**Imported libraries for machine learning algorithms**

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.linear\_model import LogisticRegression import sklearn.metrics as metrics

from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import MinMaxScaler import cx\_Oracle

import sqlalchemy

from sqlalchemy.exc import SQLAlchemyError

from sklearn.metrics import confusion\_matrix , classification\_report import seaborn as sns

import numpy as np ax= plt.subplot()

from matplotlib import pyplot as plt

from sklearn.metrics import roc\_auc\_score from sklearn.metrics import roc\_curve import tensorflow as tf

from tensorflow import keras

from sklearn.neighbors import KNeighborsClassifier from keras.models import Sequential

from keras.layers import Dense, Dropout

from keras.wrappers.scikit\_learn import KerasClassifier from sklearn.model\_selection import GridSearchCV

from sklearn.model\_selection import RepeatedStratifiedKFold from sklearn.naive\_bayes import GaussianNB

from sklearn.preprocessing import PowerTransformer import statsmodels.api as sm

**Sql query to fetch data from oracle database**

try: engine = sqlalchemy.create\_engine ("oracle+cx\_oracle://hxxxxxt:hxxxxmat@192.168.20.XXX/?service\_name=orcl", arraysize=1000)

ispdb= """ select \* from intpredecit\_xxxx\_final """; df = pd.read\_sql(ispdb, engine)

print(df)

except SQLAlchemyError as e:

print(e)

**Code to fetch data and its type**

df.dtypes

**Code to provide graphical representation count of dependent feature** get\_ipython().run\_line\_magic(’matplotlib’, ’inline’) df[’internet\_user\_name’].value\_counts().plot(kind=’bar’, figsize=(8, 6)) plt.ylabel("Count", labelpad=14)

plt.xlabel("Non Internet user / Internet user", labelpad=14) plt.title("Count of Internet user variable per category", y=1.02);

**Code for setting dependent and independent feature and dropping irrelevant column**

df=df.drop(’customer\_id’,axis=’columns’)

df["internet\_user\_name"] = df["internet\_user\_name"].astype(str).astype(int) X = df.drop(’internet\_user\_name’,axis=’columns’)

y = df[’internet\_user\_name’]

**Code for MinMax Feature Scaling**

cols\_to\_scale = [’tenure’,’rch\_freq’,’avg\_rch’,’pkg\_change\_freq’ ,’ppv\_freq’] from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

df[cols\_to\_scale] = scaler.fit\_transform(df[cols\_to\_scale])

**Code to Split data set in ratio of 8:2** X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=0.2,random\_state=5)

**Code for Artificial Neural Network Classifier without Hyperparam- eter Tuning**

model = keras.Sequential([

keras.layers.Dense(94, input\_shape=(94,), activation=’relu’), keras.layers.Dense(2, activation=’relu’),

keras.layers.Dense(1, activation=’sigmoid’)

])

opt = keras.optimizers.Adam(learning\_rate=0.0001)

model.compile(optimizer=’adam’, loss=’binary\_crossentropy’, metrics=[’accuracy’]) model.fit(X\_train, y\_train, epochs=10) model.evaluate(X\_test, y\_test)

yp = model.predict(X\_test)

**Code for Logistic Regression Classifier without Hyperparameter Tun- ing**

classifier= LogisticRegression(random\_state=0,max\_iter=500) classifier.fit(X\_train, y\_train)

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

print(’Training set accuracy: ’, metrics.accuracy\_score(y\_train, y\_hat)) print(’Test set accuracy: ’,metrics.accuracy\_score(y\_test, y\_pred))

**Code For K Nearest Neighbour Classifier without Hyperparameter Tuning**

classifier= KNeighborsClassifier(n\_neighbors=10) classifier.fit(X\_train, y\_train)

y\_pred = classifier.predict(X\_test) print ("Predicted Value test:", y\_pred)

y\_predTrain = classifier.predict(X\_train) print ("Predicted Value train:", y\_predTrain)

**Code for Naive Bayes Classifier without Hyperparameter Tuning**

classifier = GaussianNB() classifier.fit(X\_train, y\_train) y\_pred = classifier.predict(X\_test) print ("Predicted Value:", y\_pred)

print(’Training set accuracy: ’, metrics.accuracy\_score(y\_train, y\_predTrain)) print(’Test set accuracy: ’,metrics.accuracy\_score(y\_test, y\_pred))

**Code for Artificial Neural Network Classifier with Cross Validation and Hyperparameter Tuning**

def DL\_Model(activation= ’linear’, neurons= 94, optimizer=’Adam’): model = Sequential()

model.add(Dense(neurons, input\_dim= 94, activation= activation)) model.add(Dense(neurons, activation= activation)) model.add(Dropout(0.3))

model.add(Dense(1, activation=’sigmoid’)) model.compile(loss=’binary\_crossentropy’, optimizer= optimizer, metrics=[’accuracy’])

return model

**Defining grid parameters and fitting** activation = [’softmax’, ’relu’,’sigmoid’] neurons = [5, 10,50,94,100]

optimizer = [’Adam’, ’Adamax’]

param\_grid = dict(activation = activation, neurons = neurons, optimizer = op- timizer)

clf = KerasClassifier(build\_fn= DL\_Model, epochs=15, batch\_size=1024, ver- bose= 2)

model = GridSearchCV(estimator= clf, param\_grid=param\_grid, n\_jobs=-1) model.fit(X\_train,y\_train)

y\_test= model.predict(X\_test) y\_pred= model.predict(X\_train)

**Code for Logistic Regression Classifier with Cross Validation and Hyperparameter Tuning**

model = LogisticRegression(max\_iter=1000) solvers = [’newton-cg’, ’lbfgs’, ’liblinear’] penalty = [’l2’,’l1’]

c\_values = [100, 10, 1.0, 0.1, 0.01]

define grid search

grid = dict(solver=solvers,penalty=penalty,C=c\_values)

cv = RepeatedStratifiedKFold(n\_splits=10, n\_repeats=3, random\_state=1) grid\_search = GridSearchCV(estimator=model, param\_grid=grid, n\_jobs=-1,

cv=cv, scoring=’accuracy’,error\_score=0)

grid\_result = grid\_search.fit(X\_train, y\_train) summarize results

means = grid\_result.cv\_results\_[’mean\_test\_score’] stds = grid\_result.cv\_results\_[’std\_test\_score’]

params = grid\_result.cv\_results\_[’params’]

for mean, stdev, param in zip(means, stds, params):

print("%f (%f) with: %r" % (mean, stdev, param)) dt\_best = grid\_search.best\_estimator\_ dt\_best

y\_pred=dt\_best.predict(X\_test) y\_hat = dt\_best.predict(X\_train)

print(’Training set accuracy: ’, metrics.accuracy\_score(y\_train, y\_hat)) print(’Test set accuracy: ’,metrics.accuracy\_score(y\_test, y\_pred))

**Code for K Nearest Neighbour classifier with Cross Validation and Hyperparameter Tuning**

grid\_params = ’n\_neighbors’ : [5,7,9,11,13,15,40,60], ’weights’ : [’uniform’,’distance’],

’metric’ : [’minkowski’,’euclidean’,’manhattan’]

gs = GridSearchCV(KNeighborsClassifier(), grid\_params, verbose = 1, cv=5, n\_jobs = -1)

g\_res = gs.fit(X\_train, y\_train) g\_res.best\_score\_ g\_res.best\_params\_

result:’metric’: ’manhattan’, ’n\_neighbors’: 40, ’weights’: ’uniform’

knn = KNeighborsClassifier(n\_neighbors = 40, weights = ’uniform’,metric = ’manhattan’)

knn.fit(X\_train, y\_train) y\_hat = knn.predict(X\_train) y\_knn = knn.predict(X\_test)

print(’Training set accuracy: ’, metrics.accuracy\_score(y\_train, y\_hat))

print(’Test set accuracy: ’,metrics.accuracy\_score(y\_test, y\_knn))

**Code for Naive Bayes classifier with Cross Validation and Hyperpa- rameter Tuning**

cv\_method = RepeatedStratifiedKFold(n\_splits=5, n\_repeats=3,

random\_state=999)

params\_NB = ’var\_smoothing’: np.logspace(0,-9, num=100) classifier = GaussianNB()

gs\_NB = GridSearchCV(estimator=classifier, param\_grid=params\_NB,

cv=cv\_method, verbose=1, scoring=’accuracy’)

gs\_NB.fit(X\_train, y\_train) gs\_NB.best\_params\_

Result: ’var\_smoothing’: 0.12328467394420659

classifier = GaussianNB(var\_smoothing=0.12328467394420659) classifier.fit(X\_train, y\_train)

y\_hat = classifier.predict(X\_train) y\_knn = classifier.predict(X\_test)

print(’Training set accuracy: ’, metrics.accuracy\_score(y\_train, y\_hat)) print(’Test set accuracy: ’,metrics.accuracy\_score(y\_test, y\_knn))

**Code for Backward Feature Elimination techniques** Adding constant column of ones, mandatory for sm.OLS model X\_1 = sm.add\_constant(X)

Fitting sm.OLS model

model = sm.OLS(y,X\_1).fit() P-value and feature plot bar p\_values0=model.pvalues

p\_values0.sort\_values(ascending = True , inplace = True)

plt.rcParams["figure.figsize"] = (20,3) p\_values0.plot.bar()

model.pvalues model.summary()

**Backward Elimination**

cols = list(X.columns) pmax = 1

while (len(cols)>0):

p= []

X\_1 = X[cols]

X\_1 = sm.add\_constant(X\_1) model = sm.OLS(y,X\_1).fit()

p = pd.Series(model.pvalues.values[1:],index = cols) pmax = max(p)

feature\_with\_p\_max = p.idxmax() if(pmax>0.05): cols.remove(feature\_with\_p\_max) else:

break

selected\_features\_BE = cols print(selected\_features\_BE)

**plot bar for selected features** p\_values1=model.pvalues p\_values1.sort\_values(ascending = True , inplace = True)

plt.rcParams["figure.figsize"] = (20,3) p\_values1.plot.bar()

**Code for classification Report and confusion matrix**

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

group\_names = [’True Neg’,’False Pos’,’False Neg’,’True Pos’]

group\_counts = ["0:0.0f".format(value) for value in cf\_matrix.flatten()] group\_percentages = ["0:.2%".format(value) for value in

cf\_matrix.flatten()/np.sum(cf\_matrix)] labels = [f"v1v2v3" for v1, v2, v3 in

zip(group\_names,group\_counts,group\_percentages)] labels = np.asarray(labels).reshape(2,2)

sns.heatmap(cf\_matrix, annot=labels, fmt=”, cmap=’Blues’,ax=ax) ax.set\_xlabel(’Predicted labels’);ax.set\_ylabel(’True labels’);

**Code for calculating ROC Score and presenting ROC curve**

ns\_auc = roc\_auc\_score(y\_test, y\_pred) lr\_auc = roc\_auc\_score(testy, lr\_probs) print(’SVM ROC AUC=%.3f’ % (ns\_auc)) def plot\_roc\_curve(fpr, tpr):

plt.plot(fpr, tpr, color=’orange’, label=’ROC’) plt.plot([0, 1], [0, 1], color=’darkblue’, linestyle=’–’) plt.xlabel(’False Positive Rate’)

plt.ylabel(’True Positive Rate’)

plt.title(’Logistic Regression ROC curve Analysis’) plt.legend()

plt.show()

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred) plot\_roc\_curve(fpr, tpr)