1. Data preprocessing (missing values)
2. Exploratory data analysis (extract meaningful insights)
3. Feature engineering
4. Model selection (ML algorithms - logistic regression, decision trees, random forests, gradient boosting will be explored and evaluated to determine most effective model for predicting outcomes)
5. Evaluation

Data Visualisation

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

df = pd.read_csv('titanic_train.csv')
df.head()
```

Out[ ]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tick
0	631	1	1	Barkworth, Mr. Algernon Henry Wilson	male	80.0	0	0	270
1	852	0	3	Svensson, Mr. Johan	male	74.0	0	0	3470
2	97	0	1	Goldschmidt, Mr. George B	male	71.0	0	0	177
3	494	0	1	Artagaveytia, Mr. Ramon	male	71.0	0	0	176
4	117	0	3	Connors, Mr. Patrick	male	70.5	0	0	3703

In [ ]:
```python
df.shape  # Check the shape of the DataFrame (rows, columns)
```

Out[ ]: (891, 12)

In [ ]:
```python
# Removing unnecessary columns
df = df.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1)
```

In [ ]:
```python
df.describe()
```

Out[ ]:

	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.361582	0.523008	0.381594	32.204208
std	0.486592	0.836071	13.019697	1.102743	0.806057	49.693429
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	22.000000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	1.000000	3.000000	35.000000	1.000000	0.000000	31.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

In [ ]:

```
df.dtypes  # Check the data types of each column
```

Out[ ]:

```
Survived      int64
Pclass        int64
Sex          object
Age         float64
SibSp         int64
Parch         int64
Fare        float64
Embarked     object
dtype: object
```

In [ ]:

```
# Checking for unique value count
df.nunique()
```

Out[ ]:

```
Survived      2
Pclass        3
Sex           2
Age          88
SibSp         7
Parch         7
Fare        248
Embarked      3
dtype: int64
```

In [ ]:

```
# Checking for missing values
df.isnull().sum()
```

Out[ ]:

```
Survived    0
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked    2
dtype: int64
```

Data Cleaning

In [ ]:
```
# Data Cleaning : Replacing missing valus with median for age (Right skewed
df['Age'] = df['Age'].replace(np.nan,df['Age'].median(axis=0))

# Data Cleaning : Replacing missing values with mode for Embarked
df['Embarked'] = df['Embarked'].replace(np.nan, 'S')

# Typecasting age to int
df['Age'] = df['Age'].astype(int)

# Replacing 1 for male and 0 for females
df['Sex'] = df['Sex'].apply(lambda x:1 if x == 'male' else 0)
```

In [ ]:
```
# Categorising age in groups
# Infant (0-5), Child (6-20), 20s (21-30), 30s(31-40), 40s (41-50), 50s (51-

df['Age'] = pd.cut(x=df['Age'],
                   bins=[-1, 5, 20, 30, 40, 50, 60, 100],
                   labels = ['Infant', 'Child', '20', '30', '40', '50', 'Ser
                   right = True,
                   include_lowest=True)
```

In [ ]:
```
df.tail(20)
```

Out[ ]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
871	0	1	0	Infant	1	2	151.5500	S
872	1	2	1	Infant	1	1	26.0000	S
873	1	3	0	Infant	0	1	12.2875	S
874	1	2	0	Infant	1	1	26.0000	S
875	0	3	0	Infant	3	2	27.9000	S
876	0	3	1	Infant	4	1	39.6875	S
877	0	3	1	Infant	4	1	39.6875	S
878	1	3	0	Infant	1	1	11.1333	S
879	1	2	1	Infant	2	1	39.0000	S
880	1	3	0	Infant	0	2	15.7417	C
881	0	3	1	Infant	5	2	46.9000	S
882	1	3	1	Infant	1	2	20.5750	S
883	1	2	1	Infant	0	2	37.0042	C
884	1	1	1	Infant	1	2	151.5500	S
885	1	2	1	Infant	0	2	29.0000	S
886	1	2	1	Infant	1	1	18.7500	S
887	1	3	0	Infant	2	1	19.2583	C
888	1	3	0	Infant	2	1	19.2583	C
889	1	2	1	Infant	1	1	14.5000	S
890	1	3	1	Infant	0	1	8.5167	C

In [ ]:

```
df.tail()
```

Out[ ]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
886	1	2	1	Infant	1	1	18.7500	S
887	1	3	0	Infant	2	1	19.2583	C
888	1	3	0	Infant	2	1	19.2583	C
889	1	2	1	Infant	1	1	14.5000	S
890	1	3	1	Infant	0	1	8.5167	C

Exploratory Data Analysis

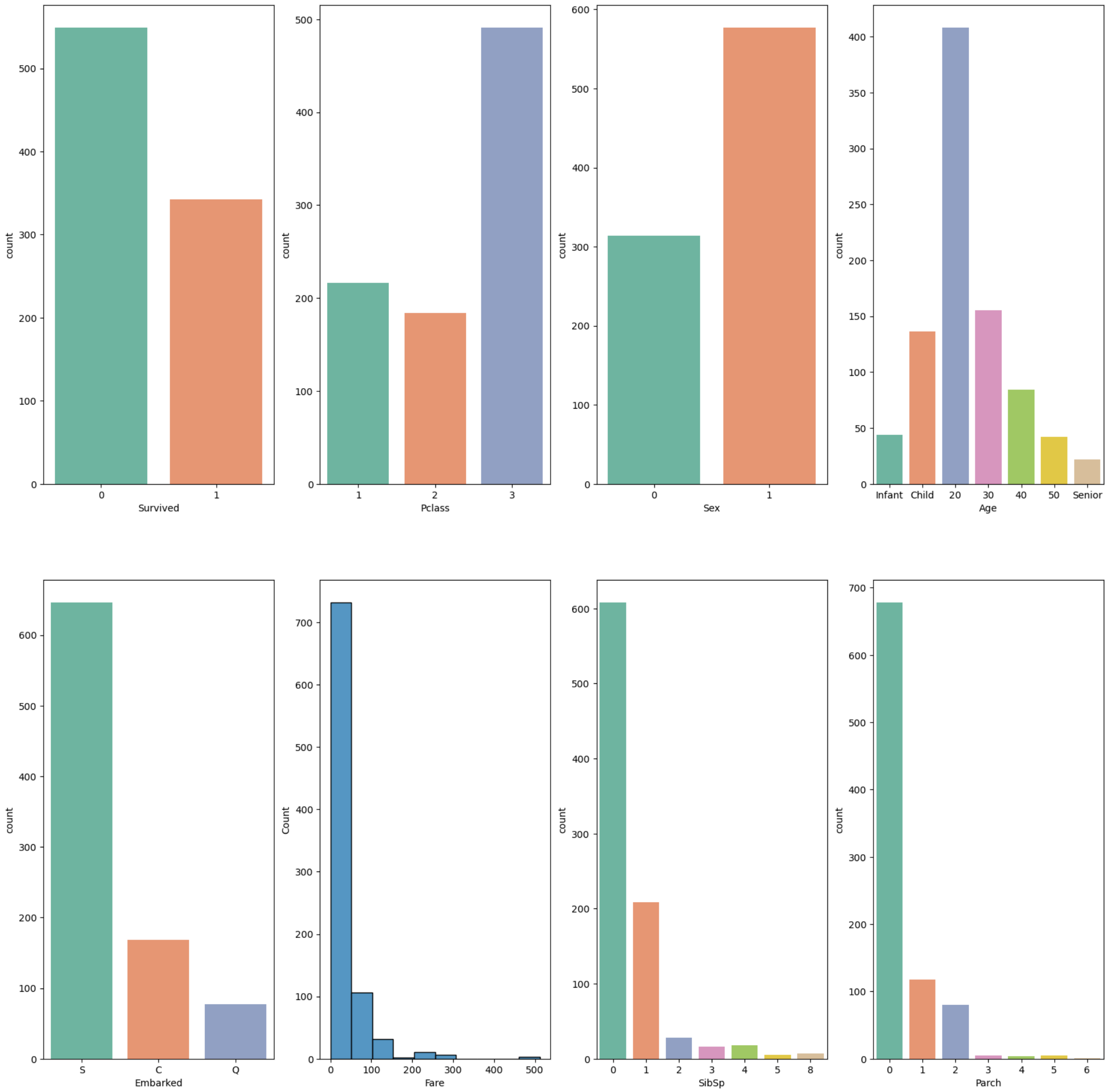
Plotting the countplot to visualize the numbers

In [ ]:
```
# Plotting the cleaned data (count vs each catgegory)
fig, ax = plt.subplots(2, 4, figsize=(20, 20))
sns.countplot(x='Survived', data=df, ax=ax[0, 0], palette='Set2')
sns.countplot(x='Pclass', data=df, ax=ax[0, 1], palette='Set2')
sns.countplot(x='Sex', data=df, ax=ax[0, 2], palette='Set2')
sns.countplot(x='Age', data=df, ax=ax[0, 3],palette='Set2')
sns.countplot(x='Embarked', data=df, ax=ax[1, 0],palette= 'Set2')
sns.histplot(x='Fare', data=df,bins = 10, ax=ax[1, 1], palette='Set2')
sns.countplot(x='SibSp', data=df, ax=ax[1, 2], palette='Set2')
sns.countplot(x='Parch', data=df, ax=ax[1, 3], palette='Set2')

# Parch means number of parents/children aboard
# Sibsp means number of siblings/spouses aboard
# Colors here have no significance, just for visualization, across diff grou
```

```
/var/folders/jn/k87rrm694dq263mh4gqmfd080000gn/T/ipykernel_8962/2506207595.p
y:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed
in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the
same effect.

  sns.countplot(x='Survived', data=df, ax=ax[0, 0], palette='Set2')
/var/folders/jn/k87rrm694dq263mh4gqmfd080000gn/T/ipykernel_8962/2506207595.p
y:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed
in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the
same effect.

  sns.countplot(x='Pclass', data=df, ax=ax[0, 1], palette='Set2')
/var/folders/jn/k87rrm694dq263mh4gqmfd080000gn/T/ipykernel_8962/2506207595.p
y:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed
in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the
same effect.

  sns.countplot(x='Sex', data=df, ax=ax[0, 2], palette='Set2')
/var/folders/jn/k87rrm694dq263mh4gqmfd080000gn/T/ipykernel_8962/2506207595.p
y:6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed
in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the
same effect.

  sns.countplot(x='Age', data=df, ax=ax[0, 3],palette='Set2')
/var/folders/jn/k87rrm694dq263mh4gqmfd080000gn/T/ipykernel_8962/2506207595.p
y:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed
in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the
same effect.

  sns.countplot(x='Embarked', data=df, ax=ax[1, 0],palette= 'Set2')
/var/folders/jn/k87rrm694dq263mh4gqmfd080000gn/T/ipykernel_8962/2506207595.p
y:8: UserWarning: Ignoring `palette` because no `hue` variable has been assi
gned.
  sns.histplot(x='Fare', data=df,bins = 10, ax=ax[1, 1], palette='Set2')
/var/folders/jn/k87rrm694dq263mh4gqmfd080000gn/T/ipykernel_8962/2506207595.p
y:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed
in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the
same effect.

  sns.countplot(x='SibSp', data=df, ax=ax[1, 2], palette='Set2')
/var/folders/jn/k87rrm694dq263mh4gqmfd080000gn/T/ipykernel_8962/2506207595.p
y:10: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed
```

```
in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the
same effect.

  sns.countplot(x='Parch', data=df, ax=ax[1, 3], palette='Set2')
```

Out[ ]: <Axes: xlabel='Parch', ylabel='count'>



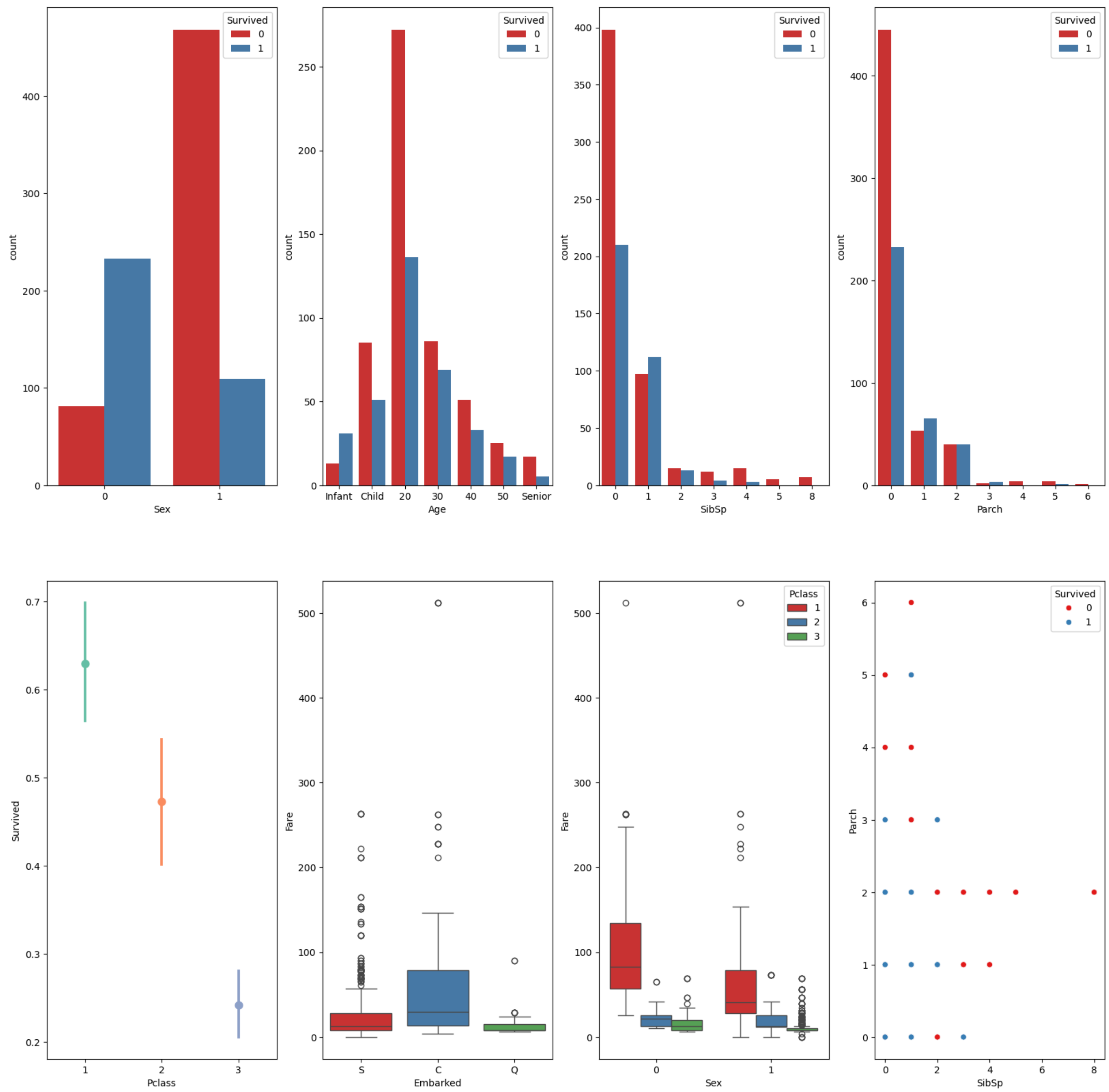
Visualizing the relationship between the features

In [ ]:
```
# Survived is 1, Dead is 0
# We want to see how does survival rate vary with sex, age, sibsp, parch, fa
fig,ax = plt.subplots(2,4,figsize=(20, 20))
sns.countplot(x='Sex', data=df, hue='Survived', ax=ax[0, 0], palette='Set1')
sns.countplot(x='Age', data=df, hue='Survived', ax=ax[0, 1], palette='Set1')
sns.countplot(x='SibSp', data=df, hue='Survived', ax=ax[0, 2], palette='Set1
sns.countplot(x='Parch', data=df, hue='Survived', ax=ax[0, 3], palette='Set1
sns.pointplot(x='Pclass', y = 'Survived', data=df, ax=ax[1, 0], palette='Set
sns.boxplot(x='Embarked', y = 'Fare', data=df, ax=ax[1, 1], palette='Set1')
```

```python
sns.boxplot(x='Sex', y = 'Fare', hue = 'Pclass', data=df, ax=ax[1, 2], palet
sns.scatterplot(x='SibSp', y='Parch', data=df, hue='Survived', ax=ax[1, 3],
```

```
/var/folders/jn/k87rrm694dq263mh4gqmfd080000gn/T/ipykernel_8962/1497080572.p
y:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed
in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the
same effect.

  sns.pointplot(x='Pclass', y = 'Survived', data=df, ax=ax[1, 0], palette='S
et2')
/var/folders/jn/k87rrm694dq263mh4gqmfd080000gn/T/ipykernel_8962/1497080572.p
y:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed
in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the
same effect.

  sns.boxplot(x='Embarked', y = 'Fare', data=df, ax=ax[1, 1], palette='Set
1')
```

Out[ ]:
```
<Axes: xlabel='SibSp', ylabel='Parch'>
```



Data preprocessing

How are is survival rate correlated to each variables?

Are the relationships statistically significant?

In [ ]:

```
from sklearn import preprocessing
# Label encoder converts categorical labels into numbers
le = preprocessing.LabelEncoder()
le.fit(['S', 'C', 'Q']) # [0,1,2]
df['Embarked'] = le.transform(df['Embarked'])
#le.transform.. takes Embarked column and replaces it.
# S = 0, C = 1, Q = 2

# Why do this? ML algorithms work with numbers, not strings.
# Label encoding is a way to convert categorical text data into numerical da
```

In [ ]:

```python
age_mapping = {
    'Infant': 0,
    'Child': 1,
    '20s': 2,
    '30s': 3,
    '40s': 4,
    '50s': 5,
    'Senior': 6}
df['Age'] = df['Age'].map(age_mapping)
df.dropna(subset=['Age'], axis = 0, inplace=True)  # Drop rows where Age is 

# We are using age_mapping dictionary to convert age categories into numeric
# Using exact ages can cause the model to fit too closely to small variation
# Deleting passengers without age data as they are not useful for our analys
# Avoids potential bias from imputation (e.g., filling with mean/median) whi
```

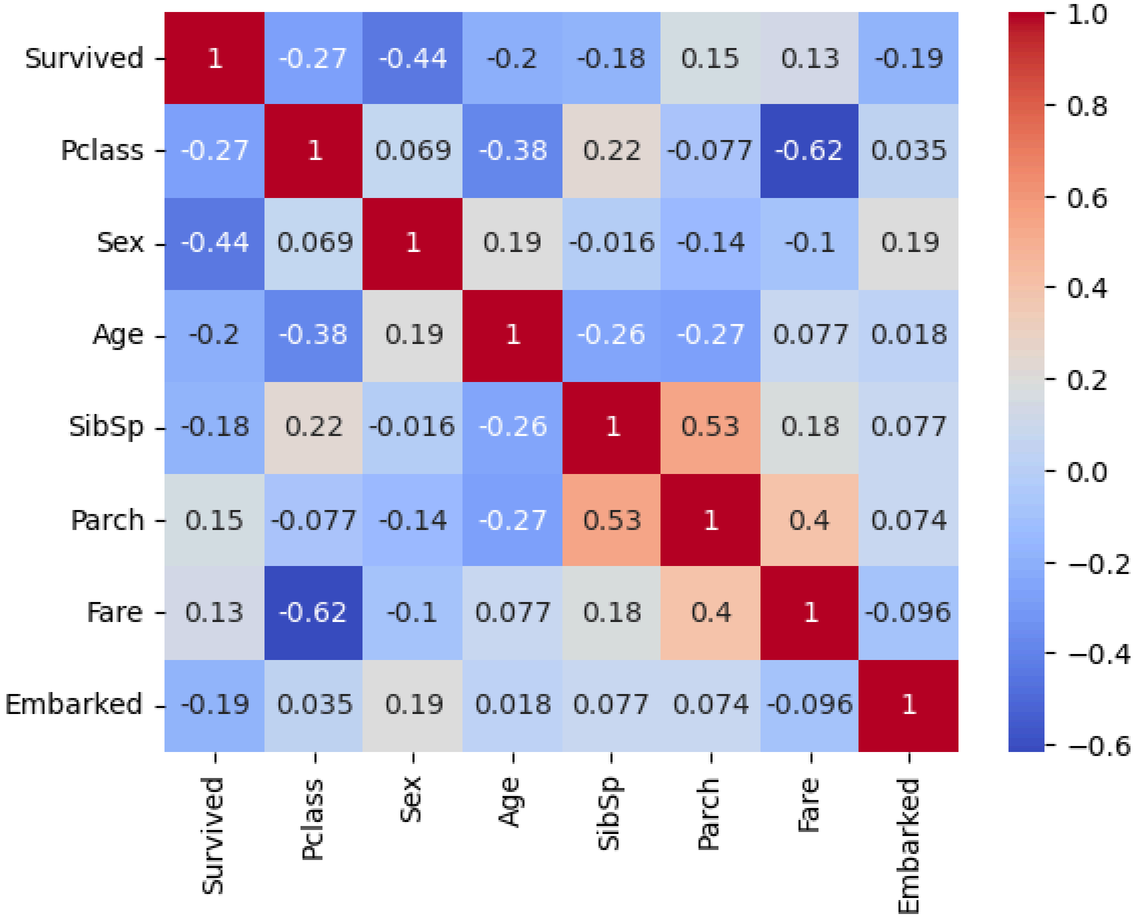
Correlation Heatmap

In [ ]:

```python
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
# Correlation Heatmap shows the strength and direction of the relationship
# between 2 features, at a time.

# To see 3-variable relationships, we can use sns.pairplot (color + size + a
# Or multivariate
```

Out[ ]:

```
<Axes: >
```



	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
Survived	1	-0.27	-0.44	-0.2	-0.18	0.15	0.13	-0.19
Pclass	-0.27	1	0.069	-0.38	0.22	-0.077	-0.62	0.035
Sex	-0.44	0.069	1	0.19	-0.016	-0.14	-0.1	0.19
Age	-0.2	-0.38	0.19	1	-0.26	-0.27	0.077	0.018
SibSp	-0.18	0.22	-0.016	-0.26	1	0.53	0.18	0.077
Parch	0.15	-0.077	-0.14	-0.27	0.53	1	0.4	0.074
Fare	0.13	-0.62	-0.1	0.077	0.18	0.4	1	-0.096
Embarked	-0.19	0.035	0.19	0.018	0.077	0.074	-0.096	1

Logistic Regression

- Testing for statistical significance with p-values.
- If $p < 0.05$, the variable is statistically significant in predicting/affecting survival rates.
- LR estimates the independent effect of each variable while controlling for others.
- Good for multivariate analysis (when survival rate is affected by sex, age, etc tgt at a time)
- Simple stats test (chi sq, t-test, mann whitney, anova) tests the rship for 2 variables at a time :(
- Doesn't account for other factors that might influence the rship.

In [ ]:

```
# Logistics Regression Significance Testing (p-values)
# We can use statsmodels to perform logistic regression and get p-values for
import statsmodels.api as sm

X = df[['Age', 'Fare', 'Pclass', 'Sex', 'Embarked', 'SibSp', 'Parch']]  # ex
```

```
X = sm.add_constant(X)  # intercept
y = df['Survived']

model = sm.Logit(y, X).fit()
print(model.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.454583
         Iterations 7
                           Logit Regression Results
=============================================================================
==
Dep. Variable:               Survived   No. Observations:                   2
02
Model:                          Logit   Df Residuals:                       1
94
Method:                           MLE   Df Model:
7
Date:                Fri, 25 Jul 2025   Pseudo R-squ.:                   0.33
49
Time:                        07:43:05   Log-Likelihood:                 -91.8
26
converged:                       True   LL-Null:                        -138.
07
Covariance Type:            nonrobust   LLR p-value:                 3.815e-
17
=============================================================================
==
                 coef    std err          z      P>|z|      [0.025      0.97
5]
-----------------------------------------------------------------------------
--
const          6.6800      1.393      4.796      0.000       3.950       9.4
10
Age           -0.5373      0.147     -3.663      0.000      -0.825      -0.2
50
Fare          -0.0144      0.007     -2.176      0.030      -0.027      -0.0
01
Pclass        -1.6528      0.410     -4.034      0.000      -2.456      -0.8
50
Sex           -1.8945      0.400     -4.739      0.000      -2.678      -1.1
11
Embarked      -0.4250      0.240     -1.774      0.076      -0.894       0.0
44
SibSp         -0.6061      0.203     -2.988      0.003      -1.004      -0.2
08
Parch          0.8420      0.314      2.682      0.007       0.227       1.4
57
=============================================================================
==
```

Explanation of Results

Are the variables relationship with survival rates statistically significant?

Logistic regression

- Multivariate model
- Controls for other predictors in the model
- Different coefficient values than corelation heatmap
- "How much does this variable affect survival when all other variables are considered too?"

Correlation heatmap

- Does not control for other variables
- Shows pairwise correlation
- Quick way to determine if there is a linear rship.
- "How much does this variable affect survival, on its own?"

The strength of relationship is in order, beginning with sex being the strongest.

1. Sex

- Most significant predictor. Highest negative coef.
- Survival rates increase for females. Males less likely to survive.

2. PClass

- Significant predictor (-ve coeff)
- Survival rate is higher for lower classes (1 = first class, 2=2nd, 3 = 3rd class)

3. Age

- Significant predictor. (-ve coeff)
- Older passengers less likely to survive.

4. SibSp

- Significant predictor.
- More siblings

Fare, Parch, Embarked

- Not statisically significant. Fare doesn't strongly predict survival
- p>0.05 (p>|z|)

Const

- Intercept term.
- Baseline log-odds of survival when others = 0 (die)

Things to note

- Non robust covariance means SE are calculated with usual assumptions (homoscedasticity - constant variance of errors)
- If data is not independent, one person's survival affects another, then the usual logistic regression assumptions are violated, affecting reliability of SE, CI, p-value inferences.
- Could lead to correlation between residuals, underestimated SE, p-values too optimistic (false positive)
- How then to handle dependent data?
- Cluster by group to adjust for SE, accounting for within group correlation.
- Identify clusters by family ID.
- But this is impossible to do with our data. We do not know for sure who were travelling as a group.
- Another factor : multicollinearity
- When 2 or more predictor variables in a regression model are highly correlated with each other.
- It makes it hard to separate out the indiv effect of each predictor on the response variable
- Regression coefficients become unstable and SE get inflated.
- This leads to large changes in coefficients, if u change ur data
- Insignificant p-values
- To detect multicollinearity, calculate coorelation matrix for predictors
- Use Variance Inflation Factor (> 5 or 10 indicates problematic multicollinearity)
- To fix MC, collect more data, remove or combine correlated predictors, use dimensionality reduction methods (PCA), regularization mtds (Ridge regression)

In [ ]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Assume X is your dataframe of predictors
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.s

print(vif_data)
```

```
    feature        VIF
0     const  39.925694
1       Age   1.378914
2      Fare   2.169132
3    Pclass   2.295560
4       Sex   1.115726
5  Embarked   1.076454
6     SibSp   1.618036
7     Parch   1.725051
```

Explanation of VIF results

- VIF for const (intercept) is high but that's normal and usually ignored
- VIFs are low for other predictors (~1)
- No serious multicollinearity issues amongst predictors
- Variables are not strongly correlated with each other :)
- Regression coefficientsand p-values are unlikely to be distorted due to multicollinearity.

Sklearn is for prediction with Machine Learning.

Stats model is for viewing if rship is statistically significant.

Key differences:

Aspect	`sklearn.linear_model.LogisticRegression`	`statsmodels.api.Logit`
Purpose	Mainly for prediction and machine learning workflows	Mainly for statistical inference and hypothesis testing
Output	Focus on prediction accuracy, provides `.predict()`, `.score()`, etc.	Detailed statistical output: p-values, confidence intervals, model diagnostics
P-values / Significance	Does *not* provide p-values or significance tests natively	Provides p-values, standard errors, and full regression summaries
API style	Fits into sklearn ecosystem (pipelines, cross-validation)	Statsmodels uses a formula or arrays and offers rich statistical details
Handling categorical variables	Usually requires preprocessing (e.g., one-hot encoding)	Can handle categorical variables with formulas
Optimization / solvers	Multiple solvers, regularization by default	Typically uses maximum likelihood estimation without regularization

Machine Learning - Model Training

Separating the target and independent variable

1. To test for statistical significance between survival rates and other variables.
2. Model Training for prediction.

In [ ]:
```
y = df['Survived']
x = df.drop(columns=['Survived'])
```

1. Logistic regression

In [ ]:
```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(max_iter=150)  # Increased max_iter for convergence
lr
# LR is not fitted yet. We have created the model, but not trained it yet
# (to fit on any data).
```

```
# In scikit-learn, we first create the model, then fit it to the data.
# Then, we can use model to predict outcomes.
# Fitting the model learns the rship between X and Y.
```

Out[ ]:

```
▾ LogisticRegression
LogisticRegression(max_iter=150)
```

In [ ]:

```
lr.fit(x,y)
lr.score(x,y)

# Predictions by this model is 80.54% accurate.
```

Out[ ]: 0.801980198019802

2. Decision Tree Classifier

In [ ]:

```
# The model has learned the relationship between the features (x) and the ta
# Decision Tree Classifier is a supervised learning algorithm used for class
# It builds a model in the form of a tree structure, where each internal noc
# (or attribute), each branch represents a decision rule,
# and each leaf node represents an outcome (class label).
# The model is trained by splitting the data into subsets based on the featu
# and it continues to split until a stopping criterion is met
# (e.g., maximum depth of the tree, minimum number of samples in a leaf node
# The goal is to create a model that predicts the target variable (in this c
# based on the input features (like age, fare, class)

from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier()
dtree
```

Out[ ]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [ ]:

```
dtree.fit(x,y)
dtree.score(x,y)
# The Decision Tree Classifier has been trained on the data and is now ready
# The score indicates the accuracy of the model on the training data.
# The model has learned the relationships between the features and the targe
# The score is the proportion of correct predictions made by the model on th
# A higher score indicates better performance, but it is important to evalua
# to ensure it generalizes well.
# The model has learned the relationships between the features and the targe
# The score is the proportion of correct predictions made by the model on th
# A higher score indicates better performance, but it is important to evalua
# to ensure it generalizes well.
```

Out[ ]: 0.9653465346534653

3. Support Vector Machine (SVM)

In [ ]:
```
from sklearn.svm import SVC
svm = SVC()
svm
```

Out[ ]:
```
▾ SVC
SVC()
```

In [ ]:
```
svm.fit(x,y)
svm.score(x,y)
# The Support Vector Machine (SVM) model has been trained on the data and is
# The score indicates the accuracy of the model on the training data.
```

Out[ ]:
```
0.6138613861386139
```

4. K-Nearest Neighbour (KNN)

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn
```

Out[ ]:
```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

In [ ]:
```
knn.fit(x, y)
knn.score(x, y)
```

Out[ ]:
```
0.8267326732673267
```

Conclusions = Decision Tree Classifier

From the above four models, Decision Tree Classifier has the highest training accuracy.

So only Decision Tree Classifier will work on the test set

Importing the test set

In [ ]:
```
df2 = pd.read_csv('titanic_test.csv')
df2.head()
```

Out[ ]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tic
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	21
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/ 3101
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373

Data Cleaning the Test Set

In [ ]:

```
# Removing unnecessary columns from the test set
df2 = df2.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1)
```

In [ ]:

```
# Replacing missing values in the test set with MEDIAN.
df2['Age'] = df2['Age'].replace(np.nan, df2['Age'].median(axis=0))
df2['Embarked'] = df2['Embarked'].replace(np.nan, 'S')
```

In [ ]:

```
# Typecasting age to int
df2['Age'] = df2['Age'].astype(int)
```

In [ ]:

```
# Replacing 1 for male and 0 for female
df2['Sex'] = df2['Sex'].apply(lambda x: 1 if x == 'male' else 0)
```

In [ ]:

```
## Categorising age in groups
df2['Age'] = pd.cut(x=df2['Age'],
                     bins=[0, 5, 20, 30, 40, 50, 60, 100], labels=['0', '1',
```

In [ ]:

```
le.fit(['S','C','Q'])
df2['Embarked'] = le.transform(df2['Embarked'])
# ML algo can only work with numbers, not strings.
```

In [ ]:

```
# Removing all NA values.
df2.dropna(subset=['Age'], axis = 0, inplace = True)
```

In [ ]:
```
df2.head()
```

Out[ ]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	1	2	1	0	7.2500	2
1	1	1	0	3	1	0	71.2833	0
2	1	3	0	2	0	0	7.9250	2
3	1	1	0	3	1	0	53.1000	2
4	0	3	1	3	0	0	8.0500	2

Separating the target and independent variable

In [ ]:
```
x = df2.drop(columns = ['Survived'])
y = df2['Survived']
```

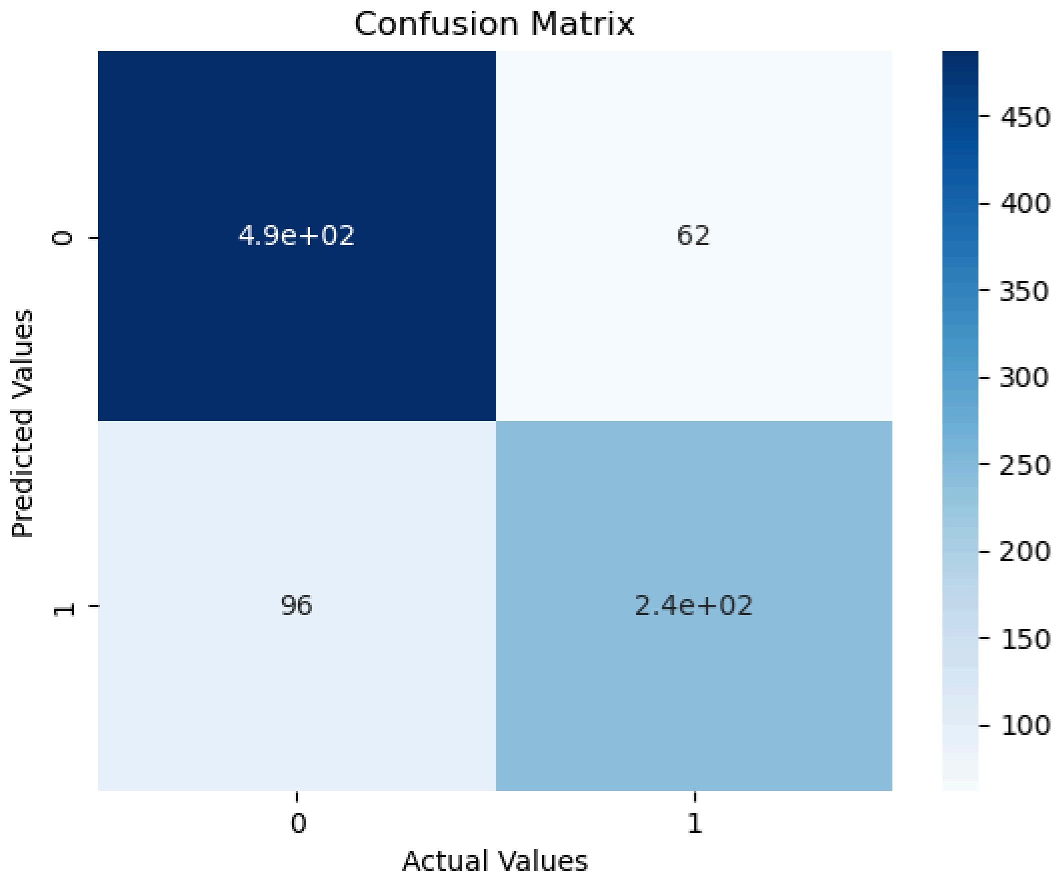
Predicting using Decision Tree Classifier

In [ ]:
```
tree_pred = dtree.predict(x)

from sklearn.metrics import accuracy_score
accuracy_score(y,tree_pred)
```

Out[ ]:
```
0.8212669683257918
```

Confusion Matrix

In [ ]:
```
from sklearn.metrics import confusion_matrix
sns.heatmap(confusion_matrix(y,tree_pred), annot = True, cmap = 'Blues')
plt.ylabel('Predicted Values')
plt.xlabel('Actual Values')
plt.title('Confusion Matrix')
plt.show()

# Confusion matrix is a performance measurement tool for classicification mo
# Shows how well the model's prediction match the actual label.
```



Conclusions of Model Predictions

Accuracy score = 82%

- ✅ 480 people who actually died were correctly predicted (True Negative)
- ✅ 240 people who actually survived were correctly predicted (True Positive)
- ❌ 66 people who actually survived were wrongly predicted as dead (False Negative)
- ❌ 93 people who actually died were wrongly predicted as alive (False Positive)