THE CHALLENGE

The sinking of the Titanic is one of the most infamous shipwrecks in history. On April 15, 1912, during her maiden voyage, the widely considered “unsinkable” RMS Titanic sank after colliding with an iceberg. Unfortunately, there weren’t enough lifeboats for everyone onboard, resulting in the death of 1502 out of 2224 passengers and crew. While there was some element of luck involved in surviving, it seems some groups of people were more likely to survive than others.

In this challenge, we will be building a predictive model that answers the question: “*what sorts of people were more likely to survive?*” using passenger data (ie name, age, gender, socio-economic class, etc).

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

```
import seaborn as sns
%matplotlib inline

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
import warnings
warnings.filterwarnings('ignore')

color_pal = sns.color_palette()
sns.set()
```

In [2]:
```
raw_df = pd.read_csv('../Titanic - Machine Learning from Disaster/train.csv')
train_df = raw_df.drop(['Survived'], axis=1)
test_df = pd.read_csv('../Titanic - Machine Learning from Disaster/test.csv')
```

In [3]:
```
raw_df
```

Out[3]:

	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
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

DATASET DESCRIPTION

The data has been split into two groups:

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

The training set would be used to build 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

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

Variable Notes

pclass: A proxy for socio-economic status (SES)

- 1st = Upper
- 2nd = Middle
- 3rd = Lower

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

sibsp: The dataset defines family relations in this way...

- Sibling = brother, sister, stepbrother, stepsister
- Spouse = husband, wife (mistresses and fiancés were ignored)

parch: The dataset defines family relations in this way...

- Parent = mother, father
- Child = daughter, son, stepdaughter, stepson *Some children travelled only with a nanny, therefore parch=0 for them*

In [4]:
```
raw_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

In [5]:
```
raw_df.isnull().sum()
```

Out[5]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

In [6]:
```
print('Shape of training file:', train_df.shape)
```

```
print('Shape of test file:', test_df.shape)
```

```
Shape of training file: (891, 11)
Shape of test file: (418, 11)
```

In [7]:
```
c_df = pd.concat([train_df, test_df], axis=0)
c_df.head()

# concatenating the training and test file into a consolidated dataframe
# this will help us during the categorical transformation so that test dataframe will have the exact same columns
```

Out[7]:

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

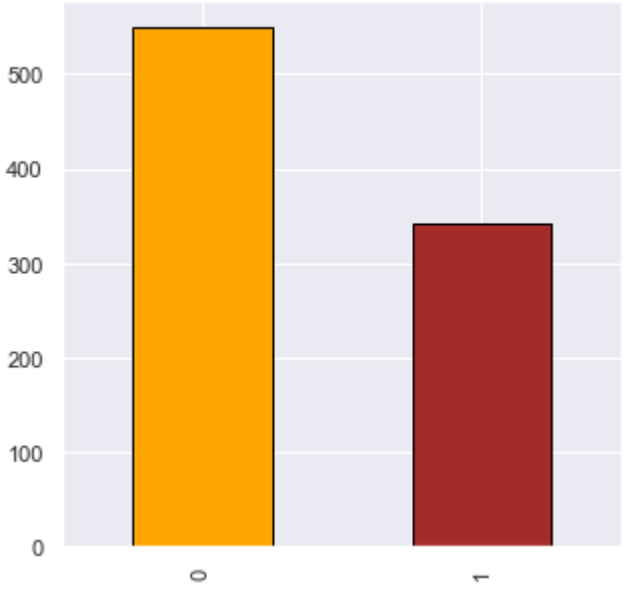
In [8]:
```
test_df.isnull().sum()
```

Out[8]:
```
PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

In [9]:
```
raw_df['Survived'].value_counts().plot(kind='bar', figsize=(5,5),
                                       color=['orange', 'brown'],
                                       edgecolor='black')

print('Survival Stats for Raw Dataframe:')
print('No. of passengers who had survived:', raw_df[raw_df['Survived'] == 1]['Survived'].count())
```

```
print('Percentage of passengers who had survived:', round((raw_df[raw_df['Survived'] == 1]['Survived'].count()/len(raw_
print('No. of passengers who failed to survive:', raw_df[raw_df['Survived'] == 0]['Survived'].count())
print('Percentage of passengers who failed survived:', round((raw_df[raw_df['Survived'] == 0]['Survived'].count()/len(r
```

```
Survival Stats for Raw Dataframe:
No. of passengers who had survived: 342
Percentage of passengers who had survived: 38.38
No. of passengers who failed to survive: 549
Percentage of passengers who failed survived: 61.62
```



In [10]:
```
# analyzing the missing values for the consolidated dataframe

print('Percentage of values missing under \'Age\' feature:', (c_df['Age'].isnull().sum()/c_df.shape[0])*100)
print('Percentage of values missing under \'Fare\' feature:', (c_df['Fare'].isnull().sum()/c_df.shape[0])*100)
print('Percentage of values missing under \'Cabin\' feature:', (c_df['Cabin'].isnull().sum()/c_df.shape[0])*100)
print('Percentage of values missing under \'Embarked\' feature:', (c_df['Embarked'].isnull().sum()/c_df.shape[0])*100)
```

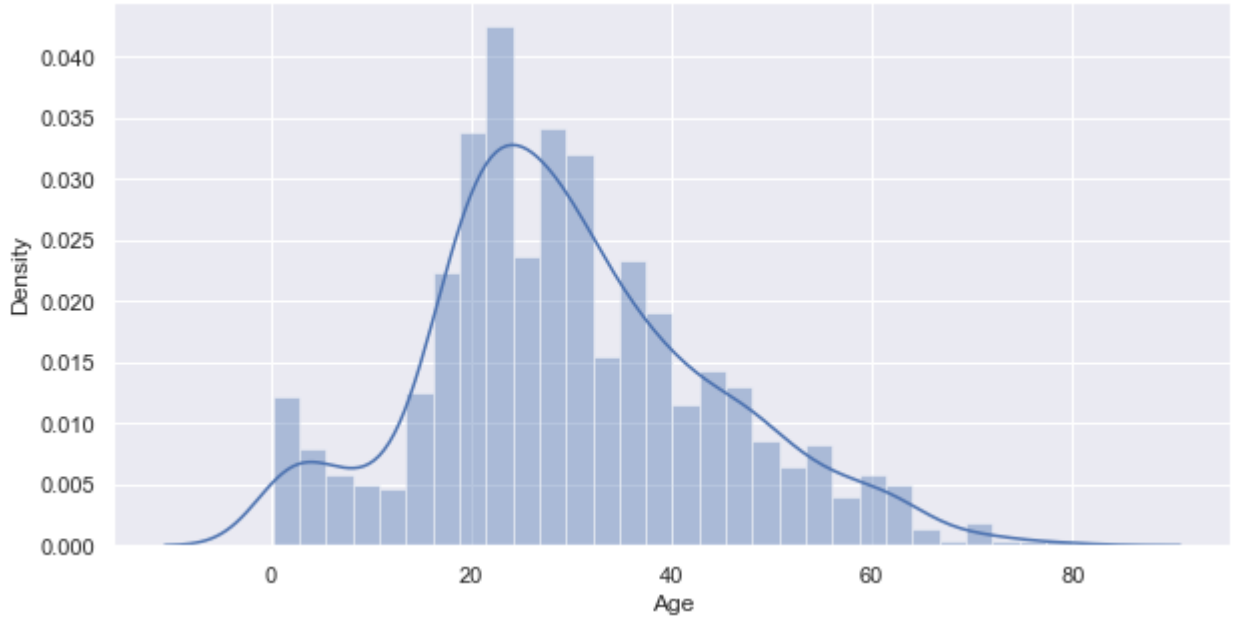
```
Percentage of values missing under 'Age' feature: 20.091673032849503
Percentage of values missing under 'Fare' feature: 0.07639419404125286
Percentage of values missing under 'Cabin' feature: 77.46371275783041
Percentage of values missing under 'Embarked' feature: 0.15278838808250572
```

LET'S ANALYZE THESE MISSING VALUE FEATURES ONE BY ONE

In [11]:
```
plt.figure(figsize=(10,5))
sns.distplot(a=c_df['Age'],
```

```
            bins=30,
            hist=True,
            kde=True)
```

Out[11]: <AxesSubplot:xlabel='Age', ylabel='Density'>



In [12]:
```
print('Most common age that feature under the consolidated dataframe:', c_df['Age'].mode()[0])
print('Average age for the consolidated dataframe:', c_df['Age'].mean())
print('Median age for the consolidated dataframe:', c_df['Age'].median())
```

```
Most common age that feature under the consolidated dataframe: 24.0
Average age for the consolidated dataframe: 29.881137667304014
Median age for the consolidated dataframe: 28.0
```

In [13]:
```
raw_df.describe()
```

Out[13]:

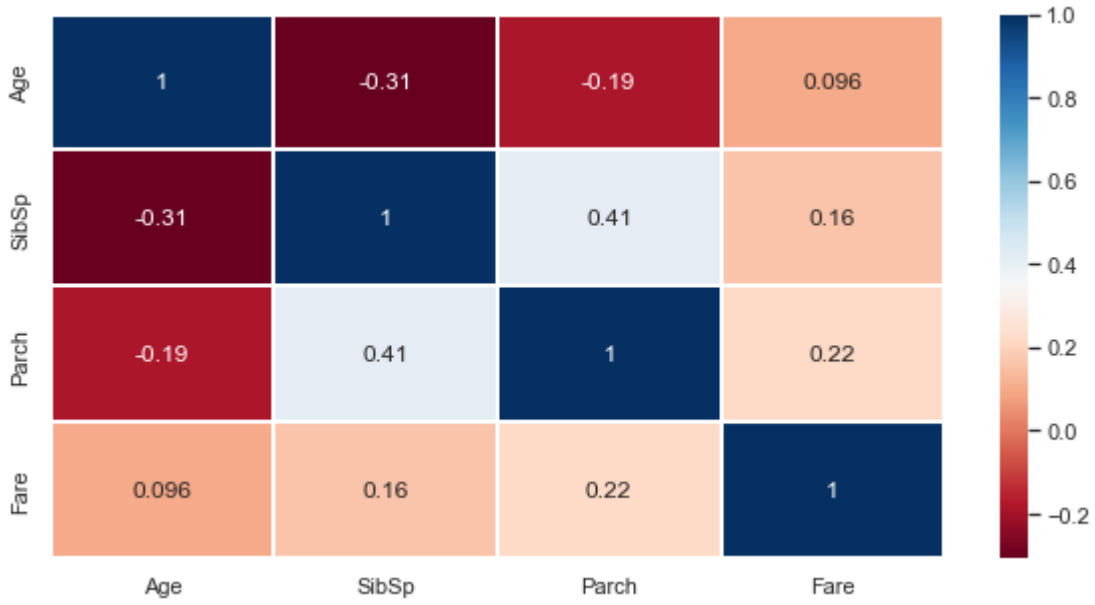
	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

In [14]:
```
# splitting the numeric and categorical values
df_numeric = raw_df[['Age','SibSp','Parch','Fare']]
df_categorical = raw_df[['Survived','Pclass','Sex','Ticket','Cabin','Embarked']]
```

In [15]:
```
# visualizing the heatmap for numeric columns

plt.figure(figsize=(10,5))
print(df_numeric.corr())
sns.heatmap(df_numeric.corr(), cmap='RdBu', annot=True, linewidths=2, linecolor='white')
```

```
             Age     SibSp     Parch      Fare
Age     1.000000 -0.308247 -0.189119  0.096067
SibSp  -0.308247  1.000000  0.414838  0.159651
Parch  -0.189119  0.414838  1.000000  0.216225
Fare    0.096067  0.159651  0.216225  1.000000
```

Out[15]:
```
<AxesSubplot:>
```



In [16]:
```
# comparing survival rate across Age, SibSp, Parch, and Fare
pd.pivot_table(raw_df, index = 'Survived', values = ['Age','SibSp','Parch','Fare'])
```

Out[16]:

Survived	Age	Fare	Parch	SibSp
0	30.626179	22.117887	0.329690	0.553734
1	28.343690	48.395408	0.464912	0.473684

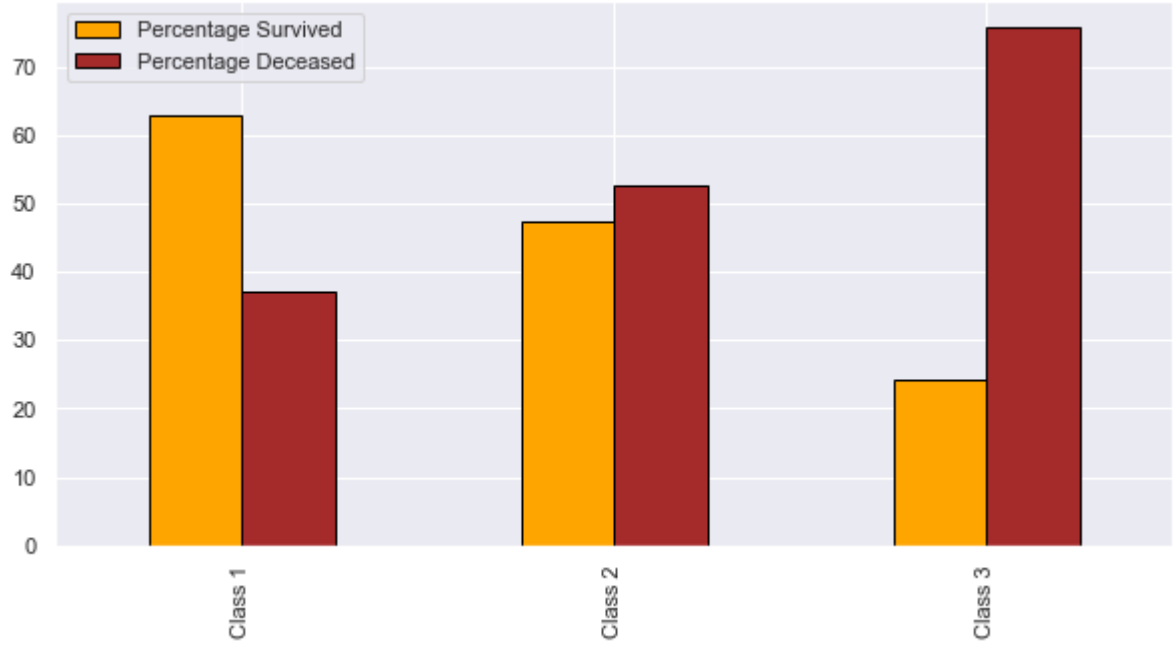
In [17]:
```
# survivor distribution based on ticket class

class1_survivor = round((raw_df[raw_df.Pclass == 1].Survived == 1).value_counts()[1]/len(raw_df[raw_df.Pclass == 1]) *
class2_survivor = round((raw_df[raw_df.Pclass == 2].Survived == 1).value_counts()[1]/len(raw_df[raw_df.Pclass == 2]) *
class3_survivor = round((raw_df[raw_df.Pclass == 3].Survived == 1).value_counts()[1]/len(raw_df[raw_df.Pclass == 3]) *

pclass_perc_df = pd.DataFrame(
    { "Percentage Survived":{"Class 1": class1_survivor,"Class 2": class2_survivor, "Class 3": class3_survivor},
     "Percentage Deceased":{"Class 1": 100-class1_survivor,"Class 2": 100-class2_survivor, "Class 3": 100-class3_surviv
pclass_perc_df
```

Out[17]:

	Percentage Survived	Percentage Deceased
Class 1	62.96	37.04
Class 2	47.28	52.72
Class 3	24.24	75.76

In [18]:
```
pclass_perc_df.plot(kind='bar', figsize=(10,5), color=['orange', 'brown'], edgecolor='black')
```

Out[18]:
```
<AxesSubplot:>
```



NOTE:

> If we visualize the graph above, it provides us a valuable insight to help reason out that the percentage of survivors is greater in case of passengers who had 'Class 1' ticket while the percentage of deceased is the maximum in case of passengers having 'Class 3' ticket. This makes us believe that the passengers having Class 1/2 ticket were prioritzed over the other passengers having Class 3 ticket and were provided the first preference to aboard the rescue boats.

In [19]:
```
# plotting age density for all the 3 ticket classes
```

```
for x in [1,2,3]:
    raw_df[raw_df.Pclass == x]['Age'].plot(kind="kde", figsize=(10,5))

plt.title("Age density based on ticket class", pad=4.5)
plt.legend(("Class 1","Class 2","Class 3"))
```

Out[19]:
```
<matplotlib.legend.Legend at 0x1c0119ae7c0>
```

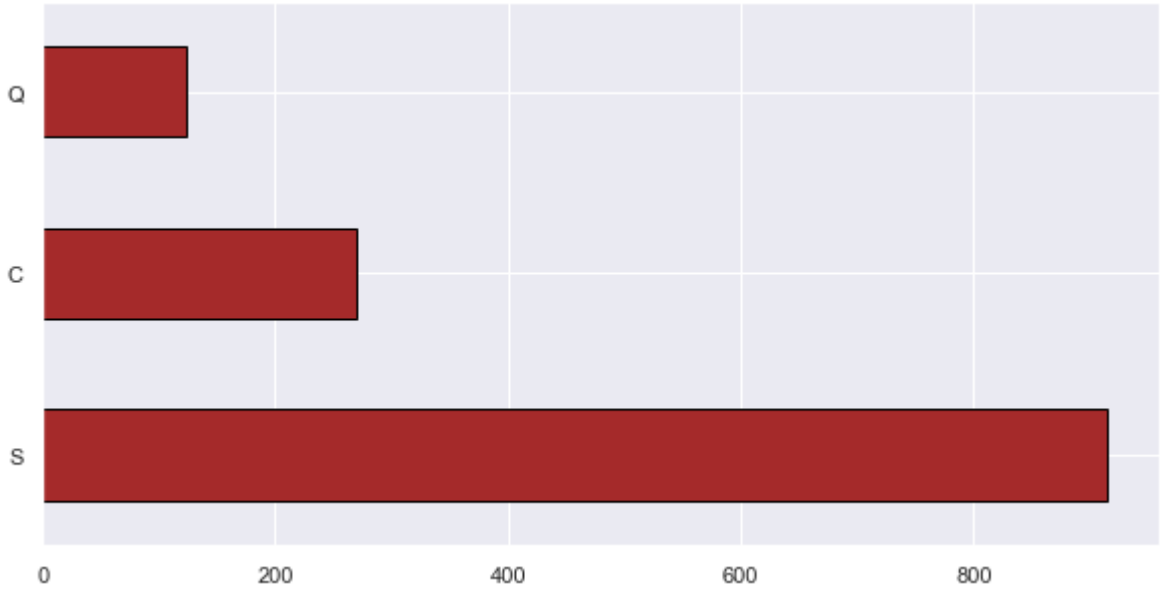
In [20]:
```
# since ~20% of the data under 'Age' feature is missing, we would be replacing the missing values with the median value
# since there is 1 missing value under 'Fare' feature in test dataframe, we would be replacing it with the median value

c_df['Age'] = c_df['Age'].fillna(value=c_df['Age'].median())
c_df['Fare'] = c_df['Fare'].fillna(value=c_df['Fare'].median())
```

In [21]:
```
# now let's go ahead and analyze the 'Embarked' feature
# 'Embarked' feature is a categorical feature having 2 categories (S/C/Q), with value missing for 2 passengers

c_df['Embarked'].value_counts()
```

Out[21]:
```
S    914
C    270
Q    123
Name: Embarked, dtype: int64
```

In [22]:
```
c_df['Embarked'].value_counts().plot(kind='barh', figsize=(10,5), color='brown', edgecolor='black')
```

Out[22]:
```
<AxesSubplot:>
```



Since ~0.2% of the data under 'Embarked' feature is missing we would be replacing the missing values with the mode value

NOTE: Since it's a categorical feature (string) we would be replacing the missing values with the most frequent value

In [23]:
```
c_df.loc[c_df['Embarked'].isnull()]
```

Out[23]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
61	62	1	Icard, Miss. Amelie	female	38.0	0	0	113572	80.0	B28	NaN
829	830	1	Stone, Mrs. George Nelson (Martha Evelyn)	female	62.0	0	0	113572	80.0	B28	NaN

There are just two missing values for 'Embarked' feature. Analyzing the same, we could reason out that both individuals had paid a similar amount, also had a Class 1 ticket and were on the same cabin. In addition to above, both of them are female individuals. So let's compare the same with other individuals onboard with similar fare, cabin and ticket class.

In [24]:
```
c_df[c_df['Pclass']==1 & c_df['Sex'].eq('female')].head(20)
```

Out[24]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	2	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
3	4	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
11	12	1	Bonnell, Miss. Elizabeth	female	58.0	0	0	113783	26.5500	C103	S
31	32	1	Spencer, Mrs. William Augustus (Marie Eugenie)	female	28.0	1	0	PC 17569	146.5208	B78	C
52	53	1	Harper, Mrs. Henry Sleeper (Myna Haxtun)	female	49.0	1	0	PC 17572	76.7292	D33	C
61	62	1	Icard, Miss. Amelie	female	38.0	0	0	113572	80.0000	B28	NaN
88	89	1	Fortune, Miss. Mabel Helen	female	23.0	3	2	19950	263.0000	C23 C25 C27	S
136	137	1	Newsom, Miss. Helen Monypeny	female	19.0	0	2	11752	26.2833	D47	S
151	152	1	Pears, Mrs. Thomas (Edith Wearne)	female	22.0	1	0	113776	66.6000	C2	S
166	167	1	Chibnall, Mrs. (Edith Martha Bowerman)	female	28.0	0	1	113505	55.0000	E33	S
177	178	1	Isham, Miss. Ann Elizabeth	female	50.0	0	0	PC 17595	28.7125	C49	C
194	195	1	Brown, Mrs. James Joseph (Margaret Tobin)	female	44.0	0	0	PC 17610	27.7208	B4	C
195	196	1	Lurette, Miss. Elise	female	58.0	0	0	PC 17569	146.5208	B80	C
215	216	1	Newell, Miss. Madeleine	female	31.0	1	0	35273	113.2750	D36	C
218	219	1	Bazzani, Miss. Albina	female	32.0	0	0	11813	76.2917	D15	C
230	231	1	Harris, Mrs. Henry Birkhardt (Irene Wallach)	female	35.0	1	0	36973	83.4750	C83	S
256	257	1	Thorne, Mrs. Gertrude Maybelle	female	28.0	0	0	PC 17585	79.2000	NaN	C
257	258	1	Cherry, Miss. Gladys	female	30.0	0	0	110152	86.5000	B77	S
258	259	1	Ward, Miss. Anna	female	35.0	0	0	PC 17755	512.3292	NaN	C

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
268	269	1	Graham, Mrs. William Thompson (Edith Junkins)	female	58.0	0	1	PC 17582	153.4625	C125	S

In [25]:
```
c_df[c_df['Pclass']==1 & c_df['Sex'].eq('female')].tail(20)
```

Out[25]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
240	1132	1	Lindstrom, Mrs. Carl Johan (Sigrid Posse)	female	55.0	0	0	112377	27.7208	NaN	C
272	1164	1	Clark, Mrs. Walter Miller (Virginia McDowell)	female	26.0	1	0	13508	136.7792	C89	C
305	1197	1	Crosby, Mrs. Edward Gifford (Catherine Elizabe...	female	64.0	1	1	112901	26.5500	B26	S
314	1206	1	White, Mrs. John Stuart (Ella Holmes)	female	55.0	0	0	PC 17760	135.6333	C32	C
324	1216	1	Kreuchen, Miss. Emilie	female	39.0	0	0	24160	211.3375	NaN	S
343	1235	1	Cardeza, Mrs. James Warburton Martinez (Charlo...	female	58.0	0	1	PC 17755	512.3292	B51 B53 B55	C
350	1242	1	Greenfield, Mrs. Leo David (Blanche Strouse)	female	45.0	0	1	PC 17759	63.3583	D10 D12	C
356	1248	1	Brown, Mrs. John Murray (Caroline Lane Lamson)	female	59.0	2	0	11769	51.4792	C101	S
364	1256	1	Harder, Mrs. George Achilles (Dorothy Annan)	female	25.0	1	0	11765	55.4417	E50	C
368	1260	1	Gibson, Mrs. Leonard (Pauline C Boeson)	female	45.0	0	1	112378	59.4000	NaN	C
371	1263	1	Wilson, Miss. Helen Alice	female	31.0	0	0	16966	134.5000	E39 E41	C
374	1266	1	Dodge, Mrs. Washington (Ruth Vidaver)	female	54.0	1	1	33638	81.8583	A34	S
375	1267	1	Bowen, Miss. Grace Scott	female	45.0	0	0	PC 17608	262.3750	NaN	C
391	1283	1	Lines, Mrs. Ernest H (Elizabeth Lindsey James)	female	51.0	0	1	PC 17592	39.4000	D28	S
395	1287	1	Smith, Mrs. Lucien Philip (Mary Eloise Hughes)	female	18.0	1	0	13695	60.0000	C31	S
397	1289	1	Frolicher-Stehli, Mrs. Maxmillian (Margaretha ...	female	48.0	1	1	13567	79.2000	B41	C
400	1292	1	Bonnell, Miss. Caroline	female	30.0	0	0	36928	164.8667	C7	S
402	1294	1	Gibson, Miss. Dorothy Winifred	female	22.0	0	1	112378	59.4000	NaN	C
411	1303	1	Minahan, Mrs. William Edward (Lillian E Thorpe)	female	37.0	1	0	19928	90.0000	C78	Q

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
414	1306	1	Oliva y Ocana, Dona. Fermina	female	39.0	0	0	PC 17758	108.9000	C105	C

NOTE:

- Most of female passengers with data (*Pclass = 1*) and (*ticket number sequence = '113XXX'*) have **Southampton**(S) under their 'Embarked' field
- Since the percentage of missing values is extremely low, we could utilize the most frequently occuring value for filling up the missing values

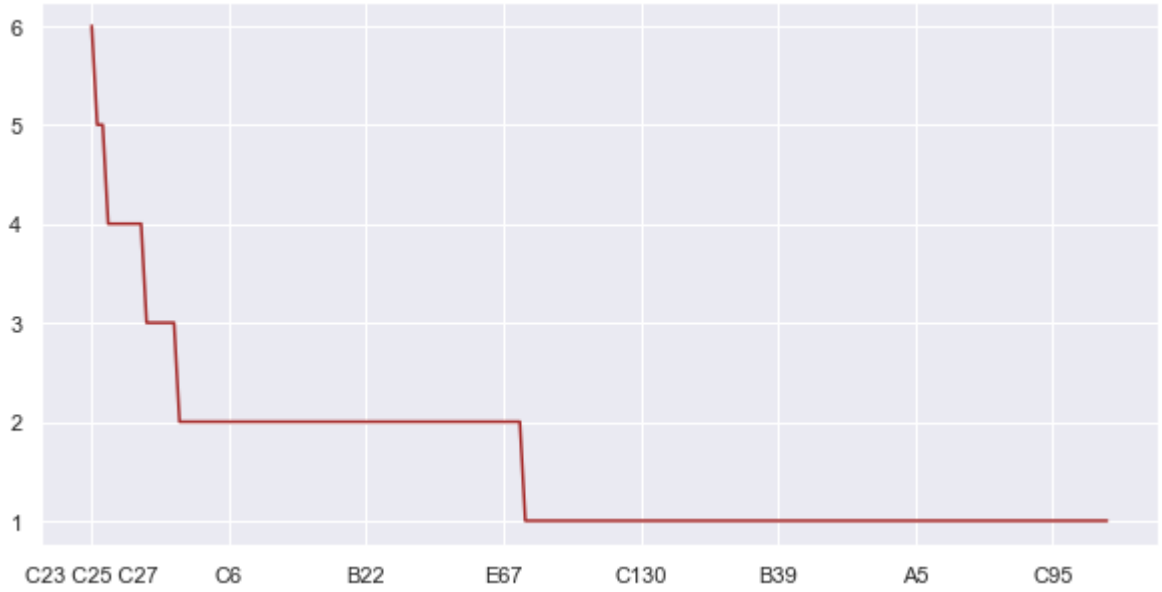
In [26]:
```
c_df['Embarked'] = c_df['Embarked'].fillna(value='S')
```

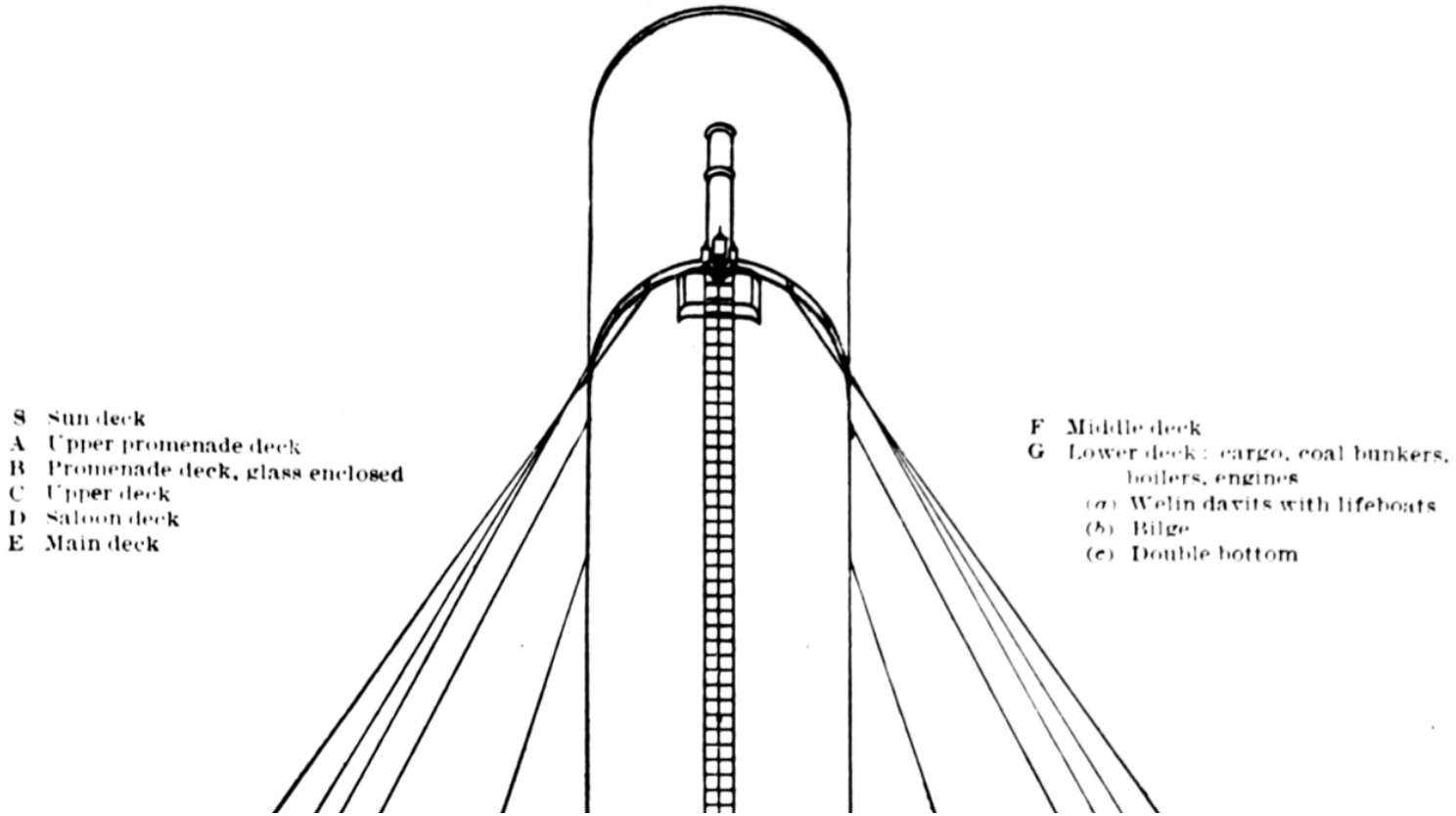
In [27]:
```
c_df.isnull().sum()
```

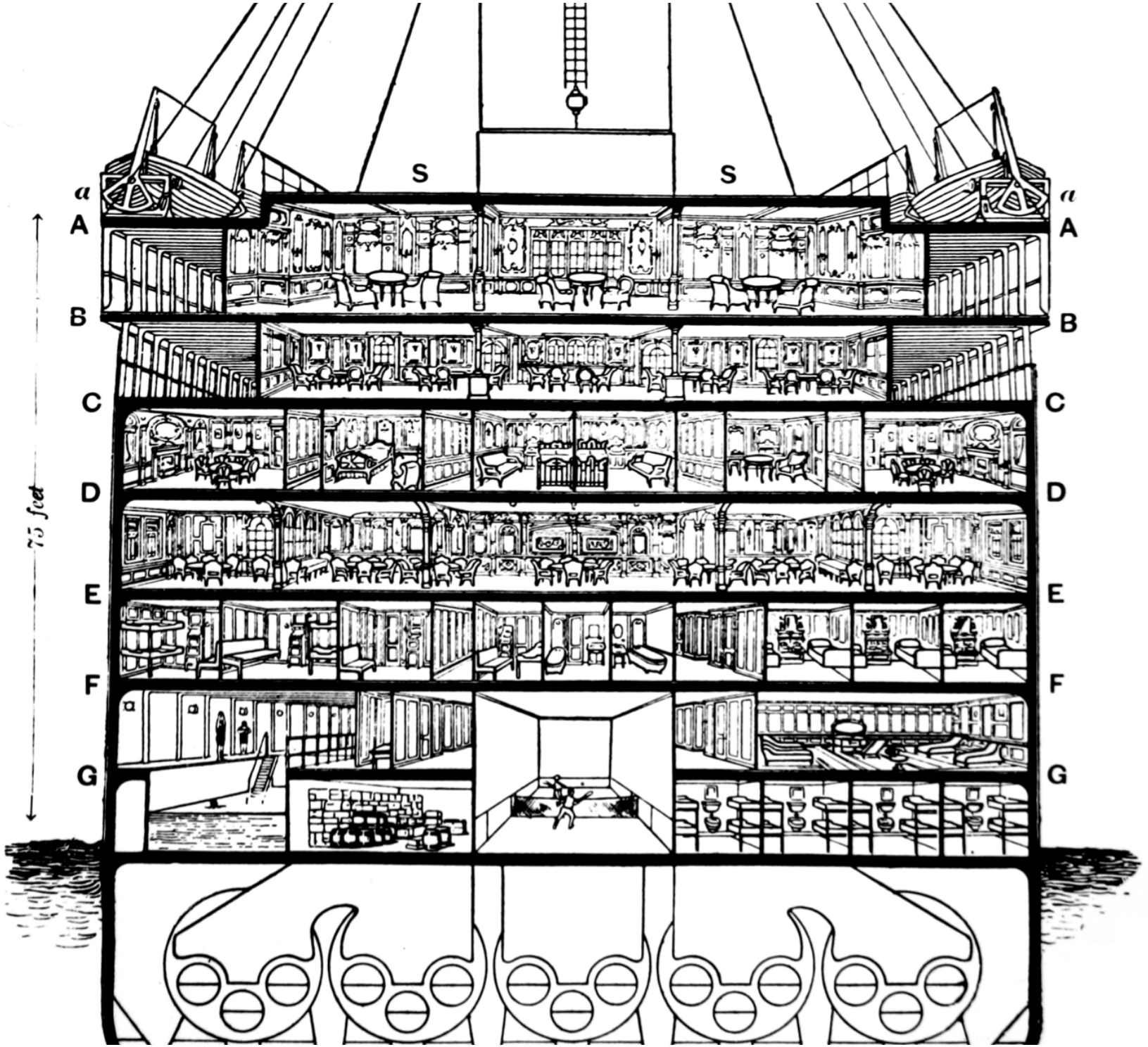
Out[27]:
```
PassengerId       0
Pclass            0
Name              0
Sex               0
Age               0
SibSp             0
Parch             0
Ticket            0
Fare              0
Cabin          1014
Embarked          0
dtype: int64
```

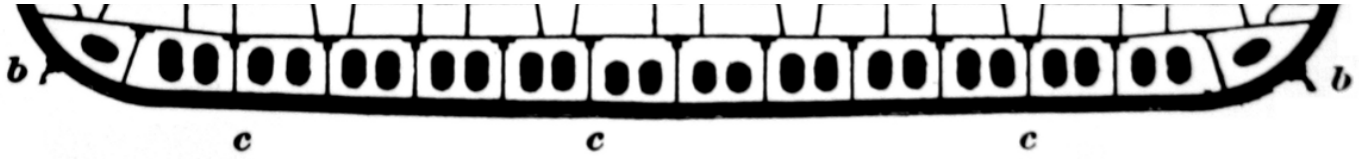
In [28]:
```
# now let's go ahead and analyze the 'Cabin' feature

c_df['Cabin'].value_counts().plot(figsize=(10,5), color='brown')
```

Out[28]:
```
<AxesSubplot:>
```


6
5
4
3
2
1
C23 C25 C27
C6
B22
E67
C130
B39
A5
C95


S Sun deck
A Upper promenade deck
B Promenade deck, glass enclosed
C Upper deck
D Saloon deck
E Main deck
F Middle deck
G Lower deck: cargo, coal bunkers, boilers, engines
(a) Welin davits with lifeboats
(b) Bilge
(c) Double bottom


a
S
S
a
A
A
B
B
C
C
D
D
E
E
F
F
G
G
75 feet



- If we visualize above first class had the cabins on deck A/B/C, a mix of it was on D/E and the third class was mainly on F/G
- We can identify the deck by the first letter based on alphabetical order -> A -- B -- C -- D -- E -- F -- G
- However as the column 'Cabin' has a lot of missing data, let's categorize all the missing data as a different class and name it as NA
- In addition to that, let's assign all the missing values with this value
- If we visualize the figure above, there were various decks and people aboard were allocated cabins in these decks based on their ticket class
- So let's utilize 'Cabin' feature to create a new feature named 'Deck' and have values assigned for each passenger
- Once we have craeted this 'Deck' feature, we will drop the 'Cabin' feature

In [29]:
```
c_df['Cabin'].value_counts()
```

Out[29]:
```
C23 C25 C27        6
G6                 5
B57 B59 B63 B66    5
C22 C26            4
F33                4
                  ..
A14                1
E63                1
E12                1
E38                1
C105               1
Name: Cabin, Length: 186, dtype: int64
```

- For 'Cabin' field having null value, we will use 'M' to fill up the corresponding 'Deck' value
- Here for cabin cells having missing values, we are replacing the corresponding field under 'Deck' feature with letter 'M'
- M here is an acronym for 'missing values'

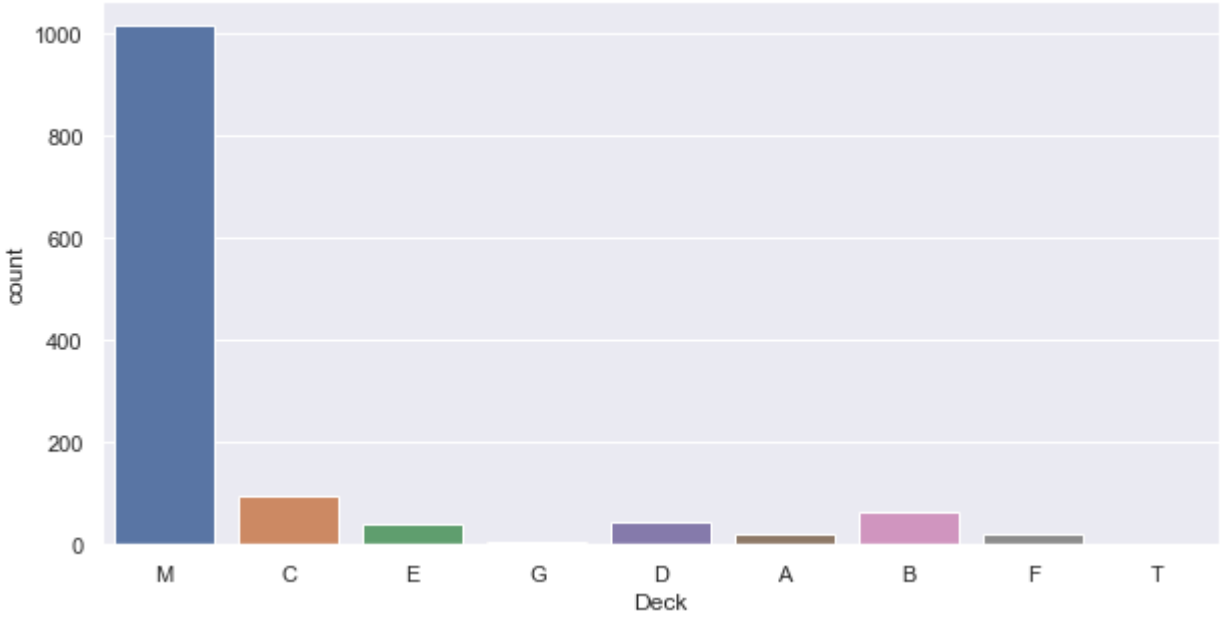
In [30]:
```
c_df['Deck'] = c_df['Cabin'].apply(lambda x:x[0] if pd.notnull(x) else 'M')
```

In [31]:
```
c_df = c_df.reset_index(drop=True)
```

In [32]:
```
plt.figure(figsize=(10,5))
print(c_df['Deck'].value_counts())
sns.countplot(x='Deck', data=c_df, )
```

```
M    1014
C      94
B      65
D      46
E      41
A      22
F      21
G       5
T       1
Name: Deck, dtype: int64
```

Out[32]:
```
<AxesSubplot:xlabel='Deck', ylabel='count'>
```



NOTE: There are significant differences in survival rates because guests on the upper decks were prioritized in terms of being provided lifejackets and aboarding the lifeboats

ASSUMPTIONS:

- Most of first class passengers were aboard the deck (A/B/C)
- A mix of first class & second class passengers were aboard the deck (D/E)

- While the third class passengers were mainly aboard the deck (F/G)
- So, we will group up these decks as **D1** [A/B/C], **D2** [D/E], **D3** [F/G]

In [33]:
```
# for Deck 'T', we only have 1 passenger so we will add it up to our missing value category 'M'
# since we do not have any details regarding the 'T' named Deck in the figure shared above

idx = c_df[c_df['Deck']=='T'].index
c_df.loc[idx, 'Deck'] = 'M'
```

In [34]:
```
c_df['Deck'] = c_df['Deck'].replace(['A', 'B', 'C'], 'D1')
c_df['Deck'] = c_df['Deck'].replace(['D', 'E'], 'D2')
c_df['Deck'] = c_df['Deck'].replace(['F', 'G'], 'D3')
```

In [35]:
```
c_df['Deck'].value_counts()
```

Out[35]:
```
M     1015
D1     181
D2      87
D3      26
Name: Deck, dtype: int64
```

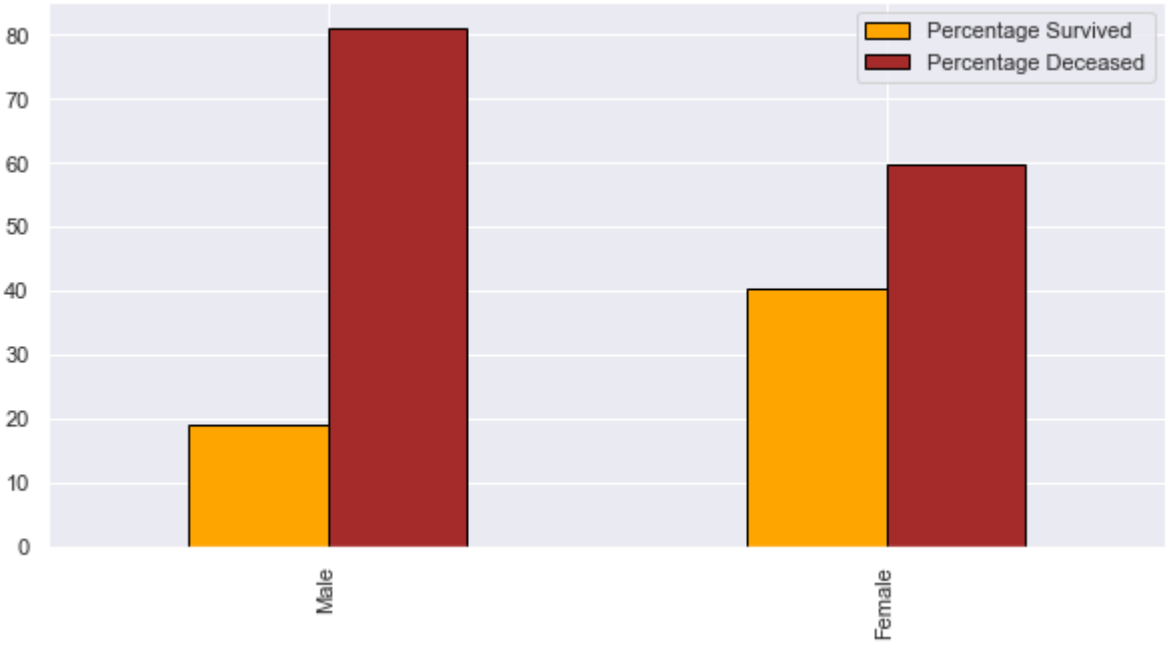
In [36]:
```
# survivor distribution based on age

male_survivor = round((raw_df[raw_df.Sex.eq('male')].Survived == 1).value_counts()[1]/len(raw_df[raw_df.Sex.eq('male')]
female_survivor = round((raw_df[raw_df.Sex.eq('female')].Survived == 1).value_counts()[1]/len(raw_df[raw_df.Sex.eq('mal


age_perc_df = pd.DataFrame(
    { "Percentage Survived":{"Male": male_survivor,"Female": female_survivor},
     "Percentage Deceased":{"Male": 100-male_survivor,"Female": 100-female_survivor}})
age_perc_df
```

Out[36]:

	Percentage Survived	Percentage Deceased
Male	18.89	81.11
Female	40.38	59.62

In [37]:
```
age_perc_df.plot(kind='bar', figsize=(10,5), color=['orange', 'brown'], edgecolor='black')
```

Out[37]:
```
<AxesSubplot:>
```



NOTE:

> If we visualize the graph above, it provides us a valuable insight to help reason out that the percentage of survivors is greater in case of female passengers while the percentage of deceased is maximum in case of male passengers. This makes us believe that the female passengers were prioritzed over the male passengers in terms of being provided lifejackets and aboarding the lifeboats.

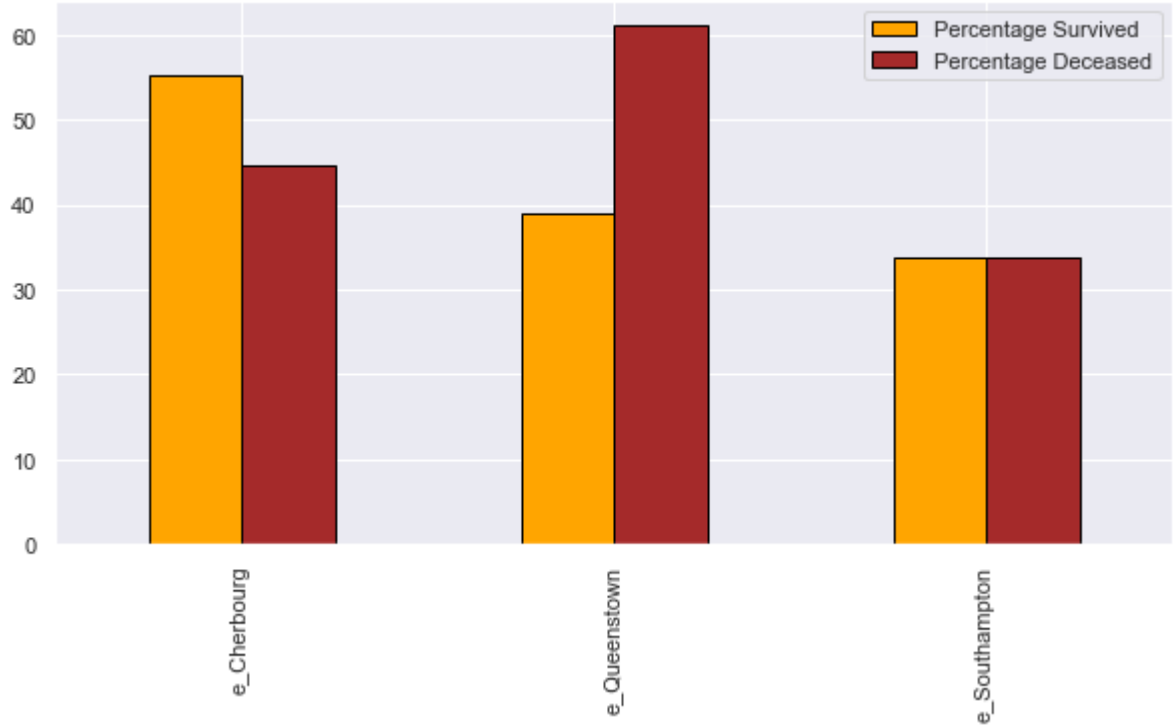
In [38]:
```
# survivor distribution based on port of embarkation feature

cherbourg_survivor = round((raw_df[raw_df.Embarked.eq('C')].Survived == 1).value_counts()[1]/len(raw_df[raw_df.Embarked
queenstown_survivor = round((raw_df[raw_df.Embarked.eq('Q')].Survived == 1).value_counts()[1]/len(raw_df[raw_df.Embarke
southampton_survivor = round((raw_df[raw_df.Embarked.eq('S')].Survived == 1).value_counts()[1]/len(raw_df[raw_df.Embark

embarked_perc_df = pd.DataFrame(
    { "Percentage Survived":{"e_Cherbourg": cherbourg_survivor,"e_Queenstown": queenstown_survivor, "e_Southampton": so
     "Percentage Deceased":{"e_Cherbourg": 100-cherbourg_survivor,"e_Queenstown": 100-queenstown_survivor, "e_Southampt
embarked_perc_df
```

Out[38]:

	Percentage Survived	Percentage Deceased
e_Cherbourg	55.36	44.64
e_Queenstown	38.96	61.04
e_Southampton	33.70	33.70

In [39]:
```
embarked_perc_df.plot(kind='bar', figsize=(10,5), color=['orange', 'brown'], edgecolor='black')
```

Out[39]:
```
<AxesSubplot:>
```



FEATURE ENGINEERING & HANDLING OUTLIERS

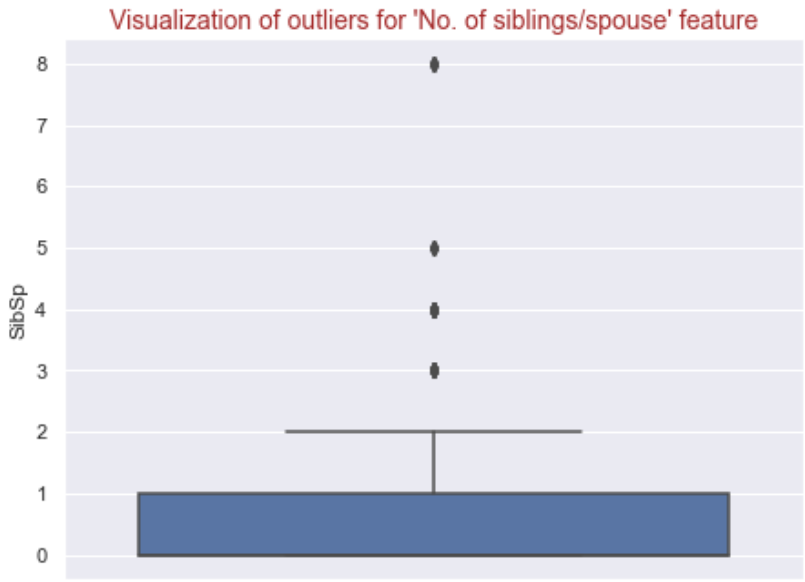
In [40]:
```
fig, sub = plt.subplots(nrows=2, ncols=2, figsize=(16,12))
fig.suptitle('BOX PLOT VISUALIZATION FOR ANALYZING THE OUTLIERS', c='green', size=22)

sns.boxplot(y=c_df['Age'], orient='v', ax=sub[0][0])
sub[0][0].set_title('Visualization of outliers for \'age\' feature', c='brown', size=14)
```

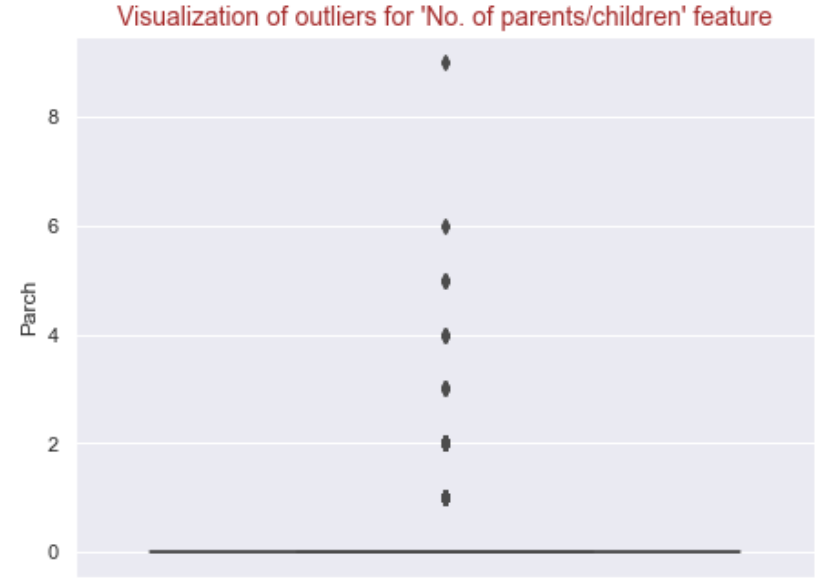
```
sns.boxplot(y=c_df['SibSp'], orient='v', ax=sub[0][1])
sub[0][1].set_title('Visualization of outliers for \'No. of siblings/spouse\' feature', c='brown', size=14)

sns.boxplot(y=c_df['Parch'], orient='v', ax=sub[1][0])
sub[1][0].set_title('Visualization of outliers for \'No. of parents/children\' feature', c='brown', size=14)

sns.boxplot(y=c_df['Fare'], orient='v', ax=sub[1][1])
sub[1][1].set_title('Visualization of outliers for \'Fare\' feature', c='brown', size=14)
```

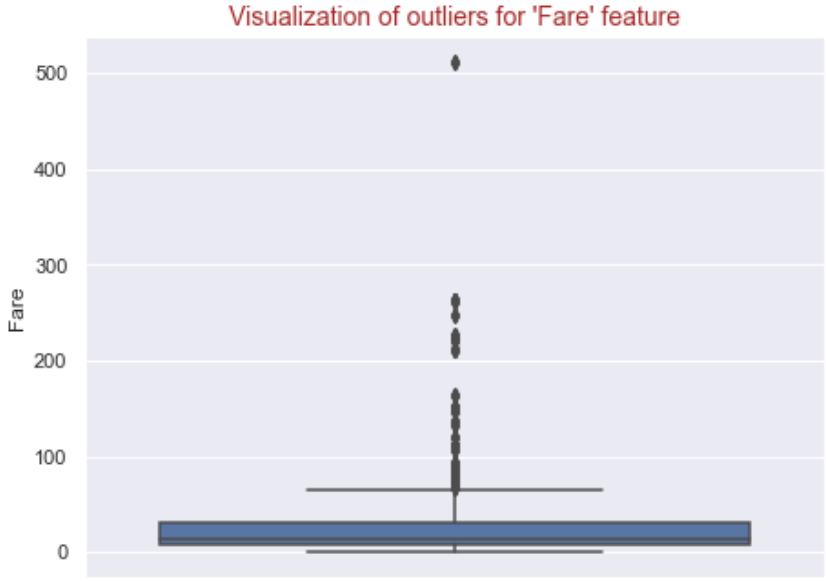
Out[40]:

```
Text(0.5, 1.0, "Visualization of outliers for 'Fare' feature")
```

BOX PLOT VISUALIZATION FOR ANALYZING THE OUTLIERS







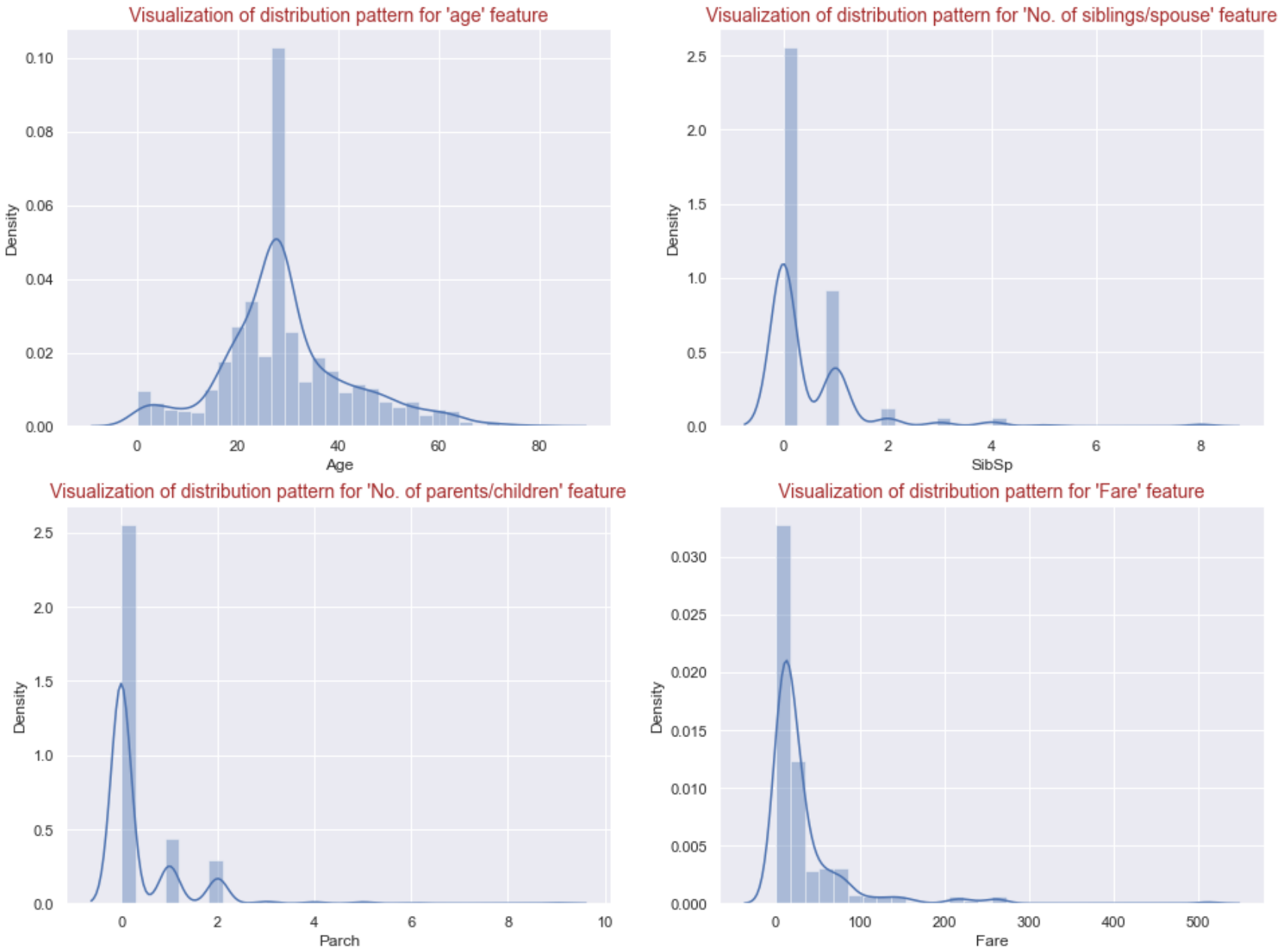
In [41]:
```
# There are 4 features having outliers: Age, Sibsp, Parch, Fare
# Let's plot them to know about the distribution pattern being followed by these features
```

```
# if we visualize 'Parch', 'SibSp' feature, the values can't be stated as outliers as there are only few instances of s
```

In [42]:
```
fig, sub = plt.subplots(nrows=2, ncols=2, figsize=(16,12))
fig.suptitle('KDE VISUALIZATION FOR ANALYZING THE DISTRIBUTION PATTERN', c='green', size=22)

sns.distplot(c_df['Age'], bins=30, hist=True, kde=True, ax=sub[0][0])
sub[0][0].set_title('Visualization of distribution pattern for \'age\' feature', c='brown', size=14)

sns.distplot(c_df['SibSp'], bins=30, hist=True, kde=True, ax=sub[0][1])
sub[0][1].set_title('Visualization of distribution pattern for \'No. of siblings/spouse\' feature', c='brown', size=14)

sns.distplot(c_df['Parch'], bins=30, hist=True, kde=True, ax=sub[1][0])
sub[1][0].set_title('Visualization of distribution pattern for \'No. of parents/children\' feature', c='brown', size=14

sns.distplot(c_df['Fare'], bins=30, hist=True, kde=True, ax=sub[1][1])
sub[1][1].set_title('Visualization of distribution pattern for \'Fare\' feature', c='brown', size=14)
```

Out[42]:
```
Text(0.5, 1.0, "Visualization of distribution pattern for 'Fare' feature")
```

KDE VISUALIZATION FOR ANALYZING THE DISTRIBUTION PATTERN



RIGHT SKEW DISTRIBUTION: These distributions tend to occur when there is a lower limit, and most values are relatively close to the

lower bound. Values can't be less than this bound but can fall far from the peak on the high end, causing them to skew positively.

NOTE: If the data is either right-skewed or left-skewed, we will use the Interquartile Range to measure the limits of outliers. On visualizing the plots above, all the features (except '*Age*') are following right skew distribution. So we will be imputing the values for these based on below formulas:

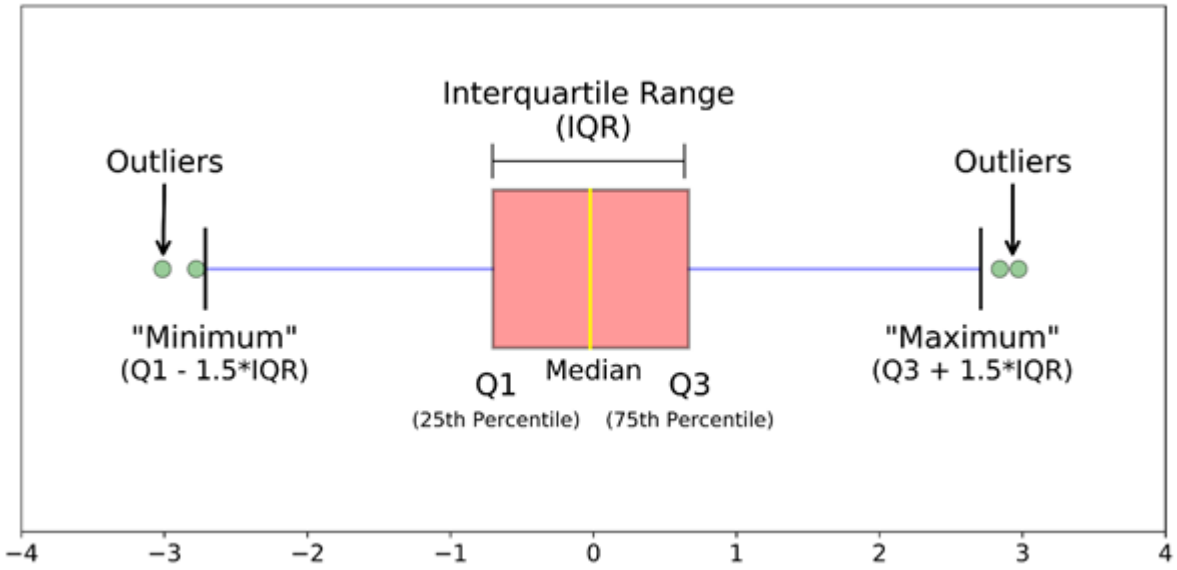
> Interquartile Range(IQR) = Q3(75th percentile) - Q1(25th percentile)

The formula for the outlier boundary can be calculated as:

> - Lower Boundary = First Quartile(Q1/25th percentile) — (1.5 * IQR)
> - Upper Boundary = Third Quartile(Q3/75th percentile) + (1.5* IQR)

If the outlier's maximum value is extremely high in comparison to the upper boundary, the boundary of outliers (also known as extreme outliers) will be calculated using the formula below:

> - Lower Boundary = First Quartile(Q1/25th percentile) — (3 * IQR)
> - Upper Boundary = Third Quartile(Q3/75th percentile) + (3 * IQR)



In [43]:
```
c_df.describe()
```

Out[43]:

	PassengerId	Pclass	Age	SibSp	Parch	Fare
count	1309.000000	1309.000000	1309.000000	1309.000000	1309.000000	1309.000000
mean	655.000000	2.294882	29.503186	0.498854	0.385027	33.281086
std	378.020061	0.837836	12.905241	1.041658	0.865560	51.741500
min	1.000000	1.000000	0.170000	0.000000	0.000000	0.000000
25%	328.000000	2.000000	22.000000	0.000000	0.000000	7.895800
50%	655.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	982.000000	3.000000	35.000000	1.000000	0.000000	31.275000
max	1309.000000	3.000000	80.000000	8.000000	9.000000	512.329200

In [44]:
```
c_df.shape
```

Out[44]:
```
(1309, 12)
```

DUMMY VARIABLE & ONE HOT ENCODING

In [45]:
```
# transforming the previous datafrmae to a new dataframe named pp_df
# pp_df here is an acronym for preprocessed dataframe
# we take a copy of our previous dataframe as we'll be creating dummies for the categorical features

pp_df = c_df.copy()
```

In [46]:
```
pp_df.head()
```

Out[46]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Deck
0	1	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S	M
1	2	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C	D1
2	3	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S	M
3	4	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S	D1
4	5	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S	M

1: Encoding the 'Sex' feature

We would be dropping any one of the dummy variable columns for 'Sex' feature to avoid multicollinearity

In [47]:
```
sex_df = pd.get_dummies(pp_df['Sex'])
pp_df = pd.concat([pp_df, sex_df], axis=1)
pp_df.drop(['Sex', 'female'], axis=1, inplace=True)
```

In [48]:
```
# renaming the 'male' feature as 'Sex_male' since the values are encoded in binary

pp_df.columns = ['PassengerId', 'Pclass', 'Name', 'Age', 'SibSp', 'Parch',
       'Ticket', 'Fare', 'Cabin', 'Embarked', 'Deck', 'Sex_male']
```

2: Encoding the 'Pclass' feature

We would be dropping any one of the dummy variable columns for 'Pclass' feature to avoid multicollinearity

In [49]:
```
pclass_df = pd.get_dummies(pp_df['Pclass'])
pp_df = pd.concat([pp_df, pclass_df], axis=1)
pp_df.columns = ['PassengerId', 'Pclass', 'Name', 'Age', 'SibSp', 'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked', 'Deck
                 'Sex_male', 'Class_1',  'Class_2', 'Class_3']
```

In [50]:
```
pp_df.drop(['Pclass', 'Class_1'], axis=1, inplace=True)
```

In [51]:
```
pp_df.head()
```

Out[51]:

	PassengerId	Name	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Deck	Sex_male	Class_2	Class_3
0	1	Braund, Mr. Owen Harris	22.0	1	0	A/5 21171	7.2500	NaN	S	M	1	0	1
1	2	Cumings, Mrs. John Bradley (Florence Briggs Th...	38.0	1	0	PC 17599	71.2833	C85	C	D1	0	0	0
2	3	Heikkinen, Miss. Laina	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S	M	0	0	1
3	4	Futrelle, Mrs. Jacques Heath (Lily May Peel)	35.0	1	0	113803	53.1000	C123	S	D1	0	0	0
4	5	Allen, Mr. William Henry	35.0	0	0	373450	8.0500	NaN	S	M	1	0	1

3: Encoding the 'age' feature

We would be grouping the age values into four categories and further drop any one of the dummy variable columns for 'age' feature

In [52]:
```
print('Maximum age value under the dataset:', pp_df['Age'].max())
print('Minimum age value under the dataset:', pp_df['Age'].min())
```

```
Maximum age value under the dataset: 80.0
Minimum age value under the dataset: 0.17
```

so let's group the age values into four categories as:

- **Young** (0-15 yrs.)
- **Adult** (16-30 yrs.)
- **Middle Aged** (31-45 yrs.)
- **Old Aged** (46+ yrs.)

In [53]:
```
age_df = pd.get_dummies(pp_df['Age'])
```

In [54]:
```
# using loc() method to extract the columns based on the column index value and further storing it in a newer dataframe
```

```
df1 = age_df.loc[:, :15]
df2 = age_df.loc[:, 16:30]
df3 = age_df.loc[:, 31:45]
df4 = age_df.loc[:, 46:]
```

In [55]:
```
young_df = df1.sum(axis=1) # dataframe from 'young' age group
adult_df = df2.sum(axis=1) # dataframe from 'adult' age group
middle_aged_df = df3.sum(axis=1) # dataframe from 'middle aged' group
old_aged_df = df4.sum(axis=1) # dataframe from 'old aged' group
```

In [56]:
```
final_age_df = pd.concat([young_df, adult_df, middle_aged_df, old_aged_df], axis=1)
```

In [57]:
```
pp_df = pd.concat([pp_df, final_age_df], axis=1)
```

In [58]:
```
pp_df.columns = ['PassengerId', 'Name', 'Age', 'SibSp', 'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked', 'Deck',
                 'Sex_male',  'Class_2', 'Class_3', 'Young',  'Adult',  'Middle_aged', 'Old_aged']
```

In [59]:
```
pp_df.drop(['Age', 'Young'], axis=1, inplace=True)
pp_df
```

Out[59]:

	PassengerId	Name	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Deck	Sex_male	Class_2	Class_3	Adult	Middle_aged
0	1	Braund, Mr. Owen Harris	1	0	A/5 21171	7.2500	NaN	S	M	1	0	1	1	0
1	2	Cumings, Mrs. John Bradley (Florence Briggs Th...	1	0	PC 17599	71.2833	C85	C	D1	0	0	0	0	1
2	3	Heikkinen, Miss. Laina	0	0	STON/O2. 3101282	7.9250	NaN	S	M	0	0	1	1	0
3	4	Futrelle, Mrs. Jacques Heath (Lily May Peel)	1	0	113803	53.1000	C123	S	D1	0	0	0	0	1
4	5	Allen, Mr. William Henry	0	0	373450	8.0500	NaN	S	M	1	0	1	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1304	1305	Spector, Mr. Woolf	0	0	A.5. 3236	8.0500	NaN	S	M	1	0	1	1	0
1305	1306	Oliva y Ocana, Dona. Fermina	0	0	PC 17758	108.9000	C105	C	D1	0	0	0	0	1
1306	1307	Saether, Mr. Simon Sivertsen	0	0	SOTON/O.Q. 3101262	7.2500	NaN	S	M	1	0	1	0	1
1307	1308	Ware, Mr. Frederick	0	0	359309	8.0500	NaN	S	M	1	0	1	1	0
1308	1309	Peter, Master. Michael J	1	1	2668	22.3583	NaN	C	M	1	0	1	1	0

1309 rows × 15 columns

4: Encoding the 'Embarked' feature

We would be dropping any one of the dummy variable columns for 'Embarked' feature to avoid multicollinearity

In [60]:
```
embarked_df = pd.get_dummies(pp_df['Embarked'])
```

In [61]:
```
pp_df = pd.concat([pp_df, embarked_df], axis=1)
```

In [62]:
```
pp_df.columns = ['PassengerId', 'Name', 'SibSp', 'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked', 'Deck',
                 'Sex_male',  'Class_2', 'Class_3', 'Adult',  'Middle_aged', 'Old_aged', 'Embarked_C', 'Embarked_Q', 'E
```

In [63]:
```
pp_df = pp_df.drop(['Embarked', 'Embarked_C'], axis=1)
```

In [64]:
```
pp_df.head()
```

Out[64]:

	PassengerId	Name	SibSp	Parch	Ticket	Fare	Cabin	Deck	Sex_male	Class_2	Class_3	Adult	Middle_aged	Old_aged	Emba
0	1	Braund, Mr. Owen Harris	1	0	A/5 21171	7.2500	NaN	M	1	0	1	1	0	0	
1	2	Cumings, Mrs. John Bradley (Florence Briggs Th...	1	0	PC 17599	71.2833	C85	D1	0	0	0	0	1	0	
2	3	Heikkinen, Miss. Laina	0	0	STON/O2. 3101282	7.9250	NaN	M	0	0	1	1	0	0	
3	4	Futrelle, Mrs. Jacques Heath (Lily May Peel)	1	0	113803	53.1000	C123	D1	0	0	0	0	1	0	
4	5	Allen, Mr. William Henry	0	0	373450	8.0500	NaN	M	1	0	1	0	1	0	

5: Encoding the 'Deck' feature

We would be dropping any missing values column for 'Deck' feature to avoid multicollinearity

In [65]:
```
pp_df['Deck'].value_counts()
```

Out[65]:
```
M     1015
D1     181
D2      87
D3      26
Name: Deck, dtype: int64
```

In [66]:
```
# we will be classifying the deck values into 4 groups:
```

```
# -> Upper deck (D1)
# -> Middle deck (D2)
# -> Lower deck (D3)
# -> Missing values (M)
```

In [67]:
```
deck_df = pd.get_dummies(pp_df['Deck'])
```

In [68]:
```
pp_df = pd.concat([pp_df, deck_df], axis=1)
```

In [69]:
```
pp_df = pp_df.drop(['Deck', 'M', 'Cabin'], axis=1)
# dropping the 'Cabin' feature as well, as we have already utilized it to introduced new feature named 'Deck'
```

6: Encoding the 'Parch' feature

We would be dropping any one of the dummy variable columns for 'Parch' feature to avoid multicollinearity

In [70]:
```
# let's proceed with the final pre-processing steps to setup our final dataframe ready to be trained
pp_df['Parch'].value_counts()
```

Out[70]:
```
0    1002
1     170
2     113
3       8
5       6
4       6
6       2
9       2
Name: Parch, dtype: int64
```

Since most of the passengers did not have any parent/children under their records as evident from above data, let's classify the 'Parch' feature into two categories:

- 0 - for passengers who did not have any parent/children registered under their records
- 1 - for passengers who had 1 or more parent/children registered under their records

In addition to above we would be dropping the 'Name', 'PassengerID' and 'Ticket' features and would be renaming few of the column labels for better classification.

In [71]:
```
parch_df = pd.get_dummies(pp_df['Parch'])
```

In [72]:
```
no_parch_df = parch_df.loc[:, :0]
yes_parch_df = pd.DataFrame(parch_df.loc[:, 1:].sum(axis=1))
```

In [73]:
```
yes_parch_df.columns = ['Has_Parch']
```

In [74]:
```
pp_df = pd.concat([pp_df, no_parch_df, yes_parch_df], axis=1)
```

In [75]:
```
pp_df = pp_df.drop(['Parch', 0], axis=1)
```

In [76]:
```
pp_df.columns = ['PassengerId', 'Name', 'SibSp', 'Ticket', 'Fare', 'Sex_male',  'Class_2', 'Class_3', 'Adult',  'Middle
```

DROPPING '*Name*'/'*Ticket*' FEATURES AND PROVIDING CONVENIENT LABELS TO THE COLUMNS**

In [77]:
```
pp_df.drop(['Name', 'Ticket'], axis=1, inplace=True)
```

In [78]:
```
pp_df.columns = ['PassengerId', 'has_SibSp', 'Fare', 'Sex_male',  'Class_2', 'Class_3', 'Adult',  'Middle_aged', 'Old_a
```

In [79]:
```
pp_df.head()
```

Out[79]:

	PassengerId	has_SibSp	Fare	Sex_male	Class_2	Class_3	Adult	Middle_aged	Old_aged	Embarked_Q	Embarked_S	D1	D2	D3	has
0	1	1	7.2500	1	0	1	1	0	0	0	1	0	0	0	
1	2	1	71.2833	0	0	0	0	1	0	0	0	1	0	0	
2	3	0	7.9250	0	0	1	1	0	0	0	1	0	0	0	
3	4	1	53.1000	0	0	0	0	1	0	0	1	1	0	0	
4	5	0	8.0500	1	0	1	0	1	0	0	1	0	0	0	

In [80]:
```
pp_df['Fare'] = (pp_df['Fare'] - pp_df['Fare'].mean()) / pp_df['Fare'].std()
pp_df
```

Out[80]:

	PassengerId	has_SibSp	Fare	Sex_male	Class_2	Class_3	Adult	Middle_aged	Old_aged	Embarked_Q	Embarked_S	D1	D2	D3
0	1	1	-0.503099	1	0	1	1	0	0	0	1	0	0	0
1	2	1	0.734463	0	0	0	0	1	0	0	0	1	0	0
2	3	0	-0.490053	0	0	1	1	0	0	0	1	0	0	0
3	4	1	0.383037	0	0	0	0	1	0	0	1	1	0	0
4	5	0	-0.487637	1	0	1	0	1	0	0	1	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1304	1305	0	-0.487637	1	0	1	1	0	0	0	1	0	0	0
1305	1306	0	1.461475	0	0	0	0	1	0	0	0	1	0	0
1306	1307	0	-0.503099	1	0	1	0	1	0	0	1	0	0	0
1307	1308	0	-0.487637	1	0	1	1	0	0	0	1	0	0	0
1308	1309	1	-0.211103	1	0	1	1	0	0	0	0	0	0	0

1309 rows × 15 columns

In [81]:
```
final_pp_df = pp_df.drop(['PassengerId'], axis=1)
```

TRAIN TEST SPLIT

In [82]:
```
from sklearn.model_selection import train_test_split
```

In [83]:
```
known_train_set = final_pp_df.loc[:890 , :]
unknown_test_set = final_pp_df.loc[891: , :]
```

In [85]:
```
print(known_train_set.shape)
print(unknown_test_set.shape)
```

```
(891, 14)
(418, 14)
```

In [88]:
```
X = known_train_set
```

```
y = raw_df['Survived']
```

In [89]:
```
rfc = RandomForestClassifier(criterion='entropy',
                             n_estimators=400,
                             oob_score=True)
```

In [90]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

In [91]:
```
rfc.fit(X_train, y_train)
```

Out[91]:
```
RandomForestClassifier(criterion='entropy', n_estimators=400, oob_score=True)
```

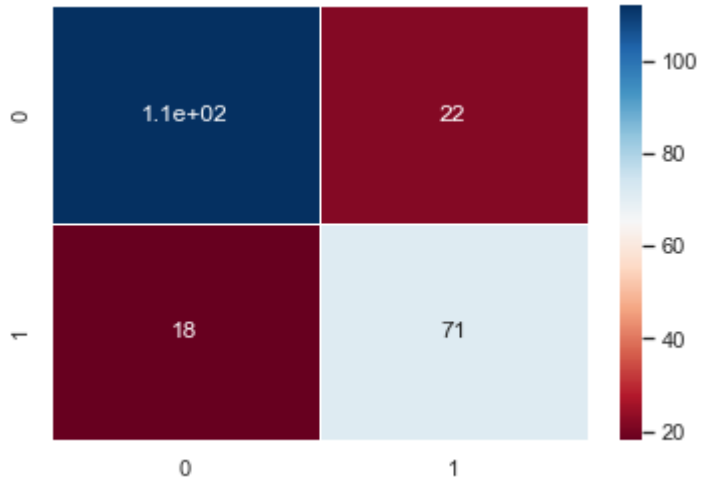
In [92]:
```
y_pred = rfc.predict(X_test)
```

In [93]:
```
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[112  22]
 [ 18  71]]
              precision    recall  f1-score   support

           0       0.86      0.84      0.85       134
           1       0.76      0.80      0.78        89

    accuracy                           0.82       223
   macro avg       0.81      0.82      0.81       223
weighted avg       0.82      0.82      0.82       223
```

In [94]:
```
sns.heatmap(confusion_matrix(y_test,y_pred), cmap='RdBu', annot=True, linewidths=1, linecolor='white')
```

Out[94]:
```
<AxesSubplot:>
```

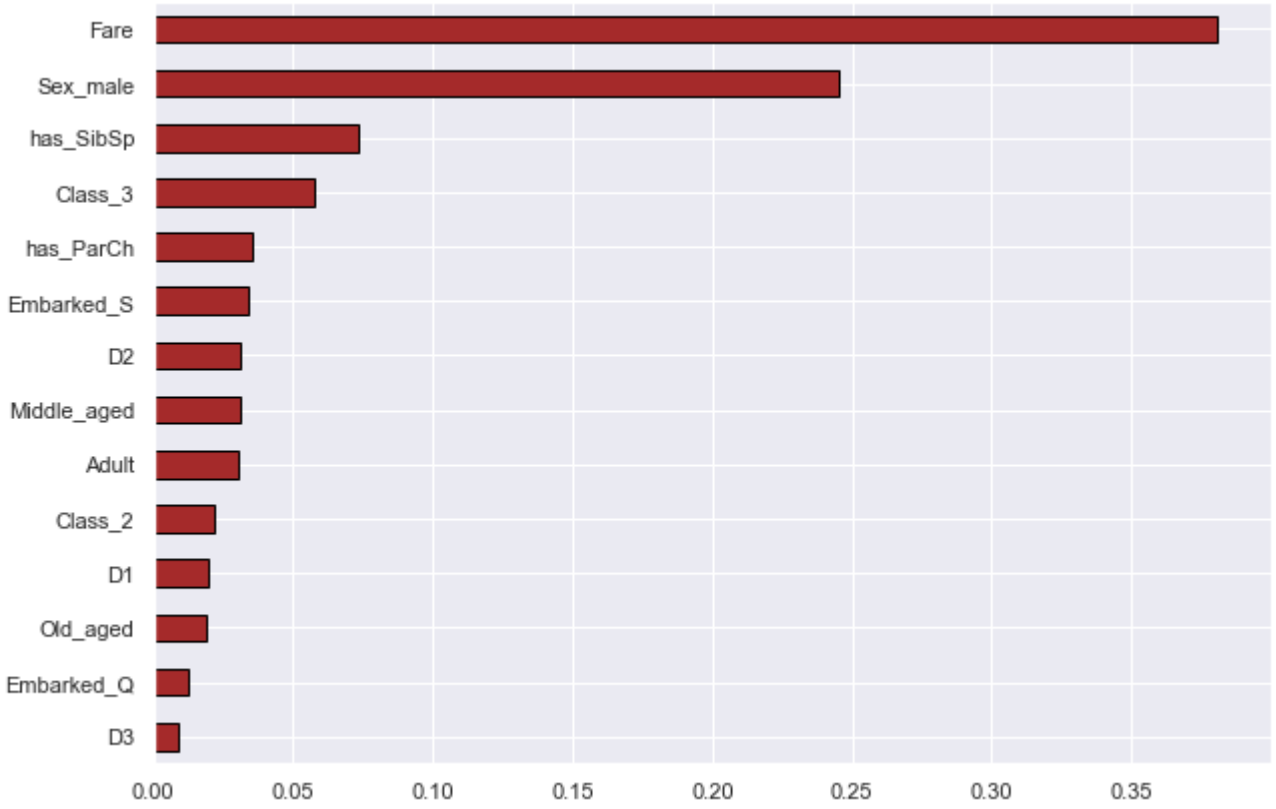


In [95]:
```
print('Accuracy Score:', round(accuracy_score(y_test, y_pred)*100, 2))
```

```
Accuracy Score: 82.06
```

In [96]:
```
feat_imp = pd.Series(rfc.feature_importances_, index=X.columns)
feat_imp.sort_values(ascending=True, inplace=True)
feat_imp.plot(kind='barh', figsize=(10,7), color='brown', edgecolor='black')
```

Out[96]:
```
<AxesSubplot:>
```



In [97]:
```
feat_imp2 = pd.DataFrame(rfc.feature_importances_, index=X.columns, columns=['Importance'])
```

In [98]:
```
feat_imp2.sort_values(by='Importance', ascending=False)
```

Out[98]:

	Importance
Fare	0.380715
Sex_male	0.245361
has_SibSp	0.073268
Class_3	0.057330
has_ParCh	0.034989
Embarked_S	0.034179
D2	0.031174
Middle_aged	0.030768
Adult	0.030546
Class_2	0.021799
D1	0.019266
Old_aged	0.019115
Embarked_Q	0.012367
D3	0.009122

APPLYING OTHER MACHINE LEARNING METHODS

In [99]:
```
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
from sklearn.ensemble import AdaBoostClassifier
```

INITIALIZE THE MODEL:

In [100…
```
svc = SVC(kernel='linear', random_state = 1)
knn = KNeighborsClassifier()
xgb = XGBClassifier()
adb = AdaBoostClassifier(n_estimators=100, random_state=1)
```

FIT AND TEST THE MODELS:

In [101…]
```
svc.fit(X_train,y_train)
knn.fit(X_train,y_train)
xgb.fit(X_train,y_train)
adb.fit(X_train,y_train)
```

Out[101]:
```
AdaBoostClassifier(n_estimators=100, random_state=1)
```

PREDICTION OVER THE TEST DATA:

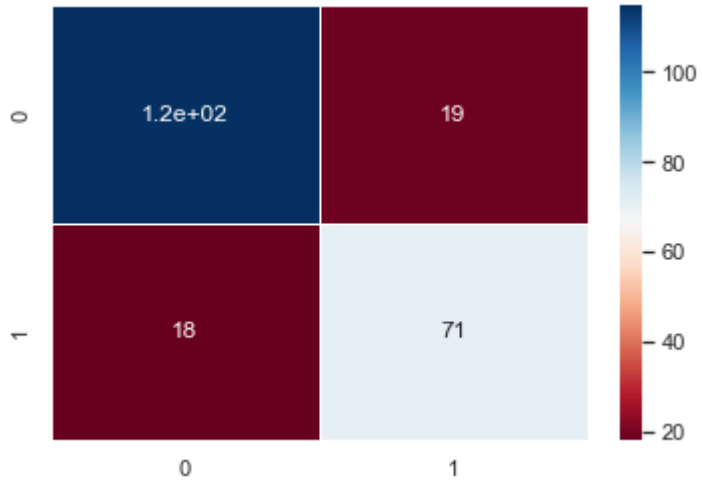
In [102…]
```
y_pred_svc= svc.predict(X_test)
y_pred_knn= knn.predict(X_test)
y_pred_xgb= xgb.predict(X_test)
y_pred_adb= adb.predict(X_test)
```

In [103…]
```
print('Accuracy Score using SupportVectorClassifier model:', round(accuracy_score(y_test, y_pred_svc)*100, 2))
print('Accuracy Score using KNearestNeighbors model:', round(accuracy_score(y_test, y_pred_knn)*100, 2))
print('Accuracy Score using XGBClassifier model:', round(accuracy_score(y_test, y_pred_xgb)*100, 2))
print('Accuracy Score using AdaBoostClassifier model:', round(accuracy_score(y_test, y_pred_adb)*100, 2))
```

```
Accuracy Score using SupportVectorClassifier model: 78.48
Accuracy Score using KNearestNeighbors model: 81.17
Accuracy Score using XGBClassifier model: 83.41
Accuracy Score using AdaBoostClassifier model: 80.72
```

In [105…]
```
# analyzing the heatmap for predicted values using XGBClassifier model
sns.heatmap(confusion_matrix(y_test,y_pred_xgb), cmap='RdBu', annot=True, linewidths=1, linecolor='white')
```

Out[105]:
```
<AxesSubplot:>
```



In [106…
```
predictions = xgb.predict(unknown_test_set)
```

In [109…
```
predictions_df = pd.DataFrame(predictions, columns=['Survived'])
```

In [116…
```
predictions_df.Survived.astype(int)
```

Out[116]:
```
0      0
1      0
2      0
3      0
4      1
      ..
413    0
414    1
415    0
416    0
417    1
Name: Survived, Length: 418, dtype: int32
```

In [117…
```
p_id = test_df['PassengerId']
p_id_df = pd.DataFrame(p_id, columns=['PassengerId'])
p_id_df
```

Out[117]:

	PassengerId
0	892
1	893
2	894
3	895
4	896
...	...
413	1305
414	1306
415	1307
416	1308
417	1309

418 rows × 1 columns

In [119…

```
pred_output_df = pd.concat([p_id_df, predictions_df], axis=1)
pred_output_df
```

Out[119]:

	PassengerId	Survived
0	892	0
1	893	0
2	894	0
3	895	0
4	896	1
...	...	...
413	1305	0
414	1306	1
415	1307	0
416	1308	0
417	1309	1

418 rows × 2 columns

In [ ]:
```
pred_output_df.to_csv('submission.csv', index=False)
```

CONGRATULATIONS! Your predictions have been submitted successfully. Wish you all the best for the results!