#data analysis libraries

import numpy as np

import pandas as pd

#visualization libraries

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

#ignore warnings

import warnings

warnings.filterwarnings('ignore')

import train and test CSV files

train = pd.read\_csv("../input/train.csv")

test = pd.read\_csv("../input/test.csv")

#

#take a look at the training data

train.describe(include="all")

#get a list of the features within the dataset

print(train.columns)

#see a sample of the dataset to get an idea of the variables

train.sample(5)

#see a summary of the training dataset

train.describe(include = "all")

#check for any other unusable values

print(pd.isnull(train).sum())

*#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["Survived"][train["Sex"] **==** 'female'].value\_counts(normalize **=** **True**)[1]**\***100)

​

print("Percentage of males who survived:", train["Survived"][train["Sex"] **==** 'male'].value\_counts(normalize **=** **True**)[1]**\***100)

#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["Survived"][train["Pclass"] == 1].value\_counts(normalize = True)[1]\*100)

print("Percentage of Pclass = 2 who survived:", train["Survived"][train["Pclass"] == 2].value\_counts(normalize = True)[1]\*100)

print("Percentage of Pclass = 3 who survived:", train["Survived"][train["Pclass"] == 3].value\_counts(normalize = True)[1]\*100)

#draw a bar plot for SibSp vs. survival

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

#I won't be printing individual percent values for all of these.

print("Percentage of SibSp = 0 who survived:", train["Survived"][train["SibSp"] == 0].value\_counts(normalize = True)[1]\*100)

print("Percentage of SibSp = 1 who survived:", train["Survived"][train["SibSp"] == 1].value\_counts(normalize = True)[1]\*100)

print("Percentage of SibSp = 2 who survived:", train["Survived"][train["SibSp"] == 2].value\_counts(normalize = True)[1]\*100)

#draw a bar plot for Parch vs. survival

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

plt.show()

#sort the ages into logical categories

train["Age"] = train["Age"].fillna(-0.5)

test["Age"] = test["Age"].fillna(-0.5)

bins = [-1, 0, 5, 12, 18, 24, 35, 60, np.inf]

labels = ['Unknown', 'Baby', 'Child', 'Teenager', 'Student', 'Young Adult', 'Adult', 'Senior']

train['AgeGroup'] = pd.cut(train["Age"], bins, labels = labels)

test['AgeGroup'] = pd.cut(test["Age"], bins, labels = labels)

#draw a bar plot of Age vs. survival

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

plt.show()

#sort the ages into logical categories

train["Age"] = train["Age"].fillna(-0.5)

test["Age"] = test["Age"].fillna(-0.5)

bins = [-1, 0, 5, 12, 18, 24, 35, 60, np.inf]

labels = ['Unknown', 'Baby', 'Child', 'Teenager', 'Student', 'Young Adult', 'Adult', 'Senior']

train['AgeGroup'] = pd.cut(train["Age"], bins, labels = labels)

test['AgeGroup'] = pd.cut(test["Age"], bins, labels = labels)

#draw a bar plot of Age vs. survival

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

plt.show()

train["CabinBool"] = (train["Cabin"].notnull().astype('int'))

test["CabinBool"] = (test["Cabin"].notnull().astype('int'))

#calculate percentages of CabinBool vs. survived

print("Percentage of CabinBool = 1 who survived:", train["Survived"][train["CabinBool"] == 1].value\_counts(normalize = True)[1]\*100)

print("Percentage of CabinBool = 0 who survived:", train["Survived"][train["CabinBool"] == 0].value\_counts(normalize = True)[1]\*100)

#draw a bar plot of CabinBool vs. survival

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

plt.show()

#we'll start off by dropping the Cabin feature since not a lot more useful information can be extracted from it.

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

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

#we can also drop the Ticket feature since it's unlikely to yield any useful information

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

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

#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)

#replacing the missing values in the Embarked feature with S

train = train.fillna({"Embarked": "S"})

#create a combined group of both datasets

combine = [train, test]

#extract a title for each Name in the train and test datasets

for dataset in combine:

dataset['Title'] = dataset.Name.str.extract(' ([A-Za-z]+)\.', expand=False)

pd.crosstab(train['Title'], train['Sex'])

#replace various titles with more common names

for dataset in combine:

dataset['Title'] = dataset['Title'].replace(['Lady', 'Capt', 'Col',

'Don', 'Dr', 'Major', 'Rev', 'Jonkheer', 'Dona'], 'Rare')

dataset['Title'] = dataset['Title'].replace(['Countess', 'Lady', 'Sir'], 'Royal')

dataset['Title'] = dataset['Title'].replace('Mlle', 'Miss')

dataset['Title'] = dataset['Title'].replace('Ms', 'Miss')

dataset['Title'] = dataset['Title'].replace('Mme', 'Mrs')

train[['Title', 'Survived']].groupby(['Title'], as\_index=False).mean()

#map each of the title groups to a numerical value

title\_mapping = {"Mr": 1, "Miss": 2, "Mrs": 3, "Master": 4, "Royal": 5, "Rare": 6}

for dataset in combine:

dataset['Title'] = dataset['Title'].map(title\_mapping)

dataset['Title'] = dataset['Title'].fillna(0)

train.head()

# fill missing age with mode age group for each title

mr\_age = train[train["Title"] == 1]["AgeGroup"].mode() #Young Adult

miss\_age = train[train["Title"] == 2]["AgeGroup"].mode() #Student

mrs\_age = train[train["Title"] == 3]["AgeGroup"].mode() #Adult

master\_age = train[train["Title"] == 4]["AgeGroup"].mode() #Baby

royal\_age = train[train["Title"] == 5]["AgeGroup"].mode() #Adult

rare\_age = train[train["Title"] == 6]["AgeGroup"].mode() #Adult

age\_title\_mapping = {1: "Young Adult", 2: "Student", 3: "Adult", 4: "Baby", 5: "Adult", 6: "Adult"}

#I tried to get this code to work with using .map(), but couldn't.

#I've put down a less elegant, temporary solution for now.

#train = train.fillna({"Age": train["Title"].map(age\_title\_mapping)})

#test = test.fillna({"Age": test["Title"].map(age\_title\_mapping)})

for x in range(len(train["AgeGroup"])):

if train["AgeGroup"][x] == "Unknown":

train["AgeGroup"][x] = age\_title\_mapping[train["Title"][x]]

for x in range(len(test["AgeGroup"])):

if test["AgeGroup"][x] == "Unknown":

test["AgeGroup"][x] = age\_title\_mapping[test["Title"][x]]

#map each Age value to a numerical value

age\_mapping = {'Baby': 1, 'Child': 2, 'Teenager': 3, 'Student': 4, 'Young Adult': 5, 'Adult': 6, 'Senior': 7}

train['AgeGroup'] = train['AgeGroup'].map(age\_mapping)

test['AgeGroup'] = test['AgeGroup'].map(age\_mapping)

train.head()

#dropping the Age feature for now, might change

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

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

#map each Sex value to a numerical value

sex\_mapping = {"male": 0, "female": 1}

train['Sex'] = train['Sex'].map(sex\_mapping)

test['Sex'] = test['Sex'].map(sex\_mapping)

train.head()

#map each Sex value to a numerical value

sex\_mapping = {"male": 0, "female": 1}

train['Sex'] = train['Sex'].map(sex\_mapping)

test['Sex'] = test['Sex'].map(sex\_mapping)

train.head()

#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)

train.head()

#fill in missing Fare value in test set based on mean fare for that Pclass

for x in range(len(test["Fare"])):

if pd.isnull(test["Fare"][x]):

pclass = test["Pclass"][x] #Pclass = 3

test["Fare"][x] = round(train[train["Pclass"] == pclass]["Fare"].mean(), 4)

#map Fare values into groups of numerical values

train['FareBand'] = pd.qcut(train['Fare'], 4, labels = [1, 2, 3, 4])

test['FareBand'] = pd.qcut(test['Fare'], 4, labels = [1, 2, 3, 4])

#drop Fare values

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

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

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 = 0.22, random\_state = 0)

# Gaussian Naive Bayes

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score

gaussian = GaussianNB()

gaussian.fit(x\_train, y\_train)

y\_pred = gaussian.predict(x\_val)

acc\_gaussian = round(accuracy\_score(y\_pred, y\_val) \* 100, 2)

print(acc\_gaussian)

# Logistic Regression

from sklearn.linear\_model import LogisticRegression

logreg = LogisticRegression()

logreg.fit(x\_train, y\_train)

y\_pred = logreg.predict(x\_val)

acc\_logreg = round(accuracy\_score(y\_pred, y\_val) \* 100, 2)

print(acc\_logreg)

# Support Vector Machines

from sklearn.svm import SVC

svc = SVC()

svc.fit(x\_train, y\_train)

y\_pred = svc.predict(x\_val)

acc\_svc = round(accuracy\_score(y\_pred, y\_val) \* 100, 2)

print(acc\_svc)

# Linear SVC

from sklearn.svm import LinearSVC

linear\_svc = LinearSVC()

linear\_svc.fit(x\_train, y\_train)

y\_pred = linear\_svc.predict(x\_val)

acc\_linear\_svc = round(accuracy\_score(y\_pred, y\_val) \* 100, 2)

print(acc\_linear\_svc)

# Perceptron

from sklearn.linear\_model import Perceptron

perceptron = Perceptron()

perceptron.fit(x\_train, y\_train)

y\_pred = perceptron.predict(x\_val)

acc\_perceptron = round(accuracy\_score(y\_pred, y\_val) \* 100, 2)

print(acc\_perceptron)

#Decision Tree

from sklearn.tree import DecisionTreeClassifier

decisiontree = DecisionTreeClassifier()

decisiontree.fit(x\_train, y\_train)

y\_pred = decisiontree.predict(x\_val)

acc\_decisiontree = round(accuracy\_score(y\_pred, y\_val) \* 100, 2)

print(acc\_decisiontree)

# Random Forest

from sklearn.ensemble import RandomForestClassifier

randomforest = RandomForestClassifier()

randomforest.fit(x\_train, y\_train)

y\_pred = randomforest.predict(x\_val)

acc\_randomforest = round(accuracy\_score(y\_pred, y\_val) \* 100, 2)

print(acc\_randomforest)

# KNN or k-Nearest Neighbors

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()

knn.fit(x\_train, y\_train)

y\_pred = knn.predict(x\_val)

acc\_knn = round(accuracy\_score(y\_pred, y\_val) \* 100, 2)

print(acc\_knn)

# Stochastic Gradient Descent

from sklearn.linear\_model import SGDClassifier

sgd = SGDClassifier()

sgd.fit(x\_train, y\_train)

y\_pred = sgd.predict(x\_val)

acc\_sgd = round(accuracy\_score(y\_pred, y\_val) \* 100, 2)

print(acc\_sgd)

# Gradient Boosting Classifier

from sklearn.ensemble import GradientBoostingClassifier

gbk = GradientBoostingClassifier()

gbk.fit(x\_train, y\_train)

y\_pred = gbk.predict(x\_val)

acc\_gbk = round(accuracy\_score(y\_pred, y\_val) \* 100, 2)

print(acc\_gbk)

models = pd.DataFrame({

'Model': ['Support Vector Machines', 'KNN', 'Logistic Regression',

'Random Forest', 'Naive Bayes', 'Perceptron', 'Linear SVC',

'Decision Tree', 'Stochastic Gradient Descent', 'Gradient Boosting Classifier'],

'Score': [acc\_svc, acc\_knn, acc\_logreg,

acc\_randomforest, acc\_gaussian, acc\_perceptron,acc\_linear\_svc, acc\_decisiontree,

acc\_sgd, acc\_gbk]})

models.sort\_values(by='Score', ascending=False)

#set ids as PassengerId and predict survival

ids = test['PassengerId']

predictions = gbk.predict(test.drop('PassengerId', axis=1))

#set the output as a dataframe and convert to csv file named submission.csv

output = pd.DataFrame({ 'PassengerId' : ids, 'Survived': predictions })

output.to\_csv('submission.csv', index=False)