In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, roc_auc_score
from sklearn.decomposition import PCA

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
```

In [2]:

```
df=pd.read_csv('C:/Users/mmi/Downloads/titanic/train.csv')

df.head()
```

Out[2]:

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

In [3]:

```
df.fillna(df.mean(),inplace=True)
```

In [4]:

```
df.isnull().sum()
```

Out[4]:

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

In [5]:

```
df.drop(['PassengerId','Name','Ticket','Cabin'], axis=1, inplace=True)
```

In [6]:

```
df.head()
```

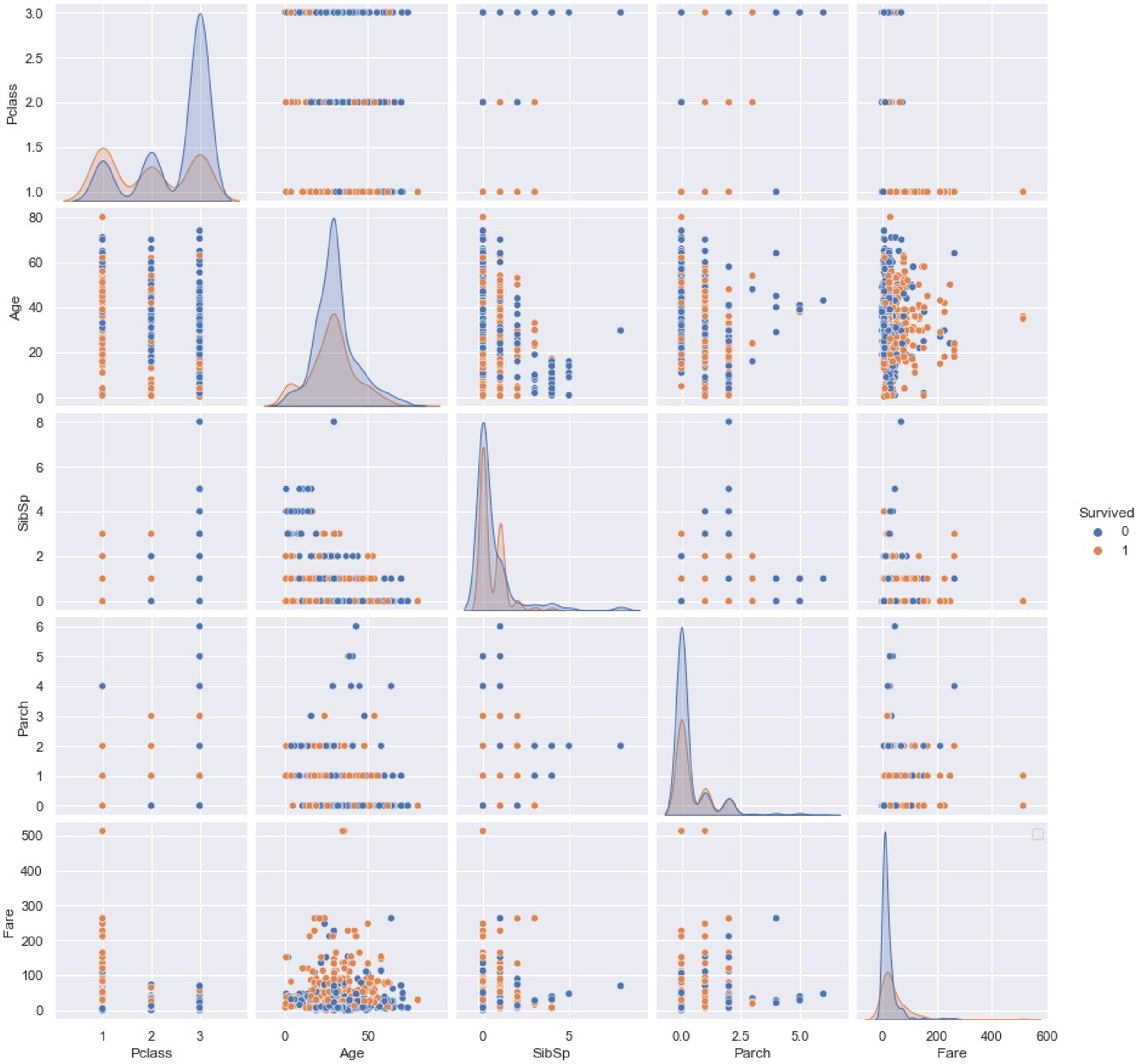
Out[6]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	male	22.0	1	0	7.2500	S
1	1	1	female	38.0	1	0	71.2833	C
2	1	3	female	26.0	0	0	7.9250	S
3	1	1	female	35.0	1	0	53.1000	S
4	0	3	male	35.0	0	0	8.0500	S

In [7]:

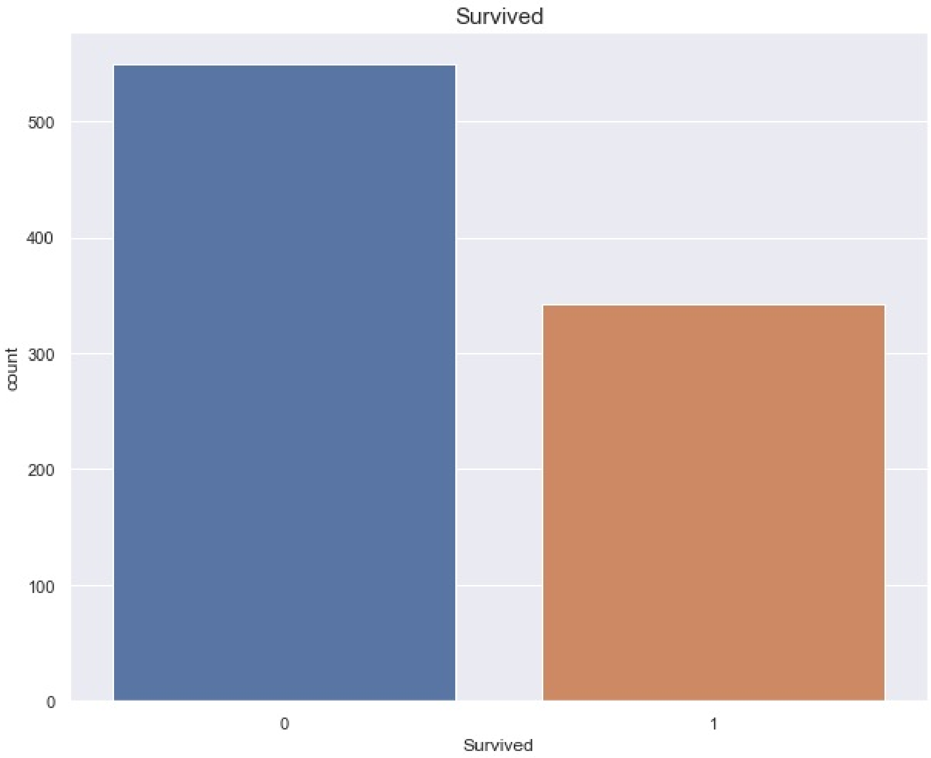
```
sns.set()

cols_to_pairplot = df.columns[:11]
sns.pairplot(df[cols_to_pairplot], hue="Survived")
plt.legend()
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label start with an underscore are igno
red when legend() is called with no argument.
```

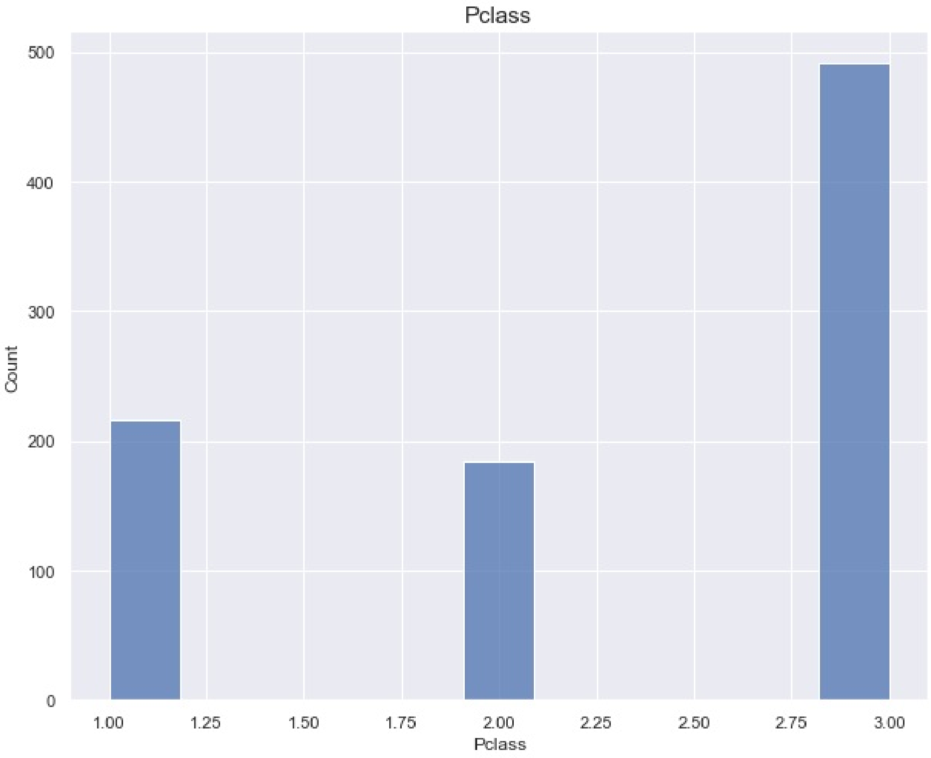


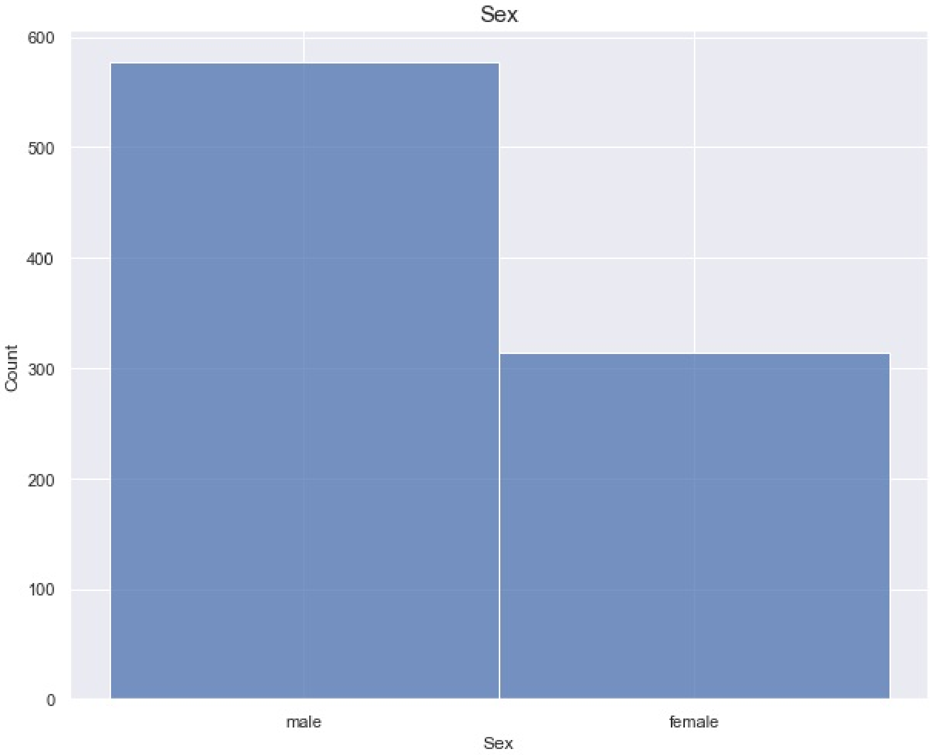
In [8]:

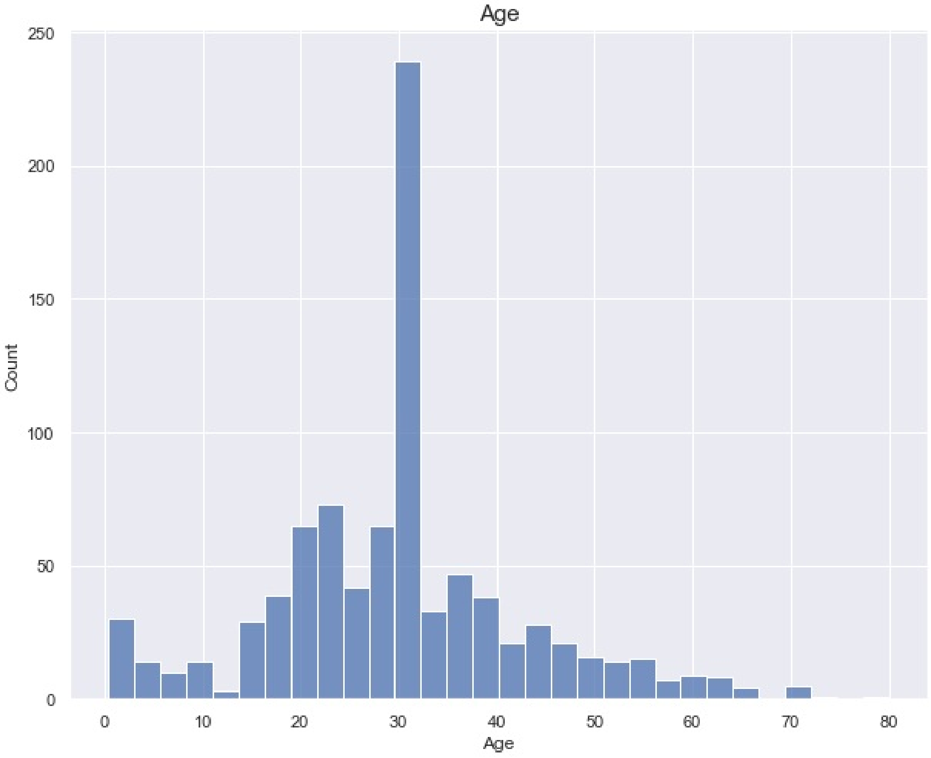
```
plt.figure(figsize=(10,8))
sns.countplot(df["Survived"])
plt.title("Survived", size=15)
plt.show()
```

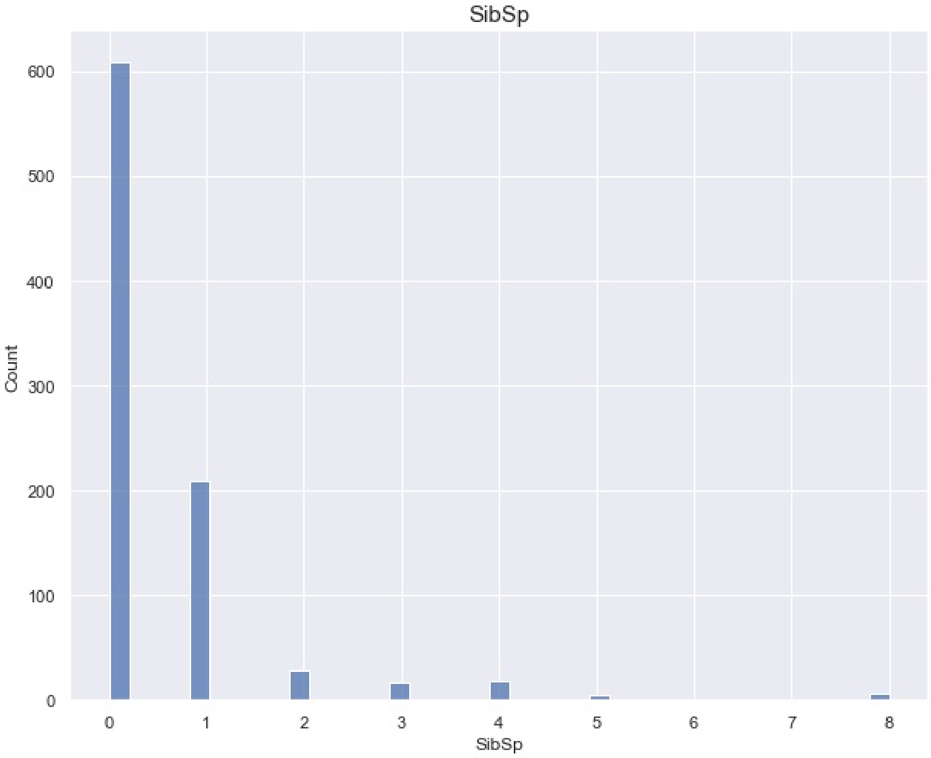


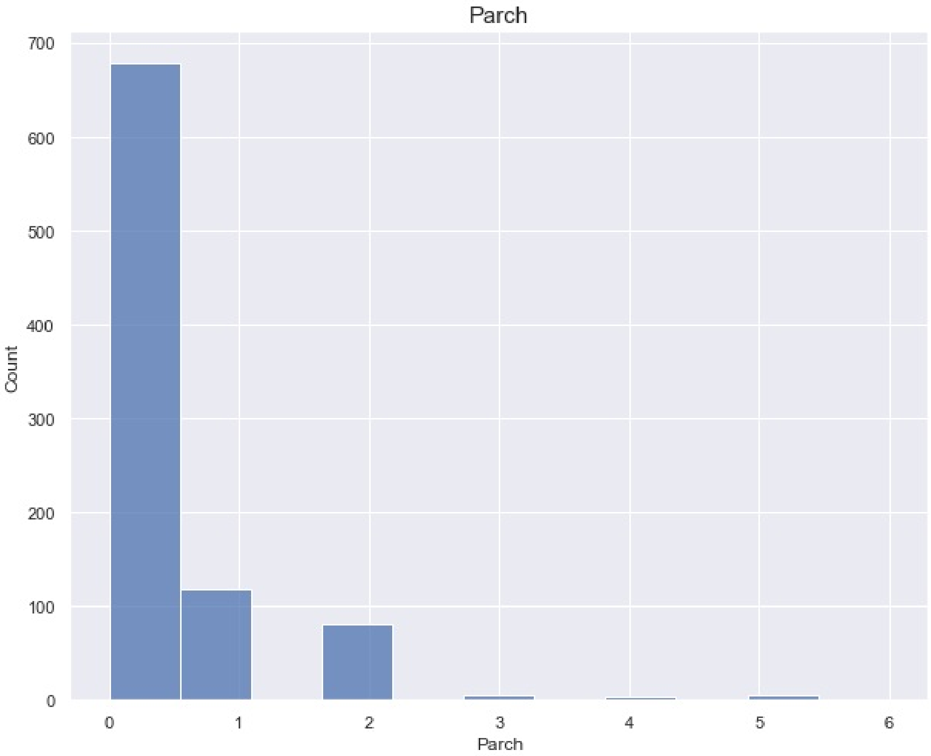
In [9]:
```
for col in df.drop("Survived", axis=1).columns:
    plt.figure(figsize=(10,8))
    sns.histplot(df[col])
    plt.title(f"{col}", size=15)
    plt.show()
```

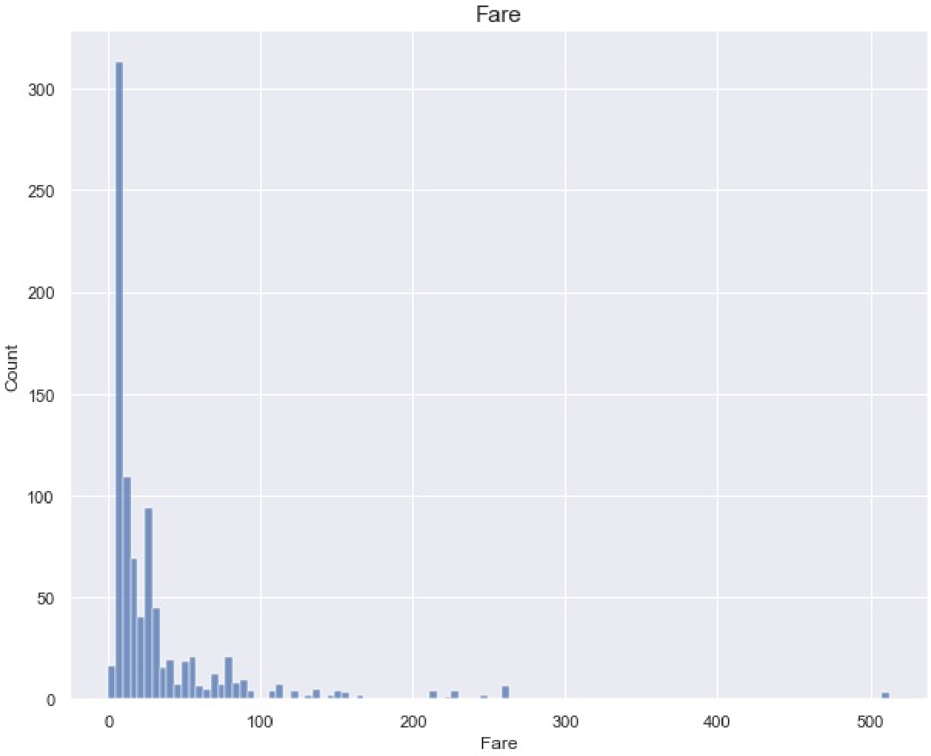


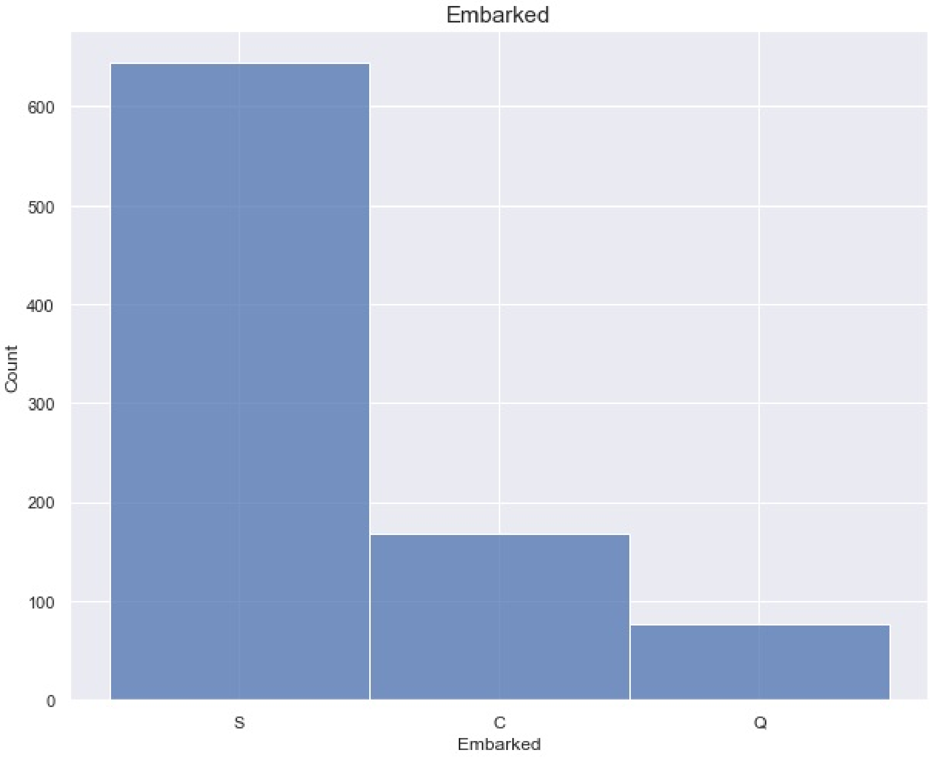

Sex
600
500
400
300
200
100
0
Count
male
female
Sex


Age
250
200
150
100
50
0
Count
0
10
20
30
40
50
60
70
80
Age

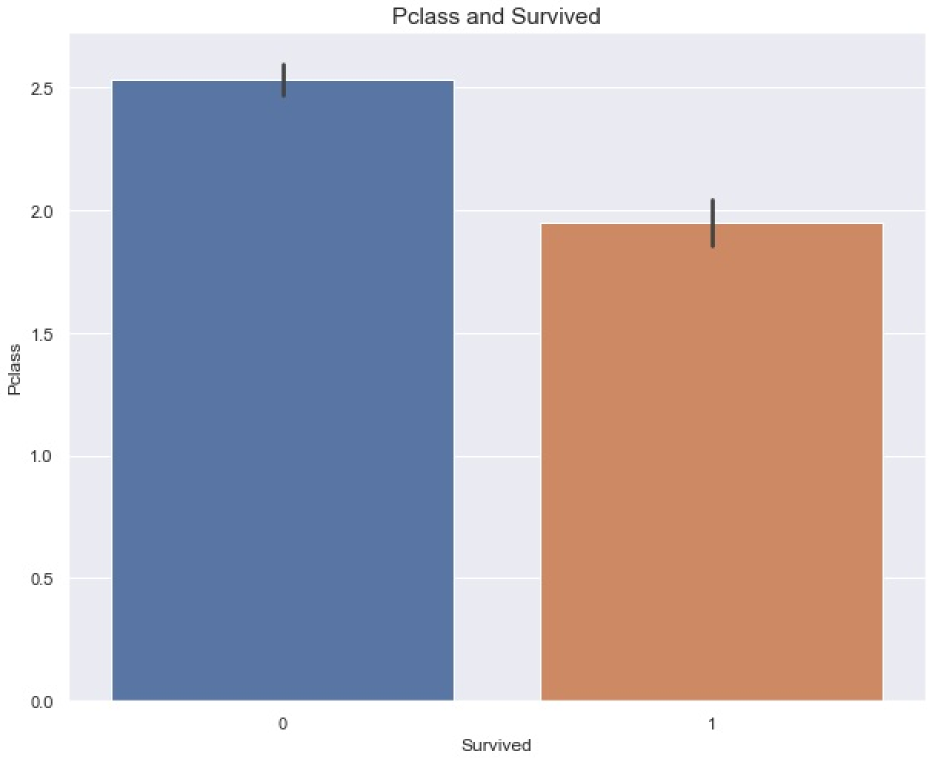

SibSp
Count
0
100
200
300
400
500
600
0
1
2
3
4
5
6
7
8
SibSp

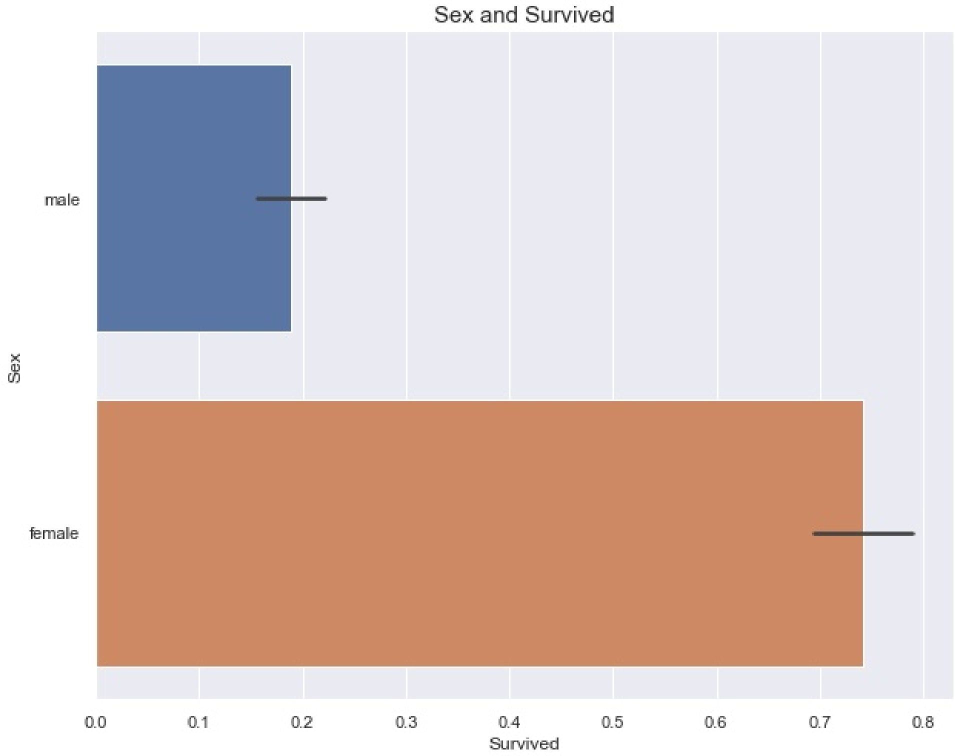

Parch
Count
0
100
200
300
400
500
600
700
0
1
2
3
4
5
6
Parch

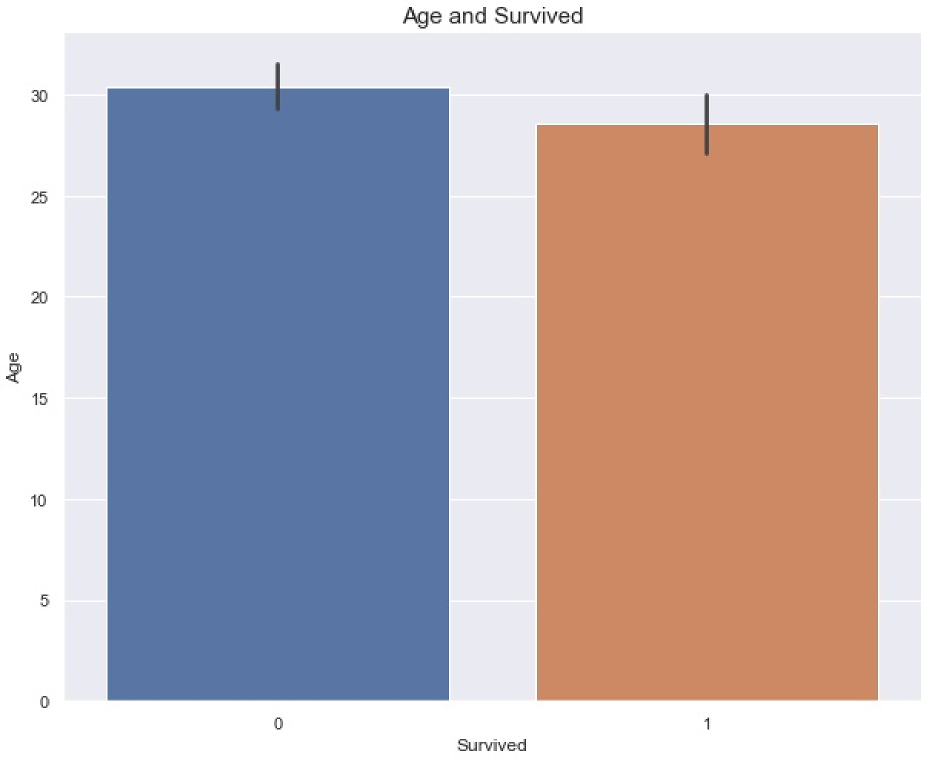


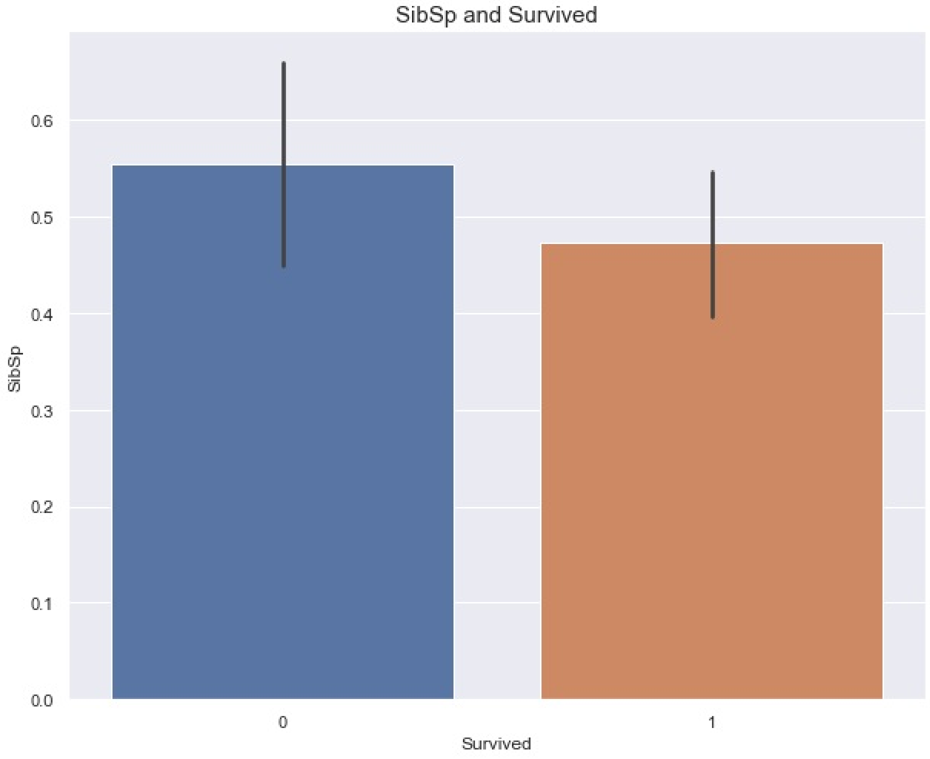


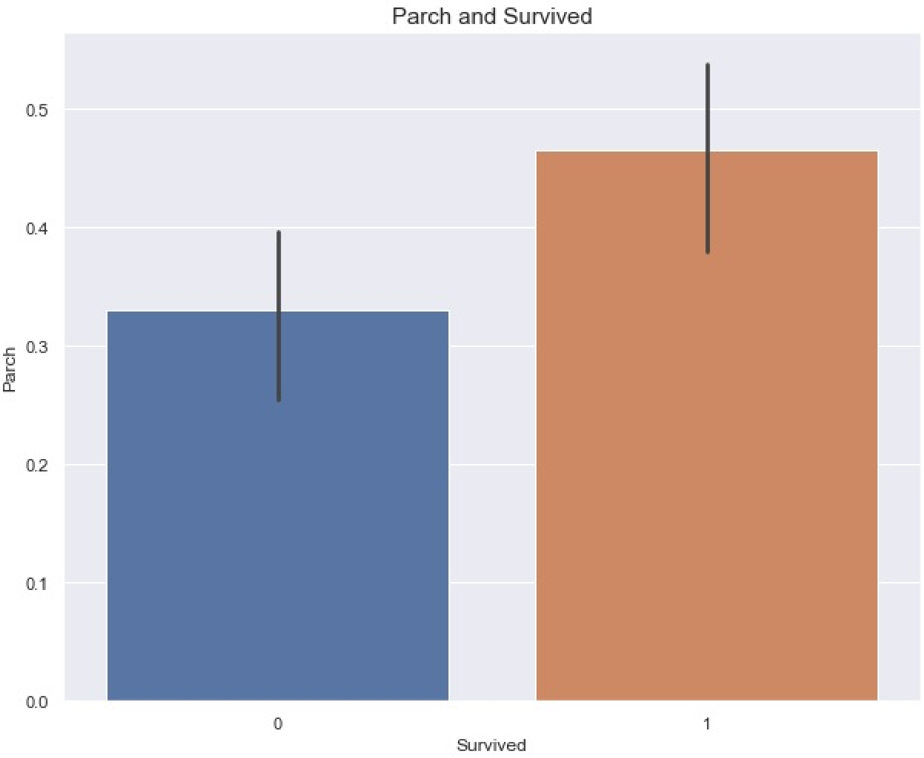
In [10]:
```
for col in df.drop("Survived", axis=1).columns:
    plt.figure(figsize=(10,8))
    sns.barplot(x=df["Survived"], y=df[col])
    plt.title(f"{col} and Survived", size=15)
    plt.show()
```

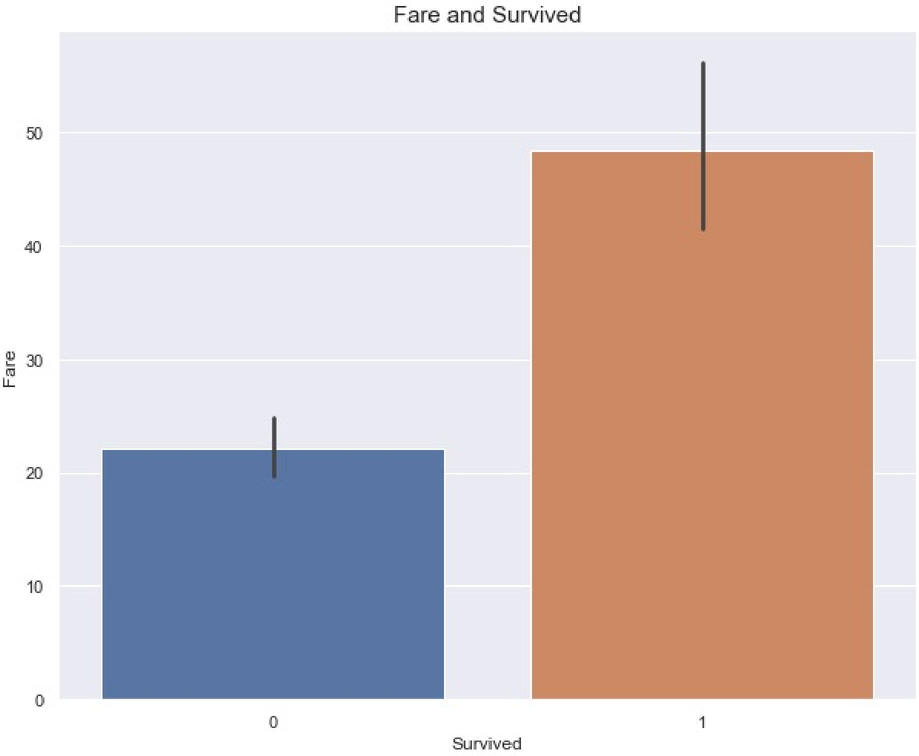

Pclass and Survived
2.5
2.0
1.5
1.0
0.5
0.0
Pclass
0
1
Survived

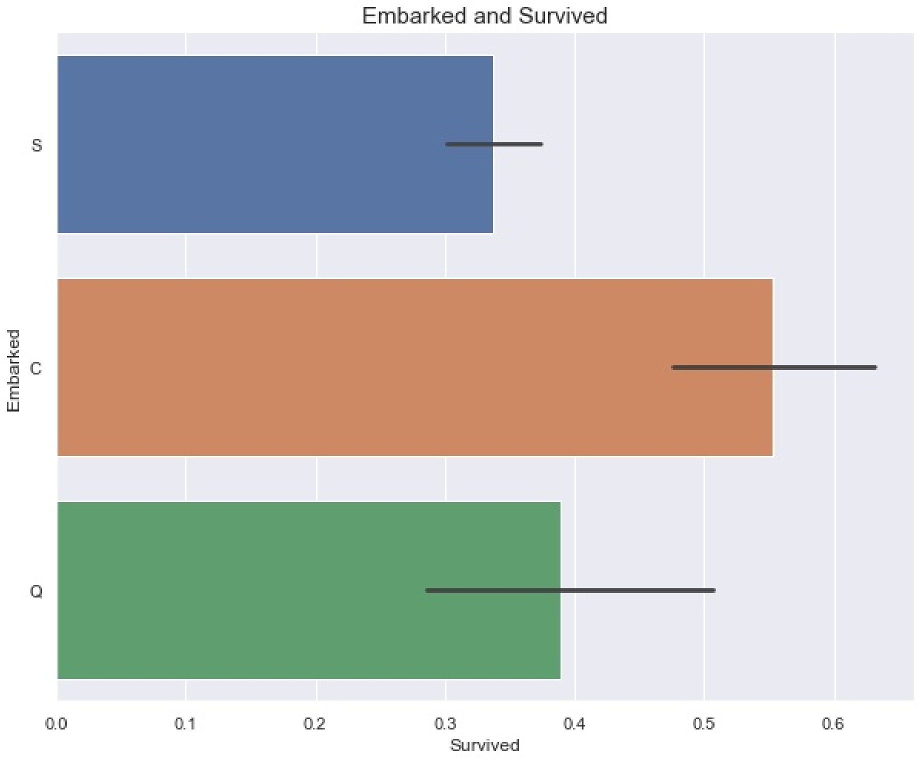

Sex and Survived
male
female
Sex
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
Survived


Age and Survived
Age
30
25
20
15
10
5
0
0
1
Survived


SibSp and Survived
SibSp
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
1
Survived

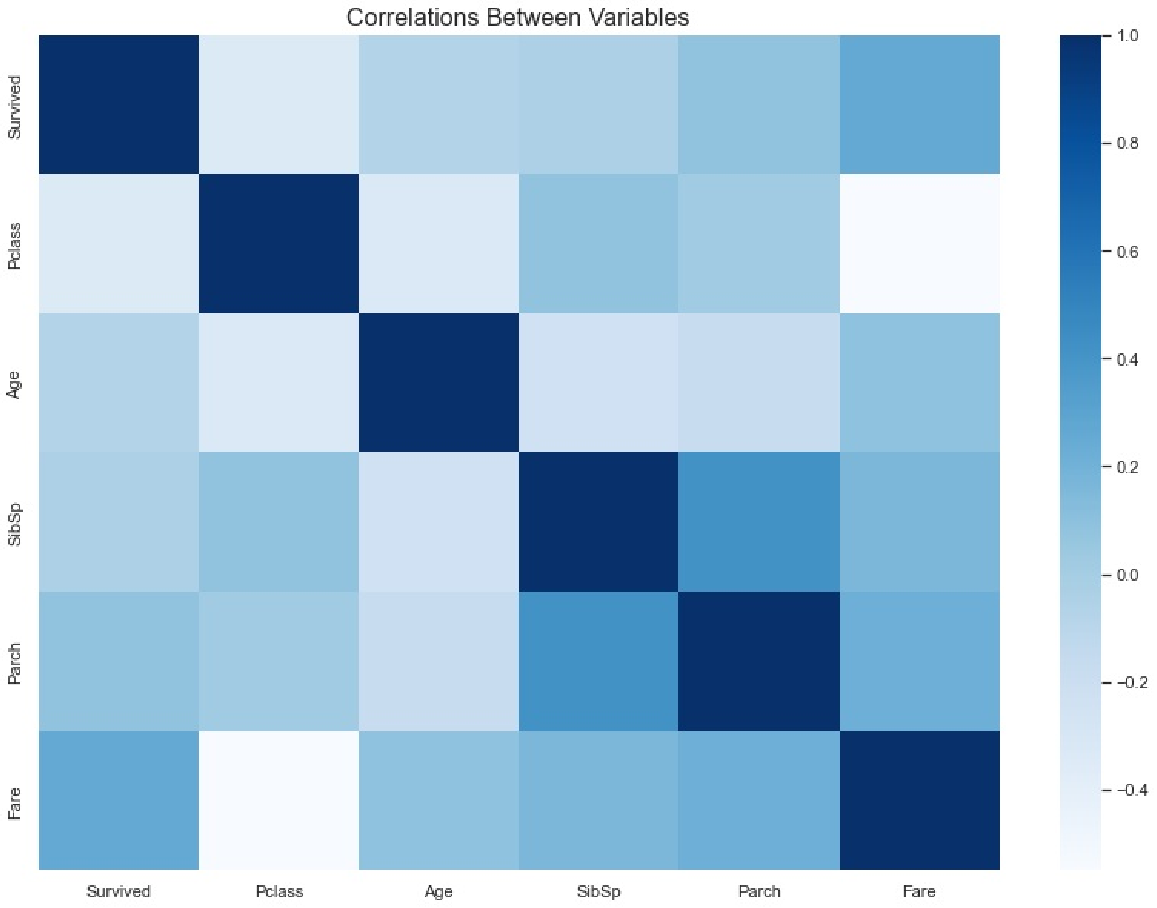

Parch and Survived
0.5
0.4
0.3
0.2
0.1
0.0
Parch
0
1
Survived


Fare and Survived
50
40
30
20
10
0
Fare
0
1
Survived



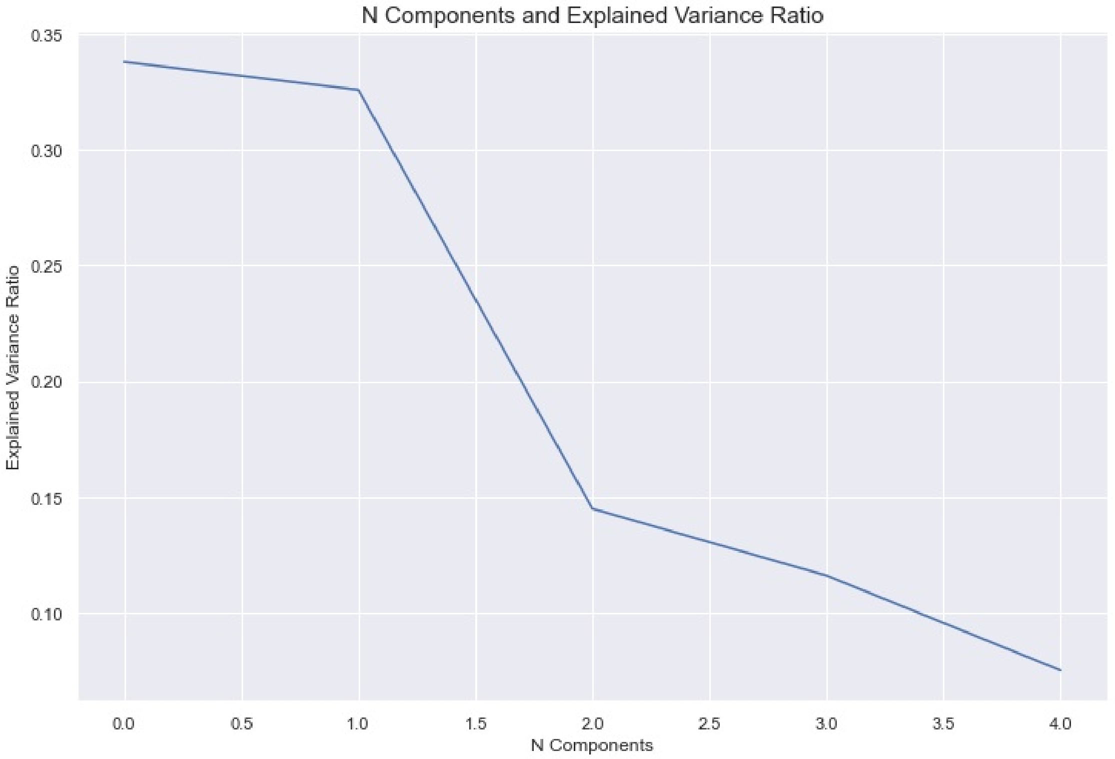
In [11]:

```
plt.figure(figsize=(14,10))
sns.heatmap(df.corr(), cmap="Blues")
plt.title("Correlations Between Variables", size=16)
plt.show()
```



In [18]:
```
X = df.drop("Survived", axis=1)
y = df["Survived"].replace({"male": 0, "female": 1})
X=df[['Pclass','Age','SibSp','Parch','Fare']]
```

In [19]:
```
scaler = StandardScaler()
X = scaler.fit_transform(X)
pca = PCA()
pca.fit(X)
plt.figure(figsize=(12,8))
plt.plot(pca.explained_variance_ratio_)
plt.title("N Components and Explained Variance Ratio", size=15)
plt.xlabel("N Components")
plt.ylabel("Explained Variance Ratio")
plt.show()
```



In [20]:
```
pca = PCA(n_components = 5)
X = pca.fit_transform(X)
pca.explained_variance_ratio_.sum()
```

Out[20]:

```
1.0
```

In [21]:

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

In [22]:

```python
models = pd.DataFrame(columns=["Model", "Accuracy Score"])
```

In [23]:

```python
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)
predictions = log_reg.predict(X_test)
score = accuracy_score(y_test, predictions)
print("Accuracy Score:", score)
new_row = {"Model": "LogisticRegression", "Accuracy Score": score}
models = models.append(new_row, ignore_index=True)
```

```
Accuracy Score: 0.7201492537313433
```

In [24]:

```python
rfc = RandomForestClassifier()
rfc.fit(X_train, y_train)
predictions = rfc.predict(X_test)
score = accuracy_score(y_test, predictions)
print("Accuracy Score:", score)

new_row = {"Model": "RandomForestClassifier", "Accuracy Score": score}
models = models.append(new_row, ignore_index=True)
```

```
Accuracy Score: 0.6455223880597015
```

In [25]:

```python
gbc = GradientBoostingClassifier()
gbc.fit(X_train, y_train)
predictions = gbc.predict(X_test)
score = accuracy_score(y_test, predictions)
print("Accuracy Score:", score)

new_row = {"Model": "GradientBoostingClassifier", "Accuracy Score": score}
models = models.append(new_row, ignore_index=True)
```

```
Accuracy Score: 0.6977611940298507
```

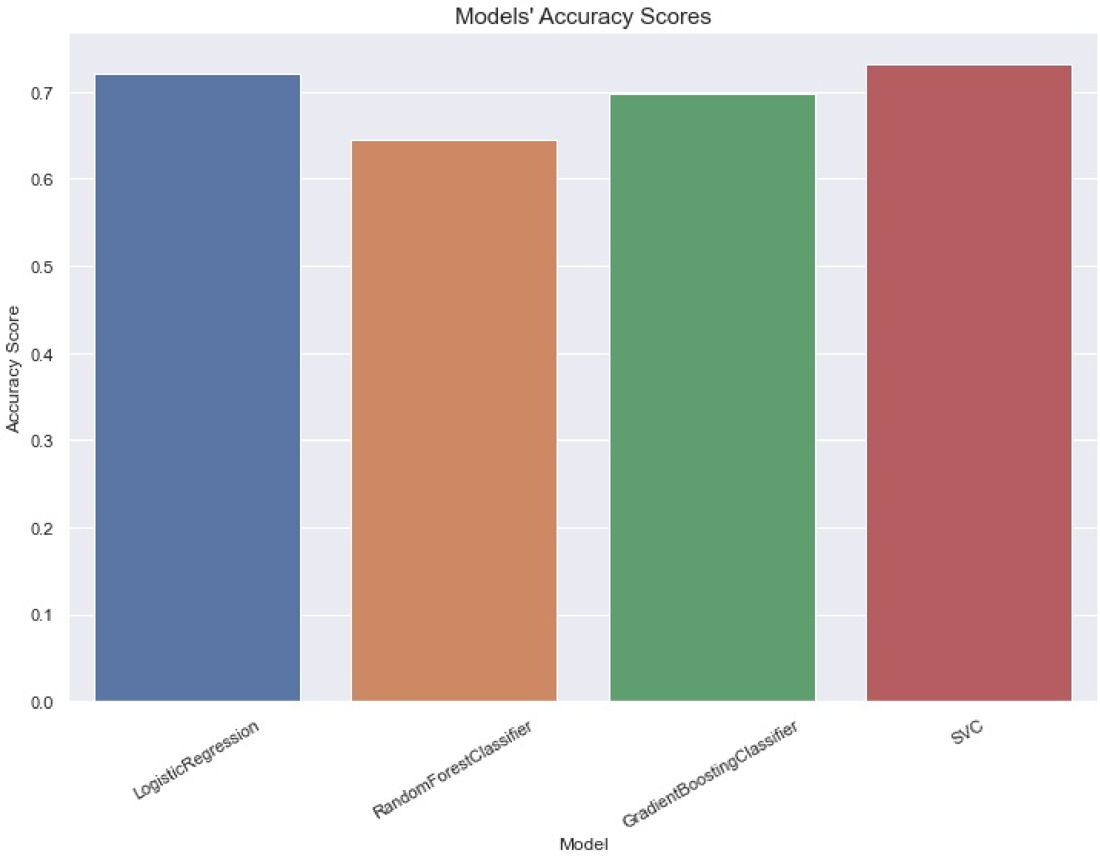
In [26]:

```python
svc = SVC()
svc.fit(X_train, y_train)
predictions = svc.predict(X_test)
score = accuracy_score(y_test, predictions)
print("Accuracy Score:", score)

new_row = {"Model": "SVC", "Accuracy Score": score}
models = models.append(new_row, ignore_index=True)
```

```
Accuracy Score: 0.7313432835820896
```

In [27]:

```python
models.sort_values(by="Accuracy Score", ascending=False)
plt.figure(figsize=(12,8))
sns.barplot(x=models["Model"], y=models["Accuracy Score"])
plt.title("Models' Accuracy Scores", size=15)
plt.xticks(rotation=30)
plt.show()
```



In [29]:

```
def visualize_roc_auc_curve(model, model_name):
    pred_prob = model.predict_proba(X_test)
    fpr, tpr, thresh = roc_curve(y_test, pred_prob[:,1], pos_label=1)

    score = roc_auc_score(y_test, pred_prob[:, 1])

    plt.figure(figsize=(10,8))
    plt.plot(fpr, tpr, linestyle="--", color="orange", label="ROC curve (area = %0.5f)" % score)
    plt.plot([0, 1], [0, 1], color="navy", lw=2, linestyle="--")

    plt.title(f"{model_name} ROC Curve", size=15)
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.legend(loc="lower right", prop={'size': 15})
    plt.show()
```

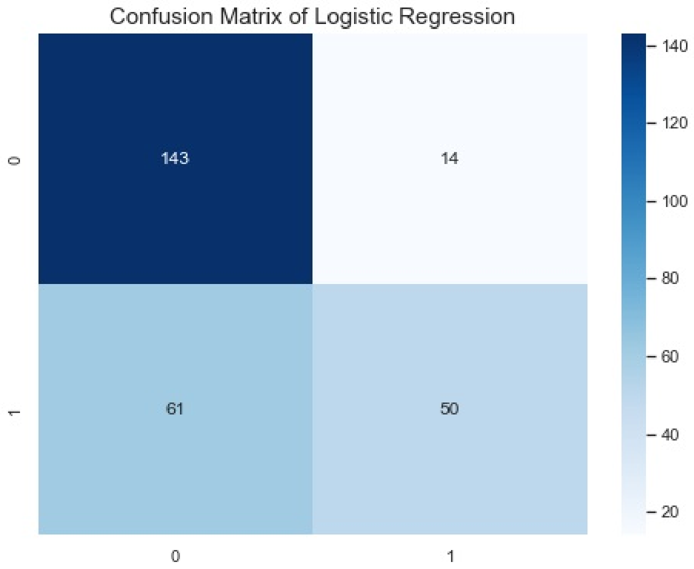
In [32]:

```
tuned_models = pd.DataFrame(columns=["Model", "Accuracy Score"])
param_grid_log_reg = {"C": [0.0001, 0.001, 0.01, 0.1, 1, 10]}
grid_log_reg = GridSearchCV(LogisticRegression(), param_grid_log_reg, scoring="accuracy", cv=5, verbose=0, n_jo

grid_log_reg.fit(X_train, y_train)
log_reg_params = grid_log_reg.best_params_
log_reg = LogisticRegression(**log_reg_params)
log_reg.fit(X_train, y_train)
predictions = log_reg.predict(X_test)
score = accuracy_score(y_test, predictions)
print("Accuracy Score:", score)

new_row = {"Model": "LogisticRegression", "Accuracy Score": score}
tuned_models = tuned_models.append(new_row, ignore_index=True)
```

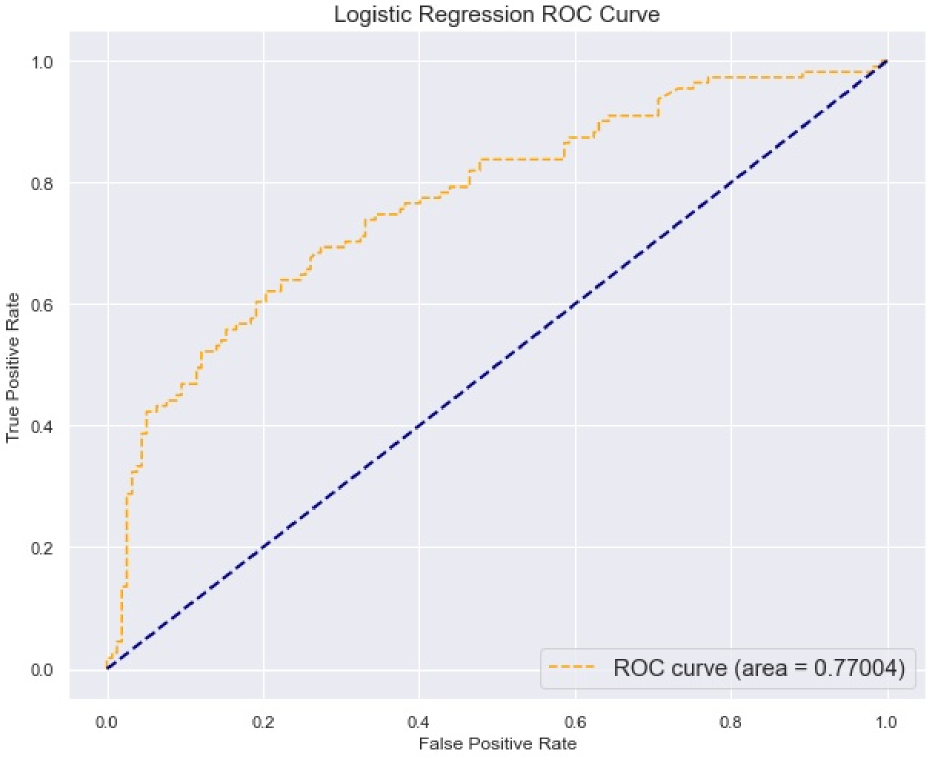
```
Accuracy Score: 0.7201492537313433
```

In [33]:

```
plt.figure(figsize=(8,6))
sns.heatmap(confusion_matrix(y_test, predictions), annot=True, cmap="Blues", fmt="d")
plt.title("Confusion Matrix of Logistic Regression", size=15)
plt.show()
```



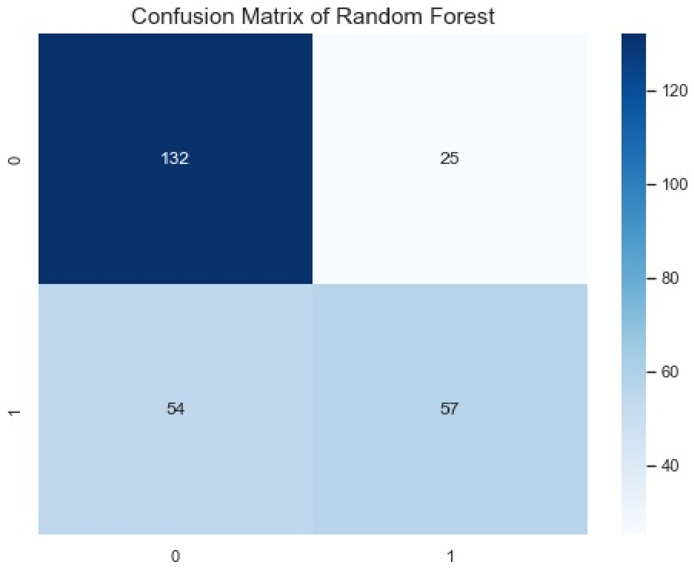
In [34]:
```
visualize_roc_auc_curve(log_reg, "Logistic Regression")
```



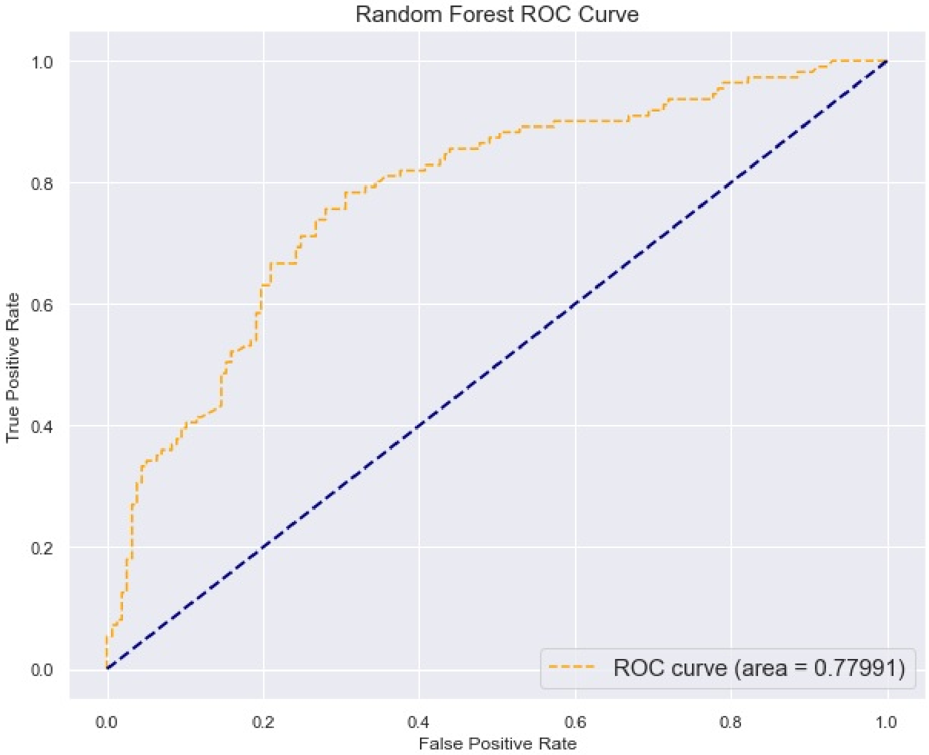
In [35]:
```
param_grid_rfc = {"min_samples_split": [2, 3, 10],
                  "min_samples_leaf": [1, 3, 10],
                  "n_estimators" :[100, 200, 500]}

grid_rfc = GridSearchCV(RandomForestClassifier(), param_grid_rfc, scoring="accuracy", cv=5, verbose=0, n_jobs=-

grid_rfc.fit(X_train, y_train)
rfc_params = grid_rfc.best_params_
rfc = RandomForestClassifier(**rfc_params)
rfc.fit(X_train, y_train)
predictions = rfc.predict(X_test)
score = accuracy_score(y_test, predictions)
print("Accuracy Score:", score)

new_row = {"Model": "RandomForestClassifier", "Accuracy Score": score}
tuned_models = tuned_models.append(new_row, ignore_index=True)
```

```
Accuracy Score: 0.7052238805970149
```

In [36]:
```
plt.figure(figsize=(8,6))
sns.heatmap(confusion_matrix(y_test, predictions), annot=True, cmap="Blues", fmt='d')
plt.title("Confusion Matrix of Random Forest", size=15)
plt.show()
```



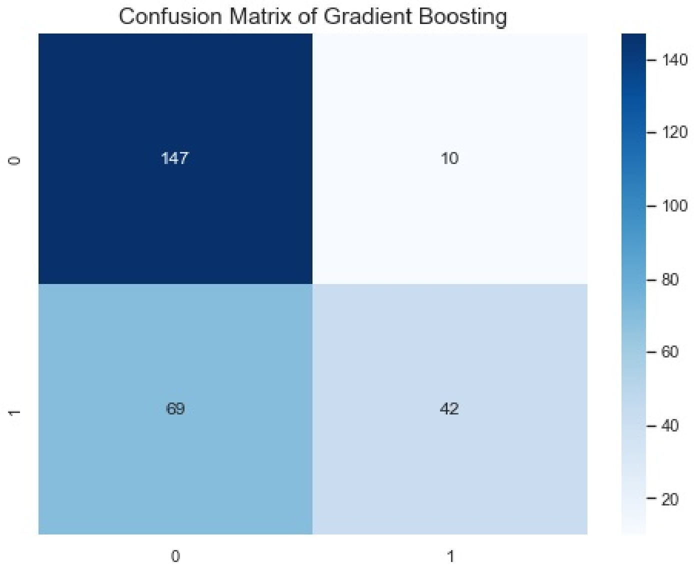
In [37]:
```
visualize_roc_auc_curve(rfc, "Random Forest")
```



In [38]:
```
param_grid_gbc = {'n_estimators' : [100, 200, 500],
                  'learning_rate': [0.1, 0.05, 0.01],
                  'max_depth': [2, 3, 6],
                  'min_samples_leaf': [1, 2, 5]}

grid_gbc = GridSearchCV(GradientBoostingClassifier(), param_grid_gbc, scoring="accuracy", cv=5, verbose=0, n_jo

grid_gbc.fit(X_train, y_train)
gbc_params = grid_gbc.best_params_
gbc = GradientBoostingClassifier(**gbc_params)
gbc.fit(X_train, y_train)
predictions = gbc.predict(X_test)
score = accuracy_score(y_test, predictions)
print("Accuracy Score:", score)

new_row = {"Model": "GradientBoostingClassifier", "Accuracy Score": score}
tuned_models = tuned_models.append(new_row, ignore_index=True)
```

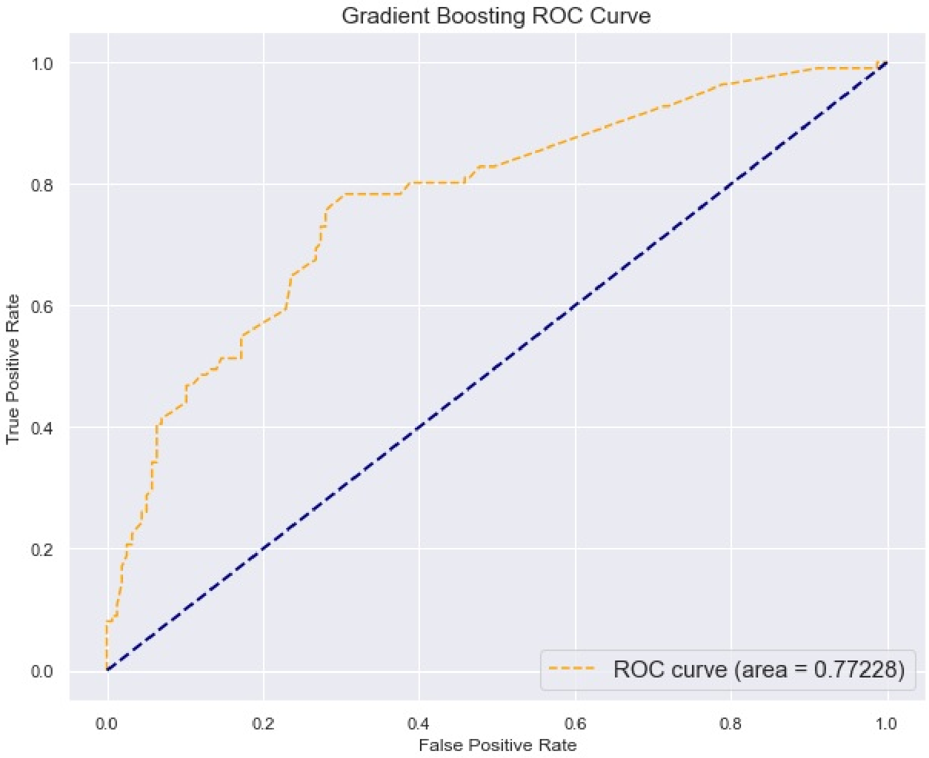
```
Accuracy Score: 0.7052238805970149
```

In [39]:
```
plt.figure(figsize=(8,6))
sns.heatmap(confusion_matrix(y_test, predictions), annot=True, cmap="Blues", fmt='d')
plt.title("Confusion Matrix of Gradient Boosting", size=15)
plt.show()
```



In [40]:

```
visualize_roc_auc_curve(gbc, "Gradient Boosting")
```



In [41]:

```
param_grid_svc = {'gamma': [ 0.001, 0.01, 0.1, 1, 10],
                  'C': [1, 10, 50, 100, 200, 300, 500, 1000]}

grid_svc = GridSearchCV(SVC(), param_grid_svc, scoring="accuracy", cv=5, verbose=0, n_jobs=-1)

grid_svc.fit(X_train, y_train)
```

Out[41]:

```
GridSearchCV(cv=5, estimator=SVC(), n_jobs=-1,
             param_grid={'C': [1, 10, 50, 100, 200, 300, 500, 1000],
                         'gamma': [0.001, 0.01, 0.1, 1, 10]},
             scoring='accuracy')
```

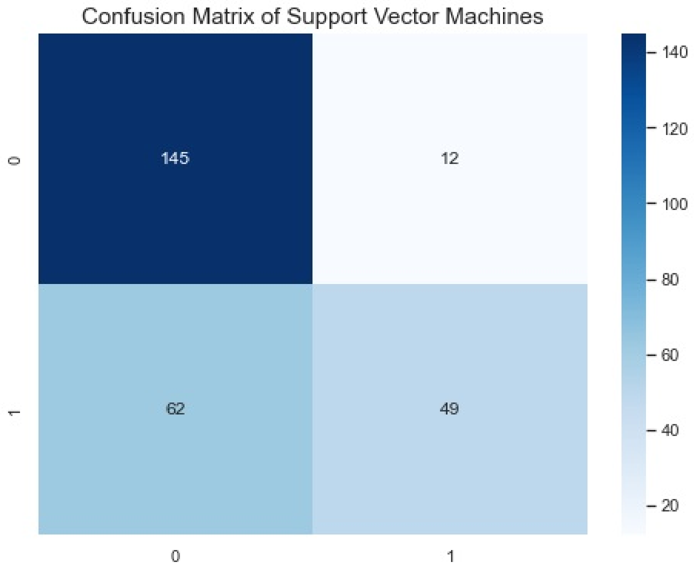
In [42]:

```
svc_params = grid_svc.best_params_
svc = SVC(**svc_params)
svc.fit(X_train, y_train)
predictions = svc.predict(X_test)
score = accuracy_score(y_test, predictions)
print("Accuracy Score:", score)

new_row = {"Model": "SVC", "Accuracy Score": score}
tuned_models = tuned_models.append(new_row, ignore_index=True)
```

```
Accuracy Score: 0.7238805970149254
```

In [43]:

```
plt.figure(figsize=(8,6))
sns.heatmap(confusion_matrix(y_test, predictions), annot=True, cmap="Blues", fmt='d')
plt.title("Confusion Matrix of Support Vector Machines", size=15)
plt.show()
```

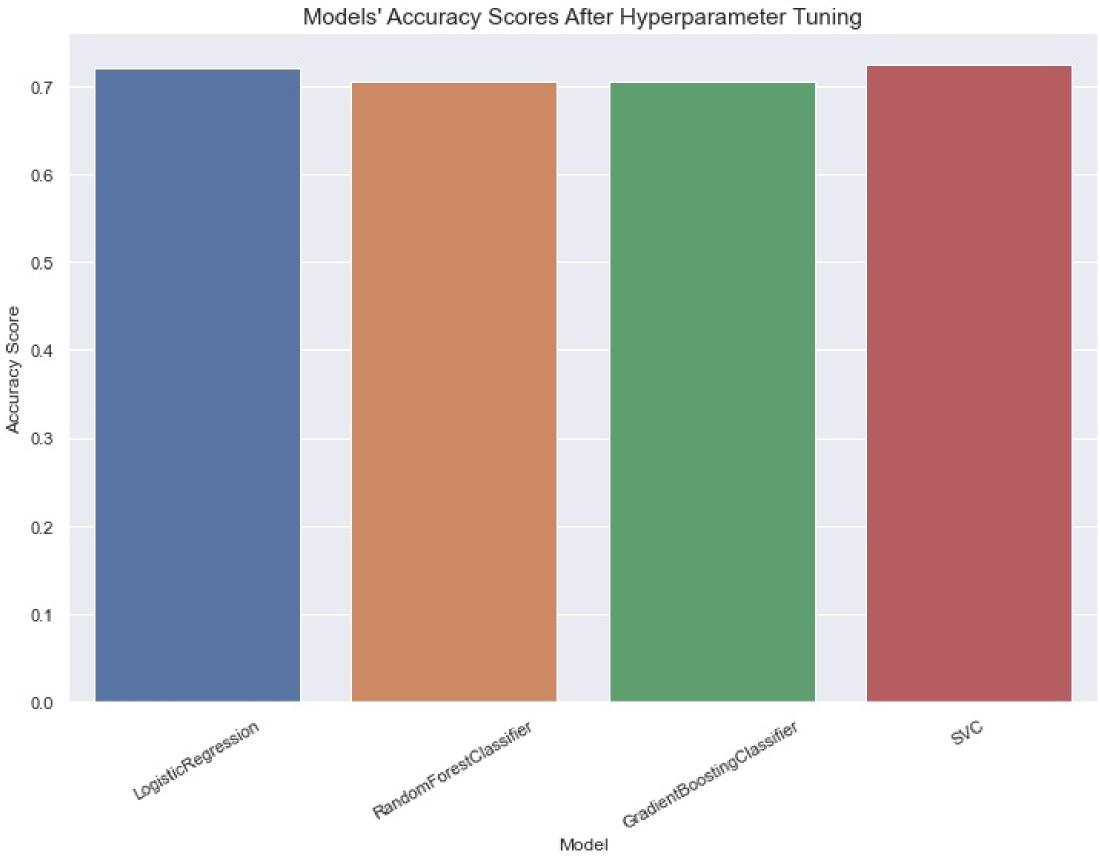

Confusion Matrix of Support Vector Machines

```
In [44]: tuned_models.sort_values(by="Accuracy Score", ascending=False)
```

Out[44]:

	Model	Accuracy Score
3	SVC	0.723881
0	LogisticRegression	0.720149
1	RandomForestClassifier	0.705224
2	GradientBoostingClassifier	0.705224

```
In [45]: plt.figure(figsize=(12, 8))
         sns.barplot(x=tuned_models["Model"], y=tuned_models["Accuracy Score"])
         plt.title("Models' Accuracy Scores After Hyperparameter Tuning", size=15)
         plt.xticks(rotation=30)
         plt.show()
```


Models' Accuracy Scores After Hyperparameter Tuning

```
In [ ]:
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js