pr01\_DataWrangling\_1

import pandas as pd

import numpy as np

pwd

data=pd.read\_csv('vehicles.csv')

data.head()

data.tail()

data.info()

data.isnull

data.shape

data.columns

type("size")

df=['year','price','size']

data[df]

data2 = data[df].head(10)

data2

caqu = pd.DataFrame(data, columns = ['state'])

caqu\_state = pd.get\_dummies(caqu['state'])

caqu\_new = pd.concat([caqu, caqu\_state], axis=1)

print(caqu\_new)

Pr02\_DataWrangling\_2

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

df = pd.read\_csv('StudentsPerformance.csv')

df

df.head()

df.tail()

df.describe()

df.isnull()

def plot\_boxplot(df,ft):

    df.boxplot(column=[ft])

    plt.grid(False)

plt.show()

plot\_boxplot(df,'math score')

def plot\_boxplot(df,ft):

    df.boxplot(column=[ft])

    plt.grid(False)

plt.show()

plot\_boxplot(df,'reading score')

def outliers(df,ft):

    Q1=df[ft].quantile(0.25)

    Q3=df[ft].quantile(0.75)

    IQR=Q3-Q1

    lower\_bound=Q1-1.5 \*IQR

    upper\_bound=Q3 +1.5 \*IQR

    ls=df.index[(df[ft] < lower\_bound) | (df[ft] > upper\_bound)]

    return ls

index\_list=[]

for features in ['math score','reading score']:

    index\_list.extend(outliers(df,features))

index\_list

def remove(df,ls):

    ls=sorted(set(ls))

    df=df.drop(ls)

    return df

df\_cleaned=remove(df,index\_list)

df\_cleaned.shape

plot\_boxplot(df\_cleaned,'math score')

plot\_boxplot(df\_cleaned,'reading score')

Pr03\_DescriptiveStatistics

import pandas as pd

import numpy as np

data = pd.read\_csv('toy\_dataset.csv')

data.head()

data.tail()

data.columns

data1 = data.iloc[0:51, 3:5]

data1

data1.mean()

data1.median()

data1.count()

data1.sum()

data1['Total'].sum()

data1[['Total','gross income']].sum()

data1.mean()

data1['Rating'].mean()

data1.std()

data1['Rating'].std()

print('The Mode of : ',data1['Payment'].mode)

Pr04\_DataAnalytics\_1

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

data = pd.read\_csv('HousingData.csv')

data.head()

data.tail()

data.describe()

data.dropna()

data.shape

data.dtypes

data.info()

data.isna().sum()

mean\_value = data['CRIM'].mean()

# Calculate the mean of each numeric column

means = data.mean()

# Impute missing values with the mean values

data.fillna(value=means, inplace=True)

# Check for any remaining missing values

print(data.isnull().sum())

target\_feature = 'MEDV'

# Splitting the dataset

x = data.drop(target\_feature, axis=1)

y = data[target\_feature]

x.head()

y.head()

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

x\_train,x\_test,y\_train,y\_test = train\_test\_split(x,y,test\_size=0.2,random\_state=0)

from sklearn.linear\_model import LinearRegression

# Create an instance of a LinearRegression() model named regression.

regression = LinearRegression()

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

# train score

train\_score=round(regression.score(x\_train,y\_train)\*100,2)

print('Train score of Linear Regression:',train\_score)

# Print out the coefficients of the model

print('Coefficients" ', regression.coef\_)

predictions = regression.predict(x\_test)

predictions

plt.scatter(y\_test, predictions)

plt.xlabel('Y Test')

plt.ylabel('Predicted Y')

from sklearn.metrics import r2\_score

score = round(r2\_score(y\_test,predictions)\*100,2)

print("r\_2 score:", score)

round(regression.score(x\_test, y\_test)\*100,2)

from sklearn import metrics

print('Mean Absolute Error on test data of Linear Regression: ',metrics.mean\_absolute\_error(y\_test, predictions))

print('Mean Squared Error on test data of Linear Regression: ',metrics.mean\_squared\_error(y\_test, predictions))

print('Root Mean Squared Error on test data of Linear Regression: ',np.sqrt(metrics.mean\_squared\_error(y\_test,predictions)))

df1 = pd.DataFrame({'Actual':y\_test, 'Predicted':predictions, 'Variance':y\_test-predictions})

df1.head()

data.head(15)

regression.predict([[0.62976,0.0,8.14,0.0,0.538,5.949,61.8,4.7075,4,307,21.0,396.60,8.26]])

regression.intercept\_

regression.coef\_

lr\_coefficient = pd.DataFrame()

lr\_coefficient["columns"] = x\_train.columns

lr\_coefficient['Coefficient Estimate'] = pd.Series(regression.coef\_)

print(lr\_coefficient)

# plotting the coefficient score

fig, ax = plt.subplots(figsize =(20, 10))

ax.bar(lr\_coefficient["columns"],

lr\_coefficient['Coefficient Estimate'])

ax.spines['bottom'].set\_position('zero')

plt.style.use('ggplot')

plt.grid()

plt.show()

fig, ax = plt.subplots(figsize =(20, 10))

color = ['tab:gray', 'tab:blue', 'tab:orange', 'tab:green', 'tab:red', 'tab:purple', 'tab:brown', 'tab:pink', 'tab:gray', 'tab:blue', 'tab:orange', 'tab:green']

ax.bar(lr\_coefficient["columns"],

lr\_coefficient['Coefficient Estimate'],color = color)

ax.spines['bottom'].set\_position('zero')

plt.style.use('ggplot')

plt.show()

Pr05\_DataAnalytics\_2

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read\_csv('Social\_Network\_Ads.csv')

dataset.head()

X = dataset.iloc[:, [2, 3]].values

y = dataset.iloc[:, 4].values

print(X[:3, :])

print('-'\*15)

print(y[:3])

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size =

0.25, random\_state = 0)

print(X\_train[:3])

print('-'\*15)

print(y\_train[:3])

print('-'\*15)

print(X\_test[:3])

print('-'\*15)

print(y\_test[:3])

from sklearn.preprocessing import StandardScaler

sc\_X = StandardScaler()

X\_train = sc\_X.fit\_transform(X\_train)

X\_test = sc\_X.transform(X\_test)

print(X\_train[:3])

print('-'\*15)

print(X\_test[:3])

from sklearn.linear\_model import LogisticRegression

classifier = LogisticRegression(random\_state = 0, solver='lbfgs' )

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

y\_pred = classifier.predict(X\_test)

print(X\_test[:10])

print('-'\*15)

print(y\_pred[:10])

print(y\_pred[:20])

print(y\_test[:20])

from sklearn.metrics import classification\_report

classification\_report(y\_test,y\_pred)

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

print(cm)

from sklearn.metrics import accuracy\_score

accuracy\_score(y\_test,y\_pred)

# Visualizing the Training set results

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_train, y\_train

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop =

X\_set[:, 0].max() + 1, step = 0.01),

 np.arange(start = X\_set[:, 1].min() - 1, stop =

X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),

X2.ravel()]).T).reshape(X1.shape),

 alpha = 0.6, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

 plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

 c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('Logistic Regression (Training set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

# Visualizing the Test set results

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_test, y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop =

X\_set[:, 0].max() + 1, step = 0.01),

 np.arange(start = X\_set[:, 1].min() - 1, stop =

X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),

X2.ravel()]).T).reshape(X1.shape),

 alpha = 0.6, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

 plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

 c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('Logistic Regression (Test set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

Pr06\_DataAnalytics\_3

import pandas as pd

import numpy as np

df=pd.read\_csv("iris.csv")

df

df.dtypes

df['class']=df['class'].astype('category')

df.dtypes

df['class']=df['class'].cat.codes

df

df.isnull().sum()

(df <= 0).sum()

print(df.shape)

# co-relation matrix

def DetectOutlier(df,var):

 Q1 = df[var].quantile(0.25)

 Q3 = df[var].quantile(0.75)

 IQR = Q3 - Q1

 high, low = Q3+1.5\*IQR, Q1-1.5\*IQR

 print("Highest allowed in variable:", var, high)

 print("lowest allowed in variable:", var, low)

 count = df[(df[var] > high) | (df[var] < low)][var].count()

 print('Total outliers in:',var,':',count)

DetectOutlier(df,'sepallength')

DetectOutlier(df,'sepalwidth')

DetectOutlier(df,'petallength')

DetectOutlier(df,'petalwidth')

import seaborn as sns

sns.boxplot(df['sepalwidth'])

def OutlierRemoval(df,var):

 Q1 = df[var].quantile(0.25)

 Q3 = df[var].quantile(0.75)

 IQR = Q3 - Q1

 high, low = Q3+1.5\*IQR, Q1-1.5\*IQR

 print("Highest allowed in variable:", var, high)

 print("lowest allowed in variable:", var, low)

 count = df[(df[var] > high) | (df[var] < low)][var].count()

 print('Total outliers in:',var,':',count)

 df = df[((df[var] >= low) & (df[var] <= high))]

 return df

print(df.shape)

df = OutlierRemoval(df,'sepalwidth')

print(df.shape)

import seaborn as sns

sns.heatmap(df.corr(),annot=True)

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import seaborn as sns

# split the data into inputs and outputs

X = df.iloc[:, [0,2,3]].values

y = df.iloc[:, 4].values

# training and testing data

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

# assign test data size 25%

X\_train, X\_test, y\_train, y\_test =train\_test\_split(X,y,test\_size=

0.25, random\_state=0)

# importing standard scaler

from sklearn.preprocessing import StandardScaler

# scalling the input data

sc\_X = StandardScaler()

X\_train = sc\_X.fit\_transform(X\_train)

X\_test = sc\_X.fit\_transform(X\_test)

# importing standard scaler

from sklearn.preprocessing import StandardScaler

# scalling the input data

sc\_X = StandardScaler()

X\_train = sc\_X.fit\_transform(X\_train)

X\_test = sc\_X.fit\_transform(X\_test)

# import Gaussian Naive Bayes classifier

from sklearn.naive\_bayes import GaussianNB

# create a Gaussian Classifier

classifer1 = GaussianNB()

# training the model

classifer1.fit(X\_train, y\_train)

# testing the model

y\_pred1 = classifer1.predict(X\_test)

# importing accuracy score

from sklearn.metrics import accuracy\_score

# printing the accuracy of the model

print(accuracy\_score(y\_test,y\_pred1))

0.8648648648648649

# importing the required modules

import seaborn as sns

from sklearn.metrics import confusion\_matrix

# passing actual and predicted values

cm = confusion\_matrix(y\_test, y\_pred1)

# true write data values in each cell of the matrix

sns.heatmap(cm, annot=True)

plt.savefig('confusion.png')

# importing classification report

from sklearn.metrics import classification\_report

# printing the report

print(classification\_report(y\_test, y\_pred1))

Pr07\_textAnalysis

import nltk

nltk.download('punkt')

nltk.download('wordnet')

nltk.download('averaged\_perceptron\_tagger')

nltk.download('stopwords')

from nltk import sent\_tokenize

from nltk import word\_tokenize

from nltk.corpus import stopwords

text='Real madrid is set to win the UCL for the season . Benzema might win Balon dor . Salah might be the runner up'

tokens\_sents = nltk.sent\_tokenize(text)

print(tokens\_sents)

tokens\_words = nltk.word\_tokenize(text)

print(tokens\_words)

from nltk.stem import PorterStemmer

from nltk.stem.snowball import SnowballStemmer

from nltk.stem import LancasterStemmer

stem=[]

for i in tokens\_words:

 ps = PorterStemmer()

 stem\_word= ps.stem(i)

 stem.append(stem\_word)

print(stem)

import nltk

from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

lemmatized\_output = ' '.join([lemmatizer.lemmatize(w) for w in stem])

print(lemmatized\_output)

leme=[]

for i in stem:

 lemetized\_word=lemmatizer.lemmatize(i)

 leme.append(lemetized\_word)

print(leme)

Pr08\_DataVisualization\_1

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

dataset = sns.load\_dataset('titanic')

dataset.head()

dataset.shape

dataset.isnull()

dataset.isnull().sum()

# remove all null values from the dataset

dataset = dataset.dropna()

# Let's see how the price of the ticket for each passenger is distributed.

sns.distplot(dataset['fare'])

sns.distplot(dataset['fare'], kde=False)

sns.distplot(dataset['fare'], kde=False, bins=10) #Here we set the number of bins to 10.

Pr09\_DataVisualization\_2

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from seaborn import load\_dataset

data = pd.read\_csv('titanic\_train.csv')

tips = load\_dataset("tips")

sns.boxplot(x=data['Sex'], y=data['Age'])

plt.show()

sns.boxplot(x='Sex', y='Age', hue='Survived', data=data)

plt.show()

data

sns.boxplot(x = 'Survived', y = 'Age', hue = 'Sex', data = data)

plt.figure(figsize = (12, 8))

sns.boxplot(x = 'Sex', y = 'Age', hue = 'Survived', palette = 'Set3',

data = data)

sns.boxplot(x = 'Sex', y = 'Age', hue = 'Survived', palette = 'Set3',

data = data, linewidth = 2.5, order = ['female', 'male'])

Pr10\_dataVisualization\_3

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

data = pd.read\_csv('iris.csv')

data.head(10)

data.tail()

data.info()

data.columns

print(data.describe(exclude=[object]))

print(data.describe(include=[object]))

categorical\_features = [column\_name for column\_name in data.columns if data[column\_name].dtype == 'O']

print("Number of Categorical Features: {}".format(len(categorical\_features)))

print("Categorical Features: ",categorical\_features)

numerical\_features = [column\_name for column\_name in data.columns if data[column\_name].dtype != 'O']

print("Number of Numerical Features: {}".format(len(numerical\_features)))

print("Numerical Features: ",numerical\_features)

data.isnull().sum()

data.shape

import matplotlib.pyplot as plt

fig, axes = plt.subplots(2, 2, figsize = (16, 9))

sns.histplot(data['sepallength'], ax = axes[0, 0])

sns.histplot(data['sepalwidth'], ax = axes[0, 1])

sns.histplot(data['petallength'], ax = axes[1, 0])

import matplotlib.pyplot as plt

fig, axes = plt.subplots(2, 2, figsize = (16, 9))

sns.boxplot(y = 'petallength', x = 'class', data = data, ax =

axes[0,0])

sns.boxplot(y = 'petalwidth', x = 'class', data = data, ax =

axes[0,1])

sns.boxplot(y = 'sepallength', x = 'class', data = data, ax =

axes[1,0])

sns.boxplot(y = 'sepalwidth', x = 'class', data = data, ax =

axes[1,1])