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:

A. 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:
  a. Emails-> spam/ not spam
  b. Outline transactions-> fraud /not fraud

      c. Tumor-> Malignant /Benign

B. 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

C. 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.

D. Classification Workflow:
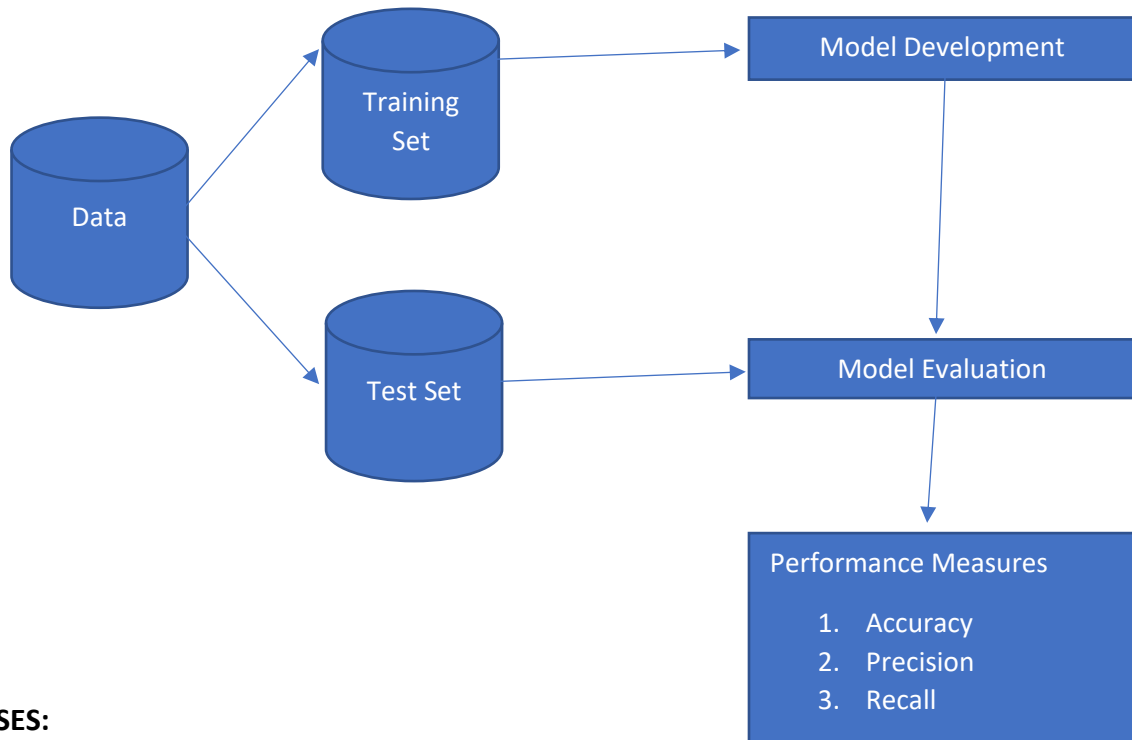
TEST CASES:

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

2. 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

3. 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:

- DecisionTree : 0.860

CONCLUSION:

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

CODE:

```python
# 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
```

```python
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()
```

In [3]: `df.head()`

Out[3]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSala
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1	101,348.8
1	2	15647311	Hill	608	Spain	Female	41	1	83,807.86	1	0	1	112,542.5
2	3	15619304	Onio	502	France	Female	42	8	159,660.80	3	1	0	113,931.5
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	0	93,826.6
4	5	15737888	Mitchell	850	Spain	Female	43	2	125,510.82	1	1	1	79,084.1

In [4]: `df.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   RowNumber        10000 non-null  int64
 1   CustomerId       10000 non-null  int64
 2   Surname          10000 non-null  object
 3   CreditScore      10000 non-null  int64
 4   Geography        10000 non-null  object
 5   Gender           10000 non-null  object
 6   Age              10000 non-null  int64
 7   Tenure           10000 non-null  int64
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64
 10  HasCrCard        10000 non-null  int64
 11  IsActiveMember   10000 non-null  int64
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

#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")

jupyter DMW project Churn Modelling Last Checkpoint: 10/03/2020 (autosaved)   Logout

File Edit View Insert Cell Kernel Widgets Help   Trusted   Python 3 O

In [5]: #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

Out[5]:

	count	mean	std	min	1%	10%	25%	50%	75%	99%	max
CreditScore	10,000.00	650.53	96.65	350.00	432.00	521.00	584.00	652.00	718.00	850.00	850.00
Age	10,000.00	38.92	10.49	18.00	21.00	27.00	32.00	37.00	44.00	72.00	92.00
Tenure	10,000.00	5.01	2.89	0.00	0.00	1.00	3.00	5.00	7.00	10.00	10.00
Balance	10,000.00	76,485.89	62,397.41	0.00	0.00	0.00	0.00	97,198.54	127,644.24	185,967.99	250,898.09
NumOfProducts	10,000.00	1.53	0.58	1.00	1.00	1.00	1.00	1.00	2.00	3.00	4.00
HasCrCard	10,000.00	0.71	0.46	0.00	0.00	0.00	0.00	1.00	1.00	1.00	1.00
IsActiveMember	10,000.00	0.52	0.50	0.00	0.00	0.00	0.00	1.00	1.00	1.00	1.00
EstimatedSalary	10,000.00	100,090.24	57,510.49	11.58	1,842.83	20,273.58	51,002.11	100,193.91	149,388.25	198,069.73	199,992.48
Exited	10,000.00	0.20	0.40	0.00	0.00	0.00	0.00	0.00	0.00	1.00	1.00

```
In [6]: 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")
```

```
        Mean_Exited
Geography
France        0.16
Germany       0.32
Spain         0.17


        Mean_Exited
Gender
Female        0.25
Male          0.16


        Mean_Exited
```
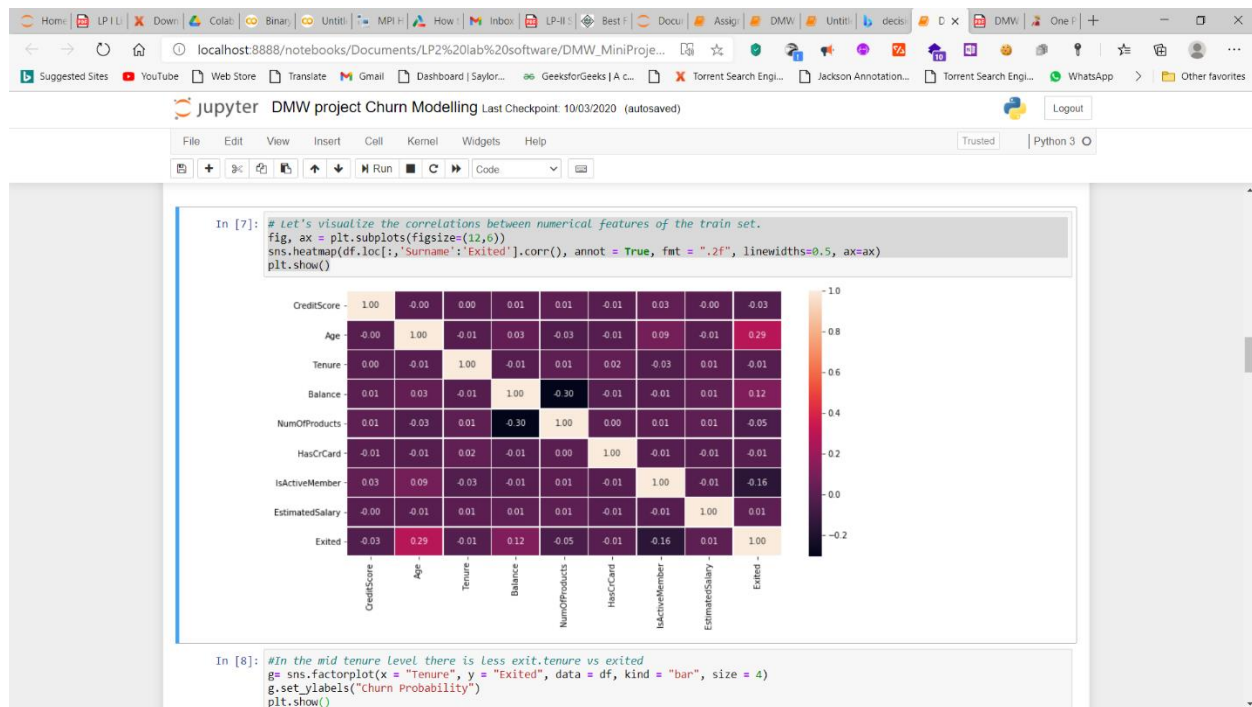
Let's visualize the correlations between numerical features of the train set.

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 [7]:  # 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 [8]:  #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()
```

#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()


```
In [8]:  #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()
```

```
In [9]:  #gender vs exited
         g= sns.factorplot(x = "Gender", y = "Exited", data = df, kind = "bar", size = 5)
```
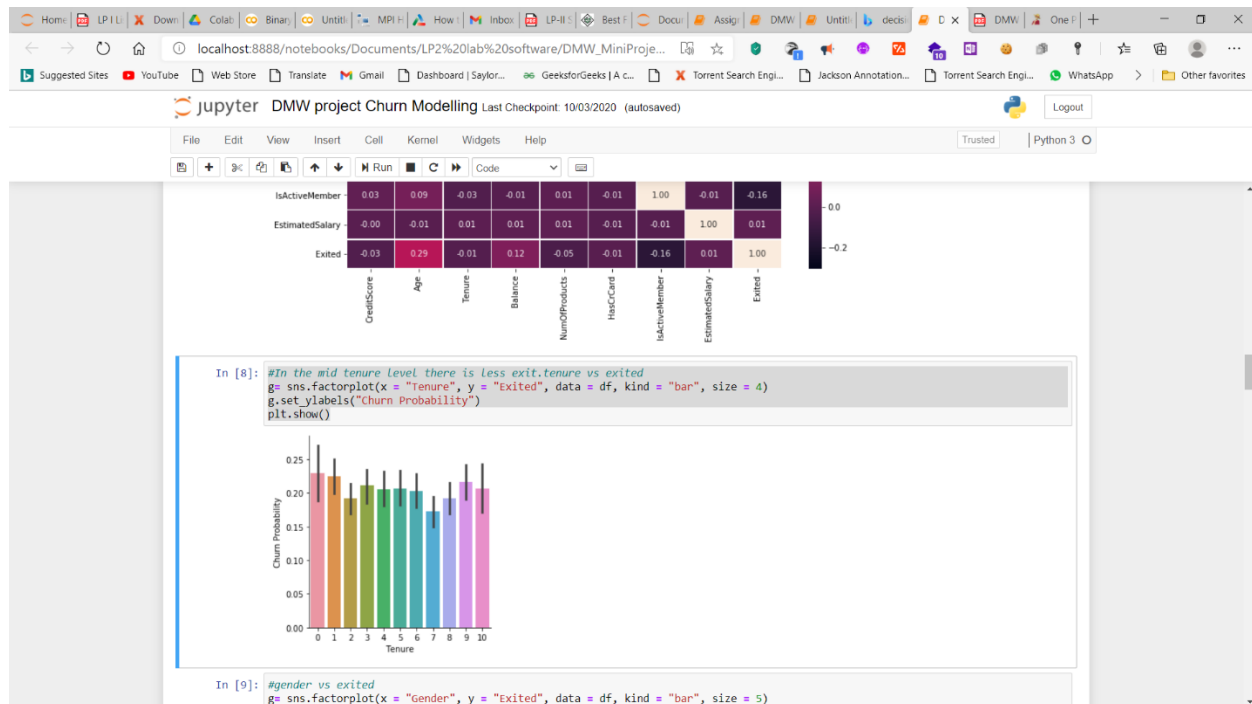
#gender vs exited

g= sns.factorplot(x = "Gender", y = "Exited", data = df, kind = "bar", size = 5)

g.set_ylabels("Churn Probability")
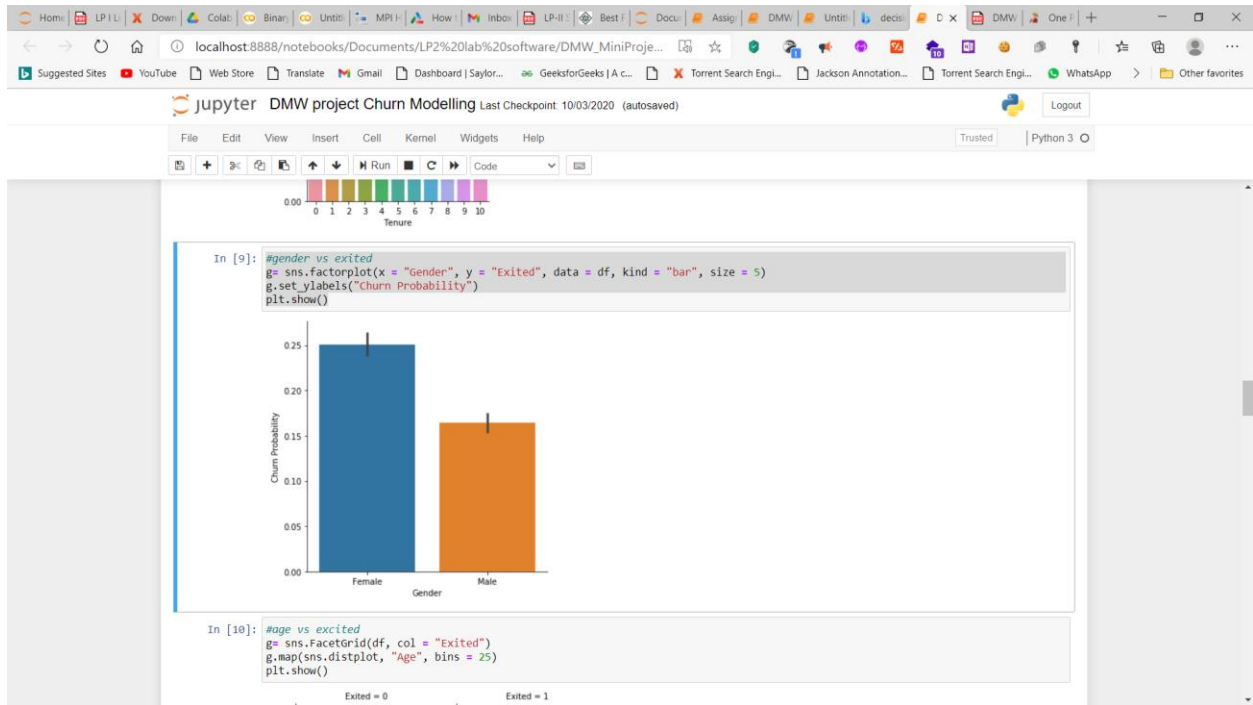
plt.show()

#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")
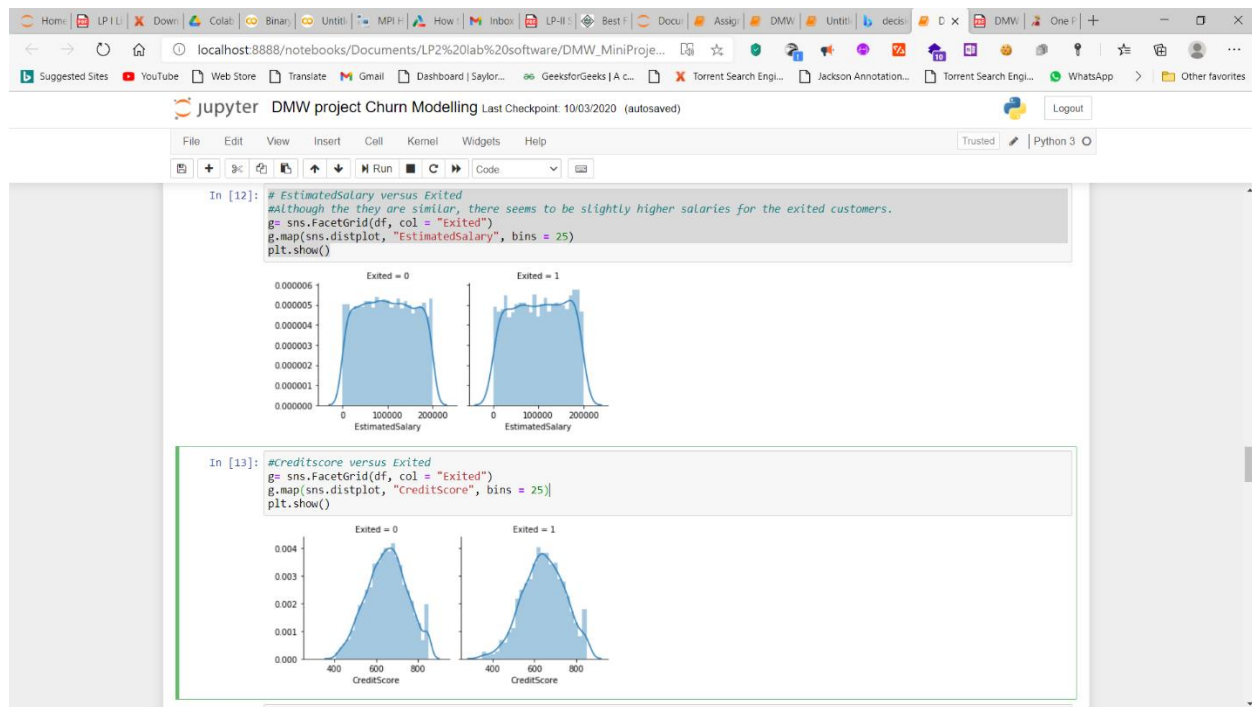
g.map(sns.distplot, "Balance", bins = 25)

plt.show()
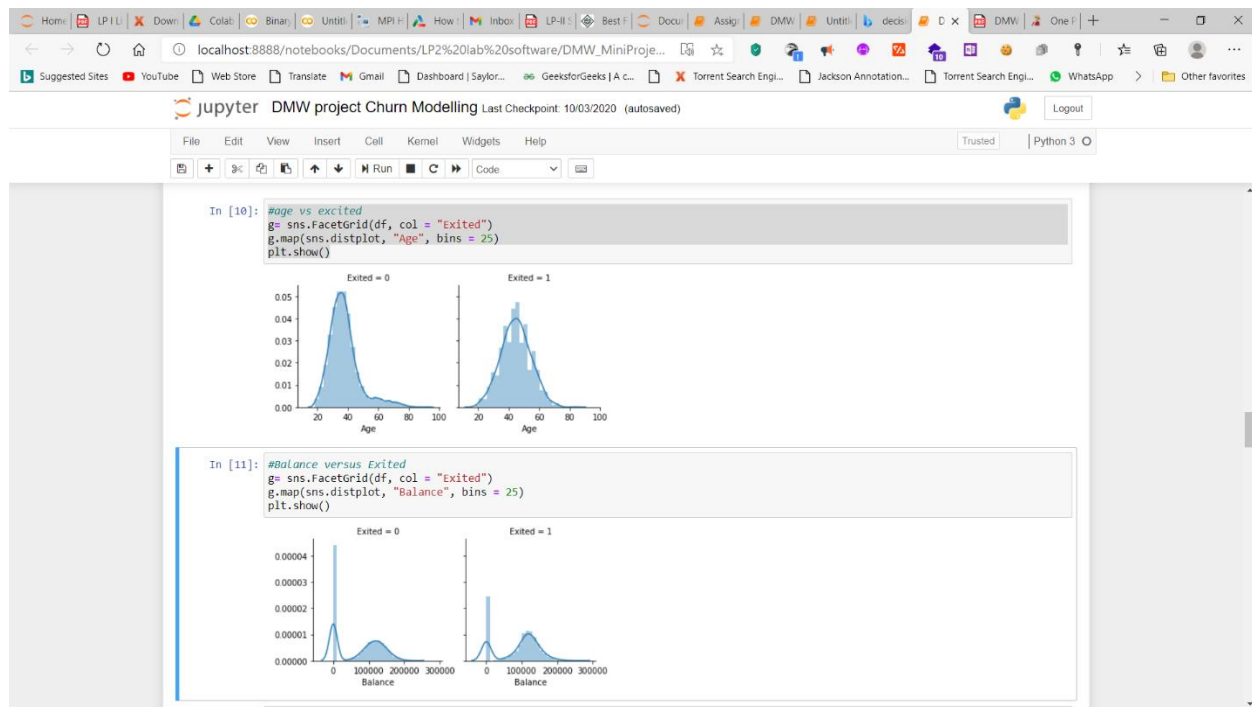
EstimatedSalary versus Exited

#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)

```python
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:
```

```python
                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)
```

```
        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
```

```
In [21]: for df in [x_train,x_val]:
             reduce_mem_usage(df)
```

```
Memory usage after optimization is: 0.21 MB
Decreased by 67.5%
Memory usage after optimization is: 0.05 MB
Decreased by 67.5%
```

```
In [22]: # Modeling, Evaluation and Model Tuning
```

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)))

```
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)))

Default model validation accuracies for the train data:


  LogisticRegression : 0.789

  Confusion matrix for Model:  LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                     intercept_scaling=1, l1_ratio=None, max_iter=100,
                     multi_class='auto', n_jobs=None, penalty='l2',
                     random_state=1309, solver='lbfgs', tol=0.0001, verbose=0,
                     warm_start=False)
[[1553   42]
 [ 380   25]]

  GaussianNB : 0.784

  Confusion matrix for Model:  GaussianNB(priors=None, var_smoothing=1e-09)
[[1534   61]
 [ 370   35]]

  DecisionTree : 0.805

  Confusion matrix for Model:  DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=1309, splitter='best')
[[1376  219]
 [ 171  234]]
```

In [24]: # Cross validation accuracy and std of the default models for all the train data

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)}

```python
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])
```

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

LogisticRegression: 0.796 (0.006)
GaussianNB: 0.796 (0.006)
DecisionTree: 0.663 (0.006)
```

```python
In [26]:  # 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]
```

```python
In [27]:  # 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])

          LogisticRegression cross validation accuracy : 0.810
          >Model tuning done in 40s
          GaussianNB cross validation accuracy : 0.796
          >Model tuning done in 2s
          DecisionTree cross validation accuracy : 0.855
          >Model tuning done in 4s
```

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]
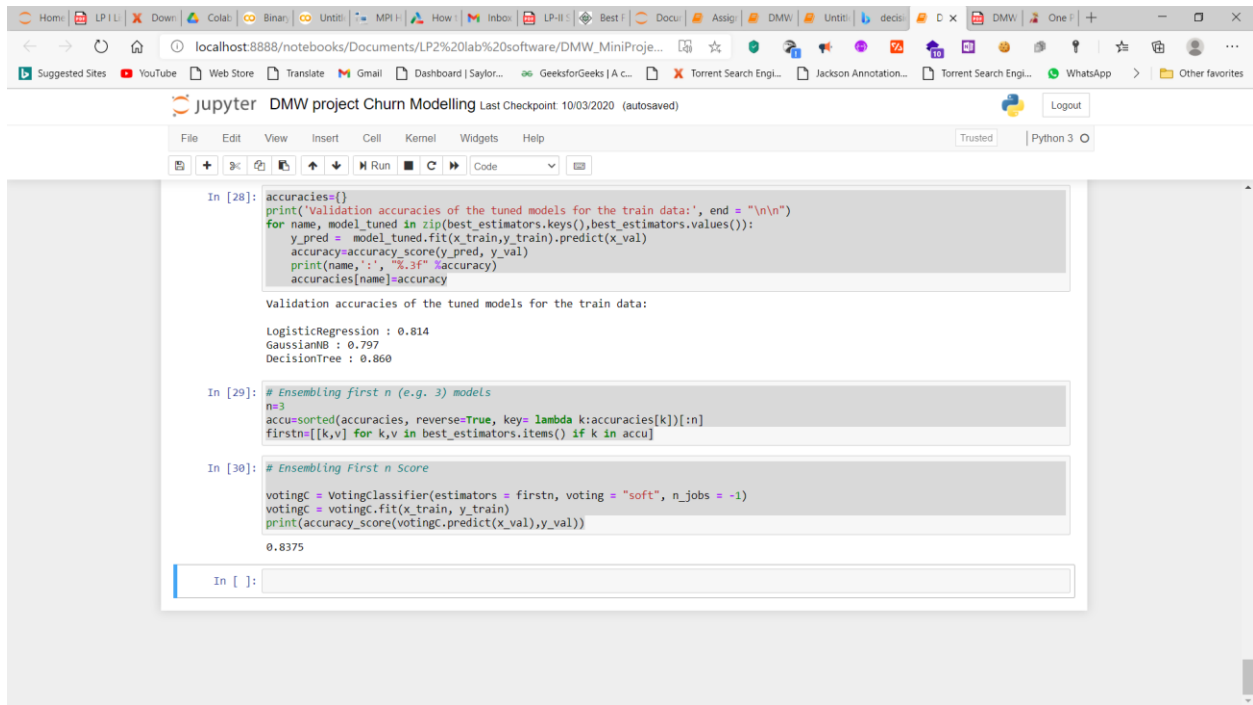

Ensembling First n Score

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))