University of Delaware
BUAD625
Capstone Project
Nam
e:- Sudarshan Kumar

Sunil Mathur

Sally Cao

Amanda Morley

Drew McKnight

Project Title : Bank Marketing

The data is related to direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to assess if the product (bank term deposit) would be subscribed (or not).

The classification goal is to predict if the client will subscribe to a term deposit (variable y).

Attribute information

For more information, visit https://www.openml.org/d/1461.

Input variables:

Bank Client Data:

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")

- output variable (desired target):

17 - y - has the client subscribed a term deposit? (binary: "yes","no")

2 = yes; 1 = no (We are changing this variable so that 1 simply means subscribed)

EDA Observations:

- The Bank phone marketing campaign data set has 45,211 rows and 17 columns including the target or Y variable.
- The dataset is very clean with no null values found
- Some key features that may be potentially important:
  - **Age**: Mean age is 41 years. Range is from 18 - 95 , mid 50% quartile is 33-48
    - **Hypothesis**: Those under 30 and greater than 60 may have more money to be able to open a deposit account
  - **Average Balance**: Mean balance is 1.3 K Range is -8k to 102K
    - **Hypothesis**: Those with more average balance will likely have money to open a deposit account
  - **Housing Loan/Personal Loan**: Either of these increases the liability.
    - **Hypothesis**:People with loans may not have enough money to open a deposit account
  - **Education**: People with higher education are likely to be high earners and may have enough spare cash to open a deposit account.
  - **Job**: People in management / technical jobs versus students / entrepreneurs / self-employed may have more money to open a deposit account. Retired folks may be good candidates as well.
    - Reasons we find interesting: It has the most categories in a feature compared to the other features and it is something we can leverage in focusing on certain jobs or removing ones that do not seem significanto
  - **Marital Status**: Singles may have more money versus married with family responsibilities
  - **Contact**: Cellular seems to be resulting in higher conversion rates
    - **Hypothesis**: Those with cellular phones may have more disposable income or calling someone on a cellular phone yields a better chance of response.
  - **Credit Default**: people with credit default may not be the best candidates to open a deposit accountoPrevious campaigns - may have a role in someone deciding to open a deposit account
- **Potential Issues**: Call Duration appears to be highly correlated. However, it is unknown till the end of the call. We are unsure if we should keep or drop this variable as it is highly correlated, however we won't know the result unless a marketing call is accepted.
- **Potential Outliers**: In the box plot for the feature 'previous' (which is the number oftimes the client was contacted before the campaign) has an outlier where there is one client who was contacted 273 times would be a feature; we would want to remove outliers for so that our dataset would not be skewed if we wanted to keep the'previous' feature for our data processing.

Recommended variables to focus on:

- Duration: highly correlated but we are unsure if this is a fair variable to use given it is necessary for a contact to be contacted definitely. If we find that itis a valuable variable this could be something that we consider to keep the customer on the phone longer as that seems to lead to more successful conversions.
- Job type is also an interesting variable to us, we see that students and retiredworkers are highly correlated so we would want to further investigate this. Perhaps the students are more educated and willing to subscribe and the retirees have more disposable income.
- Marital type is an interesting variable as those that are single seem to be highly correlated, perhaps those that are married will invest in something else like college funds.
- The type of contact also seems to make a difference, those that had cellular phones were more likely to subscribe and this could be for reasons like those that have cellular phones have more discretionary income or it is easier to contact them.
- Previous outcome success is also an interesting variable and could be used topinpoint who we make our calls to. If we call those we had previous success with first this might increase our marketing power.

Full Pipeline and Model Deployment Summary:

- We created 7 models with accuracies ranging from ~85-90%, the highest being from the Random Forest.
  - Decision Tree
  - Random Forest
  - SVM
  - XGBoost
  - Naive Bayes
- Features we used for the models:
  - age
  - balance
  - day
  - duration
  - pdays
  - housing
  - month
  - poutcome
  - contact
  - marital

Conclusion

Baseline Accuracy for predicting succesful Customer subscription using feature "poutcome" (Previous campaign success) is 88.3 %. Our best performing model is "Random Forest" with 90.63% accuracy. Based on the random forest, duration seems to be the strongest predictor for whether a customer will subscribe or not with a feature importance score of ~34%. This could be due to the fact that customers who are willing to stay longer on the phone with a bank representative have a higher chance of being persuaded to subscribe to a bank deposit.

Import Library

In [1]:
```
# import packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
plt.style.use('seaborn')  # change the default style
cbold='\033[01m'
cEnd = '\033[0m'
```

READ DATA

In [2]:
```
# read csv data into pandas dataframe
df = pd.read_csv('projectdataset-1.csv')
```

In [3]:
```
# basic shape, data type, null values: dataset looks pretty good.
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   age        45211 non-null  int64
 1   job        45211 non-null  object
 2   marital    45211 non-null  object
 3   education  45211 non-null  object
 4   default    45211 non-null  object
 5   balance    45211 non-null  int64
 6   housing    45211 non-null  object
 7   loan       45211 non-null  object
 8   contact    45211 non-null  object
 9   day        45211 non-null  int64
 10  month      45211 non-null  object
 11  duration   45211 non-null  int64
 12  campaign   45211 non-null  int64
 13  pdays      45211 non-null  int64
 14  previous   45211 non-null  int64
 15  poutcome   45211 non-null  object
 16  Class      45211 non-null  int64
dtypes: int64(8), object(9)
memory usage: 5.9+ MB
```

In [4]:
```
# first 5 lines of data
df.head()
```

Out[4]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	may	261	1	-1	0
1	44	technician	single	secondary	no	29	yes	no	unknown	5	may	151	1	-1	0
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	76	1	-1	0
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	may	92	1	-1	0
4	33	unknown	single	unknown	no	1	no	no	unknown	5	may	198	1	-1	0

In [5]:
```
# Prepare the data by separating X and y
# dropping Y variable

# axis = 1 below means dropping by columns, 0 means by rows
df.Class.replace((1,2),(0,1), inplace=True) # 0 - Not Subscribed, 1 - Subscribed

X = df.drop(['Class'], axis=1)
y = df['Class']
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 16 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   age        45211 non-null  int64
 1   job        45211 non-null  object
 2   marital    45211 non-null  object
 3   education  45211 non-null  object
 4   default    45211 non-null  object
 5   balance    45211 non-null  int64
 6   housing    45211 non-null  object
 7   loan       45211 non-null  object
 8   contact    45211 non-null  object
 9   day        45211 non-null  int64
 10  month      45211 non-null  object
 11  duration   45211 non-null  int64
 12  campaign   45211 non-null  int64
 13  pdays      45211 non-null  int64
 14  previous   45211 non-null  int64
 15  poutcome   45211 non-null  object
dtypes: int64(7), object(9)
memory usage: 5.5+ MB
```

In [6]:
```
# basic stats
df.describe(include='all')
```

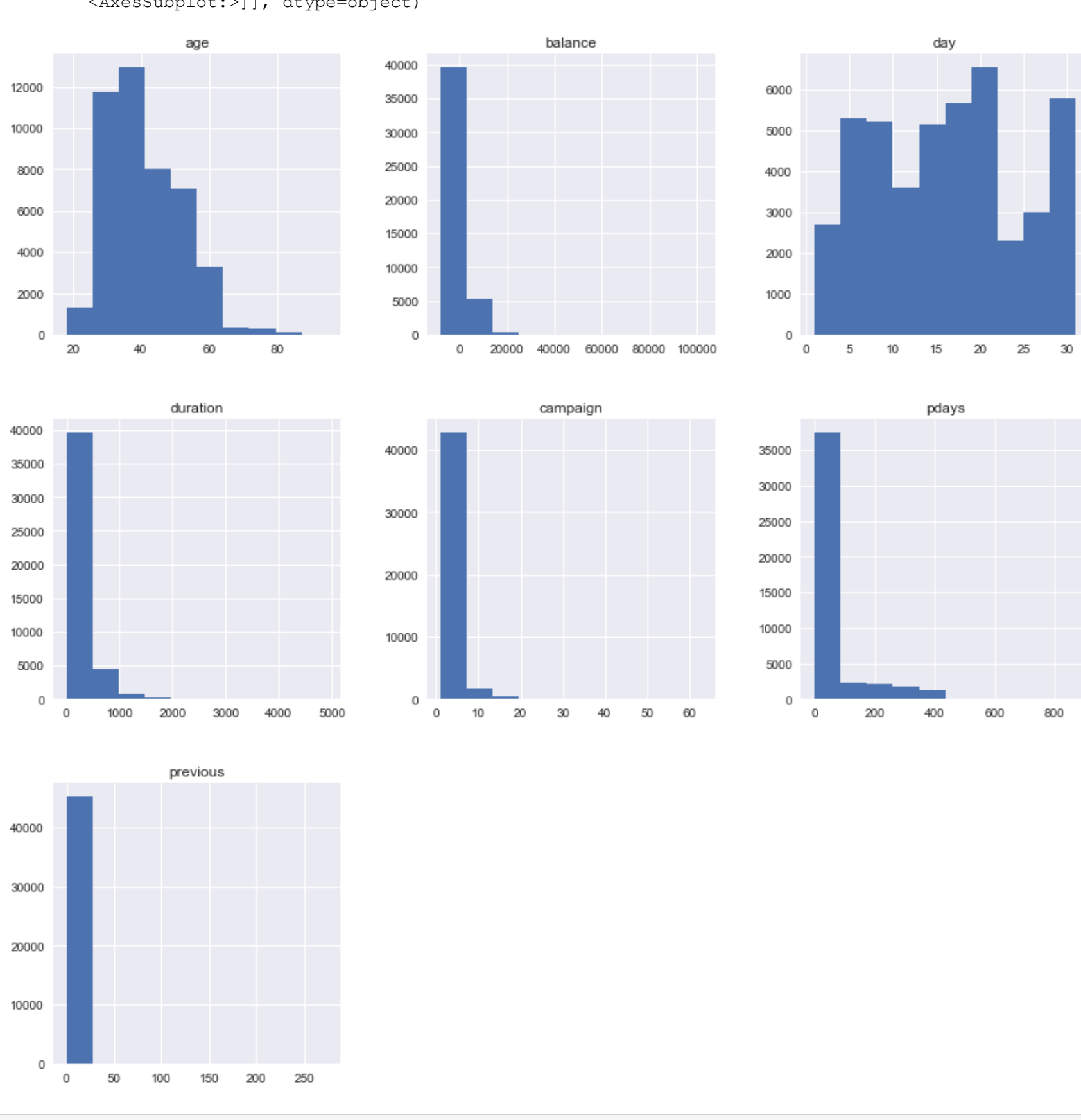
Out[6]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration
count	45211.000000	45211	45211	45211	45211	45211.000000	45211	45211	45211	45211.000000	45211	45211.000000
unique	NaN	12	3	4	2	NaN	2	2	3	NaN	12	NaN
top	NaN	blue-collar	married	secondary	no	NaN	yes	no	cellular	NaN	may	NaN
freq	NaN	9732	27214	23202	44396	NaN	25130	37967	29285	NaN	13766	NaN
mean	40.936210	NaN	NaN	NaN	NaN	1362.272058	NaN	NaN	NaN	15.806419	NaN	258.163080
std	10.618762	NaN	NaN	NaN	NaN	3044.765829	NaN	NaN	NaN	8.322476	NaN	257.527812
min	18.000000	NaN	NaN	NaN	NaN	-8019.000000	NaN	NaN	NaN	1.000000	NaN	0.000000
25%	33.000000	NaN	NaN	NaN	NaN	72.000000	NaN	NaN	NaN	8.000000	NaN	103.000000
50%	39.000000	NaN	NaN	NaN	NaN	448.000000	NaN	NaN	NaN	16.000000	NaN	180.000000
75%	48.000000	NaN	NaN	NaN	NaN	1428.000000	NaN	NaN	NaN	21.000000	NaN	319.000000
max	95.000000	NaN	NaN	NaN	NaN	102127.000000	NaN	NaN	NaN	31.000000	NaN	4918.000000

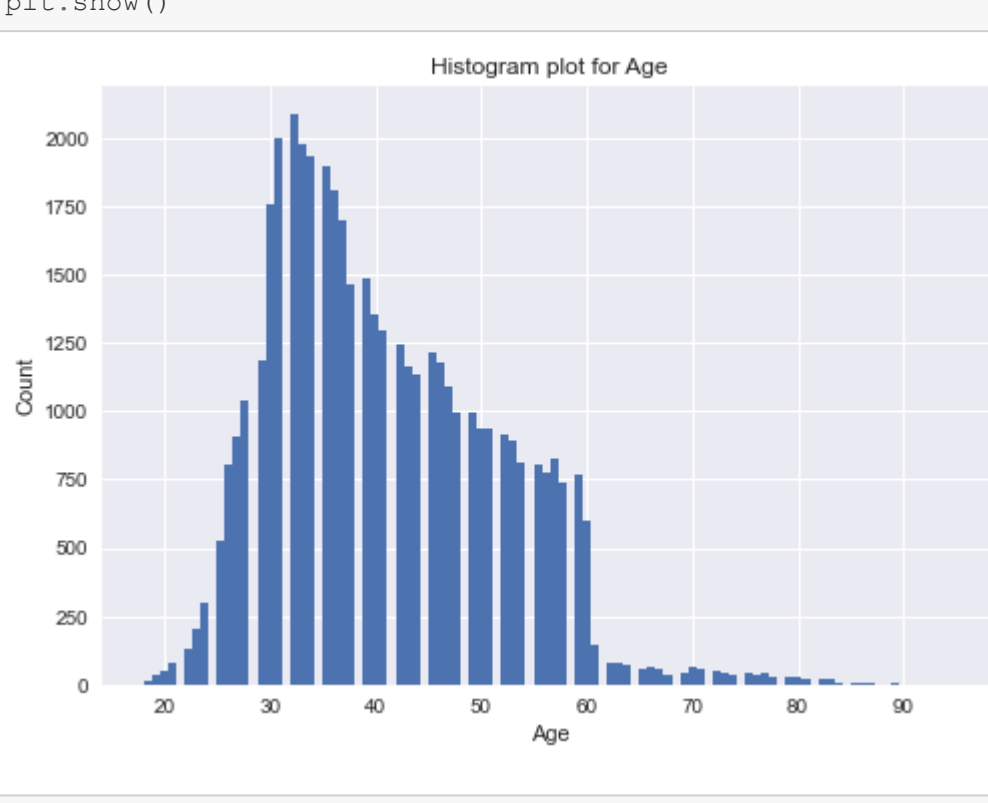
EDA

In [7]:
```
# histograms for all numerical features
X.hist(figsize=(15,15))
```

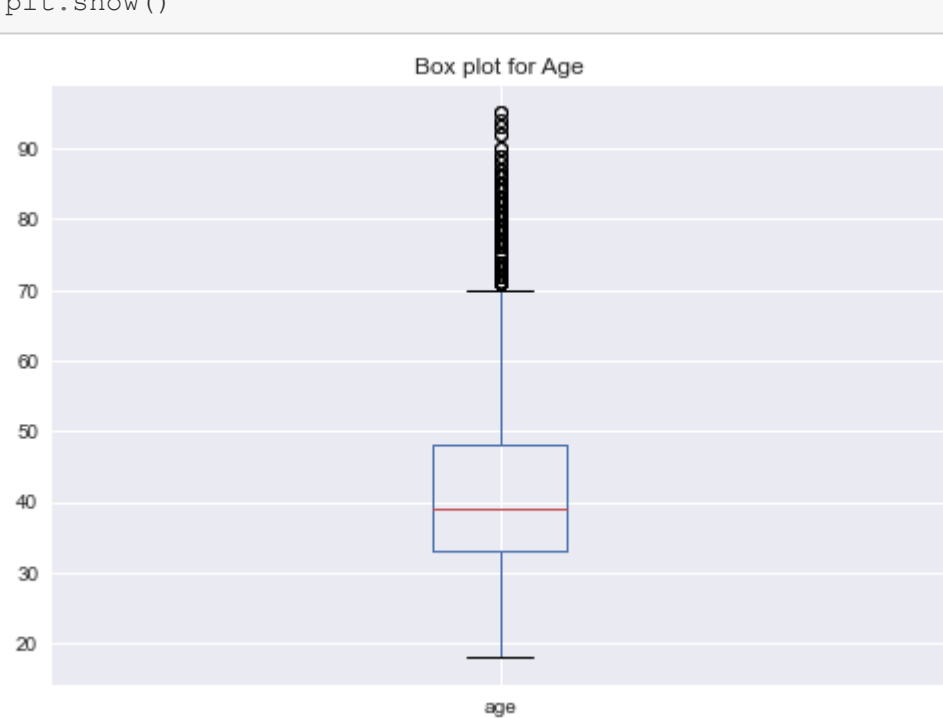
Out[7]:
```
array([[<AxesSubplot:title={'center':'age'}>,
        <AxesSubplot:title={'center':'balance'}>,
        <AxesSubplot:title={'center':'day'}>],
       [<AxesSubplot:title={'center':'duration'}>,
        <AxesSubplot:title={'center':'campaign'}>,
        <AxesSubplot:title={'center':'pdays'}>],
       [<AxesSubplot:title={'center':'previous'}>, <AxesSubplot:>,
        <AxesSubplot:>]], dtype=object)
```



In [8]:
```
# key findings with potential processing
# long tail (skewed left): log transformation
# some outliers: outlier removal

X['age'].hist(bins=100)
plt.title('Histogram plot for Age')
plt.ylabel('Count')
plt.xlabel('Age')
plt.show()
```



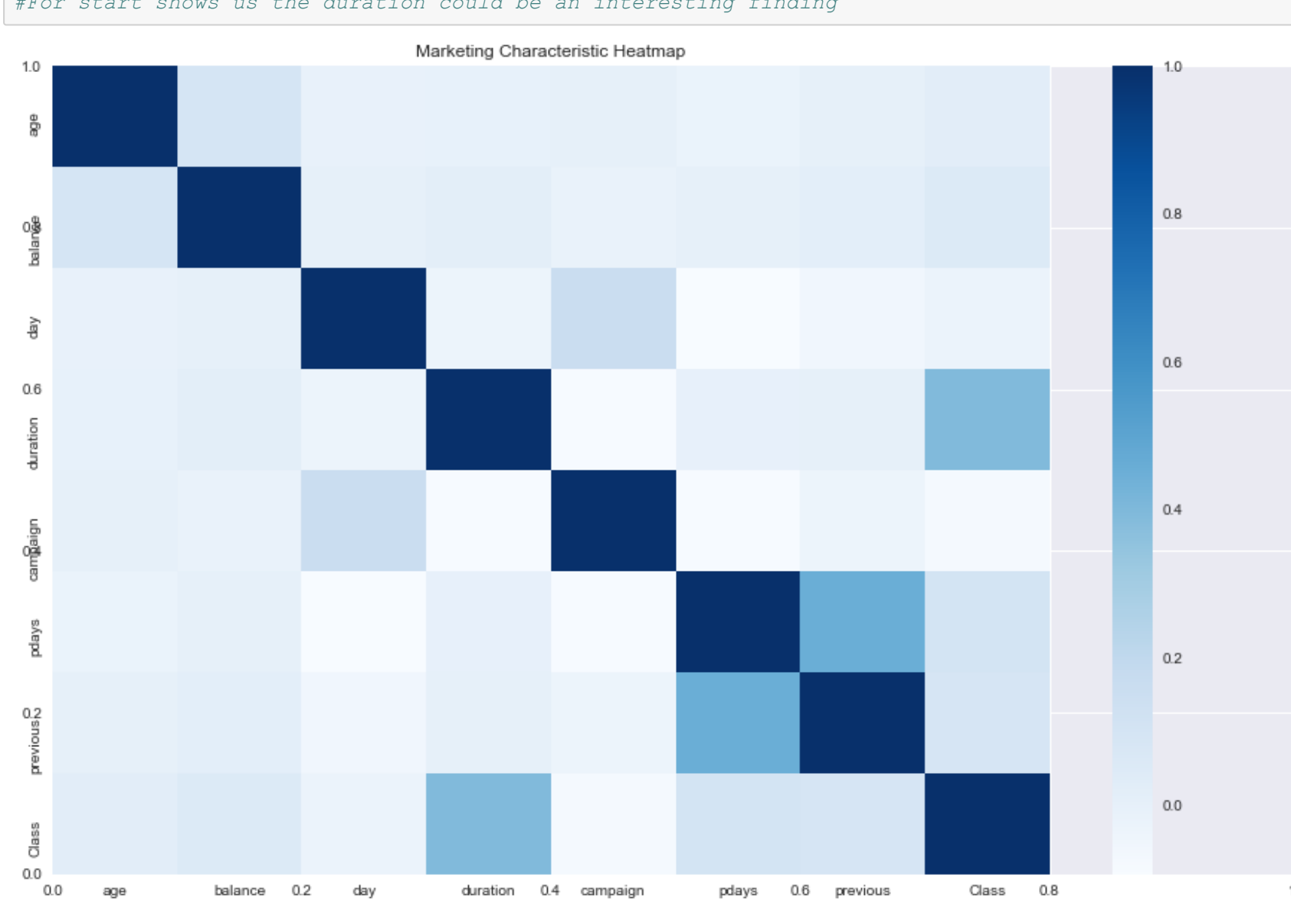
In [9]:
```
X['age'].plot.box()
plt.title('Box plot for Age')

plt.show()
```



In [10]:
```
# pairplot example using seaborn
sns.pairplot(data=X)
print("Pair plot")
```

Out[10]: `<seaborn.axisgrid.PairGrid at 0x227aab49708>`

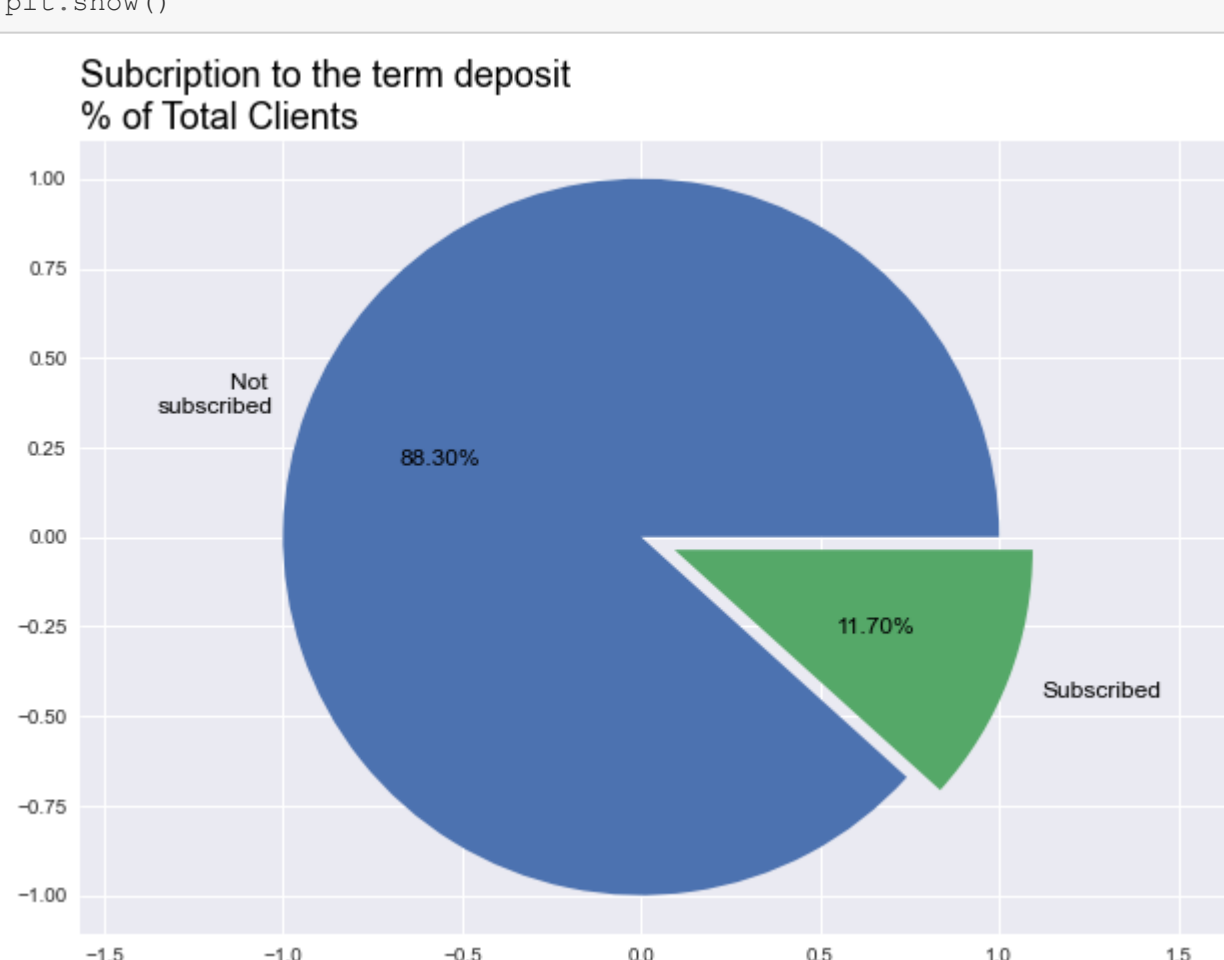
In [11]:
```
plt.subplots(figsize=(15,10))
ax = plt.axes()
ax.set_title("Marketing Characteristic Heatmap")
corr = df.corr()
sns.heatmap(corr,
            xticklabels=corr.columns.values,
            yticklabels=corr.columns.values,
           cmap="Blues")
plt.show()

#For start shows us the duration could be an interesting finding
```



In [12]:
```
fig, axs = plt.subplots(nrows = 3, ncols = 1, figsize = (15, 10))
print("Count Plot for Age, Month and Education")
sns.countplot(x = "age",data = df, ax=axs[0])
sns.countplot(x = "month", data = df, ax=axs[1])
sns.countplot(x = "education", data = df, ax=axs[2])

#We can see a few graphs that show us a little about the distribution.
```

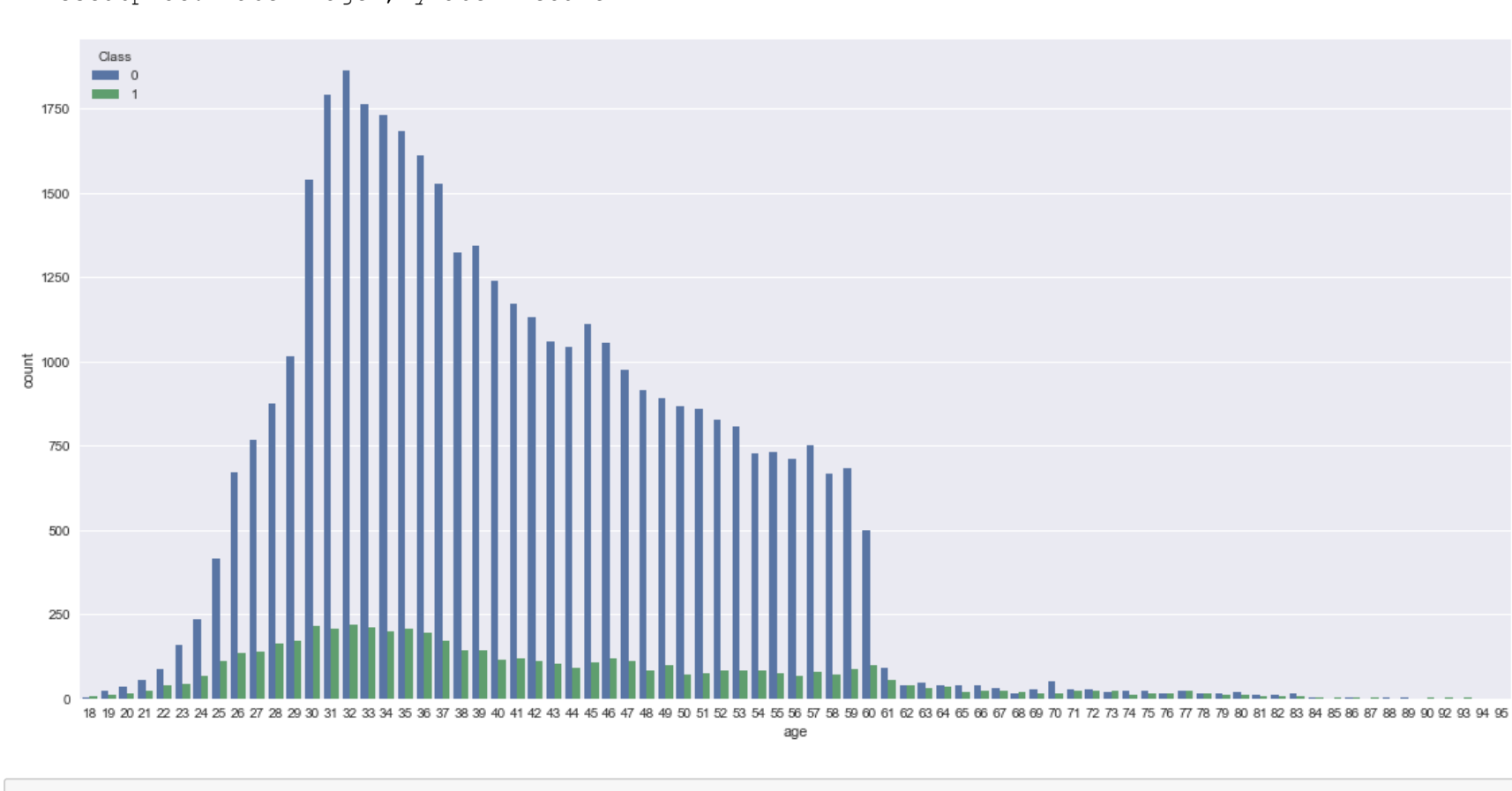
Out[12]: `<AxesSubplot:xlabel='education', ylabel='count'>`

In [13]:
```
# Pie chart
labels = ["Not \nsubscribed", "Subscribed"]
explode = (0, 0.1)  # only "explode" the second slice (i.e. 'Subscribed')

# depicting the visualization
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])

ax.pie(df['Class'].value_counts(),
       labels = labels,
       explode = explode,
       autopct ='%1.2f%%',
       frame = True,
       textprops = dict(color ="black", size=12))

ax.axis('equal')
plt.title('Subcription to the term deposit\n% of Total Clients',
     loc='left',
     color = 'black',
     fontsize = '18')

plt.show()
```



In [14]:
```
plt.figure(figsize=(19, 9))
sns.countplot(data=df, x='age', hue='Class')

print("Count plot for Age : Subscribed vs Not Subscribed")
```

Out[14]: `<AxesSubplot:xlabel='age', ylabel='count'>`



In [15]:
```
#Mean
print("Mean for the numerical feature")
df.groupby(["Class"]).mean().T # transpose makes the output easier to read
```

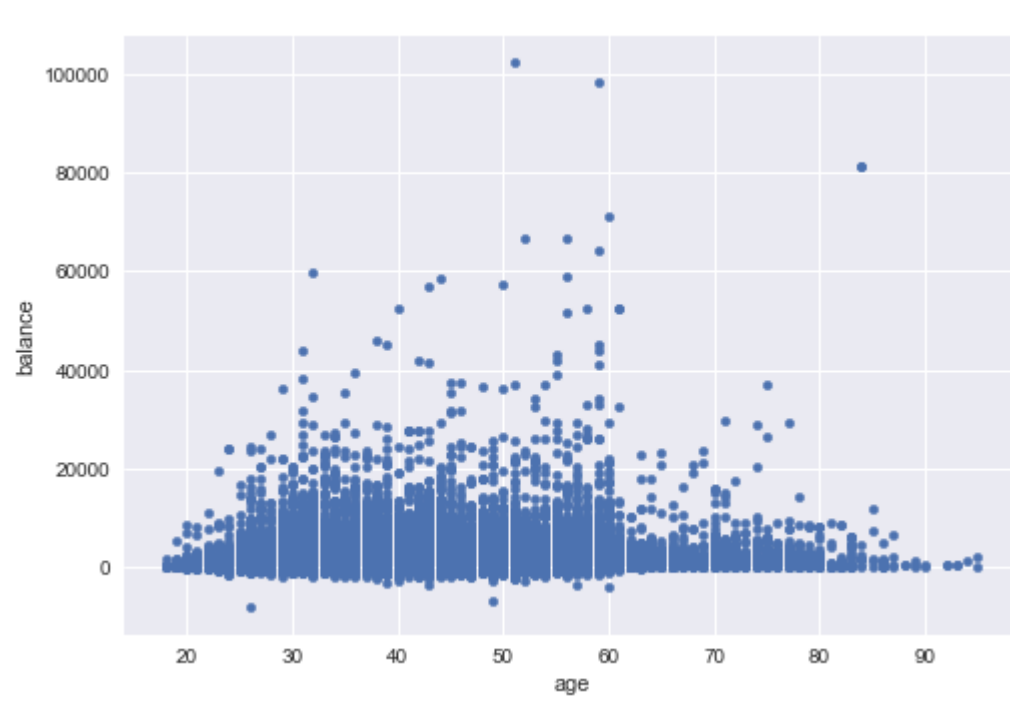
Mean for the numerical feature

Out[15]:

Class	0	1
age	40.838986	41.670070
balance	1303.714969	1804.267915
day	15.892290	15.158253
duration	221.182806	537.294574
campaign	2.846350	2.141047
pdays	36.421372	68.702968
previous	0.502154	1.170354

In [16]:
```
print("Scatter plot for age vs Balance")
df.plot(kind = 'scatter', x='age',y='balance')
```

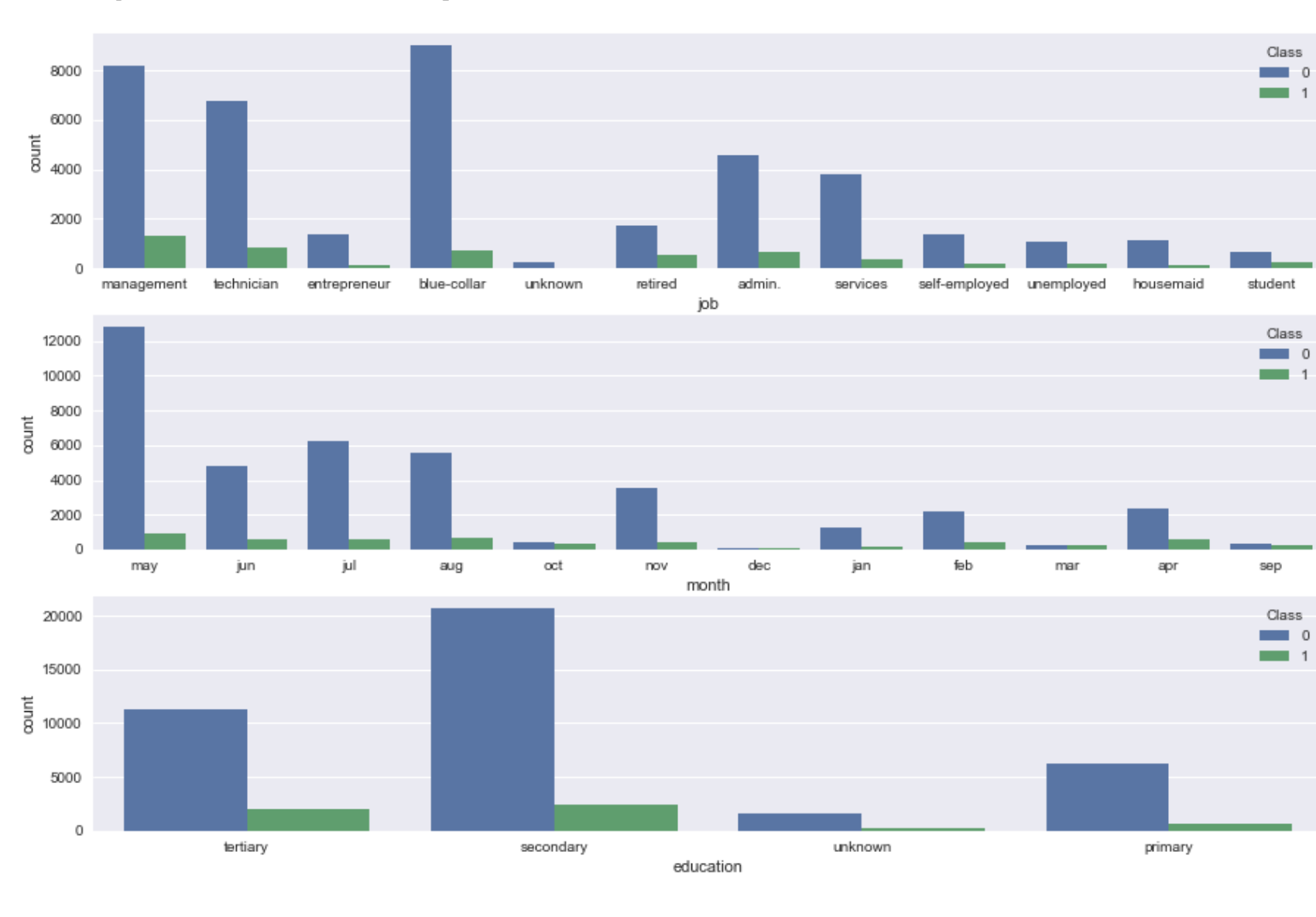
Out[16]: `<AxesSubplot:xlabel='age', ylabel='balance'>`



From the graph; Age range 50 to 60 seems to have higher balance; hence they are more likly to subscribe.

In [17]:
```
fig, axs = plt.subplots(nrows = 3, ncols = 1, figsize = (15, 10))

sns.countplot(x = "job",data = df, ax=axs[0],hue="Class")
sns.countplot(x = "month", data = df, ax=axs[1],hue="Class")
sns.countplot(x = "education", data = df, ax=axs[2],hue="Class")
```

Out[17]: `<AxesSubplot:xlabel='education', ylabel='count'>`



Train Test Split

In [18]:
```
# Select important Features
#X = X[["duration", "poutcome", "pdays", "housing", "age", "month", "day", "balance", "contact" ]]
#X = X[["duration", "default", "loan", "pdays", "housing", "age", "balance", "contact", "job", "educati
on", "poutcome" ]]
X  = X[['age', 'balance', 'housing', 'contact', 'day', 'month', 'duration',
       'pdays', 'poutcome']]

# Split the data into a training set and a test set.
# Any number for the random_state is fine, see 42: https://en.wikipedia.org/wiki/42_(number)
# We choose to use 20% (test_size=0.2) of the data set as the test set.
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,test_size=0.2, random_state=42)

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)

##added stratify option above
```

```
(36168, 9)
(9043, 9)
(36168,)
```

Baseline Prediction

Using Feature "contact" for baseline prediction; Cellular contact subscription rate is highest 14.91 %

Using Feature "poutcome" for baseline prediction; Previous campaign(poutcome) success subscription rate 64.7% is highest.

In [19]:
```
# Cellular contact subscription rate: 14.91 %
# Telephone contact subscription rate: 13.42 %
# Unknown contact subscription rate: 4.07 %

group_norm1 = df.groupby('contact')['Class'].value_counts(normalize=True)
print(cbold + "BaseLine Prediction with Contact Feature\n\n" + cEnd,group_norm1)

# based on Previous campaign(poutcome) success ; subscription rate : 64.7%


group_norm2 = df.groupby('poutcome')['Class'].value_counts(normalize=True)
print(cbold + "\nBaseLine Prediction with poutcome Feature\n\n" + cEnd,group_norm2)
```

```
BaseLine Prediction with Contact Feature

 contact    Class
cellular   0        0.850811
           1        0.149189
telephone  0        0.865795
           1        0.134205
unknown    0        0.959293
           1        0.040707
Name: Class, dtype: float64

BaseLine Prediction with poutcome Feature

 poutcome  Class
failure   0        0.873903
          1        0.126097
other     0        0.833152
          1        0.166848
success   1        0.647253
          0        0.352747
unknown   0        0.908385
          1        0.091615
Name: Class, dtype: float64
```

In [20]:
```
X_test.head()
```

Out[20]:

	age	balance	housing	contact	day	month	duration	pdays	poutcome
1392	40	640	yes	unknown	8	may	347	-1	unknown
7518	44	378	yes	unknown	30	may	203	-1	unknown
12007	31	356	yes	unknown	20	jun	228	-1	unknown
5536	36	655	yes	unknown	23	may	153	-1	unknown
29816	34	1921	yes	cellular	4	feb	61	-1	unknown

In [21]:
```
# rule-based prediction
baseline_pred1 = X_test['contact'].apply(lambda x: 1 if x == 'cellular' else 0)

# rule-based prediction
baseline_pred2 = X_test['poutcome'].apply(lambda x: 1 if x == 'Success' else 0)
```

In [22]:
```
from sklearn.metrics import accuracy_score

accuracy_score(y_test, baseline_pred2)

print("Baseline Accuracy Score with Feature 'poutcome' " , {accuracy_score(y_test, baseline_pred2)})

accuracy_score(y_test, baseline_pred1)

print("Baseline Accuracy Score with Feature 'Contact' " , {accuracy_score(y_test, baseline_pred1)})
```

```
Baseline Accuracy Score with Feature 'poutcome'  {0.8830034280659074}
Baseline Accuracy Score with Feature 'Contact'  {0.43580670131593496}
```

We shall choose poutcome feature to compare baseline accuracy. seems like Baseline accuracy with poutcome "Success" is 88.3 %

Data Preprocessing

In [56]:
```python
feature_names = num_original_feature_names + cat_new_feature_names
feature_names
```
Out[56]:
```
['age',
 'balance',
 'day',
 'duration',
 'pdays',
 'housing_no',
 'housing_yes',
 'contact_cellular',
 'contact_telephone',
 'contact_unknown',
 'month_apr',
 'month_aug',
 'month_dec',
 'month_feb',
 'month_jan',
 'month_jul',
 'month_jun',
 'month_mar',
 'month_may',
 'month_nov',
 'month_oct',
 'month_sep',
 'poutcome_failure',
 'poutcome_other',
 'poutcome_success',
 'poutcome_unknown']
```

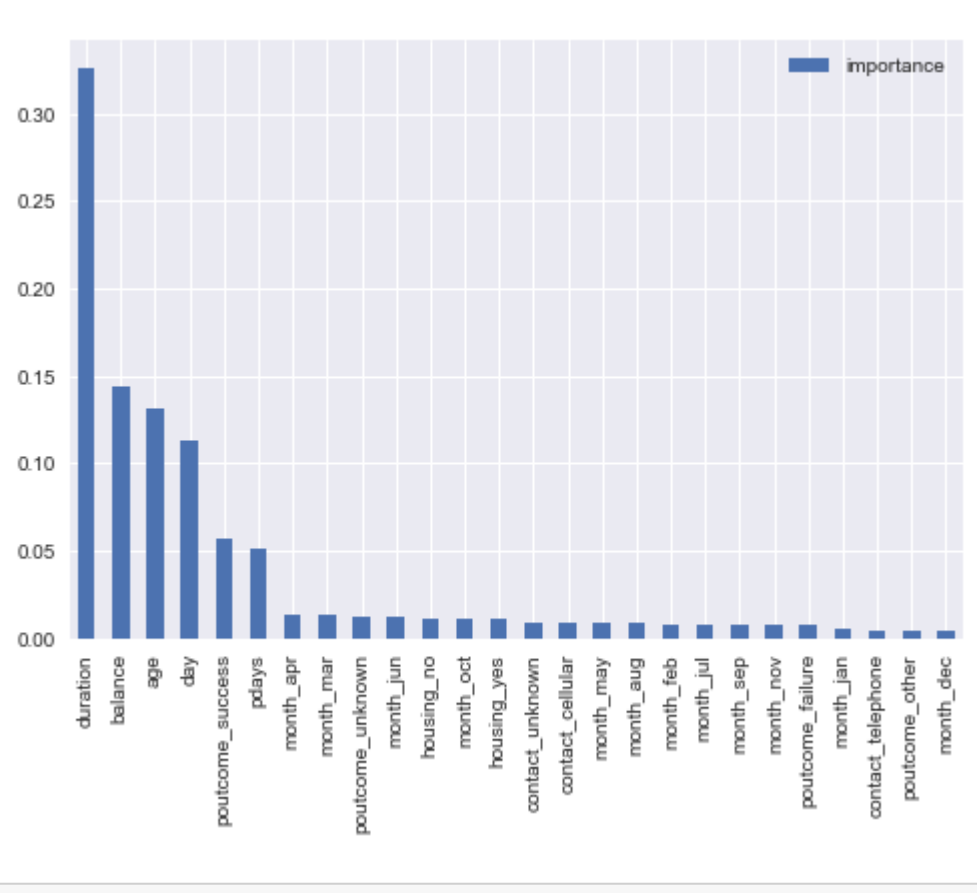
In [57]:
```python
r = pd.DataFrame(i, index=feature_names, columns=['importance'])
r
```
Out[57]:

	importance
age	0.131024
balance	0.143643
day	0.113451
duration	0.326142
pdays	0.050667
housing_no	0.011422
housing_yes	0.010521
contact_cellular	0.009086
contact_telephone	0.004562
contact_unknown	0.009179
month_apr	0.013694
month_aug	0.008426
month_dec	0.004208
month_feb	0.008181
month_jan	0.005000
month_jul	0.008174
month_jun	0.012353
month_mar	0.013381
month_may	0.008674
month_nov	0.007375
month_oct	0.011209
month_sep	0.007892
poutcome_failure	0.007258
poutcome_other	0.004462
poutcome_success	0.057358
poutcome_unknown	0.012656

In [58]:
```python
r.sort_values('importance', ascending=False)
```
Out[58]:

	importance
duration	0.326142
balance	0.143643
age	0.131024
day	0.113451
poutcome_success	0.057358
pdays	0.050667
month_apr	0.013694
month_mar	0.013381
poutcome_unknown	0.012656
month_jun	0.012353
housing_no	0.011422
month_oct	0.011209
housing_yes	0.010521
contact_unknown	0.009179
contact_cellular	0.009086
month_may	0.008674
month_aug	0.008426
month_feb	0.008181
month_jul	0.008174
month_sep	0.007892
month_nov	0.007375
poutcome_failure	0.007258
month_jan	0.005000
contact_telephone	0.004562
poutcome_other	0.004462
month_dec	0.004208

In [59]:
```python
r.sort_values('importance', ascending=False).plot.bar()
```
Out[59]: <AxesSubplot:>



In [60]:
```python
# Save the model as a pickle file
import joblib
joblib.dump(clf_best, "clf-best.pickle")
```
Out[60]: ['clf-best.pickle']

In [61]:
```python
# Load the model from a pickle file
saved_tree_clf = joblib.load("clf-best.pickle")
saved_tree_clf
```
Out[61]:
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num_pipeline',
                                                  Pipeline(steps=[('num_imputer',
                                                                   SimpleImputer()),
                                                                  ('scaler',
                                                                   StandardScaler())]),
                                                  ['age', 'balance', 'day',
                                                   'duration', 'pdays']),
                                                 ('cat_pipeline',
                                                  Pipeline(steps=[('cat_imputer',
                                                                   SimpleImputer(strategy='most_frequ
ent')),
                                                                  ('onehot',
                                                                   OneHotEncoder(handle_unknown='igno
re'))]),
                                                  ['housing', 'contact',
                                                   'month', 'poutcome'])])),
                ('clf_rf', RandomForestClassifier())])
```

In [62]:
```python
onehot_columns = list(clf_best.named_steps['preprocessor'].named_transformers_['cat_pipeline'].named_st
eps['onehot'].get_feature_names(input_features=cat_features))
```

In [63]:
```python
numeric_features_list = list(num_features)
numeric_features_list.extend(onehot_columns)
```

In [64]:
```python
print(numeric_features_list)
```
```
['age', 'balance', 'day', 'duration', 'pdays', 'housing_no', 'housing_yes', 'contact_cellular', 'cont
act_telephone', 'contact_unknown', 'month_apr', 'month_aug', 'month_dec', 'month_feb', 'month_jan',
'month_jul', 'month_jun', 'month_mar', 'month_may', 'month_nov', 'month_oct', 'month_sep', 'poutcome_
failure', 'poutcome_other', 'poutcome_success', 'poutcome_unknown']
```

SVM

In [65]:
```python
# try SVM classifer
from sklearn.svm import SVC
np.warnings.filterwarnings('ignore')

# SVC pipeline
pipeline_svc = Pipeline([
    ('preprocessor', preprocessor),
    ('clf_svc', SVC()),
])

# here we are trying three different kernel and three degree values for polynomail kernel
# in total 5 different combinations
param_grid_svc = [
    {
        'clf_svc__kernel': ['linear', 'poly', 'rbf'],
        'clf_svc__degree': [3, 4, 5],  # only for poly kernel
    }
]

# set up the grid search
grid_search_svc = GridSearchCV(pipeline_svc, param_grid_svc, cv=10, scoring='accuracy')
```

In [66]:
```python
# train the model using the full pipeline
#grid_search_svc.fit(X_train, y_train)
```

In [67]:
```python
# best test score
#grid_search_svc.best_score_
```

XG Boost

In [68]:
```python
from sklearn import preprocessing
lbl = preprocessing.LabelEncoder()
#X_train['default'] = lbl.fit_transform(X_train['default'].astype(str))
#X_train['loan'] = lbl.fit_transform(X_train['loan'].astype(str))
X_train['housing'] = lbl.fit_transform(X_train['housing'].astype(str))
X_train['contact'] = lbl.fit_transform(X_train['contact'].astype(str))
X_train['month'] = lbl.fit_transform(X_train['month'].astype(str))
X_train['poutcome'] = lbl.fit_transform(X_train['month'].astype(str))
#X_train['job'] = lbl.fit_transform(X_train['job'].astype(str))
#X_train['education'] = lbl.fit_transform(X_train['education'].astype(str))
```

In [69]:
```python
X_train.head()
```
Out[69]:

	age	balance	housing	contact	day	month	duration	pdays	poutcome
24001	36	861	0	1	29	1	140	-1	1
43409	24	4126	0	0	5	0	907	185	0
20669	44	244	1	0	12	1	1735	-1	1
18810	48	0	0	1	31	5	35	-1	7
23130	38	257	0	0	26	1	57	-1	1

In [70]:
```python
import xgboost as xgb
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
#from sklearn.preprocessing import Imputer

model = xgb.XGBClassifier()

pipeline = Pipeline([
    ('standard_scaler', StandardScaler()),
    ('pca', PCA()),
    ('model', model)
])

param_grid = {
    'pca__n_components': [5, 10, 15, 20],
    'model__max_depth': [2, 3, 5, 7, 10],
    'model__n_estimators': [10, 100, 500]
}

grid_search_xg = GridSearchCV(pipeline, param_grid, cv=5, n_jobs=-1, scoring='roc_auc')
```

In [71]:
```python
#%%time
grid_search_xg.fit(X_train, y_train)

mean_score = grid_search_xg.cv_results_["mean_test_score"][grid_search_xg.best_index_]
std_score = grid_search_xg.cv_results_["std_test_score"][grid_search_xg.best_index_]

grid_search_xg.best_params_, mean_score, std_score

print(f"Best parameters: {grid_search_xg.best_params_}")
print(f"Mean CV score: {mean_score: .6f}")
print(f"Standard deviation of CV score: {std_score: .6f}")
print(f"Best score: {grid_search_xg.best_score_: .6f}")
```
```
[18:28:25] WARNING: ..\src\learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_
metric if you'd like to restore the old behavior.
Best parameters: {'model__max_depth': 7, 'model__n_estimators': 100, 'pca__n_components': 5}
Mean CV score:  0.894718
Standard deviation of CV score:  0.004039
Best score:  0.894718
```

In [72]:
```python
clf_best = grid_search_xg.best_estimator_
clf_best
```
Out[72]:
```
Pipeline(steps=[('standard_scaler', StandardScaler()),
                ('pca', PCA(n_components=5)),
                ('model',
                 XGBClassifier(base_score=0.5, booster='gbtree',
                               colsample_bylevel=1, colsample_bynode=1,
                               colsample_bytree=1, gamma=0, gpu_id=-1,
                               importance_type='gain',
                               interaction_constraints='',
                               learning_rate=0.300000012, max_delta_step=0,
                               max_depth=7, min_child_weight=1, missing=nan,
                               monotone_constraints='()', n_estimators=100,
                               n_jobs=8, num_parallel_tree=1, random_state=0,
                               reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
                               subsample=1, tree_method='exact',
                               validate_parameters=1, verbosity=None))])
```

In [73]:
```python
# best test score
print('best dt score is: ', grid_search_dt.best_score_)
#print('best svc score is: ', grid_search_svc.best_score_)
print('best rf score is: ', grid_search_rf.best_score_)
print('best xgboost score is: ', grid_search_xg.best_score_)
```
```
best dt score is:  0.9024552506477523
best rf score is:  0.9063535929766269
best xgboost score is:  0.8947183340539521
```

Best performing Model is Random Forrest with accuracy 90.63 %

Naive Bayes

In [74]:
```python
NBdf = df
NBdf.Class.replace((1, 2), ('no', 'yes'), inplace=True)
NBdf.head()
```
Out[74]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	p
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	may	261	1	-1	0	
1	44	technician	single	secondary	no	29	yes	no	unknown	5	may	151	1	-1	0	
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	76	1	-1	0	
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	may	92	1	-1	0	
4	33	unknown	single	unknown	no	1	no	no	unknown	5	may	198	1	-1	0	

Features to focus on

- duration - numerical
- job - categorical
- martial - categorical
- contact - categorical
- poutcome - categorical

In [75]:
```python
categorical = [var for var in NBdf.columns if NBdf[var].dtype=='O']
```

In [76]:
```python
categorical
```
Out[76]:
```
['job',
 'marital',
 'education',
 'default',
 'housing',
 'loan',
 'contact',
 'month',
 'poutcome',
 'Class']
```

In [77]:
```python
numerical = [var for var in NBdf.columns if NBdf[var].dtype!='O']
```

In [78]:
```python
numerical
```
Out[78]: ['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous']

In [79]:
```python
X = NBdf
X = X.drop(['education', 'default', 'housing', 'loan', 'month', 'Class','age', 'balance', 'day','campai
gn', 'pdays', 'previous'],axis = 1)
X.columns
```
Out[79]: Index(['job', 'marital', 'contact', 'duration', 'poutcome'], dtype='object')

In [80]:
```python
y = NBdf['Class']
```

In [81]:
```python
from sklearn.model_selection import train_test_split
X_train,X_test, y_train,y_test = train_test_split(X,y,test_size=0.25,random_state=42)
```

In [82]:
```python
import category_encoders as ce

oneHot = ce.OneHotEncoder(cols=X.columns)
X_train = oneHot.fit_transform(X_train)
X_test = oneHot.transform(X_test)
```

In [84]:
```python
from sklearn.naive_bayes import GaussianNB

bayes = GaussianNB()
#bayes.fit(X_train, y_train)
```

In [ ]:
```python
y_pred = bayes.predict(X_test)
```

In [ ]:
```python
from sklearn.metrics import accuracy_score

print('Model Accuracy Score is {0:0.4f}'.format(accuracy_score(y_test, y_pred)))
```

In [ ]:
```python
print('Training-set Accuracy Score is {0:0.4f}'. format(accuracy_score(y_train, bayes.predict(X_train
))))
```

In [ ]:
```python
y_test.value_counts()
```

In [ ]:

In [ ]: