#Titanic Tutorial Part 3

#Graphics Analysis

#Feature Reduction (Extraction/Selection)

#Filling in Missing Values

#Split\_Train\_Test

#Model Selection and Evaluation

#For Part 3 of the Titanic Tutorial, complete Steps 14-15.

import pandas as pd

import yellowbrick

#Step 1: Load data into a dataframe

addr1 = "D://From SD Card/DSC550/data/train.csv"

data = pd.read\_csv(addr1)

# Step 2: check the dimension of the table

print("The dimension of the table is: ", data.shape)

#Step 3: Look at the data

print(data.head(5))

#Step 5: what type of variables are in the table

print("Describe Data")

print(data.describe())

print("Summarized Data")

print(data.describe(include=['O']))

#Step 6: import visulization packages

import matplotlib.pyplot as plt

# set up the figure size

plt.rcParams['figure.figsize'] = (20, 10)

# make subplots

fig, axes = plt.subplots(nrows = 2, ncols = 2)

# Specify the features of interest

num\_features = ['Age', 'SibSp', 'Parch', 'Fare']

xaxes = num\_features

yaxes = ['Counts', 'Counts', 'Counts', 'Counts']

# draw histograms

axes = axes.ravel()

for idx, ax in enumerate(axes):

ax.hist(data[num\_features[idx]].dropna(), bins=40)

ax.set\_xlabel(xaxes[idx], fontsize=20)

ax.set\_ylabel(yaxes[idx], fontsize=20)

ax.tick\_params(axis='both', labelsize=15)

#plt.show()

#7: Barcharts: set up the figure size

#%matplotlib inline

plt.rcParams['figure.figsize'] = (20, 10)

# make subplots

fig, axes = plt.subplots(nrows = 2, ncols = 2)

# make the data read to feed into the visulizer

X\_Survived = data.replace({'Survived': {1: 'yes', 0: 'no'}}).groupby('Survived').size().reset\_index(name='Counts')['Survived']

Y\_Survived = data.replace({'Survived': {1: 'yes', 0: 'no'}}).groupby('Survived').size().reset\_index(name='Counts')['Counts']

# make the bar plot

axes[0, 0].bar(X\_Survived, Y\_Survived)

axes[0, 0].set\_title('Survived', fontsize=25)

axes[0, 0].set\_ylabel('Counts', fontsize=20)

axes[0, 0].tick\_params(axis='both', labelsize=15)

# make the data read to feed into the visulizer

X\_Pclass = data.replace({'Pclass': {1: '1st', 2: '2nd', 3: '3rd'}}).groupby('Pclass').size().reset\_index(name='Counts')['Pclass']

Y\_Pclass = data.replace({'Pclass': {1: '1st', 2: '2nd', 3: '3rd'}}).groupby('Pclass').size().reset\_index(name='Counts')['Counts']

# make the bar plot

axes[0, 1].bar(X\_Pclass, Y\_Pclass)

axes[0, 1].set\_title('Pclass', fontsize=25)

axes[0, 1].set\_ylabel('Counts', fontsize=20)

axes[0, 1].tick\_params(axis='both', labelsize=15)

# make the data read to feed into the visulizer

X\_Sex = data.groupby('Sex').size().reset\_index(name='Counts')['Sex']

Y\_Sex = data.groupby('Sex').size().reset\_index(name='Counts')['Counts']

# make the bar plot

axes[1, 0].bar(X\_Sex, Y\_Sex)

axes[1, 0].set\_title('Sex', fontsize=25)

axes[1, 0].set\_ylabel('Counts', fontsize=20)

axes[1, 0].tick\_params(axis='both', labelsize=15)

# make the data read to feed into the visulizer

X\_Embarked = data.groupby('Embarked').size().reset\_index(name='Counts')['Embarked']

Y\_Embarked = data.groupby('Embarked').size().reset\_index(name='Counts')['Counts']

# make the bar plot

axes[1, 1].bar(X\_Embarked, Y\_Embarked)

axes[1, 1].set\_title('Embarked', fontsize=25)

axes[1, 1].set\_ylabel('Counts', fontsize=20)

axes[1, 1].tick\_params(axis='both', labelsize=15)

#plt.show()

#Step 8: Pearson Ranking

#set up the figure size

#%matplotlib inline

plt.rcParams['figure.figsize'] = (15, 7)

# import the package for visulization of the correlation

from yellowbrick.features import Rank2D

# extract the numpy arrays from the data frame

X = data[num\_features].as\_matrix()

# instantiate the visualizer with the Covariance ranking algorithm

visualizer = Rank2D(features=num\_features, algorithm='pearson')

visualizer.fit(X) # Fit the data to the visualizer

visualizer.transform(X) # Transform the data

visualizer.poof(outpath="d://pcoords1.png") # Draw/show/poof the data

#plt.show()

# Step 9: Compare variables against Survived and Not Survived

#set up the figure size

#%matplotlib inline

plt.rcParams['figure.figsize'] = (15, 7)

plt.rcParams['font.size'] = 50

# setup the color for yellowbrick visulizer

from yellowbrick.style import set\_palette

set\_palette('sns\_bright')

# import packages

from yellowbrick.features import ParallelCoordinates

# Specify the features of interest and the classes of the target

classes = ['Not-survived', 'Survived']

num\_features = ['Age', 'SibSp', 'Parch', 'Fare']

# copy data to a new dataframe

data\_norm = data.copy()

# normalize data to 0-1 range

for feature in num\_features:

data\_norm[feature] = (data[feature] - data[feature].mean(skipna=True)) / (data[feature].max(skipna=True) - data[feature].min(skipna=True))

# Extract the numpy arrays from the data frame

X = data\_norm[num\_features].as\_matrix()

y = data.Survived.as\_matrix()

# Instantiate the visualizer

# Instantiate the visualizer

visualizer = ParallelCoordinates(classes=classes, features=num\_features)

visualizer.fit(X, y) # Fit the data to the visualizer

visualizer.transform(X) # Transform the data

#visualizer.poof(outpath="d://pcoords2.png") # Draw/show/poof the data

plt.show();

# Step 10 - stacked bar charts to compare survived/not survived

#set up the figure size

#%matplotlib inline

plt.rcParams['figure.figsize'] = (20, 10)

# make subplots

fig, axes = plt.subplots(nrows = 2, ncols = 2)

# make the data read to feed into the visulizer

Sex\_survived = data.replace({'Survived': {1: 'Survived', 0: 'Not-survived'}})[data['Survived']==1]['Sex'].value\_counts()

Sex\_not\_survived = data.replace({'Survived': {1: 'Survived', 0: 'Not-survived'}})[data['Survived']==0]['Sex'].value\_counts()

Sex\_not\_survived = Sex\_not\_survived.reindex(index = Sex\_survived.index)

# make the bar plot

p1 = axes[0, 0].bar(Sex\_survived.index, Sex\_survived.values)

p2 = axes[0, 0].bar(Sex\_not\_survived.index, Sex\_not\_survived.values, bottom=Sex\_survived.values)

axes[0, 0].set\_title('Sex', fontsize=25)

axes[0, 0].set\_ylabel('Counts', fontsize=20)

axes[0, 0].tick\_params(axis='both', labelsize=15)

axes[0, 0].legend((p1[0], p2[0]), ('Survived', 'Not-survived'), fontsize = 15)

# make the data read to feed into the visulizer

Pclass\_survived = data.replace({'Survived': {1: 'Survived', 0: 'Not-survived'}}).replace({'Pclass': {1: '1st', 2: '2nd', 3: '3rd'}})[data['Survived']==1]['Pclass'].value\_counts()

Pclass\_not\_survived = data.replace({'Survived': {1: 'Survived', 0: 'Not-survived'}}).replace({'Pclass': {1: '1st', 2: '2nd', 3: '3rd'}})[data['Survived']==0]['Pclass'].value\_counts()

Pclass\_not\_survived = Pclass\_not\_survived.reindex(index = Pclass\_survived.index)

# make the bar plot

p3 = axes[0, 1].bar(Pclass\_survived.index, Pclass\_survived.values)

p4 = axes[0, 1].bar(Pclass\_not\_survived.index, Pclass\_not\_survived.values, bottom=Pclass\_survived.values)

axes[0, 1].set\_title('Pclass', fontsize=25)

axes[0, 1].set\_ylabel('Counts', fontsize=20)

axes[0, 1].tick\_params(axis='both', labelsize=15)

axes[0, 1].legend((p3[0], p4[0]), ('Survived', 'Not-survived'), fontsize = 15)

# make the data read to feed into the visulizer

Embarked\_survived = data.replace({'Survived': {1: 'Survived', 0: 'Not-survived'}})[data['Survived']==1]['Embarked'].value\_counts()

Embarked\_not\_survived = data.replace({'Survived': {1: 'Survived', 0: 'Not-survived'}})[data['Survived']==0]['Embarked'].value\_counts()

Embarked\_not\_survived = Embarked\_not\_survived.reindex(index = Embarked\_survived.index)

# make the bar plot

p5 = axes[1, 0].bar(Embarked\_survived.index, Embarked\_survived.values)

p6 = axes[1, 0].bar(Embarked\_not\_survived.index, Embarked\_not\_survived.values, bottom=Embarked\_survived.values)

axes[1, 0].set\_title('Embarked', fontsize=25)

axes[1, 0].set\_ylabel('Counts', fontsize=20)

axes[1, 0].tick\_params(axis='both', labelsize=15)

axes[1, 0].legend((p5[0], p6[0]), ('Survived', 'Not-survived'), fontsize = 15)

#plt.show()

# Step 11 - fill in missing values and eliminate features

#fill the missing age data with median value

def fill\_na\_median(data, inplace=True):

return data.fillna(data.median(), inplace=inplace)

fill\_na\_median(data['Age'])

# check the result

print(data['Age'].describe())

# fill with the most represented value

def fill\_na\_most(data, inplace=True):

return data.fillna('S', inplace=inplace)

fill\_na\_most(data['Embarked'])

# check the result

print(data['Embarked'].describe())

# import package

import numpy as np

# log-transformation

def log\_transformation(data):

return data.apply(np.log1p)

data['Fare\_log1p'] = log\_transformation(data['Fare'])

# check the data

#print(data.describe())

#Step 12 - adjust skewed data (fare)

#check the distribution using histogram

# set up the figure size

#%matplotlib inline

plt.rcParams['figure.figsize'] = (10, 5)

plt.hist(data['Fare\_log1p'], bins=40)

plt.xlabel('Fare\_log1p', fontsize=20)

plt.ylabel('Counts', fontsize=20)

plt.tick\_params(axis='both', labelsize=15)

#plt.show()

#Step 13 - convert categorical data to numbers

#get the categorical data

cat\_features = ['Pclass', 'Sex', "Embarked"]

data\_cat = data[cat\_features]

data\_cat = data\_cat.replace({'Pclass': {1: '1st', 2: '2nd', 3: '3rd'}})

# One Hot Encoding

data\_cat\_dummies = pd.get\_dummies(data\_cat)

# check the data

print(data\_cat\_dummies.head(8))

#Step 14 - create a whole features dataset that can be used for train and validation data splitting

# here we will combine the numerical features and the dummie features together

features\_model = ['Age', 'SibSp', 'Parch', 'Fare\_log1p']

data\_model\_X = pd.concat([data[features\_model], data\_cat\_dummies], axis=1)

# create a whole target dataset that can be used for train and validation data splitting

data\_model\_y = data.replace({'Survived': {1: 'Survived', 0: 'Not\_survived'}})['Survived']

# separate data into training and validation and check the details of the datasets

# import packages

from sklearn.model\_selection import train\_test\_split

# split the data

X\_train, X\_val, y\_train, y\_val = train\_test\_split(data\_model\_X, data\_model\_y, test\_size =0.3, random\_state=11)

# number of samples in each set

print("No. of samples in training set: ", X\_train.shape[0])

print("No. of samples in validation set:", X\_val.shape[0])

# Survived and not-survived

print('\n')

print('No. of survived and not-survived in the training set:')

print(y\_train.value\_counts())

print('\n')

print('No. of survived and not-survived in the validation set:')

print(y\_val.value\_counts())

# Step 15 - Eval Metrics

from sklearn.linear\_model import LogisticRegression

from yellowbrick.classifier import ConfusionMatrix

from yellowbrick.classifier import ClassificationReport

from yellowbrick.classifier import ROCAUC

# Instantiate the classification model

model = LogisticRegression()

#The ConfusionMatrix visualizer taxes a model

classes = ['Not\_survived','Survived']

cm = ConfusionMatrix(model, classes=classes, percent=False)

#Fit fits the passed model. This is unnecessary if you pass the visualizer a pre-fitted model

cm.fit(X\_train, y\_train)

#To create the ConfusionMatrix, we need some test data. Score runs predict() on the data

#and then creates the confusion\_matrix from scikit learn.

cm.score(X\_val, y\_val)

# change fontsize of the labels in the figure

for label in cm.ax.texts:

label.set\_size(20)

#How did we do?

cm.poof()

# Precision, Recall, and F1 Score

# set the size of the figure and the font size

#%matplotlib inline

plt.rcParams['figure.figsize'] = (15, 7)

plt.rcParams['font.size'] = 20

# Instantiate the visualizer

visualizer = ClassificationReport(model, classes=classes)

visualizer.fit(X\_train, y\_train) # Fit the training data to the visualizer

visualizer.score(X\_val, y\_val) # Evaluate the model on the test data

g = visualizer.poof()

# ROC and AUC

#Instantiate the visualizer

visualizer = ROCAUC(model)

visualizer.fit(X\_train, y\_train) # Fit the training data to the visualizer

visualizer.score(X\_val, y\_val) # Evaluate the model on the test data

g = visualizer.poof()