Naive Bayes and Decision Tree models on Titanic Classification Problem

We start by

1. Exploratory Data Analysis (EDA) on the dataset, then
2. Apply the Naive Bayes and Decision Tree models subsequently

Given 2 sets of data: A training set, train.csv, and a testing set, test.csv.

1. The training set has information related to the passengers aboard the Titanic along with the answer to whether they survived the shipwreck or not.
2. While the test data contains only the passenger information. We have to use the training data to make a model that can predict the chances of survival for the passengers in the testing data.

Loading important libraries that we may use:

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Loading out training and testing data into a Data Frame :

In [4]:
```
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")
```

In [5]:
```
train.head()
```

Out[5]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

In [6]:
```
test.head()
```

Out[6]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S

Exploratory Data Analysis

We try to see how different features affect the survival rate by Data Visualization

Cleaning the Data :

We try to find the features that affect Survival rate the most and clean the data accordingly to make it ready for our model

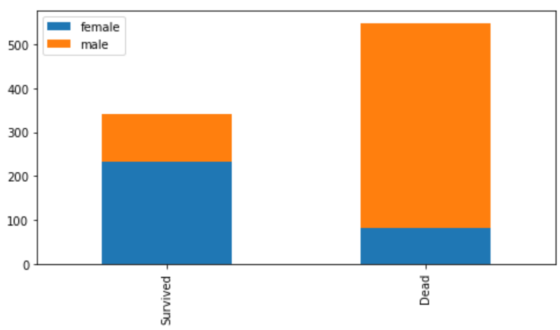
1. Name Vs Survival rate :

We try to see how the name affects the survival rate of the person

Our data set has names with prefixes like "Mr.", "Mrs.", "Miss" and so on, this will affect a person's survival rate.
It is known that Females had a better chance of survival compared to males
We plot a bar graph to check this :

In [7]:
```
survived = train[train["Survived"]==1]["Sex"].value_counts()
dead = train[train["Survived"]==0]["Sex"].value_counts()
df_sex = pd.DataFrame([survived,dead])
df_sex.index = ["Survived","Dead"]
df_sex.plot(kind="bar",stacked = True, figsize = (8,4))
```

Out[7]: <AxesSubplot:>



The chart confirms that males have a lesser chance of surviving compared to females.
Prefixes like **"Mr."**, **"Sir."** would have lesser chance of survival compared to Prefixes like **"Miss."**, **"Mrs."**
Any other prefixes like **"Rev"** or **"Dr."** can be put into a separate category.

In [8]:
```
combined_data = [train,test]
```

```python
for data in combined_data:
    data["Prefix"] = data["Name"].str.extract(' ([A-Za-z]+)\.', expand = False)
```

In [9]:

```python
train["Prefix"].value_counts()
```

Out[9]:

```
Mr          517
Miss        182
Mrs         125
Master       40
Dr            7
Rev           6
Major         2
Col           2
Mlle          2
Sir           1
Countess      1
Mme           1
Capt          1
Jonkheer      1
Don           1
Lady          1
Ms            1
Name: Prefix, dtype: int64
```

Mapping is defined as :

1. Mr, Master as 0
2. Miss, Mlle, Ms as 1
3. Mrs, Mme, Lady as 2
4. Others as 3

In [10]:

```python
Prefix_mapping = {"Mr":0,"Miss":1,"Mrs":2,"Master":0,"Dr":3,"Rev":3,"Major":3,"Mlle":1,"Col":3,"Capt":3,"Sir":3,"Ms":1,"Lady":3,"Mme":2,"Countess":3,"Jonkheer":3,"Don":3}
```

In [11]:

```python
for data in combined_data:
    data["Prefix"] = data["Prefix"].map(Prefix_mapping)
```

In [12]:

```python
train.head()
```

Out[12]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Prefix
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S	0
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C	2
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S	1
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S	2
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S	0

In [13]:

```python
test.head()
```

Out[13]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Prefix
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q	0.0
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S	2.0
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q	0.0
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S	0.0
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S	2.0

In [14]:

```python
test["Prefix"] = test["Prefix"].fillna(3)
test.describe()
```

Out[14]:

	PassengerId	Pclass	Age	SibSp	Parch	Fare	Prefix
count	418.000000	418.000000	332.000000	418.000000	418.000000	417.000000	418.000000
mean	1100.500000	2.265550	30.272590	0.447368	0.392344	35.627188	0.576555
std	120.810458	0.841838	14.181209	0.896760	0.981429	55.907576	0.822423
min	892.000000	1.000000	0.170000	0.000000	0.000000	0.000000	0.000000
25%	996.250000	1.000000	21.000000	0.000000	0.000000	7.895800	0.000000
50%	1100.500000	3.000000	27.000000	0.000000	0.000000	14.454200	0.000000
75%	1204.750000	3.000000	39.000000	1.000000	0.000000	31.500000	1.000000
max	1309.000000	3.000000	76.000000	8.000000	9.000000	512.329200	3.000000

In [15]:

```python
for data in combined_data:
    data.drop(columns = "Name",inplace=True)
```

In [16]:

```python
train.head()
```

Out[16]:

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Prefix
0	1	0	3	male	22.0	1	0	A/5 21171	7.2500	NaN	S	0
1	2	1	1	female	38.0	1	0	PC 17599	71.2833	C85	C	2
2	3	1	3	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S	1
3	4	1	1	female	35.0	1	0	113803	53.1000	C123	S	2
4	5	0	3	male	35.0	0	0	373450	8.0500	NaN	S	0

In [17]:

```python
test.head()
```

Out[17]:

	PassengerId	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Prefix
0	892	3	male	34.5	0	0	330911	7.8292	NaN	Q	0.0

	PassengerId	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Prefix
1	893	3	female	47.0	1	0	363272	7.0000	NaN	S	2.0
2	894	2	male	62.0	0	0	240276	9.6875	NaN	Q	0.0
3	895	3	male	27.0	0	0	315154	8.6625	NaN	S	0.0
4	896	3	female	22.0	1	1	3101298	12.2875	NaN	S	2.0

2. Cabin Vs Survival :

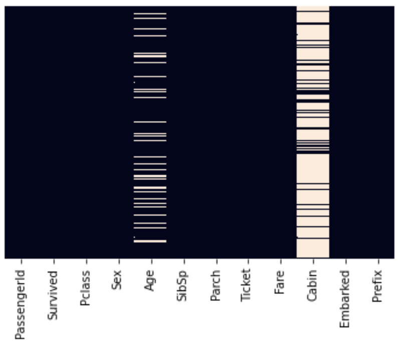
We first see how many null values are there in the Cabin column of our train data set :

In [18]:

```
import seaborn as sns

sns.heatmap(train.isnull(), yticklabels = False, cbar = False)
```

Out[18]: <AxesSubplot:>



In [19]:

```
train["Cabin"].isnull().value_counts()
```

Out[19]:

```
True     687
False    204
Name: Cabin, dtype: int64
```

We check to see if having a Cabin number has anything to do with the survivar rate of the passenger:

In [20]:

```
train["Cabin"] = train["Cabin"].fillna(0)
for i in range(891):
    if(train.at[i,"Cabin"]!=0):
        train.at[i,"Cabin"]=1
train.head()
```

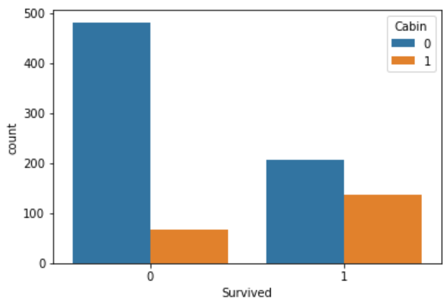
Out[20]:

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Prefix
0	1	0	3	male	22.0	1	0	A/5 21171	7.2500	0	S	0
1	2	1	1	female	38.0	1	0	PC 17599	71.2833	1	C	2
2	3	1	3	female	26.0	0	0	STON/O2. 3101282	7.9250	0	S	1
3	4	1	1	female	35.0	1	0	113803	53.1000	1	S	2
4	5	0	3	male	35.0	0	0	373450	8.0500	0	S	0

In [21]:

```
sns.countplot(x = "Survived", hue = "Cabin", data= train)
```

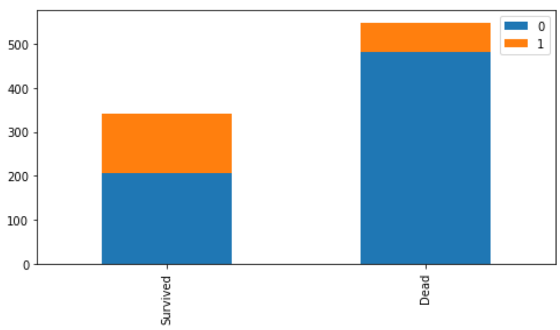
Out[21]: <AxesSubplot:xlabel='Survived', ylabel='count'>



In [22]:

```
survived = train[train["Survived"]==1]["Cabin"].value_counts()
dead = train[train["Survived"]==0]["Cabin"].value_counts()
df_cabin = pd.DataFrame([survived,dead])
df_cabin.index = ["Survived","Dead"]
df_cabin.plot(kind="bar",stacked = True, figsize = (8,4))
```

Out[22]: <AxesSubplot:>



We can see that Cabin has too many Null values, but most passengers that did not survive also didnt have a cabin number, we can code our data as

Having a Cabin : 1
Not Having a Cabin : 0

In [23]:
```
test["Cabin"] = test["Cabin"].fillna(0)
for i in range(417):
    if(test.at[i,"Cabin"]!=0):
        test.at[i,"Cabin"]=1
```

In [24]:
```
test.head()
```

Out[24]:

	PassengerId	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Prefix
0	892	3	male	34.5	0	0	330911	7.8292	0	Q	0.0
1	893	3	female	47.0	1	0	363272	7.0000	0	S	2.0
2	894	2	male	62.0	0	0	240276	9.6875	0	Q	0.0
3	895	3	male	27.0	0	0	315154	8.6625	0	S	0.0
4	896	3	female	22.0	1	1	3101298	12.2875	0	S	2.0

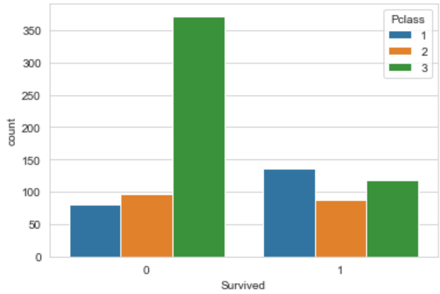
The **Cabin** column has been encoded.

3. Passenger Class Vs Survival Rate :

Let us vizualise the data to see if the class of the passenger impacts the survival rate or not:

In [25]:
```
sns.set_style("whitegrid")
sns.countplot(x = "Survived", hue = "Pclass", data = train)
```

Out[25]: `<AxesSubplot:xlabel='Survived', ylabel='count'>`



From the bar plot it is clear that the passengers from **First Class** had a higher chance of surviving.
Majority of the passengers in **Third Class** did not survive. Therefore, Passenger Class is an important factor while predicting the survival rate of the passengers.

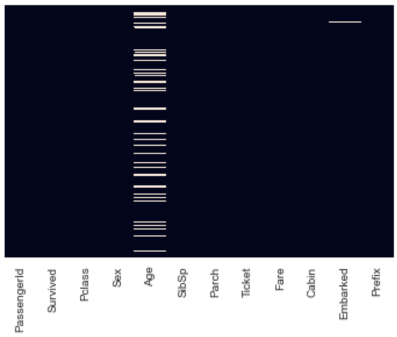
4. Age

we use data visulaization to see how age influences the survival rate of the passengers : -

We check to see if Age has any null values -

In [26]:
```
sns.heatmap(train.isnull(), yticklabels = False, cbar = False)
```

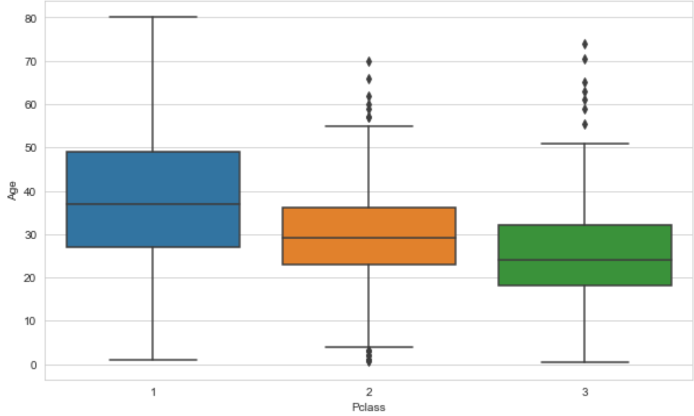
Out[26]: `<AxesSubplot:>`



Age has a few null values which should be filled up.
We can fill the null values using the average age of the passengers in each Passenger class.

In [27]:
```
plt.figure(figsize=(10,6))
sns.boxplot(x="Pclass",y="Age",data=train)
```

Out[27]: `<AxesSubplot:xlabel='Pclass', ylabel='Age'>`



In [28]:

```
avg_first = train["Age"][train["Pclass"]==1].mean()
avg_second = train["Age"][train["Pclass"]==2].mean()
avg_third = train["Age"][train["Pclass"]==3].mean()
print("Average age for First class Passenger : ",avg_first)
print("Average age for Second class Passenger : ",avg_second)
print("Average age for Third class Passenger : ",avg_third)
```

```
Average age for First class Passenger :  38.233440860215055
Average age for Second class Passenger :  29.87763005780347
Average age for Third class Passenger :  25.14061971830986
```

In [29]:

```
for data in combined_data:
    data["Age"] = data["Age"].fillna(0)

for i in range(891):
    if(train.at[i,"Age"]==0):
        if(train.at[i,"Pclass"]==1):
            train.at[i,"Age"]=avg_first
        elif(train.at[i,"Pclass"]==2):
            train.at[i,"Age"]=avg_second
        else:
            train.at[i,"Age"]=avg_third
for i in range(418):
    if(test.at[i,"Age"]==0):
        if(test.at[i,"Pclass"]==1):
            test.at[i,"Age"]=avg_first
        elif(test.at[i,"Pclass"]==2):
            test.at[i,"Age"]=avg_second
        else:
            test.at[i,"Age"]=avg_third
```

In [30]:

```
train.describe()
```

Out[30]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	Prefix
count	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.292875	0.523008	0.381594	32.204208	0.567901
std	257.353842	0.486592	0.836071	13.210527	1.102743	0.806057	49.693429	0.826963
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	22.000000	0.000000	0.000000	7.910400	0.000000
50%	446.000000	0.000000	3.000000	26.000000	0.000000	0.000000	14.454200	0.000000
75%	668.500000	1.000000	3.000000	37.000000	1.000000	0.000000	31.000000	1.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200	3.000000

In [31]:

```
test.describe()
```

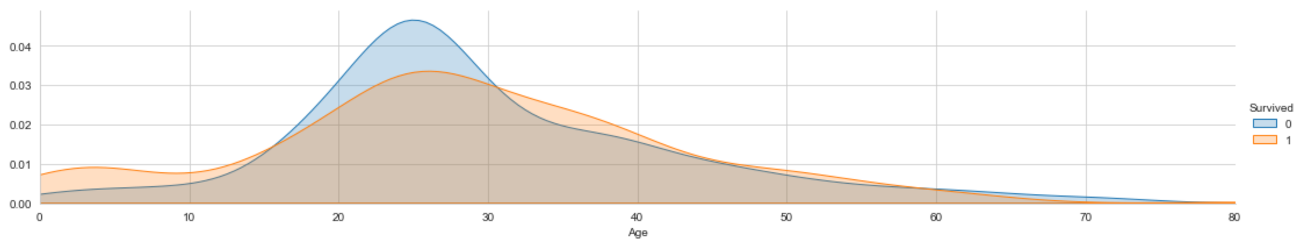
Out[31]:

	PassengerId	Pclass	Age	SibSp	Parch	Fare	Prefix
count	418.000000	418.000000	418.000000	418.000000	418.000000	417.000000	418.000000
mean	1100.500000	2.265550	29.555296	0.447368	0.392344	35.627188	0.576555
std	120.810458	0.841838	12.846509	0.896760	0.981429	55.907576	0.822423
min	892.000000	1.000000	0.170000	0.000000	0.000000	0.000000	0.000000
25%	996.250000	1.000000	23.000000	0.000000	0.000000	7.895800	0.000000
50%	1100.500000	3.000000	25.140620	0.000000	0.000000	14.454200	0.000000
75%	1204.750000	3.000000	36.375000	1.000000	0.000000	31.500000	1.000000
max	1309.000000	3.000000	76.000000	8.000000	9.000000	512.329200	3.000000

In [32]:

```
fac = sns.FacetGrid(train,hue = "Survived", aspect = 5)
fac.map(sns.kdeplot,'Age',shade=True)
fac.set(xlim=(0,train["Age"].max()))
fac.add_legend()
```

Out[32]: <seaborn.axisgrid.FacetGrid at 0x1c4dd0f2c70>

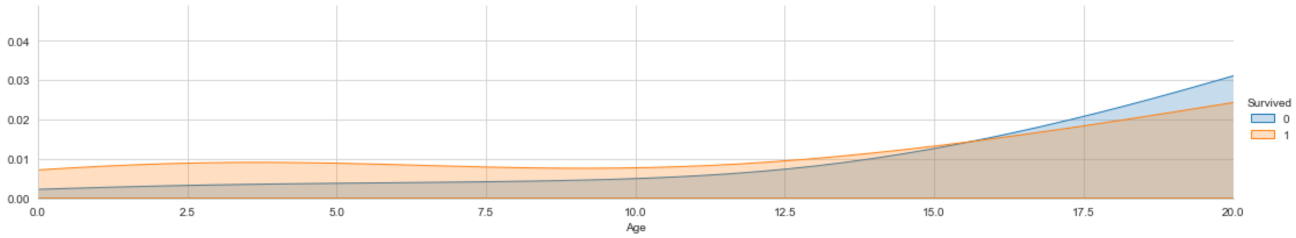


We look closely at the different age ranges:

Age range : 0 - 20

In [33]:
```
fac = sns.FacetGrid(train,hue = "Survived", aspect = 5)
fac.map(sns.kdeplot,'Age',shade=True)
fac.set(xlim=(0,train["Age"].max()))
fac.add_legend()
plt.xlim(0,20)
```

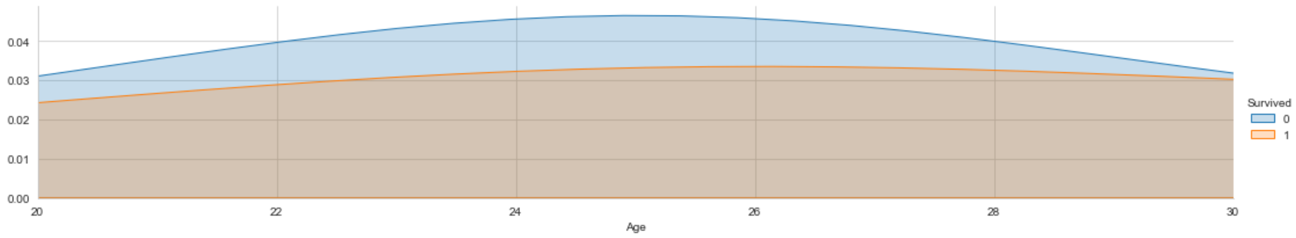
Out[33]: (0.0, 20.0)



Age range : 20 - 30

In [34]:
```
fac = sns.FacetGrid(train,hue = "Survived", aspect = 5)
fac.map(sns.kdeplot,'Age',shade=True)
fac.set(xlim=(0,train["Age"].max()))
fac.add_legend()
plt.xlim(20,30)
```

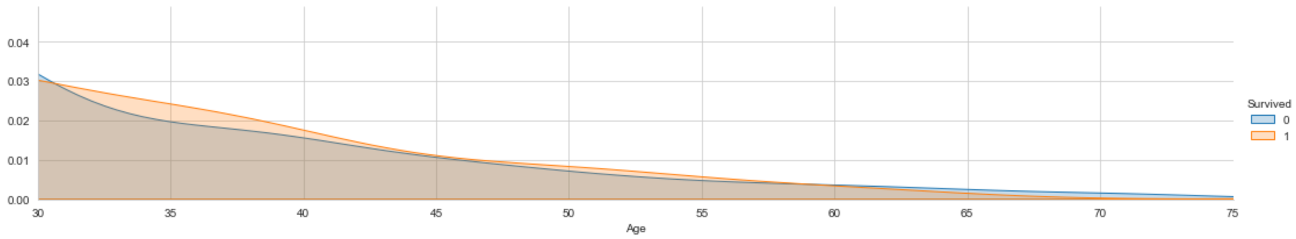
Out[34]: (20.0, 30.0)



Age range : 30 above

In [35]:
```
fac = sns.FacetGrid(train,hue = "Survived", aspect = 5)
fac.map(sns.kdeplot,'Age',shade=True)
fac.set(xlim=(0,train["Age"].max()))
fac.add_legend()
plt.xlim(30,75)
```

Out[35]: (30.0, 75.0)



Observation :

1. Younger people, age 0 - 20 are more likely to survive than to die
2. People in the age group of 20 - 30 are more likely to die
3. Older people will more likely survive

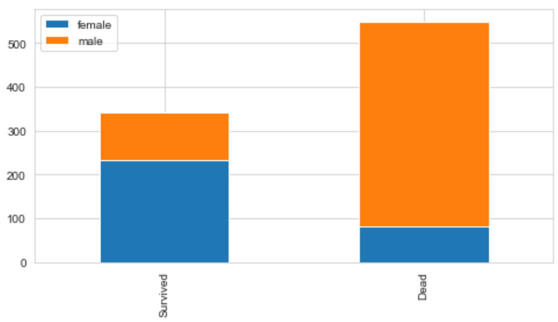
In general, the young adult passengers have the highest probablity of dying compard to children and older adults

5. Sex v/s Survival Rate :

We try to see if the sex of a passenger has anything to do with their survival rate.

In [36]:
```
survived = train[train["Survived"]==1]["Sex"].value_counts()
dead = train[train["Survived"]==0]["Sex"].value_counts()
df_sex = pd.DataFrame([survived,dead])
df_sex.index = ["Survived","Dead"]
df_sex.plot(kind="bar",stacked = True, figsize = (8,4))
```

Out[36]: <AxesSubplot:>



Observation : Male passengers have a higher chance of dying compared to female passengers.

We can encode the data in the following way :

Female : 1

Male : 0

In [37]:

```
dummy = pd.get_dummies(train["Sex"])
dummy.head()
```

Out[37]:

	female	male
0	0	1
1	1	0
2	1	0
3	1	0
4	0	1

In [38]:

```
train["Sex"] = dummy["female"]
train.head()
```

Out[38]:

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Prefix
0	1	0	3	0	22.0	1	0	A/5 21171	7.2500	0	S	0
1	2	1	1	1	38.0	1	0	PC 17599	71.2833	1	C	2
2	3	1	3	1	26.0	0	0	STON/O2. 3101282	7.9250	0	S	1
3	4	1	1	1	35.0	1	0	113803	53.1000	1	S	2
4	5	0	3	0	35.0	0	0	373450	8.0500	0	S	0

In [39]:

```
dummy2 = pd.get_dummies(test["Sex"])
test["Sex"] = dummy2["female"]
test.head()
```

Out[39]:

	PassengerId	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Prefix
0	892	3	0	34.5	0	0	330911	7.8292	0	Q	0.0
1	893	3	1	47.0	1	0	363272	7.0000	0	S	2.0
2	894	2	0	62.0	0	0	240276	9.6875	0	Q	0.0
3	895	3	0	27.0	0	0	315154	8.6625	0	S	0.0
4	896	3	1	22.0	1	1	3101298	12.2875	0	S	2.0

6. Embarked V/s Survival Rate

The **Embarked** column has 3 categorical values. To use this column for data analysis, we will have to encode them. We can create 2 additinal columns for the **Embarked** values as these two columns can depict the 3 categorical values.

In [40]:

```
emb_dummies = pd.get_dummies(train["Embarked"])
emb_dummies.head()
```

Out[40]:

	C	Q	S
0	0	0	1
1	1	0	0
2	0	0	1
3	0	0	1
4	0	0	1

In [41]:

```
train["Q"] = emb_dummies["Q"]
train["S"] = emb_dummies["S"]
train.drop(columns="Embarked",inplace = True)
train.head()
```

Out[41]:

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Prefix	Q	S
0	1	0	3	0	22.0	1	0	A/5 21171	7.2500	0	0	0	1
1	2	1	1	1	38.0	1	0	PC 17599	71.2833	1	2	0	0
2	3	1	3	1	26.0	0	0	STON/O2. 3101282	7.9250	0	1	0	1
3	4	1	1	1	35.0	1	0	113803	53.1000	1	2	0	1
4	5	0	3	0	35.0	0	0	373450	8.0500	0	0	0	1

In [42]:

```
emb_dumm = pd.get_dummies(test["Embarked"])
test["Q"] = emb_dumm["Q"]
test["S"] = emb_dumm["S"]
test.drop(columns="Embarked",inplace = True)
test.head()
```

Out[42]:

	PassengerId	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Prefix	Q	S
0	892	3	0	34.5	0	0	330911	7.8292	0	0.0	1	0
1	893	3	1	47.0	1	0	363272	7.0000	0	2.0	0	1
2	894	2	0	62.0	0	0	240276	9.6875	0	0.0	1	0
3	895	3	0	27.0	0	0	315154	8.6625	0	0.0	0	1
4	896	3	1	22.0	1	1	3101298	12.2875	0	2.0	0	1

7. PassengerId Vs Survival Rate :

The ID of the passenger has nothing to do with the survival rate as it is a unique value for each passenger.
We drop this value.

In [43]:
```python
train.drop(columns = "PassengerId", inplace = True)
test.drop(columns = "PassengerId", inplace = True)
train.head()
```

Out[43]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Prefix	Q	S
0	0	3	0	22.0	1	0	A/5 21171	7.2500	0	0	0	1
1	1	1	1	38.0	1	0	PC 17599	71.2833	1	2	0	0
2	1	3	1	26.0	0	0	STON/O2. 3101282	7.9250	0	1	0	1
3	1	1	1	35.0	1	0	113803	53.1000	1	2	0	1
4	0	3	0	35.0	0	0	373450	8.0500	0	0	0	1

8. Ticket Vs Survival Rate :

The Ticket of the passenger has nothing to do with the survival rate. Ticket values will have a large number of categorical values which do not provide us with the right information to predict wheather a passenger survived or not.
People will likely have different Ticket Numbers.
We drop this value.

In [44]:
```python
train.drop(columns = "Ticket", inplace = True)
test.drop(columns = "Ticket", inplace = True)
train.head()
```

Out[44]:

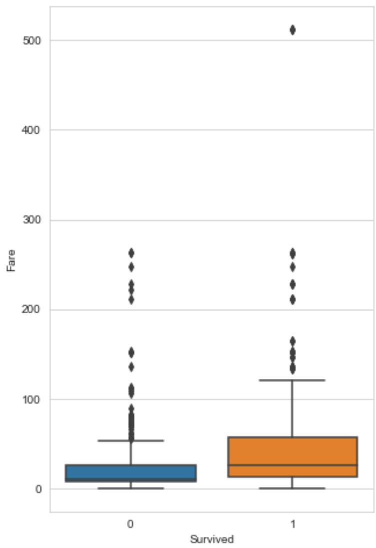
	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Cabin	Prefix	Q	S
0	0	3	0	22.0	1	0	7.2500	0	0	0	1
1	1	1	1	38.0	1	0	71.2833	1	2	0	0
2	1	3	1	26.0	0	0	7.9250	0	1	0	1
3	1	1	1	35.0	1	0	53.1000	1	2	0	1
4	0	3	0	35.0	0	0	8.0500	0	0	0	1

9. Fare v/s Survival Rate :

We analyze the amount of money each passenger paid for their ticket and try to find if this affects the survival rate of the passenger.

In [45]:
```python
plt.figure(figsize=(5,8))
sns.boxplot(x="Survived",y="Fare",data=train)
```

Out[45]: <AxesSubplot:xlabel='Survived', ylabel='Fare'>



Observation :

The average money spent on the ticket was more for the passengers that survived.

In [46]:
```python
train["Fare"].isnull().value_counts()
```

Out[46]:
```
False    891
Name: Fare, dtype: int64
```

In [47]:
```python
test["Fare"].isnull().value_counts()
```

Out[47]:
```
False    417
True       1
Name: Fare, dtype: int64
```

The test dataset has one null value.
We can replace this with the average fare of the people in the same passenger class.

In [48]:
```python
fare_first = train["Fare"][train["Pclass"]==1].mean()
fare_second = train["Fare"][train["Pclass"]==2].mean()
fare_third = train["Fare"][train["Pclass"]==3].mean()
print("Average Fare for First class Passenger : ",fare_first)
print("Average Fare for Second class Passenger : ",fare_second)
print("Average Fare for Third class Passenger : ",fare_third)
```

```
Average Fare for First class Passenger :  84.15468749999992
Average Fare for Second class Passenger :  20.66218315217391
Average Fare for Third class Passenger :  13.675550101832997
```

In [49]:
```python
test["Fare"] = test["Fare"].fillna(0)
```

```python
for i in range(418):
    if(test.at[i,"Fare"]==0):
        if(test.at[i,"Pclass"]==1):
            test.at[i,"Fare"]=fare_first
        elif(test.at[i,"Pclass"]==2):
            test.at[i,"Fare"]=fare_second
        else:
            test.at[i,"Fare"]=fare_third

test["Fare"].isnull().value_counts()
```

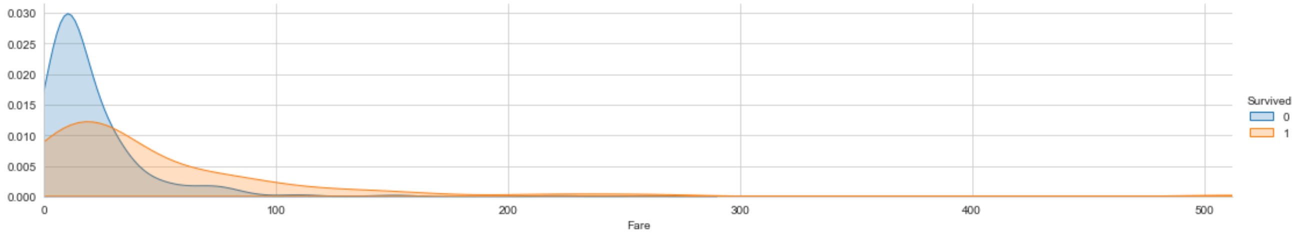
Out[49]:

```
False    418
Name: Fare, dtype: int64
```

In [50]:

```python
fac = sns.FacetGrid(train,hue = "Survived", aspect = 5)
fac.map(sns.kdeplot,'Fare',shade=True)
fac.set(xlim=(0,train["Fare"].max()))
fac.add_legend()
```

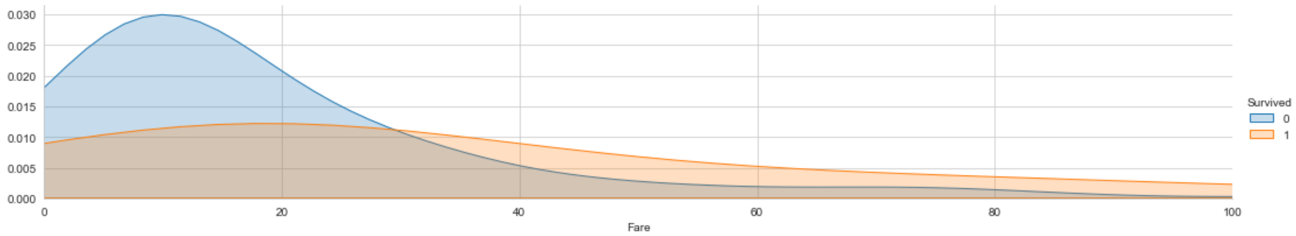
Out[50]: `<seaborn.axisgrid.FacetGrid at 0x1c4de337d60>`



In [51]:

```python
fac = sns.FacetGrid(train,hue = "Survived", aspect = 5)
fac.map(sns.kdeplot,'Fare',shade=True)
fac.set(xlim=(0,train["Fare"].max()))
fac.add_legend()
plt.xlim(0,100)
```

Out[51]: `(0.0, 100.0)`



People who paid a lower fare were most likely from Second or Third class and therefore had a lesser chance of surviving.

10. SibSp and Parch V/s Survival Rate :

We first add the values of SibSp and Parch as both represent the same thing, i.e. family size:

In [52]:

```python
train["Family"] = train["SibSp"] + train["Parch"] + 1
test["Family"] = test["SibSp"] + test["Parch"] + 1

for data in combined_data:
    data.drop(columns = ["SibSp","Parch"],inplace =True)

train.head()
```

Out[52]:

	Survived	Pclass	Sex	Age	Fare	Cabin	Prefix	Q	S	Family
0	0	3	0	22.0	7.2500	0	0	0	1	2
1	1	1	1	38.0	71.2833	1	2	0	0	2
2	1	3	1	26.0	7.9250	0	1	0	1	1
3	1	1	1	35.0	53.1000	1	2	0	1	2
4	0	3	0	35.0	8.0500	0	0	0	1	1

In [53]:

```python
test.head()
```

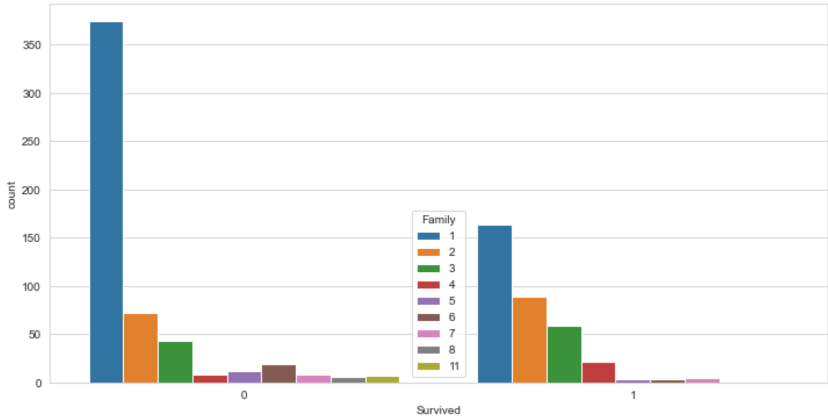
Out[53]:

	Pclass	Sex	Age	Fare	Cabin	Prefix	Q	S	Family
0	3	0	34.5	7.8292	0	0.0	1	0	1
1	3	1	47.0	7.0000	0	2.0	0	1	2
2	2	0	62.0	9.6875	0	0.0	1	0	1
3	3	0	27.0	8.6625	0	0.0	0	1	1
4	3	1	22.0	12.2875	0	2.0	0	1	3

In [54]:

```python
plt.figure(figsize = (12,6))
sns.set_style("whitegrid")
sns.countplot(x = "Survived", hue = "Family", data = train)
```

Out[54]: `<AxesSubplot:xlabel='Survived', ylabel='count'>`



Observation :

People whoe were travelling with someone had more chnaces to survive compared to those who were travelling alone.

Feature Scaling and Train - Test - Split

Before we make our model using various algorithms, we should scale the data.

In [57]:
```
X = train[["Pclass","Sex","Age","Fare","Cabin","Prefix","Q","S","Family"]]
Y = train["Survived"]
X_TEST = test[["Pclass","Sex","Age","Fare","Cabin","Prefix","Q","S","Family"]]
```

In [58]:
```
print(X)
```

```
     Pclass  Sex       Age     Fare Cabin  Prefix  Q  S  Family
0         3    0  22.00000   7.2500     0       0  0  1       2
1         1    1  38.00000  71.2833     1       2  0  0       2
2         3    1  26.00000   7.9250     0       1  0  1       1
3         1    1  35.00000  53.1000     1       2  0  1       2
4         3    0  35.00000   8.0500     0       0  0  1       1
..      ...  ...       ...      ...   ...     ... .. ..     ...
886       2    0  27.00000  13.0000     0       3  0  1       1
887       1    1  19.00000  30.0000     1       1  0  1       1
888       3    1  25.14062  23.4500     0       1  0  1       4
889       1    0  26.00000  30.0000     1       0  0  0       1
890       3    0  32.00000   7.7500     0       0  1  0       1

[891 rows x 9 columns]
```

In [59]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,Y,test_size = 0.2, random_state=1)
```

Making the Model

Decision Tree Algorithm

We try out the Decision Tree algorithm for this classification problem.

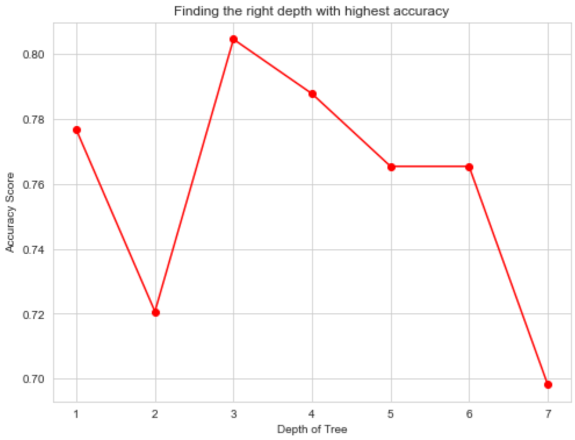
In [61]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

depth = [];

for i in range(1,8):
    clf_tree = DecisionTreeClassifier(criterion="entropy", random_state = 100, max_depth = i)
    clf_tree.fit(X_train,y_train)
    yhat = clf_tree.predict(X_test)
    depth.append(accuracy_score(y_test,yhat))
    print("For max depth = ",i, " : ",accuracy_score(y_test,yhat))
```

```
For max depth =  1  :  0.776536312849162
For max depth =  2  :  0.7206703910614525
For max depth =  3  :  0.8044692737430168
For max depth =  4  :  0.7877094972067039
For max depth =  5  :  0.7653631284916201
For max depth =  6  :  0.7653631284916201
For max depth =  7  :  0.6983240223463687
```

In [62]:
```
plt.figure(figsize=(8,6))
plt.plot(range(1,8),depth,color="red", marker = "o")
plt.xlabel("Depth of Tree")
plt.ylabel("Accuracy Score")
plt.title("Finding the right depth with highest accuracy")
plt.xticks(range(1,8))
plt.show()
```



Highest accuracy is obtained with depth = 3.

predictions are:

In [70]:
```
clf_tr = DecisionTreeClassifier(criterion="entropy", random_state = 100, max_depth = 3)
clf_tr.fit(X,Y)
pred_tree = clf_tr.predict(X_TEST)
print(pred_tree)
```

```
[0 1 0 0 1 0 1 0 1 0 0 0 1 0 1 1 0 0 1 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 0 0 1
 1 0 0 0 0 0 1 1 0 0 0 1 1 0 0 1 1 0 0 0 0 0 1 0 0 0 1 1 1 1 0 0 1 1 0 1 0
 1 0 0 1 0 1 0 0 0 0 0 0 1 1 1 0 1 0 1 0 0 0 1 0 1 0 1 0 0 0 1 0 0 0 0 0 0
 1 1 1 1 0 0 1 0 1 1 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0
 0 0 1 0 0 1 0 0 1 1 0 1 1 0 1 0 0 1 0 0 1 1 0 0 0 0 0 1 1 0 1 1 0 0 1 0 1
 0 1 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 1 0 0 1 0 1 0 0 0 0 1 1 0 1 0 1 0 1 0
 1 0 1 1 0 1 0 0 0 1 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 0 1 1 1 0 0 0 0 0 0 0 1
 0 0 0 1 1 0 0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 1 0 1 1 1 0 0 0 0 0 0 1 0 0 0 0
 1 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 1 1 1 0 0 0 0 0 0 0 1 0 1 0 0 0 1 0 0
 1 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 1 0 0 0 1 0 1 0 0 1 0 1 1 0 1 0 0 0 1 0
 0 1 0 0 1 1 1 0 0 0 0 0 1 1 0 1 0 0 0 0 0 1 0 0 0 1 0 1 0 0 1 0 1 0 0 0 0
 0 1 1 1 1 1 0 1 0 0 0]
```

In [81]:
```
test_accuracy = clf_tr.score(X_test, y_test)
print(test_accuracy)
```

```
0.7988826815642458
```

Naive Bayes Algorithm

We try out the Naive Bayes Algorithm for this classification problem.

In [78]:
```
from sklearn.naive_bayes import GaussianNB
clf_NB = GaussianNB()
clf_NB.fit(X_train,y_train)
y_hat = clf_NB.predict(X_test)
print("Accuracy for training data : ",accuracy_score(y_test,y_hat))
```

```
Accuracy for training data :  0.7430167597765364
```

In [79]:
```
clf_NB = GaussianNB()
clf_NB.fit(X,Y)
pred_NB = clf_NB.predict(X_TEST)
pred_NB
```

Out[79]:
```
array([0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0,
       1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1,
       1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1,
       1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1,
       1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1,
       0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1,
       1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
       0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0,
       1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1,
       1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0,
       0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0,
       1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0,
       0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0,
       1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0],
      dtype=int64)
```

In [82]:
```
test_accuracy = clf_NB.score(X_test, y_test)
print(test_accuracy)
```

```
0.7486033519553073
```

In [ ]:
```
```