**Data Science Notes**

## Exploratory Data Analysis

1. Import pandas as pd
2. Import and review data set
   1. **df = pd.readcsv(“dataname.csv”)** # places dataset into a Pandas data frame.
   2. **df.head()** # shows the first 5 rows of the data for inspection, passing in a parameter will change the number of rows provided.
   3. **df.shape** # shows the number of rows and columns in the data set.
   4. **df.types()** # shows the type of each column.
   5. **df[‘area\_error’].unique()** # shows the unique values for a given column.
   6. **df.info()** # provides more information about the dataset.
   7. **df.describe** # shows a set of basic descriptive statistics for each column.
   8. **df.describe(percentiles=[0.25, 0.5])** # shows the percentile values for points provided
   9. **df[‘area error’].value\_count()** # shows count for categorial variables
   10. **df.corr() –** produces a correlation matrix
3. Inspect for NaN Values
   1. **df.isna()**
   2. df.isna().sum()
   3. df.isna().sum()/df.shape[0] # percent NaNs
   4. df.isna().sum()/df.shape[0].sort\_values()
   5. df.isna().sum()/df.shape[0].sort\_values().plot(kind=”bar”)
   6. I need to include a method to deal with NaNs.
4. Dealing with Missing Values – **Simple Imputer**
   1. Import numpy as np
   2. from sklearn.impute import SimpleImputer
   3. imputer = SimpleImputer(missing\_values = np.nan, strategy = ‘mean’)
      1. missing\_values = [np.nan, 0, np.inf]
      2. other methods include: median, most\_frequent, and constant
   4. data = target dataframe
   5. imputer.fit(data)
   6. transformed\_data = imputer.transform(data)
   7. print(transformed\_data)
   8. transformed\_data.head()
5. Dealing with Missing Vales – **Iterative Imputer**
   1. Import numpy as np
   2. From sklearn.experimental import enable\_iterative\_imputer
   3. From sklearn.impute import IterativeImputer
   4. Imputer = IterativeImputer()
   5. Data = target dataframe
   6. Imputer.fit(data)
   7. transformed\_data = imputer.transform(data)
   8. print(transformed\_data)
   9. transformed\_data.head()
6. Dealing with Missing Values – **KNN Imputer**
   1. Import numpy as np
   2. From sklearn.impute import KNNImputer
   3. Imputer = KNNImputer(n\_neighbors = 2, weights = ‘uniform’)
   4. Data = target dataframe
   5. Imputer.fit(Data)
   6. Transformed\_data = imputer.transform(data)
   7. print(transformed\_data)
   8. transformed\_data.head()
7. Dealing with Missing Values Alternative Approach
   1. Replace NaN with the Mean in One Column
      1. df[‘Time\_taken’].fillna(value=df[‘Time\_taken’].mean(), inplace=True)
   2. Replace all missing values in a Dataframe
      1. df = df.fillna(df.mean())
8. Convert Categorical Variables to Dummies
   1. df = pd.**get\_dummies**(df, columns=[‘target1’, ‘target2’], drop\_first=True)
9. Standardize Data with the **StandardScaler()** – Fits everything to a normal curve, most observations should be between -3 and +3, other might be outliers.
   1. Import numpy as py
   2. From sklearn import preprocessing
   3. Import matplotlib.pyplot as plt
   4. Data = target dataframe
      1. Sample: data = np.random.normal(10,20,10000).reshape(10000,1)
   5. Scaler = preposcessing.StandardScaler()
   6. Scaler.fit(data)
   7. Transformed\_data = scaler.transform(data)
   8. Plt.hist(data, bons = 100)
   9. Plt.show()
   10. Plt.hist(transformed\_data, bins = 100)
   11. Plt.show()
10. Normalize Data – keeps the data in it natural distribution, does not force to a curve.
    1. Import numpy as py
    2. From sklearn import preprocessing
    3. Import matplotlib.pyplot as plt
    4. Data = target dataframe
       1. Sample: data = np.random.random((1, 5)) \* 100
    5. Transformed\_data = preprocessing.normalize(data)
    6. Print(data)
    7. Print(transformed\_data)
11. **RobustScaler()** – used like the standard scaler
12. **Discretization** – This is to turn continuous variables into discrete variables
    1. Import numpy as py
    2. From sklearn import preprocessing
    3. Import matplotlib.pyplot as plt
    4. Data = (np.random.random((10)\*10).reshape(10,1)
       1. Or target dataframe
    5. Discretizer = preprocessing.KBinsDiscretizer(n\_bins = 3, encode = ‘ordinal’)
    6. Discretizer.fit(data)
    7. Transformed\_data = discretizer.transform(data)
    8. Print(data)
    9. Print(transformed\_data)
13. **One Hot Encoding** – Categorical Variables to Numbers
    1. Import numpy as py
    2. From sklearn import preprocessing
    3. Import matplotlib.pyplot as plt
    4. Data = target dataframe
    5. Encoder = preprocessing.OneHotEncoder()
    6. Encoder.fit(data)
    7. Transformed\_data = encoder.transform(data).toarray()
    8. Print(data)
    9. Print(transformed\_data)
14. **Outlier Processing** 
    1. Use Seaborn to inspect for potential outliers. Assumes data set is loaded in a data frame call df. This process will use capping and flooring to limit the effects of outliers.
       1. df.info()
       2. df.describe()
    2. **High Outlier Processing**
       1. np.percentile(df.fieldname,[99])
          1. # this command returns an array of observations at or above the 99th percentile as an array
       2. np.percentile(df.fieldname,[99])[0]
          1. # this command gets the first item of the returned array
       3. uv = np.percentile(df.fieldname,[99])[0]
          1. # stores the above item in a variable – upper value
       4. df[(df.fieldname > uv)]
          1. # Shows all observations in this column above 99 percentile
       5. df.fieldname[(df.fieldname > 3\*uv)] = 3\*uv
       6. # This command caps all high outliers at upper value times 3. This still seems high to me but must reduce the effect of the high outliers.
15. **Low Outlier Processing**
    1. lv = np.percentile(df.fieldname,[1])[0]
    2. df[(df.fieldname) < lv]
    3. df.fieldname[(df.fieldname < 0.3 \* lv)] = lv\*0.3Basic plots to review data
16. **Repeated Data**
    1. Used when multiple variable contain the same information. This approach will use the mean, but min and max values could also be used.
       1. df[‘avg\_dist’] = (df.dist1 + df.dist2 + df.dist3 + df.dist4 ) / 4
    2. then remove old variables
       1. del df[‘dist1’], del df[‘dist2’], del df[‘dist3’], del df[‘dist4’]
17. **Correlation Analysis**
    1. Correlation Matrix = df.corr()
    2. Drop one of highly correlated variables, and insignificant variables
       1. del df.[‘parks’]
18. **Non-linear Data**
    1. This approach can be used to improve data that is non-linear, such as an exponential relationship into something that is more linear.
    2. Examine the relationship between two variables
       1. sns.jointplot(data=df, x=”crime\_rate”, y=”price”)
       2. # shows the relationship between two variables. In this case between one independent variable and the dependent variable.
       3. Apply 1 + log to the data
       4. df.crime\_rate = np.log(1 + df.craime\_rate)
    3. Regraph the data
       1. sns.jointplot(data=df, x=”crime\_rate”, y=”price”)
19. **MatPlotLib Plots**

# MatpoltLib Scatter Plots - First Look at Relationships

fig,axes = plt.subplots(nrows=1,ncols=3,figsize=(16,6))

axes[0].plot(df['TV'],df['sales'],'o')

axes[0].set\_ylabel("Sales")

axes[0].set\_title("TV Spend")

axes[1].plot(df['radio'],df['sales'],'o')

axes[1].set\_ylabel("Sales")

axes[1].set\_title("Radio Spend")

axes[2].plot(df['newspaper'],df['sales'],'o')

axes[2].set\_ylabel("Sales")

axes[2].set\_title("Newspaper Spend")

plt.tight\_layout();

1. **Seaborn Pair Plots** – Compares Features
   1. sns.pairplot(df)
   2. sns.pairplot(df.iloc[:, 0:5))
   3. kind=hist, kde
   4. hue=”target” # for categorical variable
   5. corner=True
2. **Histograms**
   1. df.hist() # basic Pandas histograms for all number variables
   2. df.iloc[:, 0:4].hist() # histograms of the first four columns
   3. can also pass in bins=? Or bins=”rice” to alter number of bins
3. **Box Plots**
   1. df.columns # lists column names
   2. df[[‘mean radius’]].boxplot()
   3. df[[‘mean radius’, ‘next column’]].boxplot()
   4. df.iloc[:, 0:5].boxplot()
4. **Sweetviz** – data analysis
   1. import sweetviz # had to install with pip first, not conda
   2. report = sweetviz.analyze(df, target\_feat="target")
   3. report.show\_html(layout="vertical")

# Machine Learning Algorithms

### Sci-Kit Learn for Python

Claims to be the most popular Machine Library for Python

**statsmodels** – model metrics for Python

### Train Test Split

Assumes basic libraries and dataset have been imported and df.head() has been called.

### Split Features and Labels

**X** = df.drop(‘label\_column’,axis=1)

**y** = df[‘label\_column]

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

# help(train\_test\_split) # learn more about the hyperparameters of this functions

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=101)**

# Classification Models

## Classification Metics

**Accuracy** – How often is the model correct? (True Positives + True Negatives) / Total Observations

**Recall** – When positive how often is the model correct? True Positives / Total Positives

**Precision** – When positive how often is the prediction correct? Total Positives / Total Predicted Positives

**F1 Score** – F1 Score = 2 X precision X recall / precision + recall

## Logistic Regression – Simple

The

### Exploratory Data Analysis

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

df = pd.read\_csv('hearing\_test.csv')

df.head()

df.describe()

df['test\_result'].value\_counts()

plt.figure(dpi=150)

sns.countplot(data=df, x='test\_result')

plt.figure(dpi=150)

sns.boxplot(x='test\_result', y='age', data=df)

plt.figure(dpi=150)

sns.boxplot(x='test\_result', y='physical\_score', data=df)

plt.figure(dpi=150)

sns.scatterplot(x='age', y='physical\_score', data=df, hue='test\_result', alpha=0.75)

plt.figure(dpi=150)

sns.pairplot(data=df, hue='test\_result')

plt.figure(dpi=150)

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

from mpl\_toolkits.mplot3d import Axes3D

# plt.figure(dpi=550)

fig = plt.figure()

ax = fig.add\_subplot(111, projection='3d')

ax.scatter(df['age'],df['physical\_score'],df['test\_result'],c=df['test\_result'])

### Split and Train Logistic Regression Model

df.head()

X = df.drop('test\_result', axis=1)

y = df['test\_result']

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

from sklearn.preprocessing import StandardScaler

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.1, random\_state=101)

scaler = StandardScaler()

scaled\_X\_train = scaler.fit\_transform(X\_train)

scaled\_X\_test = scaler.transform(X\_test)

from sklearn.linear\_model import LogisticRegression

log\_model = LogisticRegression()

log\_model.fit(scaled\_X\_train, y\_train)

log\_model.coef\_

y\_pred = log\_model.predict(scaled\_X\_test)

y\_pred

y\_pred = log\_model.predict\_proba(scaled\_X\_test)

y\_pred

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

y\_pred = log\_model.predict(scaled\_X\_test)

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

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

from sklearn.metrics import plot\_confusion\_matrix

plot\_confusion\_matrix(log\_model, scaled\_X\_test, y\_test)

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

from sklearn.metrics import precision\_score, recall\_score

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

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

from sklearn.metrics import plot\_precision\_recall\_curve, plot\_roc\_curve

## Multiple Logistic Regression

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

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

df.head()

df.describe()

df.info()

df['species'].value\_counts()

plt.figure(dpi=150)

sns.countplot(x='species', data=df)

plt.figure(dpi=150)

sns.scatterplot(x='petal\_length', y='petal\_width', data=df, hue='species')

plt.figure(dpi=150)

sns.pairplot(data=df, hue='species')

plt.figure(dpi=150)

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

#### Train\_Test\_Split

X = df.drop('species', axis=1)

y = df['species']

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

from sklearn.preprocessing import StandardScaler

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=101)

scaler = StandardScaler()

scaled\_X\_train = scaler.fit\_transform(X\_train)

scaled\_X\_test = scaler.transform(X\_test)

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import GridSearchCV

log\_model = LogisticRegression(solver='saga', multi\_class='ovr',

max\_iter=5000 )

penalty = ['l1', 'l2', 'elasticnet']

l1\_ratio = np.linspace(0, 1, 20)

C = np.logspace(0, 10, 20)

param\_grid = {'penalty': penalty,

'l1\_ratio': l1\_ratio,

'C': C }

grid\_model = GridSearchCV(log\_model, param\_grid=param\_grid)

grid\_model.fit(scaled\_X\_train, y\_train)

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report, plot\_confusion\_matrix

grid\_model.best\_params\_

y\_pred = grid\_model.predict(scaled\_X\_test)

y\_pred

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

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

plt.figure(dpi=150)

plot\_confusion\_matrix(grid\_model, scaled\_X\_test, y\_test)

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

## KNN – K Nearest Neighbor

Can but should not be used for regression, performs poorly

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

df = pd.read\_csv('gene\_expression.csv')

df.head()

df.describe()

plt.figure(dpi=150)

sns.scatterplot(data=df, x = 'Gene One', y = 'Gene Two',

hue = 'Cancer Present', alpha = 0.75, style='Cancer Present' )

# plt.xlim(2, 6)

# plt.ylim(4, 8)

len(df)

sns.pairplot(data=df, hue='Cancer Present')

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

from sklearn.preprocessing import StandardScaler

X = df.drop('Cancer Present', axis=1)

y = df['Cancer Present']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

scaler = StandardScaler()

scaled\_X\_train = scaler.fit\_transform(X\_train)

scaled\_X\_test = scaler.transform(X\_test)

from sklearn.neighbors import KNeighborsClassifier

# help(KNeighborsClassifier

knn\_model = KNeighborsClassifier(n\_neighbors=1)

knn\_model.fit(scaled\_X\_train, y\_train)

y\_pred = knn\_model.predict(scaled\_X\_test)

from sklearn.metrics import confusion\_matrix, classification\_report

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

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

df['Cancer Present'].value\_counts()

from sklearn.metrics import accuracy\_score

# Elbow Method to Find the Best K

test\_error\_rates = []

# A function to find the best K value

for k in range(1, 30):

knn\_model = KNeighborsClassifier(n\_neighbors=k)

knn\_model.fit(scaled\_X\_train, y\_train)

y\_pred\_test = knn\_model.predict(scaled\_X\_test)

test\_error = 1-accuracy\_score(y\_test, y\_pred\_test)

test\_error\_rates.append(test\_error)

test\_error\_rates

plt.figure(dpi=150)

plt.plot(range(1,30), test\_error\_rates)

plt.ylabel('Error Rate')

plt.xlabel('K Neighbors')

### PIPELINE --> GRID SEARCH CV

scaler = StandardScaler()

knn = KNeighborsClassifier()

knn.get\_params().keys()

operations = [('scaler', scaler), ('knn', knn)]

from sklearn.pipeline import Pipeline

pipe = Pipeline(operations)

from sklearn.model\_selection import GridSearchCV

k\_values = list(range(1, 20))

k\_values

param\_grid = {'knn\_\_n\_neighboes':k\_values}

full\_cv\_classifier = GridSearchCV(pipe, param\_grid, cv=5, scoring='accuracy')

full\_cv\_classifier.fit(X\_train, y\_train)

full\_cv\_classifier.best\_estimator\_.get\_params()

full\_pred = full\_cv\_classifier.predict(X\_test)

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

#### New Data Example

new\_patient = [[6.4, 3.8]]

full\_cv\_classifier.predict(new\_patient)

full\_cv\_classifier.predict\_proba(new\_patient)

## EDA (Exploratory Data Analysis)

The

Includes basic visualization

### One Approach to EDA

From a Udemy class that I did not finish, but this may be helpful

1. Variable Identification
2. Univariate Analysis
3. Multivariate Analysis
4. Missing Values Treatment
5. Outlier Treatment
6. Variable Transformation
7. Variable Creation

### **Draft Directory Structure** This is from Super Data Science

1. Original Data
2. Prepared Data
3. Uploaded Data
4. Analysis
5. Insights
6. Final

# Python Notes

Various related notes on the Python programing language.

## Python Virtual Environments

Why do we do these

### Capture Project Dependencies

Pip freeze > requirements.txt

### Install Project Dependencies

Python -m pip install requirements.txt

### Check for outdated packages

Pip list –outdated

### Upgrade Packages in \*.txt

Pip install -r requirements.txt --upgrade

## Python Type Hinting

**def** greeting(name: str) -> str:

**return** 'Hello ' + name

""" This is the Main file for the Adventure Game in PyCharm """

def print\_hi(name):

# Use a breakpoint in the code line below to debug your script.

print(f'Hi, {name}') # Press Ctrl+F8 to toggle the breakpoint.

if \_\_name\_\_ == '\_\_main\_\_':

print\_hi('Craig D Murray, SPHR') or other main function

## Active User Count Using OOP

Class User:

active\_users = 0

def \_\_init\_\_(self, first, last, age):

self.first = first

self.last = last

self.age = age

User.active\_users += 1

def logout(self):

User.active\_users -= 1

Logout logic and message

# Data Cleaning Outline

The first attempt

## Basic Data Science Imports

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

## Load and Test Data

# Read In Data

df = pd.read\_csv('hearing\_test.csv')

df.head()

df.describe()

df.shape

## Basic Plots

sns.boxplot(data=df)

sns.stripplot(data=df, marker='o', color='k', alpha=.5)

# Plotting 1st Round

sns.pairplot(df);

df.corr()

# Check For Repeated Rows

areRepeats = df.duplicated()

print(f'There are {np.sum(areRepeats)} repeated rows.')

# df = df.drop\_duplicates()

df.shape

# create new columns as z-scored versions

# Remember this only works with numbers

# create a lambda function (a simple one-line function)

zscore = lambda x: (x-x.mean()) / x.std()

# loop through the variables in the dataframe

for c in df.columns:

# apply the zscore function and map to a new column

df[c + '\_z'] = zscore(df[c])

# let's have a look!

print( df )

# df.describe()

## Remove Outliers

zThresh = 3

np.abs(df) > zThresh

# find outliers

row2kill = np.array([])

for c in df.columns:

if c[-1]=='z':

# find the outliers and update the list of rows to reject

hasoutliers = np.where(np.abs(df[c])>zThresh)[0]

row2kill = np.append(row2kill,hasoutliers)

# print a message

print(f'{c[:-2]} has an outlier (z>{zThresh}) in row(s) {hasoutliers}')

# let's see all the rows to reject:

row2kill

# remove those rows

df\_dropped = df.drop(row2kill)

df\_dropped.describe()

df\_na\_dropped = df\_dropped.dropna()

df\_na\_dropped

# Visualization

## Seaborn

Import seaborn as sns

plt.figure(dpi=150) # provides a bit better chart

### Line Plot

sns.lineplot(data=pokemon, x=”HP”, y=”Attack”)

### Scatter Plot

sns.scatterplot(data=pokemon, x=”HP”, y=”Attack”, (optional) hue=”Type”, col=”Type”)

### Relationship Plot

sns.relplot(data=pokemon, x=”HP”, y=”Attack”, (optional) hue=”Type”, col=”Type”, col\_wrap=3)

### Regression Plot – Plot data and a linear regression model fit.

sns.regplot(data=pokemon, x=”HP”, y=”Attack”)

### Pair Plot – relationships of all numeric features

sns.pairplot(pokemon)

sns.scatterplot(x='age', y='physical\_score', data=df, hue='test\_result', alpha=0.8)

### Histogram Plot

sns.histplot(data=pokemon, x="Attack") (optional bins=10 or other number)

### Kernel Density Estimate Plot

sns.kdeplot(data=pokemon, x="Attack")

### Combined Histogram and KDE Plot

sns.histplot(data=pokemon, x="Attack", bins=8, kde=True)

### Distribution Plot

sns.displot(data=pokemon, x="Attack", bins=10, col="Type", col\_wrap=3)

### Categorical Plots

sns.stripplot(data=pokemon, x='Type', y='Attack')

sns.catplot(kind='strip', data=pokemon, x='Type', y='Attack', aspect=2)

### Categorical Box Plot

sns.catplot(kind='box', data=pokemon, x='Type', y='Attack', aspect=2)

### Categorical Violin Plot

sns.catplot(kind='violin', data=pokemon, x='Type', y='Attack', aspect=2)

### Categorical Bar Plot

sns.catplot(kind='bar', data=pokemon, x='Type', y='Attack', aspect=2)

### Categorical Count Plot

sns.catplot(kind='count', data=pokemon, x='Type', aspect=2)

### Count Plot

sns.countplot(data=df, x='test\_result')

### Box Plot

sns.boxplot(x='test\_result', y='physical\_score', data=df)

## MatPlotLib Pyplot

### 3d Scatter Plot

from mpl\_toolkits.mplot3d import Axes3D

fig = plt.figure()

ax = fig.add\_subplot(111, projection='3d')

ax.scatter(df['age'],df['physical\_score'],df['test\_result'],c=df['test\_result'])

## Naïve Bays

By The Lazy Programmer

### Key Concepts

1. Applied machine learning is nothing but geometry.
2. All data is the same
   1. Naïve Bays will work on any data in any field.
3. Where does data come from
   1. Kaggle
   2. Survey and experiments
   3. Automated data collection
   4. Where can I find data
      1. Work
      2. Google

## Unsupervised Machine Learning

### K Means Clustering

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

df = pd.read\_csv('bank-full.csv')

df.head()

df.describe()

df.info()

# sns.pairplot(data=df)

plt.figure(figsize=(12, 6), dpi=200)

sns.histplot(data=df, x = 'age', bins=40, hue='loan', kde=True)

plt.figure(figsize=(12, 6), dpi=200)

sns.countplot(data=df, x='job', order=df['job'].value\_counts().index)

plt.xticks(rotation=90);

rotation=90);plt.figure(figsize=(12, 6), dpi=200)

sns.countplot(data=df, x='education', hue='default', order=df['education'].value\_counts().index)

plt.xticks(rotation=90);

sns.countplot(data=df, x='default')

df['default'].value\_counts()

df['loan'].value\_counts()

**Deal with Categorical Variables – convert to dummy variable**

df.head()

X = pd.get\_dummies(df)

X

**Scale Data to Bring Constant**

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaled\_X = scaler.fit\_transform(X)

**Build Cluster Model**

from sklearn.cluster import KMeans

# help(KMeans)

model = KMeans(n\_clusters=2)

cluster\_labels = model.fit\_predict(scaled\_X)

cluster\_labels

X['Cluster'] = cluster\_labels

**Review Model Results**

X

X.corr()['Cluster'].iloc[:-1].sort\_values()

plt.figure(figsize=(12,6), dpi=200)

X.corr()['Cluster'].iloc[:-1].sort\_values().plot(kind='bar')

**SSD – Sum of Squared Distances**

ssd = []

for k in range(2, 10):

model = KMeans(n\_clusters=k)

model.fit(scaled\_X)

ssd.append(model.inertia\_) # SSD point to cluster center

ssd

plt.plot(range(2, 10), ssd, 'o--')

pd.Series(ssd)

pd.Series(ssd).diff()

## Sweetviz Data Review

report = sweetviz.analyze(dataset, target\_feat="PE")

report.show\_html(layout="vertical")

report.show\_notebook()

## Decision Trees

This is the base decision tree model. Rarely used since there are now better tree methods.

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import sweetviz

df = pd.read\_csv('penguins\_size.csv')

df.head()

df.describe()

df['species'].unique()

df.isnull().sum()

df.info()

df = df.dropna()

df.info()

df.isnull().sum()

df.head()

df['island'].unique()

df['sex'].unique()

df[df['sex']=='.'] # find the data point with bad data “.”

df[df['species']=='Gentoo'].groupby('sex').describe().transpose()

df.at[336, 'sex'] = 'FEMALE'

df.loc[336]

# report = sweetviz.analyze(df, target\_feat="body\_mass\_g")

# report.show\_notebook()

plt.figure(figsize=(12,6), dpi=200)

sns.pairplot(df, hue='species')

sns.catplot(x='species', y='culmen\_length\_mm', data=df, kind='box', col='sex')

X = pd.get\_dummies(df.drop('species', axis=1), drop\_first=True)

y = df['species']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=101)

from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier()

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

base\_preds = model.predict(X\_test)

from sklearn.metrics import classification\_report, plot\_confusion\_matrix, ConfusionMatrixDisplay

print(classification\_report(y\_test, base\_preds))

plot\_confusion\_matrix(model, X\_test, y\_test) # old method, use next

ConfusionMatrixDisplay.from\_predictions(y\_test, base\_preds)

model.feature\_importances\_

X.columns

pd.DataFrame(index=X.columns, data=model.feature\_importances\_, columns=['Feature Importance']).sort\_values('Feature Importance')

from sklearn.tree import plot\_tree

plt.figure(figsize=(12,8), dpi=200)

plot\_tree(model, feature\_names=X.columns, filled=True);

len(X\_train)

def report\_model(model):

model\_preds = model.predict(X\_test)

print(classification\_report(y\_test, model\_preds))

print('\n')

plt.figure(figsize=(12,8), dpi=200)

plot\_tree(model, feature\_names=X.columns, filled=True);

report\_model(model)

pruned\_tree = DecisionTreeClassifier(max\_depth=3)

pruned\_tree.fit(X\_train, y\_train)

report\_model(pruned\_tree)

max\_leaf\_tree = DecisionTreeClassifier(max\_leaf\_nodes=3)

max\_leaf\_tree.fit(X\_train, y\_train)

report\_model(max\_leaf\_tree)

entropy\_tree = DecisionTreeClassifier(criterion='entropy')

entropy\_tree.fit(X\_train, y\_train)

report\_model(entropy\_tree)

## Random Forests

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import sweetviz

df = pd.read\_csv('data\_banknote\_authentication.csv')

df.head()

# report = sweetviz.analyze(df, target\_feat="Class")

# report.show\_notebook()

df.info()

df.isnull().sum()

plt.figure(dpi=250)

sns.pairplot(data=df, hue='Class')

X = df.drop('Class', axis=1)

y = df['Class']

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.15, random\_state=101)

from sklearn.model\_selection import GridSearchCV

from sklearn.ensemble import RandomForestClassifier

# help(RandomForestClassifier)

n\_estimators = [64, 100, 128, 200]

max\_features = [2, 3, 4]

bootstrap = [True, False]

oob\_score = [True, False] # Is this really needed?

param\_grid = {'n\_estimators': n\_estimators,

'max\_features': max\_features,

'bootstrap': bootstrap,

'oob\_score': oob\_score}

rfc = RandomForestClassifier()

grid = GridSearchCV(rfc, param\_grid)

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

grid.best\_params\_

rfc = RandomForestClassifier(max\_features=2, n\_estimators=200, oob\_score=True)

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

rfc.oob\_score\_

predictions = rfc.predict(X\_test)

from sklearn.metrics import plot\_confusion\_matrix, classification\_report, ConfusionMatrixDisplay, accuracy\_score

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

# plot\_confusion\_matrix(rfc, X\_test, y\_test) # old method

ConfusionMatrixDisplay.from\_predictions(y\_test, predictions)

errors = []

misclassifications = []

for n in range(1, 200):

rfc = RandomForestClassifier(n\_estimators=n, max\_features=2)

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

preds = rfc.predict(X\_test)

err = 1 - accuracy\_score(y\_test, preds)

n\_missed = np.sum(preds != y\_test)

errors.append(err)

misclassifications.append(n\_missed)

plt.plot(range(1, 200), errors)

plt.plot(range(1, 200), misclassifications)

## Bias Variance Trade Off

**Bias** – high bias results in underfitting, the model cannot generalize. This can be seen by **poor** performance on both the training and test data.

**Variance** – high variance results in overfitting, and the model over generalizes. This can be seen in **good** performance on the training data and **bad** performance on the test data. We should always plot the model complexity versus error to verify results.

## Support Vector Machines – Classification

The

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

df = pd.read\_csv('mouse\_viral\_study.csv')

df.head()

# Data review visual

plt.figure(figsize=(12,8), dpi=250)

sns.scatterplot(x='Med\_1\_mL', y='Med\_2\_mL', hue='Virus Present', data=df)

# Create a false hyperplane

x = np.linspace(0, 10, 100)

m = -1

b = 11

y = m \* x + b

plt.plot(x, y, 'black')

from sklearn.svm import SVC

# help(SVC)

y = df['Virus Present']

X = df.drop('Virus Present', axis = 1)

model = SVC(kernel='linear', C = 1000)

model.fit(X, y)

from svm\_margin\_plot import plot\_svm\_boundary

plot\_svm\_boundary(model, X, y)

model = SVC(kernel='linear', C=0.05)

model.fit(X, y)

plot\_svm\_boundary(model, X, y)

model = SVC(kernel='rbf', C=1, gamma='scale')

model.fit(X, y)

plot\_svm\_boundary(model, X, y)

model = SVC(kernel='sigmoid')

model.fit(X, y)

plot\_svm\_boundary(model, X, y)

model = SVC(kernel='poly', C=15, degree=3)

model.fit(X, y)

plot\_svm\_boundary(model, X, y)

from sklearn.model\_selection import GridSearchCV

svm = SVC()

param\_grid = {'C':[0.01, 0.1, 1], 'kernel':['linear', 'rbf']}

grid = GridSearchCV(svm, param\_grid)

grid.fit(X,y)

grid.best\_params\_

## Support Vector Machines – Regression

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

df = pd.read\_csv('cement\_slump.csv')

df.head()

plt.figure(figsize=(8, 8), dpi=200)

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

df.columns

X = df.drop('Compressive Strength (28-day)(Mpa)', axis=1)

y = df['Compressive Strength (28-day)(Mpa)']

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=101)

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaled\_X\_train = scaler.fit\_transform(X\_train)

scaled\_X\_test = scaler.transform(X\_test)

from sklearn.svm import SVR, LinearSVR

# help(SVR)

base\_model = SVR()

base\_model.fit(scaled\_X\_train, y\_train)

base\_preds = base\_model.predict(scaled\_X\_test)

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

mean\_absolute\_error(y\_test, base\_preds)

np.sqrt(mean\_squared\_error(y\_test, base\_preds))

y\_test.mean()

param\_grid = {'C':[0.001,0.01, 0.1, 0.5, 1], 'kernel':['linear', 'rbf', 'poly'],

'gamma':['scale', 'auto'], 'degree':[2, 3, 4], 'epsilon':[0, 0.01, 0.1, 0.5, 1, 2]}

from sklearn.model\_selection import GridSearchCV

svr = SVR()

grid = GridSearchCV(svr, param\_grid)

grid.fit(scaled\_X\_train, y\_train)

grid.best\_params\_

grid\_preds = grid.predict(scaled\_X\_test)

mean\_absolute\_error(y\_test, grid\_preds)

np.sqrt(mean\_squared\_error(y\_test, grid\_preds))

## ADA Boost Classification

# ADA Boost

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import sweetviz

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

df.head()

# ### Data Set Information:

# This data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family (pp. 500-525). Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. This latter class was combined with the poisonous one. The Guide clearly states that there is no simple rule for determining the edibility of a mushroom; no rule like ``leaflets three, let it be'' for Poisonous Oak and Ivy.

# ### Attribute Information:

# 1. cap-shape: bell=b,conical=c,convex=x,flat=f, knobbed=k,sunken=s

# 2. cap-surface: fibrous=f,grooves=g,scaly=y,smooth=s

# 3. cap-color: brown=n,buff=b,cinnamon=c,gray=g,green=r, pink=p,purple=u,red=e,white=w,yellow=y

# 4. bruises?: bruises=t,no=f

# 5. odor: almond=a,anise=l,creosote=c,fishy=y,foul=f, musty=m,none=n,pungent=p,spicy=s

# 6. gill-attachment: attached=a,descending=d,free=f,notched=n

# 7. gill-spacing: close=c,crowded=w,distant=d

# 8. gill-size: broad=b,narrow=n

# 9. gill-color: black=k,brown=n,buff=b,chocolate=h,gray=g, green=r,orange=o,pink=p,purple=u,red=e, white=w,yellow=y

# 10. stalk-shape: enlarging=e,tapering=t

# 11. stalk-root: bulbous=b,club=c,cup=u,equal=e, rhizomorphs=z,rooted=r,missing=?

# 12. stalk-surface-above-ring: fibrous=f,scaly=y,silky=k,smooth=s

# 13. stalk-surface-below-ring: fibrous=f,scaly=y,silky=k,smooth=s

# 14. stalk-color-above-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o, pink=p,red=e,white=w,yellow=y

# 15. stalk-color-below-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o, pink=p,red=e,white=w,yellow=y

# 16. veil-type: partial=p,universal=u

# 17. veil-color: brown=n,orange=o,white=w,yellow=y

# 18. ring-number: none=n,one=o,two=t

# 19. ring-type: cobwebby=c,evanescent=e,flaring=f,large=l, none=n,pendant=p,sheathing=s,zone=z

# 20. spore-print-color: black=k,brown=n,buff=b,chocolate=h,green=r, orange=o,purple=u,white=w,yellow=y

# 21. population: abundant=a,clustered=c,numerous=n, scattered=s,several=v,solitary=y

# 22. habitat: grasses=g,leaves=l,meadows=m,paths=p, urban=u,waste=w,woods=d

sns.countplot(data=df, x='class')

df.describe()

df.describe().transpose().reset\_index().sort\_values('unique')

feat\_uni = df.describe().transpose().reset\_index().sort\_values('unique')

plt.figure(figsize=(14,6), dpi=250)

sns.barplot(data=feat\_uni, x='index', y='unique')

plt.xticks(rotation=90);

X = df.drop('class', axis=1)

# X.isnull().sum()

X = pd.get\_dummies(X, drop\_first=True)

y = df['class']

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.15, random\_state=101)

from sklearn.ensemble import AdaBoostClassifier

model = AdaBoostClassifier(n\_estimators=1)

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

from sklearn.metrics import classification\_report, plot\_confusion\_matrix, accuracy\_score

preditions = model.predict(X\_test)

preditions

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

model.feature\_importances\_

model.feature\_importances\_.argmax()

X.columns[22]

sns.countplot(data=df, x='odor', hue='class')

len(X.columns)

error\_rates = []

for n in range(1, 96):

model = AdaBoostClassifier(n\_estimators=n)

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

preds = model.predict(X\_test)

err = 1 - accuracy\_score(y\_test, preds)

error\_rates.append(err)

plt.plot(range(1, 96), error\_rates)

model

imp\_feats = pd.DataFrame(index=X.columns, data=model.feature\_importances\_, columns=['Importance'])

imp\_feats

plt.figure(figsize=(14,6), dpi=250)

sns.barplot(data=imp\_feats.sort\_values("Importance"), x=imp\_feats.index, y='Importance')

plt.xticks(rotation=90);

# My work

cdm\_model = AdaBoostClassifier(n\_estimators=18)

cdm\_model.fit(X\_train, y\_train)

cdm\_preditions = cdm\_model.predict(X\_test)

cdm\_preditions

print(classification\_report(y\_test, cdm\_preditions))

cdm\_feats = pd.DataFrame(index=X.columns, data=cdm\_model.feature\_importances\_, columns=['Importance'])

cdm\_feats = feats[feats['Importance']>0]

cdm\_feats

plt.figure(figsize=(14,6), dpi=250)

sns.barplot(data=cdm\_feats.sort\_values("Importance"), x=cdm\_feats.index, y='Importance')

plt.xticks(rotation=90);

## Gradient Boost Classification

# Gradient Boosting Classification

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import sweetviz

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

df.head()

X = df.drop('class', axis=1)

X = pd.get\_dummies(X, drop\_first=True)

y = df['class']

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.15, random\_state=101)

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.model\_selection import GridSearchCV

param\_grid = {'n\_estimators':[50, 100], 'learning\_rate':[0.1, 0.05, 0.2], 'max\_depth':[3, 4, 5]}

gb\_model = GradientBoostingClassifier()

grid = GridSearchCV(gb\_model, param\_grid)

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

from sklearn.metrics import classification\_report, plot\_confusion\_matrix, accuracy\_score

predicitions = grid.predict(X\_test)

predicitions

grid.best\_estimator\_

grid.best\_params\_

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

# grid.best\_estimator\_.feature\_importances\_

feat\_import = grid.best\_estimator\_.feature\_importances\_

imp\_feat = pd.DataFrame(index=X.columns, data=feat\_import, columns=['Importance'])

imp\_feat = imp\_feat[imp\_feat['Importance']>0.0005]

imp\_feat = imp\_feat.sort\_values("Importance")

plt.figure(figsize=(14,8), dpi=250)

sns.barplot(data=imp\_feat, x=imp\_feat.index, y='Importance')

plt.xticks(rotation=90);

### Natural Language Processing

NLP-1

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

df = pd.read\_csv('airline\_tweets.csv')

df.head()

sns.countplot(data=df, x='airline\_sentiment')

sns.countplot(data=df, x='negativereason')

plt.xticks(rotation=90);

sns.countplot(data=df, x='airline', hue='airline\_sentiment')

data = df[['airline\_sentiment', 'text']]

data

X = data['text']

y = data['airline\_sentiment']

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=101)

from sklearn.feature\_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(stop\_words='english')

tfidf.fit(X\_train)

X\_train\_tfidf = tfidf.transform(X\_train)

X\_test\_tfidf = tfidf.transform(X\_test)

X\_train\_tfidf

from sklearn.naive\_bayes import MultinomialNB

nb = MultinomialNB()

nb.fit(X\_train\_tfidf, y\_train)

from sklearn.linear\_model import LogisticRegression

log\_model = LogisticRegression(max\_iter=1000)

log\_model.fit(X\_train\_tfidf, y\_train)

from sklearn.svm import SVC, LinearSVC

rbf\_svc = SVC()

rbf\_svc.fit(X\_train\_tfidf, y\_train)

linearsvc = LinearSVC()

linearsvc.fit(X\_train\_tfidf, y\_train)

from sklearn.metrics import plot\_confusion\_matrix, classification\_report

def report(model):

preds = model.predict(X\_test\_tfidf)

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

plot\_confusion\_matrix(model, X\_test\_tfidf, y\_test)

report(nb)

report(rbf\_svc)

report(log\_model)

report(linearsvc)

from sklearn.pipeline import Pipeline

pipe = Pipeline([('tfidf', TfidfVectorizer()),

('svc', LinearSVC())])

pipe.fit(X, y)

pipe.predict([('good flight')])

pipe.predict([('bad flight')])

pipe.predict([('ok flight')])

# # DBSCAN vs. K Means Clustering

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import sweetviz

blobs = pd.read\_csv('cluster\_blobs.csv')

blobs.head()

sns.scatterplot(data=blobs, x='X1', y='X2')

moons = pd.read\_csv('cluster\_moons.csv')

moons.head()

sns.scatterplot(data=moons, x='X1', y='X2')

circles = pd.read\_csv('cluster\_circles.csv')

circles.head()

sns.scatterplot(data=circles, x='X1', y='X2')

def display\_categories(model, data):

labels = model.fit\_predict(data)

sns.scatterplot(data=data, x='X1', y='X2', hue=labels, palette='Set1')

from sklearn.cluster import KMeans

model = KMeans(n\_clusters=3)

display\_categories(model, blobs)

moon\_model = KMeans(n\_clusters=2)

display\_categories(moon\_model, moons)

circle\_model = KMeans(n\_clusters=2)

display\_categories(circle\_model, circles)

from sklearn.cluster import DBSCAN

model = DBSCAN()

display\_categories(model, blobs)

model = DBSCAN(eps=0.15)

display\_categories(model, moons)

model = DBSCAN(eps=0.15)

display\_categories(model, circles)

## Artificial Neural Networks

"""Artificial Neural Network - CCPP

"""

from google.colab import drive

drive.mount('/content/drive')

**"""### Importing the libraries"""**

import numpy as np

import pandas as pd

import tensorflow as tf

tf.\_\_version\_\_

"""## Part 1 - Data Preprocessing

**### Importing the dataset**

"""

dataset = pd.read\_excel("/content/drive/MyDrive/Data Science/Folds5x2\_pp.xlsx")

# /content/drive/MyDrive/Data Science/Folds5x2\_pp.xlsx

dataset.head()

X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, -1].values

"""### Splitting the dataset into the Training set and Test set"""

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)

"""## Part 2 - Building the ANN

### Initializing the ANN

"""

ann = tf.keras.models.Sequential()

"""### Adding the input layer and the first hidden layer"""

ann.add(tf.keras.layers.Dense(units=6, activation='relu'))

"""### Adding the second hidden layer"""

ann.add(tf.keras.layers.Dense(units=6, activation='relu'))

"""### Adding the output layer"""

ann.add(tf.keras.layers.Dense(units=1))

"""## Part 3 - Training the ANN

### Compiling the ANN

"""

ann.compile(optimizer='adam', loss='mean\_squared\_error')

"""### Training the ANN model on the Training set"""

ann.fit(X\_train, y\_train, batch\_size=32, epochs=100)

"""### Predicting the results of the Test set"""

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

np.set\_printoptions(precision=2)

print(np.concatenate((y\_pred.reshape(len(y\_pred),1), y\_test.reshape(len(y\_pred),1)), 1))

# Algorithms

This section includes the algorithms that I have used in my data science studies. I am currently not an expert on these procedures but I am trying to learn them. These algorithms were typed by me into Notebooks so the actual source code should be available.

## Feature Scaling

These two feature scaling techniques are taken from the Udemy videos of Jose Portilla – Python for Machine Learning and Data Science.

### Standardization

The standardization approach places the observation under a normal distribution. From -3 to +3.

X - µ (Subtract the mean from the observation)

X changed = --------------------------

Standard Deviation

### Normalization

The normalization approach places all observations in order in range of from 0 to 1.

X – X min

X changed = --------------------------

X max – X min

### Feature Scaling Process

1. Perform train test split
2. Fit to training feature data
3. Transform training feature data
4. Transform test feature data

## Outliers

### IQR – Inter Quartile Range

Start with a Panadas Series = ser

Describe the variable

IQR = 75%tile – 25%tile

Lower\_limit = 25%tile – 1.5 \* IQR

Lower\_limit

Upper\_limit = 75%tile + 1.5 \* IQR

Upper\_limit

Ser = [Ser > Lower\_limit]

Ser = [Ser < Upper\_limit]

q75, q25 = np.percentile(sample, [75,25])

iqr = q75 – q25

iqr

lower\_limit = q25 – 1.5 \* iqr

### **Feature Selection**

1. Variance Threshold
   1. Import pandas as pd
   2. From sklearn.feature\_selection import VarianceThreshold
   3. # X = target dataframe
   4. Selector = VarianceThreshold(threshold=1)
   5. Selector.fit(X)
   6. Transformed = selector.transform(X)
   7. Print(‘Shape of the data – ()’.format(transformed.shape))
2. Recursive Feature Selection - RFE()
   1. Import pandas as pd
   2. From sklearn.feature\_selection import RFE
   3. From sklearn.linear\_model import LogisticRegression
   4. Selector = RFE(estimator=LogisticRegression(), n\_features\_to\_select = 5)
   5. Selector.fit(X,y)
   6. Variable\_chosen = selector.support\_
   7. Print(‘Selector chosen variables – ()’.format(variables\_chosen)
   8. Variable\_ranks = selector.ranking\_
   9. Print(“Selectors variable rank – {}”.format(variable\_ranks)