finalprojectfraud

April 7, 2024

[1]:
```
# Load some necessary packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

1 Importing Datasets

[2]:
```
import pandas as pd

# Load the labeled training data to understand its structure and the labels
train_labels_df = pd.read_csv('medicalfraud/Train-1542865627584.csv')

# Display the first few rows of the dataset to understand its structure
train_labels_df.head()
```

[2]:

	Provider	PotentialFraud
0	PRV51001	No
1	PRV51003	Yes
2	PRV51004	No
3	PRV51005	Yes
4	PRV51007	No

[3]:
```
# Load the data to understand its structure
beneficiary_data_df = pd.read_csv('medicalfraud/
 ↪Train_Beneficiarydata-1542865627584.csv')
inpatient_data_df = pd.read_csv('medicalfraud/Train_Inpatientdata-1542865627584.
 ↪csv')
outpatient_data_df = pd.read_csv('medicalfraud/
 ↪Train_Outpatientdata-1542865627584.csv')
```

[4]:
```
import pandas as pd

# Show the unique count of providers
print(f"Unique count of providers: {train_labels_df['Provider'].nunique()}")

# Show providers with 'Potential Fraud' as 'Yes' and 'No'
```

```
fraud_counts = train_labels_df['PotentialFraud'].value_counts()
total_providers = len(train_labels_df)

print("\nProviders with 'Potential Fraud':")
for fraud_value, count in fraud_counts.items():
    percentage = (count / total_providers) * 100
    print(f"{fraud_value}: {count} ({percentage:.2f}%)")
```

```
Unique count of providers: 5410

Providers with 'Potential Fraud':
No: 4904 (90.65%)
Yes: 506 (9.35%)
```

As we can see this is complete imbalance dataset. Having only 506 Potential Fraud Providers

1.1 Understanding the Data & It's Features

[5]:
```
## Data exploration

# Look at the column names and data types of the data frame
print(beneficiary_data_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 138556 entries, 0 to 138555
Data columns (total 25 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   BeneID                          138556 non-null  object
 1   DOB                             138556 non-null  object
 2   DOD                             1421 non-null    object
 3   Gender                          138556 non-null  int64
 4   Race                            138556 non-null  int64
 5   RenalDiseaseIndicator           138556 non-null  object
 6   State                           138556 non-null  int64
 7   County                          138556 non-null  int64
 8   NoOfMonths_PartACov             138556 non-null  int64
 9   NoOfMonths_PartBCov             138556 non-null  int64
 10  ChronicCond_Alzheimer           138556 non-null  int64
 11  ChronicCond_Heartfailure        138556 non-null  int64
 12  ChronicCond_KidneyDisease       138556 non-null  int64
 13  ChronicCond_Cancer              138556 non-null  int64
 14  ChronicCond_ObstrPulmonary      138556 non-null  int64
 15  ChronicCond_Depression          138556 non-null  int64
 16  ChronicCond_Diabetes            138556 non-null  int64
 17  ChronicCond_IschemicHeart       138556 non-null  int64
 18  ChronicCond_Osteoporasis        138556 non-null  int64
```

```
 19  ChronicCond_rheumatoidarthritis  138556 non-null  int64
 20  ChronicCond_stroke               138556 non-null  int64
 21  IPAnnualReimbursementAmt         138556 non-null  int64
 22  IPAnnualDeductibleAmt            138556 non-null  int64
 23  OPAnnualReimbursementAmt         138556 non-null  int64
 24  OPAnnualDeductibleAmt            138556 non-null  int64
dtypes: int64(21), object(4)
memory usage: 26.4+ MB
None
```

[6]:
```
beneficiary_data_df.head()
```

[6]:

	BeneID	DOB	DOD	Gender	Race	RenalDiseaseIndicator	State	\
0	BENE11001	1943-01-01	NaN	1	1	0	39	
1	BENE11002	1936-09-01	NaN	2	1	0	39	
2	BENE11003	1936-08-01	NaN	1	1	0	52	
3	BENE11004	1922-07-01	NaN	1	1	0	39	
4	BENE11005	1935-09-01	NaN	1	1	0	24	

	County	NoOfMonths_PartACov	NoOfMonths_PartBCov	…	\
0	230	12	12	…	
1	280	12	12	…	
2	590	12	12	…	
3	270	12	12	…	
4	680	12	12	…	

	ChronicCond_Depression	ChronicCond_Diabetes	ChronicCond_IschemicHeart	\
0	1	1	1	
1	2	2	2	
2	2	2	1	
3	2	1	1	
4	2	1	2	

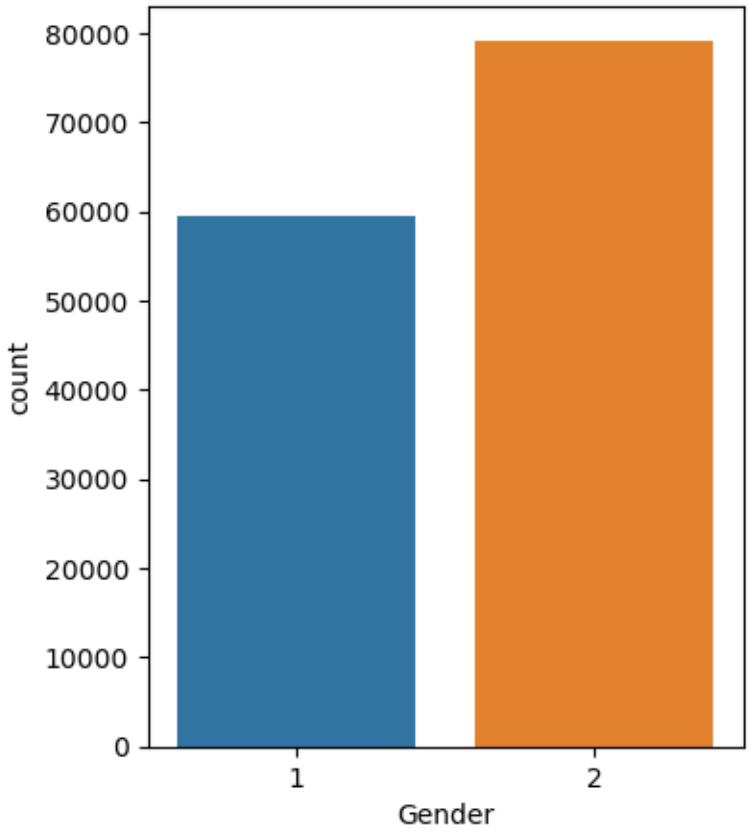
	ChronicCond_Osteoporasis	ChronicCond_rheumatoidarthritis	\
0	2	1	
1	2	2	
2	2	2	
3	1	1	
4	2	2	

	ChronicCond_stroke	IPAnnualReimbursementAmt	IPAnnualDeductibleAmt	\
0	1	36000	3204	
1	2	0	0	
2	2	0	0	
3	2	0	0	
4	2	0	0	

```
   OPAnnualReimbursementAmt  OPAnnualDeductibleAmt
0                        60                     70
1                        30                     50
2                        90                     40
3                      1810                    760
4                      1790                   1200

[5 rows x 25 columns]
```

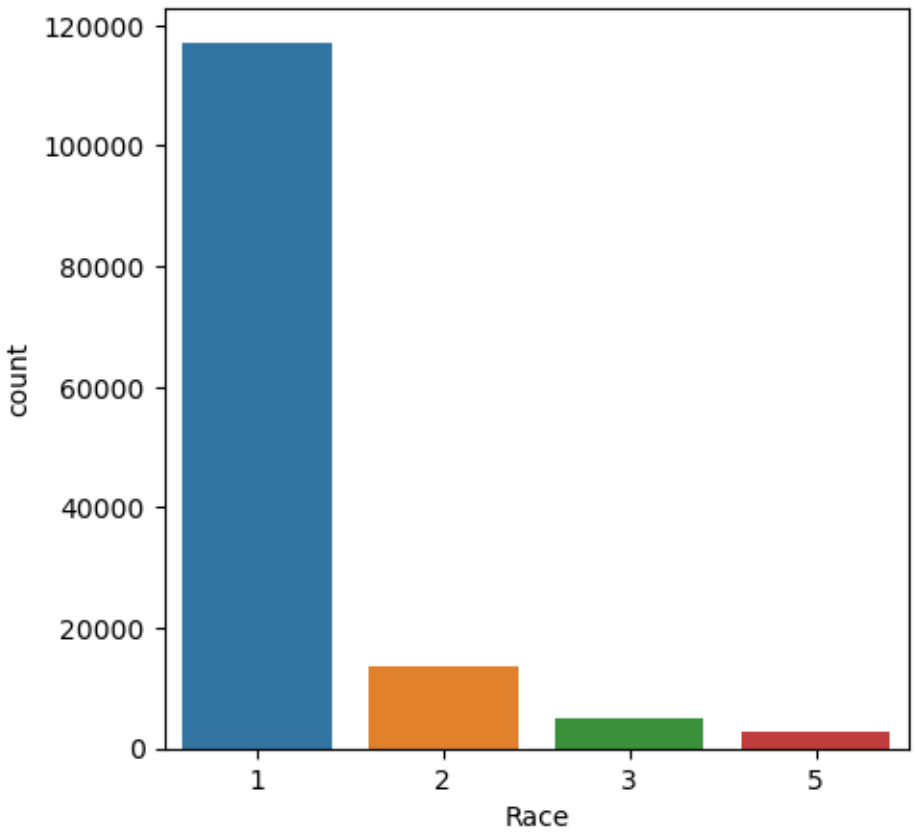
[7]:
```
# Gender
plt.rcParams["figure.figsize"] = [4, 5]
sns.countplot(x = beneficiary_data_df['Gender'])
```

[7]: <Axes: xlabel='Gender', ylabel='count'>



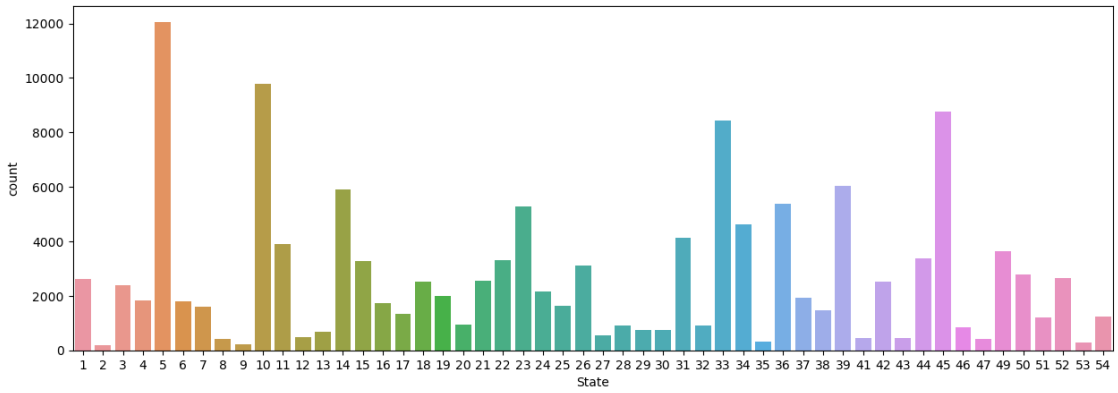
[8]:
```
#boolean values for race
plt.rcParams["figure.figsize"] = [5, 5]
sns.countplot(x = beneficiary_data_df['Race'])
```

[8]: <Axes: xlabel='Race', ylabel='count'>

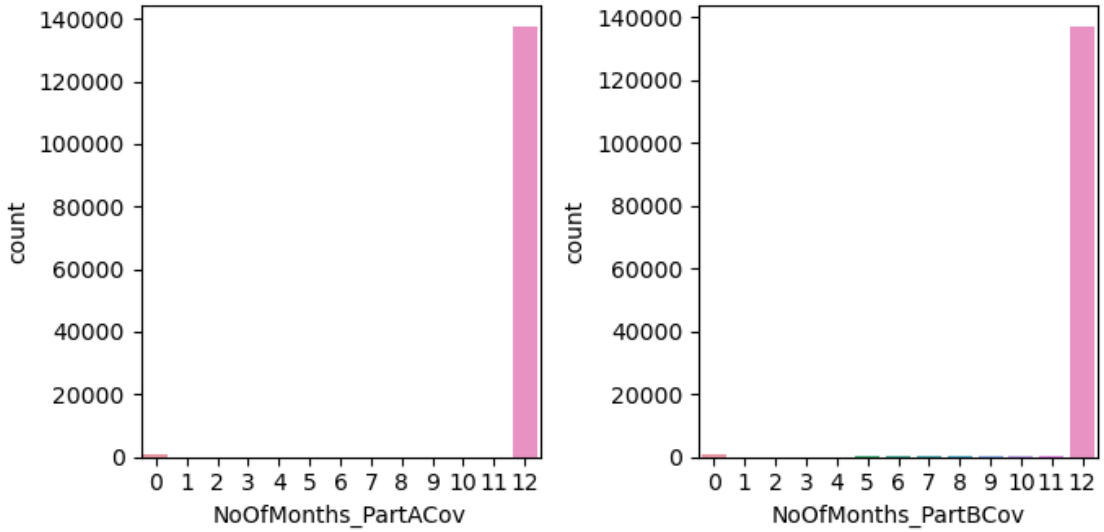


[9]:
```
# Plot the distribution across states
plt.rcParams["figure.figsize"] = [15, 5]
sns.countplot(x = beneficiary_data_df['State'])
```

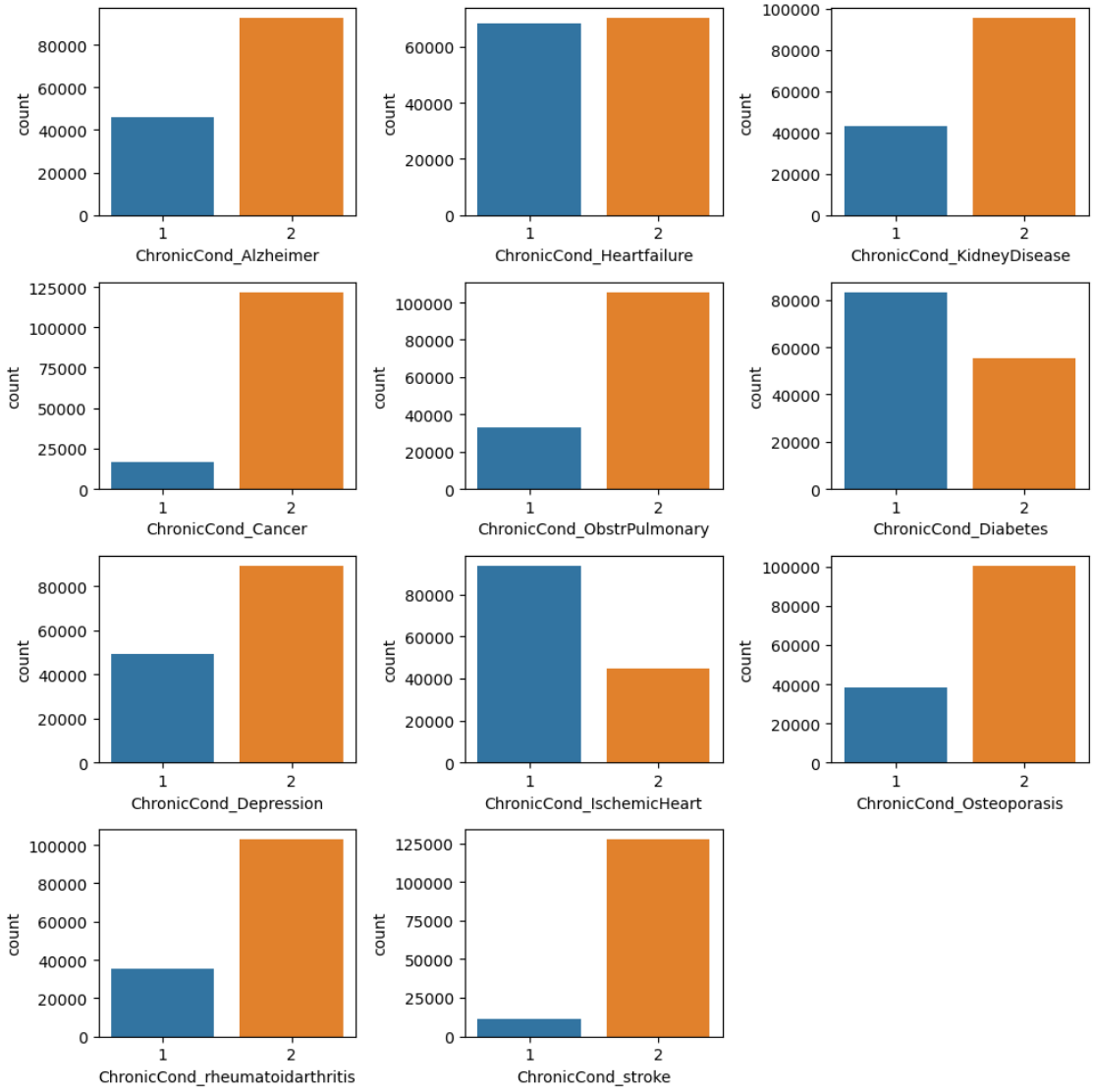
[9]: <Axes: xlabel='State', ylabel='count'>



[10]:
```
# Plotting the distribution of Part-A and Part-B coverage
plt.rcParams["figure.figsize"] = [7.00, 3.50]
plt.rcParams["figure.autolayout"] = True
fig, axs = plt.subplots(1, 2)
sns.countplot(x= beneficiary_data_df['NoOfMonths_PartACov'] ,ax=axs[0])
sns.countplot(x= beneficiary_data_df['NoOfMonths_PartBCov'] ,ax=axs[1])
plt.show()
```



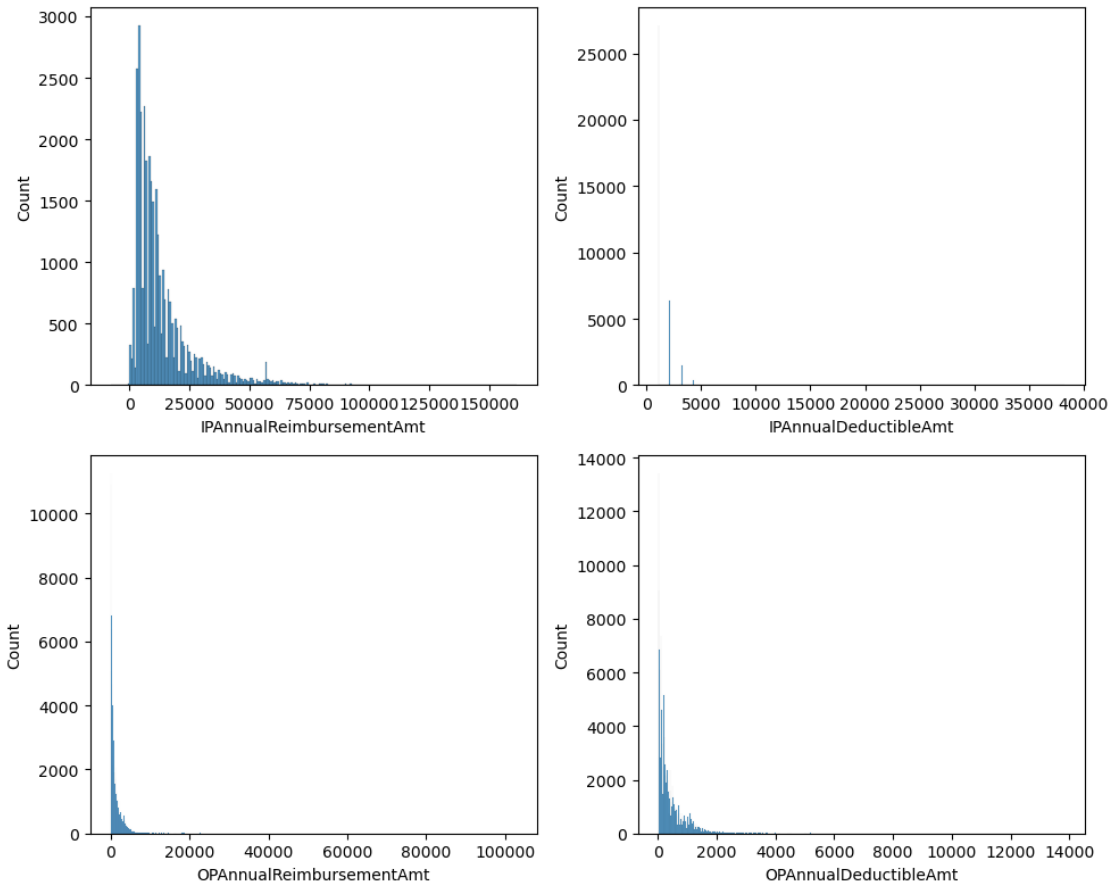
[11]:
```
# Plotting the distribution of Chronic Conditions
plt.rcParams["figure.figsize"] = [10,10]
plt.rcParams["figure.autolayout"] = True
fig, ax = plt.subplots(4, 3)
sns.countplot(x = beneficiary_data_df['ChronicCond_Alzheimer'], ax=ax[0,0])
sns.countplot(x = beneficiary_data_df['ChronicCond_Heartfailure'], ax=ax[0,1])
sns.countplot(x = beneficiary_data_df['ChronicCond_KidneyDisease'], ax=ax[0,2])
sns.countplot(x = beneficiary_data_df['ChronicCond_Cancer'], ax=ax[1,0])
sns.countplot(x = beneficiary_data_df['ChronicCond_ObstrPulmonary'], ax=ax[1,1])
sns.countplot(x = beneficiary_data_df['ChronicCond_Diabetes'], ax=ax[1,2])
sns.countplot(x = beneficiary_data_df['ChronicCond_Depression'], ax=ax[2,0])
sns.countplot(x = beneficiary_data_df['ChronicCond_IschemicHeart'], ax=ax[2,1])
sns.countplot(x = beneficiary_data_df['ChronicCond_Osteoporasis'], ax=ax[2,2])
sns.countplot(x = beneficiary_data_df['ChronicCond_rheumatoidarthritis'],␣
 ↪ax=ax[3,0])
sns.countplot(x = beneficiary_data_df['ChronicCond_stroke'], ax=ax[3,1])
fig.delaxes(ax[3,2])
plt.show()
```



[12]:
```
# Exploring quantitative variebles in the beneficiary_data_df data frame
plt.rcParams["figure.figsize"] = [10,8]
plt.rcParams["figure.autolayout"] = True
fig, ax = plt.subplots(2, 2)
sns.histplot(beneficiary_data_df.
 ↪loc[beneficiary_data_df['IPAnnualReimbursementAmt']!=0,␣
 ↪'IPAnnualReimbursementAmt'], kde =False, ax=ax[0,0])
sns.histplot(beneficiary_data_df.
 ↪loc[beneficiary_data_df['IPAnnualDeductibleAmt']!=0,␣
 ↪'IPAnnualDeductibleAmt'], kde =False, ax=ax[0,1])
```

```
sns.histplot(beneficiary_data_df.
  ↪loc[beneficiary_data_df['OPAnnualReimbursementAmt']!=0,␣
  ↪'OPAnnualReimbursementAmt'], kde =False, ax=ax[1,0])
sns.histplot(beneficiary_data_df.
  ↪loc[beneficiary_data_df['OPAnnualDeductibleAmt']!=0,␣
  ↪'OPAnnualDeductibleAmt'], kde =False, ax=ax[1,1])
plt.show
```

[12]: <function matplotlib.pyplot.show(close=None, block=None)>



[13]:
```
print(beneficiary_data_df[['IPAnnualReimbursementAmt', 'IPAnnualDeductibleAmt',␣
  ↪'OPAnnualReimbursementAmt', 'OPAnnualDeductibleAmt']].describe())
```

```
       IPAnnualReimbursementAmt  IPAnnualDeductibleAmt  \
count                138556.000000          138556.000000
mean                   3660.346502             399.847296
std                    9568.621827             956.175202
min                   -8000.000000               0.000000
25%                       0.000000               0.000000
```

```
50%                    0.000000              0.000000
75%                 2280.000000           1068.000000
max               161470.000000          38272.000000

       OPAnnualReimbursementAmt  OPAnnualDeductibleAmt
count             138556.000000          138556.000000
mean                1298.219348             377.718258
std                 2493.901134             645.530187
min                  -70.000000               0.000000
25%                  170.000000              40.000000
50%                  570.000000             170.000000
75%                 1500.000000             460.000000
max               102960.000000           13840.000000
```

[14]:
```
print(inpatient_data_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40474 entries, 0 to 40473
Data columns (total 30 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   BeneID                  40474 non-null  object
 1   ClaimID                 40474 non-null  object
 2   ClaimStartDt            40474 non-null  object
 3   ClaimEndDt              40474 non-null  object
 4   Provider                40474 non-null  object
 5   InscClaimAmtReimbursed  40474 non-null  int64
 6   AttendingPhysician      40362 non-null  object
 7   OperatingPhysician      23830 non-null  object
 8   OtherPhysician          4690 non-null   object
 9   AdmissionDt             40474 non-null  object
 10  ClmAdmitDiagnosisCode   40474 non-null  object
 11  DeductibleAmtPaid       39575 non-null  float64
 12  DischargeDt             40474 non-null  object
 13  DiagnosisGroupCode      40474 non-null  object
 14  ClmDiagnosisCode_1      40474 non-null  object
 15  ClmDiagnosisCode_2      40248 non-null  object
 16  ClmDiagnosisCode_3      39798 non-null  object
 17  ClmDiagnosisCode_4      38940 non-null  object
 18  ClmDiagnosisCode_5      37580 non-null  object
 19  ClmDiagnosisCode_6      35636 non-null  object
 20  ClmDiagnosisCode_7      33216 non-null  object
 21  ClmDiagnosisCode_8      30532 non-null  object
 22  ClmDiagnosisCode_9      26977 non-null  object
 23  ClmDiagnosisCode_10     3927 non-null   object
 24  ClmProcedureCode_1      23148 non-null  float64
 25  ClmProcedureCode_2      5454 non-null   float64
 26  ClmProcedureCode_3      965 non-null    float64
```

```
 27  ClmProcedureCode_4      116 non-null    float64
 28  ClmProcedureCode_5      9 non-null      float64
 29  ClmProcedureCode_6      0 non-null      float64
dtypes: float64(7), int64(1), object(22)
memory usage: 9.3+ MB
None
```

[15]:
```
inpatient_data_df.head()
```

[15]:

	BeneID	ClaimID	ClaimStartDt	ClaimEndDt	Provider	InscClaimAmtReimbursed	AttendingPhysician	OperatingPhysician	OtherPhysician	AdmissionDt	…	ClmDiagnosisCode_7	ClmDiagnosisCode_8	ClmDiagnosisCode_9	ClmDiagnosisCode_10	ClmProcedureCode_1	ClmProcedureCode_2	ClmProcedureCode_3	ClmProcedureCode_4	ClmProcedureCode_5	ClmProcedureCode_6
0	BENE11001	CLM46614	2009-04-12	2009-04-18	PRV55912	26000	PHY390922	NaN	NaN	2009-04-12	…	2724	19889	5849	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	BENE11001	CLM66048	2009-08-31	2009-09-02	PRV55907	5000	PHY318495	PHY318495	NaN	2009-08-31	…	NaN	NaN	NaN	NaN	7092.0	NaN	NaN	NaN	NaN	NaN
2	BENE11001	CLM68358	2009-09-17	2009-09-20	PRV56046	5000	PHY372395	NaN	PHY324689	2009-09-17	…	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	BENE11011	CLM38412	2009-02-14	2009-02-22	PRV52405	5000	PHY369659	PHY392961	PHY349768	2009-02-14	…	25062	40390	4019	NaN	331.0	NaN	NaN	NaN	NaN	NaN
4	BENE11014	CLM63689	2009-08-13	2009-08-30	PRV56614	10000	PHY379376	PHY398258	NaN	2009-08-13	…	5119	29620	20300	NaN	3893.0	NaN	NaN	NaN	NaN	NaN

```
2                NaN
3                NaN
4                NaN

[5 rows x 30 columns]
```

[16]:
```
print(outpatient_data_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 517737 entries, 0 to 517736
Data columns (total 27 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   BeneID                  517737 non-null  object
 1   ClaimID                 517737 non-null  object
 2   ClaimStartDt            517737 non-null  object
 3   ClaimEndDt              517737 non-null  object
 4   Provider                517737 non-null  object
 5   InscClaimAmtReimbursed  517737 non-null  int64
 6   AttendingPhysician      516341 non-null  object
 7   OperatingPhysician      90617 non-null   object
 8   OtherPhysician          195046 non-null  object
 9   ClmDiagnosisCode_1      507284 non-null  object
 10  ClmDiagnosisCode_2      322357 non-null  object
 11  ClmDiagnosisCode_3      203257 non-null  object
 12  ClmDiagnosisCode_4      125596 non-null  object
 13  ClmDiagnosisCode_5      74344 non-null   object
 14  ClmDiagnosisCode_6      48756 non-null   object
 15  ClmDiagnosisCode_7      32961 non-null   object
 16  ClmDiagnosisCode_8      22912 non-null   object
 17  ClmDiagnosisCode_9      14838 non-null   object
 18  ClmDiagnosisCode_10     1083 non-null    object
 19  ClmProcedureCode_1      162 non-null     float64
 20  ClmProcedureCode_2      36 non-null      float64
 21  ClmProcedureCode_3      4 non-null       float64
 22  ClmProcedureCode_4      2 non-null       float64
 23  ClmProcedureCode_5      0 non-null       float64
 24  ClmProcedureCode_6      0 non-null       float64
 25  DeductibleAmtPaid       517737 non-null  int64
 26  ClmAdmitDiagnosisCode   105425 non-null  object
dtypes: float64(6), int64(2), object(19)
memory usage: 106.7+ MB
None
```

[17]:
```
outpatient_data_df.head()
```

[17]:
```
      BeneID    ClaimID ClaimStartDt  ClaimEndDt  Provider  \
0  BENE11002  CLM624349   2009-10-11  2009-10-11  PRV56011
1  BENE11003  CLM189947   2009-02-12  2009-02-12  PRV57610
2  BENE11003  CLM438021   2009-06-27  2009-06-27  PRV57595
3  BENE11004  CLM121801   2009-01-06  2009-01-06  PRV56011
4  BENE11004  CLM150998   2009-01-22  2009-01-22  PRV56011

   InscClaimAmtReimbursed AttendingPhysician OperatingPhysician  \
0                      30          PHY326117                NaN
1                      80          PHY362868                NaN
2                      10          PHY328821                NaN
3                      40          PHY334319                NaN
4                     200          PHY403831                NaN

  OtherPhysician ClmDiagnosisCode_1  … ClmDiagnosisCode_9  \
0            NaN              78943  …                NaN
1            NaN               6115  …                NaN
2            NaN               2723  …                NaN
3            NaN              71988  …                NaN
4            NaN              82382  …                NaN

  ClmDiagnosisCode_10 ClmProcedureCode_1 ClmProcedureCode_2  \
0                 NaN                NaN                NaN
1                 NaN                NaN                NaN
2                 NaN                NaN                NaN
3                 NaN                NaN                NaN
4                 NaN                NaN                NaN

  ClmProcedureCode_3 ClmProcedureCode_4 ClmProcedureCode_5 ClmProcedureCode_6  \
0                NaN                NaN                NaN                NaN
1                NaN                NaN                NaN                NaN
2                NaN                NaN                NaN                NaN
3                NaN                NaN                NaN                NaN
4                NaN                NaN                NaN                NaN

  DeductibleAmtPaid  ClmAdmitDiagnosisCode
0                 0                 56409
1                 0                 79380
2                 0                   NaN
3                 0                   NaN
4                 0                 71947

[5 rows x 27 columns]
```

[18]:
```
# Dianosis and Procedure codes for outpatients
```

```
outpatient_data_df['ClmProcedureCodes'] = outpatient_data_df[[col for col in␣
 ↪outpatient_data_df.columns if col.startswith('ClmProcedureCode_')]].
 ↪apply(lambda row: list(row.dropna()), axis=1)
outpatient_data_df['ClmDiagnosisCodes'] = outpatient_data_df[[col for col in␣
 ↪outpatient_data_df.columns if col.startswith('ClmDiagnosisCode_')]].
 ↪apply(lambda row: list(row.dropna()), axis=1)
```

[19]:
```
!pip install wordcloud
```

```
Requirement already satisfied: wordcloud in
/Users/raja/anaconda3/lib/python3.11/site-packages (1.9.3)
Requirement already satisfied: numpy>=1.6.1 in
/Users/raja/anaconda3/lib/python3.11/site-packages (from wordcloud) (1.24.3)
Requirement already satisfied: pillow in
/Users/raja/anaconda3/lib/python3.11/site-packages (from wordcloud) (10.2.0)
Requirement already satisfied: matplotlib in
/Users/raja/anaconda3/lib/python3.11/site-packages (from wordcloud) (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in
/Users/raja/anaconda3/lib/python3.11/site-packages (from matplotlib->wordcloud)
(1.0.5)
Requirement already satisfied: cycler>=0.10 in
/Users/raja/anaconda3/lib/python3.11/site-packages (from matplotlib->wordcloud)
(0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in
/Users/raja/anaconda3/lib/python3.11/site-packages (from matplotlib->wordcloud)
(4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/Users/raja/anaconda3/lib/python3.11/site-packages (from matplotlib->wordcloud)
(1.4.4)
Requirement already satisfied: packaging>=20.0 in
/Users/raja/anaconda3/lib/python3.11/site-packages (from matplotlib->wordcloud)
(23.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/Users/raja/anaconda3/lib/python3.11/site-packages (from matplotlib->wordcloud)
(3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in
/Users/raja/anaconda3/lib/python3.11/site-packages (from matplotlib->wordcloud)
(2.8.2)
Requirement already satisfied: six>=1.5 in
/Users/raja/anaconda3/lib/python3.11/site-packages (from python-
dateutil>=2.7->matplotlib->wordcloud) (1.16.0)
```

[20]:
```
# Lets ploit a word cloud for the diagnosis codes for outpatient
from wordcloud import WordCloud
from collections import Counter
DiagnosisCounts_out = Counter(outpatient_data_df['ClmDiagnosisCodes'].explode().
 ↪dropna().to_list())
```

```
wordcloud = WordCloud(width = 800, height = 800).
  ↪generate_from_frequencies(DiagnosisCounts_out)
plt.figure(figsize=(6,6))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```

[21]:
```
# Lets ploit a word cloud for the procedures codes for outpatients
from wordcloud import WordCloud
from collections import Counter
ProcedureCounts_out = Counter([str(int(f)) for f in␣
  ↪outpatient_data_df['ClmProcedureCodes'].explode().dropna().to_list()])
```

```
wordcloud = WordCloud(width = 800, height = 800).
  ↪generate_from_frequencies(ProcedureCounts_out)
plt.figure(figsize=(6,6))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```

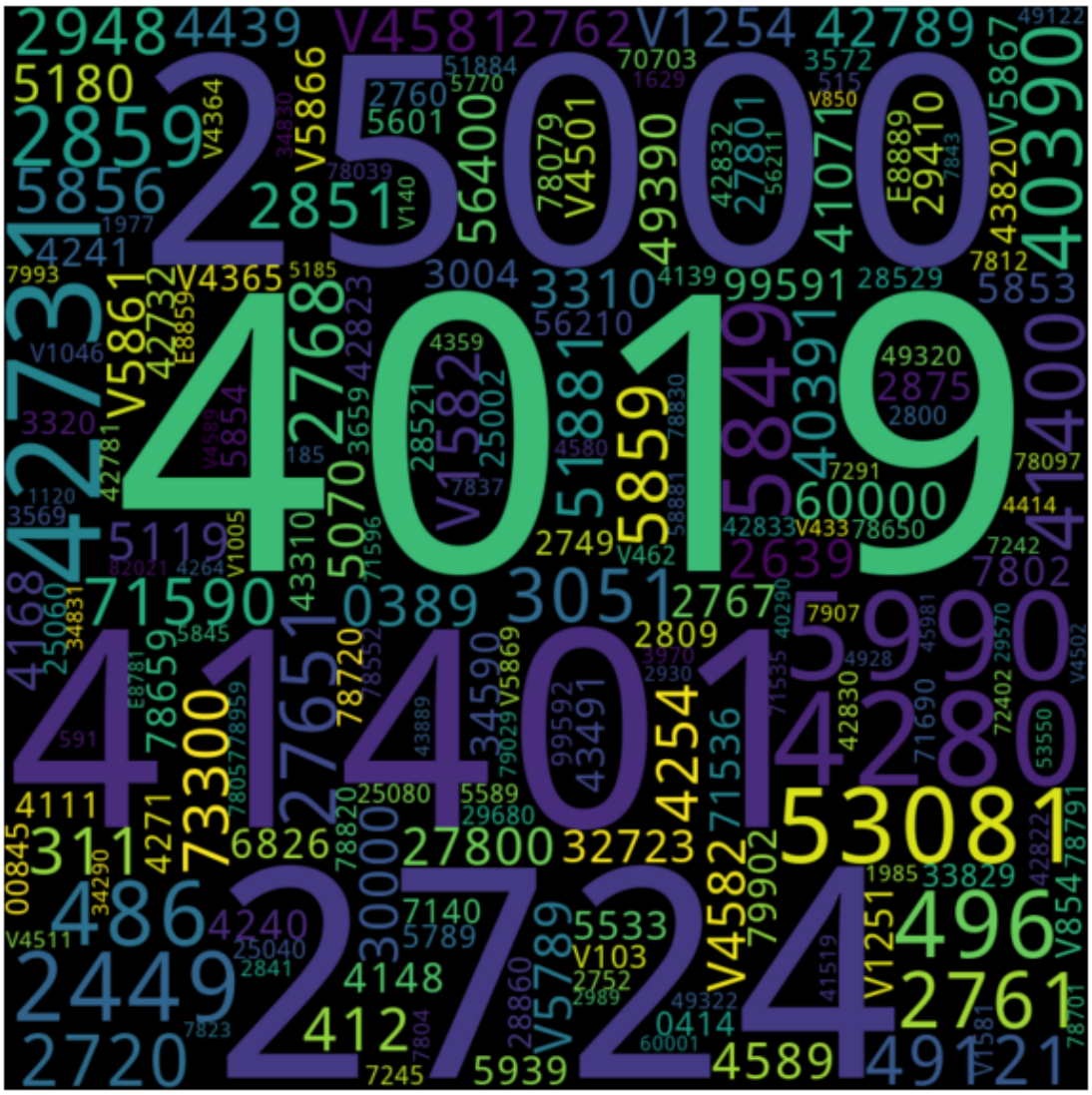
[22]:
```
# Lets plot a word cloud for the diagnosis codes for outpatients
from wordcloud import WordCloud
from collections import Counter
AdmitDiagnosisCounts_out = Counter(outpatient_data_df['ClmAdmitDiagnosisCode'].
  ↪dropna().to_list())
```

```
wordcloud = WordCloud(width = 800, height = 800).
  ↪generate_from_frequencies(AdmitDiagnosisCounts_out)
plt.figure(figsize=(10,10))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```

[23]:
```
# Let us look at all the Dianosis and Procedure codes for inpateints
inpatient_data_df['ClmProcedureCodes'] = inpatient_data_df[[col for col in␣
  ↪inpatient_data_df.columns if col.startswith('ClmProcedureCode_')]].
  ↪apply(lambda row: list(row.dropna()), axis=1)
```

```
inpatient_data_df['ClmDiagnosisCodes'] = inpatient_data_df[[col for col in␣
 ↪inpatient_data_df.columns if col.startswith('ClmDiagnosisCode_')]].
 ↪apply(lambda row: list(row.dropna()), axis=1)
```

[24]:
```
# word cloud for the diagnosis codes for inpatients
from wordcloud import WordCloud
from collections import Counter
DiagnosisCounts_in = Counter(inpatient_data_df['ClmDiagnosisCodes'].explode().
 ↪dropna().to_list())
wordcloud = WordCloud(width = 800, height = 800).
 ↪generate_from_frequencies(DiagnosisCounts_in)
plt.figure(figsize=(6,6))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```



[25]:
```
# word cloud for the procedures codes for inpatients
from wordcloud import WordCloud
from collections import Counter
ProcedureCounts_in = Counter([str(int(f)) for f in␣
 ↪inpatient_data_df['ClmProcedureCodes'].explode().dropna().to_list()])
wordcloud = WordCloud(width = 800, height = 800).
 ↪generate_from_frequencies(ProcedureCounts_in)
plt.figure(figsize=(6,6))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```

[26]:
```
# Lets plot a word cloud for the diagnosis codes for inpatients
from wordcloud import WordCloud
from collections import Counter
AdmitDiagnosisCounts_in = Counter(inpatient_data_df['ClmAdmitDiagnosisCode'].
 ↪dropna().to_list())
wordcloud = WordCloud(width = 800, height = 800).
 ↪generate_from_frequencies(AdmitDiagnosisCounts_in)
plt.figure(figsize=(10,10))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```

[27]:
```
print(inpatient_data_df[['InscClaimAmtReimbursed', 'DeductibleAmtPaid']].
  ↪describe())
```

```
       InscClaimAmtReimbursed  DeductibleAmtPaid
count            40474.000000            39575.0
mean             10087.884074             1068.0
std              10303.099402                0.0
min                  0.000000             1068.0
25%               4000.000000             1068.0
50%               7000.000000             1068.0
75%              12000.000000             1068.0
max             125000.000000             1068.0
```

[28]:
```
print(outpatient_data_df[['InscClaimAmtReimbursed', 'DeductibleAmtPaid']].
  ↪describe())
```

```
       InscClaimAmtReimbursed  DeductibleAmtPaid
count           517737.000000      517737.000000
mean               286.334799           2.779233
std                694.034343          15.785839
min                  0.000000           0.000000
25%                 40.000000           0.000000
50%                 80.000000           0.000000
75%                200.000000           0.000000
max             102500.000000         897.000000
```

2 Data Pre-Processing & Feature Engeenering

[29]:
```
# Here 0 indicates male and 1 as female
beneficiary_data_df['Gender'] = beneficiary_data_df['Gender'].map({1: 0, 2: 1})
```

[30]:
```
# Race category variable
beneficiary_data_df['Race'] = beneficiary_data_df['Race'].astype("category")
```

[31]:
```
# Convert the date of birth and date of death to datetime format
import datetime
beneficiary_data_df['DOB'] = pd.to_datetime(beneficiary_data_df['DOB'],␣
  ↪format='%Y-%m-%d')
beneficiary_data_df['DOD'] = pd.to_datetime(beneficiary_data_df['DOD'],␣
  ↪format='%Y-%m-%d')
```

[32]:
```
# creating a new boolean variable coulumn 'dead' and assigning it a particular␣
  ↪value if it is non-empty
beneficiary_data_df['dead'] =  ~ pd.isnull(beneficiary_data_df['DOD'])
beneficiary_data_df[['dead', 'DOD']].head()
```

[32]:
```
    dead DOD
0  False NaT
1  False NaT
2  False NaT
3  False NaT
4  False NaT
```

[33]:
```
from datetime import datetime
beneficiary_data_df['age']= beneficiary_data_df['DOB'].map(lambda x:␣
 ↪(int((datetime.now()-x).days/365.25))-5)
```

[34]:
```
beneficiary_data_df[['age']].head()
```

[34]:
```
   age
0   76
1   82
2   82
3   96
4   83
```

[35]:
```
# Since state and county are categorical values
beneficiary_data_df['State'] = beneficiary_data_df['State'].astype('category')
beneficiary_data_df['County'] = beneficiary_data_df['County'].astype('category')
```

[36]:
```
# Convert Renal Disease Indicator to Boolean
beneficiary_data_df['RenalDiseaseIndicator'] =␣
 ↪beneficiary_data_df['RenalDiseaseIndicator'].map({'0': 0, 'Y': 1})

#convert the categorical variables to boolean variable such that if covered for␣
 ↪12 months then 1 else 0
beneficiary_data_df['12Months_PartACov'] =␣
 ↪beneficiary_data_df['NoOfMonths_PartACov'].apply(lambda x: 1 if x == 12 else␣
 ↪0)
beneficiary_data_df['12Months_PartBCov'] =␣
 ↪beneficiary_data_df['NoOfMonths_PartBCov'].apply(lambda x: 1 if x == 12 else␣
 ↪0)

#Boolean conversion for ChronicCond
for col in beneficiary_data_df.columns:
    if col.startswith('ChronicCond'):
        beneficiary_data_df[col] = beneficiary_data_df[col].map({1: 1, 2: 0})
```

[37]:
```
# Days spent in the hostpital
inpatient_data_df['DischargeDt'] = pd.
 ↪to_datetime(inpatient_data_df['DischargeDt'], format='%Y-%m-%d')
inpatient_data_df['AdmissionDt'] = pd.
 ↪to_datetime(inpatient_data_df['AdmissionDt'], format='%Y-%m-%d')
```

```
inpatient_data_df['days_in_hospital'] = (inpatient_data_df['DischargeDt'] -␣
 ↪inpatient_data_df['AdmissionDt']).dt.days
```

[38]:
```
# Time for the claim process
inpatient_data_df['ClaimProcessTime'] = (pd.
 ↪to_datetime(inpatient_data_df['ClaimEndDt'], format='%Y-%m-%d') - pd.
 ↪to_datetime(inpatient_data_df['ClaimStartDt'], format='%Y-%m-%d')).dt.days
outpatient_data_df['ClaimProcessTime'] = (pd.
 ↪to_datetime(outpatient_data_df['ClaimEndDt'], format='%Y-%m-%d') - pd.
 ↪to_datetime(outpatient_data_df['ClaimStartDt'], format='%Y-%m-%d')).dt.days
```

[39]:
```
# Converting Physicians into boolean values
outpatient_data_df['AttPhy?'] =  ~ pd.
 ↪isnull(outpatient_data_df['AttendingPhysician'])
outpatient_data_df['OpPhy?'] =  ~ pd.
 ↪isnull(outpatient_data_df['OperatingPhysician'])
outpatient_data_df['OthPhy?'] =  ~ pd.
 ↪isnull(outpatient_data_df['OtherPhysician'])
inpatient_data_df['AttPhy?'] =  ~ pd.
 ↪isnull(inpatient_data_df['AttendingPhysician'])
inpatient_data_df['OpPhy?'] =  ~ pd.
 ↪isnull(inpatient_data_df['OperatingPhysician'])
inpatient_data_df['OthPhy?'] =  ~ pd.isnull(inpatient_data_df['OtherPhysician'])
```

[40]:
```
# top 100 admit codes for outpatients and converting them to a dataframe
AdmitDiagnosisCounts_out_top = dict(sorted(AdmitDiagnosisCounts_out.items(),␣
 ↪key=lambda x:x[1], reverse=True)[:100])
outpatient_AdmitDiagCodes_df = pd.DataFrame(np.zeros((outpatient_data_df.
 ↪shape[0], 100)), columns = AdmitDiagnosisCounts_out_top.keys())
for col in outpatient_AdmitDiagCodes_df.columns:
    outpatient_AdmitDiagCodes_df.loc[:,col] =␣
 ↪outpatient_data_df['ClmAdmitDiagnosisCode'].apply(lambda x: 1 if str(col)␣
 ↪==x else 0)
```

```
/var/folders/bh/46kz97xn2d7cj3l5gp7l32nc0000gn/T/ipykernel_3142/4293655691.py:5:
DeprecationWarning: In a future version, `df.iloc[:, i] = newvals` will attempt
to set the values inplace instead of always setting a new array. To retain the
old behavior, use either `df[df.columns[i]] = newvals` or, if columns are non-
unique, `df.isetitem(i, newvals)`
  outpatient_AdmitDiagCodes_df.loc[:,col] =
outpatient_data_df['ClmAdmitDiagnosisCode'].apply(lambda x: 1 if str(col) ==x
else 0)
```

[41]:
```
# Gettting the top 100 admit codes for inpatients and converting them to a␣
 ↪dataframe
AdmitDiagnosisCounts_in_top = dict(sorted(AdmitDiagnosisCounts_in.items(),␣
 ↪key=lambda x:x[1], reverse=True)[:100])
```

```
inpatient_AdmitDiagCodes_df = pd.DataFrame(np.zeros((inpatient_data_df.
 ↪shape[0], 100)), columns = AdmitDiagnosisCounts_in_top.keys())
for col in inpatient_AdmitDiagCodes_df.columns:
    inpatient_AdmitDiagCodes_df.loc[:,col] =␣
 ↪inpatient_data_df['ClmAdmitDiagnosisCode'].apply(lambda x: 1 if str(col) ==x␣
 ↪else 0)
```

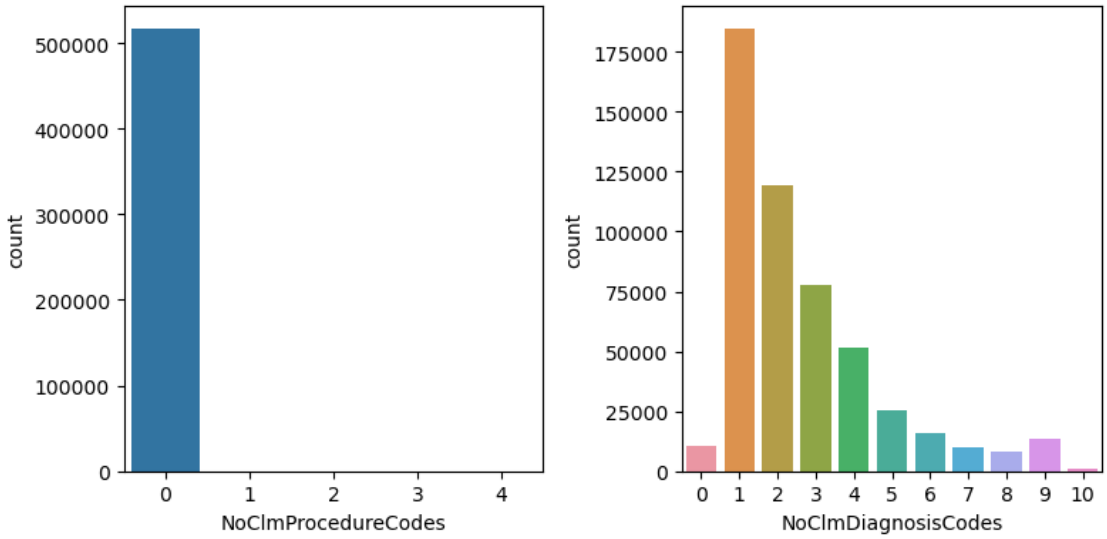
```
/var/folders/bh/46kz97xn2d7cj3l5gp7l32nc0000gn/T/ipykernel_3142/1386464342.py:5:
DeprecationWarning: In a future version, `df.iloc[:, i] = newvals` will attempt
to set the values inplace instead of always setting a new array. To retain the
old behavior, use either `df[df.columns[i]] = newvals` or, if columns are non-
unique, `df.isetitem(i, newvals)`
  inpatient_AdmitDiagCodes_df.loc[:,col] =
inpatient_data_df['ClmAdmitDiagnosisCode'].apply(lambda x: 1 if str(col) ==x
else 0)
```

[42]:
```
# Gettting the top 100 diagnosis codes for outpatients and converting them to a␣
 ↪dataframe
DiagnosisCounts_out_top = dict(sorted(DiagnosisCounts_out.items(), key=lambda x:
 ↪x[1], reverse=True)[:100])
outpatient_DiagCodes_df = pd.DataFrame(np.zeros((outpatient_data_df.shape[0],␣
 ↪100)), columns = DiagnosisCounts_out_top.keys())
for col in outpatient_DiagCodes_df.columns:
    outpatient_DiagCodes_df.loc[:,col] =␣
 ↪outpatient_data_df['ClmDiagnosisCodes'].apply(lambda x: 1 if str(col) in x␣
 ↪else 0)

# Gettting the top 100 procedure codes for outpatients and converting them to a␣
 ↪dataframe

ProcedureCounts_out_top = dict(sorted(ProcedureCounts_out.items(), key=lambda x:
 ↪x[1], reverse=True)[:100])
outpatient_ProcCodes_df = pd.DataFrame(np.zeros((outpatient_data_df.shape[0],␣
 ↪100)), columns = ProcedureCounts_out_top.keys())
for col in outpatient_ProcCodes_df.columns:
    outpatient_ProcCodes_df.loc[:,col] =␣
 ↪outpatient_data_df['ClmProcedureCodes'].apply(lambda x: 1 if str(col) in x␣
 ↪else 0)

# Gettting the top 100 diagnosis codes for inpatients and converting them to a␣
 ↪dataframe
DiagnosisCounts_in_top = dict(sorted(DiagnosisCounts_in.items(), key=lambda x:
 ↪x[1], reverse=True)[:100])
inpatient_DiagCodes_df = pd.DataFrame(np.zeros((inpatient_data_df.shape[0],␣
 ↪100)), columns = DiagnosisCounts_in_top.keys())
for col in inpatient_DiagCodes_df.columns:
```

```
    inpatient_DiagCodes_df.loc[:,col] = inpatient_data_df['ClmDiagnosisCodes'].
 ↪apply(lambda x: 1 if str(col) in x else 0)

# Gettting the top 100 procedure codes for inpatients and converting them to a␣
 ↪dataframe
ProcedureCounts_in_top = dict(sorted(ProcedureCounts_in.items(), key=lambda x:
 ↪x[1], reverse=True)[:100])
inpatient_ProcCodes_df = pd.DataFrame(np.zeros((inpatient_data_df.shape[0],␣
 ↪100)), columns = ProcedureCounts_in_top.keys())
for col in inpatient_ProcCodes_df.columns:
    inpatient_ProcCodes_df.loc[:,col] = inpatient_data_df['ClmProcedureCodes'].
 ↪apply(lambda x: 1 if str(col) in x else 0)
```

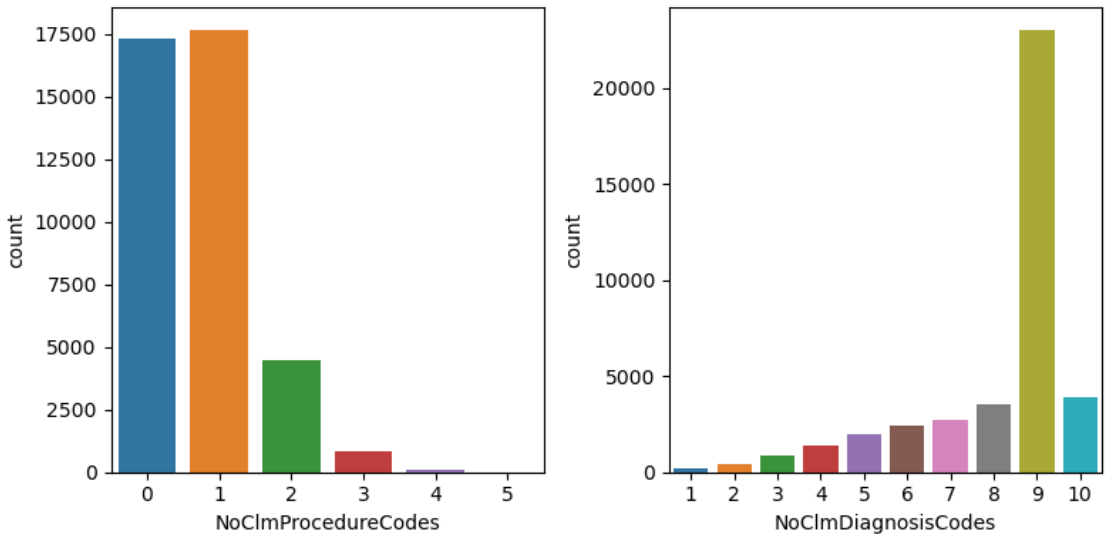
```
/var/folders/bh/46kz97xn2d7cj3l5gp7l32nc0000gn/T/ipykernel_3142/3423786791.py:5:
DeprecationWarning: In a future version, `df.iloc[:, i] = newvals` will attempt
to set the values inplace instead of always setting a new array. To retain the
old behavior, use either `df[df.columns[i]] = newvals` or, if columns are non-
unique, `df.isetitem(i, newvals)`
  outpatient_DiagCodes_df.loc[:,col] =
outpatient_data_df['ClmDiagnosisCodes'].apply(lambda x: 1 if str(col) in x else
0)
/var/folders/bh/46kz97xn2d7cj3l5gp7l32nc0000gn/T/ipykernel_3142/3423786791.py:12
: DeprecationWarning: In a future version, `df.iloc[:, i] = newvals` will
attempt to set the values inplace instead of always setting a new array. To
retain the old behavior, use either `df[df.columns[i]] = newvals` or, if columns
are non-unique, `df.isetitem(i, newvals)`
  outpatient_ProcCodes_df.loc[:,col] =
outpatient_data_df['ClmProcedureCodes'].apply(lambda x: 1 if str(col) in x else
0)
/var/folders/bh/46kz97xn2d7cj3l5gp7l32nc0000gn/T/ipykernel_3142/3423786791.py:18
: DeprecationWarning: In a future version, `df.iloc[:, i] = newvals` will
attempt to set the values inplace instead of always setting a new array. To
retain the old behavior, use either `df[df.columns[i]] = newvals` or, if columns
are non-unique, `df.isetitem(i, newvals)`
  inpatient_DiagCodes_df.loc[:,col] =
inpatient_data_df['ClmDiagnosisCodes'].apply(lambda x: 1 if str(col) in x else
0)
/var/folders/bh/46kz97xn2d7cj3l5gp7l32nc0000gn/T/ipykernel_3142/3423786791.py:24
: DeprecationWarning: In a future version, `df.iloc[:, i] = newvals` will
attempt to set the values inplace instead of always setting a new array. To
retain the old behavior, use either `df[df.columns[i]] = newvals` or, if columns
are non-unique, `df.isetitem(i, newvals)`
  inpatient_ProcCodes_df.loc[:,col] =
inpatient_data_df['ClmProcedureCodes'].apply(lambda x: 1 if str(col) in x else
0)
```

[43]:
```
# Let us look at all the Dianosis and Procedure codes for outpatients
outpatient_data_df['NoClmProcedureCodes'] = outpatient_data_df[[col for col in␣
 ↪outpatient_data_df.columns if col.startswith('ClmProcedureCode_')]].
 ↪count(axis=1)
outpatient_data_df['NoClmDiagnosisCodes'] = outpatient_data_df[[col for col in␣
 ↪outpatient_data_df.columns if col.startswith('ClmDiagnosisCode_')]].
 ↪count(axis=1)

# Plotting the distribution of No of Diagnosis and Procedure codes
plt.rcParams["figure.figsize"] = [8, 4]
plt.rcParams["figure.autolayout"] = True
fig, axs = plt.subplots(1, 2)
sns.countplot(x= outpatient_data_df['NoClmProcedureCodes'] ,ax=axs[0])
sns.countplot(x= outpatient_data_df['NoClmDiagnosisCodes'] ,ax=axs[1])
plt.show()
```



[44]:
```
# Let us look at all the Dianosis and Procedure codes for inpatients
inpatient_data_df['NoClmProcedureCodes'] = inpatient_data_df[[col for col in␣
 ↪inpatient_data_df.columns if col.startswith('ClmProcedureCode_')]].
 ↪count(axis=1)
inpatient_data_df['NoClmDiagnosisCodes'] = inpatient_data_df[[col for col in␣
 ↪inpatient_data_df.columns if col.startswith('ClmDiagnosisCode_')]].
 ↪count(axis=1)

# Plotting the distribution of No of Diagnosis and Procedure codes
plt.rcParams["figure.figsize"] = [8, 4]
plt.rcParams["figure.autolayout"] = True
fig, axs = plt.subplots(1, 2)
```

```
sns.countplot(x= inpatient_data_df['NoClmProcedureCodes'] ,ax=axs[0])
sns.countplot(x= inpatient_data_df['NoClmDiagnosisCodes'] ,ax=axs[1])
plt.show()
```



[45]:
```
outpatient_data_df['Admit?'] = ~pd.
 ↪isnull(outpatient_data_df['ClmAdmitDiagnosisCode'])  # Outpatient admitted ?

outpatient_data_df[['Admit?']].head()
```

[45]:

	Admit?
0	True
1	True
2	False
3	False
4	True

[46]:
```
ip_features_1= inpatient_data_df[['BeneID',␣
 ↪'ClaimID','Provider','ClaimStartDt', 'InscClaimAmtReimbursed',␣
 ↪'DeductibleAmtPaid', 'AttPhy?', 'OpPhy?', 'OthPhy?', 'days_in_hospital',␣
 ↪'ClaimProcessTime', 'NoClmProcedureCodes', 'NoClmDiagnosisCodes']].
 ↪groupby(['BeneID','Provider']).agg(
    {'ClaimID': 'count', 'InscClaimAmtReimbursed': 'sum', 'DeductibleAmtPaid':␣
 ↪'sum', 'days_in_hospital': 'sum', 'NoClmProcedureCodes': 'sum',␣
 ↪'NoClmDiagnosisCodes': 'sum', 'AttPhy?': 'sum', 'OpPhy?': 'sum', 'OthPhy?':␣
 ↪'sum', 'ClaimProcessTime': 'mean'}).reset_index()
```

```
ip_features_2 = pd.concat([inpatient_data_df[['BeneID','Provider']],␣
 ↪inpatient_DiagCodes_df, inpatient_ProcCodes_df,␣
 ↪inpatient_AdmitDiagCodes_df]).groupby(['BeneID','Provider']).agg('sum').
 ↪reset_index()

ip_features= ip_features_1.merge(ip_features_2, how='inner', on= ['BeneID',␣
 ↪'Provider'])

ip_bene_df = ip_features.join(beneficiary_data_df.fillna(0).
 ↪set_index('BeneID'), on='BeneID').reset_index()
```

[47]:
```
ip_bene_df.head()
```

[47]:

	index	BeneID	Provider	ClaimID	InscClaimAmtReimbursed	DeductibleAmtPaid	days_in_hospital	NoClmProcedureCodes	NoClmDiagnosisCodes	AttPhy?	…	ChronicCond_rheumatoidarthritis	ChronicCond_stroke	IPAnnualReimbursementAmt	IPAnnualDeductibleAmt	OPAnnualReimbursementAmt	OPAnnualDeductibleAmt	dead	age
0	0	BENE100002	PRV54894	1	12000	1068.0	25	2	9	1	…	0	1	12250	1068	1760	660	False	80
1	1	BENE100004	PRV52890	1	3000	1068.0	2	0	9	1	…	0	0	14270	2136	1880	700	False	75
2	2	BENE100006	PRV57317	1	17000	1068.0	6	2	9	1	…	0	0	17000	1068	160	20	False	67
3	3	BENE100007	PRV54875	1	4000	1068.0	4	0	9	1	…	0	0	4400	1068	200	140	False	80
4	4	BENE100010	PRV55916	2	12000	2136.0	18	1	18	2	…	1	0	13400	2136	1050	760	False	91

```
   12Months_PartACov  12Months_PartBCov
0                  1                  1
1                  1                  1
2                  1                  1
3                  1                  1
4                  1                  1

[5 rows x 294 columns]
```

[48]:
```
ip_bene_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36616 entries, 0 to 36615
Columns: 294 entries, index to 12Months_PartBCov
dtypes: bool(1), category(3), datetime64[ns](1), float64(255), int64(31),
object(3)
memory usage: 81.2+ MB
```

[49]:
```
ip_final_features1 = ip_bene_df.groupby('Provider').agg('mean').reset_index().
 ↪rename(columns  = {'ClaimID':"AvgClaims"})
ip_final_features2 = ip_bene_df[['Provider', 'BeneID', 'ClaimID']].
 ↪groupby('Provider').agg(TotalClaims =('ClaimID', 'sum'), NoBene =('ClaimID',␣
 ↪'count')).reset_index()
ip_final_features = ip_final_features1.merge(ip_final_features2, how='inner',␣
 ↪on= ['Provider'])

ip_final_features.head()
```

```
/var/folders/bh/46kz97xn2d7cj3l5gp7l32nc0000gn/T/ipykernel_3142/2669241409.py:1:
FutureWarning: The default value of numeric_only in DataFrameGroupBy.mean is
deprecated. In a future version, numeric_only will default to False. Either
specify numeric_only or select only columns which should be valid for the
function.
  ip_final_features1 =
ip_bene_df.groupby('Provider').agg('mean').reset_index().rename(columns  =
{'ClaimID':"AvgClaims"})
```

[49]:
```
    Provider         index  AvgClaims  InscClaimAmtReimbursed  \
0  PRV51001  18671.600000   1.000000            19400.000000
1  PRV51003  17060.773585   1.169811            10811.320755
2  PRV51007  29581.666667   1.000000             6333.333333
3  PRV51008   8283.000000   1.000000            12500.000000
4  PRV51011  21723.000000   1.000000             5000.000000

   DeductibleAmtPaid  days_in_hospital  NoClmProcedureCodes  \
0        1068.000000          5.000000             0.600000
1        1249.358491          6.037736             0.905660
```

```
2        1068.000000          5.333333             0.333333
3        1068.000000          4.000000             1.000000
4        1068.000000          5.000000             0.000000

   NoClmDiagnosisCodes    AttPhy?    OpPhy?  …  IPAnnualReimbursementAmt  \
0             7.200000   1.000000  0.400000  …                77902.00000
1             9.490566   1.169811  0.754717  …                11925.09434
2             7.333333   1.000000  0.333333  …                11710.00000
3             7.500000   1.000000  1.000000  …                18750.00000
4             8.000000   1.000000  0.000000  …                 5000.00000

   IPAnnualDeductibleAmt  OPAnnualReimbursementAmt  OPAnnualDeductibleAmt  \
0            2563.200000               1350.000000              236.00000
1            1473.433962               1751.698113              584.90566
2            2136.000000               2413.333333              470.00000
3            1602.000000                320.000000              165.00000
4            1068.000000                590.000000               90.00000

       dead         age  12Months_PartACov  12Months_PartBCov  TotalClaims  \
0  0.000000   87.200000           1.000000           1.000000            5
1  0.018868   77.584906           0.981132           0.981132           62
2  0.000000   87.666667           1.000000           1.000000            3
3  0.000000   60.500000           1.000000           1.000000            2
4  0.000000  107.000000           1.000000           1.000000            1

   NoBene
0       5
1      53
2       3
3       2
4       1

[5 rows x 290 columns]
```

[50]:
```
op_features_1= outpatient_data_df[['BeneID',␣
 ↪'ClaimID','Provider','ClaimStartDt', 'InscClaimAmtReimbursed',␣
 ↪'DeductibleAmtPaid', 'AttPhy?', 'OpPhy?', 'OthPhy?', 'Admit?',␣
 ↪'ClaimProcessTime', 'NoClmProcedureCodes', 'NoClmDiagnosisCodes']].
 ↪groupby(['BeneID','Provider']).agg(
    {'ClaimID': 'count', 'InscClaimAmtReimbursed': 'sum', 'DeductibleAmtPaid':␣
 ↪'sum', 'Admit?': 'sum', 'NoClmProcedureCodes': 'sum', 'NoClmDiagnosisCodes':␣
 ↪'sum', 'AttPhy?': 'sum', 'OpPhy?': 'sum', 'OthPhy?': 'sum',␣
 ↪'ClaimProcessTime': 'mean'}).reset_index()
op_features_2 = pd.concat([outpatient_data_df[['BeneID','Provider']],␣
 ↪outpatient_DiagCodes_df, outpatient_ProcCodes_df,␣
 ↪outpatient_AdmitDiagCodes_df]).groupby(['BeneID','Provider']).agg('sum').
 ↪reset_index()
```

```
op_features= op_features_1.merge(op_features_2, how='inner', on= ['BeneID',␣
 ↪'Provider'])

op_bene_df = op_features.join(beneficiary_data_df.fillna(0).
 ↪set_index('BeneID'), on='BeneID').reset_index()

op_final_features1 = op_bene_df.groupby('Provider').agg('mean').reset_index().
 ↪rename(columns  = {'ClaimID':"AvgClaims"})
op_final_features2 = op_bene_df[['Provider', 'BeneID', 'ClaimID']].
 ↪groupby('Provider').agg(TotalClaims =('ClaimID', 'sum'), NoBene =('ClaimID',␣
 ↪'count')).reset_index()
op_final_features = op_final_features1.merge(op_final_features2, how='inner',␣
 ↪on= ['Provider'])

op_final_features.head()
```

```
/var/folders/bh/46kz97xn2d7cj3l5gp7l32nc0000gn/T/ipykernel_3142/3862218470.py:9:
FutureWarning: The default value of numeric_only in DataFrameGroupBy.mean is
deprecated. In a future version, numeric_only will default to False. Either
specify numeric_only or select only columns which should be valid for the
function.
  op_final_features1 =
op_bene_df.groupby('Provider').agg('mean').reset_index().rename(columns  =
{'ClaimID':"AvgClaims"})
```

[50]:

	Provider	index	AvgClaims	InscClaimAmtReimbursed	\
0	PRV51001	157734.736842	1.052632	402.105263	
1	PRV51003	175224.000000	1.060606	495.000000	
2	PRV51004	164949.021739	1.079710	378.043478	
3	PRV51005	172328.208081	2.353535	567.494949	
4	PRV51007	162450.785714	1.232143	262.678571	

	DeductibleAmtPaid	Admit?	NoClmProcedureCodes	NoClmDiagnosisCodes	\
0	0.000000	0.315789	0.0	2.315789	
1	1.060606	0.242424	0.0	2.878788	
2	2.246377	0.202899	0.0	2.789855	
3	7.474747	0.509091	0.0	6.092929	
4	1.071429	0.250000	0.0	3.446429	

	AttPhy?	OpPhy?	…	IPAnnualReimbursementAmt	IPAnnualDeductibleAmt	\
0	1.052632	0.157895	…	2296.842105	449.684211	
1	1.060606	0.075758	…	2800.151515	339.818182	
2	1.079710	0.195652	…	4596.739130	454.144928	
3	2.349495	0.448485	…	3717.232323	398.698990	
4	1.232143	0.196429	…	2825.535714	362.357143	

```
   OPAnnualReimbursementAmt  OPAnnualDeductibleAmt      dead        age  \
0               2850.000000             537.789474  0.000000  87.578947
1               3045.757576             735.606061  0.000000  78.136364
2               2095.144928             600.869565  0.007246  81.724638
3               1798.808081             475.965657  0.006061  79.296970
4               1447.857143             436.785714  0.017857  76.821429

   12Months_PartACov  12Months_PartBCov  TotalClaims  NoBene
0           1.000000           1.000000           20      19
1           0.984848           0.984848           70      66
2           0.978261           0.992754          149     138
3           0.985859           0.985859         1165     495
4           0.982143           0.982143           69      56

[5 rows x 279 columns]
```

[51]:
```
for col in ip_final_features.columns[1:]:
    ip_final_features.rename(columns={col: 'IP_'+col}, inplace=True)

op_final_features.drop(columns ='index', inplace=True)
for col in op_final_features.columns[1:]:
    op_final_features.rename(columns={col: 'OP_'+col}, inplace=True)
```

[52]:
```
#### Merging datasets together
```

[53]:
```
final_features = op_final_features.merge(ip_final_features, how='left',␣
 ↪on="Provider")
final_features.fillna(0, inplace=True)

train_df = final_features.merge(train_labels_df, on = 'Provider')
```

[54]:
```
print(train_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5012 entries, 0 to 5011
Columns: 568 entries, Provider to PotentialFraud
dtypes: float64(564), int64(2), object(2)
memory usage: 21.8+ MB
None
```

[55]:
```
train_df.head()
```

[55]:
```
   Provider  OP_AvgClaims  OP_InscClaimAmtReimbursed  OP_DeductibleAmtPaid  \
0  PRV51001      1.052632                 402.105263              0.000000
1  PRV51003      1.060606                 495.000000              1.060606
2  PRV51004      1.079710                 378.043478              2.246377
3  PRV51005      2.353535                 567.494949              7.474747
4  PRV51007      1.232143                 262.678571              1.071429
```

```
   OP_Admit?  OP_NoClmProcedureCodes  OP_NoClmDiagnosisCodes  OP_AttPhy?  \
0   0.315789                     0.0                2.315789    1.052632
1   0.242424                     0.0                2.878788    1.060606
2   0.202899                     0.0                2.789855    1.079710
3   0.509091                     0.0                6.092929    2.349495
4   0.250000                     0.0                3.446429    1.232143

   OP_OpPhy?  OP_OthPhy?  …  IP_IPAnnualDeductibleAmt  \
0   0.157895    0.473684  …               2563.200000
1   0.075758    0.378788  …               1473.433962
2   0.195652    0.456522  …                  0.000000
3   0.448485    0.965657  …                  0.000000
4   0.196429    0.464286  …               2136.000000

   IP_OPAnnualReimbursementAmt  IP_OPAnnualDeductibleAmt   IP_dead     IP_age  \
0                  1350.000000                 236.00000  0.000000  87.200000
1                  1751.698113                 584.90566  0.018868  77.584906
2                     0.000000                   0.00000  0.000000   0.000000
3                     0.000000                   0.00000  0.000000   0.000000
4                  2413.333333                 470.00000  0.000000  87.666667

   IP_12Months_PartACov  IP_12Months_PartBCov  IP_TotalClaims  IP_NoBene  \
0              1.000000              1.000000             5.0        5.0
1              0.981132              0.981132            62.0       53.0
2              0.000000              0.000000             0.0        0.0
3              0.000000              0.000000             0.0        0.0
4              1.000000              1.000000             3.0        3.0

   PotentialFraud
0              No
1             Yes
2              No
3             Yes
4              No

[5 rows x 568 columns]
```

[56]:
```
# Print all column names
print(train_df.columns.tolist())
```

```
['Provider', 'OP_AvgClaims', 'OP_InscClaimAmtReimbursed',
'OP_DeductibleAmtPaid', 'OP_Admit?', 'OP_NoClmProcedureCodes',
'OP_NoClmDiagnosisCodes', 'OP_AttPhy?', 'OP_OpPhy?', 'OP_OthPhy?',
'OP_ClaimProcessTime', 'OP_4019', 'OP_25000', 'OP_2724', 'OP_V5869', 'OP_4011',
'OP_V5861', 'OP_2720', 'OP_42731', 'OP_2449', 'OP_78079', 'OP_53081', 'OP_2859',
'OP_496', 'OP_4280', 'OP_28521', 'OP_41400', 'OP_41401', 'OP_2809', 'OP_58881',
'OP_3051', 'OP_2722', 'OP_73300', 'OP_311', 'OP_V5883', 'OP_5990', 'OP_71590',
```

'OP_7242', 'OP_78650', 'OP_5859', 'OP_7295', 'OP_V4581', 'OP_V5866', 'OP_78605',
'OP_72887', 'OP_7812', 'OP_7823', 'OP_56210', 'OP_412', 'OP_30000', 'OP_49390',
'OP_V7644', 'OP_2721', 'OP_5856', 'OP_V1254', 'OP_7862', 'OP_7245', 'OP_42789',
'OP_4439', 'OP_78659', 'OP_185', 'OP_2948', 'OP_V103', 'OP_2723', 'OP_5853',
'OP_7197', 'OP_60000', 'OP_7802', 'OP_79029', 'OP_V4582', 'OP_78900',
'OP_73390', 'OP_71690', 'OP_V1582', 'OP_V5867', 'OP_27800', 'OP_71941',
'OP_71946', 'OP_V0481', 'OP_5939', 'OP_V4501', 'OP_7804', 'OP_78609',
'OP_V1046', 'OP_V420', 'OP_4279', 'OP_V4589', 'OP_78651', 'OP_7840', 'OP_7231',
'OP_25001', 'OP_5533', 'OP_7291', 'OP_3320', 'OP_7851', 'OP_79021', 'OP_2689',
'OP_78791', 'OP_40390', 'OP_5854', 'OP_56400', 'OP_72252', 'OP_42781',
'OP_51889', 'OP_7906', 'OP_71945', 'OP_2768', 'OP_V4365', 'OP_3310', 'OP_25002',
'OP_V571', 'OP_9904', 'OP_4516', 'OP_3722', 'OP_66', 'OP_5123', 'OP_9672',
'OP_3893', 'OP_9952', 'OP_8152', 'OP_9955', 'OP_3995', 'OP_8622', 'OP_9390',
'OP_4513', 'OP_3895', 'OP_8703', 'OP_966', 'OP_7820', 'OP_1741', 'OP_9425',
'OP_5849', 'OP_3794', 'OP_3422', 'OP_5794', 'OP_8872', 'OP_51', 'OP_3723',
'OP_9919', 'OP_4573', 'OP_3811', 'OP_239', 'OP_8154', 'OP_7939', 'OP_4571',
'OP_9982', 'OP_527', 'OP_9923', 'OP_6929', 'OP_5305', 'OP_5459', 'OP_5185',
'OP_3787', 'OP_8151', 'OP_9411', 'OP_4432', 'OP_7971', 'OP_4142', 'OP_4401',
'OP_151', 'OP_4701', 'OP_604', 'OP_5503', 'OP_8102', 'OP_8669', 'OP_3772',
'OP_2763', 'OP_9671', 'OP_9359', 'OP_9915', 'OP_9961', 'OP_2731', 'OP_8051',
'OP_3783', 'OP_554', 'OP_7931', 'OP_8659', 'OP_3950', 'OP_387', 'OP_6859',
'OP_4523', 'OP_4311', 'OP_5369', 'OP_3607', 'OP_9999', 'OP_5119', 'OP_5304',
'OP_3776', 'OP_8627', 'OP_9339', 'OP_9723', 'OP_4299', 'OP_9394', 'OP_3522',
'OP_9971', 'OP_5739', 'OP_4945', 'OP_4674', 'OP_9929', 'OP_9604', 'OP_4023',
'OP_7915', 'OP_V7612', 'OP_V7283', 'OP_1629', 'OP_78909', 'OP_V7651',
'OP_V5811', 'OP_7140', 'OP_95901', 'OP_78701', 'OP_72981', 'OP_7244',
'OP_79380', 'OP_61172', 'OP_V580', 'OP_V7284', 'OP_V6709', 'OP_9597',
'OP_V7611', 'OP_V6759', 'OP_3669', 'OP_36616', 'OP_59970', 'OP_43310',
'OP_78906', 'OP_5920', 'OP_1539', 'OP_V5331', 'OP_78039', 'OP_32723', 'OP_7881',
'OP_20280', 'OP_72402', 'OP_7821', 'OP_4414', 'OP_7847', 'OP_78820', 'OP_78901',
'OP_71947', 'OP_5693', 'OP_78703', 'OP_70715', 'OP_79093', 'OP_25080',
'OP_V1272', 'OP_V7281', 'OP_7866', 'OP_99673', 'OP_7213', 'OP_4359', 'OP_45340',
'OP_20300', 'OP_Gender', 'OP_RenalDiseaseIndicator', 'OP_NoOfMonths_PartACov',
'OP_NoOfMonths_PartBCov', 'OP_ChronicCond_Alzheimer',
'OP_ChronicCond_Heartfailure', 'OP_ChronicCond_KidneyDisease',
'OP_ChronicCond_Cancer', 'OP_ChronicCond_ObstrPulmonary',
'OP_ChronicCond_Depression', 'OP_ChronicCond_Diabetes',
'OP_ChronicCond_IschemicHeart', 'OP_ChronicCond_Osteoporasis',
'OP_ChronicCond_rheumatoidarthritis', 'OP_ChronicCond_stroke',
'OP_IPAnnualReimbursementAmt', 'OP_IPAnnualDeductibleAmt',
'OP_OPAnnualReimbursementAmt', 'OP_OPAnnualDeductibleAmt', 'OP_dead', 'OP_age',
'OP_12Months_PartACov', 'OP_12Months_PartBCov', 'OP_TotalClaims', 'OP_NoBene',
'IP_index', 'IP_AvgClaims', 'IP_InscClaimAmtReimbursed', 'IP_DeductibleAmtPaid',
'IP_days_in_hospital', 'IP_NoClmProcedureCodes', 'IP_NoClmDiagnosisCodes',
'IP_AttPhy?', 'IP_OpPhy?', 'IP_OthPhy?', 'IP_ClaimProcessTime', 'IP_4019',
'IP_2724', 'IP_25000', 'IP_41401', 'IP_4280', 'IP_42731', 'IP_5990', 'IP_53081',
'IP_2449', 'IP_5849', 'IP_496', 'IP_486', 'IP_2859', 'IP_40390', 'IP_41400',
'IP_2761', 'IP_27651', 'IP_3051', 'IP_2768', 'IP_2720', 'IP_5859', 'IP_49121',

```
'IP_311', 'IP_71590', 'IP_2948', 'IP_2851', 'IP_51881', 'IP_5856', 'IP_73300',
'IP_412', 'IP_V4581', 'IP_40391', 'IP_V1582', 'IP_4254', 'IP_0389', 'IP_42789',
'IP_2639', 'IP_V1254', 'IP_4439', 'IP_3310', 'IP_30000', 'IP_5119', 'IP_60000',
'IP_27800', 'IP_V5789', 'IP_V4582', 'IP_V5861', 'IP_5180', 'IP_41071',
'IP_56400', 'IP_2762', 'IP_4589', 'IP_4168', 'IP_49390', 'IP_5070', 'IP_32723',
'IP_2767', 'IP_99591', 'IP_V4501', 'IP_34590', 'IP_27801', 'IP_29410',
'IP_4148', 'IP_7802', 'IP_71536', 'IP_V5866', 'IP_5853', 'IP_78659', 'IP_4240',
'IP_6826', 'IP_43491', 'IP_2875', 'IP_4241', 'IP_5939', 'IP_3004', 'IP_5533',
'IP_V5867', 'IP_79902', 'IP_42732', 'IP_V1251', 'IP_V4365', 'IP_43820',
'IP_42823', 'IP_4111', 'IP_4271', 'IP_V854', 'IP_2809', 'IP_5854', 'IP_56210',
'IP_7140', 'IP_78079', 'IP_25002', 'IP_43310', 'IP_78720', 'IP_5601', 'IP_2749',
'IP_0414', 'IP_2760', 'IP_33829', 'IP_V103', 'IP_9904', 'IP_8154', 'IP_66',
'IP_3893', 'IP_3995', 'IP_4516', 'IP_3722', 'IP_8151', 'IP_8872', 'IP_9671',
'IP_4513', 'IP_9390', 'IP_9672', 'IP_5123', 'IP_7935', 'IP_8152', 'IP_9339',
'IP_3812', 'IP_3491', 'IP_3950', 'IP_3772', 'IP_5185', 'IP_8108', 'IP_4523',
'IP_309', 'IP_9921', 'IP_387', 'IP_7915', 'IP_8622', 'IP_4525', 'IP_5491',
'IP_4311', 'IP_8703', 'IP_9604', 'IP_5794', 'IP_3324', 'IP_4139', 'IP_6029',
'IP_8102', 'IP_3783', 'IP_3612', 'IP_3723', 'IP_3613', 'IP_9929', 'IP_9971',
'IP_331', 'IP_3895', 'IP_8604', 'IP_3521', 'IP_4573', 'IP_8051', 'IP_8659',
'IP_4542', 'IP_4443', 'IP_9925', 'IP_4576', 'IP_3929', 'IP_8891', 'IP_7936',
'IP_9462', 'IP_5459', 'IP_4562', 'IP_3971', 'IP_9907', 'IP_5122', 'IP_9955',
'IP_3794', 'IP_605', 'IP_3572', 'IP_8191', 'IP_51', 'IP_9607', 'IP_8180',
'IP_3404', 'IP_3734', 'IP_8801', 'IP_6849', 'IP_9915', 'IP_966', 'IP_5749',
'IP_8628', 'IP_4011', 'IP_7855', 'IP_5551', 'IP_8741', 'IP_7749', 'IP_9962',
'IP_9961', 'IP_8411', 'IP_3327', 'IP_78650', 'IP_78605', 'IP_78097', 'IP_78900',
'IP_5789', 'IP_78060', 'IP_78609', 'IP_78701', 'IP_8208', 'IP_7804', 'IP_71535',
'IP_5609', 'IP_4359', 'IP_7295', 'IP_78009', 'IP_71596', 'IP_2989', 'IP_78791',
'IP_82021', 'IP_71945', 'IP_78909', 'IP_72402', 'IP_7242', 'IP_5693', 'IP_5770',
'IP_29570', 'IP_78039', 'IP_7245', 'IP_7862', 'IP_5781', 'IP_5589', 'IP_56211',
'IP_71946', 'IP_41519', 'IP_78906', 'IP_4414', 'IP_71516', 'IP_25080',
'IP_7840', 'IP_78901', 'IP_78703', 'IP_185', 'IP_29680', 'IP_72981', 'IP_41091',
'IP_82009', 'IP_99859', 'IP_71595', 'IP_5780', 'IP_436', 'IP_72210', 'IP_7851',
'IP_7823', 'IP_431', 'IP_60001', 'IP_42781', 'IP_45340', 'IP_99673', 'IP_51884',
'IP_72989', 'IP_49122', 'IP_29530', 'IP_59970', 'IP_Gender',
'IP_RenalDiseaseIndicator', 'IP_NoOfMonths_PartACov', 'IP_NoOfMonths_PartBCov',
'IP_ChronicCond_Alzheimer', 'IP_ChronicCond_Heartfailure',
'IP_ChronicCond_KidneyDisease', 'IP_ChronicCond_Cancer',
'IP_ChronicCond_ObstrPulmonary', 'IP_ChronicCond_Depression',
'IP_ChronicCond_Diabetes', 'IP_ChronicCond_IschemicHeart',
'IP_ChronicCond_Osteoporasis', 'IP_ChronicCond_rheumatoidarthritis',
'IP_ChronicCond_stroke', 'IP_IPAnnualReimbursementAmt',
'IP_IPAnnualDeductibleAmt', 'IP_OPAnnualReimbursementAmt',
'IP_OPAnnualDeductibleAmt', 'IP_dead', 'IP_age', 'IP_12Months_PartACov',
'IP_12Months_PartBCov', 'IP_TotalClaims', 'IP_NoBene', 'PotentialFraud']
```

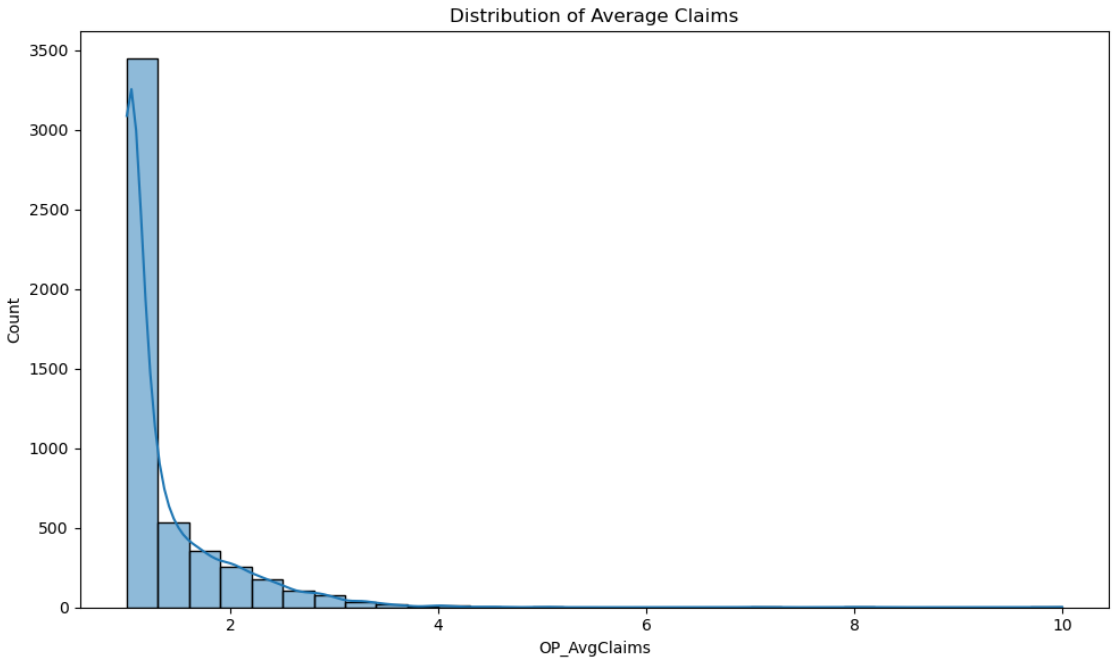
[57]:
```
# Check for null values and get the count of null values per column
null_values_count = train_df.isnull().sum()
```

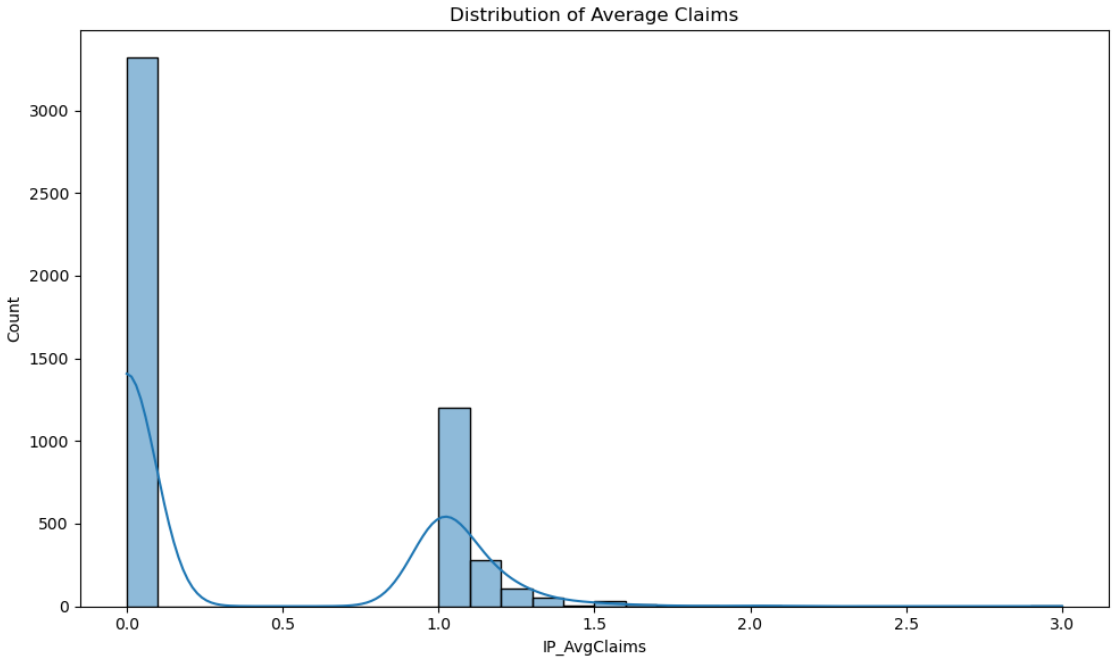
```
# Filter out columns that have null values and their counts
null_values_count = null_values_count[null_values_count > 0]

# Print the columns with null values and their counts
print(null_values_count)
```

```
Series([], dtype: int64)
```

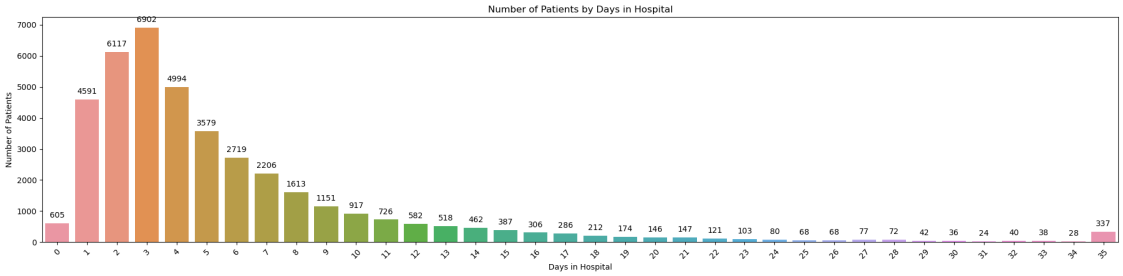
3 EDA

[58]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Histogram for numerical feature
plt.figure(figsize=(10, 6))
sns.histplot(data=train_df, x='OP_AvgClaims', kde=True, bins=30)
plt.title('Distribution of Average Claims')
plt.show()

# Histogram for numerical feature
plt.figure(figsize=(10, 6))
sns.histplot(data=train_df, x='IP_AvgClaims', kde=True, bins=30)
plt.title('Distribution of Average Claims')
plt.show()
```

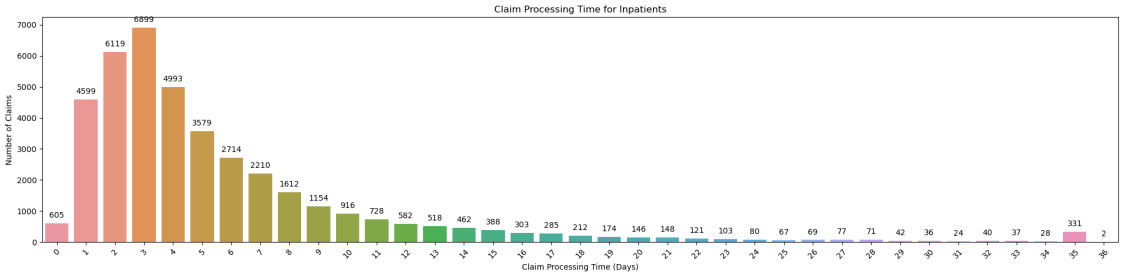




[59]:
```
plt.rcParams["figure.figsize"] = [20, 5]
ax = sns.countplot(x=inpatient_data_df['days_in_hospital'])

# Adding labels on top of each bar
for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}',  # Text to be displayed; convert to␣
 ↪int for cleanliness
                (p.get_x() + p.get_width() / 2., p.get_height()),  # Position
                ha='center',  # Center the text horizontally
                va='center',  # Center the text vertically
                xytext=(0, 10),  # Distance from the top of the bar
                textcoords='offset points')

plt.title('Number of Patients by Days in Hospital')
plt.xlabel('Days in Hospital')
plt.ylabel('Number of Patients')
plt.xticks(rotation=45)  # Rotate labels if they overlap
plt.show()
```

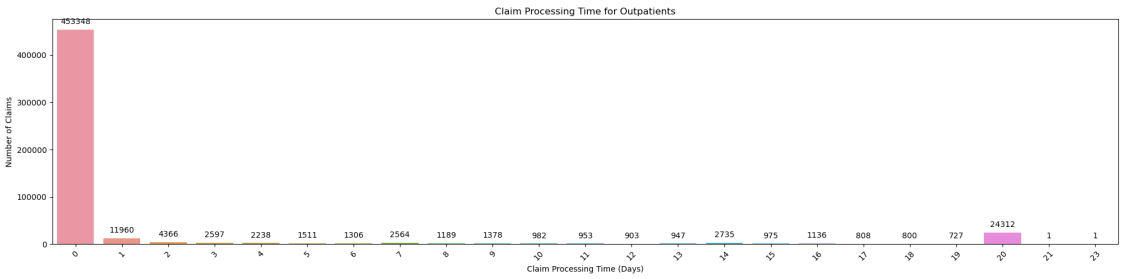


[60]:
```
plt.rcParams["figure.figsize"] = [20, 5]
ax = sns.countplot(x=inpatient_data_df['ClaimProcessTime'])

# Adding labels on top of each bar
for p in ax.patches:
    # The text to display (the count) and its position
    ax.annotate(f'{int(p.get_height())}',  # Convert count to int for␣
 ↪cleanliness
                (p.get_x() + p.get_width() / 2., p.get_height()),  # Position
                ha='center',  # Center the text horizontally on the bar
                va='center',  # Center the text vertically in relation to the␣
 ↪bar
                xytext=(0, 10),  # Distance (in points) from the top of the bar
                textcoords='offset points')  # Offset (in points) from the xy␣
 ↪value

plt.title('Claim Processing Time for Inpatients')
plt.xlabel('Claim Processing Time (Days)')
plt.ylabel('Number of Claims')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability if needed
plt.show()
```



[61]:
```
plt.rcParams["figure.figsize"] = [20, 5]
ax = sns.countplot(x=outpatient_data_df['ClaimProcessTime'])
```

```
# Adding labels on top of each bar
for p in ax.patches:
    # The text to display (the count) and its position
    ax.annotate(f'{int(p.get_height())}',  # Convert count to int for␣
 ↪cleanliness
                (p.get_x() + p.get_width() / 2., p.get_height()),  # Position
                ha='center',  # Center the text horizontally on the bar
                va='center',  # Center the text vertically in relation to the␣
 ↪bar
                xytext=(0, 10),  # Distance (in points) from the top of the bar
                textcoords='offset points')  # Offset (in points) from the xy␣
 ↪value

plt.title('Claim Processing Time for Outpatients')
plt.xlabel('Claim Processing Time (Days)')
plt.ylabel('Number of Claims')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability if needed
plt.show()
```



[62]:
```
# Count the number of claims per provider and sort in descending order
provider_claims_count = inpatient_data_df['Provider'].value_counts()

# Get the provider with the highest number of claims
highest_claims_provider = provider_claims_count.idxmax()
highest_claims_count = provider_claims_count.max()

print(f"The provider with the highest number of claims for inpatients is␣
 ↪{highest_claims_provider} with {highest_claims_count} claims.")
```

```
The provider with the highest number of claims for inpatients is PRV52019 with
516 claims.
```

[63]:
```
# Count the number of claims per provider and sort in descending order
provider_claims_count = outpatient_data_df['Provider'].value_counts()

# Get the provider with the highest number of claims
```

```
highest_claims_provider = provider_claims_count.idxmax()
highest_claims_count = provider_claims_count.max()

print(f"The provider with the highest number of claims for outpatients is␣
 ↪{highest_claims_provider} with {highest_claims_count} claims.")
```

```
The provider with the highest number of claims for outpatients is PRV51459 with
8240 claims.
```

4 Machine Learning Models

4.0.1 Define & Split

[64]:
```
Y= train_df['PotentialFraud'].map(lambda x: 1 if x == 'Yes' else 0)
X = train_df.drop(['PotentialFraud', 'Provider'], axis=1)
```

[65]:
```
#split the data into training and test data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,␣
 ↪random_state=42)

#number of training and test samples
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
```

```
(4009, 566) (4009,) (1003, 566) (1003,)
```

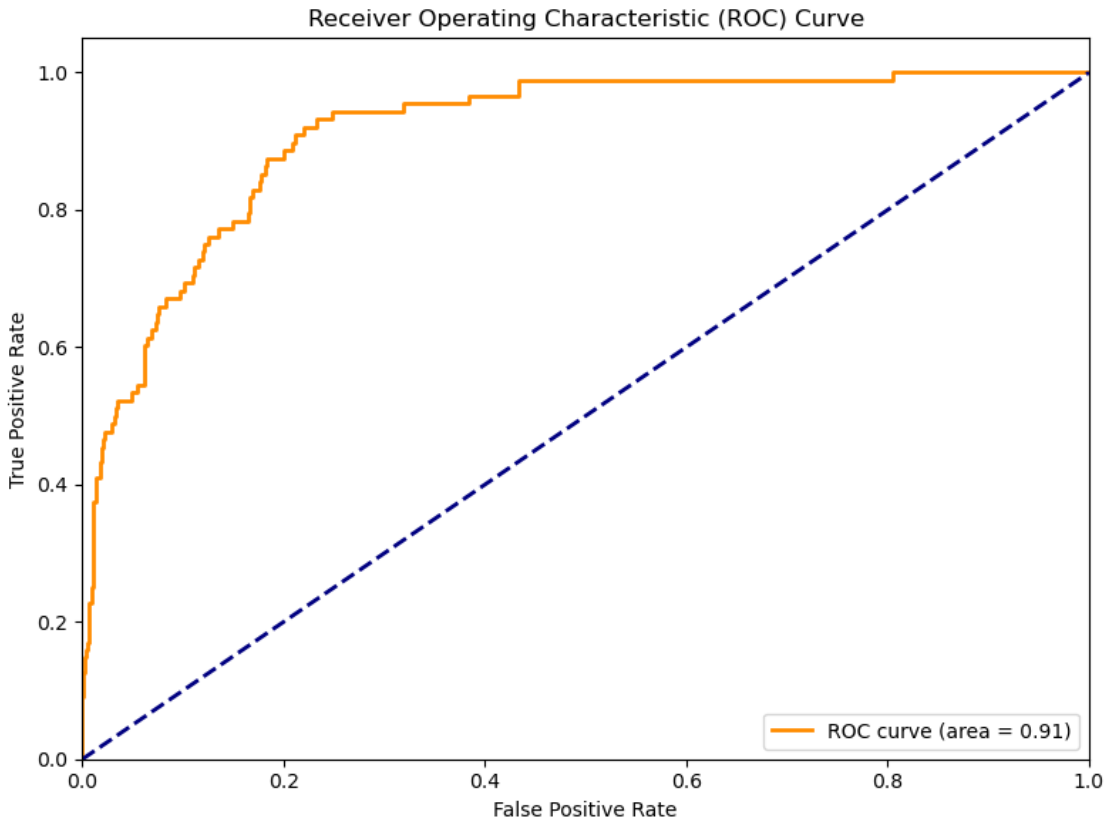
4.1 Logistic Regression

[66]:
```
from scipy.stats import uniform
from sklearn.model_selection import RandomizedSearchCV
from sklearn.linear_model import LogisticRegression
import warnings
from sklearn.exceptions import ConvergenceWarning

# Hyperparameter distribution for randomized search
param_dist_lr = {
    'l1_ratio': uniform(0, 1),
}

# Suppressing convergence warnings
with warnings.catch_warnings():
    warnings.filterwarnings("ignore", category=ConvergenceWarning)

    # Initializing the Logistic Regression model with elasticnet penalty
    clf_logreg = LogisticRegression(penalty='elasticnet', solver='saga',␣
 ↪l1_ratio=0.5, max_iter=5000, random_state=42, class_weight='balanced')

    # Setting up the randomized search with Logistic Regression
```

```
    paramsearch_random = RandomizedSearchCV(
        estimator=clf_logreg, param_distributions=param_dist_lr, n_iter=20,␣
 ↪random_state=42
    )

    # Fitting the model
    paramsearch_random.fit(X_train, y_train)

# Getting the best estimator after the randomized search
logreg_clf = paramsearch_random.best_estimator_
```

[67]:
```
from sklearn.metrics import roc_curve, roc_auc_score, classification_report,␣
 ↪confusion_matrix
import matplotlib.pyplot as plt

# Predict probabilities for the test set
y_probs = logreg_clf.predict_proba(X_test)[:, 1]  # Get the probability of the␣
 ↪positive class

# Calculate the ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_probs)

# Calculate the AUC score
auc_score = roc_auc_score(y_test, y_probs)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %␣
 ↪auc_score)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

# Predict classes for the test set
y_pred = logreg_clf.predict(X_test)

# Calculate accuracy on the training set
train_accuracy = logreg_clf.score(X_train, y_train)
print(f"Training Accuracy: {train_accuracy:.4f}")

# Calculate accuracy on the test set
test_accuracy = logreg_clf.score(X_test, y_test)
```

```python
print(f"Test Accuracy: {test_accuracy:.4f}")


# Generate the classification report
class_report = classification_report(y_test, y_pred)
print("Classification Report:\n", class_report)

# Generate the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", conf_matrix)
```




```
Training Accuracy: 0.7832
Test Accuracy: 0.7986
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.79      0.88       915
           1       0.29      0.90      0.44        88

    accuracy                           0.80      1003
   macro avg       0.64      0.84      0.66      1003
```

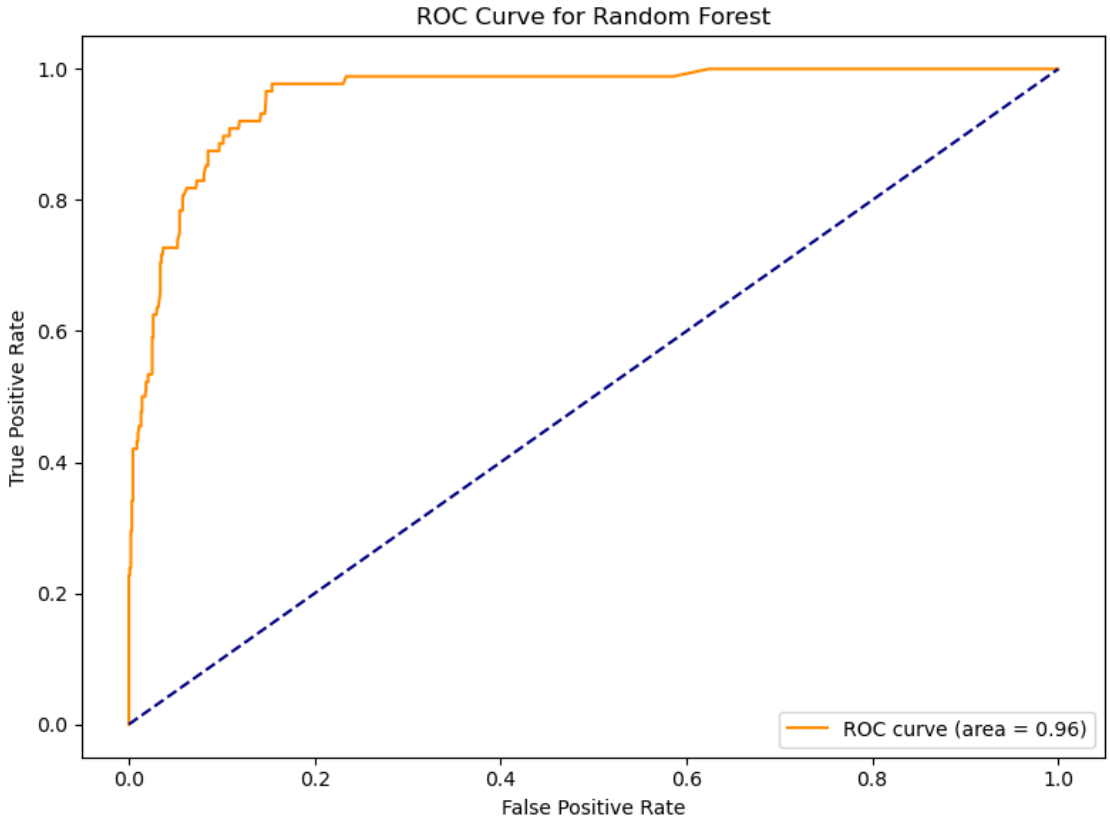
```
weighted avg       0.93      0.80      0.84      1003

Confusion Matrix:
 [[722 193]
 [  9  79]]
```

4.2 Random Forest

[68]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# Initialize the RandomForestClassifier with 1000 trees
clf_rf = RandomForestClassifier(n_estimators=1000)

# Define the hyperparameter grid to be used in GridSearchCV
param_grid_rf = {
 'criterion': ["gini", "entropy", "log_loss"],  # Criteria for splitting
 'max_features': ["sqrt", "log2"],  # The number of features to consider when␣
 ↪looking for the best split
 'class_weight': ['balanced']  # Weights associated with classes to handle␣
 ↪imbalanced classes
}

# Initialize GridSearchCV with the random forest classifier, the hyperparameter␣
 ↪grid, and parallel jobs
paramsearch_random = GridSearchCV(
    clf_rf, param_grid=param_grid_rf, n_jobs=3
)

# Fit GridSearchCV to the training data
paramsearch_random.fit(X_train, y_train)

# Extract the best estimator (random forest model with best hyperparameters)
rf_clf = paramsearch_random.best_estimator_
```

[69]:
```
from sklearn.metrics import roc_curve, roc_auc_score, classification_report,␣
 ↪confusion_matrix
import matplotlib.pyplot as plt

# Predict probabilities for the test set
y_probs_rf = rf_clf.predict_proba(X_test)[:, 1]

# ROC curve and AUC score
fpr_rf, tpr_rf, thresholds_rf = roc_curve(y_test, y_probs_rf)
auc_score_rf = roc_auc_score(y_test, y_probs_rf)
plt.figure(figsize=(8, 6))
```

```
plt.plot(fpr_rf, tpr_rf, color='darkorange', label=f'ROC curve (area =␣
 ↪{auc_score_rf:.2f})')
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Random Forest')
plt.legend(loc="lower right")
plt.show()

# Classification report and confusion matrix
y_pred_rf = rf_clf.predict(X_test)
print("Random Forest Classification Report:\n", classification_report(y_test,␣
 ↪y_pred_rf))
print("Random Forest Confusion Matrix:\n", confusion_matrix(y_test, y_pred_rf))

# Accuracy on training and test set
train_accuracy_rf = rf_clf.score(X_train, y_train)
test_accuracy_rf = rf_clf.score(X_test, y_test)
print(f"Random Forest Training Accuracy: {train_accuracy_rf:.4f}")
print(f"Random Forest Test Accuracy: {test_accuracy_rf:.4f}")
```



```
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.99      0.97       915
           1       0.80      0.45      0.58        88

    accuracy                           0.94      1003
   macro avg       0.87      0.72      0.77      1003
weighted avg       0.94      0.94      0.93      1003

Random Forest Confusion Matrix:
 [[905  10]
 [ 48  40]]
Random Forest Training Accuracy: 1.0000
Random Forest Test Accuracy: 0.9422
```

4.3 XGBoost

[70]:
```
!pip install xgboost
```

```
Requirement already satisfied: xgboost in
/Users/raja/anaconda3/lib/python3.11/site-packages (2.0.3)
Requirement already satisfied: numpy in
/Users/raja/anaconda3/lib/python3.11/site-packages (from xgboost) (1.24.3)
Requirement already satisfied: scipy in
/Users/raja/anaconda3/lib/python3.11/site-packages (from xgboost) (1.10.1)
```

[71]:
```
import xgboost as xgb
from sklearn.model_selection import train_test_split

# Initialize the XGBoost Classifier
xgb_clf = xgb.XGBClassifier(tree_method="hist", early_stopping_rounds=2,␣
 ↪verbosity=1, random_state=42, n_jobs=3)

# Split the training data for early stopping
X_trainss, X_val, y_trainss, y_val = train_test_split(X_train, y_train,␣
 ↪test_size=0.2, random_state=42)

# Fit the model; the validation set is used for early stopping
xgb_clf.fit(X_trainss, y_trainss, eval_set=[(X_val, y_val)], verbose=True)
```

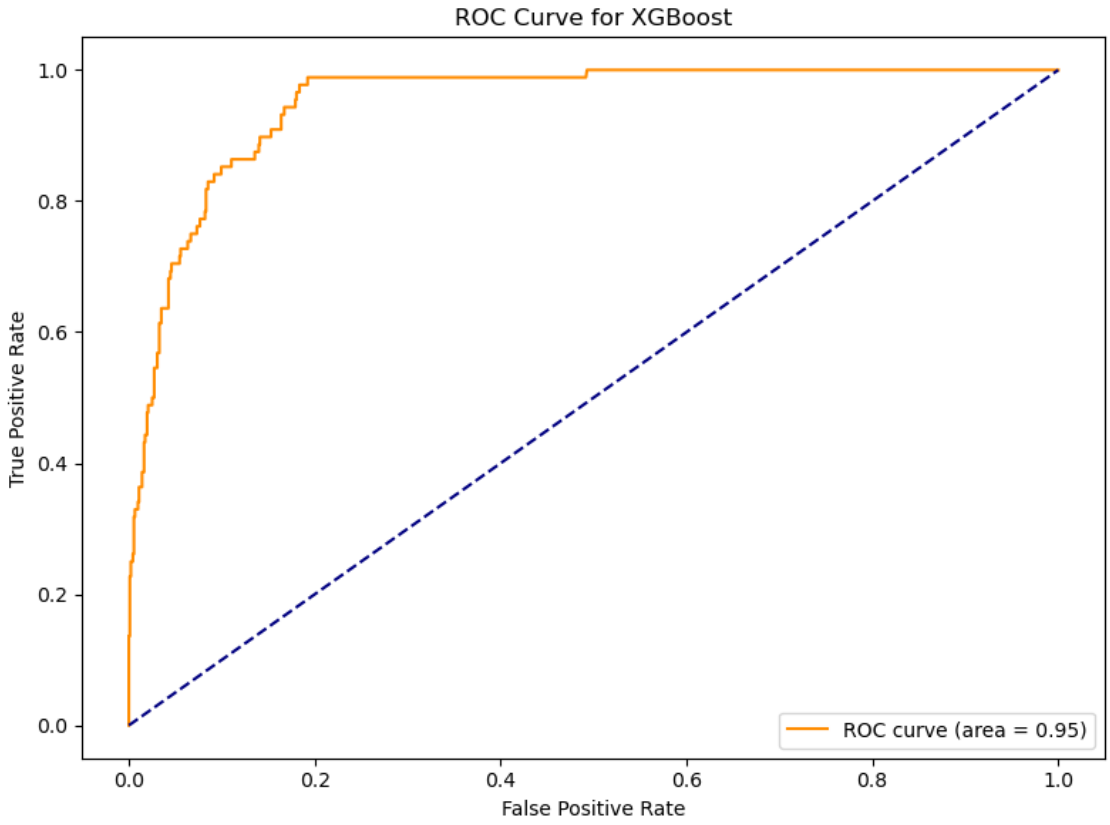
```
[0]     validation_0-logloss:0.24967
[1]     validation_0-logloss:0.21121
[2]     validation_0-logloss:0.19283
[3]     validation_0-logloss:0.17737
[4]     validation_0-logloss:0.16865
[5]     validation_0-logloss:0.16312
[6]     validation_0-logloss:0.15767
[7]     validation_0-logloss:0.15534
```

```
[8]     validation_0-logloss:0.15224
[9]     validation_0-logloss:0.14949
[10]    validation_0-logloss:0.15012
[11]    validation_0-logloss:0.14892
[12]    validation_0-logloss:0.14722
[13]    validation_0-logloss:0.14593
[14]    validation_0-logloss:0.14613
[15]    validation_0-logloss:0.14616
```

[71]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=2,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=3,
              num_parallel_tree=None, random_state=42, …)
```

[72]:
```
from sklearn.metrics import roc_curve, roc_auc_score, classification_report,␣
 ↪confusion_matrix
import matplotlib.pyplot as plt

# Predict probabilities for the test set
y_probs_xgb = xgb_clf.predict_proba(X_test)[:, 1]

# Calculate the ROC curve and AUC score
fpr_xgb, tpr_xgb, thresholds_xgb = roc_curve(y_test, y_probs_xgb)
auc_score_xgb = roc_auc_score(y_test, y_probs_xgb)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr_xgb, tpr_xgb, color='darkorange', label=f'ROC curve (area =␣
 ↪{auc_score_xgb:.2f})')
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for XGBoost')
plt.legend(loc="lower right")
plt.show()

# Generate the classification report and confusion matrix
y_pred_xgb = xgb_clf.predict(X_test)
print("XGBoost Classification Report:\n", classification_report(y_test,␣
 ↪y_pred_xgb))
```

```python
print("XGBoost Confusion Matrix:\n", confusion_matrix(y_test, y_pred_xgb))

# Calculate accuracy on the training and test set
train_accuracy_xgb = xgb_clf.score(X_train, y_train)
test_accuracy_xgb = xgb_clf.score(X_test, y_test)
print(f"XGBoost Training Accuracy: {train_accuracy_xgb:.4f}")
print(f"XGBoost Test Accuracy: {test_accuracy_xgb:.4f}")
```




```
XGBoost Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.98      0.97       915
           1       0.69      0.49      0.57        88

    accuracy                           0.94      1003
   macro avg       0.82      0.73      0.77      1003
weighted avg       0.93      0.94      0.93      1003

XGBoost Confusion Matrix:
 [[896  19]
 [ 45  43]]
```

```
XGBoost Training Accuracy: 0.9818
XGBoost Test Accuracy: 0.9362
```

[ ]: