Machine Learning Classification: Predicting Survival in the Titanic Shipwreck

Rodrigo Lope Prieto

Contents

Data Analysis Section

Main Objectives

We will work with a dataset of passengers of the Titanic, provided by a Kaggle Machine Learning competition, which can be found here:

https://www.kaggle.com/competitions/titanic/data

The dataset contains 421 unique vales and consists of 10 features which we will explain in due course.

The main objective of the analysis is determine whether a passenger survived or not, based on models which will be trained on the features provided in the dataset above.

Dataset Description

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.00	1	0	A/5 21171	7.25	29.70	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.00	1	0	PC 17599	71.28	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.00	0	0	STON/O2. 3101282	7.92	29.70	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.00	1	0	113803	53.10	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.00	0	0	373450	8.05	29.70	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.00	0	0	211536	13.00	29.70	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.00	0	0	112053	30.00	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	29.70	1	2	W./C. 6607	23.45	29.70	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.00	0	0	111369	30.00	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.00	0	0	370376	7.75	29.70	Q

891 rows × 12 columns

Variable	Definition	Key
survival	Survival	0 = No, 1 = Yes
Class	Ticket Class	1 = 1st, 2 = 2nd, 3 = 3rd
Sex		0 = Female, 1 = Male
Age	Age in years	
sibsp	Number of siblings/spouses aboard	
parch	Number of parents/children aboard	
ticket	Ticket number	
Fare	Fare paid	
Cabin	Cabin number	
Embarked	Port of embarkation	S = Southhampton, Q = Queenstown, C = Cherbourg

Data Cleaning

Only null values were in the ‘Age’ column. We also replace all possible infinity values. We also clean all possible `float’ type cells so that LabelEncoder can process our data. These happen to be only in the ‘Embarked’ and ‘Cabin’ columns.

```
data.Embarked.map(type)==float
np.array(data.Embarked.map(type)==float).sum()
for i in range(0, len(data['Embarked'])):
    if type(data['Embarked'][i] == float):
        data['Embarked'][i] = str(data['Embarked'][i])


data.Cabin.map(type)==float
np.array(data.Cabin.map(type)==float).sum()
for i in range(0, len(data['Cabin'])):
    if type(data['Cabin'][i] == float):
        data['Cabin'][i] = str(data['Cabin'][i])
```

```
data.replace([np.inf, -np.inf], 0, inplace=True)

mean_age = data['Age'].mean()

data.fillna(mean_age, inplace=True)
```

Data Analysis

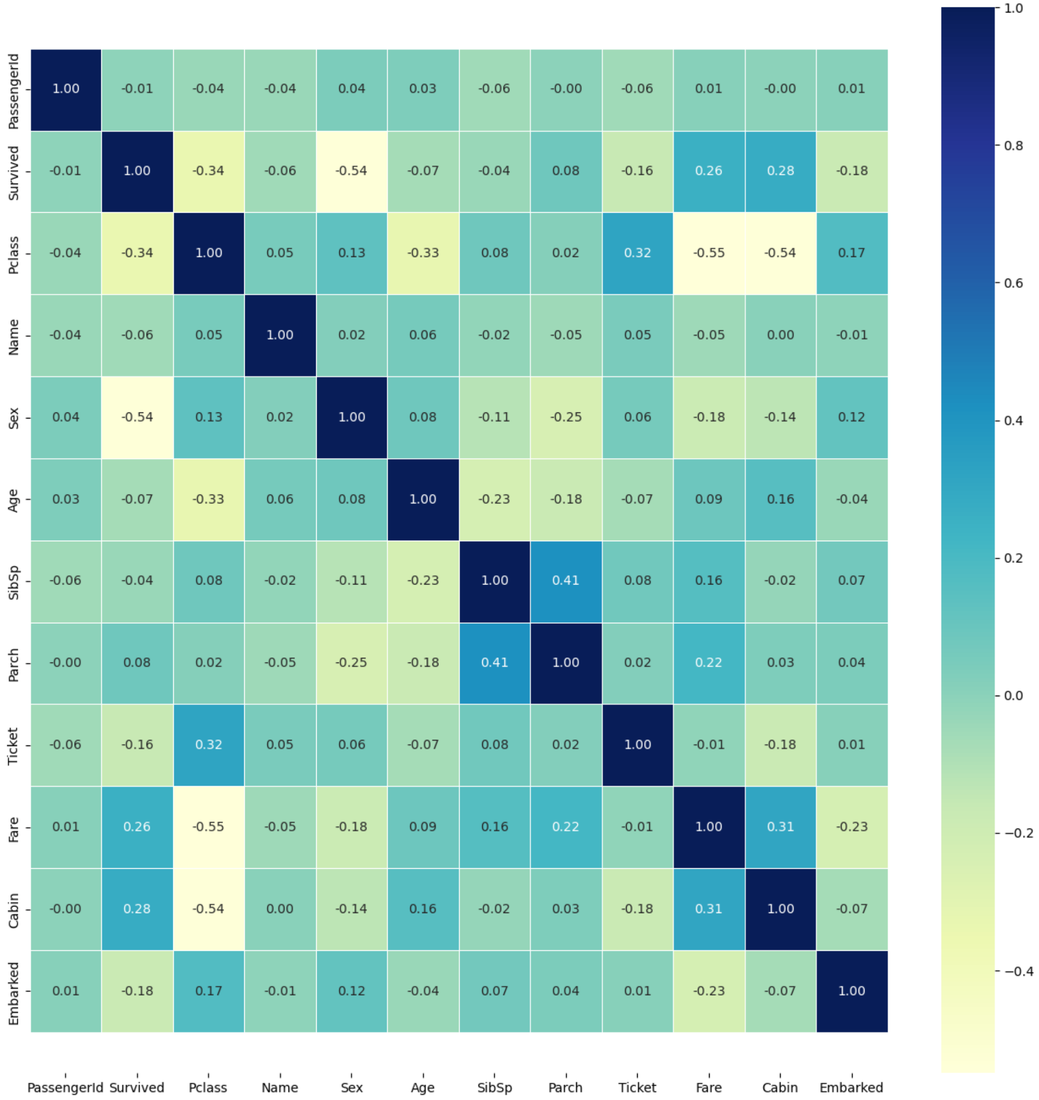
Then, proceed to identify categorical and numeric variables, and binary and non-binary variables:

```
#Identify binary variables

data_uniques = pd.DataFrame([[i, len(data[i].unique())] for i in data.columns], columns=['Variable', 'Unique Values']).set_index('Variable')

binary_variables = list(data_uniques[data_uniques['Unique Values'] == 2].index)
non_binary_variables = [column for column in data.columns if column not in binary_variables]



#Identify categorical and numeric data

categorical_val = []
numeric_val = []
numeric_data = data.select_dtypes(include=[np.number])
categorical_data = data.select_dtypes(exclude=[np.number])

for column in data.columns:
    if column in numeric_data.columns:
        numeric_val.append(column)


for column in data.columns:
    if column in categorical_data.columns:
        categorical_val.append(column)
```

Data Analysis

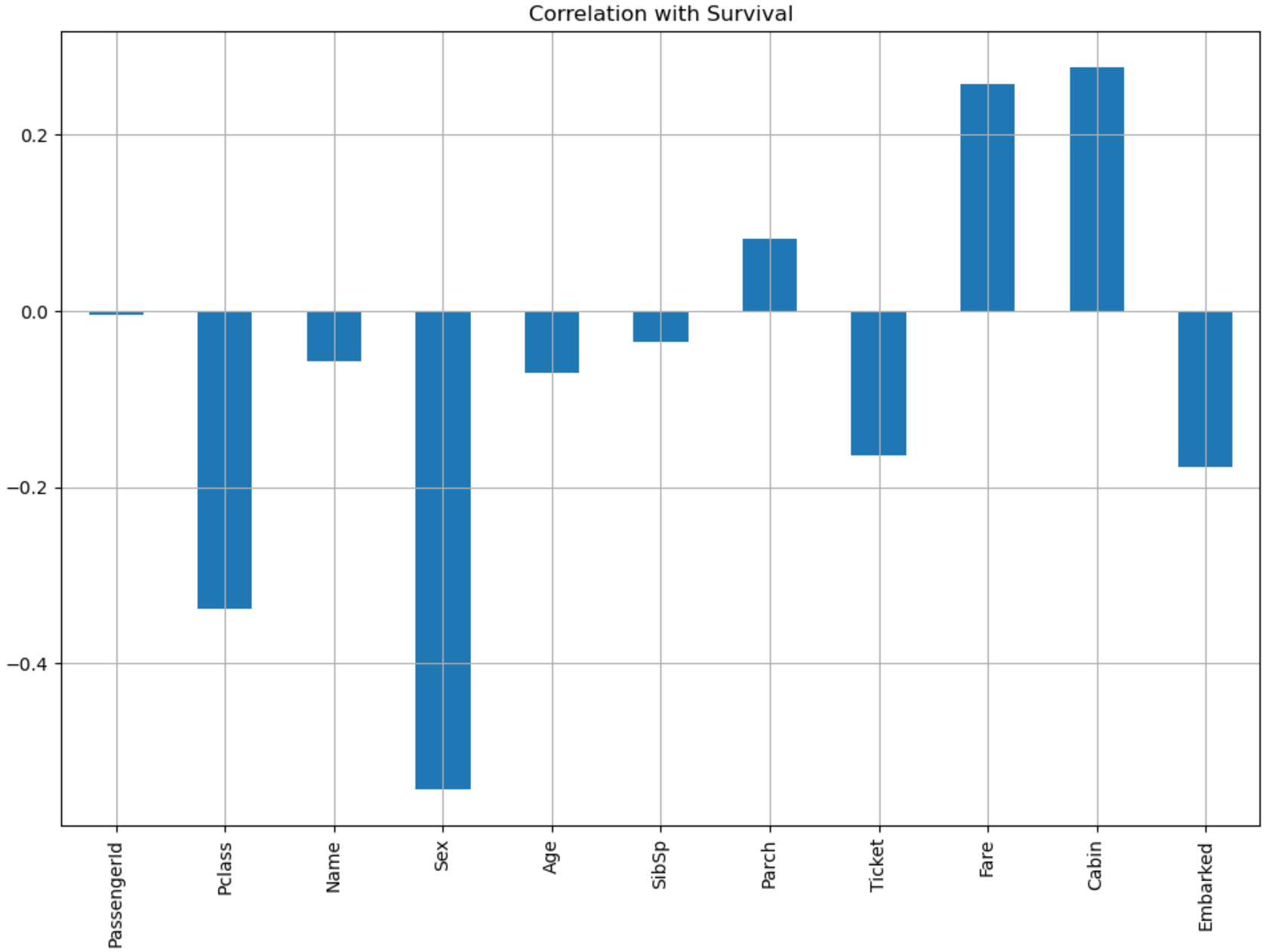
```
             Unique Values
Variable
PassengerId            891
Survived                 2
Pclass                   3
Name                   891
Sex                      2
Age                     89
SibSp                    7
Parch                    7
Ticket                 681
Fare                   248
Cabin                  148
Embarked                 4
Binary Variables ['Survived', 'Sex']
Non Binary Variables ['PassengerId', 'Pclass', 'Name', 'Age', 'SibSp', 'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked']
Categorical Variables ['Name', 'Sex', 'Ticket', 'Cabin', 'Embarked']
Numeric Variables ['PassengerId', 'Survived', 'Pclass', 'Age', 'SibSp', 'Parch', 'Fare']
```

Data Analysis



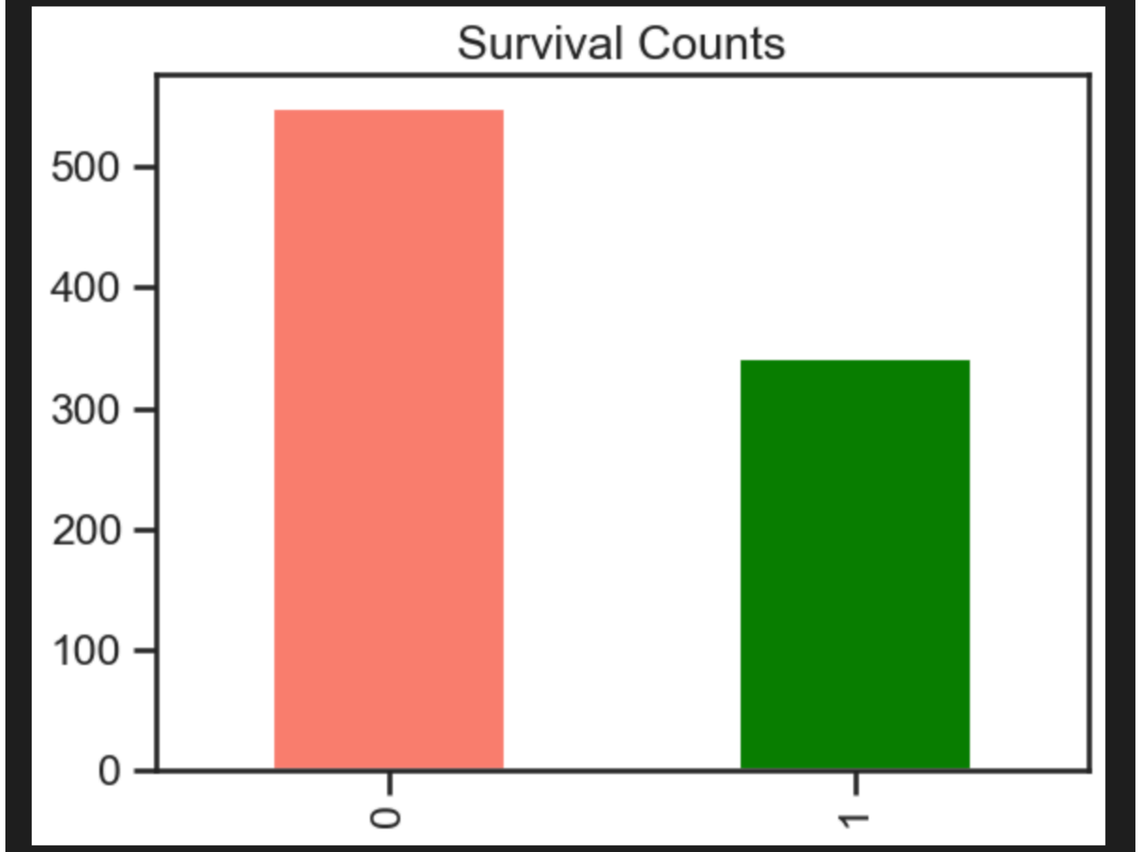
Two obvious features have a low correlation with the chance of survival: these are the Passenger ID and the passenger's name.

Other than that, SibSp has the lowest correlation of the rest of the features, with Sex and Class being the highest correlated features


Correlation with Survival
0.2
0.0
−0.2
−0.4
PassengerId
Pclass
Name
Sex
Age
SibSp
Parch
Ticket
Fare
Cabin
Embarked

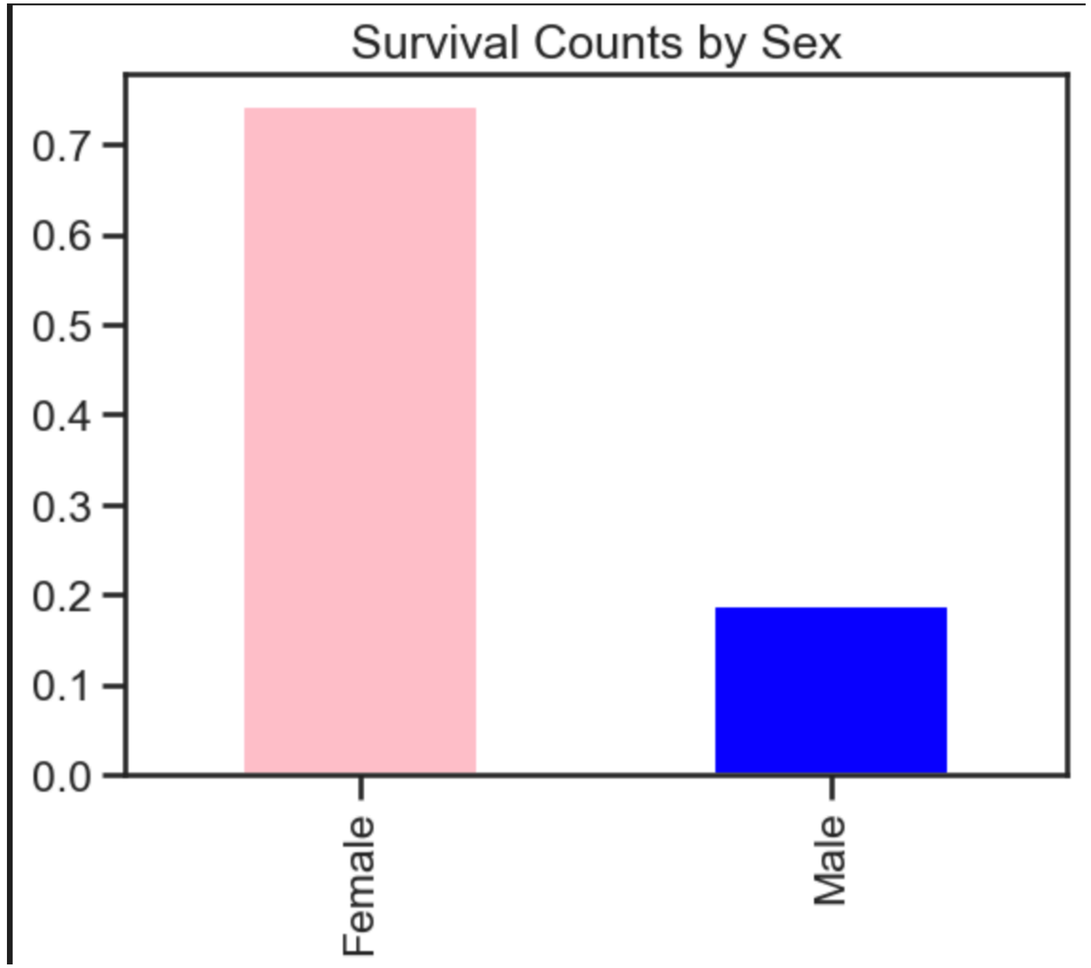
Data Analysis

```
0    549
1    342
Name: Survived, dtype: int64

<AxesSubplot:title={'center':'Survival Counts'}>
```



In our dataset, 549 people died and 342 survived in total. The data we are trying to predict is somewhat out of balance but still a fair split.

Data Analysis

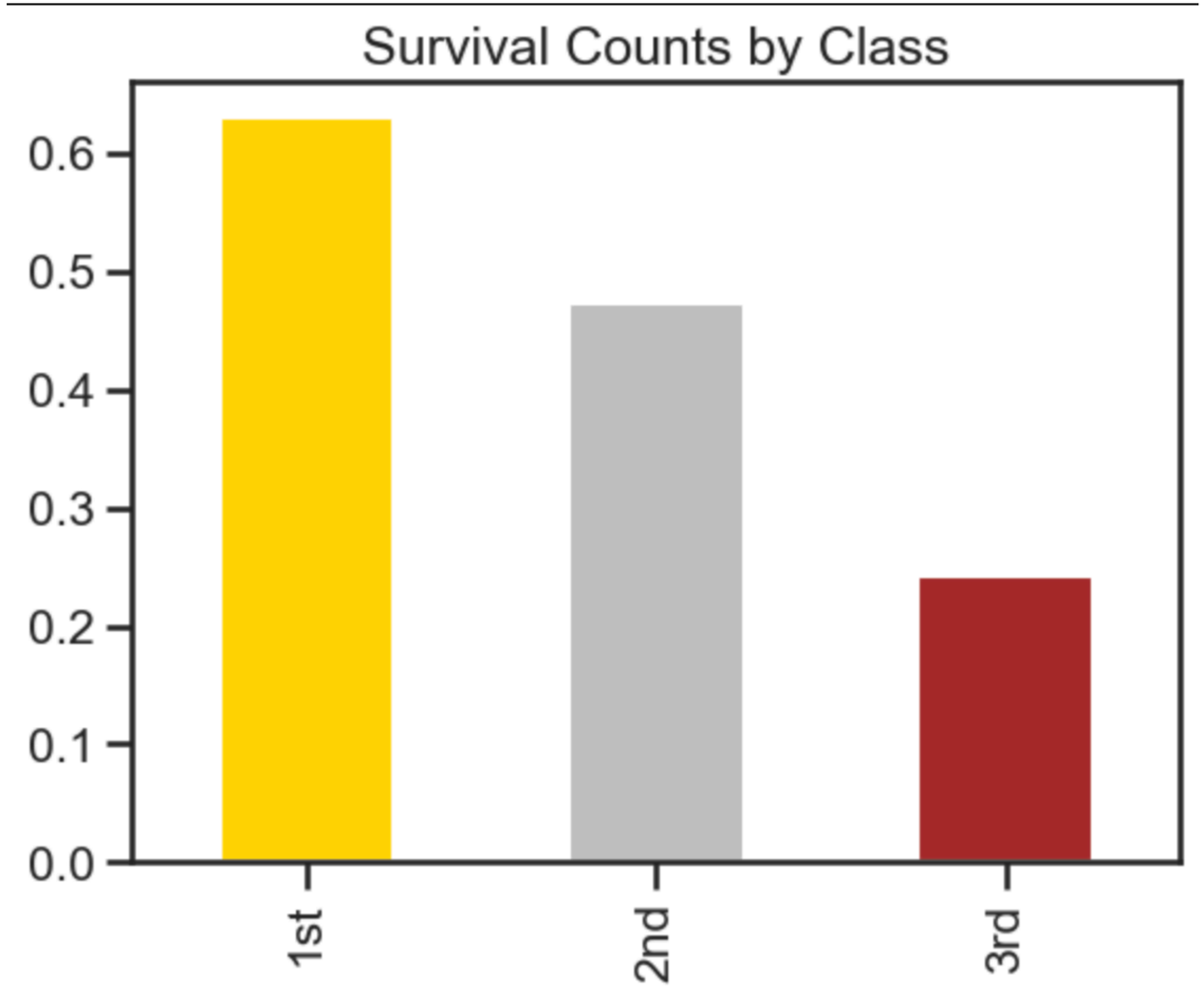


The data is highly skewed when split by sex, as the chance of survival for women is approximately three times higher than for men.

Data Analysis

Similarly, we see there is a considerable difference between the cancers of survival of passengers in the three ticket classes.

Passengers in 1st class were again approximately three times more likely to survive as those in 3rd class, with passengers in second class roughly in the middle.



Machine Learning Section

In this section we will use several algorithms to predict whether a passenger survived or not the Titanic's shipwreck based on the provided data. These models will be the Support Vector Machine, the k-Nearest Neighbours algorithm and several types of Logistic Regression (standard, L1 regularised and L2 regularised).

We will look at several techniques to improve our data and our models, such as standard scaling and label encoding for our dataset and F1 score, accuracy and precision scores for our models

Splitting the Data

```
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
feature_cols = [col_name for col_name in data.columns if col_name != 'Survived']



# Get the split indexes
strat_shuf_split = StratifiedShuffleSplit(n_splits=1, test_size=0.3, random_state=42)

train_idx, test_idx = next(strat_shuf_split.split(data[feature_cols], data.Survived))

# Create the dataframes
X_train = data.loc[train_idx, feature_cols]
y_train = data.loc[train_idx, 'Survived']

X_test  = data.loc[test_idx, feature_cols]
y_test  = data.loc[test_idx, 'Survived']
```

Logistic Regression

For the report we use three different types of logistic regression: standard, L1 regularised and L2 regularised, with respective cost functions:

L1 Regularization

$$\text{Cost} = \sum_{i=0}^{N} (y_i - \sum_{j=0}^{M} x_{ij} W_j)^2 + \lambda \sum_{j=0}^{M} |W_j|$$

L2 Regularization

$$\text{Cost} = \sum_{i=0}^{N} (y_i - \sum_{j=0}^{M} x_{ij} W_j)^2 + \lambda \sum_{j=0}^{M} W_j^2$$

Loss function

Regularization Term

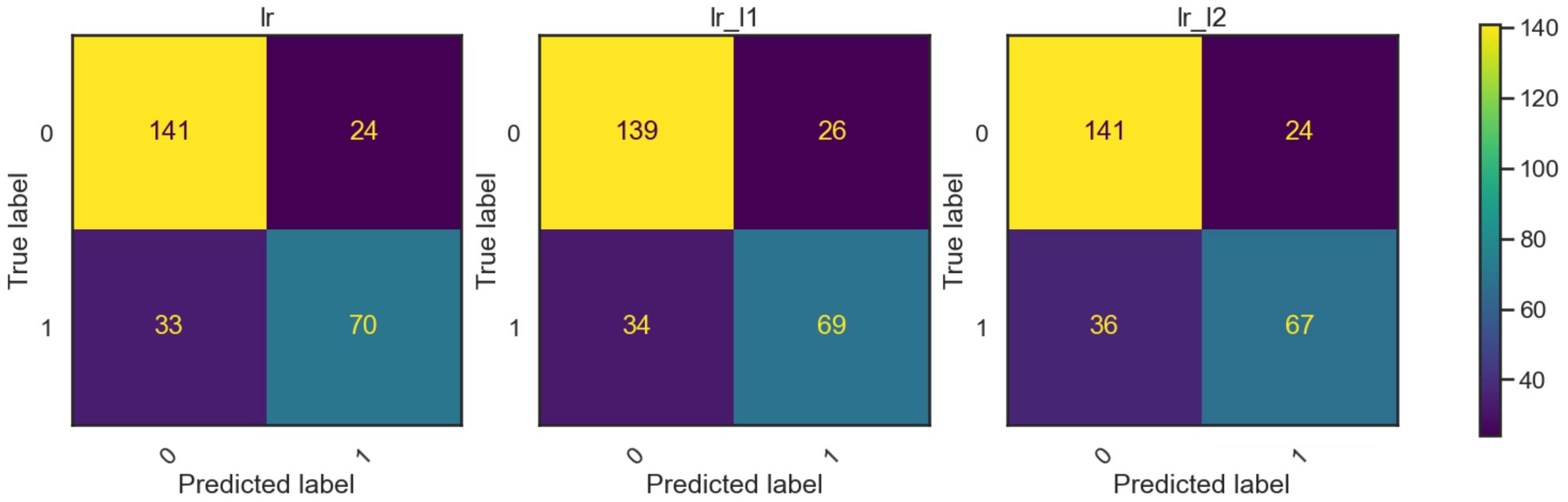
Logistic Regression

Syntax:

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


#Regular Logistic Regression
lr = LogisticRegression(solver='liblinear')
lr = lr.fit(X_train, y_train)


#L1 Regularised Logistic Regression
lr_l1 = LogisticRegressionCV(Cs=10, cv=4, penalty='l1', solver='liblinear')
lr_l1 = lr_l1.fit(X_train, y_train)


#L2 Regularised Logistic Regression
lr_l2 = LogisticRegressionCV(Cs=10, cv=4, penalty='l2', solver='liblinear')
lr_l2 = lr_l2.fit(X_train, y_train)
```

Logistic Regression

We plot our findings in three respective confusion matrices:



Logistic Regression

We find L2 regularised logistic regression to be the most effective and the one with the most accurate predictions. We obtain the model's statistics via the code below:

	0	1	accuracy	macro avg	weighted avg
precision	0.80	0.74	0.78	0.77	0.77
recall	0.85	0.65	0.78	0.75	0.78
f1-score	0.82	0.69	0.78	0.76	0.77
support	165.00	103.00	0.78	268.00	268.00

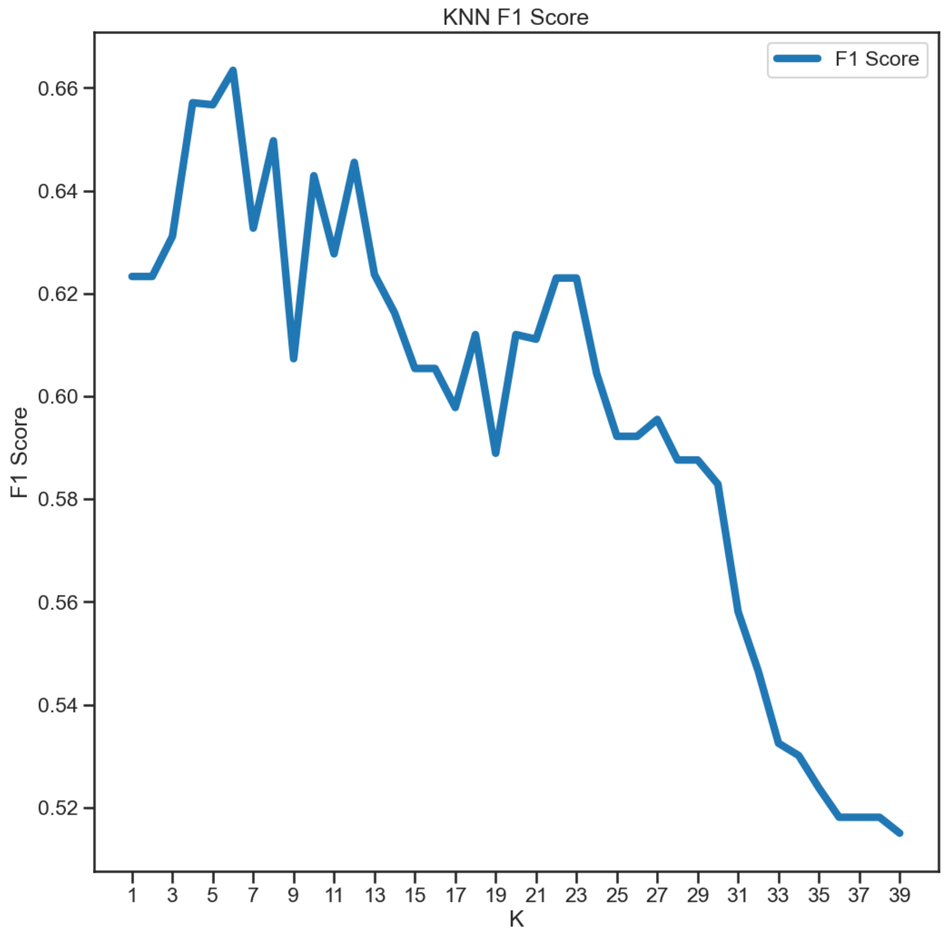
```
#Regular Logistic Regression
lr = LogisticRegression(solver='liblinear')
lr = lr.fit(X_train, y_train)
lr_stats = pd.DataFrame(classification_report(y_test, y_pred, output_dict=True))
```

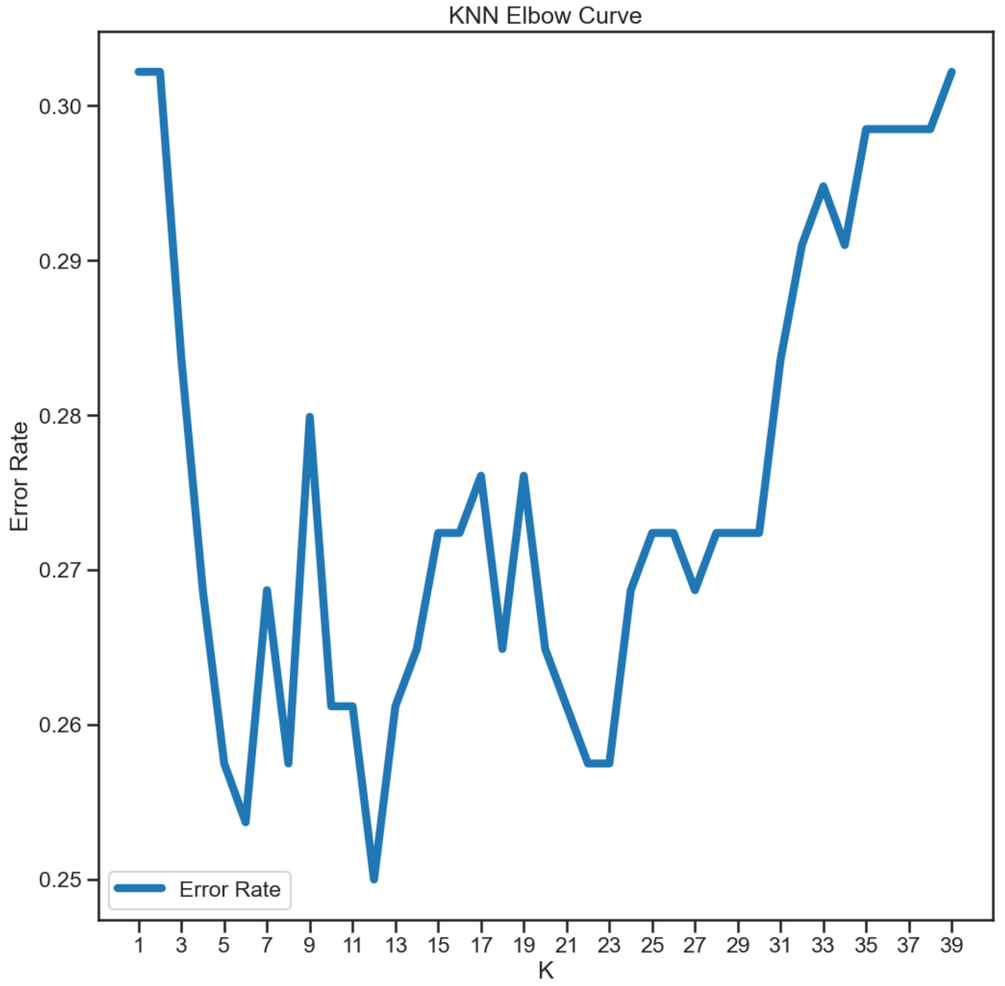
KNN Algorithm

We first try to obtain the fluctuations of the F1-score for different k, and the KNN elbow curve. We then find the most effective k for the algorithm.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report, f1_score, ConfusionMatrixDisplay

max_k = 40
f1_scores = list()
error_rates = list() # 1-accuracy

for k in range(1, max_k):

    knn = KNeighborsClassifier(n_neighbors=k, weights='distance')
    knn = knn.fit(X_train, y_train)

    y_pred = knn.predict(X_test)
    f1 = f1_score(y_pred, y_test)
    f1_scores.append((k, round(f1_score(y_test, y_pred), 4)))
    error = 1-round(accuracy_score(y_test, y_pred), 4)
    error_rates.append((k, error))

f1_results = pd.DataFrame(f1_scores, columns=['K', 'F1 Score'])
error_results = pd.DataFrame(error_rates, columns=['K', 'Error Rate'])
```

KNN Algorithm





KNN Algorithm

The most effective is k=12. Using this, we run the 12-NN algorithm with the code below

```
#Get the best k
min_error_index = error_results['Error Rate'].idxmin()
error_results.loc[min_error_index]
```

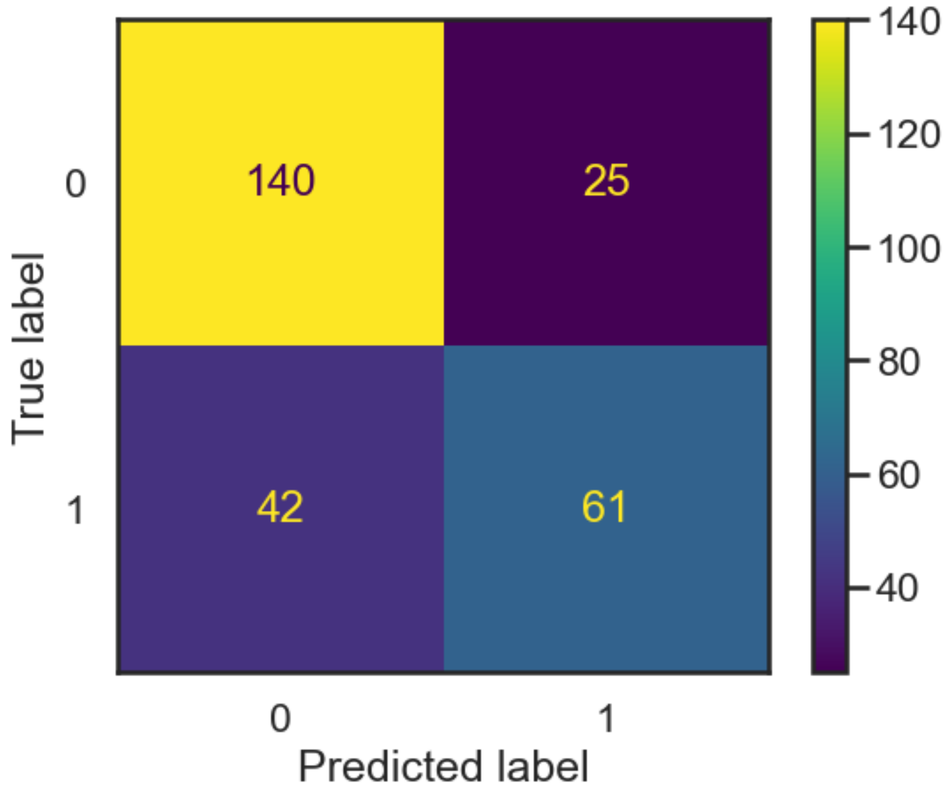
✓ 0.1s

```
K            12.00
Error Rate    0.25
Name: 11, dtype: float64
```

```
#Best possible model

knn = KNeighborsClassifier(n_neighbors=12, weights='distance')
knn = knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
```

KNN Algorithm

Findings:



	0	1	accuracy	macro avg	weighted avg
precision	0.77	0.71	0.75	0.74	0.75
recall	0.85	0.59	0.75	0.72	0.75
f1-score	0.81	0.65	0.75	0.73	0.74
support	165.00	103.00	0.75	268.00	268.00

Random Forest

We firstly fit a simple random forest classifier, using the whole of our columns as a maximum number of features and setting the number of trees to 20

We also create functions to check the model's accuracy and the correlation between our features (syntax of this in the next slide)

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV


#Simple Random Forest Search

num_features=data.shape[1]
max_features=round(np.sqrt(num_features))-1

RanFor = RandomForestClassifier( max_features=max_features,n_estimators=20, random_state=0)
RanFor.fit(X_train,y_train)

print(get_accuracy(X_train, X_test, y_train, y_test, RanFor))
print(get_correlation(X_test, y_test,RanFor).style.background_gradient(cmap='coolwarm'))
```

```
{'test Accuracy': 0.8097014925373134, 'trian Accuracy': 0.9967897271268058}
Average correlation between predictors:  0.320238849663336
```

Random Forest

```python
from sklearn import metrics

def get_accuracy(X_train, X_test, y_train, y_test, model):
    return  {"test Accuracy":metrics.accuracy_score(y_test, model.predict(X_test)),"trian Accuracy": metrics.accuracy_score(y_train, model.predict(X_train))}



def get_correlation(X_test, y_test,models):
    #This function calculates the average correlation between predictors
    n_estimators=len(models.estimators_)
    prediction=np.zeros((y_test.shape[0],n_estimators))
    predictions=pd.DataFrame({'estimator '+str(n+1):[] for n in range(n_estimators)})

    for key,model in zip(predictions.keys(),models.estimators_):
        predictions[key]=model.predict(X_test.to_numpy())

    corr=predictions.corr()
    print("Average correlation between predictors: ", corr.mean().mean()-1/n_estimators)
    return corr
```

Random Forest

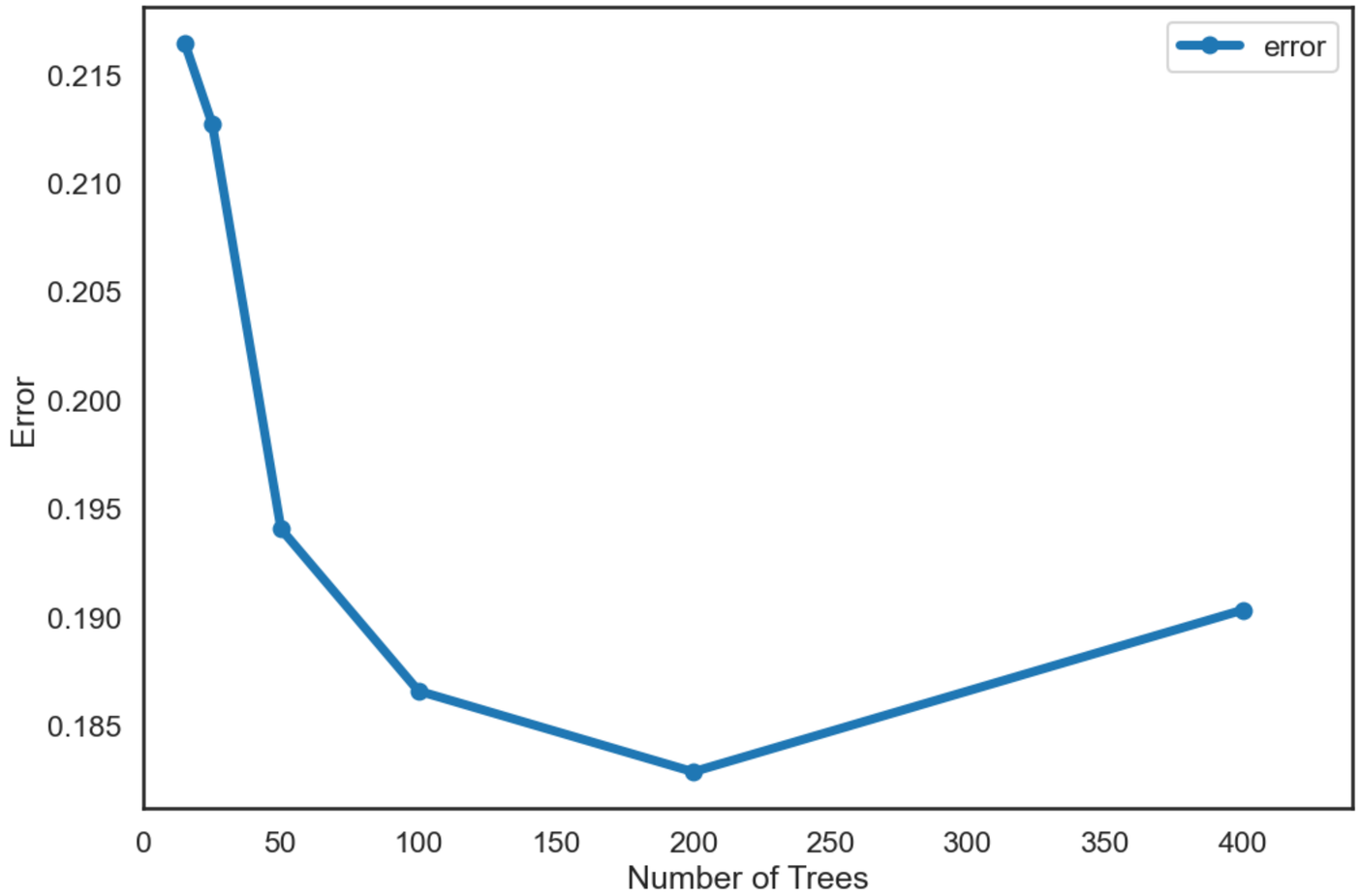
Now we iterate a random forest search through several possible maximum number of trees. We then plot the error for each one of the possibilities

```python
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score

error_list = list()

# Iterate through various possibilities for number of trees #Faster
tree_list = [15, 25, 50, 100, 200, 400]
for n_trees in tree_list:


    GBC = GradientBoostingClassifier(n_estimators=n_trees, random_state=42)
    print(f'Fitting model with {n_trees} trees')
    GBC.fit(X_train.values, y_train.values)
    y_pred = GBC.predict(X_test)

    error = 1.0 - accuracy_score(y_test, y_pred)
    error_list.append(pd.Series({'n_trees': n_trees, 'error': error}))

error_df = pd.concat(error_list, axis=1).T.set_index('n_trees')
```

n_trees	error
15.0	0.216418
25.0	0.212687
50.0	0.194030
100.0	0.186567
200.0	0.182836
400.0	0.190299

The minimum error must be in the range between 100 and 200 trees. Plots of the accuracy can be found in the next slide.

Random Forest



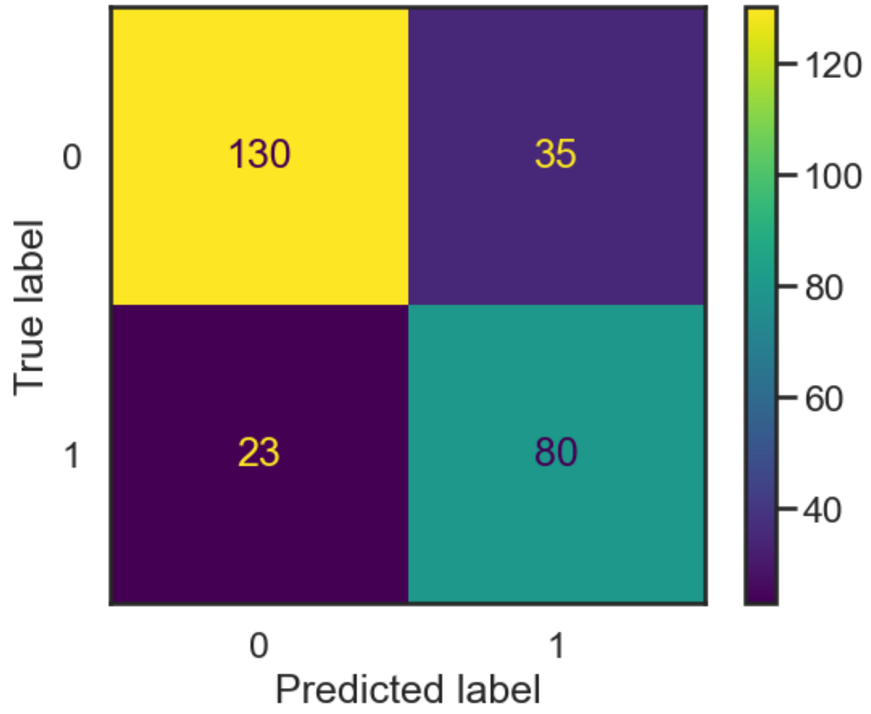

```
sns.set_context('talk')
sns.set_style('white')

# Create the plot
ax = error_df.plot(marker='o', figsize=(12, 8), linewidth=5)

# Set parameters
ax.set(xlabel='Number of Trees', ylabel='Error')
ax.set_xlim(0, max(error_df.index)*1.1);
```

✓ 0.3s

Support Vector Classifier

Findings:

```
from sklearn.svm import SVC

kwargs = {'kernel': 'rbf'}
svc = SVC(**kwargs)

SVC_cl = svc.fit(X_train, y_train)
y_pred = SVC_cl.predict(X_test)
SVC_cl_report = pd.DataFrame(classification_report(y_test, y_pred, output_dict=True))
SVC_cl_report

cm = confusion_matrix(y_test, y_pred, labels=SVC_cl.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=SVC_cl.classes_)
disp.plot()
plt.grid(False)
plt.show()
```



Next steps

We have found that features such as sex and passenger's ticket class had a major impact on the likeliness of a passenger surviving.

Similar models and analytics could be expanded further to predict likeliness of survival of any other type of accident, i.e. one could look at features such as sex, civil status etc when looking at survival statistics in plane crashes, traffic accidents etc. This is commonly used in the insurance business for pricing and risk analysis.