**DA Mini-Project TITLE:** Classification Algorithms on Churn Modelling

**PROBLEM STATEMENT:** Consider a labeled dataset belonging to an application domain. Apply suitable data preprocessing steps such as handling of null values, data reduction, discretization. For prediction of class labels of given data instances, build classifier models using different techniques (minimum 3), analyze the confusion matrix and compare these models. Also apply cross validation while preparing the training and testing datasets. For Example: Health Care Domain for predicting disease

# OBJECTIVE:

* To learn different Classification algorithms
* To implement Classification models
* Analyze and compare various techniques

**OUTCOME:** We will be able to –

* Learn different classification models in ML
* Implementation of classification of models
* Analyze and compare these models

# REQUIREMENTS:

* 2 GB RAM
* 500 GB HDD
* sklearn library

# THEORY:

1. **Logistic Regression:**
   * Logistic regression is one of the most simple machine learning models. It is a classification algorithm used to assign observations to discrete set of classes.
   * It transforms is output using logistic Sigmoid function
   * They are very easy to understand, interpretable and give pretty good results.
   * Types of logistic regression are:
     + Binary (e.g. Tumor malignant (benign))
     + Multilinear functions fails Class (e.g. cats, dogs, sheeps)
   * Logistic regression is very much interpretable considering the business needs & explanation regarding how the model works considering different independent variables used in the model.

Examples:

1. Emails-> spam/ not spam
2. Outline transactions-> fraud /not fraud
3. Tumor-> Malignant /Benign

# Naive Bayes:

* + It is the most straightforward and fast classification algorithm which is suitable for which is suitable for a large chunk of data
  + It is a statistical classification technique based on Bayes Theorem. It is one of the simplest supervised learning algorithm.
  + Naive Bayes classifier assumes that the effect of particular feature in class is independent of other features.
  + For example, a loan applicant is desirable or not depending on his/her income, previous loan & transaction history, age & location
  + Even if these features are interdependent these considered independently.
  + This assumption simplifies computation & that's why is considered naive.
  + This assumption is also called class conditional independence.
  + Applications
    - Spam filtering
    - Text classification
    - Sentiment Analysis
    - Recommender Systems

# Decision Tree:

* + Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.
  + In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
  + The decisions or the test are performed on the basis of features of the given dataset.
  + It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.
  + It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.
  + In order to build a tree, we use the CART algorithm, which stands for Classification and Regression Tree algorithm.
  + A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees
  + Applications:
    - Selecting the best flight to travel to a destination.
    - Decision-making process based on different circumstantial situations.
    - Churn Analysis.
    - Sentiment Analysis.

# Classification Workflow:

Training

Set

Data

Test Set

Model Evaluation

Model Development

**TEST CASES:**

Performance Measures

1. Accuracy
2. Precision
3. Recall

Input Data ->Churn\_Modelling.csv

1. Logistic Regression

Training Result

* + Accuracy score: 0.789
  + Confusion matrix:

[[1553 42]

[370 35]]

3 fold Cross validation accuracy and std of the default models for the train data:

* LogisticRegression: 0.796 (0.006) #0.006 is the standard deviation
* LogisticRegression cross validation accuracy after tuning : 0.810

>Model tuning done in 27s

Validation accuracies of the tuned models for the train data:

* LogisticRegression : 0.814

1. GaussianNB

Training Result

* + Accuracy score: 0.784
  + Confusion matrix:

[[1534 61]

[ 370 35]]

3 fold Cross validation accuracy and std of the default models for the train data:

* GaussianNB: 0.796 (0.006)
* GaussianNB cross validation accuracy after tuning : 0.796

>Model tuning done in 2s

Validation accuracies of the tuned models for the train data:

* GaussianNB : 0.797

1. Decision Tree

Training Result

* + Accuracy score: 0.784
  + Confusion matrix:

[[1534 61]

[ 370 35]]

3 fold Cross validation accuracy and std of the default models for the train data:

* DecisionTree: 0.663 (0.006)
* DecisionTree cross validation accuracy after tuning : 0.855

>Model tuning done in 4s

Validation accuracies of the tuned models for the train data:

# CONCLUSION:

* DecisionTree : 0.860

Thus, we have successfully studied and implemented different classification algorithms and analyzed and compared them.

# CODE:

# data analysis libraries:

import numpy as np import pandas as pd import re

# data visualization libraries:

import matplotlib.pyplot as plt import seaborn as sns

# to ignore warnings:

import sys

if not sys.warnoptions: import os, warnings

warnings.simplefilter("ignore") os.environ["PYTHONWARNINGS"] = "ignore"

# to display all columns: pd.set\_option('display.max\_columns', None)

#timer import time

from contextlib import contextmanager

# Importing modelling libraries

from sklearn.model\_selection import train\_test\_split,GridSearchCV,cross\_val\_score,KFold from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score,confusion\_matrix from sklearn.naive\_bayes import GaussianNB

from sklearn.tree import DecisionTreeClassifier

from sklearn.linear\_model import LogisticRegression from sklearn.ensemble import VotingClassifier pd.options.display.float\_format = "{:,.2f}".format

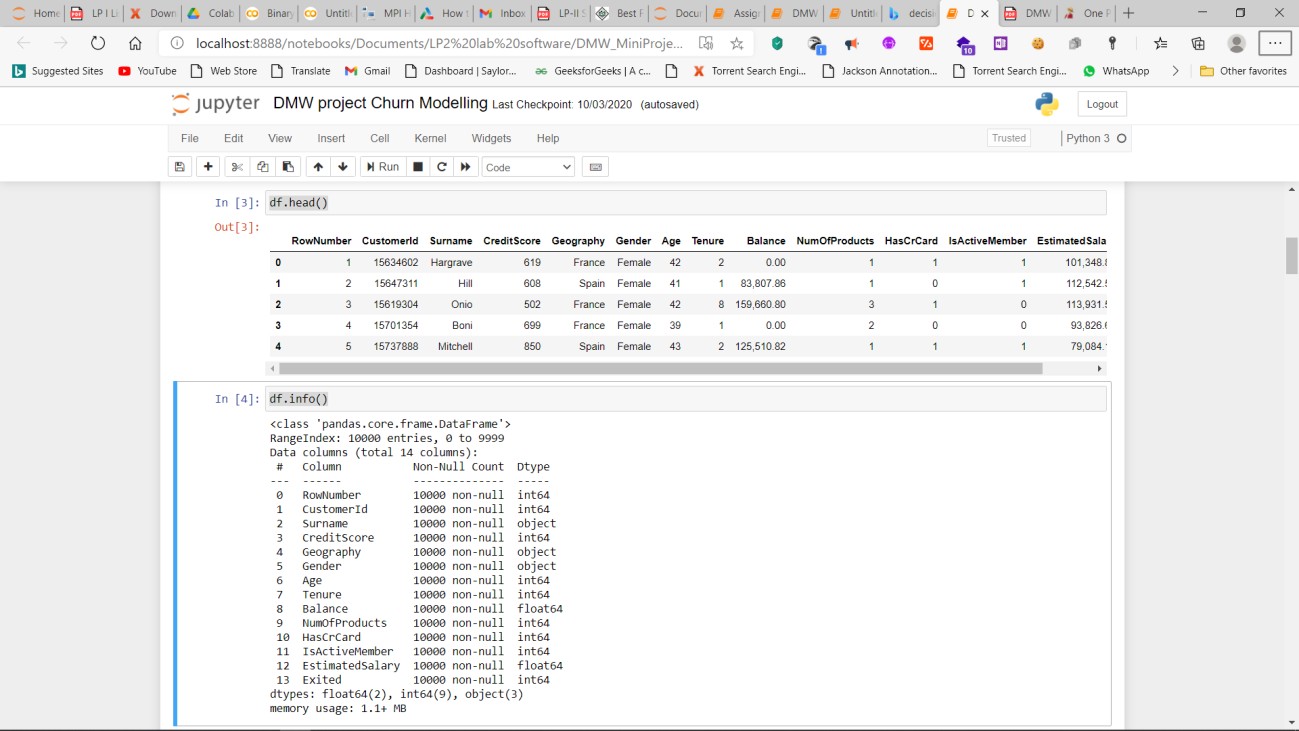
@contextmanager def timer(title):

t0 = time.time() yield

print("{} done in {:.0f}s".format(title, time.time() - t0)) # Read train and test data with pd.read\_csv():

df = pd.read\_csv("Churn\_Modelling.csv") df.head()

df.info()



#Descriptive statistics excluding CustomerId and row number which do not carry any meaningful information for Survival.

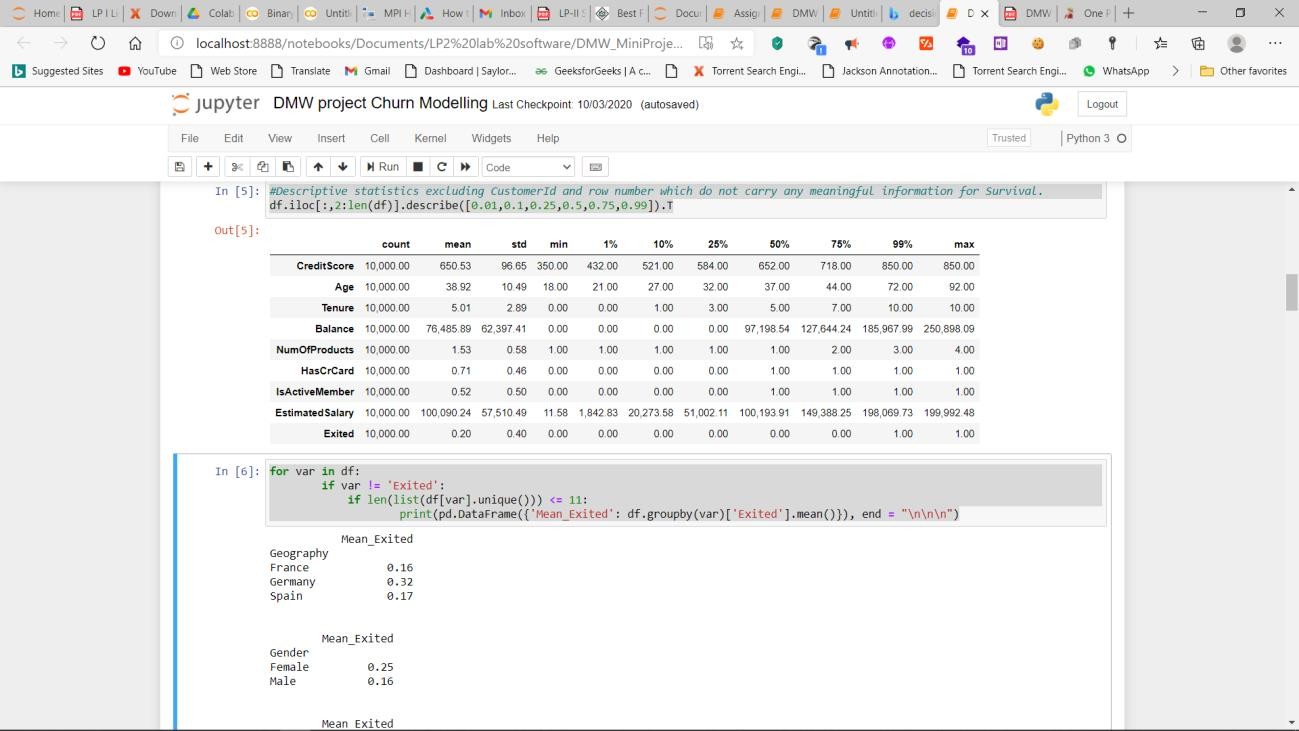
df.iloc[:,2:len(df)].describe([0.01,0.1,0.25,0.5,0.75,0.99]).T for var in df:

if var != 'Exited':

if len(list(df[var].unique())) <= 11:

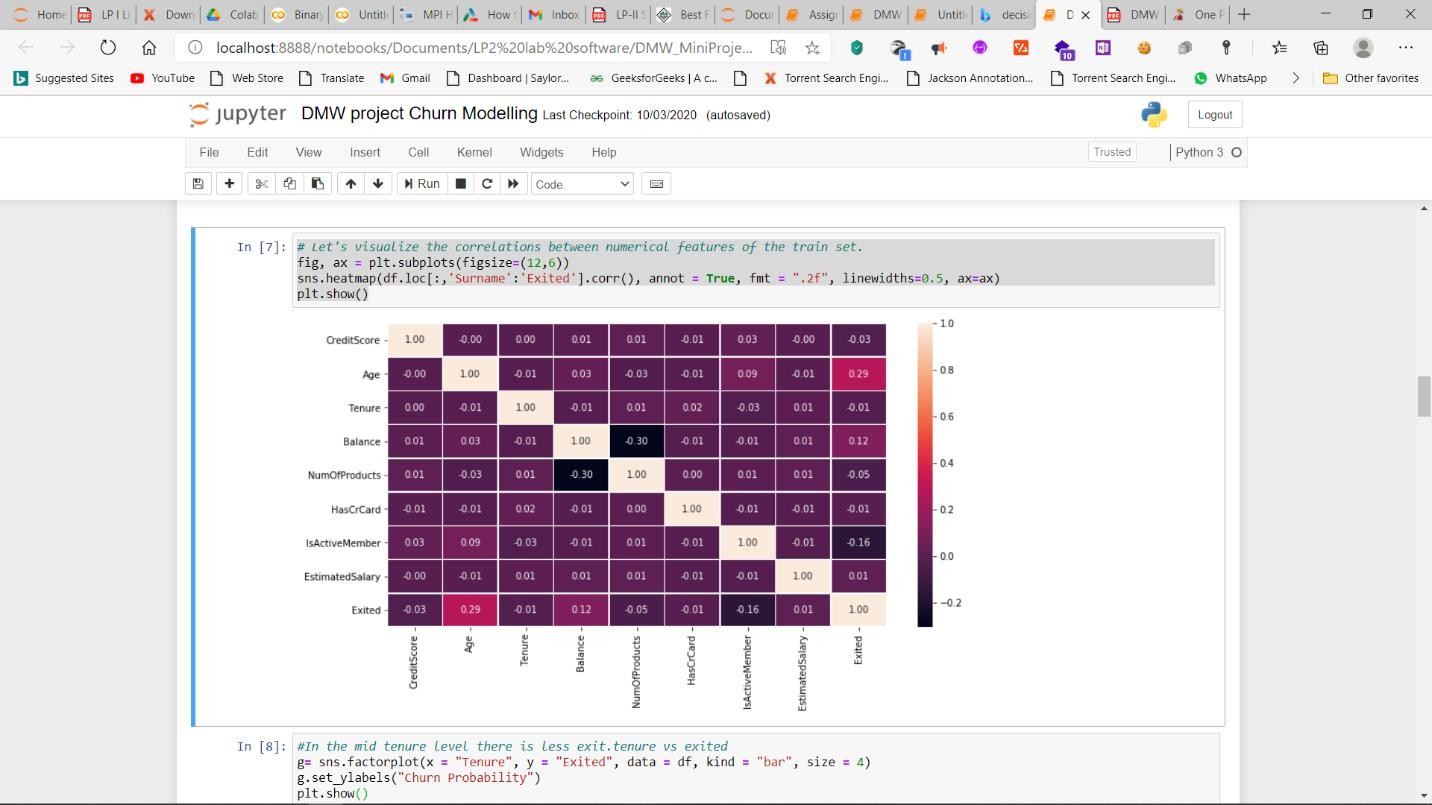
print(pd.DataFrame({'Mean\_Exited': df.groupby(var)['Exited'].mean()}), end =

"\n\n\n")



# Let's visualize the correlations between numerical features of the train set. fig, ax = plt.subplots(figsize=(12,6))

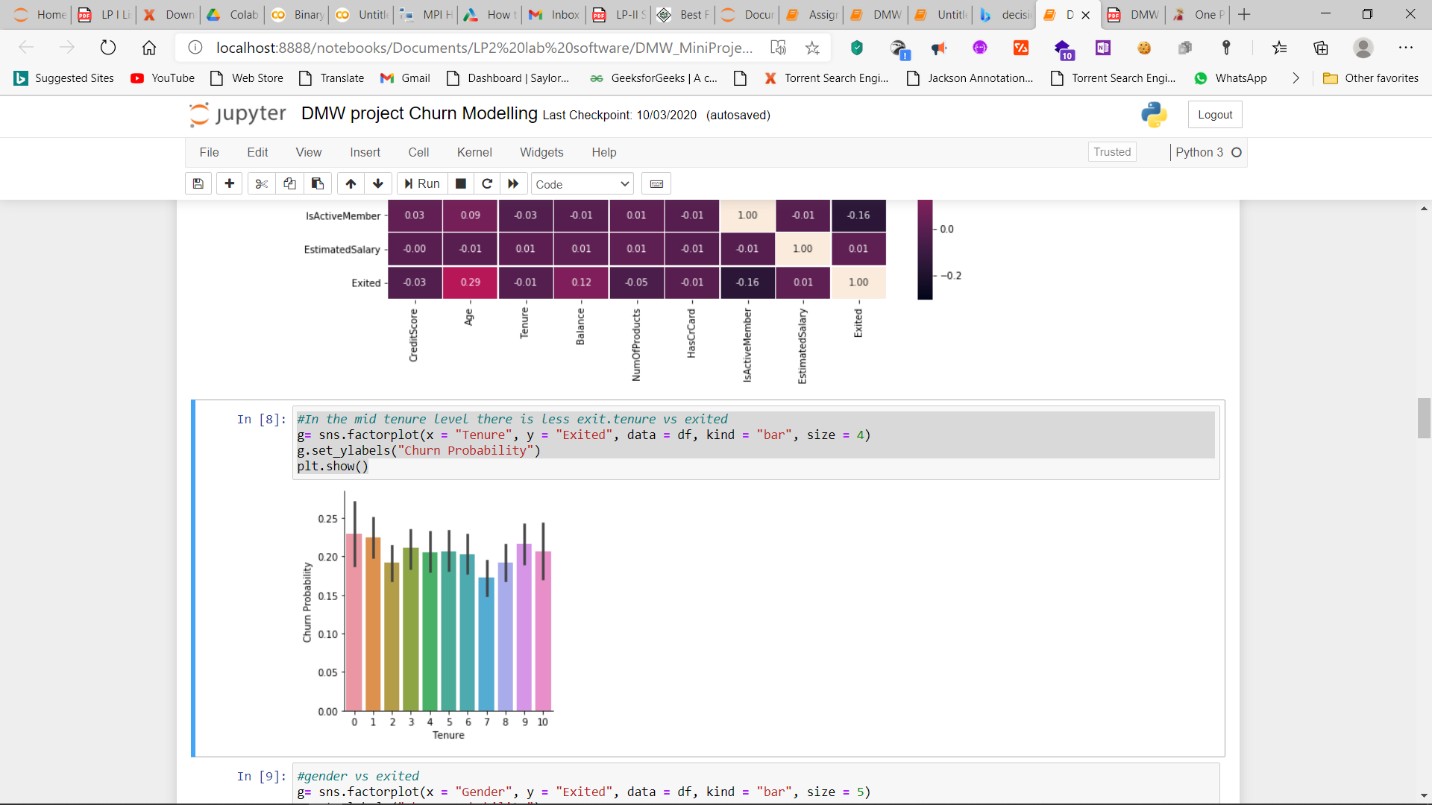
sns.heatmap(df.loc[:,'Surname':'Exited'].corr(), annot = True, fmt = ".2f", linewidths=0.5, ax=ax) plt.show()



#In the mid tenure level there is less exit.tenure vs exited

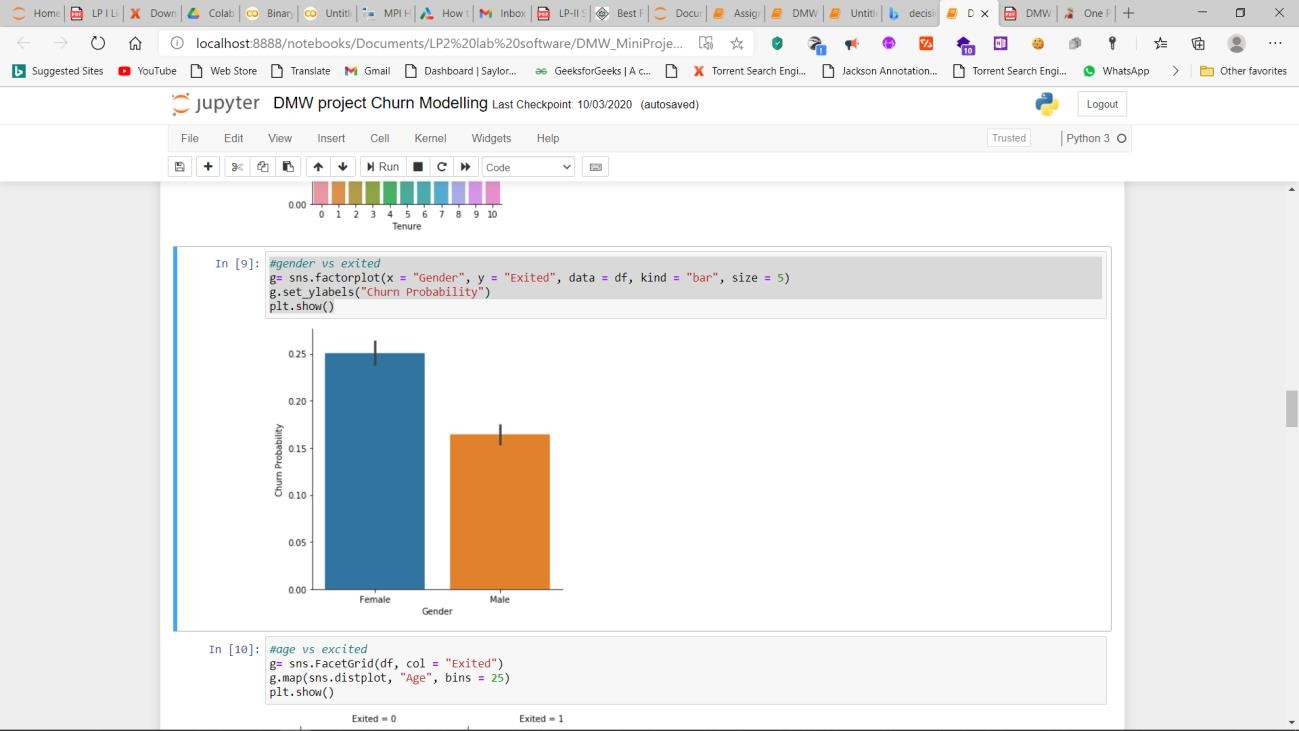
g= sns.factorplot(x = "Tenure", y = "Exited", data = df, kind = "bar", size = 4) g.set\_ylabels("Churn Probability")

plt.show()



#gender vs exited

g= sns.factorplot(x = "Gender", y = "Exited", data = df, kind = "bar", size = 5) g.set\_ylabels("Churn Probability")

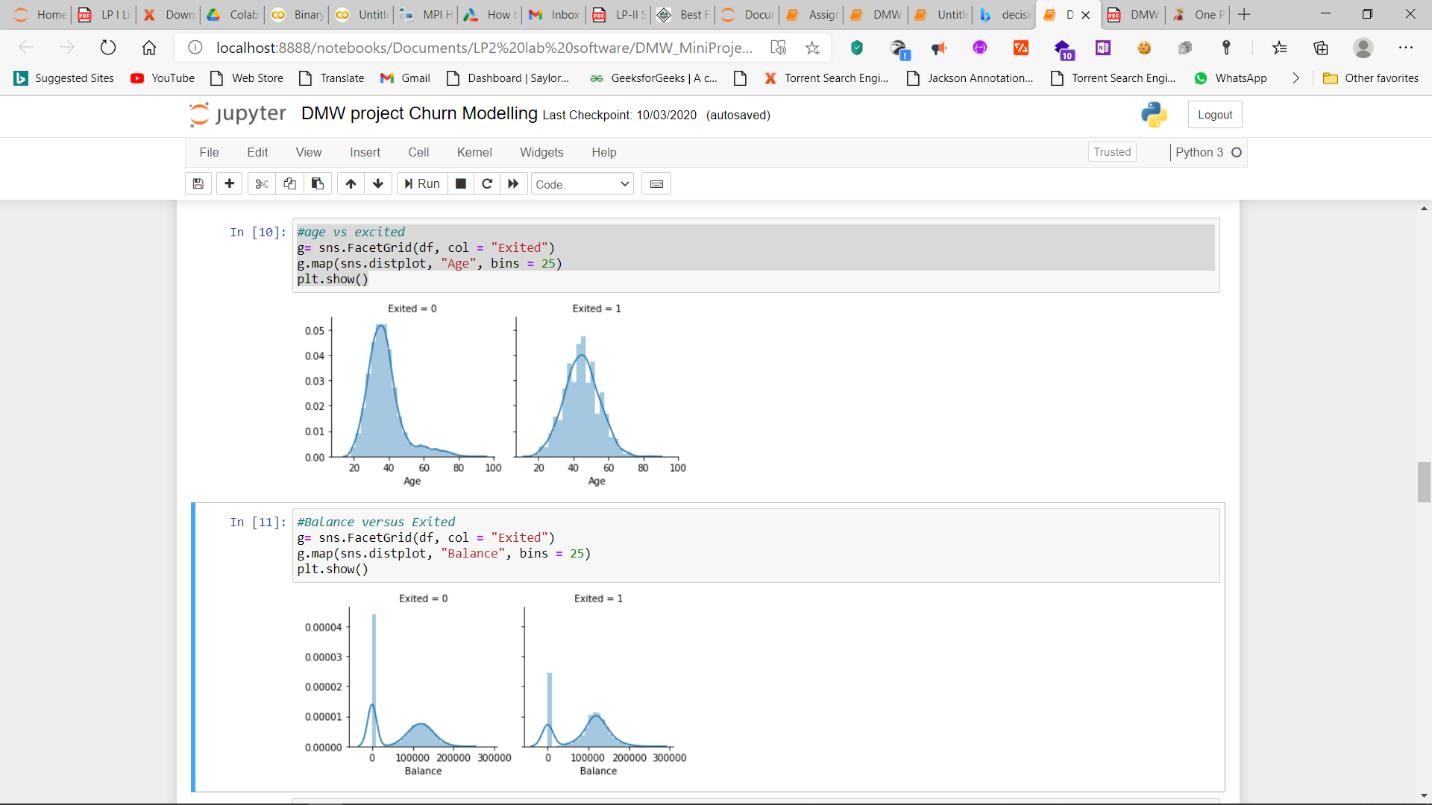
plt.show()

#age vs excited

g= sns.FacetGrid(df, col = "Exited") g.map(sns.distplot, "Age", bins = 25) plt.show()

#Balance versus Exited

g= sns.FacetGrid(df, col = "Exited") g.map(sns.distplot, "Balance", bins = 25) plt.show()



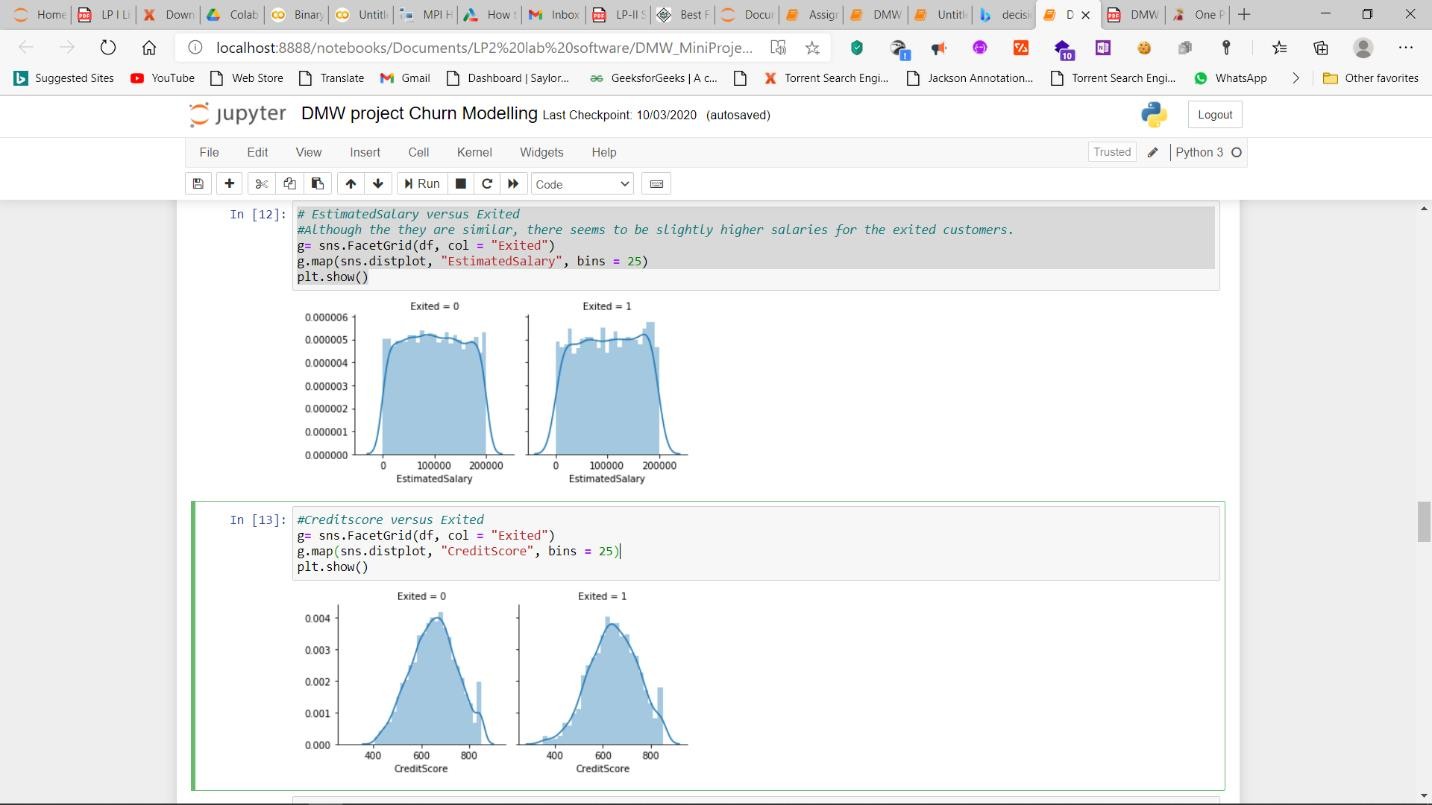
# EstimatedSalary versus Exited

#Although the they are similar, there seems to be slightly higher salaries for the exited customers.

g= sns.FacetGrid(df, col = "Exited") g.map(sns.distplot, "EstimatedSalary", bins = 25) plt.show()

#Creditscore versus Exited

g= sns.FacetGrid(df, col = "Exited") g.map(sns.distplot, "CreditScore", bins = 25) plt.show()



# Data Preprocessing

# There is no missing value in the data as seen in section. In addition, from decriptive statistics we can see that median and mean values are very similar for most of the numerical variables.

# Splitting the data as train and validation data

# The given data is splitted into train and validation sets to test the accuracy of training with the untrained 20% of the sample.

##

xs = df.drop(['RowNumber',"Exited"], axis=1) target = df["Exited"]

x\_train, x\_val, y\_train, y\_val = train\_test\_split(xs, target, test\_size = 0.20, random\_state = 0)

val\_ids = x\_val['CustomerId'] train\_ids=x\_train['CustomerId']

x\_train = x\_train.drop(['CustomerId'], axis=1)

x\_val= x\_val.drop(['CustomerId'], axis=1)

df\_train=df[df['CustomerId'].isin(train\_ids)] df\_val=df[df['CustomerId'].isin(val\_ids)]

x\_train.shape

# Handling Categorical Variables

# Label encoding of gender variable and removing surname for df in [x\_train,x\_val]:

df["Gender"]=df["Gender"].map(lambda x: 0 if x=='Female' else 1) df.drop(['Surname'], axis = 1, inplace=True)

# One hot encoding of Geography (Country)

x\_train,x\_val= [ pd.get\_dummies(data, columns = ['Geography']) for data in [x\_train,x\_val]] x\_train.shape

x\_train.info()

# Memory Reduction

def reduce\_mem\_usage(df, verbose=True):

numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64'] start\_mem = df.memory\_usage().sum() / 1024\*\*2

for col in df.columns: col\_type = df[col].dtypes if col\_type in numerics:

c\_min = df[col].min() c\_max = df[col].max()

if str(col\_type)[:3] == 'int':

if c\_min > np.iinfo(np.int8).min and c\_max < np.iinfo(np.int8).max:

df[col] = df[col].astype(np.int8)

elif c\_min > np.iinfo(np.int16).min and c\_max < np.iinfo(np.int16).max: df[col] = df[col].astype(np.int16)

elif c\_min > np.iinfo(np.int32).min and c\_max < np.iinfo(np.int32).max: df[col] = df[col].astype(np.int32)

elif c\_min > np.iinfo(np.int64).min and c\_max < np.iinfo(np.int64).max: df[col] = df[col].astype(np.int64)

else:

if c\_min > np.finfo(np.float16).min and c\_max < np.finfo(np.float16).max: df[col] = df[col].astype(np.float16)

elif c\_min > np.finfo(np.float32).min and c\_max < np.finfo(np.float32).max: df[col] = df[col].astype(np.float32)

else:

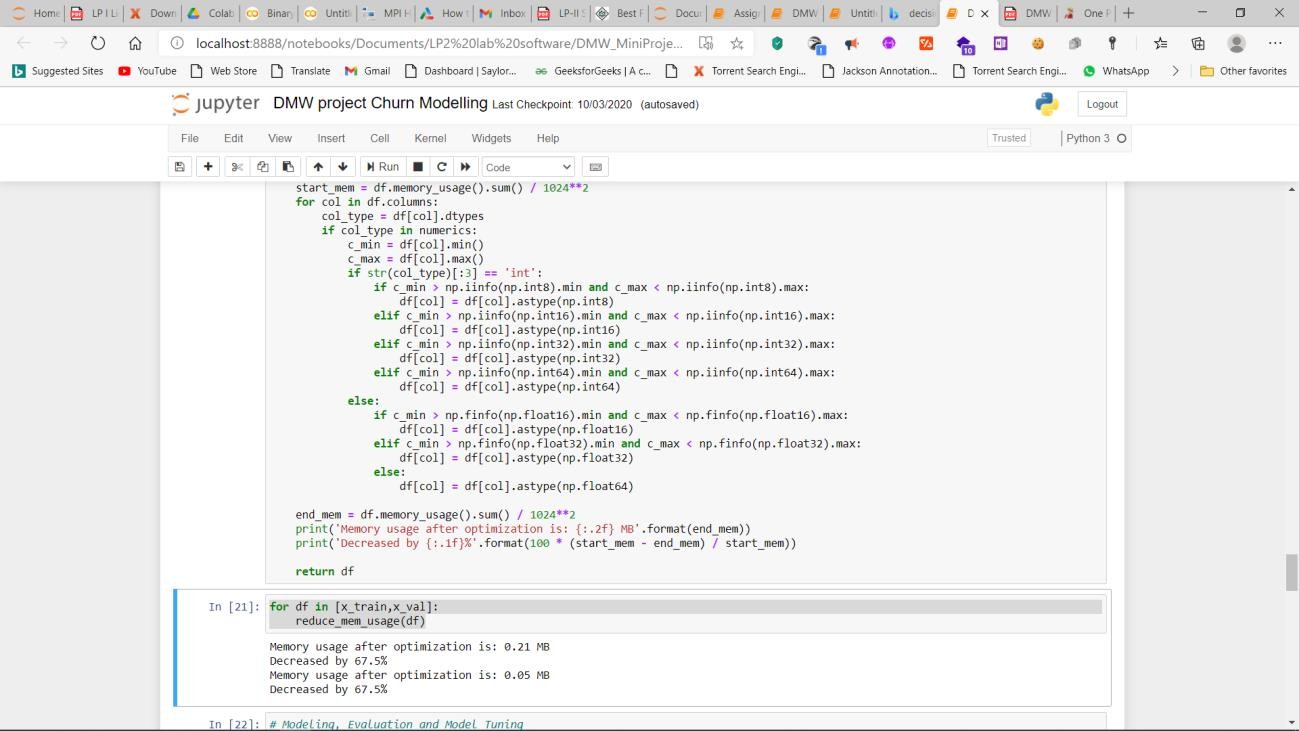
df[col] = df[col].astype(np.float64)

end\_mem = df.memory\_usage().sum() / 1024\*\*2

print('Memory usage after optimization is: {:.2f} MB'.format(end\_mem)) print('Decreased by {:.1f}%'.format(100 \* (start\_mem - end\_mem) / start\_mem))

return df

for df in [x\_train,x\_val]: reduce\_mem\_usage(df)



# Modeling, Evaluation and Model Tuning

# Validation Set Accuracy for the default models r=1309

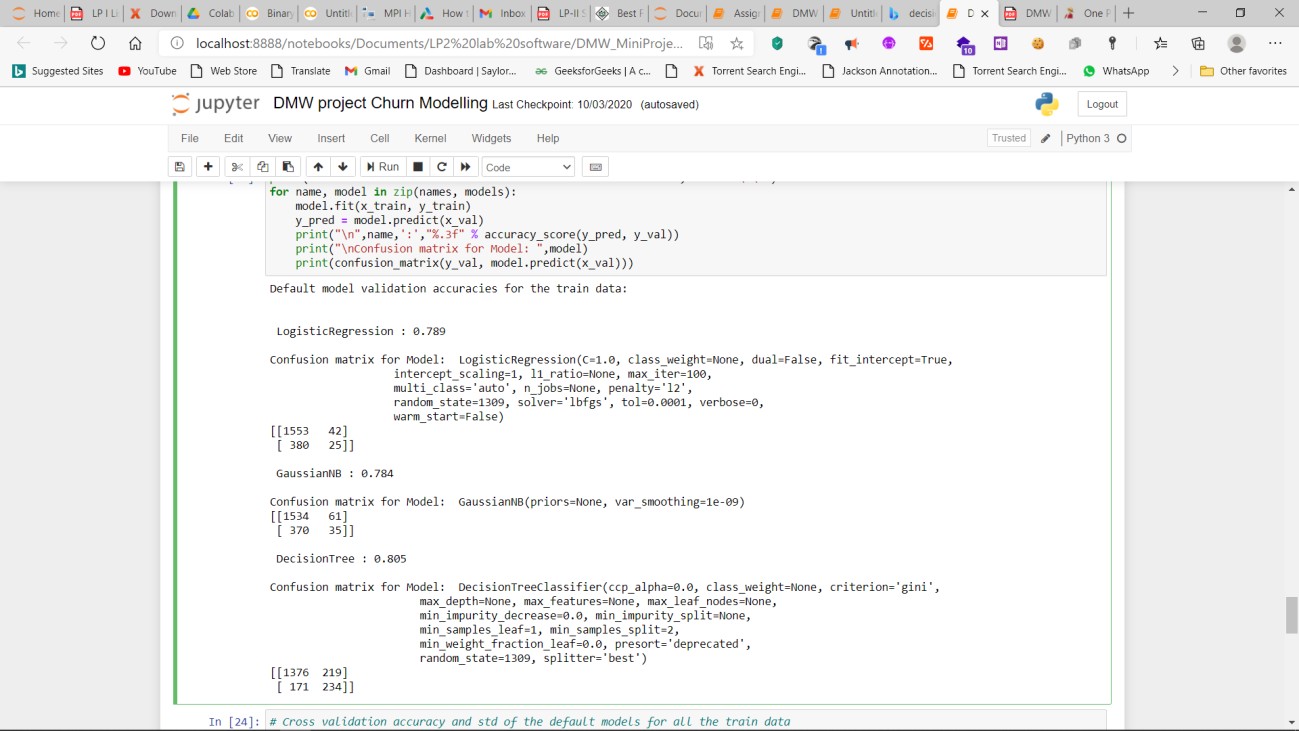
models = [LogisticRegression(random\_state=r),GaussianNB(), DecisionTreeClassifier(random\_state=r)]

names = ["LogisticRegression","GaussianNB","DecisionTree"]

print('Default model validation accuracies for the train data:', end = "\n\n") for name, model in zip(names, models):

model.fit(x\_train, y\_train) y\_pred = model.predict(x\_val)

print("\n",name,':',"%.3f" % accuracy\_score(y\_pred, y\_val)) print("\nConfusion matrix for Model: ",model) print(confusion\_matrix(y\_val, model.predict(x\_val)))



# Cross validation accuracy and std of the default models for all the train data predictors=pd.concat([x\_train,x\_val])

results = []

print('3 fold Cross validation accuracy and std of the default models for the train data:', end = "\n\n")

for name, model in zip(names, models):

kfold = KFold(n\_splits=3, random\_state=1001)

cv\_results = cross\_val\_score(model, predictors, target, cv = kfold, scoring = "accuracy") results.append(cv\_results)

print("{}: {} ({})".format(name, "%.3f" % cv\_results.mean() ,"%.3f" % cv\_results.std())) # Model tuning using crossvalidation

# Possible hyper parameters logreg\_params= {"C":np.logspace(-1, 1, 10),

"penalty": ["l1","l2"], "solver":['lbfgs', 'liblinear', 'sag', 'saga'], "max\_iter":[1000]}

NB\_params = {'var\_smoothing': np.logspace(0,-9, num=100)}

dtree\_params = {"min\_samples\_split" : range(10,500,20), "max\_depth": range(1,20,2)}

classifier\_params = [logreg\_params,NB\_params,dtree\_params]

# Tuning by Cross Validation cv\_result = {} best\_estimators = {}

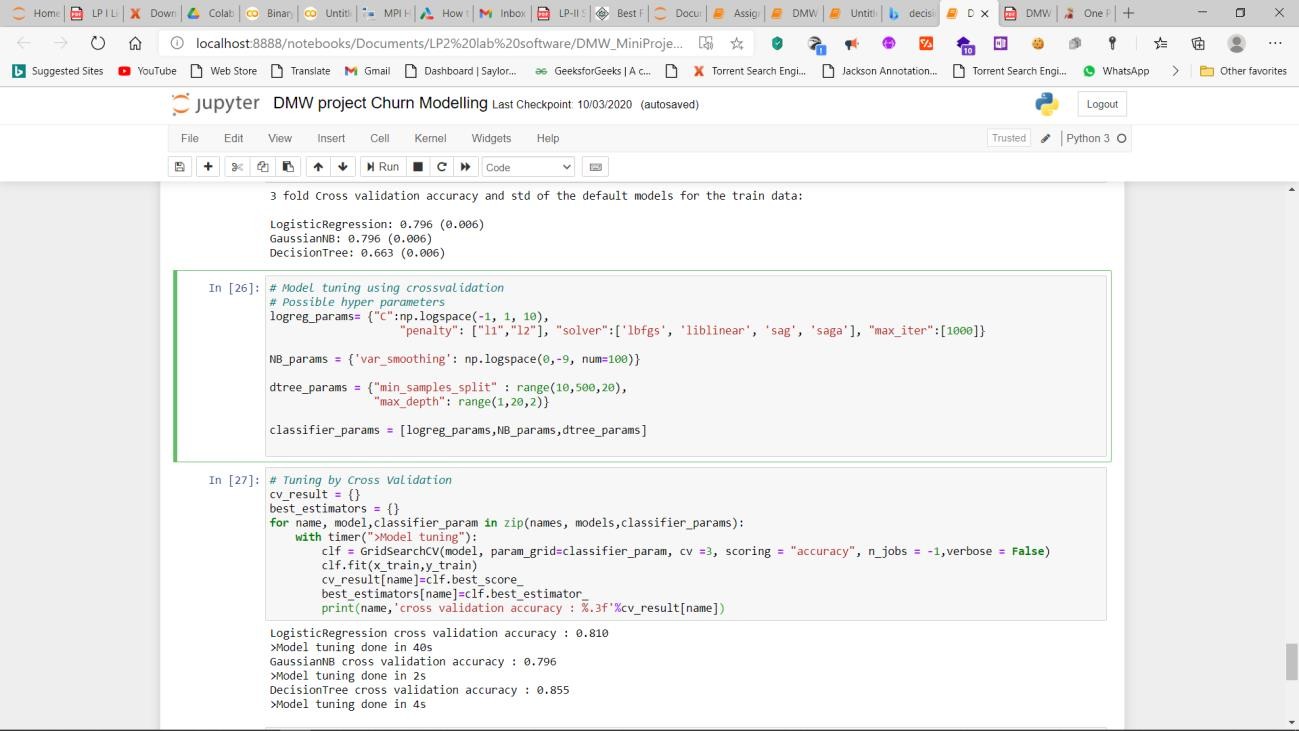
for name, model,classifier\_param in zip(names, models,classifier\_params): with timer(">Model tuning"):

clf = GridSearchCV(model, param\_grid=classifier\_param, cv =3, scoring = "accuracy", n\_jobs

= -1,verbose = False) clf.fit(x\_train,y\_train) cv\_result[name]=clf.best\_score\_

best\_estimators[name]=clf.best\_estimator\_

print(name,'cross validation accuracy : %.3f'%cv\_result[name])



accuracies={}

print('Validation accuracies of the tuned models for the train data:', end = "\n\n") for name, model\_tuned in zip(best\_estimators.keys(),best\_estimators.values()):

y\_pred = model\_tuned.fit(x\_train,y\_train).predict(x\_val) accuracy=accuracy\_score(y\_pred, y\_val)

print(name,':', "%.3f" %accuracy) accuracies[name]=accuracy

# Ensembling first n (e.g. 3) models n=3

accu=sorted(accuracies, reverse=True, key= lambda k:accuracies[k])[:n] firstn=[[k,v] for k,v in best\_estimators.items() if k in accu]

# Ensembling First n Score

votingC = VotingClassifier(estimators = firstn, voting = "soft", n\_jobs = -1)

votingC = votingC.fit(x\_train, y\_train) print(accuracy\_score(votingC.predict(x\_val),y\_val))

