Data Wrangling

September 28, 2021

1 Data Wrangling

1.1 Content

- Import Necessary Modules and Datasets
- Test Data by Linear Regression
- Record Linkage: Penalty vs Quality
- Record Linkage: Quality vs Info
- Record Linkage: Info vs Penalty
- Convert Dtypes
- Data Quality Analysis
- Measure Quality
- States and Cities
- Four Quarter Scores
- Used in Quality Measure Five Star Rating

1.1.1 Import Necessary Packages

[1]:
```
## Necessary packages
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
import pandas_profiling

from io import BytesIO
from zipfile import ZipFile
import urllib
import recordlinkage

from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import sklearn.model_selection as m_sel
```

[2]:
```
## Unzip and read file of Penality dataset
url = urllib.request.urlopen('https://github.com/jonahwinninghoff/
↪Springboard_Capstone_Project/blob/main/Assets/NH_Penalties_Jun2021.csv.zip?
↪raw=true')
```

```
file = ZipFile(BytesIO(url.read()))
csv= file.open('NH_Penalties_Jun2021.csv')
penalty = pd.read_csv(csv, encoding='cp1252')
file.close()
```

[3]:
```
url = urllib.request.urlopen('https://github.com/jonahwinninghoff/
 ↪Springboard_Capstone_Project/blob/main/Assets/NH_ProviderInfo_Aug2021.csv.
 ↪zip?raw=true')
file = ZipFile(BytesIO(url.read()))
csv = file.open('NH_ProviderInfo_Aug2021.csv')
information = pd.read_csv(csv, encoding='cp1252')
file.close()
```

[4]:
```
## Unzip and read file of MDS Measure Quality dataset
url = urllib.request.urlopen('https://github.com/jonahwinninghoff/
 ↪Springboard_Capstone_Project/raw/main/Assets/NH_QualityMsr_MDS_Jun2021.csv.
 ↪zip')
file = ZipFile(BytesIO(url.read()))
csv = file.open("NH_QualityMsr_MDS_Jun2021.csv")
quality = pd.read_csv(csv, encoding='cp1252')
file.close()
```

```
/opt/anaconda3/lib/python3.8/site-
packages/IPython/core/interactiveshell.py:3165: DtypeWarning: Columns (0) have
mixed types.Specify dtype option on import or set low_memory=False.
  has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
```

1.1.2 Test Data by Linear Regression

[5]:
```
## Split data by 70 and 30
df = quality[['Provider Zip Code','Measure Code',
              'Four Quarter Average Score']].dropna(how = 'any', axis = 0)
dataset_X, dataset_y = df[['Provider Zip Code','Measure Code']], df['Four␣
 ↪Quarter Average Score']
X_train, X_test, y_train, y_test = m_sel.train_test_split(dataset_X, dataset_y,
                                                          test_size=0.20,
                                                          random_state=666)
```

[6]:
```
# Create 10 folds of cross validation and use linear regression
model = LinearRegression()
cv = m_sel.KFold(n_splits=10, random_state=444, shuffle=True)
print(model.fit(X_train, y_train))
```

```
LinearRegression()
```

[7]:
```
# Create the 10 fold average metrics
thenames = ['Accuracy','Mean Absolute Error','Brier Score']
```

```
scorings = ['r2','neg_mean_absolute_error','neg_mean_squared_error']
scores = {}

for i in enumerate(thenames):
    scores[thenames[i[0]]] = m_sel.cross_val_score(model,
                                                   X_train, y_train,
                                                   cv=cv,␣
 ↪scoring=scorings[i[0]])
    scores[thenames[i[0]]] = np.absolute(round(np.
 ↪mean(scores[thenames[i[0]]]),3))
```

[8]:
```
print('The data problem that seems solvable is because the result is:␣
 ↪'+str(scores)+
      ', even though both average MSE and average MAE is extremely high.')
```

The data problem that seems solvable is because the result is: {'Accuracy': 0.209, 'Mean Absolute Error': 26.53, 'Brier Score': 1053.954}, even though both average MSE and average MAE is extremely high.

1.1.3 Record Linkage: Penalty vs Quality

[9]:
```
# Check if both have some same names of columns
compare = {}
for i in penalty.columns:
    if i in quality.columns:
        compare[i] = ['V']
    else:
        compare[i] = ['X']

display(pd.DataFrame(compare).T.reset_index().rename(columns={
    'index':'Both datasets',0:'do have'}))
```

	Both datasets	do have
0	Federal Provider Number	V
1	Provider Name	V
2	Provider Address	V
3	Provider City	V
4	Provider State	V
5	Provider Zip Code	V
6	Penalty Date	X
7	Penalty Type	X
8	Fine Amount	X
9	Payment Denial Start Date	X
10	Payment Denial Length in Days	X
11	Location	V
12	Processing Date	V

[10]:
```
# Instaniate the record linkage
indexer = recordlinkage.Index()
indexer.block('Federal Provider Number')
potential_links = indexer.index(quality, penalty)
```

[11]:
```
# Create the criteria for record linkage
compare = recordlinkage.Compare()

compare.string('Provider Name', 'Provider Name', threshold = 0.95)
compare.string('Provider Address', 'Provider Address', threshold = 0.95)
compare.exact('Provider State', 'Provider State')
compare.exact('Provider Zip Code', 'Provider Zip Code')
compare.string('Location', 'Location', threshold = 0.7)
compare.exact('Processing Date', 'Processing Date')

# The comparison vectors
compare_vectors = compare.compute(potential_links, quality, penalty)
```

[12]:
```
# Describe the comparing vectors
display(compare_vectors.describe())
```

	0	1	2	3	4	5
count	11348.0	11348.0	11348.0	11348.0	11348.0	11348.0
mean	1.0	1.0	1.0	1.0	1.0	1.0
std	0.0	0.0	0.0	0.0	0.0	0.0
min	1.0	1.0	1.0	1.0	1.0	1.0
25%	1.0	1.0	1.0	1.0	1.0	1.0
50%	1.0	1.0	1.0	1.0	1.0	1.0
75%	1.0	1.0	1.0	1.0	1.0	1.0
max	1.0	1.0	1.0	1.0	1.0	1.0

[13]:
```
# Check if all are matched
print(compare_vectors.sum(axis=1).value_counts().sort_index(ascending=False))
```

```
6.0    11348
dtype: int64
```

[14]:
```
df = penalty.merge(quality, how = 'right', on = 'Federal Provider Number',␣
↪suffixes = ('_merged',''), copy = True)
```

[15]:
```
# Remove redundancies in column names and use for check duplicaties
nomerged = []
for i in range(len(df.columns)):
    if df.columns.str.endswith('merged', na=False)[i]:
        None
    else:
        nomerged.append(df.columns[i])
```

[16]:
```
result = any(df[nomerged].duplicated())

if result == False:
    print('There is none of duplications existed in data')
else:
    cleaned_df[cleaned_df.duplicated()]
```

```
There is none of duplications existed in data
```

[17]:
```
# Complete dataset
dataset = df[nomerged].copy()
```

[18]:
```
# The problem is that this dataset only has Texas.
print(dataset[~dataset['Penalty Type'].isna()]['Provider State'].unique())
```

```
['TX']
```

1.1.4 Record Linkage: Quality vs Info

[19]:
```
# Check if both have some same names of columns
compare = {}
for i in information.columns:
    if i in quality.columns:
        compare[i] = ['V']
    else:
        compare[i] = ['X']

pd.set_option('display.max_rows', 1000)
compare = pd.DataFrame(compare).T.reset_index().rename(columns={
    'index':'Both datasets',0:'do have'})
display(compare[compare['do have'] == 'V'])
```

	Both datasets	do have
0	Federal Provider Number	V
1	Provider Name	V
2	Provider Address	V
3	Provider City	V
4	Provider State	V
5	Provider Zip Code	V
86	Location	V
87	Processing Date	V

[20]:
```
# Instaniate the record linkage
indexer = recordlinkage.Index()
indexer.block('Federal Provider Number')
potential_links = indexer.index(quality, information)
```

[21]:
```
# Create the criteria for record linkage
compare = recordlinkage.Compare()

compare.string('Provider Name', 'Provider Name', threshold = 0.95)
compare.string('Provider Address', 'Provider Address', threshold = 0.95)
compare.exact('Provider State', 'Provider State')
compare.exact('Provider Zip Code', 'Provider Zip Code')
compare.string('Location', 'Location', threshold = 0.95)
compare.exact('Processing Date', 'Processing Date')

# The comparison vectors
compare_vectors = compare.compute(potential_links, quality, information)
```

[22]:
```
# Describe the comparing vectors
display(compare_vectors.describe())
```

```
                   0              1         2              3              4  \
count  261532.000000  261532.000000  261532.0  261532.000000  261532.000000
mean        0.987474       0.998279       1.0       0.999794       0.001789
std         0.111218       0.041445       0.0       0.014368       0.042264
min         0.000000       0.000000       1.0       0.000000       0.000000
25%         1.000000       1.000000       1.0       1.000000       0.000000
50%         1.000000       1.000000       1.0       1.000000       0.000000
75%         1.000000       1.000000       1.0       1.000000       0.000000
max         1.000000       1.000000       1.0       1.000000       1.000000

              5
count  261532.0
mean        0.0
std         0.0
min         0.0
25%         0.0
50%         0.0
75%         0.0
max         0.0
```

1.1.5 Record Linkage: Info vs Penalty

[23]:
```
# Check if both have some same names of columns
compare = {}
for i in information.columns:
    if i in penalty.columns:
        compare[i] = ['V']
    else:
        compare[i] = ['X']

pd.set_option('display.max_rows', 1000)
```

```
compare = pd.DataFrame(compare).T.reset_index().rename(columns={
    'index':'Both datasets',0:'do have'})
display(compare[compare['do have'] == 'V'])
```

```
               Both datasets do have
0   Federal Provider Number       V
1             Provider Name       V
2          Provider Address       V
3             Provider City       V
4            Provider State       V
5         Provider Zip Code       V
86                 Location       V
87          Processing Date       V
```

[24]:
```
# Instaniate the record linkage
penalty['Federal Provider Number']= penalty['Federal Provider Number'].
 ↪astype('object')
indexer = recordlinkage.Index()
indexer.block('Federal Provider Number')
potential_links = indexer.index(information, penalty)
```

[25]:
```
# Create the criteria for record linkage
compare = recordlinkage.Compare()

compare.string('Provider Name', 'Provider Name', threshold = 0.95)
compare.string('Provider Address', 'Provider Address', threshold = 0.95)
compare.exact('Provider State', 'Provider State')
compare.exact('Provider Zip Code', 'Provider Zip Code')
compare.string('Location', 'Location', threshold = 0.95)
compare.exact('Processing Date', 'Processing Date')

# The comparison vectors
compare_vectors = compare.compute(potential_links, information, penalty)
```

[26]:
```
print(compare_vectors)
```

```
Empty DataFrame
Columns: [0, 1, 2, 3, 4, 5]
Index: []
```

In conclusion, the penalty dataset is unable to match information and quality datasets, so it is not useful at all. The information and quality datasets are not matched enough. So, both cannot be merged. But Feasible Generalized Least Square (FGLS) may be in use to solve this problem by having two-equation system.

1.1.6 Convert Dtypes

[27]:
```
## Create easy and fast way to convert data types
def multi_astypes(data,string):
    diction = {}                                   # Create empty dict
    thedict = {'o':'object', 'i':'int64', 'f':'float64',
               'b':'bool', 'd':'datetime64', 't':'timedelta',
               'c':'category'}
    thelist = list(string.lower())          # EX: obc -> [o,b,c]
    Error = 'Please use any letters of o, i, f, b, d, t, and c. Each letter␣
 ↪represents the first letters of '+str(thedict)

    for i in range(0,len(thelist)):
        if thelist[i] in thedict:
            thetype = thedict[thelist[i]]# EX: thedict[o] -> object
            diction[data.columns[i]] = thetype
        else:
            raise ValueError(Error)
            break

    return data.astype(diction)             # Convert all columns
```

[28]:
```
# Convert all data types at once
clean_quality = multi_astypes(quality,'oooooiiocfffffffffcood')
```

See what changes in data types

[29]:
```
dataset_dtypes = pd.DataFrame(quality.dtypes).reset_index()
clean_dataset_dtypes = pd.DataFrame(clean_quality.dtypes).reset_index()

dataset_dtypes.columns = ['variable', 'dtype']
clean_dataset_dtypes.columns = ['variable', 'cleaned dtype']
comp = dataset_dtypes.merge(clean_dataset_dtypes, how = 'inner',
                     on = 'variable')

comp['dtype'] = comp['dtype'].astype('str')
comp['cleaned dtype'] = comp['cleaned dtype'].astype('str')

display(comp[comp['dtype'] != comp['cleaned dtype']])
```

	variable	dtype	cleaned dtype
8	Resident type	object	category
19	Used in Quality Measure Five Star Rating	object	category
22	Processing Date	object	datetime64[ns]

1.1.7 Data Quality Analysis

[30]:
```
profile = pandas_profiling.ProfileReport(
    clean_quality, title="Data Quality Analysis: MDS Quality Measure")
```

[31]:
```
display(profile)
```

```
Summarize dataset:   0%|          | 0/36 [00:00<?, ?it/s]

Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]

Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]

<IPython.core.display.HTML object>
```

[32]:
```
profile.to_file("MDS_quality_measure.html")
```

```
Export report to file:   0%|          | 0/1 [00:00<?, ?it/s]
```

Data Quality Analysis - Key Points:

- `Federal Provider`, `Measure Period`, and `Processing Date` are useless.
- `Footnote` in each variable is useless.
- `Measure Score` in each variable is bimodally distributed.
- `Provider Name` < `Provider Address`
- `Provider State` variable contains 53 different states, not 50.
- `Provider Zip Code` is unique 9181 values in total and nonparametrically distributed.
- `Measure Code` = `Measure Description` in total 18 unique values.
- `Resident Type` has most number of long stay than that of short stay.
- `Used in Quality Measure Five Star Rating` has number of `No` equal to that of `Yes`.

[33]:
```
clean_quality = clean_quality.drop(columns=['Federal Provider Number','Footnote␣
↪for Q1 Measure Score',
                              'Footnote for Q2 Measure Score','Footnote for Q3␣
↪Measure Score',
                              'Footnote for Q4 Measure Score','Footnote for Four␣
↪Quarter Average Score',
                              'Measure Period','Processing Date'])
```

[34]:
```
profile = pandas_profiling.ProfileReport(
    information, title="Data Quality Analysis: Information")
```

[35]:
```
display(profile)
```

```
Summarize dataset:   0%|          | 0/101 [00:00<?, ?it/s]

Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]

Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

```
<IPython.core.display.HTML object>
```

[36]:
```
profile.to_file("information.html")
```

```
Export report to file:   0%|          | 0/1 [00:00<?, ?it/s]
```

Data Quality Analysis - Key Points:

- `Federal Provider`, `Provider Phone Number`, `Provider SSA`, `Provider County Code`, `Provider County Name`, `Legal Business Name`, `Special Focus Status`, `Reported Staffing Footnote`, `Physical Therapist Staffing Footnote`, and `Processing Date` are useless.
- `Footnote` in some variables can replace NA in other variables
- Several variables have normally distributed.
- `Abuse Icon` might be useful to predict the potential abuse in nursing home.

[37]:
```
information = information.drop(columns=['Federal Provider Number','Provider␣
↪Phone Number',
                                        'Provider SSA County Code','Provider␣
↪County Name','Legal Business Name',
                                        'Special Focus Status','Reported␣
↪Staffing Footnote',
                                        'Physical Therapist Staffing␣
↪Footnote','Processing Date'])
```

1.1.8 Measure Quality

Check duplications

[38]:
```
result = any(clean_quality.duplicated())

if result == False:
    print('There is none of duplications existed in data')
else:
    clean_quality[clean_quality.duplicated()]
```

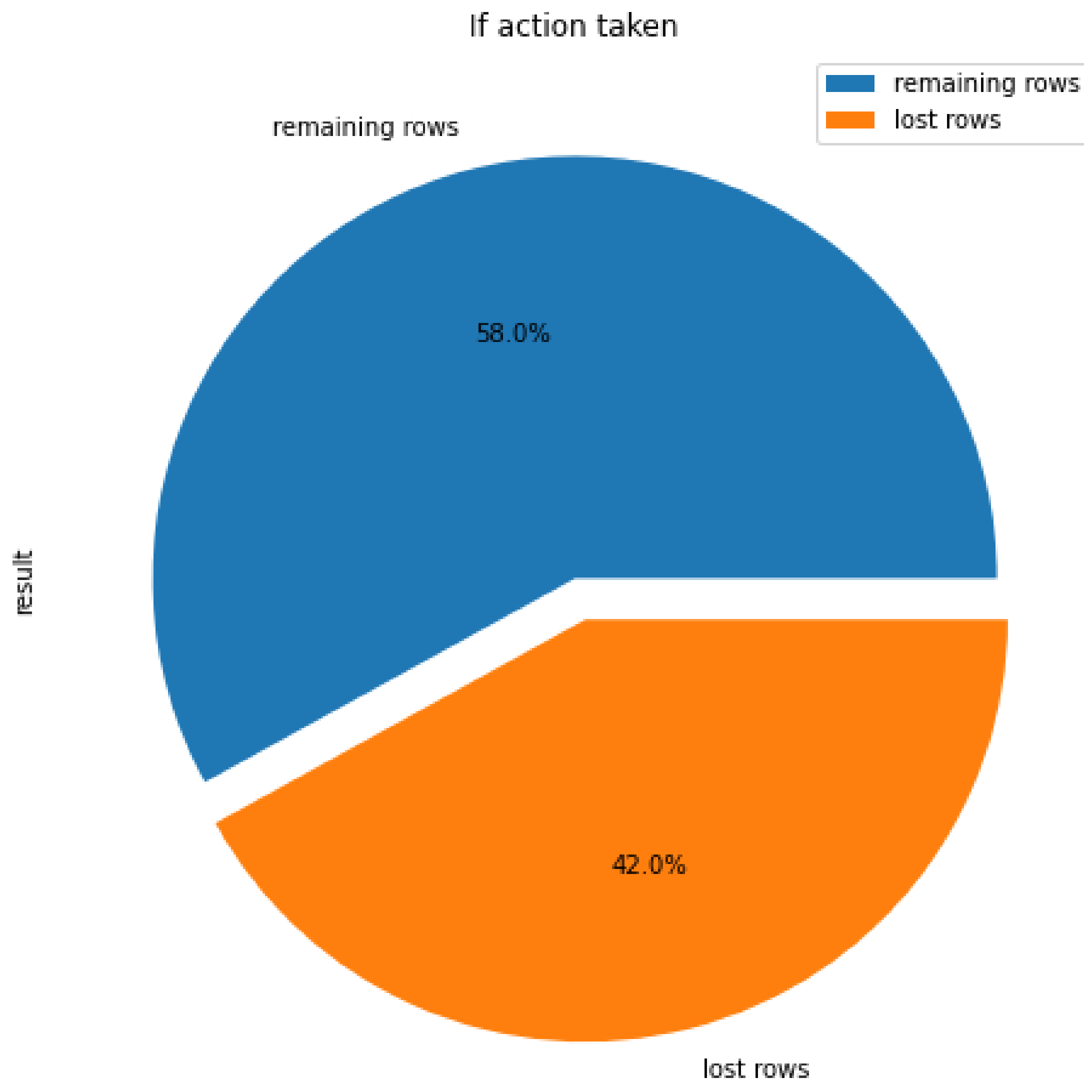
```
There is none of duplications existed in data
```

[39]:
```
result = any(information.duplicated())

if result == False:
    print('There is none of duplications existed in data')
else:
    information[information.duplicated()]
```

```
There is none of duplications existed in data
```

What if measure scores remove all missing values and zeros?

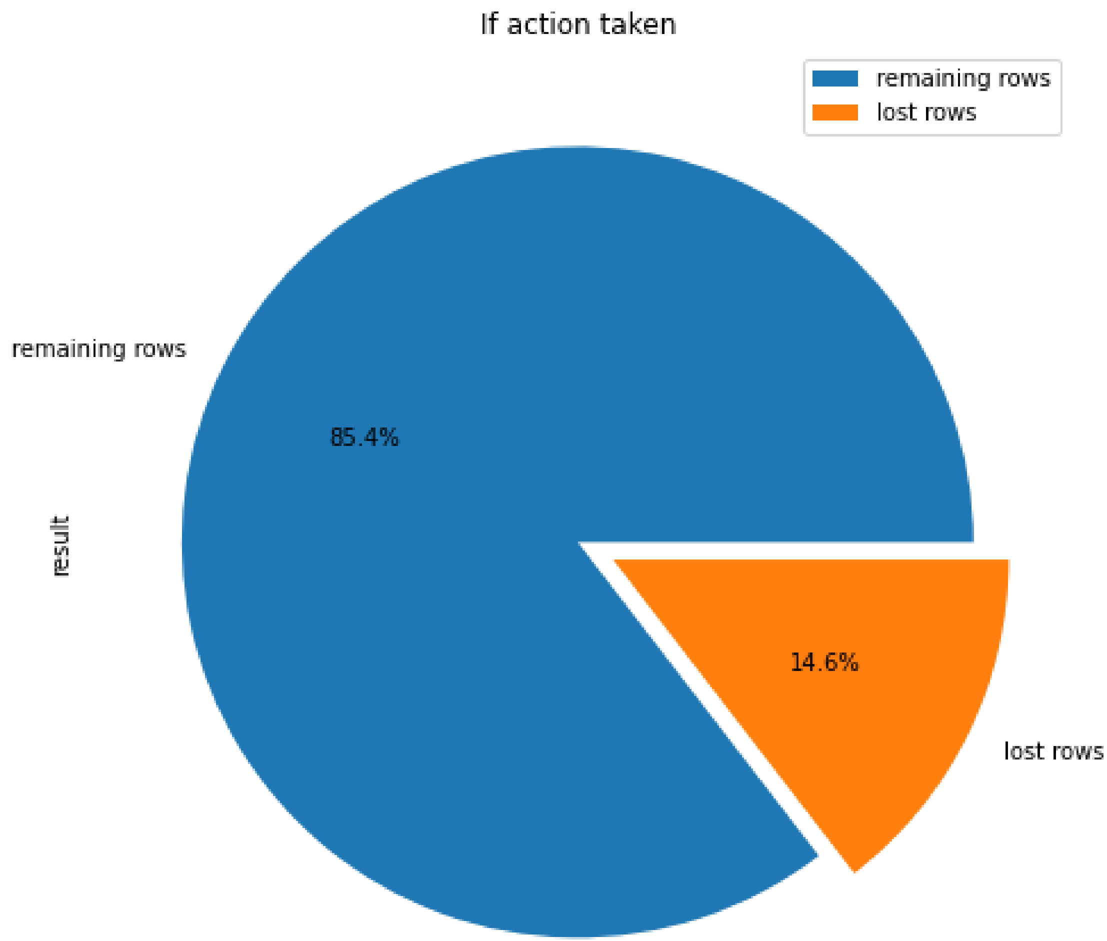
[40]:
```
## Copy the data for experiment
guinea_pig = clean_quality.copy()

guinea_pig = guinea_pig.dropna(how = 'any', axis = 0)     # Drop all NaN in␣
↪every row
guinea_pig = guinea_pig[~(guinea_pig==0.0).any(           # Drop all zeros in␣
↪every row
    axis = 1)]

# The result is:
pie_chart = pd.DataFrame({"result":
↪[len(guinea_pig),len(clean_quality)-len(guinea_pig)]},
                        index = ['remaining rows', 'lost rows'])

pie_chart.plot.pie(y='result', figsize=(8, 8), autopct='%1.1f%%', explode=(0, 0.
↪1), title = 'If action taken')
plt.show()
```



What if average measure score alone remove all missing values and zeros?

[41]:
```
## Copy the dataframe for second experiment
guinea_pig = clean_quality.copy()

guinea_pig = guinea_pig[guinea_pig['Four Quarter Average Score'].notnull()]  #␣
↪Remove NaN only in one column
guinea_pig = guinea_pig[guinea_pig['Four Quarter Average Score'] != 0]       #␣
↪Remove zeros only in one column



# The result is:
```

```
pie_chart = pd.DataFrame({"result":
↪[len(guinea_pig),len(clean_quality)-len(guinea_pig)]},
                        index = ['remaining rows', 'lost rows'])

pie_chart.plot.pie(y='result', figsize=(8, 8), autopct='%1.1f%%', explode=(0, 0.
↪1), title = 'If action taken')
plt.show()
```



The latter course of action is chosen

```
[42]: clean_quality = clean_quality[clean_quality['Four Quarter Average Score'].
↪notnull()]
clean_quality = clean_quality[clean_quality['Four Quarter Average Score'] != 0]
```

1.1.9 States and Cities

[43]:
```
# Copy and paste from https://gist.github.com/JeffPaine/3083347
states = ["AL", "AK", "AZ", "AR", "CA", "CO", "CT", "DC", "DE", "FL", "GA",
          "HI", "ID", "IL", "IN", "IA", "KS", "KY", "LA", "ME", "MD",
          "MA", "MI", "MN", "MS", "MO", "MT", "NE", "NV", "NH", "NJ",
          "NM", "NY", "NC", "ND", "OH", "OK", "OR", "PA", "RI", "SC",
          "SD", "TN", "TX", "UT", "VT", "VA", "WA", "WV", "WI", "WY"]
```

[44]:
```
# PR and GU are US territories, so remove them
print(clean_quality[~clean_quality['Provider State'].isin(states)]['Provider␣
 ↪State'].unique())
clean_quality = clean_quality[clean_quality['Provider State'].isin(states)]
```

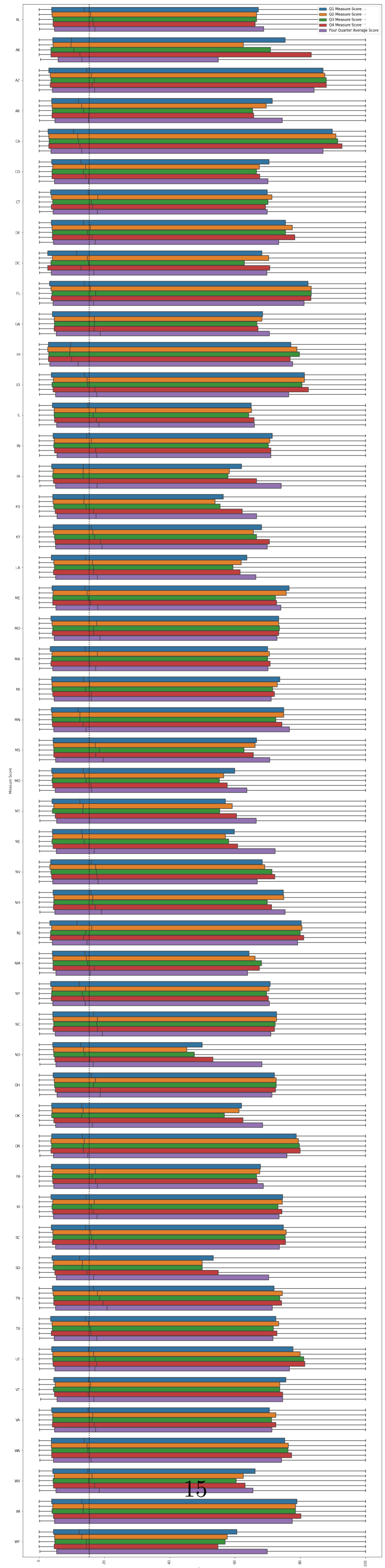
```
['PR' 'GU']
```

[45]:
```
# PR and GU are also removed in information dataset
print(information[~information['Provider State'].isin(states)]['Provider␣
 ↪State'].unique())
information = information[information['Provider State'].isin(states)]
```

```
['PR' 'GU']
```

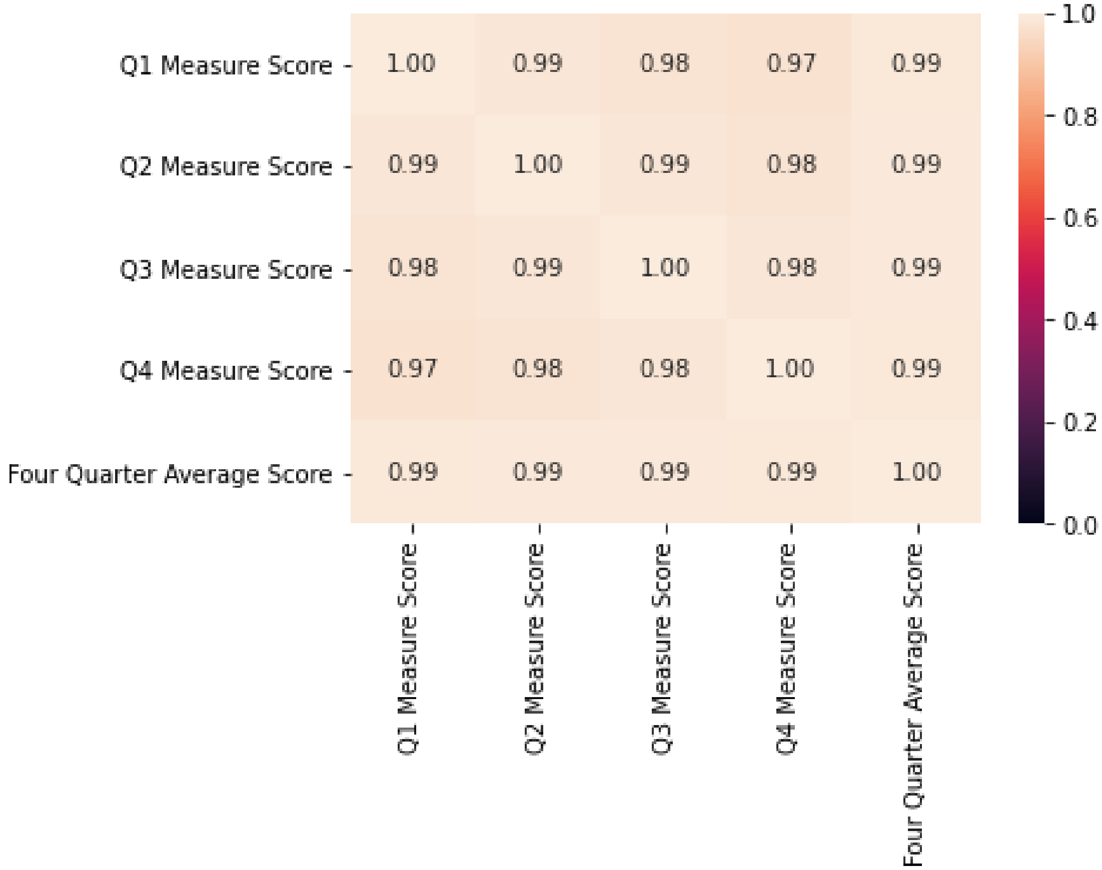
[46]:
```
melted_df = clean_quality.melt(id_vars = ['Provider Name', 'Provider Address',␣
 ↪'Provider City', 'Provider State',
                           'Provider Zip Code', 'Measure Code', 'Measure␣
 ↪Description',
                           'Resident type','Used in Quality Measure Five Star␣
 ↪Rating',
                           'Location'],
                value_vars =['Q1 Measure Score', 'Q2 Measure Score','Q3 Measure␣
 ↪Score',
                             'Q4 Measure Score','Four Quarter Average Score'],
                var_name ='Type of MS', value_name ='MS')

threshold = melted_df['MS'].agg('median')
```

[47]:
```
plt.subplots(figsize=(18, 80))
sns.boxplot(y='Provider State', x='MS', hue='Type of MS',␣
 ↪data=melted_df,orient="h")
plt.xticks(rotation='vertical')
plt.axvline(x=threshold,linewidth=2, color='black',linestyle=':')
plt.legend( loc = 'upper right')
plt.ylabel('Measure Score')
plt.xlabel('State')
plt.show()
```


Q1 Measure Score
Q2 Measure Score
Q3 Measure Score
Q4 Measure Score
Four Quarter Average Score
AL
AK
AZ
AR
CA
CO
CT
DE
DC
FL
GA
HI
ID
IL
IN
IA
KS
KY
LA
ME
MD
MA
MI
MN
MS
MO
MT
NE
NV
NH
NJ
NM
NY
NC
ND
OH
OK
OR
PA
RI
SC
SD
TN
TX
UT
VT
VA
WA
WV
WI
WY
Measure Score
0
20
40
60
80
100
State

1.1.10 Four Quarter Scores

[48]:
```
sns.heatmap(clean_quality[['Q1 Measure Score', 'Q2 Measure Score',
                            'Q3 Measure Score', 'Q4 Measure Score',
                            'Four Quarter Average Score']].corr(),
            vmin=0, vmax=1, annot=True,  fmt='.2f')
plt.show()
```



[49]:
```
## As indicated by above, Quarter 1-4 scores are highly correlated to Average␣
↪score, so remove them
clean_quality = clean_quality.drop(['Q1 Measure Score', 'Q2 Measure Score',
                                    'Q3 Measure Score', 'Q4 Measure␣
↪Score'],axis=1)
```

[50]:
```
## Create the function to identify variables that contain something
def identify_columns(data, contain):
```

```
    # If it is dataframe
    if type(data) == type(pd.DataFrame()):
        thelist = []
        for i in range(len(data.columns)):
            if contain in str.split(list(data.columns)[i]):
                thelist.append(list(data.columns)[i])
        return thelist
    # If it is a list
    elif type(data) == type([]):
        thelist = []
        for i in range(len(data)):
            if contain in str.split(data[i]):
                thelist.append(data[i])
        return thelist

## Create the function that removes several strings at once
def remove_columns(fulllist, somelist):
    for i in range(len(somelist)):
        fulllist.remove(somelist[i])
    return fulllist
```

[51]:
```
# Replace . with NaN
def fill_NaN(data):
    for j in data.columns:
        thelist = []
        for i in data[j]:
            if i == '.':
                thelist.append(np.nan)
            else:
                thelist.append(i)
        data[j] = thelist
    return data
```

[52]:
```
# Create the algorithm to see if rating and rating footnote can combine or not
def rating_footnote(data):
    # Identify each column that has its footnote
    footnote = identify_columns(data, 'Footnote')
    thelist = {}

    # Same variable with and without footnote
    for i in range(len(footnote)):
        thelist[footnote[i].replace(' Footnote','')] = [footnote[i].replace('␣
↪Footnote',''),footnote[i]]

    # Finally, three times check if rating and rating footnote are combinable
    for i in thelist.keys():
```

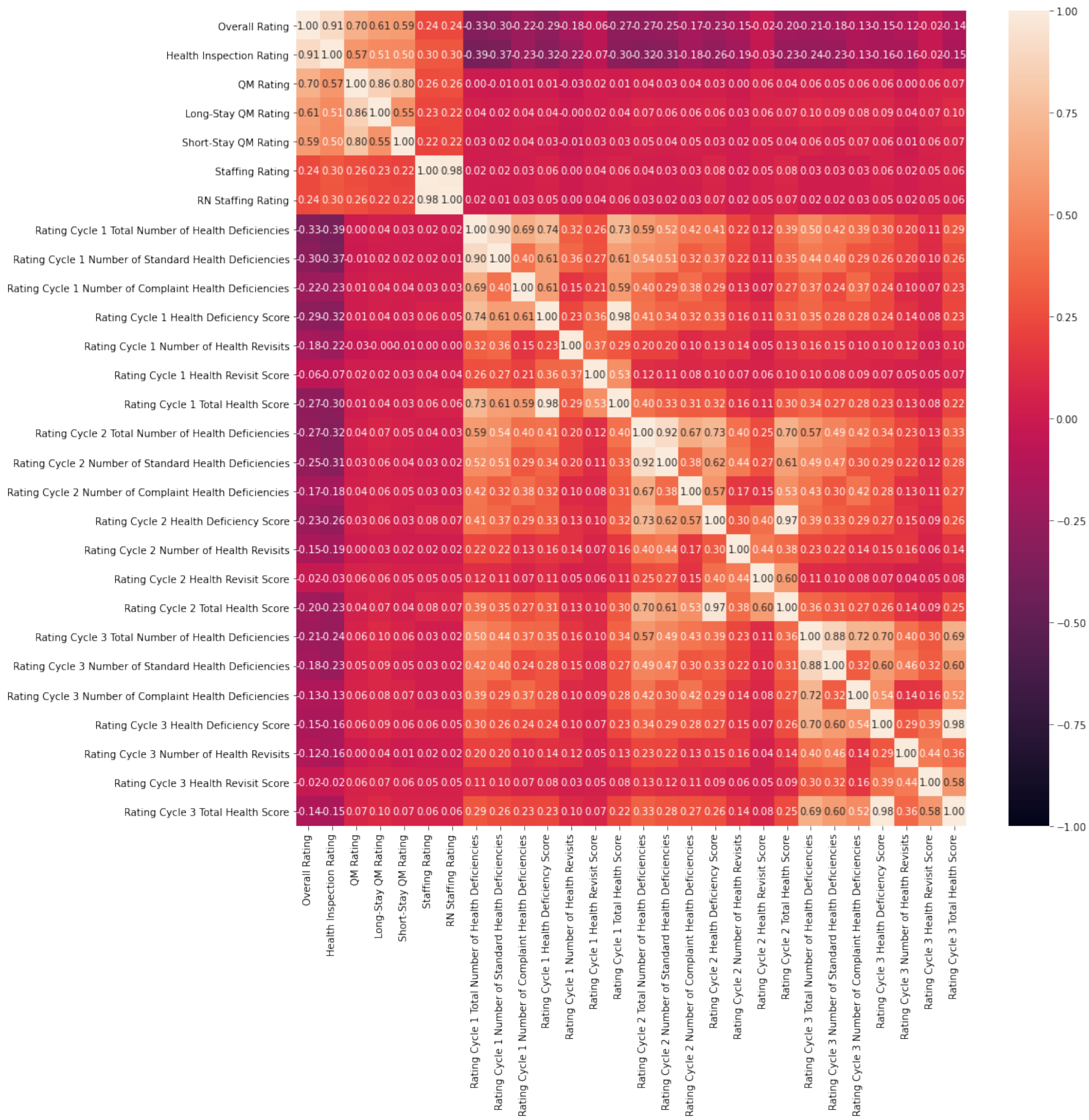
```
        if data[data[thelist[i][0]].isna()][thelist[i][1]].isna().any() ==␣
↪False:
            if data[data[thelist[i][1]].isna()][thelist[i][0]].isna().any() ==␣
↪False:
                n1 = len(data[data[thelist[i][0]].isna()][thelist[i][1]])
                n2 = len(data[data[thelist[i][1]].isna()][thelist[i][0]])
                if len(data) >= n1 + n2:
                    data = pd.concat([data[~data[thelist[i][1]].isna()].
↪drop(thelist[i][0], axis = 1).rename(
                        columns = {thelist[i][1]:thelist[i][0]}),
                        data[data[thelist[i][1]].isna()].drop(thelist[i][1],␣
↪axis = 1)], axis = 0)
    return data
```

[53]:
```
information = fill_NaN(information)
information = rating_footnote(information)
```

[54]:
```
# Identify rating-related columns and remove date-related columns
ratings = identify_columns(information, 'Rating')
ratings = remove_columns(ratings, identify_columns(ratings, 'Date'))
```

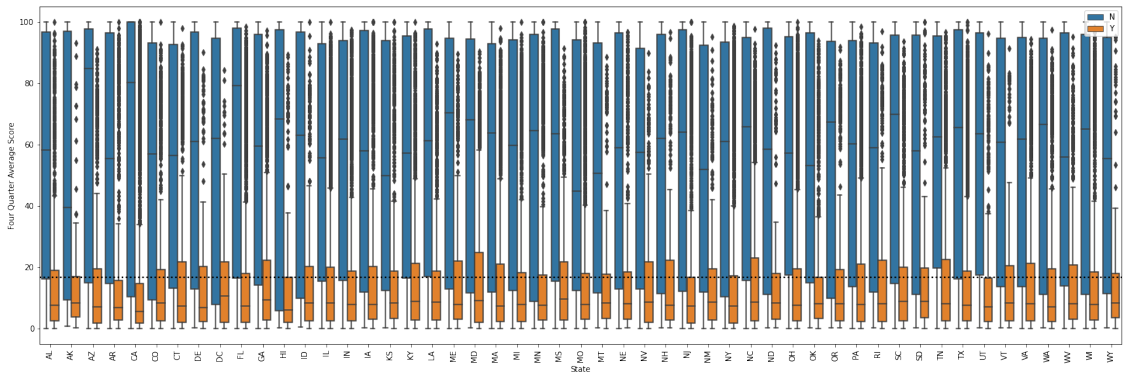
[55]:
```
# Coerce all rating-related variables into float
ratings = multi_astypes(information[ratings],len(information[ratings].
↪columns)*'f')
```

[56]:
```
plt.figure(figsize=(15,15))
sns.heatmap(ratings.corr(),
            vmin=-1, vmax=1, annot=True,  fmt='.2f')
plt.show()
```



1.1.11 Used in Quality Measure Five Star Rating

[57]:
```
plt.subplots(figsize=(25, 8))
sns.boxplot(x='Provider State', y='Four Quarter Average Score', hue='Used in␣
↪Quality Measure Five Star Rating', data=clean_quality,orient="v")
plt.xticks(rotation='vertical')
plt.axhline(y=clean_quality['Four Quarter Average Score'].
↪agg('median'),linewidth=2, color='black',linestyle=':')
plt.legend( loc = 'upper right')
plt.ylabel('Four Quarter Average Score')
plt.xlabel('State')
plt.show()
```



[58]:
```
missing_checks = ['Provider Name', 'Provider State', 'Provider Address',
                  'Provider Zip Code', 'Provider City','Location',
                  'Resident type', 'Used in Quality Measure Five Star Rating',
                 'Measure Description', 'Measure Code']
for i in missing_checks:
    print('Missing '+i+'?: '+str(clean_quality[i].isna().any()))
```

```
Missing Provider Name?: False
Missing Provider State?: False
Missing Provider Address?: False
Missing Provider Zip Code?: False
Missing Provider City?: False
Missing Location?: False
Missing Resident type?: False
Missing Used in Quality Measure Five Star Rating?: False
Missing Measure Description?: False
Missing Measure Code?: False
```

[61]:
```
clean_quality.to_csv("clean_quality.csv")
```

[62]:
```
information.to_csv("information.csv")
```