In [1]:

```python
import numpy as np
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as plt
import sklearn
from sklearn.tree import export_graphviz
#from sklearn import metrics
#from sklearn.metrics import classification_report

Url = "https://raw.githubusercontent.com/BigDataGal/Python-for-Data-Science/master/titanic-train.csv"
```

In [2]:

```python
from plotly.offline import iplot
import plotly as ply
import plotly.tools as tls
import cufflinks as cf
cf.go_offline()
```

In [3]:

```python
titanic = pd.read_csv(Url)
titanic.head()
```

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

You use only Pclass, Sex, Age, SibSp (Siblings aboard), Parch (Parents/children aboard), and Fare to predict whether a passenger survived

In [4]:

```python
titanic.columns
```

Out[4]:

```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

Droping Columns Pclass, Sex, Age, SibSp (Siblings aboard), Parch (Parents/children aboard), and Fare

In [5]:

```python
titanic.drop(columns = ['PassengerId', 'Name',
       'Ticket', 'Cabin', 'Embarked'],axis=1,inplace=True)

# Changing Name of the Columns 'Parch' to 'Parents/children_aboard','SibSp' to 'Siblings aboard'
titanic.rename(columns={'Parch':'Parn/chldrnAboard','SibSp':'SiblingsAboard'},inplace=True)
```

In [6]:

```python
titanic.head()
```

Out[6]:

	Survived	Pclass	Sex	Age	SiblingsAboard	Parn/chldrnAboard	Fare
0	0	3	male	22.0	1	0	7.2500
1	1	1	female	38.0	1	0	71.2833
2	1	3	female	26.0	0	0	7.9250
3	1	1	female	35.0	1	0	53.1000
4	0	3	male	35.0	0	0	8.0500

Description of titanic Data.

To Check Whether there are null or insignificant values are there.

In [7]:

```python
titanic.describe()
```

Out[7]:

	Survived	Pclass	Age	SiblingsAboard	Parn/chldrnAboard	Fare
count	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

As we can see that Age having only 714 values while other features have value.

Checking the info of all the columns to see the data type and Null columns.

In [8]:

```python
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 7 columns):
Survived             891 non-null int64
Pclass               891 non-null int64
Sex                  891 non-null object
Age                  714 non-null float64
SiblingsAboard       891 non-null int64
Parn/chldrnAboard    891 non-null int64
Fare                 891 non-null float64
dtypes: float64(2), int64(4), object(1)
memory usage: 48.9+ KB
```

In [9]:

```python
titanic.isnull().sum()
```

Out[9]:

```
Survived               0
Pclass                 0
Sex                    0
Age                  177
SiblingsAboard         0
Parn/chldrnAboard      0
Fare                   0
dtype: int64
```

We can see that Age Column have 177 Null values.

Removing Null Values

In [10]:
```python
# Dropna removes the Null values from the Data Frame.
titanic.dropna(inplace= True)
titanic.isnull().info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 714 entries, 0 to 890
Data columns (total 7 columns):
Survived            714 non-null bool
Pclass              714 non-null bool
Sex                 714 non-null bool
Age                 714 non-null bool
SiblingsAboard      714 non-null bool
Parn/chldrnAboard   714 non-null bool
Fare                714 non-null bool
dtypes: bool(7)
memory usage: 10.5 KB
```

Changing Male = 1 and Female = 0 in Sex Column of DataFrame.

In [11]:
```python
# Function to change value
def bool_att(value):
    """
    :param: value: Take a single value as input.
    :return: 1 in case of male, 0 in case of Female.
    """

    if value.lower() == 'male':
        return 1
    elif value.lower() == 'female':
        return 0

titanic['Sex'] = titanic['Sex'].apply(bool_att)
```

In [12]:
```python
titanic
```

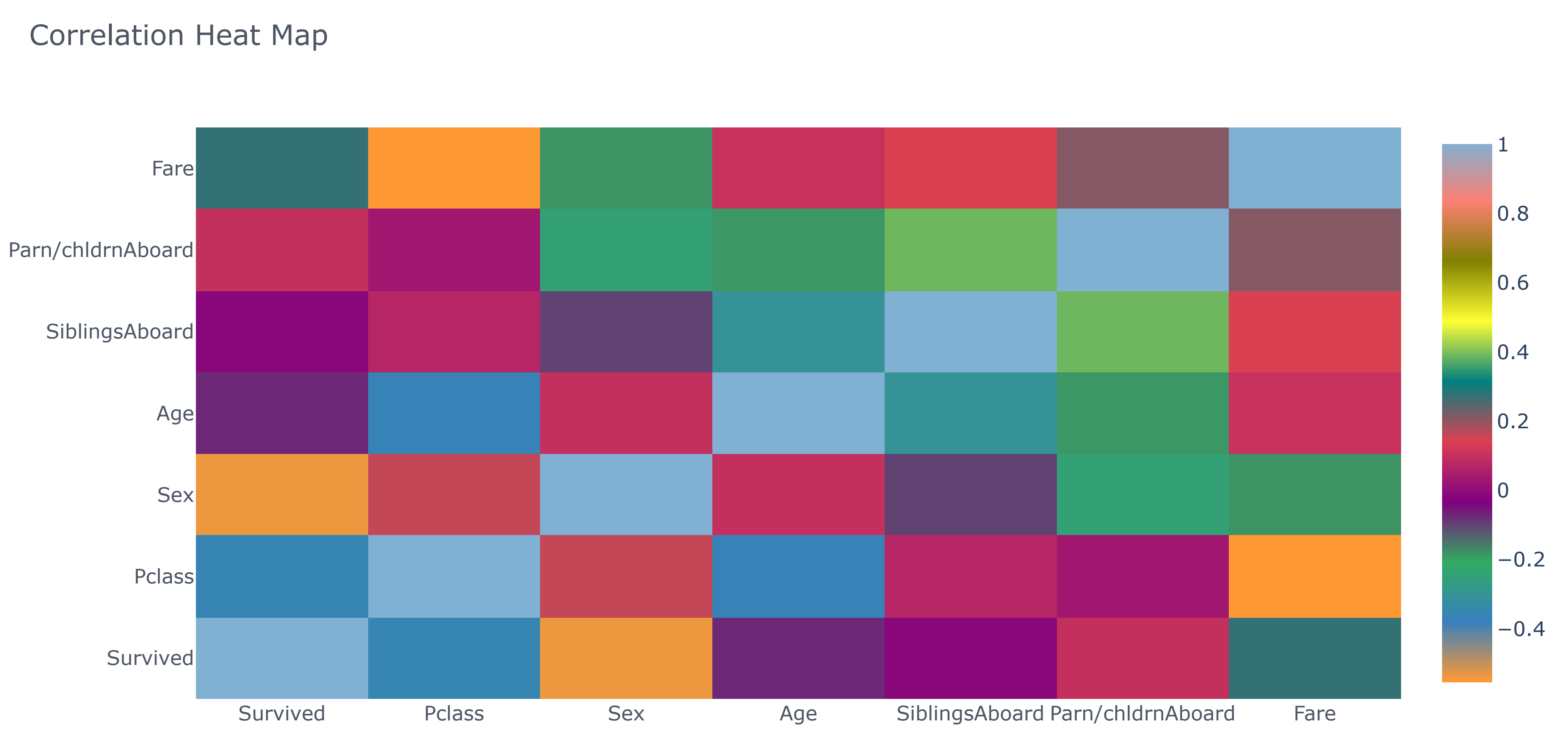
Out[12]:

	Survived	Pclass	Sex	Age	SiblingsAboard	Parn/chldrnAboard	Fare
0	0	3	1	22.0	1	0	7.2500
1	1	1	0	38.0	1	0	71.2833
2	1	3	0	26.0	0	0	7.9250
3	1	1	0	35.0	1	0	53.1000
4	0	3	1	35.0	0	0	8.0500
...	...	...	...	...	...	...	...
885	0	3	0	39.0	0	5	29.1250
886	0	2	1	27.0	0	0	13.0000
887	1	1	0	19.0	0	0	30.0000
889	1	1	1	26.0	0	0	30.0000
890	0	3	1	32.0	0	0	7.7500

714 rows × 7 columns

Checking Correlation

In [13]:
```python
correlation_df = titanic.corr()
correlation_df.iplot(kind='heatmap',title='Correlation Heat Map') #
print("Correlation Values:\n", correlation_df)
```



Export to plot.ly »

```
Correlation Values:
                    Survived    Pclass       Sex       Age  SiblingsAboard  \
Survived           1.000000 -0.359653 -0.538826 -0.077221       -0.017358
Pclass            -0.359653  1.000000  0.155460 -0.369226        0.067247
Sex               -0.538826  0.155460  1.000000  0.093254       -0.103950
Age               -0.077221 -0.369226  0.093254  1.000000       -0.308247
SiblingsAboard    -0.017358  0.067247 -0.103950 -0.308247        1.000000
Parn/chldrnAboard  0.093317  0.025683 -0.246972 -0.189119        0.383820
Fare               0.268189 -0.554182 -0.184994  0.096067        0.138329

                   Parn/chldrnAboard      Fare
Survived                    0.093317  0.268189
Pclass                      0.025683 -0.554182
Sex                        -0.246972 -0.184994
Age                        -0.189119  0.096067
SiblingsAboard              0.383820  0.138329
Parn/chldrnAboard           1.000000  0.205119
Fare                        0.205119  1.000000
```

As from the above graph it is clear that there is no such high correlation values. So we can move forward with this DataFrame.

Spliting the Data Frame into Dependent and Independent Features

In [14]:

```
def tree_img(name,classifier):
    export_graphviz(
     classifier,
     out_file=name+".dot",
     feature_names=X.columns,
     class_names=y.name,
     rounded=True,
     filled=True
     )



```

In [15]:

```
X = titanic.drop('Survived',axis=1)
y = titanic.Survived
```

In [16]:

```
from sklearn.tree import DecisionTreeClassifier
decision_tree1 = DecisionTreeClassifier()
decision_tree1.fit(X,y)
```

Out[16]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

In [17]:

```
tree_img("decision_tree_1",decision_tree1)
! dot -Tpng decision_tree_1.dot -o decision_tree_1.png
```

Model Perfromance on Whole Data

In [18]:

```

print("Model Accuracy: ",decision_tree1.score(X,y))
```

```
Model Accuracy:  0.9859943977591037
```

Performing Hold Out Validation

In [19]:

```
from sklearn.model_selection import train_test_split
```

In [20]:

```
decision_tree = DecisionTreeClassifier()
# Spliting the fetaures into traning and Testing Part
x_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.20,random_state=355)
```

In [22]:

```
decision_tree.fit(x_train,y_train)
tree_img("decision_tree_2",decision_tree)
! dot -Tpng decision_tree_2.dot -o decision_tree_2.png

```

Testing Accuracy

In [23]:

```
decision_tree.score(x_train,y_train)
```

Out[23]:

```
0.9912434325744308
```

Traning Accuracy

In [24]:

```
decision_tree.score(x_test,y_test)
```

Out[24]:

```
0.7692307692307693
```

Prediction

In [25]:

```
decision_tree.predict([[3,1,23.00,0,0,7.8542]])[0]
```

Out[25]:

```
0
```

Performing Cross Validation

In [26]:

```
from sklearn.model_selection import cross_val_score,KFold
```

In [27]:

```
c_v = KFold(n_splits=5)
c_v_score = cross_val_score(decision_tree,X,y,cv=c_v)
print("Average Cross Validation Scores on 5 Splits: ",c_v_score.sum()/5)
```

```
Average Cross Validation Scores on 5 Splits:  0.7759479956663056
```

In [28]:

```
c_v = KFold(n_splits=10)
c_v_score = cross_val_score(decision_tree,X,y,cv=c_v)
print("Average Cross Validation Scores on 10 Splits: \n",c_v_score.sum()/10)
```

```
Average Cross Validation Scores on 10 Splits:
 0.7634194053208138
```

In [29]:

```
c_v = KFold(n_splits=50)
c_v_score = cross_val_score(decision_tree,X,y,cv=c_v)
print("Average Cross Validation Scores on 50 Splits: \n",c_v_score.sum()/50)
```

```
Average Cross Validation Scores on 50 Splits:
 0.7679047619047619
```

In [30]:

```
c_v = KFold(n_splits=100)
c_v_score = cross_val_score(decision_tree,X,y,cv=c_v)
print("Average Cross Validation Scores on 100 Splits: \n",c_v_score.sum()/100)
```

```
Average Cross Validation Scores on 100 Splits:
 0.7621428571428571
```

In [31]:

```
c_v = KFold(n_splits=500)
c_v_score = cross_val_score(decision_tree,X,y,cv=c_v)
print("Average Cross Validation Scores on 500 Splits: \n",c_v_score.sum()/500)
```

```
Average Cross Validation Scores on 500 Splits:
 0.781
```

From above score on Model traning on Whole data to Hold Out Validation to Cross Validation we can see different Scores. If we are going to consider accuracy as an approach to select the Model then Model Tranined on Whole data is more Accurate, but we cannot use this in production as the model tree is made on that data. So it can predict the data present in that More accurately than in Hold Out and Cross Validation Approaches.

In Hold Out Validation the Traning Accuracy is 99 % and Testing Accuracy 77 %. It a clear Indiaction of Overfitting Model having High Variance.

So we are Using Coss Validation to check and get Model on different data test set to get an average score for the Model Accuracy.

From the above score it is clear that our model accuracy is going to be around 77% .

Now how to get the parameters on which we can get this Model.

Performing Hyperparameter tunning.

Decision Tree Classifier Function Contains Different Prameters(Listed Below). Hyperparamter Tunning is the way to get a combination of paramter to acchieve the desired model having a stable accuracy.

DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort='deprecated', ccp_alpha=0.0,)

In [32]:
```
Decision_tree_obj = DecisionTreeClassifier()
from sklearn.model_selection import GridSearchCV
grid_parameters = {
    'criterion':['ginni','entropy'],
    'splitter': ['best','random'],
    'max_depth' : range(2,30,2),
    'min_samples_split': range(2,30,2),
    'min_samples_leaf': range(2,30,2)
}

grid_search = GridSearchCV(estimator=Decision_tree_obj,
                          param_grid=grid_parameters,
                          cv=5,
                          n_jobs=-1)

```

In [38]:
```
grid_search.fit(x_train,y_train)
```

Out[38]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort='deprecated',
                                              random_state=None,
                                              splitter='best'),
             iid='deprecated', n_jobs=-1,
             param_grid={'criterion': ['ginni', 'entropy'],
                         'max_depth': range(2, 30, 2),
                         'min_samples_leaf': range(2, 30, 2),
                         'min_samples_split': range(2, 30, 2),
                         'splitter': ['best', 'random']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In [39]:
```
best_parameters = grid_search.best_params_
print("Best Parameters for the Model Making\n\n",best_parameters)
```

```
Best Parameters for the Model Making

 {'criterion': 'entropy', 'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 14, 'splitter': 'best'}
```

In [ ]:
```
best_score = grid_search.best_score_
print("Best Score We got From Hyper Parameter Tunning is : ",best_score)
```

Using the Best Parameters for Model Making

In [40]:
```
model = DecisionTreeClassifier(criterion = 'entropy', max_depth = 10, min_samples_leaf = 2, min_samples_split = 14, splitter = 'best')
model.fit(x_train,y_train)
```

Out[40]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=10, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=2, min_samples_split=14,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

In [41]:
```
tree_img("decision_tree_model",model)
! dot -Tpng decision_tree_model.dot -o decision_tree_model.png
```

Testing Accuracy

In [42]:
```
model.score(x_train,y_train)
```

Out[42]:
```
0.8896672504378283
```

Traning Accuracy

In [43]:
```
model.score(x_test,y_test)
```

Out[43]:
```
0.7972027972027972
```

Saving the model file

In [44]:
```
import pickle
```

In [45]:
```
with open('decision_tree_model.pickle','wb') as file_open:
    pickle.dump(model,file_open)
```

In [46]:
```
with open('decision_tree_model.pickle','rb') as file_read:
    model = pickle.load(file_read)
```

Since it's a classification task. We cannot only measure accuracy in terms of Validation Accuracy alone. In classification we need to know how he model perfroms on different criteria, whether it is able to remember it's value, whether it is predicting the true to be true and false to be false.

Precision

Recall

Specificity

Specificity
F1 Score

AUC & ROC

In [47]:
```
from sklearn.metrics import confusion_matrix,f1_score, recall_score, precision_score,accuracy_score,roc_auc_score,roc_curve
from sklearn.model_selection import cross_val_predict
```

Confusion Matrix

--	Actual	Values
Predicted	True Positive	False Positive
Values	False Negative	True Negative

Model 1 (Hyperparameter tunning)

In [48]:
```
model_predict  = cross_val_predict(model,x_train,y_train,cv=3)
confusion_matrix(y_train,model_predict)
```

Out[48]:
```
array([[285,  59],
       [ 57, 170]], dtype=int64)
```

From the above confussion Matrix we get to know that

- 285 values were actually Positive and Classified as positive by the Model.
- 59 Values were actually Negative that were classified as positive by the Model.
- 57 values were actually positive but classified as Negative by the Model.
- 170 values were actually negative and were classified as negative by the Model.

In [49]:
```
print(f"Recall Score: {round(recall_score(y_train,model_predict,2)*100)}%")
print("Model is able to recall 73% of the Data on which it is trained.")
```

```
Recall Score: 75.0%
Model is able to recall 73% of the Data on which it is trained.
```

In [50]:
```
print(f"Precision Score: {round(precision_score(y_train,model_predict,2)*100)}%")
print("Model is able to accuratly classify 74% of the Data on which it is trained.")
```

```
Precision Score: 74.0%
Model is able to accuratly classify 74% of the Data on which it is trained.
```

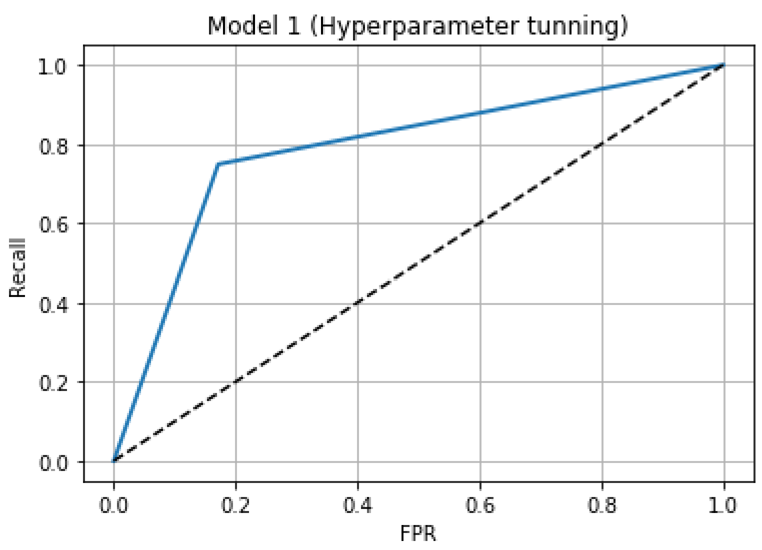
In [51]:
```
print(f"f1 Score: {round(f1_score(y_train,model_predict,2)*100)}%")

```

```
f1 Score: 75.0%
```

In [52]:
```
print(f"AUC ROC Score: {round(roc_auc_score(y_train,model_predict),2)*100}")

```

```
AUC ROC Score: 79.0
```

In [53]:
```
fpr, tpr, thresholds = roc_curve(y_train, model_predict)  #fpr: false positive rate , tpr: True Positive rate(Recall)


def plot_roc_curve(fpr, tpr,model,label=None):
    plt.title(model)
    plt.plot(fpr, tpr, linewidth=2, label=label)
    plt.plot([0, 1], [0, 1], 'k--') # dashed diagonal
    plt.ylabel('Recall')
    plt.xlabel('FPR')
    plt.grid()
    plt.show()

plot_roc_curve(fpr, tpr,model="Model 1 (Hyperparameter tunning)")
print("More the blue line away from the dotted line better the Model.")
```




```
More the blue line away from the dotted line better the Model.
```

Model 2 (hold out validation)

In [54]:
```
model_predict_decision_tree  = cross_val_predict(decision_tree,x_train,y_train,cv=3)
confusion_matrix(y_train,model_predict_decision_tree)

```

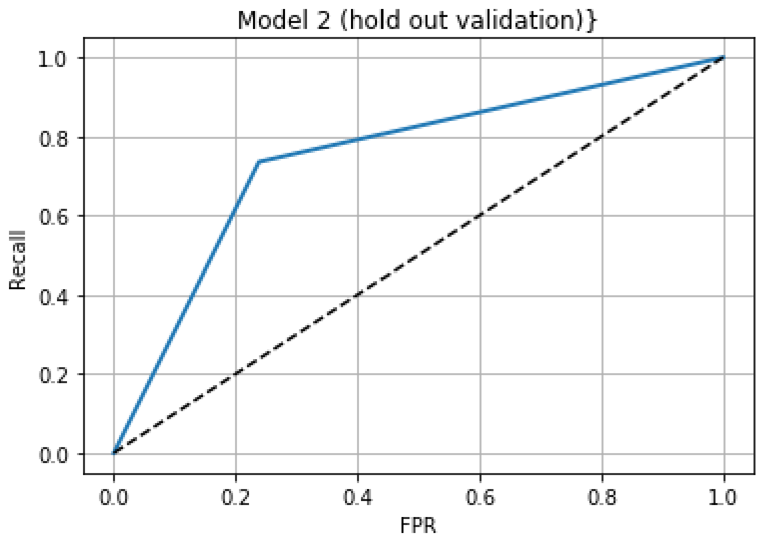
Out[54]:
```
array([[262,  82],
       [ 60, 167]], dtype=int64)
```

From the above confussion Matrix we get to know that

- 262 values were actually Positive and Classified as positive by the Model.
- 82 Values were actually Negative that were classified as positive by the Model.
- 60 values were actually positive but classified as Negative by the Model.
- 167 values were actually negative and were classified as negative by the Model.

In [55]:

```python
fpr, tpr, thresholds = roc_curve(y_train, model_predict_decision_tree) #fpr: false positive rate , tpr: True Positive rate(Recall)


def plot_roc_curve(fpr, tpr,model,label=None):
    plt.title(model)
    plt.plot(fpr, tpr, linewidth=2, label=label)
    plt.plot([0, 1], [0, 1], 'k--') # dashed diagonal
    plt.ylabel('Recall')
    plt.xlabel('FPR')
    plt.grid()
    plt.show()

plot_roc_curve(fpr, tpr,model="Model 2 (hold out validation)}")
```



In [56]:

```python
print(f"f1 Score: {round(f1_score(y_train,model_predict_decision_tree,2)*100)}%")

```

```
f1 Score: 70.0%
```

-- END --

In [ ]:

```python

```