In [401]:
```
#Loading Libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import statistics
```

In [403]:
```
#Load Data
x1 = pd.read_csv('C:/Users/nneam/OneDrive/Documents/540Assignments/dsc680_final.csv')
df = pd.DataFrame(data=x1, columns = x1.columns)
```

Data Overview

In [404]:
```
#Data Preview
df.head()
```

Out[404]:

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

In [405]:
```
#df info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49732 entries, 0 to 49731
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   age        49732 non-null  int64 
 1   job        49732 non-null  object
 2   marital    49732 non-null  object
 3   education  49732 non-null  object
 4   default    49732 non-null  object
 5   balance    49732 non-null  int64 
 6   housing    49732 non-null  object
 7   loan       49732 non-null  object
 8   contact    49732 non-null  object
 9   day        49732 non-null  int64 
 10  month      49732 non-null  object
 11  duration   49732 non-null  int64 
 12  campaign   49732 non-null  int64 
 13  pdays      49732 non-null  int64 
 14  previous   49732 non-null  int64 
 15  poutcome   49732 non-null  object
 16  y          49732 non-null  object
dtypes: int64(7), object(10)
memory usage: 6.5+ MB
```

In [406]:
```
#Dropping unneccesary columns
df = df.drop(columns=['contact', 'month', 'job'])
```

In [407]:
```
#Data Preview
df.head()
```

Out[407]:

	age	marital	education	default	balance	housing	loan	day	duration	campaign	pdays	previous	poutcome	y
0	58	married	tertiary	no	2143	yes	no	5	261	1	-1	0	unknown	no
1	44	single	secondary	no	29	yes	no	5	151	1	-1	0	unknown	no
2	33	married	secondary	no	2	yes	yes	5	76	1	-1	0	unknown	no
3	47	married	unknown	no	1506	yes	no	5	92	1	-1	0	unknown	no
4	33	single	unknown	no	1	no	no	5	198	1	-1	0	unknown	no

In [408]:
```
#Count of outcome types. Will be grouped by success(1) or any other outcome(0)
df['poutcome'].value_counts(ascending=False)
```

Out[408]:
```
unknown    40664
failure     5391
other       2037
success     1640
Name: poutcome, dtype: int64
```

In [409]:
```
#count of y value
df['y'].value_counts(ascending=False)
```

Out[409]:
```
no     43922
yes     5810
Name: y, dtype: int64
```

In [410]:
```
#Mariral Status
df['marital'].value_counts(ascending=False)
```

Out[410]:
```
married     30011
single      13986
divorced     5735
Name: marital, dtype: int64
```

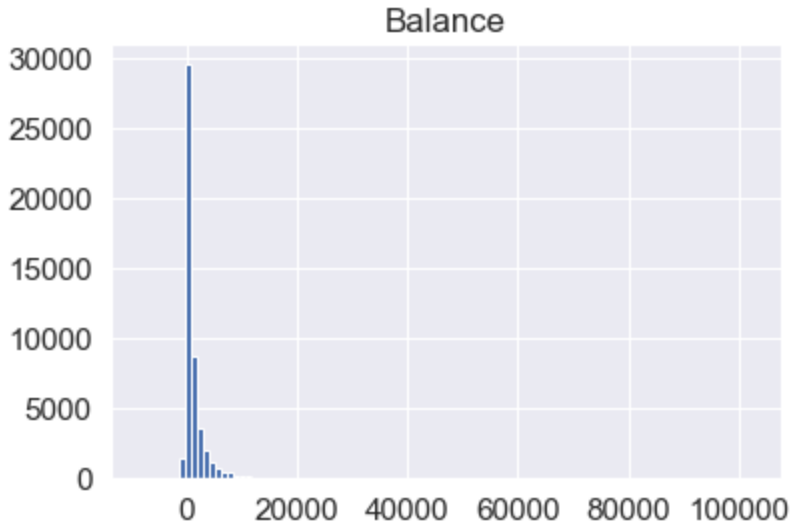
In [411]:
```
#education will be grouped by either Above HS (1) or HS and Below(0)
df['education'].value_counts(ascending=False)
```

Out[411]:
```
secondary    25508
tertiary     14651
primary       7529
unknown       2044
Name: education, dtype: int64
```

In [412]:
```
#Previous outcome will be grouped by either succesful(1) or any other outcome(0)
df['poutcome'].value_counts(ascending=False)
```

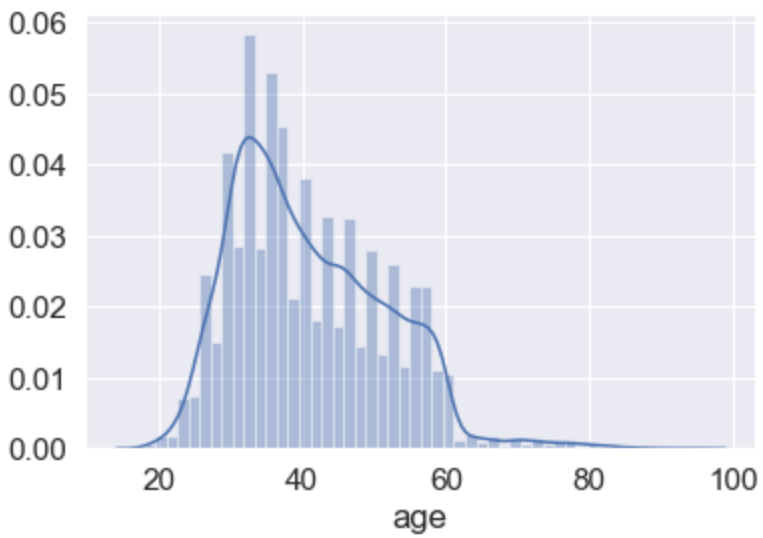
Out[412]:
```
unknown    40664
failure     5391
other       2037
success     1640
Name: poutcome, dtype: int64
```

In [413]:
```
# Histogram for balance
df['balance'].hist(bins=100)
plt.title('Balance')
plt.show()
```



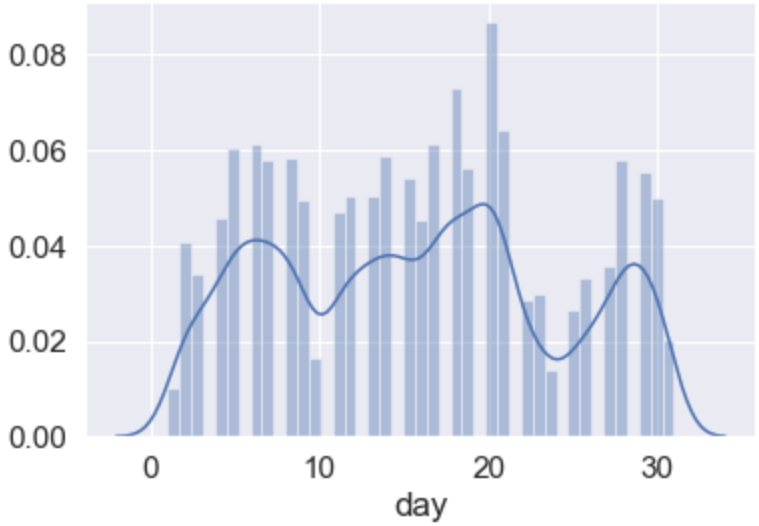
In [414]:
```
# Desnity and Histogram for Age
sns.distplot(a=df.age)
```

Out[414]: <matplotlib.axes._subplots.AxesSubplot at 0x234070a1790>

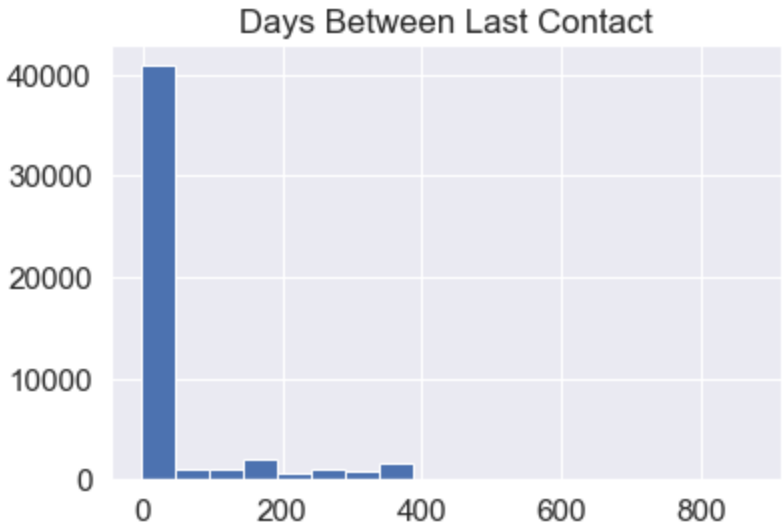


In [415]:
```
# Histogram for Day
sns.distplot(a=df.day)
```

Out[415]: <matplotlib.axes._subplots.AxesSubplot at 0x234073f8f40>



In [416]:
```
# Histogram for Day
df['pdays'].hist(bins=18)
plt.title('Days Between Last Contact')
plt.show()
```



Data Clean

In [417]:
```
#Replcing yes/no and gender column value for model
df['default'] = df['default'].replace(['yes','no'],['1','0'])
df['housing'] = df['housing'].replace(['yes','no'],['1','0'])
df['loan'] = df['loan'].replace(['yes','no'],['1','0'])
df['y'] = df['y'].replace(['yes','no'],['1','0'])
df['pdays'] = df['pdays'].replace([-1],[0])
df['marital'] = df['marital'].replace(['married','single', 'divorced'],['1','0', '0'])
df['education'] = df['education'].replace(['tertiary','secondary','unknown', 'primary'],['1', '0','0', '0'])
df['poutcome'] = df['poutcome'].replace(['success','other','failure', 'unknown'],['1', '0','0', '0'])
#Data Preview
df.head()
```

Out[417]:

	age	marital	education	default	balance	housing	loan	day	duration	campaign	pdays	previous	poutcome	y
0	58	1	1	0	2143	1	0	5	261	1	0	0	0	0
1	44	0	0	0	29	1	0	5	151	1	0	0	0	0
2	33	1	0	0	2	1	1	5	76	1	0	0	0	0
3	47	1	0	0	1506	1	0	5	92	1	0	0	0	0
4	33	0	0	0	1	0	0	5	198	1	0	0	0	0

In [418]:
```
#Changing value type
df['default'] = df.default.astype(int)
df['housing'] = df.housing.astype(int)
df['loan'] = df.loan.astype(int)
df['y'] = df.y.astype(int)
df['pdays'] = df.pdays.astype(int)
df['marital'] = df.pdays.astype(int)
df['education'] = df.pdays.astype(int)
df['poutcome'] = df.pdays.astype(int)
```

Correlation Testing

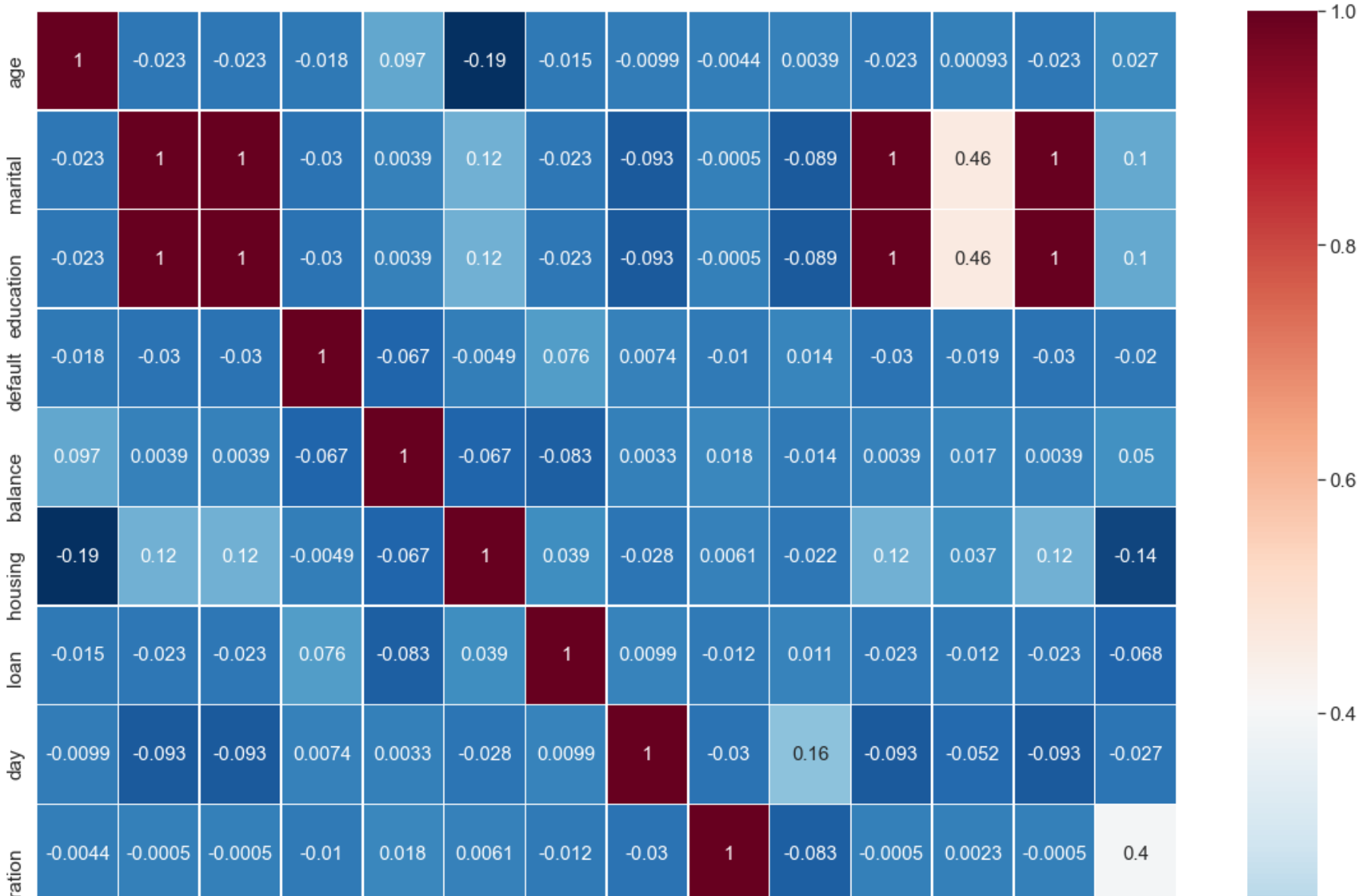
In [419]:
```
#Assign pearson correlation
pearson = df.corr(method = 'pearson')
pearson
```

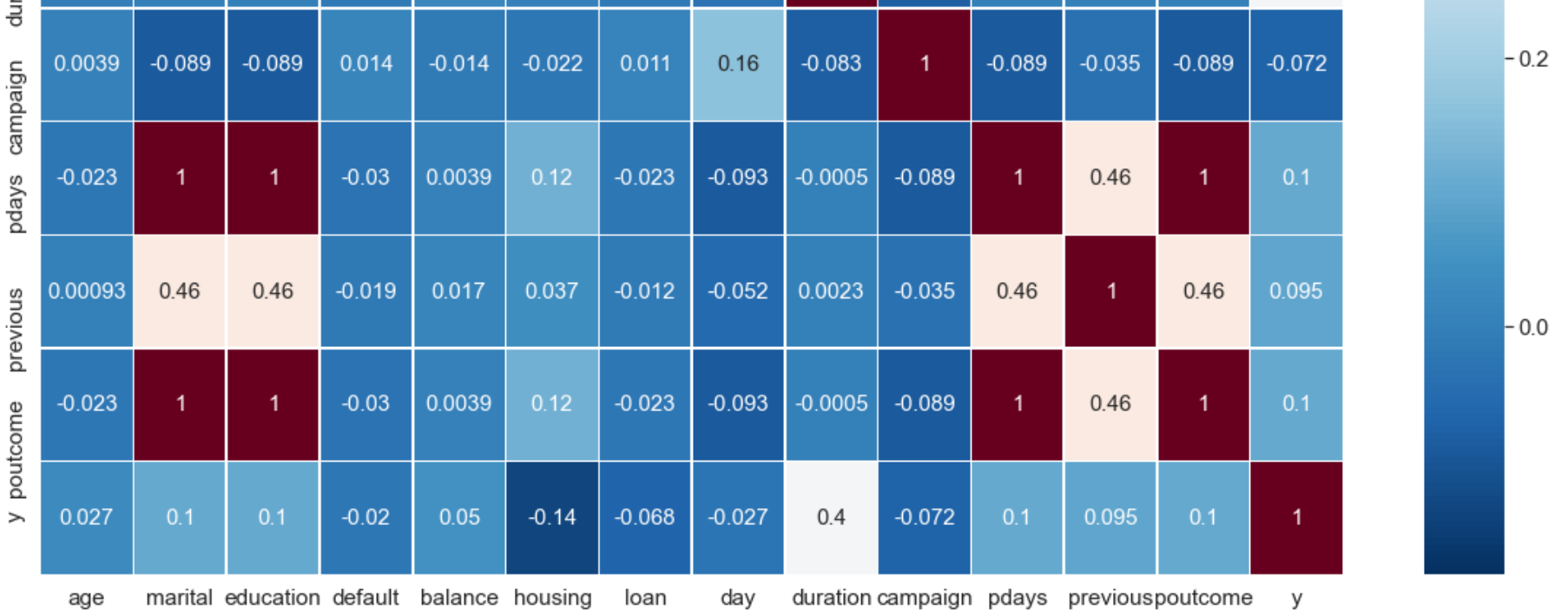
Out[419]:

	age	marital	education	default	balance	housing	loan	day	duration	campaign	pdays	previous	pou
age	1.000000	-0.022504	-0.022504	-0.017895	0.096564	-0.186225	-0.015299	-0.009880	-0.004399	0.003877	-0.022504	0.000928	-0.0
marital	-0.022504	1.000000	1.000000	-0.029599	0.003863	0.123678	-0.023445	-0.093149	-0.000498	-0.088920	1.000000	0.461281	1.0
education	-0.022504	1.000000	1.000000	-0.029599	0.003863	0.123678	-0.023445	-0.093149	-0.000498	-0.088920	1.000000	0.461281	1.0
default	-0.017895	-0.029599	-0.029599	1.000000	-0.067118	-0.004906	0.076101	0.007432	-0.010179	0.014234	-0.029599	-0.018825	-0.0
balance	0.096564	0.003863	0.003863	-0.067118	1.000000	-0.067068	-0.083231	0.003348	0.018195	-0.014147	0.003863	0.017243	0.0
housing	-0.186225	0.123678	0.123678	-0.004906	-0.067068	1.000000	0.039248	-0.028256	0.006088	-0.021760	0.123678	0.037042	0.1
loan	-0.015299	-0.023445	-0.023445	0.076101	-0.083231	0.039248	1.000000	0.009908	-0.011781	0.010603	-0.023445	-0.011729	-0.0
day	-0.009880	-0.093149	-0.093149	0.007432	0.003348	-0.028256	0.009908	1.000000	-0.029674	0.162336	-0.093149	-0.052083	-0.0
duration	-0.004399	-0.000498	-0.000498	-0.010179	0.018195	0.006088	-0.011781	-0.029674	1.000000	-0.083061	-0.000498	0.002330	-0.0
campaign	0.003877	-0.088920	-0.088920	0.014234	-0.014147	-0.021760	0.010603	0.162336	-0.083061	1.000000	-0.088920	-0.035162	-0.0
pdays	-0.022504	1.000000	1.000000	-0.029599	0.003863	0.123678	-0.023445	-0.093149	-0.000498	-0.088920	1.000000	0.461281	1.0
previous	0.000928	0.461281	0.461281	-0.018825	0.017243	0.037042	-0.011729	-0.052083	0.002330	-0.035162	0.461281	1.000000	0.4
poutcome	-0.022504	1.000000	1.000000	-0.029599	0.003863	0.123678	-0.023445	-0.093149	-0.000498	-0.088920	1.000000	0.461281	1.0
y	0.026939	0.103369	0.103369	-0.020336	0.049705	-0.136070	-0.068381	-0.026821	0.395099	-0.072085	0.103369	0.094567	0.1

In [420]:
```
#Pearson Correlation Heatmap
fig, ax = plt.subplots(figsize=(20,20))
sns.heatmap(pearson,
            xticklabels=pearson.columns,
            yticklabels=pearson.columns,
            cmap='RdBu_r',
            annot=True,
            linewidth= 0.5,
            annot_kws={"size": 15},
            ax=ax)
```

Out[420]: <matplotlib.axes._subplots.AxesSubplot at 0x234071d9ca0>





In [427]:
```
#Variance Testing
df.var()
```

Out[427]:
```
age           1.126784e+02
marital       9.958289e+03
education     9.958289e+03
default       1.759540e-02
balance       9.251384e+06
housing       2.467828e-01
loan          1.341000e-01
day           6.915053e+01
duration      6.643153e+04
campaign      9.604264e+00
pdays         9.958289e+03
previous      5.084293e+00
poutcome      9.958289e+03
y             1.031799e-01
dtype: float64
```

Model Creation

In [369]:
```
#Load Model Libraries
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.inspection import permutation_importance
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score,precision_score,recall_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
```

In [322]:
```
#Creating Data Variables
X = df.drop('y', axis=1)
y = df['y']
```

In [359]:
```
#Creating the Test, Train, and Split
X_train, X_test, y_train, y_test=train_test_split(X,y,test_size=0.2,random_state=40)
```

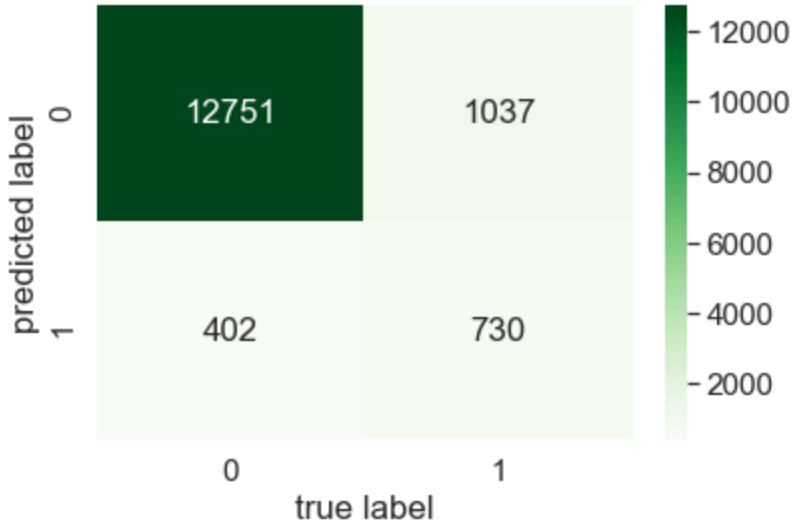
In [360]:
```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [365]:
```
#Model
clf = RandomForestClassifier(n_estimators=20, random_state=40)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

Model Analysis

In [367]:
```
#Confusion Matrix
mat = confusion_matrix(y_test,y_pred)
sns.heatmap(mat.T, annot=True, fmt='d', cmap=plt.cm.Greens, cbar=True)
plt.xlabel('true label')
plt.ylabel('predicted label')
```

Out[367]: Text(26.5, 0.5, 'predicted label')



In [366]:
```
#Classification score
print(classification_report(y_test,y_pred.round()))
print("Total Accuracy:", accuracy_score(y_test, y_pred.round()))
```

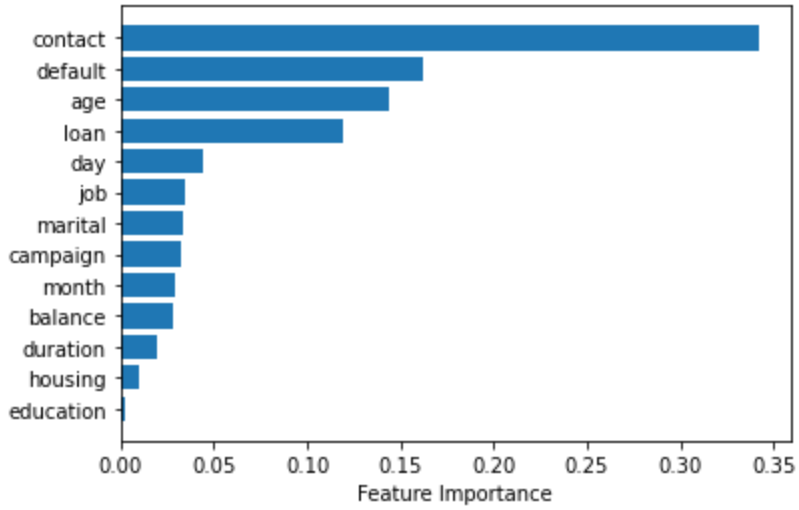
```
              precision    recall  f1-score   support

           0       0.92      0.97      0.95     13153
           1       0.64      0.41      0.50      1767

    accuracy                           0.90     14920
   macro avg       0.78      0.69      0.73     14920
weighted avg       0.89      0.90      0.89     14920

Total Accuracy: 0.9035522788203754
```

In [327]:
```
#Compare Dataset Results
print('Training set metrics:')
print('Accuracy:', accuracy_score(y_train, clf.predict(X_train)))
print('Precision:', precision_score(y_train, clf.predict(X_train)))
print('Recall:', recall_score(y_train, clf.predict(X_train)))

print('-----------------------')

print('Test set metrics:')
print('Accuracy:', accuracy_score(y_test, clf.predict(X_test)))
print('Precision:', precision_score(y_test, clf.predict(X_test)))
print('Recall:', recall_score(y_test, clf.predict(X_test)))
```

```
Training set metrics:
Accuracy: 0.9999497297976624
Precision: 1.0
Recall: 0.9995664426620421
-----------------------
Test set metrics:
Accuracy: 0.912335377500754
Precision: 0.7124183006535948
Recall: 0.4553049289891395
```

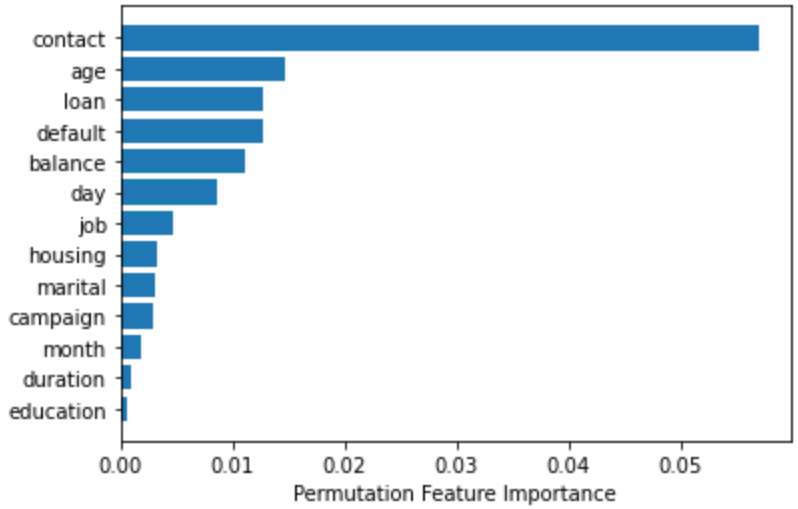
In [333]:
```
# Feature Importance
clf.feature_importances_
sorted_idx = clf.feature_importances_.argsort()
plt.barh(x1.columns[sorted_idx], clf.feature_importances_[sorted_idx])
plt.xlabel("Feature Importance")
```

Out[333]:
```
Text(0.5, 0, 'Feature Importance')
```



In [334]:
```
# Permutation Feature Importance
perm_importance = permutation_importance(clf, X_test, y_test)
sorted_idx = perm_importance.importances_mean.argsort()
plt.barh(x1.columns[sorted_idx], perm_importance.importances_mean[sorted_idx])
plt.xlabel("Permutation Feature Importance")
```

Out[334]:
```
Text(0.5, 0, 'Permutation Feature Importance')
```



Hyperparameter Model Tuning

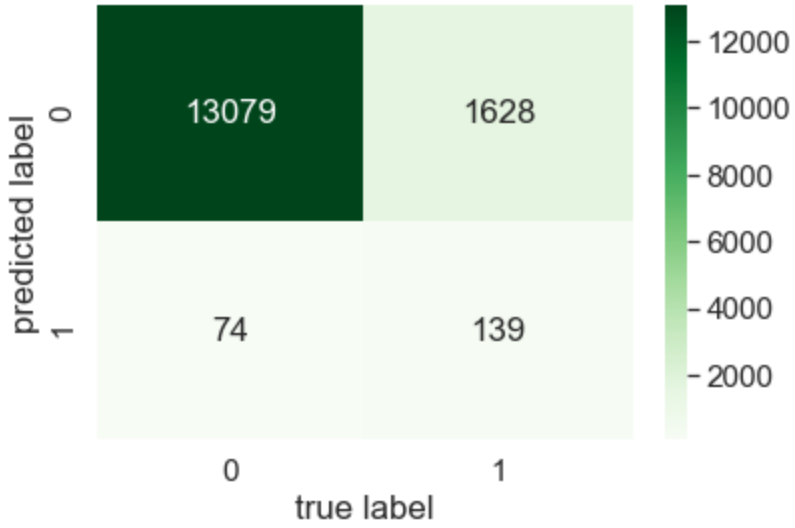
In [372]:
```
# Grid Search
n_estimators = [100, 300, 500, 800, 1200]
max_depth = [5, 8, 15, 25, 30]
min_samples_split = [2, 5, 10, 15, 100]
min_samples_leaf = [1, 2, 5, 10]

hyperF = dict(n_estimators = n_estimators, max_depth = max_depth,
              min_samples_split = min_samples_split,
             min_samples_leaf = min_samples_leaf)

gridF = GridSearchCV(clf, hyperF, cv = 1, verbose = 1,
                     n_jobs = 1)
bestF = gridF.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 500 candidates, totalling 1500 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:  3.9min
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed: 14.4min
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed: 32.3min
[Parallel(n_jobs=-1)]: Done 792 tasks      | elapsed: 67.2min
[Parallel(n_jobs=-1)]: Done 1242 tasks      | elapsed: 118.5min
[Parallel(n_jobs=-1)]: Done 1500 out of 1500 | elapsed: 147.1min finished
```

In [374]:
```
#Create model
forestOpt = RandomForestClassifier(random_state = 1, max_depth = 5, n_estimators = 20, min_samples_split = 2, min_samp

#Fit data
modelOpt = forestOpt.fit(X_train, y_train)
y_pred = modelOpt.predict(X_test)
```

In [375]:
```
#Confusion Matrix for Grid Search
mat = confusion_matrix(y_test,y_pred)
sns.heatmap(mat.T, annot=True, fmt='d', cmap=plt.cm.Greens, cbar=True)
plt.xlabel('true label')
plt.ylabel('predicted label')
```

Out[375]:
```
Text(26.5, 0.5, 'predicted label')
```



In [376]:
```
#Classification Report
print(classification_report(y_test,y_pred.round()))
print("Total Accuracy:", accuracy_score(y_test, y_pred.round()))
```

```
              precision    recall  f1-score   support

           0       0.89      0.99      0.94     13153
           1       0.65      0.08      0.14      1767

    accuracy                           0.89     14920
   macro avg       0.77      0.54      0.54     14920
weighted avg       0.86      0.89      0.84     14920

Total Accuracy: 0.8859249329758713
```

Standard Logistic Regression Model

In [382]:
```
#Load Library
from sklearn.linear_model import LogisticRegression

#Create Model
logmodel = LogisticRegression(solver='liblinear', max_iter=200, penalty='l2')
logmodel.fit(X_train,y_train)
predictions = logmodel.predict(X_test)
```

In [383]:
```
#Classification Report
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.90      0.98      0.94     13153
           1       0.57      0.19      0.28      1767

    accuracy                           0.89     14920
   macro avg       0.73      0.58      0.61     14920
weighted avg       0.86      0.89      0.86     14920
```

In [384]:
```
#Confusion Matrix
mat = confusion_matrix(y_test,y_pred)
sns.heatmap(mat.T, annot=True, fmt='d', cmap=plt.cm.Greens, cbar=True)
plt.xlabel('true label')
plt.ylabel('predicted label')
```

Out[384]:
```
Text(26.5, 0.5, 'predicted label')
```