homework9-skel

November 24, 2020

NAME: Nigel Mansell
SECTION: 995
CS 5970: Machine Learning Practices

1 Homework 9: Decision Tree Classifiers

1.1 Assignment Overview

Follow the TODOs and read through and understand any provided code. Post any questions you might have to the class Slack. For all plots, make sure all necessary axes and curves are clearly and accurately labeled. Include figure/plot titles appropriately as well.

1.1.1 Task

For this assignment you will be exploring Decision Tree Classifiers.

1.1.2 Data set

These data were re-configured from a dataset collected for the purpose of detecting Health care Provider Fraud. Total Medicare spending increases exponentially due to fraud in Medicare claims. Healthcare fraud involves health care providers, physicians, patients, and beneficiaries acting in tandum to construct fraudulent claims.

The goal is to "predict potentially fraudulent providers" from summary statistics of their filed healthcare claims.

Features
The features are aggregate statistics computed as either the mean or the sum. For the following features, the column represents the average value for the provider's claims:
* InscClaimAmtReimbursed
* DeductibleAmtPaid * NoOfMonths_PartACov * NoOfMonths_PartBCov * IPAnnualReimbursementAmt * IPAnnualDeductibleAmt * OPAnnualReimbursementAmt * OPAnnualDeductibleAmt * NumPhysiciansSeen * NumProcedures * NumDiagnosisClaims * Age

For the following features, the column represents the total number among the provider's claims:
* ChronicCond_Alzheimer
* ChronicCond_Heartfailure
* ChronicCond_KidneyDisease
* ChronicCond_Cancer
* ChronicCond_ObstrPulmonary

* ChronicCond_Depression
* ChronicCond_Diabetes
* ChronicCond_IschemicHeart
* ChronicCond_Osteoporasis
* ChronicCond_rheumatoidarthritis
* ChronicCond_stroke
* RenalDiseaseIndicator

These data were amalagmated from the HEALTHCARE PROVIDER FRAUD DETECTION ANALYSIS data set on Kaggle.

1.1.3 Objectives

- Introduction to Decision Trees

1.1.4 Notes

- Please make sure to select "enable scrolling for outputs" for any cell that displays a large section of plots

1.1.5 General References

- Guide to Jupyter
- Python Built-in Functions
- Python Data Structures
- Numpy Reference
- Numpy Cheat Sheet
- Summary of matplotlib
- DataCamp: Matplotlib
- Pandas DataFrames
- Sci-kit Learn Linear Models
- Sci-kit Learn Ensemble Models
- Sci-kit Learn Metrics
- Sci-kit Learn Model Selection
- Sci-kit Learn Pipelines
- Sci-kit Learn Preprocessing

1.1.6 Hand-In Procedure

- Execute all cells so they are showing correct results
- Notebook:
  - Submit this file (.ipynb) to the Canvas HW9 dropbox
- PDF:
  - File/Print/Print to file -> Produces a copy of the notebook in PDF format
  - Submit the PDF file to the Gradescope HW9 dropbox

[1]:
```
# TODO
# THESE FIRST 3 IMPORTS ARE FROM FILES IN THE HW9 FOLDER.  MAKE SURE TO COPY
# THEM TO THE SAME FOLDER AS YOUR NOTEBOOK
import visualize
```

```python
import metrics_plots
from pipeline_components import DataSampleDropper, DataFrameSelector, DataScaler

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import re, os, pathlib
import time as timelib
from IPython.display import Image

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, RobustScaler
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import confusion_matrix, roc_curve, auc
from sklearn.metrics import log_loss, f1_score
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.linear_model import SGDClassifier, LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import DecisionTreeRegressor, export_graphviz
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, OrdinalEncoder
import joblib
import pickle as pkl

FIGW = 5
FIGH = 5
FONTSIZE = 12

plt.rcParams['figure.figsize'] = (FIGW, FIGH)
plt.rcParams['font.size'] = FONTSIZE

plt.rcParams['xtick.labelsize'] = FONTSIZE
plt.rcParams['ytick.labelsize'] = FONTSIZE

%matplotlib inline
plt.style.use('ggplot')
```

[2]:
```python
""" PROVIDED
Display current working directory of this notebook. If you are using
relative paths for your data, then it needs to be relative to the CWD.
"""
HOME_DIR = pathlib.Path.home()
pathlib.Path.cwd()
```

[2]: PosixPath('/home/nigel/Desktop/mlp/homework9')

2 LOAD DATA

[3]:
```
# TODO: set path appropriately.
fname = "/home/nigel/Desktop/mlp/homework9/health_provider_fraud.csv"
claims_data = pd.read_csv(fname)
claims_data.shape
```

[3]:
```
(5410, 25)
```

[4]:
```
""" PROVIDED
Display data info
"""
claims_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5410 entries, 0 to 5409
Data columns (total 25 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Provider                        5410 non-null   object
 1   PotentialFraud                  5410 non-null   bool
 2   Age                             5410 non-null   float64
 3   NumPhysiciansSeen               5410 non-null   float64
 4   NumProcedures                   5410 non-null   float64
 5   NumDiagnosisClaims              5410 non-null   float64
 6   InscClaimAmtReimbursed          5410 non-null   float64
 7   DeductibleAmtPaid               5409 non-null   float64
 8   NoOfMonths_PartACov             5410 non-null   float64
 9   NoOfMonths_PartBCov             5410 non-null   float64
 10  IPAnnualReimbursementAmt        5410 non-null   float64
 11  IPAnnualDeductibleAmt           5410 non-null   float64
 12  OPAnnualReimbursementAmt        5410 non-null   float64
 13  OPAnnualDeductibleAmt           5410 non-null   float64
 14  ChronicCond_Alzheimer           5410 non-null   int64
 15  ChronicCond_Heartfailure        5410 non-null   int64
 16  ChronicCond_KidneyDisease       5410 non-null   int64
 17  ChronicCond_Cancer              5410 non-null   int64
 18  ChronicCond_ObstrPulmonary      5410 non-null   int64
 19  ChronicCond_Depression          5410 non-null   int64
 20  ChronicCond_Diabetes            5410 non-null   int64
 21  ChronicCond_IschemicHeart       5410 non-null   int64
 22  ChronicCond_Osteoporasis        5410 non-null   int64
 23  ChronicCond_rheumatoidarthritis 5410 non-null   int64
 24  ChronicCond_stroke              5410 non-null   int64
dtypes: bool(1), float64(12), int64(11), object(1)
memory usage: 1019.8+ KB
```

[5]:
```python
""" PROVIDED
Display the head of the data
"""
claims_data.head()
```

[5]:
```
  Provider  PotentialFraud        Age  NumPhysiciansSeen  NumProcedures  \
0 PRV51001           False  78.840000           1.280000       0.120000
1 PRV51003            True  70.022727           1.181818       0.363636
2 PRV51004           False  72.161074           1.322148       0.000000
3 PRV51005            True  70.475536           1.209442       0.000000
4 PRV51007           False  69.291667           1.125000       0.013889

  NumDiagnosisClaims  InscClaimAmtReimbursed  DeductibleAmtPaid  \
0           3.640000             4185.600000         213.600000
1           5.765152             4588.409091         502.166667
2           2.751678              350.134228           2.080537
3           2.786266              241.124464           3.175966
4           3.208333              468.194444          45.333333

  NoOfMonths_PartACov  NoOfMonths_PartBCov  …  ChronicCond_Heartfailure  \
0           12.000000            12.000000  …                        19
1           11.818182            11.871212  …                        80
2           11.865772            11.959732  …                        88
3           11.907296            11.939914  …                       680
4           11.833333            11.833333  …                        40

  ChronicCond_KidneyDisease  ChronicCond_Cancer  ChronicCond_ObstrPulmonary  \
0                        17                   5                          10
1                        64                  10                          41
2                        50                  16                          41
3                       507                 165                         295
4                        22                  12                          16

  ChronicCond_Depression  ChronicCond_Diabetes  ChronicCond_IschemicHeart  \
0                      9                    21                         23
1                     54                   100                        112
2                     63                   105                        108
3                    485                   799                        895
4                     29                    49                         51

  ChronicCond_Osteoporasis  ChronicCond_rheumatoidarthritis  \
0                        6                                8
1                       33                               38
2                       49                               46
3                      344                              331
4                       21                               22
```

```
   ChronicCond_stroke
0                   6
1                  12
2                  17
3                 124
4                  12

[5 rows x 25 columns]
```

[6]:

```
""" PROVIDED
Display the summary statistics
Make sure you skim this
"""
claims_data.describe()
```

[6]:

```
                Age  NumPhysiciansSeen  NumProcedures  NumDiagnosisClaims  \
count   5410.000000        5410.000000    5410.000000         5410.000000
mean      73.731027           1.227410       0.108011            3.676631
std        4.712307           0.220822       0.246305            1.882603
min       34.000000           0.500000       0.000000            0.000000
25%       71.768368           1.000000       0.000000            2.696134
50%       73.863636           1.200000       0.000000            3.000000
75%       75.760000           1.375000       0.083333            3.847902
max      101.000000           3.000000       3.000000           11.000000

       InscClaimAmtReimbursed  DeductibleAmtPaid  NoOfMonths_PartACov  \
count             5410.000000        5409.000000          5410.000000
mean              1740.679369         155.643175            11.919716
std               3484.473124         306.489453             0.395682
min                  0.000000           0.000000             0.000000
25%                232.394593           0.312500            11.994207
50%                356.085106           4.285714            12.000000
75%               1490.154301         137.418605            12.000000
max              57000.000000        1068.000000            12.000000

       NoOfMonths_PartBCov  IPAnnualReimbursementAmt  IPAnnualDeductibleAmt  \
count          5410.000000               5410.000000            5410.000000
mean             11.930647               6166.692586             666.980865
std               0.310612               6203.422910             623.108956
min               0.000000                  0.000000               0.000000
25%              11.965836               2902.238095             356.000000
50%              12.000000               4729.047927             527.580008
75%              12.000000               7336.173195             801.000000
max              12.000000             103000.000000           12068.000000

       …  ChronicCond_Heartfailure  ChronicCond_KidneyDisease  \
count  …               5410.000000                5410.000000
```

	…		
mean	…	60.921072	42.510906
std	…	158.698296	110.048136
min	…	0.000000	0.000000
25%	…	6.000000	4.000000
50%	…	18.000000	13.000000
75%	…	52.750000	37.000000
max	…	4638.000000	3111.000000

	ChronicCond_Cancer	ChronicCond_ObstrPulmonary	ChronicCond_Depression \
count	5410.000000	5410.000000	5410.000000
mean	15.620148	32.288540	44.863956
std	41.558020	82.958866	117.563035
min	0.000000	0.000000	0.000000
25%	1.000000	3.000000	4.000000
50%	5.000000	10.000000	13.000000
75%	13.000000	29.000000	39.000000
max	1238.000000	2312.000000	3592.000000

	ChronicCond_Diabetes	ChronicCond_IschemicHeart \
count	5410.000000	5410.000000
mean	72.783549	78.341959
std	190.919202	205.233787
min	0.000000	0.000000
25%	7.000000	7.000000
50%	22.000000	23.000000
75%	62.750000	67.000000
max	5784.000000	6074.000000

	ChronicCond_Osteoporasis	ChronicCond_rheumatoidarthritis \
count	5410.000000	5410.000000
mean	32.775231	32.107024
std	85.862305	84.497824
min	0.000000	0.000000
25%	3.000000	3.000000
50%	10.000000	9.000000
75%	28.000000	28.000000
max	2531.000000	2511.000000

	ChronicCond_stroke
count	5410.000000
mean	10.495564
std	27.171512
min	0.000000
25%	1.000000
50%	3.000000
75%	9.000000
max	810.000000

[8 rows x 23 columns]

3 PRE-PROCESS DATA

[7]:
```python
""" PROVIDED
Construct preprocessing pipeline
"""
selected_features = claims_data.columns.drop(['Provider'])
scaled_features = ['InscClaimAmtReimbursed', 'DeductibleAmtPaid',
                   'IPAnnualReimbursementAmt', 'IPAnnualDeductibleAmt',
                   'OPAnnualReimbursementAmt', 'OPAnnualDeductibleAmt']

pipe = Pipeline([
    ('RowDropper', DataSampleDropper()),
    ('FeatureSelector', DataFrameSelector(selected_features)),
    ('Scale', DataScaler(scaled_features))
])
```

[8]:
```python
""" TODO
Pre-process the data using the defined pipeline
"""
processed_data = pipe.fit_transform(claims_data)
processed_data.shape
```

[8]: (5409, 24)

[9]:
```python
""" TODO
Verify all NaNs removed
"""
processed_data.isnull().values.any()
```

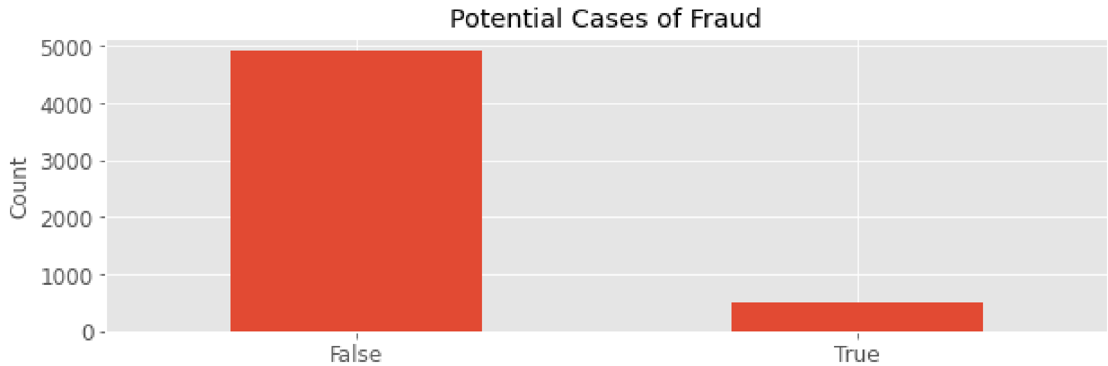
[9]: False

4 VISUALIZE DATA

[10]:
```python
""" PROVIDED
Plot the class distributions for no potential fraud and potential fraud
"""
class_counts = pd.value_counts(processed_data['PotentialFraud'])
class_counts.plot(kind='bar', rot=0, figsize=(10,3))
plt.title("Potential Cases of Fraud")
plt.ylabel("Count")

# Display the class fractions
```

```
nsamples, nfeatures = processed_data.shape
class_counts / nsamples
```

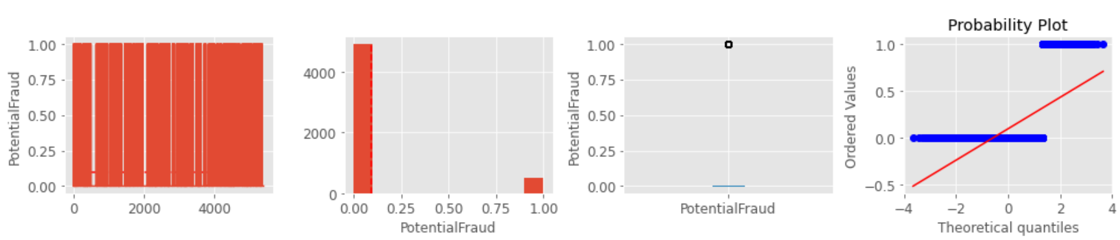
[10]:
```
False    0.906452
True     0.093548
Name: PotentialFraud, dtype: float64
```



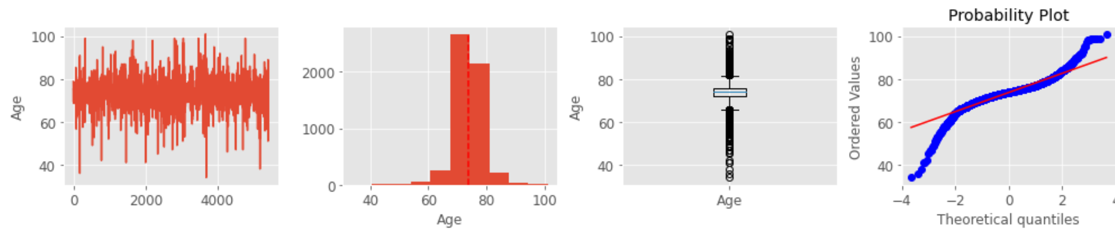
[11]:
```
""" PROVIDED
Extract positions of the postive and negative cases
"""
pos = processed_data['PotentialFraud'] == 1
neg = processed_data['PotentialFraud'] == 0
```

[12]:
```
""" PROVIDED
Visualize the data using visualize.featureplots
"""
# Please make sure to "Enable Scrolling for Outputs" before generating the PDF
cdata = processed_data.astype('float64')
visualize.featureplots(cdata.values, cdata.columns)
```

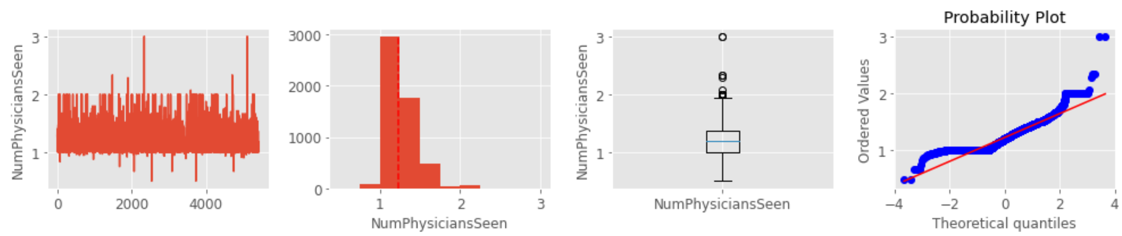
FEATURE: PotentialFraud



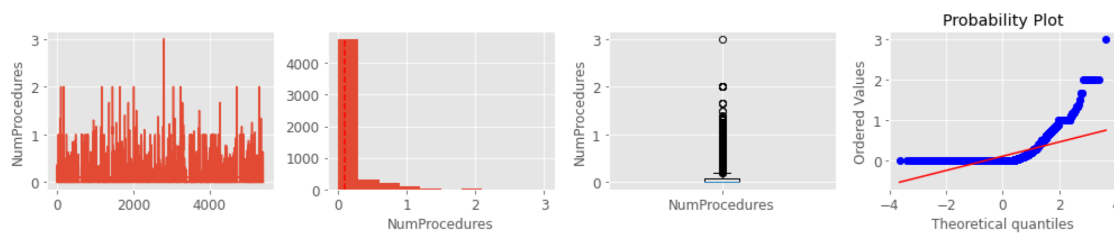
FEATURE: Age



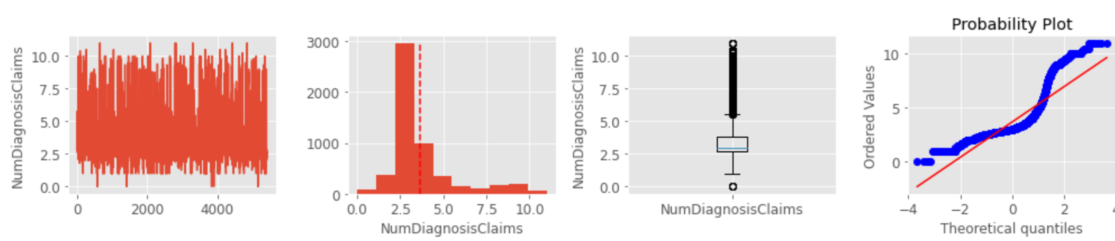
FEATURE: NumPhysiciansSeen



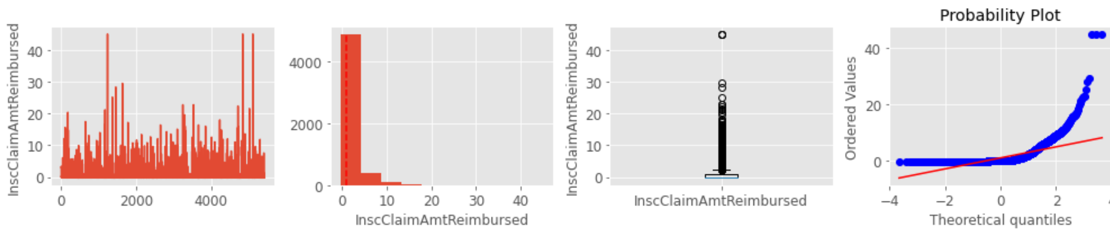
FEATURE: NumProcedures



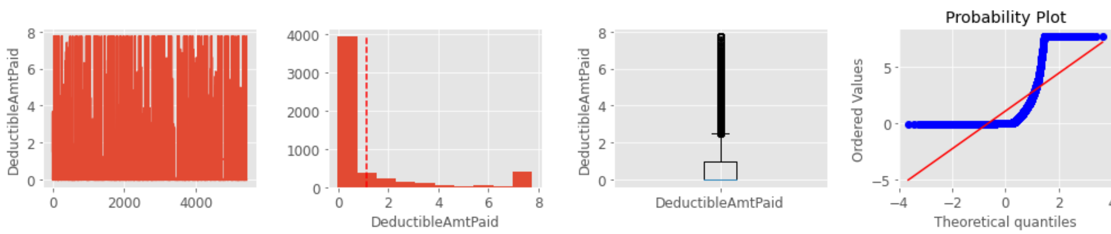
FEATURE: NumDiagnosisClaims



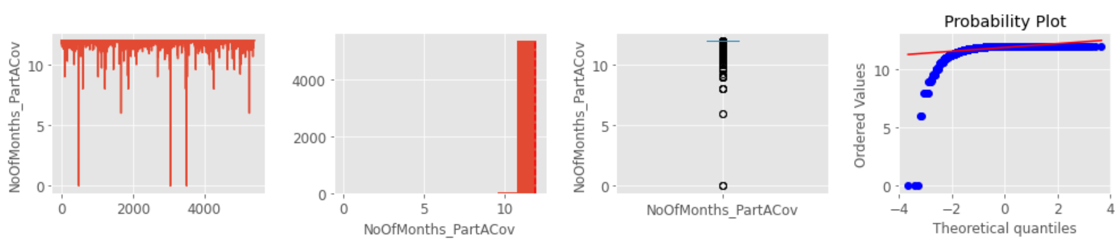
FEATURE: InscClaimAmtReimbursed



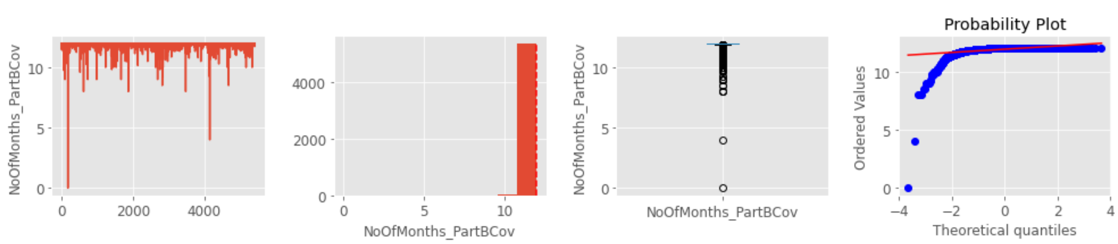
FEATURE: DeductibleAmtPaid



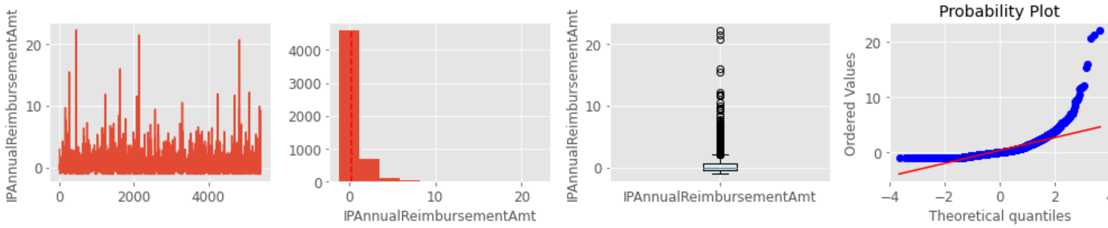
FEATURE: NoOfMonths_PartACov



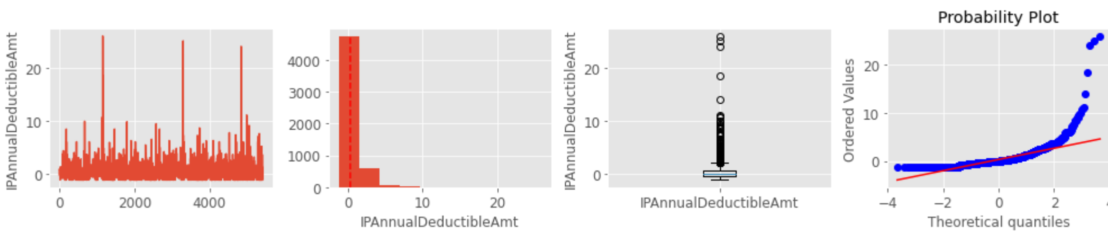
FEATURE: NoOfMonths_PartBCov



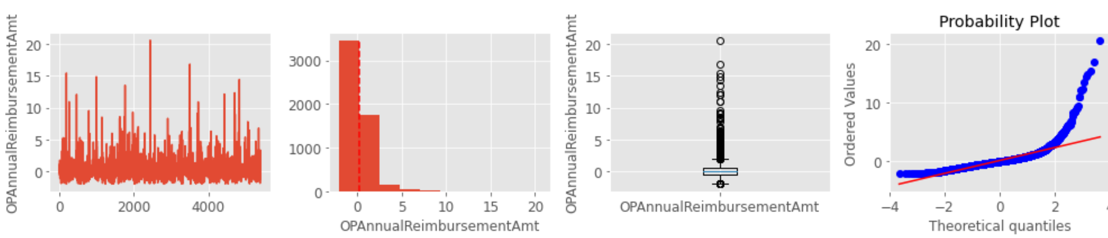
FEATURE: IPAnnualReimbursementAmt



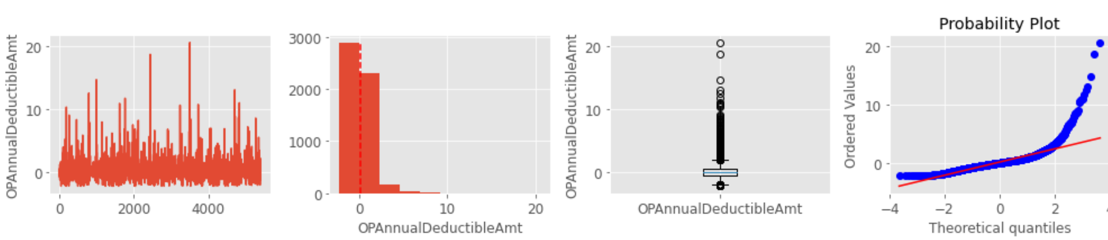
FEATURE: IPAnnualDeductibleAmt



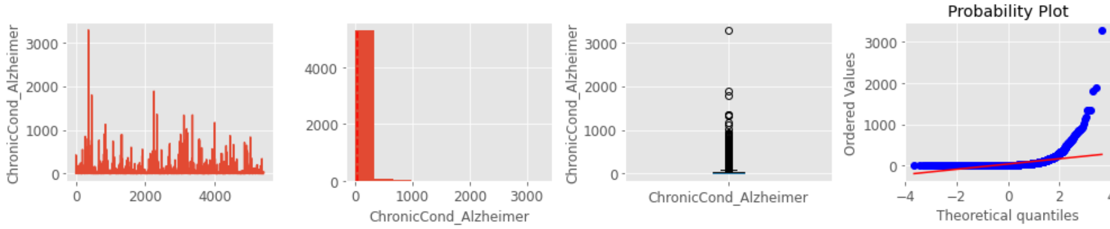
FEATURE: OPAnnualReimbursementAmt



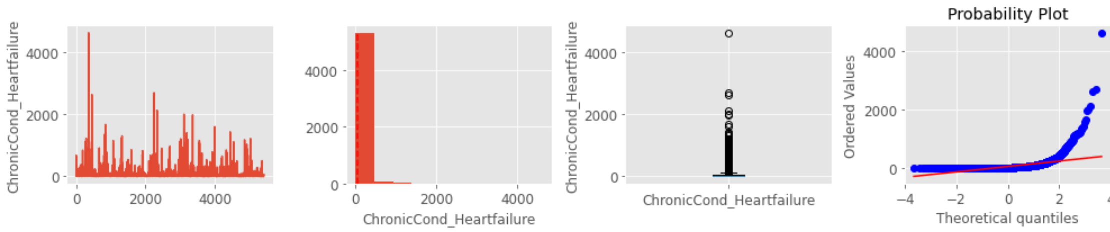
FEATURE: OPAnnualDeductibleAmt



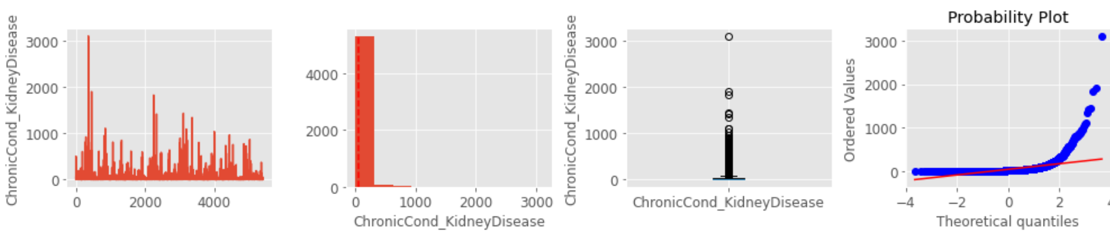
FEATURE: ChronicCond_Alzheimer



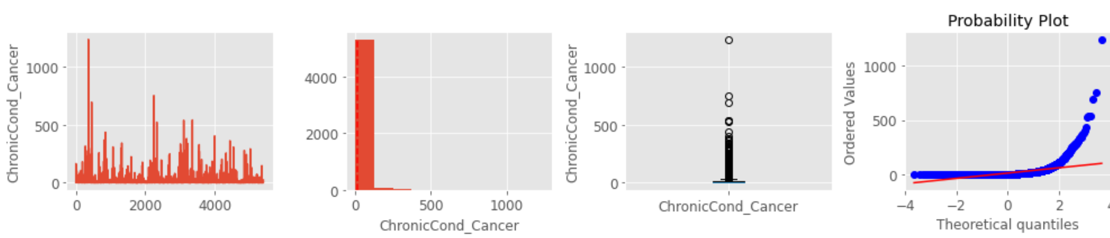
FEATURE: ChronicCond_Heartfailure



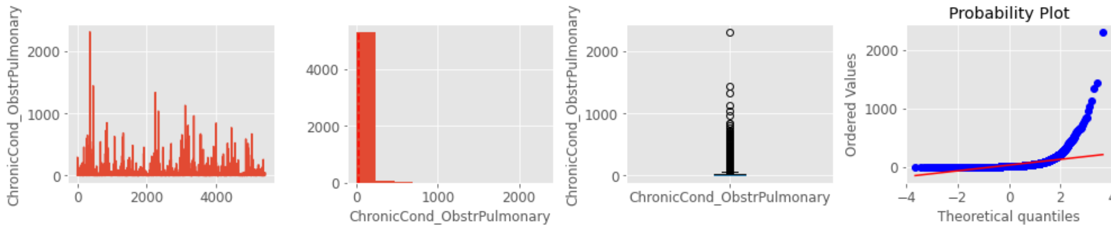
FEATURE: ChronicCond_KidneyDisease



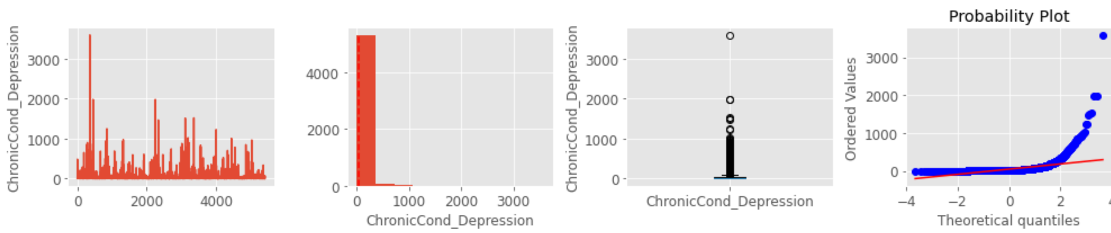
FEATURE: ChronicCond_Cancer



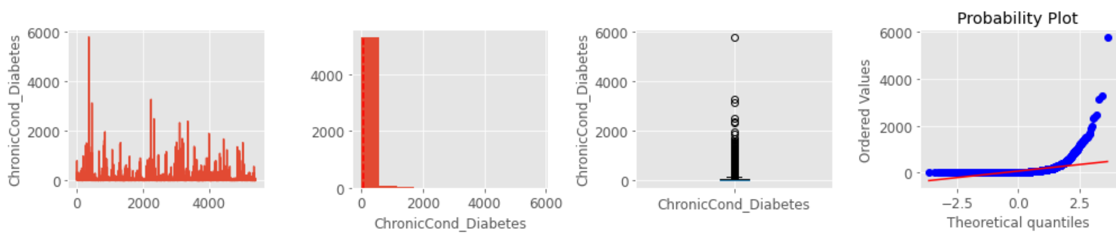
FEATURE: ChronicCond_ObstrPulmonary



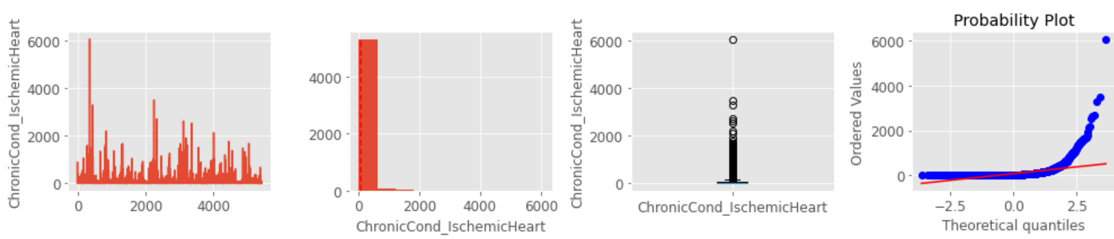
FEATURE: ChronicCond_Depression



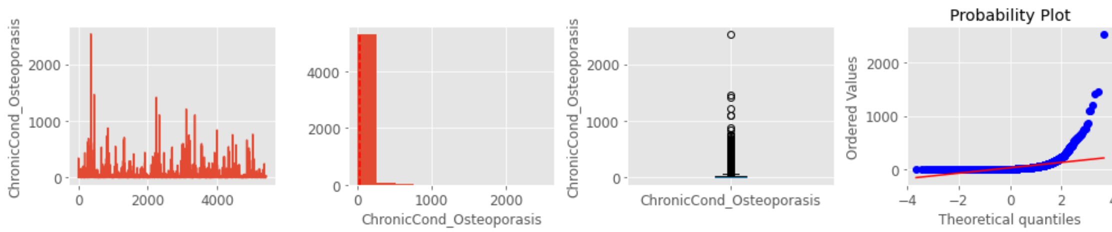
FEATURE: ChronicCond_Diabetes



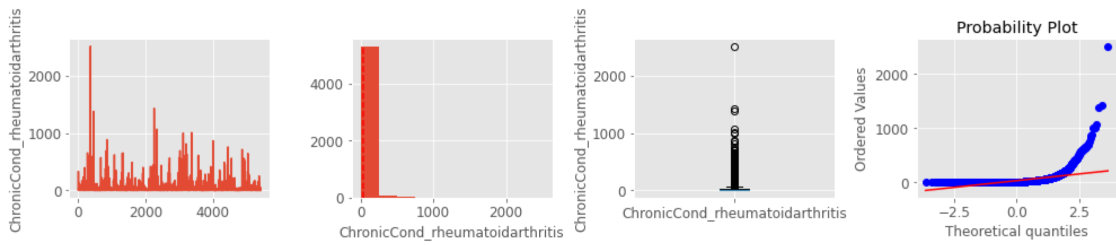
FEATURE: ChronicCond_IschemicHeart



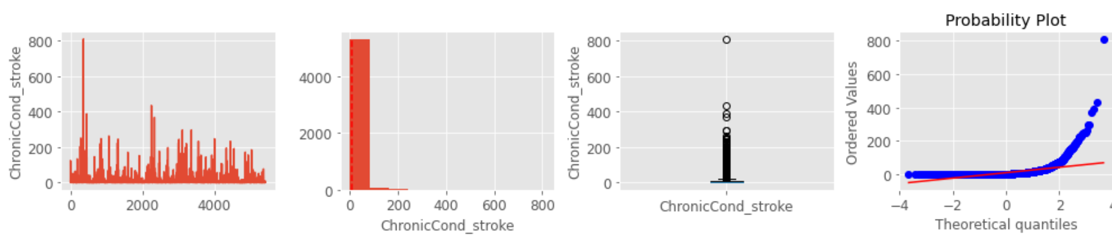
FEATURE: ChronicCond_Osteoporasis



FEATURE: ChronicCond_rheumatoidarthritis



FEATURE: ChronicCond_stroke



5 Decision Tree Classifiers

5.0.1 Model Exploration

[13]:
```
""" TODO
Split data into X (the inputs) and y (the outputs)

Hold out a subset of the data, before training and cross validation
using train_test_split, with stratify NOT equal to None, and a test_size
fraction of .2.

For this exploratory section, the held out set of data is a validation set.
For the GridSearch section, the held out set of data is a test set.
"""
```

```
targetnames = ['NonFraud', 'Fraud']

# TODO: Separate the data into X and y
X = processed_data[processed_data.columns.drop(['PotentialFraud'])]
y = processed_data['PotentialFraud']

# TODO: Split data into train and test sets and display the shapes of the train␣
↪and test sets
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.2, stratify=y)
Xtrain.shape, Xtest.shape
```

[13]: ((4327, 23), (1082, 23))

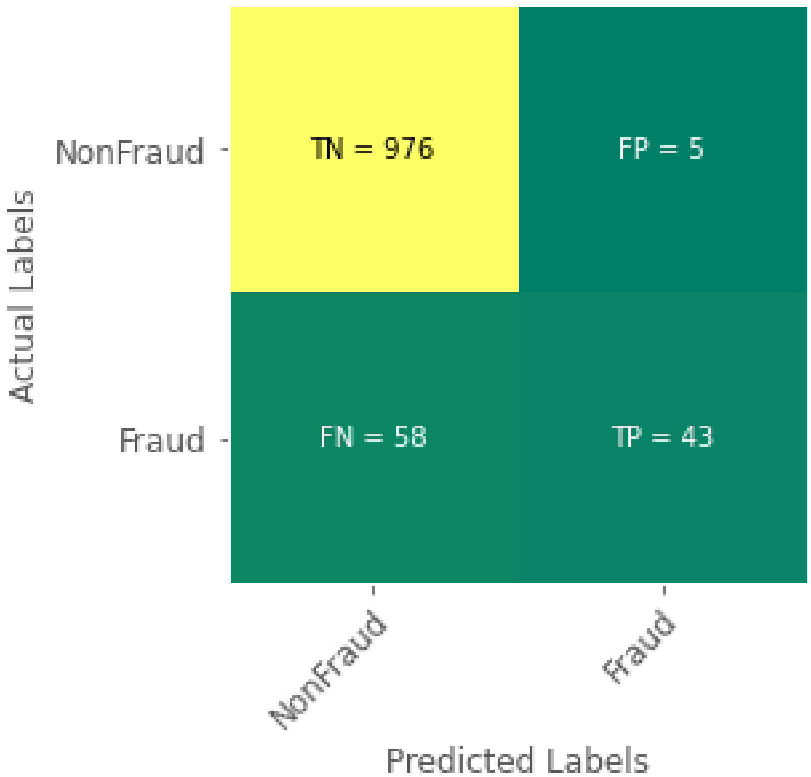
[14]:
```
""" TODO
Play around with the hyper-parameters. Pick your favorite model to leave with␣
↪in
your submitted report.
"""
# TODO: Create and fit the model
tree_model = DecisionTreeClassifier(random_state = 0, max_depth=4)
# clf = SGDClassifier(loss='log', random_state=42, max_iter=10000, tol=1e-3)
tree_model.fit(Xtest,ytest)

# TODO: Predict with the model on the validation set
preds_val = tree_model.predict(Xtest)


# TODO: Obtain prediction probabilities for the validation set, using
# cross_val_predict with cv=10 and method='predict_proba'
proba_val = cross_val_predict(tree_model, Xtest, ytest, method='predict_proba',␣
↪cv=10)


# TODO: The mean CV accuracy on the given validation data and labels, using
# cross_val_score and cv=10
scorescv = cross_val_score(tree_model, Xtest, ytest, cv=10)
np.mean(scorescv)
```

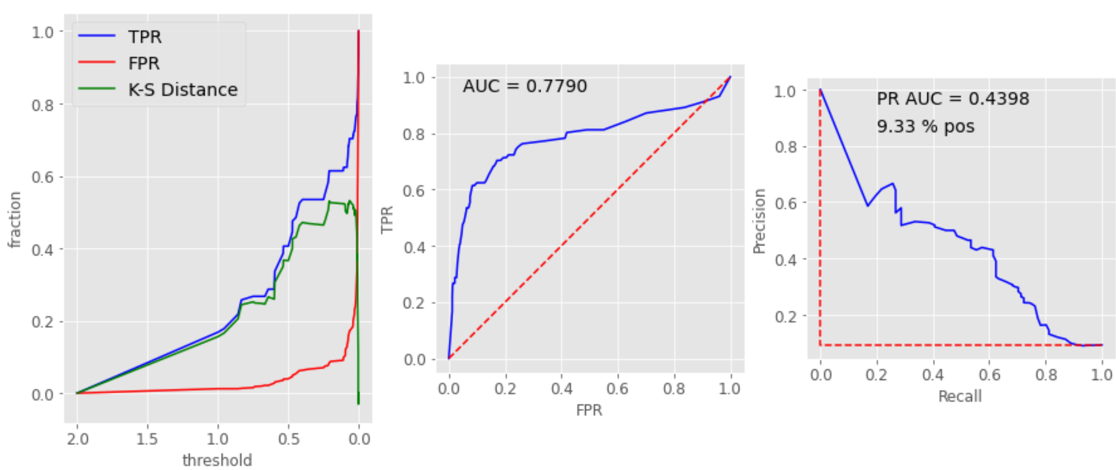
[14]: 0.9094376486578322

[19]:
```
""" TODO
Display the confusion matrix, KS plot, ROC curve, and PR curve for the␣
↪validation set
using metrics_plots.ks_roc_prc_plot

The red dashed line in the PRC is indicative of a the expected performance for␣
↪a random
```

```
classifier, which would predict predict postives at the rate of occurance␣
↪within the data set
"""
# TODO: Confusion Matrix
confusion = confusion_matrix(ytest, preds_val)
metrics_plots.confusion_mtx_colormap(confusion, targetnames, targetnames)

# TODO: Curves
# Note, you'll want the probability class predictions for the class label 1
# See the API page for the DecisionTreeClassifier predict_proba; proba_val[:,1]
scores = proba_val[:,1]
metrics_plots.ks_roc_prc_plot(ytest, scores)

# Obtain the PSS and F1 Score
pss_val = metrics_plots.skillScore(ytest, preds_val)
f1_val = f1_score(ytest, preds_val)
print("PSS: %.4f" % pss_val[0])
print("F1 Score %.4f" % f1_val)
print(tree_model.get_params().keys())
```

```
PSS: 0.4206
F1 Score 0.5772
dict_keys(['ccp_alpha', 'class_weight', 'criterion', 'max_depth',
'max_features', 'max_leaf_nodes', 'min_impurity_decrease', 'min_impurity_split',
'min_samples_leaf', 'min_samples_split', 'min_weight_fraction_leaf', 'presort',
'random_state', 'splitter'])
```

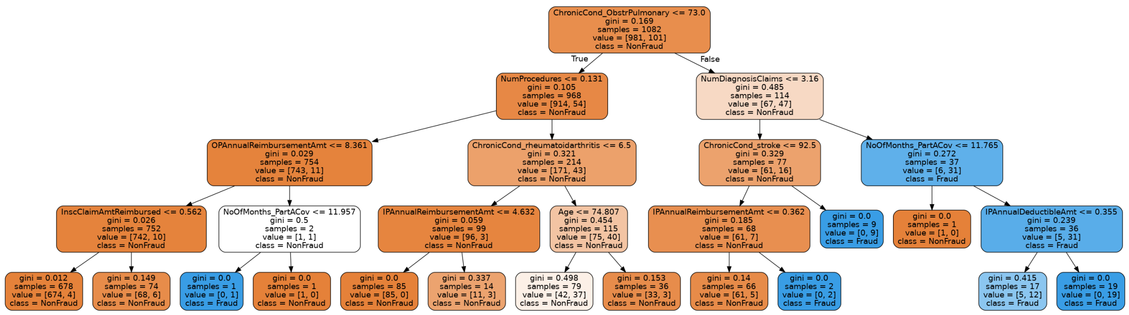




[16]:
```
""" PROVIDED
Export the image of the tree model
"""
export_graphviz(tree_model, out_file='exploratory_model.dot',
                feature_names=X.columns, class_names=targetnames,
```

```
                 rounded=True, filled=True)
!dot -Tpng exploratory_model.dot > e_model.png
Image(filename='e_model.png')
```

[16]:



6 GRID SEARCH CV

[21]:

```
""" TODO
Estimated time: <25 min on Oscer
Set up and run the grid search using GridSearchCV and the following
settings:
* The below scoring dictionary for scoring,
* refit set to 'f1' as the optimized metric
* Twenty for the number of cv folds,
* n_jobs=-1,
* verbose=2,
* return_train_score=True
"""
# Optimized metric
opt_metric = 'f1'
scoring = {opt_metric:opt_metric}

# Flag to re-load previous run regardless of whether the file exists
force = False
# File previous run is saved to
srchfname = "hw9_search_" + opt_metric + ".pkl"

# SETUP EXPERIMENT HYPERPARAMETERS
max_depths = [None, 200, 100, 10, 8, 6, 4]
max_leaf_nodes = [None, 10, 5, 2]

ndepths = len(max_depths)
nleaves = len(max_leaf_nodes)

# TODO: Create the dictionary of hyper-parameters to try
```

```python
hyperparams = {'max_depth':max_depths, 'max_leaf_nodes':max_leaf_nodes}

# RUN EXPERIMENT
time0 = timelib.time()
search = None
if force or (not os.path.exists(srchfname)):
    # TODO: Create the GridSearchCV object
    search = GridSearchCV(tree_model, hyperparams, scoring=scoring,␣
↪refit=opt_metric,
                          cv=20, n_jobs=-1, verbose=2, return_train_score=True)

    # TODO: Execute the grid search by calling fit using the training data
    search.fit(Xtrain, ytrain)

    # TODO: Save the grid search object
    joblib.dump(search, srchfname)

    print("Saved %s" % srchfname)
else:
    # TODO: Re-load the grid search object
    search = joblib.load(srchfname)
    print("Loaded %s" % srchfname)

time1 = timelib.time()
duration = time1 - time0
print("Elapsed Time: %.2f min" % (duration / 60))

search
```

```
Loaded hw9_search_f1.pkl
Elapsed Time: 0.00 min
```

[21]:
```
GridSearchCV(cv=20,
             estimator=DecisionTreeClassifier(max_depth=4, random_state=0),
             n_jobs=-1,
             param_grid={'max_depth': [None, 200, 100, 10, 8, 6, 4],
                         'max_leaf_nodes': [None, 10, 5, 2]},
             refit='f1', return_train_score=True, scoring={'f1': 'f1'},
             verbose=2)
```

7 RESULTS

[22]:
```python
""" PROVIDED
Display the head of the results for the grid search
See the cv_results_ attribute
"""
```

```python
all_results = search.cv_results_
df_res = pd.DataFrame(all_results)
df_res.head(3)
```

[22]:
```
   mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
0       0.104415      0.005764         0.005838        0.000834
1       0.048879      0.002344         0.006330        0.002224
2       0.044034      0.005997         0.005620        0.000780

  param_max_depth param_max_leaf_nodes  \
0            None                 None
1            None                   10
2            None                    5

                                      params  split0_test_f1  \
0  {'max_depth': None, 'max_leaf_nodes': None}        0.622222
1    {'max_depth': None, 'max_leaf_nodes': 10}        0.648649
2     {'max_depth': None, 'max_leaf_nodes': 5}        0.545455

   split1_test_f1  split2_test_f1  …  split12_train_f1  split13_train_f1  \
0        0.756757        0.476190  …          1.000000          1.000000
1        0.648649        0.628571  …          0.646809          0.629738
2        0.461538        0.516129  …          0.486188          0.473881

   split14_train_f1  split15_train_f1  split16_train_f1  split17_train_f1  \
0          1.000000          1.000000          1.000000          1.000000
1          0.624633          0.642755          0.644979          0.612557
2          0.372277          0.492701          0.490909          0.490909

   split18_train_f1  split19_train_f1  mean_train_f1  std_train_f1
0          1.000000          1.000000       1.000000      0.000000
1          0.640227          0.647887       0.627048      0.014087
2          0.478821          0.486289       0.475601      0.040368

[3 rows x 52 columns]
```

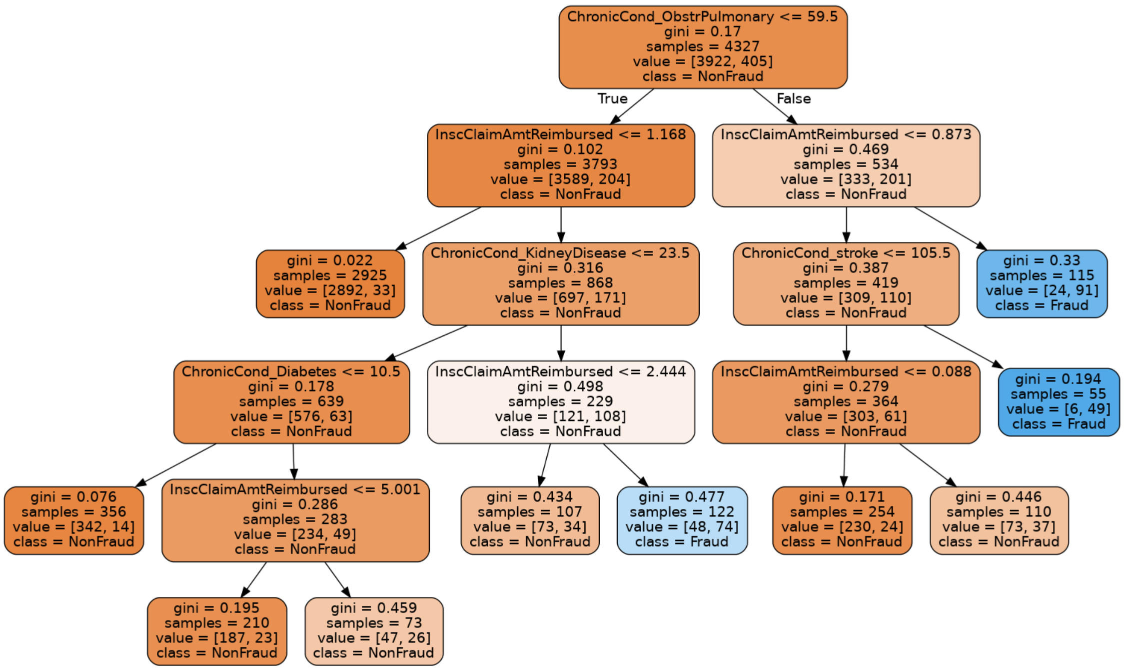
[23]:
```python
""" TODO
Obtain the best model from the grid search and
fit it to the full training data
"""
best_model = search.best_estimator_
best_model.fit(Xtrain, ytrain)
```

[23]:
```
DecisionTreeClassifier(max_leaf_nodes=10, random_state=0)
```

[24]:
```python
""" PROVIDED
Export the image of the best model
```

```python
use export_graphviz
"""
export_graphviz(best_model, out_file='best_model.dot',
                feature_names=X.columns, class_names=targetnames,
                rounded=True, filled=True)
!dot -Tpng best_model.dot > b_model.png
Image(filename='b_model.png')
```

[24]:

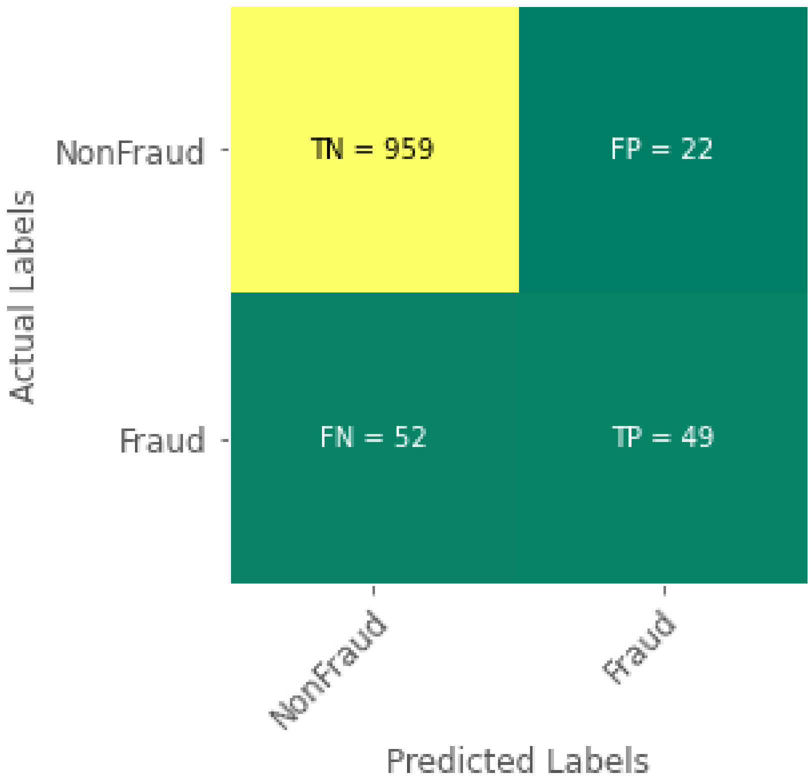


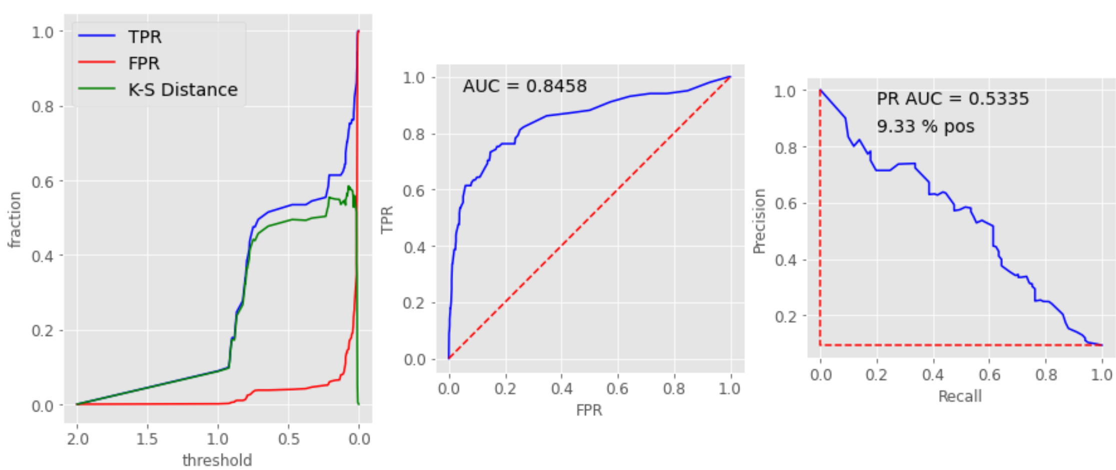
[28]:

```python
""" TODO
Display the confusion matrix, KS plot, ROC curve, and PR curve for the test
set using metrics_plots.ks_roc_prc_plot

The red dashed line in the PRC is indicative of a the expected performance for
a random classifier, which would predict predict postives at the rate of
occurance within the data set
"""
# TODO: Predict with the best model on the test set
preds_test = best_model.predict(Xtest)

# TODO: Obtain prediction probabilities for the test set using cross_val_predict
# 'predict_proba' as the method
proba_test = cross_val_predict(best_model, Xtest, ytest,␣
 ↪method='predict_proba', cv=10)
```

```python
# TODO: Compute mean accuracy (using cross_val_score) on the given test data
↪and labels
best_scorescv = cross_val_score(best_model, Xtest, ytest, cv=10)
print('Mean accuracy: ', np.mean(best_scorescv))

# TODO: Confusion Matrix
confusion = confusion_matrix(ytest, preds_test)
metrics_plots.confusion_mtx_colormap(confusion, targetnames, targetnames)


# TODO: Curves (i.e. ROC, PRC, etc) use metrics_plots.ks_roc_prc_plot and the
# the probabilities for the class label of 1
best_scores = proba_test[:,1]
metrics_plots.ks_roc_prc_plot(ytest, best_scores)


# Obtain the PSS and F1 Score
pss_test = metrics_plots.skillScore(ytest, preds_test)
f1_test = f1_score(ytest, preds_test)
print("PSS: %.4f" % pss_test[0])
print("F1 Score %.4f" % f1_test)
```

```
Mean accuracy:  0.920497791369351
PSS: 0.4627
F1 Score 0.5698
```

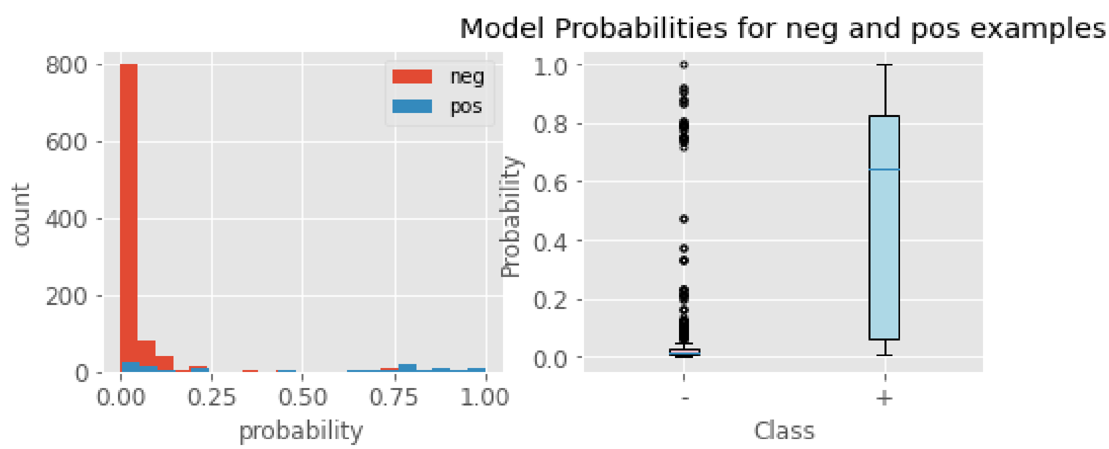




[26]:
```
""" PROVIDED
Plot a histogram of the test scores from the best model.
Compare the distribution of scores for positive and negative examples
using boxplots.
```

```python
Create one subplot of the distribution of all the scores, with a histogram.
Create a second subplot comparing the distribution of the scores of the
positive examples with the distribution of the negative examples, with boxplots.
"""
# Obtain the pos and neg indices
pos_inds = np.where(ytest)[0]
neg_inds = np.where(ytest == 0)[0]

# Separate the scores for the pos and neg examples
proba_pos = proba_test[pos_inds, 1]
proba_neg = proba_test[neg_inds, 1]

# Plot the distribution of all scores
nbins = 21
plt.figure(figsize=(8,3))
plt.subplot(1,2,1)
plt.hist(proba_neg, bins=nbins)
plt.hist(proba_pos, bins=nbins)
plt.xlabel('probability', fontsize=FONTSIZE)
plt.ylabel('count', fontsize=FONTSIZE)
plt.legend(['neg', 'pos'])

# Plot the boxplots of the pos and neg examples
plt.subplot(1,2,2)
boxplot = plt.boxplot([proba_neg, proba_pos], patch_artist=True, sym='.')
boxplot['boxes'][0].set_facecolor('pink')
boxplot['boxes'][1].set_facecolor('lightblue')
plt.xticks(ticks=[1, 2], labels=['-', '+'])
plt.xlabel("Class")
plt.ylabel("Probability")
plt.title("Model Probabilities for neg and pos examples")
```

[26]: Text(0.5, 1.0, 'Model Probabilities for neg and pos examples')



8 Discussion

In a few paragraphs, discuss and interpret the test results for the best model. Include a brief discussion of the histogram and boxplots of the scores. Compare the best model from the grid search to the one you chose in the exploration section.

From the last hw, I talked about how ROC AUC has better accuracy compared to PRC AUC when there are more negative than positive. This is reflected in the best model becuase the ROC AUC is 0.8458 where as PRC AUC is 0.5335. We can see from the histogram and boxplots what there are vastly more negative than positive as well as negative class having a large amount of outliers where the positive class does not.

Comparing best model to the one that I chose, best model has a better ROC AUC and PRC AUC score. The difference is only 0.0668 between the two for ROC AUC. Although close, a ROC AUC score ov .8 to .9 is excellent compared to .7 to .8 being acceptable.