Financial: Analysis on Customer Retention Data

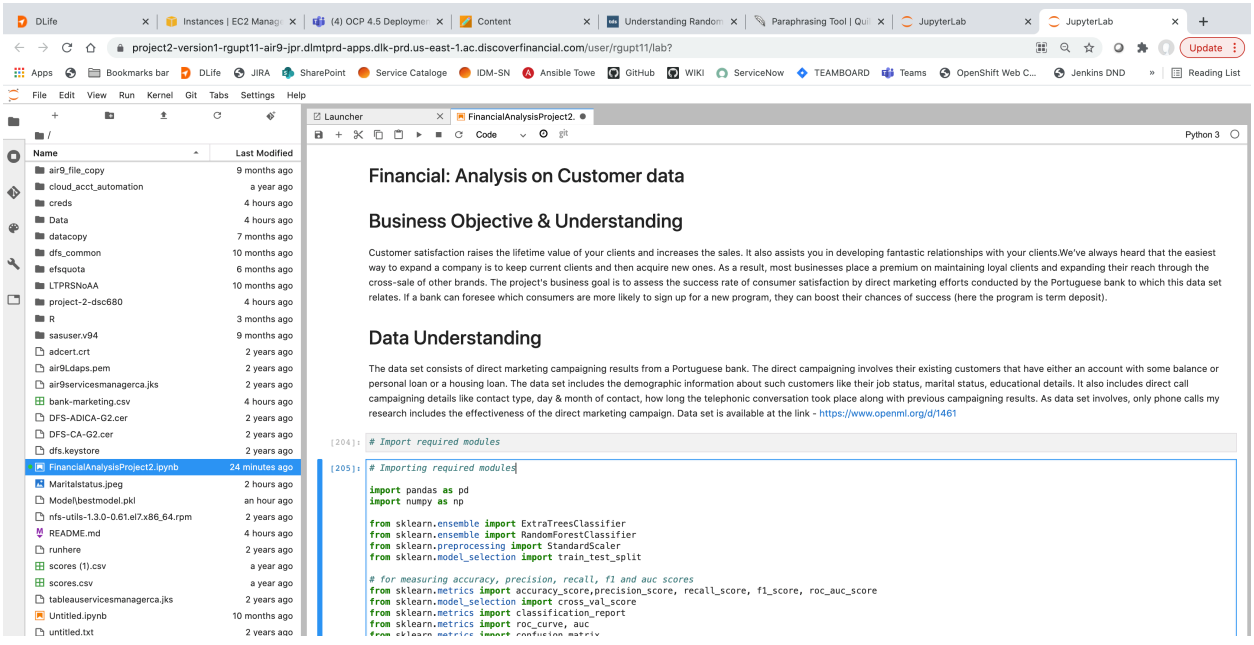
Rahul Gupta - DSC 680 - Spring 2021

https://github.com/rahulgupta271/DSC680_Project_2_Financial_Analysis_Customer_data

Project 2 Draft Report

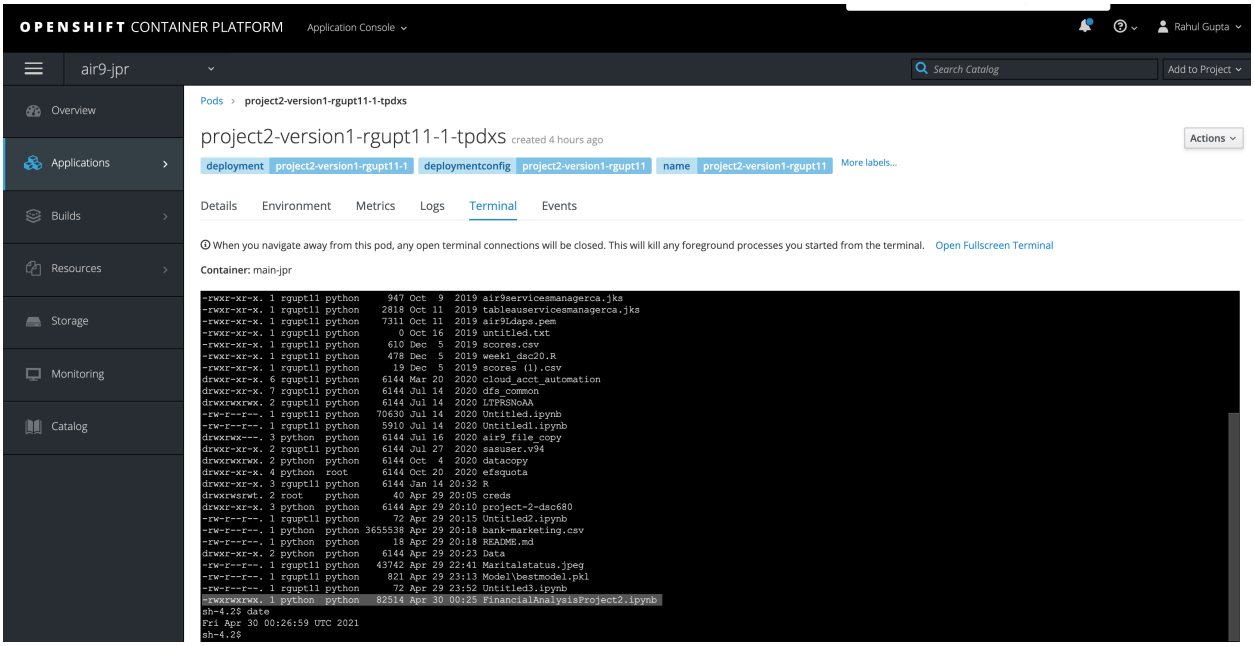
I am using JupyterLab for my Project and which is installed on Redhat OpenShift Platform.

" JupyterLab enables you to arrange your work area with notebooks, text files, terminals, and notebook outputs. JupyterLab provides a high level of integration between notebooks, documents, and activities: Drag-and-drop to reorder notebook cells and copy them between notebooks. "



"OpenShift is a family of containerization software products developed by Red Hat. Its flagship product is the OpenShift Container Platform — an on-premises platform as a service built around

Docker containers orchestrated and managed by Kubernetes on a foundation of Red Hat Enterprise Linux “



1. Understanding Goals and Objectives in Business

Customer satisfaction raises the lifetime value of your clients and increases the sales. It also assis ts you in developing fantastic relationships with your clients.We’ve always heard that the easiest way to expand a company is to keep current clients and then acquire new ones. As a result, most businesses place a premium on maintaining loyal clients and expanding their reach through the cross-sale of other brands. The project's business goal is to assess the success rate of consumer satisfaction by direct marketing efforts conducted by the Portuguese bank to which this data set relates. If a bank can foresee which consumers are more likely to sign up for a new program, they can boost their chances of success (here the program is term deposit).

Business objective:

In terms of modeling, the market goal is to describe a statistical model that involves the binary division of consumers into two groups. –

1) Customers who may open the bank's term deposit account after the campaign.

2) Customers who do not open the bank's term deposit account.

This classification really lets banks recognize consumers that they can invest their marketing efforts so that they have the greatest chance of succeeding in their campaign, resulting in more long-term deposit accounts (indirectly raising the success rate of their program).

Approach:

CRISP – DM Technique has been used to accomplish this task in supervised learning classification.

2. Data Understanding

The data collection consists of the direct marketing performance of the Portuguese bank. Direct lobbying includes their current clients who may have a balance account or a personal loan or a housing loan. The data collection contains personal information for clients such as their working status, marital status, educational records. It also provides direct call lobbying information such as contact form, day & month of contact, how long a telephone chat took place along with past campaign data. As data set includes, just phone calls to my study include the success of a targeted marketing strategy.

Data set can be found at the link - https://www.openml.org/d/1461

Feature Variables: There are 16 input variables, some of which are categorical and some of which are numeric. Below is the whole collection

1 - age (numeric)

2 - job : type of job (categorical: "admin.","unknown","unemployed","management","housemaid","entrepreneur", "student","blue-collar","self-employed","retired","technician","services")

3 - marital : marital status (categorical: "married","divorced","single"; note: "divorced" means divorced or widowed)

4 - education (categorical: "unknown","secondary","primary","tertiary") 5 - default: has credit in default? (binary: "yes","no")
6 - balance: average yearly balance, in euros (numeric)
7 - housing: has housing loan? (binary: "yes","no")

8 - loan: has personal loan? (binary: "yes","no")

related with the last contact of the current campaign:
9 - contact: contact communication type (categorical: "unknown","telephone","cellular")

10 - day: last contact day of the month (numeric)
11 - month: last contact month of year (categorical: "jan", "feb", "mar", ..., "nov", "dec") 12 - duration: last contact duration, in seconds (numeric)

other attributes:
13 - campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)

14 - pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric, -1 means client was not previously contacted)

15 - previous: number of contacts performed before this campaign and for this client (numeric)

16 - poutcome: outcome of the previous marketing campaign (categorical: "unknown","other","failure","success")

Target Variable: Just one attribute that has to be estimated is the customer's classification, the Boolean form is yes/no (represented by 1, 0 in the data set). Yeah, means that the customer has subscribed for the term deposit and No means that the customer has not subscribed. The original data collection has values 1 & 2 that have been updated as 0 & 1 prior to exploratory data processing.

Exploratory Data Analysis: First, I searched for missing values in the data collection. Luckily, the data collection available was very tidy for some missed values.

[98]:
```
# Checking for blank values for each column
import datetime
now = datetime.datetime.now()

print()
def personal_details():
    name, age = "Rahul Gupta", 680
    address = "Fadi Alsaleem"
    print("Student Name: {}\nCourse: {}\nInstructor: {}".format(name, age, address))

personal_details()

print()
print ("Current date and time : ")
print (now.strftime("%Y-%m-%d %H:%M:%S"))

print()

nullcols = bankdf.isnull().sum()
print(nullcols)
```

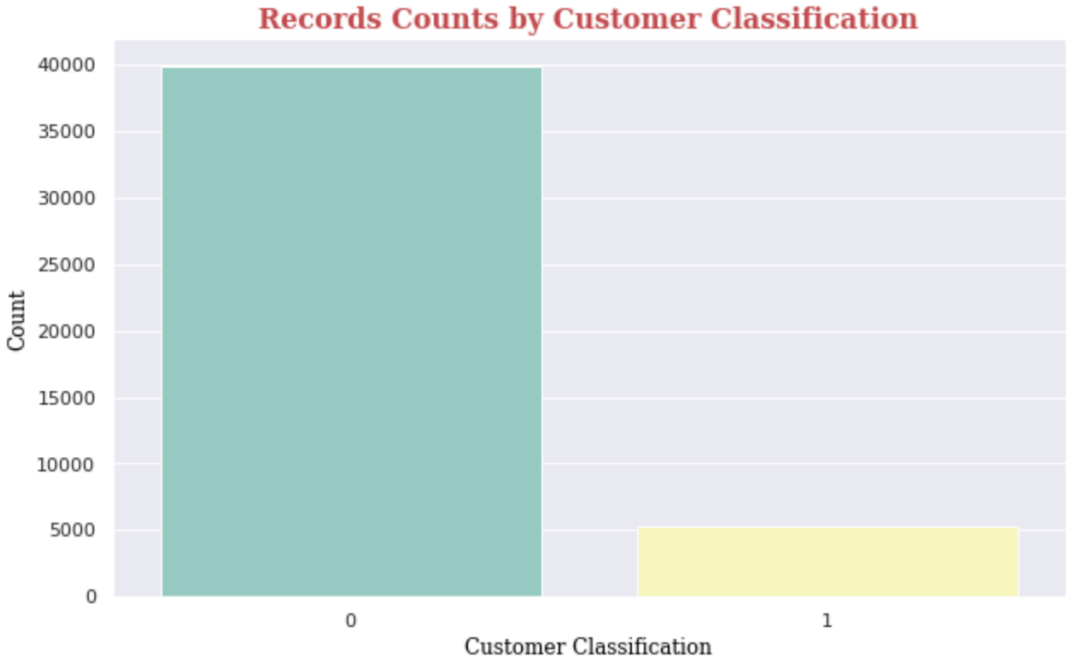
```
Student Name: Rahul Gupta
Course: 680
Instructor: Fadi Alsaleem

Current date and time :
2021-04-29 22:10:06

age               0
job               0
marital_status    0
education         0
default           0
balance           0
housing           0
loan              0
contact           0
day               0
month             0
duration          0
campaign          0
pdays             0
previous          0
poutcome          0
class             0
dtype: int64
```

All columns are populated in this data set.

Exploring counts by each category (of categorical variables)

```
Student Name: Rahul Gupta
Course: 680
Instructor: Fadi Alsaleem

Current date and time :
2021-04-29 22:10:06
```

```
105]: Text(0.5, 1.0, 'Records Counts by Customer Classification')
```

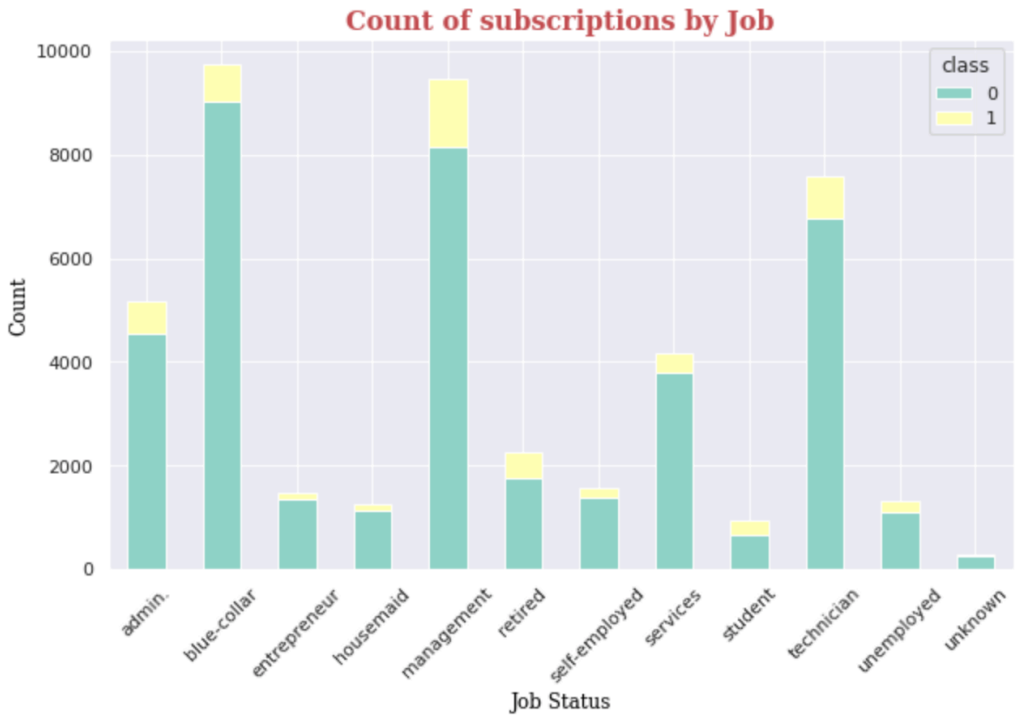


In order to better understand the data, I conducted an initial exploratory data analysis for all variables.

1) Job Status: The top 3 jobs for existing customers are blue-collar, administration and technician.

[107]:
```
sns.set(style="darkgrid", palette="Set3")

plt.rcParams["figure.figsize"] = [10, 6]
bankdf1.plot(kind='bar', stacked=True)

plt.xlabel("Job Status", fontdict=labelfont)
plt.ylabel("Count", fontdict=labelfont)
plt.xticks(rotation=45)
plt.title("Count of subscriptions by Job", fontdict=titlefont , color='r')
```

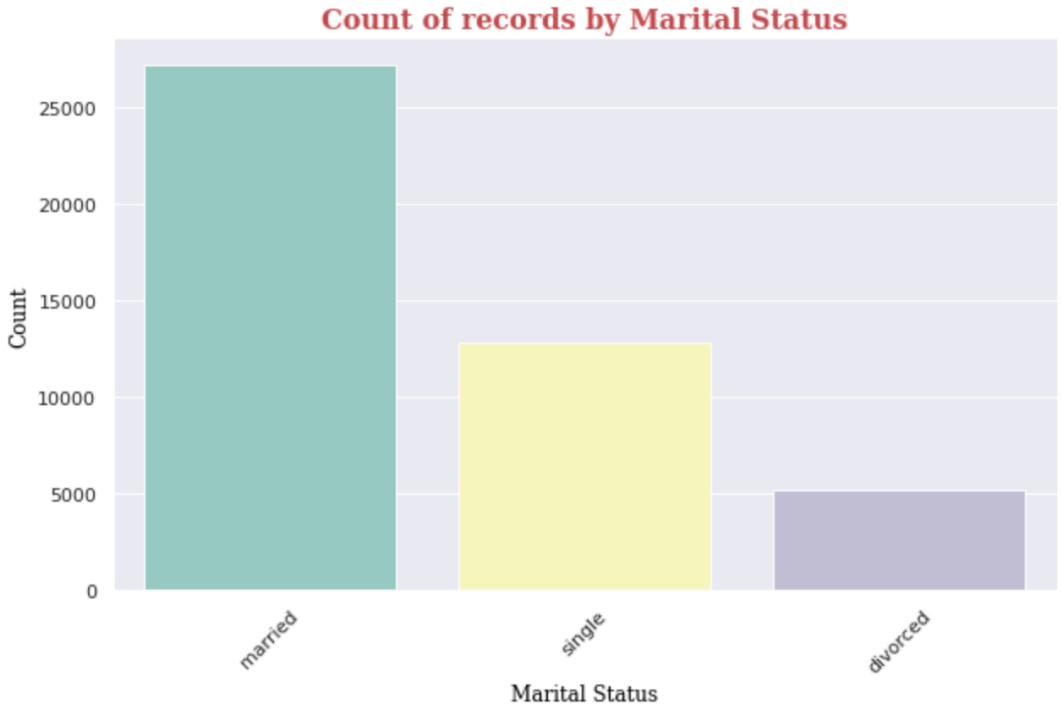
[107]:
```
Text(0.5, 1.0, 'Count of subscriptions by Job')
```



2) Marital Status: Most new clients are married as obviously seen in the bar plot below.

[111]:
```
sns.set(style="darkgrid", palette="Set3")
plt.figure(figsize=(10, 6))
ax = sns.countplot(x="marital_status", data=bankdf)

plt.xlabel("Marital Status", fontdict=labelfont)
plt.ylabel("Count", fontdict=labelfont)

plt.xticks(rotation=45)
plt.title("Count of records by Marital Status", fontdict=titlefont , color='r')
```

[111]:
```
Text(0.5, 1.0, 'Count of records by Marital Status')
```

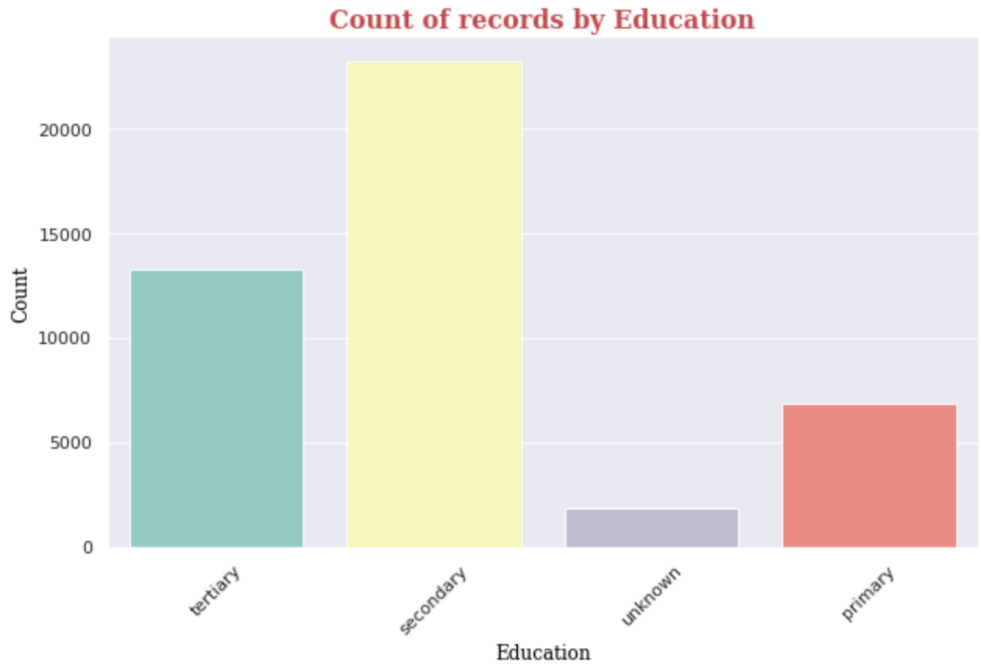


3) Education status – Most of the clients have at least secondary and tertiary education.

Most of the current customers are married as clearly depicted in below bar plot.

[154]:
```
sns.set(style="darkgrid", palette="Set3")
plt.figure(figsize=(10, 6))
ax = sns.countplot(x="education", data=bankdf)

plt.xlabel("Education", fontdict=labelfont)
plt.ylabel("Count", fontdict=labelfont)

plt.xticks(rotation=45)
plt.title("Count of records by Education", fontdict=titlefont , color='r')
```

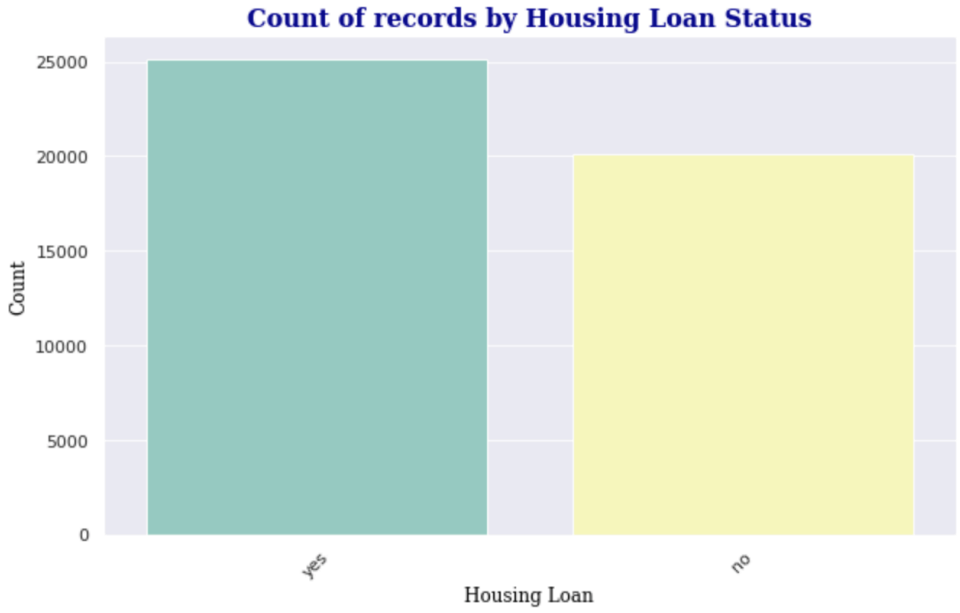
[154]:
```
Text(0.5, 1.0, 'Count of records by Education')
```



4) Housing Loan: Much of the new client base has a housing loan with the bank and seems like consumers who do not have a housing loan with the bank are most likely to subscribe to a term deposit.

[116]:
```
sns.set(style="darkgrid", palette="Set3")
plt.figure(figsize=(10, 6))
ax = sns.countplot(x="housing", data=bankdf)

plt.xlabel("Housing Loan", fontdict=labelfont)
plt.ylabel("Count", fontdict=labelfont)
plt.xticks(rotation=45)
plt.title("Count of records by Housing Loan Status",fontdict=titlefont , color='r' )
```

[116]:
```
Text(0.5, 1.0, 'Count of records by Housing Loan Status')
```



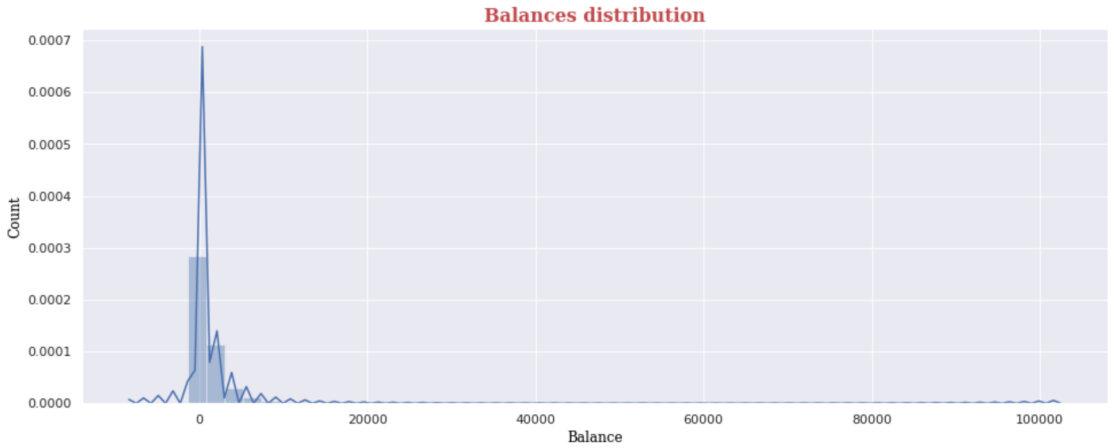
5) Personal Loan: Much of the new client base may not have a personal loan with the bank and looks like clients who do not have a personal loan with the bank are most likely to have a term deposit.

```
[156]: sns.set(style="darkgrid", palette="Set3")
       plt.figure(figsize=(10, 6))
       ax = sns.countplot(x="loan", data=bankdf)

       plt.xlabel("Personal Loan", fontdict=labelfont)
       plt.ylabel("Count", fontdict=labelfont)
       plt.xticks(rotation=45)
       plt.title("Count of records by Personal Loan Status",fontdict=titlefont , color='r' )
```

```
[156]: Text(0.5, 1.0, 'Count of records by Personal Loan Status')
```

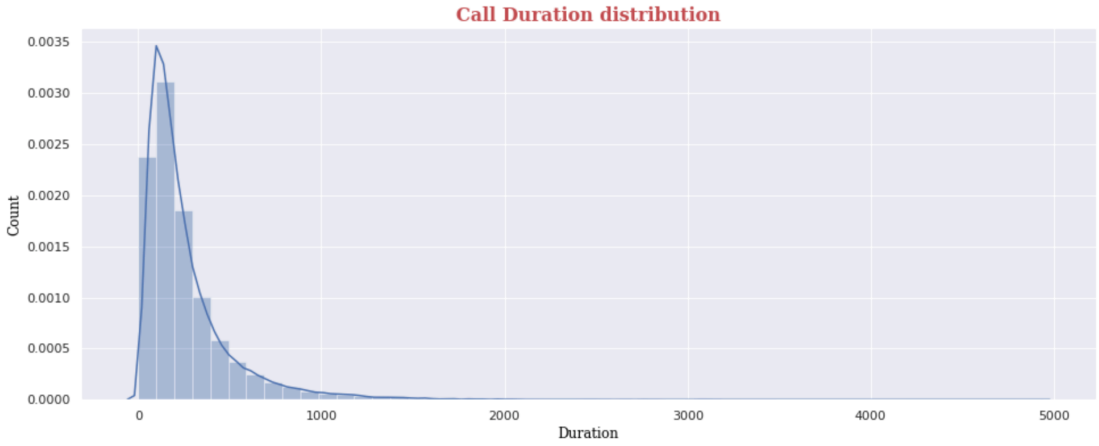
6) Balance: Balance distribution of existing clients banks is rather biased with a long tail towards a higher end of the spectrum. We can assume that most existing customers have a balance of less than 10,000.

```
[159]: sns.set(style="darkgrid", palette="Set3")
       plt.figure(figsize=(16, 6))
       sns.distplot(bankdf.balance, kde=True,color="b")

       plt.xlabel("Balance", fontdict=labelfont)
       plt.ylabel("Count", fontdict=labelfont)
       plt.title("Balances distribution", fontdict=titlefont , color='r' )

[159]: Text(0.5, 1.0, 'Balances distribution')
```



7) Last Call Duration: Most existing consumers have talked for fewer than 500 seconds during the last call campaign. In other words, on average, campaign members would expend less than 10 minutes to figure out whether the client is involved or not in contributing to the term deposit scheme.

```
[160]: sns.set(style="darkgrid", palette="Set3")
       plt.figure(figsize=(16, 6))
       sns.distplot(bankdf.duration, kde=True,color="b")

       plt.xlabel("Duration", fontdict=labelfont)
       plt.ylabel("Count", fontdict=labelfont)
       plt.title("Call Duration distribution", fontdict=titlefont , color='r' )

[160]: Text(0.5, 1.0, 'Call Duration distribution')
```

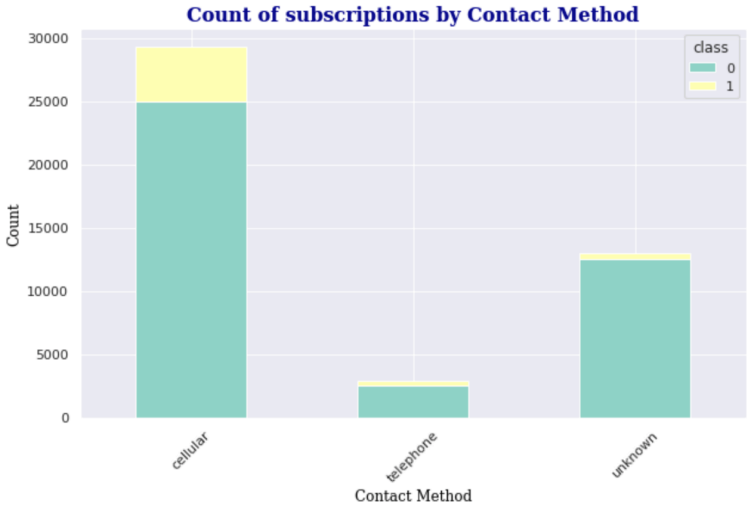


8) Communication Form: Most consumers prefer a mobile phone over a main contact method. It is understandable that mobile phones offer freedom to be carried all the time.

Contact Method

[121]:
```
bankdf6 = bankdf.groupby(['contact', 'class']).size().reset_index().pivot(columns='class', index='contact', values=0)
bankdf6

sns.set(style="darkgrid", palette="Set3")

plt.rcParams["figure.figsize"] = [10, 6]
bankdf6.plot(kind='bar', stacked=True)

plt.xlabel("Contact Method", fontdict=labelfont)
plt.ylabel("Count", fontdict=labelfont)
plt.xticks(rotation=45)
plt.title("Count of subscriptions by Contact Method", fontdict=titlefont , color='r' )
```

[121]: `Text(0.5, 1.0, 'Count of subscriptions by Contact Method')`



9) Customer Classification: The data collection is actually somewhat unbalanced. Many consumers did not adhere to the Term Deposit Scheme. Predictive modeling needs to resolve this disparity in the data collection.

JIRA SharePoint Service Cataloge IDM-SN Ansible Towe GitHub WIKI Se

ngs Help

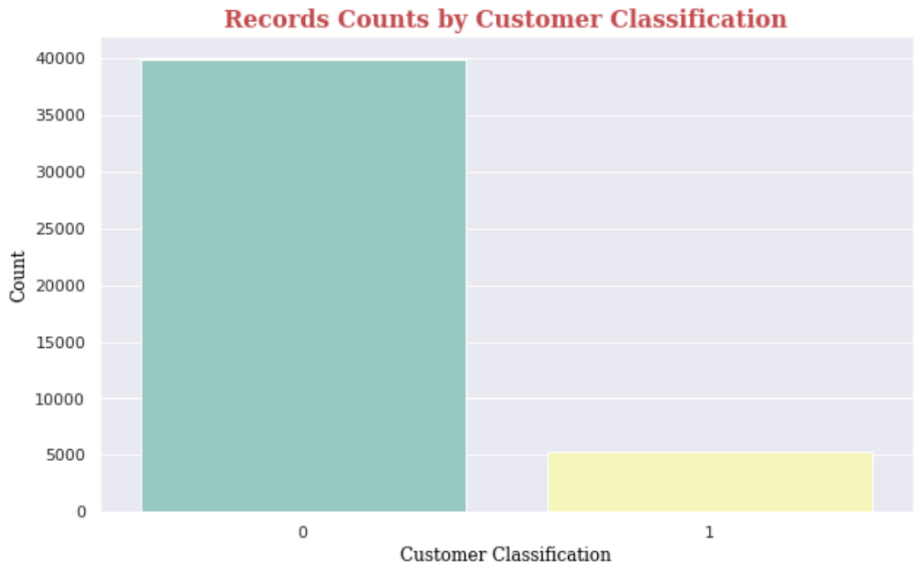
Financial_Analysis_Custor

Code git

[105]:
```
print()
def personal_details():
    name, age = "Rahul Gupta", 680
    address = "Fadi Alsaleem"
    print("Student Name: {}\nCourse: {}\nInstructor: {}".format(name, age, address))

personal_details()

print()
print ("Current date and time : ")
print (now.strftime("%Y-%m-%d %H:%M:%S"))

print()

# Draw count plot
sns.set(style="darkgrid", palette="Set3")
plt.figure(figsize=(10, 6))
ax = sns.countplot(x="class", data=bankdf , palette="Set3")


plt.xlabel("Customer Classification", fontdict=labelfont)
plt.ylabel("Count", fontdict=labelfont)
plt.title("Records Counts by Customer Classification",fontdict=titlefont , color='r' ))
```

```
Student Name: Rahul Gupta
Course: 680
Instructor: Fadi Alsaleem

Current date and time :
2021-04-29 22:10:06
```

[105]:
```
Text(0.5, 1.0, 'Records Counts by Customer Classification')
```



For more detailed exploratory data analysis, please visit my Git Hub portfolio -

Created By: Rahul Gupta

https://iamnrr.github.io/

3. Data Preparation

During the early stages of the exploratory data study, it was found that the target variable – 'class' – carries values 1 & 2. In order to translate it to a binary format, the values shifted to 0 and 1 respectively. Owing to the lack of consistency in the available documents, I have considered that the existing value 1 represents 'NO' in binary format and is thus substituted by the numeric value 0; the value 2 – represents the binary value 'YES' – is replaced by the numeric value 1.

```
[162]: # Changing class variable values to 0 (no) or 1 (yes)

import datetime
now = datetime.datetime.now()

print()
def personal_details():
    name, age = "Rahul Gupta", 680
    address = "Fadi Alsaleem"
    print("Student Name: {}\nCourse: {}\nInstructor: {}".format(name, age, address))

personal_details()

print()
print ("Current date and time : ")
print (now.strftime("%Y-%m-%d %H:%M:%S"))


bankdf.loc[bankdf['class'] == 1, 'class'] = 0    # Changing all NO to 0
bankdf.loc[bankdf['class'] == 2, 'class'] = 1    # Changing all YES to 1
```

Feature Engineering

The input features actually include a lot of categorical variables. Categorical variables must be encoded to match the predictive models. Here, I used pandas – getdummies() function to create dummy variables for all of the categorical variables.

Feature Engineering

The input features actually include a lot of categorical variables. Categorical variables must be encoded to match the predictive models. Here, I used pandas – getdummies() function to create dummy variables for all of the categorical variables.

```
bankdf_wdummy = pd.get_dummies(bankdf)
bankdf_wdummy.head()
```

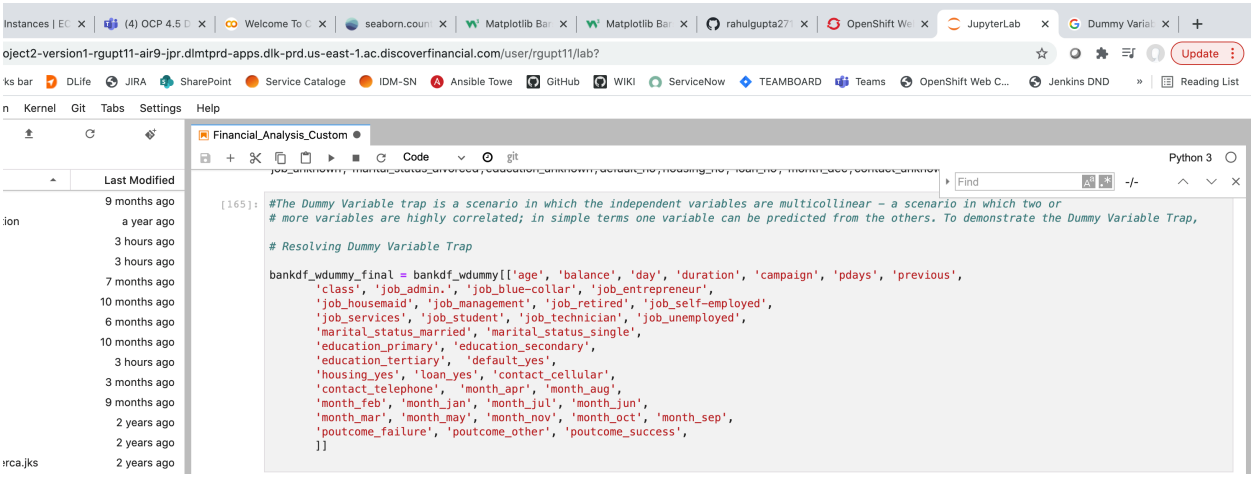
	age	balance	day	duration	campaign	pdays	previous	class	job_admin.	job_blue-collar	...	month_jun	month_mar	month_may	month_nov	month_oct	month_sep	poutcome_
0	58	2143	5	261	1	-1	0	0	0	0	...	0	0	1	0	0	0	
1	44	29	5	151	1	-1	0	0	0	0	...	0	0	1	0	0	0	
2	33	2	5	76	1	-1	0	0	0	0	...	0	0	1	0	0	0	
3	47	1506	5	92	1	-1	0	0	0	1	...	0	0	1	0	0	0	
4	33	1	5	198	1	-1	0	0	0	0	...	0	0	1	0	0	0	

5 rows × 52 columns

```
1 bank
2 bank
```

To stop dummy variable pits, one dummy variable for each categorical variable should be omitted. The following vector columns are removed:

'job_unknown', 'marital_status_divorced','education_unknown','default_no','housing_no', 'loan_no', 'month_dec','contact_unknown', 'poutcome_unknown'



4. Predictive Modelling

The predictive problem (predicting whether or not the consumer subscribes) that we have at our hands is basically a binary classification problem. In comparison, the data collection also comprises the aim classification, which means that this is a supervised learning classification. Thanks to the existence of this simple classification, which we can use to train the model that we are about to build, the benefit of supervised learning also applies here. To solve this problem of supervised binary classification, I want to try Random Forest Classifier.

Why Random Forest?

The Random Forest is a group of decision trees. Since then, I've heard about Random Forest, it's been one of my favorite models to think, with the awareness that it does not suit all the problems. Yet I see the following advantages:

1) First of all, our data has unbalanced groups and Random Forest treats unbalanced data well by minimizing the total error rate.
2) At each random break of the decision tree, the algorithm chooses a random subset of feat ures, which decreases the similarity between the chosen features and increases the varian ce of the model.
3) Random Forest is not influenced by the existence of outliers and non-linear data.

In the study of the exploratory results, we found that the 'balance' variable has outliers. The existence of outliers is negated by picking the Random Tree.

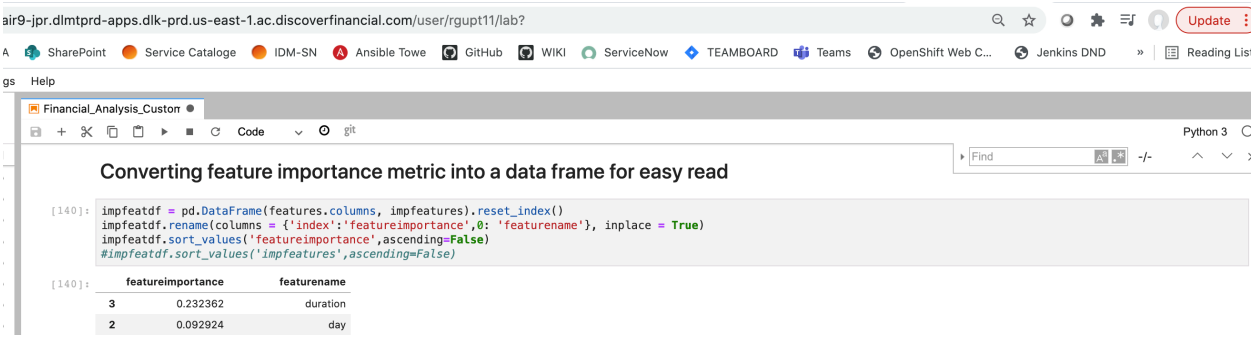
Feature Selection

In order to increase the precision and consistency of our forecasts, it is appropriate to pick essential variables that have more say in predictions. This tends to minimize errors and increase noise due to the inclusion of noise in the data collection.

I used the ExtraTreesClassifier algorithm to pick relevant features from our current data collection.

```
[166]: #Feature selection is the process of reducing the number of input variables when developing a predictive model. It is desirable to reduce the number of input variables to both reduce the computational
       # in some cases, to improve the performance of the model.

       # feature selection

       import datetime
       now = datetime.datetime.now()

       print()
       def personal_details():
           name, age = "Rahul Gupta", 680
           address = "Fadi Alsaleem"
           print("Student Name: {}\nCourse: {}\nInstructor: {}".format(name, age, address))

       personal_details()

       print()
       print ("Current date and time : ")
       print (now.strftime("%Y-%m-%d %H:%M:%S"))

       print()


       fsmodel = ExtraTreesClassifier(n_estimators=10)
       fsmodel.fit(features, target)
       impfeatures = fsmodel.feature_importances_
       print(impfeatures)
       print(features.columns)

       Student Name: Rahul Gupta
       Course: 680
       Instructor: Fadi Alsaleem

       Current date and time :
       2021-04-29 23:05:02

       [0.09101072 0.09005224 0.09220293 0.22743648 0.05816953 0.02839449
        0.0229781  0.00902757 0.00747141 0.00466997 0.00351129 0.01093691
        0.00649563 0.00538037 0.00740775 0.00589276 0.01158327 0.00587057
        0.01489191 0.01034862 0.00776367 0.01304631 0.01030331 0.00295984
        0.0233738  0.01117721 0.01651917 0.00545736 0.0093652  0.0088711
        0.00773229 0.00475624 0.00857069 0.01074771 0.01433363 0.00804685
        0.00754935 0.0119635  0.01215943 0.00947829 0.00734373 0.07474882]
       Index(['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous',
              'job_admin.', 'job_blue-collar', 'job_entrepreneur', 'job_housemaid',
              'job_management', 'job_retired', 'job_self-employed', 'job_services',
              'job_student', 'job_technician', 'job_unemployed',
              'marital_status_married', 'marital_status_single', 'education_primary',
              'education_secondary', 'education_tertiary', 'default_yes',
              'housing_yes', 'loan_yes', 'contact_cellular', 'contact_telephone',
              'month_apr', 'month_aug', 'month_feb', 'month_jan', 'month_jul',
              'month_jun', 'month_mar', 'month_may', 'month_nov', 'month_oct',
              'month_sep', 'poutcome_failure', 'poutcome_other', 'poutcome_success'],
             dtype='object')
```

Converting feature importance metric into a data frame for easy read.

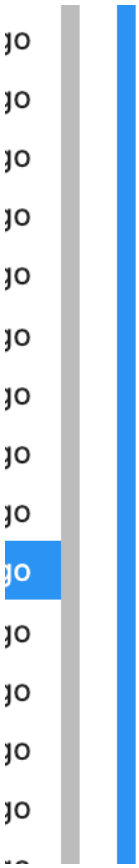


Below is the table that shows each variable and its importance. The most important variable among that has considerable say in the outcome of predicting subscription is “duration” of call.

[140]:

	featureimportance	featurename
3	0.232362	duration
2	0.092924	day
1	0.090325	balance
0	0.089757	age
41	0.072994	poutcome_success
4	0.059659	campaign
5	0.029943	pdays
6	0.024496	previous
24	0.023394	housing_yes
26	0.016318	contact_cellular
18	0.014321	marital_status_married
34	0.013809	month_mar
21	0.013631	education_secondary
37	0.012727	month_oct
33	0.012183	month_jun
22	0.011373	education_tertiary
19	0.011060	marital_status_single
25	0.011026	loan_yes
16	0.010801	job_technician
38	0.009903	month_sep
39	0.009812	poutcome_failure
11	0.009667	job_management
7	0.009083	job_admin.
35	0.008942	month_may
29	0.008775	month_aug
20	0.008688	education_primary
28	0.008051	month_apr
36	0.007359	month_nov
30	0.007160	month_feb
12	0.006952	job_retired

Saving failed

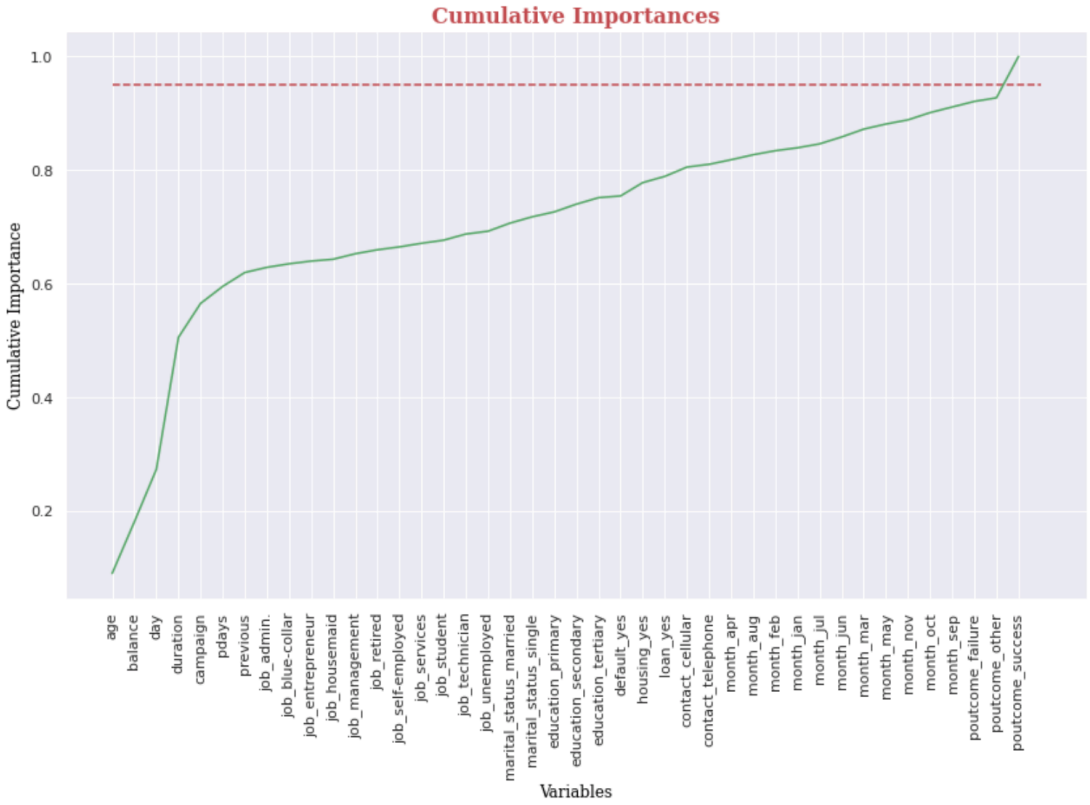


30	0.007160	month_feb
12	0.006952	job_retired
32	0.006916	month_jul
14	0.006351	job_services
40	0.006335	poutcome_other
8	0.006234	job_blue-collar
15	0.005475	job_student
13	0.005106	job_self-employed
31	0.005085	month_jan
17	0.004903	job_unemployed
27	0.004901	contact_telephone
9	0.004818	job_entrepreneur
10	0.003269	job_housemaid
23	0.003115	default_yes

From the plot below, which indicates the combined value of the features in order to hit 95% of the importance, we need to consider nearly all the features. Therefore, no further selection of features is needed for this model, we need to consider all possible variables.

```
plt.xlabel('Variables', fontdict=labelfont)
plt.ylabel('Cumulative Importance', fontdict=labelfont)
plt.title('Cumulative Importances', fontdict=titlefont , color='r' )

Student Name: Rahul Gupta
Course: 680
Instructor: Fadi Alsaleem

Current date and time :
2021-04-29 23:11:47
```

]: Text(0.5, 1.0, 'Cumulative Importances')



Model Training

Generated a base model using the Random Forest Classifier with the parameters below and trained the dataset.

Model Tuning

Using GridSearchCV which tries every combination of parameter given in the parameter grid, we can find out the best parameters. Though we might get better accuracy with this approach, it could lot of time to search through all parameters. So, we need to be cautious consider the tradeoff between performance and accuracy before defining the parameters for GridSearchCV.

```
import datetime
now = datetime.datetime.now()

print()
def personal_details():
    name, age = "Rahul Gupta", 680
    address = "Fadi Alsaleem"
    print("Student Name: {}\nCourse: {}\nInstructor: {}".format(name, age, address))

personal_details()

print()
print ("Current date and time : ")
print (now.strftime("%Y-%m-%d %H:%M:%S"))

# Create the parameter grid based on the results of random search
param_grid = {
    'n_estimators': [100, 250, 500, 1000, 2000],
    "criterion": ["gini", "entropy"],
     }

# Create a default rf model
rf = RandomForestClassifier(random_state=1,           # for consistent results
                                    oob_score=True,           # OOB Score to get performance
                                    bootstrap=True,
                                    n_jobs=-1,                # for using all cores
                                    class_weight="balanced" )
# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                           cv = 3, n_jobs = -1, verbose = 2)
```

```
Student Name: Rahul Gupta
Course: 680
Instructor: Fadi Alsaleem

Current date and time :
2021-04-29 23:17:15
```

5. Evaluation

Accuracy cannot be used as a criterion to measure the efficiency of our predictive model, as our existing data collection has unbalanced groups. So, we need to remember other metrics such as the uncertainty matrix, the classification report and the ROC curve.

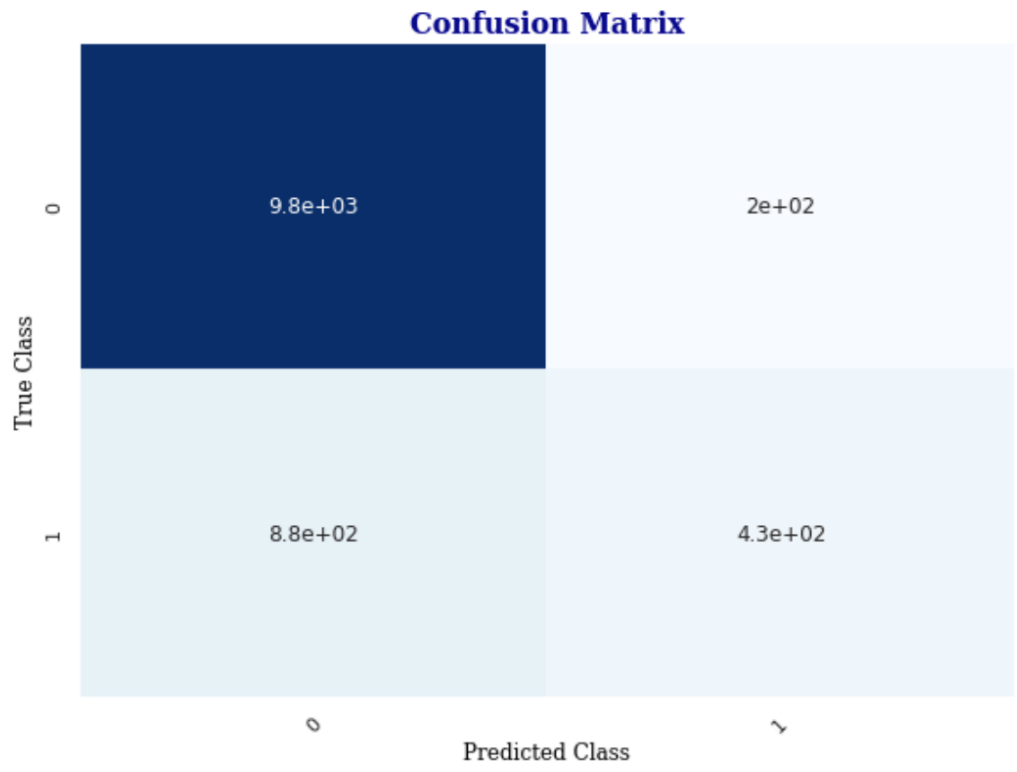
Below are the resultant metrics from evaluating our random classifier classification model:

Confusion Matrix

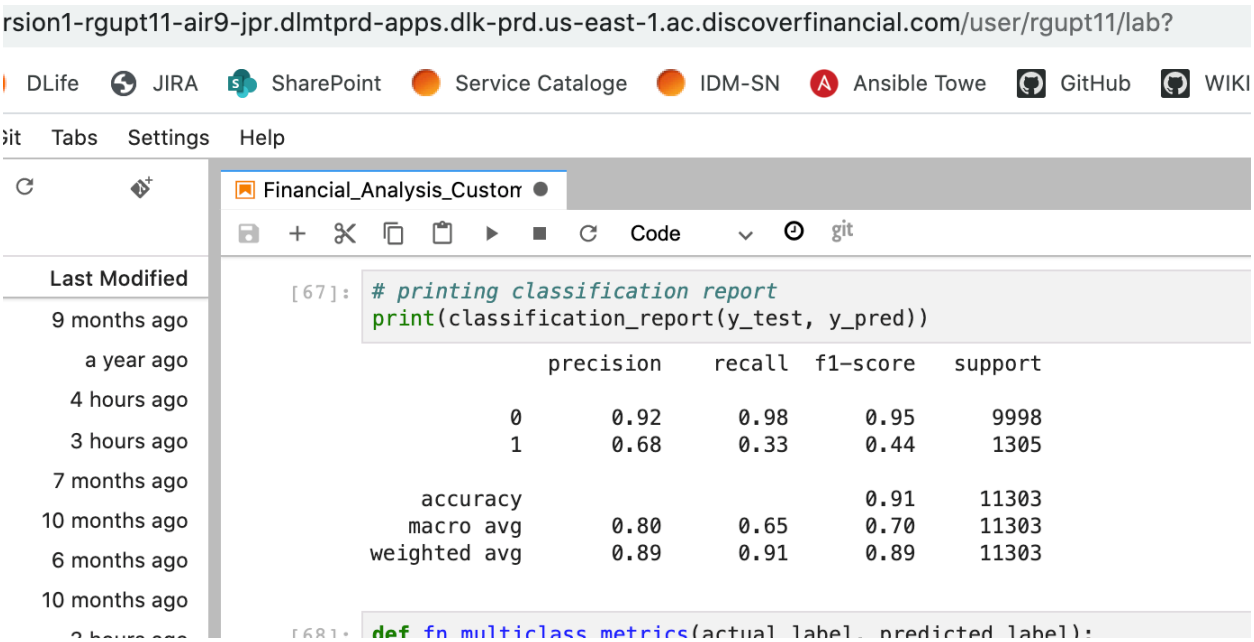
[65]:
```
# Create confusion matrix
matrix = confusion_matrix(y_test, y_pred)
matrix
```

[65]:
```
array([[9800,  198],
       [ 875,  430]])
```

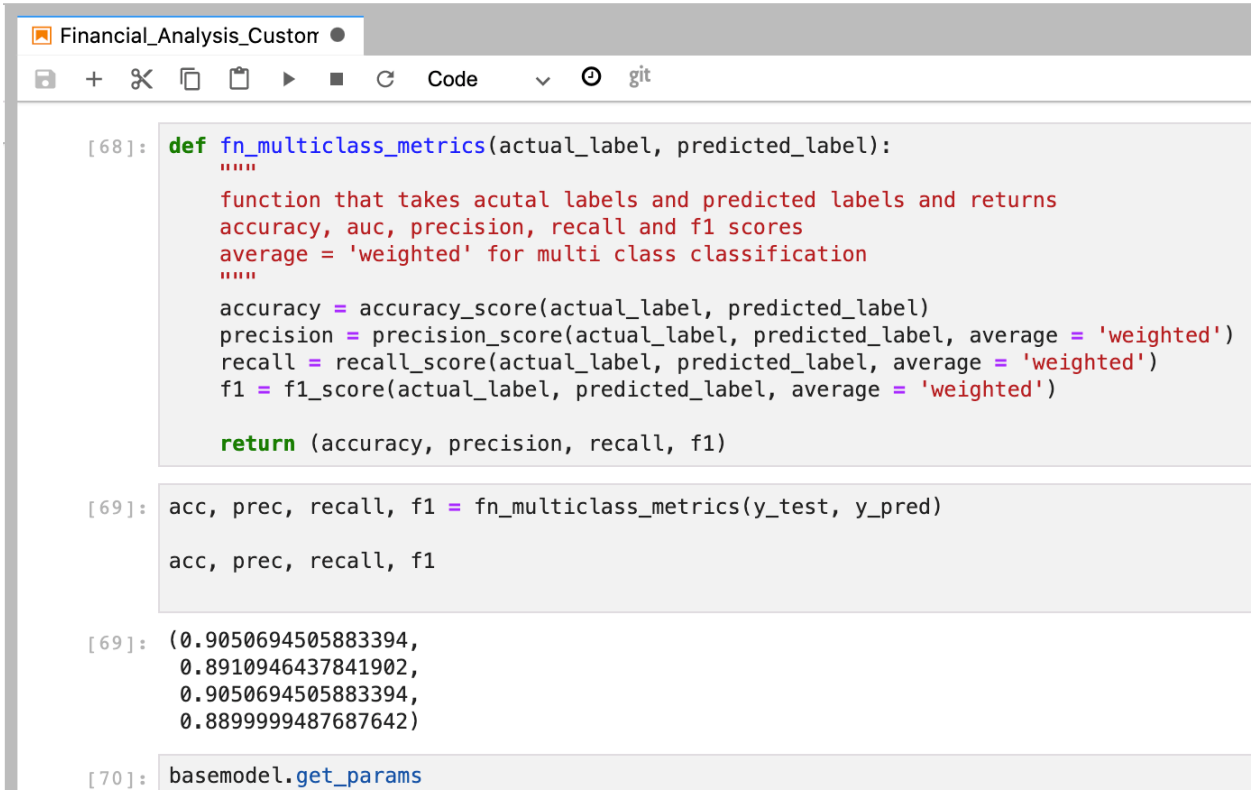
[66]:
```
# Create pandas dataframe
dataframe = pd.DataFrame(matrix) #, index=class_names, columns=class_names)


# Create heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(dataframe, annot=True, cbar=None, cmap="Blues")
plt.title("Confusion Matrix")
plt.tight_layout()
plt.xlabel("Predicted Class", fontdict=labelfont)
plt.ylabel("True Class", fontdict=labelfont)

plt.xticks(rotation=45)
plt.title("Confusion Matrix" , fontdict=titlefont , color='r' )
plt.show()
```



Classification Report



Accuracy, Precision, Recall, F1 Score

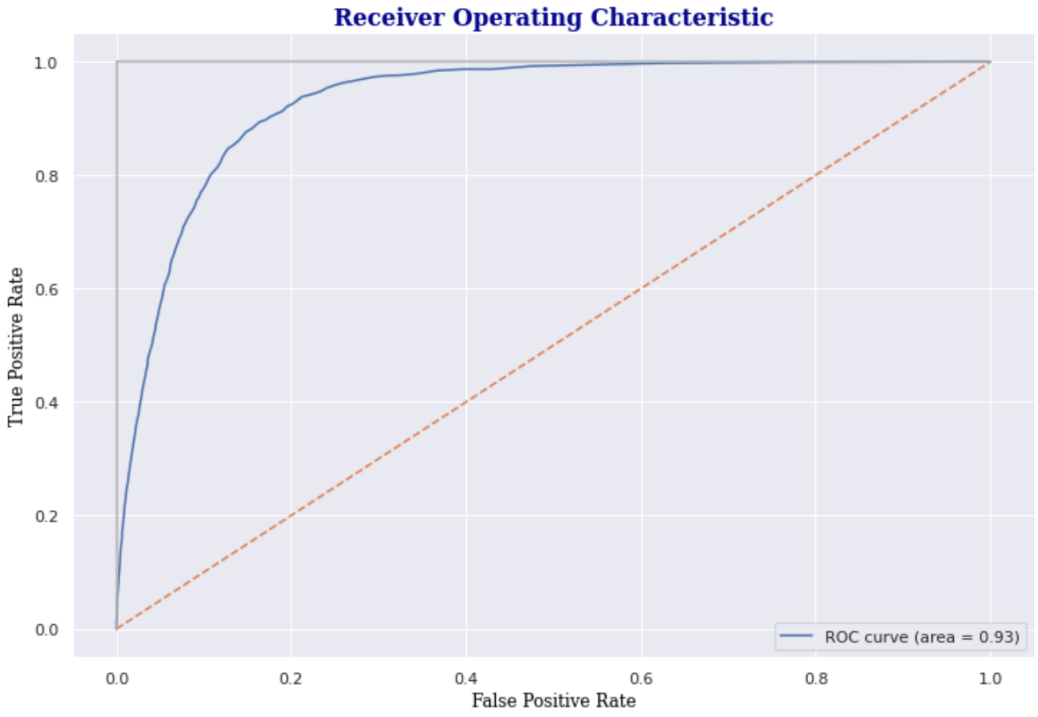


Precision informs us whether or not the consumers we expected were those who might contribute to the term service. Metric Recall shows us how many consumers who subscribed were expected by our model as others who could subscribe to it.

Our above evaluation shows that we have a Precision of 89% and Recall factor of 90%.

ROC Curve:

```
[74]: # Plot ROC curve
plotroc(false_positive_rate, true_positive_rate)
```



The above Receiver Operational Characteristics plot reveals that the base model was generated with an AUC value of 0.93 per cent and shows that the model distinguishes between the two groups well.

6. Model Tuning

To further tune the model, I built the grid of the parameter below and used GridSearchCV, which attempts any parameter combination defined in the grid parameter and finds the best parameter. While we could get better accuracy with this method, it might take a lot of time to scan for all the parameters. We also need to be careful to understand the tradeoff between efficiency and accuracy before specifying the criteria for GridSearchCV.

Model Tuning

Using GridSearchCV which tries every combination of parameter given in the parameter grid, we can find out the best parameters. Though we might get better accuracy with this approach, it could lot of time to search through all parameters. So, we need to be cautious consider the tradeoff between performance and accuracy before defining the parameters for GridSearchCV.

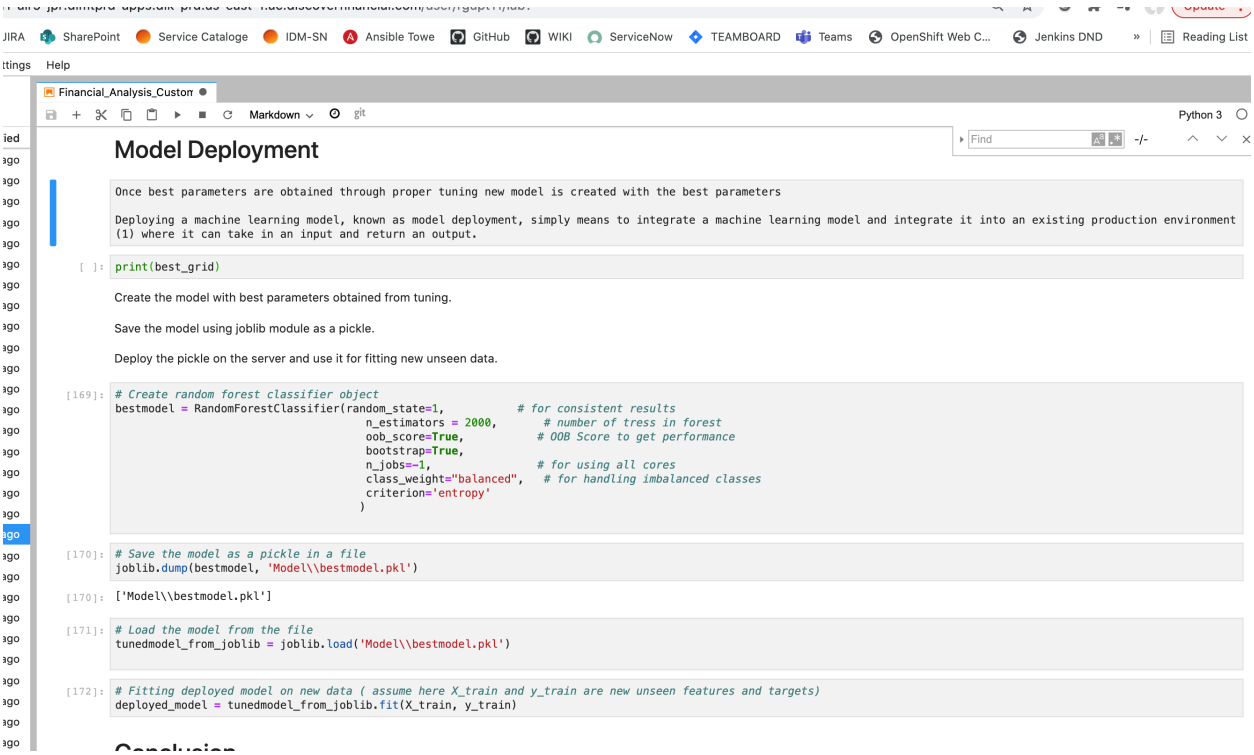
```
import datetime
now = datetime.datetime.now()

print()
def personal_details():
    name, age = "Rahul Gupta", 680
    address = "Fadi Alsaleem"
    print("Student Name: {}\nCourse: {}\nInstructor: {}".format(name, age, address))

personal_details()

print()
print ("Current date and time : ")
print (now.strftime("%Y-%m-%d %H:%M:%S"))

# Create the parameter grid based on the results of random search
param_grid = {
    'n_estimators': [100, 250, 500, 1000, 2000],
    "criterion": ["gini", "entropy"],
     }

# Create a default rf model
rf = RandomForestClassifier(random_state=1,           # for consistent results
                                    oob_score=True,           # OOB Score to get performance
                                    bootstrap=True,
                                    n_jobs=-1,                # for using all cores
                                    class_weight="balanced" )
# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                           cv = 3, n_jobs = -1, verbose = 2)
```

```
Student Name: Rahul Gupta
Course: 680
Instructor: Fadi Alsaleem

Current date and time :
2021-04-29 23:17:15
```

For the specified grid parameter, GridSearchCV has defined the following parameters as the best estimators for the Random Forest Classifier model.

7. Deployment

For the specified grid parameter, GridSearchCV has defined the following parameters as the best estimators for the Random Forest Classifier model.



8. Conclusion

To conclude using the Random Forest Classifier algorithm and hyper tuning it further using either GridSearchCV or RandomSearchCV, we should be able to determine whether or not the consumer would subscribe to the term deposit service in an effective manner. Next step is to see how the efficiency of the model can be further enhanced by using Deep Learning and Neural Networks.

9. Assumptions

Due to lack of clarity in the code book and misrepresentation of the binary variable – class with values 1 & 2, I had to make below assumptions:

1) The value ‘1’ represents binary ‘no’ and hence replaced it with numeric 0. 2) The value ‘2’ represents binary ‘yes and hence replaced it with numeric 1.

10. Techniques

Used the below modules in python to accomplish this supervised classification task.

- pandas
- numpy
- sklearn
- matplotlib
- seaborn
- joblib
- os

11. References

1. Data source link - https://www.openml.org/d/1461
2. Chris Albon, April 2018, “Python Machine Learning Cookbook”, ISBN 9781491989388.
3. Jake Huneycutt, May 18 2018, "Implementing a Random Forest Classification Model in Python", https://medium.com/@hjhuney/implementing-a-random-forest-classification-model-in-python-583891c99652
4. Will Koehrsen, Jan 9 2018, "Hyperparameter Tuning the Random Forest in Python", https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74
5. Project Reference: https://iamnrr.github.io/
6. Mohammed Sunasra, Nov 11, 2017, "Performance Metrics for Classification problems in Machine Learning", https://medium.com/thalus-ai/performance-metrics-for-classification-problems-in-machine-learning-part-i-b085d432082b
7. Random Forest Algo - https://towardsdatascience.com/understanding-random-forest-58381e0602d2

12. Appendix

#. Understanding Random Forest

Classification is a large part of machine learning; we want to know what class (a.k.a. group) an observation belongs to. The ability to accurately distinguish findings is highly useful for a variety of market purposes, such as predicting whether a specific customer will purchase a commodity or estimating whether a given loan will default or not.

Classification algorithms in data science include logistic regression, help vector machine, naive Bayes classifier, and decision trees. However, the random forest classifier is at the top of the classifier hierarchy (there is also the random forest regressor but that is a topic for another day).

The random forest, as the name suggests, is made up of a large number of individual decision trees that work together as an ensemble. Each individual tree in the random forest produces a class prediction, and the class with the most votes becomes the prediction of our model.

Random forests are one of my absolute favorites. Coming from the world of finance and investing, the holy grail was often to create a portfolio of uncorrelated models, each with a positive estimated return, and then combine them to earn huge alpha (alpha = market beating returns). It's a lot better said than done!