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: vo1kod4v@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.

2 Libraries

```
# @title
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

from datetime import datetime
import json
import scipy.stats as st
```

3 Reading Train Data

```
# deleteme

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()
```

Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
-1	3	Niklasson, Mr. Samuel	male	28.00	0	0	363611	8.05	nan	S
-1	1	Borebank, Mr. John James	male	42.00	0	0	110489	26.55	D22	S
-1	3	Pedersen, Mr. Olaf	male	nan	0	0	345498	7.78	nan	S
0	2	Meyer, Mr. August	male	39.00	0	0	248723	13.00	nan	S
-1	3	McCarthy, Miss. Catherine Katie""	female	nan	0	0	383123	7.75	nan	Q
0	3	Zabour, Miss. Thamine	female	nan	1	0	2665	14.45	nan	C
-1	3	McNeill, Miss. Bridget	female	nan	0	0	370368	7.75	nan	Q
1	2	Leitch, Miss. Jessie Wills	female	nan	0	0	248727	33.00	nan	S
-1	3	Johnston, Mrs. Andrew G (Elizabeth Lily" Watson)"	female	nan	1	2	W./C. 6607	23.45	nan	S
1	3	Moubarek, Master. Gerios	male	nan	1	1	2661	15.25	nan	C

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.

Check if there any null values

```
print(passenger_df_train.isna().any())
print('_'*40)
passenger_df_test.isna().any()
```

```
Survived    False
Pclass      False
Name        False
Sex         False
Age          True
SibSp       False
Parch       False
Ticket      False
Fare        False
Cabin        True
Embarked     True
dtype: bool
________________________________________
Pclass      False
Name        False
Sex         False
Age          True
SibSp       False
Parch       False
Ticket      False
Fare         True
Cabin        True
Embarked    False
Survived    False
dtype: bool
```

4 Explatory Data Analysis (EDA) and Data Visualization

4.1 Part 1 - Data Visualization

4.1.1 Describe Data

```
passenger_df_train.info()
```

```
print('_'*40)
passenger_df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 891 entries, 1 to 891
Data columns (total 11 columns):
 #   Column    Non -Null Count  Dtype
 - - -  - - - - - -    - - - - - - - - - - - - - -  - - - - -
 0   Survived  891 non -null    int64
 1   Pclass    891 non -null    int64
 2   Name      891 non -null    object
 3   Sex       891 non -null    object
 4   Age       714 non -null    float64
 5   SibSp     891 non -null    int64
 6   Parch     891 non -null    int64
 7   Ticket    891 non -null    object
 8   Fare      891 non -null    float64
 9   Cabin     204 non -null    object
 10  Embarked  889 non -null    object
dtypes: float64(2), int64(4), object(5)
memory usage: 83.5+ KB
________________________________________
<class 'pandas.core.frame.DataFrame'>
Index: 418 entries, 892 to 1309
Data columns (total 11 columns):
 #   Column    Non -Null Count  Dtype
 - - -  - - - - - -    - - - - - - - - - - - - - -  - - - - -
 0   Pclass    418 non -null    int64
 1   Name      418 non -null    object
 2   Sex       418 non -null    object
 3   Age       332 non -null    float64
 4   SibSp     418 non -null    int64
 5   Parch     418 non -null    int64
 6   Ticket    418 non -null    object
 7   Fare      417 non -null    float64
 8   Cabin     91 non -null     object
 9   Embarked  418 non -null    object
 10  Survived  418 non -null    int64
dtypes: float64(2), int64(4), object(5)
memory usage: 39.2+ KB
```

```
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

```
print(f'Train: There are {len(passenger_df_train["Ticket"].unique())} unique Ticket names and {len(passe
print(f'Test: There are {len(passenger_df_test["Ticket"].unique())} unique Ticket names and {len(passeng
```

```
Train: There are 681 unique Ticket names and 148 unique Cabins.
Test: There are 363 unique Ticket names and 77 unique Cabins.
```

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

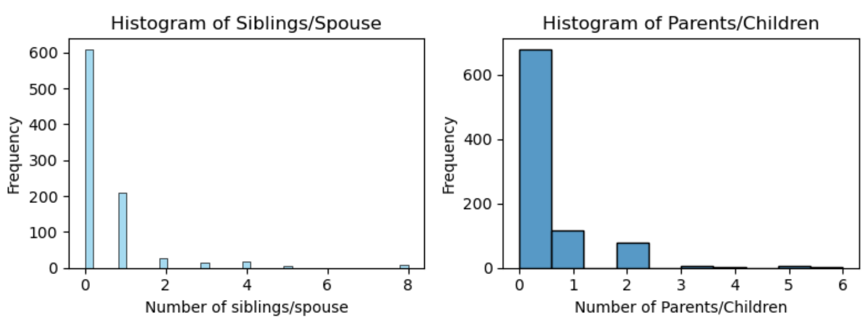
```
create_pie_chart_of_count(passenger_df_train, 'Survived')
```

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.

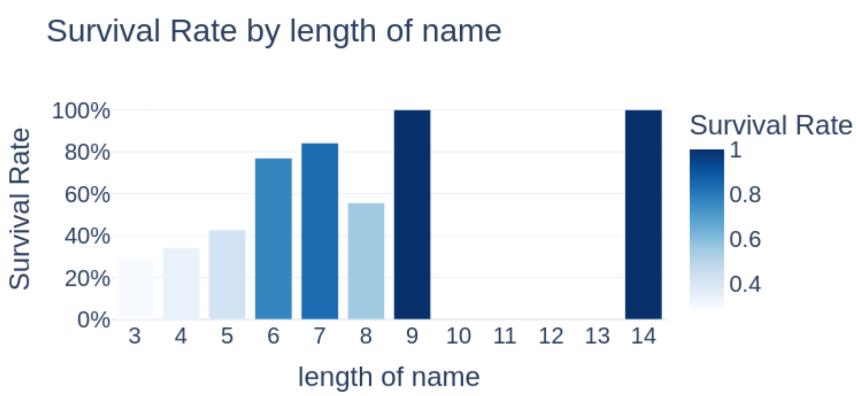
4.4 Data Exploration

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 do not have any empty values. It also makes sense doing so only for features which are categorical (Sex), ordinal (Pclass) or discrete (SibSp, Parch) type.

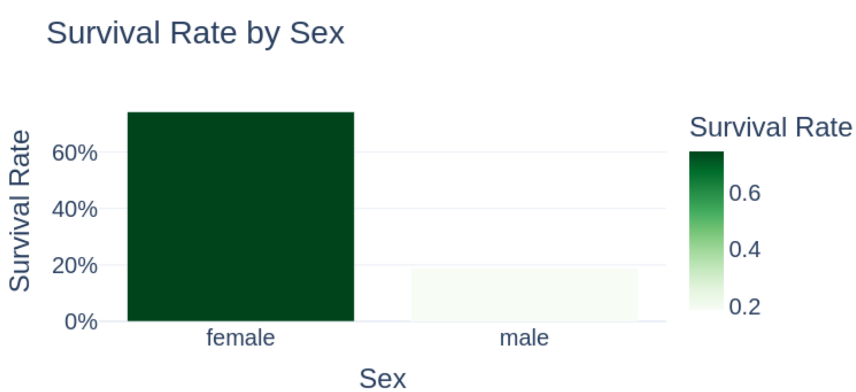
- **Pclass** We observe significant correlation (>0.5) among Pclass=1 and Survived (classifying #3). We decide to include this feature in our model.
- **Sex** We confirm the observation during problem definition that Sex=female had very high survival rate at 74% (classifying #1).
- **SibSp and Parch** 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

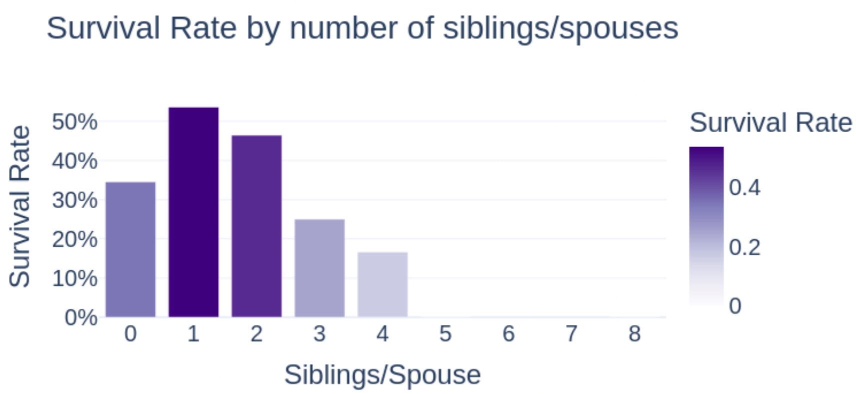
```
display(fig)
```



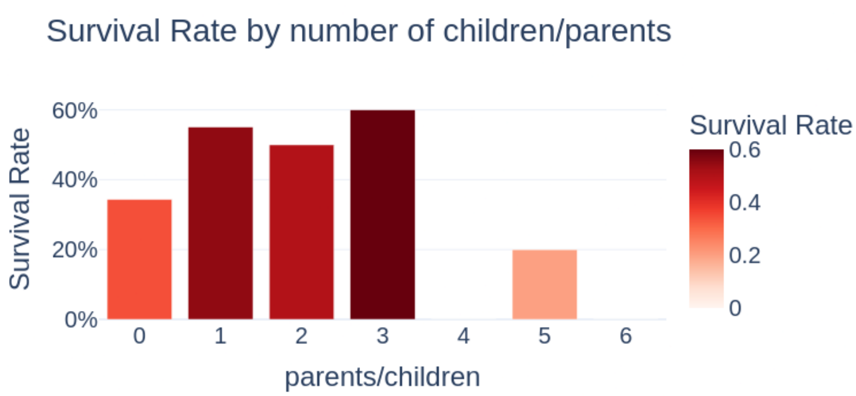

```
fig
```



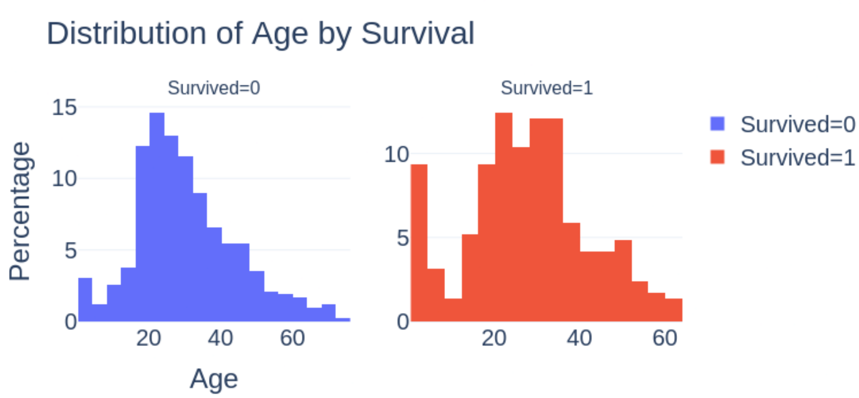

```
fig
```




```
fig
```




```
fig
```



4.4.2 Based on the Age vs Survived Histograms:

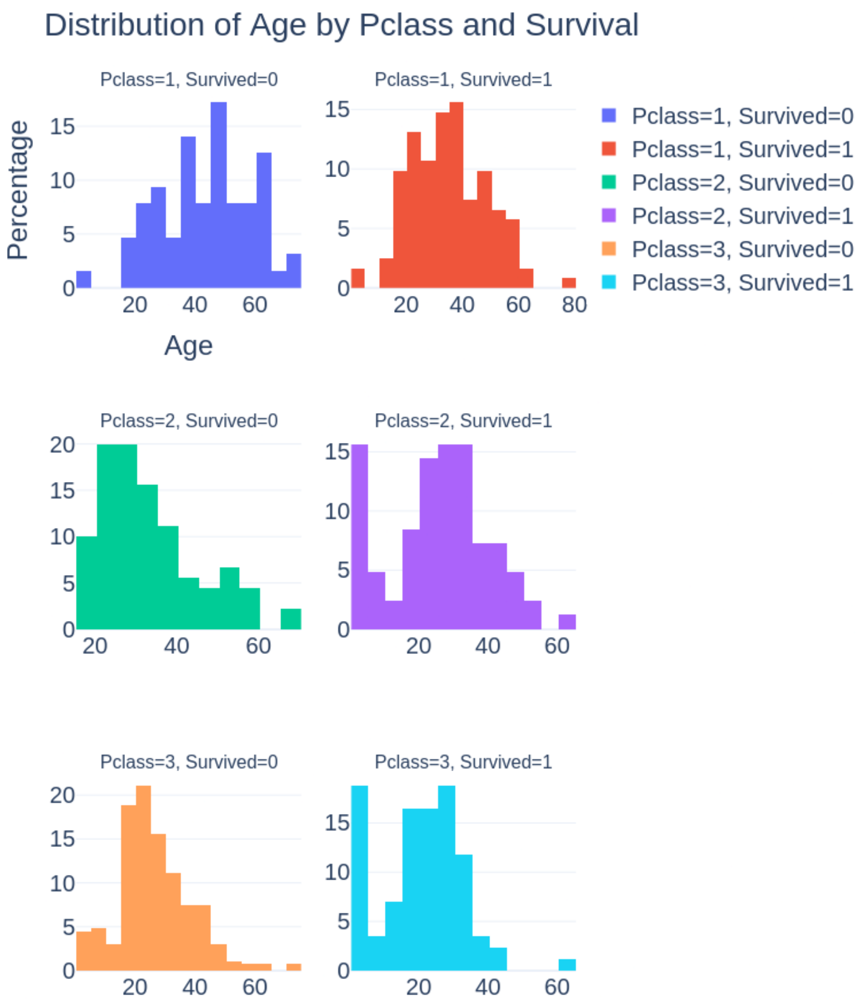
Observations

- 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).

```
fig
```



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.

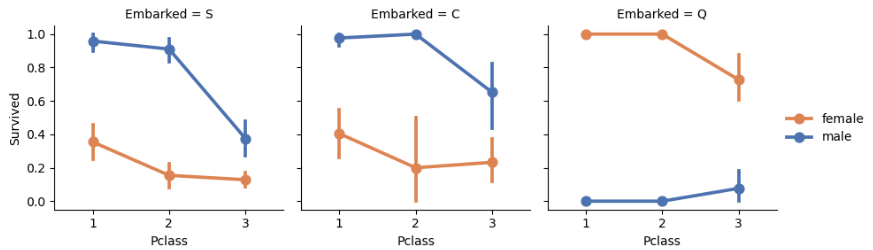
Decisions

- Consider Pclass for model training.

```
grid = sns.FacetGrid(passenger_df_train, col='Embarked')
grid.map(sns.pointplot, 'Pclass', 'Survived', 'Sex', palette='deep')
grid.add_legend()
```

```
/home/michael/anaconda3/envs/Intelligent/lib/python3.11/site -packages/seaborn/axisgrid.py:718: UserWarn

Using the pointplot function without specifying `order` is likely to produce an incorrect plot.

/home/michael/anaconda3/envs/Intelligent/lib/python3.11/site -packages/seaborn/axisgrid.py:723: UserWarn

Using the pointplot function without specifying `hue_order` is likely to produce an incorrect plot.
```

```
<seaborn.axisgrid.FacetGrid at 0x7ce50dcb44d0>
```



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).

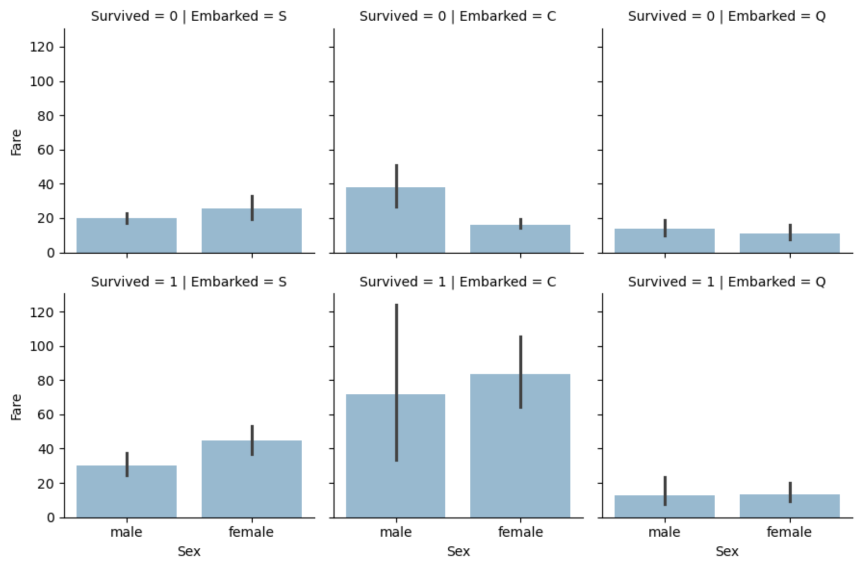
Decisions

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

```
grid = sns.FacetGrid(passenger_df_train, col ='Embarked', row ='Survived')
grid.map(sns.barplot, 'Sex', 'Fare', alpha=.5)
grid.add_legend()
```

```
/home/michael/anaconda3/envs/Intelligent/lib/python3.11/site -packages/seaborn/axisgrid.py:718: UserWarn

Using the barplot function without specifying `order` is likely to produce an incorrect plot.
```

```
<seaborn.axisgrid.FacetGrid at 0x7ce50da5c4d0>
```



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

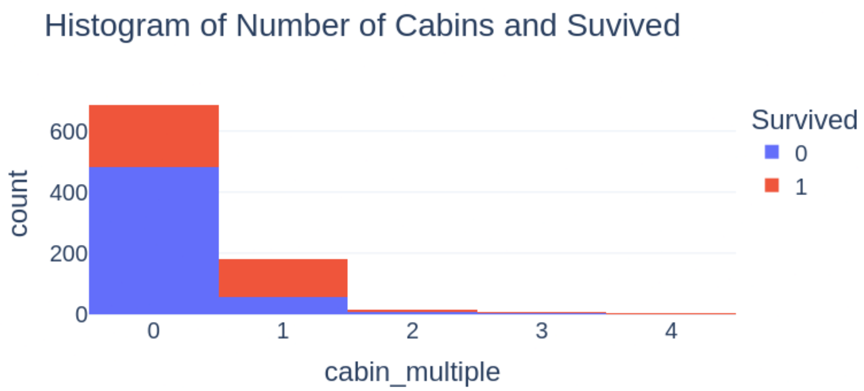
Observations

- 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 o
```



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')
```

cabin_deck	A	B	C	D	E	F	G	T	n
Survived									
0	8.0	12.0	24.0	8.0	8.0	5.0	2.0	1.0	481.0
1	7.0	35.0	35.0	25.0	24.0	8.0	2.0	NaN	206.0

4.4.6 Based on the Cabins Pivot Tables:

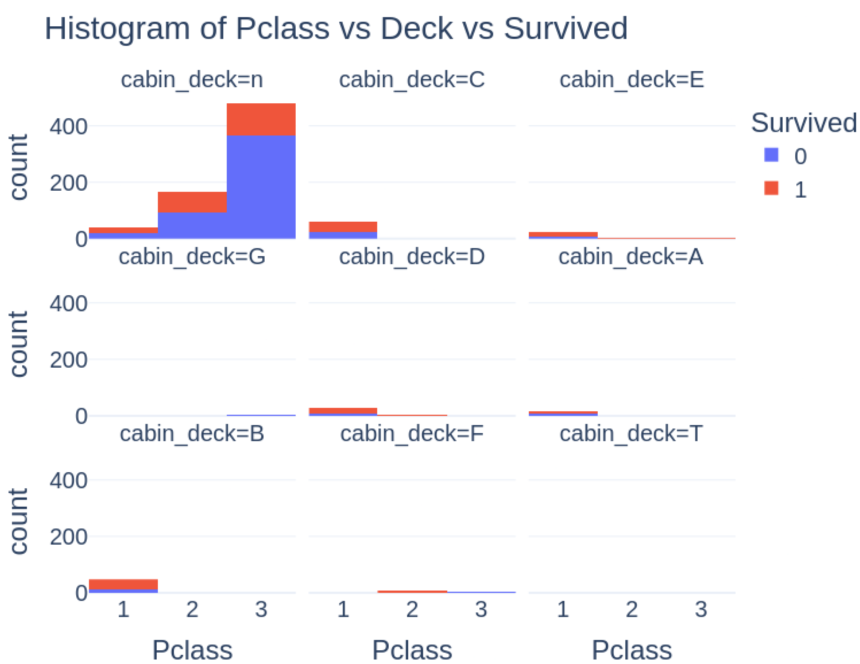
Observations

- 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

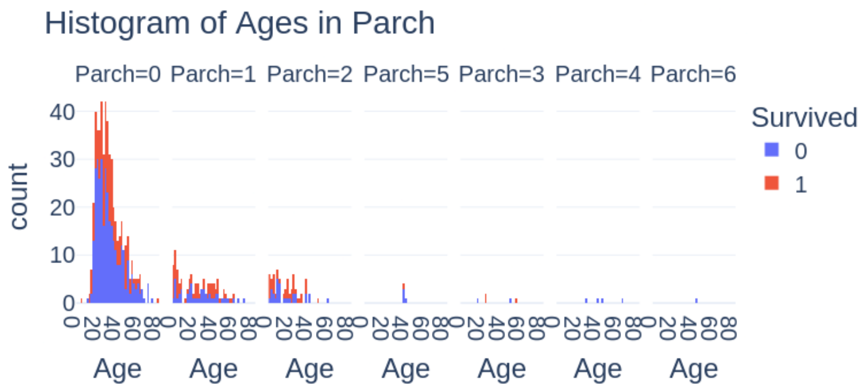
- Consider Seperating the cabin feature into only cabin letters.
- Consider creating a number of Cabins feature.

```
fig
```



Ages vs ParCh

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



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.

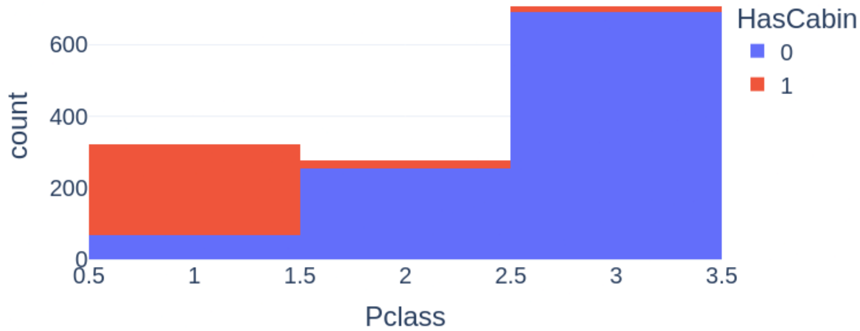
```
passenger_df.loc[passenger_df.Survived.isna(),"Survived"] = -1
passenger_df[original].vu(10)
```

Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
-1	3	Niklasson, Mr. Samuel	male	28.00	0	0	363611	8.05	nan	S
-1	1	Borebank, Mr. John James	male	42.00	0	0	110489	26.55	D22	S
-1	3	Pedersen, Mr. Olaf	male	nan	0	0	345498	7.78	nan	S
0	2	Meyer, Mr. August	male	39.00	0	0	248723	13.00	nan	S
-1	3	McCarthy, Miss. Catherine Katie""	female	nan	0	0	383123	7.75	nan	Q
0	3	Zabour, Miss. Thamine	female	nan	1	0	2665	14.45	nan	C
-1	3	McNeill, Miss. Bridget	female	nan	0	0	370368	7.75	nan	Q
1	2	Leitch, Miss. Jessie Wills	female	nan	0	0	248727	33.00	nan	S
-1	3	Johnston, Mrs. Andrew G (Elizabeth Lily" Watson)"	female	nan	1	2	W./C. 6607	23.45	nan	S
1	3	Moubarek, Master. Gerios	male	nan	1	1	2661	15.25	nan	C

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 most empty of all. Only 22% of the passengers have it, with majority of them being in first class. Upon examining the Titanic deck plans, we've seen 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.

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 cabins. Since the families prefered to occupy cabins close to each other, our splitting is good enough. Those are only in 1st class. sdfasdf

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	25.00	3	0.00	LINE	LINE	1	LINE1
Johnson, Mr. Alfred	male	49.00	3	0.00	LINE	LINE	1	LINE1
Leonard, Mr. Lionel	male	36.00	3	0.00	LINE	LINE	1	LINE1
Johnson, Mr. William Cahoone Jr	male	19.00	3	0.00	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)
```

	tPref	tickets
0		957
1	PC	92
2	C.A.	46
3	SOTON/O.Q.	16
4	W./C.	14
5	STON/O 2.	14
6	CA.	12
7	A/5	12
8	SC/PARIS	11
9	CA	10
10	A/5.	10
11	F.C.C.	9
12	SOTON/OQ	8

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`

```
q = """
select Pclass, tPrefTr, count(*) as count
from passenger_df
group by Pclass, tPrefTr
order by Pclass, count desc
```

```
limit 13
"""
ps.sqldf(q)
```

	Pclass	tPrefTr	count
0	1		224
1	1	PC	92
2	1	WEP	4
3	1	FC	3
4	2		184
5	2	CA	34
6	2	SC	26
7	2	FCC	9
8	2	SOC	8
9	2	WC	5
10	2	SOPP	4
11	2	SWPP	2
12	2	PPP	2

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

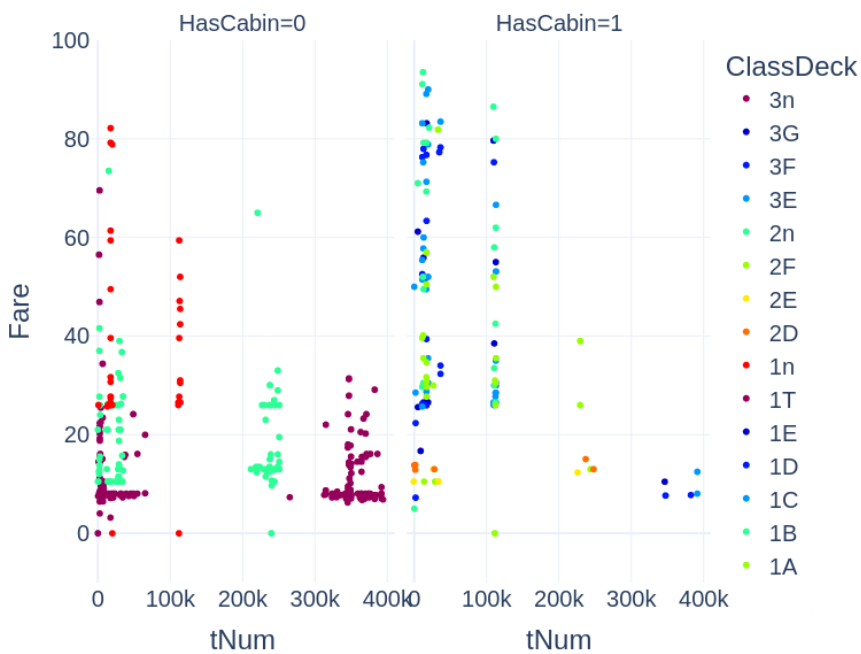
- Ticket number
- Ticket prefix
- Fare
- Class
- Deck
- Cabin number

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.

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_col="HasCabin", \
          color_discrete_sequence= px.colors.sequential.Rainbow, category_orders=co ,
          height=600, range_x= [ -10000, 4.1e5], range_y=[ -10,100]) #, facet_col= "Survived")
fig
```



4.5.3 Age

```
print( int(passenger_df[passenger_df.Age.isna()].shape[0] / passenger_df.shape[0] *100) , "%")
```

```
20 %
```

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.head(20)
```

	word	count
1153	Mr.	517
1122	Miss.	182
1154	Mrs.	125
1633	William	62
858	John	44
1068	Master.	40
758	Henry	33
828	James	24
382	Charles	23
645	George	22
1523	Thomas	21
522	Edward	18
869	Joseph	16
625	Frederick	15
850	Johan	15
226	Arthur	13
1343	Richard	13
1401	Samuel	13
1065	Mary	13
177	Alfred	12

```
ttls = tokens[tokens.word.str.contains('\\.', )]
comment = "% of tokens in in the whole Name column are the following tokens:"
print(int(ttls["count"].sum()/ tokens["count"].sum() *100), comment, sep="" )
ttls
```

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

	word	count
1153	Mr.	517
1122	Miss.	182
1154	Mrs.	125
1068	Master.	40
500	Dr.	7
1340	Rev.	6
412	Col.	2
1127	Mlle.	2
1025	Major.	2
1155	Ms.	1
1128	Mme.	1
1442	Sir.	1
353	Capt.	1
490	Don.	1
432	Countess.	1
933	L.	1
934	Lady.	1
862	Jonkheer.	1

We see that people 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 :) .

```
passenger_df.drop("lName Title fName sName".split(" "),axis=1, inplace=True, errors='ignore')
```

```
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)
cols = ['Pclass', 'Name', 'Sex', 'Age', 'lName', 'Title', 'fName', 'sName']

passenger_df[cols].vu(10)
```

Pclass	Name	Sex	Age	lName	Title	fName	sName
3	Niklasson, Mr. Samuel	male	28.00	Niklasson	Mr	Samuel	nan
1	Borebank, Mr. John James	male	42.00	Borebank	Mr	John James	nan
3	Pedersen, Mr. Olaf	male	nan	Pedersen	Mr	Olaf	nan
2	Meyer, Mr. August	male	39.00	Meyer	Mr	August	nan
3	McCarthy, Miss. Catherine Katie""	female	nan	McCarthy	Miss	Catherine Katie""	nan
3	Zabour, Miss. Thamine	female	nan	Zabour	Miss	Thamine	nan
3	McNeill, Miss. Bridget	female	nan	McNeill	Miss	Bridget	nan
2	Leitch, Miss. Jessie Wills	female	nan	Leitch	Miss	Jessie Wills	nan
3	Johnston, Mrs. Andrew G (Elizabeth Lily" Watson)"	female	nan	Johnston	Mrs	Andrew G	Elizabeth Lily" Watson
3	Moubarek, Master. Gerios	male	nan	Moubarek	Master	Gerios	nan

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)
```

	Title	cnt
0	Mr	757
1	Miss	260
2	Mrs	197
3	Master	61
4	Rev	8
5	Dr	8
6	Col	4
7	Ms	2
8	Mlle	2
9	Major	2
10	the Countess	1
11	Sir	1
12	Mme	1
13	Lady	1
14	Jonkheer	1
15	Dona	1
16	Don	1
17	Capt	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

This may be used to estimate Age where it's unknown

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)
```

	Title	count
0	Mr	759
1	Miss	264
2	Mrs	201
3	Master	61
4	Rare	24

Second name Let's explore the second name:

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

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

Fare	Sex	Age	Pclass	Title	fName	lName	sName
26.00	female	44.00	2	Mrs	Ernest Courtenay	Carter	Lilian Hughes
27.72	female	24.00	2	Mrs	Sebastiano	del Carlo	Argenia Genovesi
55.90	female	39.00	1	Mrs	William Baird	Silvey	Alice Munger
53.10	female	18.00	1	Mrs	Daniel Warner	Marvin	Mary Graham Carmichael Farquarson
23.00	female	34.00	2	Mrs	John T	Doling	Ada Julia Bone
86.50	female	33.00	1	Mrs	of	Rothes	Lucy Noel Martha Dyer-Edwards
26.55	male	42.00	1	Mr	Erik Gustaf	Lindeberg-Lind	Mr Edward Lingrey"
21.00	female	31.00	2	Mrs	John James	Ware	Florence Louise Long
90.00	female	35.00	1	Mrs	Frederick Maxfield	Hoyt	Jane Anne Forby
29.12	female	39.00	3	Mrs	William	Rice	Margaret Norton

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.head(20)
```

	word	count
407	nan	1088
272	Mary	13
117	Elizabeth	12
264	Maria	8
33	Anna	7
106	E	6
36	Annie	5
75	Catherine	5
16	Ada	5
261	Margaret	5
140	Florence	5
24	Alice	4
108	Edith	4
7	"Mr	4
127	Emma	4
270	Martha	4
245	Louise	4
188	Hughes	3
177	Helen	3
80	Charlotte	3

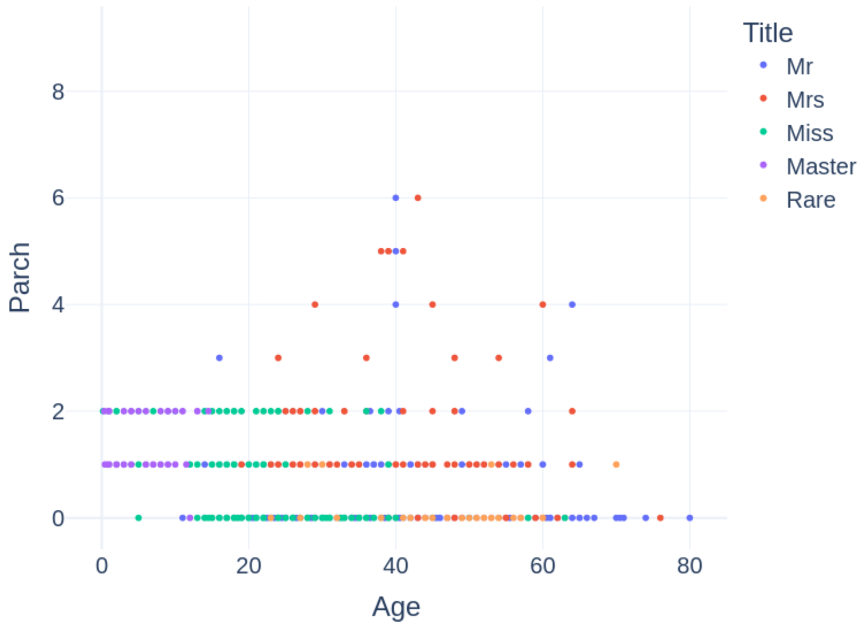
```
passenger_df.loc[passenger_df.sName.astype(str).str.contains("Mr"), show].vu(9)
```

Fare	Sex	Age	Pclass	Title	fName	lName	sName
26.55	male	42.00	1	Mr	Erik Gustaf	Lindeberg-Lind	Mr Edward Lingrey"
13.00	female	24.00	2	Miss	Henriette	Yrois	"Mrs Harbeck"
26.00	male	39.00	2	Mr	Henry Samuel	Morley	"Mr Henry Marshall"
26.55	male	45.00	1	Mr	Charles Hallace	Romaine	"Mr C Rolmane"
79.20	male	46.00	1	Mr	George	Rosenshine	Mr George Thorne"
26.00	female	19.00	2	Miss	Kate Florence	Phillips	"Mrs Kate Louise Phillips Marshall"
26.55	male	35.00	1	Mr	Harry	Homer	"Mr E Haven"
56.93	male	49.00	1	Mr	Cosmo Edmund	Duff Gordon	"Mr Morgan"
49.50	female	24.00	1	Miss	Berthe Antonine	Mayne	"Mrs de Villiers"

Seems like majority of these contain full/maiden names of women travelling with ticket under their husbands' names.
Maybe this property could be used for Age estimation...

4.5.5 Family composition data

```
px.scatter(passenger_df, x = "Age", y = "Parch", color="Title", hover_data=["Fare"], \
          category_orders=co, height= 600 )
```



We clearly see that the titles Master and Miss, 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.

But first, there is one person without Fare. We'll just put a number manually there.

```
passenger_df.loc[passenger_df.Fare.isna(), "Fare"] = 7.2500
```

Prepare dataset for training and imputation

```
Cx = ["Fare", "Sex", "SibSp", "Parch", "Pclass", "Title"]
Cy = "Age"
categorical_columns = ["Sex", "Title"]

# Convert categorical variables into dummy variables using one -hot encoding
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
7.78	0	0	3	False	True	False	False	True	False	False
13.00	0	0	2	False	True	False	False	True	False	False
7.75	0	0	3	True	False	False	True	False	False	False
14.45	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
14.46	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
7.90	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
15.90	1	1	3	False	True	True	False	False	False	False
79.20	0	0	1	False	True	False	False	True	False	False
7.85	0	0	3	True	False	False	True	False	False	False
0.00	0	0	1	False	True	False	False	True	False	False
26.00	1	0	2	True	False	False	False	False	True	False
8.66	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 with maximum 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

```
#deleteme

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 2 candidates, totalling 20 fits
Elapsed Time: 00:00:01
====================
Best Score: 0.434
Best Parameters: {'max_depth': 17, 'max_features': 7, 'max_leaf_nodes': 20, 'n_estimators': 357}
```

```
abr = AdaBoostRegressor()
abr.get_params()

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 2 candidates, totalling 20 fits
Elapsed Time: 00:00:00
====================
Best Score: -2.552
Best Parameters: {'learning_rate': 6, 'n_estimators': 344}
```

GradientBoosting

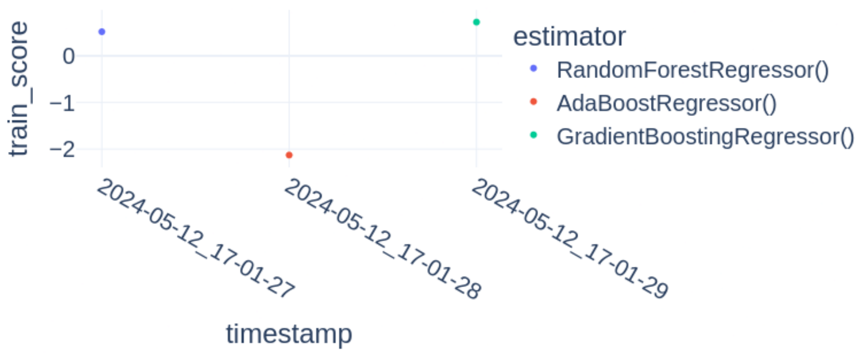
```
gbr = GradientBoostingRegressor()
param_grid ={'max_depth': st.randint(6, 20),
             'n_estimators': st.randint(10, 500),
             'max_features': np.arange(5,12),
             'max_leaf_nodes': st.randint(6, 30)}


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 2 candidates, totalling 20 fits
Elapsed Time: 00:00:00
====================
Best Score: 0.360
Best Parameters: {'max_depth': 14, 'max_features': 10, 'max_leaf_nodes': 10, 'n_estimators': 287}
```

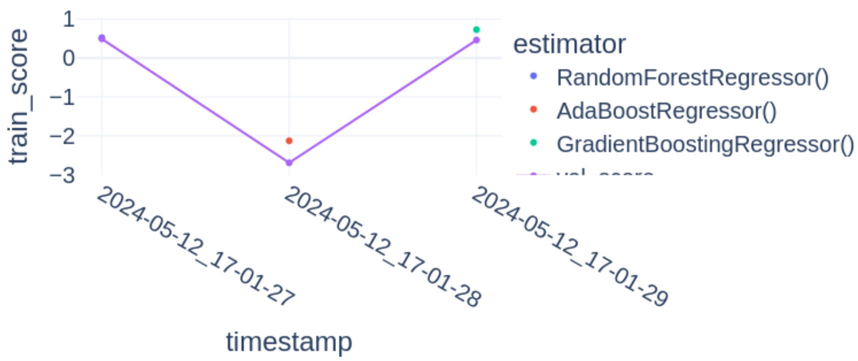
```
CV_df = pd.DataFrame(CV_Runs)
CV_df[['elapsed', 'estimator', 'best_params', 'train_score',
       'val_score', 'cv', 'n_iter']]
```

	elapsed	estimator	best_params	train_score	val_score	cv	n_iter
0	00:00:01	RandomForestRegressor()	{'max_depth': 17, 'max_features': 7, 'max_leaf...	0.52	0.49	10	2
1	00:00:00	AdaBoostRegressor()	{'learning_rate': 6, 'n_estimators': 344}	-2.13	-2.70	10	2
2	00:00:00	GradientBoostingRegressor()	{'max_depth': 14, 'max_features': 10, 'max_lea...	0.73	0.46	10	2

```
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)

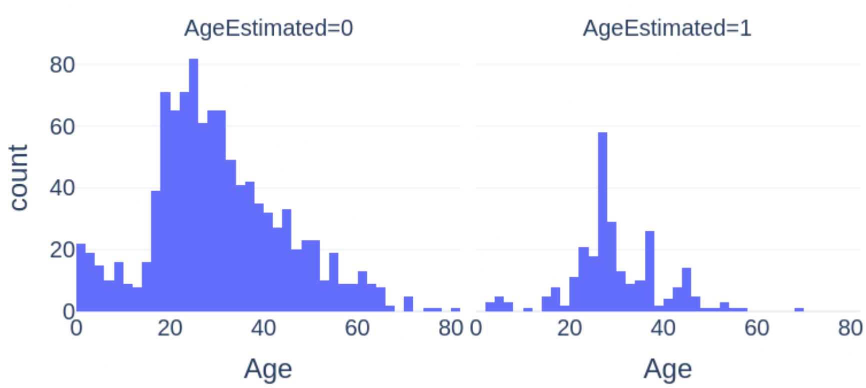
```
best_params = {'max_depth': 13, 'max_features': 5, 'max_leaf_nodes': 29, 'n_estimators': 435}
```

```
rfc = GradientBoostingRegressor( **best_params)
rfc.fit(X,y)
y_hat = rfc.predict(Ximp)
y_hat = pd.Series(rfc.predict(Ximp), index=Ximp.index)
y_hat.head(7)
```

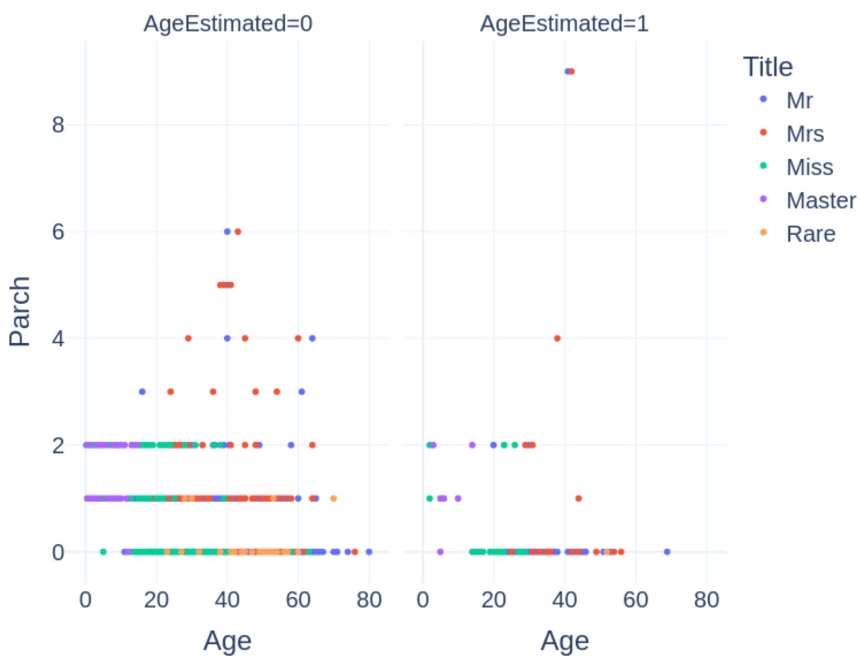
```
PassengerId
6     22.55
18    32.22
20    44.75
27    27.43
29    21.71
30    27.89
32    44.28
dtype: float64
```

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, x = "Age", y = "Parch", color="Title", facet_col= "AgeEstimated",
           hover_data=["SibSp", "Fare", "Name"],
           category_orders=co, height= 600)
```



It seems that imputation went quite well.

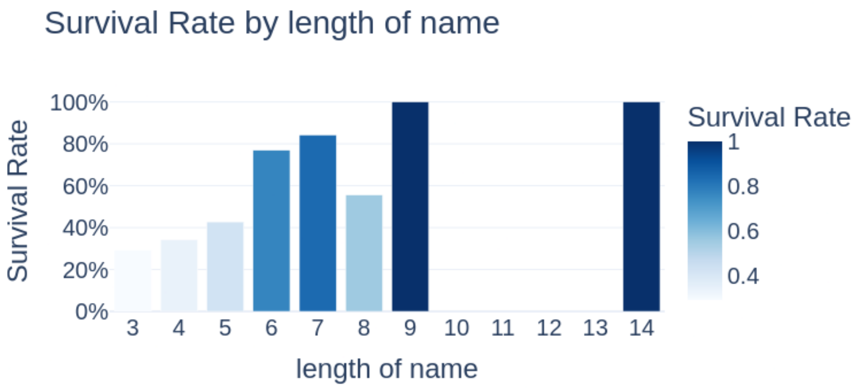
4.8 Construct More features

The length of the Name

```
passenger_df['Words_Count'] = passenger_df['Name'].apply(lambda x: len(x.split()))
print(passenger_df.Words_Count.value_counts())
```

```
Words_Count
4      558
3      449
5      144
6       81
7       59
8       16
14       1
9        1
Name: count, dtype: int64
```

```
fig
```



Create new features cabin_multiple and cabin_deck that shows number of cabins each passenger had.
Create new feature FamilySize as a combination of SibSp and Parch
Create new feature IsAlone from FamilySize

```
side_by_side(df1,df2)
```

	FamilySize	Survived
3	4	0.72
2	3	0.58
1	2	0.55
6	7	0.33
0	1	0.30
4	5	0.20
5	6	0.14
7	8	0.00
8	11	0.00

	IsAlone	Survived
0	0	0.51
1	1	0.30

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

```
df
```

	CategoricalFare	Survived
0	(-0.001, 7.775]	0.21
1	(7.775, 8.662]	0.19
2	(8.662, 14.454]	0.37
3	(14.454, 26.0]	0.44
4	(26.0, 53.1]	0.44
5	(53.1, 512.329]	0.70

Create a New feature CategoricalAge

```
df.head(8)
```

	CategoricalAge	Survived
0	(0.169, 18.5]	0.50
1	(18.5, 24.0]	0.36
2	(24.0, 28.0]	0.32
3	(28.0, 34.0]	0.37
4	(34.0, 43.0]	0.38
5	(43.0, 80.0]	0.37

4.9 Mapping Categorical and High Ordinal Features

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

	Age*Class	Age	Pclass
PassengerId			
1	3.0	1.0	3
2	4.0	4.0	1
3	6.0	2.0	3
4	4.0	4.0	1
5	12.0	4.0	3
6	3.0	1.0	3
7	5.0	5.0	1
8	0.0	0.0	3
9	6.0	2.0	3
10	0.0	0.0	2

4.10 Feature Selection

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

Survived	Pclass	Sex	Age	Parch	Fare	Embarked	Words_Count	Cabin_multiple	Cabin_deck	FamilySize	IsAlone	Age*Class
-1	3	1	3.00	0	1	0	3	0	0	1	1	9.00
-1	1	1	5.00	0	4	0	4	1	4	1	1	5.00
-1	3	1	2.00	0	1	0	3	0	0	1	1	6.00
0	2	1	4.00	0	2	0	3	0	0	1	1	8.00
-1	3	0	2.00	0	1	2	4	0	0	1	1	6.00
0	3	0	0.00	0	2	1	3	0	0	2	0	0.00
-1	3	0	2.00	0	1	2	3	0	0	1	1	6.00
1	2	0	1.00	0	5	0	4	0	0	1	1	2.00
-1	3	0	3.00	2	3	0	7	0	0	4	0	9.00
1	3	1	0.00	1	2	1	3	0	0	3	0	0.00

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

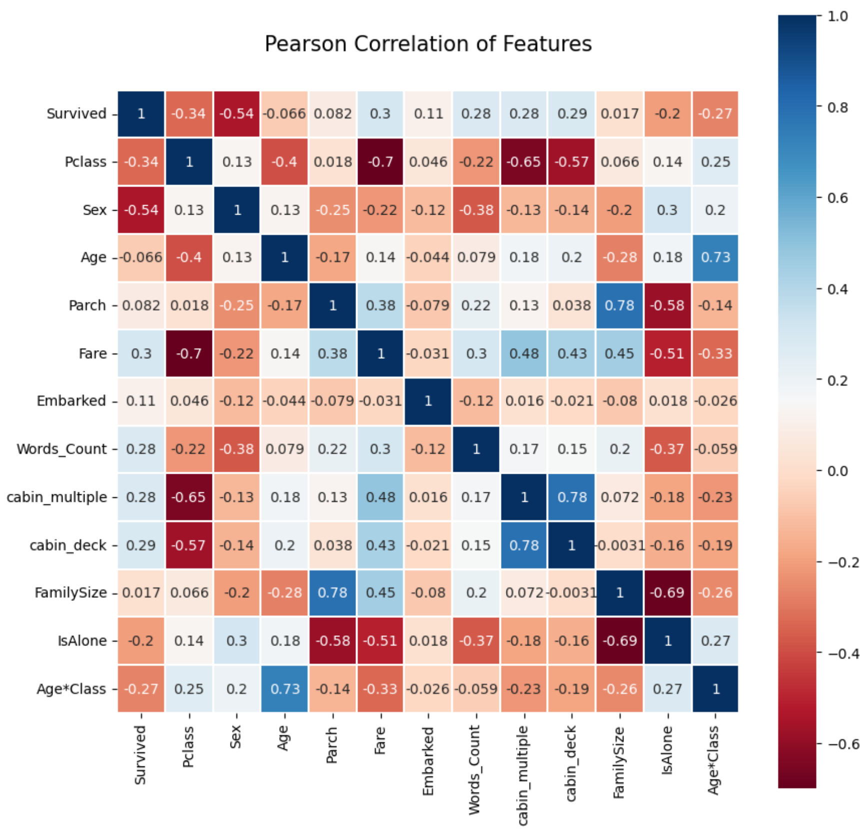
Survived	Pclass	Sex	Age	Parch	Fare	Embarked	Words_Count	Cabin_multiple	Cabin_deck	FamilySize	IsAlone	Age*Class
1	3	1	0.00	1	2	1	6	0	0	3	0	0.00
0	2	1	3.00	0	1	0	5	0	0	1	1	6.00
0	3	1	1.00	0	1	0	4	0	0	1	1	3.00
1	2	0	0.00	1	5	0	5	0	0	2	0	0.00
1	3	0	0.00	0	1	1	3	0	0	2	0	0.00
1	1	0	2.00	0	5	0	4	0	0	1	1	2.00
1	3	0	2.00	0	1	2	6	0	0	1	1	6.00
0	3	1	0.00	0	3	0	5	0	0	3	0	0.00
1	3	0	0.00	0	1	2	4	0	0	1	1	0.00
1	1	0	1.00	2	4	0	4	1	4	3	0	1.00

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

	Pclass	Sex	Age	Parch	Fare	Embarked	Words_Count	Cabin_multiple	Cabin_deck	FamilySize	IsAlone	Age*Class
PassengerId												
892	3	1	4.0	0	1	2	3	0	0	1	1	12.0
893	3	0	5.0	0	1	0	5	0	0	2	0	15.0
894	2	1	5.0	0	1	2	4	0	0	1	1	10.0
895	3	1	2.0	0	1	0	3	0	0	1	1	6.0
896	3	0	1.0	1	2	0	6	0	0	3	0	3.0
897	3	1	0.0	0	1	0	4	0	0	1	1	0.0
898	3	0	3.0	0	1	2	3	0	0	1	1	9.0
899	2	1	2.0	1	4	0	4	0	0	3	0	4.0
900	3	0	1.0	0	1	1	6	0	0	1	1	3.0
901	3	1	1.0	0	4	0	4	0	0	3	0	3.0

```
passenger_df_corr=passenger_df_train.astype(float).corr()
```

```
colormap=plt.cm.RdBu
plt.figure(figsize=(10,10))
plt.title('Pearson Correlation of Features', y=1.05, size=15)
sns.heatmap(passenger_df_corr,linewidths=0.1,vmax=1.0,
            square=True, cmap=colormap, linecolor='white', annot=True)
plt.show()
```



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

```
X = passenger_df_train.drop('Survived', axis=1)
y = passenger_df_train['Survived']

# Splitting data into train and validation sets
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

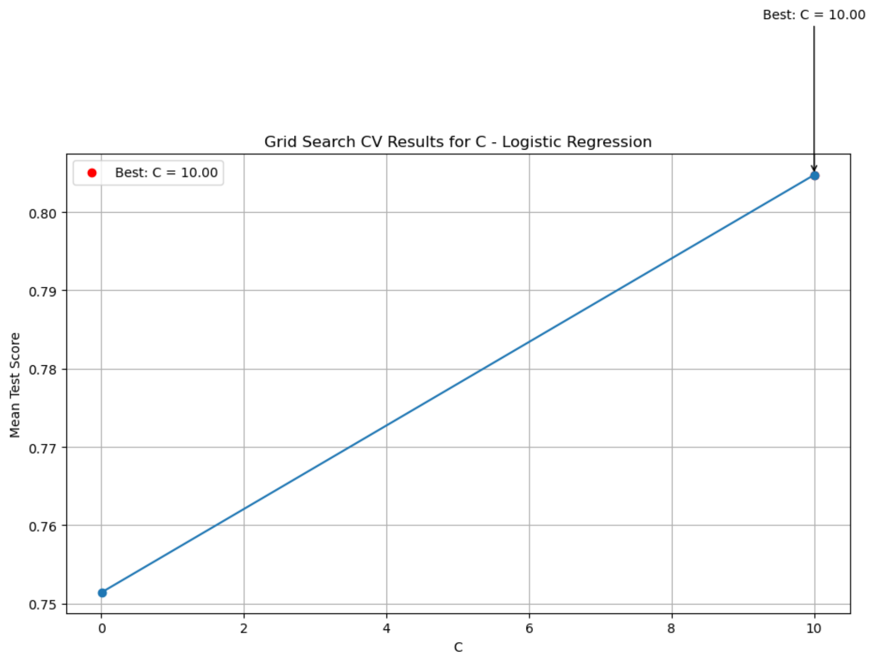
5.2.3 Confusion Matrix for Best Model

5.3 Classical models

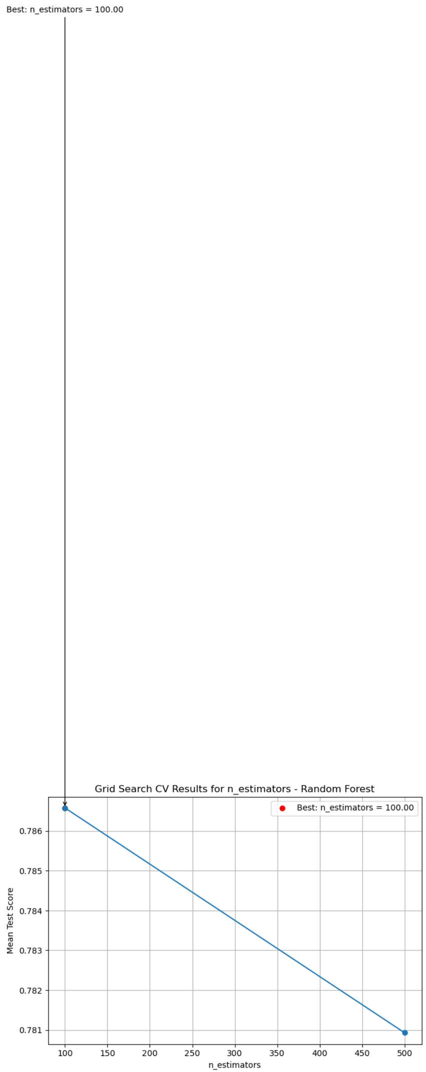
Initialize models with Hyperparameters

```
# Define models
```

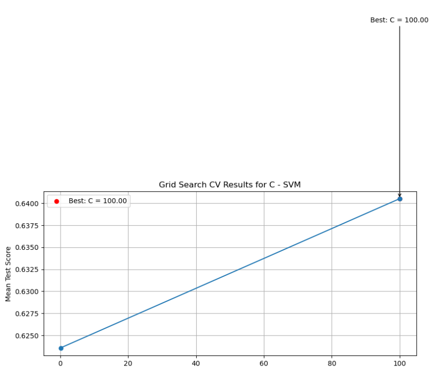
```
models = {
    'Logistic Regression': LogisticRegression(),
    'Random Forest': RandomForestClassifier(),
    'SVM': SVC(),
    #'Lasso': Lasso(),
    #'Ridge': Ridge(),
    'Gradient Boosting': GradientBoostingClassifier(),
    'Decision Tree': DecisionTreeClassifier()
}

# 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, 
    'Decision Tree': {'max_depth': [None, 10, 20, 30, 50, 100]}
    #'Lasso': {'alpha': [0.01, 0.1, 1]},
    #'Ridge': {'alpha': [0.01, 0.1, 1]},
}



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

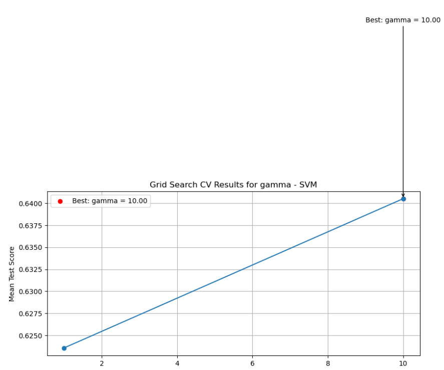


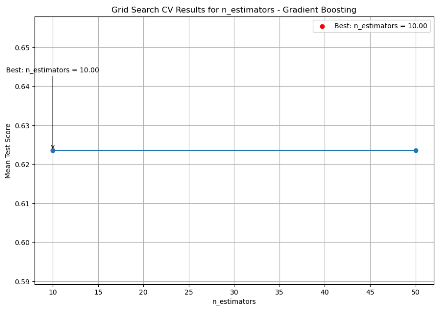

```
/tmp/ipykernel_3272649/2637939331.py:20: UserWarning:

Tight layout not applied. The bottom and top margins cannot be made large enough to accommodate all axes
```

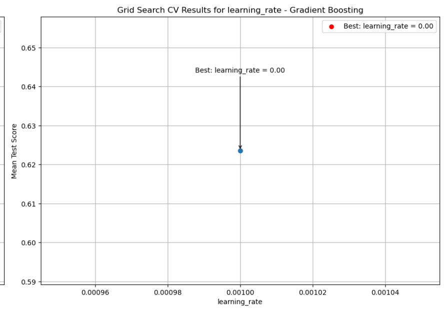

Best: n_estimators = 100.00
Grid Search CV Results for n_estimators - Random Forest
Best: n_estimators = 100.00
Mean Test Score
n_estimators

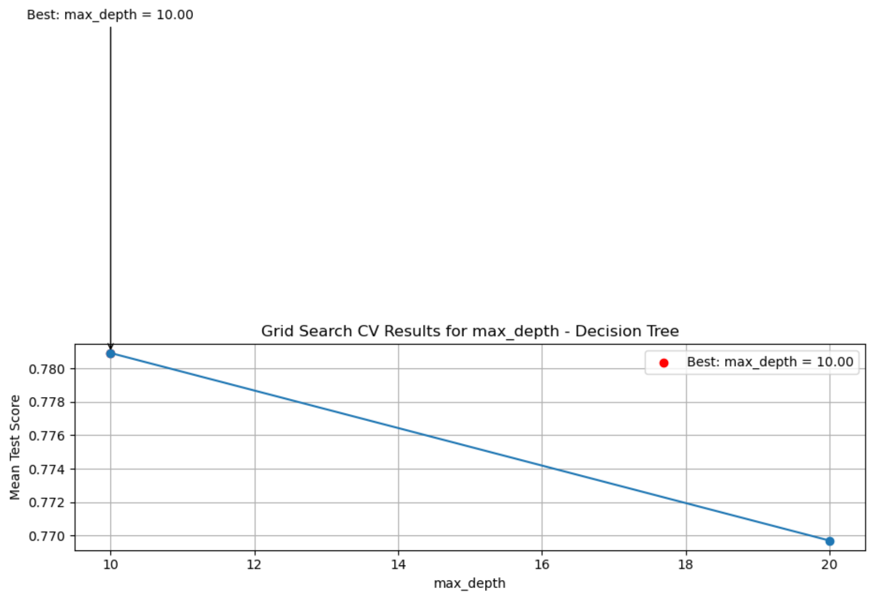

Best: max_depth = 10.00
Grid Search CV Results for max_depth - Random Forest
Best: max_depth = 10.00
Mean Test Score
max_depth


Best: C = 100.00
Grid Search CV Results for C - SVM
Best: C = 100.00
Mean Test Score
C


Best: gamma = 10.00
Grid Search CV Results for gamma - SVM
Best: gamma = 10.00
Mean Test Score
gamma







	Model	Best Parameters	Best Score (CV)	Validation Score
0	Logistic Regression	{'C': 10}	0.80	0.81
1	Random Forest	{'n_estimators': 100, 'max_depth': 10}	0.79	0.83
2	SVM	{'gamma': 10, 'C': 100}	0.64	0.61
3	Gradient Boosting	{'n_estimators': 10, 'learning_rate': 0.001}	0.62	0.59
4	Decision Tree	{'max_depth': 10}	0.78	0.79

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("Best Validation Score: {:.4f}" .format(best_validation_score))
best_model
```

```
Best Model:  Random Forest
Best Validation Score: 0.8324
```

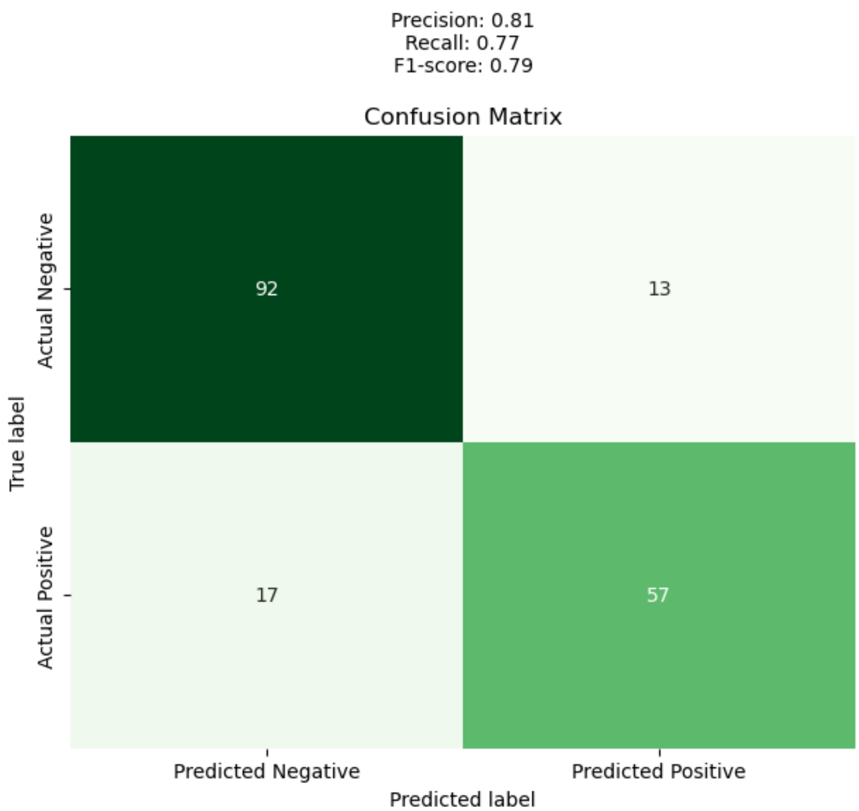
```
RandomForestClassifier(max_depth=10)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

[x]RandomForestClassifier

```
RandomForestClassifier(max_depth=10)
```

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




```
Train Accuracy: 0.9256
Validation Accuracy: 0.8324
```

```
(0.925561797752809,
 0.8324022346368715,
 0.8142857142857143,
 0.7702702702702703,
 0.7916666666666666)
```

*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.head(10)
```

	Pclass	Sex	Age	Parch	Fare	Embarked	Words_Count	Cabin_multiple	cabin_deck	FamilySize	IsAlone	Age*Class
PassengerId												
892	3	1	4.0	0	1	2	3	0	0	1	1	12.0
893	3	0	5.0	0	1	0	5	0	0	2	0	15.0
894	2	1	5.0	0	1	2	4	0	0	1	1	10.0
895	3	1	2.0	0	1	0	3	0	0	1	1	6.0
896	3	0	1.0	1	2	0	6	0	0	3	0	3.0
897	3	1	0.0	0	1	0	4	0	0	1	1	0.0
898	3	0	3.0	0	1	2	3	0	0	1	1	9.0
899	2	1	2.0	1	4	0	4	0	0	3	0	4.0
900	3	0	1.0	0	1	1	6	0	0	1	1	3.0
901	3	1	1.0	0	4	0	4	0	0	3	0	3.0

```
predictions = best_model.predict(passenger_df_test)
#predictions = best_svm_classifier.predict(passenger_df_test)
#predictions = best_gb_classifier.predict(passenger_df_test)
#predictions = lasso_model.predict(passenger_df_test)
#predictions = ridge_model.predict(passenger_df_test)
predictions = predictions.astype(int)

# Convert predictions into binary output
#binary_predictions = (predictions >= 0.5).astype(int)

output = pd.DataFrame({'PassengerId': passenger_df_test.index, 'Survived': predictions})
#output = pd.DataFrame({'PassengerId': passenger_df_test.index, 'Survived': binary_predictions})
output.to_csv('submission.csv', index=False)
print("Your submission was successfully saved!")
```

```
Your submission was successfully saved!
```

```
output.sample(10)
```

	PassengerId	Survived
314	1206	1
8	900	1
320	1212	0
354	1246	0
27	919	0
125	1017	1
17	909	0
94	986	0
358	1250	0
209	1101	0