**Functions**

1. Reading Dataset

import pandas as pd

import matplotlib.pyplot as plt

admissions = pd.read\_csv("admissions.csv")

plt.scatter(admissions['gpa'], admissions['admit'])

plt.show()

1. Logit Function

# Logit Function

def logit(x):

# np.exp(x) raises x to the exponential power, ie e^x. e ~= 2.71828

return np.exp(x) / (1 + np.exp(x))

# Generate 50 real values, evenly spaced, between -6 and 6.

x = np.linspace(-6,6,50, dtype=float)

# Transform each number in t using the logit function.

y = logit(x)

# Plot the resulting data.

plt.plot(x, y)

plt.ylabel("Probability")

plt.show()

1. Regressions

from sklearn.linear\_model import LinearRegression

linear\_model = LinearRegression()

linear\_model.fit(admissions[["gpa"]], admissions["admit"])

from sklearn.linear\_model import LogisticRegression

logistic\_model = LogisticRegression()

logistic\_model.fit(admissions[["gpa"]], admissions["admit"])

1. Logistic regression

logistic\_model = LogisticRegression()

logistic\_model.fit(admissions[["gpa"]], admissions["admit"])

pred\_probs = logistic\_model.predict\_proba(admissions[["gpa"]])

plt.scatter(admissions["gpa"], pred\_probs[:,1])

logistic\_model = LogisticRegression()

logistic\_model.fit(admissions[["gpa"]], admissions["admit"])

fitted\_labels = logistic\_model.predict(admissions[["gpa"]])

plt.scatter(admissions["gpa"], fitted\_labels)

1. Binary classifiers

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.linear\_model import LogisticRegression

admissions = pd.read\_csv("admissions.csv")

model = LogisticRegression()

model.fit(admissions[["gpa"]], admissions["admit"])

admissions = pd.read\_csv("admissions.csv")

model = LogisticRegression()

model.fit(admissions[["gpa"]], admissions["admit"])

labels = model.predict(admissions[["gpa"]])

admissions["predicted\_label"] = labels

print(admissions["predicted\_label"].value\_counts())

print(admissions.head())

## Logistics prediction

admissions["actual\_label"] = admissions["admit"]

matches = admissions["predicted\_label"] == admissions["actual\_label"]

correct\_predictions = admissions[matches]

print(correct\_predictions.head())

accuracy = len(correct\_predictions) / len(admissions)

print(accuracy)

1. TP and FP

true\_positive\_filter = (admissions["predicted\_label"] == 1) & (admissions["actual\_label"] == 1)

true\_positives = len(admissions[true\_positive\_filter])

true\_negative\_filter = (admissions["predicted\_label"] == 0) & (admissions["actual\_label"] == 0)

true\_negatives = len(admissions[true\_negative\_filter])

print(true\_positives)

print(true\_negatives)

sensitivity = true\_positives / (true\_positives + false\_negatives)

print(sensitivity)

true\_negatives = len(admissions[true\_negative\_filter])

false\_positive\_filter = (admissions["predicted\_label"] == 1) & (admissions["actual\_label"] == 0)

false\_positives = len(admissions[false\_positive\_filter])

specificity = (true\_negatives) / (false\_positives + true\_negatives)

print(specificity)

1. Multi-class classification

import pandas as pd

cars = pd.read\_csv("auto.csv")

print(cars.head())

unique\_regions = cars["origin"].unique()

print(unique\_regions)

## Dummy variables for categorical variables

dummy\_cylinders = pd.get\_dummies(cars["cylinders"], prefix="cyl")

cars = pd.concat([cars, dummy\_cylinders], axis=1)

dummy\_years = pd.get\_dummies(cars["year"], prefix="year")

cars = pd.concat([cars, dummy\_years], axis=1)

cars = cars.drop("year", axis=1)

cars = cars.drop("cylinders", axis=1)

print(cars.head())

## reshuffling rows

shuffled\_rows = np.random.permutation(cars.index)

shuffled\_cars = cars.iloc[shuffled\_rows]

highest\_train\_row = int(cars.shape[0] \* .70)

train = shuffled\_cars.iloc[0:highest\_train\_row]

test = shuffled\_cars.iloc[highest\_train\_row:]

##Multi-class classification

from sklearn.linear\_model import LogisticRegression

unique\_origins = cars["origin"].unique()

unique\_origins.sort()

models = {}

features = [c for c in train.columns if c.startswith("cyl") or c.startswith("year")]

for origin in unique\_origins:

model = LogisticRegression()

X\_train = train[features]

y\_train = train["origin"] == origin

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

models[origin] = model

##prediction for every model

testing\_probs = pd.DataFrame(columns=unique\_origins)

for origin in unique\_origins:

# Select testing features.

X\_test = test[features]

# Compute probability of observation being in the origin.

testing\_probs[origin] = models[origin].predict\_proba(X\_test)[:,1]

## getting predictions

predicted\_origins = testing\_probs.idxmax(axis=1)

print(predicted\_origins)

1. Intermediate Linear Regression

import pandas

import matplotlib.pyplot as plt

pisa = pandas.DataFrame({"year": range(1975, 1988),

"lean": [2.9642, 2.9644, 2.9656, 2.9667, 2.9673, 2.9688, 2.9696,

2.9698, 2.9713, 2.9717, 2.9725, 2.9742, 2.9757]})

print(pisa)

plt.scatter(pisa["year"], pisa["lean"])

import statsmodels.api as sm

y = pisa.lean # target

X = pisa.year # features

X = sm.add\_constant(X) # add a column of 1's as the constant term

# OLS -- Ordinary Least Squares Fit

linear = sm.OLS(y, X)

# fit model

linearfit = linear.fit()

print(linearfit.summary())

## Residuals

# Our predicted values of y

yhat = linearfit.predict(X)

print(yhat)

residuals = yhat – y

## Linear fit equation

import numpy as np

# sum the (predicted - observed) squared

SSE = np.sum((y.values-yhat)\*\*2)

# Average y

ybar = np.mean(y.values)

# sum the (mean - predicted) squared

RSS = np.sum((ybar-yhat)\*\*2)

# sum the (mean - observed) squared

TSS = np.sum((ybar-y.values)\*\*2)

SSE = np.sum((y.values-yhat)\*\*2)

ybar = np.mean(y.values)

RSS = np.sum((ybar-yhat)\*\*2)

TSS = np.sum((y.values-ybar)\*\*2)

R2 = RSS/TSS

print(R2)

# Print the models summary

print(linearfit.summary())

#The models parameters

print("\n",linearfit.params)

delta = linearfit.params["year"] \* 15

# Enter your code here.

# Compute SSE

SSE = np.sum((y.values - yhat)\*\*2)

# Compute variance in X

xvar = np.sum((pisa.year - pisa.year.mean())\*\*2)

# Compute variance in b1

s2b1 = SSE / ((y.shape[0] - 2) \* xvar)

##T-test

from scipy.stats import t

# 100 values between -3 and 3

x = np.linspace(-3,3,100)

# Compute the pdf with 3 degrees of freedom

print(t.pdf(x=x, df=3))

# Pdf with 3 degrees of freedom

tdist3 = t.pdf(x=x, df=3)

# Pdf with 30 degrees of freedom

tdist30 = t.pdf(x=x, df=30)

# Plot pdfs

plt.plot(x, tdist3)

plt.plot(x, tdist30)

# The variable s2b1 is in memory. The variance of beta\_1

tstat = linearfit.params["year"] / np.sqrt(s2b1)

# At the 95% confidence interval for a two-sided t-test we must use a p-value of 0.975

pval = 0.975

# The degrees of freedom

df = pisa.shape[0] - 2

# The probability to test against

p = t.cdf(tstat, df=df)

beta1\_test = p > pval