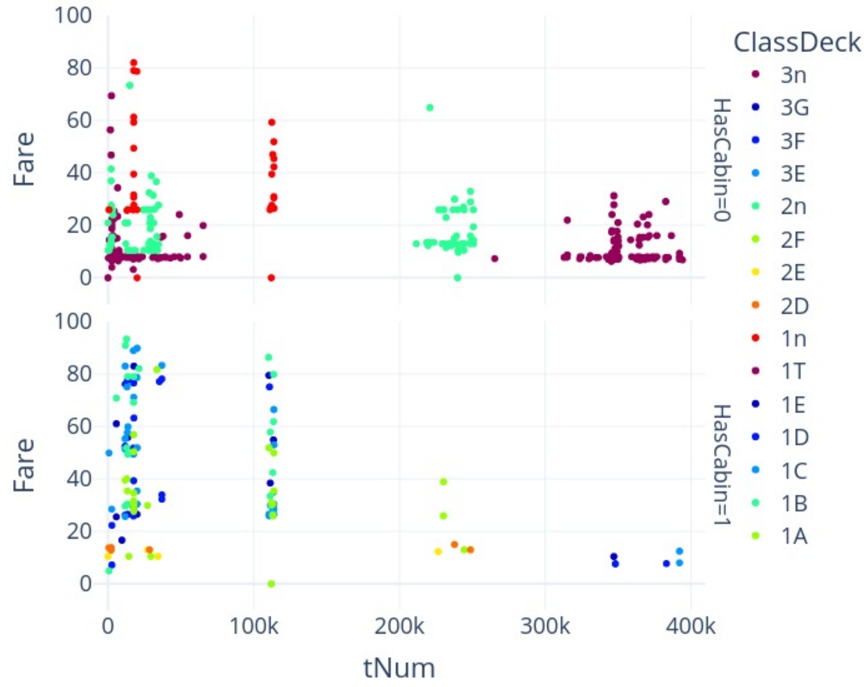
Ticket numbers seem to be concentrated into several groups. Internet seems to suggest that these groups come from individual stores from which the tickets were purchased. The order inside group is probably the order in which the tickets were purchased, so probably not very relevant to current research.

For now I could not identify any interaction pattern of tickets numbers with other features.

#TODO: describe this more

Zooming down to only 2, 3 classes, I was wondering if cabin deck, number or side might somehow be "encoded" in Fare and ticket number. But the data is too sparse to make any judjement on that.

```
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=600, range_x= [ -10000, 4.1e5], range_y=[ -10,100])
fig
```



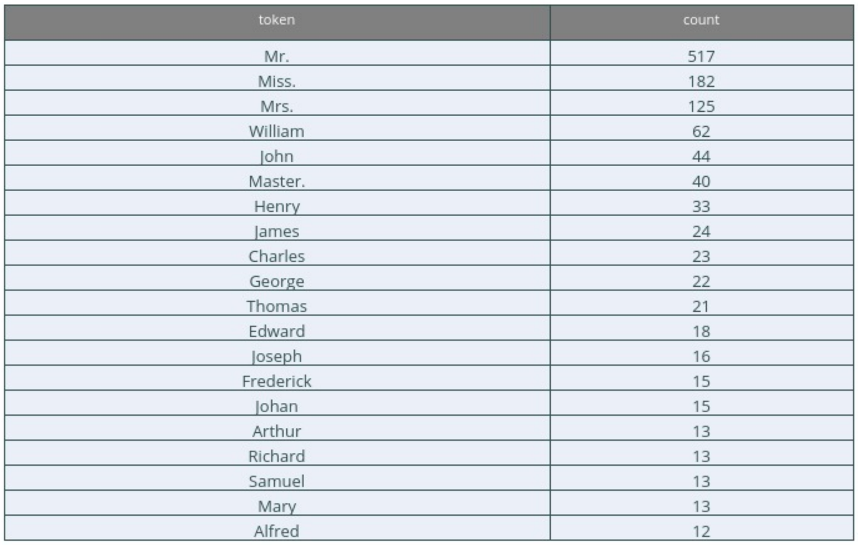
4.5.3 Age

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

4.5.4 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)
```

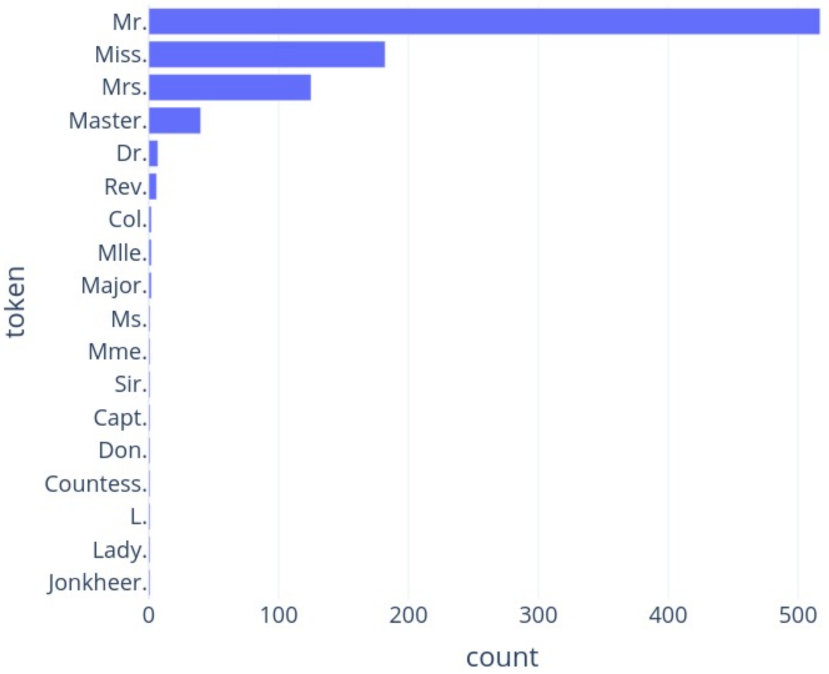


token	count
Mr.	517
Miss.	182
Mrs.	125
William	62
John	44
Master.	40
Henry	33
James	24
Charles	23
George	22
Thomas	21
Edward	18
Joseph	16
Frederick	15
Johan	15
Arthur	13
Richard	13
Samuel	13
Mary	13
Alfred	12

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

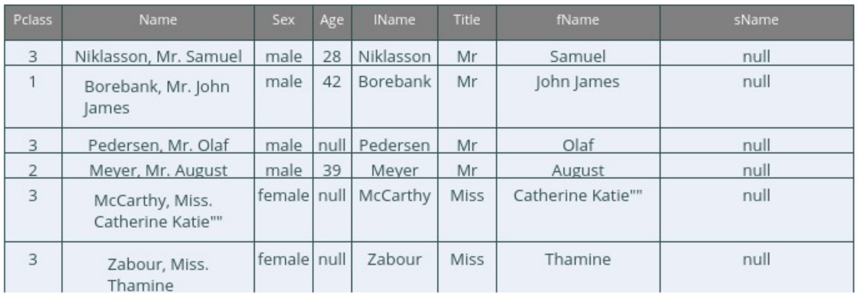
```
print(perc)
fig
```

```
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)
```

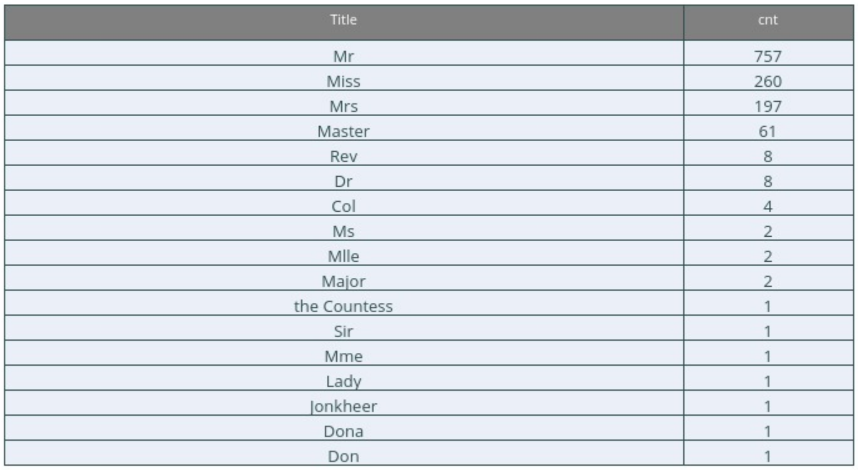


Pclass	Name	Sex	Age	lName	Title	fName	sName
3	Niklasson, Mr. Samuel	male	28	Niklasson	Mr	Samuel	null
1	Borebank, Mr. John James	male	42	Borebank	Mr	John James	null
3	Pedersen, Mr. Olaf	male	null	Pedersen	Mr	Olaf	null
2	Meyer, Mr. August	male	39	Meyer	Mr	August	null
3	McCarthy, Miss. Catherine Katie""	female	null	McCarthy	Miss	Catherine Katie""	null
3	Zabour, Miss. Thamine	female	null	Zabour	Miss	Thamine	null

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)
```



Title	cnt
Mr	757
Miss	260
Mrs	197
Master	61
Rev	8
Dr	8
Col	4
Ms	2
Mlle	2
Major	2
the Countess	1
Sir	1
Mme	1
Lady	1
Jonkheer	1
Dona	1
Don	1

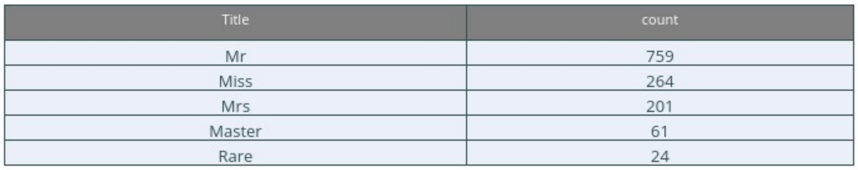
We have 17 titles in total, most of which are common: Mr, Mrs, Miss and Master, with the rest being rare:

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)
```

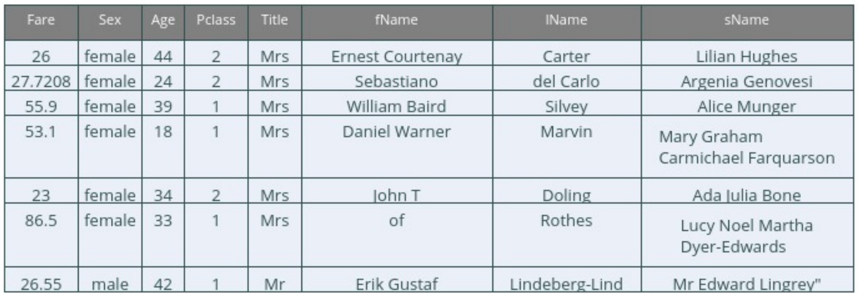


Title	count
Mr	759
Miss	264
Mrs	201
Master	61
Rare	24

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

Second name Let's explore the second name (sName column):

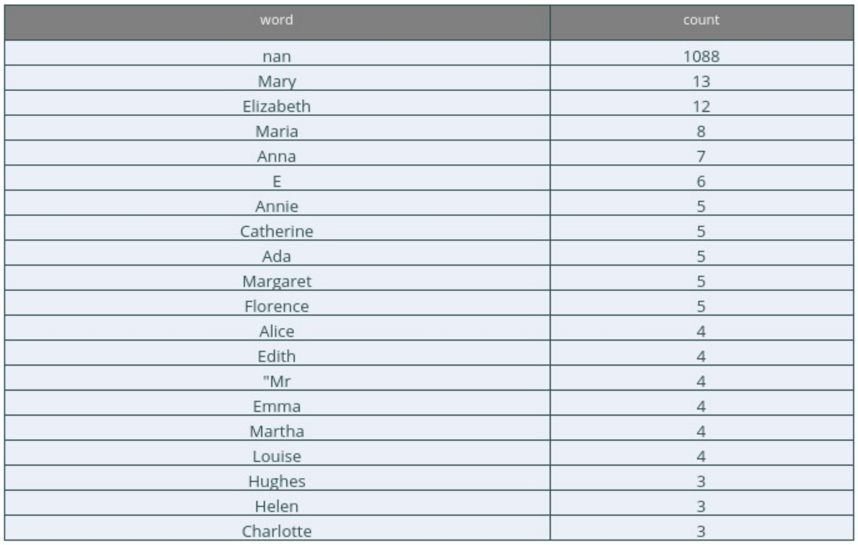
```
coi = "Fare Sex Age Pclass Title fName lName sName".split(" ")

passenger_df.loc[~passenger_df.sName.isna(),coi].vu(10)
```



Fare	Sex	Age	Pclass	Title	fName	lName	sName
26	female	44	2	Mrs	Ernest Courtenay	Carter	Lilian Hughes
27.7208	female	24	2	Mrs	Sebastiano	del Carlo	Argenia Genovesi
55.9	female	39	1	Mrs	William Baird	Silvey	Alice Munger
53.1	female	18	1	Mrs	Daniel Warner	Marvin	Mary Graham Carmichael Farquarson
23	female	34	2	Mrs	John T	Doling	Ada Julia Bone
86.5	female	33	1	Mrs	of	Rothes	Lucy Noel Martha Dyer-Edwards
26.55	male	42	1	Mr	Erik Gustaf	Lindeberg-Lind	Mr Edward Lingrey"

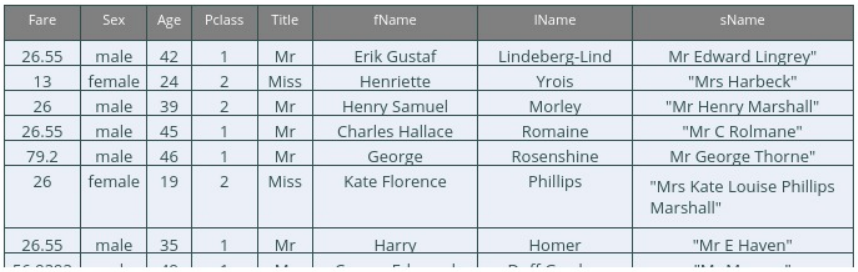
We perform a similar token analysis with these as with the full name before

```
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)
```



word	count
nan	1088
Mary	13
Elizabeth	12
Maria	8
Anna	7
E	6
Annie	5
Catherine	5
Ada	5
Margaret	5
Florence	5
Alice	4
Edith	4
"Mr	4
Emma	4
Martha	4
Louise	4
Hughes	3
Helen	3
Charlotte	3

```
passenger_df.loc[passenger_df.sName.astype(str).str.contains("Mr"), coi].vu(9)
```



Fare	Sex	Age	Pclass	Title	fName	lName	sName
26.55	male	42	1	Mr	Erik Gustaf	Lindeberg-Lind	Mr Edward Lingrey"
13	female	24	2	Miss	Henriette	Yrois	"Mrs Harbeck"
26	male	39	2	Mr	Henry Samuel	Morley	"Mr Henry Marshall"
26.55	male	45	1	Mr	Charles Hallace	Romaine	"Mr C Rolmane"
79.2	male	46	1	Mr	George	Rosenshine	Mr George Thorne"
26	female	19	2	Miss	Kate Florence	Phillips	"Mrs Kate Louise Phillips Marshall"
26.55	male	35	1	Mr	Harry	Homer	"Mr E Haven"

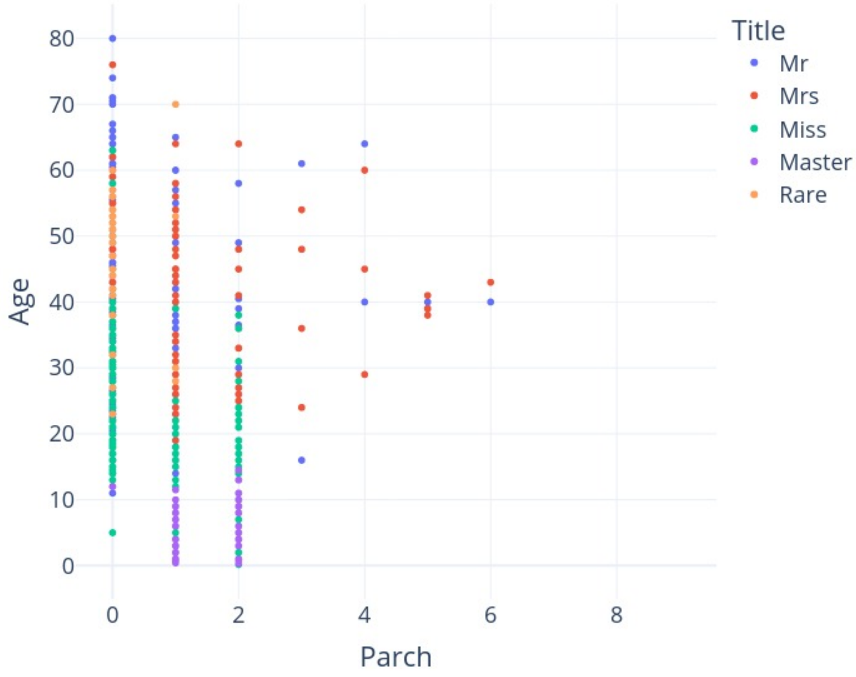
Seems like majority of these contain full/maiden names of women travelling with ticket under their husbands' names.
Maybe this property could be used to separate SibSp column into Siblings and Spouses and ultimately, for Age estimation.

4.5.5 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.

4.6 Part 2 - Data Engineering + Encoding Categorical Values

4.7 Data Imputation

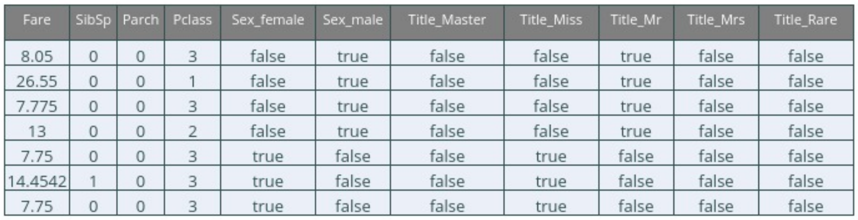
As discussed in Exploration section, about 20% of passengers have no Are registered. We would like to impute the null values of Age with an estimate based on other variables.

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)
```

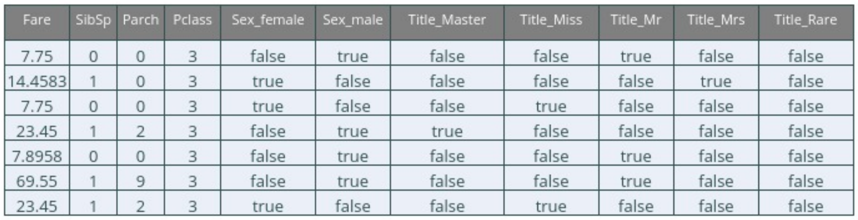
```
y = passenger_df[Cy]
X.vu(7)
```



Fare	SibSp	Parch	Pclass	Sex_female	Sex_male	Title_Master	Title_Miss	Title_Mr	Title_Mrs	Title_Rare
8.05	0	0	3	false	true	false	false	true	false	false
26.55	0	0	1	false	true	false	false	true	false	false
7.775	0	0	3	false	true	false	false	true	false	false
13	0	0	2	false	true	false	false	true	false	false
7.75	0	0	3	true	false	false	true	false	false	false
14.4542	1	0	3	true	false	false	true	false	false	false
7.75	0	0	3	true	false	false	true	false	false	false

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

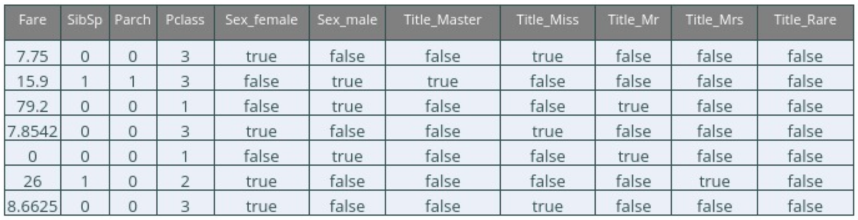
```
Ximp = X[y.isna()]
yimp = y[y.isna()]
Ximp.vu(7)
```



Fare	SibSp	Parch	Pclass	Sex_female	Sex_male	Title_Master	Title_Miss	Title_Mr	Title_Mrs	Title_Rare
7.75	0	0	3	false	true	false	false	true	false	false
14.4583	1	0	3	true	false	false	false	false	true	false
7.75	0	0	3	true	false	false	true	false	false	false
23.45	1	2	3	false	true	true	false	false	false	false
7.8958	0	0	3	false	true	false	false	true	false	false
69.55	1	9	3	false	true	false	false	true	false	false
23.45	1	2	3	true	false	false	true	false	false	false

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)
```



Fare	SibSp	Parch	Pclass	Sex_female	Sex_male	Title_Master	Title_Miss	Title_Mr	Title_Mrs	Title_Rare
7.75	0	0	3	true	false	false	true	false	false	false
15.9	1	1	3	false	true	true	false	false	false	false
79.2	0	0	1	false	true	false	false	true	false	false
7.8542	0	0	3	true	false	false	true	false	false	false
0	0	0	1	false	true	false	false	true	false	false
26	1	0	2	true	false	false	false	false	true	false
8.6625	0	0	3	true	false	false	true	false	false	false

And split the Dataset into Train and Validation

4.7.1 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 4 candidates, totalling 40 fits
Elapsed Time: 00:00:01
====================
Best Score: 0.441
Best Parameters: {'max_depth': 10, 'max_features': 6, 'max_leaf_nodes': 23, 'n_estimators': 50}
```

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)
```

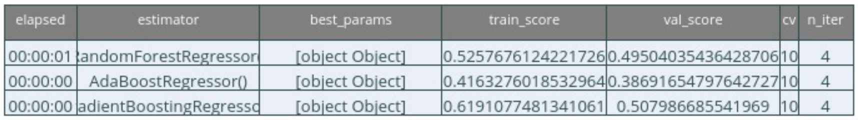
```
Fitting 10 folds for each of 4 candidates, totalling 40 fits
Elapsed Time: 00:00:00
====================
Best Score: 0.391
Best Parameters: {'learning_rate': 1, 'n_estimators': 147}
```

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)
```

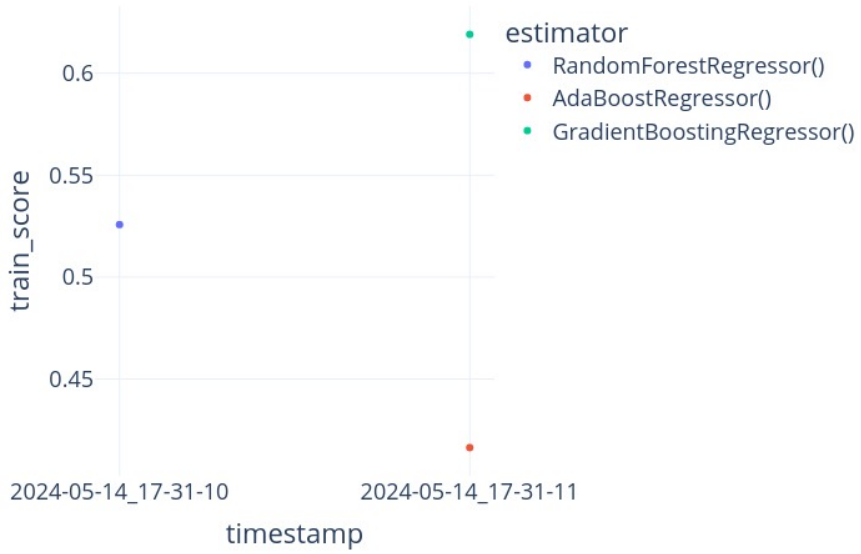
```
Fitting 10 folds for each of 4 candidates, totalling 40 fits
Elapsed Time: 00:00:00
====================
Best Score: 0.433
Best Parameters: {'max_depth': 12, 'max_features': 7, 'max_leaf_nodes': 11, 'n_estimators': 83}
```

```
CV_df = pd.DataFrame(CV_Runs)
CV_df[['elapsed', 'estimator', 'best_params', 'train_score',
       'val_score', 'cv', 'n_iter']].vu(r=0)
```



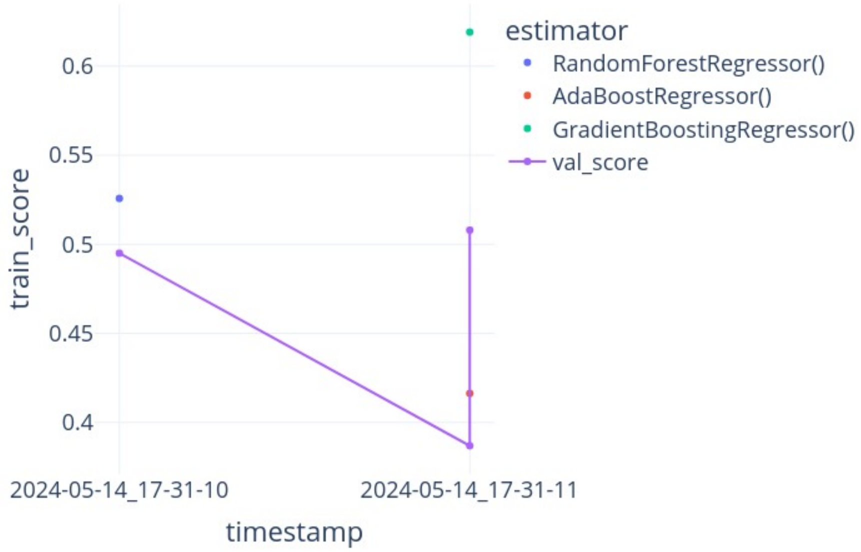
elapsed	estimator	best_params	train_score	val_score	cv	n_iter
00:00:01	RandomForestRegressor	[object Object]	0.5257676124221726	0.49504035436428706	10	4
00:00:00	AdaBoostRegressor()	[object Object]	0.4163276018532964	0.38691654797642727	10	4
00:00:00	adientBoostingRegresso	[object Object]	0.6191077481341061	0.507986685541969	10	4

```
fig = px.scatter(CV_df, x="timestamp", y="train_score", color="estimator")
fig
```



```
fig.add_trace( go.Scatter(x=CV_df["timestamp"], y=CV_df["val_score"], name="val_score", )) #, fill=CV_d
```

```
fig
```



4.7.2 Estimate missing ages

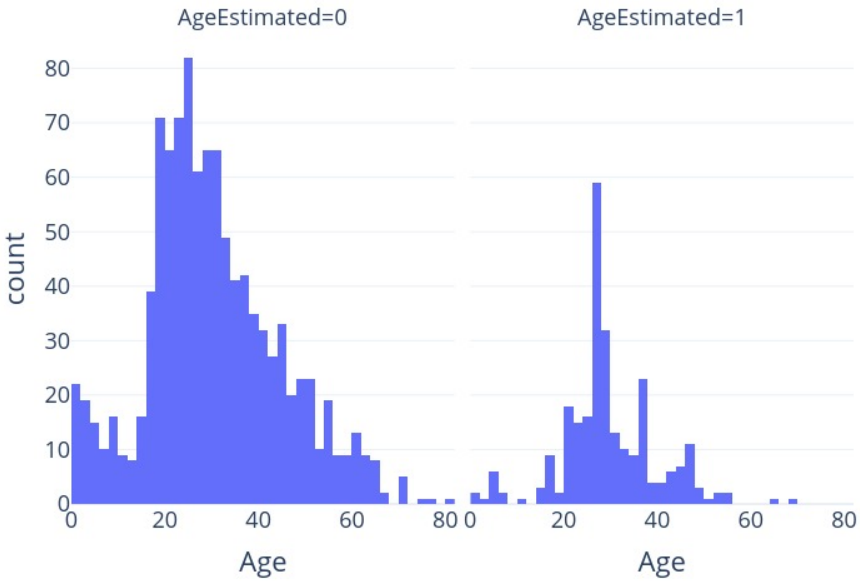
Based on the benchmarking results above, we decided to choose model 3 (GradientBoostingRegressor)

```
best_params = {'max_depth': 13, 'max_features': 5, 'max_leaf_nodes': 29, 'n_estimators': 435}
```

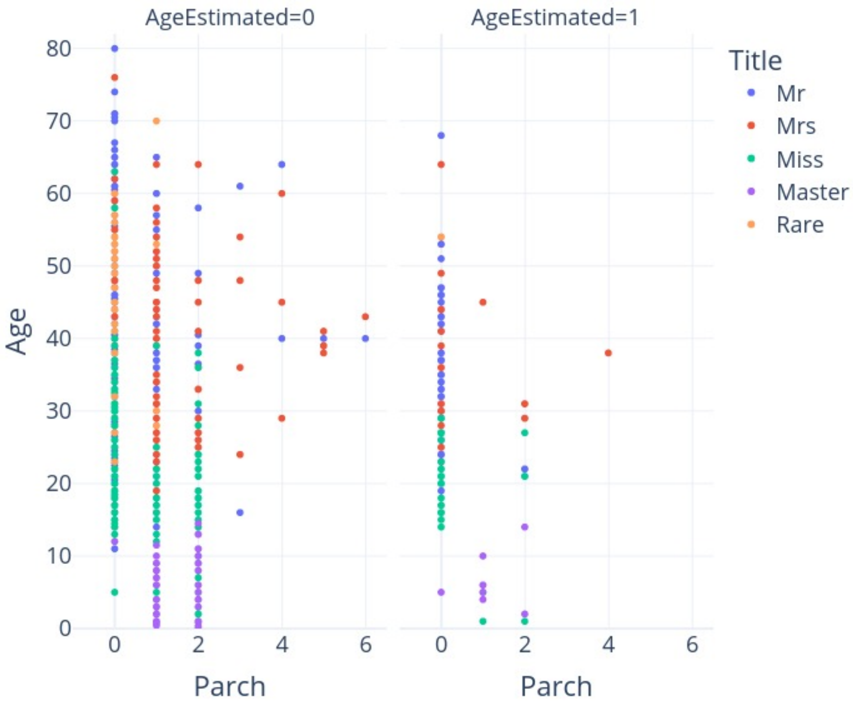
```
rfc = GradientBoostingRegressor(**best_params)
rfc.fit(X,y)
y_hat = rfc.predict(Ximp)
y_hat = pd.Series(rfc.predict(Ximp), index=Ximp.index)
```

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)
```

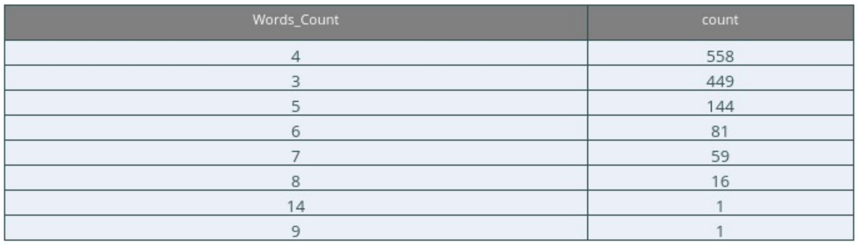


It seems that imputation went quite well.

4.8 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)
```

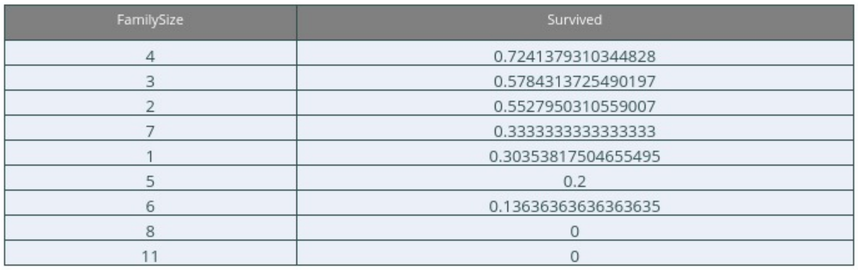


Words_Count	count
4	558
3	449
5	144
6	81
7	59
8	16
14	1
9	1

Creating new features:

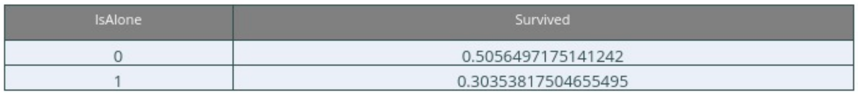
- cabin_multiple: number of cabins each passenger had.
- cabin_deck: the ship's deck where cabnin is located
- FamilySize as a combination of SibSp and Parch
- IsAlone from FamilySize

```
df1.vu(r=0)
```



FamilySize	Survived
4	0.7241379310344828
3	0.5784313725490197
2	0.5527950310559007
7	0.3333333333333333
1	0.30353817504655495
5	0.2
6	0.13636363636363635
8	0
11	0

```
df2.vu(r=0)
```

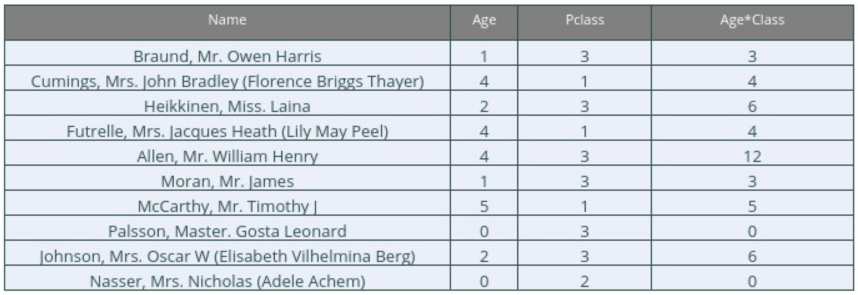


IsAlone	Survived
0	0.5056497175141242
1	0.30353817504655495

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

4.9 Mapping Categorical and High Ordinal Features

```
passenger_df.loc[:, ['Name', 'Age', 'Pclass', 'Age*Class']].vu(10,0)
```

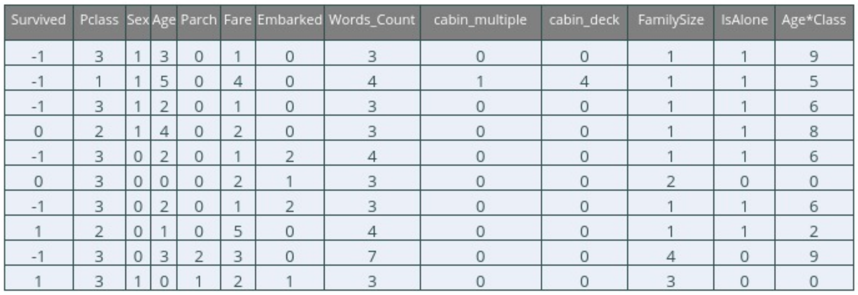


Name	Age	Pclass	Age*Class
Braund, Mr. Owen Harris	1	3	3
Cumings, Mrs. John Bradley (Florence Briggs Thayer)	4	1	4
Heikkinen, Miss. Laina	2	3	6
Futrelle, Mrs. Jacques Heath (Lily May Peel)	4	1	4
Allen, Mr. William Henry	4	3	12
Moran, Mr. James	1	3	3
McCarthy, Mr. Timothy J	5	1	5
Palsson, Master. Gosta Leonard	0	3	0
Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	2	3	6
Nasser, Mrs. Nicholas (Adele Achem)	0	2	0

4.10 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.

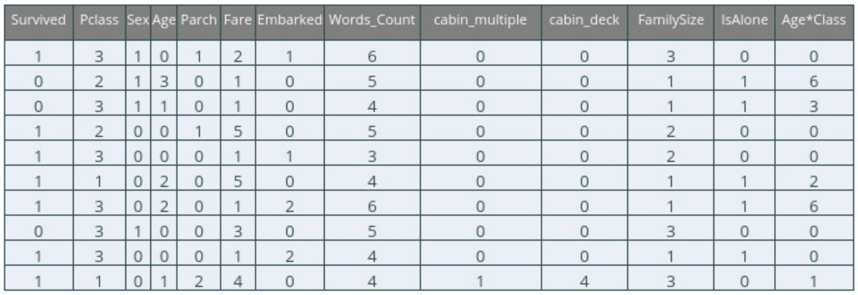
```
passenger_df.vu(10)
```



Survived	Pclass	Sex	Age	Parch	Fare	Embarked	Words_Count	cabin_multiple	cabin_deck	FamilySize	IsAlone	Age*Class
-1	3	1	3	0	1	0	3	0	0	1	1	9
-1	1	1	5	0	4	0	4	1	4	1	1	5
-1	3	1	2	0	1	0	3	0	0	1	1	6
0	2	1	4	0	2	0	3	0	0	1	1	8
-1	3	0	2	0	1	2	4	0	0	1	1	6
0	3	0	0	0	2	1	3	0	0	2	0	0
-1	3	0	2	0	1	2	3	0	0	1	1	6
1	2	0	1	0	5	0	4	0	0	1	1	2
-1	3	0	3	2	3	0	7	0	0	4	0	9
1	3	1	0	1	2	1	3	0	0	3	0	0

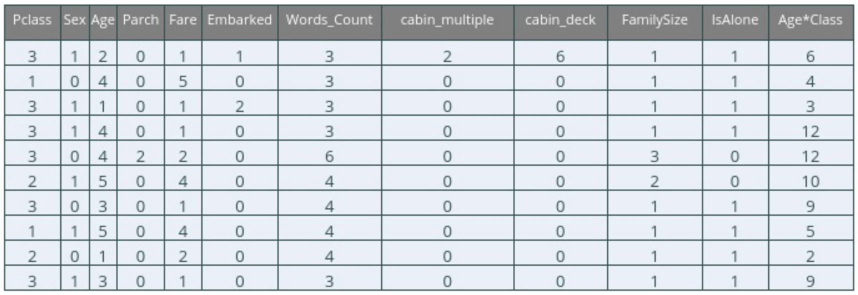
Splitting the dataset back into training and testing

```
passenger_df_train = passenger_df[passenger_df.Survived != -1]
passenger_df_train.vu(10)
```



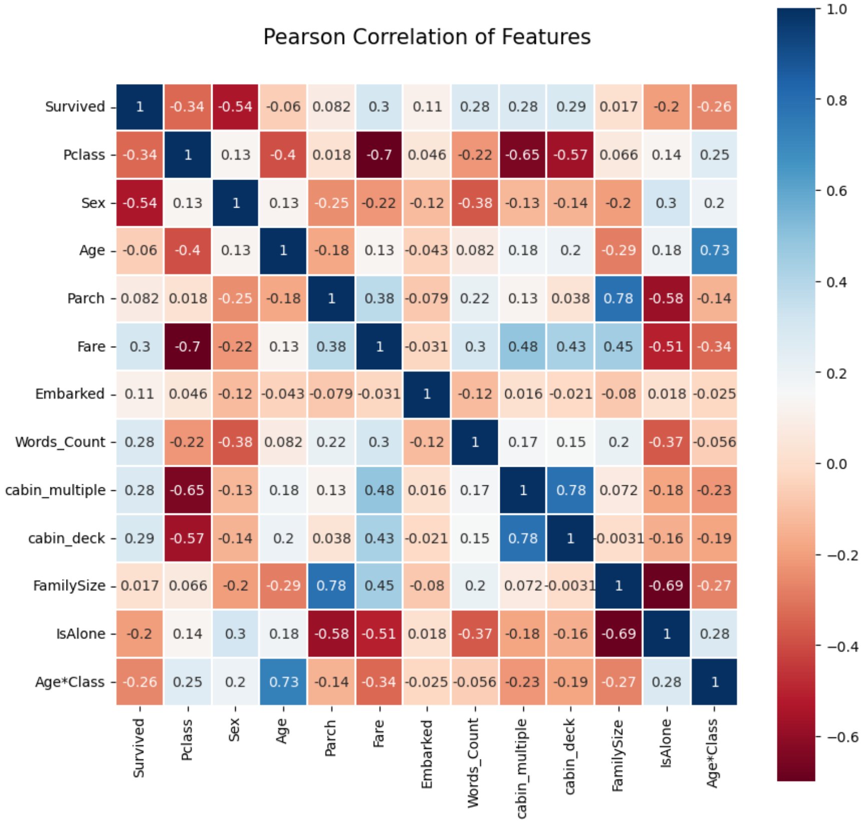
Survived	Pclass	Sex	Age	Parch	Fare	Embarked	Words_Count	cabin_multiple	cabin_deck	FamilySize	IsAlone	Age*Class
1	3	1	0	1	2	1	6	0	0	3	0	0
0	2	1	3	0	1	0	5	0	0	1	1	6
0	3	1	1	0	1	0	4	0	0	1	1	3
1	2	0	0	1	5	0	5	0	0	2	0	0
1	3	0	0	0	1	1	3	0	0	2	0	0
1	1	0	2	0	5	0	4	0	0	1	1	2
1	3	0	2	0	1	2	6	0	0	1	1	6
0	3	1	0	0	3	0	5	0	0	3	0	0
1	3	0	0	0	1	2	4	0	0	1	1	0
1	1	0	1	2	4	0	4	1	4	3	0	1

```
passenger_df_test = passenger_df[passenger_df.Survived == -1].drop("Survived", axis=1)
passenger_df_test.vu(10)
```



Pclass	Sex	Age	Parch	Fare	Embarked	Words_Count	cabin_multiple	cabin_deck	FamilySize	IsAlone	Age*Class
3	1	2	0	1	1	3	2	6	1	1	6
1	0	4	0	5	0	3	0	0	1	1	4
3	1	1	0	1	2	3	0	0	1	1	3
3	1	4	0	1	0	3	0	0	1	1	12
3	0	4	2	2	0	6	0	0	3	0	12
2	1	5	0	4	0	4	0	0	2	0	10
3	0	3	0	1	0	4	0	0	1	1	9
1	1	5	0	4	0	4	0	0	1	1	5
2	0	1	0	2	0	4	0	0	1	1	2
3	1	3	0	1	0	3	0	0	1	1	9

fig



	Survived	Pclass	Sex	Age	Parch	Fare	Embarked	Words_Count	cabin_multiple	cabin_deck	FamilySize	IsAlone	Age*Class
Survived	1	-0.34	-0.54	-0.06	0.082	0.3	0.11	0.28	0.28	0.29	0.017	-0.2	-0.26
Pclass	-0.34	1	0.13	-0.4	0.018	-0.7	0.046	-0.22	-0.65	-0.57	0.066	0.14	0.25
Sex	-0.54	0.13	1	0.13	-0.25	-0.22	-0.12	-0.38	-0.13	-0.14	-0.2	0.3	0.2
Age	-0.06	-0.4	0.13	1	-0.18	0.13	-0.043	0.082	0.18	0.2	-0.29	0.18	0.73
Parch	0.082	0.018	-0.25	-0.18	1	0.38	-0.079	0.22	0.13	0.038	0.78	-0.58	-0.14
Fare	0.3	-0.7	-0.22	0.13	0.38	1	-0.031	0.3	0.48	0.43	0.45	-0.51	-0.34
Embarked	0.11	0.046	-0.12	-0.043	-0.079	-0.031	1	-0.12	0.016	-0.021	-0.08	0.018	-0.025
Words_Count	0.28	-0.22	-0.38	0.082	0.22	0.3	-0.12	1	0.17	0.15	0.2	-0.37	-0.056
cabin_multiple	0.28	-0.65	-0.13	0.18	0.13	0.48	0.016	0.17	1	0.78	0.072	-0.18	-0.23
cabin_deck	0.29	-0.57	-0.14	0.2	0.038	0.43	-0.021	0.15	0.78	1	-0.0031	-0.16	-0.19
FamilySize	0.017	0.066	-0.2	-0.29	0.78	0.45	-0.08	0.2	0.072	-0.0031	1	-0.69	-0.27
IsAlone	-0.2	0.14	0.3	0.18	-0.58	-0.51	0.018	-0.37	-0.18	-0.16	-0.69	1	0.28
Age*Class	-0.26	0.25	0.2	0.73	-0.14	-0.34	-0.025	-0.056	-0.23	-0.19	-0.27	0.28	1

4.11 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.

5 Model Learning

5.1 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)
```

5.2 Model Functions

5.2.1 Train and Evaluate models

5.2.2 Visualize Results

5.2.3 Confusion Matrix for Best Model

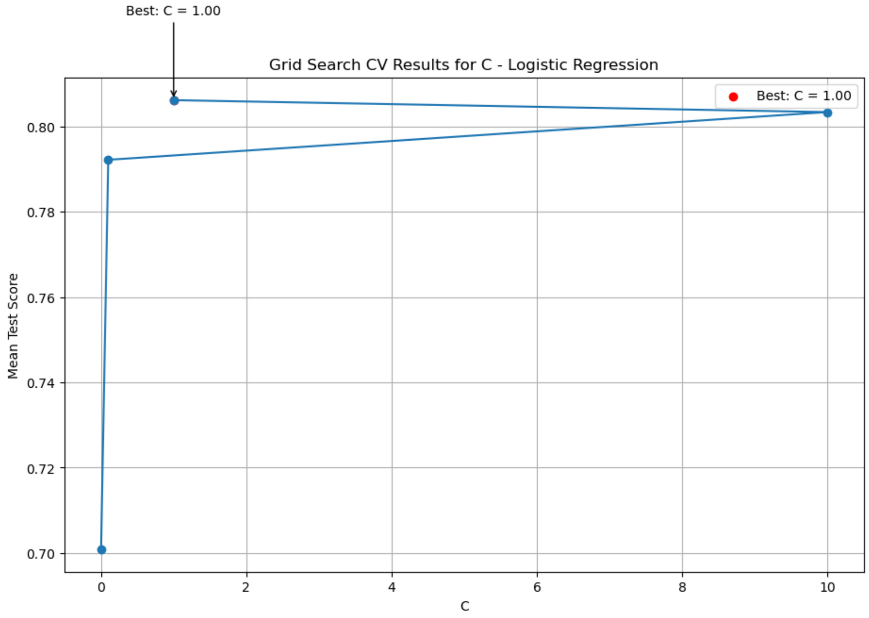
5.3 Classic models

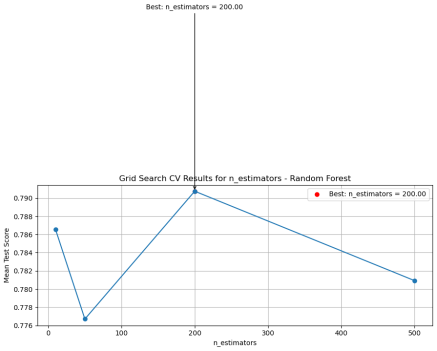
Initialize models with Hyperparameters

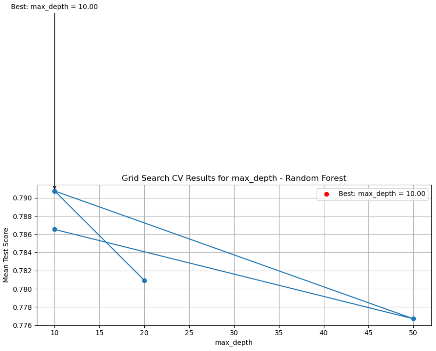
```
# Define models
models = {
    'Logistic Regression': LogisticRegression(),
    'Random Forest': RandomForestClassifier(),
    'SVM': SVC(),
    'Gradient Boosting': GradientBoostingClassifier(),
    'Decision Tree': DecisionTreeClassifier()
}

# 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]}
}
```

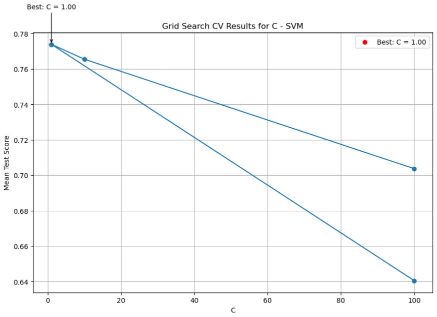
Training and Evaluating the models Showing the process of Cross Validation using each and every combination of parameters for each model, so we can show in the final dataframe their best parameter combination (highest model mean test score)

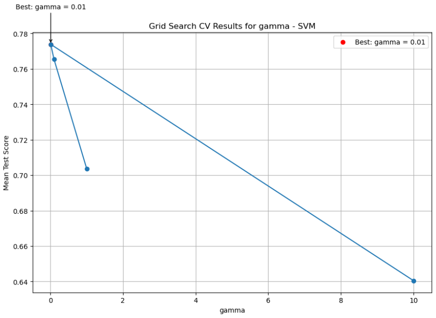
```
results, evaluation_df = train_and_evaluate_models(models, X_train, y_train, X_val, y_val,
                                                   param_grids, scoring='accuracy', cv=10)
```

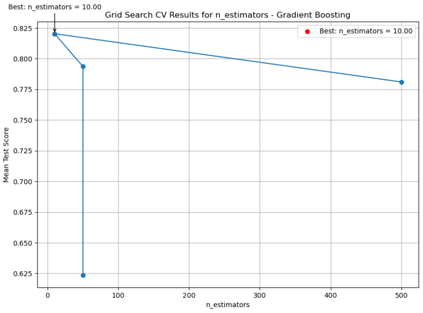

Best: C = 1.00
Grid Search CV Results for C - Logistic Regression
● Best: C = 1.00
Mean Test Score
0.70
0.72
0.74
0.76
0.78
0.80
0
2
4
6
8
10
C

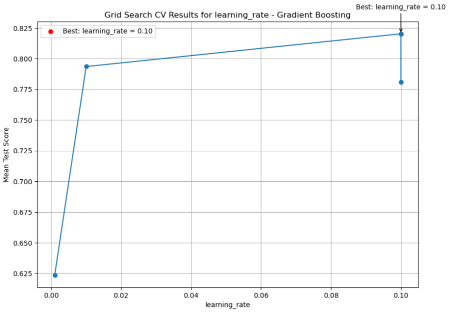

Best: n_estimators = 200.00
Grid Search CV Results for n_estimators - Random Forest
● Best: n_estimators = 200.00
Mean Test Score
0.776
0.778
0.780
0.782
0.784
0.786
0.788
0.790
0
100
200
300
400
500
n_estimators

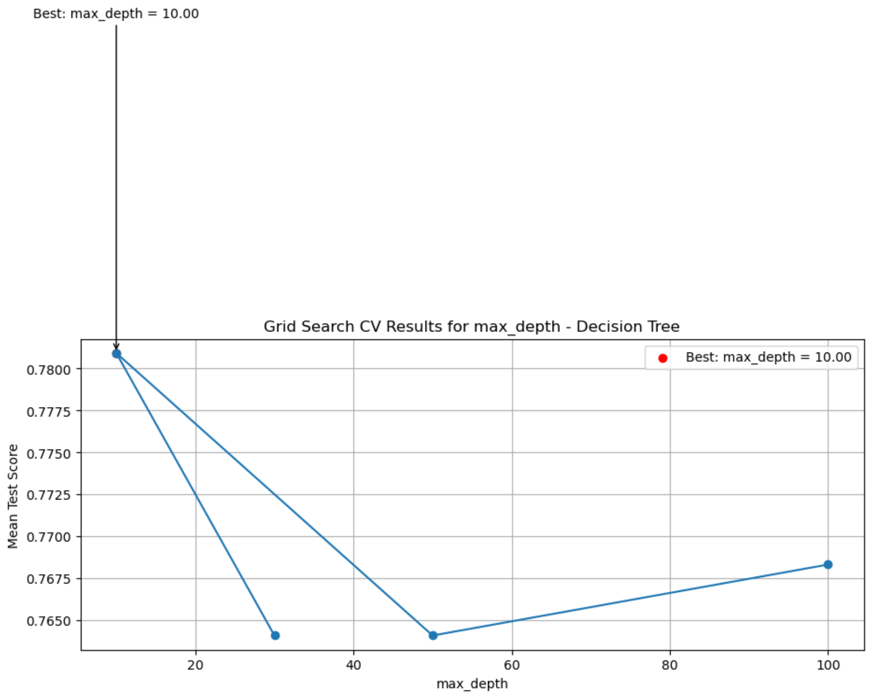

Best: max_depth = 10.00
Grid Search CV Results for max_depth - Random Forest
● Best: max_depth = 10.00
Mean Test Score
0.776
0.778
0.780
0.782
0.784
0.786
0.788
0.790
10
15
20
25
30
35
40
45
50
max_depth


Best: C = 1.00
Grid Search CV Results for C - SVM
● Best: C = 1.00
Mean Test Score
0.64
0.66
0.68
0.70
0.72
0.74
0.76
0.78
0
20
40
60
80
100
C

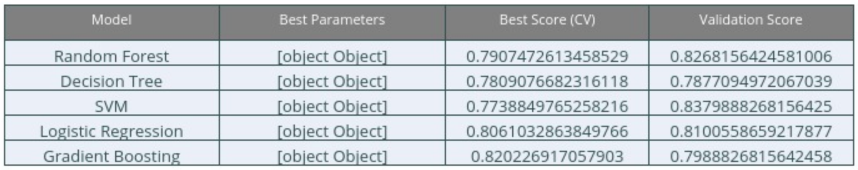

Best: gamma = 0.01
Grid Search CV Results for gamma - SVM
● Best: gamma = 0.01
Mean Test Score
0.64
0.66
0.68
0.70
0.72
0.74
0.76
0.78
0
2
4
6
8
10
gamma


Best: n_estimators = 10.00
Grid Search CV Results for n_estimators - Gradient Boosting
● Best: n_estimators = 10.00
Mean Test Score
0.625
0.650
0.675
0.700
0.725
0.750
0.775
0.800
0.825
0
100
200
300
400
500
n_estimators


Best: learning_rate = 0.10
Grid Search CV Results for learning_rate - Gradient Boosting
● Best: learning_rate = 0.10
Mean Test Score
0.625
0.650
0.675
0.700
0.725
0.750
0.775
0.800
0.825
0.00
0.02
0.04
0.06
0.08
0.10
learning_rate



```
#TODO: deal with dict and float representation
evaluation_df.vu(5)
```



Model	Best Parameters	Best Score (CV)	Validation Score
Random Forest	[object Object]	0.7907472613458529	0.8268156424581006
Decision Tree	[object Object]	0.7809076682316118	0.7877094972067039
SVM	[object Object]	0.7738849765258216	0.8379888268156425
Logistic Regression	[object Object]	0.8061032863849766	0.8100558659217877
Gradient Boosting	[object Object]	0.820226917057903	0.7988826815642458

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

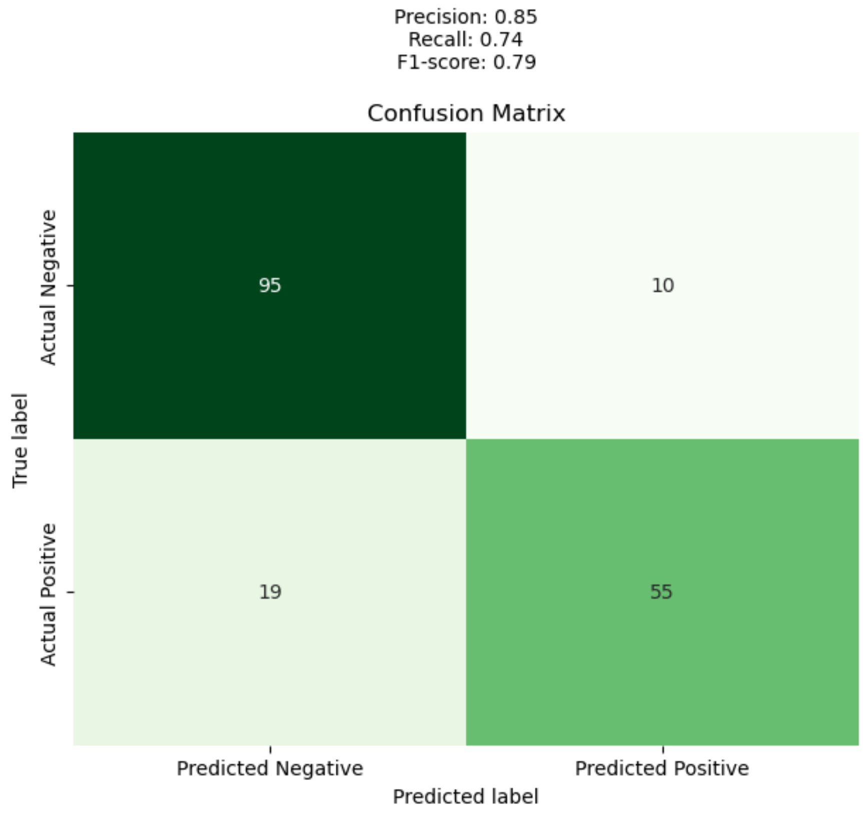
```
best_model_row = evaluation_df.loc[evaluation_df['Validation Score'].idxmax()]
best_model_name = best_model_row['Model']
best_validation_score = best_model_row['Validation Score']
best_model = models[best_model_name]

print("Best Model: ", best_model_name)
print(f"Best Parameters: {best_model_row['Best Parameters']}")
print("Best Validation Score: {:.4f}" .format(best_validation_score))
```

```
Best Model:  SVM
Best Parameters: {'gamma': 0.01, 'C': 1}
Best Validation Score: 0.8380
```

```
if iterations > 500 and best_model_name != "Gradient Boosting":
    raise AssertionError("best model is not Gradient Boosting model! ")
```

Short summary of the best model:

```
sc = evaluate_best_model(best_model, X_train, y_train, X_val, y_val)
```



```
Train Accuracy: 0.8062
Validation Accuracy: 0.8380
```

5.3.1 Results

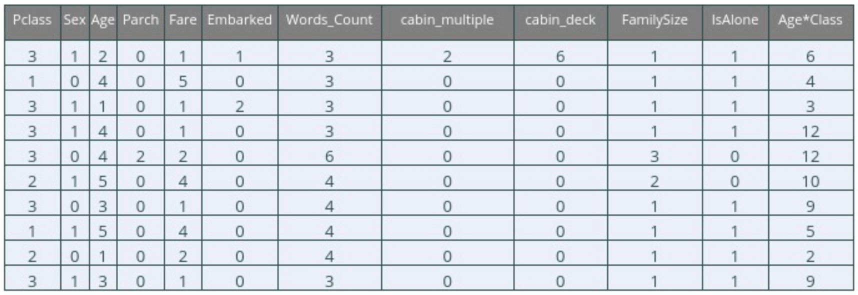
Final Model that was running on the validation data created:

- 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.

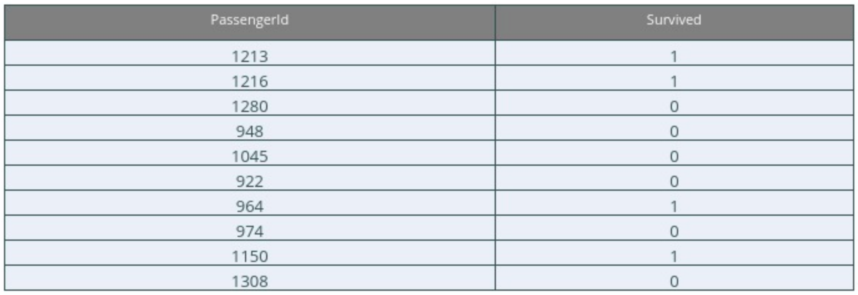
6 Test input data for submission

```
passenger_df_test = passenger_df[passenger_df.Survived == -1].drop("Survived", axis=1)
passenger_df_test.vu(10)
```



Pclass	Sex	Age	Parch	Fare	Embarked	Words_Count	cabin_multiple	cabin_deck	FamilySize	IsAlone	Age*Class
3	1	2	0	1	1	3	2	6	1	1	6
1	0	4	0	5	0	3	0	0	1	1	4
3	1	1	0	1	2	3	0	0	1	1	3
3	1	4	0	1	0	3	0	0	1	1	12
3	0	4	2	2	0	6	0	0	3	0	12
2	1	5	0	4	0	4	0	0	2	0	10
3	0	3	0	1	0	4	0	0	1	1	9
1	1	5	0	4	0	4	0	0	1	1	5
2	0	1	0	2	0	4	0	0	1	1	2
3	1	3	0	1	0	3	0	0	1	1	9

```
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)
```



PassengerId	Survived
1213	1
1216	1
1280	0
948	0
1045	0
922	0
964	1
974	0
1150	1
1308	0

6.1 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:

6.1.1 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.

6.1.2 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.

6.1.3 Postprocessing

After model training different models, we pick the model based on its accuracy metrics with new data, in othe the 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, Gradient Boosting gives the best validation accuracy score, therefore we'll use this model for estimating test data.