Task 09: Titanic Survival Prediction Using Various Classification models

Objectives:

1. Import Necessary Libraries
2. Read in and Explore the Data
3. Data Analysis
4. Data Visualization
5. Cleaning Data
6. Build model using Naives Bayes, Logistic Regression, Decision Trees, SVM and KNN.

1. Import necessary libraries

```
import numpy as np
import pandas as pd

#visualization libraries
import matplotlib.pyplot as plt
import seaborn as sns
```

2. Read in and explore the dataset

```
#import train and test CSV files
train = pd.read_csv("../input/train.csv")
test = pd.read_csv("../input/test.csv")

train.head()
```

3. Data Analysis

```
#get a list of the features within the dataset
print(train.columns)
```

- **Numerical Features:** Age (Continuous), Fare (Continuous), SibSp (Discrete), Parch (Discrete)
- **Categorical Features:** Survived, Sex, Embarked, Pclass
- **Alphanumeric Features:** Ticket, Cabin

What are the data types for each feature?

- Survived: int
- Pclass: int
- Name: string
- Sex: string
- Age: float
- SibSp: int
- Parch: int
- Ticket: string
- Fare: float
- Cabin: string
- Embarked: string

Some Observations:

- There are a total of 891 passengers in our training set.
- The Age feature is missing approximately 19.8% of its values. The Age feature seems important to survival, so we should probably attempt to fill these gaps.
- The Cabin feature is missing approximately 77.1% of its values. Since so much of the feature is missing, it would be hard to fill in the missing values. We'll probably drop these values from our dataset.
- The Embarked feature is missing 0.22% of its values, which should be relatively harmless.

Some Predictions:

- Sex: Females are more likely to survive.
- SibSp/Parch: People traveling alone are more likely to survive.
- Age: Young children are more likely to survive.
- Pclass: People of higher socioeconomic class are more likely to survive.

4. Data Visualization
   Visualize the data to see if the predictions were correct

   a. Visualize Gender feature
      #draw a bar plot of survival by sex

```
sns.barplot(x="Sex", y="Survived", data=train)

#print percentages of females vs. males that survive
print("Percentage of females who survived:",
train.loc[train["Sex"] == 'female', "Survived"].mean() * 100)

print("Percentage of males who survived:",
train.loc[train["Sex"] == 'male', "Survived"].mean() * 100)
```

b. Visualize Pclass feature

```
#draw a bar plot of survival by Pclass
sns.barplot(x="Pclass", y="Survived", data=train)

#print percentage of people by Pclass that survived
print("Percentage of Pclass = 1 who survived:",
train.loc[train["Pclass"] == 1, "Survived"].mean()*100)

print("Percentage of Pclass = 2 who survived:",
train.loc[train["Pclass"] == 2, "Survived"].mean()*100)

print("Percentage of Pclass = 3 who survived:",
train.loc[train["Pclass"] == 3, "Survived"].mean()*100)
```

c. Visualize Sibsp feature

```
#draw a bar plot for SibSp vs. survival
sns.barplot(x="SibSp", y="Survived", data=train)
print("Percentage of SibSp = 0 who survived:",
train.loc[train["SibSp"] == 0,"Survived"].mean()*100)

print("Percentage of SibSp = 1 who survived:",
train.loc[train["SibSp"] == 1,"Survived"].mean()*100)

print("Percentage of SibSp = 2 who survived:",
train.loc[train["SibSp"] == 2,"Survived"].mean()*100)
```

5. Cleaning Data
   Look at the test and train data

```
print(train.describe(include="all"))

print(test.describe(include="all"))
```

- We have a total of 418 passengers.
- 1 value from the Fare feature is missing.
- Around 20.5% of the Age feature is missing, we will need to fill that in.

a. Cabin, Ticket, Name and Fare features
   We can drop the Cabin, Ticket, Name and Fare features since it's unlikely that these features yield any useful information

```
train = train.drop(['Cabin'], axis = 1)
test = test.drop(['Cabin'], axis = 1)

train = train.drop(['Ticket'], axis = 1)
test = test.drop(['Ticket'], axis = 1)

train = train.drop(['Name'], axis = 1)
test = test.drop(['Name'], axis = 1)


train = train.drop(['Fare'], axis = 1)
test = test.drop(['Fare'], axis = 1)
```

b. Embarked Feature
   Now we need to fill in the missing values in the Embarked feature.

```
print("Number of people embarking in Southampton (S):")
southampton = train[train["Embarked"] == "S"].shape[0]
```

```
print(southampton)

print("Number of people embarking in Cherbourg (C):")
cherbourg = train[train["Embarked"] == "C"].shape[0]
print(cherbourg)

print("Number of people embarking in Queenstown (Q):")
queenstown = train[train["Embarked"] == "Q"].shape[0]
print(queenstown)
```

It's clear that the majority of people embarked in Southampton (S). Let's go ahead and fill in the missing values with S.

```
#replacing the missing values in the Embarked feature
with S
train = train.fillna({"Embarked": "S"})
test = test.fillna({"Embarked": "S"})


#map each Embarked value to a numerical value
embarked_mapping = {"S": 1, "C": 2, "Q": 3}
train['Embarked'] =
train['Embarked'].map(embarked_mapping)
test['Embarked'] =
test['Embarked'].map(embarked_mapping)
```

c. Age feature
   Fill in the missing values of age with mean (or median) age

```
train.Age = train.Age.fillna(train.Age.mean())
test.Age = test.Age.fillna(test.Age.mean())
```

d. Gender feature

Map each gender value to a numerical value.

```
mapping = {"male": 0, "female": 1}
train['Sex'] = train['Sex'].map(mapping)
test['Sex'] = test['Sex'].map(mapping)
```

6. Train Model for Predicting survival rate

   a. Train-test split

```
from sklearn.model_selection import train_test_split

predictors = train.drop(['Survived', 'PassengerId'], axis=1)
target = train["Survived"]
x_train, x_val, y_train, y_val = train_test_split(predictors,
target, test_size = 0.20, random_state = 0)
```

   b. Model using different classification models

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.svm import LinearSVC

# Initialize models
log_reg = LogisticRegression()
decision_tree = DecisionTreeClassifier()
knn = KNeighborsClassifier(n_neighbors=5)
linear_svc = LinearSVC()    # Linear SVC
svm = SVC(probability=True)

# Train models
log_reg.fit(X_train, y_train)
decision_tree.fit(X_train, y_train)
knn.fit(X_train, y_train)
svm.fit(X_train, y_train)
linear_svc.fit(X_train, y_train)
```

c. Plot ROC curve and confusion matrices for each model. Also print the accuracy and AUC for each.

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score

models = [("Logistic Regression", log_reg),
        ("Decision Tree", decision_tree),
        ("k-NN", knn),
        ("SVM", svm)]

plt.figure(figsize=(10, 8))

for name, model in models:
   y_prob = model.predict_proba(x_val)[:,1]

   fpr, tpr, thresholds = roc_curve(y_val, y_prob)
   roc_auc = auc(fpr, tpr)

   plt.plot(fpr, tpr, label=f'ROC curve (area = {roc_auc:.2f})')
   plt.plot([0, 1], [0, 1], 'k--')  # Dashed diagonal line
   plt.xlabel('False Positive Rate')
   plt.ylabel('True Positive Rate')
   plt.title('Receiver Operating Characteristic (ROC) Curve')
   plt.legend([name for name, _ in models],loc="lower right")
   plt.show()

   y_pred = model.predict(x_val)
   accuracy = accuracy_score(y_val, y_pred)
   print(f"{name} - Accuracy: {accuracy:.2f}, AUC: roc_auc:.2f}")
```