Bagging

Bagging (Bootstrap Aggregation) is used when our goal is to reduce the variance of a decision tree. Here idea is to create several subsets of data from training sample chosen randomly with replacement. Average of all the predictions from different trees are used which is more robust than a single decision tree.

In [5]:
```
import pandas as pd, numpy as np, matplotlib.pyplot as plt, os, sys, seabor
```

Customer Churn Data of Telecom industry is used for Study Bagging.

In [8]:
```
filepath = 'churndata_processed.csv'
data = pd.read_csv(filepath)
```

In [12]:
```
data.describe().T
```

Out[12]:

	count	mean	std	min	25%	50%	75%	max
months	7043.0	0.433551	0.398231	0.0	0.000000	0.250000	0.750000	1.0
multiple	7043.0	0.421837	0.493888	0.0	0.000000	0.000000	1.000000	1.0
gb_mon	7043.0	0.241358	0.240223	0.0	0.035294	0.200000	0.317647	1.0
security	7043.0	0.286668	0.452237	0.0	0.000000	0.000000	1.000000	1.0
backup	7043.0	0.344881	0.475363	0.0	0.000000	0.000000	1.000000	1.0
protection	7043.0	0.343888	0.475038	0.0	0.000000	0.000000	1.000000	1.0
support	7043.0	0.290217	0.453895	0.0	0.000000	0.000000	1.000000	1.0
unlimited	7043.0	0.673719	0.468885	0.0	0.000000	1.000000	1.000000	1.0
contract	7043.0	0.377396	0.424234	0.0	0.000000	0.000000	1.000000	1.0
paperless	7043.0	0.592219	0.491457	0.0	0.000000	1.000000	1.000000	1.0
monthly	7043.0	0.462803	0.299403	0.0	0.171642	0.518408	0.712438	1.0
satisfaction	7043.0	0.561231	0.300414	0.0	0.500000	0.500000	0.750000	1.0
churn_value	7043.0	0.265370	0.441561	0.0	0.000000	0.000000	1.000000	1.0
payment_Credit Card	7043.0	0.390317	0.487856	0.0	0.000000	0.000000	1.000000	1.0
payment_Mailed Check	7043.0	0.054664	0.227340	0.0	0.000000	0.000000	0.000000	1.0
internet_type_DSL	7043.0	0.234559	0.423753	0.0	0.000000	0.000000	0.000000	1.0
internet_type_Fiber Optic	7043.0	0.430924	0.495241	0.0	0.000000	0.000000	1.000000	1.0
internet_type_None	7043.0	0.216669	0.412004	0.0	0.000000	0.000000	0.000000	1.0
offer_Offer A	7043.0	0.073832	0.261516	0.0	0.000000	0.000000	0.000000	1.0
offer_Offer B	7043.0	0.116996	0.321438	0.0	0.000000	0.000000	0.000000	1.0
offer_Offer C	7043.0	0.058924	0.235499	0.0	0.000000	0.000000	0.000000	1.0
offer_Offer D	7043.0	0.085475	0.279607	0.0	0.000000	0.000000	0.000000	1.0
offer_Offer E	7043.0	0.114298	0.318195	0.0	0.000000	0.000000	0.000000	1.0

In [15]:
```
data.corr()
```

Out[15]:

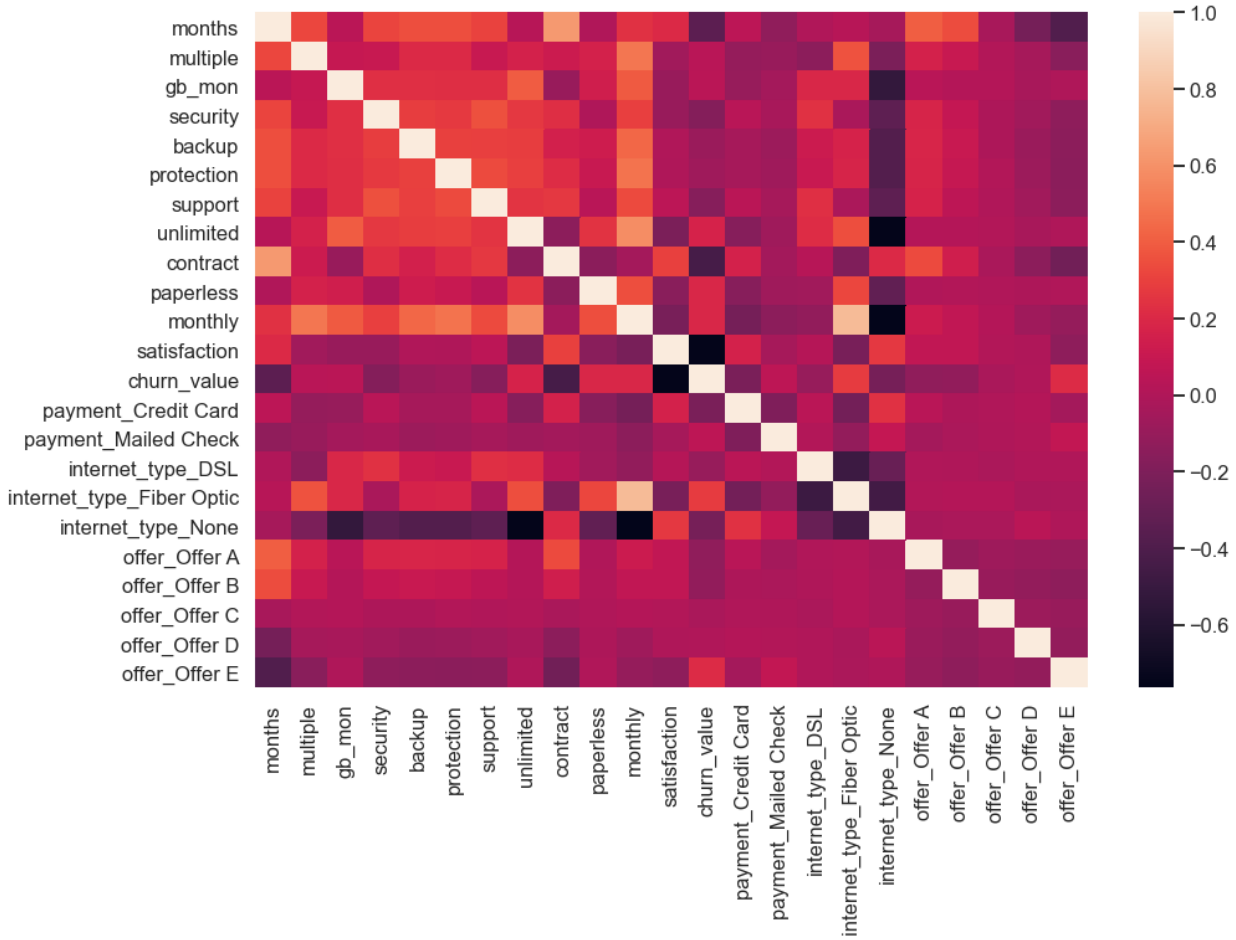
	months	multiple	gb_mon	security	backup	protection	support	unl
months	1.000000	0.321235	0.046863	0.315527	0.349682	0.350511	0.310377	0.0
multiple	0.321235	1.000000	0.091679	0.098108	0.202237	0.201137	0.100571	0.1
gb_mon	0.046863	0.091679	1.000000	0.234738	0.229254	0.225877	0.223924	0.3
security	0.315527	0.098108	0.234738	1.000000	0.283832	0.275438	0.354931	0.2
backup	0.349682	0.202237	0.229254	0.283832	1.000000	0.303546	0.294233	0.2
protection	0.350511	0.201137	0.225877	0.275438	0.303546	1.000000	0.333313	0.2
support	0.310377	0.100571	0.223924	0.354931	0.294233	0.333313	1.000000	0.2
unlimited	0.031219	0.159669	0.395209	0.265037	0.283855	0.296619	0.251496	1.0
contract	0.629463	0.120475	-0.093588	0.228371	0.159355	0.217698	0.269251	-0.1
paperless	0.003924	0.163530	0.142999	-0.003636	0.126735	0.103797	0.037880	0.2
monthly	0.241080	0.490434	0.391787	0.296594	0.441780	0.482692	0.338304	0.5
satisfaction	0.204274	-0.054236	-0.091657	-0.093158	0.003252	-0.000549	0.050086	-0.2
churn_value	-0.337205	0.040102	0.048868	-0.171226	-0.082255	-0.066160	-0.164674	0.1
payment_Credit Card	0.055491	-0.102332	-0.096806	0.041805	-0.035564	-0.038203	0.042449	-0.1
payment_Mailed Check	-0.123784	-0.091576	-0.043116	-0.029513	-0.081179	-0.062323	-0.036790	-0.0
internet_type_DSL	0.007804	-0.141726	0.189763	0.240403	0.120729	0.110683	0.229289	0.2
internet_type_Fiber Optic	0.029646	0.359797	0.191512	-0.024751	0.167258	0.175230	-0.015673	0.3
internet_type_None	-0.037735	-0.210564	-0.528450	-0.333403	-0.381593	-0.380754	-0.336298	-0.7
offer_Offer A	0.401638	0.163428	0.038480	0.177625	0.181239	0.177382	0.169983	0.0
offer_Offer B	0.340577	0.101441	0.018397	0.089664	0.105777	0.091731	0.058263	0.0
offer_Offer C	-0.026761	0.014575	0.021679	-0.009290	-0.007771	0.010519	0.004729	0.0
offer_Offer D	-0.234338	-0.039021	-0.030202	-0.057919	-0.081859	-0.074860	-0.057860	-0.0
offer_Offer E	-0.391121	-0.154137	-0.002380	-0.132006	-0.136721	-0.147336	-0.143183	0.0

23 rows × 23 columns

Satisfaction is inversly correlated to churn value.If a customer is more satisfied he is more likely to stay Tenture in months positively corelated.

In [37]:
```
fig, ax = plt.subplots(figsize=(15,10))
sns.heatmap(data.corr())
```

Out[37]: <AxesSubplot:>



Examining the Target and Preprocessing

- Examine distribution of the predicted variable (`churn_value`).
- Split the data into train and test sets. Decide if a stratified split should be used or not based on the distribution.
- Examine the distribution of the predictor variable in the train and test data.

In [38]:
```
# Data are skewed at ~85% towards non-churned customers
# This will be important to remember when model building
target = 'churn_value'
data[target].value_counts()
```

Out[38]:
```
0    5174
1    1869
Name: churn_value, dtype: int64
```

In [39]:
```
data[target].value_counts(normalize=True)
```

Out[39]:
```
0    0.73463
1    0.26537
Name: churn_value, dtype: float64
```

In [40]:
```
from sklearn.model_selection import StratifiedShuffleSplit


feature_cols = [x for x in data.columns if x != target]


# Split the data into two parts with 1500 points in the test data
# This creates a generator
strat_shuff_split = StratifiedShuffleSplit(n_splits=1, test_size=1500, rand

# Get the index values from the generator
train_idx, test_idx = next(strat_shuff_split.split(data[feature_cols], data

# Create the data sets
X_train = data.loc[train_idx, feature_cols]
y_train = data.loc[train_idx, target]

X_test = data.loc[test_idx, feature_cols]
y_test = data.loc[test_idx, target]
```

In [41]:
```
y_train.value_counts(normalize=True)
```

Out[41]:
```
0    0.73462
1    0.26538
Name: churn_value, dtype: float64
```

In [42]:
```
y_test.value_counts(normalize=True)
```

Out[42]:
```
0    0.734667
1    0.265333
Name: churn_value, dtype: float64
```

Random Forest and Out-of-bag Error

- Fit random forest models with a range of tree numbers and evaluate the out-of-bag error for each of these models.
- Plot the resulting oob errors as a function of the number of trees.

In [22]:
```
# Suppress warnings about too few trees from the early models
import warnings
warnings.filterwarnings("ignore", category=UserWarning)
warnings.filterwarnings("ignore", category=RuntimeWarning)
```

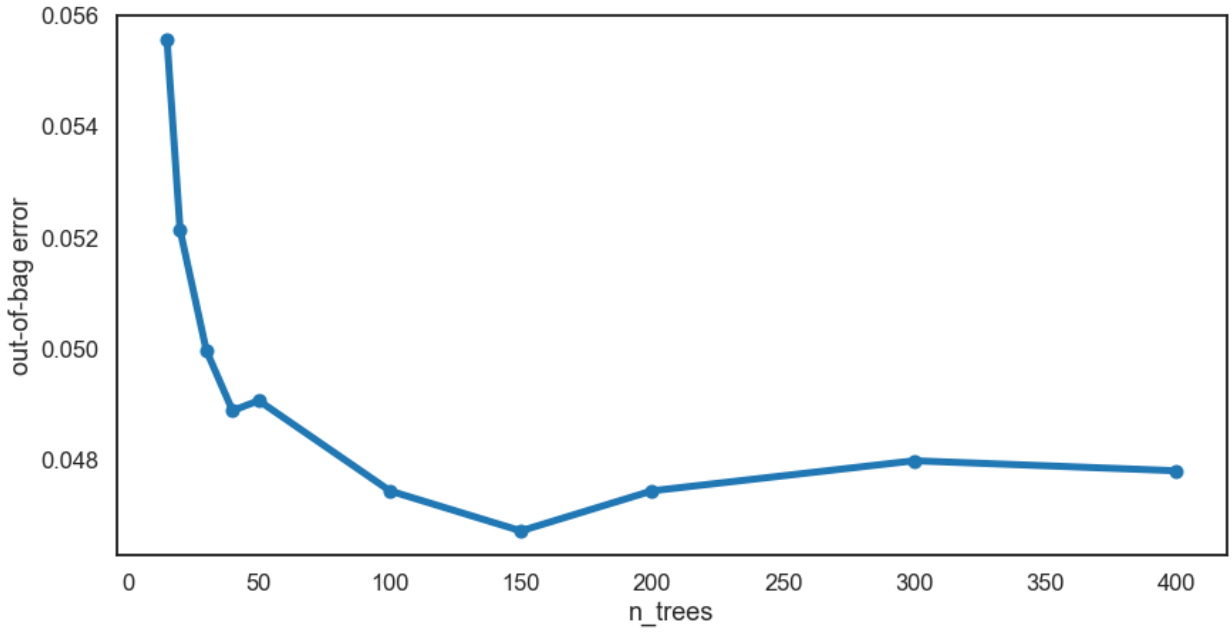
In [23]:
```
from sklearn.ensemble import RandomForestClassifier

# Initialize the random forest estimator
# Note that the number of trees is not setup here
RF = RandomForestClassifier(oob_score=True,
                            random_state=42,
                            warm_start=True,
                            n_jobs=-1)

oob_list = list()

# Iterate through all of the possibilities for
# number of trees
for n_trees in [15, 20, 30, 40, 50, 100, 150, 200, 300, 400]:

    # Use this to set the number of trees
    RF.set_params(n_estimators=n_trees)

    # Fit the model
    RF.fit(X_train, y_train)

    # Get the oob error
    oob_error = 1 - RF.oob_score_

    # Store it
    oob_list.append(pd.Series({'n_trees': n_trees, 'oob': oob_error}))

rf_oob_df = pd.concat(oob_list, axis=1).T.set_index('n_trees')

rf_oob_df
```

Out[23]:

n_trees	oob
15.0	0.055566
20.0	0.052138
30.0	0.049973
40.0	0.048890
50.0	0.049071
100.0	0.047447
150.0	0.046726
200.0	0.047447
300.0	0.047988
400.0	0.047808

In [24]:
```
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
```

In [25]:
```
sns.set_context('talk')
sns.set_style('white')

ax = rf_oob_df.plot(legend=False, marker='o', figsize=(14, 7), linewidth=5)
ax.set(ylabel='out-of-bag error');
```



Gathering Results

- Model performs well at 100 and calculate error metrics and a confusion matrix on the test data set.

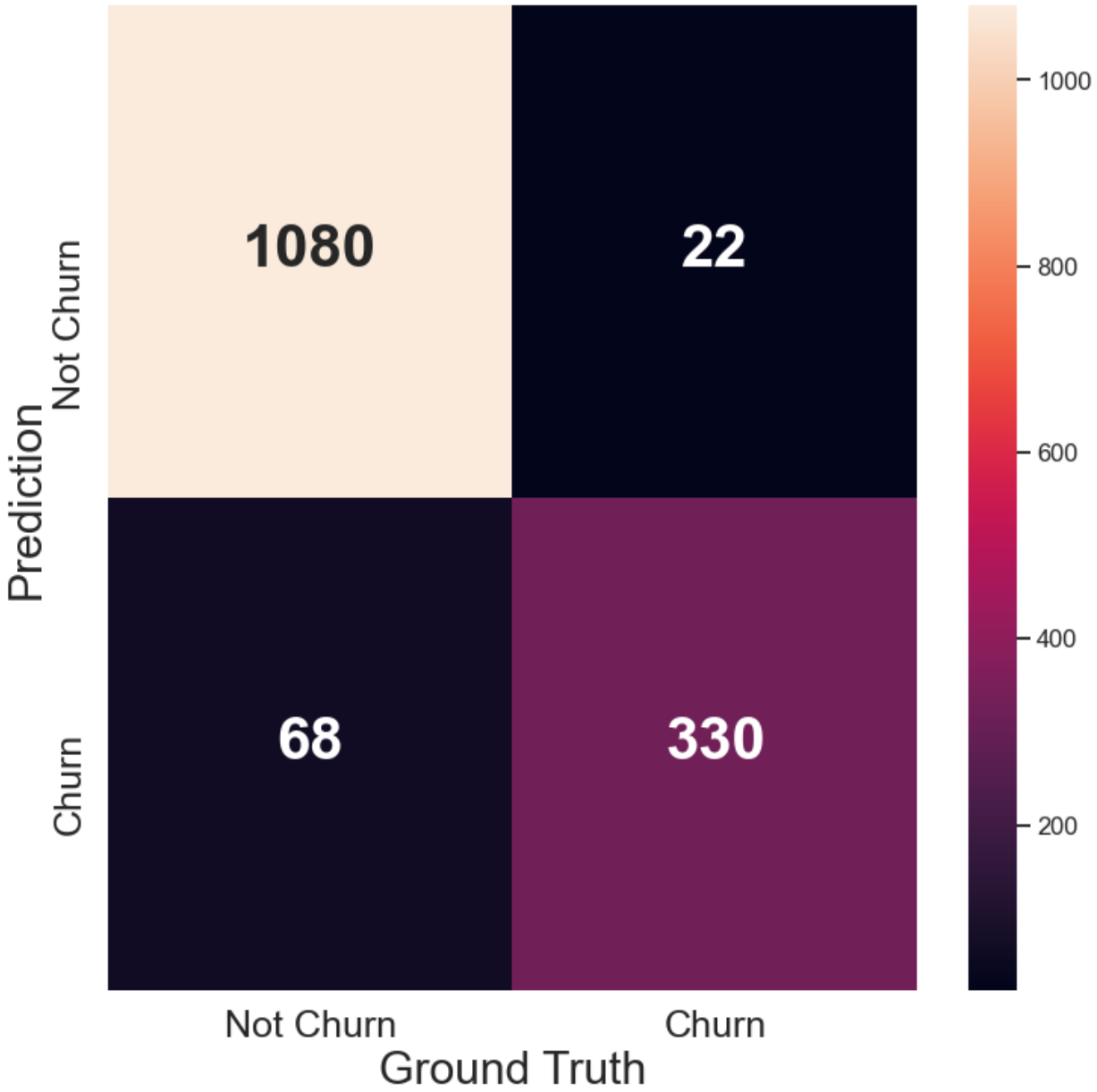
In [43]:
```
# Random forest with 100 estimators
model = RF.set_params(n_estimators=100)

y_pred = model.predict(X_test)
```

In [44]:
```
from sklearn.metrics import classification_report, accuracy_score, precisio
from sklearn.metrics import f1_score, roc_auc_score

cr = classification_report(y_test, y_pred)
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.94      0.98      0.96      1102
           1       0.94      0.83      0.88       398

    accuracy                           0.94      1500
   macro avg       0.94      0.90      0.92      1500
weighted avg       0.94      0.94      0.94      1500
```

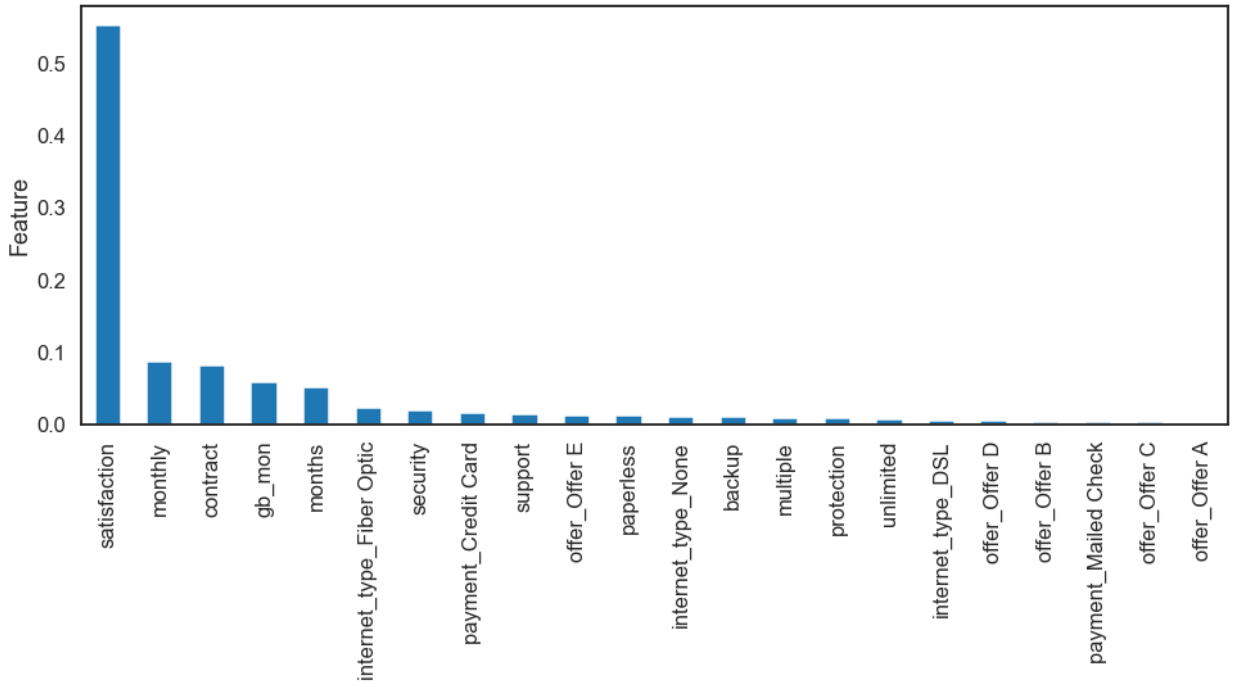
In [45]:
```
from sklearn.metrics import roc_curve, precision_recall_curve, confusion_ma

sns.set_context('talk')
cm = confusion_matrix(y_test, y_pred)
_, ax = plt.subplots(figsize=(12,12))
ax = sns.heatmap(cm, annot=True, fmt='d', annot_kws={"size": 40, "weight":

ylabels = ['Not Churn', 'Churn']
xlabels = ['Not Churn', 'Churn']
ax.set_xticklabels(xlabels, fontsize=25);
ax.set_yticklabels(ylabels, fontsize=25);
ax.set_ylabel('Prediction', fontsize=30);
ax.set_xlabel('Ground Truth', fontsize=30)
```

Out[45]:
```
Text(0.5, 76.5, 'Ground Truth')
```



Feature Importance

In [47]:
```
feature_imp = pd.Series(model.feature_importances_, index=feature_cols).sor

ax = feature_imp.plot(kind='bar', figsize=(16, 6))
ax.set(ylabel='Relative Importance');
ax.set(ylabel='Feature');
```



Summary :

We can also use ExtraTreesClassifier and compare results. Recall is not a good option for prediction here.From Feature importance we can easily interpret whisch feature is really important.

In [ ]: