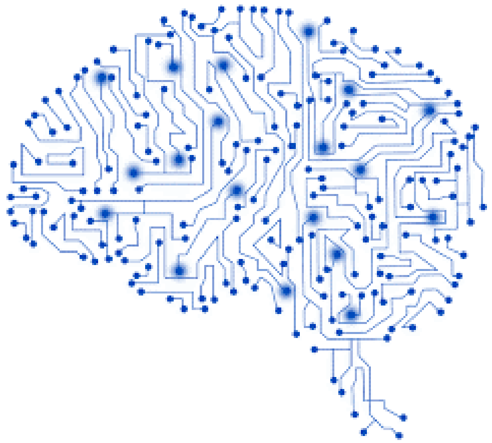
University of Minho
School of Engineering

Intelligent Systems Associate Laboratory



Dados e Aprendizagem Automática

Linear & Logistic Regression

DAA @ MEI-1º/MiEI-4º – 1º Semestre

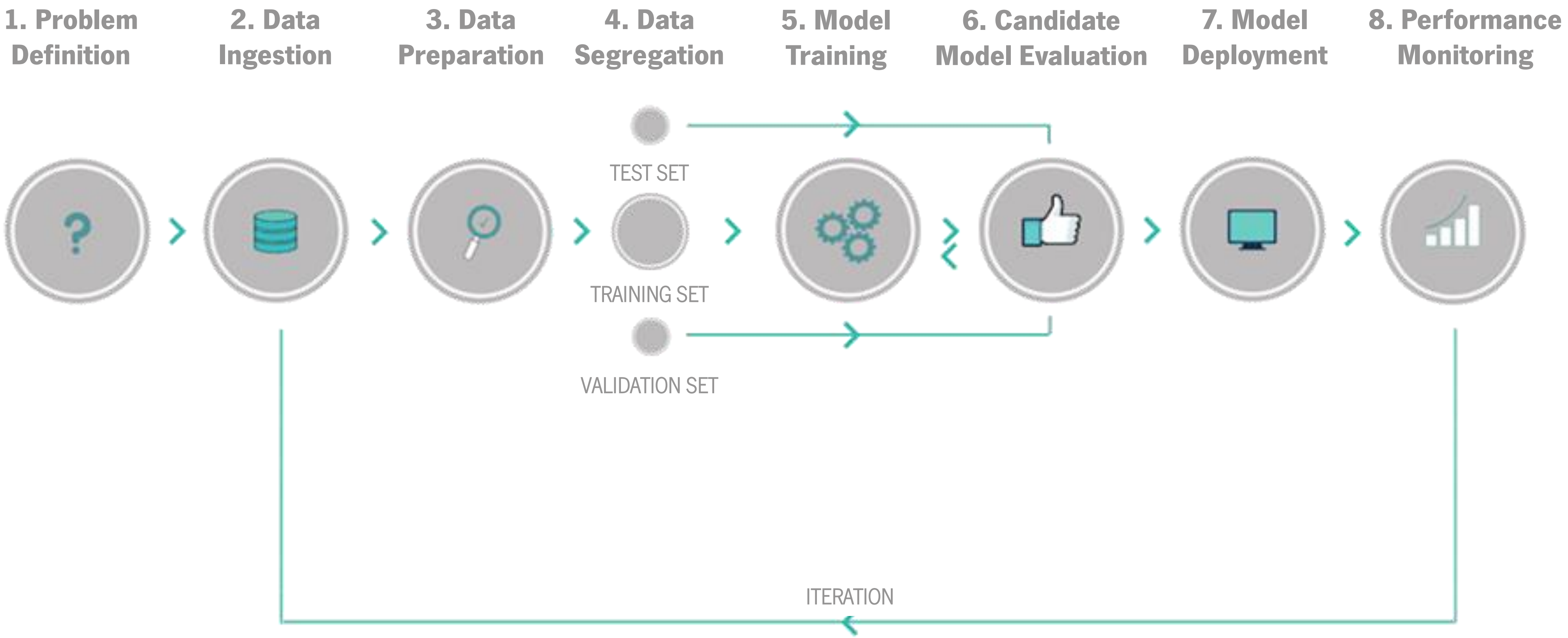
Bruno Fernandes, Dalila Alves, Filipa Ferraz, Victor Alves

Part IV

Contents

- Linear Regression
- Logistic Regression
- Hands On

Supervised Learning



We will explore the **Supervised Learning** algorithms.
The **choice of the algorithm** to implement depends on **the type of data and the context** provided.
If we have **labelled data** or we know in advance what the **output** should be, we choose the Supervised Learning paradigm.

Linear Regression

The Problem and the Data

Problem: development of a Machine Learning model capable of **predicting house prices** for regions in the USA
Approach: **Linear Regression** approach
Dataset: table with information about houses in regions of the United States, including:

- ***Avg. Area Income***: average income of the residents of the city where the house is located
- ***Avg. Area House Age***: average age of the houses in the same city
- ***Avg. Area Number of Rooms***: average number of rooms of the houses in the same city
- ***Avg. Area Number of Bedrooms***: average number of bedrooms of the houses in the same city
- ***Area Population***: population of the city where the house is located
- ***Price***: price at which the house was sold
- ***Address***: address of the house

Exploratory Data Analysis


```
USAhousing = pd.read_csv('USA_Housing.csv')
```

```
USAhousing.head()
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address
0	79545.458574	5.682861	7.009188	4.09	23086.800503	1.059034e+06	208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1	79248.642455	6.002900	6.730821	3.09	40173.072174	1.505891e+06	188 Johnson Views Suite 079\nLake Kathleen, CA...
2	61287.067179	5.865890	8.512727	5.13	36882.159400	1.058988e+06	9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3	63345.240046	7.188236	5.586729	3.26	34310.242831	1.260617e+06	USS Barnett\nFPO AP 44820
4	59982.197226	5.040555	7.839388	4.23	26354.109472	6.309435e+05	USNS Raymond\nFPO AE 09386

```
USAhousing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Avg. Area Income              5000 non-null   float64
 1   Avg. Area House Age           5000 non-null   float64
 2   Avg. Area Number of Rooms     5000 non-null   float64
 3   Avg. Area Number of Bedrooms  5000 non-null   float64
 4   Area Population               5000 non-null   float64
 5   Price                         5000 non-null   float64
 6   Address                       5000 non-null   object 
dtypes: float64(6), object(1)
memory usage: 273.6+ KB
```

Exploratory Data Analysis


```
USAhousing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Avg. Area Income              5000 non-null   float64
 1   Avg. Area House Age           5000 non-null   float64
 2   Avg. Area Number of Rooms     5000 non-null   float64
 3   Avg. Area Number of Bedrooms  5000 non-null   float64
 4   Area Population               5000 non-null   float64
 5   Price                         5000 non-null   float64
 6   Address                       5000 non-null   object 
dtypes: float64(6), object(1)
memory usage: 273.6+ KB
```

```
USAhousing.columns
```

```
Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
       'Avg. Area Number of Bedrooms', 'Area Population', 'Price', 'Address'],
      dtype='object')
```

Exploratory Data Analysis

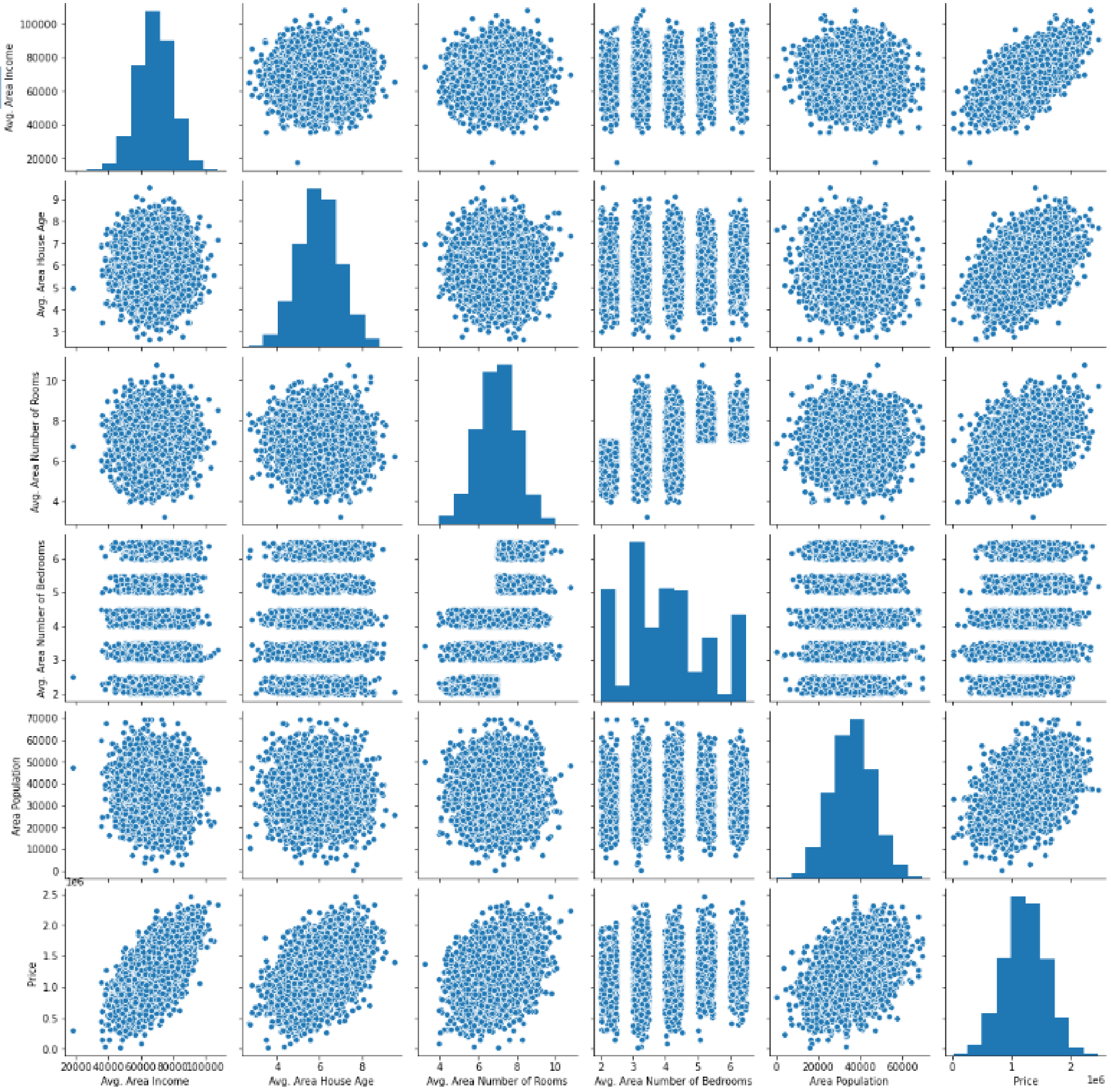

```
USAhousing.describe()
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5.000000e+03
mean	68583.108984	5.977222	6.987792	3.981330	36163.516039	1.232073e+06
std	10657.991214	0.991456	1.005833	1.234137	9925.650114	3.531176e+05
min	17796.631190	2.644304	3.236194	2.000000	172.610686	1.593866e+04
25%	61480.562388	5.322283	6.299250	3.140000	29403.928702	9.975771e+05
50%	68804.286404	5.970429	7.002902	4.050000	36199.406689	1.232669e+06
75%	75783.338666	6.650808	7.665871	4.490000	42861.290769	1.471210e+06
max	107701.748378	9.519088	10.759588	6.500000	69621.713378	2.469066e+06

Exploratory Data Analysis

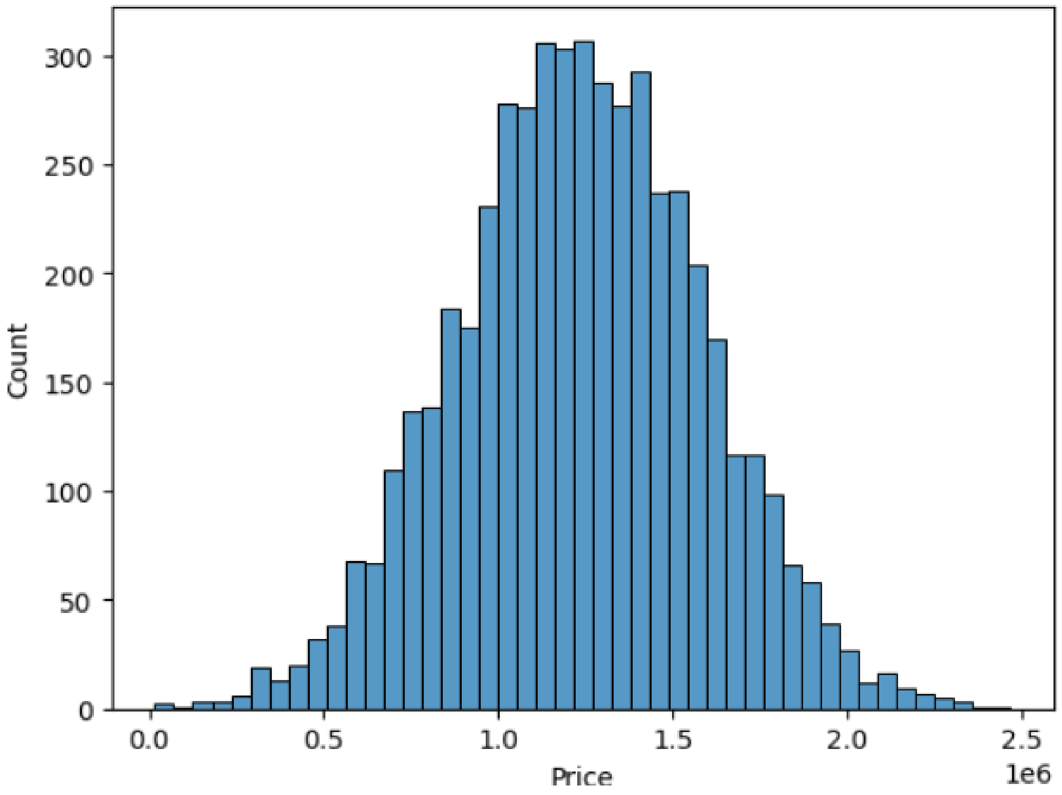
Let’s create some plots to check the data:

```
sns.pairplot(USAhousing, hue="Price")
```

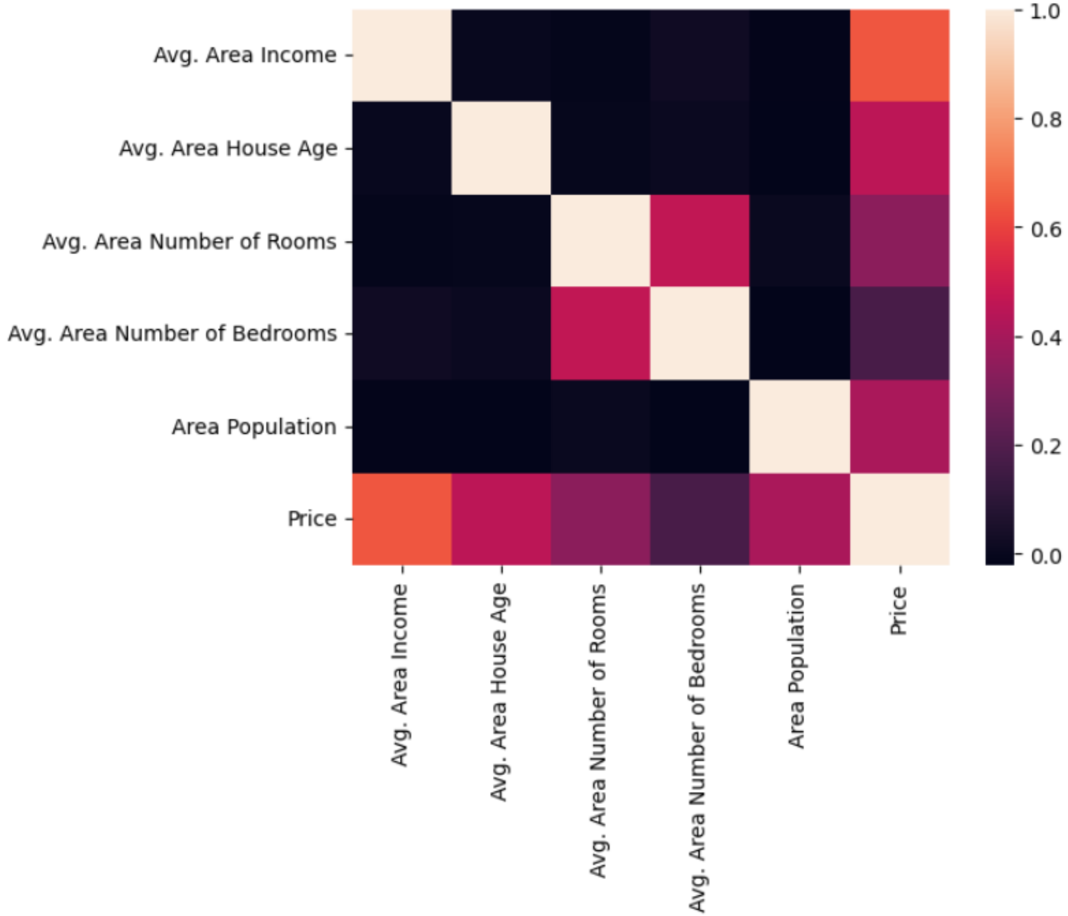


Exploratory Data Analysis


```
sns.histplot(USAhousing['Price'])
```




```
sns.heatmap(USAhousing.corr(numeric_only=True))
```



Linear Regression

Let's now begin to train our model.
The target is the **Price** so we will implement a Linear Regression model.
The feature **Address** will not be consider since is not relevant nor a numeric variable.

```
X = USAhousing[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
               'Avg. Area Number of Bedrooms', 'Area Population']]
y = USAhousing['Price']
```

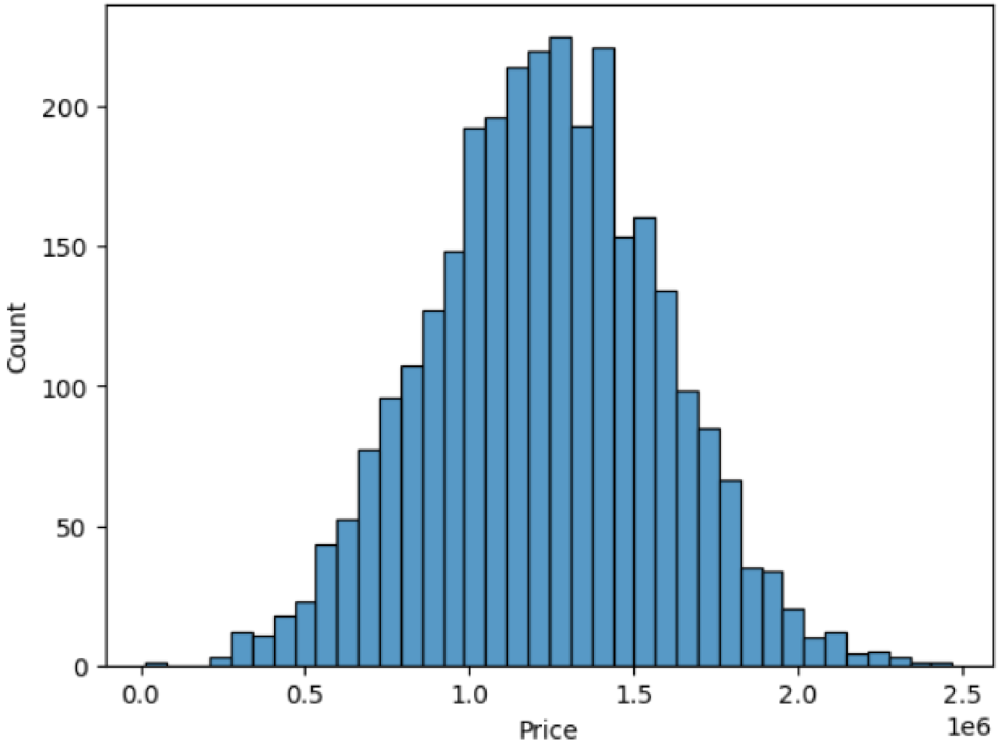
```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=101)
```

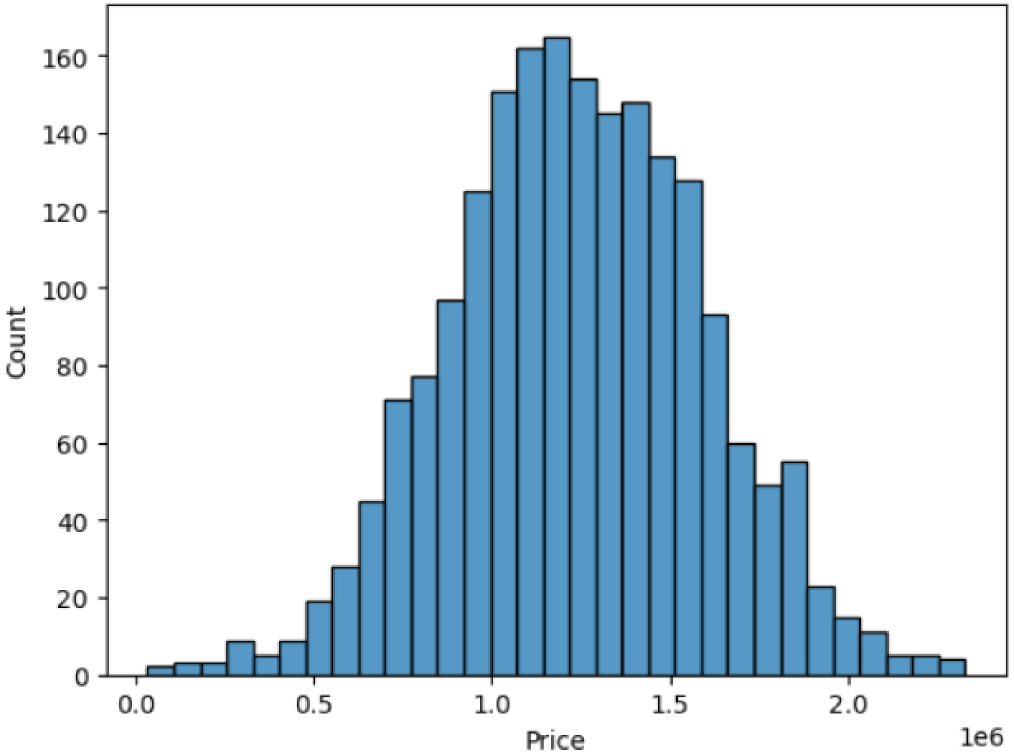
Linear Regression

Let’s check the distribution of each subset:

```
sns.histplot(y_train)
```




```
sns.histplot(y_test)
```



Linear Regression


```
sklearn.linear_model.LinearRegression(*, fit_intercept=True, copy_X=True, n_jobs=None, positive=False)
```

Creating and training the model:

```
from sklearn.linear_model import LinearRegression
```

```
lm = LinearRegression()
lm.fit(X_train,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

Then, evaluating the model by checking its coefficients and interpreting them:

```
print(lm.intercept_)
```

```
-2640159.7968525267
```

Linear Regression


```
coeff_df = pd.DataFrame(lm.coef_,X.columns,columns=['Coefficient'])
coeff_df
```

	Coefficient
Avg. Area Income	21.528276
Avg. Area House Age	164883.282027
Avg. Area Number of Rooms	122368.678027
Avg. Area Number of Bedrooms	2233.801864
Area Population	15.150420

Holding all the other features fixed:

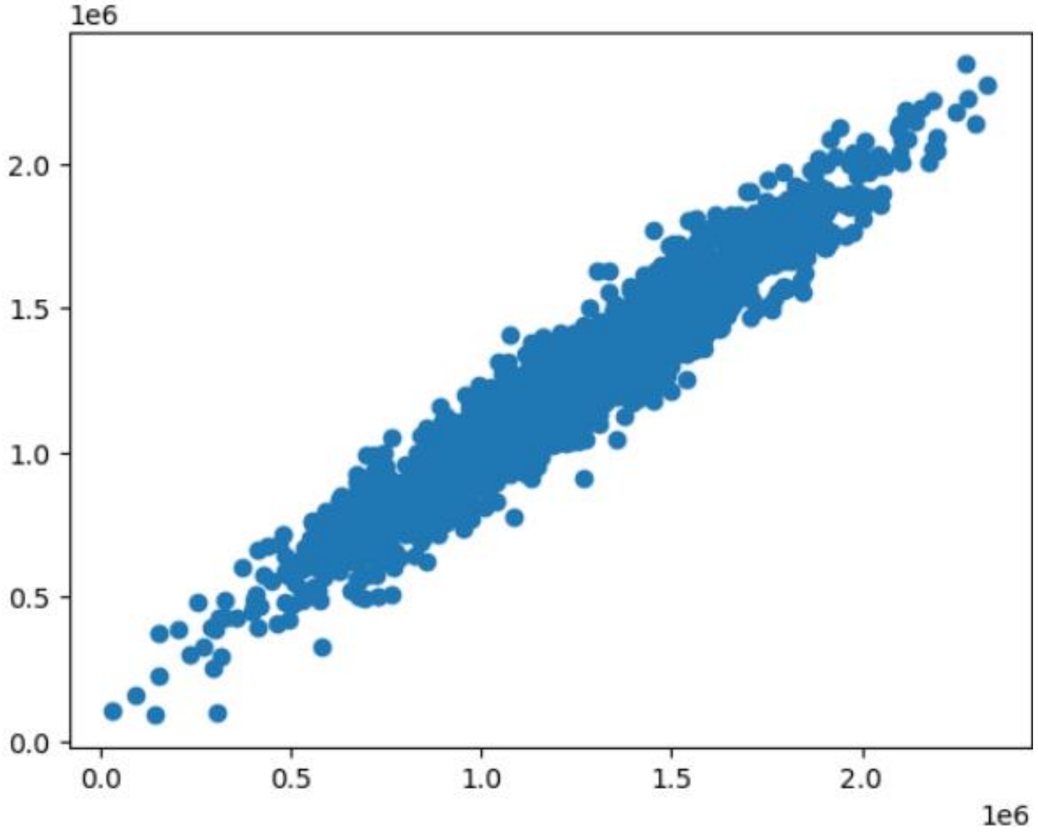
- 1 unit increase in ***Avg. Area Income*** is associated with an increase of *21,53 $*
- 1 unit increase in ***Avg. Area House Age*** is associated with an increase of *164883,28 $*
- 1 unit increase in ***Avg. Area Number of Rooms*** is associated with an increase of *122368,68 $*
- 1 unit increase in ***Avg. Area Number of Bedrooms*** is associated with an increase of *2233,80 $*
- 1 unit increase in ***Area Population*** is associated with an increase of *15,15 $*

Linear Regression

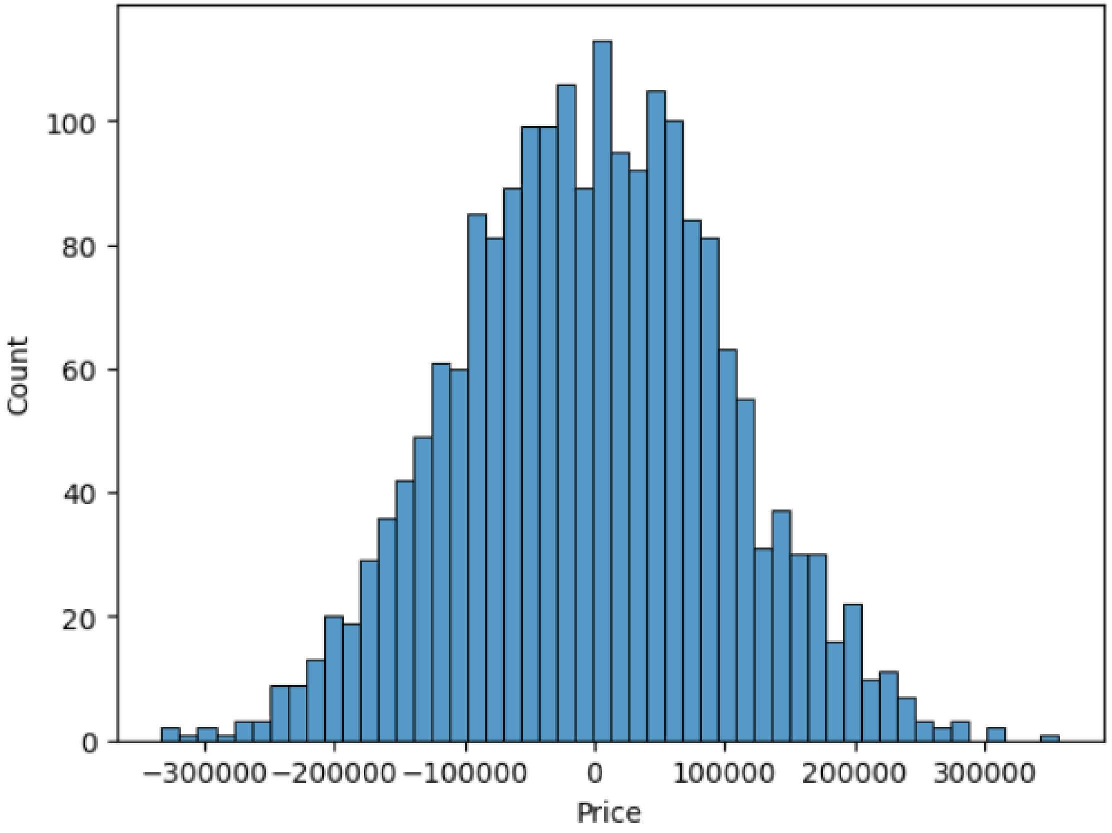
Let's analyze the predictions and plot it:

```
predictions = lm.predict(X_test)
```

```
plt.scatter(y_test,predictions)
```




```
sns.histplot((y_test-predictions),bins=50);
```



Linear Regression

Regression Evaluation Metrics

The three most common evaluation metrics for regression problems are:

- Mean Absolute Error (MAE)
- Mean Squared Error (MSE)
- Root Mean Squared Error (RMSE)

Comparing them:

- **MAE** is the easiest to understand because it's the *average error*;
- **MSE** is more popular than MAE because MSE *"punishes" large errors*;
- **RMSE** is even more popular than MSE because RMSE is interpretable in *units of the target variable*

All of these are **loss functions** because we want to *minimize the error*.

```
from sklearn import metrics
```

```
print('MAE:', metrics.mean_absolute_error(y_test, predictions))
print('MSE:', metrics.mean_squared_error(y_test, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
```

```
MAE: 82288.22251914947
MSE: 10460958907.209057
RMSE: 102278.82922290935
```

Logistic Regression

The Problem and the Data

Problem: development of a Machine Learning model that predicts **which passengers survived** the Titanic shipwreck

Approach: **Logistic Regression** approach

Dataset: table with information regarding passengers' information, including:

- ***survival***: if the passenger survived (0: No, 1: Yes)
- ***pclass***: ticket class (1: 1st, 2: 2nd, 3: 3rd)
- ***sex***: M: Male, F: Female
- ***Age***: age in years
- ***sibsp***: number of siblings per spouses aboard
- ***parch***: number pf parents per children aboard
- ***ticket***: ticket number
- ***fare:*** passenger fare
- ***cabin:*** cabin number
- ***embarked:*** port of embarkation (C: Cherbourg, Q: Queenstown, S: Southampton)

Exploratory Data Analysis


```
train = pd.read_csv('titanic_train.csv')
```

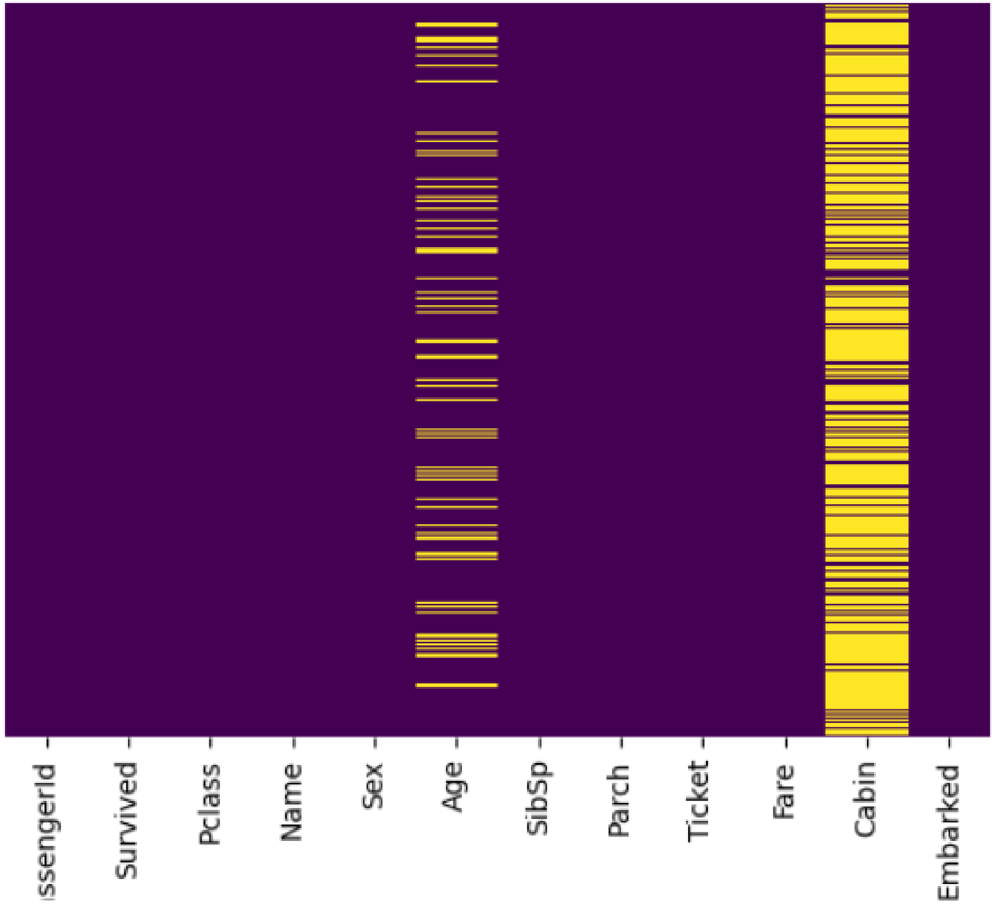
```
train.head()
```

	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

Exploratory Data Analysis

Let's check if there are missing data:

```
sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```



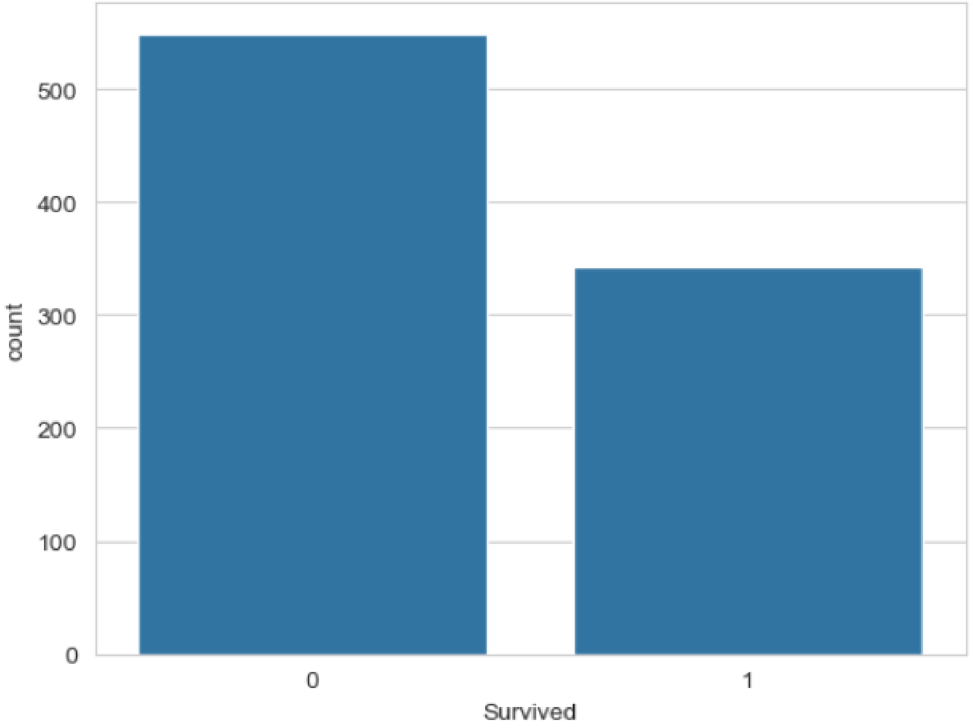
About 20% of the *Age* data is **missing**. The proportion of *Age* data missing is likely small enough for reasonable replacement with some form of imputation.

Looking at the *Cabin* column it looks like we are just **missing too much** of that data to do something useful with at a basic level. We will probably drop this later, or change it into another feature like "Cabin Known: 1 or 0"

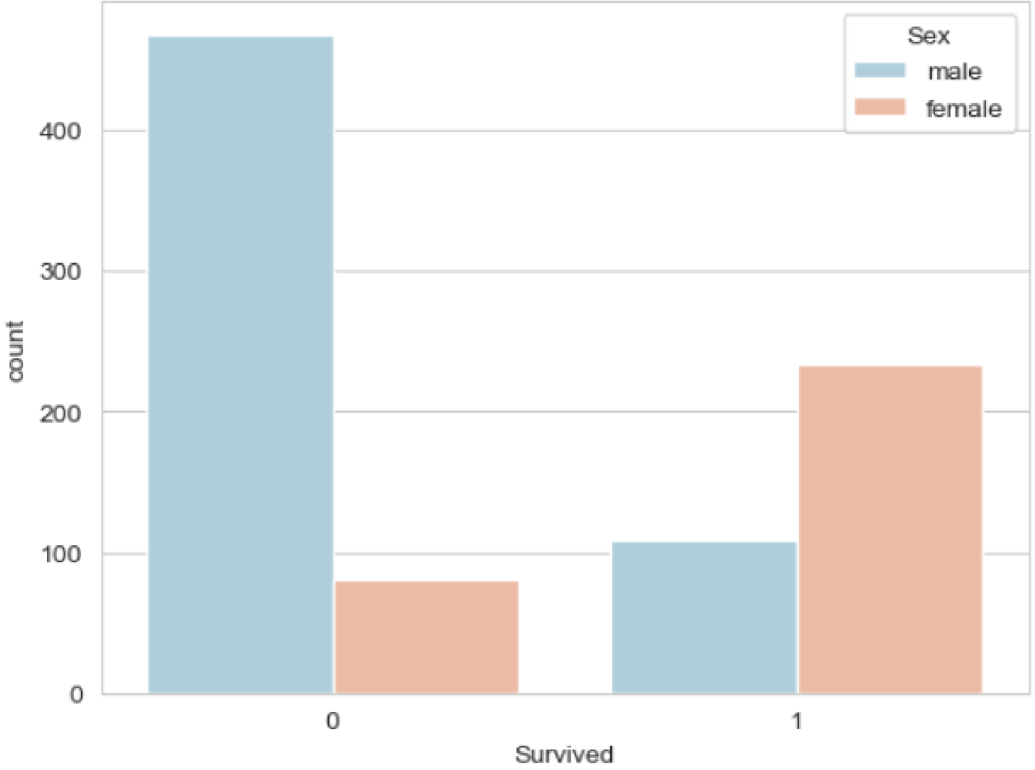
Exploratory Data Analysis

Let's continue on by visualizing some more of the data:

```
sns.set_style('whitegrid')
sns.countplot(x='Survived', data=train)
```

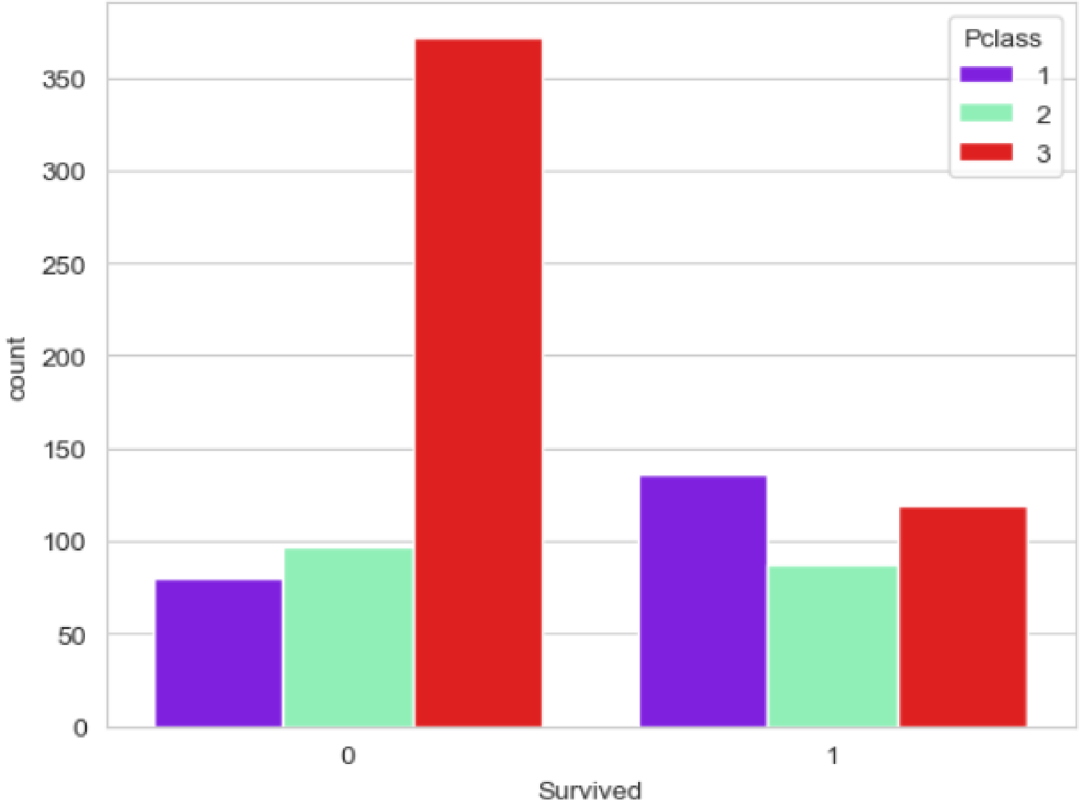



```
sns.set_style('whitegrid')
sns.countplot(x='Survived',hue='Sex',data=train,palette='RdBu_r')
```

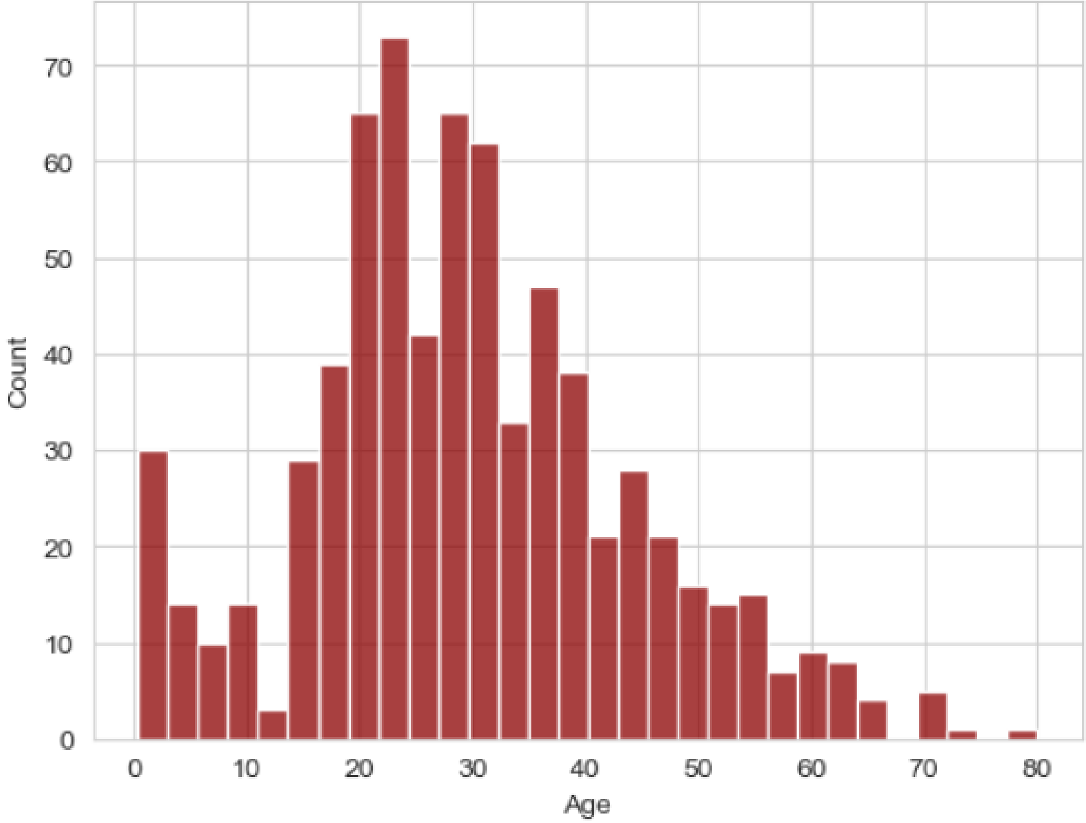


Exploratory Data Analysis


```
sns.set_style('whitegrid')
sns.countplot(x='Survived',hue='Pclass',data=train,palette='rainbow')
```

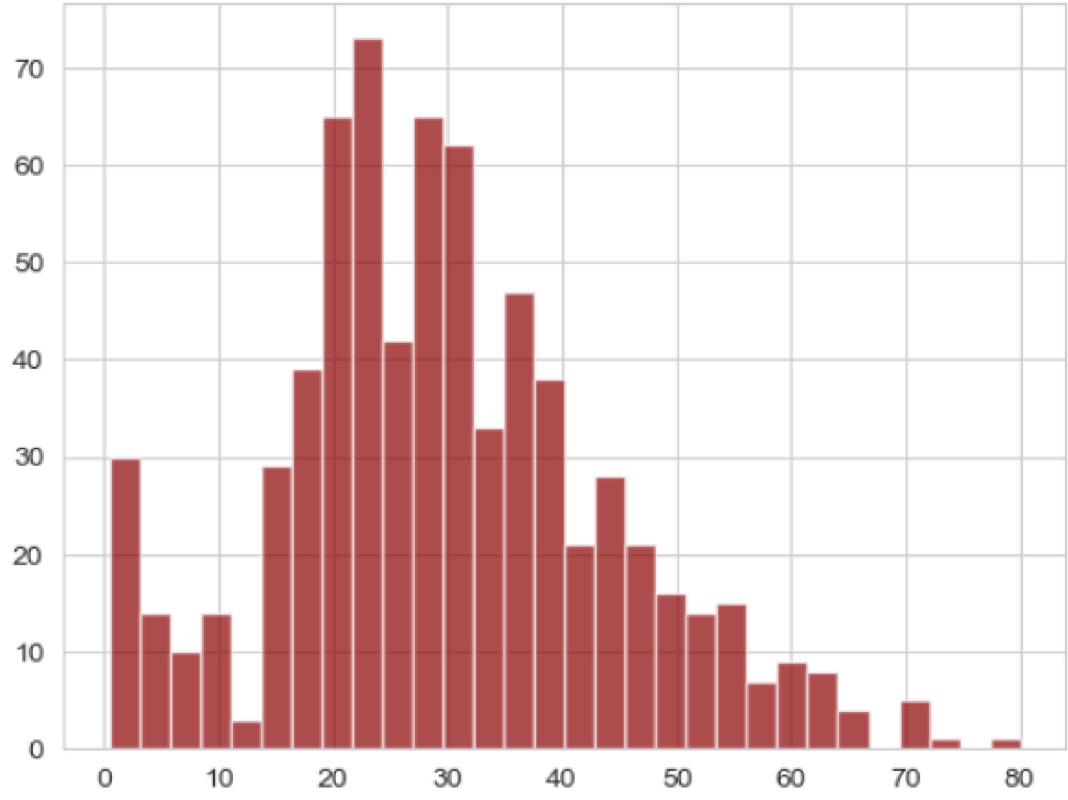



```
sns.histplot(train['Age'].dropna(),kde=False,color='darkred',bins=30)
```

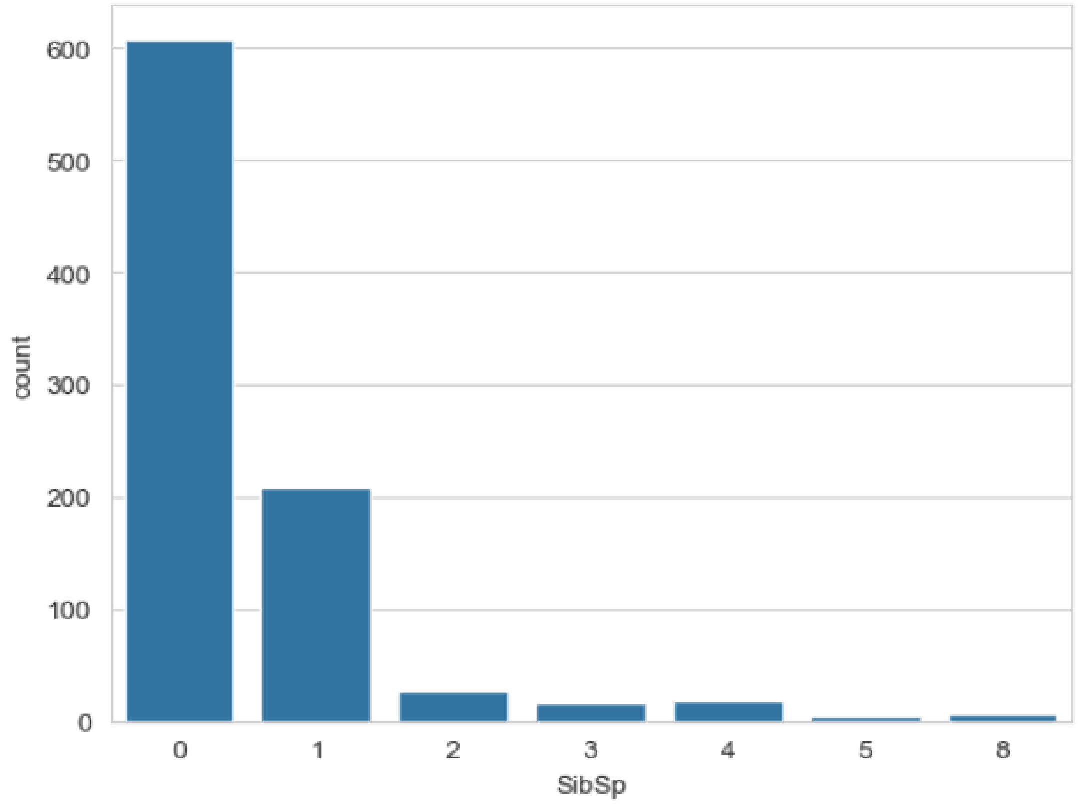


Exploratory Data Analysis


```
train['Age'].hist(bins=30,color='darkred',alpha=0.7)
```

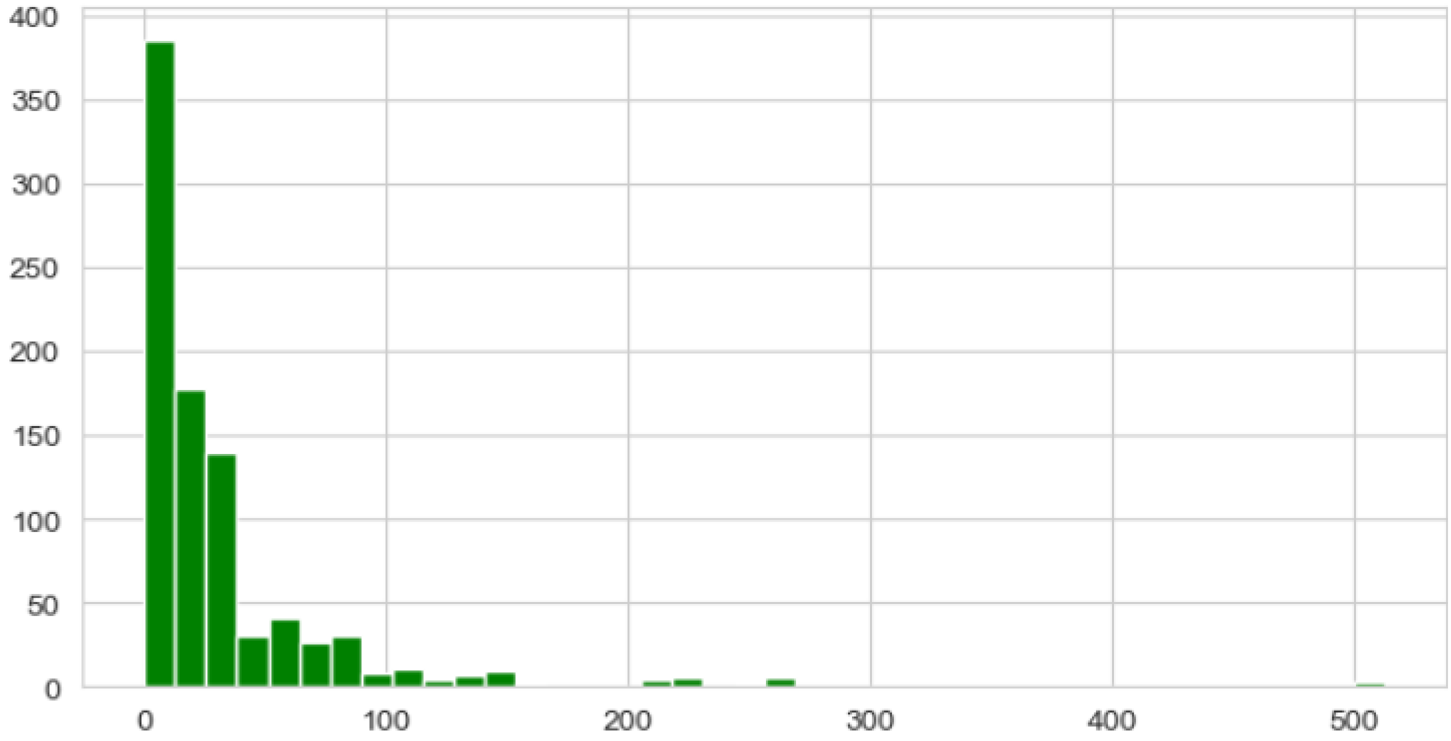



```
sns.countplot(x='SibSp',data=train)
```



Exploratory Data Analysis

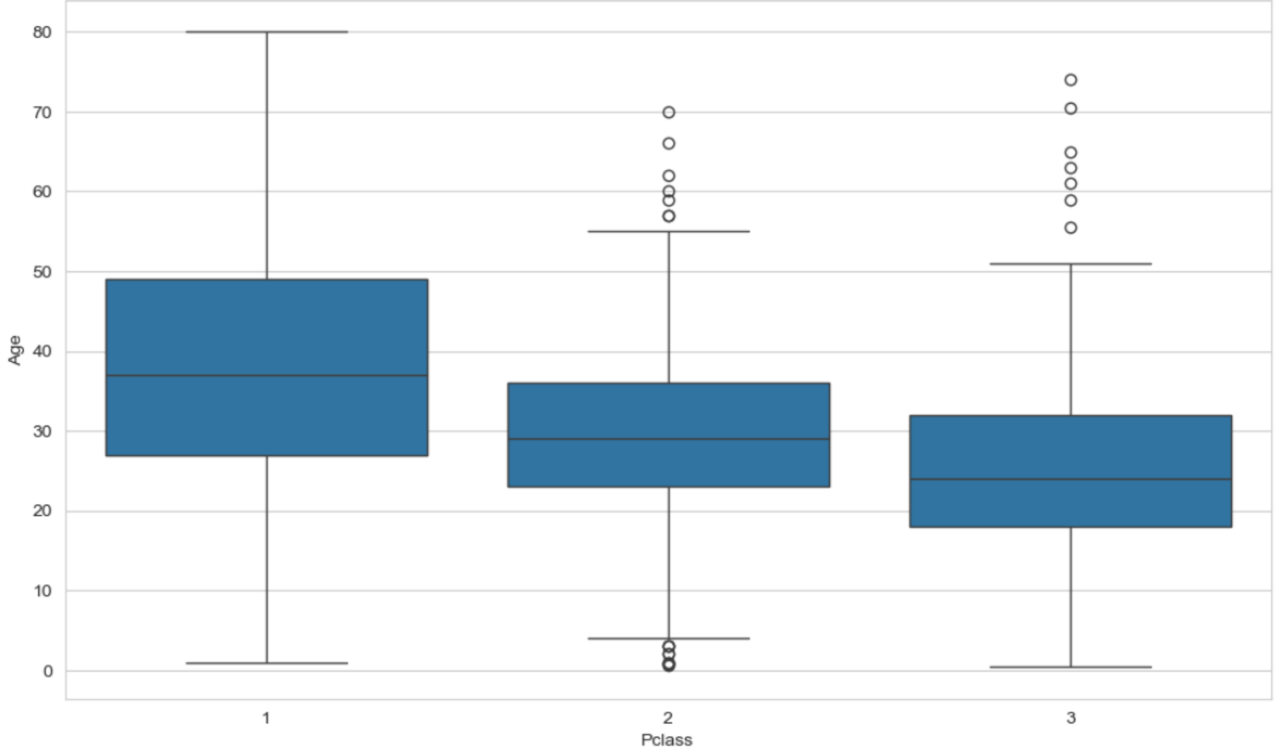

```
train['Fare'].hist(color='green',bins=40,figsize=(8,4))
```



Data Preprocessing

We want to **fill in missing values** instead of dropping it. One way to do this is by filling the mean age of all passengers (**imputation**). However, we can be smarter about this and check the average age by passenger class:

```
plt.figure(figsize=(12, 7))
sns.boxplot(x='Pclass', y='Age', data=train)
```



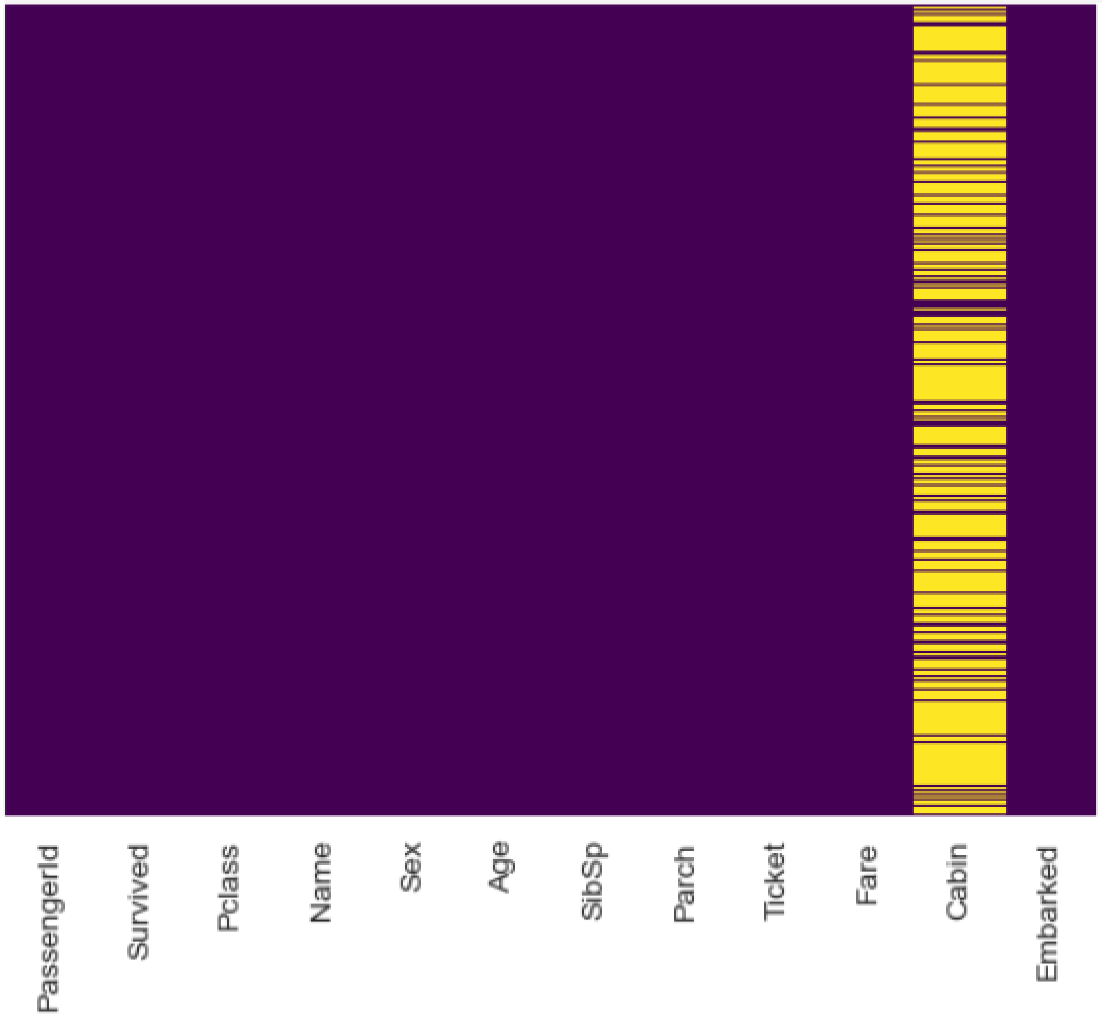
We can see that wealthier passengers in the higher classes tend to be older which makes sense. We will use these average age values to impute based on *pclass*.

Data Preprocessing


```python
def impute_age(cols):
    Age = cols[0]
    Pclass = cols[1]

    if pd.isnull(Age):

        if Pclass == 1:
            return 37

        elif Pclass == 2:
            return 29

        else:
            return 24

    else:
        return Age
```

```python
train['Age'] = train[['Age','Pclass']].apply(impute_age,axis=1)
```

```python
sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```



Data Preprocessing

Now let's drop the column *cabin* and the rows in *Embarked* that are NaN:

```
train.drop('Cabin',axis=1,inplace=True)
```

```
train.head()
```

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

```
train.dropna(inplace=True)
```

Data Preprocessing

We will need to convert categorical features into dummy variables using pandas' library:

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 889 entries, 0 to 890
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  889 non-null    int64
 1   Survived     889 non-null    int64
 2   Pclass       889 non-null    int64
 3   Name         889 non-null    object
 4   Sex          889 non-null    object
 5   Age          889 non-null    float64
 6   SibSp        889 non-null    int64
 7   Parch        889 non-null    int64
 8   Ticket       889 non-null    object
 9   Fare         889 non-null    float64
 10  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(4)
memory usage: 83.3+ KB
```

```
sex = pd.get_dummies(train['Sex'],drop_first=True)
embark = pd.get_dummies(train['Embarked'],drop_first=True)
```

```
train.drop(['Sex','Embarked','Name','Ticket'],axis=1,inplace=True)
```

```
train = pd.concat([train,sex,embark],axis=1)
```

```
train.head()
```

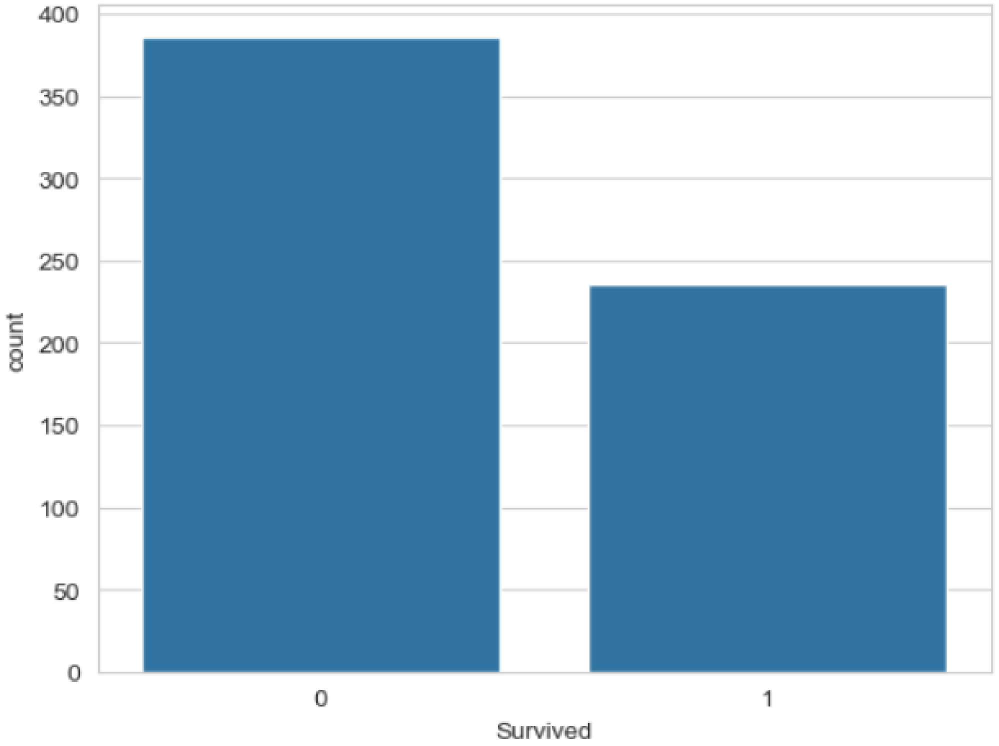
	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	male	Q	S
0	1	0	3	22.0	1	0	7.2500	True	False	True
1	2	1	1	38.0	1	0	71.2833	False	False	False
2	3	1	3	26.0	0	0	7.9250	False	False	True
3	4	1	1	35.0	1	0	53.1000	False	False	True
4	5	0	3	35.0	0	0	8.0500	True	False	True

Logistic Regression

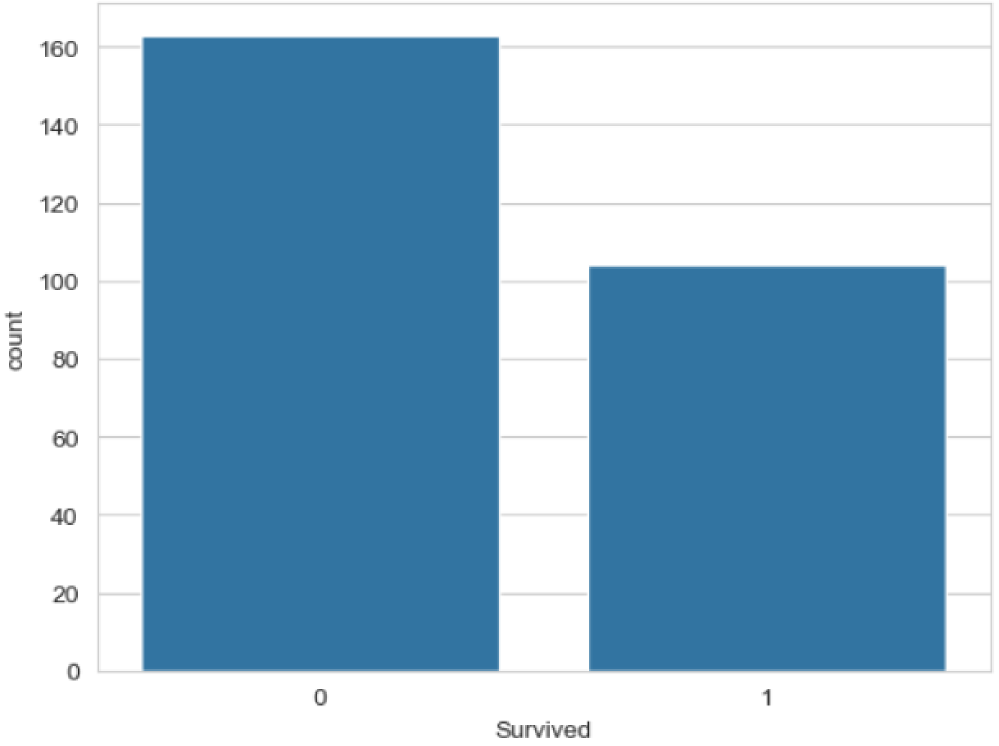

```
from sklearn.model_selection import train_test_split
X = train.drop('Survived',axis=1)
y = train['Survived']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=101)
```

```
sns.set_style('whitegrid')
sns.countplot(x='Survived', data = pd.DataFrame(y_train,columns=['Survived']) )
```




```
sns.set_style('whitegrid')
sns.countplot(x='Survived', data = pd.DataFrame(y_test,columns=['Survived']) )
```



Logistic Regression


```
sklearn.linear_model.LogisticRegression(penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None,
                        random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None)
```

Logistic Regression' **solvers**:

- For **small datasets** *liblinear* is a good choice whereas *sag* and *saga* are faster for **larger ones**;
- For **multiclass** problems only *newton-cg*, *sag*, *saga* and *lbfgs* handle multinomial loss;
- *liblinear* is limited to **one-versus-rest schemes**.

Supported **penalties** by solver:

- ***newton-cg*** – *[l2, none]*
- ***lbfgs*** – *[l2, none]*
- ***liblinear*** – *[l1, l2]*
- ***sag*** – *[l2, none]*
- ***saga*** – *[elasticnet, l1, l2, none]*

```
from sklearn.linear_model import LogisticRegression
```

Logistic Regression

Model 1: `random_state = 2022, solver = 'newton-cg'`

```
starttime = time.process_time()

logmodel1 = LogisticRegression(random_state=2022, solver='newton-cg')
print(logmodel1)
logmodel1.fit(X_train,y_train)

endtime = time.process_time()
print(f"Time spent: {endtime - starttime} seconds")
```

```
LogisticRegression(random_state=2022, solver='newton-cg')
Time spent: 0.015625 seconds
```

```
predictions1 = logmodel1.predict(X_test)
```

Model 2: `random_state = 2022, solver = 'lbfgs'`

```
starttime = time.process_time()

logmodel2 = LogisticRegression(random_state=2022, solver='lbfgs', max_iter=800)
print(logmodel2)
logmodel2.fit(X_train,y_train)

endtime = time.process_time()
print(f"Time spent: {endtime - starttime} seconds")
```

```
LogisticRegression(max_iter=800, random_state=2022)
Time spent: 0.078125 seconds
```

Model 3: `random_state = 2022, solver = 'liblinear'`

```
starttime = time.process_time()

logmodel3 = LogisticRegression(random_state=2022, solver='liblinear')
print(logmodel3)
logmodel3.fit(X_train,y_train)

endtime = time.process_time()
print(f"Time spent: {endtime - starttime} seconds")
```

```
LogisticRegression(random_state=2022, solver='liblinear')
Time spent: 0.0 seconds
```

```
predictions3 = logmodel3.predict(X_test)
```

Logistic Regression

Let's evaluate the model using precision, recall, f1-score and confusion matrix:

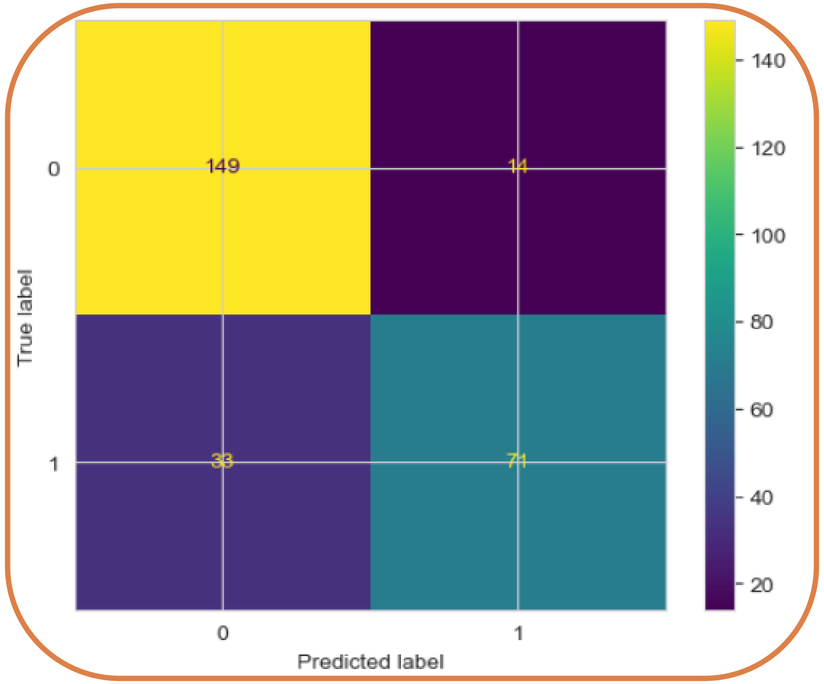
```
from sklearn.metrics import classification_report, ConfusionMatrixDisplay
```

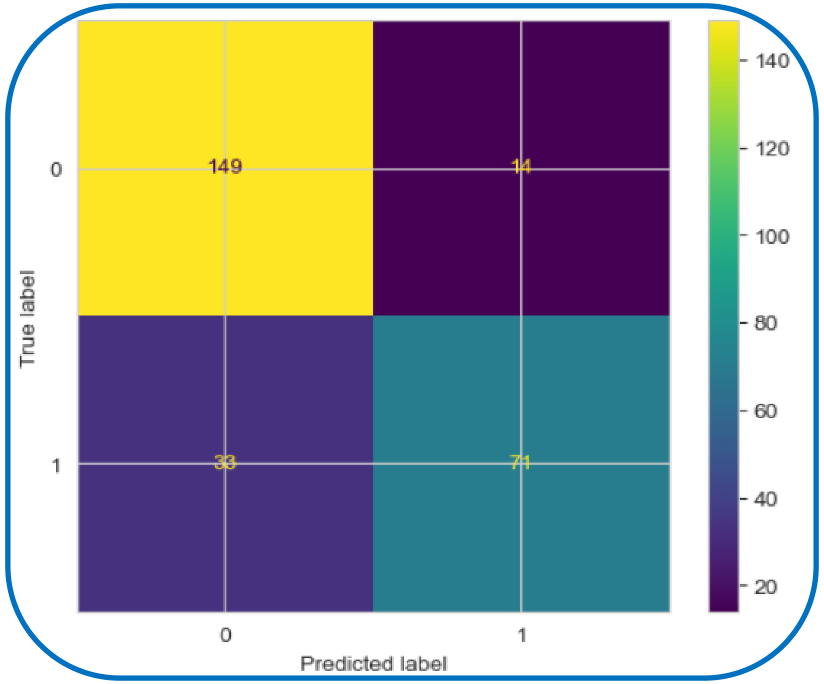
```
print("With 'newton-cg': \n", classification_report(y_test,predictions1))
print("With 'lbfgs': \n", classification_report(y_test,predictions2))
print("With 'liblinear': \n", classification_report(y_test,predictions3))
```

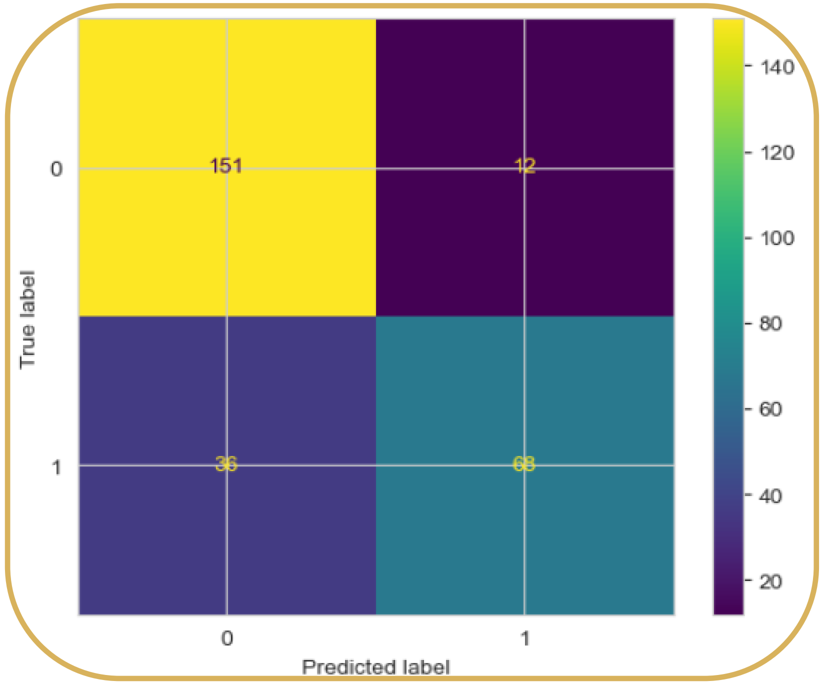
```
With 'newton-cg':
              precision    recall  f1-score   support

           0       0.82      0.91      0.86       163
           1       0.84      0.68      0.75       104

    accuracy                           0.82       267
   macro avg       0.83      0.80      0.81       267
weighted avg       0.83      0.82      0.82       267

With 'lbfgs':
              precision    recall  f1-score   support

           0       0.82      0.91      0.86       163
           1       0.84      0.68      0.75       104

    accuracy                           0.82       267
   macro avg       0.83      0.80      0.81       267
weighted avg       0.83      0.82      0.82       267

With 'liblinear':
              precision    recall  f1-score   support

           0       0.81      0.93      0.86       163
           1       0.85      0.65      0.74       104

    accuracy                           0.82       267
   macro avg       0.83      0.79      0.80       267
weighted avg       0.82      0.82      0.81       267
```

Logistic Regression


```
ConfusionMatrixDisplay.from_predictions(y_test, predictions1)
ConfusionMatrixDisplay.from_predictions(y_test, predictions2)
ConfusionMatrixDisplay.from_predictions(y_test, predictions3)
plt.show()
```







Hands On