In this tutorial, we will do a refresher on some python basics and popular libraries used in data science. Some of you are probably familiar with these, but we want to make sure everyone is on the same page.

You can open this file on Colab at the following link:
https://colab.research.google.com/drive/1p7jZ6sXRwAlGQOulK8BJEedh1ODAnvFi

# Introduction to Colab

Colab is very similar to the Jupyter Notebook that you run on a local machine. Colab runs with resources in the cloud provided and preconfigured by Google, but you can configure some of these resources for your instance depending on your needs.

Similar to Google Docs for word documents, you can invite and collaborate with others using Colab as it supports realtime updates. However, Colab can also yield merge conflicts if multiple people make updates simultaneously, so be careful!

# Configuring your environment

Colab comes with a set of preinstalled libraries. These may be subject to change by Google. You can import these popular data science libraries without installing them:

- **Pandas** - useful for data cleaning, manipulation, and analysis.
- **Scikitlearn** - comes with a set of useful machine learning tools and provides a set of standard ml algorithms.
- **Matplotlib** - one of the most popular library for plotting.

which we will be using for this tutorial.

To see the current list of installed libraries, make a new cell (using the "+ Code" button in the top left of your browser) and run the following command in the cell:

> !pip list

However, if you wish to use libraries that are not included, you can use the following command:

> !pip install [library]

We will first import all the necessary libraries to get started:

Note that your session times out on idle or upon closing the tab, you would have to reinstall them when the session times out and rerun all the cells.

In [ ]:
```
# use for data processing
import pandas as pd  # import pandas with alias pd

# use for training the model
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler, OneHotEncoder

# use for evaluation
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_s

# use for plotting the results
import matplotlib.pyplot as plt
```

# Data Processing

There are several ways to import the data. But, Colab provides direct support for importing the data files from your google drive. You can upload your data to Google Drive and access the files by mounting your drive.

We'll be using the popular Titanic dataset as an example - for more info, please check https://www.kaggle.com/competitions/titanic.

In [ ]:
```
# loading from google drive
#from google.colab import drive
#drive.mount('/content/drive/')
#train_data = pd.read_csv('/content/drive/My Drive/titanic_dataset/train.csv')
#test_data = pd.read_csv('/content/drive/My Drive/titanic_dataset/test.csv')

# loading from public repo
url = 'https://raw.githubusercontent.com/datasciencedojo/datasets/master/titan
titanic_data = pd.read_csv(url)
display(titanic_data)
```

| | PassengerId | Survived | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 0 | 3 | Braund, Mr. Owen Harris | male | 22.0 | 1 | 0 | A/5 21171 | 7.2500 |
| **1** | 2 | 1 | 1 | Cumings, Mrs. John Bradley (Florence Briggs Th... | female | 38.0 | 1 | 0 | PC 17599 | 71.2833 |
| **2** | 3 | 1 | 3 | Heikkinen, Miss. Laina | female | 26.0 | 0 | 0 | STON/O2. 3101282 | 7.9250 |
| **3** | 4 | 1 | 1 | Futrelle, Mrs. Jacques Heath (Lily May Peel) | female | 35.0 | 1 | 0 | 113803 | 53.1000 |
| **4** | 5 | 0 | 3 | Allen, Mr. William Henry | male | 35.0 | 0 | 0 | 373450 | 8.0500 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **886** | 887 | 0 | 2 | Montvila, Rev. Juozas | male | 27.0 | 0 | 0 | 211536 | 13.0000 |
| **887** | 888 | 1 | 1 | Graham, Miss. Margaret Edith | female | 19.0 | 0 | 0 | 112053 | 30.0000 |
| **888** | 889 | 0 | 3 | Johnston, Miss. Catherine Helen "Carrie" | female | NaN | 1 | 2 | W./C. 6607 | 23.4500 |
| **889** | 890 | 1 | 1 | Behr, Mr. Karl Howell | male | 26.0 | 0 | 0 | 111369 | 30.0000 |
| **890** | 891 | 0 | 3 | Dooley, Mr. Patrick | male | 32.0 | 0 | 0 | 370376 | 7.7500 |

891 rows × 12 columns

If you are using the Kaggle files, you will be provided with train.csv, test.csv, and gender_submission.csv, we will only be using train.csv as test.csv does not have the actual labels, they are hidden by Kaggle until submission. We will split (7:3) the titanic training data into train and test data for this tutorial.

In [ ]:
```
train_data, test_data = train_test_split(titanic_data, test_size=0.3, random_s
```

Pandas provides many useful functions to work with your data, for example you can find out the shape of the data you will be working with. This is really helpful for sanity check.

Use some of the functions below to understand your data and transform your data in preparation for training.

In [ ]:
```
print(train_data.shape)
print(test_data.shape)
```

```
(623, 12)
(268, 12)
```

You can also view the data as tables.

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

Out[ ]:

| | PassengerId | Survived | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare |
|---|---|---|---|---|---|---|---|---|---|---|
| **445** | 446 | 1 | 1 | Dodge, Master. Washington | male | 4.0 | 0 | 2 | 33638 | 81.8583 |
| **650** | 651 | 0 | 3 | Mitkoff, Mr. Mito | male | NaN | 0 | 0 | 349221 | 7.8958 |
| **172** | 173 | 1 | 3 | Johnson, Miss. Eleanor Ileen | female | 1.0 | 1 | 1 | 347742 | 11.1333 |
| **450** | 451 | 0 | 2 | West, Mr. Edwy Arthur | male | 36.0 | 1 | 2 | C.A. 34651 | 27.7500 |
| **314** | 315 | 0 | 2 | Hart, Mr. Benjamin | male | 43.0 | 1 | 1 | F.C.C. 13529 | 26.2500 |

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

Out[ ]:

| | PassengerId | Survived | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fa |
|---|---|---|---|---|---|---|---|---|---|---|
| **709** | 710 | 1 | 3 | Moubarek, Master. Halim Gonios ("William George") | male | NaN | 1 | 1 | 2661 | 15.24 |
| **439** | 440 | 0 | 2 | Kvillner, Mr. Johan Henrik Johannesson | male | 31.0 | 0 | 0 | C.A. 18723 | 10.50 |
| **840** | 841 | 0 | 3 | Alhomaki, Mr. Ilmari Rudolf | male | 20.0 | 0 | 0 | SOTON/O2 3101287 | 7.92 |
| **720** | 721 | 1 | 2 | Harper, Miss. Annie Jessie "Nina" | female | 6.0 | 0 | 1 | 248727 | 33.00 |
| **39** | 40 | 1 | 3 | Nicola-Yarred, Miss. Jamila | female | 14.0 | 1 | 0 | 2651 | 11.24 |

Information provided by Kaggle about the dataset

| Variable | Definition | Key |
|---|---|---|
| survival | Survival | 0 = No, 1 = Yes |
| pclass | Ticket class | 1 = 1st, 2 = 2nd, 3 = 3rd |
| sex | Sex | |
| Age | Age in years | |
| sibsp | # of siblings / spouses aboard the Titanic | |
| parch | # of parents / children aboard the Titanic | |
| ticket | Ticket number | |
| fare | Passenger fare | |
| cabin | Cabin number | |
| embarked | Port of Embarkation | C = Cherbourg, Q = Queenstown, S = Southampton |

Note that Age, for example, has many missing values. And, Cabin has mostly NaN values. Given that most features in Cabin are NaN, we might want to drop this column. Let's validate this now:

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

Out[ ]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

It seems that both passsengers with null values in Embarked survived.

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

Out[ ]:

| | PassengerId | Survived | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare | Cabin |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **61** | 62 | 1 | 1 | Icard, Miss. Amelie | female | 38.0 | 0 | 0 | 113572 | 80.0 | B28 |
| **829** | 830 | 1 | 1 | Stone, Mrs. George Nelson (Martha Evelyn) | female | 62.0 | 0 | 0 | 113572 | 80.0 | B28 |

These are probably not useful. Let's just drop the two rows in Embarked with null values.

In [ ]:
```
titanic_data.dropna(subset=['Embarked'], inplace=True)

titanic_data.isnull().sum()
```

Out[ ]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         0
dtype: int64
```

describe() provides some useful information about your dataset, it may help you decide which features you want to use.

In [ ]:
```
print(titanic_data.describe())
```

```
       PassengerId    Survived      Pclass         Age       SibSp  \
count   889.000000  889.000000  889.000000  712.000000  889.000000
mean    446.000000    0.382452    2.311586   29.642093    0.524184
std     256.998173    0.486260    0.834700   14.492933    1.103705
min       1.000000    0.000000    1.000000    0.420000    0.000000
25%     224.000000    0.000000    2.000000   20.000000    0.000000
50%     446.000000    0.000000    3.000000   28.000000    0.000000
75%     668.000000    1.000000    3.000000   38.000000    1.000000
max     891.000000    1.000000    3.000000   80.000000    8.000000

            Parch        Fare
count  889.000000  889.000000
mean     0.382452   32.096681
std      0.806761   49.697504
min      0.000000    0.000000
25%      0.000000    7.895800
50%      0.000000   14.454200
75%      0.000000   31.000000
max      6.000000  512.329200
```

Let's explore more about the Gender column by doing a group-by aggregation that counts the number of passengers who survived, broken down by gender. There could be a clear imblance.

In [ ]:
```
titanic_data.groupby('Sex')['Survived'].sum()
```

Out[ ]:
```
Sex
female    231
male      109
Name: Survived, dtype: int64
```

In [ ]:
```
titanic_data.groupby('Sex').size()
```

Out[ ]:
```
Sex
female    312
male      577
dtype: int64
```

It seems that a much larger proportion of women survived despite there being more men than women on board.

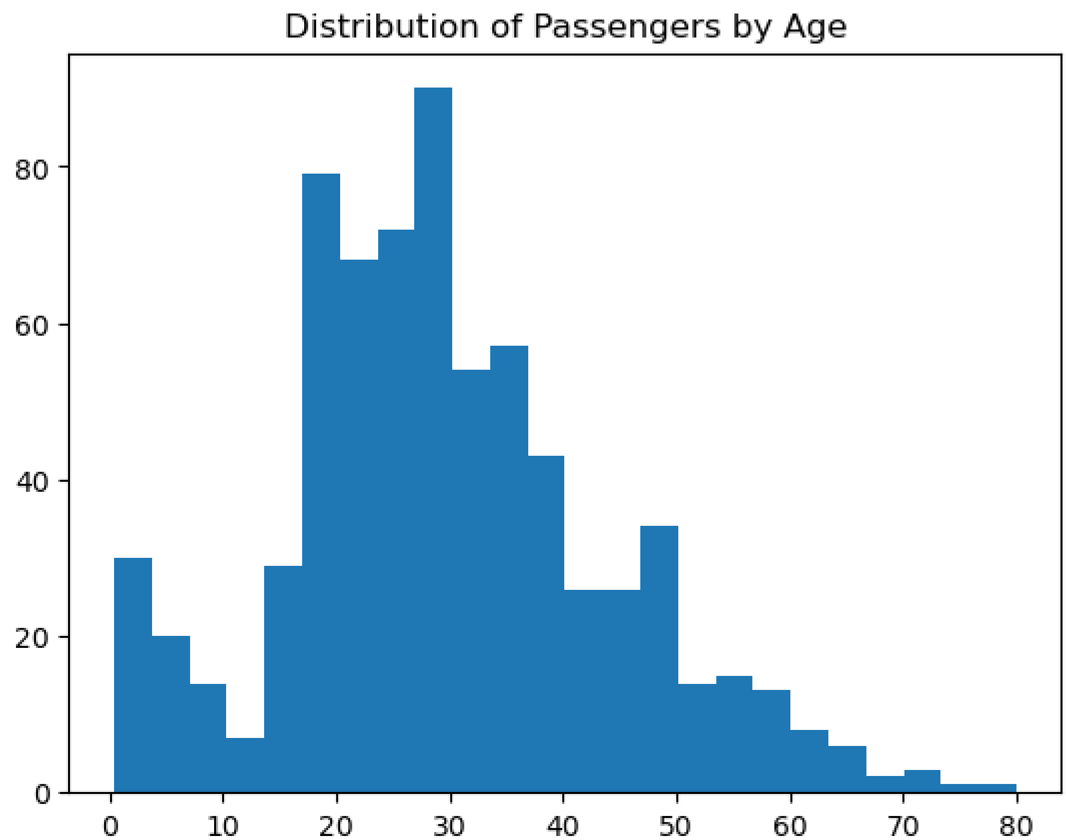
Maybe we'd like to explore the connection between Age and Fare (maybe older people paid more expensive fares?). Let's plot them out to see how they are distributed and to examine their patterns and relationships.

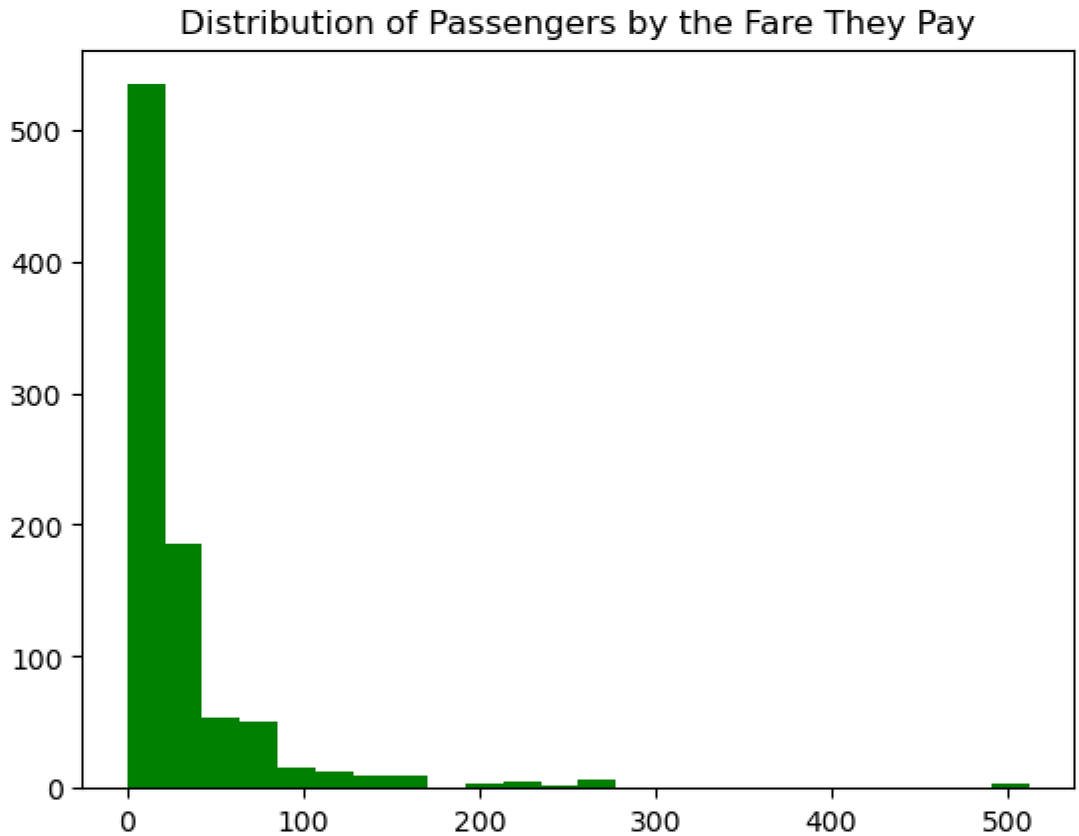
You can plot them out with the `matplotlib` library or directly with `pandas` since its plotting functions are built directly on top of `matplotlib`. But if you want more fine-grained customization with functional support, you can use `matplotlib` directly (or more advanced packages for plotting, like `seaborn`).

In [ ]:
```
age_data, fare_data = titanic_data['Age'], titanic_data['Fare']
```

In [ ]:
```
#plot with pandas directly
#titanic_data['Age'].plot(kind='hist', bins=24)
#plt.show()
```

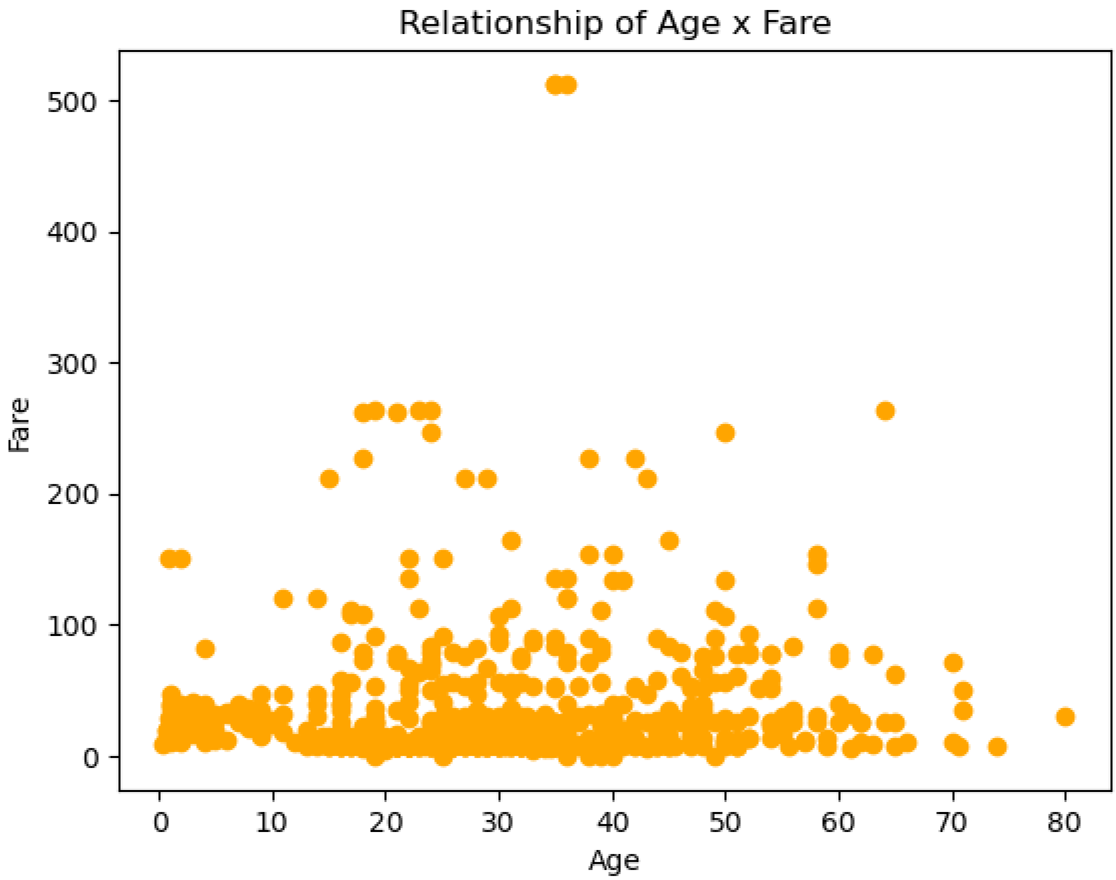
```
plt.title('Distribution of Passengers by Age')
plt.hist(age_data, bins=24)
plt.show()

plt.title('Distribution of Passengers by the Fare They Pay')
plt.hist(fare_data, bins=24, color='green')
plt.show()
```






Looks like most people are young adults with some children and infants onboard. A couple of individuals paid for significantly higher fares, probably VIP tickets. Then it's reasonable to wonder wether the "children and women first" rule or class statuses will impact their chance of survival. These features should be useful.

In [ ]:
```
plt.xlabel('Age')
plt.ylabel('Fare')
plt.title('Relationship of Age x Fare')
plt.scatter(age_data, fare_data, color='orange')
plt.show()
```




There doesn't seem to be any apparent relationships between Age and Fare, but two individuals in their 30s paid significantly more (Maybe it's Rose and her fiance with the VIP tickets...).

Here, we can drop columns like PassengerId, Name, Ticket Number, and Cabin, as they won't provide any useful information as features.

In [ ]:
```
train_labels = train_data["Survived"]

desired_features = ["Pclass", "Sex", "SibSp", "Parch", "Age", "Fare", "Embarke
train_features = train_data[desired_features]
display(train_features)

print(train_features.shape)
print(train_labels.shape)
```

| | Pclass | Sex | SibSp | Parch | Age | Fare | Embarked |
|---|---|---|---|---|---|---|---|
| **445** | 1 | male | 0 | 2 | 4.0 | 81.8583 | S |
| **650** | 3 | male | 0 | 0 | NaN | 7.8958 | S |
| **172** | 3 | female | 1 | 1 | 1.0 | 11.1333 | S |
| **450** | 2 | male | 1 | 2 | 36.0 | 27.7500 | S |
| **314** | 2 | male | 1 | 1 | 43.0 | 26.2500 | S |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **106** | 3 | female | 0 | 0 | 21.0 | 7.6500 | S |
| **270** | 1 | male | 0 | 0 | NaN | 31.0000 | S |
| **860** | 3 | male | 2 | 0 | 41.0 | 14.1083 | S |
| **435** | 1 | female | 1 | 2 | 14.0 | 120.0000 | S |
| **102** | 1 | male | 0 | 1 | 21.0 | 77.2875 | S |

623 rows × 7 columns

```
(623, 7)
(623,)
```

Age also has many missing values. For now, we can impute the missing values by replacing the NaNs with the average age value.

In [ ]:
```
train_data["Age"].fillna(train_data["Age"].mean(), inplace = True)
```

Some algorithms (such as SVM) are sensitive to the scale of the feature, so we may want to normalize values such as Age and Fare (the Fare could be much larger than Age!).

In [ ]:
```
train_data[["Age", "Fare"]]
```

Out[ ]:

| | Age | Fare |
|---|---|---|
| **445** | 4.000000 | 81.8583 |
| **650** | 29.256353 | 7.8958 |
| **172** | 1.000000 | 11.1333 |
| **450** | 36.000000 | 27.7500 |
| **314** | 43.000000 | 26.2500 |
| ... | ... | ... |
| **106** | 21.000000 | 7.6500 |
| **270** | 29.256353 | 31.0000 |
| **860** | 41.000000 | 14.1083 |
| **435** | 14.000000 | 120.0000 |
| **102** | 21.000000 | 77.2875 |

623 rows × 2 columns

In [ ]:
```
train_data[["Age", "Fare"]] = StandardScaler().fit_transform(train_data[["Age"

train_data.head()
```

Out[ ]:

| | PassengerId | Survived | Pclass | Name | Sex | Age | SibSp | Parch | Ticket |
|---|---|---|---|---|---|---|---|---|---|
| **445** | 446 | 1 | 1 | Dodge, Master. Washington | male | -1.940356e+00 | 0 | 2 | 33638 |
| **650** | 651 | 0 | 3 | Mitkoff, Mr. Mito | male | 2.729423e-16 | 0 | 0 | 349221 |
| **172** | 173 | 1 | 3 | Johnson, Miss. Eleanor Ileen | female | -2.170835e+00 | 1 | 1 | 347742 |
| **450** | 451 | 0 | 2 | West, Mr. Edwy Arthur | male | 5.180904e-01 | 1 | 2 | C.A. 34651 |
| **314** | 315 | 0 | 2 | Hart, Mr. Benjamin | male | 1.055876e+00 | 1 | 1 | F.C.C. 13529 |

We can also turn the data in the Sex and Embarked columns into "one-hot encoded features" (i.e., binary dummies) using the pandas.get_dummies() function. Note that pandas knows to only do this for the non-numeric columns within `desired_features`. We apply the parameter drop_first to drop a category from each encoded features to be used as a reference class (given that Sex_male and Sex_female, for example, are collinear and provide similar information).

In [ ]:
```
train_features = pd.get_dummies(train_data[desired_features], drop_first=True)

train_features.head()
```

Out[ ]:

| | Pclass | SibSp | Parch | Age | Fare | Sex_male | Embarked_Q | Embarked_S |
|---|---|---|---|---|---|---|---|---|
| **445** | 1 | 0 | 2 | -1.940356e+00 | 0.980998 | True | False | True |
| **650** | 3 | 0 | 0 | 2.729423e-16 | -0.469634 | True | False | True |
| **172** | 3 | 1 | 1 | -2.170835e+00 | -0.406136 | False | False | True |
| **450** | 2 | 1 | 2 | 5.180904e-01 | -0.080232 | True | False | True |
| **314** | 2 | 1 | 1 | 1.055876e+00 | -0.109651 | True | False | True |

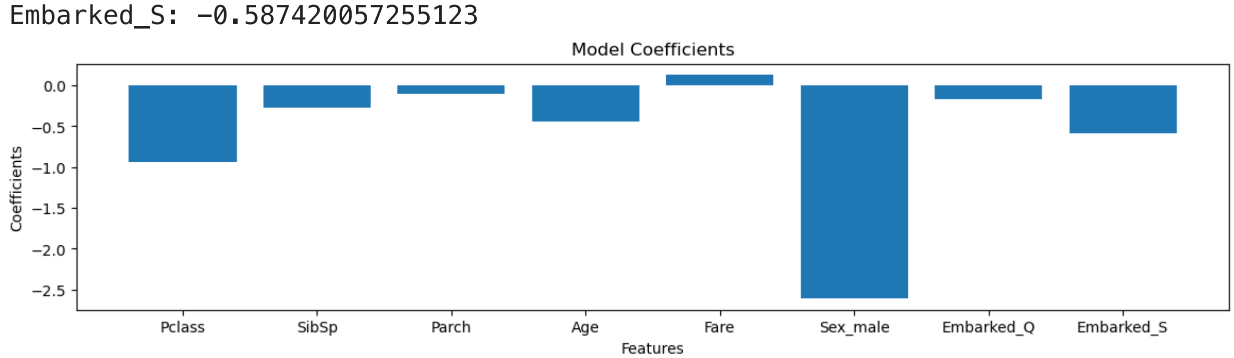
# Model Training

**The sklearn package provides many useful builtin models. Once you have your data ready, you just have to input the training data and labels and use these functions to train your classifier. Below, we're training three different classifiers using three different methods.**

In [ ]:
```
lrc = LogisticRegression(penalty=None).fit(train_features, train_labels)
```

In [ ]:
```
feature_names = train_features.columns
coefficients = lrc.coef_[0]

for feature_name, coefficient in zip(feature_names, coefficients):
  print(f"{feature_name}: {coefficient}")

plt.figure(figsize=(14, 3))
plt.bar(feature_names, coefficients)
plt.xlabel('Features')
plt.ylabel('Coefficients')
plt.title('Model Coefficients')
plt.show()
```

```
Pclass: -0.9427032723435657
SibSp: -0.2741195648187688
Parch: -0.10252105514353024
Age: -0.444742149998205
Fare: 0.12818472014845114
Sex_male: -2.60757900182309
Embarked_Q: -0.17307333492619056
Embarked_S: -0.587420057255123
```




Here are some observations:

- It seems that gender does have a significant correlation with survival; and women have a higher chance of survival (perhaps the "children and women first" rule was in effect?)
- Paying higher fares does seem to slightly increase your odds of survival
- Younger people have a slightly higher chance of survival
- The kaggle data dictionary tell us the following about that column: “pclass: A proxy for socio-economic status (SES). 1st = Upper, 2nd = Middle, 3rd = Lower”, which implies that lower SES passengers had a lower survival rate
- The Embarked_Q and Embarked_S categories should be interpreted relative to embarked_C, which we dropped as a reference. And since they’re negative, you’re more likely to survive if you embarked at C relative to embarking at Q or S.

In [ ]:
```
rfc = RandomForestClassifier(n_estimators=100, max_depth=5, random_state=42).fi
```

In [ ]:
```
svc = SVC(C=100, kernel='rbf').fit(train_features, train_labels)
```

# Model Evaluation

**The sklearn package also provides many useful tools for evaluating your classifiers and for understanding their performance.**

For example:

**confusion_matrix** provides true positive, true negative, false positive, and false negative numbers of your classifier on the test cases.

**roc_auc score** - aggregates a measure of performance across all possible classification thresholds. A higher score indicates robust performance to distinguish positive and negative cases.

**classification_report** - provides overview of performance scores on classification problems (such as precision, recall, and F1-score).

In [ ]:
```
# First do the same data cleaning steps for the test data.
# Remember that you should always normalize within train/test set, and not acro
test_labels = test_data["Survived"]
test_data["Age"].fillna(test_data["Age"].mean(), inplace = True)
test_data[["Age", "Fare"]] = StandardScaler().fit_transform(test_data[["Age", '
test_features = pd.get_dummies(test_data[desired_features], drop_first=True)
```

***You can comment out the different classifiers and rerun the following cells to see and compare their performances!***

In [ ]:
```
predicted_labels = lrc.predict(test_features)

#predicted_labels = rfc.predict(test_features)

#predicted_labels = svc.predict(test_features)
```

In [ ]:
```
print(classification_report(test_labels, predicted_labels, target_names=["Surv
```

```
              precision    recall  f1-score   support

    Survived       0.82      0.87      0.84       157
        Dead       0.79      0.73      0.76       111

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

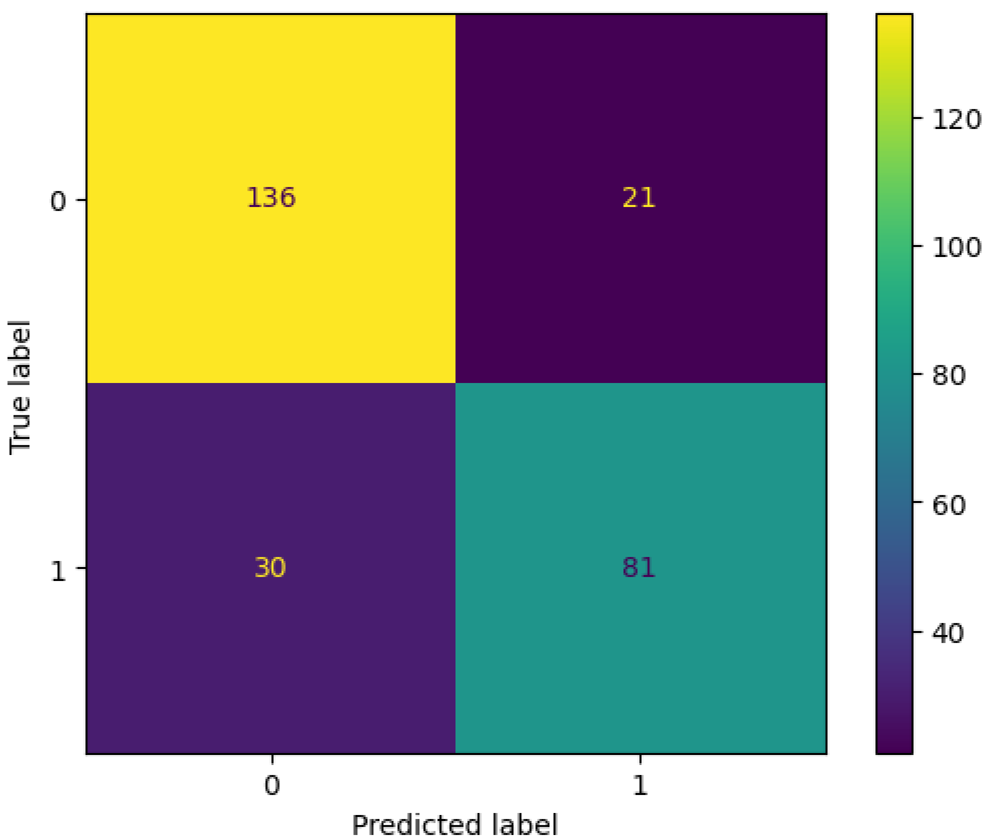
In [ ]:
```
tn, fp, fn, tp = confusion_matrix(test_labels, predicted_labels).ravel()

print("True Negative", tn, "\nFalse Positive", fp, "\nFalse Negative", fn, "\n
```

```
True Negative 136
False Positive 21
False Negative 30
True Positive 81
```

The ConfusionMatrixDisplay plots these values in an intuitive way:

In [ ]:
```
ConfusionMatrixDisplay.from_predictions(test_labels, predicted_labels)
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f95a0b474
c0>
```




In [ ]:
```
auc = roc_auc_score(test_labels, predicted_labels)

print("AUC:", auc)
```

```
AUC: 0.7979858839731452
```

Feel free to tweak the code and evaluate the performance of your classifier. It is up to you to deicide the appropriate measures and algorithms!

# Resources

- Sklearn: Sklearn Library
- Pandas: Pandas Library
- Numpy: Numpy Library
- Matplotlib: Matplotlib Library
- Kaggle: Kaggle Titanic Datasest
- ROC/AUC: ROC/AUC