Titanic Survival Analysis and Prediction

Author

DINESH S
my linkedin profile | github link | Data Play

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 ask you to build 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).

Overview of Data

The data has been split into two groups:

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

The training set should be used to build your machine learning models.

The test set should be used to see how well your model performs on unseen data.

use the model you trained to predict whether or not they survived the sinking of the Titanic.

Data set Link: https://www.kaggle.com/c/titanic/data

importing the neccessary libraries

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

import warnings
warnings.filterwarnings('ignore')
```

In [ ]:
```
#Default theme
sns.set_theme(context='notebook',
              style='whitegrid',
              palette='rainbow',
              font='Lucida Calligraphy',
              font_scale=1.5,
              rc=None)
import matplotlib
matplotlib.rcParams['figure.figsize'] = [8, 8]
matplotlib.rcParams.update({'font.size': 15})
matplotlib.rcParams['font.family'] = 'sans-serif'
```

Analyzing the Data set

In [ ]:
```
titanic = pd.read_csv('/content/train.csv')
titanic.head().style.set_properties(
    **{
        'background-color': 'LightBlue',
        'color': 'Black',
        'border-color': 'darkblack'
    })
```

Out[ ]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch
0	1	0	3	Braund, Mr. Owen Harris	male	22.000000	1	0
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Thayer)	female	38.000000	1	0
2	3	1	3	Heikkinen, Miss. Laina	female	26.000000	0	0
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.000000	1	0
4	5	0	3	Allen, Mr. William Henry	male	35.000000	0	0

In [ ]:
```
print('Shape of Titanic data set is :',titanic.shape)
print('Size of Titanic data set is  :',titanic.size)
```

```
Shape of Titanic data set is : (891, 12)
Size of Titanic data set is  : 10692
```

In [ ]:
```
titanic.describe()
```

Out[ ]:

	PassengerId	Survived	Pclass	Age	SibSp	Parc
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.00000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.38159
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.80605
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.00000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.00000
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.00000
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.00000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.00000

Variable Notes

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

1. 1st = Upper
2. 2nd = Middle
3. 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

In [ ]:

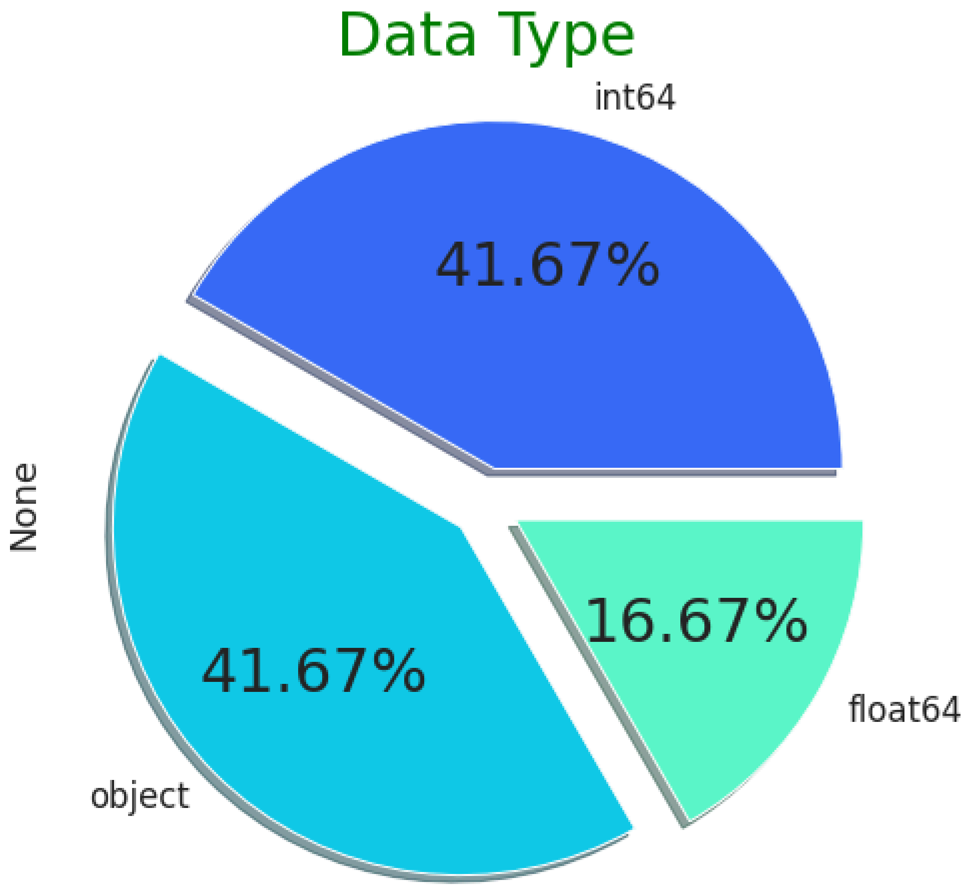
```
titanic.info()
```

```
<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
```

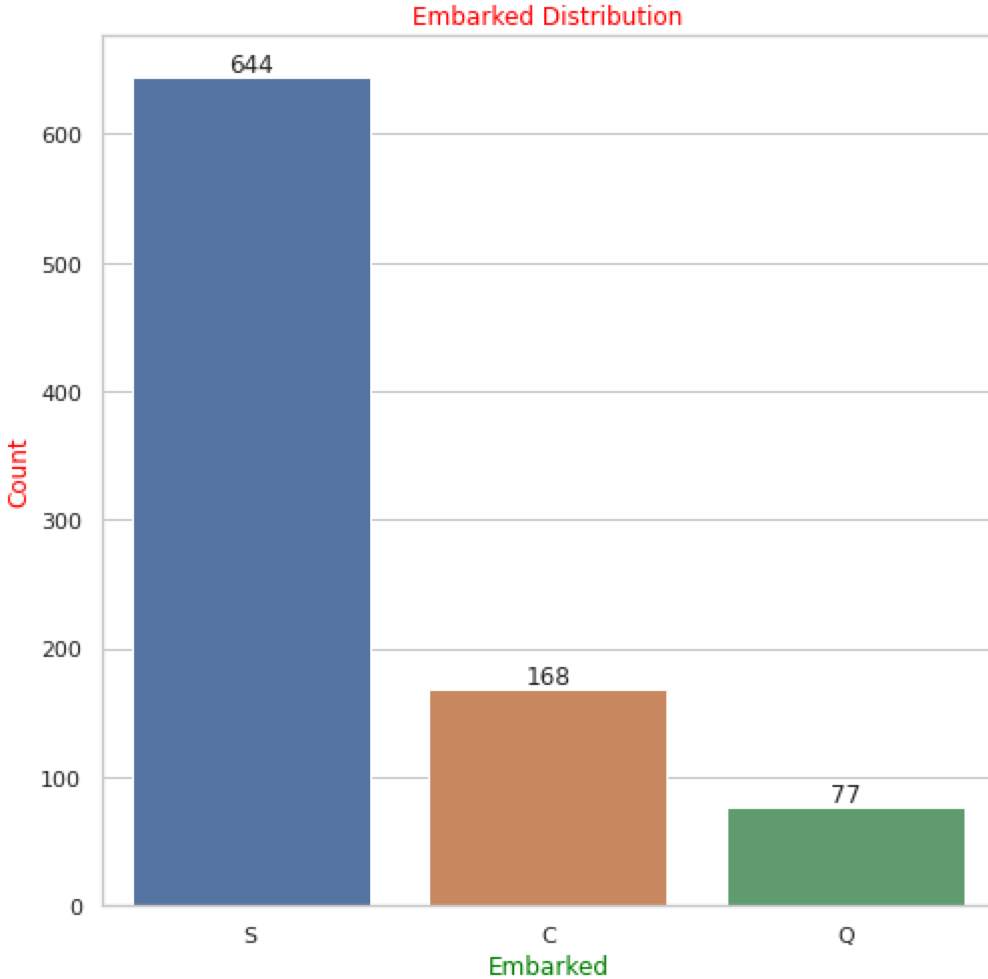
Data Visualization

In [ ]:

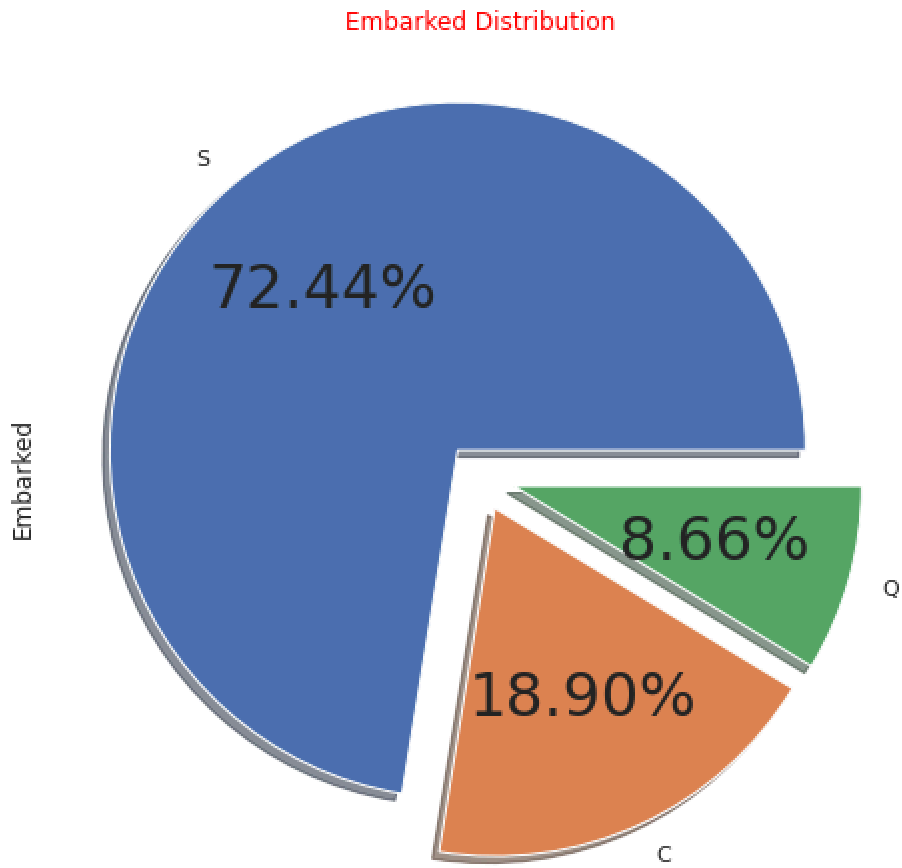
```
matplotlib.rcParams.update({'font.size': 30})

titanic.dtypes.value_counts().plot.pie(explode=[0.1, 0.1, 0.1],
                                       autopct='%1.2f%%',
                                       shadow=True)
plt.title('Data Type',
          color='Green',
          loc='center',
          font='Lucida Calligraphy');
```



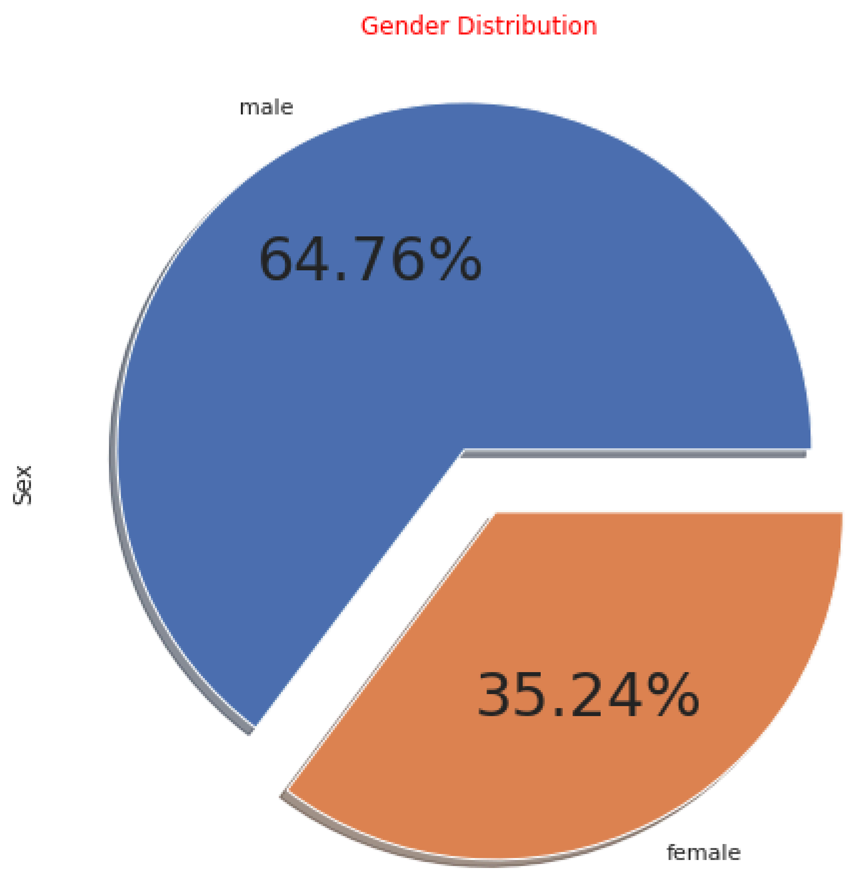
In [ ]:
```
ax = sns.set(style="whitegrid")
ax = sns.countplot(data=titanic,x='Embarked');
ax.bar_label(ax.containers[0])

plt.title('Embarked Distribution',color='Red',loc='center',font='Lucida Call
plt.xlabel('Embarked',color='Green',loc='center',font='Lucida Calligraphy')
plt.ylabel('Count',color='Red',loc='center',font='Lucida Calligraphy');
```



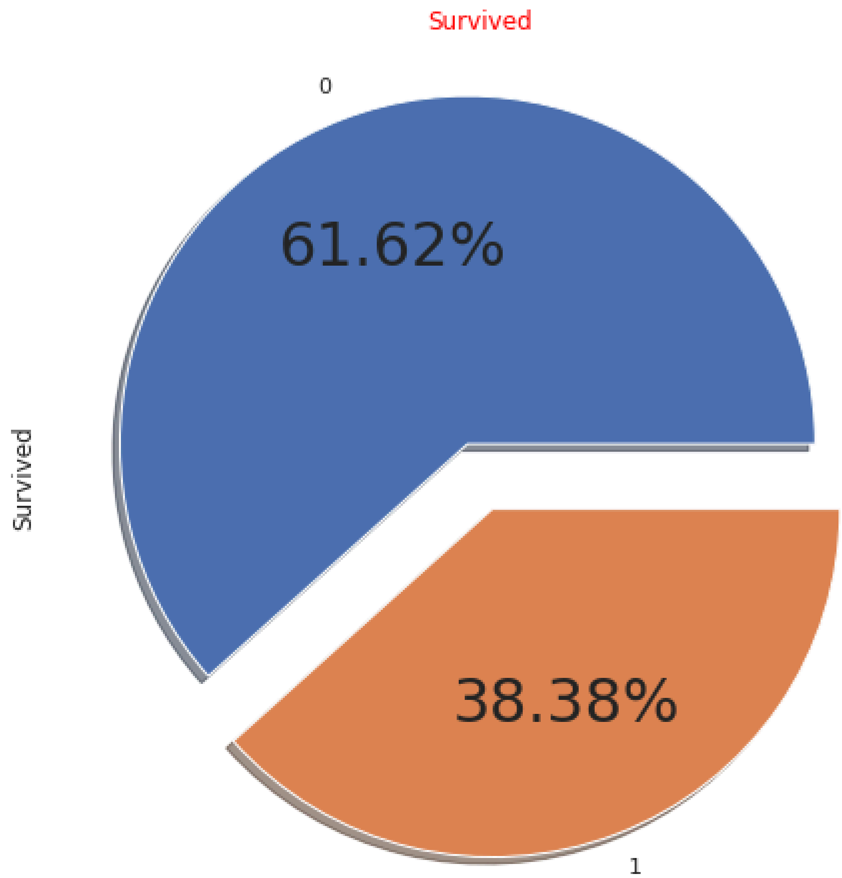
In [ ]:
```
matplotlib.rcParams.update({'font.size': 30})
titanic['Embarked'].value_counts().plot.pie(explode=[0.1, 0.1, 0.1],
                                             autopct='%1.2f%%',
                                             shadow=True)
plt.title('Embarked Distribution',color='Red',loc='center');
```



In [ ]:
```
matplotlib.rcParams.update({'font.size': 30})
titanic['Sex'].value_counts().plot.pie(explode=[0.1, 0.1],
                                       autopct='%1.2f%%',
                                       shadow=True)
plt.title('Gender Distribution',color='Red',loc='center');
```



In [ ]:
```
matplotlib.rcParams.update({'font.size': 30})
titanic['Survived'].value_counts().plot.pie(explode=[0.1, 0.1],
                                            autopct='%1.2f%%',
                                            shadow=True)
plt.title('Survived',color='Red',loc='center');
```



In [ ]:

```
titanic.corr().style.background_gradient(cmap='coolwarm').set_precision(3)
```

Out[ ]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
PassengerId	1.000	-0.005	-0.035	0.037	-0.058	-0.002	0.013
Survived	-0.005	1.000	-0.338	-0.077	-0.035	0.082	0.257
Pclass	-0.035	-0.338	1.000	-0.369	0.083	0.018	-0.549
Age	0.037	-0.077	-0.369	1.000	-0.308	-0.189	0.096
SibSp	-0.058	-0.035	0.083	-0.308	1.000	0.415	0.160
Parch	-0.002	0.082	0.018	-0.189	0.415	1.000	0.216
Fare	0.013	0.257	-0.549	0.096	0.160	0.216	1.000

In [ ]:

```
corr=titanic.corr()#["Survived"]
plt.figure(figsize=(20, 15))
sns.heatmap(corr, vmax=.8, linewidths=0.01, square=True,annot=True,cmap='YlG
plt.title('Correlation between features')
plt.show()
```

In [ ]:

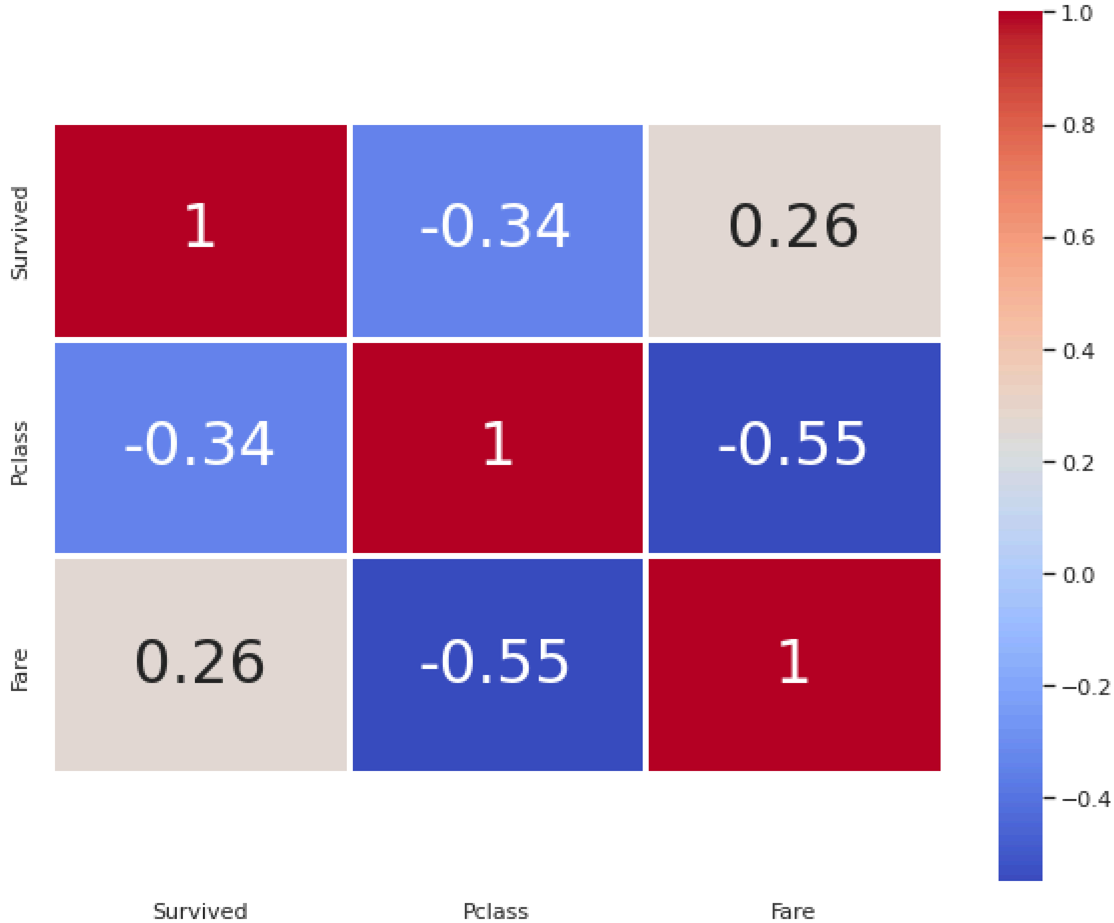
```
# correlation heatmap of higly correlated features with SalePrice

hig_corr = titanic.corr()
hig_corr_features = hig_corr.index[abs(hig_corr["Fare"]) >= 0.25]
hig_corr_features
```

Out[ ]:

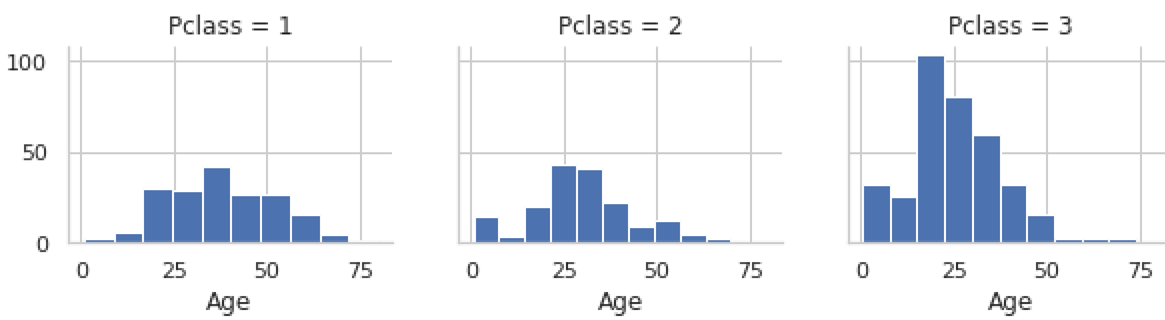
```
Index(['Survived', 'Pclass', 'Fare'], dtype='object')
```

In [ ]:

```
plt.figure(figsize=(10,8))
ax = sns.heatmap(titanic[hig_corr_features].corr(), cmap = "coolwarm", annot
# to fix the bug "first and last row cut in half of heatmap plot"
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
plt.show()
```



In [ ]:
```
g = sns.FacetGrid(titanic, col="Pclass")
g = g.map(plt.hist, "Age")
```



Methods to find Missing Values

In [ ]:
```
def missing_value (df):
    missing_Number = df.isnull().sum().sort_values(ascending=False)[df.isnul
    missing_percent=round((df.isnull().sum()/df.isnull().count())*100,2)[rou
    missing = pd.concat([missing_Number,missing_percent],axis=1,keys=['Missi
    return missing
```

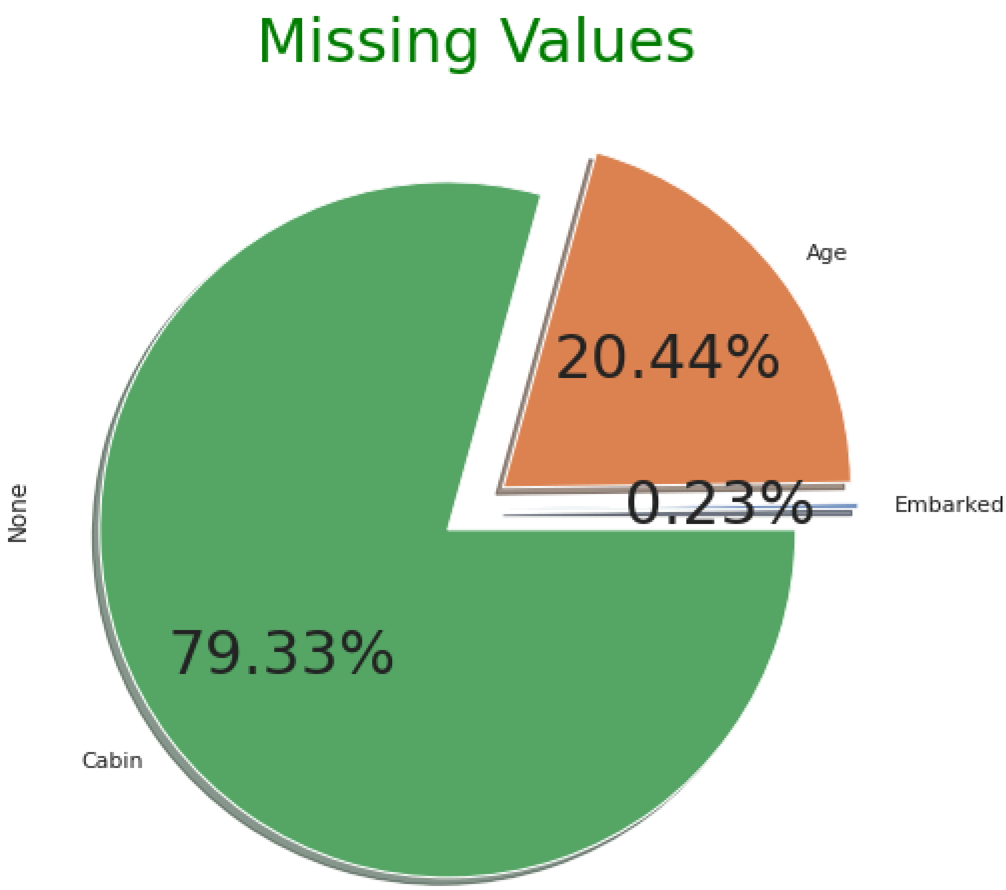
In [ ]:
```
missing_value(titanic).style.background_gradient(cmap='coolwarm').set_precis
```

Out[ ]:

	Missing Number	Missing Percentage
Cabin	687	77.10
Age	177	19.87
Embarked	2	0.22

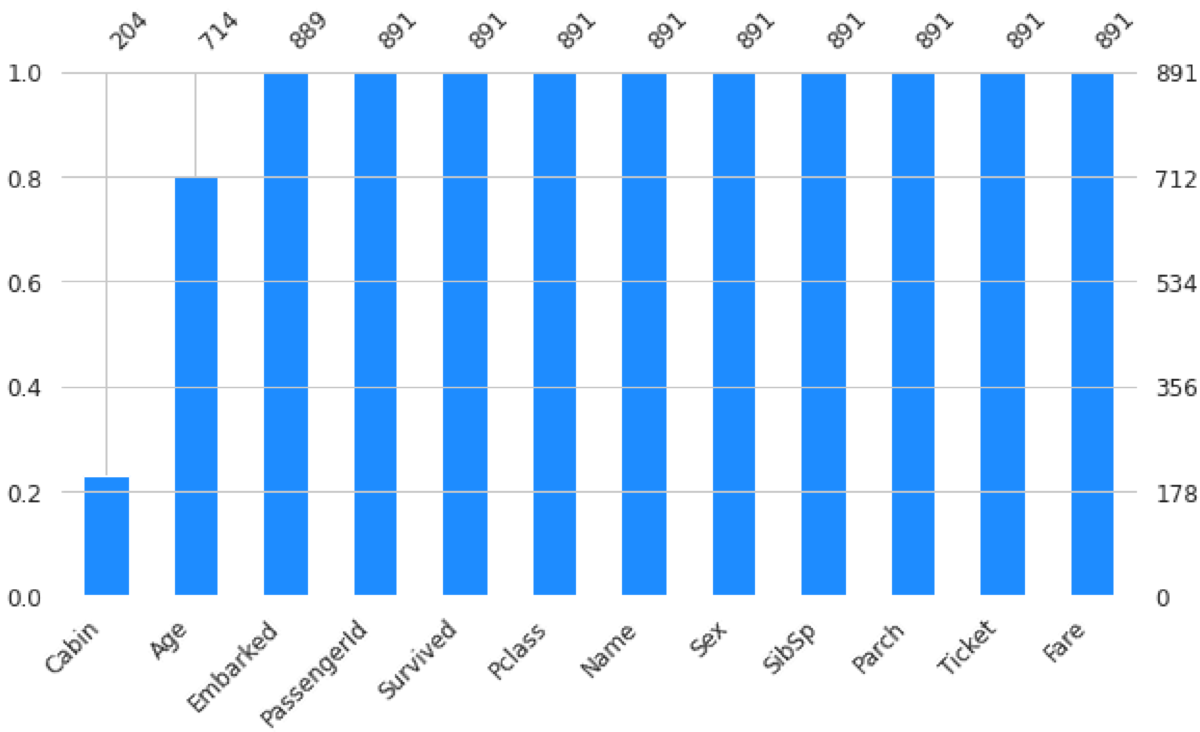
In [ ]:

```
missing_values = titanic.isnull().sum()
missing_values = missing_values[missing_values > 0]
missing_values.sort_values(inplace=True)
missing_values.plot.pie(explode=[0.1, 0.1, 0.1],
                        autopct='%1.2f%%',
                        shadow=True)

plt.title('Missing Values',
          color='Green',
          loc='center',
          font='Lucida Calligraphy');
```

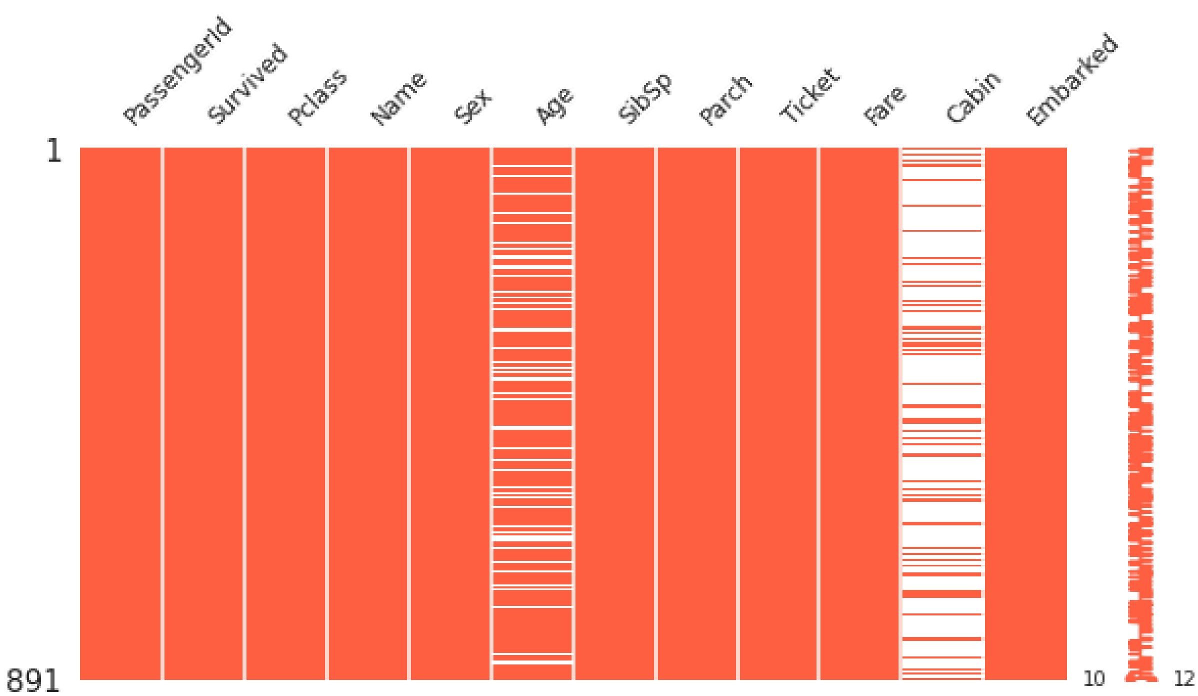


In [ ]:

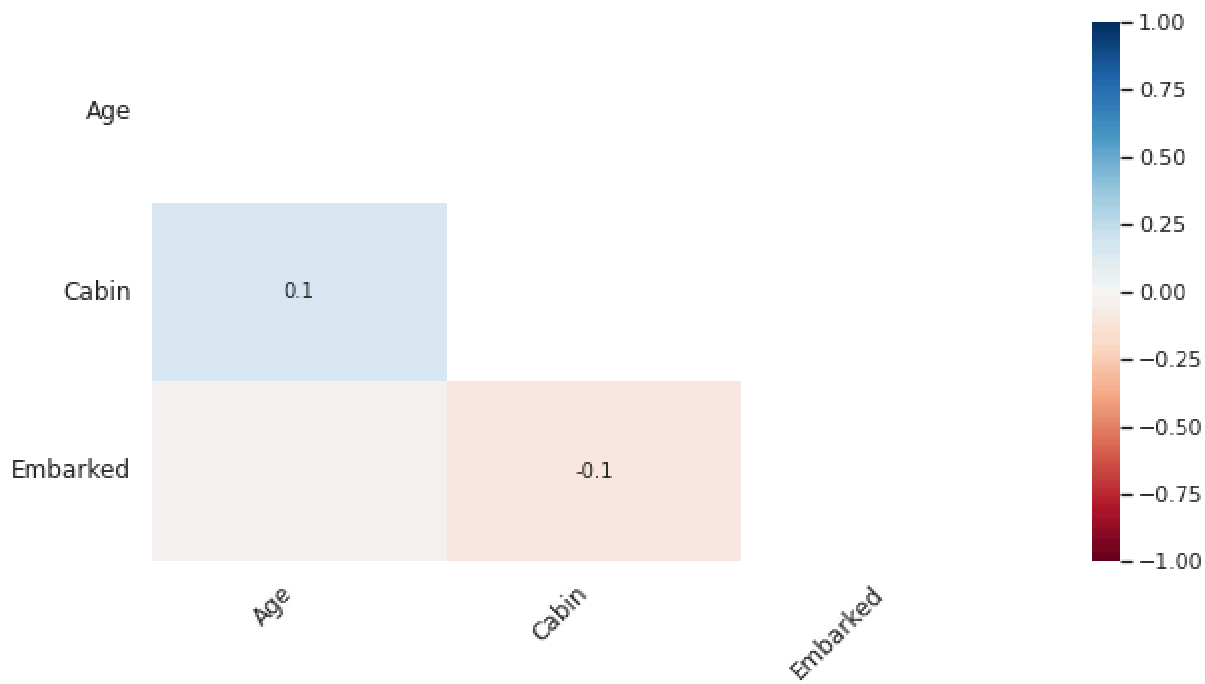
```
import missingno
missingno.bar(titanic, color="dodgerblue", sort="ascending", figsize=(10,5),
```



In [ ]:
```
missingno.matrix(titanic, figsize=(10,5), fontsize=12, color=(1, 0.38, 0.27)
```



In [ ]:
```
missingno.heatmap(titanic, figsize=(10,5), fontsize=12);
```



Feature Engineering

Filling/Removing Missing Values

In [ ]:
```
titanic['Age'] = titanic['Age'].fillna(titanic['Age'].mean())
```

In [ ]:
```
titanic[titanic['Embarked'].isnull()]
```

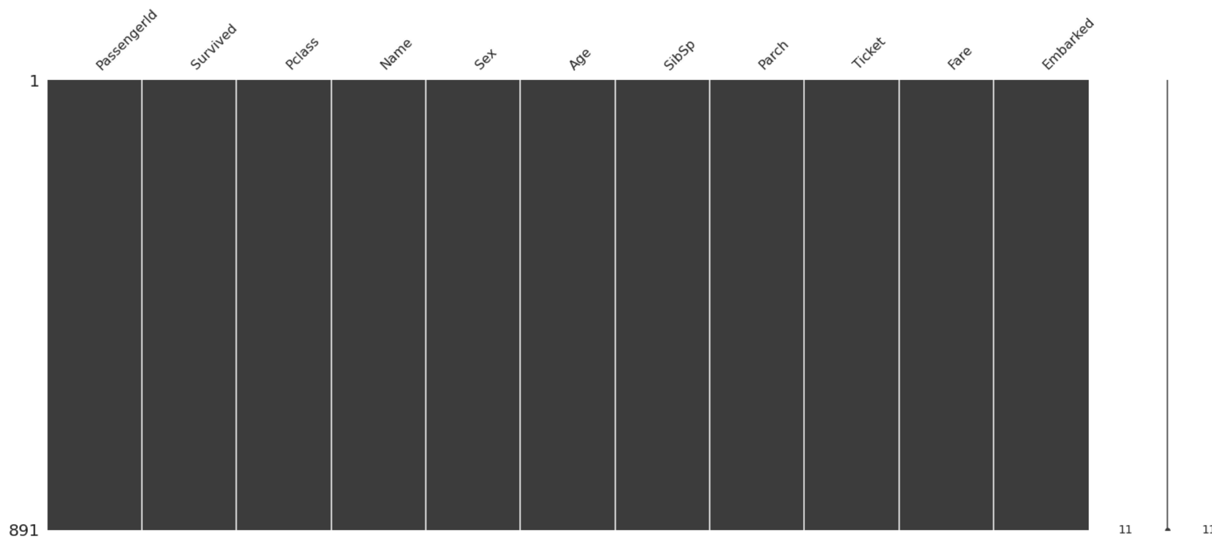
Out[ ]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticke
61	62	1	1	Icard, Miss. Amelie	female	38.0	0	0	11357
829	830	1	1	Stone, Mrs. George Nelson (Martha Evelyn)	female	62.0	0	0	11357

In [ ]:
```
titanic['Embarked'] = titanic['Embarked'].fillna(method='bfill')
```

In [ ]:
```
titanic = titanic.drop(['Cabin'],axis=1)
```

In [ ]:
```
import missingno as msno
msno.matrix(titanic)
plt.show()
```



In [ ]:
```
titanic.isnull().sum()
```

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

All the Missing Value is Filled/Removed

In [ ]:
```
titanic = titanic.drop(['Name','Ticket'],axis=1)
```

In [ ]:
```
titanic.head()
```

Out[ ]:

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embark
0	1	0	3	male	22.0	1	0	7.2500	
1	2	1	1	female	38.0	1	0	71.2833	
2	3	1	3	female	26.0	0	0	7.9250	
3	4	1	1	female	35.0	1	0	53.1000	
4	5	0	3	male	35.0	0	0	8.0500	

In [ ]:
```
titanic = pd.get_dummies(titanic,columns=['Sex','Embarked'],drop_first=True)
titanic.head()
```

Out[ ]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	Sex_male	Emb
0	1	0	3	22.0	1	0	7.2500	1	
1	2	1	1	38.0	1	0	71.2833	0	
2	3	1	3	26.0	0	0	7.9250	0	
3	4	1	1	35.0	1	0	53.1000	0	
4	5	0	3	35.0	0	0	8.0500	1	

Train Test Split

In [ ]:

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

In [ ]:

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_st
```

Standardizing the data

In [ ]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

X_train = pd.DataFrame(X_train, columns=X.columns)
X_test = pd.DataFrame(X_test, columns=X.columns)
```

In [ ]:

```
display(X_train.head())
display(X_test.head())
```

	PassengerId	Pclass	Age	SibSp	Parch	Fare	Sex_male
0	1.360492	-1.584396	0.010681	-0.479698	-0.460682	-0.018600	0.728823
1	-1.632266	-1.584396	-0.119643	-0.479698	-0.460682	0.079245	0.728823
2	-1.344650	-1.584396	-0.503148	-0.479698	0.810657	0.646624	0.728823
3	-1.686680	-0.381742	-1.193456	0.493365	-0.460682	-0.031329	-1.372075
4	-1.111449	0.820913	0.033758	-0.479698	-0.460682	-0.479818	0.728823

	PassengerId	Pclass	Age	SibSp	Parch	Fare	Sex_male
0	0.676433	0.820913	-0.273045	0.493365	-0.460682	-0.315867	-1.372075
1	-0.248601	0.820913	-0.809952	-0.479698	-0.460682	-0.485419	0.728823
2	1.096196	0.820913	-0.733251	-0.479698	-0.460682	-0.467343	0.728823
3	1.488753	0.820913	0.010681	-0.479698	-0.460682	0.506858	0.728823
4	0.027354	-0.381742	0.493964	0.493365	2.081997	-0.078596	0.728823

Model Implementation

LogisticRegression

In [ ]:

```
from sklearn.metrics import accuracy_score
# Logistic Regression
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
Y_pred = logreg.predict(X_test)

log_train = round(logreg.score(X_train, y_train) * 100, 2)
log_accuracy = round(accuracy_score(Y_pred, y_test) * 100, 2)

print("Training Accuracy    :",log_train)
print("Model Accuracy Score :",log_accuracy)
```

```
Training Accuracy    : 80.2
Model Accuracy Score : 79.89
```

Support Vector Machines

In [ ]:

```
# Support Vector Machines
from sklearn.svm import SVC
svc = SVC()
svc.fit(X_train, y_train)
Y_pred = svc.predict(X_test)

svc_train = round(svc.score(X_train, y_train) * 100, 2)
svc_accuracy = round(accuracy_score(Y_pred, y_test) * 100, 2)

print("Training Accuracy     :",svc_train)
print("Model Accuracy Score  :",svc_accuracy)
```

```
Training Accuracy     : 85.11
Model Accuracy Score  : 80.45
```

KNeighborsClassifier

In [ ]:

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(X_train, y_train)
Y_pred = knn.predict(X_test)

knn_train = round(knn.score(X_train, y_train) * 100, 2)
knn_accuracy = round(accuracy_score(Y_pred, y_test) * 100, 2)

print("Training Accuracy     :",knn_train)
print("Model Accuracy Score  :",knn_accuracy)
```

```
Training Accuracy     : 90.03
Model Accuracy Score  : 75.98
```

GaussianNB

In [ ]:

```
# Gaussian Naive Bayes
from sklearn.naive_bayes import GaussianNB
gaussian = GaussianNB()
gaussian.fit(X_train, y_train)
Y_pred = gaussian.predict(X_test)

gaussian_train = round(gaussian.score(X_train, y_train) * 100, 2)
gaussian_accuracy = round(accuracy_score(Y_pred, y_test) * 100, 2)

print("Training Accuracy     :",gaussian_train)
print("Model Accuracy Score  :",gaussian_accuracy)
```

```
Training Accuracy     : 79.21
Model Accuracy Score  : 81.56
```

LinearSVC

In [ ]:

```
# Linear SVC
from sklearn.svm import LinearSVC
linear_svc = LinearSVC()
linear_svc.fit(X_train, y_train)
Y_pred = linear_svc.predict(X_test)

linear_svc_train = round(linear_svc.score(X_train, y_train) * 100, 2)
linear_svc_accuracy = round(accuracy_score(Y_pred, y_test) * 100, 2)

print("Training Accuracy    :",linear_svc_train)
print("Model Accuracy Score :",linear_svc_accuracy)
```

```
Training Accuracy    : 80.34
Model Accuracy Score : 81.01
```

DecisionTreeClassifier

```
In [ ]:
# Decision Tree
from sklearn.tree import DecisionTreeClassifier
decision = DecisionTreeClassifier()
decision.fit(X_train, y_train)
Y_pred = decision.predict(X_test)

decision_train = round(decision.score(X_train, y_train) * 100, 2)
decision_accuracy = round(accuracy_score(Y_pred, y_test) * 100, 2)

print("Training Accuracy     :",decision_train)
print("Model Accuracy Score  :",decision_accuracy)
```

```
Training Accuracy     : 100.0
Model Accuracy Score  : 72.63
```

RandomForestClassifier

```
In [ ]:
# Random Forest
from sklearn.ensemble import RandomForestClassifier
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(X_train, y_train)
Y_pred = random_forest.predict(X_test)
random_forest.score(X_train, y_train)

random_forest_train = round(random_forest.score(X_train, y_train) * 100, 2)
random_forest_accuracy = round(accuracy_score(Y_pred, y_test) * 100, 2)

print("Training Accuracy     :",random_forest_train)
print("Model Accuracy Score  :",random_forest_accuracy)
```

```
Training Accuracy     : 100.0
Model Accuracy Score  : 81.56
```

XGBClassifier

```
In [ ]:
import xgboost as Xgb
xgb = Xgb.XGBClassifier()
xgb.fit(X_train,y_train)
Y_pred = xgb.predict(X_test)
xgb.score(X_train, y_train)

xgb_train = round(xgb.score(X_train, y_train) * 100, 2)
xgb_accuracy = round(accuracy_score(Y_pred, y_test) * 100, 2)

print("Training Accuracy     :",xgb_train)
print("Model Accuracy Score  :",xgb_accuracy)
```

```
Training Accuracy     : 100.0
Model Accuracy Score  : 77.65
```

Comparing Models

In [ ]:
```
models = pd.DataFrame({
    'Model': [
        'Support Vector Machines', 'KNN', 'Logistic Regression',
        'Random Forest',  'Perceptron',
        'Stochastic Gradient Decent', 'Linear SVC', 'Decision Tree',
        'GaussianNB', 'MLPClassifier', 'XGBClassifier'
    ],
    'Training Accuracy': [
        log_train, svc_train, knn_train, gaussian_train, perceptron_train,
        linear_svc_train, sgd_train, decision_train, random_forest_train,
        mlp_train, xgb_train
    ],
    'Model Accuracy Score': [
        log_accuracy, svc_accuracy, knn_accuracy, gaussian_accuracy, percept
        linear_svc_accuracy, sgd_accuracy, decision_accuracy, random_forest_
        mlp_accuracy, xgb_accuracy
    ]
})
```

In [ ]:
```
models.sort_values(by='Training Accuracy', ascending=False)
```

Out[ ]:

	Model	Training Accuracy	Model Accuracy Score
7	Decision Tree	100.00	72.63
8	GaussianNB	100.00	81.56
10	XGBClassifier	100.00	77.65
2	Logistic Regression	90.03	75.98
9	MLPClassifier	86.38	79.89
1	KNN	85.11	80.45
5	Stochastic Gradient Decent	80.34	81.01
0	Support Vector Machines	80.20	79.89
6	Linear SVC	79.78	81.01
3	Random Forest	79.21	81.56
4	Perceptron	73.17	72.07

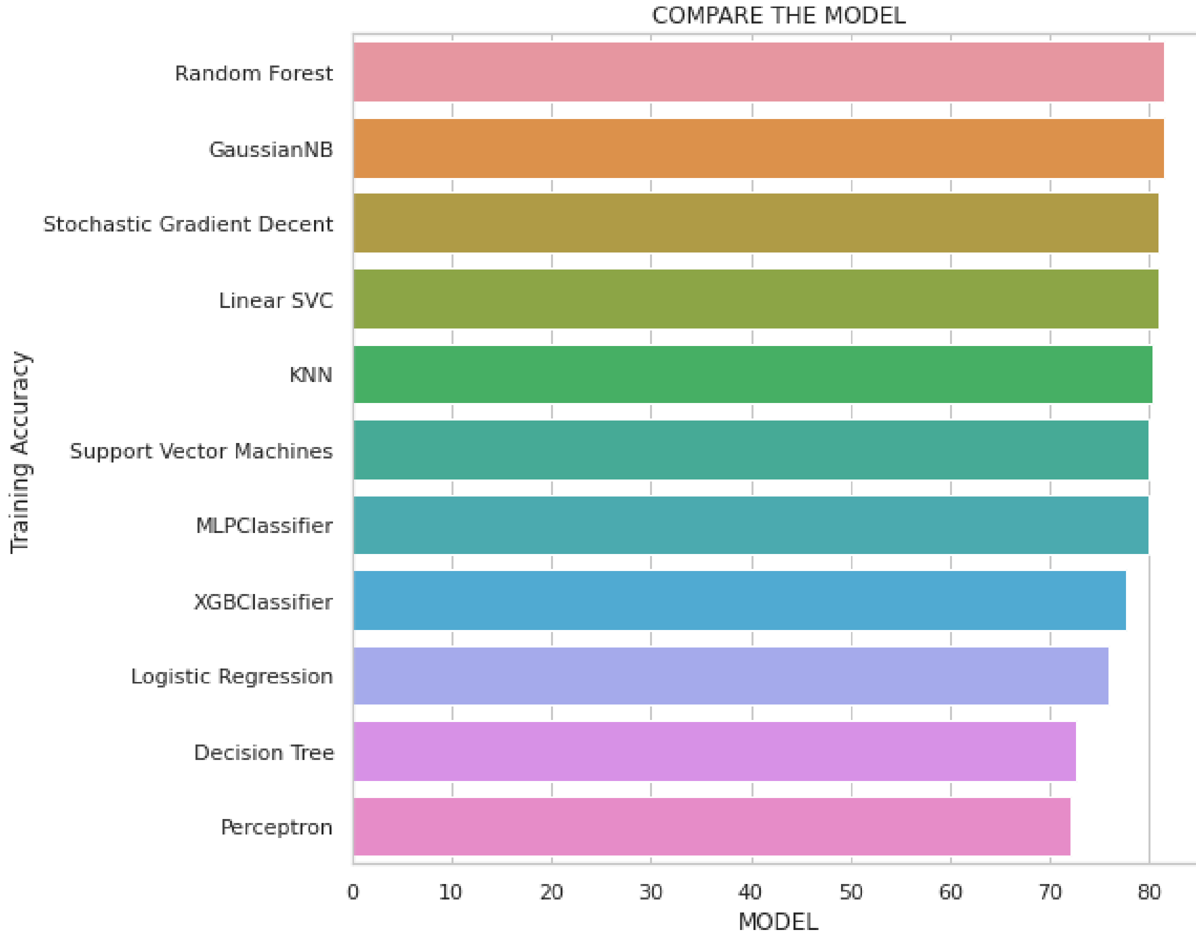
In [ ]:
```
models.sort_values(by='Model Accuracy Score', ascending=False).style.backgro
        cmap='coolwarm').hide_index().set_properties(**{
            'font-family': 'Lucida Calligraphy',
            'color': 'LigntGreen',
            'font-size': '15px'
        })
```

Out[ ]:

	Model	Training Accuracy	Model Accuracy Score
	Random Forest	79.210000	81.560000
	GaussianNB	100.000000	81.560000
	Stochastic Gradient Decent	80.340000	81.010000
	Linear SVC	79.780000	81.010000
	KNN	85.110000	80.450000
	Support Vector Machines	80.200000	79.890000
	MLPClassifier	86.380000	79.890000
	XGBClassifier	100.000000	77.650000
	Logistic Regression	90.030000	75.980000
	Decision Tree	100.000000	72.630000
	Perceptron	73.170000	72.070000

In [ ]:

```
models=models.sort_values(by='Model Accuracy Score',ascending=False)[:20]

sns.barplot(y= 'Model', x= 'Model Accuracy Score', data= models)
plt.title('COMPARE THE MODEL')
plt.xlabel('MODEL')
plt.ylabel('Training Accuracy');
```



Conclusion

In this notebook, we compared multiple machine learning models on both their training and test accuracy scores. Several observations stand out:

1. **Overfitting Indicators**: Models like Decision Tree, GaussianNB, and XGBClassifier achieved 100% training accuracy, but their test accuracies dropped—most notably, the Decision Tree's test accuracy fell to 72.63%, indicating overfitting.

2. **Top Performers**: Despite having perfect training scores, **GaussianNB** and **Random Forest** both reached the highest test accuracy of **81.56%**, suggesting they generalize well to unseen data compared to other models.

3. **Competitive Alternatives**: Stochastic Gradient Descent, Linear SVC, and KNN also performed relatively well, with test accuracies around 80–81%. Logistic Regression and MLPClassifier hovered in the mid-70s to high-70s range.

4. **Next Steps**:

- **Hyperparameter Tuning**: Fine-tune the top models (GaussianNB, Random Forest, etc.) to see if performance can be further improved.
- **Cross-Validation**: Employ more robust evaluation methods (e.g., k-fold cross-validation) for reliable performance estimates.
- **Additional Metrics**: Consider metrics like precision, recall, and F1-score to gain deeper insights into model performance.

Overall, GaussianNB and Random Forest emerged as strong contenders for this dataset. Further refinement and additional validation will help confirm which model offers the best balance of accuracy, efficiency, and interpretability.

Acknowledgements

Special Thanks:
I would like to extend my heartfelt gratitude to DataPlay Company for the fellowship. This opportunity has been instrumental in enhancing my skills and enabling projects like this to flourish.

End of Notebook

This notebook was converted with convert.ploomber.io