KNN

In this lab, you are going to practice data preprocessing and building the KNN model using MLLib and other spark tools. Go to the following website and download the dataset. Training data is already given to you as train.csv. Your goal is to build a model that can accurately predict survival in test.csv. https://www.kaggle.com/competitions/titanic/overview (https://www.kaggle.com/competitions/titanic/overview)

Part 1 - Build a KNN classifier to classify the dataset.

- • Write standard scaler from scratch - do not scale/z-score features using off-the-shelf scaler from sklearn
- • Scale the data(where appropriate) using standard scaler
- • Split the dataset into training and testing
- • Determine the K value, and create a visualization of the accuracy. Report the best Kvalue
- • Run 5 fold cross validations - report mean and standard deviation
- • Evaluate using confusion matrix
- • Use MARKDOWN cell to explain the accuracy of your model

Data Dictionary

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

Variable Notes

- pclass: A proxy for socio-economic status (SES) 1st = Upper 2nd = Middle 3rd = Lower
- age: Age is fractional if less than 1. If the age is estimated, is it in the form of xx.5
- sibsp: The dataset defines family relations in this way... Sibling = brother, sister, stepbrother, stepsister Spouse = husband, wife (mistresses and fiancés were ignored)
- parch: The dataset defines family relations in this way... Parent = mother, father Child = daughter, son, stepdaughter, stepson Some children travelled only with a nanny, therefore parch=0 for them.

In [1]:
```
#Importing relevant libraries
import warnings
warnings.filterwarnings('ignore')

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
pd.set_option("display.float_format", lambda x: "%.3f" % x)
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
```

In [2]:
```
#importing dataset
data = pd.read_csv('/Users/cheerycheena/Downloads/train.csv')
```

In [3]:
```
data.head(10)
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.000	1	0	A/5 21171	7.250
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.000	1	0	PC 17599	71.283
2	3	1	3	Heikkinen, Miss. Laina	female	26.000	0	0	STON/O2. 3101282	7.925
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.000	1	0	113803	53.100
4	5	0	3	Allen, Mr. William Henry	male	35.000	0	0	373450	8.050
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.458
6	7	0	1	McCarthy, Mr. Timothy J	male	54.000	0	0	17463	51.862
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.000	3	1	349909	21.075
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.000	0	2	347742	11.133
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.000	1	0	237736	30.071

In [4]:
```
#Checking for duplicate record on train set
data.duplicated().sum()
```

Out[4]:
```
0
```

In [5]:
```
#Checking for missing values
pd.DataFrame(data={'% of Missing Values':round(data.isna().sum()/data.
```

Out[5]:

	% of Missing Values
PassengerId	0.000
Survived	0.000
Pclass	0.000
Name	0.000
Sex	0.000
Age	19.870
SibSp	0.000
Parch	0.000
Ticket	0.000
Fare	0.000
Cabin	77.100
Embarked	0.220

- Missing values are found in the Age, Cabin and embarked attributes. The percentages are quite high.
- Age has 19.87% missing values out of the total observations.
- Cabin has 77.10% missing values out of the total observations.

Making sense of the data (Exploratory Data Analysis)

In [6]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

- There are missing data in the dataset.
- There are 891 records and 12 attributes.
- Name, Passenger ID and Ticket no do not provide any predictive value and are therefore not necesary for our model. They should be dropped.

In [7]:
```
data.nunique()
```

Out[7]:
```
PassengerId    891
Survived         2
Pclass           3
Name           891
Sex              2
Age             88
SibSp            7
Parch            7
Ticket         681
Fare           248
Cabin          147
Embarked         3
dtype: int64
```

- This confirms that PassengerID and Name should be dropped because they will add no value to the model.

In [8]:
```
#Getting the number of people that survived in each socio-economic sta
data.groupby('Pclass')['Survived'].sum()
```

Out[8]:
```
Pclass
1    136
2     87
3    119
Name: Survived, dtype: int64
```

- The upper class survived the most with 136 observations.
- The middle class survived the least with 87 observations.

In [9]:
```
data.describe().T
```

Out[9]:

	count	mean	std	min	25%	50%	75%	max
PassengerId	891.000	446.000	257.354	1.000	223.500	446.000	668.500	891.000
Survived	891.000	0.384	0.487	0.000	0.000	0.000	1.000	1.000
Pclass	891.000	2.309	0.836	1.000	2.000	3.000	3.000	3.000
Age	714.000	29.699	14.526	0.420	20.125	28.000	38.000	80.000
SibSp	891.000	0.523	1.103	0.000	0.000	0.000	1.000	8.000
Parch	891.000	0.382	0.806	0.000	0.000	0.000	0.000	6.000
Fare	891.000	32.204	49.693	0.000	7.910	14.454	31.000	512.329

- Survived is the output label and should be a categorical variable.
- Max age in the dataset is 80. the mean and median is 28 and 29.7 respectively. The fractions in the age represents estimated ages and records of people less than one year old.
- There are outliers in the Fare attribute.

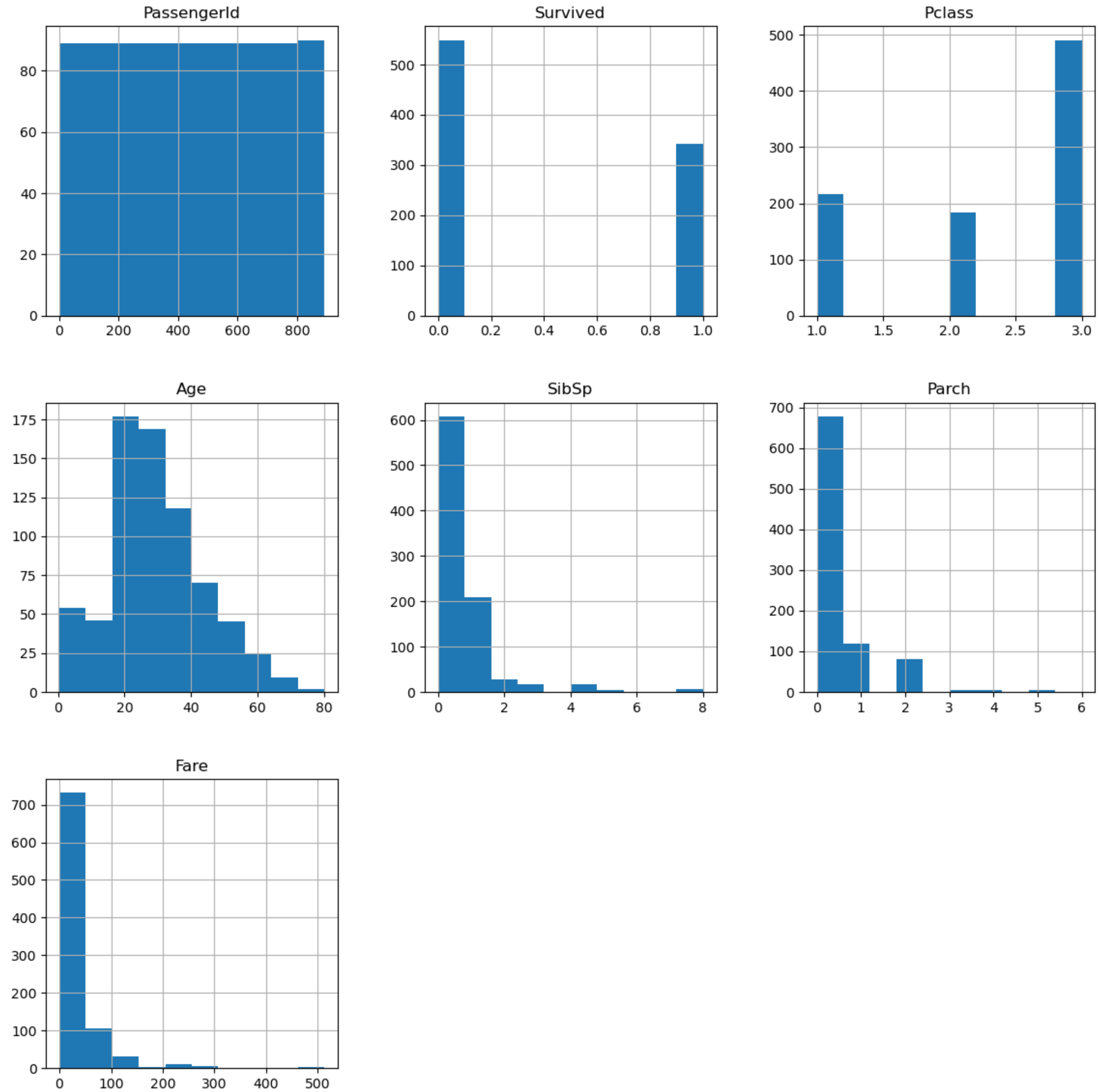
In [10]:
```
data.describe(include='object').T
```

Out[10]:

	count	unique	top	freq
Name	891	891	Braund, Mr. Owen Harris	1
Sex	891	2	male	577
Ticket	891	681	347082	7
Cabin	204	147	B96 B98	4
Embarked	889	3	S	644

- Males are the majority with a count of 577
- Port of embarkment S (Southampton) has the highest record of 644.

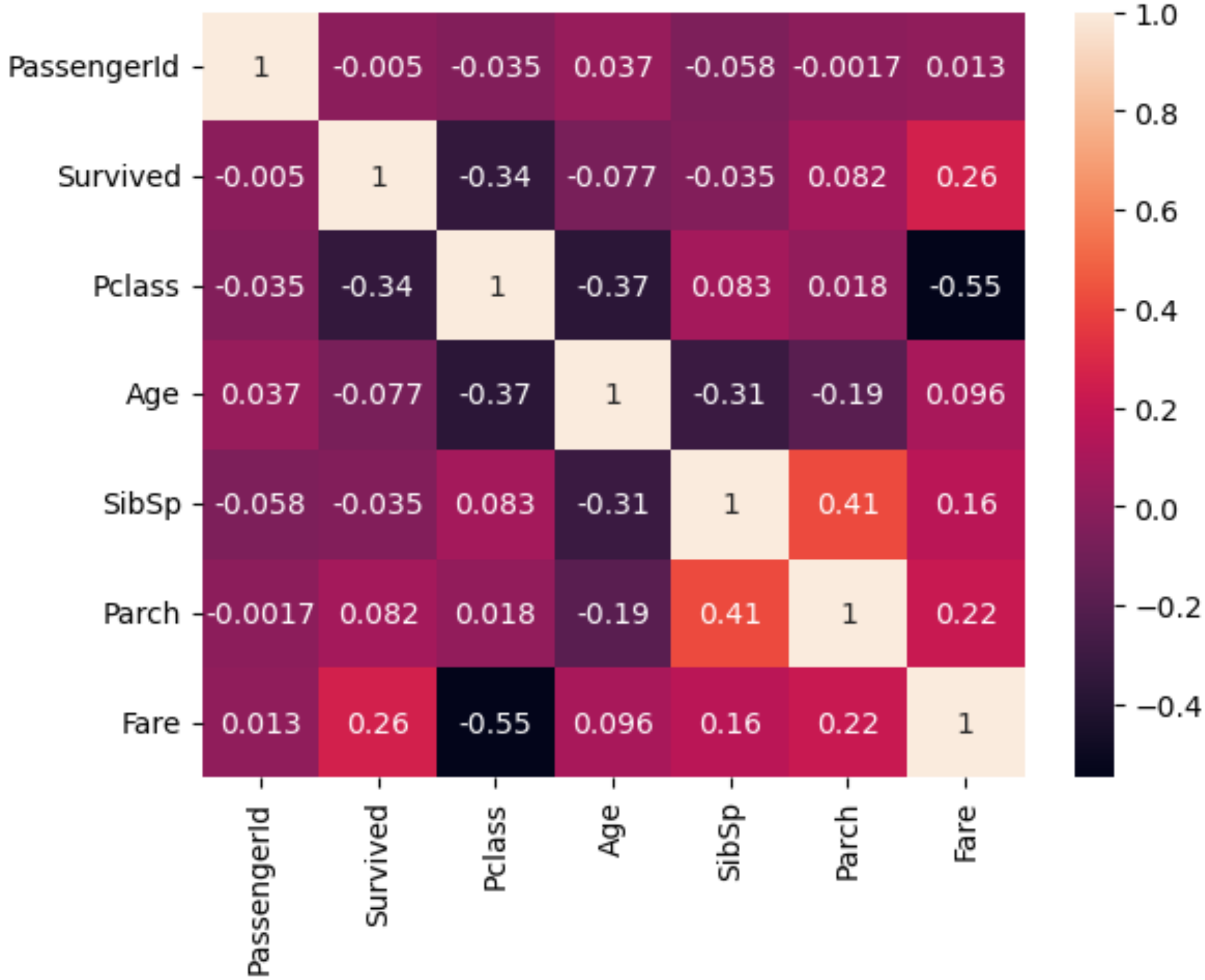
In [11]:
```
data.hist(figsize = (14,14));
plt.show()
```



The dataset has:

- A record of more people who did not survive than those who did.
- Lower class (pclass=3) is most frequent.
- Most of the people are in the age range of 19-22years.
- Most Fare fell within the range of 0 to 150.

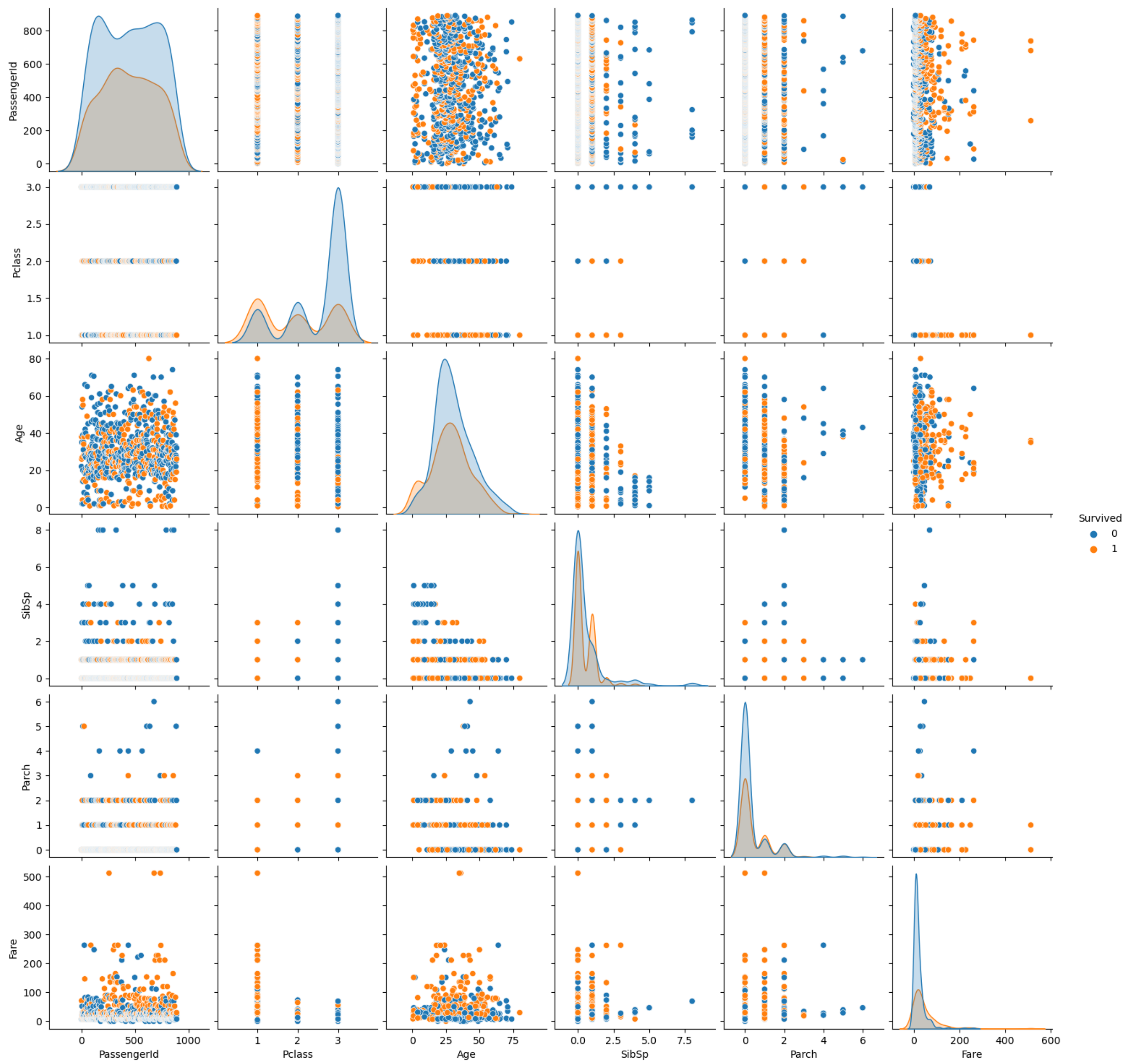
In [12]:
```
sns.heatmap(data=data.corr(), annot=True);
```



From the heatmap:

- Fair and survived are more correlated to each other than other attributes.
- Same goes for Parch and Sibsp.

In [13]:
```
sns.pairplot(data=data, hue='Survived');
```



- Those with lower age and higher fare in pclass of 1 survived greatly.

In [14]:
```
#Making a copy of the dataset to avoid changes to the original dataset
df = data.copy()
df.tail(5)
```

Out[14]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
886	887	0	2	Montvila, Rev. Juozas	male	27.000	0	0	211536	13.000
887	888	1	1	Graham, Miss. Margaret Edith	female	19.000	0	0	112053	30.000
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.450
889	890	1	1	Behr, Mr. Karl Howell	male	26.000	0	0	111369	30.000
890	891	0	3	Dooley, Mr. Patrick	male	32.000	0	0	370376	7.750

In [15]:
```
#dropping PassengerID, Name and Cabin and Ticket attributes.

df.drop(columns = ['PassengerId', 'Name', 'Cabin', 'Ticket'], inplace=
```

Dealing with missing values

The Age and embarked attribute has missing values. We will use simple imputer to affix the missing values.

In [16]:
```
df.isna().sum()
```

Out[16]:
```
Survived      0
Pclass        0
Sex           0
Age         177
SibSp         0
Parch         0
Fare          0
Embarked      2
dtype: int64
```

In [17]:
```
#For Age, we will replace the missing values with the median value.
median_value = df['Age'].median()
df["Age"].fillna(value = median_value, inplace=True)
```

In [18]:
```
#For Embarked, missing values constitute of only 2 observations. We ca
#significant effect on dataset.
df=df.dropna()
```

In [19]:
```
df.isna().sum()
```

Out[19]:
```
Survived    0
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked    0
dtype: int64
```

- All missing values have been dealt with.

One Hot Encoding.

One hot encoding converts categorical data needs to be represented in numerical format to enable us build the model. The Sex and Embarked column are the two columns that needs to be one-hot encoded.

In [20]:
```
df1 = pd.get_dummies(df, columns=['Sex', 'Embarked'])
df1
```

Out[20]:

	Survived	Pclass	Age	SibSp	Parch	Fare	Sex_female	Sex_male	Embarked_C	Eml
0	0	3	22.000	1	0	7.250	0	1	0	
1	1	1	38.000	1	0	71.283	1	0	1	
2	1	3	26.000	0	0	7.925	1	0	0	
3	1	1	35.000	1	0	53.100	1	0	0	
4	0	3	35.000	0	0	8.050	0	1	0	
...	...	...	...	...	...	...	...	...	...	
886	0	2	27.000	0	0	13.000	0	1	0	
887	1	1	19.000	0	0	30.000	1	0	0	
888	0	3	28.000	1	2	23.450	1	0	0	
889	1	1	26.000	0	0	30.000	0	1	1	
890	0	3	32.000	0	0	7.750	0	1	0	

889 rows × 11 columns

In [21]:
```
#dropping the Sex_female column. It's a redundant attribute because Se

df1 = df1.drop(columns='Sex_female')
df1
```

Out[21]:

	Survived	Pclass	Age	SibSp	Parch	Fare	Sex_male	Embarked_C	Embarked_Q	Er
0	0	3	22.000	1	0	7.250	1	0	0	
1	1	1	38.000	1	0	71.283	0	1	0	
2	1	3	26.000	0	0	7.925	0	0	0	
3	1	1	35.000	1	0	53.100	0	0	0	
4	0	3	35.000	0	0	8.050	1	0	0	
...	...	...	...	...	...	...	...	...	...	
886	0	2	27.000	0	0	13.000	1	0	0	
887	1	1	19.000	0	0	30.000	0	0	0	
888	0	3	28.000	1	2	23.450	0	0	0	
889	1	1	26.000	0	0	30.000	1	1	0	
890	0	3	32.000	0	0	7.750	1	0	1	

889 rows × 10 columns

Building the Model

The train and test data set has been given differently, therefore there in no need to split the dataset.

For the train dataset:

In [22]:
```
X = df1.drop(columns='Survived', axis = 1)
Y = df1['Survived']
```

Standardization of the dataset

Most of the attributes have values btw 0 and 1. Therefore we will only standardize the age and fare attributes.

In [23]:
```
# defining a function to standardize the dataset
def standardize_dataset(df):
    # Calculate the mean and standard deviation of each attribute
    means = df.mean(axis=0)
    stds = df.std(axis=0)

    # Standardize each attribute
    for col in df.columns:
        df[col] = (df[col] - means[col]) / stds[col]
    return df
```

In [24]:
```
# Standardize the dataset
X = standardize_dataset(X)
X
```

Out[24]:

	Pclass	Age	SibSp	Parch	Fare	Sex_male	Embarked_C	Embarked_Q	Embarked_S
0	0.825	-0.563	0.431	-0.474	-0.500	0.735	-0.482	-0.308	0.616
1	-1.571	0.669	0.431	-0.474	0.789	-1.359	2.070	-0.308	-1.620
2	0.825	-0.255	-0.475	-0.474	-0.486	-1.359	-0.482	-0.308	0.616
3	-1.571	0.438	0.431	-0.474	0.423	-1.359	-0.482	-0.308	0.616
4	0.825	0.438	-0.475	-0.474	-0.484	0.735	-0.482	-0.308	0.616
...	...	...	...	...	...	...	...	...	...
886	-0.373	-0.178	-0.475	-0.474	-0.384	0.735	-0.482	-0.308	0.616
887	-1.571	-0.794	-0.475	-0.474	-0.042	-1.359	-0.482	-0.308	0.616
888	0.825	-0.101	0.431	2.005	-0.174	-1.359	-0.482	-0.308	0.616
889	-1.571	-0.255	-0.475	-0.474	-0.042	0.735	2.070	-0.308	-1.620
890	0.825	0.207	-0.475	-0.474	-0.490	0.735	-0.482	3.246	-1.620

889 rows × 9 columns

Splitting the dataset

In [25]:
```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.
print(X_train.shape, X_test.shape)
```

```
(622, 9) (267, 9)
```

Feature engineering for the Test Dataset.

This includes dealing with missing values, dropping unnecessary columns, normalization and hot encoding.

In [26]:
```
data_test = pd.read_csv('/Users/cheerycheena/Downloads/test.csv')
```

In [27]:
```
data_test.head(10)
```

Out[27]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Emb
0	892	3	Kelly, Mr. James	male	34.500	0	0	330911	7.829	NaN	
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.000	1	0	363272	7.000	NaN	
2	894	2	Myles, Mr. Thomas Francis	male	62.000	0	0	240276	9.688	NaN	
3	895	3	Wirz, Mr. Albert	male	27.000	0	0	315154	8.662	NaN	
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.000	1	1	3101298	12.287	NaN	
5	897	3	Svensson, Mr. Johan Cervin	male	14.000	0	0	7538	9.225	NaN	
6	898	3	Connolly, Miss. Kate	female	30.000	0	0	330972	7.629	NaN	
7	899	2	Caldwell, Mr. Albert Francis	male	26.000	1	1	248738	29.000	NaN	
8	900	3	Abrahim, Mrs. Joseph (Sophie Halaut Easu)	female	18.000	0	0	2657	7.229	NaN	
9	901	3	Davies, Mr. John Samuel	male	21.000	2	0	A/4 48871	24.150	NaN	

In [28]:
```
data_test.duplicated().sum()
```

Out[28]: 0

In [29]:
```
data_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  418 non-null    int64
 1   Pclass       418 non-null    int64
 2   Name         418 non-null    object
 3   Sex          418 non-null    object
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64
 6   Parch        418 non-null    int64
 7   Ticket       418 non-null    object
 8   Fare         417 non-null    float64
 9   Cabin        91 non-null     object
 10  Embarked     418 non-null    object
dtypes: float64(2), int64(4), object(5)
memory usage: 36.1+ KB
```

- There are issing values in the Age, Fare and Cabin columns.
- The missing values in the age column will be filled with median value.
- We need to drop the Cabin column and the 2 null rows in the Fare columns.
- The PassengerId, Ticket and Name column which do not have any predictive value, should also be dropped.

In [30]:
```
#For Age, we will replace the missing values with the median value.
median_value = data_test['Age'].median()
data_test["Age"].fillna(value = median_value, inplace=True)

#Dropping the Cabin column
data_test.drop(columns=['Cabin', 'PassengerId', 'Name', 'Ticket'], inp

#Dropping the rows with missing values
data_test = data_test.dropna()
data_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 417 entries, 0 to 417
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Pclass    417 non-null    int64  
 1   Sex       417 non-null    object 
 2   Age       417 non-null    float64
 3   SibSp     417 non-null    int64  
 4   Parch     417 non-null    int64  
 5   Fare      417 non-null    float64
 6   Embarked  417 non-null    object 
dtypes: float64(2), int64(3), object(2)
memory usage: 26.1+ KB
```

- the test data set has 417 observations and 6 columns

In [31]:
```
#One-Hot-Encoding for test dataset
data_test_encoded = pd.get_dummies(data_test.copy(), columns=['Sex', '

#Standardardization for the test dataset
data_test_stand = standardize_dataset(data_test_encoded.copy())
data_test_stand
```

Out[31]:

	Pclass	Age	SibSp	Parch	Fare	Sex_female	Sex_male	Embarked_C	Embarked_Q
0	0.874	0.394	-0.500	-0.400	-0.497	-0.756	0.756	-0.568	2.837
1	0.874	1.384	0.615	-0.400	-0.512	1.319	-1.319	-0.568	-0.352
2	-0.313	2.572	-0.500	-0.400	-0.464	-0.756	0.756	-0.568	2.837
3	0.874	-0.200	-0.500	-0.400	-0.482	-0.756	0.756	-0.568	-0.352
4	0.874	-0.596	0.615	0.618	-0.417	1.319	-1.319	-0.568	-0.352
...	...	...	...	...	...	...	...	...	...
413	0.874	-0.200	-0.500	-0.400	-0.493	-0.756	0.756	-0.568	-0.352
414	-1.501	0.750	-0.500	-0.400	1.311	1.319	-1.319	1.755	-0.352
415	0.874	0.711	-0.500	-0.400	-0.508	-0.756	0.756	-0.568	-0.352
416	0.874	-0.200	-0.500	-0.400	-0.493	-0.756	0.756	-0.568	-0.352
417	0.874	-0.200	0.615	0.618	-0.237	-0.756	0.756	1.755	-0.352

417 rows × 10 columns

In [32]:
```
x = data_test_stand.drop(columns='Sex_female')
```

In [33]:
```
X.shape
```

Out[33]: (889, 9)

In [34]:
```
x.shape
```

Out[34]: (417, 9)

Determining the value of neighbours(k)

There are various methods to determining the optimal value of k. This includes elbow method, silhoutte coefficient, gap statistics, etc.

The Elbow Method:

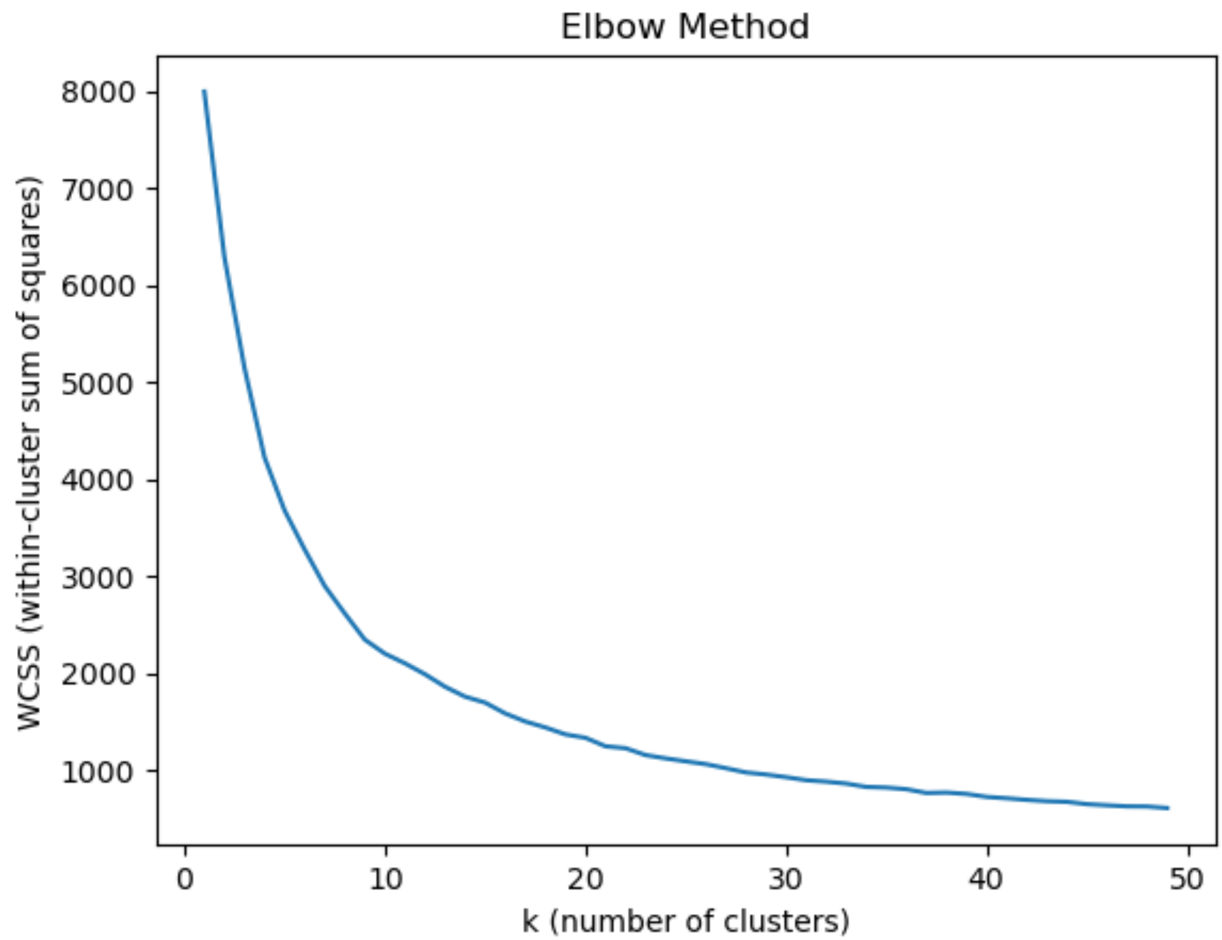
This Computes the within-cluster sum of squares (WCSS) for different values of k (the number of clusters) and plots the results.

In [35]:
```
def elbow_method(X, k_range):
    wcss = []
    for k in k_range:
        kmeans = KMeans(n_clusters=k)
        kmeans.fit(X)
        wcss.append(kmeans.inertia_)

    plt.plot(k_range, wcss)
    plt.xlabel('k (number of clusters)')
    plt.ylabel('WCSS (within-cluster sum of squares)')
    plt.title('Elbow Method')
    plt.show()
```

In [36]:
```
from sklearn.cluster import KMeans
```

In [37]:
```
k_range = range(1, 50)
elbow_method(X, k_range)
```



- We can see from the graph that 9,15 and 24 will make good number of clusters. We will build models with the different number of clusters and compare their accuracies.

KNN for 9 clusters

In [41]:
```
knn_9 = KNeighborsClassifier(n_neighbors=9)
knn_9.fit(X_train, y_train)
```

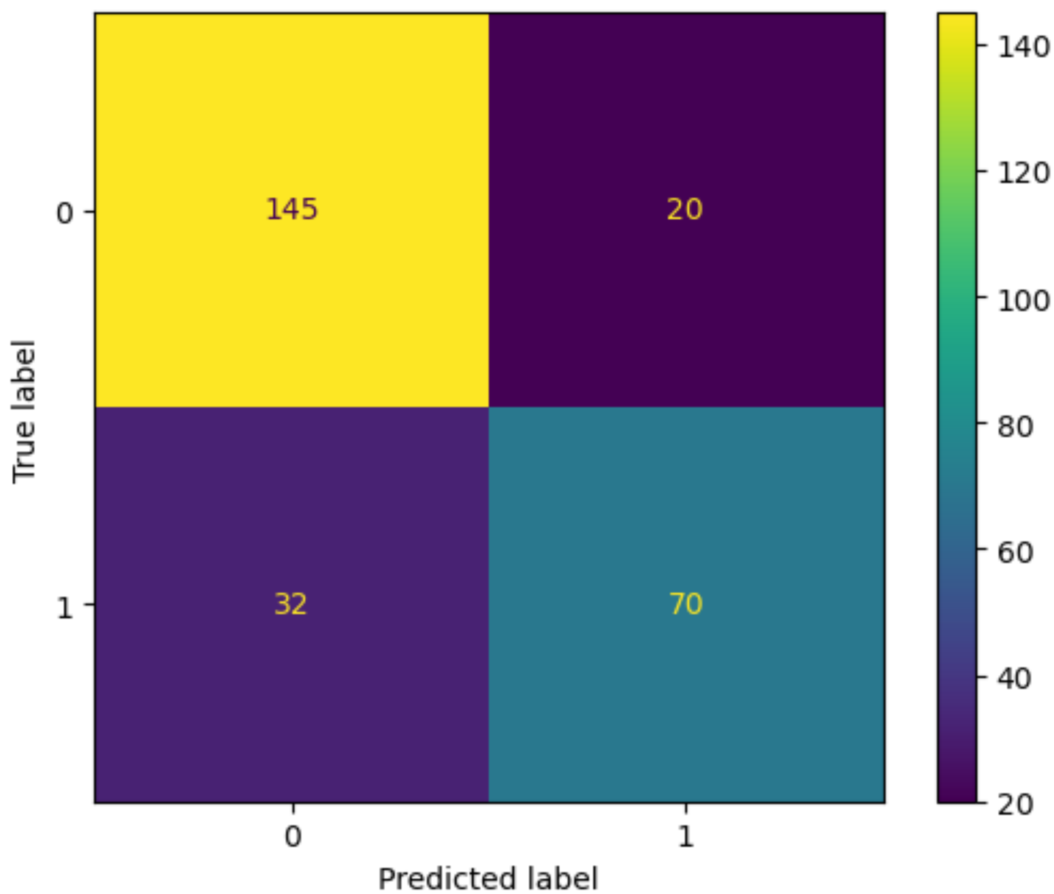
Out[41]:
```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=9)
```

In [43]:
```
y_pred = knn_9.predict(X_test)
```

In [45]:
```
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)
```

```
Accuracy: 0.8052434456928839
```

In [48]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.show()
```



KNN for 15 clusters

In [53]:
```
knn_15 = KNeighborsClassifier(n_neighbors=15)
knn_15.fit(X_train, y_train)
```

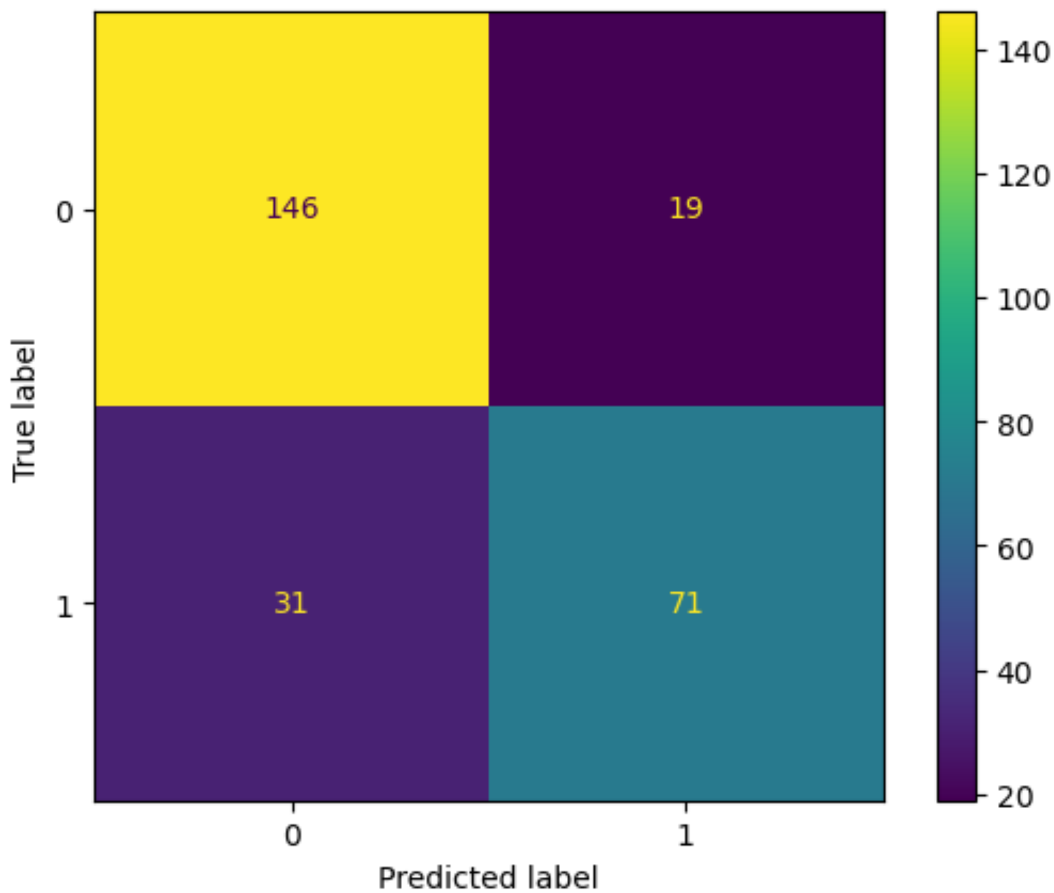
Out[53]:
```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=15)
```

In [54]:
```
y_pred_15 = knn_15.predict(X_test)
```

In [55]:
```
accuracy = accuracy_score(y_test, y_pred_15)
print('Accuracy:', accuracy)
```

```
Accuracy: 0.8127340823970037
```

In [56]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(y_test, y_pred_15)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.show()
```



In [57]:
```
knn_24 = KNeighborsClassifier(n_neighbors=24)
knn_24.fit(X_train, y_train)
```

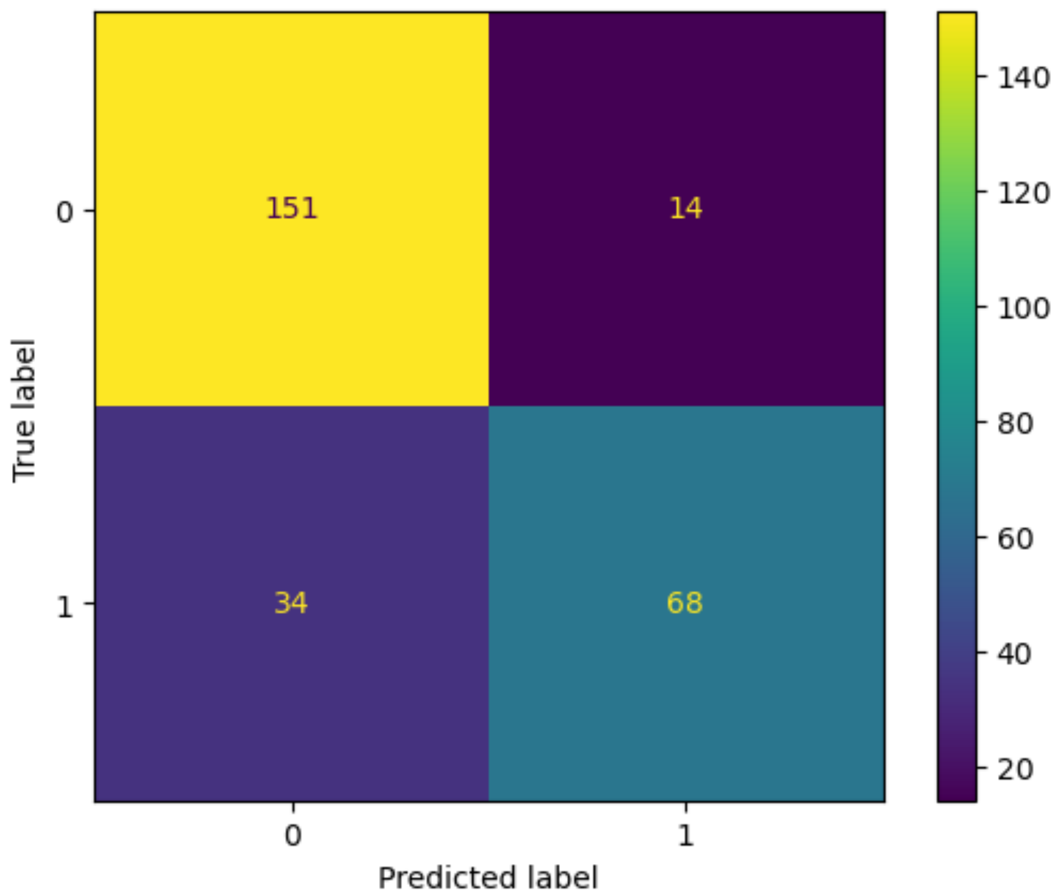
Out[57]:
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=24)
```

In [58]:
```
y_pred_24 = knn_24.predict(X_test)
```

In [60]:
```
accuracy = accuracy_score(y_test, y_pred_24)
print('Accuracy:', accuracy)
```

```
Accuracy: 0.8202247191011236
```

In [59]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(y_test, y_pred_24)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.show()
```



At k = 24:

The KNN classifier model has the highest accuracy with an accuracy of 82%.

In [ ]: