Missing Value Analysis

```
import pandas as pd
import numpy as np
import missingno as msno
%matplotlib inline
import numpy as np
```

```
import os
os.environ['KAGGLE_CONFIG_DIR'] = "/content/gdrive/My Drive/Kaggle"
```

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive
```

```
%cd /content/gdrive/My Drive/Kaggle
```

```
/content/gdrive/My Drive/Kaggle
```

```
!kaggle datasets download -d imsparsh/jobathon-analytics-vidhya
```

```
!ls
```

```
!unzip \*.zip  && rm *.zip
```

```
train=pd.read_csv('/content/gdrive/MyDrive/Kaggle/train.csv')
test=pd.read_csv('/content/gdrive/MyDrive/Kaggle/test.csv')
```

Let's view column-wise missing value counts

```
train.isnull().sum()
```

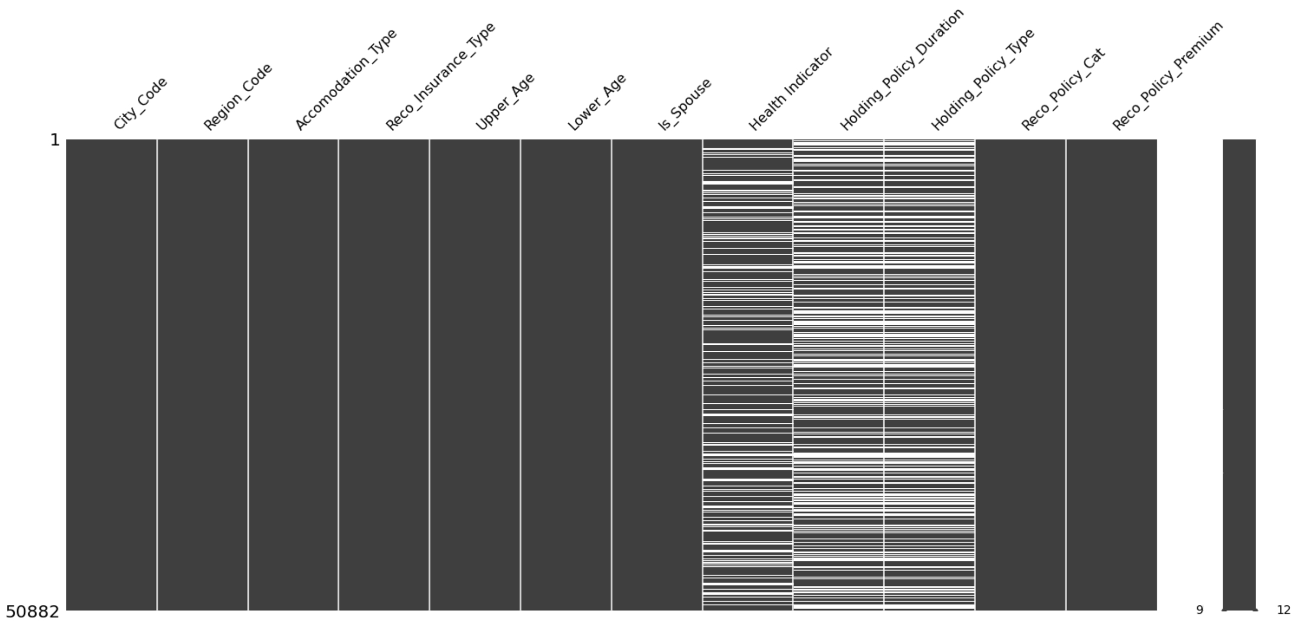
```
ID                             0
City_Code                      0
Region_Code                    0
Accomodation_Type              0
Reco_Insurance_Type            0
Upper_Age                      0
Lower_Age                      0
Is_Spouse                      0
Health Indicator           11691
Holding_Policy_Duration    20251
Holding_Policy_Type        20251
Reco_Policy_Cat                0
Reco_Policy_Premium            0
```

```
Response                     0
dtype: int64
```

Nullity Matrix - missing value pattern

```
msno.matrix(train.iloc[:,1:13])
```

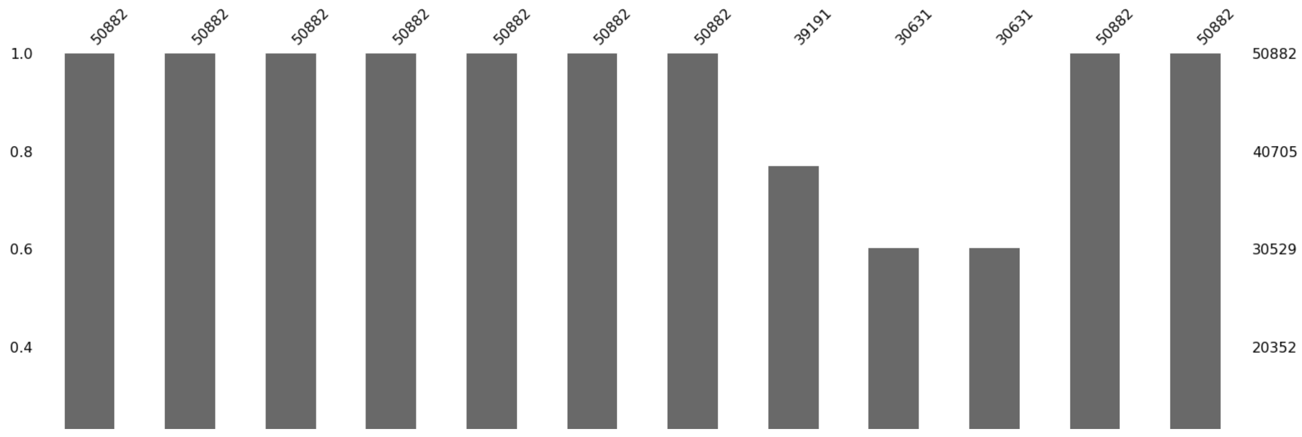
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3f75abed10>
```



Missing value by column

```
msno.bar(train.iloc[:,1:13])
```

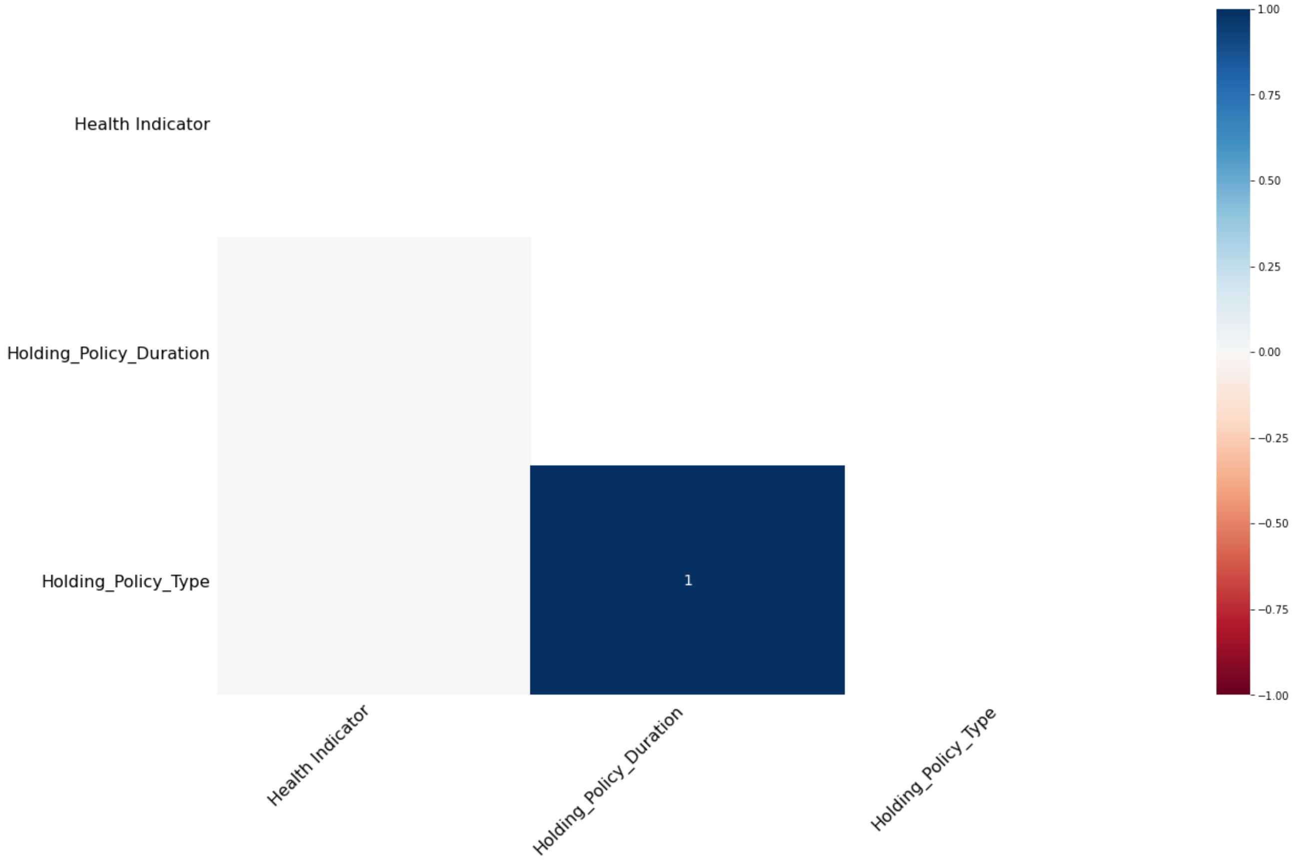
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3f75a4d590>
```



Pair-Wise nullity correlation analysis


```
msno.heatmap(train.iloc[:,1:13])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3f75999650>
```

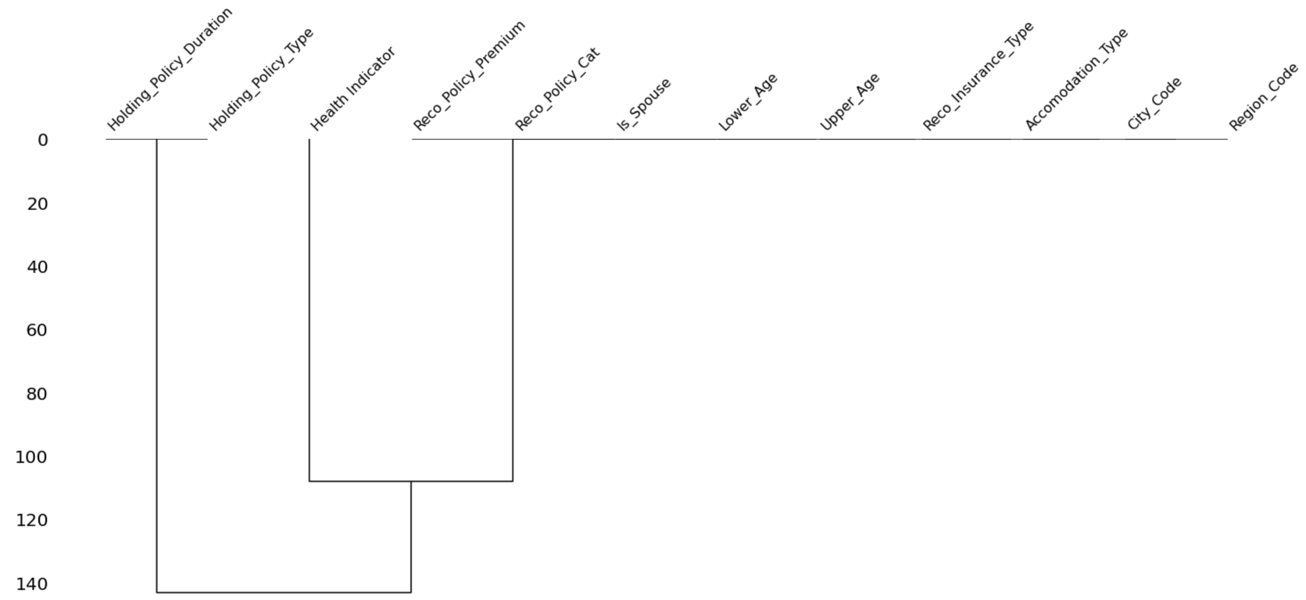


The above heatmap explains that both holding policy type and holding policy duration variables have 1 nullity correlation, that is if one variable appears the other definitely also does.

- Dendrogram-reveals deeper trend than pair-wise analysis heatmap

```
msno.dendrogram(train.iloc[:,1:13])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3f7588a790>
```



The above dendrogram explains that the horizontal line(top) is zero binary distance. Predicts one variable another variable's presence. if the height of the cluster is large then there is a mismatch between the variable records.

- Test Dataset

- Test dataset column-wise missing value count.

```
test.isnull().sum()
```

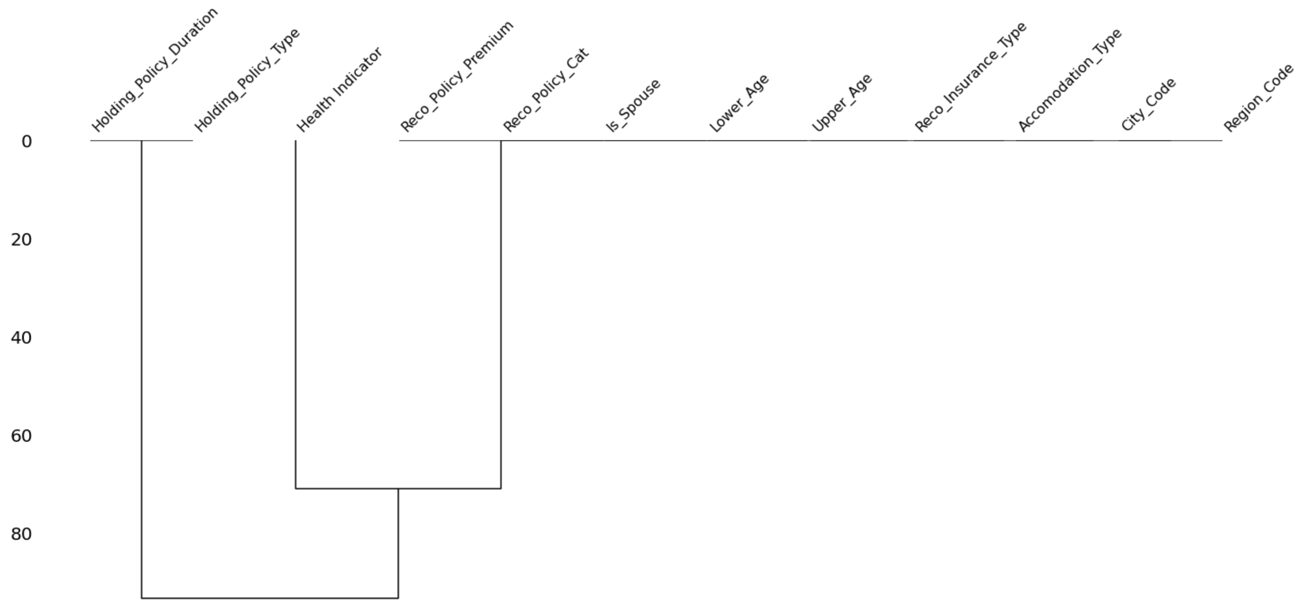
```
ID                         0
City_Code                  0
Region_Code                0
Accomodation_Type          0
Reco_Insurance_Type        0
```

```
Upper_Age                  0
Lower_Age                  0
Is_Spouse                  0
Health Indicator        5027
Holding_Policy_Duration 8603
Holding_Policy_Type     8603
Reco_Policy_Cat            0
Reco_Policy_Premium        0
dtype: int64
```

Dendrogram

```
msno.dendrogram(test.iloc[:,1:13])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3f75873ad0>
```



The above dendrogram shows that the test dataset also follows the same missing pattern as train data.

Missing Value Imputation

For the missing value imputation the category vairables should be encoded as numbers.

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50882 entries, 0 to 50881
Data columns (total 14 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   ID                       50882 non-null  int64  
 1   City_Code                50882 non-null  object 
 2   Region_Code              50882 non-null  int64  
 3   Accomodation_Type        50882 non-null  object 
 4   Reco_Insurance_Type      50882 non-null  object 
 5   Upper_Age                50882 non-null  int64  
 6   Lower_Age                50882 non-null  int64  
 7   Is_Spouse                50882 non-null  object 
 8   Health Indicator         39191 non-null  object 
 9   Holding_Policy_Duration  30631 non-null  object 
 10  Holding_Policy_Type      30631 non-null  float64
 11  Reco_Policy_Cat          50882 non-null  int64  
 12  Reco_Policy_Premium      50882 non-null  float64
 13  Response                 50882 non-null  int64  
dtypes: float64(2), int64(6), object(6)
memory usage: 5.4+ MB
```

```
train.select_dtypes(include='object')
```

	City_Code	Accomodation_Type	Reco_Insurance_Type	Is_Spouse	Health Indicator	Hold
0	C3	Rented	Individual	No	X1	
1	C5	Owned	Joint	No	X2	
2	C5	Owned	Individual	No	NaN	
3	C24	Owned	Joint	No	X1	
4	C8	Rented	Individual	No	X2	
...	...	...	...	...	...	
50877	C4	Rented	Individual	No	X3	
50878	C5	Rented	Individual	No	X3	
50879	C1	Rented	Individual	No	X2	
50880	C1	Owned	Joint	No	X2	
50881	C3	Rented	Individual	No	X3	

Before imputing missing values, a separate missing value indicator to be created for the column that has missing values.

```
train.select_dtypes(include='object').columns.values
```

```
array(['City_Code', 'Accomodation_Type', 'Reco_Insurance_Type',
       'Is_Spouse', 'Health Indicator', 'Holding_Policy_Duration'],
      dtype=object)
```

```
train['HI_miss']=train['Health Indicator'].isnull().astype(int)
```

```
train['Hpd_miss']=train['Holding_Policy_Duration'].isnull().astype(int)
```

```
train['Hpt_miss']=train['Holding_Policy_Type'].isnull().astype(int)
```

Let's re-arrange the column

```
train=train.iloc[:,np.r_[0:9,14,9,15,10,16,11:14]]
train
```

	ID	City_Code	Region_Code	Accomodation_Type	Reco_Insurance_Type	Upper_
0	1	C3	3213	Rented	Individual	
1	2	C5	1117	Owned	Joint	
2	3	C5	3732	Owned	Individual	
3	4	C24	4378	Owned	Joint	
4	5	C8	2190	Rented	Individual	
...	...	...	...	...	...	
50877	50878	C4	845	Rented	Individual	
50878	50879	C5	4188	Rented	Individual	
50879	50880	C1	442	Rented	Individual	
50880	50881	C1	4	Owned	Joint	
50881	50882	C3	3866	Rented	Individual	

50882 rows × 17 columns

City code column has mixed type values(character and number) so we remove those character and make is as nummerical column.

```
train['City_Code']=train['City_Code'].str.replace(r'\D', '').astype(int)
```

Accomodation Type category column has two levels rented and owned.Let's encode **owned as 1 and rented as 0**

```
acc_encode={'Owned':1,'Rented':0}
```

```
train['Accomodation_Type']=train['Accomodation_Type'].map(acc_encode)
```

Reco Insurance Type category column has two levels individual and joint .Let's encode **joined as 0 and individual as 1**

```
rectype_encode={'Individual':1,'Joint':0}

train['Reco_Insurance_Type']=train['Reco_Insurance_Type'].map(rectype_encode)
```

Is Spouse category column has two levels yes and no .Let's encode **no as 0 and yes as 1**

```
spouse_encode={'Yes':1,'No':0}

train['Is_Spouse']=train['Is_Spouse'].map(spouse_encode)
```

Health Indicator column has mixed type values(character and number) so we remove those character and make is as nummerical column.

```
import re

train['Health Indicator']=train['Health Indicator'].str.replace(r'\D','').astype('int32',e
```

Holding Policy Duration has mixed type values(addition operator symbol and number) so we remove those operator symbol and make is as nummerical column.

```
train['Holding_Policy_Duration']=train['Holding_Policy_Duration'].replace('14+','15.0').as

train['Holding_Policy_Duration']=train['Holding_Policy_Duration'].str.replace(r'\D0','')
```

Let's impute missing value using Knn algorithm

```
from sklearn.impute import KNNImputer

imputer=KNNImputer(n_neighbors=2)

train
```

eco_Insurance_Type	Upper_Age	Lower_Age	Is_Spouse	Health Indicator	HI_miss	Holding_Policy
1	36	36	0	1	0	
0	75	22	0	2	0	
1	32	32	0	NaN	1	
0	52	48	0	1	0	
1	44	44	0	2	0	
...	...	...	...	...	...	
1	22	22	0	3	0	
1	27	27	0	3	0	
1	63	63	0	2	0	
0	71	49	0	2	0	
1	24	24	0	3	0	

```
X_imputed = imputer.fit_transform(train.iloc[:,np.r_[8,10,12]])
```

Imputed array

```
X_imputed=pd.DataFrame(X_imputed,columns=train.iloc[:,np.r_[8,10,12]].columns)
```

```
X_imputed
```

	Health Indicator	Holding_Policy_Duration	Holding_Policy_Type
0	1.0	15.0	3.0
1	2.0	4.0	1.0
2	1.5	1.0	1.0
3	1.0	15.0	3.0
4	2.0	3.0	1.0
...	...	...	...
50877	3.0	2.5	3.0
50878	3.0	7.0	3.0
50879	2.0	15.0	1.0
50880	2.0	2.0	2.0
50881	3.0	2.0	3.0

50882 rows × 3 columns

Write imputed Health Indicator value to the train dataset column.

```
train['Health Indicator']=X_imputed['Health Indicator'].astype('int').round()
```

Write imputed Holding Policy Duration value to the train dataset column.

```
train['Holding_Policy_Duration']=X_imputed['Holding_Policy_Duration'].astype('int').round(
```

Write imputed Holding Policy value Type to the train dataset column.

```
train['Holding_Policy_Type']=X_imputed['Holding_Policy_Type'].astype('int').round()
```

```
y = train['Response']
x = train.iloc[:,np.r_[1:16]]
```

Split data into train and test

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

```
xtrain,xtest,ytrain,ytest = train_test_split(x,y,train_size=0.99,random_state=1236)
```

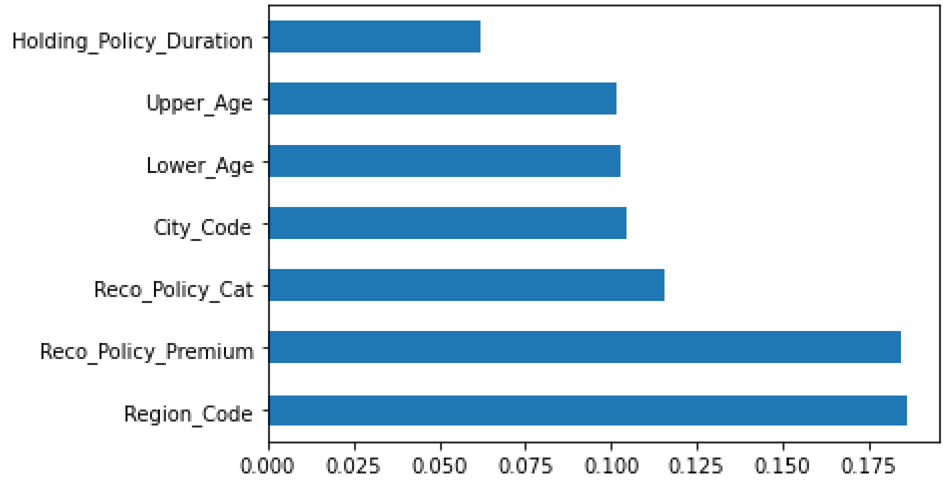
Simple random forest model

```
# importing random forest classifier from assemble module
from sklearn.ensemble import RandomForestClassifier

# creating a RF classifier
clf = RandomForestClassifier(n_estimators = 500)

# Training the model on the training dataset
# fit function is used to train the model using the training sets as parameters
clf.fit(xtrain,ytrain)

# performing predictions on the test dataset
y_pred = clf.predict(xtest)

# metrics are used to find accuracy or error
from sklearn import metrics
print()

# using metrics module for accuracy calculation
print("ACCURACY OF THE MODEL: ", metrics.roc_auc_score(ytest, y_pred))
```

```
ACCURACY OF THE MODEL:  0.5060761603930859
```

```
pd.Series(clf.feature_importances_,index=xtrain.columns).nlargest(7).plot(kind='barh')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3f5569d550>
```



The accuarancy of the model is 0.50

Let's try xgboost model with random search

```
import xgboost as xgb
model=xgb.XGBClassifier(random_state=1,learning_rate=0.4,eval_metric='auc')
model.fit(xtrain, ytrain)
model.score(xtest,ytest)
```

```
0.7721021611001965
```

```
# A parameter grid for XGBoost
params = {
        'min_child_weight': [1, 5, 10],
        'gamma': [0.5, 1, 1.5, 2, 5],
        'subsample': [0.6, 0.8, 1.0],
        'colsample_bytree': [0.6, 0.8, 1.0],
        'max_depth': [3, 4, 5]
        }
```

```
from xgboost import XGBClassifier
xgb = XGBClassifier(learning_rate=0.02, n_estimators=600, objective='binary:logistic',
                    silent=True, nthread=1)
```

```
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import StratifiedKFold

folds = 3
```

```
folds = 3
param_comb = 5

skf = StratifiedKFold(n_splits=folds, shuffle = True, random_state = 1001)

random_search = RandomizedSearchCV(xgb, param_distributions=params, n_iter=param_comb, sco

random_search.fit(xtrain, ytrain)
```

```
Fitting 3 folds for each of 5 candidates, totalling 15 fits
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  15 out of  15 | elapsed:  3.3min finished
RandomizedSearchCV(cv=<generator object _BaseKFold.split at 0x7f3f6fad5ed0>,
                   error_score=nan,
                   estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                           colsample_bylevel=1,
                                           colsample_bynode=1,
                                           colsample_bytree=1, gamma=0,
                                           learning_rate=0.02, max_delta_step=0,
                                           max_depth=3, min_child_weight=1,
                                           missing=None, n_estimators=600,
                                           n_jobs=1, nthread=1,
                                           objective='binary:logist...
                                           reg_lambda=1, scale_pos_weight=1,
                                           seed=None, silent=True, subsample=1,
                                           verbosity=1),
                   iid='deprecated', n_iter=5, n_jobs=4,
                   param_distributions={'colsample_bytree': [0.6, 0.8, 1.0],
                                        'gamma': [0.5, 1, 1.5, 2, 5],
                                        'max_depth': [3, 4, 5],
                                        'min_child_weight': [1, 5, 10],
                                        'subsample': [0.6, 0.8, 1.0]},
                   pre_dispatch='2*n_jobs', random_state=1001, refit=True,
                   return_train_score=False, scoring='roc_auc', verbose=3)
```

```
print(random_search.best_score_)
```

```
0.6570303307672747
```

```
print("The best parameters\n"+"_"*100+"\n\n",random_search.best_params_)
```

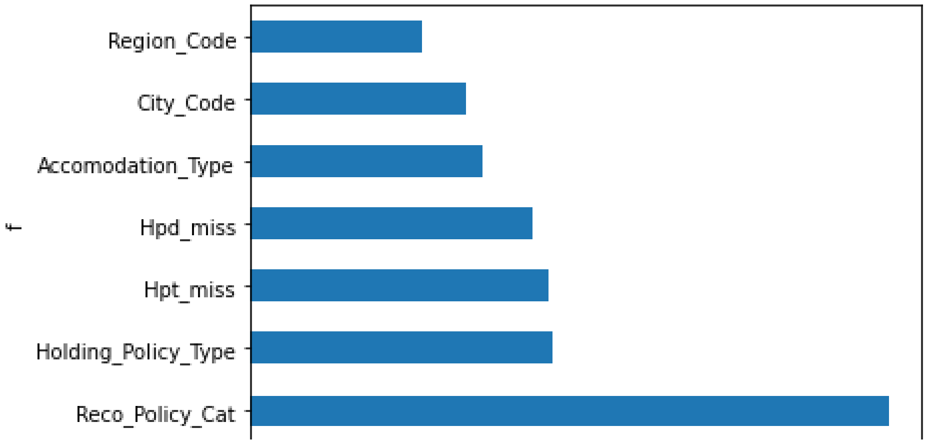
```
The best parameters
____________________________________________________________________________________________________

 {'subsample': 0.8, 'min_child_weight': 5, 'max_depth': 5, 'gamma': 1, 'colsample_by
```

```
feat_import=pd.Series(random_search.best_estimator_.feature_importances_,index=xtrain.colu
```

Feature Importance Plot

```
feat_import.nlargest(7).plot(kind='barh')
```

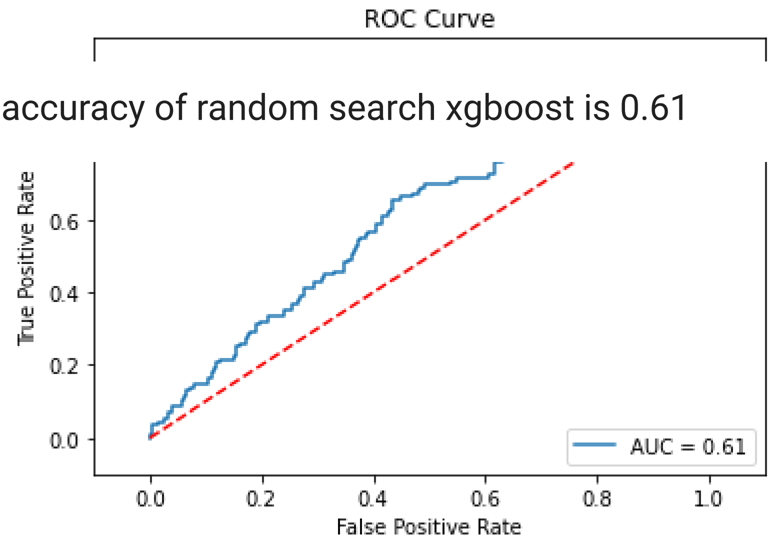
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3f55607d90>
```




```
pred = random_search.predict_proba(xtest)[:,1]
```

ROC Plot

```
fpr, tpr, _ = metrics.roc_curve(ytest, pred)

auc_score = metrics.auc(fpr, tpr)

# clear current figure
plt.clf()

plt.title('ROC Curve')
plt.plot(fpr, tpr, label='AUC = {:.2f}'.format(auc_score))

# it's helpful to add a diagonal to indicate where chance
# scores lie (i.e. just flipping a coin)
plt.plot([0,1],[0,1],'r--')

plt.xlim([-0.1,1.1])
plt.ylim([-0.1,1.1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')

plt.legend(loc='lower right')
plt.show()
```



The accuracy of random search xgboost is 0.61