Titanic - Machine Learning from Disaster

Assignment 1 submission

Michael Berger[1]

Monday 19th February, 2024

Keywords

```
#!conda install -y matplotlib numpy  seaborn  plotly scikit -learn  pandasql  scipy
```

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

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

[1] Correspondence to: vo1kod4v@gmail.com

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

```
#delete me
```

```
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(pass
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.

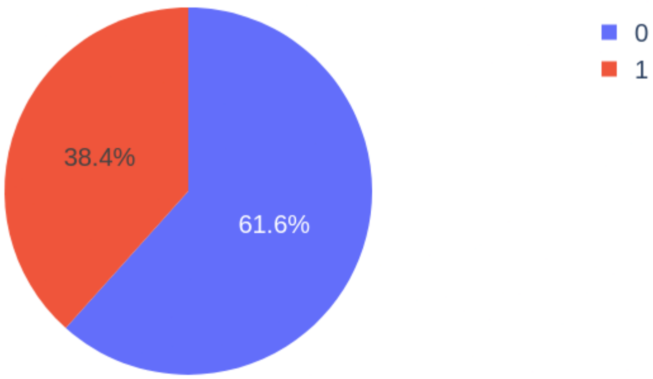
What are the data types for various features?

Helping us during converting goal.

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

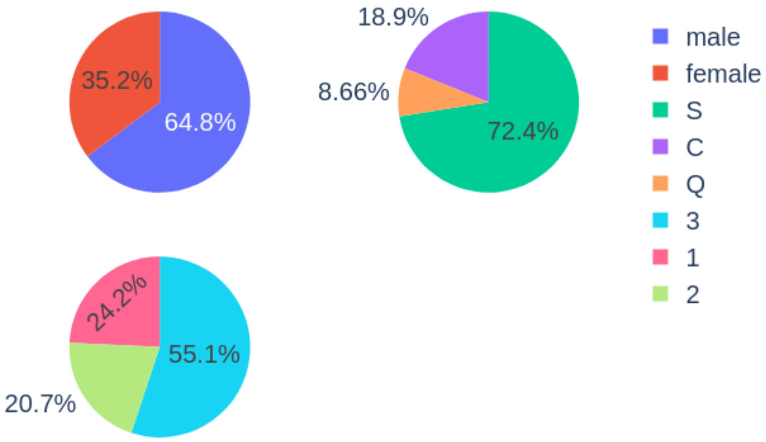
4.1.2 Amount of Survivors

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



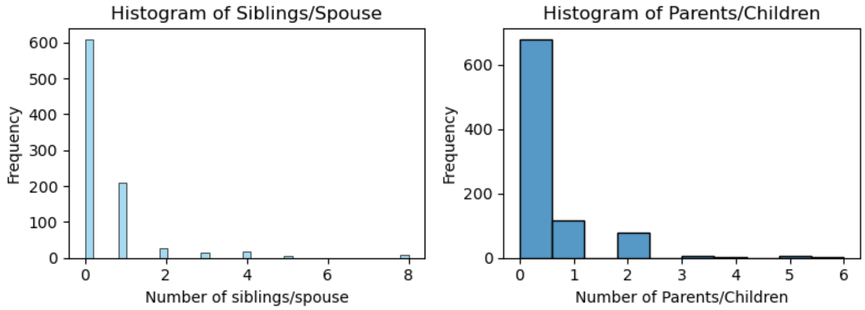
4.1.3 Pie Charts for Embark, Sex and Pclass

```
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.

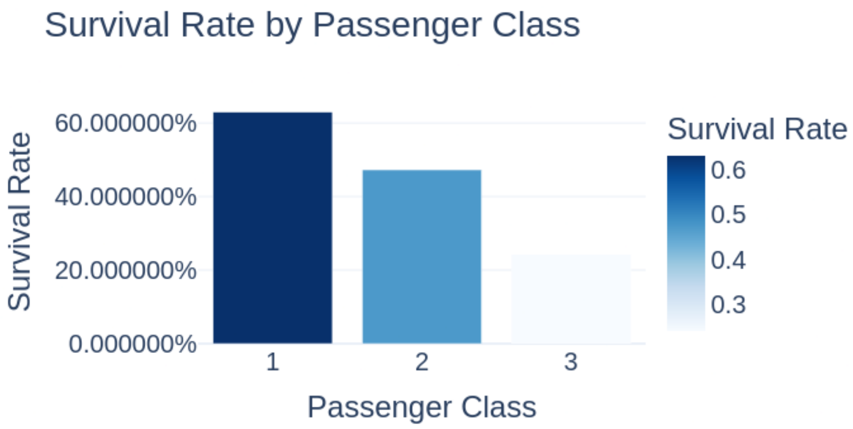
4.4 Data Exploration

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.

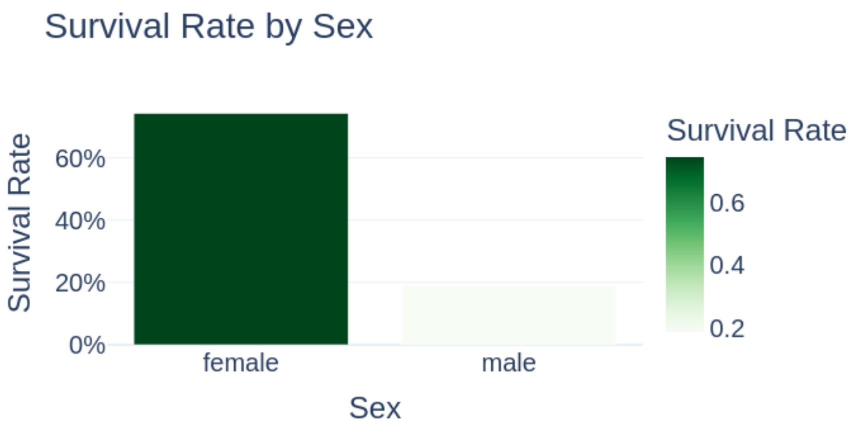
- **Pclass** We observe significant correlation (>0.5) among Pclass=1 and Survived (classifying #3). We decide to include this feature in our model.
- **Sex** We confirm the observation during problem definition that Sex=female had very high survival rate at 74% (classifying #1).
- **SibSp and Parch**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

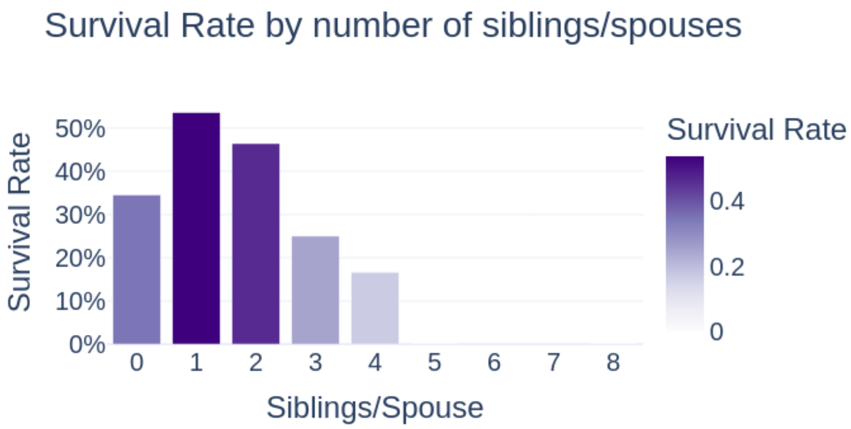
```
fig.show()
```



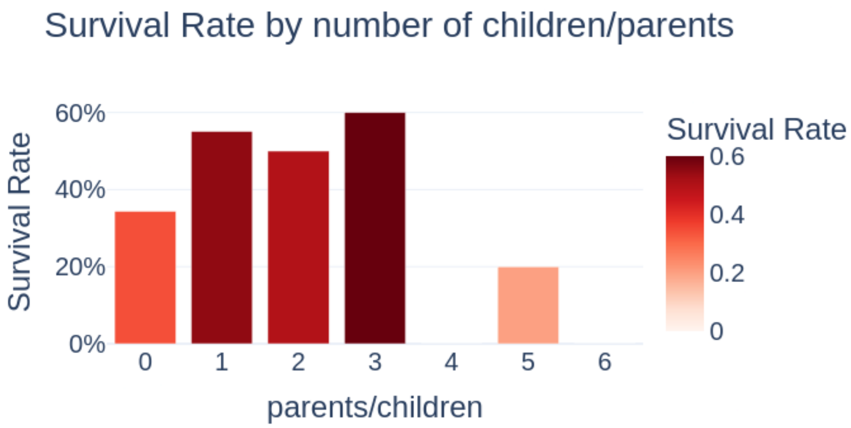
```
fig.show()
```



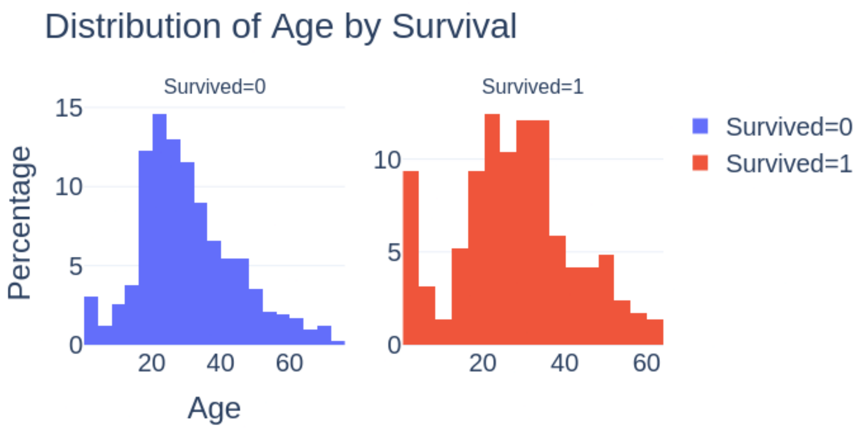
```
fig.show()
```



```
fig.show()
```



```
fig.show()
```



4.4.2 Based on the Age vs Survived Histograms:

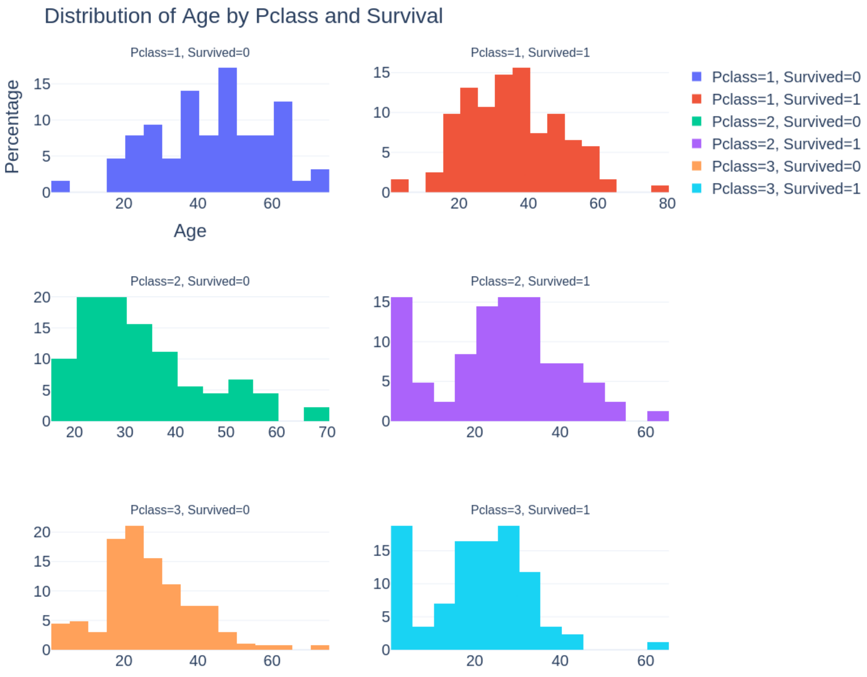
Observations

- 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.show()
```



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.

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.

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

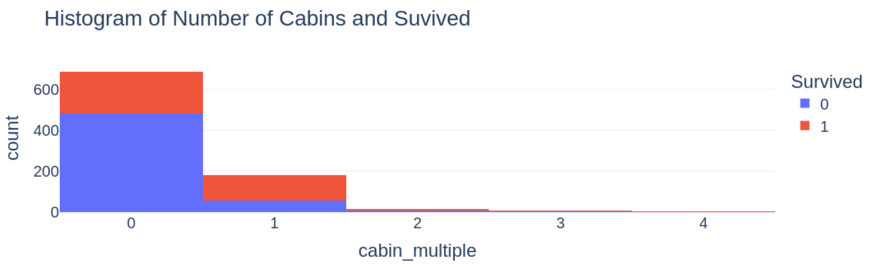
Observations

- 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')
```

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:

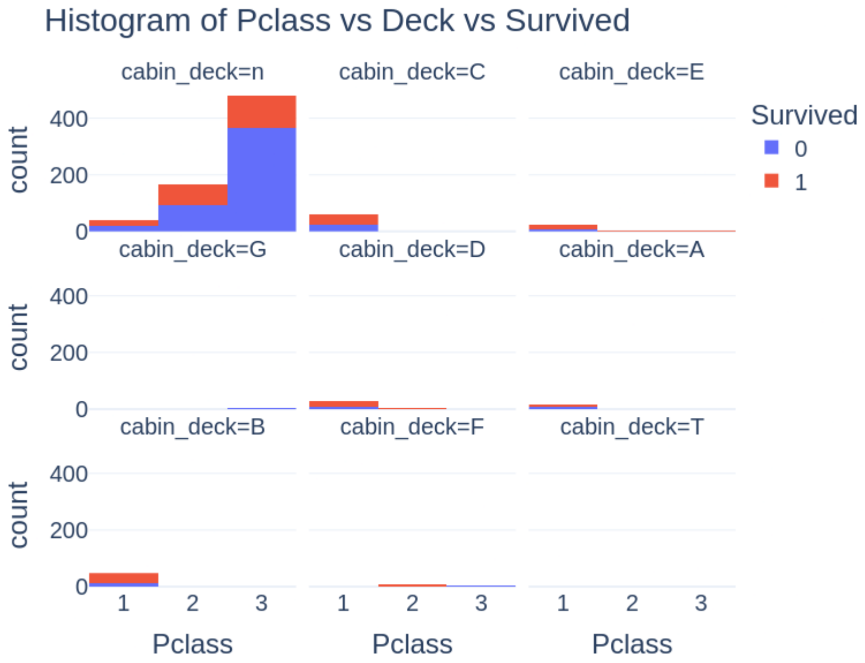
Observations

- 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

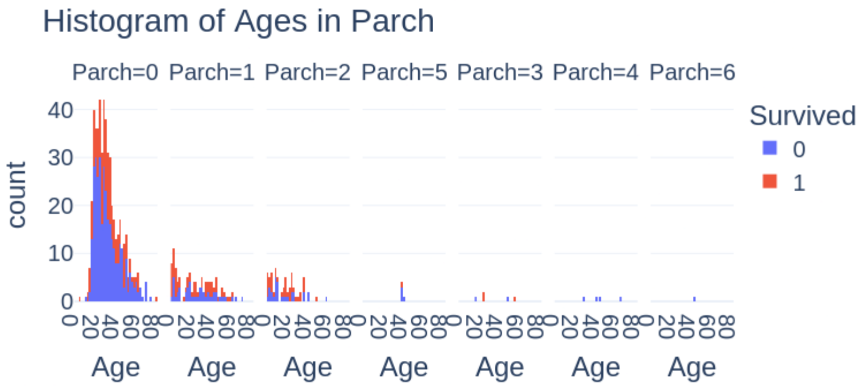
- Consider Seperating the cabin feature into only cabin letters.
- Consider creating a number of Cabins feature.

```
fig.show()
```



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.sample(10)
```

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
PassengerId											
556	0	1	Wright, Mr. George	male	62.0	0	0	113807	26.550	NaN	S
847	0	3	Sage, Mr. Douglas Bullen	male	NaN	8	2	CA. 2343	69.550	NaN	S
1105	-1	2	Howard, Mrs. Benjamin (Ellen Truelove Arman)	female	60.0	1	0	24065	26.000	NaN	S
437	0	3	Ford, Miss. Doolina Margaret "Daisy"	female	21.0	2	2	W./C. 6608	34.375	NaN	S
953	-1	2	McCrae, Mr. Arthur Gordon	male	32.0	0	0	237216	13.500	NaN	S
1077	-1	2	Maybery, Mr. Frank Hubert	male	40.0	0	0	239059	16.000	NaN	S
886	0	3	Rice, Mrs. William (Margaret Norton)	female	39.0	0	5	382652	29.125	NaN	Q
357	1	1	Bowerman, Miss. Elsie Edith	female	22.0	0	1	113505	55.000	E33	S
912	-1	1	Rothschild, Mr. Martin	male	55.0	1	0	PC 17603	59.400	NaN	C
20	1	3	Masselmani, Mrs. Fatima	female	NaN	0	0	2649	7.225	NaN	C

4.5.1 Cabin

```
passenger_df["HasCabin"] = ~passenger_df.Cabin.isnull() *1
```

```
px.histogram(passenger_df, x = "Pclass", color="HasCabin")
```

```
interesting_passengers = [823,  831,  829,  828,  827,  826,  825,  822,  833,
                          584,  938,  600,  285,   24, 1266, 1185,  648,
                          1001, 129, 1180, 1213, 873, 1114, 67, 517, 346]
passenger_df.loc[interesting_passengers].sample(10)
```

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	HasCabin
PassengerId												
600	1	1	Duff Gordon, Sir. Cosmo Edmund ("Mr Morgan")	male	49.0	1	0	PC 17485	56.9292	A20	C	1
873	0	1	Carlsson, Mr. Frans Olof	male	33.0	0	0	695	5.0000	B51 B53 B55	S	1
833	0	3	Saad, Mr. Amin	male	NaN	0	0	2671	7.2292	NaN	C	0
938	-1	1	Chevre, Mr. Paul Romaine	male	45.0	0	0	PC 17594	29.7000	A9	C	1
831	1	3	Yasbeck, Mrs. Antoni (Selini Alexander)	female	15.0	1	0	2659	14.4542	NaN	C	0
1180	-1	3	Mardirosian, Mr. Sarkis	male	NaN	0	0	2655	7.2292	F E46	C	1
346	1	2	Brown, Miss. Amelia "Mildred"	female	24.0	0	0	248733	13.0000	F33	S	1
1001	-1	2	Swane, Mr. George	male	18.5	0	0	248734	13.0000	F	S	1
1266	-1	1	Dodge, Mrs. Washington (Ruth Vidaver)	female	54.0	1	1	33638	81.8583	A34	S	1
825	0	3	Panula, Master. Urho Abraham	male	2.0	4	1	3101295	39.6875	NaN	S	0

Some cabins seem to not have splitted correctly. However, upon examining these cabins we can conclude that these are families occupying several cabins. Since the families occupies cabins very close to each other, our splitting is good enough. Those are only in 1st class.

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)
passenger_df[passenger_df['Ticket'] != passenger_df["tCheck"]]
```

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	HasCabin	cDeck	cNum	cCheck	tPref	tNum	tCheck
PassengerId																		
180	0	3	Leonard, Mr. Lionel	male	36.0	0	0	LINE	0.0	NaN	S	0	n	NaN	NaN	LINE1		LINE1
272	1	3	Tornquist, Mr. William Henry	male	25.0	0	0	LINE	0.0	NaN	S	0	n	NaN	NaN	LINE1		LINE1
303	0	3	Johnson, Mr. William Cahoone Jr	male	19.0	0	0	LINE	0.0	NaN	S	0	n	NaN	NaN	LINE1		LINE1
598	0	3	Johnson, Mr. Alfred	male	49.0	0	0	LINE	0.0	NaN	S	0	n	NaN	NaN	LINE1		LINE1

Analyzing ticket prefixes

```
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

```
q = """
select Pclass, tPrefTr, tPref,  count(*) as cnt
from passenger_df
group by Pclass, tPrefTr, tPref
order by Pclass, tPrefTr, cnt desc
limit 13
"""
ps.sqldf(q)
```

	Pclass	tPrefTr	tPref	cnt
0	1			224
1	1	FC	F.C.	3
2	1	PC	PC	92
3	1	WEP	WE/P	2
4	1	WEP	W.E.P.	2
5	2			184
6	2	CA	C.A.	31
7	2	CA	CA	2
8	2	CA	C.A./SOTON	1
9	2	FCC	F.C.C.	9
10	2	PPP	P/PP	2
11	2	SC	SC/PARIS	11
12	2	SC	SC/Paris	5

Unfortunately, I could not guess what most of these mean, and no clues was found on internet

Hypothesis: ticket number has meaning During exploration I had an hipothesis that ticket number could somehow contain an encoding to placement of the passenger on the ship. To explore this hypothesis, I 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. Interet 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

```
passenger_df[passenger_df.Age.isna()].shape[0] / passenger_df.shape[0]
```

```
0.20091673032849502
```

We see that about 20% of passengers have no age registered. Maybe it could be estimated from other features?

4.5.4 Name

There appears to be a lot of information that can be extracted from passengers names.
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

```
tokens.loc[tokens.word.str.contains("\.", ),"count"].sum()/ passenger_df.shape[0]
```

```
0.6814362108479756
```

We see that people titles have high token frequencies, suggesting that lots of people have them. Specifically, 68% of the people have them.

Moreover, the Name column appears to have a very consistent structure:
`Last_name, Title.  First_Name (Second_Name)`
This allows to relatively easy split the Name column into its components.

```
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]+)\.(?:\s(?P<fName>[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].sample(10)
```

	Pclass	Name	Sex	Age	lName	Title	fName	sName
PassengerId								
692	3	Karun, Miss. Manca	female	4.0	Karun	Miss	Manca	NaN
681	3	Peters, Miss. Katie	female	NaN	Peters	Miss	Katie	NaN
719	3	McEvoy, Mr. Michael	male	NaN	McEvoy	Mr	Michael	NaN
143	3	Hakkarainen, Mrs. Pekka Pietari (Elin Matilda ...	female	24.0	Hakkarainen	Mrs	Pekka Pietari	Elin Matilda Dolck
1124	3	Wiklund, Mr. Karl Johan	male	21.0	Wiklund	Mr	Karl Johan	NaN
208	3	Albimona, Mr. Nassef Cassem	male	26.0	Albimona	Mr	Nassef Cassem	NaN
459	2	Toomey, Miss. Ellen	female	50.0	Toomey	Miss	Ellen	NaN
82	3	Sheerlinck, Mr. Jan Baptist	male	29.0	Sheerlinck	Mr	Jan Baptist	NaN
1088	1	Spedden, Master. Robert Douglas	male	6.0	Spedden	Master	Robert Douglas	NaN
989	3	Makinen, Mr. Kalle Edvard	male	29.0	Makinen	Mr	Kalle Edvard	NaN

Title

```
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

Most titles are Mr, Mrs, Miss and Master. This may be used to estimate age where it's unknown

There are 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 equivalent of Mr, Mrs... 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

```
passenger_df.loc[passenger_df.Title.isin(["Col", "Major", "Jonkheer", "Capt", "Dr", "Rev"]),show].sort_
```

	Pclass	Name	Sex	Age	Fare	lName	Title	fName	sName
PassengerId									
399	2	Pain, Dr. Alfred	male	23.0	10.5000	Pain	Dr	Alfred	NaN
887	2	Montvila, Rev. Juozas	male	27.0	13.0000	Montvila	Rev	Juozas	NaN
849	2	Harper, Rev. John	male	28.0	33.0000	Harper	Rev	John	NaN
1041	2	Lahtinen, Rev. William	male	30.0	26.0000	Lahtinen	Rev	William	NaN
633	1	Stahelin-Maeglin, Dr. Max	male	32.0	30.5000	Stahelin-Maeglin	Dr	Max	NaN
823	1	Reuchlin, Jonkheer. John George	male	38.0	0.0000	Reuchlin	Jonkheer	John George	NaN
1056	2	Peruschitz, Rev. Joseph Maria	male	41.0	13.0000	Peruschitz	Rev	Joseph Maria	NaN
150	2	Byles, Rev. Thomas Roussel Davids	male	42.0	13.0000	Byles	Rev	Thomas Roussel Davids	NaN
246	1	Minahan, Dr. William Edward	male	44.0	90.0000	Minahan	Dr	William Edward	NaN
537	1	Butt, Major. Archibald Willingham	male	45.0	26.5500	Butt	Major	Archibald Willingham	NaN
1094	1	Astor, Col. John Jacob	male	47.0	227.5250	Astor	Col	John Jacob	NaN
797	1	Leader, Dr. Alice (Farnham)	female	49.0	25.9292	Leader	Dr	Alice	Farnham
661	1	Frauenthal, Dr. Henry William	male	50.0	133.6500	Frauenthal	Dr	Henry William	NaN
151	2	Bateman, Rev. Robert James	male	51.0	12.5250	Bateman	Rev	Robert James	NaN
450	1	Peuchen, Major. Arthur Godfrey	male	52.0	30.5000	Peuchen	Major	Arthur Godfrey	NaN
1023	1	Gracie, Col. Archibald	male	53.0	28.5000	Gracie	Col	Archibald IV	NaN

After the replacing we have just 5 categories in title:

```
q = """
select Title, count(*) as cnt
from passenger_df
group by Title
order by cnt desc
"""
ps.sqldf(q)
```

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

Second name Let’s explore the second name:

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

	Pclass	Name	Sex	Age	Fare	lName	Title	fName	sName
PassengerId									
1116	1	Candee, Mrs. Edward (Helen Churchill Hungerford)	female	53.0	27.4458	Candee	Mrs	Edward	Helen Churchill Hungerford
957	2	Corey, Mrs. Percy C (Mary Phyllis Elizabeth Mi...	female	NaN	21.0000	Corey	Mrs	Percy C	Mary Phyllis Elizabeth Miller
328	2	Ball, Mrs. (Ada E Hall)	female	36.0	13.0000	Ball	Mrs	NaN	Ada E Hall
572	1	Appleton, Mrs. Edward Dale (Charlotte Lamson)	female	53.0	51.4792	Appleton	Mrs	Edward Dale	Charlotte Lamson
368	3	Moussa, Mrs. (Mantoura Boulos)	female	NaN	7.2292	Moussa	Mrs	NaN	Mantoura Boulos
348	3	Davison, Mrs. Thomas Henry (Mary E Finck)	female	NaN	16.1000	Davison	Mrs	Thomas Henry	Mary E Finck
9	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	11.1333	Johnson	Mrs	Oscar W	Elisabeth Vilhelmina Berg
1239	3	Whabee, Mrs. George Joseph (Shawneene Abi-Saab)	female	38.0	7.2292	Whabee	Mrs	George Joseph	Shawneene Abi-Saab
191	2	Pinsky, Mrs. (Rosa)	female	32.0	13.0000	Pinsky	Mrs	NaN	Rosa
1114	2	Cook, Mrs. (Selena Rogers)	female	22.0	10.5000	Cook	Mrs	NaN	Selena Rogers

We perform a similar token analysis with these as 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]
```

	Pclass	Name	Sex	Age	Fare	lName	Title	fName	sName
PassengerId									
188	1	Romaine, Mr. Charles Hallace ("Mr C Rolmane")	male	45.0	26.5500	Romaine	Mr	Charles Hallace	"Mr C Rolmane"
200	2	Yrois, Miss. Henriette ("Mrs Harbeck")	female	24.0	13.0000	Yrois	Miss	Henriette	"Mrs Harbeck"
428	2	Phillips, Miss. Kate Florence ("Mrs Kate Louis...	female	19.0	26.0000	Phillips	Miss	Kate Florence	"Mrs Kate Louise Phillips Marshall"
600	1	Duff Gordon, Sir. Cosmo Edmund ("Mr Morgan")	male	49.0	56.9292	Duff Gordon	Mr	Cosmo Edmund	"Mr Morgan"
605	1	Homer, Mr. Harry ("Mr E Haven")	male	35.0	26.5500	Homer	Mr	Harry	"Mr E Haven"
706	2	Morley, Mr. Henry Samuel ("Mr Henry Marshall")	male	39.0	26.0000	Morley	Mr	Henry Samuel	"Mr Henry Marshall"
711	1	Mayne, Mlle. Berthe Antonine ("Mrs de Villiers")	female	24.0	49.5042	Mayne	Miss	Berthe Antonine	"Mrs de Villiers"
1036	1	Lindeberg-Lind, Mr. Erik Gustaf (Mr Edward Lin...	male	42.0	26.5500	Lindeberg-Lind	Mr	Erik Gustaf	Mr Edward Lingrey"
1219	1	Rosenshine, Mr. George (Mr	male	46.0	79.2000	Rosenshine	Mr	George	Mr George Thorne"

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 people's age

4.5.6 Some interaction variables

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 estimation 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.head(7)
```

	Fare	SibSp	Parch	Pclass	Sex_female	Sex_male	Title_Master	Title_Miss	Title_Mr	Title_Mrs	Title_Rare
PassengerId											
1	7.2500	1	0	3	False	True	False	False	True	False	False
2	71.2833	1	0	1	True	False	False	False	False	True	False
3	7.9250	0	0	3	True	False	False	True	False	False	False
4	53.1000	1	0	1	True	False	False	False	False	True	False
5	8.0500	0	0	3	False	True	False	False	True	False	False
6	8.4583	0	0	3	False	True	False	False	True	False	False
7	51.8625	0	0	1	False	True	False	False	True	False	False

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

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

	Fare	SibSp	Parch	Pclass	Sex_female	Sex_male	Title_Master	Title_Miss	Title_Mr	Title_Mrs	Title_Rare
PassengerId											
6	8.4583	0	0	3	False	True	False	False	True	False	False
18	13.0000	0	0	2	False	True	False	False	True	False	False
20	7.2250	0	0	3	True	False	False	False	False	True	False
27	7.2250	0	0	3	False	True	False	False	True	False	False
29	7.8792	0	0	3	True	False	False	True	False	False	False
30	7.8958	0	0	3	False	True	False	False	True	False	False
32	146.5208	1	0	1	True	False	False	False	False	True	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.head(7)
```

	Fare	SibSp	Parch	Pclass	Sex_female	Sex_male	Title_Master	Title_Miss	Title_Mr	Title_Mrs	Title_Rare
PassengerId											
1	7.2500	1	0	3	False	True	False	False	True	False	False
2	71.2833	1	0	1	True	False	False	False	False	True	False
3	7.9250	0	0	3	True	False	False	True	False	False	False
4	53.1000	1	0	1	True	False	False	False	False	True	False
5	8.0500	0	0	3	False	True	False	False	True	False	False
7	51.8625	0	0	1	False	True	False	False	True	False	False
8	21.0750	3	1	3	False	True	True	False	False	False	False

Train, validation, test split

4.7.1 Cross-validate ensemble models

This convenience function will be used for training, evaluation and summarization of various ML models. At this stage we will concentrate on ensemble family of 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: 0.433
Best Parameters: {'max_depth': 6, 'max_features': 9, 'max_leaf_nodes': 27, 'n_estimators': 334}
```

AdaBoost

```
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: 0.294
Best Parameters: {'learning_rate': 3, 'n_estimators': 173}
```

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: 0.430
Best Parameters: {'max_depth': 10, 'max_features': 6, 'max_leaf_nodes': 8, 'n_estimators': 149}
```

```
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': 6, 'max_features': 9, 'max_leaf_...	0.533011	0.478867	10	2
1	00:00:00	AdaBoostRegressor()	{'learning_rate': 3, 'n_estimators': 173}	0.349628	0.321810	10	2
2	00:00:00	GradientBoostingRegressor()	{'max_depth': 10, 'max_features': 6, 'max_leaf...	0.629950	0.494468	10	2

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

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

fig.show()
```

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     22.482209
18    32.216009
20    44.775079
27    27.427907
29    21.672006
30    27.910477
32    45.170013
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

```
# Gives 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.show()
```

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

```
display(df1), display(df2)
```

	FamilySize	Survived
3	4	0.724138
2	3	0.578431
1	2	0.552795
6	7	0.333333
0	1	0.303538
4	5	0.200000
5	6	0.136364
7	8	0.000000
8	11	0.000000

	IsAlone	Survived
0	0	0.505650
1	1	0.303538

```
(None, None)
```

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

```
df
```

	CategoricalFare	Survived
0	(-0.001, 7.775]	0.205128
1	(7.775, 8.662]	0.190789
2	(8.662, 14.454]	0.366906
3	(14.454, 26.0]	0.436242
4	(26.0, 53.1]	0.435065
5	(53.1, 512.329]	0.695035

Create a New feature CategoricalAge

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

	CategoricalAge	Survived
0	(0.169, 18.0]	0.503185
1	(18.0, 24.0]	0.352941
2	(24.0, 28.0]	0.315385
3	(28.0, 34.0]	0.365672
4	(34.0, 43.0]	0.384615
5	(43.0, 80.0]	0.368056

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.head(10)
```

	Survived	Pclass	Sex	Age	Parch	Fare	Embarked	Words_count	Cabin_multiple	Cabin_deck	FamilySize	IsAlone	Age*Class
PassengerId													
1	0	3	1	1.0	0	1	0	4	0	0	2	0	3.0
2	1	1	0	4.0	0	5	1	7	1	3	2	0	4.0
3	1	3	0	2.0	0	1	0	3	0	0	1	1	6.0
4	1	1	0	4.0	0	5	0	7	1	3	2	0	4.0
5	0	3	1	4.0	0	1	0	4	0	0	1	1	12.0
6	0	3	1	1.0	0	1	2	3	0	0	1	1	3.0
7	0	1	1	5.0	0	5	0	4	1	5	1	1	5.0
8	0	3	1	0.0	1	3	0	4	0	0	5	0	0.0
9	1	3	0	2.0	2	1	0	7	0	0	3	0	6.0
10	1	2	0	0.0	0	5	1	5	0	0	2	0	0.0

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

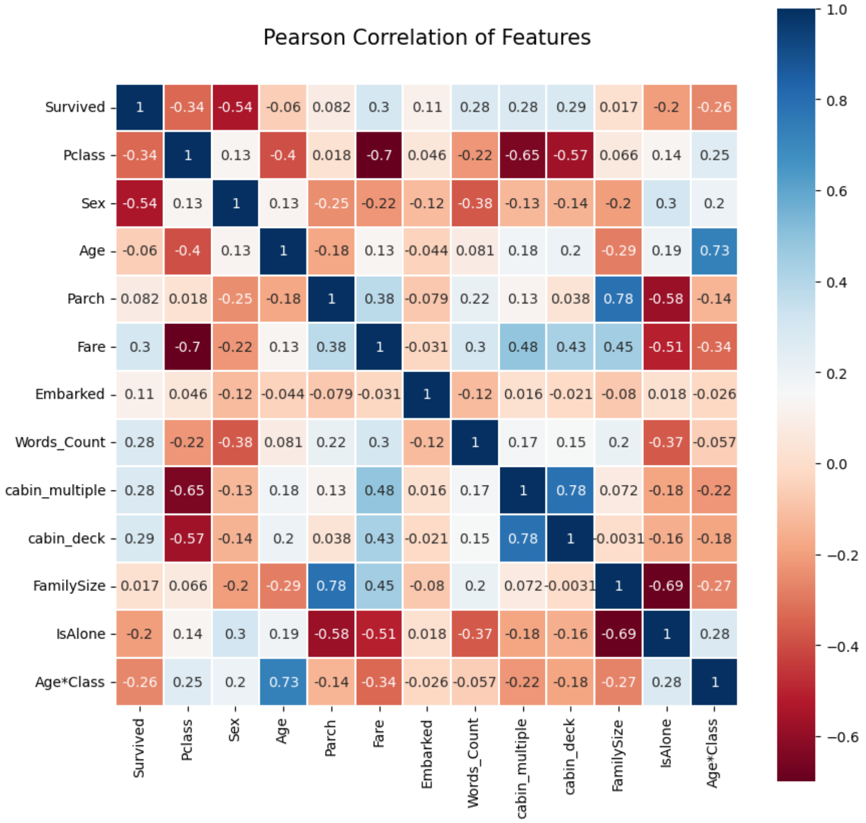
	Survived	Pclass	Sex	Age	Parch	Fare	Embarked	Words_count	Cabin_multiple	Cabin_deck	FamilySize	IsAlone	Age*Class
PassengerId													
1	0	3	1	1.0	0	1	0	4	0	0	2	0	3.0
2	1	1	0	4.0	0	5	1	7	1	3	2	0	4.0
3	1	3	0	2.0	0	1	0	3	0	0	1	1	6.0
4	1	1	0	4.0	0	5	0	7	1	3	2	0	4.0
5	0	3	1	4.0	0	1	0	4	0	0	1	1	12.0
6	0	3	1	1.0	0	1	2	3	0	0	1	1	3.0
7	0	1	1	5.0	0	5	0	4	1	5	1	1	5.0
8	0	3	1	0.0	1	3	0	4	0	0	5	0	0.0
9	1	3	0	2.0	2	1	0	7	0	0	3	0	6.0
10	1	2	0	0.0	0	5	1	5	0	0	2	0	0.0

```
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', a
plt.show()
```



	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	-0.044	0.081	0.18	0.2	-0.29	0.19	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	-0.044	-0.079	-0.031	1	-0.12	0.016	-0.021	-0.08	0.018	-0.026
Words_Count	0.28	-0.22	-0.38	0.081	0.22	0.3	-0.12	1	0.17	0.15	0.2	-0.37	-0.057
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	-0.22
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	-0.18
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	0.19	-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	-0.026	-0.057	-0.22	-0.18	-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

```
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

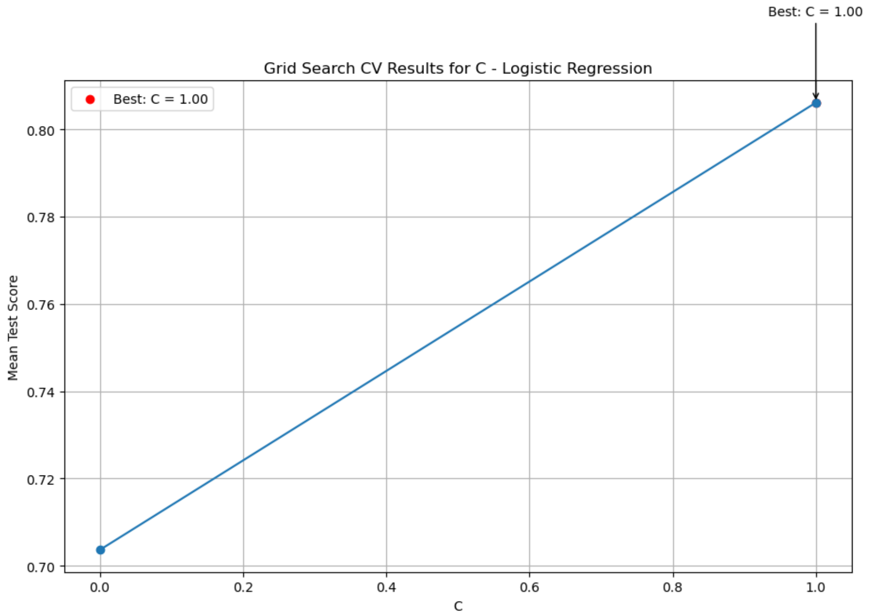
5.2.3 Confusion Matrix for Best Model

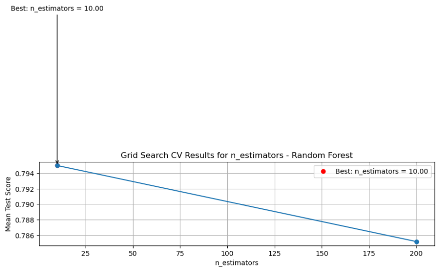
5.3 Classical models

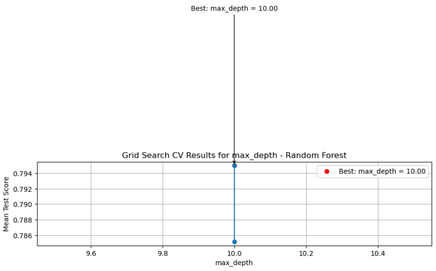
Initialize models with Hyperparameters

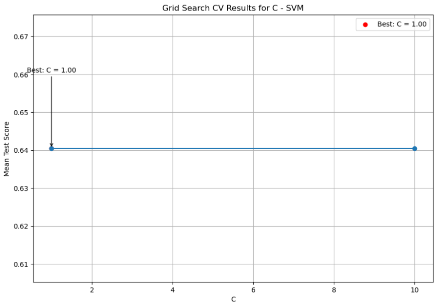
```
# Define models
```

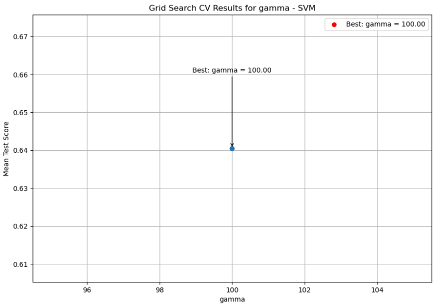
```
models = {
    'Logistic Regression': LogisticRegression(),
    'Random Forest': RandomForestClassifier(),
    'SVM': SVC(),
    #'Lasso': Lasso(),
    #'Ridge': Ridge(),
    'Gradient Boosting': GradientBoostingClassifier(),
    'Decision Tree': DecisionTreeClassifier()
}

# 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
```

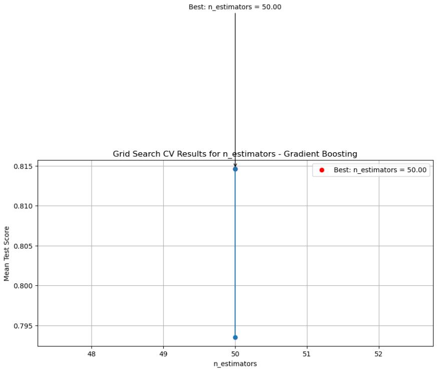


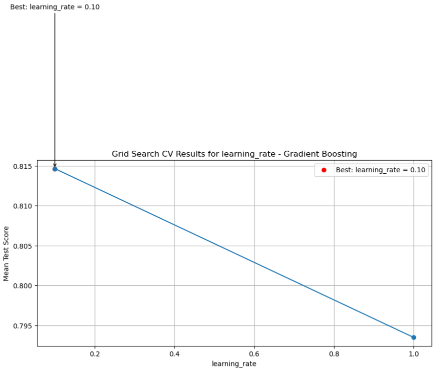


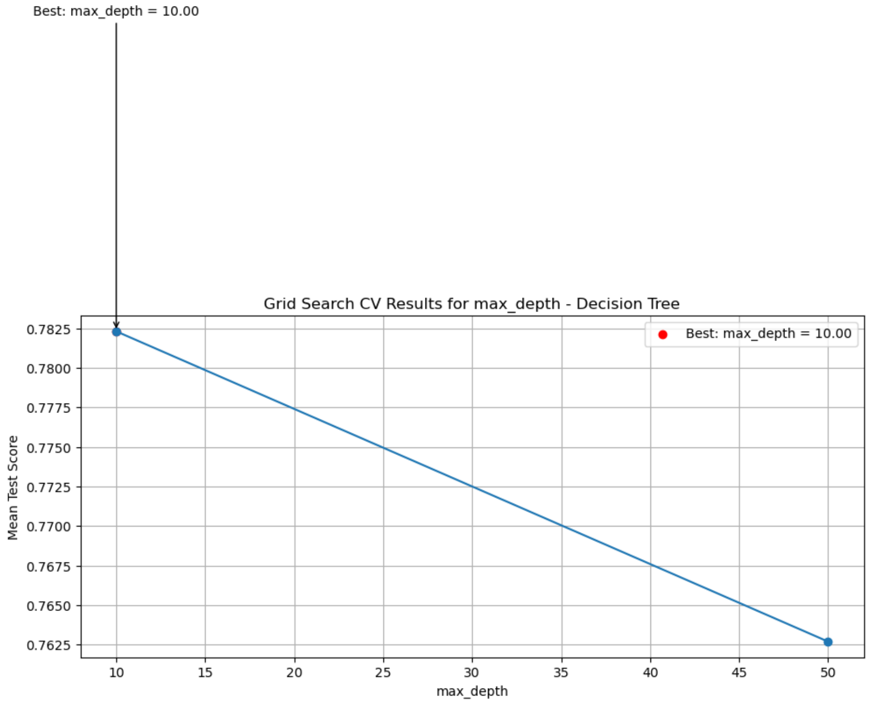



Grid Search CV Results for C - SVM
Best: C = 1.00
Best: C = 1.00
Mean Test Score
0.67
0.66
0.65
0.64
0.63
0.62
0.61
2
4
6
8
10
C


Grid Search CV Results for gamma - SVM
Best: gamma = 100.00
Best: gamma = 100.00
Mean Test Score
0.67
0.66
0.65
0.64
0.63
0.62
0.61
96
98
100
102
104
gamma


Best: n_estimators = 50.00
Grid Search CV Results for n_estimators - Gradient Boosting
Best: n_estimators = 50.00
Mean Test Score
0.815
0.810
0.805
0.800
0.795
48
49
50
51
52
n_estimators


Best: learning_rate = 0.10
Grid Search CV Results for learning_rate - Gradient Boosting
Best: learning_rate = 0.10
Mean Test Score
0.815
0.810
0.805
0.800
0.795
0.2
0.4
0.6
0.8
1.0
learning_rate


Best: max_depth = 10.00
Grid Search CV Results for max_depth - Decision Tree
Best: max_depth = 10.00
Mean Test Score
0.7825
0.7800
0.7775
0.7750
0.7725
0.7700
0.7675
0.7650
0.7625
10
15
20
25
30
35
40
45
50
max_depth

	Model	Best Parameters	Best Score (CV)	Validation Score
0	Logistic Regression	{'C': 1}	0.806103	0.810056
1	Random Forest	{'n_estimators': 10, 'max_depth': 10}	0.794992	0.810056
2	SVM	{'gamma': 100, 'C': 1}	0.640493	0.620112
3	Gradient Boosting	{'n_estimators': 50, 'learning_rate': 0.1}	0.814632	0.832402
4	Decision Tree	{'max_depth': 10}	0.782316	0.787709

We can see that SVM gives us the best Validation score, meaning SVM 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:  Gradient Boosting
Best Validation Score: 0.8324
```

```
GradientBoostingClassifier(n_estimators=50)
```

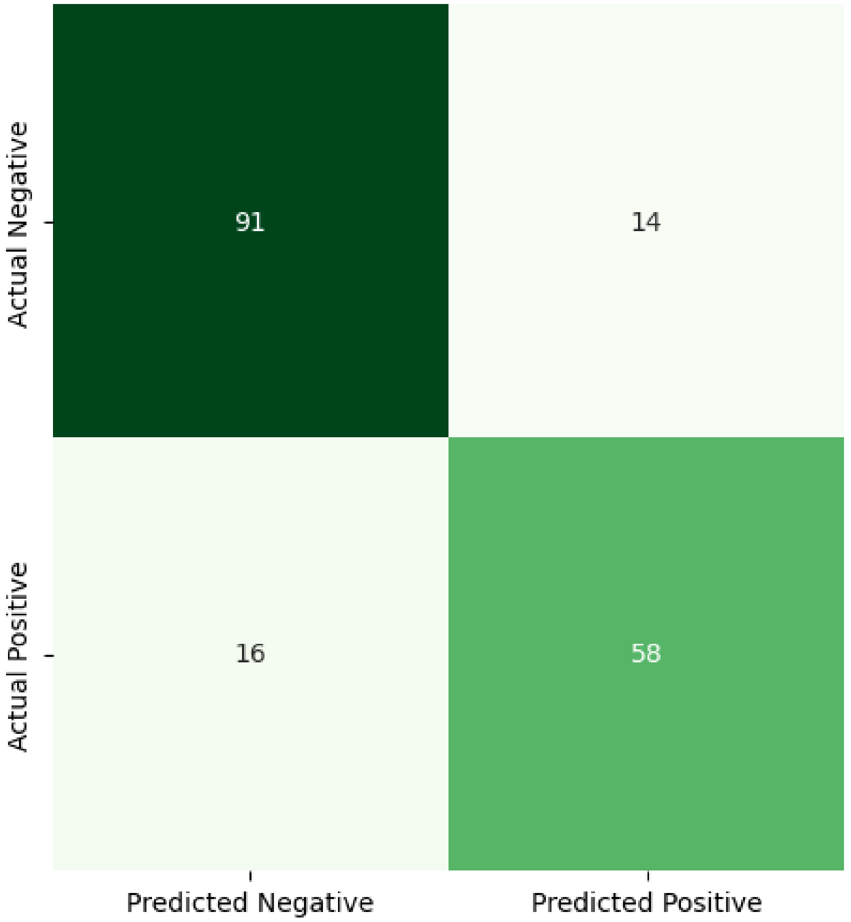
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

[x]GradientBoostingClassifier

```
GradientBoostingClassifier(n_estimators=50)
```

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

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
AttributeError                            Traceback (most recent call last)
Cell In[159], line 1
 - - - -> 1 evaluate_best_model(best_model, X_train, y_train, X_val, y_val)

Cell In[155], line 22, in evaluate_best_model(best_model, X_train, y_train, X_val, y_val)
     18 plt.subplot(1, 2, 1)
     19 sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Greens", cbar=False,
     20             xticklabels=['Predicted Negative', 'Predicted Positive'],
     21             yticklabels=['Actual Negative', 'Actual Positive'])
 - - -> 22 plt.title(f"Confusion Matrix for {str(best_model.estimator)} model")
     23 plt.xlabel("Predicted label")
     24 plt.ylabel("True label")

AttributeError: 'GradientBoostingClassifier' object has no attribute 'estimator'
```



5.4 Lasso and Ridge Regularization

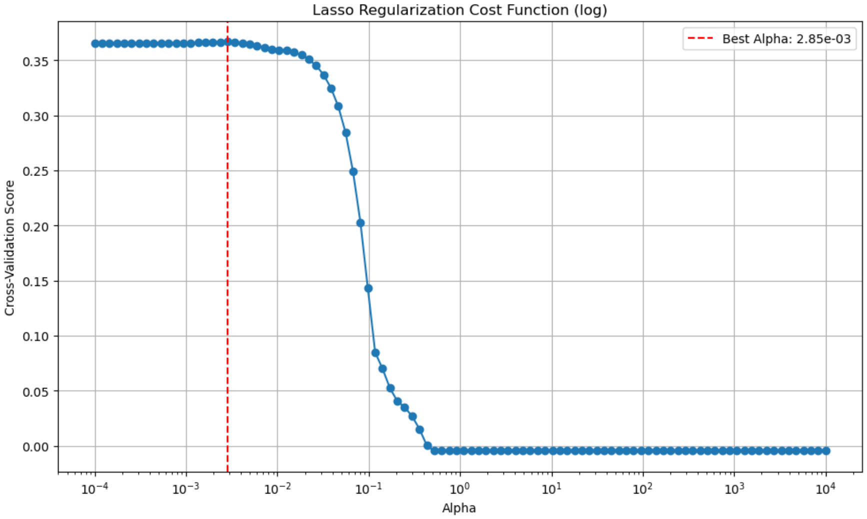
```
# Creating a Lasso Regularization model
lasso_model = Lasso(alpha=0.1)
# Creating a Ridge Regularization model
ridge_model = Ridge(alpha=0.1)

# Cross -validation to plot the cost function
alphas = np.logspace( -4, 4, 100)  # Range of alpha values for cross -validation
scores_lasso = []
scores_ridge = []
```

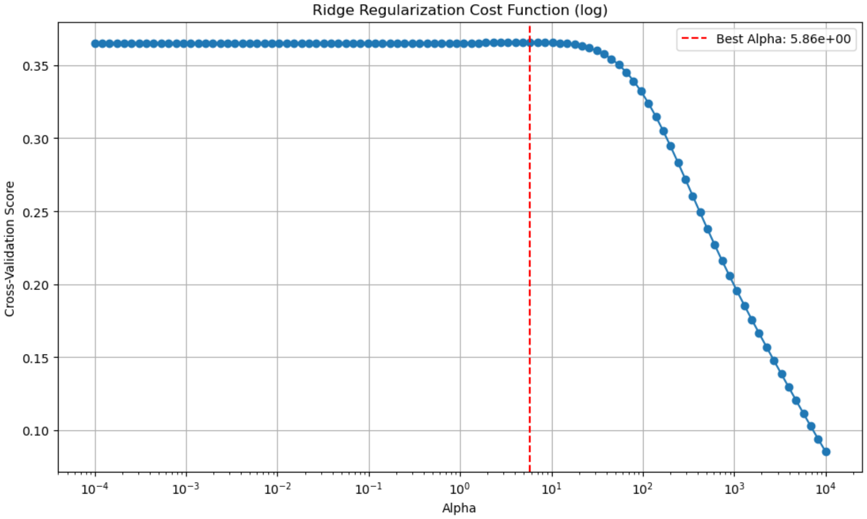
5.4.1 10-fold Cross Validation using initial Hyper Parameters

In order to tune Performance in every model, we'll have multiple values of the hyper paramaters and find the values that give us the best cross validation score, which in this case is accuracy scores:

```
for alpha in alphas:
    lasso_model.alpha = alpha
    cv_scores_lasso = cross_val_score(lasso_model, X_train, y_train, cv=10)  # 10 -fold cross -validati
    scores_lasso.append(np.mean(cv_scores_lasso))

    ridge_model.alpha = alpha
    cv_scores_ridge = cross_val_score(ridge_model, X_train, y_train, cv=10)  # 10 -fold cross -validati
    scores_ridge.append(np.mean(cv_scores_ridge))
```

```
# Finding the best alpha value
best_alpha_lasso = alphas[np.argmax(scores_lasso)]
best_alpha_ridge = alphas[np.argmax(scores_ridge)]

# Create subplots
fig, axs = plt.subplots(2, 1, figsize=(10, 12))

# Plotting the cost function for Lasso Regularization
axs[0].plot(alphas, scores_lasso, ’ -o’)
axs[0].set_xlabel(’Alpha’)
axs[0].set_ylabel(’Cross -Validation Score’)
axs[0].set_title(’Lasso Regularization Cost Function (log)’)
axs[0].set_xscale(’log’)
axs[0].grid(True)

# Highlighting the best alpha value for Lasso Regularization
axs[0].axvline(x=best_alpha_lasso, color=’r’, linestyle=’ - -’, label=’Best Alpha: {:.2e}’.format(best_a
axs[0].legend()

# Plotting the cost function for another model (assuming you have another set of data)
axs[1].plot(alphas, scores_ridge, ’ -o’)  # Replace scores_another_model with your actual scores
axs[1].set_xlabel(’Alpha’)
axs[1].set_ylabel(’Cross -Validation Score’)
axs[1].set_title(’Ridge Regularization Cost Function (log)’)
axs[1].set_xscale(’log’)
axs[1].grid(True)

# Highlighting the best alpha value for the another model
axs[1].axvline(x=best_alpha_ridge, color=’r’, linestyle=’ - -’, label=’Best Alpha: {:.2e}’.format(best_a
axs[1].legend()

# Adjust layout
plt.tight_layout()

plt.show()
```



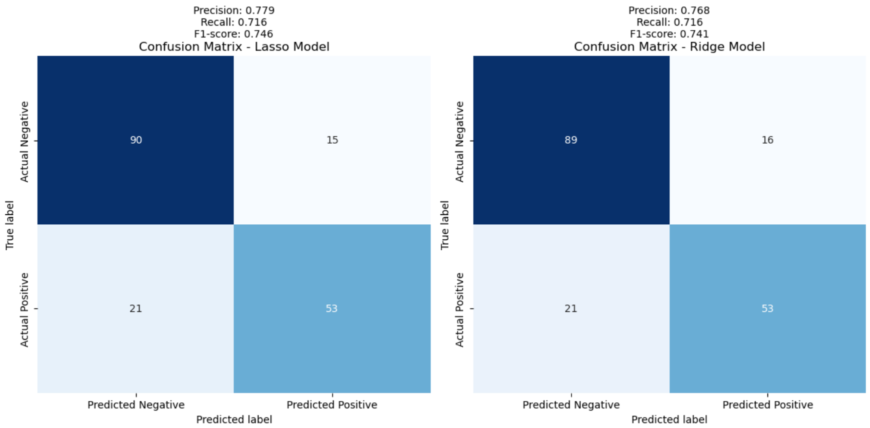


After tuning, we fit the hyperparameters into the model and fit the trained data to this model:

```
# Training the model with the best alpha
lasso_model.alpha = best_alpha_lasso
lasso_model.fit(X_train, y_train)

# Training the model with the best alpha
ridge_model.alpha = best_alpha_ridge
ridge_model.fit(X_train, y_train)

# Predictions on train and validation data
train_predictions_lasso = lasso_model.predict(X_train)
train_predictions_ridge = ridge_model.predict(X_train)
val_predictions_lasso = lasso_model.predict(X_val)
val_predictions_ridge = ridge_model.predict(X_val)

# Accuracy metrics
train_accuracy_lasso = accuracy_score(y_train, np.round(train_predictions_lasso))
train_accuracy_ridge = accuracy_score(y_train, np.round(train_predictions_ridge)) # Round predictions to
val_accuracy_lasso = accuracy_score(y_val, np.round(val_predictions_lasso))
val_accuracy_ridge = accuracy_score(y_val, np.round(val_predictions_ridge))

print("Train Accuracy: \nLasso - {:.4f}" .format(train_accuracy_lasso) ,"\nRidge - {:.4f} \n" .format(t
print("Validation Accuracy: \nLasso - {:.4f}" .format(val_accuracy_lasso) ,"\nRidge - {:.4f} " .format(v
```

```
Train Accuracy:
Lasso - 0.7978
Ridge - 0.8020

Validation Accuracy:
Lasso - 0.7989
Ridge - 0.7933
```

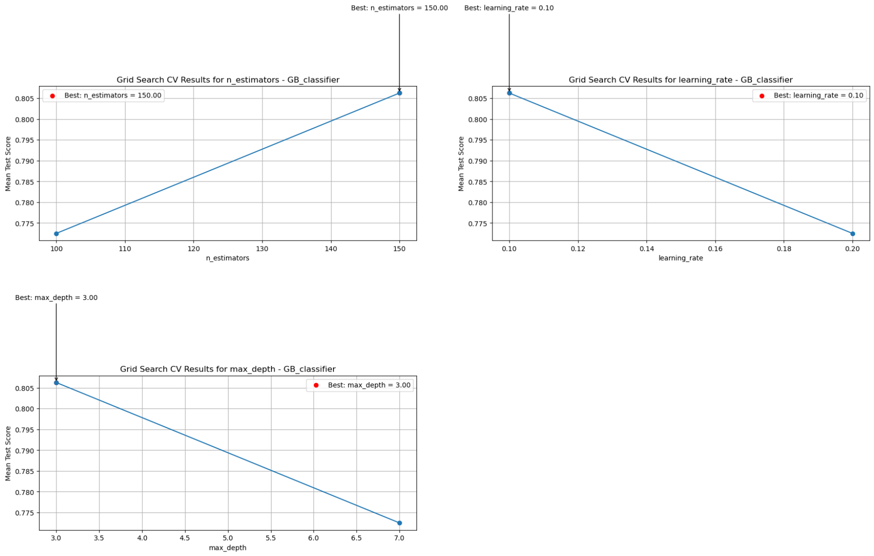
```
# Confusion matrix for Lasso model
conf_matrix_lasso = confusion_matrix(y_val, np.round(val_predictions_lasso))

# Confusion matrix for Ridge model
conf_matrix_ridge = confusion_matrix(y_val, np.round(val_predictions_ridge))

# Calculate precision, recall, and F1 -score for Lasso Model
precision_lasso = precision_score(y_val, np.round(val_predictions_lasso))
recall_lasso = recall_score(y_val, np.round(val_predictions_lasso))
f1_lasso = f1_score(y_val, np.round(val_predictions_lasso))

# Calculate precision, recall, and F1 -score for Ridge Model
precision_ridge = precision_score(y_val, np.round(val_predictions_ridge))
recall_ridge = recall_score(y_val, np.round(val_predictions_ridge))
f1_ridge = f1_score(y_val, np.round(val_predictions_ridge))

# Plot confusion matrix with additional metrics for Lasso Model
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
sns.heatmap(conf_matrix_lasso, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=['Predicted Negative', 'Predicted Positive'],
            yticklabels=['Actual Negative', 'Actual Positive'])
plt.title("Confusion Matrix - Lasso Model")
plt.xlabel("Predicted label")
plt.ylabel("True label")
plt.text(0.5, 1.1, f"Precision: {precision_lasso:.3f}\nRecall: {recall_lasso:.3f}\nF1 -score: {f1_lasso
         horizontalalignment='center', verticalalignment='center', transform=plt.gca().transAxes)

# Plot confusion matrix with additional metrics for Ridge Model
plt.subplot(1, 2, 2)
sns.heatmap(conf_matrix_ridge, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=['Predicted Negative', 'Predicted Positive'],
            yticklabels=['Actual Negative', 'Actual Positive'])
plt.title("Confusion Matrix - Ridge Model")
plt.xlabel("Predicted label")
plt.ylabel("True label")
plt.text(0.5, 1.1, f"Precision: {precision_ridge:.3f}\nRecall: {recall_ridge:.3f}\nF1 -score: {f1_ridge
         horizontalalignment='center', verticalalignment='center', transform=plt.gca().transAxes)

plt.tight_layout()
plt.show()
```



5.5 Gradient Boosting

Tuning Boosting Hyperparameters using Grid search 10-fold Cross Validation:

```
# Define hyperparameters grid
param_grid = {
    ’n_estimators’: [50, 100, 150],
    ’learning_rate’: [0.01, 0.1, 0.2],
    ’max_depth’: [3, 5, 7]
}

# Create a Gradient Boosting Classifier
gb_classifier = GradientBoostingClassifier(random_state=42)

models = {’GB_classifier’: gb_classifier}

param_grids = {’GB_classifier’: param_grid}

train_and_evaluate_models(models, X_train, y_train, X_val, y_val, param_grids, scoring=’accuracy’, cv=1

# # Perform GridSearchCV to find the best hyperparameters
# grid_search = model_selection.RandomizedSearchCV( n_iter= iterations,
#     estimator=gb_classifier, param_distributions=param_grid, cv=10, scoring=’accuracy’)
# grid_search.fit(X_train, y_train)

# # Get the best hyperparameters
# best_params = grid_search.best_params_

# # Create a new Gradient Boosting Classifier with the best hyperparameters
# best_gb_classifier = GradientBoostingClassifier(**best_params, random_state=42)
# best_gb_classifier.fit(X_train, y_train)

# # Plotting the cost function
# results = grid_search.cv_results_
# scores = results[’mean_test_score’]
# params = results[’params’]
```



```
({'GB_classifier': {'best_params': {'n_estimators': 150,
    'max_depth': 3,
    'learning_rate': 0.1},
   'best_score': 0.8062597809076681,
   'validation_score': 0.8156424581005587}},
            Model                                    Best Parameters  \
 0  GB_classifier  {'n_estimators': 150, 'max_depth': 3, 'learnin...

    Best Score (CV)  Validation Score
 0          0.80626          0.815642  )
```

5.5.1 Results and Fitting model:

```
# plt.figure(figsize=(18, 12))

# for i, param_name in enumerate(param_grid.keys()):
#     plt.subplot(2, 2, i + 1)
#     param_values = [param[param_name] for param in params]
#     plt.plot(param_values, scores, marker='o')
#     plt.xlabel(param_name)
#     plt.ylabel('Mean Test Score')
#     plt.title('Grid Search CV Results for {}'.format(param_name))
#     plt.grid(True)

#   # Marking the best parameter value
#     best_value = best_params[param_name]
#     best_score = grid_search.best_score_
#     plt.scatter(best_value, best_score, color='red', label='Best: {} = {:.2f}'.format(param_name, best
#     plt.annotate('Best: {} = {:.2f}'.format(param_name, best_value), xy=(best_value, best_score), xyte
#                  arrowprops=dict(facecolor='black', arrowstyle=' ->'), horizontalalignment='center')
#     plt.legend()

# plt.tight_layout()
# plt.show()


visualize_grid_search_results(param_grid, params, scores, best_params, best_score, title='')
```

```
# Print the best hyperparameters
print("Best Hyperparameters:", best_params)

# Predictions on validation data
train_predictions = best_gb_classifier.predict(X_train)

# Predictions on validation data
val_predictions = best_gb_classifier.predict(X_val)

# Accuracy metrics
train_accuracy = accuracy_score(y_train, np.round(train_predictions))
val_accuracy = accuracy_score(y_val, np.round(val_predictions))

print("Train Accuracy: {:.4f}" .format(train_accuracy))
print("Validation Accuracy: {:.4f}" .format(val_accuracy))
```

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
NameError                                 Traceback (most recent call last)
Cell In[166], line 24
      1 # plt.figure(figsize=(18, 12))
      2
      3 # for i, param_name in enumerate(param_grid.keys()):
   (...)
     20 # plt.tight_layout()
     21 # plt.show()
 - - -> 24 visualize_grid_search_results(param_grid, params, scores, best_params, best_score,title='')
     27 # Print the best hyperparameters
     28 print("Best Hyperparameters:", best_params)

NameError: name 'params' is not defined
```

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

# # Confusion matrix for Gradient Boosting Model
# conf_matrix = confusion_matrix(y_val, np.round(val_predictions))

# # Calculate precision, recall, and F1 -score for Gradient Boosting Model
# precision_gb = precision_score(y_val, np.round(val_predictions))
# recall_gb = recall_score(y_val, np.round(val_predictions))
# f1_gb = f1_score(y_val, np.round(val_predictions))

# # Plot confusion matrix with additional metrics for Gradient Boosting Model
# plt.figure(figsize=(12, 6))
# plt.subplot(1, 2, 1)
# sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Greens", cbar=False,
#             xticklabels=['Predicted Negative', 'Predicted Positive'],
#             yticklabels=['Actual Negative', 'Actual Positive'])
# plt.title("Confusion Matrix - Gradient Boosting Model")
# plt.xlabel("Predicted label")
# plt.ylabel("True label")
# plt.text(0.5, 1.15, f"Precision: {precision_gb:.2f}\nRecall: {recall_gb:.2f}\nF1 -score: {f1_gb:.2f}"
#          horizontalalignment='center', verticalalignment='center', transform=plt.gca().transAxes)
# plt.tight_layout()
# plt.show()
```

5.6 SVM

```
plt.show()
```

And the best hyperparameters are:

```
print("Best Hyperparameters:", best_params)
print("Train Accuracy: {:.4f}" .format(train_accuracy))
print("Validation Accuracy: {:.4f}" .format(val_accuracy))
```

*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.

```
plt.show()
```

6 Test input data for submission

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

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!")

output.sample(10)
```