```
# Balancing dataset using SMOTE oversampling method
# Then applied Logistic Regression, Naive Bayes Classifier, and Decision Tree algorithms
# Compared changes in accuracy rates when 10-fold cross-validation applied
```

```
import sys
!{sys.executable} -m pip install -U pandas-profiling[notebook]
!jupyter nbextension enable --py widgetsnbextension
!pip install matplotlib
!pip install graphviz
```

```
Requirement already satisfied: pandas-profiling[notebook] in /usr/local/lib/python3.7/di
WARNING: pandas-profiling 1.4.1 does not provide the extra 'notebook'
Requirement already satisfied: jinja2>=2.8 in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: six>=1.9 in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: matplotlib>=1.4 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pandas>=0.19 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: numpy>=1.11 in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/li
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (f
Enabling notebook extension jupyter-js-widgets/extension...
Paths used for configuration of notebook:
        /root/.jupyter/nbconfig/notebook.json
      - Validating: OK
Paths used for configuration of notebook:
        /root/.jupyter/nbconfig/notebook.json
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packages (3.2
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/li
Requirement already satisfied: numpy>=1.11 in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: graphviz in /usr/local/lib/python3.7/dist-packages (0.10
```

```
from google.colab import files
uploaded = files.upload()
```

Choose files stroke preprocessed.arff

- **stroke preprocessed.arff**(n/a) - 368681 bytes, last modified: 06/03/2022 - 100% done

```
Saving stroke preprocessed.arff to stroke preprocessed.arff
```

```
import pandas as pd
from scipy.io import arff
```

```
import numpy as np

# Timing how long predictors take to run for efficiency calculations
# Import libraries
import time


data_file = "stroke preprocessed.arff"
data = arff.loadarff(data_file)


df = pd.DataFrame(data[0])
for col in df.columns:
  if df[col].dtype == 'object':
    # Ensure data isn't read as bytes but rather as strings from file
    df[col] = df[col].str.decode('utf-8')
# Examine data types
print(df.dtypes)
```

```
     "id"                   float64
     "gender"                object
     "age"                  float64
     "hypertension"          object
     "heart_disease"         object
     "ever_married"          object
     "work_type"             object
     "residence_type"        object
     "avg_glucose_level"    float64
     "bmi"                  float64
     "smoking_status"        object
     "stroke"                object
     dtype: object
```

```
# Display first 10 rows
df.head(10)
```

	"id"	"gender"	"age"	"hypertension"	"heart_disease"	"ever_married"	"work_type
0	9046.0	Male	67.0	0	1	Yes	Privat
1	51676.0	Female	61.0	0	0	Yes	Sel employe
2	31112.0	Male	80.0	0	1	Yes	Privat

```
# Examine meta info about data
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   "id"                 5110 non-null   float64
 1   "gender"             5110 non-null   object 
 2   "age"                5110 non-null   float64
 3   "hypertension"       5110 non-null   object 
 4   "heart_disease"      5110 non-null   object 
 5   "ever_married"       5110 non-null   object 
 6   "work_type"          5110 non-null   object 
 7   "residence_type"     5110 non-null   object 
 8   "avg_glucose_level"  5110 non-null   float64
 9   "bmi"                5110 non-null   float64
 10  "smoking_status"     5110 non-null   object 
 11  "stroke"             5110 non-null   object 
dtypes: float64(4), object(8)
memory usage: 479.2+ KB
```

```
# The original 201 null values were all from bmi column, and they have been replaced by place
# Convert the 5000 values back into null values
df = df.replace(5000.0, np.nan)
```

```
# Check head of dataset again
df.head(10)
```

	"id"	"gender"	"age"	"hypertension"	"heart_disease"	"ever_married"	"work_type
0	9046.0	Male	67.0	0	1	Yes	Privat
1	51676.0	Female	61.0	0	0	Yes	Sel employe
2	31112.0	Male	80.0	0	1	Yes	Privat
3	60182.0	Female	49.0	0	0	Yes	Privat
4	1665.0	Female	79.0	1	0	Yes	Sel

```
# Check structure of data types to ensure bmi remains float
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   "id"                 5110 non-null   float64
 1   "gender"             5110 non-null   object
 2   "age"                5110 non-null   float64
 3   "hypertension"       5110 non-null   object
 4   "heart_disease"      5110 non-null   object
 5   "ever_married"       5110 non-null   object
 6   "work_type"          5110 non-null   object
 7   "residence_type"     5110 non-null   object
 8   "avg_glucose_level"  5110 non-null   float64
 9   "bmi"                4909 non-null   float64
 10  "smoking_status"     5110 non-null   object
 11  "stroke"             5110 non-null   object
dtypes: float64(4), object(8)
memory usage: 479.2+ KB
```

```
# Remove records with NAs from dataset
df_noNA = df
df_noNA = df_noNA.dropna()
df_noNA.head(10)
```

	"id"	"gender"	"age"	"hypertension"	"heart_disease"	"ever_married"	"work_typ
0	9046.0	Male	67.0	0	1	Yes	Priva
2	31112.0	Male	80.0	0	1	Yes	Priva
3	60182.0	Female	49.0	0	0	Yes	Priva
4	1665.0	Female	79.0	1	0	Yes	Se employ
5	56669.0	Male	81.0	0	0	Yes	Priva

```
# See if data is imbalanced on the variable of interest, stroke
# Count how many '1's (stroke) and '0's (no stroke) appear
print(df_noNA['"stroke"'].value_counts())
# Dataset is quite unbalanced on the stroke variable
# Around 209/(4700+209) = 4.3% of dataset is positive for stroke
```

```
0    4700
1     209
Name: "stroke", dtype: int64
```

```
# Change 'stroke' attribute into data type float
df_noNA['"stroke"'] = df_noNA['"stroke"'].astype(float)
df_noNA.head(10)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
print(df_noNA.dtypes)
```

```
"id"                   float64
"gender"                object
"age"                  float64
"hypertension"          object
"heart_disease"         object
"ever_married"          object
"work_type"             object
"residence_type"        object
"avg_glucose_level"    float64
"bmi"                  float64
"smoking_status"        object
"stroke"               float64
dtype: object
```

```
# See if there are any extreme values in numeric data
df_noNA.describe()
```

	"id"	"age"	"avg_glucose_level"	"bmi"	"stroke"
count	4909.000000	4909.000000	4909.000000	4909.000000	4909.000000
mean	37064.313506	42.865374	105.305150	28.893237	0.042575
std	20995.098457	22.555115	44.424341	7.854067	0.201917
min	77.000000	0.080000	55.120000	10.300000	0.000000
25%	18605.000000	25.000000	77.070000	23.500000	0.000000
50%	37608.000000	44.000000	91.680000	28.100000	0.000000
75%	55220.000000	60.000000	113.570000	33.100000	0.000000
max	72940.000000	82.000000	271.740000	97.600000	1.000000

```
# Normalize continuous numeric variables
# Such as age, avg_glucose_level, and bmi
# Using z-score methods

# Import libraries for normalization
from sklearn import preprocessing
scaler = preprocessing.MinMaxScaler()

# Only need to normalize continuous numeric variables
var_to_norm = ['"age"', '"avg_glucose_level"', '"bmi"']
df_noNA[var_to_norm] = scaler.fit_transform(df_noNA[var_to_norm])

# Examine first 10 rows of normalized dataset
```

```
df_noNA.head()

# The 3 columns are now standarized to values between 0-1
```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/frame.py:3678: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
  self[col] = igetitem(value, i)
```

	"id"	"gender"	"age"	"hypertension"	"heart_disease"	"ever_married"	"work_t
0	9046.0	Male	0.816895	0	1	Yes	Pr
2	31112.0	Male	0.975586	0	1	Yes	Pr
3	60182.0	Female	0.597168	0	0	Yes	Pr
4	1665.0	Female	0.963379	1	0	Yes	empl
5	56669.0	Male	0.987793	0	0	Yes	Pr

```
# Create list of categorical columns
cat_cols = ['"gender"', '"hypertension"', '"heart_disease"', '"ever_married"', '"work_type"',


# Create copy of a data frame in memory w/ a different name
df_dummy = df_noNA.copy()
# Convert only categorical feature into dummy/one-hot features
df_dummy = pd.get_dummies(df_noNA, columns = cat_cols, prefix = cat_cols)
# Print dataset
df_dummy
```

	"id"	"age"	"avg_glucose_level"	"bmi"	"stroke"	"gender"_Female	"gend
0	9046.0	0.816895	0.801265	0.301260	1.0	0	
2	31112.0	0.975586	0.234512	0.254296	1.0	0	
3	60182.0	0.597168	0.536008	0.276060	1.0	1	
4	1665.0	0.963379	0.549349	0.156930	1.0	1	
5	56669.0	0.987793	0.605161	0.214204	1.0	0	

```
# Create train test set split
from sklearn.model_selection import train_test_split
```

5106	44873.0	0.987793	0.323516	0.340206	0.0	1

```
# Balancing Dataset: Import libraries needed for SMOTE Oversampling method
from collections import Counter
from imblearn.over_sampling import SMOTE
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

4909 rows × 25 columns

```
print(df_dummy.dtypes)
```

```
"id"                                float64
"age"                               float64
"avg_glucose_level"                 float64
"bmi"                               float64
"stroke"                            float64
"gender"_Female                       uint8
"gender"_Male                         uint8
"gender"_Other                        uint8
"hypertension"_0                      uint8
"hypertension"_1                      uint8
"heart_disease"_0                     uint8
"heart_disease"_1                     uint8
"ever_married"_No                     uint8
"ever_married"_Yes                    uint8
"work_type"_Govt_job                  uint8
"work_type"_Never_worked              uint8
"work_type"_Private                   uint8
"work_type"_Self-employed             uint8
"work_type"_children                  uint8
"residence_type"_Rural                uint8
"residence_type"_Urban                uint8
"smoking_status"_Unknown              uint8
"smoking_status"_formerly smoked      uint8
"smoking_status"_never smoked         uint8
"smoking_status"_smokes               uint8
dtype: object
```

```
# Partition the class of interest, stroke (Dependent Variable), from the Independent Variable
iv_classes = df_dummy.iloc[:,:-1]
```

```
iv_classes = df_dummy.iloc[:,:-1]
dv_class = df_dummy.iloc[:,-1]


# Set class name as "stroke", all other attributes will be used as features
class_col_name = '"stroke"'
# Obtain necessary dummy feature names
dummy_feature_name = df_dummy.columns.values.tolist()
dummy_feature_names = dummy_feature_name[5:]


# 70% training, 30% test set split
x_train, x_test, y_train, y_test = train_test_split(df_dummy.loc[:, dummy_feature_names], df_


# Display the class distribution in its original split
print("Original class split prior to SMOTE oversampling: ", Counter(y_train))
```

```
     Original class split prior to SMOTE oversampling:  Counter({0.0: 3291, 1.0: 145})
```

```
# Implement the SMOTE Oversampler Model
SMOTEOversampler = SMOTE()


# Apply data into Random Undersampler Model
x_train_SMOTE, y_train_SMOTE = SMOTEOversampler.fit_resample(x_train, y_train)


# Display the class distribution after the SMOTE Oversampler is applied
print("Class split after SMOTE Oversampling: ",Counter(y_train_SMOTE))
```

```
     Class split after SMOTE Oversampling:  Counter({0.0: 3291, 1.0: 3291})
```

```
#·Import·required·libraries
from·sklearn.svm·import·SVC
from·sklearn.metrics·import·classification_report,·roc_auc_score


# Evaluate the SMOTE Oversampler
model = SVC()
clf_SMOTEOversampler = model.fit(x_train_SMOTE, y_train_SMOTE)
pred_SMOTEOversampler = clf_SMOTEOversampler.predict(x_test)


# Display ROC AUC score
print("SMOTE Oversampled data - ROC AUC: ", roc_auc_score(y_test, pred_SMOTEOversampler))
```

```
     SMOTE Oversampled data - ROC AUC:  0.5
```

```
start = time.time()


# Import needed libraries for Logistic Regression Model
```

```
from sklearn.linear_model import LogisticRegression

# Begin to Implement Logistic Regression Model
log_regr = LogisticRegression()

# Apply data into Logistic Regression Model
log_regr.fit(x_train, y_train)
y_pred = log_regr.predict(x_test)
```

```
# Obtain Confusion Matrix and Evaluation Metrics for the Logistic Regression Model
from sklearn import metrics
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix
```

```
array([[1409,    0],
       [  64,    0]])
```

```
# Display Evaluation Metrics for Logistic Regression Model
print("Logistic Regression Accuracy:\t", metrics.accuracy_score(y_test, y_pred))
print("Logistic Regression Precision:\t",metrics.precision_score(y_test, y_pred))
print("Logistic Regression Recall:\t",metrics.recall_score(y_test, y_pred))
```

```
Logistic Regression Accuracy:    0.956551255940258
Logistic Regression Precision:   0.0
Logistic Regression Recall:      0.0
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: Undefine
  _warn_prf(average, modifier, msg_start, len(result))
```

```
# Import libraries for cross-validation
from sklearn.model_selection import cross_val_score, cross_val_predict

# 10-Fold Cross Validation for Logistic Regression
cv_lr = cross_val_score(log_regr, df_dummy, df_dummy[class_col_name], cv=10)
print("Cross-validated scores:\t", cv_lr)

# Increased Accuracy score from 0.957 to 1
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Convergenc
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Convergenc
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
```

```
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
Cross-validated scores:  [1.          1.          1.          1.          1.          1.
 1.          1.          0.95723014 0.95918367]
```

```
# Cross validation accuracy for Logistic Regression (R2 score)
predictions = cross_val_predict(log_regr, df_dummy, df_dummy[class_col_name], cv=10)
accuracy = metrics.r2_score(df_dummy[class_col_name], predictions)
print("Cross-Predicted Accuracy for Logistic Regression: ", accuracy)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Convergenc
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Convergenc
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
Cross-Predicted Accuracy for Logistic Regression:  0.7951043469408531
```

```
end = time.time()

print("Time to run Logistic Regression: ", end - start)
```

```
Time to run Logistic Regression:  3.1480841636657715
```

```
start = time.time()

# Naive Bayes modeling
from sklearn.naive_bayes import MultinomialNB


# Create Multinomial NB Classifier
nb = MultinomialNB()


# Train model using training sets
nb.fit(x_train_SMOTE, y_train_SMOTE)
```

```
MultinomialNB()
```

```
# Predict response for test dataset
y_pred = nb.predict(x_test)


# Print Naive Bayes output
print("Number of features used: ", nb.n_features_)
print("Classes: ", nb.classes_)
print("Number of records for classes: ", nb.class_count_)
print("Log prior probability for classes: ", nb.class_log_prior_)
print("Log conditional probability for each feature given a class: ", nb.feature_log_prob_)
```

```
Number of features used:  20
Classes:  [0. 1.]
Number of records for classes:  [3291. 3291.]
Log prior probability for classes:  [-0.69314718 -0.69314718]
Log conditional probability for each feature given a class:  [[ -2.47270743  -2.8400895
   -1.99056696  -5.08288006  -2.97245497  -2.38986067  -3.98660149
   -7.15535293  -2.49664198  -3.8615758   -3.90153906  -2.67927954
   -2.60089142  -3.11914766  -3.7690812   -2.91883388  -3.83311859]
 [ -3.04260823  -2.72915412 -12.73907161  -3.96491332  -2.66402858
   -3.69762338  -2.55636268  -2.63730714  -4.35953259  -2.21559915
  -12.73907161  -2.89257778  -2.53056564 -12.73907161  -2.93711857
   -3.21611268  -2.7126145   -2.46927567  -3.09325436  -2.55059311]]
```

```
# Get Naive Bayes Classifier Confusion matrix
from sklearn.metrics import confusion_matrix
cf = confusion_matrix(y_test, y_pred)
print("Confusion Matrix")
print(cf)
tn, fp, fn, tp = cf.ravel()
print("TP: ", tp, ", FP: ", fp, ", TN: ", tn, ", FN: ", fn)
```

```
Confusion Matrix
[[1357   52]
 [  58    6]]
TP:  6 , FP:  52 , TN:  1357 , FN:  58
```

```
# Get Naive Bayes Classifier Classifier report
from sklearn.metrics import classification_report
from sklearn import metrics

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

```
              precision    recall  f1-score   support

         0.0       0.96      0.96      0.96      1409
         1.0       0.10      0.09      0.10        64

    accuracy                           0.93      1473
   macro avg       0.53      0.53      0.53      1473
```

```
weighted avg       0.92      0.93      0.92      1473
```

```
# Display Evaluation Metrics for Naive Bayes Classifier
print("Naive Bayes Classifier Accuracy:\t", metrics.accuracy_score(y_test, y_pred))
print("Naive Bayes Classifier Precision:\t",metrics.precision_score(y_test, y_pred))
print("Naive Bayes Classifier Recall:\t\t",metrics.recall_score(y_test, y_pred))
```

```
Naive Bayes Classifier Accuracy:         0.9253224711473184
Naive Bayes Classifier Precision:        0.10344827586206896
Naive Bayes Classifier Recall:           0.09375
```

```
# 10-Fold Cross Validation for Naive Bayes Classifier
cv_nb = cross_val_score(nb, df_dummy, df_dummy[class_col_name], cv=10)
print("Cross-validated scores:\t", cv_nb)

# Increased Accuracy score from 0.926 to 0.998
```

```
Cross-validated scores:  [0.99796334 0.99592668 0.99796334 0.99796334 0.99592668 0.9938
 0.99592668 0.99796334 0.99796334 0.99795918]
```

```
# Cross validation accuracy for Naive Bayes Classifier (R2 score)
predictions = cross_val_predict(nb, df_dummy, df_dummy[class_col_name], cv=10)
accuracy = metrics.r2_score(df_dummy[class_col_name], predictions)
print("Cross-Predicted Accuracy for Naive Bayes Classifier: ", accuracy)
```

```
Cross-Predicted Accuracy for Naive Bayes Classifier:  0.9250381757100682
```

```
end = time.time()

print("Time to run Naive Bayes Classifier: ", end - start)
```

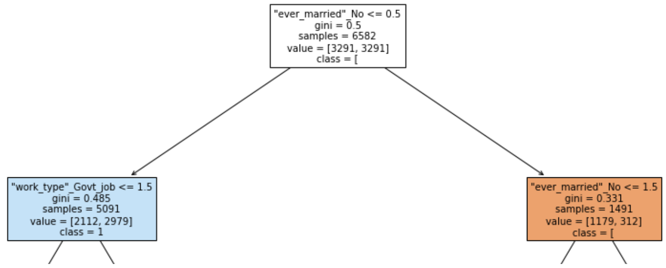
```
Time to run Naive Bayes Classifier:  0.7576520442962646
```

```
start = time.time()

# Decision tree on dummy encoded data
from sklearn import tree
clf = tree.DecisionTreeClassifier(max_depth = 5) # 5 levels set
clf = clf.fit(x_train_SMOTE, y_train_SMOTE)

import graphviz
# Obtain unique class values to show on tree
class_values = df_dummy[class_col_name]. unique()
print("class names: ", class_values)
```

```
class names:  [1. 0.]
```

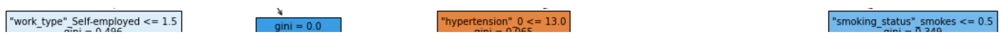
```
# Import libraries for plotting the decision tree
```

```
import matplotlib
from matplotlib import pyplot as plt


# Plot decision tree
fig = plt.figure(figsize=(25,20))
_ = tree.plot_tree(clf, feature_names = dummy_feature_names, class_names = str(class_values),
```




```
# Save decision tree figure
fig.savefig("decision_tree8.png")
```


```
# Perform prediction on test set
y_pred = clf.predict(x_test)
```




```
# Get decision tree confusion matrix
cf = confusion_matrix(y_test, y_pred)
print("Confusion Matrix")
print(cf)
tn, fp, fn, tp = cf.ravel()
print("TP: ", tp, ", FP: ", fp, ", TN: ", tn, ", FN: ", fn)
```

```
Confusion Matrix
[[495 914]
 [  6  58]]
TP:  58 , FP:  914 , TN:  495 , FN:  6
```

```
# Get decision tree report
from sklearn.metrics import classification_report
from sklearn import metrics
print(classification_report(y_test, y_pred))

# Oversampling can lead to overfitting
```

```
              precision    recall  f1-score   support

         0.0       0.99      0.35      0.52      1409
         1.0       0.06      0.91      0.11        64

    accuracy                           0.38      1473
   macro avg       0.52      0.63      0.32      1473
weighted avg       0.95      0.38      0.50      1473
```

```
# Display Evaluation Metrics for Decision Tree
print("Decision Tree Accuracy:\t\t", metrics.accuracy_score(y_test, y_pred))
print("Decision Tree Precision:\t",metrics.precision_score(y_test, y_pred))
print("Decision Tree Recall:\t\t",metrics.recall_score(y_test, y_pred))
```

```
Decision Tree Accuracy:          0.37542430414120 84
Decision Tree Precision:         0.05967078189300 4114
Decision Tree Recall:            0.90625
```

```
# 10-Fold Cross Validation for Decision Tree
cv_dt = cross_val_score(clf, df_dummy, df_dummy[class_col_name], cv=10)
print("Cross-validated scores:\t", cv_dt)

# Increased Accuracy score from 0.375 to 1
```

```
Cross-validated scores:  [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

```
# Cross validation accuracy for Decision Tree (R2 score)
predictions = cross_val_predict(clf, df_dummy, df_dummy[class_col_name], cv=10)
accuracy = metrics.r2_score(df_dummy[class_col_name], predictions)
print("Cross-Predicted Accuracy for Decision Tree: ", accuracy)
```

```
Cross-Predicted Accuracy for Decision Tree:  1.0
```

```
end = time.time()

print("Time to run Decision Tree: ", end - start)
```

```
Time to run Decision Tree:  4.123949289321899
```

✓ 0s completed at 6:21 AM