DSC 550 Week 11 & 12

Final Case Study Analysis

Lara Clasen

I decided to work with a data set called “Cars” which provides information about different vehicles and several of their characteristics. I chose this data set because it had a similar layout to the Titanic data set which we used to learn about this project initially. Ultimately, for both sets of data, the goal was to investigate a relationship between two variables. In the Titanic data set variables related to survival are looked into, and in my data set I look at variables as the are related to the origin of the vehicles.

To begin I loaded the data from my file “cars.csv” into a Data Frame. I used PyCharm for this as I have not been able to get yellowbrick installed in my Anaconda in order to use in Jupyter, unfortunately.

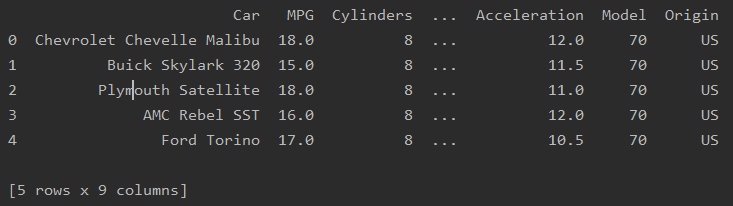
By displaying the dimensions of the data set we can get an idea of the size of our set.

#Case Study Part 3  
  
import pandas as pd  
import yellowbrick  
  
  
#Step 1: Load data into a dataframe  
addr1 = "cars.csv"  
data = pd.read\_csv(addr1)  
  
# Step 2: check the dimension of the table  
print("The dimension of the table is: ", data.shape)



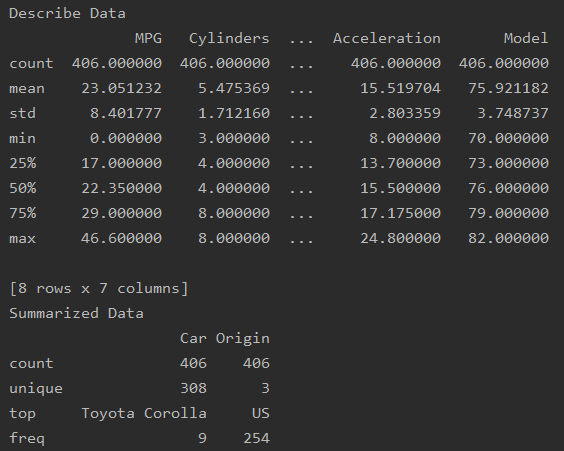
I can then take a look at the first 5 rows of my data to get a better idea of its contents.

#Step 3: Look at the data  
print(data.head(5))



The next important step is to review the variables in the data set.

#Step 5: what type of variables are in the table  
print("Describe Data")  
print(data.describe())  
print("Summarized Data")  
print(data.describe(include=['O']))



Now we can begin to look at some preliminary visualizations of the data to hopefully gain some initial insight and find direction as to the questions we would like to ask. I import Matplotlob for better visualization options.

#Step 6: import visualization packages  
import matplotlib.pyplot as plt

I specify which features to look at with these visualizations, and draw histograms.

# Specify the features of interest  
num\_features = ['MPG', 'Weight', 'Acceleration', 'Origin']  
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()

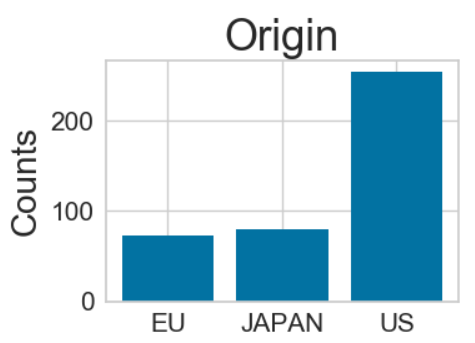
A screenshot of a cell phone

Description automatically generated

Figure : Histograms

From here I will look at a bar chart of the Origin of the vehicles, and Pearson correlation for four variables.

# make the data read to feed into the visulizer  
X\_Origin = data.replace({'Origin': {1: 'JAPAN', 0: 'US', 3: 'EU'}}).groupby('Origin').size().reset\_index(name='Counts')['Origin']  
Y\_Origin = data.replace({'Origin': {1: 'JAPAN', 0: 'US', 3: 'EU'}}).groupby('Origin').size().reset\_index(name='Counts')['Counts']  
# make the bar plot  
axes[0, 0].bar(X\_Origin, Y\_Origin)  
axes[0, 0].set\_title('Origin', fontsize=25)  
axes[0, 0].set\_ylabel('Counts', fontsize=20)  
axes[0, 0].tick\_params(axis='both', labelsize=15)  
plt.show()



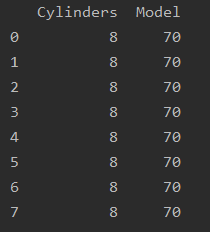
#Step 8: Pearson Ranking  
plt.rcParams['figure.figsize'] = (15, 7)  
  
# import the package for visualization of the correlation  
from yellowbrick.features import Rank2D  
  
# extract the numpy arrays from the data frame  
X = data[num\_features].as\_matrix()  
  
# instantiate the visualizer with 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()

A screenshot of a cell phone

Description automatically generated

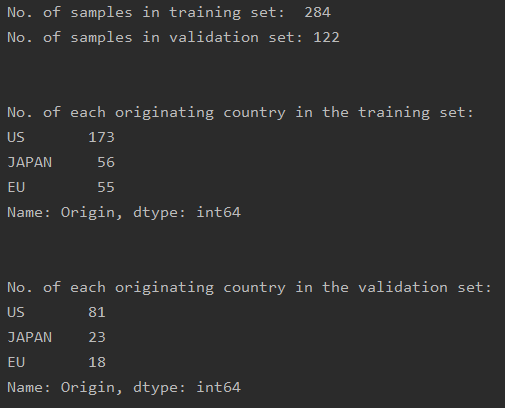
I change a few of my categorical variables into numerical ones.

#Step 13 - convert categorical data to numbers  
#get the categorical data  
cat\_features = ['Cylinders', 'Model']  
data\_cat = data[cat\_features]  
# One Hot Encoding  
data\_cat\_dummies = pd.get\_dummies(data\_cat)  
# check the data  
print(data\_cat\_dummies.head(8))



Step 14 directs us to split up our data into two sets: Training and Testing.

#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 = ['MPG', 'Cylinders', 'Weight', 'Origin']  
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({'Origin': {0: 'US', 1: 'JAPAN', 2: 'EU'}})['Origin']  
# 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])  
  
# Origins  
print('\n')  
print('No. of each originating country in the training set:')  
print(y\_train.value\_counts())  
  
print('\n')  
print('No. of each originating country in the validation set:')  
print(y\_val.value\_counts())



For Step 15 I use the following metrics for evaluation purposes:

Confusion Matrix, Precision Recall & F1 score, and ROC curve.

# 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 = ['EU','JAPAN','US']  
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()

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generatedA close up of a map

Description automatically generated