|  |  |
| --- | --- |
| ***EDA*** | |
| ***## All imports***  ***(Data Libraries)*** | ***import numpy as np***  ***import pandas as pd***  ***import matplotlib.pyplot as plt***  ***import seaborn as sns*** |
| ***## import the warning*** | ***import warnings***  ***warnings.filterwarnings ("ignore")*** |
| ***## Upload to Colab*** | ***from google.colab import files***  ***uploaded = files.upload()*** |
| ***## Load the dataset*** | ***df = pd.read\_csv('superstore.csv')*** |
|  |  |
| ***Primary Set of Operations on a DataFrame*** | |
| ***## Check the Shape*** | ***df.shape*** |
| ***## Total Number of Missing Values*** | ***df.isnull().sum()*** |
| ***## print the information of variables to check their data types*** | ***df.info()*** |
| ***## Check the summary for the numeric columns*** | ***df.describe()*** |
|  |  |
|  |  |
|  |  |
|  |  |
| ***## Check the number of nulls in the specific Column***  ***## To check the count with null values***  ***## Check the table with missing values*** | ***df.Rating.isnull().sum()***  ***df['Android Ver'].isnull()***  ***df [df['Android Ver'].isnull()]*** |
| ***## Finding the mode value:*** | ***df['Android Ver'].mode()[0]*** |
| ***## Replace the NAN value with mode of the clm:*** | ***df['Android Ver'] = df['Android Ver'].fillna(df['Android Ver'].mode()[0])***  ***df['Android Ver']*** |
| ***## Individual counts for each entry in the clm*** | ***df['Current Ver'].value\_counts()***  ***df['Android Ver'].value\_counts()*** |
| ***## Function to replace $***  ***## Changing the data type of numerical values after removing $*** | ***lambda x:0 if x=="0" else float (x[1:])***  ***df.Price = df.Price.apply(lambda x:0 if x=="0" else float (x[1:]) )***  ***df.Price*** |
| ***## Check number of Duplicate Rows:*** | ***df.duplicated().sum()*** |
| ***## Drop all the Duplicate Rows:*** | ***df.drop\_duplicates(subset=None, keep='first', inplace=True, ignore\_index=False)*** |
|  |  |
|  |  |

DATA CLEANING

1. ***Identifying the data types***

|  |  |
| --- | --- |
| ***## Check the Shape*** | ***df.shape*** |
| ***## print the information of variables to check their data types*** | ***df.info()*** |
| ***## Total Number of Missing Values*** | ***df.isnull().sum()*** |
| ***## Check the number of nulls in the specific Column***  ***## To check the count with null values***  ***## Check the set of table with missing values in a given Column*** | ***df.Rating.isnull().sum()***  ***df['Android Ver'].isnull()***  ***df [df['Android Ver'].isnull()]*** |
|  |  |

1. ***Fixing the rows and columns***

|  |  |
| --- | --- |
| ***## Read the file in inp0 without first two rows as it is of no use*** | ***inp0= pd.read\_csv("bank.csv", skiprows=2)***  ***inp0*** |
| ***## Deleting a Row:*** | ***Based on index number***  ***df.drop([0,1,2,3,4])***  ***0,1,2,3,4 are the index of rows to be dropped.*** |
| ***Deleting all the rows with missing value in one column***  ***#drop the records with age missing in inp0 and copy in inp1 dataframe.***  ***inp1= inp0[-inp0.age.isnull()].copy()***  ***inp1*** |
|  |
| ***## Deleting a Column based on column name:*** | ***df.drop(columns='customerid', inplace=True)***  ***df.head()*** |
| ***## Splitting a column to form 2 Columns: job,edu to job and edu*** | ***#Extract job in newly created 'job' column from "jobedu" column.***  ***inp0['job']= inp0.jobedu.apply (lambda x: x.split(",")[0])***  ***inp0.head()*** |
| ***#Extract education in newly created 'education' column from "jobedu" column.***  ***inp0['education']= inp0.jobedu.apply (lambda x: x.split(",")[1])***  ***inp0.head()*** |
| ***## Percentage of Missing Values*** | ***#calculate the percentage of missing values in response column.***  ***float(100.0\*30/45211)*** |
|  |  |
|  |  |

1. ***Imputing/removing missing values***

|  |  |
| --- | --- |
|  |  |
| ***## Remove the rows with null in age*** | ***bank1= bank[~(bank.age.isnull())].copy()***  ***bank1.shape*** |
| ***Replace the null values with Mode (2 Steps)*** | |
| ***## step 1: identify the mode value*** | ***month\_mode= bank1.month.mode()[0]***  ***month\_mode*** |
| ***## step 2: Replacing the missing value*** | ***bank1.month.fillna(month\_mode, inplace=True)***  ***bank1.month.isnull().sum()*** |
|  |  |
|  |  |

1. ***Handling outliers***

|  |  |
| --- | --- |
|  |  |
| ***## Histogram*** | ***#plot the histogram of age variable.***  ***plt.figure(figsize=[8,2])***  ***inp1.age.plot.hist()***  ***plt.show()*** |
| ***## Boxplot*** | ***#plot the boxplot of age variable.***  ***plt.figure(figsize=[8,2])***  ***sns.boxplot(inp1.age)***  ***plt.show()*** |
|  |  |

1. ***Standardizing the values***

|  |  |
| --- | --- |
| ***## Quantile Value*** | ***#print the quantile (0.5, 0.7, 0.9, 0.95 and 0.99) of balance variable***  ***inp1.balance.quantile([0.5, 0.7, 0.9, 0.95 ,0.99])*** |
| ***## Converting from Second to Minute*** | ***#convert the duration variable into single unit i.e. minutes. and remove the sec or min prefix.***  ***inp1.duration = inp1.duration.apply(lambda x:float(x.split()[0])/60 if x.find('sec')>0 else float(x.split()[0]))*** |
|  |  |
|  |  |

1. ***Fixing invalid values***
2. ***Filtering the data***

df[column].unique()

***Doubt Session:***

1. ***#describe the pdays column of inp1.***

split date and time+pandas column+pd.datetime+python+pandas+stackoverflow

Python sample problems

|  |  |
| --- | --- |
|  | ***df[column].unique()*** |
|  | ***df[column].value\_counts()*** |
| ***use a bar plot in order to look at our missing values:*** | ***fig, ax = plt.subplots(figsize=(15, 5))x = df.isna().sum().index y = df.isna().sum() ax.bar(x=x, height=y) ax.set\_xticklabels(x, rotation = 45) plt.tight\_layout();*** |
|  |  |
|  |  |
|  |  |
|  |  |
| Table of Contents   * [**Rename column**](https://queirozf.com/entries/pandas-dataframe-examples-column-operations#rename-column) * [**Apply function to column names**](https://queirozf.com/entries/pandas-dataframe-examples-column-operations#apply-function-to-column-names) * [**Apply function to column**](https://queirozf.com/entries/pandas-dataframe-examples-column-operations#apply-function-to-column) * [**Create derived column**](https://queirozf.com/entries/pandas-dataframe-examples-column-operations#create-derived-column) * [**Number of NaNs in column**](https://queirozf.com/entries/pandas-dataframe-examples-column-operations#number-of-nans-in-column) * [**Get column names**](https://queirozf.com/entries/pandas-dataframe-examples-column-operations#get-column-names) * [**Get number of columns**](https://queirozf.com/entries/pandas-dataframe-examples-column-operations#get-number-of-columns) * [**Change column order**](https://queirozf.com/entries/pandas-dataframe-examples-column-operations#change-column-order) * [**Drop column**](https://queirozf.com/entries/pandas-dataframe-examples-column-operations#drop-column) * [**Drop multiple columns**](https://queirozf.com/entries/pandas-dataframe-examples-column-operations#drop-multiple-columns) * [**Append new column**](https://queirozf.com/entries/pandas-dataframe-examples-column-operations#append-new-column) * [**Check if column exists**](https://queirozf.com/entries/pandas-dataframe-examples-column-operations#check-if-column-exists) * [**Insert column at specific index**](https://queirozf.com/entries/pandas-dataframe-examples-column-operations#insert-column-at-specific-index) * [**Convert column to another type**](https://queirozf.com/entries/pandas-dataframe-examples-column-operations#convert-column-to-another-type) * [**Convert column to date/datetime**](https://queirozf.com/entries/pandas-dataframe-examples-column-operations#convert-column-to-date-datetime) * [**map example**](https://queirozf.com/entries/pandas-dataframe-examples-column-operations#map-example)   + [**map vs apply: time comparison**](https://queirozf.com/entries/pandas-dataframe-examples-column-operations#map-vs-apply-time-comparison) |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | |
| ***##map vs apply: time comparison (One of the most striking differences between the .map() and .apply() functions is that apply() can be used to employ Numpy vectorized functions. This gives massive (more than 70x) performance gains, as can be seen in the following example: Time comparison: create a dataframe with 10,000,000 rows and multiply a numeric column by 2)***  ***Step1:***  ***#create a sample dataframe with 10,000,000 rows***  ***df = pd.DataFrame({'x': np.random.normal(loc=0.0, scale=1.0, size=10000000)})***  ***Step2:***  ***Using map function multiply 'x' column by 2***  ***def multiply\_by\_two\_map(x):***  ***return x\*2***  ***df['2x\_map'] = df['x'].map(multiply\_by\_two)***  ***# >>> CPU times: user 14.4 s, sys: 300 ms, total: 14.7 s***  ***# >>> Wall time: 14.7 s***  ***Step3:***  ***Using apply function multiply 'x' column by 2***  ***import numpy as np***  ***def multiply\_by\_two(arr):***  ***return np.multiply(arr,2)***  ***# note the double square brackets around the 'x'!!***  ***# this is because we want to use DataFrame.apply,***  ***# not Series.apply!!***  ***df['2x\_apply'] = df[['x']].apply(multiply\_by\_two)***  ***# >>> CPU times: user 80 ms, sys: 112 ms, total: 192 ms***  ***# >>> Wall time: 188 ms***  [***Refer Here>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>***](https://queirozf.com/entries/pandas-dataframe-examples-column-operations#rename-column) | |
|  | |
|  | |
| ***#Rename column***  ***df.rename(columns={'name':'person\_name'})***  ***#change multiple column names***  ***df.rename(columns={'name':'person\_name','age':'age\_in\_years'})***  ***# convert column NAMES to uppercase***  ***df.columns = [col.upper() for col in df.columns]***  ***df***  ***# convert all the names in a column to uppercase***  ***df['name'] = df['name'].map(lambda name: name.upper())***  ***#Create derived column***  ***df['age\_times\_two']= df['age'] \*2***  ***df***  ***#Create derived column from name column***  ***df["name\_uppercase"] = df['name'].map(lambda name: name.upper())***  ***df***  ***#Number of NaNs in column***  ***df['name'].isnull().sum()***  ***#Get column names***  ***df.columns.values***  ***df***  ***#Get number of columns***  ***len(df.columns.values)***  ***#Change column order (reassigning the dataframe by selecting the columns in the order you want), if initial order is [age, name , state]***  ***df = df[['name','age','state']]***  ***df***  ***#Drop column***  ***df.drop(columns=['age'])***  ***#Drop multiple columns***  ***df.drop(columns=['age','name'])***  ***#Append new column (In order to add a new column to a DataFrame, create a Series and assign it as a new column)***  ***states = pd.Series(['dc','ca','ny'])***  ***df['state'] = states***  ***df***  ***#Check if certain column exists or not (from a series of name say candidate\_names = ['name','gender','age'])***  ***candidate\_names = ['name','gender','age']***  ***for name in candidate\_names:***  ***if name in df.columns.values:***  ***print('"{}" is a column name'.format(name))***  ***The outputs will come with name of the column which are available in dataframe (for ex: "name" is a column name, "age" is a column name)***  ***#Insert column at specific index***  ***df.insert(<position>,<column\_name>, <data>)***  ***#Insert column at specific index ( from a series, say column\_data = pd.Series(['female','male','male'] to insert at position = 1 & column\_name = 'gender')***  ***df.insert(position=1, column='gender' , column\_data)***  ***Check data type of a data frame:***  ***df2['age'].dtype***  ***#Convert the data type of a column to another another data type***  ***df2['age'] = df2['age'].astype(str)***  ***df2['age'] = df2['age'].astype(np.uint8)***  ***#Convert column to date/datetime (where %d/%m/%Y should be given as per input format of date and time)***  ***df['date\_of\_birth'] = pd.to\_datetime(df['date\_of\_birth'],format='%d/%m/%Y')***  ***#map() to create derived columns***  ***Divide one column by another to create/add a new column in Dataframe:***  ***df7['average'] = df7['age']/df7['number\_of\_children']***  ***df7['average']*** | |

Machine Learning Model Process

# Import Libraries

# Read, Understand and Basic Manipulation of the Data

|  |
| --- |
| bikedata = pd.read\_csv("/content/gdrive/My Drive/day.csv")  bikedata.head() |
| bikedata.shape |
| bikedata.info() |
| bikedata.describe() |
| bikedata['dteday'] = bikedata['dteday'].astype('datetime64') |
| Segregate the **Categorical & Numerical** Variables #Unique Values to identify variable type  bikedata.nunique().sort\_values() |

# Data Visualization

|  |
| --- |
| Data Visualization of Numerical Data sns.pairplot(bikedata)  plt.show() |
| Create a dataframe with numerical features to be viewed on pairplot  bikedata\_numerical = bikedata[['temp','atemp','hum','windspeed','cnt']]  #sns.pairplot(bikedata\_numerical,x\_vars=['cnt'], y\_vars=['temp','atemp','hum','windspeed'])  sns.pairplot(bikedata\_numerical)  plt.show() |
| View the correlation between the numerical variables  plt.figure(figsize=(12, 8))  sns.heatmap(bikedata\_numerical.corr(),annot=True)  plt.show() |
| Data Visualization of Categorical Data |
| plt.figure(figsize=(20, 12))  plt.subplot(2,3,1)  sns.boxplot(x = 'yr', y = 'cnt', data = bikedata)  plt.subplot(2,3,2)  sns.boxplot(x = 'mnth', y = 'cnt', data = bikedata)  plt.subplot(2,3,3)  sns.boxplot(x = 'weekday', y = 'cnt', data = bikedata)  plt.subplot(2,3,4)  sns.boxplot(x = 'weathersit', y = 'cnt', data = bikedata)  plt.subplot(2,3,5)  sns.boxplot(x = 'season', y = 'cnt', data = bikedata)  plt.subplot(2,3,6)  sns.boxplot(x = 'holiday', y = 'cnt', data = bikedata)  plt.show() |

# Data Preparation

# Split the data into Train and Test

# Rescaling the Features

# Model Building

# Residual Analysis

# Insights and Predictions

# from \_\_future\_\_ import division

import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

import os

for dirname, \_, filenames in os.walk('nyc\_airbnb'):

    for filename in filenames:

        print(os.path.join(dirname, filename))

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline

import warnings

warnings.filterwarnings('ignore')

from sklearn import preprocessing

from sklearn.linear\_model import LinearRegression

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

from sklearn import metrics

sns.set\_style('darkgrid')