Capstone Project:

Titanic Survivors Prediction

Problem Statement

The sinking of the RMS Titanic is one of the most infamous shipwrecks in history. On April 15, 1912, during her maiden voyage, the Titanic sank after colliding with an iceberg, killing numerous passengers and crew. This sensational tragedy shocked the international community and led to better safety regulations for ships. One of the reasons that the shipwreck led to such loss of life was that there were not enough lifeboats for the passengers and crew. Although there was some element of luck involved in surviving the sinking, some groups of people were more likely to survive than others, such as women, children, and the upper-class. In this, we ask you to complete the analysis of what sorts of people were likely to survive. In particular, we ask you to apply the tools of machine learning to predict which passengers survived the tragedy.

Data Dictionary Variables:

1. **survived:** Survival (0 = No, 1 = Yes)
2. **Pclass:** Ticket class (1 = 1st, 2 = 2nd, 3 = 3rd)
3. **Sex:** Gender (Male or Female)
4. **Age:** Age in years
5. **Sibsp:** # of siblings / spouses aboard the Titanic
6. **Parch:** # of parents / children aboard the Titanic
7. **Ticket:** Ticket number
8. **Fare:** Passenger fare
9. **Cabin:** Cabin number
10. **embarked:** Port of Embarkation (C= Cherbourg, Q = Queenstown, S =Southampton)

Importing Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import  mean_squared_error
from sklearn.preprocessing import StandardScaler
import scipy
import matplotlib.pyplot
import statistics
from   pylab import rcParams
import warnings
from matplotlib.pyplot import figure
warnings.filterwarnings("ignore")
```

In [2]:
```
df=pd.read_csv('train.csv')
df.sample(5)
```

Out[2]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
593	594	0	3	Bourke, Miss. Mary	female	NaN	0	2	364848	7.7500	NaN	Q
136	137	1	1	Newsom, Miss. Helen Monypeny	female	19.0	0	2	11752	26.2833	D47	S
29	30	0	3	Todoroff, Mr. Lalio	male	NaN	0	0	349216	7.8958	NaN	S
868	869	0	3	van Melkebeke, Mr. Philemon	male	NaN	0	0	345777	9.5000	NaN	S
196	197	0	3	Mernagh, Mr. Robert	male	NaN	0	0	368703	7.7500	NaN	Q

In [3]:
```
print('The number of rows (observations) is',df.shape[0],'\n''The number of columns (variables) is',df.shape[1],'
```

```
The number of rows (observations) is 891
The number of columns (variables) is 12
Total values in dataset: 10692
```

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

```
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]:

```
pd.DataFrame(data={'# Missing Values':df.isna().sum(),
                '% of Missing Values':round(df.isnull().sum()/df.isnull().count()*100,2)}).sort_values(by='# Miss
```

Out[5]:

	# Missing Values	% of Missing Values
Cabin	687	77.10
Age	177	19.87
Embarked	2	0.22
PassengerId	0	0.00
Survived	0	0.00
Pclass	0	0.00
Name	0	0.00
Sex	0	0.00
SibSp	0	0.00
Parch	0	0.00
Ticket	0	0.00
Fare	0	0.00

In [6]:

```
df.describe()
```

Out[6]:

	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

Imputing Missing Values

In [7]:

```
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace = True)
pd.DataFrame(data={'# Missing Values':df.isna().sum(),
                '% of Missing Values':round(df.isnull().sum()/df.isnull().count()*100,2)}).sort_values(by='# Miss
```

Out[7]:

	# Missing Values	% of Missing Values
Cabin	687	77.10
Age	177	19.87
PassengerId	0	0.00

Survived	0	0.00
Pclass	0	0.00
Name	0	0.00
Sex	0	0.00
SibSp	0	0.00
Parch	0	0.00
Ticket	0	0.00
Fare	0	0.00
Embarked	0	0.00

In [8]:

```
from sklearn.impute import KNNImputer

dfB = pd.DataFrame(df.Age)

imputer = KNNImputer(n_neighbors=10)
dfA = pd.DataFrame(imputer.fit_transform(dfB), columns=['Age'])
dfA.Age=dfA.Age.round()
df.Age=dfA.Age
df
```

Out[8]:

	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	30.0	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

In [9]:

```
df.to_csv('new.csv')
```

In [10]:

```
pd.DataFrame(data={'# Missing Values':df.isna().sum(),
                   '% of Missing Values':round(df.isnull().sum()/df.isnull().count()*100,2)}).sort_values(by='# Miss
```

Out[10]:

	# Missing Values	% of Missing Values
Cabin	687	77.1
PassengerId	0	0.0
Survived	0	0.0
Pclass	0	0.0
Name	0	0.0
Sex	0	0.0
Age	0	0.0
SibSp	0	0.0
Parch	0	0.0
Ticket	0	0.0
Fare	0	0.0
Embarked	0	0.0

In [11]:

```python
df1=df.copy()

df1.Age=df1.Age.astype('int64')

df1.drop(['PassengerId','Cabin','Name','Ticket'],axis=1,inplace=True)
df1= df1[['Sex','Age','SibSp','Parch','Pclass','Fare','Embarked','Survived']]
df1.sample(10)
```

Out[11]:

	Sex	Age	SibSp	Parch	Pclass	Fare	Embarked	Survived
870	male	26	0	0	3	7.8958	S	0
550	male	17	0	2	1	110.8833	C	1
660	male	50	2	0	1	133.6500	S	1
848	male	28	0	1	2	33.0000	S	0
351	male	30	0	0	1	35.0000	S	0
734	male	23	0	0	2	13.0000	S	0
20	male	35	0	0	2	26.0000	S	0
412	female	33	1	0	1	90.0000	Q	1
64	male	30	0	0	1	27.7208	C	0
203	male	46	0	0	3	7.2250	C	0

In [12]:

```python
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 8 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Sex       891 non-null    object 
 1   Age       891 non-null    int64  
 2   SibSp     891 non-null    int64  
 3   Parch     891 non-null    int64  
 4   Pclass    891 non-null    int64  
 5   Fare      891 non-null    float64
 6   Embarked  891 non-null    object 
 7   Survived  891 non-null    int64  
dtypes: float64(1), int64(5), object(2)
memory usage: 55.8+ KB
```

In [13]:

```python
print('The number of rows (observations) is',df1.shape[0],'\n''The number of columns (variables) is',df1.shape[1]
df_eda=df1.copy()
```

```
The number of rows (observations) is 891
The number of columns (variables) is 8
Total values in dataset: 7128
```

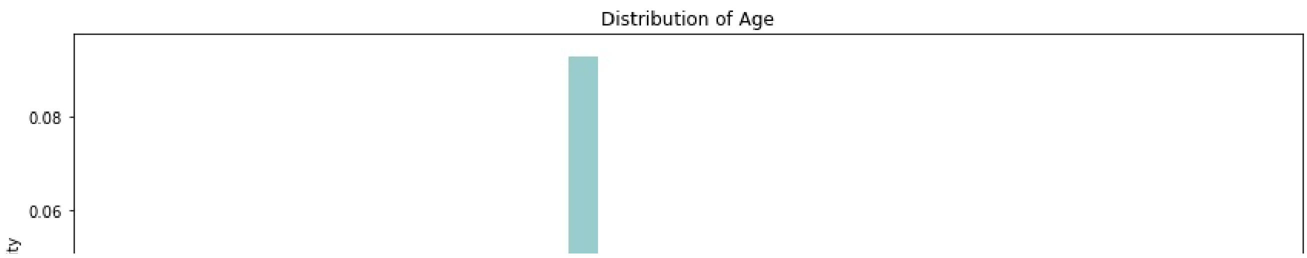
EDA

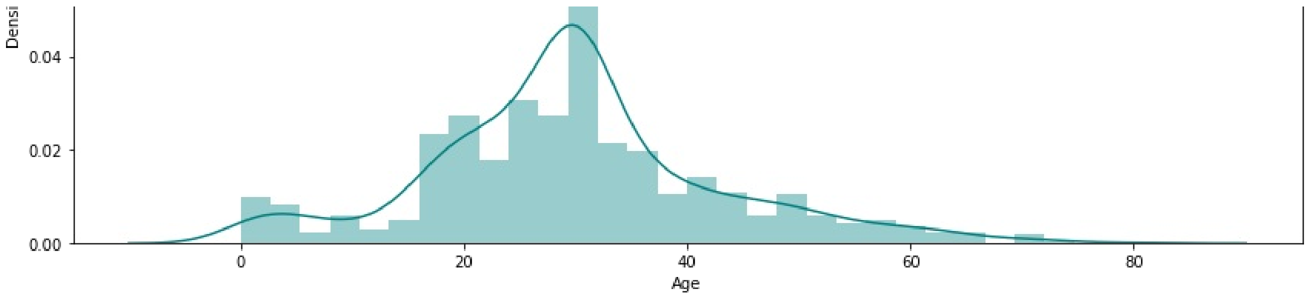
In [14]:

```python
def uniplot(x):
    fig,axs = plt.subplots(2,figsize=(12,10))
    axs[0].set_title('Distribution of '+x)
    sns.distplot(df_eda[x],ax=axs[0],color='teal')
    axs[1].set_title('Boxplot of '+x+' to see outliers')
    sns.boxplot(df_eda[x],ax=axs[1],palette='dark')
    plt.tight_layout()
    plt.show()
```

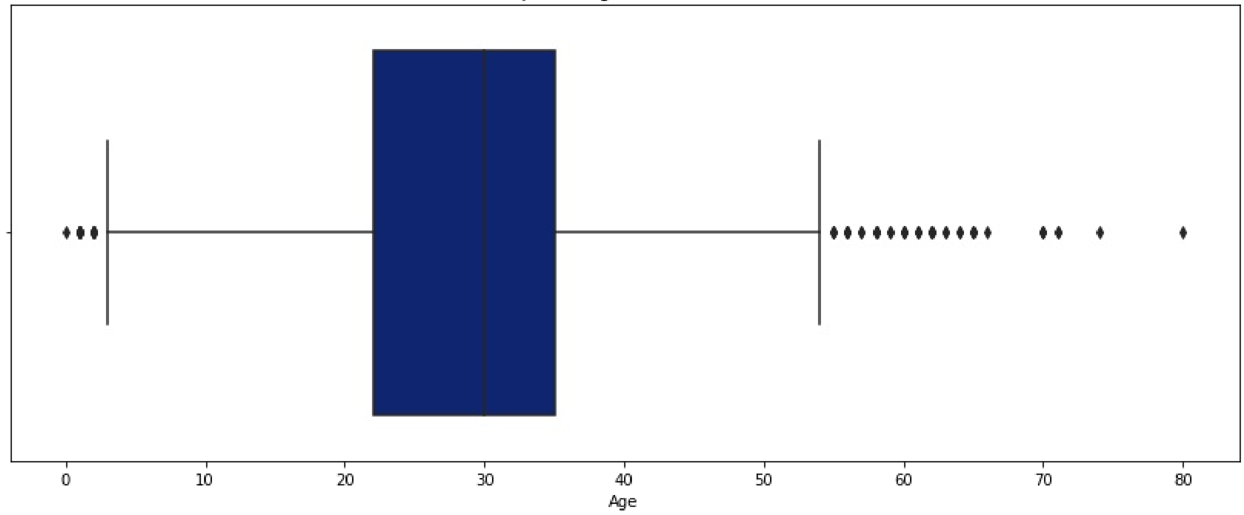
In [15]:

```python
uniplot('Age')
```



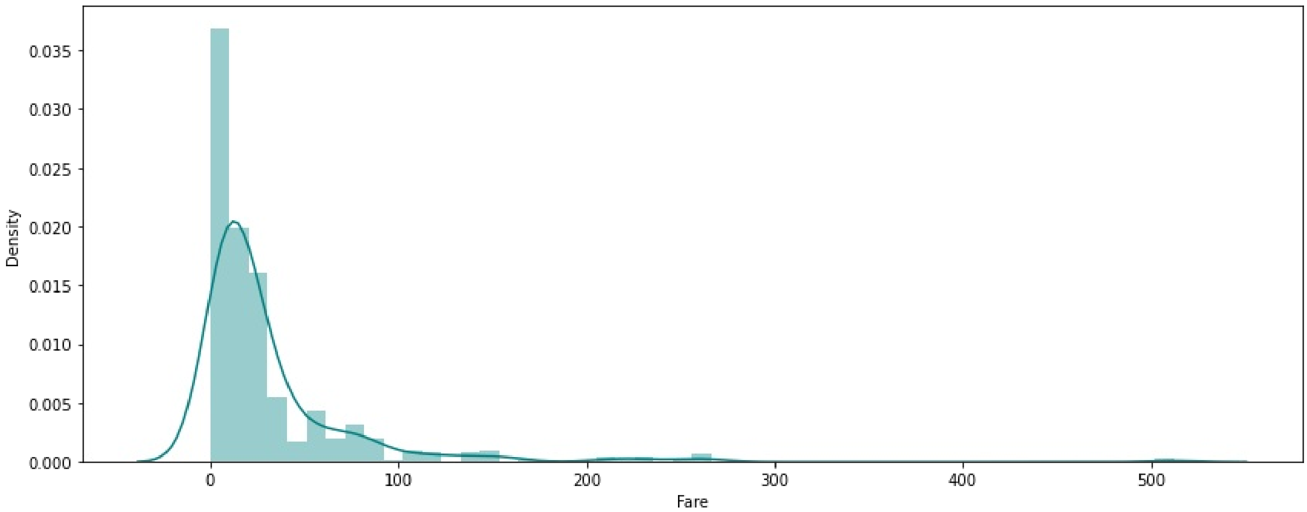


Boxplot of Age to see outliers

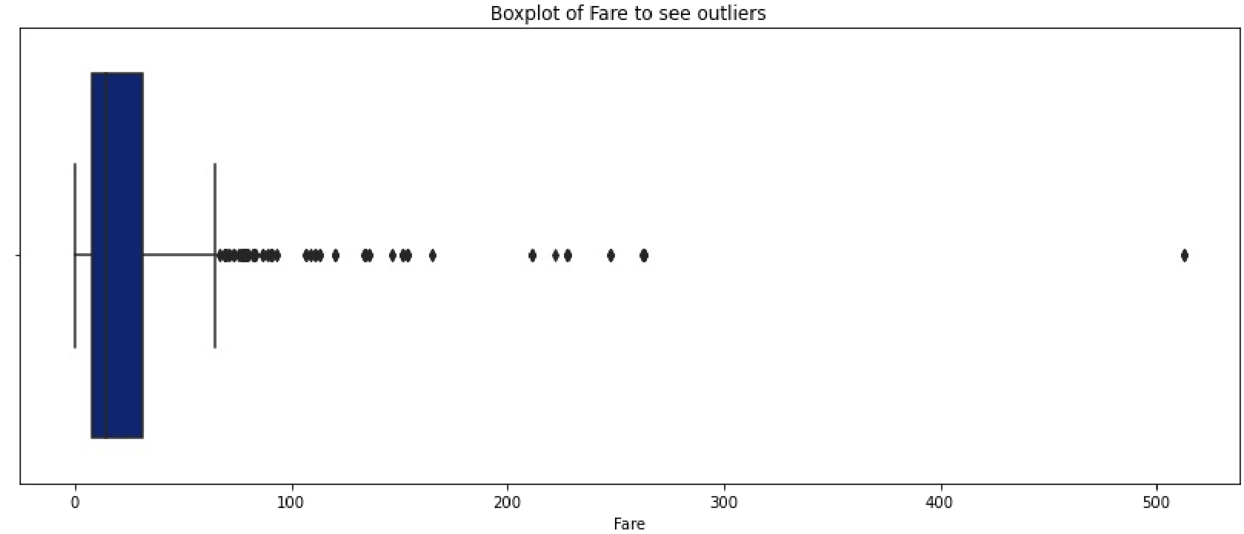


In [16]:
```python
uniplot('Fare')
```

Distribution of Fare



Boxplot of Fare to see outliers



In [17]:
```python
# Survived
```

```
print(df_eda.Survived.value_counts())
srate = round((sum(df_eda[df_eda['Survived']==1]['Survived'])/len(df_eda['Survived'].index))*100,2)
print('Survival Rate : ',srate,'%')
```

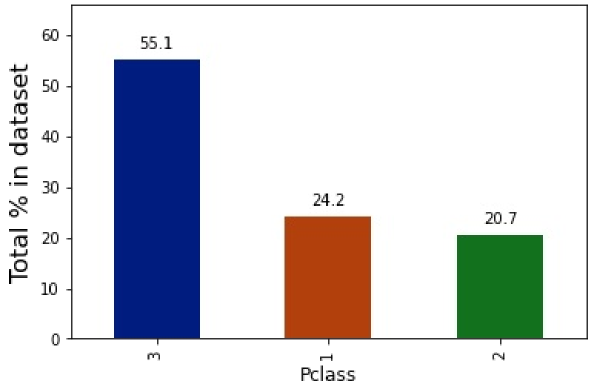
```
0    549
1    342
Name: Survived, dtype: int64
Survival Rate :  38.38 %
```

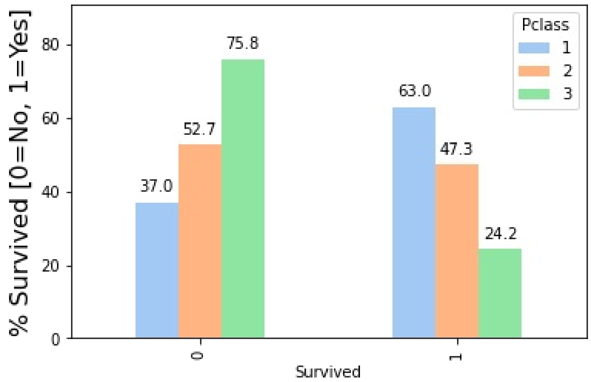
In [18]:

```
# Functions for visualization and beautifications

## Wonderful solution to show labels in bar plots - copied from https://stackoverflow.com/a/48372659 and edited
def showLabels(ax, d=None):
    plt.margins(0.2, 0.2)
    rects = ax.patches
    i = 0
    locs, labels = plt.xticks()
    counts = {}
    if not d is None:
        for key, value in d.items():
            counts[str(key)] = value

    # For each bar: Place a label
    for rect in rects:
        # Get X and Y placement of label from rect.
        y_value = rect.get_height()
        x_value = rect.get_x() + rect.get_width() / 2

        # Number of points between bar and label. Change to your liking.
        space = 5
        # Vertical alignment for positive values
        va = 'bottom'

        # If value of bar is negative: Place label below bar
        if y_value < 0:
            # Invert space to place label below
            space *= -1
            # Vertically align label at top
            va = 'top'

        # Use Y value as label and format number with one decimal place
        if d is None:
            label = "{:.1f}".format(y_value)
        else:
            try:
                label = "{:.1f}%".format(y_value) + "\nof " + str(counts[str(labels[i].get_text())])
            except:
                label = "{:.1f}%".format(y_value)

        i = i+1

        # Create annotation
        plt.annotate(
            label,                      # Use `label` as label
            (x_value, y_value),         # Place label at end of the bar
            xytext=(0, space),          # Vertically shift label by `space`
            textcoords="offset points", # Interpret `xytext` as offset in points
            ha='center',                # Horizontally center label
            va=va)                      # Vertically align label differently for
                                        # positive and negative values.

def plot_percentages(dataframe, by, sortbyindex=False):
    #plt.subplot(1, 2, 1)
    values = (dataframe[by].value_counts(normalize=True)*100)
    if sortbyindex:
        values = values.sort_index()
    ax = values.plot.bar(color=sns.color_palette('dark',16))
    ax.set_ylabel('Total % in dataset', fontsize=16)
    ax.set_xlabel(by, fontsize=12)
    showLabels(ax)

    grp = dataframe.groupby(['Survived',by])[by].count()
    cnt = dataframe.groupby(by)[by].count()

    percentages = grp.unstack() * 100 / cnt.T

    ax = percentages.plot.bar(color=sns.color_palette('pastel', 16))
    ax.set_ylabel('% Survived [0=No, 1=Yes]', fontsize=16)
    showLabels(ax)
```

In [19]:

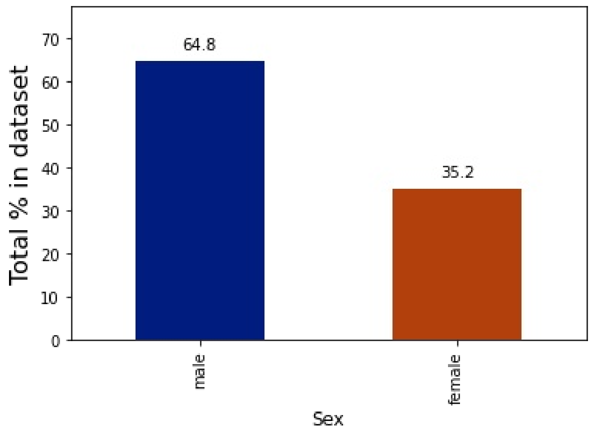
```
# Pclass
plot_percentages(df_eda,'Pclass')
plt.show()
```

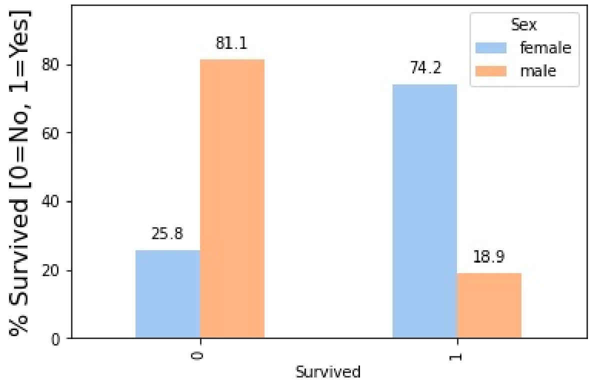




In [20]:

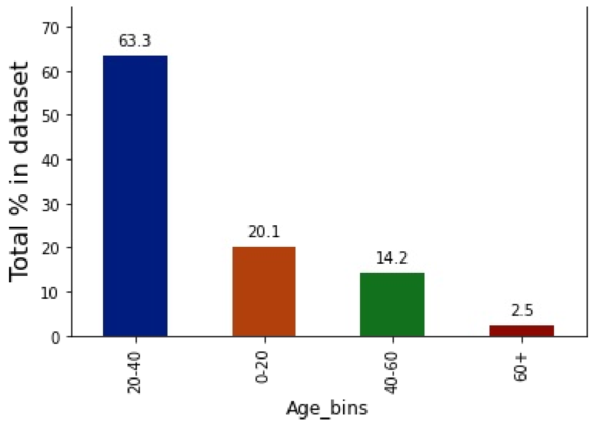
```
# Sex
plot_percentages(df_eda,'Sex')
plt.show()
```

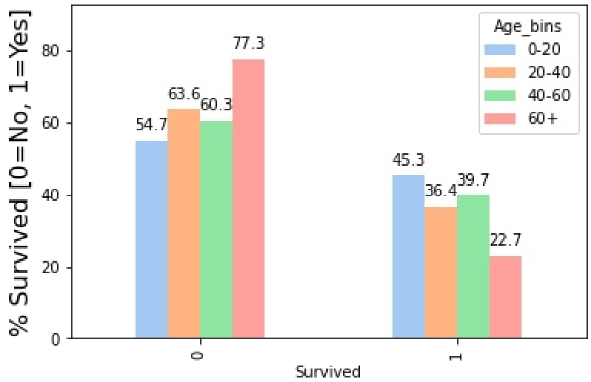




In [21]:

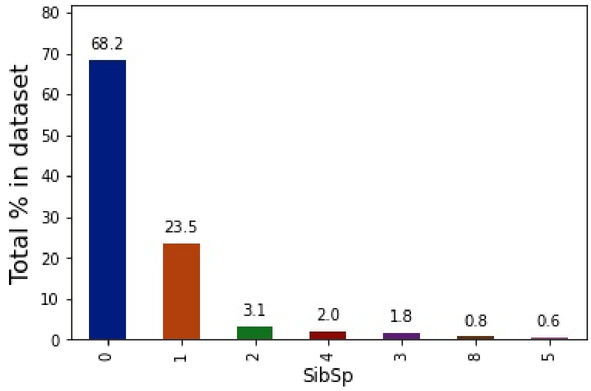
```
# Age
bins = [0,20,40,60,np.inf]
labels = ['0-20','20-40','40-60','60+']
age_bins = pd.cut(df1.Age, bins, labels=labels)
df1['Age_bins'] = age_bins

plot_percentages(df1,'Age_bins')
```

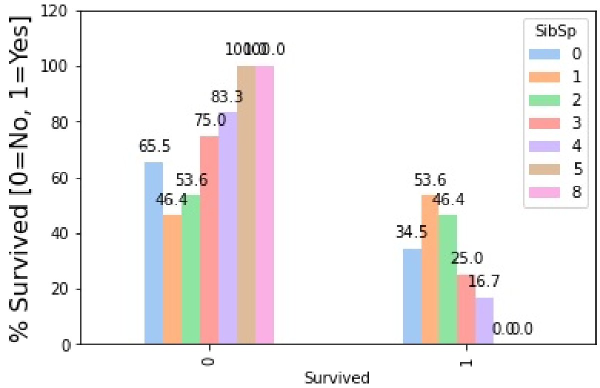




In [22]:

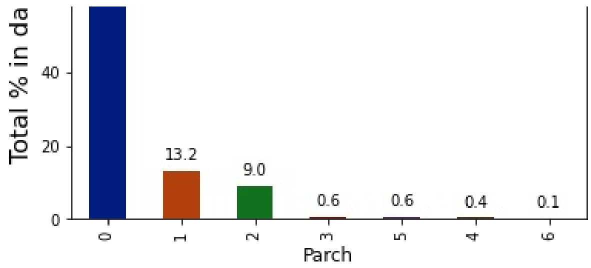
```
# SibSp
plot_percentages(df_eda,'SibSp')
plt.show()
```

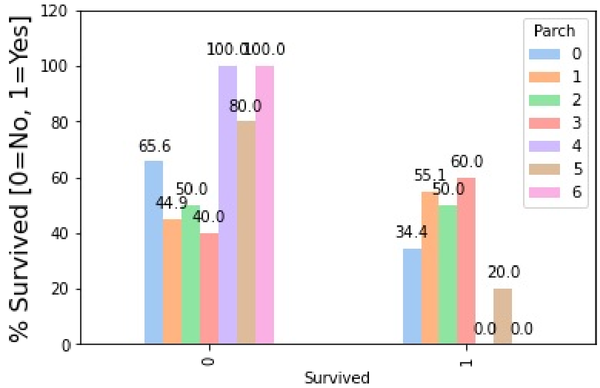




In [23]:

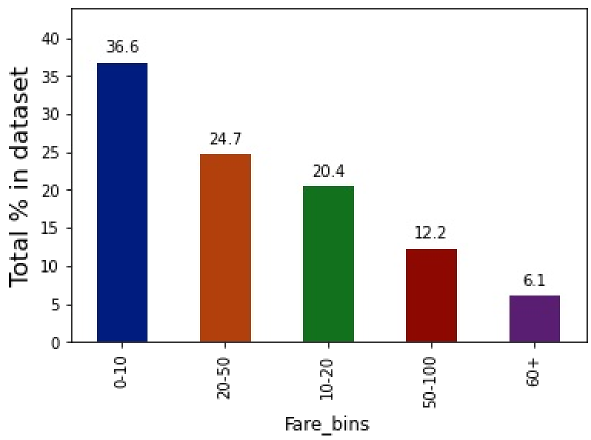
```
# Parch
plot_percentages(df_eda,'Parch')
plt.show()
```

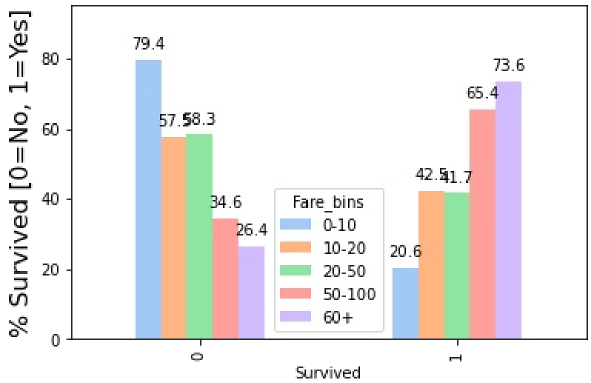




In [24]:

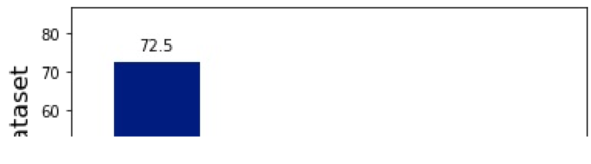
```
# Fare
bins = [0,10,20,50,100,np.inf]
labels = ['0-10','10-20','20-50','50-100','60+']
fare_bins = pd.cut(df1.Fare, bins, labels=labels)
df1['Fare_bins'] = fare_bins

plot_percentages(df1,'Fare_bins')
```

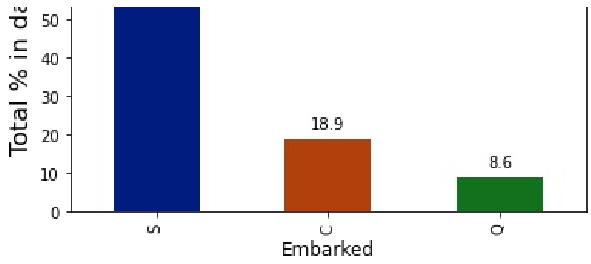


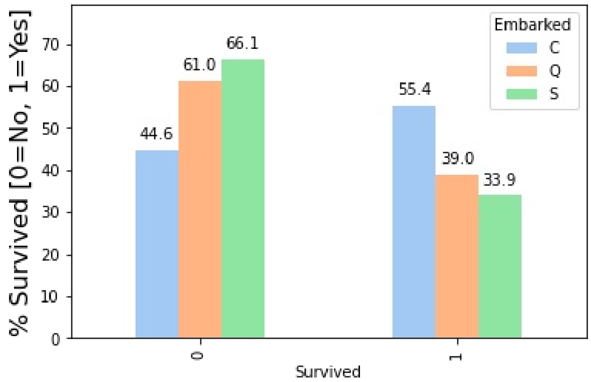


In [25]:

```
# Embark
plot_percentages(df_eda,'Embarked')
plt.show()
```







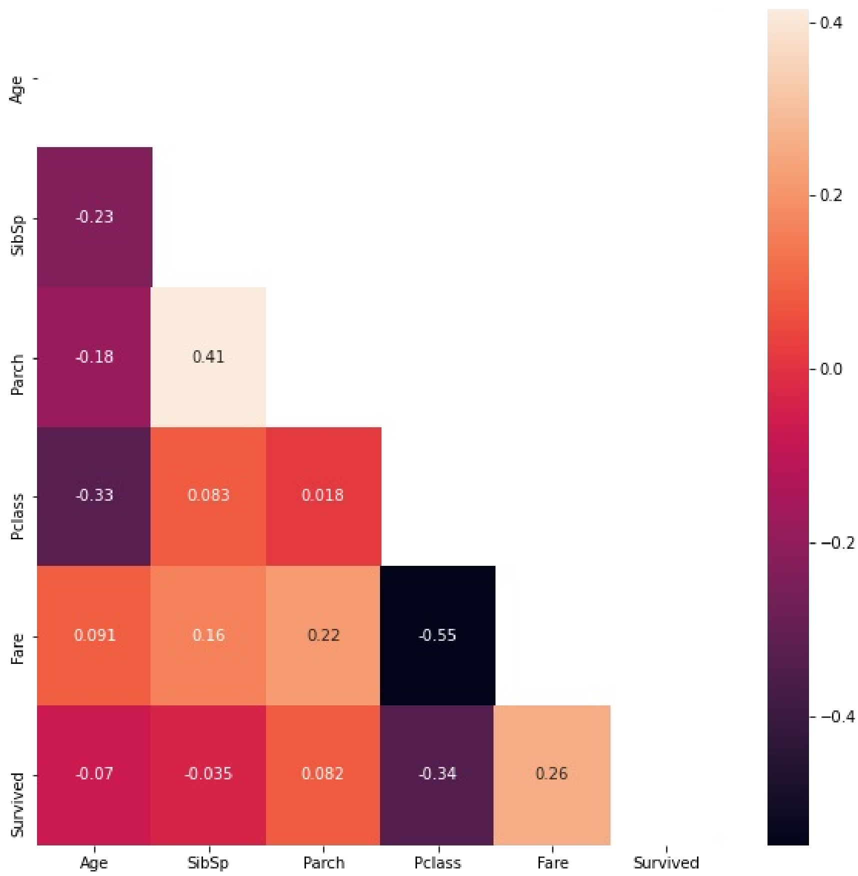
In [26]:

```
# Tickets, Cabins and Names

print('There are {} unique Tickets values in the provided data'.format(len(df.Ticket.unique())))
print('\nThere are {} unique Cabin values excluding the nulls in the provided data'.format(len(df.Cabin.unique())
print('\nThere are {} Unique passanger names'.format(len(df.Name.unique())))
```

```
There are 681 unique Tickets values in the provided data

There are 148 unique Cabin values excluding the nulls in the provided data

There are 891 Unique passanger names
```

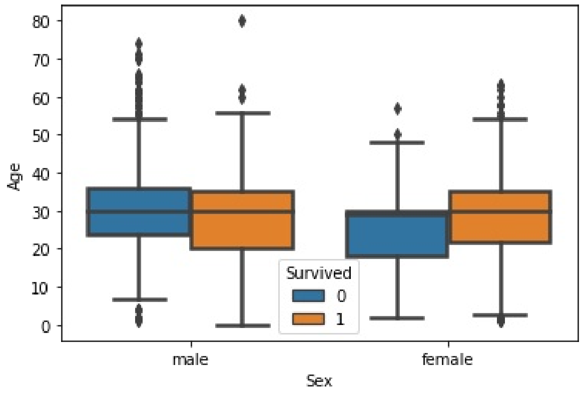
In [27]:

```
corr = df1.corr()
mask = np.triu(np.ones_like(corr, dtype=np.bool)) #masks upper triangle
fig = plt.subplots(figsize=(10, 10))
sns.heatmap(corr, annot=True,mask=mask)
plt.show()
```



In [28]:

```
ax = sns.boxplot(x="Sex", y="Age", hue="Survived",data=df1, linewidth=2.5)
```



In [29]:

```
#df1.drop(['Age','Fare'],axis=1,inplace=True)
df1= df1[['Sex','Age_bins','SibSp','Parch','Pclass','Fare_bins','Embarked','Survived']]
df1.sample(5)
```

Out[29]:

	Sex	Age_bins	SibSp	Parch	Pclass	Fare_bins	Embarked	Survived
287	male	20-40	0	0	3	0-10	S	0
657	female	20-40	1	1	3	10-20	Q	0
179	male	20-40	0	0	3	NaN	S	0
762	male	0-20	0	0	3	0-10	C	1
472	female	20-40	1	2	2	20-50	S	1

In [30]:

```
dummy=pd.get_dummies(df1,drop_first=True)
dummy.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   SibSp            891 non-null    int64
 1   Parch            891 non-null    int64
 2   Pclass           891 non-null    int64
 3   Survived         891 non-null    int64
 4   Sex_male         891 non-null    uint8
 5   Age_bins_20-40   891 non-null    uint8
 6   Age_bins_40-60   891 non-null    uint8
 7   Age_bins_60+     891 non-null    uint8
 8   Fare_bins_10-20  891 non-null    uint8
 9   Fare_bins_20-50  891 non-null    uint8
 10  Fare_bins_50-100 891 non-null    uint8
 11  Fare_bins_60+    891 non-null    uint8
 12  Embarked_Q       891 non-null    uint8
 13  Embarked_S       891 non-null    uint8
dtypes: int64(4), uint8(10)
memory usage: 36.7 KB
```

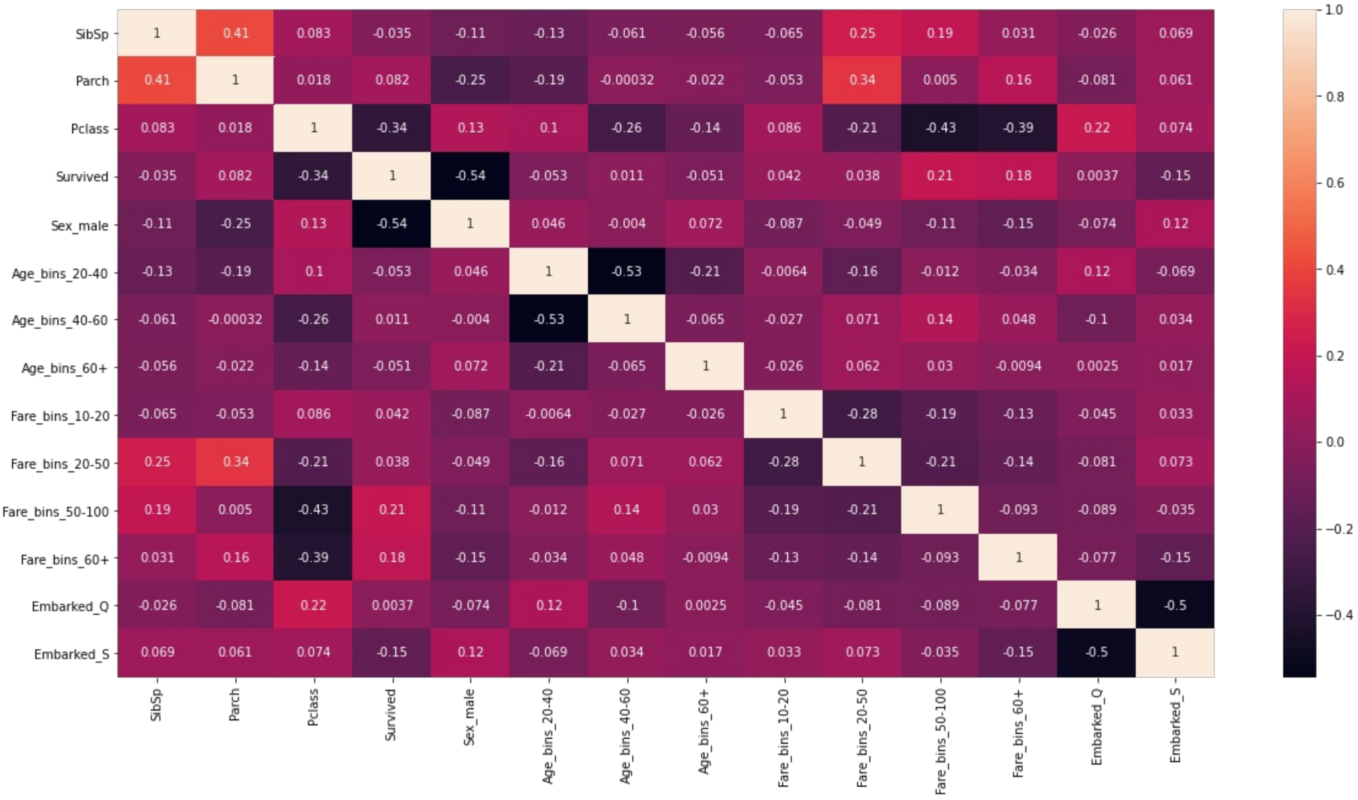
In [31]:

```
dummy.sample(5)
```

Out[31]:

	SibSp	Parch	Pclass	Survived	Sex_male	Age_bins_20-40	Age_bins_40-60	Age_bins_60+	Fare_bins_10-20	Fare_bins_20-50	Fare_bins_50-100	Fare_
315	0	0	3	1	0	1	0	0	0	0	0	
214	1	0	3	0	1	1	0	0	0	0	0	
445	0	2	1	1	1	0	0	0	0	0	1	
602	0	0	1	0	1	1	0	0	0	1	0	
622	1	1	3	1	1	0	0	0	1	0	0	

```
In [32]: plt.figure(figsize = (20,10))         # Size of the figure
         sns.heatmap(dummy.corr(),annot = True)
         plt.show()
```




```
In [33]: from sklearn.model_selection import train_test_split
         from sklearn.model_selection import GridSearchCV

         X= dummy.drop('Survived', axis =1) #features
         Y= dummy.pop ('Survived') #target

         Xtrain, Xtest, Ytrain, Ytest = train_test_split(X, Y, test_size=0.3, random_state=50)
         Xtrain.shape
```

```
Out[33]: (623, 13)
```

Logistic Regression

```
In [34]: #Method 1- Default Parameters
         from sklearn.linear_model import LogisticRegression
         from sklearn import metrics,model_selection
         from sklearn.metrics import roc_auc_score,roc_curve,classification_report,confusion_matrix,plot_confusion_matrix

         model = LogisticRegression()
         model.fit(Xtrain, Ytrain)

         LR_Ytrain_predict = model.predict(Xtrain)
         LR_Ytest_predict = model.predict(Xtest)

         print('Train Set Accuracy:-',model.score(Xtrain, Ytrain))
         print('Test Set Accuracy:-',model.score(Xtest, Ytest))
         print('\nConfusion Matrix for Train Set:-\n',confusion_matrix(Ytrain, LR_Ytrain_predict))
         print('\nClassification Report for Train Set:-\n',classification_report(Ytrain, LR_Ytrain_predict))
         print('\nConfusion Matrix for Test Set:-\n',confusion_matrix(Ytest, LR_Ytest_predict))
         print('\nClassification Report for Test Set:-\n',classification_report(Ytest, LR_Ytest_predict))
```

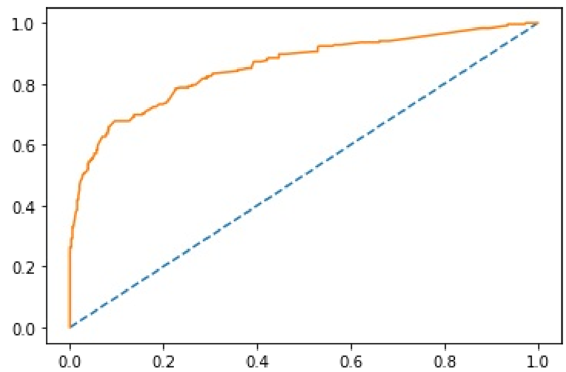
```
Train Set Accuracy:- 0.8041733547351525
Test Set Accuracy:- 0.8059701492537313

Confusion Matrix for Train Set:-
 [[341  46]
 [ 76 160]]

Classification Report for Train Set:-
               precision    recall  f1-score   support
```

```
           0       0.82      0.88      0.85       387
           1       0.78      0.68      0.72       236

    accuracy                           0.80       623
   macro avg       0.80      0.78      0.79       623
weighted avg       0.80      0.80      0.80       623


Confusion Matrix for Test Set:-
 [[143  19]
 [ 33  73]]

Classification Report for Test Set:-
               precision    recall  f1-score   support

           0       0.81      0.88      0.85       162
           1       0.79      0.69      0.74       106

    accuracy                           0.81       268
   macro avg       0.80      0.79      0.79       268
weighted avg       0.80      0.81      0.80       268
```

In [35]:

```python
#AUC and ROC for train data wrt LR
probs = model.predict_proba(Xtrain)
probs = probs[:, 1]
LRbasic_train_auc = roc_auc_score(Ytrain, probs)
print('AUC: %.3f' % LRbasic_train_auc)
train_fpr, train_tpr, train_thresholds = roc_curve(Ytrain, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(train_fpr, train_tpr);
```

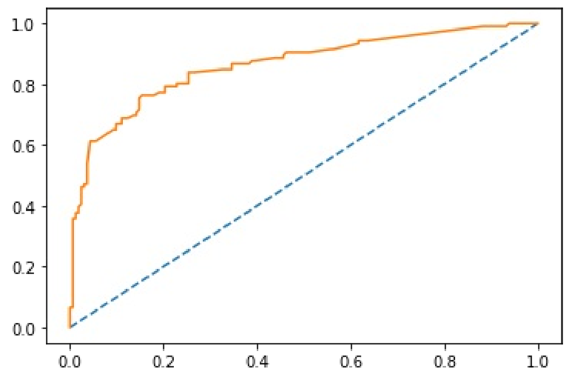
```
AUC: 0.855
```



In [36]:

```python
#AUC and ROC for test data wrt LR
probs = model.predict_proba(Xtest)
probs = probs[:, 1]
LRbasic_test_auc = roc_auc_score(Ytest, probs)
print('AUC: %.3f' % LRbasic_test_auc)
test_fpr, test_tpr, test_thresholds = roc_curve(Ytest, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(test_fpr, test_tpr);
```

```
AUC: 0.863
```



In [37]:
```
#Method 2- Grid Search Logistic Regression
grid={'penalty':['l2','none'],
      'solver':['liblinear','lbfgs','saga'],
      'tol':[0.001,0.00001]}

LR = LogisticRegression(max_iter=1000,n_jobs=2)
grid_search = GridSearchCV(estimator = LR, param_grid = grid, cv = 3,scoring='f1')
grid_search.fit(Xtrain, Ytrain)
```

Out[37]:
```
GridSearchCV(cv=3, estimator=LogisticRegression(max_iter=1000, n_jobs=2),
             param_grid={'penalty': ['l2', 'none'],
                         'solver': ['liblinear', 'lbfgs', 'saga'],
                         'tol': [0.001, 1e-05]},
             scoring='f1')
```

In [38]:
```
print(grid_search.best_params_,'\n')
print(grid_search.best_estimator_)
```

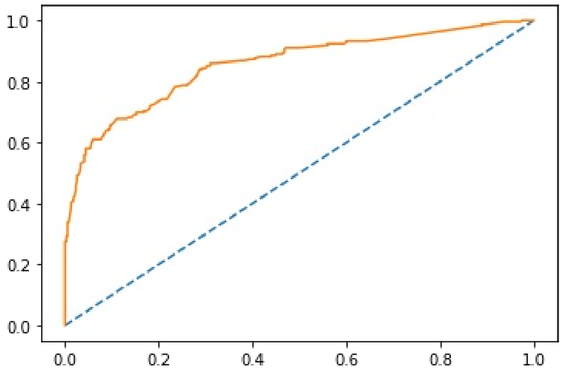
```
{'penalty': 'l2', 'solver': 'liblinear', 'tol': 0.001}

LogisticRegression(max_iter=1000, n_jobs=2, solver='liblinear', tol=0.001)
```

In [39]:
```
bestmodel = grid_search.best_estimator_

LRM_Ytrain_predict = bestmodel.predict(Xtrain)
LRM_train_score = bestmodel.score(Xtrain, Ytrain)
print('Train Set Accuracy:',LRM_train_score)
print('Confusion Matrix:-\n',confusion_matrix(Ytrain, LRM_Ytrain_predict))
print('\nClassification Report:-\n',classification_report(Ytrain, LRM_Ytrain_predict))

LRM_Ytest_predict = bestmodel.predict(Xtest)
LRM_test_score = bestmodel.score(Xtest, Ytest)
print('Test Set Accuracy:',LRM_test_score)
print('Confusion Matrix:-\n',confusion_matrix(Ytest, LRM_Ytest_predict))
print('\nClassification Report:-\n',classification_report(Ytest, LRM_Ytest_predict))
```

```
Train Set Accuracy: 0.8025682182985554
Confusion Matrix:-
 [[340  47]
 [ 76 160]]

Classification Report:-
               precision    recall  f1-score   support

           0       0.82      0.88      0.85       387
           1       0.77      0.68      0.72       236

    accuracy                           0.80       623
   macro avg       0.80      0.78      0.78       623
weighted avg       0.80      0.80      0.80       623

Test Set Accuracy: 0.8022388059701493
Confusion Matrix:-
 [[143  19]
 [ 34  72]]

Classification Report:-
               precision    recall  f1-score   support

           0       0.81      0.88      0.84       162
           1       0.79      0.68      0.73       106

    accuracy                           0.80       268
   macro avg       0.80      0.78      0.79       268
weighted avg       0.80      0.80      0.80       268
```

In [40]:
```
#AUC and ROC for train data wrt LRM
probs = bestmodel.predict_proba(Xtrain)
probs = probs[:, 1]
LR_train_auc = roc_auc_score(Ytrain, probs)
print('AUC: %.3f' % LR_train_auc)
train_fpr, train_tpr, train_thresholds = roc_curve(Ytrain, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(train_fpr, train_tpr);
```

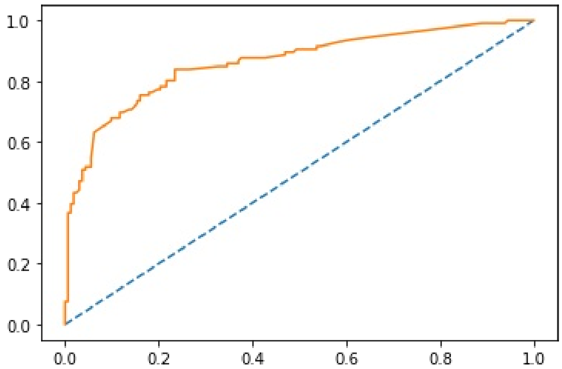
AUC: 0.855



In [41]:

```
#AUC and ROC for test data wrt LRM
probs = bestmodel.predict_proba(Xtest)
probs = probs[:, 1]
LR_test_auc = roc_auc_score(Ytest, probs)
print('AUC: %.3f' % LR_test_auc)
test_fpr, test_tpr, test_thresholds = roc_curve(Ytest, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(test_fpr, test_tpr);
```

AUC: 0.862



KNN

In [42]:

```
from sklearn.neighbors import KNeighborsClassifier

KNN=KNeighborsClassifier()
KNN.fit(Xtrain,Ytrain)
```

Out[42]: KNeighborsClassifier()

In [43]:

```
#Performance on Train Dataset wrt KNN
KNN_Ytrain_predict = KNN.predict(Xtrain)
KNN_train_score = KNN.score(Xtrain, Ytrain)
print('Train Data Accuracy:-',KNN_train_score)
print('\nConfusion Matrix:-\n',metrics.confusion_matrix(Ytrain, KNN_Ytrain_predict))
print('\nClassification Report:-\n',metrics.classification_report(Ytrain, KNN_Ytrain_predict))

#Performance on Test Dataset wrt KNN
KNN_Ytest_predict = KNN.predict(Xtest)
KNN_test_score = KNN.score(Xtest, Ytest)
print('Test Data Accuracy:-',KNN_test_score)
print('\nConfusion Matrix:-\n',metrics.confusion_matrix(Ytest, KNN_Ytest_predict))
print('\nClassification Report:-\n',metrics.classification_report(Ytest, KNN_Ytest_predict))
```

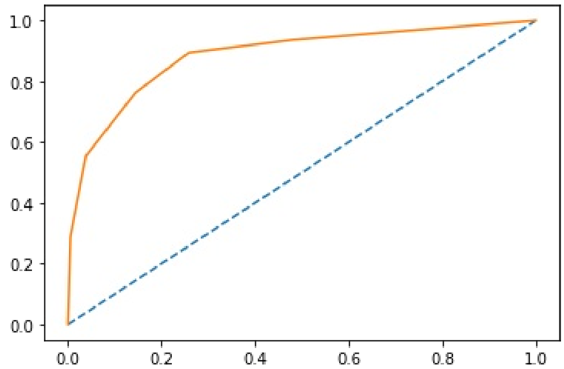
```
Train Data Accuracy:- 0.8202247191011236

Confusion Matrix:-
 [[331  56]
 [ 56 180]]

Classification Report:-
```

```
              precision    recall  f1-score   support

           0       0.86      0.86      0.86       387
           1       0.76      0.76      0.76       236

    accuracy                           0.82       623
   macro avg       0.81      0.81      0.81       623
weighted avg       0.82      0.82      0.82       623

Test Data Accuracy:- 0.7798507462686567

Confusion Matrix:-
 [[141  21]
 [ 38  68]]

Classification Report:-
              precision    recall  f1-score   support

           0       0.79      0.87      0.83       162
           1       0.76      0.64      0.70       106

    accuracy                           0.78       268
   macro avg       0.78      0.76      0.76       268
weighted avg       0.78      0.78      0.78       268
```

In [44]:

```
#AUC and ROC for train data wrt KNN
probs = KNN.predict_proba(Xtrain)
probs = probs[:, 1]
KNN_train_auc = roc_auc_score(Ytrain, probs)
print('AUC: %.3f' % KNN_train_auc)
train_fpr, train_tpr, train_thresholds = roc_curve(Ytrain, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(train_fpr, train_tpr);
```

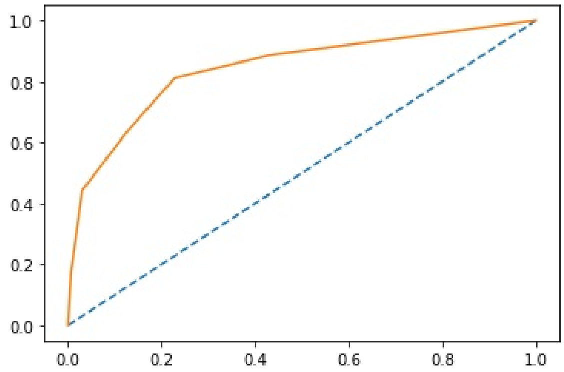
AUC: 0.885



In [45]:

```
#AUC and ROC for test data wrt KNN
probs = KNN.predict_proba(Xtest)
probs = probs[:, 1]
KNN_test_auc = roc_auc_score(Ytest, probs)
print('AUC: %.3f' % KNN_test_auc)
test_fpr, test_tpr, test_thresholds = roc_curve(Ytest, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(test_fpr, test_tpr);
```

AUC: 0.842



In [ ]:

In [46]:

```
KNN = KNeighborsClassifier(n_jobs=-1)
params = {'n_neighbors':[3,5,7,9],
          'leaf_size':[5,10,15],
          'weights':['uniform', 'distance'],
          'algorithm':['auto', 'ball_tree','kd_tree','brute'],
          'n_jobs':[-1]}
gridKNN = GridSearchCV(KNN, param_grid=params, n_jobs=1)
gridKNN.fit(Xtrain,Ytrain)
```

Out[46]:

```
GridSearchCV(estimator=KNeighborsClassifier(n_jobs=-1), n_jobs=1,
             param_grid={'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
                         'leaf_size': [5, 10, 15], 'n_jobs': [-1],
                         'n_neighbors': [3, 5, 7, 9],
                         'weights': ['uniform', 'distance']})
```

In [47]:

```
print(gridKNN.best_params_,'\n')
print(gridKNN.best_estimator_)
```

```
{'algorithm': 'ball_tree', 'leaf_size': 10, 'n_jobs': -1, 'n_neighbors': 9, 'weights': 'distance'}

KNeighborsClassifier(algorithm='ball_tree', leaf_size=10, n_jobs=-1,
                     n_neighbors=9, weights='distance')
```

In [48]:

```
bestmodelKNN = gridKNN.best_estimator_

KNNM_Ytrain_predict = bestmodelKNN.predict(Xtrain)
KNNM_train_score = bestmodelKNN.score(Xtrain, Ytrain)
print('Train Set Accuracy:',KNNM_train_score)
print('Confusion Matrix:-\n',confusion_matrix(Ytrain, KNNM_Ytrain_predict))
print('\nClassification Report:-\n',classification_report(Ytrain, KNNM_Ytrain_predict))

KNNM_Ytest_predict = bestmodelKNN.predict(Xtest)
KNNM_test_score = bestmodelKNN.score(Xtest, Ytest)
print('Test Set Accuracy:',KNNM_test_score)
print('Confusion Matrix:-\n',confusion_matrix(Ytest, KNNM_Ytest_predict))
print('\nClassification Report:-\n',classification_report(Ytest, KNNM_Ytest_predict))
```

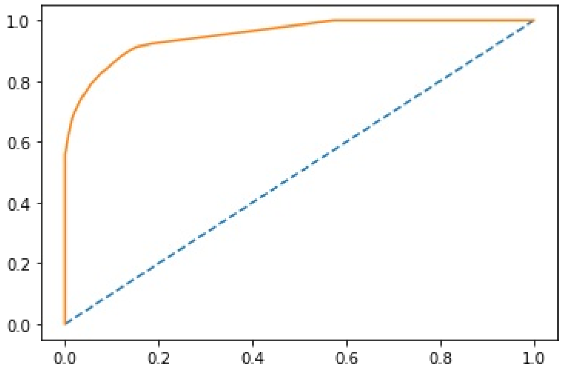
```
Train Set Accuracy: 0.8860353130016051
Confusion Matrix:-
 [[366  21]
 [ 50 186]]

Classification Report:-
               precision    recall  f1-score   support

           0       0.88      0.95      0.91       387
           1       0.90      0.79      0.84       236

    accuracy                           0.89       623
   macro avg       0.89      0.87      0.88       623
weighted avg       0.89      0.89      0.88       623

Test Set Accuracy: 0.8097014925373134
Confusion Matrix:-
 [[150  12]
 [ 39  67]]

Classification Report:-
               precision    recall  f1-score   support

           0       0.79      0.93      0.85       162
           1       0.85      0.63      0.72       106

    accuracy                           0.81       268
   macro avg       0.82      0.78      0.79       268
weighted avg       0.82      0.81      0.80       268
```

In [49]:

```
#AUC and ROC for train data wrt KNN-M
```

```
probs = bestmodelKNN.predict_proba(Xtrain)
probs = probs[:, 1]
KNNM_train_auc = roc_auc_score(Ytrain, probs)
print('AUC: %.3f' % KNNM_train_auc)
train_fpr, train_tpr, train_thresholds = roc_curve(Ytrain, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(train_fpr, train_tpr);
```

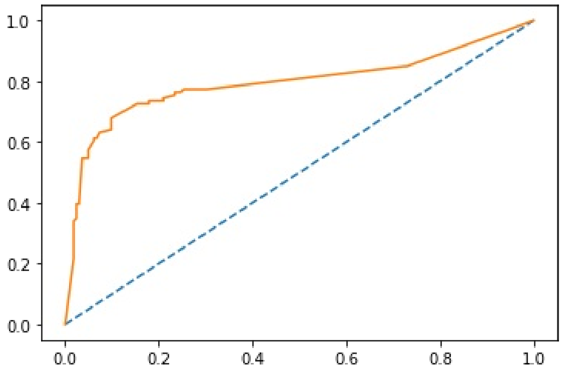
```
AUC: 0.953
```



In [50]:

```
#AUC and ROC for test data wrt KNN-M
probs = bestmodelKNN.predict_proba(Xtest)
probs = probs[:, 1]
KNNM_test_auc = roc_auc_score(Ytest, probs)
print('AUC: %.3f' % KNNM_test_auc)
test_fpr, test_tpr, test_thresholds = roc_curve(Ytest, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(test_fpr, test_tpr);
```

```
AUC: 0.794
```



AdaBoost

In [51]:

```
from sklearn.ensemble import AdaBoostClassifier

ADaBo = AdaBoostClassifier(n_estimators=100,random_state=1)
ADaBo.fit(Xtrain,Ytrain)
```

Out[51]:

```
AdaBoostClassifier(n_estimators=100, random_state=1)
```

In [52]:

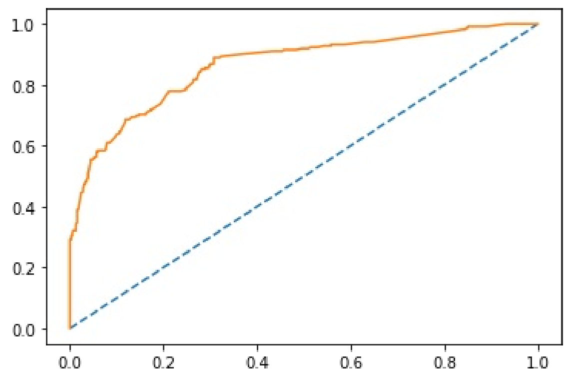
```
## Performance Matrix on train data set
ADaBo_Ytrain_predict = ADaBo.predict(Xtrain)
ADaBo_train_score = ADaBo.score(Xtrain, Ytrain)
print('Train Data Accuracy:-',ADaBo_train_score)
print('\nConfusion Matrix:-\n',metrics.confusion_matrix(Ytrain, ADaBo_Ytrain_predict))
print('\nClassification Report:-\n',metrics.classification_report(Ytrain, ADaBo_Ytrain_predict))

## Performance Matrix on test data set
ADaBo_Ytest_predict = ADaBo.predict(Xtest)
ADaBo_test_score = ADaBo.score(Xtest, Ytest)
print('Test Data Accuracy:-',ADaBo_test_score)
print('\nConfusion Matrix:-\n',metrics.confusion_matrix(Ytest, ADaBo_Ytest_predict))
print('\nClassification Report:-\n',metrics.classification_report(Ytest, ADaBo_Ytest_predict))
```

```
Train Data Accuracy:- 0.7961476725521669

Confusion Matrix:-
 [[330  57]
 [ 70 166]]

Classification Report:-
               precision    recall  f1-score   support

           0       0.82      0.85      0.84       387
           1       0.74      0.70      0.72       236

    accuracy                           0.80       623
   macro avg       0.78      0.78      0.78       623
weighted avg       0.79      0.80      0.79       623

Test Data Accuracy:- 0.7947761194029851

Confusion Matrix:-
 [[136  26]
 [ 29  77]]

Classification Report:-
               precision    recall  f1-score   support

           0       0.82      0.84      0.83       162
           1       0.75      0.73      0.74       106

    accuracy                           0.79       268
   macro avg       0.79      0.78      0.78       268
weighted avg       0.79      0.79      0.79       268
```

In [53]:

```
#AUC and ROC for train data wrt AdaBoost
probs = ADaBo.predict_proba(Xtrain)
probs = probs[:, 1]
ADaBo_train_auc = roc_auc_score(Ytrain, probs)
print('AUC: %.3f' % ADaBo_train_auc)
train_fpr, train_tpr, train_thresholds = roc_curve(Ytrain, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(train_fpr, train_tpr);
```

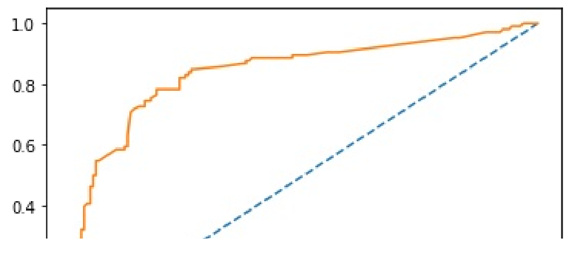
```
AUC: 0.864
```

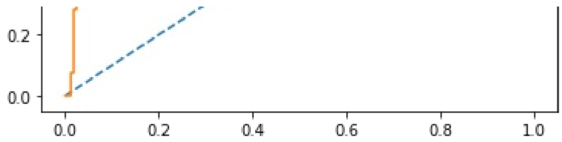


In [54]:

```
#AUC and ROC for test data wrt AdaBoost
probs = ADaBo.predict_proba(Xtest)
probs = probs[:, 1]
ADaBo_test_auc = roc_auc_score(Ytest, probs)
print('AUC: %.3f' % ADaBo_test_auc)
test_fpr, test_tpr, test_thresholds = roc_curve(Ytest, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(test_fpr, test_tpr);
```

```
AUC: 0.839
```





GradientBoost

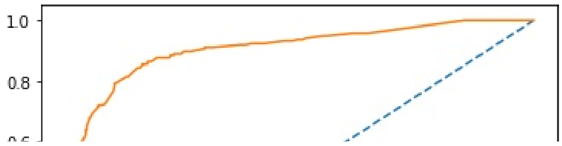
In [55]:
```
from sklearn.ensemble import GradientBoostingClassifier
GBo = GradientBoostingClassifier(random_state=1)
GBo = GBo.fit(Xtrain, Ytrain)
```

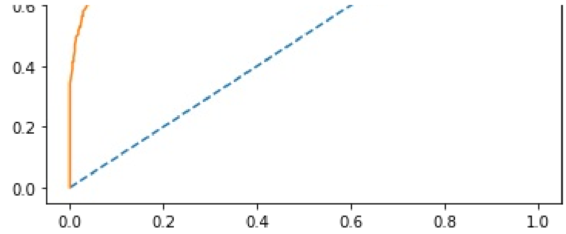
In [56]:
```
## Performance Matrix on train data set
GBo_Ytrain_predict = GBo.predict(Xtrain)
GBo_train_score = GBo.score(Xtrain, Ytrain)
print('Train Data Accuracy',GBo_train_score)
print('\nConfusion Matrix:-\n',metrics.confusion_matrix(Ytrain, GBo_Ytrain_predict))
print('\nClassification Report:-\n',metrics.classification_report(Ytrain, GBo_Ytrain_predict))

## Performance Matrix on test data set
GBo_Ytest_predict = GBo.predict(Xtest)
GBo_test_score = GBo.score(Xtest, Ytest)
print('Test Data Accuracy',GBo_test_score)
print('\nConfusion Matrix:-\n',metrics.confusion_matrix(Ytest, GBo_Ytest_predict))
print('\nClassification Report:-\n',metrics.classification_report(Ytest, GBo_Ytest_predict))
```

```
Train Data Accuracy 0.8475120385232745

Confusion Matrix:-
 [[360  27]
 [ 68 168]]

Classification Report:-
               precision    recall  f1-score   support

           0       0.84      0.93      0.88       387
           1       0.86      0.71      0.78       236

    accuracy                           0.85       623
   macro avg       0.85      0.82      0.83       623
weighted avg       0.85      0.85      0.84       623

Test Data Accuracy 0.8171641791044776

Confusion Matrix:-
 [[151  11]
 [ 38  68]]

Classification Report:-
               precision    recall  f1-score   support

           0       0.80      0.93      0.86       162
           1       0.86      0.64      0.74       106

    accuracy                           0.82       268
   macro avg       0.83      0.79      0.80       268
weighted avg       0.82      0.82      0.81       268
```

In [57]:
```
#AUC and ROC for train data wrt GradientBoost
probs = GBo.predict_proba(Xtrain)
probs = probs[:, 1]
GBo_train_auc = roc_auc_score(Ytrain, probs)
print('AUC: %.3f' % GBo_train_auc)
train_fpr, train_tpr, train_thresholds = roc_curve(Ytrain, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(train_fpr, train_tpr);
```

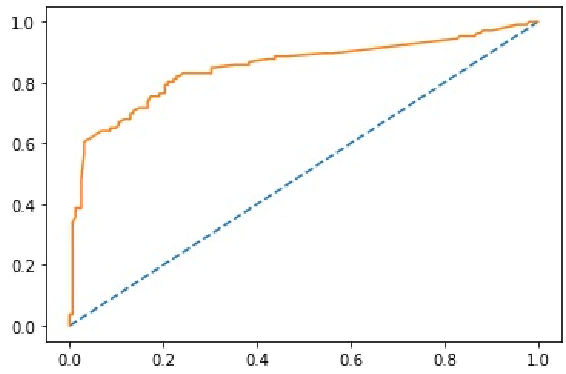
```
AUC: 0.908
```





```
In [58]: #AUC and ROC for test data wrt GradientBoost
probs = GBo.predict_proba(Xtest)
probs = probs[:, 1]
GBo_test_auc = roc_auc_score(Ytest, probs)
print('AUC: %.3f' % GBo_test_auc)
test_fpr, test_tpr, test_thresholds = roc_curve(Ytest, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(test_fpr, test_tpr);
```

```
AUC: 0.850
```



Random Forest

```
In [59]: from sklearn.ensemble import RandomForestClassifier

RF=RandomForestClassifier(n_estimators=100,random_state=1)
RF.fit(Xtrain, Ytrain)
```

```
Out[59]: RandomForestClassifier(random_state=1)
```

```
In [60]: ## Performance Matrix on train data set
RF_Ytrain_predict = RF.predict(Xtrain)
RF_train_score =RF.score(Xtrain, Ytrain)
print('Train Data Accuracy:-',RF_train_score)
print('\nConfusion Matrix:-\n',metrics.confusion_matrix(Ytrain, RF_Ytrain_predict))
print('\nClassification Report:-\n',metrics.classification_report(Ytrain, RF_Ytrain_predict))

## Performance Matrix on test data set
RF_Ytest_predict = RF.predict(Xtest)
RF_test_score = RF.score(Xtest, Ytest)
print('Test Data Accuracy:-',RF_test_score)
print('\nConfusion Matrix:-\n',metrics.confusion_matrix(Ytest, RF_Ytest_predict))
print('\nClassification Report:-\n',metrics.classification_report(Ytest, RF_Ytest_predict))
```

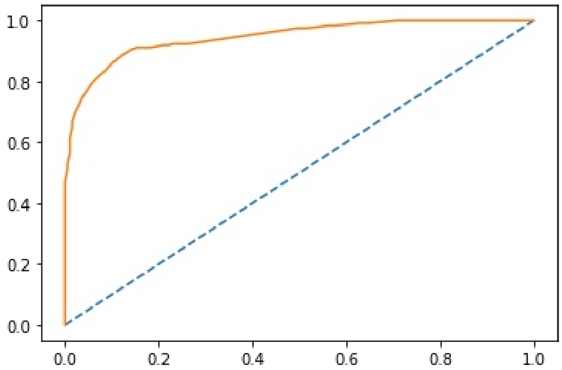
```
Train Data Accuracy:- 0.8860353130016051

Confusion Matrix:-
 [[360  27]
 [ 44 192]]

Classification Report:-
               precision    recall  f1-score   support

           0       0.89      0.93      0.91       387
           1       0.88      0.81      0.84       236

    accuracy                           0.89       623
   macro avg       0.88      0.87      0.88       623
```

```
weighted avg       0.89      0.89      0.89       623

Test Data Accuracy:- 0.8171641791044776

Confusion Matrix:-
 [[147  15]
 [ 34  72]]

Classification Report:-
               precision    recall  f1-score   support

           0       0.81      0.91      0.86       162
           1       0.83      0.68      0.75       106

    accuracy                           0.82       268
   macro avg       0.82      0.79      0.80       268
weighted avg       0.82      0.82      0.81       268
```

In [61]:

```
#AUC and ROC for train data wrt RandomForest
probs = RF.predict_proba(Xtrain)
probs = probs[:, 1]
RF_train_auc = roc_auc_score(Ytrain, probs)
print('AUC: %.3f' % RF_train_auc)
train_fpr, train_tpr, train_thresholds = roc_curve(Ytrain, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(train_fpr, train_tpr);
```

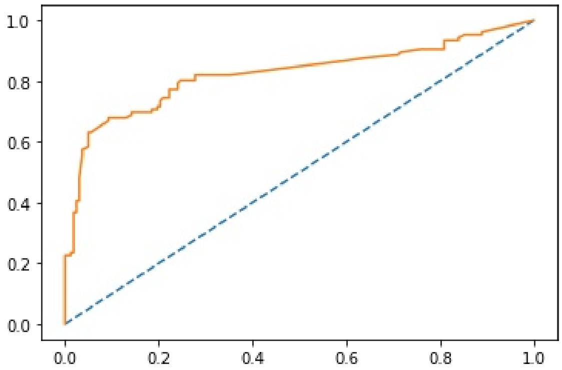
AUC: 0.945



In [62]:

```
#AUC and ROC for test data wrt Random Forest
probs = RF.predict_proba(Xtest)
probs = probs[:, 1]
RF_test_auc = roc_auc_score(Ytest, probs)
print('AUC: %.3f' % RF_test_auc)
test_fpr, test_tpr, test_thresholds = roc_curve(Ytest, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(test_fpr, test_tpr);
```

AUC: 0.823



In [63]:

```
params = {'n_estimators': [15,25,30],
              'criterion': ['gini','entropy'],
              'max_depth': [3,7,None],
```

```
               'min_samples_split': [5,10,],
               'min_samples_leaf': [1,3,5],
               'max_features': [2,'auto'],
              }

gridRF = GridSearchCV(RF, param_grid=params, n_jobs=1)
gridRF.fit(Xtrain,Ytrain)
```

Out[63]:
```
GridSearchCV(estimator=RandomForestClassifier(random_state=1), n_jobs=1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [3, 7, None], 'max_features': [2, 'auto'],
                         'min_samples_leaf': [1, 3, 5],
                         'min_samples_split': [5, 10],
                         'n_estimators': [15, 25, 30]})
```

In [64]:
```
print(gridRF.best_params_,'\n')
print(gridRF.best_estimator_)
```

```
{'criterion': 'gini', 'max_depth': None, 'max_features': 2, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_est
imators': 30}

RandomForestClassifier(max_features=2, min_samples_split=5, n_estimators=30,
                       random_state=1)
```

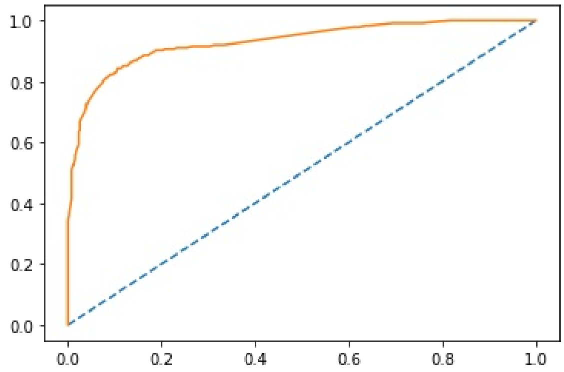
In [65]:
```
bestmodelRF = gridRF.best_estimator_

RFM_Ytrain_predict = bestmodelRF.predict(Xtrain)
RFM_train_score = bestmodelRF.score(Xtrain, Ytrain)
print('Train Set Accuracy:',RFM_train_score)
print('Confusion Matrix:-\n',confusion_matrix(Ytrain, RFM_Ytrain_predict))
print('\nClassification Report:-\n',classification_report(Ytrain, RFM_Ytrain_predict))

RFM_Ytest_predict = bestmodelRF.predict(Xtest)
RFM_test_score = bestmodelRF.score(Xtest, Ytest)
print('Test Set Accuracy:',RFM_test_score)
print('Confusion Matrix:-\n',confusion_matrix(Ytest, RFM_Ytest_predict))
print('\nClassification Report:-\n',classification_report(Ytest, RFM_Ytest_predict))
```

```
Train Set Accuracy: 0.8731942215088283
Confusion Matrix:-
 [[348  39]
 [ 40 196]]

Classification Report:-
               precision    recall  f1-score   support

           0       0.90      0.90      0.90       387
           1       0.83      0.83      0.83       236

    accuracy                           0.87       623
   macro avg       0.87      0.86      0.87       623
weighted avg       0.87      0.87      0.87       623

Test Set Accuracy: 0.8022388059701493
Confusion Matrix:-
 [[143  19]
 [ 34  72]]

Classification Report:-
               precision    recall  f1-score   support

           0       0.81      0.88      0.84       162
           1       0.79      0.68      0.73       106

    accuracy                           0.80       268
   macro avg       0.80      0.78      0.79       268
weighted avg       0.80      0.80      0.80       268
```

In [66]:
```
#AUC and ROC for train data wrt RF-M
probs = bestmodelRF.predict_proba(Xtrain)
probs = probs[:, 1]
RFM_train_auc = roc_auc_score(Ytrain, probs)
print('AUC: %.3f' % RFM_train_auc)
train_fpr, train_tpr, train_thresholds = roc_curve(Ytrain, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
```

```
plt.plot(train_fpr, train_tpr);
```

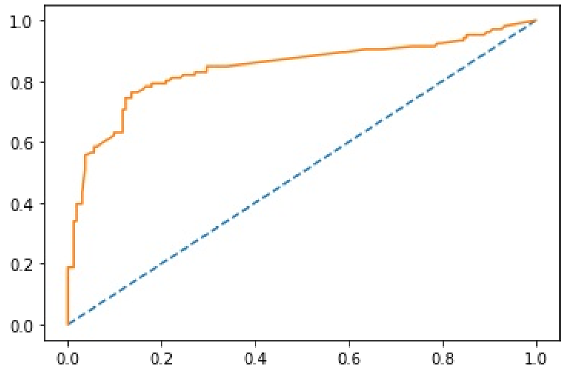
AUC: 0.930



In [67]:

```
#AUC and ROC for test data wrt RF-M
probs = bestmodelRF.predict_proba(Xtest)
probs = probs[:, 1]
RFM_test_auc = roc_auc_score(Ytest, probs)
print('AUC: %.3f' % RFM_test_auc)
test_fpr, test_tpr, test_thresholds = roc_curve(Ytest, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(test_fpr, test_tpr);
```

AUC: 0.842



So, we can predict around 88% values in train set and 82% in test set using Random Forest model.