Lab 4: Trees and Forests

In this assignment, you will explore tree-based classification models to implement predictions for broadband deployment, using a combination of available datasets from the Census and the Federal Communications Commission (FCC).

You will work with these datasets to understand the nature of broadband deployment across Chicago, as well as what features turn out to be good predictors of broadband deployment. The existence of connectivity in a certain area can also be measured in a variety of ways (e.g., available speed tiers, subscriptions, measured performance), and reported in different ways (e.g., by ISPs, by citizens/subscribers). You will explore how reporting and characteristics vary across different datasets and how this may affect your model.

Learning Objectives ¶

In this assignment, you will learn the following:

- The mechanisms of decision trees and bootstrapping methods, including Random Forest classifiers
- How to work with various broadband connectivity data (ACS, FCC, etc.)
- How to apply tree-based classification models to data (decision tree, Random Forest)
- How to take advantage of various characteristics of Random Forest models, such as feature importance
- How to evaluate classification models using various metrics and approaches

Part 1: Tree-Based Models by Hand

In this section, you will manually implement a decision tree using entropy and information gain. Please do not use code in your calculations for this section. You may either type your solutions using Markdown, or attach a scanned copy of your hand written solutions in your Canvas file submission.

For this section, you will predict the user experience when streaming a video, based on three attributes: the speed that they have purchased from their ISP, the quality of their WiFi Router, and the type of device they are using to stream.

ISP Speed	WiFi Router	Device Type	User Experience
Fast	Regular	Old	Low
Medium	Regular	Average	High
Slow	Premium	New	Low
Fast	Premium	New	High
Fast	Premium	Average	Low
Medium	Regular	Old	High
Slow	Regular	Old	High
Slow	Premium	Average	Low
Slow	Premium	Average	High
Slow	Regular	Average	High

1.1 Calculating Entropy

What is the entropy for the User Experience attribute? Were you expecting a result around this number? Why?

The Mathematical formula for Entropy is as follows:

$$E(S) = \sum_{i=1}^{n} -p_i log_2 p_i$$

The entropy for the User Experience attribute is:

$$E(UserExperience) = -\frac{4}{10} \times log_2 \frac{4}{10} - \frac{6}{10} \times log_2 \frac{6}{10} \approx 0.97$$

This is considered a high entropy, it means a low level of purity in the data. It was expected since the observations are almost split evenly between low and high user experience. In order to have a high level of purity or 0 in Entropy measure we should have 0 observations in one of the categories and 10 in the other, since 10 is the total number of observations in this dataset.

In [1]:
```
import censusdata
import geopandas as gpd
from geopandas import GeoDataFrame
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

%matplotlib inline
```

In [2]:
```
import math
entropy = -(4/10) * math.log(4/10, 2) - (6/10) * math.log(6/10, 2)
entropy
```

Out[2]:
```
0.9709505944546686
```

1.2 Calculating Information Gain

What is the information gain on User Experience if you split the observations on the WiFi Router attribute?

In [3]:
```
data = [[1,4,5],[3,2,5], [4,6,10]]
infogain = pd.DataFrame(data, columns=["Low", "High", "Total"], index = ["Regu
lar", "Premium", "Total"])
print("Contigency table to calculate entropy based on WiFi Router")
infogain
```

```
Contigency table to calculate entropy based on WiFi Router
```

Out[3]:

	Low	High	Total
Regular	1	4	5
Premium	3	2	5
Total	4	6	10

$$E(UserExperience|WifiRouter = Regular) = -\frac{1}{5} \times log_2\frac{1}{5} - \frac{4}{5} \times log_2\frac{4}{5} \approx 0.72$$
$$E(UserExperience|WifiRouter = Premium) = -\frac{3}{5} \times log_2\frac{3}{5} - \frac{2}{5} \times log_2\frac{2}{5} \approx 0.97$$

Weighted Average:

$$E(UserExperience|WifiRouter) = -\frac{5}{10} \times 0.72 + \frac{5}{10} \times 0.97 \approx 0.85$$

Compute the Information Gain on User Experience from Wifi Router. Information Gain:

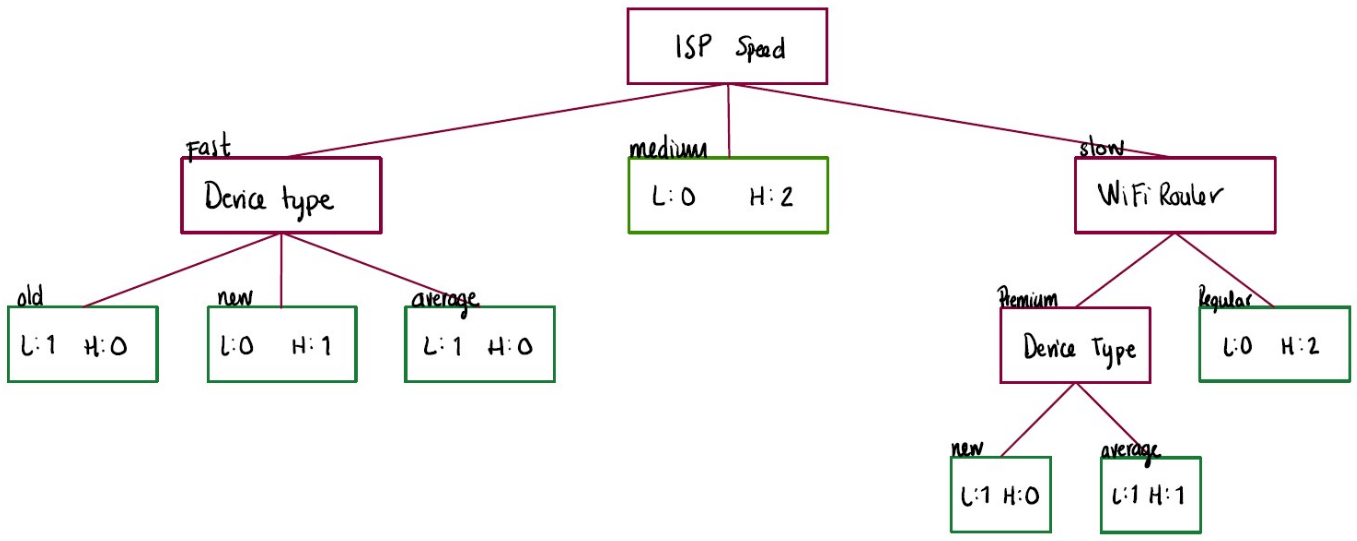
$$IG(UserExperience, WifiRouter) = E(UserExperience) - E(UserExperience|WifiRouter)$$
$$IG(UserExperience, WifiRouter) = 0.97 - 0.85$$
$$= 0.124$$

Wifi Router helps reducing the uncertainty around the target variable:User Experience.

1.3 Building a Simple Decision Tree

Construct a simple two-level decision tree that can be used to predict User Experience. Use information gain as your splitting criterion. Feel free to use a greedy approach, recursively splitting on the attribute that gives the greatest information gain at each step.

Draw your final tree and include an image of it with your final submission.



Part 2: Loading and Exploring Datasets

2.1 FCC Broadband Map Data

In the first part of the lab, you will compile the datasets that you will use in your analysis. Given that the FCC dataset is quite large, we will work with a truncated version of the data for Chicago. We have also included the original source in case you wish to explore the full data yourself.

The FCC makes its data for broadband maps available on its website (https://broadbandmap.fcc.gov). Data is also available for download (https://broadbandmap.fcc.gov/#/data-download). Internet service providers are required to fill out a "Form 477", which reports service offerings in each Census Block.

The specific data that we will use for this lab is from June 2019 in Cook County, IL. We provide the the following code for downloading the data, according to the API documentation (https://dev.socrata.com/foundry/opendata.fcc.gov/sgz3-kiqt). You should have 794,141 rows. The download should take no more than 5 minutes.

In [4]:
```python
import pandas as pd
from sodapy import Socrata

client = Socrata("opendata.fcc.gov", None)

# GEOID is the only geographic attribute, so we limit the data to Cook County
# Returned as JSON from API / converted to Python list of dictionaries by soda
py
results = client.get("sgz3-kiqt", limit=800000, where="starts_with(blockcode,
 '17031')")

# Convert to pandas DataFrame
fcc_df = pd.DataFrame.from_records(results)

print(fcc_df.shape)
fcc_df.head(2)
```

```
WARNING:root:Requests made without an app_token will be subject to strict thr
ottling limits.

(794141, 17)
```

Out[4]:

	logrecno	provider_id	frn	providername	dbaname	holdingcompanyname	hoc
0	48702817	52979	0001568880	GCI Communication Corp.	GCI Communication Corp.	GCI Holdings LLC	13
1	48702818	52979	0001568880	GCI Communication Corp.	GCI Communication Corp.	GCI Holdings LLC	13

First, produce the following three maps for the City of Chicago:

1. The maximum contractual downstream speed offered by any provider in each Census block group.
2. The number of unique ISPs that offer service in each Census block group.
3. The number of unique ISPs that offer service at or above 25 Mbps downstream and 3 Mbps upstream in each Census block group. (This is the FCC's definition of broadband Internet access, which you can read about more in the 2019 broadband deployment report (https://docs.fcc.gov/public/attachments/FCC-19-44A1.pdf)). Use contractual downstream and upstream speed.

Note: Geographic boundaries are only directly available on the Chicago Open Data Portal at the Census block and Census tract level. However, the GeoPandas `dissolve` command can be used to obtain boundaries at the Census block group level. Recall that Census block groups correspond to 12 digits of the FIPS code based on the Census geography hierarchy (https://www.census.gov/programs-surveys/geography/guidance/geo-identifiers.html).

In [5]:
```
census_gdf = gpd.read_file("https://data.cityofchicago.org/resource/bt9m-d2mf.
geojson?$limit=9999999")
```

In [6]:
```
census_gdf.head(2)
```

Out[6]:

	statefp10	name10	blockce10	tract_bloc	geoid10	tractce10	countyfp10	ge
0	17	Block 2010	2010	6903002010	170316903002010	690300	031	MULTIPO (((-8 41 -87.6
1	17	Block 3007	3007	6809003007	170316809003007	680900	031	MULTIPO (((-8 41 -87.6

In [7]:
```
fcc_df["geo_12"] = fcc_df["blockcode"].map(lambda x: str(x)[:12])
census_gdf["geo_12"] = census_gdf["geoid10"].map(lambda x: str(x)[:12])
```

In [8]:
```
blockgroups = census_gdf.dissolve(by='geo_12')
blockgroups.crs = "EPSG:4326"
blockgroups['area'] = blockgroups.to_crs('EPSG:3857').area / (10**6)
```

In [9]:
```
blockgroups.head(2)
```

Out[9]:

	geometry	statefp10	name10	blockce10	tract_bloc	geoid10	tractce1
geo_12							
170310101001	POLYGON ((-87.67009 42.02115, -87.67047 42.021...	17	Block 1000	1000	0101001000	170310101001000	01010
170310101002	POLYGON ((-87.66950 42.01936, -87.66963 42.019...	17	Block 2002	2002	0101002002	170310101002002	01010

In [10]:
```
columns = ['maxcirdown', 'maxcirup']
fcc_df[columns] = fcc_df[columns].apply(pd.to_numeric)

fcc_df['target'] = 0
fcc_df.loc[(fcc_df['maxcirdown'] >= 25) & (fcc_df['maxcirup'] >= 3), 'target']
= 1

fcc_df['isp_broadband'] = None
fcc_df.loc[(fcc_df['maxcirdown'] >= 25) & (fcc_df['maxcirup'] >= 3), 'isp_broa
dband'] = \
fcc_df.loc[(fcc_df['maxcirdown'] >= 25) & (fcc_df['maxcirup'] >= 3), 'provider
_id']

fcc_df = fcc_df.groupby('geo_12').agg({'maxcirdown': 'max', 'target': 'max',
'provider_id':'nunique',
                                       'isp_broadband': 'nunique'}).reset_index
()

fcc_df
```

Out[10]:

	geo_12	maxcirdown	target	provider_id	isp_broadband
0	170310101001	0.0	0	7	0
1	170310101002	1000.0	1	9	1
2	170310101003	1000.0	1	9	2
3	170310102011	0.0	0	7	0
4	170310102012	1000.0	1	9	1
...	...	...	...	...	...
3988	170318439004	500.0	1	7	1
3989	170318439005	500.0	1	7	1
3990	170319800001	1000.0	1	20	5
3991	170319801001	1000.0	1	12	3
3992	170319900000	0.0	0	1	0

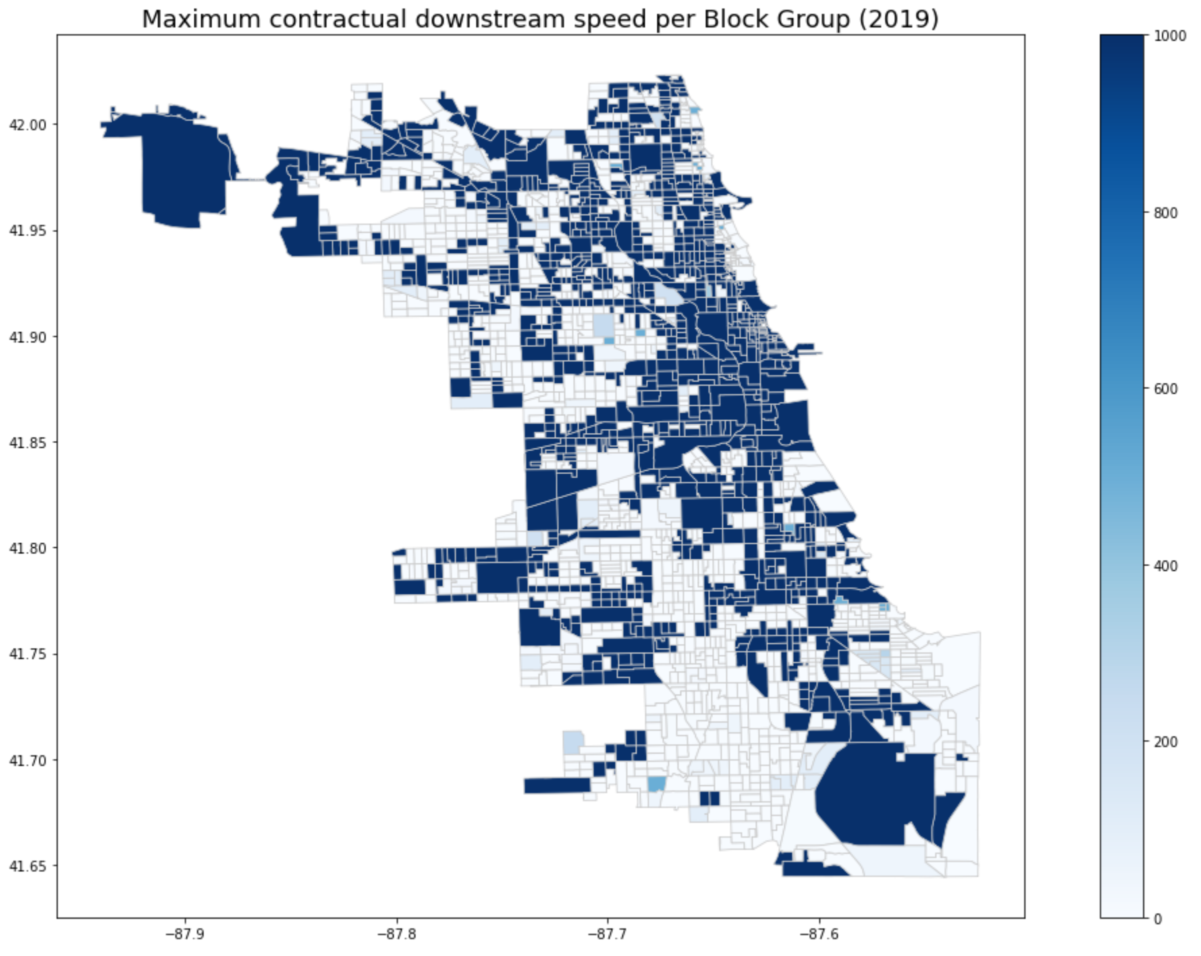
3993 rows × 5 columns

In [11]:
```
gdf_blockgroups = (gpd.GeoDataFrame(fcc_df.merge(blockgroups[['geometry','are
a']], on="geo_12", how="inner"), crs=blockgroups.crs))
gdf_blockgroups.head(2)
```

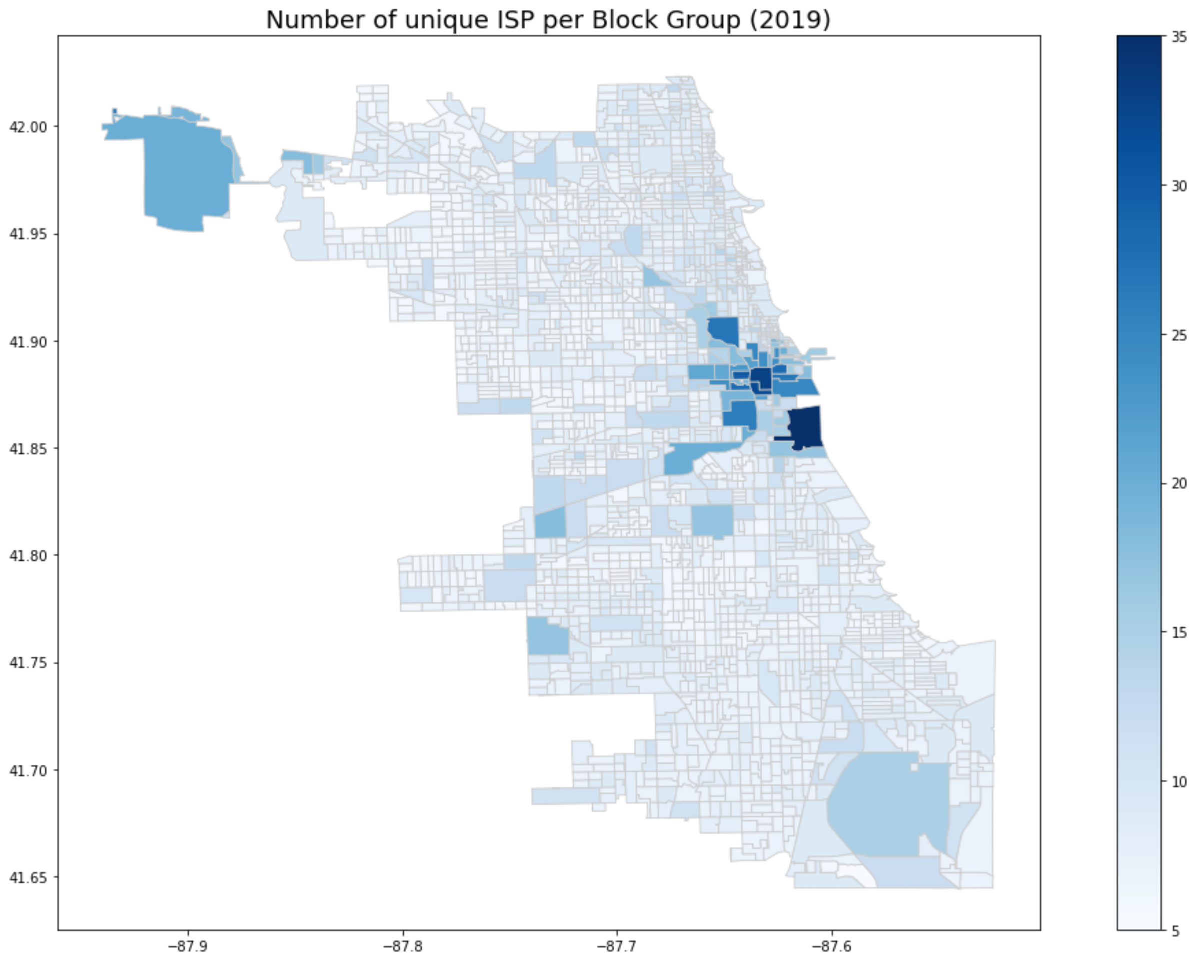
Out[11]:

	geo_12	maxcirdown	target	provider_id	isp_broadband	geometry	area
0	170310101001	0.0	0	7	0	POLYGON ((-87.67009 42.02115, -87.67047 42.021...	0.113260
1	170310101002	1000.0	1	9	1	POLYGON ((-87.66950 42.01936, -87.66963 42.019...	0.334453

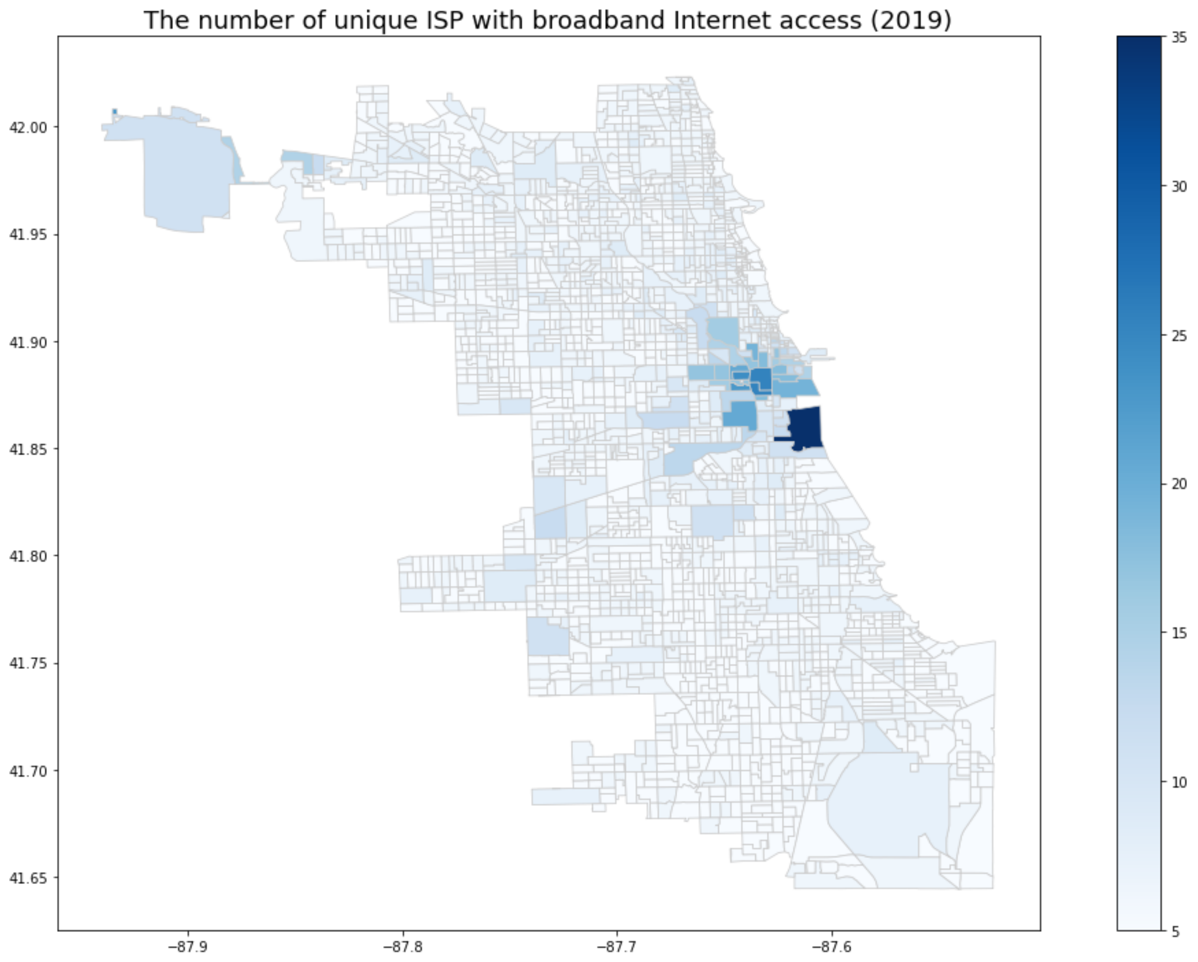
In [12]:
```
variable = 'maxcirdown'
vmin, vmax = 0, 1000
fig, ax = plt.subplots(1, figsize= (20, 12))
gdf_blockgroups.plot(column=variable, cmap='Blues', linewidth=0.8, ax=ax, edge
color='0.8')
ax.set_title('Maximum contractual downstream speed per Block Group (2019)', fo
ntdict={'fontsize': '18', 'fontweight' : '3'})
sm = plt.cm.ScalarMappable(cmap='Blues', norm=plt.Normalize(vmin=vmin, vmax=vm
ax))
sm._A=[]
cbar = fig.colorbar(sm)
```



In [13]:
```
variable = 'provider_id'
vmin, vmax = 5, 35
fig, ax = plt.subplots(1, figsize= (20, 12))
gdf_blockgroups.plot(column=variable, cmap='Blues', linewidth=0.8, ax=ax, edge
color='0.8')
ax.set_title('Number of unique ISP per Block Group (2019)', fontdict={'fontsiz
e': '18', 'fontweight' : '3'})
sm = plt.cm.ScalarMappable(cmap='Blues', norm=plt.Normalize(vmin=vmin, vmax=vm
ax))
sm._A=[]
cbar = fig.colorbar(sm)
```



In [14]:
```
variable = 'isp_broadband'
vmin, vmax = 5, 35
fig, ax = plt.subplots(1, figsize= (20, 12))
gdf_blockgroups.plot(column=variable, cmap='Blues', linewidth=0.8, ax=ax, edge
color='0.8')
ax.set_title('The number of unique ISP with broadband Internet access (2019)',
             fontdict={'fontsize': '18', 'fontweight' : '3'})
sm = plt.cm.ScalarMappable(cmap='Blues', norm=plt.Normalize(vmin=vmin, vmax=vm
ax))
sm._A=[]
cbar = fig.colorbar(sm)
```



2.2 ACS Data

The American Community Survey (ACS) provides data on broadband Internet access subscriptions, as reported by participants. For this lab, we will use the ACS 5-year estimates at the Census block group level for 2018.

Use the Census API to perform the following for block groups in the City of Chicago:

1. Load ACS data for the following field:
   - the percentages of broadband Internet access of any type
2. Load ACS data for Total Population, White, Black, and Median Income – and then compute the following:
   - the percentage of each Census block group's population that is White and Black;
   - the median income for the block gloup;
   - the population density of the block group (e.g. in units of population per square kilometer)

Present your answer as a dataframe showing only the first few rows. Some of these computations should be familiar from the previous lab. We've provided code using the `censusdata` package for pulling the necessary ACS data.

In [15]:
```python
import censusdata

# Useful for finding the ACS tables you want
# censusdata.search('acs5', 2018, 'label', 'broadband')
# censusdata.printtable(censusdata.censustable('acs5', 2018, 'B28011'))

# Pull ACS data
census_tables = {
    'GEO_ID': 'GEO_ID',
    'B28011_001E': 'Internet Total',
    'B28011_004E': 'Broadband',
    'B02001_001E': 'Race Total',
    'B02001_002E': 'White',
    'B02001_003E': 'Black',
    'B19013_001E': 'Median Income'}

acs_df = censusdata.download("acs5",
                              2018,
                              censusdata.censusgeo([("state", "17"),
                                                    ("county", "031"),
                                                    ("tract", "*"),
                                                    ("block group", "*")]),
                              list(census_tables.keys()))

# Rename columns
acs_df.rename(columns=census_tables, inplace=True)

print(acs_df.shape)
acs_df.head(2)
```

```
(3993, 7)
```

Out[15]:

	GEO_ID	Internet Total	Broadband	Race Total	White	Black	Median Income
Block Group 1, Census Tract 8088, Cook County, Illinois: Summary level: 150, state:17> county:031> tract:808800> block group:1	1500000US170318088001	788	690	1738	1538	70	96111
Block Group 2, Census Tract 8088, Cook County, Illinois: Summary level: 150, state:17> county:031> tract:808800> block group:2	1500000US170318088002	601	516	1994	1672	39	173646

In [16]:
```
acs_df["geo_12"] = acs_df["GEO_ID"].map(lambda x: str(x)[-12:])
acs_df["perc_broadband"] = acs_df["Broadband"]/ acs_df["Internet Total"]
acs_df["perc_white"] = acs_df["White"]/acs_df["Race Total"]
acs_df["perc_black"] = acs_df["Black"]/acs_df["Race Total"]
acs_df.head(2)
```

Out[16]:

	GEO_ID	Internet Total	Broadband	Race Total	White	Black	Median Income	
Block Group 1, Census Tract 8088, Cook County, Illinois: Summary level: 150, state:17> county:031> tract:808800> block group:1	1500000US170318088001	788	690	1738	1538	70	96111	17031
Block Group 2, Census Tract 8088, Cook County, Illinois: Summary level: 150, state:17> county:031> tract:808800> block group:2	1500000US170318088002	601	516	1994	1672	39	173646	17031

In [17]:
```
gdf = (gpd.GeoDataFrame(acs_df[["perc_broadband","perc_white","perc_black","ge
o_12","Race Total","Median Income"]].merge(gdf_blockgroups, on="geo_12", how=
"inner"), crs=blockgroups.crs))
gdf["pop_density"] = gdf["Race Total"]/gdf["area"]
gdf.head(2)
```

Out[17]:

	perc_broadband	perc_white	perc_black	geo_12	Race Total	Median Income	maxcirdown	targe
0	0.698413	0.964523	0.000000	170318220002	451	48869	1000.0	
1	0.619277	0.590588	0.017647	170311404004	850	-666666666	1000.0	

Part 3: Prediction with Tree-Based Models

In this part of the assignment, you will use the data and features that you computed to develop tree-based prediction models for broadband Internet deployment at the Census block group level.

3.1 Decision Trees

3.1.1 Training Decison Trees

First, train a decision tree classifier to predict if a Census block group has broadband Internet access or not (i.e., at least one ISP that offers service at or above 25 Mbps downstream and 3 Mbps upstream). Tune your classifier with a hyperparameter grid and use k-fold cross validation.

In the previous lab, you were asked to construct hyperparameter grids and manually send data through the pipeline. For this lab, you may use scikit-learn's inbuilt `Pipeline` and `GridSearchCV` objects to simplify this step.

Some additional notes on training:

- Use `random_state=0` when both splitting your data and training your classifiers.
- Given the size of this dataset, it will be sufficient to forgo the validation set and only separate your data into training and testing sets (i.e., use `sklearn.model_selection.test_train_split`)
- Your classifier should use the following features:
  - The ACS population characteristics from Section 2.2 (percentage of population that is White and Black, median income, population density)
  - The number of ISPs serving that block group
- Tune your classifier with a hyperparameter grid, using the following hyperparameter options:
  - `criterion` : `gini` or `entropy`
  - `max_depth` : 1, 3, 5
  - `min_samples_split` : 2, 5, 10
- Use K-fold cross validation with `k = 10` , using accuracy as your scoring metric.
- For evaluation metrics, print accuracy, precision, and recall on the test set for each hyperparameter combination.

Print a dataframe summarizing your grid search results. This should have one row for each classifier that you trained (18 total rows) specifying the hyperparamters for that classifier along with its evaluation metrics averaged across the folds (accuracy, precision, and recall).

In [18]:
```python
gdf = gdf[gdf['Median Income'] > 0]
```

In [19]:
```
x = gdf[["perc_white","perc_black","Median Income", "pop_density", "provider_i
d"]]
y = gdf[["target"]]

X_train, X_test, y_train, y_test = train_test_split (x,y, test_size = 0.20, ra
ndom_state=0)

print('Number of observations in \033[1mX_train\033[0m:{:,}'.format(X_train.sh
ape[0]),
      '\nNumber of observations in \033[1mX_test\033[0m:{:,}'.format(X_test.sh
ape[0]),'\n'
     '\nNumber of observations in \033[1my_train\033[0m:{:,}'.format(y_train.s
hape[0]),
     '\nNumber of observations in \033[1my_test\033[0m:{:,}'.format(y_test.sha
pe[0]),)
```

```
Number of observations in X_train:1,664
Number of observations in X_test:416

Number of observations in y_train:1,664
Number of observations in y_test:416
```

In [20]:
```
hyperparams = {'criterion': ['gini', 'entropy'],
                'max_depth': [1,3,5],
                'min_samples_split': [2,5,10]}


grid_search = GridSearchCV(estimator = DecisionTreeClassifier(random_state=0),
                           param_grid=hyperparams,
                           cv=10,
                           return_train_score=True,
                           scoring =['accuracy','precision','recall'],
                           refit = 'accuracy')

grid_search.fit(X_train, y_train)

cv_results =  pd.DataFrame(grid_search.cv_results_)

pd.set_option('max_colwidth', 400)
cv_results[['params','mean_train_accuracy', 'mean_train_precision','mean_train
_recall']]
```

Out[20]:

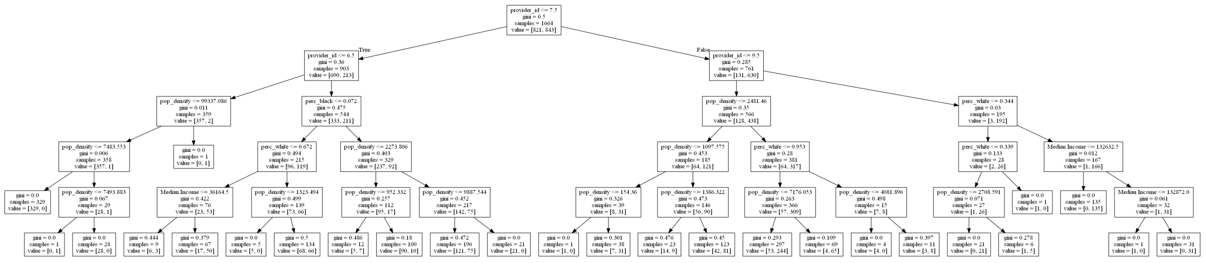
	params	mean_train_accuracy	mean_train_precision	mean_train_recall
0	{'criterion': 'gini', 'max_depth': 1, 'min_samples_split': 2}	0.793269	0.827844	0.747333
1	{'criterion': 'gini', 'max_depth': 1, 'min_samples_split': 5}	0.793269	0.827844	0.747333
2	{'criterion': 'gini', 'max_depth': 1, 'min_samples_split': 10}	0.793269	0.827844	0.747333
3	{'criterion': 'gini', 'max_depth': 3, 'min_samples_split': 2}	0.807558	0.766976	0.890867
4	{'criterion': 'gini', 'max_depth': 3, 'min_samples_split': 5}	0.807558	0.766976	0.890867
5	{'criterion': 'gini', 'max_depth': 3, 'min_samples_split': 10}	0.807558	0.766976	0.890867
6	{'criterion': 'gini', 'max_depth': 5, 'min_samples_split': 2}	0.830862	0.820725	0.853296
7	{'criterion': 'gini', 'max_depth': 5, 'min_samples_split': 5}	0.830862	0.820725	0.853296
8	{'criterion': 'gini', 'max_depth': 5, 'min_samples_split': 10}	0.830462	0.820673	0.852373
9	{'criterion': 'entropy', 'max_depth': 1, 'min_samples_split': 2}	0.793269	0.827844	0.747333
10	{'criterion': 'entropy', 'max_depth': 1, 'min_samples_split': 5}	0.793269	0.827844	0.747333
11	{'criterion': 'entropy', 'max_depth': 1, 'min_samples_split': 10}	0.793269	0.827844	0.747333
12	{'criterion': 'entropy', 'max_depth': 3, 'min_samples_split': 2}	0.807558	0.766976	0.890867
13	{'criterion': 'entropy', 'max_depth': 3, 'min_samples_split': 5}	0.807558	0.766976	0.890867
14	{'criterion': 'entropy', 'max_depth': 3, 'min_samples_split': 10}	0.807558	0.766976	0.890867
15	{'criterion': 'entropy', 'max_depth': 5, 'min_samples_split': 2}	0.827390	0.812777	0.857776
16	{'criterion': 'entropy', 'max_depth': 5, 'min_samples_split': 5}	0.827390	0.812777	0.857776
17	{'criterion': 'entropy', 'max_depth': 5, 'min_samples_split': 10}	0.827123	0.812685	0.857248

3.1.2 Evaluation I: Interpreting Trees

Print the tree with the highest average test accuracy. Note that `sklearn.tree.export_graphviz` may be helpful.

In [21]:
```
from sklearn import tree
import pydotplus
from IPython.display import Image


dt = DecisionTreeClassifier(criterion = 'gini',
                             max_depth = 5,
                             random_state = 0)
dt.fit(X_train, y_train)

dot_data = tree.export_graphviz(dt, out_file=None, feature_names=["perc_white"
,"perc_black","Median Income", "pop_density", "provider_id"])

graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```

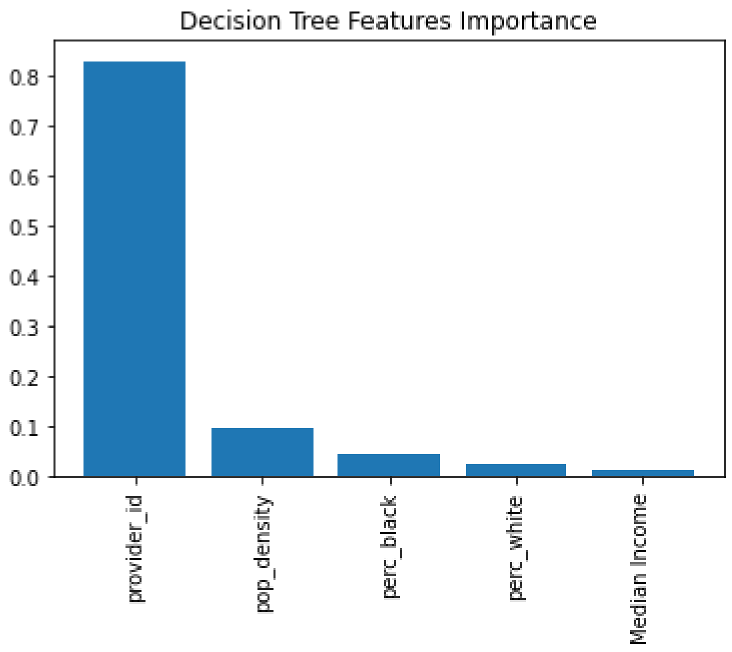
Out[21]:


provider_id <= 7.5 gini = 0.5 samples = 1664 value = [821, 843]
True
False
provider_id <= 6.5 gini = 0.36 samples = 903 value = [690, 213]
provider_id <= 9.5 gini = 0.285 samples = 761 value = [131, 630]
pop_density <= 99337.086 gini = 0.011 samples = 359 value = [357, 2]
perc_black <= 0.072 gini = 0.475 samples = 544 value = [333, 211]
pop_density <= 2481.46 gini = 0.35 samples = 566 value = [128, 438]
perc_white <= 0.344 gini = 0.03 samples = 195 value = [3, 192]
pop_density <= 7483.553 gini = 0.006 samples = 358 value = [357, 1]
gini = 0.0 samples = 1 value = [0, 1]
perc_white <= 0.672 gini = 0.494 samples = 215 value = [96, 119]
pop_density <= 2273.806 gini = 0.403 samples = 329 value = [237, 92]
pop_density <= 1097.575 gini = 0.453 samples = 185 value = [64, 121]
perc_white <= 0.953 gini = 0.28 samples = 381 value = [64, 317]
perc_white <= 0.339 gini = 0.133 samples = 28 value = [2, 26]
Median Income <= 132632.5 gini = 0.012 samples = 167 value = [1, 166]
gini = 0.0 samples = 329 value = [329, 0]
pop_density <= 7493.883 gini = 0.067 samples = 29 value = [28, 1]
Median Income <= 36164.5 gini = 0.422 samples = 76 value = [23, 53]
pop_density <= 1323.494 gini = 0.499 samples = 139 value = [73, 66]
pop_density <= 952.332 gini = 0.257 samples = 112 value = [95, 17]
pop_density <= 9887.544 gini = 0.452 samples = 217 value = [142, 75]
pop_density <= 154.36 gini = 0.326 samples = 39 value = [8, 31]
pop_density <= 1386.322 gini = 0.473 samples = 146 value = [56, 90]
pop_density <= 7176.053 gini = 0.263 samples = 366 value = [57, 309]
pop_density <= 4081.896 gini = 0.498 samples = 15 value = [7, 8]
pop_density <= 2708.591 gini = 0.071 samples = 27 value = [1, 26]
gini = 0.0 samples = 1 value = [1, 0]
gini = 0.0 samples = 135 value = [0, 135]
Median Income <= 132872.0 gini = 0.061 samples = 32 value = [1, 31]
gini = 0.0 samples = 1 value = [0, 1]
gini = 0.0 samples = 28 value = [28, 0]
gini = 0.444 samples = 9 value = [6, 3]
gini = 0.379 samples = 67 value = [17, 50]
gini = 0.0 samples = 5 value = [5, 0]
gini = 0.5 samples = 134 value = [68, 66]
gini = 0.486 samples = 12 value = [5, 7]
gini = 0.18 samples = 100 value = [90, 10]
gini = 0.472 samples = 196 value = [121, 75]
gini = 0.0 samples = 21 value = [21, 0]
gini = 0.0 samples = 1 value = [1, 0]
gini = 0.301 samples = 38 value = [7, 31]
gini = 0.476 samples = 23 value = [14, 9]
gini = 0.45 samples = 123 value = [42, 81]
gini = 0.293 samples = 297 value = [53, 244]
gini = 0.109 samples = 69 value = [4, 65]
gini = 0.0 samples = 4 value = [4, 0]
gini = 0.397 samples = 11 value = [3, 8]
gini = 0.0 samples = 21 value = [0, 21]
gini = 0.278 samples = 6 value = [1, 5]
gini = 0.0 samples = 1 value = [1, 0]
gini = 0.0 samples = 31 value = [0, 31]

The tree with **criterion** = gini and **max_depth** = 5, with varying **min_samples_split** where the ones with highest average test accuracy (tie between **min_sample_split** equals 2 and 5; and close when equaling 10). That is why I decided to plot the tree with the to first especifications.

What was the most important feature in the tree that yielded the highest accuracy?

In [22]:
```
labels = ["perc_white","perc_black","Median Income", "pop_density", "provider_
id"]
importances = dt.feature_importances_
indices = np.argsort(importances)[::-1]

names = [labels[i] for i in indices]

plt.figure()
plt.bar(range(x.shape[1]),importances[indices])
plt.xticks(range(x.shape[1]), names, rotation=90)
plt.title("Decision Tree Features Importance")
plt.show()
```



The most important feature in the plotted tree was **the number of unique ISP providers** per Block Group.

3.1.3 Evaluation II: Confusion Matrices and Precision-Recall Curves

For all questions in this section, use the decision tree from above that yielded the highest average accuracy.

Report the confusion matrix for this decision tree. Then use the output of this confusion matrix to manually calculate the following:

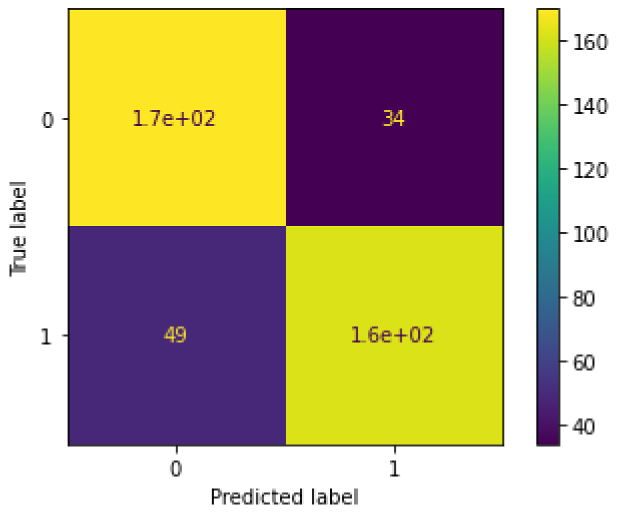
1. What is the test precision for this tree?
2. What is the test recall for this tree?
3. What is the test F1 score for this tree?

After manually computing these metrics, use the built-in `scikit-learn.metrics` functions to verify your answers.

In [23]:
```
y_pred = dt.predict(X_test)
```

In [24]:
```
from sklearn import metrics
from sklearn.metrics import plot_confusion_matrix, classification_report, plot
_roc_curve, plot_precision_recall_curve

plot_confusion_matrix(dt,X_test,y_test)
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[170  34]
 [ 49 163]]
```



- Precision: $\frac{TP}{TP+FP} = \frac{163}{163+34} = \frac{163}{197} = 0.827$
- Recall: $\frac{TP}{TP+FN} = \frac{163}{163+49} = \frac{163}{212} = 0.768$
- F1 Score: $2 \cdot \frac{P \cdot R}{P+R} = 2 \cdot \frac{0.827 \cdot [0.768]}{0.827+0.768} = 2 \cdot \frac{0.635}{1.595} = 2 \cdot 0.398 = 0.796$

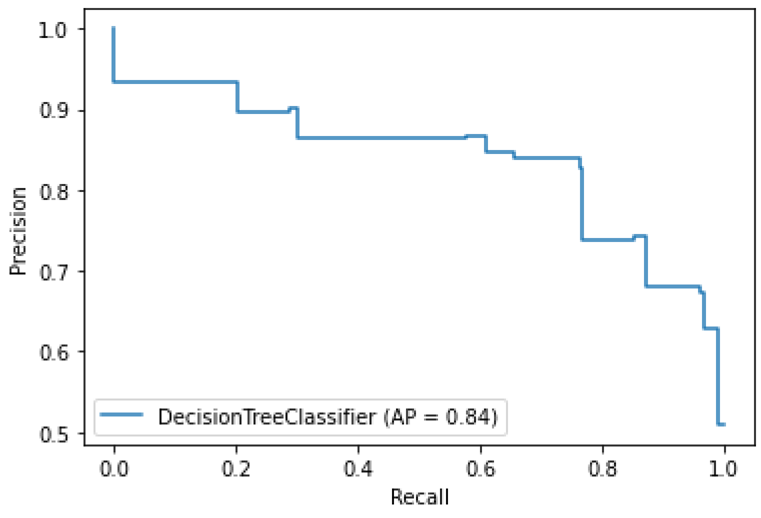
In [25]:
```
print("Precision:\t{}\nRecall:\t\t{}\nF1 Score:\t{}\n".format(metrics.precisio
n_score(y_test, y_pred),
                                                              metrics.recall_s
core(y_test, y_pred),
                                                              metrics.f1_score
(y_test, y_pred)
                                                                            ))
```

```
Precision:      0.8274111675126904
Recall:         0.7688679245283019
F1 Score:       0.7970660146699267
```

Plot the precision-recall curve for this decision tree. Then answer the following questions:

1. Describe what you observe in the precision-recall curve in plain language.
2. Does it make sense for precision to decrease as recall increases? Why or why not?
3. Can you identify a 'sweet spot' in the graph? (i.e., a threshold that balances nicely precision and recall)

In [26]:
```
plot_precision_recall_curve(dt,X_test,y_test)
plt.show()
```



1. Precision-recall curves plot the positive predictive value (precision) against the true positive rate (recall). This classifier that separates the two classes well but not perfectly, with an area under the curve of 0.84, close to 1.
2. Yes, because as our recall increases, our precision decreases because in addition to increasing the true positives, we also increase the false positives.
3. At threshold ~0.75.

3.1.4 Subsetting Features

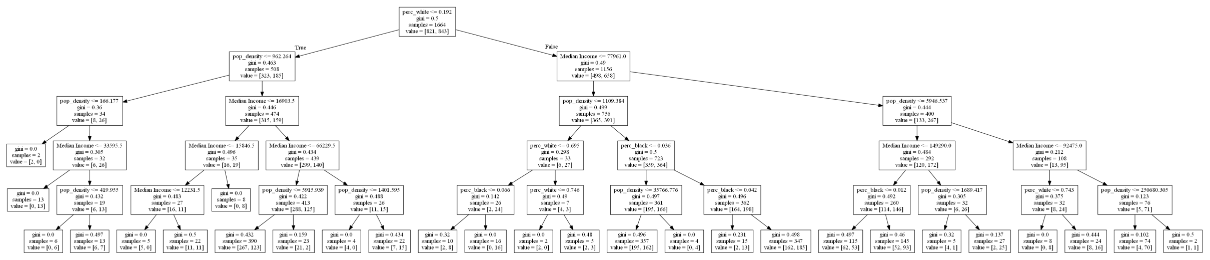
Note that the number of ISPs is closely (and directly) related to the predicted outcome. Suppose you no longer have access to the FCC data. Re-run your decision tree classification using just the ACS demographic features. How does this model perform compared to the model including the number of ISPs as a predictor?

In [27]:
```
x = gdf[["perc_white","perc_black","Median Income", "pop_density"]]
y = gdf["target"].values

X_train, X_test, y_train, y_test = train_test_split (x,y, test_size = 0.20, ra
ndom_state=0)

print('Number of observations in \033[1mX_train\033[0m:{:,}'.format(X_train.sh
ape[0]),
      '\nNumber of observations in \033[1mX_test\033[0m:{:,}'.format(X_test.sh
ape[0]),'\n'
     '\nNumber of observations in \033[1my_train\033[0m:{:,}'.format(y_train.s
hape[0]),
     '\nNumber of observations in \033[1my_test\033[0m:{:,}'.format(y_test.sha
pe[0]),)
```

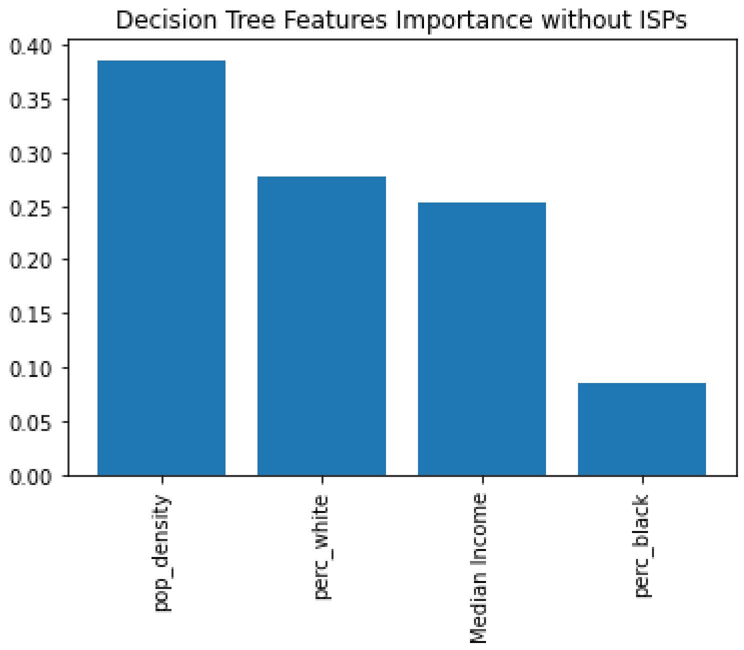
```
Number of observations in X_train:1,664
Number of observations in X_test:416

Number of observations in y_train:1,664
Number of observations in y_test:416
```

In [28]:
```
dt = DecisionTreeClassifier(criterion = 'gini',
                             max_depth = 5,
                             random_state = 0)
dt.fit(X_train, y_train)

dot_data = tree.export_graphviz(dt, out_file=None, feature_names=["perc_white"
,"perc_black","Median Income", "pop_density"])

graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```

Out[28]:


perc_white <= 0.192 gini = 0.5 samples = 1664 value = [821, 843]
True
False
pop_density <= 962.264 gini = 0.463 samples = 508 value = [323, 185]
Median Income <= 77961.0 gini = 0.49 samples = 1156 value = [498, 658]
pop_density <= 166.177 gini = 0.36 samples = 34 value = [8, 26]
Median Income <= 16903.5 gini = 0.446 samples = 474 value = [315, 159]
pop_density <= 1109.384 gini = 0.499 samples = 756 value = [365, 391]
pop_density <= 5946.537 gini = 0.444 samples = 400 value = [133, 267]
gini = 0.0 samples = 2 value = [2, 0]
Median Income <= 33595.5 gini = 0.305 samples = 32 value = [6, 26]
Median Income <= 15846.5 gini = 0.496 samples = 35 value = [16, 19]
Median Income <= 66229.5 gini = 0.434 samples = 439 value = [299, 140]
perc_white <= 0.695 gini = 0.298 samples = 33 value = [6, 27]
perc_black <= 0.036 gini = 0.5 samples = 723 value = [359, 364]
Median Income <= 149290.0 gini = 0.484 samples = 292 value = [120, 172]
Median Income <= 92475.0 gini = 0.212 samples = 108 value = [13, 95]
gini = 0.0 samples = 13 value = [0, 13]
pop_density <= 419.955 gini = 0.432 samples = 19 value = [6, 13]
Median Income <= 12231.5 gini = 0.483 samples = 27 value = [16, 11]
gini = 0.0 samples = 8 value = [0, 8]
pop_density <= 5915.939 gini = 0.422 samples = 413 value = [288, 125]
pop_density <= 1401.595 gini = 0.488 samples = 26 value = [11, 15]
perc_black <= 0.066 gini = 0.142 samples = 26 value = [2, 24]
perc_white <= 0.746 gini = 0.49 samples = 7 value = [4, 3]
pop_density <= 35766.776 gini = 0.497 samples = 361 value = [195, 166]
perc_black <= 0.042 gini = 0.496 samples = 362 value = [164, 198]
perc_black <= 0.012 gini = 0.492 samples = 260 value = [114, 146]
pop_density <= 1689.417 gini = 0.305 samples = 32 value = [6, 26]
perc_white <= 0.743 gini = 0.375 samples = 32 value = [8, 24]
pop_density <= 250680.305 gini = 0.123 samples = 76 value = [5, 71]
gini = 0.0 samples = 6 value = [0, 6]
gini = 0.497 samples = 13 value = [6, 7]
gini = 0.0 samples = 5 value = [5, 0]
gini = 0.5 samples = 22 value = [11, 11]
gini = 0.432 samples = 390 value = [267, 123]
gini = 0.159 samples = 23 value = [21, 2]
gini = 0.0 samples = 4 value = [4, 0]
gini = 0.434 samples = 22 value = [7, 15]
gini = 0.32 samples = 10 value = [2, 8]
gini = 0.0 samples = 16 value = [0, 16]
gini = 0.0 samples = 2 value = [2, 0]
gini = 0.48 samples = 5 value = [2, 3]
gini = 0.496 samples = 357 value = [195, 162]
gini = 0.0 samples = 4 value = [0, 4]
gini = 0.231 samples = 15 value = [2, 13]
gini = 0.498 samples = 347 value = [162, 185]
gini = 0.497 samples = 115 value = [62, 53]
gini = 0.46 samples = 145 value = [52, 93]
gini = 0.32 samples = 5 value = [4, 1]
gini = 0.137 samples = 27 value = [2, 25]
gini = 0.0 samples = 8 value = [0, 8]
gini = 0.444 samples = 24 value = [8, 16]
gini = 0.102 samples = 74 value = [4, 70]
gini = 0.5 samples = 2 value = [1, 1]

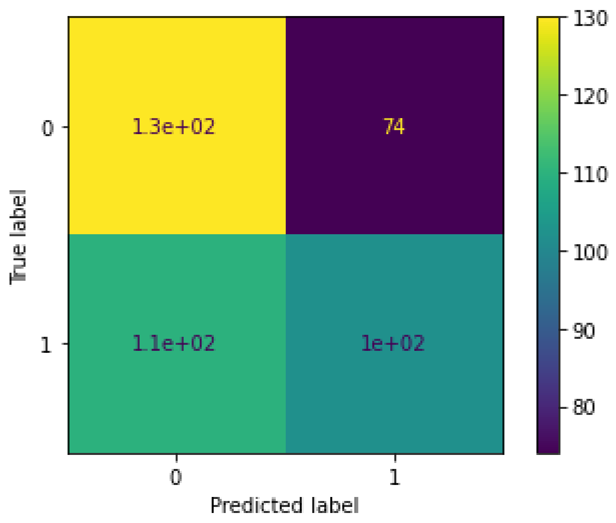
In [29]:
```
labels = ["perc_white","perc_black","Median Income", "pop_density", ]
importances = dt.feature_importances_
indices = np.argsort(importances)[::-1]

names = [labels[i] for i in indices]

plt.figure()
plt.bar(range(x.shape[1]),importances[indices])
plt.xticks(range(x.shape[1]), names, rotation=90)
plt.title("Decision Tree Features Importance without ISPs")
plt.show()
```



In [30]:
```
y_pred = dt.predict(X_test)
```

In [31]:
```
plot_confusion_matrix(dt,X_test,y_test)
print(metrics.confusion_matrix(y_test, y_pred))
```

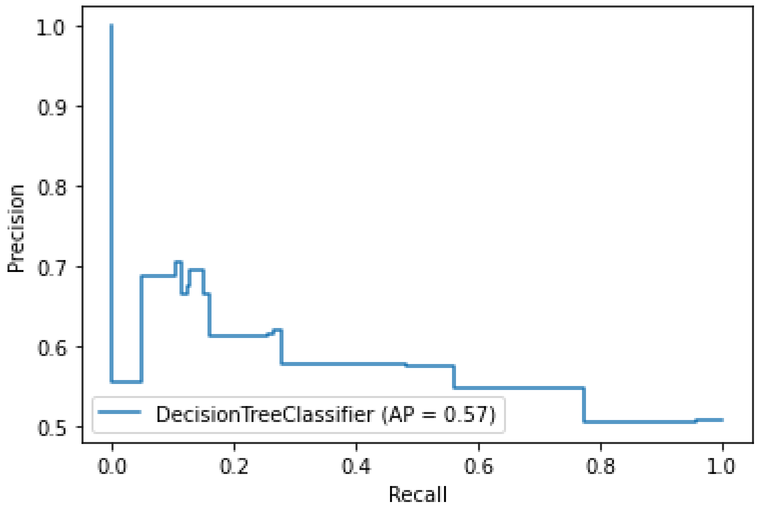
```
[[130  74]
 [110 102]]
```



In [32]:
```
print("Precision:\t{}\nRecall:\t\t{}\nF1 Score:\t{}\n".format(metrics.precisio
n_score(y_test, y_pred),
                                                                 metrics.recall_s
core(y_test, y_pred),
                                                                 metrics.f1_score
(y_test, y_pred)
                                                                            ))
```

```
Precision:      0.5795454545454546
Recall:         0.4811320754716981
F1 Score:       0.5257731958762887
```

In [33]:
```
plot_precision_recall_curve(dt,X_test,y_test)
plt.show()
```



The model without the most important feature is worse than the one with number of ISPs as a feature. All the metrics of the model decrease their value. As we can see in the Confusion Matrix, true positives and true negatives went down, while false positives and false negatives increased. Finally the Precision-Recall curve shows a poorer performance with an Area Under the Curve of 0.57.

3.2 Random Forests

3.2.1 Training Random Forests

Next, train a Random Forest classifier to predict broadband Internet deployment. As before, tune your classifier with a hyperparameter grid and use k-fold cross validation. Again, print a dataframe summarizing the classifiers, the hyperparameters for each classifier, and their respective evaluation metrics.

As with the previous section:

- You may use scikit-learn's inbuilt `Pipeline` and `GridSearchCV` functions if you prefer.
- Use `random_state=0` when both splitting your data and training your classifiers.
- Given the size of this dataset, it will be sufficient to forgo the validation set and only separate your data into training and testing sets (i.e., use `sklearn.model_selection.test_train_split`)
- Your classifier should use the following features:
  - The ACS population characteristics from Section 2.2 (percentage of population that is White and Black, median income, population density)
  - The number of ISPs serving that block group
- Tune your classifier with a hyperparameter grid, using the following hyperparameter options (note the differences for Random Forests):
  - `n_estimators` : 100, 1000, 5000
  - `criterion` : `gini` or `entropy`
  - `max_depth` : 1, 3, 5
  - `min_samples_split` : 2, 5, 10
- Use K-fold cross validation with `k = 10` , using accuracy as your scoring metric.
- For evaluation metrics, print accuracy, precision, and recall on the test set for each hyperparameter combination.

Note: This will take noticeably longer than training the single decision tree, but should not exceed 40 minutes. We highly recommend creating a smaller sample of this data to work with while you test your code.

In [34]:
```
x = gdf[["perc_white","perc_black","Median Income", "pop_density", "provider_i
d"]]
y = gdf["target"].values

X_train, X_test, y_train, y_test = train_test_split (x,y, test_size = 0.20, ra
ndom_state=0)

print('Number of observations in \033[1mX_train\033[0m:{:,}'.format(X_train.sh
ape[0]),
      '\nNumber of observations in \033[1mX_test\033[0m:{:,}'.format(X_test.sh
ape[0]),'\n'
     '\nNumber of observations in \033[1my_train\033[0m:{:,}'.format(y_train.s
hape[0]),
     '\nNumber of observations in \033[1my_test\033[0m:{:,}'.format(y_test.sha
pe[0]),)
```

```
Number of observations in X_train:1,664
Number of observations in X_test:416

Number of observations in y_train:1,664
Number of observations in y_test:416
```

In [35]:
```
from sklearn.ensemble import RandomForestClassifier

hyperparams = {'n_estimators':[100, 1000, 5000],
               'criterion': ['gini', 'entropy'],
               'max_depth': [1,3,5],
               'min_samples_split': [2,5,10]}


grid_search = GridSearchCV(estimator = RandomForestClassifier(random_state=0),
                           param_grid=hyperparams,
                           cv=10,
                           return_train_score=True,
                           scoring =['accuracy','precision','recall'],
                           refit = 'accuracy')

grid_search.fit(X_train, y_train)

cv_results =  pd.DataFrame(grid_search.cv_results_)

pd.set_option('max_colwidth', 400)
cv_results[['params','mean_train_accuracy', 'mean_train_precision','mean_train
_recall']]
```

Out[35]:

	params	mean_train_accuracy	mean_train_precision	mean_train_recall
0	{'criterion': 'gini', 'max_depth': 1, 'min_samples_split': 2, 'n_estimators': 100}	0.793269	0.827844	0.747333
1	{'criterion': 'gini', 'max_depth': 1, 'min_samples_split': 2, 'n_estimators': 1000}	0.793269	0.827844	0.747333
2	{'criterion': 'gini', 'max_depth': 1, 'min_samples_split': 2, 'n_estimators': 5000}	0.793269	0.827844	0.747333
3	{'criterion': 'gini', 'max_depth': 1, 'min_samples_split': 5, 'n_estimators': 100}	0.793269	0.827844	0.747333
4	{'criterion': 'gini', 'max_depth': 1, 'min_samples_split': 5, 'n_estimators': 1000}	0.793269	0.827844	0.747333
5	{'criterion': 'gini', 'max_depth': 1, 'min_samples_split': 5, 'n_estimators': 5000}	0.793269	0.827844	0.747333
6	{'criterion': 'gini', 'max_depth': 1, 'min_samples_split': 10, 'n_estimators': 100}	0.793269	0.827844	0.747333
7	{'criterion': 'gini', 'max_depth': 1, 'min_samples_split': 10, 'n_estimators': 1000}	0.793269	0.827844	0.747333
8	{'criterion': 'gini', 'max_depth': 1, 'min_samples_split': 10, 'n_estimators': 5000}	0.793269	0.827844	0.747333
9	{'criterion': 'gini', 'max_depth': 3, 'min_samples_split': 2, 'n_estimators': 100}	0.793537	0.827568	0.748387
10	{'criterion': 'gini', 'max_depth': 3, 'min_samples_split': 2, 'n_estimators': 1000}	0.793403	0.827423	0.748256
11	{'criterion': 'gini', 'max_depth': 3, 'min_samples_split': 2, 'n_estimators': 5000}	0.793403	0.827329	0.748387
12	{'criterion': 'gini', 'max_depth': 3, 'min_samples_split': 5, 'n_estimators': 100}	0.793537	0.827568	0.748387
13	{'criterion': 'gini', 'max_depth': 3, 'min_samples_split': 5, 'n_estimators': 1000}	0.793403	0.827423	0.748256
14	{'criterion': 'gini', 'max_depth': 3, 'min_samples_split': 5, 'n_estimators': 5000}	0.793403	0.827329	0.748387
15	{'criterion': 'gini', 'max_depth': 3, 'min_samples_split': 10, 'n_estimators': 100}	0.793804	0.827860	0.748651
16	{'criterion': 'gini', 'max_depth': 3, 'min_samples_split': 10, 'n_estimators': 1000}	0.793403	0.827423	0.748256

	params	mean_train_accuracy	mean_train_precision	mean_train_recall
17	{'criterion': 'gini', 'max_depth': 3, 'min_samples_split': 10, 'n_estimators': 5000}	0.793403	0.827329	0.748387
18	{'criterion': 'gini', 'max_depth': 5, 'min_samples_split': 2, 'n_estimators': 100}	0.829793	0.822481	0.847360
19	{'criterion': 'gini', 'max_depth': 5, 'min_samples_split': 2, 'n_estimators': 1000}	0.828725	0.824068	0.841960
20	{'criterion': 'gini', 'max_depth': 5, 'min_samples_split': 2, 'n_estimators': 5000}	0.828925	0.824028	0.842487
21	{'criterion': 'gini', 'max_depth': 5, 'min_samples_split': 5, 'n_estimators': 100}	0.829192	0.822842	0.845386
22	{'criterion': 'gini', 'max_depth': 5, 'min_samples_split': 5, 'n_estimators': 1000}	0.828191	0.823242	0.841959
23	{'criterion': 'gini', 'max_depth': 5, 'min_samples_split': 5, 'n_estimators': 5000}	0.828725	0.823969	0.842092
24	{'criterion': 'gini', 'max_depth': 5, 'min_samples_split': 10, 'n_estimators': 100}	0.826922	0.820986	0.842615
25	{'criterion': 'gini', 'max_depth': 5, 'min_samples_split': 10, 'n_estimators': 1000}	0.825921	0.822860	0.836819
26	{'criterion': 'gini', 'max_depth': 5, 'min_samples_split': 10, 'n_estimators': 5000}	0.826789	0.823163	0.838532
27	{'criterion': 'entropy', 'max_depth': 1, 'min_samples_split': 2, 'n_estimators': 100}	0.790398	0.815901	0.758401
28	{'criterion': 'entropy', 'max_depth': 1, 'min_samples_split': 2, 'n_estimators': 1000}	0.790064	0.812205	0.763412
29	{'criterion': 'entropy', 'max_depth': 1, 'min_samples_split': 2, 'n_estimators': 5000}	0.789997	0.811962	0.763676
30	{'criterion': 'entropy', 'max_depth': 1, 'min_samples_split': 5, 'n_estimators': 100}	0.790398	0.815901	0.758401
31	{'criterion': 'entropy', 'max_depth': 1, 'min_samples_split': 5, 'n_estimators': 1000}	0.790064	0.812205	0.763412
32	{'criterion': 'entropy', 'max_depth': 1, 'min_samples_split': 5, 'n_estimators': 5000}	0.789997	0.811962	0.763676

	params	mean_train_accuracy	mean_train_precision	mean_train_recall
33	{'criterion': 'entropy', 'max_depth': 1, 'min_samples_split': 10, 'n_estimators': 100}	0.790398	0.815901	0.758401
34	{'criterion': 'entropy', 'max_depth': 1, 'min_samples_split': 10, 'n_estimators': 1000}	0.790064	0.812205	0.763412
35	{'criterion': 'entropy', 'max_depth': 1, 'min_samples_split': 10, 'n_estimators': 5000}	0.789997	0.811962	0.763676
36	{'criterion': 'entropy', 'max_depth': 3, 'min_samples_split': 2, 'n_estimators': 100}	0.802218	0.807390	0.805584
37	{'criterion': 'entropy', 'max_depth': 3, 'min_samples_split': 2, 'n_estimators': 1000}	0.800680	0.799707	0.817051
38	{'criterion': 'entropy', 'max_depth': 3, 'min_samples_split': 2, 'n_estimators': 5000}	0.801281	0.801381	0.815205
39	{'criterion': 'entropy', 'max_depth': 3, 'min_samples_split': 5, 'n_estimators': 100}	0.802218	0.807390	0.805584
40	{'criterion': 'entropy', 'max_depth': 3, 'min_samples_split': 5, 'n_estimators': 1000}	0.800680	0.799538	0.817314
41	{'criterion': 'entropy', 'max_depth': 3, 'min_samples_split': 5, 'n_estimators': 5000}	0.801281	0.801323	0.815337
42	{'criterion': 'entropy', 'max_depth': 3, 'min_samples_split': 10, 'n_estimators': 100}	0.802752	0.807969	0.805848
43	{'criterion': 'entropy', 'max_depth': 3, 'min_samples_split': 10, 'n_estimators': 1000}	0.800547	0.799281	0.817578
44	{'criterion': 'entropy', 'max_depth': 3, 'min_samples_split': 10, 'n_estimators': 5000}	0.801014	0.800845	0.815469
45	{'criterion': 'entropy', 'max_depth': 5, 'min_samples_split': 2, 'n_estimators': 100}	0.830528	0.808467	0.873198
46	{'criterion': 'entropy', 'max_depth': 5, 'min_samples_split': 2, 'n_estimators': 1000}	0.832331	0.810484	0.873991

	params	mean_train_accuracy	mean_train_precision	mean_train_recall
47	{'criterion': 'entropy', 'max_depth': 5, 'min_samples_split': 2, 'n_estimators': 5000}	0.831864	0.810008	0.873596
48	{'criterion': 'entropy', 'max_depth': 5, 'min_samples_split': 5, 'n_estimators': 100}	0.830127	0.807723	0.873331
49	{'criterion': 'entropy', 'max_depth': 5, 'min_samples_split': 5, 'n_estimators': 1000}	0.831263	0.809741	0.872541
50	{'criterion': 'entropy', 'max_depth': 5, 'min_samples_split': 5, 'n_estimators': 5000}	0.831263	0.809585	0.872805
51	{'criterion': 'entropy', 'max_depth': 5, 'min_samples_split': 10, 'n_estimators': 100}	0.829594	0.808240	0.871090
52	{'criterion': 'entropy', 'max_depth': 5, 'min_samples_split': 10, 'n_estimators': 1000}	0.830261	0.809758	0.869905
53	{'criterion': 'entropy', 'max_depth': 5, 'min_samples_split': 10, 'n_estimators': 5000}	0.830194	0.809293	0.870564

3.2.2 Evaluation I: Interpreting Random Forests

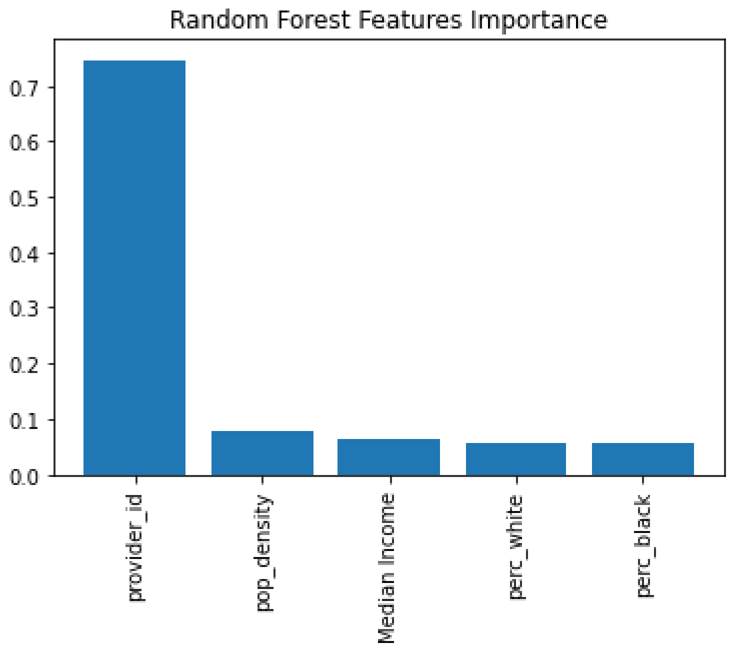
Identify the Random Forest with the highest average test accuracy. Create a plot showing feature importance.

In [36]:
```
cv_results['params'][cv_results.mean_train_accuracy == cv_results.mean_train_a
ccuracy.max()]
```

Out[36]:
```
46    {'criterion': 'entropy', 'max_depth': 5, 'min_samples_split': 2, 'n_est
imators': 1000}
Name: params, dtype: object
```

I used this method because **best*estimator*** was not returning the parameters with the highest average test accuracy.

In [37]:
```
rf = RandomForestClassifier(criterion = 'entropy',
                            max_depth = 5,
                            min_samples_split = 2,
                            n_estimators = 1000,
                            random_state = 0)
rf.fit(X_train, y_train)

labels = ["perc_white","perc_black","Median Income", "pop_density", "provider_
id"]
importances_rf = rf.feature_importances_
indices_rf = np.argsort(importances_rf)[::-1]
names_rf = [labels[i] for i in indices_rf]

plt.figure()
plt.bar(range(x.shape[1]),importances_rf[indices_rf])
plt.xticks(range(x.shape[1]), names_rf, rotation=90)
plt.title("Random Forest Features Importance")
plt.show()
```



Again, the most important feature was de number of unique ISPs provideers per Block Group

3.2.3 Evaluation II: Confusion Matrices and Precision Recall Curves

For all questions in this section, use the Random Forest classifier that yielded the highest average accuracy.

Report the confusion matrix for this Random Forest classifier using the test data. Then manually calculate the following (and feel free to then check your work with the built-in functions):

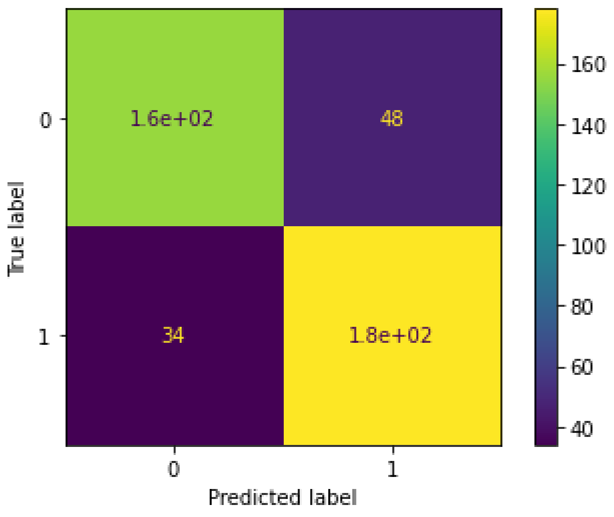
1. What is the test precision for this classifier?
2. What is the test recall for this classifier?
3. What is the test F1 score for this classifier?

How do these figures compare to the test metrics for the decision tree?

```
In [38]: y_pred = rf.predict(X_test)
```

```
In [39]: plot_confusion_matrix(rf,X_test,y_test)
         print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[156  48]
 [ 34 178]]
```



- Precision: $\frac{TP}{TP+FP} = \frac{178}{178+48} = \frac{178}{226} = 0.787$
- Recall: $\frac{TP}{TP+FN} = \frac{178}{178+34} = \frac{178}{212} = 0.839$
- F1 Score: $2 \cdot \frac{P \cdot R}{P+R} = 2 \cdot \frac{0.787 \cdot [0.839]}{0.787+0.839} = 2 \cdot \frac{0.660}{1.626} = 2 \cdot 0.406 = 0.812$

Plot the precision-recall curve for this Random Forest. On the same axes, plot the precision-recall curve for the decision tree that you previously plotted in Section 3.1.3. How do they compare? Which would you choose if you wanted to maximize precision? What if you wanted to maximize recall?

In [40]:
```
print("Precision:\t{}\nRecall:\t\t{}\nF1 Score:\t{}\n".format(metrics.precisio
n_score(y_test, y_pred),
                                                                     metrics.recall_s
core(y_test, y_pred),
                                                                     metrics.f1_score
(y_test, y_pred)
                                                                       ))
```

```
Precision:      0.7876106194690266
Recall:         0.839622641509434
F1 Score:       0.812785388127854
```

The **precision** or the fraction of true positives decreased, which means this model has more false positive cases (48 instead of 34 in the decision tree). There would be more cases in where the model indentifies that a Block Group has broadband but it does not. The **recall** or correct answers in the positive class increased, which means that this model has less false negative cases (34 instead of 49 in the decision tree). There would be less cases in which the model predicts a Block Group does not have broadband but it has. The **F1 Score**, which is the measure that balances precision and recall, is also higher.

3.3 Policy Applications and Model Selection

Imagine you have been hired by the City of Chicago, who is administering a survey about broadband quality. However, they do not know which block groups have broadband and must choose a model to predict broadband deployment.

1. It is important for you to reach as many block groups with broadband as possible without visiting every single block group. Of the classifiers you trained above, which should you use to predict broadband deployment in this situation? Why?
2. Budget cuts have struck! Priorities have shifted, and you are now told that you can only survey a small number of block groups. You are not worried about representativeness, and your main goal is to avoid wasting resources by visiting block groups that do not have broadband. Which classifier do you use now? Why?
3. You find out only after training that your population density data was in units of people per square mile, not people per square kilometer, meaning that your data would need to be rescaled to be properly interpreted. Do you now need to retrain your model with the rescaled data to get an updated set of predictions? Why or why not?
4. The City of Chicago is interested in better understanding the predictive features in your model. Why do you think the number of ISP providers and population density are important predictors?

There is a deep policy debate around broadband access related to the role of public investment, public-private-partnerships, and subsidies. This includes a discussion of 'rate-of-return' carriers and what ISPs should be allowed to charge for broadband access in rural areas. For further reading, see this discussion (https://www.usac.org/high-cost/resources/rules-orders/rate-of-return-reform-order/) on rate-of-return reform and this FCC briefing (https://docs.fcc.gov/public/attachments/FCC-19-77A1.pdf).

1. If I want to reach as many block groups with broadband as possible, that means I am interested in true positives then I want to minimize the cases that say they have broadband but they don't actually have it. In that case I would prefer the model that minimizes false negatives. I would choose Random Forest then, since the recall is higher, meaning that this model would have less false negative cases: 34 instead of 49 in the decision tree.
2. Since I want to avoid false positive cases, the ones that don't have broadband band but are reported as they do, I would choose the model with higher precision and less recall. In this case, that model would be Decision Tree.
3. No, rescaling won't change the results of decision trees either random forest. Rescale density population will be helpful only for the interpretation of the model.
4. More population density would mean more demand of both the service and better quality of the service. More population density creates more competition. Given that competition the number of ISPs would increase as well as the quality of their service. Assuming ISPs want to get more market share and maximize their profits.

3.4 Working with Thresholds

So far, we've been using GridSearch and Cross-Validation (`GridSearchCV`) to find the best models. Internally, this method is running `predict` on each validation fold, computing the evaluation metrics (accuracy, precision, etc.) for each fold and returning the average for the metrics. Recall that `predict` uses 0.5 as the default classification threshold, meaning every observation with a probability score above 0.5 is classified as True.

In many cases, `predict` is not ideal. Instead, we might want to choose a classification threshold ourselves, rather than use the default 0.5 cutoff. The threshold might be a fixed value (say every observation with a probability above 0.7 receives a predicted label of True). Alternatively, the threshold might be based on a percentage of the total observations (e.g. the top 10% of observations with the highest probability scores will be labelled as True, while the rest will be labelled as False). Which approach to choose will depend on your policy problem. In this exercise, we will explore the effect of using custom thresholds through both approaches.

Use the best Random Forest classifier identified in the previous section to output prediction probabilities for your testing dataset. Note that you will need to use `predict_proba` rather than `predict` to get these probability scores. Be sure that you understand what the output of `predict_proba` returns and its dimensions. Then, compute precision and recall for different thresholds.

3.4.1 Thresholds I: Fixed Values Approach

Use the following fixed values to convert probability scores (the output of `predict_proba`) into binary predictions: 0, 0.3, 0.5, 0.7, 0.9. Output a table summarizing the precision and recall at each of these thresholds.

In [41]:
```
probability_1 = rf.predict_proba(X_test)[:,1]
thresholds_df = pd.DataFrame(columns = ["Cutoff", 'Precision', "Recall"])
cutoffs = [0, 0.3, 0.5, 0.7, 0.9]
```

In [42]:
```
for x in cutoffs:
    y_pred = (probability_1 > x)
    precision = metrics.precision_score(y_test, y_pred)
    recall = metrics.recall_score(y_test, y_pred)
    thresholds_df = thresholds_df.append({"Cutoff": x, "Precision": precision,
"Recall": recall}, ignore_index = True)
```

In [43]:
```
thresholds_df
```

Out[43]:

	Cutoff	Precision	Recall
0	0.0	0.509615	1.000000
1	0.3	0.680782	0.985849
2	0.5	0.787611	0.839623
3	0.7	0.880795	0.627358
4	0.9	1.000000	0.070755

3.4.2 Thresholds II: Percentage Approach

Now, use the percentage-based approach using the following threshold percentages: 1%, 5%, 10%, 20%, 50% 100%.

Using the 1% threshold, for example, you would need to output probability scores for each observation, sort these probabilities in descending order based on this score, and then assign the top 1% True (and the bottom 99% False). If your dataset had 1,000 observations, that would mean that just 10 observations would be marked True.

Again, output a table summarizing the precision and recall at each of these thresholds.

In [44]:
```
probability_1[::-1].sort()
percentage_approach = pd.DataFrame(columns = ["Cutoff", 'Precision', "Recall"
])
cutoffs = [0.01, 0.05, 0.1, 0.2, 0.5, 1]
```

In [45]:
```
for x in cutoffs:
    i = round(len(probability_1)*x)
    y_pred[:i] = True
    y_pred[i:] = False
    precision = metrics.precision_score(y_test, y_pred)
    recall = metrics.recall_score(y_test, y_pred)
    percentage_approach = percentage_approach.append({"Cutoff": x, "Precision"
: precision, "Recall": recall}, ignore_index = True)
```

```
In [46]: percentage_approach
```

```
Out[46]:
```

	Cutoff	Precision	Recall
0	0.01	1.000000	0.018868
1	0.05	0.428571	0.042453
2	0.10	0.428571	0.084906
3	0.20	0.421687	0.165094
4	0.50	0.490385	0.481132
5	1.00	0.509615	1.000000

3.4.3 Summarizing Thresholds

Describe what you observed in the two summary tables above. How do precision and recall change when adjusting the thresholds using the two approaches? Are these results consistent with your precision-recall curves? Discuss a situation when using a threshold approach would make sense when evaluating classification models.

Thresholds with fixed values: When the cutoff is 0, all the observations will be set to True, that means that it will match correctly all the true positive cases and will identify the rest as false positve, that is why at this cutoff the precision is low. On the contrary, recall es high since it won't have false negative cases since the won't be any False in the prediction. On the other hand, when the cutoff is set to 0.9, there woul be more labels in the prediction and the false positive will decrease, making precision higher and recall lower.

Thresholds with percentages: This works the other way around. If you have the cutoff set to 0.01 that would mean that 10% of the cases would be set to True and 90% of the cases would be set to False, which means that, at the beginning, precision would be higher than recall because there is less likely to do a false positive match. The inverse is true as the cutoff increases.