Load Python libraries

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import SimpleImputer
from pprint import pprint
from scipy import stats
```

Read CSV & Load Data in to Pandas Dataframe

In [2]:
```
data = pd.read_csv(r'C:\Users\JoshFunderburk\Desktop\D206 Project\medical_raw_data.csv', index_col=0)
```

In [3]:
```
data.head()
```

Out[3]:

	CaseOrder	Customer_id	Interaction	UID	City	State
1	1	C412403	8cd49b13-f45a-4b47-a2bd-173ffa932c2f	3a83ddb66e2ae73798bdf1d705dc0932	Eva	AL
2	2	Z919181	d2450b70-0337-4406-bdbb-bc1037f1734c	176354c5eef714957d486009feabf195	Marianna	FL
3	3	F995323	a2057123-abf5-4a2c-abad-8ffe33512562	e19a0fa00aeda885b8a436757e889bc9	Sioux Falls	SD
4	4	A879973	1dec528d-eb34-4079-adce-0d7a40e82205	cd17d7b6d152cb6f23957346d11c3f07	New Richland	MN
5	5	C544523	5885f56b-d6da-43a3-8760-83583af94266	d2f0425877b10ed6bb381f3e2579424a	West Point	VA

5 rows × 52 columns

List Column Names, Data Types, & Unique Values

In [4]:
```
dt = data.dtypes
print(dt)
```

```
CaseOrder              int64
Customer_id           object
Interaction           object
UID                   object
City                  object
State                 object
County                object
Zip                    int64
Lat                  float64
Lng                  float64
Population             int64
Area                  object
Timezone              object
Job                   object
Children             float64
Age                  float64
Education             object
Employment            object
Income               float64
Marital               object
Gender                object
ReAdmis               object
VitD_levels          float64
Doc_visits             int64
Full_meals_eaten       int64
VitD_supp              int64
Soft_drink            object
Initial_admin         object
HighBlood             object
Stroke                object
Complication_risk     object
Overweight           float64
Arthritis             object
Diabetes              object
Hyperlipidemia        object
BackPain              object
Anxiety              float64
Allergic_rhinitis     object
Reflux_esophagitis    object
Asthma                object
Services              object
Initial_days         float64
TotalCharge          float64
Additional_charges   float64
Item1                  int64
Item2                  int64
Item3                  int64
Item4                  int64
Item5                  int64
Item6                  int64
Item7                  int64
Item8                  int64
dtype: object
```

In [5]:
```
for col in data:
    pprint(data[col].unique())
```

```
array([    1,     2,     3, ...,  9998,  9999, 10000], dtype=int64)
array(['C412403', 'Z919181', 'F995323', ..., 'R778890', 'E344109',
       'I569847'], dtype=object)
array(['8cd49b13-f45a-4b47-a2bd-173ffa932c2f',
       'd2450b70-0337-4406-bdbb-bc1037f1734c',
       'a2057123-abf5-4a2c-abad-8ffe33512562', ...,
       '1d79569d-8e0f-4180-a207-d67ee4527d26',
       'f5a68e69-2a60-409b-a92f-ac0847b27db0',
       'bc482c02-f8c9-4423-99de-3db5e62a18d5'], dtype=object)
array(['3a83ddb66e2ae73798bdf1d705dc0932',
       '176354c5eef714957d486009feabf195',
       'e19a0fa00aeda885b8a436757e889bc9', ...,
       '41b770aeee97a5b9e7f69c906a8119d7',
       '2bb491ef5b1beb1fed758cc6885c167a',
       '95663a202338000abdf7e09311c2a8a1'], dtype=object)
array(['Eva', 'Marianna', 'Sioux Falls', ..., 'Milmay', 'Quinn',
       'Coraopolis'], dtype=object)
array(['AL', 'FL', 'SD', 'MN', 'VA', 'OK', 'OH', 'MS', 'WI', 'IA', 'CA',
       'IN', 'MO', 'MI', 'NE', 'PA', 'AR', 'WV', 'KS', 'MA', 'KY', 'NY',
       'VT', 'DC', 'IL', 'ND', 'SC', 'AK', 'NM', 'NH', 'GA', 'NC', 'MD',
       'TN', 'WA', 'TX', 'CO', 'NJ', 'LA', 'OR', 'AZ', 'ME', 'ID', 'UT',
       'RI', 'MT', 'PR', 'NV', 'CT', 'HI', 'WY', 'DE'], dtype=object)
array(['Morgan', 'Jackson', 'Minnehaha', ..., 'Navarro', 'Los Alamos',
       'Sterling'], dtype=object)
array([35621, 32446, 57110, ...,  8340, 57775, 15108], dtype=int64)
array([34.3496 , 30.84513, 43.54321, ..., 39.43609, 44.10354, 40.49998])
array([ -86.72508,  -85.22907,  -96.63772, ...,  -74.87302, -102.01593,
        -80.19959])
array([ 2951, 11303, 17125, ...,  8368,  7908, 41524], dtype=int64)
array(['Suburban', 'Urban', 'Rural'], dtype=object)
array(['America/Chicago', 'America/New_York', 'America/Los_Angeles',
       'America/Indiana/Indianapolis', 'America/Detroit',
       'America/Denver', 'America/Nome', 'America/Anchorage',
       'America/Phoenix', 'America/Boise', 'America/Puerto_Rico',
       'America/Yakutat', 'Pacific/Honolulu', 'America/Menominee',
       'America/Kentucky/Louisville', 'America/Indiana/Vincennes',
       'America/Toronto', 'America/Indiana/Marengo',
       'America/Indiana/Winamac', 'America/Indiana/Tell_City',
       'America/Sitka', 'America/Indiana/Knox',
       'America/North_Dakota/New_Salem', 'America/Indiana/Vevay',
       'America/Adak', 'America/North_Dakota/Beulah'], dtype=object)
array(['Psychologist, sport and exercise', 'Community development worker',
       'Chief Executive Officer', 'Early years teacher',
       'Health promotion specialist', 'Corporate treasurer',
       'Hydrologist', 'Psychiatric nurse', 'Computer games developer',
       'Production assistant, radio', 'Contractor',
       'Surveyor, planning and development',
       'English as a second language teacher', 'Actuary', 'Media planner',
       'Fast food restaurant manager', 'Horticulturist, commercial',
       'Secretary, company', 'Designer, graphic', 'Personnel officer',
       'Telecommunications researcher', 'Restaurant manager, fast food',
       'Surveyor, minerals', 'Architectural technologist',
       'Therapist, speech and language', 'Accounting technician',
       'Glass blower/designer', 'Travel agency manager', 'Illustrator',
       'Police officer', 'Accountant, chartered public finance',
       'Sport and exercise psychologist', 'Pensions consultant',
       'Community education officer', 'Radio producer',
```

```
'Designer, television/film set', 'Conference centre manager',
'Advertising account executive', 'Civil Service fast streamer',
'Training and development officer', 'Buyer, retail',
'Event organiser', 'IT technical support officer',
'Historic buildings inspector/conservation officer',
'Research scientist (physical sciences)', 'Games developer',
'Manufacturing engineer', 'Embryologist, clinical',
'Merchant navy officer', 'Television floor manager',
'Web designer', 'Industrial buyer', 'Aid worker',
'Systems developer', 'Probation officer',
'Scientific laboratory technician',
'Environmental health practitioner', 'Prison officer',
'Naval architect', 'Pilot, airline',
'Medical sales representative', 'Learning disability nurse',
'Agricultural engineer', 'Multimedia programmer', 'Cartographer',
'Company secretary', 'Operations geologist',
'Conservation officer, nature', 'Therapist, art',
'Therapist, sports', 'Oncologist',
'Armed forces logistics/support/administrative officer',
'Podiatrist', 'Translator', 'Geochemist',
'Engineer, technical sales',
'Production designer, theatre/television/film', 'Site engineer',
'Teacher, primary school', 'Clinical molecular geneticist',
'Armed forces operational officer', 'Careers information officer',
'Camera operator', 'Engineer, aeronautical', 'Learning mentor',
'Neurosurgeon', 'Clothing/textile technologist',
'Financial controller', 'Education officer, museum',
'Set designer', 'Accountant, chartered certified', 'Solicitor',
'Forensic psychologist', 'Outdoor activities/education manager',
'Heritage manager', 'Hospital doctor', 'Engineer, chemical',
'Musician', 'Engineer, control and instrumentation',
'Engineer, mining', 'Editor, commissioning',
'Sports development officer', 'Teacher, music',
"Nurse, children's", 'Editor, film/video', 'Acupuncturist',
'Data scientist', 'Tax inspector', 'Engineer, maintenance',
'Radiographer, therapeutic', 'Surveyor, commercial/residential',
'Engineer, civil (contracting)', 'Therapist, nutritional',
'Public affairs consultant', 'Warehouse manager',
'Consulting civil engineer', 'Museum/gallery exhibitions officer',
'Risk manager', 'Air traffic controller', 'Health service manager',
'Teacher, adult education', 'Theatre stage manager',
'Designer, fashion/clothing', 'Engineer, site',
'Psychologist, counselling',
'Product/process development scientist', 'Financial adviser',
'Quarry manager', 'Librarian, public', 'Presenter, broadcasting',
'Structural engineer', 'Trade mark attorney',
'Amenity horticulturist', 'Building services engineer',
'Primary school teacher', 'Network engineer',
'Psychotherapist, child', 'Archaeologist',
'Publishing rights manager', 'Economist', 'Herbalist',
'Legal secretary', 'Engineer, manufacturing systems',
'Psychologist, occupational', 'Journalist, broadcasting',
'Lexicographer', 'Clinical psychologist',
'Scientist, water quality',
'Chartered legal executive (England and Wales)', 'Statistician',
'Chartered accountant', 'Operational investment banker',
'Nutritional therapist', 'Actor', 'Ecologist',
```

```
'Conservator, furniture', 'Archivist',
'Industrial/product designer', 'Air broker', 'Sports coach',
'Chief Technology Officer', 'Arts administrator',
'Restaurant manager', 'Editorial assistant', 'Cytogeneticist',
'Scientist, marine', 'Surveyor, quantity',
'Designer, exhibition/display', 'Curator',
'Human resources officer', 'Osteopath', 'Therapist, music',
'Volunteer coordinator', 'Office manager',
'Research officer, government', 'Quality manager', 'Artist',
'Museum education officer', 'Exercise physiologist',
'Administrator, charities/voluntary organisations',
'Purchasing manager', 'Therapeutic radiographer', 'Farm manager',
'Tour manager', 'Writer', 'Designer, industrial/product',
'Science writer', 'Engineer, biomedical',
'Development worker, international aid', 'Journalist, newspaper',
'Multimedia specialist', 'Dealer', 'Water engineer',
'Scientist, clinical (histocompatibility and immunogenetics)',
'Special effects artist', 'Engineer, agricultural',
'Corporate investment banker', 'Best boy',
'Production assistant, television', 'Chiropractor',
'Jewellery designer', 'Energy engineer', 'Scientist, forensic',
'Biomedical engineer', 'Insurance account manager',
'Occupational psychologist', 'Diagnostic radiographer', 'Banker',
'Medical technical officer', 'Quantity surveyor',
'Biochemist, clinical', 'Broadcast engineer',
'Chartered management accountant', 'Theatre manager',
'Animal technologist', 'Animator', 'Producer, radio',
'Chiropodist', 'Exhibition designer', 'Occupational therapist',
'Database administrator', 'Arts development officer',
'Health and safety inspector', 'Press photographer',
'Recruitment consultant', 'Dance movement psychotherapist',
'Audiological scientist', 'Soil scientist', 'Equities trader',
'Orthoptist', 'Engineer, materials', 'Regulatory affairs officer',
'Trade union research officer', 'Research scientist (maths)',
'Television production assistant', 'Chief of Staff',
'Advertising copywriter',
'Programme researcher, broadcasting/film/video',
'Technical sales engineer', 'Music therapist',
'Electronics engineer', 'Waste management officer',
'Plant breeder/geneticist', 'Operational researcher',
'Further education lecturer', 'Electrical engineer',
'Television camera operator', 'Runner, broadcasting/film/video',
'Pharmacist, community', 'Ophthalmologist', 'Wellsite geologist',
'Psychologist, educational', 'Advertising account planner',
'Sports therapist', 'Surveyor, building control', 'Engineer, land',
'Clinical embryologist', 'Marine scientist',
'Teacher, secondary school', 'Chief Financial Officer',
'Landscape architect', 'Community pharmacist', 'Product manager',
'Financial risk analyst', 'Administrator',
'Civil engineer, contracting', 'Engineer, maintenance (IT)',
'Scientist, audiological', 'Management consultant', 'Dentist',
'Barrister', 'Surveyor, insurance', 'Customer service manager',
'Clinical cytogeneticist', 'Forest/woodland manager',
'Insurance underwriter', 'Speech and language therapist',
'Trading standards officer', 'Surveyor, building',
'Engineering geologist', 'Investment analyst',
'Research scientist (life sciences)', 'Firefighter',
```

```
'Higher education careers adviser', 'Theatre director',
'Passenger transport manager',
'English as a foreign language teacher',
'Research officer, trade union',
'Conservation officer, historic buildings',
'Scientist, product/process development', 'Air cabin crew',
'Colour technologist', 'Research officer, political party',
'Chemist, analytical', 'Hydrogeologist', 'Music tutor',
'Therapist, drama', 'Health physicist',
'Lecturer, higher education', 'Records manager',
'Scientist, research (medical)', 'Field trials officer',
'Adult guidance worker', 'Fine artist',
'Social research officer, government',
'Interior and spatial designer', 'Freight forwarder',
'Production engineer', 'Accommodation manager', 'Retail banker',
'Research scientist (medical)', 'Occupational hygienist',
'Diplomatic Services operational officer', "Barrister's clerk",
'Call centre manager', 'Tourism officer',
'Agricultural consultant', 'Armed forces technical officer',
"Politician's assistant",
'Geographical information systems officer',
'Chief Operating Officer', 'Higher education lecturer',
'Therapist, occupational', 'Land', 'Print production planner',
'Tree surgeon', 'Physiological scientist',
'Producer, television/film/video', 'Facilities manager',
'Designer, blown glass/stained glass', 'Location manager',
'Maintenance engineer', 'Meteorologist',
'Local government officer', 'Energy manager', 'Estate agent',
'Counsellor', 'Dispensing optician', 'Geophysical data processor',
'Adult nurse', 'Educational psychologist', 'Mental health nurse',
'IT sales professional', 'Water quality scientist',
'Advice worker', 'Intelligence analyst', 'Community arts worker',
'Optometrist', 'Patent examiner',
'Psychotherapist, dance movement', 'Gaffer', 'Risk analyst',
'Financial trader', 'Sales promotion account executive',
'Equality and diversity officer', 'Administrator, education',
'Medical secretary', 'Claims inspector/assessor',
'Child psychotherapist', 'Immigration officer', 'Metallurgist',
'Education administrator', 'Fitness centre manager',
'Chief Strategy Officer', 'Public librarian',
'Furniture conservator/restorer', 'Photographer',
'Production manager', 'Nature conservation officer',
'Phytotherapist', 'Therapist, horticultural',
'Aeronautical engineer', 'Engineer, civil (consulting)',
'Television/film/video producer', 'Solicitor, Scotland',
'Psychologist, forensic', 'Development worker, community',
'Engineer, manufacturing', 'Garment/textile technologist',
'Charity officer', 'Insurance risk surveyor',
'Broadcast presenter', 'Secretary/administrator',
'Civil Service administrator', 'Surveyor, hydrographic',
'Loss adjuster, chartered', 'Secondary school teacher',
'Teacher, special educational needs', 'Engineer, petroleum',
'Surveyor, rural practice', 'Information systems manager',
'Designer, furniture', 'Engineer, energy',
'Conservator, museum/gallery', 'Environmental consultant',
'Doctor, general practice', 'Nurse, mental health',
'Graphic designer', 'Investment banker, corporate', 'Astronomer',
```

```
'Data processing manager', 'Stage manager', 'Textile designer',
'Drilling engineer', 'Scientist, research (life sciences)',
'Furniture designer', 'Ambulance person', 'Buyer, industrial',
'Copywriter, advertising', 'Academic librarian',
'Scientist, research (maths)',
'International aid/development worker', 'Engineer, structural',
'Lecturer, further education', 'Interpreter',
'Chief Marketing Officer', 'Transport planner',
'Pharmacist, hospital', 'Toxicologist', 'Proofreader',
'Contracting civil engineer', 'Psychologist, clinical',
'Retail manager', 'Manufacturing systems engineer',
'Art therapist', 'Chartered certified accountant',
'Sales professional, IT', 'Dramatherapist',
'Designer, interior/spatial', 'Administrator, Civil Service',
'Printmaker', 'Engineer, electrical',
'Planning and development surveyor', 'Paediatric nurse',
'Designer, multimedia', 'Herpetologist', 'Mudlogger',
'Engineer, water', 'Arboriculturist', 'Sub',
'Sports administrator', 'Mechanical engineer',
'Physicist, medical',
'Armed forces training and education officer',
'Marketing executive', 'Magazine features editor', 'Ergonomist',
'Mining engineer', 'Dancer', 'Optician, dispensing',
'Designer, textile', 'Ranger/warden', 'Psychiatrist',
'Bonds trader', 'Technical brewer', 'Engineer, building services',
'Field seismologist', 'Engineer, electronics',
'Medical illustrator', 'Architect', 'Engineer, production',
'Licensed conveyancer', 'Surveyor, mining',
'Applications developer', 'Museum/gallery curator',
'Market researcher', 'Radiation protection practitioner',
'Control and instrumentation engineer', 'Programmer, applications',
'Advertising art director',
'Clinical scientist, histocompatibility and immunogenetics',
'Professor Emeritus', 'Horticulturist, amenity', 'Physiotherapist',
'Race relations officer', 'Surveyor, land/geomatics',
'Youth worker', 'Horticultural therapist', 'IT consultant', 'Make',
'Public relations account executive', 'Private music teacher',
'Fashion designer', 'Hospital pharmacist', 'Tax adviser',
'Engineer, broadcasting (operations)',
'Commercial art gallery manager', 'Legal executive',
'Visual merchandiser', 'Commercial/residential surveyor',
'Personal assistant', 'Insurance claims handler',
'Financial manager', 'Tourist information centre manager',
'Scientist, physiological', 'Designer, ceramics/pottery',
'Accountant, chartered management', 'Psychotherapist',
'Health visitor', 'Pharmacologist',
'Special educational needs teacher', 'Public relations officer',
'Town planner', 'Animal nutritionist', 'Building control surveyor',
'Engineer, automotive', 'Information officer',
'Senior tax professional/tax inspector', 'Film/video editor',
'Cabin crew', 'Radiographer, diagnostic', 'Warden/ranger',
'Video editor', 'Airline pilot', 'Newspaper journalist',
'Education officer, community', 'Geologist, engineering',
'Librarian, academic', 'Paramedic', 'Recycling officer',
'Merchandiser, retail', 'Retail merchandiser',
'Administrator, local government', 'Counselling psychologist',
'Estate manager/land agent', 'Oceanographer', 'Haematologist',
```

```
       'Scientist, research (physical sciences)', 'Medical physicist',
       'Communications engineer', 'Surgeon', 'Homeopath',
       'Charity fundraiser', 'Theme park manager', 'Barista',
       'Chartered public finance accountant',
       'Teaching laboratory technician', 'Microbiologist',
       'Programmer, multimedia', 'Automotive engineer',
       'Holiday representative', 'Systems analyst', 'Product designer',
       'Forensic scientist', 'Museum/gallery conservator',
       'Patent attorney', 'Ship broker', 'Technical author',
       'Pension scheme manager', 'Ceramics designer', 'Careers adviser',
       'Building surveyor', 'Public house manager',
       'Environmental education officer', 'Journalist, magazine',
       'Magazine journalist', 'Analytical chemist',
       'Teacher, English as a foreign language',
       'Lighting technician, broadcasting/film/video',
       'Teacher, early years/pre', 'Commercial horticulturist',
       'Publishing copy', 'Clinical biochemist', 'IT trainer',
       'Programmer, systems', 'Logistics and distribution manager',
       'Horticultural consultant', 'Hotel manager', 'Associate Professor',
       'Nurse, learning disability', 'Hydrographic surveyor',
       'Nurse, adult', 'Fisheries officer', 'Administrator, sports',
       'Insurance broker', 'Veterinary surgeon', 'Designer, jewellery',
       'Lobbyist', 'Chemical engineer', 'Chartered loss adjuster',
       'Social researcher', 'Petroleum engineer', 'Social worker',
       'Education officer, environmental', 'Futures trader',
       'Fish farm manager', 'Lawyer', 'Seismic interpreter',
       'TEFL teacher', 'Immunologist', 'Engineer, drilling',
       'Emergency planning/management officer', 'Pathologist',
       'Broadcast journalist', 'Geologist, wellsite',
       'Investment banker, operational', 'Biomedical scientist',
       'Bookseller', 'Copy', 'Midwife', 'Media buyer',
       'Geneticist, molecular', 'Housing manager/officer',
       'Geophysicist/field seismologist', 'Art gallery manager',
       'Food technologist', 'Land/geomatics surveyor',
       'Radio broadcast assistant',
       'Psychologist, prison and probation services', 'Dietitian',
       'Civil engineer, consulting', 'Sales executive',
       'Leisure centre manager', 'Scientist, biomedical',
       'Exhibitions officer, museum/gallery', 'Engineer, communications',
       'Catering manager', 'Administrator, arts', 'Software engineer',
       'Medical laboratory scientific officer', 'Commissioning editor',
       'Geoscientist', 'Materials engineer', 'Financial planner',
       'Brewing technologist', 'Minerals surveyor',
       'Editor, magazine features', 'General practice doctor',
       'Health and safety adviser', 'Doctor, hospital',
       'Environmental manager', 'Clinical research associate',
       'Sound technician, broadcasting/film/video', 'Press sub',
       'Retail buyer', 'Comptroller',
       'Government social research officer', 'Rural practice surveyor',
       'Accountant, chartered'], dtype=object)
array([ 1.,  3.,  0., nan,  7.,  2.,  4., 10.,  5.,  6.,  9.,  8.])
array([53., 51., 78., 22., 76., 50., 40., 48., 55., 64., 41., 45., 85.,
       nan, 44., 54., 72., 84., 52., 31., 75., 70., 56., 32., 86., 65.,
       79., 25., 58., 60., 33., 83., 66., 73., 43., 63., 57., 36., 39.,
       20., 47., 18., 59., 69., 82., 26., 34., 74., 37., 77., 27., 89.,
       30., 87., 23., 29., 80., 19., 67., 24., 88., 62., 49., 71., 21.,
       61., 81., 42., 35., 38., 28., 68., 46.])
```

```
array(['Some College, Less than 1 Year',
       'Some College, 1 or More Years, No Degree',
       'GED or Alternative Credential', 'Regular High School Diploma',
       "Bachelor's Degree", "Master's Degree",
       'Nursery School to 8th Grade',
       '9th Grade to 12th Grade, No Diploma', 'Doctorate Degree',
       "Associate's Degree", 'Professional School Degree',
       'No Schooling Completed'], dtype=object)
array(['Full Time', 'Retired', 'Unemployed', 'Student', 'Part Time'],
      dtype=object)
array([86575.93, 46805.99, 14370.14, ..., 65917.81, 29702.32, 62682.63])
array(['Divorced', 'Married', 'Widowed', 'Never Married', 'Separated'],
      dtype=object)
array(['Male', 'Female', 'Prefer not to answer'], dtype=object)
array(['No', 'Yes'], dtype=object)
array([17.80233049, 18.99463952, 17.4158887 , ..., 15.75275136,
       21.95630508, 20.42188348])
array([6, 4, 5, 7, 3, 2, 8, 9, 1], dtype=int64)
array([0, 2, 1, 3, 4, 5, 7, 6], dtype=int64)
array([0, 1, 2, 3, 4, 5], dtype=int64)
array([nan, 'No', 'Yes'], dtype=object)
array(['Emergency Admission', 'Elective Admission',
       'Observation Admission'], dtype=object)
array(['Yes', 'No'], dtype=object)
array(['No', 'Yes'], dtype=object)
array(['Medium', 'High', 'Low'], dtype=object)
array([ 0.,  1., nan])
array(['Yes', 'No'], dtype=object)
array(['Yes', 'No'], dtype=object)
array(['No', 'Yes'], dtype=object)
array(['Yes', 'No'], dtype=object)
array([ 1., nan,  0.])
array(['Yes', 'No'], dtype=object)
array(['No', 'Yes'], dtype=object)
array(['Yes', 'No'], dtype=object)
array(['Blood Work', 'Intravenous', 'CT Scan', 'MRI'], dtype=object)
array([10.58576971, 15.12956221,  4.77217721, ..., 68.66823748,
       63.35690285, 70.85059182])
array([3191.048774, 4214.905346, 2177.586768, ..., 7725.953391,
       8462.831883, 8700.856021])
array([17939.40342 , 17612.99812 , 17505.19246 , ..., 15281.21466 ,
        7781.678412, 11643.18993 ])
array([3, 2, 4, 1, 5, 7, 6, 8], dtype=int64)
array([3, 4, 5, 1, 2, 6, 7], dtype=int64)
array([2, 3, 4, 5, 1, 6, 7, 8], dtype=int64)
array([2, 4, 3, 5, 6, 1, 7], dtype=int64)
array([4, 3, 5, 2, 6, 1, 7], dtype=int64)
array([3, 4, 5, 2, 6, 1, 7], dtype=int64)
array([3, 5, 4, 2, 6, 1, 7], dtype=int64)
array([4, 3, 5, 6, 2, 1, 7], dtype=int64)
```

Data Cleaning - Part 2C

Identify columns that do not contain values for all 10000 rows in the data set

In [6]:
```
#Identify columns that do not contain values for all 10000 rows in the data se
t
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10000 entries, 1 to 10000
Data columns (total 52 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   CaseOrder           10000 non-null  int64  
 1   Customer_id         10000 non-null  object 
 2   Interaction         10000 non-null  object 
 3   UID                 10000 non-null  object 
 4   City                10000 non-null  object 
 5   State               10000 non-null  object 
 6   County              10000 non-null  object 
 7   Zip                 10000 non-null  int64  
 8   Lat                 10000 non-null  float64
 9   Lng                 10000 non-null  float64
 10  Population          10000 non-null  int64  
 11  Area                10000 non-null  object 
 12  Timezone            10000 non-null  object 
 13  Job                 10000 non-null  object 
 14  Children            7412 non-null   float64
 15  Age                 7586 non-null   float64
 16  Education           10000 non-null  object 
 17  Employment          10000 non-null  object 
 18  Income              7536 non-null   float64
 19  Marital             10000 non-null  object 
 20  Gender              10000 non-null  object 
 21  ReAdmis             10000 non-null  object 
 22  VitD_levels         10000 non-null  float64
 23  Doc_visits          10000 non-null  int64  
 24  Full_meals_eaten    10000 non-null  int64  
 25  VitD_supp           10000 non-null  int64  
 26  Soft_drink          7533 non-null   object 
 27  Initial_admin       10000 non-null  object 
 28  HighBlood           10000 non-null  object 
 29  Stroke              10000 non-null  object 
 30  Complication_risk   10000 non-null  object 
 31  Overweight          9018 non-null   float64
 32  Arthritis           10000 non-null  object 
 33  Diabetes            10000 non-null  object 
 34  Hyperlipidemia      10000 non-null  object 
 35  BackPain            10000 non-null  object 
 36  Anxiety             9016 non-null   float64
 37  Allergic_rhinitis   10000 non-null  object 
 38  Reflux_esophagitis  10000 non-null  object 
 39  Asthma              10000 non-null  object 
 40  Services            10000 non-null  object 
 41  Initial_days        8944 non-null   float64
 42  TotalCharge         10000 non-null  float64
 43  Additional_charges  10000 non-null  float64
 44  Item1               10000 non-null  int64  
 45  Item2               10000 non-null  int64  
 46  Item3               10000 non-null  int64  
 47  Item4               10000 non-null  int64  
 48  Item5               10000 non-null  int64  
 49  Item6               10000 non-null  int64  
 50  Item7               10000 non-null  int64  
 51  Item8               10000 non-null  int64  
```

```
dtypes: float64(11), int64(14), object(27)
memory usage: 4.0+ MB
None
```

i. Set index column for the dataframe

In [7]:
```
#Set index column for the dataframe
data.set_index('CaseOrder')

#ref: https://realpython.com/python-data-cleaning-numpy-pandas/
```

Out[7]:

CaseOrder	Customer_id	Interaction	UID	City	State	
1	C412403	8cd49b13-f45a-4b47-a2bd-173ffa932c2f	3a83ddb66e2ae73798bdf1d705dc0932	Eva	AL	
2	Z919181	d2450b70-0337-4406-bdbb-bc1037f1734c	176354c5eef714957d486009feabf195	Marianna	FL	
3	F995323	a2057123-abf5-4a2c-abad-8ffe33512562	e19a0fa00aeda885b8a436757e889bc9	Sioux Falls	SD	M
4	A879973	1dec528d-eb34-4079-adce-0d7a40e82205	cd17d7b6d152cb6f23957346d11c3f07	New Richland	MN	
5	C544523	5885f56b-d6da-43a3-8760-83583af94266	d2f0425877b10ed6bb381f3e2579424a	West Point	VA	Ki
...	...	...	...	...	...	
9996	B863060	a25b594d-0328-486f-a9b9-0567eb0f9723	39184dc28cc038871912ccc4500049e5	Norlina	NC	
9997	P712040	70711574-f7b1-4a17-b15f-48c54564b70f	3cd124ccd43147404292e883bf9ec55c	Milmay	NJ	
9998	R778890	1d79569d-8e0f-4180-a207-d67ee4527d26	41b770aeee97a5b9e7f69c906a8119d7	Southside	TN	Mc
9999	E344109	f5a68e69-2a60-409b-a92f-ac0847b27db0	2bb491ef5b1beb1fed758cc6885c167a	Quinn	SD	F
10000	I569847	bc482c02-f8c9-4423-99de-3db5e62a18d5	95663a202338000abdf7e09311c2a8a1	Coraopolis	PA	

10000 rows × 51 columns

ii. Update confusing or misleading column names

In [8]:
```
#Update confusing or misleading column names
data.rename(columns={'Item1': 'Survey_Timely_Admission',
                     'Item2': 'Survey_Timely_Treatment',
                     'Item3': 'Survey_Timely_Visit',
                     'Item4': 'Survey_Reliability',
                     'Item5': 'Survey_Options',
                     'Item6': 'Survey_Hours_of_Treatment',
                     'Item7': 'Survey_Courteous_Staff',
                     'Item8': 'Survey_Doctor_Active_Listening',
                    },
            inplace=True)

#Check for update column names
print(data.columns[-8:])
```

```
Index(['Survey_Timely_Admission', 'Survey_Timely_Treatment',
       'Survey_Timely_Visit', 'Survey_Reliability', 'Survey_Options',
       'Survey_Hours_of_Treatment', 'Survey_Courteous_Staff',
       'Survey_Doctor_Active_Listening'],
      dtype='object')
```

iii. Update nan and null values

In [9]:
```
#Replace null values
#7 of 52 columns do not contain 10,000 values: Children, Age, Income, Soft_dri
nk, Overweight, Anxiety, & Initial_days

#Children will be assigned a value of 0 with the assumption that a blank field
on the patient form is the equivalent of 0 children.
#Soft Drink and Anxiety will be assigned a "No" with the assumption that a bla
nk field on the patient form is the equivalent of "No."
data['Children'] = data['Children'].fillna(0)
data['Soft_drink'] = data['Soft_drink'].fillna('No')
data['Anxiety'] = data['Soft_drink'].fillna('No')

#Assign Imputer strategies to variables
from sklearn.impute import SimpleImputer
impNumeric = SimpleImputer(missing_values=np.nan, strategy='mean')
impCategorical = SimpleImputer(missing_values=np.nan,
                               strategy='most_frequent')

#A simple mean will be used to fill blank values for the Age, Income, and Init
ial Days columns.
impute_mean = data[['Age', 'Income', 'Initial_days']]
data[['Age', 'Income', 'Initial_days']] = impNumeric.fit(impute_mean).transfor
m(impute_mean)

#The Overweight blank fields will be filled with the most frequently occuring
 value. Since this is a binary column, a mean is not appropriate.
impute_cat = data[['Overweight']]
data[['Overweight']] = impCategorical.fit(impute_cat).transform(impute_cat)

data

#Ref: https://www.geeksforgeeks.org/replace-nan-values-with-zeros-in-pandas-da
taframe/
#Ref: https://towardsdatascience.com/a-brief-guide-to-data-imputation-with-pyt
hon-and-r-5dc551a95027
```

Out[9]:

	CaseOrder	Customer_id	Interaction	UID	City	S
1	1	C412403	8cd49b13-f45a-4b47-a2bd-173ffa932c2f	3a83ddb66e2ae73798bdf1d705dc0932	Eva	
2	2	Z919181	d2450b70-0337-4406-bdbb-bc1037f1734c	176354c5eef714957d486009feabf195	Marianna	
3	3	F995323	a2057123-abf5-4a2c-abad-8ffe33512562	e19a0fa00aeda885b8a436757e889bc9	Sioux Falls	
4	4	A879973	1dec528d-eb34-4079-adce-0d7a40e82205	cd17d7b6d152cb6f23957346d11c3f07	New Richland	
5	5	C544523	5885f56b-d6da-43a3-8760-83583af94266	d2f0425877b10ed6bb381f3e2579424a	West Point	
...	...	...	...	...	...	
9996	9996	B863060	a25b594d-0328-486f-a9b9-0567eb0f9723	39184dc28cc038871912ccc4500049e5	Norlina	
9997	9997	P712040	70711574-f7b1-4a17-b15f-48c54564b70f	3cd124ccd43147404292e883bf9ec55c	Milmay	
9998	9998	R778890	1d79569d-8e0f-4180-a207-d67ee4527d26	41b770aeee97a5b9e7f69c906a8119d7	Southside	
9999	9999	E344109	f5a68e69-2a60-409b-a92f-ac0847b27db0	2bb491ef5b1beb1fed758cc6885c167a	Quinn	
10000	10000	I569847	bc482c02-f8c9-4423-99de-3db5e62a18d5	95663a202338000abdf7e09311c2a8a1	Coraopolis	

10000 rows × 52 columns

Export copy of the clean data set

In [10]:
```
data.to_csv(r'C:\Users\JoshFunderburk\Desktop\D206 Project\medical_raw_data_cleaned-video.csv', index = False)
```

iv. Reexpress categorical data as numeric data

Reexpress State column as numerical

In [11]:
```
#Reexpress State column as numerical

#Identify unique values and sort in alphabetical order
print('Sorted unique values')
print(sorted(data['State'].unique()))

#Prepare to set State equal to State in Numerical form
data['State_Numeric'] = data['State']

#Assign numbers 1 to 52 to each state based on the alphabetical sort
state_dict = {'State_Numeric': { 'AK' : 1, 'AL' : 2, 'AR' : 3, 'AZ' : 4, 'CA'
: 5, 'CO' : 6, 'CT' : 7, 'DC' : 8, 'DE' : 9,
                                 'FL' : 10, 'GA' : 11, 'HI' : 12, 'IA' : 13, 'I
D' : 14, 'IL' : 15, 'IN' : 16, 'KS' : 17,
                                  'KY' : 18, 'LA' : 19, 'MA' : 20, 'MD' : 21,
'ME' : 22, 'MI' : 23, 'MN' : 24, 'MO' : 25,
                                  'MS' : 26, 'MT' : 27, 'NC' : 28, 'ND' : 29,
'NE' : 30, 'NH' : 31, 'NJ' : 32, 'NM' : 33,
                                  'NV' : 34, 'NY' : 35, 'OH' : 36, 'OK' : 37,
'OR' : 38, 'PA' : 39, 'PR' : 40, 'RI' : 41,
                                  'SC' : 42, 'SD' : 43, 'TN' : 44, 'TX' : 45,
'UT' : 46, 'VA' : 47, 'VT' : 48, 'WA' : 49,
                                 'WI' : 50,  'WV' : 51, 'WY' : 52}
             }

#Replace data
data.replace(state_dict, inplace = True)

#Verify new column
print('')
print('Sorted unique numerical values')
print(sorted(data['State_Numeric'].unique()))
```

```
Sorted unique values
['AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA', 'HI', 'I
A', 'ID', 'IL', 'IN', 'KS', 'KY', 'LA', 'MA', 'MD', 'ME', 'MI', 'MN', 'MO',
'MS', 'MT', 'NC', 'ND', 'NE', 'NH', 'NJ', 'NM', 'NV', 'NY', 'OH', 'OK', 'OR',
'PA', 'PR', 'RI', 'SC', 'SD', 'TN', 'TX', 'UT', 'VA', 'VT', 'WA', 'WI', 'WV',
'WY']

Sorted unique numerical values
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 2
2, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 4
1, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52]
```

Reexpress Area column as numerical

In [12]:
```
#Reexpress Area column as numerical

#Identify unique values and sort in alphabetical order
print('Sorted unique values')
print(sorted(data['Area'].unique()))

#Prepare to set Area equal to Area in Numerical form
data['Area_Numeric'] = data['Area']

#Assign numbers each Area based on the alphabetical sort
area_dict = {'Area_Numeric': { 'Rural' : 1, 'Suburban' : 2, 'Urban' : 3}
             }

#Replace data
data.replace(area_dict, inplace = True)

#Verify new column
print('')
print('Sorted unique numerical values')
print(sorted(data['Area_Numeric'].unique()))
```

```
Sorted unique values
['Rural', 'Suburban', 'Urban']

Sorted unique numerical values
[1, 2, 3]
```

Reexpress Timezone column as numerical

In [13]:
```
#Reexpress Timezone column as numerical

#Identify unique values and sort in alphabetical order
print('Sorted unique values')
print(sorted(data['Timezone'].unique()))

#Prepare to set Timezone equal to Timezone in Numerical form
data['Timezone_Numeric'] = data['Timezone']

#Assign numbers to each Timezone based on the alphabetical sort
timezone_dict = {'Timezone_Numeric': { 'America/Adak' : -10,'America/Anchorag
e' : -9,'America/Boise' : -7,
                                       'America/Chicago' : -6,'America/Denver'
: -7,'America/Detroit' : -5,
                                       'America/Indiana/Indianapolis' : -5,'Ame
rica/Indiana/Knox' : -6,
                                       'America/Indiana/Marengo' : -5,'America/
Indiana/Tell_City' : -6,
                                       'America/Indiana/Vevay' : -5,'America/In
diana/Vincennes' : -5,
                                       'America/Indiana/Winamac' : -5,'America/
Kentucky/Louisville' : -5,
                                       'America/Los_Angeles' : -8,'America/Meno
minee' : -6,'America/New_York' : -5,
                                       'America/Nome' : -9,'America/North_Dakot
a/Beulah' : -6,
                                       'America/North_Dakota/New_Salem' : -6,'A
merica/Phoenix' : -7,'America/Puerto_Rico' : -4,
                                       'America/Sitka' : -9,'America/Toronto' :
-5,'America/Yakutat' : -9,'Pacific/Honolulu' : -10}
                     }

#Replace data
data.replace(timezone_dict, inplace = True)

#Verify new column
print('')
print('Sorted unique numerical values')
print(sorted(data['Timezone_Numeric'].unique()))
```

```
Sorted unique values
['America/Adak', 'America/Anchorage', 'America/Boise', 'America/Chicago', 'Am
erica/Denver', 'America/Detroit', 'America/Indiana/Indianapolis', 'America/In
diana/Knox', 'America/Indiana/Marengo', 'America/Indiana/Tell_City', 'Americ
a/Indiana/Vevay', 'America/Indiana/Vincennes', 'America/Indiana/Winamac', 'Am
erica/Kentucky/Louisville', 'America/Los_Angeles', 'America/Menominee', 'Amer
ica/New_York', 'America/Nome', 'America/North_Dakota/Beulah', 'America/North_
Dakota/New_Salem', 'America/Phoenix', 'America/Puerto_Rico', 'America/Sitka',
'America/Toronto', 'America/Yakutat', 'Pacific/Honolulu']

Sorted unique numerical values
[-10, -9, -8, -7, -6, -5, -4]
```

Reexpress Education column as numerical

In [14]:
```
#Reexpress Education column as numerical

#Identify unique values and sort in alphabetical order
print('Sorted unique values')
print(sorted(data['Education'].unique()))

#Prepare to set Education equal to Education in Numerical form
data['Education_Numeric'] = data['Education']

#Assign numbers to each Education based on the alphabetical sort
education_dict = {'Education_Numeric': {  "No Schooling Completed" : 0,
                                          "Nursery School to 8th Grade" : 8,
                                          "9th Grade to 12th Grade, No Diploma"
: 10,
                                          "GED or Alternative Credential" : 12,
                                          "Regular High School Diploma" : 12,
                                          "Some College, Less than 1 Year" : 13
,
                                          "Some College, 1 or More Years, No De
gree" : 14,
                                          "Associate's Degree" : 14,
                                          "Bachelor's Degree" : 16,
                                          "Master's Degree" : 18,
                                          "Professional School Degree" : 18,
                                          "Doctorate Degree" : 20}
                  }

#Replace data
data.replace(education_dict, inplace = True)

#Verify new column
print('')
print('Sorted unique numerical values')
print(sorted(data['Education_Numeric'].unique()))
```

```
Sorted unique values
['9th Grade to 12th Grade, No Diploma', "Associate's Degree", "Bachelor's Deg
ree", 'Doctorate Degree', 'GED or Alternative Credential', "Master's Degree",
'No Schooling Completed', 'Nursery School to 8th Grade', 'Professional School
Degree', 'Regular High School Diploma', 'Some College, 1 or More Years, No De
gree', 'Some College, Less than 1 Year']

Sorted unique numerical values
[0, 8, 10, 12, 13, 14, 16, 18, 20]
```

Reexpress Employment column as numerical

In [15]:
```
#Reexpress Employment column as numerical

#Identify unique values and sort in alphabetical order
print('Sorted unique values')
print(sorted(data['Employment'].unique()))

#Prepare to set Employment equal to Employment in Numerical form
data['Employment_Numeric'] = data['Employment']

#Assign numbers to each Employment based on the alphabetical sort
employment_dict = {'Employment_Numeric': {  'Full Time' : 3, 'Part Time' : 2,
'Retired' : 4, 'Student' : 1, 'Unemployed' : 0}
                    }

#Replace data
data.replace(employment_dict, inplace = True)

#Verify new column
print('')
print('Sorted unique numerical values')
print(sorted(data['Employment_Numeric'].unique()))
```

```
Sorted unique values
['Full Time', 'Part Time', 'Retired', 'Student', 'Unemployed']

Sorted unique numerical values
[0, 1, 2, 3, 4]
```

Reexpress Marital column as numerical

In [16]:
```
#Reexpress Marital column as numerical

#Identify unique values and sort in alphabetical order
print('Sorted unique values')
print(sorted(data['Marital'].unique()))

#Prepare to set Marital equal to Marital in Numerical form
data['Marital_Numeric'] = data['Marital']

#Assign numbers to each Marital based on the alphabetical sort
marital_dict = {'Marital_Numeric': {  'Divorced' : 3,
                                      'Married' : 1,
                                      'Never Married' : 0,
                                      'Separated' : 2,
                                      'Widowed' : 4}
                    }

#Replace data
data.replace(marital_dict, inplace = True)

#Verify new column
print('')
print('Sorted unique numerical values')
print(sorted(data['Marital_Numeric'].unique()))
```

```
Sorted unique values
['Divorced', 'Married', 'Never Married', 'Separated', 'Widowed']

Sorted unique numerical values
[0, 1, 2, 3, 4]
```

Reexpress Gender column as numerical

In [17]:
```
#Reexpress Gender column as numerical

#Identify unique values and sort in alphabetical order
print('Sorted unique values')
print(sorted(data['Gender'].unique()))

#Prepare to set Gender equal to Gender in Numerical form
data['Gender_Numeric'] = data['Gender']

#Assign numbers to each Gender based on the alphabetical sort
gender_dict = {'Gender_Numeric': {  'Male' : 1, 'Female' : 2, 'Prefer not to a
nswer' : 0}
                    }

#Replace data
data.replace(gender_dict, inplace = True)

#Verify new column
print('')
print('Sorted unique numerical values')
print(sorted(data['Gender_Numeric'].unique()))
```

```
Sorted unique values
['Female', 'Male', 'Prefer not to answer']

Sorted unique numerical values
[0, 1, 2]
```

Reexpress ReAdmis column as numerical

In [18]:
```
#Reexpress ReAdmis column as numerical

#Identify unique values and sort in alphabetical order
print('Sorted unique values')
print(sorted(data['ReAdmis'].unique()))

#Prepare to set ReAdmis equal to ReAdmis in Numerical form
data['ReAdmis_Numeric'] = data['ReAdmis']

#Assign numbers to each ReAdmis based on the alphabetical sort
readmis_dict = {'ReAdmis_Numeric': {  'Yes' : 1, 'No' : 0}
                    }

#Replace data
data.replace(readmis_dict, inplace = True)

#Verify new column
print('')
print('Sorted unique numerical values')
print(sorted(data['ReAdmis_Numeric'].unique()))
```

```
Sorted unique values
['No', 'Yes']

Sorted unique numerical values
[0, 1]
```

Reexpress Soft_drink column as numerical

In [19]:
```
#Reexpress Soft_drink column as numerical

#Replace null values
data['Soft_drink'] = data['Soft_drink'].fillna('No')

#Identify unique values and sort in alphabetical order
print('Sorted unique values')
print(sorted(data['Soft_drink'].unique()))

#Prepare to set Soft_drink equal to Soft_drink in Numerical form
data['Soft_drink_Numeric'] = data['Soft_drink']

#Assign numbers to each Soft_drink based on the alphabetical sort
soft_drink_dict = {'Soft_drink_Numeric': {  'Yes' : 1, 'No' : 0, 'Unknown' : 2
}
                    }
#Replace data
data.replace(soft_drink_dict, inplace = True)

#Cast value from float to int
data['Soft_drink_Numeric'] = data['Soft_drink_Numeric'].astype(int)

#Verify new column
print('')
print('Sorted unique numerical values')
print(sorted(data['Soft_drink_Numeric'].unique()))
```

```
Sorted unique values
['No', 'Yes']

Sorted unique numerical values
[0, 1]
```

Reexpress Initial_admin column as numerical

In [20]:
```
#Reexpress Initial_admin column as numerical

#Identify unique values and sort in alphabetical order
print('Sorted unique values')
print(sorted(data['Initial_admin'].unique()))

#Prepare to set Initial_admin equal to Initial_admin in Numerical form
data['Initial_admin_Numeric'] = data['Initial_admin']

#Assign numbers to each Initial_admin based on the alphabetical sort
initial_admin_dict = {'Initial_admin_Numeric': {  'Elective Admission' : 1, 'E
mergency Admission' : 2, 'Observation Admission' : 3}
                    }

#Replace data
data.replace(initial_admin_dict, inplace = True)

#Verify new column
print('')
print('Sorted unique numerical values')
print(sorted(data['Initial_admin_Numeric'].unique()))
```

```
Sorted unique values
['Elective Admission', 'Emergency Admission', 'Observation Admission']

Sorted unique numerical values
[1, 2, 3]
```

Reexpress HighBlood column as numerical

In [21]:
```
#Reexpress HighBlood column as numerical

#Identify unique values and sort in alphabetical order
print('Sorted unique values')
print(sorted(data['HighBlood'].unique()))

#Prepare to set HighBlood equal to HighBlood in Numerical form
data['HighBlood_Numeric'] = data['HighBlood']

#Assign numbers to each HighBlood based on the alphabetical sort
highblood_dict = {'HighBlood_Numeric': {  'Yes' : 1, 'No' : 0}
                    }

#Replace data
data.replace(highblood_dict, inplace = True)

#Verify new column
print('')
print('Sorted unique numerical values')
print(sorted(data['HighBlood_Numeric'].unique()))
```

```
Sorted unique values
['No', 'Yes']

Sorted unique numerical values
[0, 1]
```

Reexpress Stroke column as numerical

In [22]:
```
#Reexpress Stroke column as numerical

#Identify unique values and sort in alphabetical order
print('Sorted unique values')
print(sorted(data['Stroke'].unique()))

#Prepare to set Stroke equal to Stroke in Numerical form
data['Stroke_Numeric'] = data['Stroke']

#Assign numbers to each Stroke based on the alphabetical sort
stroke_dict = {'Stroke_Numeric': {  'Yes' : 1, 'No' : 0}
                    }

#Replace data
data.replace(stroke_dict, inplace = True)

#Verify new column
print('')
print('Sorted unique numerical values')
print(sorted(data['Stroke_Numeric'].unique()))
```

```
Sorted unique values
['No', 'Yes']

Sorted unique numerical values
[0, 1]
```

Reexpress Complication_risk column as numerical

In [23]:
```
#Reexpress Complication_risk column as numerical

#Identify unique values and sort in alphabetical order
print('Sorted unique values')
print(sorted(data['Complication_risk'].unique()))

#Prepare to set Complication_risk equal to Complication_risk in Numerical form
data['Complication_risk_Numeric'] = data['Complication_risk']

#Assign numbers to each Complication_risk based on the alphabetical sort
complication_risk_dict = {'Complication_risk_Numeric': {  'Low' : 1, 'Medium'
: 2, 'High' : 3}
                    }

#Replace data
data.replace(complication_risk_dict, inplace = True)

#Verify new column
print('')
print('Sorted unique numerical values')
print(sorted(data['Complication_risk_Numeric'].unique()))
```

```
Sorted unique values
['High', 'Low', 'Medium']

Sorted unique numerical values
[1, 2, 3]
```

Reexpress Arthritis column as numerical

In [24]:
```
#Reexpress Arthritis column as numerical

#Identify unique values and sort in alphabetical order
print('Sorted unique values')
print(sorted(data['Arthritis'].unique()))

#Prepare to set Arthritis equal to Arthritis in Numerical form
data['Arthritis_Numeric'] = data['Arthritis']

#Assign numbers to each Arthritis based on the alphabetical sort
arthritis_dict = {'Arthritis_Numeric': {  'Yes' : 1, 'No' : 0}
                    }

#Replace data
data.replace(arthritis_dict, inplace = True)

#Verify new column
print('')
print('Sorted unique numerical values')
print(sorted(data['Arthritis_Numeric'].unique()))
```

```
Sorted unique values
['No', 'Yes']

Sorted unique numerical values
[0, 1]
```

Reexpress Diabetes column as numerical

In [25]:
```
#Reexpress Diabetes column as numerical

#Identify unique values and sort in alphabetical order
print('Sorted unique values')
print(sorted(data['Diabetes'].unique()))

#Prepare to set Diabetes equal to Diabetes in Numerical form
data['Diabetes_Numeric'] = data['Diabetes']

#Assign numbers to each Diabetes based on the alphabetical sort
diabetes_dict = {'Diabetes_Numeric': {  'Yes' : 1, 'No' : 0}
                    }

#Replace data
data.replace(diabetes_dict, inplace = True)

#Verify new column
print('')
print('Sorted unique numerical values')
print(sorted(data['Diabetes_Numeric'].unique()))
```

```
Sorted unique values
['No', 'Yes']

Sorted unique numerical values
[0, 1]
```

Reexpress Hyperlipidemia column as numerical

In [26]:
```
#Reexpress Hyperlipidemia column as numerical

#Identify unique values and sort in alphabetical order
print('Sorted unique values')
print(sorted(data['Hyperlipidemia'].unique()))

#Prepare to set Hyperlipidemia equal to Hyperlipidemia in Numerical form
data['Hyperlipidemia_Numeric'] = data['Hyperlipidemia']

#Assign numbers to each Hyperlipidemia based on the alphabetical sort
hyperlipidemia_dict = {'Hyperlipidemia_Numeric': {  'Yes' : 1, 'No' : 0}
                     }

#Replace data
data.replace(hyperlipidemia_dict, inplace = True)

#Verify new column
print('')
print('Sorted unique numerical values')
print(sorted(data['Hyperlipidemia_Numeric'].unique()))
```

```
Sorted unique values
['No', 'Yes']

Sorted unique numerical values
[0, 1]
```

Reexpress BackPain column as numerical

In [27]:
```
#Reexpress BackPain column as numerical

#Identify unique values and sort in alphabetical order
print('Sorted unique values')
print(sorted(data['BackPain'].unique()))

#Prepare to set BackPain equal to BackPain in Numerical form
data['BackPain_Numeric'] = data['BackPain']

#Assign numbers to each BackPain based on the alphabetical sort
backpain_dict = {'BackPain_Numeric': {  'Yes' : 1, 'No' : 0}
                    }

#Replace data
data.replace(backpain_dict, inplace = True)

#Verify new column
print('')
print('Sorted unique numerical values')
print(sorted(data['BackPain_Numeric'].unique()))
```

```
Sorted unique values
['No', 'Yes']

Sorted unique numerical values
[0, 1]
```

Reexpress Allergic_rhinitis column as numerical

In [28]:
```
#Reexpress Allergic_rhinitis column as numerical

#Identify unique values and sort in alphabetical order
print('Sorted unique values')
print(sorted(data['Allergic_rhinitis'].unique()))

#Prepare to set Allergic_rhinitis equal to Allergic_rhinitis in Numerical form
data['Allergic_rhinitis_Numeric'] = data['Allergic_rhinitis']

#Assign numbers to each Allergic_rhinitis based on the alphabetical sort
allergic_rhinitis_dict = {'Allergic_rhinitis_Numeric': {  'Yes' : 1, 'No' : 0}
                     }

#Replace data
data.replace(allergic_rhinitis_dict, inplace = True)

#Verify new column
print('')
print('Sorted unique numerical values')
print(sorted(data['Allergic_rhinitis_Numeric'].unique()))
```

```
Sorted unique values
['No', 'Yes']

Sorted unique numerical values
[0, 1]
```

Reexpress Reflux_esophagitis column as numerical

In [29]:
```
#Reexpress Reflux_esophagitis column as numerical

#Identify unique values and sort in alphabetical order
print('Sorted unique values')
print(sorted(data['Reflux_esophagitis'].unique()))

#Prepare to set Reflux_esophagitis equal to Reflux_esophagitis in Numerical fo
rm
data['Reflux_esophagitis_Numeric'] = data['Reflux_esophagitis']

#Assign numbers to each Reflux_esophagitis based on the alphabetical sort
reflux_esophagitis_dict = {'Reflux_esophagitis_Numeric': {  'Yes' : 1, 'No' :
0}
                    }

#Replace data
data.replace(reflux_esophagitis_dict, inplace = True)

#Verify new column
print('')
print('Sorted unique numerical values')
print(sorted(data['Reflux_esophagitis_Numeric'].unique()))
```

```
Sorted unique values
['No', 'Yes']

Sorted unique numerical values
[0, 1]
```

Reexpress Asthma column as numerical

In [30]:
```
#Reexpress Asthma column as numerical

#Identify unique values and sort in alphabetical order
print('Sorted unique values')
print(sorted(data['Asthma'].unique()))

#Prepare to set Asthma equal to Asthma in Numerical form
data['Asthma_Numeric'] = data['Asthma']

#Assign numbers to each Asthma based on the alphabetical sort
asthma_dict = {'Asthma_Numeric': {  'Yes' : 1, 'No' : 0}
                    }

#Replace data
data.replace(asthma_dict, inplace = True)

#Verify new column
print('')
print('Sorted unique numerical values')
print(sorted(data['Asthma_Numeric'].unique()))
```

```
Sorted unique values
['No', 'Yes']

Sorted unique numerical values
[0, 1]
```

Reexpress Services column as numerical

In [31]:
```
#Reexpress Services column as numerical

#Identify unique values and sort in alphabetical order
print('Sorted unique values')
print(sorted(data['Services'].unique()))

#Prepare to set Services equal to Services in Numerical form
data['Services_Numeric'] = data['Services']

#Assign numbers to each Services based on the alphabetical sort
services_dict = {'Services_Numeric': {  'Blood Work' : 1, 'Intravenous' : 2,
'CT Scan' : 3, 'MRI' : 4}
                    }

#Replace data
data.replace(services_dict, inplace = True)

#Verify new column
print('')
print('Sorted unique numerical values')
print(sorted(data['Services_Numeric'].unique()))
```

```
Sorted unique values
['Blood Work', 'CT Scan', 'Intravenous', 'MRI']

Sorted unique numerical values
[1, 2, 3, 4]
```

v. Identify Outliers

In [32]:
```
#Create new dataframe focusing on numerical columns
df = data[['ReAdmis_Numeric',
        'City',
        'County',
        'State_Numeric',
        'Zip',
        'Lat',
        'Lng',
        'Population',
        'Area_Numeric',
        'Timezone_Numeric',
        'Gender_Numeric',
        'Age',
        'Education_Numeric',
        'Employment_Numeric',
        'Income',
        'Marital_Numeric',
        'Children',
        'Initial_days',
        'TotalCharge',
        'Additional_charges',
        'Doc_visits',
        'Initial_admin_Numeric',
        'Full_meals_eaten',
        'VitD_levels',
        'VitD_supp',
        'Soft_drink_Numeric',
        'HighBlood_Numeric',
        'Stroke_Numeric',
        'Complication_risk_Numeric',
        'Overweight',
        'Arthritis_Numeric',
        'Diabetes_Numeric',
        'Hyperlipidemia_Numeric',
        'BackPain_Numeric',
        'Allergic_rhinitis_Numeric',
        'Reflux_esophagitis_Numeric',
        'Asthma_Numeric',
        'Services_Numeric',
        'Survey_Timely_Admission',
        'Survey_Timely_Treatment',
        'Survey_Timely_Visit',
        'Survey_Reliability',
        'Survey_Options',
        'Survey_Hours_of_Treatment',
        'Survey_Courteous_Staff',
        'Survey_Doctor_Active_Listening']]

df
```

Out[32]:

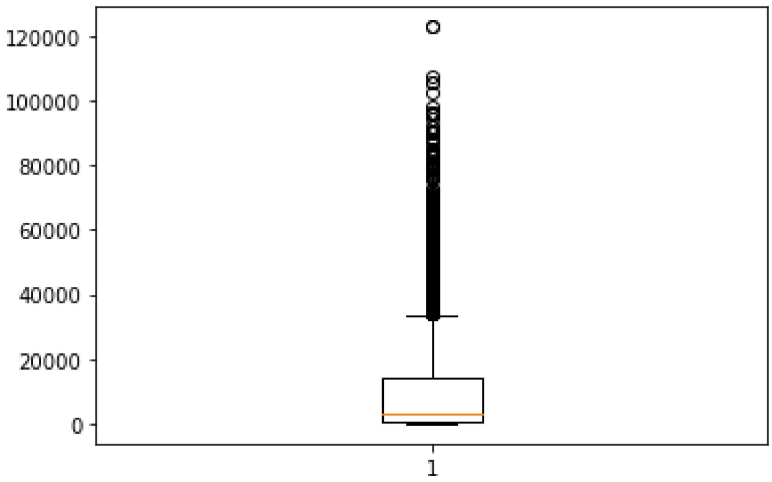
	ReAdmis_Numeric	City	County	State_Numeric	Zip	Lat	Lng	P
1	0	Eva	Morgan	2	35621	34.34960	-86.72508	
2	0	Marianna	Jackson	10	32446	30.84513	-85.22907	
3	0	Sioux Falls	Minnehaha	43	57110	43.54321	-96.63772	
4	0	New Richland	Waseca	24	56072	43.89744	-93.51479	
5	0	West Point	King William	47	23181	37.59894	-76.88958	
...	...	...	...	...	...	...	...	
9996	0	Norlina	Warren	28	27563	36.42886	-78.23716	
9997	1	Milmay	Atlantic	32	8340	39.43609	-74.87302	
9998	1	Southside	Montgomery	44	37171	36.36655	-87.29988	
9999	1	Quinn	Pennington	43	57775	44.10354	-102.01593	
10000	1	Coraopolis	Allegheny	39	15108	40.49998	-80.19959	

10000 rows × 46 columns

Population Boxplot

In [33]:
```
plt.boxplot(df.Population)
```

Out[33]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x285016de0c8>,
  <matplotlib.lines.Line2D at 0x2850161ee88>],
 'caps': [<matplotlib.lines.Line2D at 0x2850160cc08>,
  <matplotlib.lines.Line2D at 0x2850160cd48>],
 'boxes': [<matplotlib.lines.Line2D at 0x2850161ec48>],
 'medians': [<matplotlib.lines.Line2D at 0x2850160cdc8>],
 'fliers': [<matplotlib.lines.Line2D at 0x2850161e348>],
 'means': []}
```

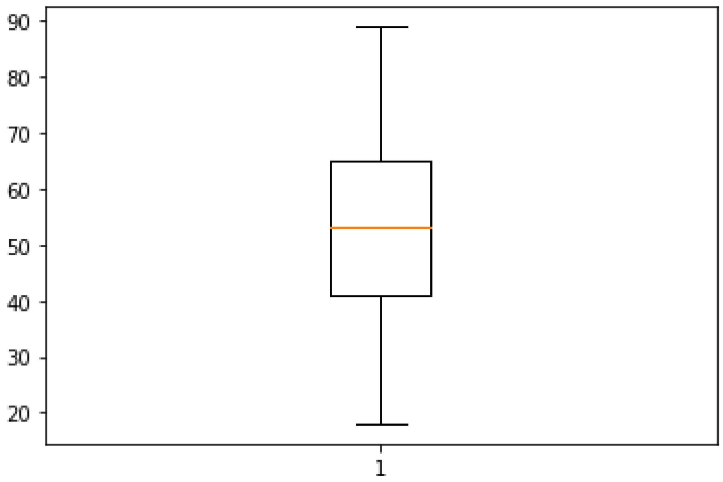


The population variable has several outliers and must be stanardized.

Age Boxplot

In [34]:
```
plt.boxplot(df.Age)
```

Out[34]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x2850164d308>,
  <matplotlib.lines.Line2D at 0x2850165e288>],
 'caps': [<matplotlib.lines.Line2D at 0x2850165aac8>,
  <matplotlib.lines.Line2D at 0x2850165afc8>],
 'boxes': [<matplotlib.lines.Line2D at 0x2850165e4c8>],
 'medians': [<matplotlib.lines.Line2D at 0x2850165acc8>],
 'fliers': [<matplotlib.lines.Line2D at 0x28501726a48>],
 'means': []}
```

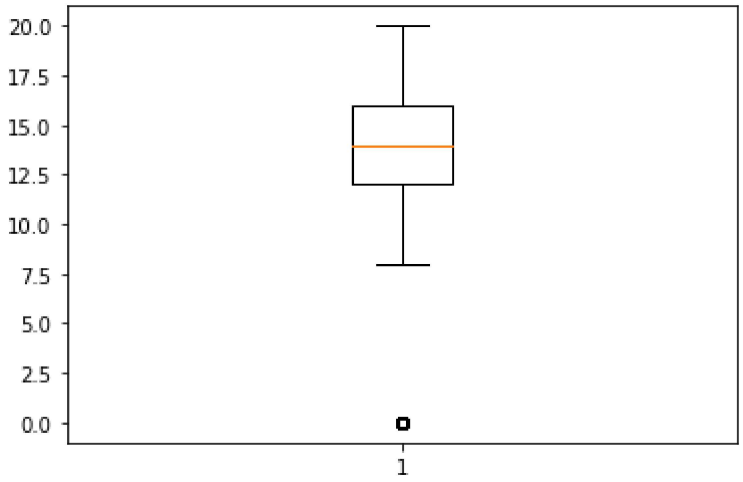


The age variable has no outliers.

Education Boxplot

In [35]:
```
plt.boxplot(df.Education_Numeric)
```

Out[35]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x2850168a208>,
  <matplotlib.lines.Line2D at 0x28501689448>],
 'caps': [<matplotlib.lines.Line2D at 0x28501689308>,
  <matplotlib.lines.Line2D at 0x28501682608>],
 'boxes': [<matplotlib.lines.Line2D at 0x2850168a3c8>],
 'medians': [<matplotlib.lines.Line2D at 0x285016824c8>],
 'fliers': [<matplotlib.lines.Line2D at 0x28501c22888>],
 'means': []}
```

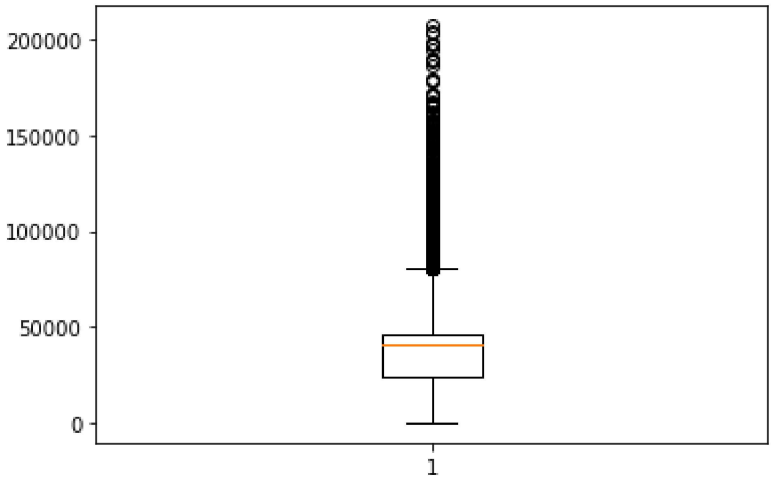


Education has one extreme outlier for those with no education. Considering this outlier and due to the fact that assigning values to the Education categories is highly subjective, this variable will be standardized.

Income Boxplot

In [36]:
```
plt.boxplot(df.Income)
```

Out[36]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x28501c8fe08>,
  <matplotlib.lines.Line2D at 0x28501c8ff48>],
 'caps': [<matplotlib.lines.Line2D at 0x28501c93bc8>,
  <matplotlib.lines.Line2D at 0x28501c93d08>],
 'boxes': [<matplotlib.lines.Line2D at 0x28501c8f688>],
 'medians': [<matplotlib.lines.Line2D at 0x28501c97a48>],
 'fliers': [<matplotlib.lines.Line2D at 0x28501c97b88>],
 'means': []}
```

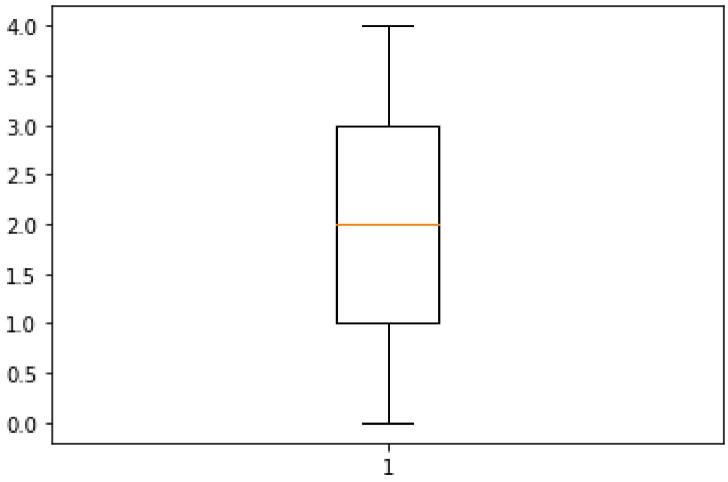


The income variable has several outliers and must be standarized.

Marital Boxplot

In [37]:
```
plt.boxplot(df.Marital_Numeric)
```

Out[37]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x28501cf7f88>,
  <matplotlib.lines.Line2D at 0x28501cfc688>],
 'caps': [<matplotlib.lines.Line2D at 0x28501cfcd48>,
  <matplotlib.lines.Line2D at 0x28501cfce88>],
 'boxes': [<matplotlib.lines.Line2D at 0x28501cf7808>],
 'medians': [<matplotlib.lines.Line2D at 0x28501d00bc8>],
 'fliers': [<matplotlib.lines.Line2D at 0x28501d00d08>],
 'means': []}
```

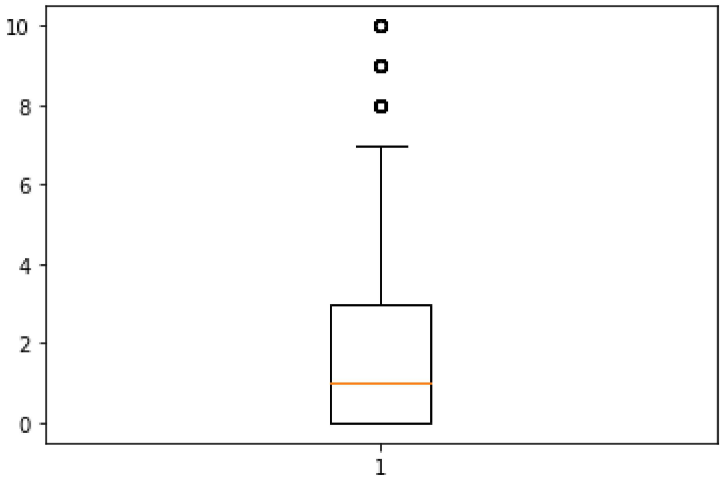


The marital variable has no outliers.

Children Boxplot

In [38]:
```
plt.boxplot(df.Children)
```

Out[38]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x28501d6fc08>,
  <matplotlib.lines.Line2D at 0x28501d6fd48>],
 'caps': [<matplotlib.lines.Line2D at 0x28501d72a08>,
  <matplotlib.lines.Line2D at 0x28501d72b48>],
 'boxes': [<matplotlib.lines.Line2D at 0x28501d6f388>],
 'medians': [<matplotlib.lines.Line2D at 0x28501d75888>],
 'fliers': [<matplotlib.lines.Line2D at 0x28501d75f88>],
 'means': []}
```

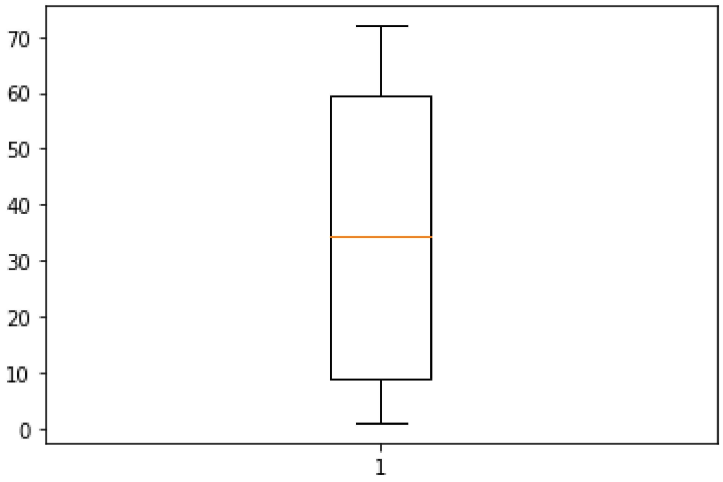


The children variable has 3 outliers. It will be standardized to ensure model effectiveness.

Initial_days Boxplot

In [39]:
```
plt.boxplot(df.Initial_days)
```

Out[39]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x28501ddc7c8>,
  <matplotlib.lines.Line2D at 0x28501ddce88>],
 'caps': [<matplotlib.lines.Line2D at 0x28501ddcfc8>,
  <matplotlib.lines.Line2D at 0x28501ddfcc8>],
 'boxes': [<matplotlib.lines.Line2D at 0x28501dd4f08>],
 'medians': [<matplotlib.lines.Line2D at 0x28501ddfe08>],
 'fliers': [<matplotlib.lines.Line2D at 0x28501de5b48>],
 'means': []}
```

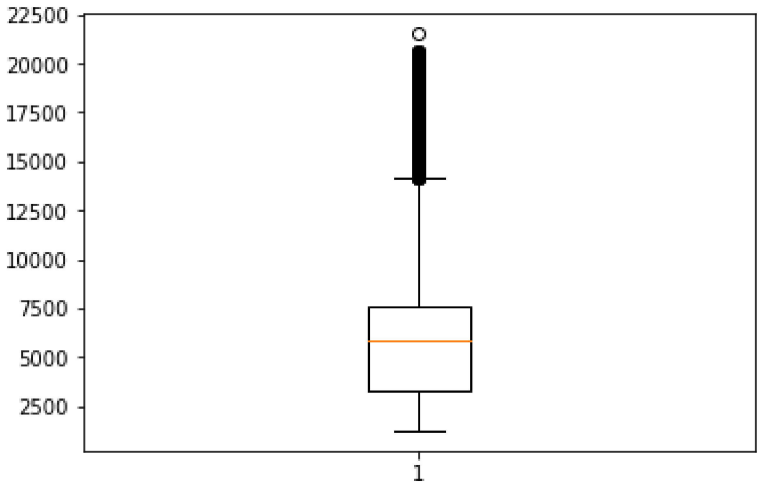


The Initial_days variable has no outliers.

TotalCharge Boxplot

In [40]:
```
plt.boxplot(df.TotalCharge)
```

Out[40]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x28501c42108>,
  <matplotlib.lines.Line2D at 0x28501c51588>],
 'caps': [<matplotlib.lines.Line2D at 0x28501c9e6c8>,
  <matplotlib.lines.Line2D at 0x28501717bc8>],
 'boxes': [<matplotlib.lines.Line2D at 0x28501c42888>],
 'medians': [<matplotlib.lines.Line2D at 0x28501c62cc8>],
 'fliers': [<matplotlib.lines.Line2D at 0x28501cb3788>],
 'means': []}
```

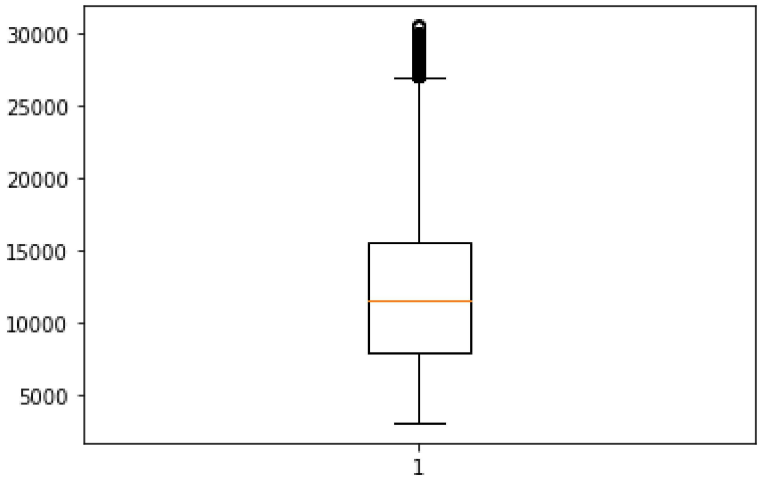


The TotalCharge variable has several outliers and will be standardized. One caveat to this variable is that it is not specified whether the data includes charges for readmissions.

Additional_charges Boxplot

In [41]:
```
plt.boxplot(df.Additional_charges)
```

Out[41]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x28501e9ce88>,
  <matplotlib.lines.Line2D at 0x28501ea0c48>],
 'caps': [<matplotlib.lines.Line2D at 0x28501ea0d88>,
  <matplotlib.lines.Line2D at 0x28501ea5a88>],
 'boxes': [<matplotlib.lines.Line2D at 0x28501e9ccc8>],
 'medians': [<matplotlib.lines.Line2D at 0x28501ea5bc8>],
 'fliers': [<matplotlib.lines.Line2D at 0x28501eab908>],
 'means': []}
```

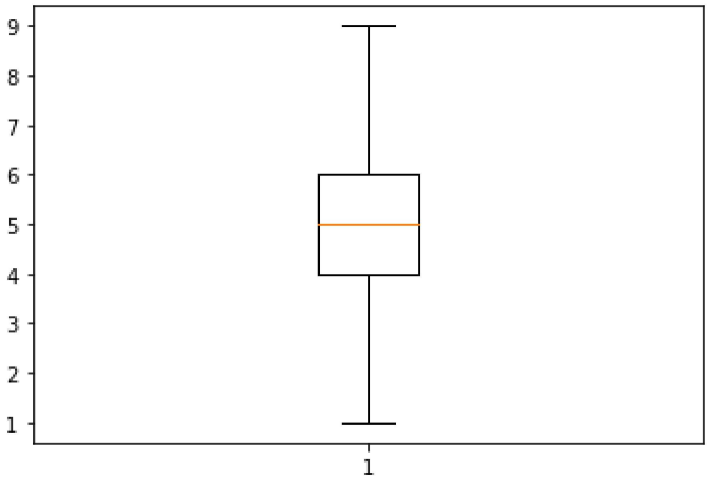


The Additional_charges variable has several outliers and will be standardized.

Doc_visits Boxplot

In [42]:
```
plt.boxplot(df.Doc_visits)
```

Out[42]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x28501f08a08>,
  <matplotlib.lines.Line2D at 0x28501f0e7c8>],
 'caps': [<matplotlib.lines.Line2D at 0x28501f0eec8>,
  <matplotlib.lines.Line2D at 0x28501d7e888>],
 'boxes': [<matplotlib.lines.Line2D at 0x28501f08848>],
 'medians': [<matplotlib.lines.Line2D at 0x28501f13c48>],
 'fliers': [<matplotlib.lines.Line2D at 0x28501f13d88>],
 'means': []}
```

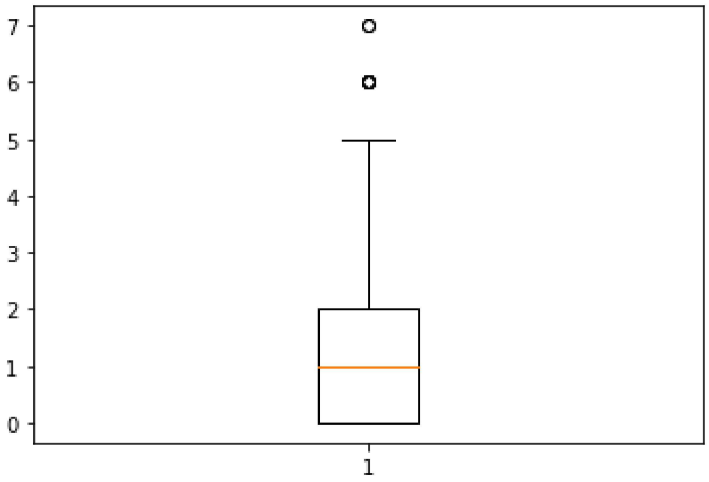


The Doc_visits variable has no outliers.

Full_meals_eaten Boxplot

In [43]:
```
plt.boxplot(df.Full_meals_eaten)
```

Out[43]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x28501f7eb08>,
  <matplotlib.lines.Line2D at 0x28501f7ec48>],
 'caps': [<matplotlib.lines.Line2D at 0x28501f84908>,
  <matplotlib.lines.Line2D at 0x28501f84a48>],
 'boxes': [<matplotlib.lines.Line2D at 0x28501f7e288>],
 'medians': [<matplotlib.lines.Line2D at 0x28501f8a788>],
 'fliers': [<matplotlib.lines.Line2D at 0x28501f8ae88>],
 'means': []}
```

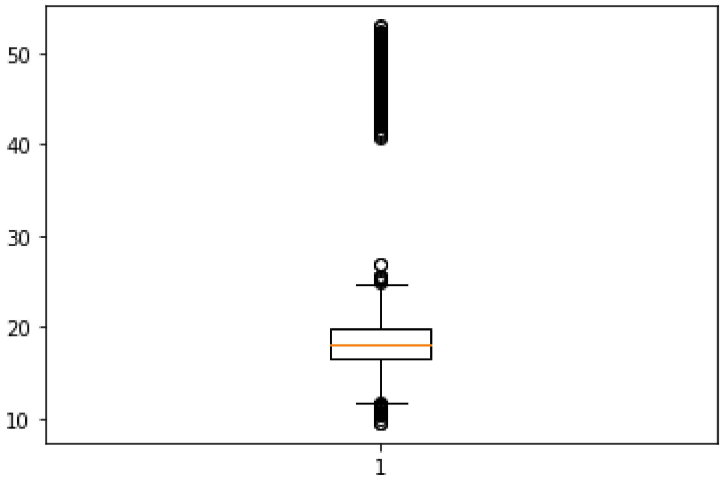


The Full_meals_eaten variable has two outliers. The variable will be standardized.

VitD_levels Boxplot

In [44]:
```
plt.boxplot(df.VitD_levels)
```

Out[44]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x28501ff3988>,
  <matplotlib.lines.Line2D at 0x28501ff3f88>],
 'caps': [<matplotlib.lines.Line2D at 0x28501ff7688>,
  <matplotlib.lines.Line2D at 0x28501ff7d88>],
 'boxes': [<matplotlib.lines.Line2D at 0x28501fedfc8>],
 'medians': [<matplotlib.lines.Line2D at 0x28501ff7f08>],
 'fliers': [<matplotlib.lines.Line2D at 0x28501ffdd88>],
 'means': []}
```

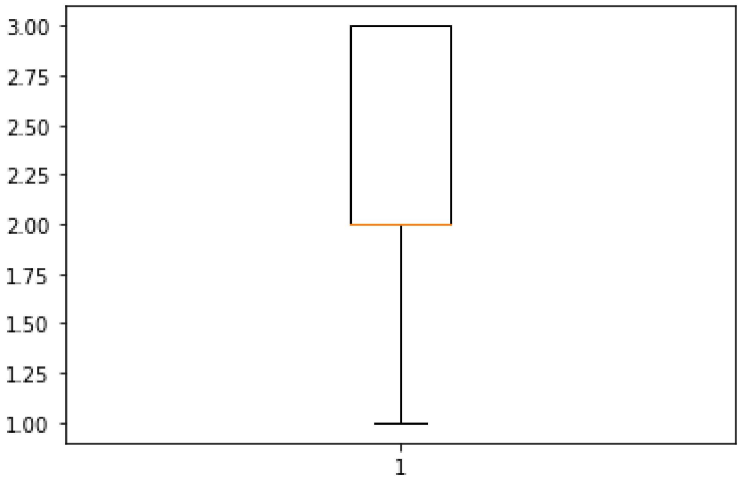


The VitD_levels variables has several outliers on both sides of the mean. The variable will be standardized.

Complication_risk Boxplot

In [45]:
```
plt.boxplot(df.Complication_risk_Numeric)
```

Out[45]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x2850205acc8>,
  <matplotlib.lines.Line2D at 0x2850205ae08>],
 'caps': [<matplotlib.lines.Line2D at 0x2850205fac8>,
  <matplotlib.lines.Line2D at 0x2850205fc08>],
 'boxes': [<matplotlib.lines.Line2D at 0x2850205a448>],
 'medians': [<matplotlib.lines.Line2D at 0x28502065948>],
 'fliers': [<matplotlib.lines.Line2D at 0x28502065a88>],
 'means': []}
```

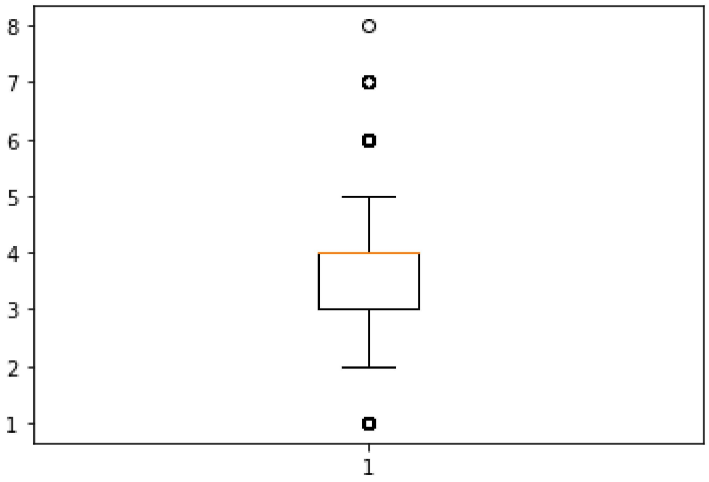


The Complication_risk variable has no outliers.

Survey_Timely_Admission Boxplot

In [46]:
```
plt.boxplot(df.Survey_Timely_Admission)
```

Out[46]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x285020d1a88>,
  <matplotlib.lines.Line2D at 0x285020d1bc8>],
 'caps': [<matplotlib.lines.Line2D at 0x285020d4888>,
  <matplotlib.lines.Line2D at 0x285020d4f88>],
 'boxes': [<matplotlib.lines.Line2D at 0x285020d1208>],
 'medians': [<matplotlib.lines.Line2D at 0x285020db708>],
 'fliers': [<matplotlib.lines.Line2D at 0x285020dbe08>],
 'means': []}
```

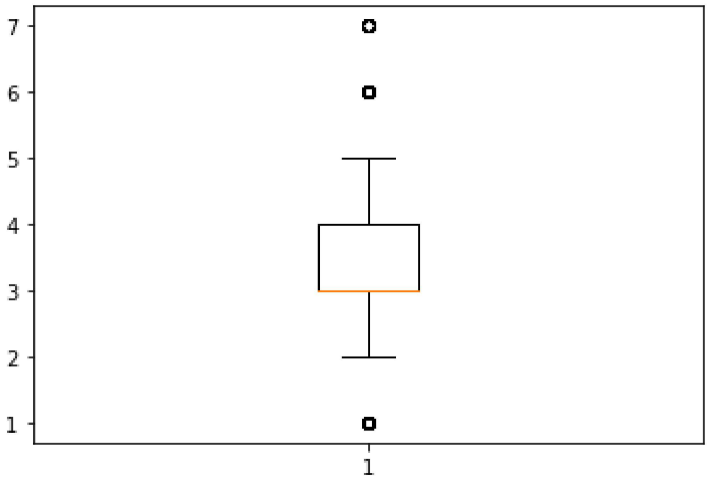


This survey question has a few outliers. Whether this will be standardized will be determined after all survey boxplots are analyzed.

Survey_Timely_Treatment Boxplot

In [47]:
```
plt.boxplot(df.Survey_Timely_Treatment)
```

Out[47]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x28502142708>,
  <matplotlib.lines.Line2D at 0x28502142dc8>],
 'caps': [<matplotlib.lines.Line2D at 0x28502142f08>,
  <matplotlib.lines.Line2D at 0x28502148c08>],
 'boxes': [<matplotlib.lines.Line2D at 0x2850213be48>],
 'medians': [<matplotlib.lines.Line2D at 0x28502148d88>],
 'fliers': [<matplotlib.lines.Line2D at 0x2850214cc08>],
 'means': []}
```

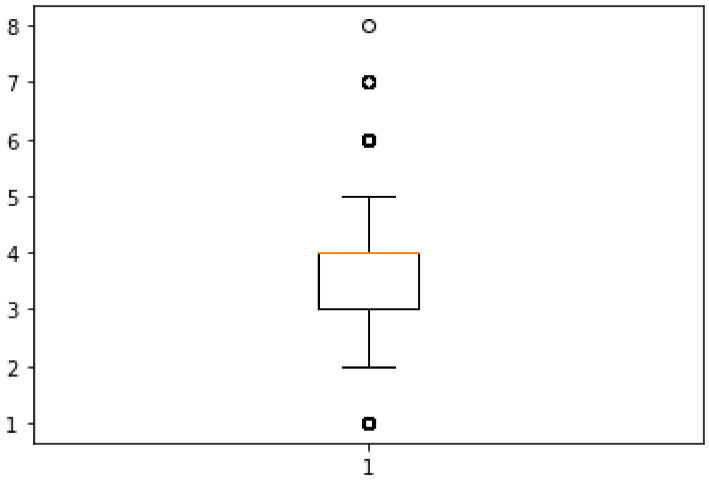


This survey question has a few outliers. Whether this will be standardized will be determined after all survey boxplots are analyzed.

Survey_Timely_Visit Boxplot

In [48]:
```
plt.boxplot(df.Survey_Timely_Visit)
```

Out[48]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x28502471c08>,
  <matplotlib.lines.Line2D at 0x28502471d48>],
 'caps': [<matplotlib.lines.Line2D at 0x28502474a08>,
  <matplotlib.lines.Line2D at 0x28502474b48>],
 'boxes': [<matplotlib.lines.Line2D at 0x285024713c8>],
 'medians': [<matplotlib.lines.Line2D at 0x285024798c8>],
 'fliers': [<matplotlib.lines.Line2D at 0x28502479a48>],
 'means': []}
```

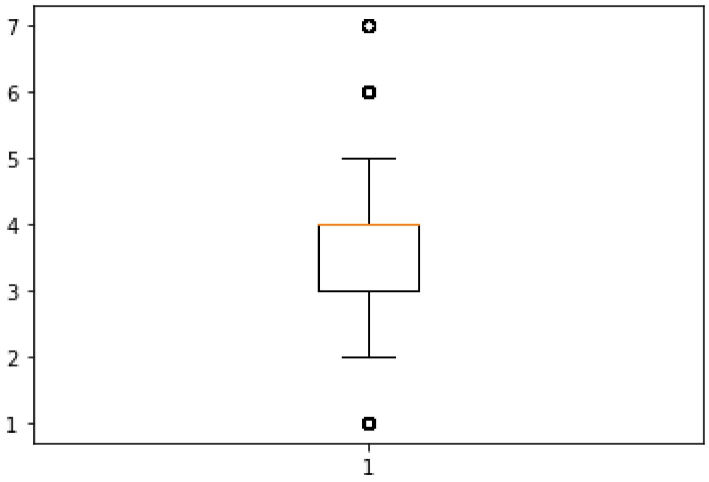


This survey question has a few outliers. Whether this will be standardized will be determined after all survey boxplots are analyzed.

Survey_Reliability Boxplot

In [49]:
```
plt.boxplot(df.Survey_Reliability)
```

Out[49]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x285024e3f88>,
  <matplotlib.lines.Line2D at 0x285024e5688>],
 'caps': [<matplotlib.lines.Line2D at 0x285024e5d48>,
  <matplotlib.lines.Line2D at 0x285024e5e88>],
 'boxes': [<matplotlib.lines.Line2D at 0x285024e3808>],
 'medians': [<matplotlib.lines.Line2D at 0x285024ebbc8>],
 'fliers': [<matplotlib.lines.Line2D at 0x285024ebd08>],
 'means': []}
```

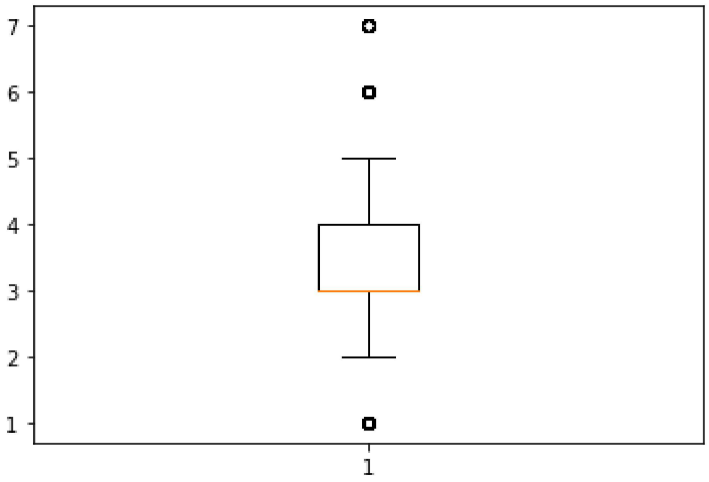


This survey question has a few outliers. Whether this will be standardized will be determined after all survey boxplots are analyzed.

Survey_Options Boxplot

In [50]:
```
plt.boxplot(df.Survey_Options)
```

Out[50]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x2850254dc48>,
  <matplotlib.lines.Line2D at 0x28502551a08>],
 'caps': [<matplotlib.lines.Line2D at 0x28502551b48>,
  <matplotlib.lines.Line2D at 0x28502559848>],
 'boxes': [<matplotlib.lines.Line2D at 0x2850254da88>],
 'medians': [<matplotlib.lines.Line2D at 0x28502559f88>],
 'fliers': [<matplotlib.lines.Line2D at 0x2850255f6c8>],
 'means': []}
```

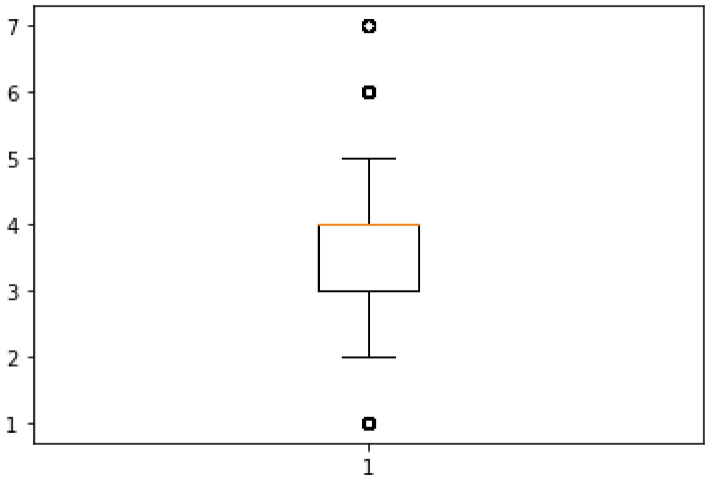


This survey question has a few outliers. Whether this will be standardized will be determined after all survey boxplots are analyzed.

Survey_Hours_of_Treatment Boxplot

In [51]:
```
plt.boxplot(df.Survey_Hours_of_Treatment)
```

Out[51]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x285025c2788>,
  <matplotlib.lines.Line2D at 0x285025c2e48>],
 'caps': [<matplotlib.lines.Line2D at 0x285025c2f88>,
  <matplotlib.lines.Line2D at 0x285025c6c08>],
 'boxes': [<matplotlib.lines.Line2D at 0x285025bcf88>],
 'medians': [<matplotlib.lines.Line2D at 0x285025c6d48>],
 'fliers': [<matplotlib.lines.Line2D at 0x285025cbac8>],
 'means': []}
```

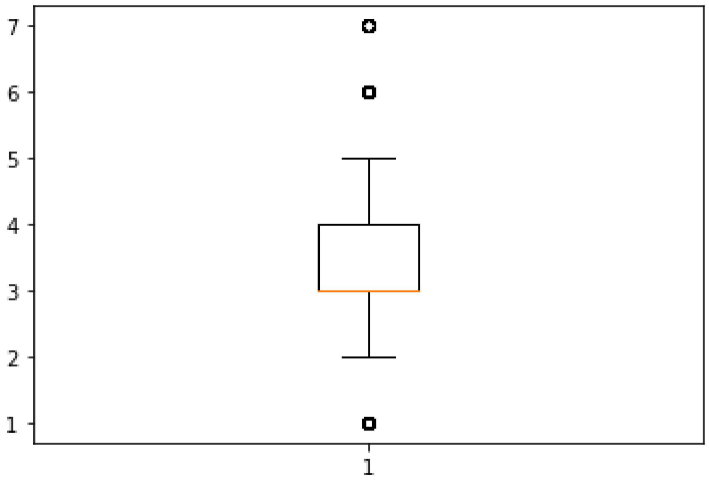


This survey question has a few outliers. Whether this will be standardized will be determined after all survey boxplots are analyzed.

Survey_Courteous_Staff Boxplot

In [52]:
```
plt.boxplot(df.Survey_Courteous_Staff)
```

Out[52]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x2850261cdc8>,
  <matplotlib.lines.Line2D at 0x285025ffdc8>],
 'caps': [<matplotlib.lines.Line2D at 0x28502632a88>,
  <matplotlib.lines.Line2D at 0x28502632bc8>],
 'boxes': [<matplotlib.lines.Line2D at 0x285025ff4c8>],
 'medians': [<matplotlib.lines.Line2D at 0x28502636888>],
 'fliers': [<matplotlib.lines.Line2D at 0x28502636f88>],
 'means': []}
```

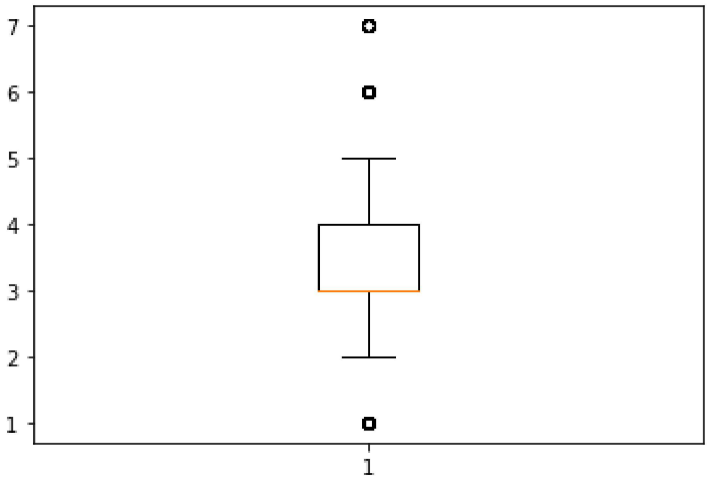


This survey question has a few outliers. Whether this will be standardized will be determined after all survey boxplots are analyzed.

Survey_Doctor_Active_Listening Boxplot

In [53]:
```
plt.boxplot(df.Survey_Doctor_Active_Listening)
```

Out[53]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x2850269a9c8>,
  <matplotlib.lines.Line2D at 0x2850269ab08>],
 'caps': [<matplotlib.lines.Line2D at 0x285026a0788>,
  <matplotlib.lines.Line2D at 0x285026a0e48>],
 'boxes': [<matplotlib.lines.Line2D at 0x2850269a208>],
 'medians': [<matplotlib.lines.Line2D at 0x285026a0f88>],
 'fliers': [<matplotlib.lines.Line2D at 0x285026a6cc8>],
 'means': []}
```



This survey question has a few outliers. Whether this will be standardized will be determined after all survey boxplots are analyzed.

***The 8 survey question boxplots are very similar and consistent with one another. Due to this consistency, standardization is probably not necessary, but it will be completed any way to ensure accuracy of the model.**

vi. Standardize numeric fields

In [54]:
```
data['Population_z'] = stats.zscore(data['Population'])
data['Education_z'] = stats.zscore(data['Education_Numeric'])
data['Income_z'] = stats.zscore(data['Income'])
data['Children_z'] = stats.zscore(data['Children'])
data['TotalCharge_z'] = stats.zscore(data['TotalCharge'])
data['Additional_charges_z'] = stats.zscore(data['Additional_charges'])
data['Full_meals_eaten_z'] = stats.zscore(data['Full_meals_eaten'])
data['VitD_levels_z'] = stats.zscore(data['VitD_levels'])
data['Survey_Courteous_Staff_z'] = stats.zscore(data['Survey_Courteous_Staf
f'])
data['Survey_Doctor_Active_Listening_z'] = stats.zscore(data['Survey_Doctor_
Active_Listening'])
data['Survey_Hours_of_Treatment_z'] = stats.zscore(data['Survey_Hours_of_Tre
atment'])
data['Survey_Options_z'] = stats.zscore(data['Survey_Options'])
data['Survey_Reliability_z'] = stats.zscore(data['Survey_Reliability'])
data['Survey_Timely_Admission_z'] = stats.zscore(data['Survey_Timely_Admissi
on'])
data['Survey_Timely_Treatment_z'] = stats.zscore(data['Survey_Timely_Treatme
nt'])
data['Survey_Timely_Visit_z'] = stats.zscore(data['Survey_Timely_Visit'])

standardized_data = data[data.columns[-8:]]
standardized_data
```

Out[54]:

	Survey_Courteous_Staff_z	Survey_Doctor_Active_Listening_z	Survey_Hours_of_Treatment_z
1	-0.483672	0.470420	-0.506139
2	-0.483672	-0.489033	0.462548
3	-0.483672	-0.489033	0.462548
4	1.474513	1.429874	1.431236
5	0.495421	-0.489033	-0.506139
...	...	...	..
9996	0.495421	-1.448487	-0.506139
9997	0.495421	0.470420	-0.506139
9998	-0.483672	-1.448487	-1.474827
9999	0.495421	-0.489033	-0.506139
10000	0.495421	-0.489033	2.399924

10000 rows × 8 columns

Data Cleaning (Part III)

In [55]:
```
#Create new dataframe with only numerical columns

#Removed City, County, lat, long, Readmis_numeric, timezone
df = data[[
        'State_Numeric',
        'Zip',
        'Population_z',
        'Area_Numeric',
        'Gender_Numeric',
        'Age',
        'Education_z',
        'Employment_Numeric',
        'Income_z',
        'Marital_Numeric',
        'Children_z',
        'Initial_days',
        'TotalCharge_z',
        'Additional_charges_z',
        'Doc_visits',
        'Initial_admin_Numeric',
        'Full_meals_eaten_z',
        'VitD_levels_z',
        'VitD_supp',
        'Soft_drink_Numeric',
        'HighBlood_Numeric',
        'Stroke_Numeric',
        'Complication_risk_Numeric',
        'Overweight',
        'Arthritis_Numeric',
        'Diabetes_Numeric',
        'Hyperlipidemia_Numeric',
        'BackPain_Numeric',
        'Allergic_rhinitis_Numeric',
        'Reflux_esophagitis_Numeric',
        'Asthma_Numeric',
        'Services_Numeric',
        'Survey_Timely_Admission_z',
        'Survey_Timely_Treatment_z',
        'Survey_Timely_Visit_z',
        'Survey_Reliability_z',
        'Survey_Options_z',
        'Survey_Hours_of_Treatment_z',
        'Survey_Courteous_Staff_z',
        'Survey_Doctor_Active_Listening_z']]

df.head()
```

Out[55]:

	State_Numeric	Zip	Population_z	Area_Numeric	Gender_Numeric	Age	Education_z	Empl
1	2	35621	-0.473168	2	1	53.0	-0.094120	
2	10	32446	0.090242	3	2	51.0	0.237522	
3	43	57110	0.482983	2	2	53.0	0.237522	
4	24	56072	-0.526393	2	1	78.0	-0.425761	
5	47	23181	-0.315586	1	2	22.0	-0.425761	

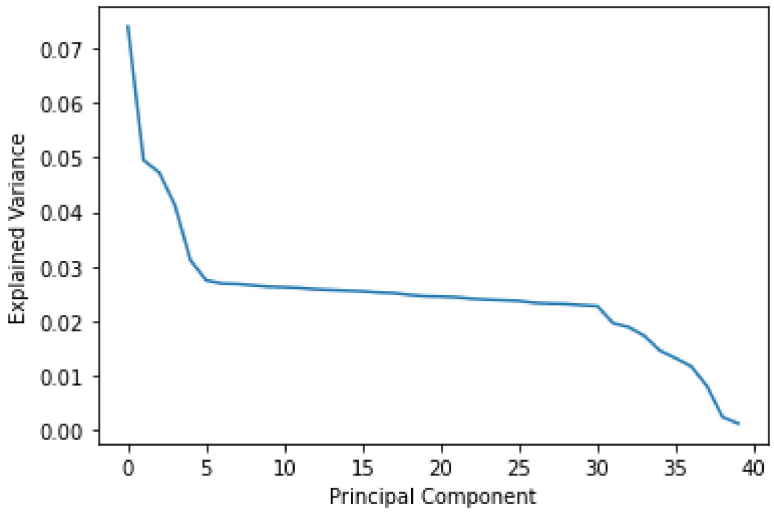
5 rows × 40 columns

In [56]:
```
#Normalize the data
df_normalized = (df-df.mean())/df.std()

#Assign all components to be extracted from the analysis
pca = PCA(n_components=df.shape[1])

#Call PCA application in the data set
pca.fit(df_normalized)
```

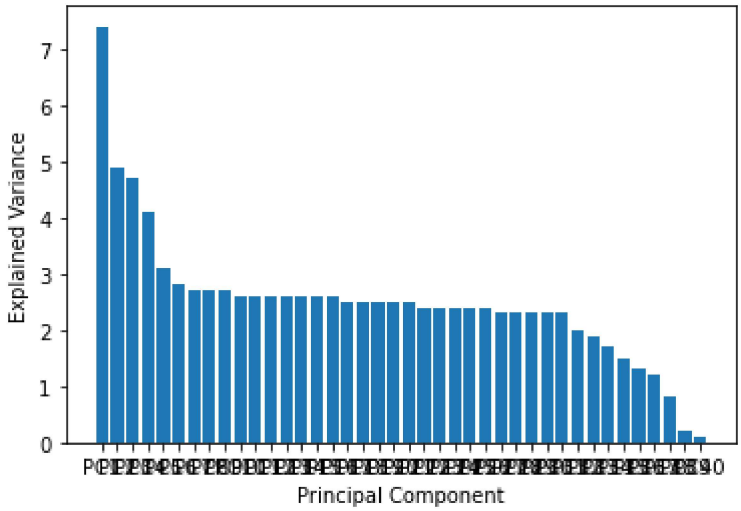
Out[56]:
```
PCA(copy=True, iterated_power='auto', n_components=40, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

Scree plots

In [57]:
```
plt.plot(pca.explained_variance_ratio_)
plt.xlabel('Principal Component')
plt.ylabel('Explained Variance')
plt.show()
```



In [58]:
```
per_var = np.round(pca.explained_variance_ratio_*100, decimals=1)
labels = ['PC' + str(x) for x in range(1, len(per_var)+1)]

plt.bar(x=range(1,len(per_var)+1), height = per_var, tick_label = labels)
plt.xlabel('Principal Component')
plt.ylabel('Explained Variance')
plt.show()
```



Create chart for PC, Explained Variance & Cumulative Variance

In [59]:
```
#ref: https://www.youtube.com/watch?v=oiusrJ0btwA

pcs_names = []
for i, col in enumerate (df.columns):
        pcs_names.append('PC' + str(i+1))

pc_dict = {'PC': pcs_names}
pc_df = pd.DataFrame(pc_dict)

explained_variance = pca.explained_variance_ratio_
cumulative_variance = np.cumsum(explained_variance)

explained_variance_df = pd.DataFrame(explained_variance, columns=['Explained V
ariance'])
cumulative_variance_df = pd.DataFrame(cumulative_variance, columns=['Cumulativ
e Variance'])

df2 = pd.concat([pc_df, explained_variance_df, cumulative_variance_df], axis =
1)
df2
```

Out[59]:

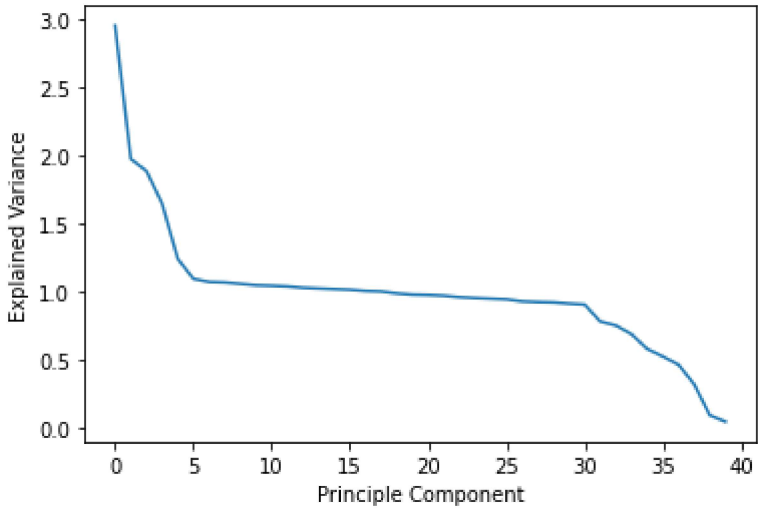
	PC	Explained Variance	Cumulative Variance
0	PC1	0.073965	0.073965
1	PC2	0.049488	0.123452
2	PC3	0.047286	0.170738
3	PC4	0.041304	0.212042
4	PC5	0.031177	0.243219
5	PC6	0.027506	0.270724
6	PC7	0.026934	0.297658
7	PC8	0.026838	0.324496
8	PC9	0.026586	0.351082
9	PC10	0.026326	0.377409
10	PC11	0.026239	0.403648
11	PC12	0.026108	0.429755
12	PC13	0.025859	0.455614
13	PC14	0.025735	0.481349
14	PC15	0.025611	0.506961
15	PC16	0.025500	0.532461
16	PC17	0.025278	0.557738
17	PC18	0.025176	0.582915
18	PC19	0.024824	0.607739
19	PC20	0.024599	0.632337
20	PC21	0.024534	0.656871
21	PC22	0.024398	0.681269
22	PC23	0.024130	0.705400
23	PC24	0.023978	0.729378
24	PC25	0.023859	0.753237
25	PC26	0.023738	0.776975
26	PC27	0.023362	0.800337
27	PC28	0.023240	0.823577
28	PC29	0.023171	0.846748
29	PC30	0.022949	0.869697
30	PC31	0.022780	0.892478
31	PC32	0.019671	0.912148
32	PC33	0.018941	0.931089
33	PC34	0.017337	0.948426
34	PC35	0.014640	0.963066

	PC	Explained Variance	Cumulative Variance
35	PC36	0.013255	0.976321
36	PC37	0.011754	0.988074
37	PC38	0.008162	0.996236
38	PC39	0.002470	0.998706
39	PC40	0.001294	1.000000

Eigenvalues

In [60]:
```
cov_matrix = np.dot(df_normalized.T, df_normalized) / df.shape[0]
eigenvalues = [np.dot(eigenvector.T, np.dot(cov_matrix, eigenvector)) for eige
nvector in pca.components_]
```

In [61]:
```
#Eigenvalues scree plot
plt.plot(eigenvalues)
plt.xlabel('Principle Component')
plt.ylabel('Explained Variance')
plt.show()
```



Output loadings

In [62]:
```
loadings = pd.DataFrame(pca.components_.T,
columns=labels,
index=df.columns)
loadings
```

Out[62]:

	PC1	PC2	PC3	PC4	PC5	PC
State_Numeric	0.000749	-0.010256	-0.000288	-0.016894	-0.664622	-0.02254
Zip	0.006772	0.007689	-0.001431	0.008393	0.631159	0.02978
Population_z	-0.010413	0.016551	0.017589	0.029349	0.282061	-0.34804
Area_Numeric	0.003038	0.028842	0.008182	0.001300	0.095280	-0.10174
Gender_Numeric	0.006277	-0.022550	-0.004350	-0.048169	-0.032301	0.16629
Age	0.000371	0.254303	-0.411300	0.030313	-0.043297	0.05690
Education_z	0.003771	-0.006874	0.017639	0.022411	0.004581	-0.06912
Employment_Numeric	0.020735	0.003402	0.005931	0.018041	0.013927	-0.01471
Income_z	0.002100	-0.003073	-0.002831	-0.018009	0.084955	-0.07312
Marital_Numeric	0.000725	-0.008129	-0.006751	-0.019424	0.010482	0.17345
Children_z	-0.001905	0.002385	-0.010280	0.025239	0.084415	0.11306
Initial_days	0.020673	0.370826	0.248350	0.066446	0.017133	-0.25541
TotalCharge_z	0.018538	0.603843	0.353786	0.079501	-0.016730	0.02370
Additional_charges_z	-0.002217	0.360473	-0.603529	0.022431	-0.004164	-0.00740
Doc_visits	-0.006999	0.002303	-0.014318	-0.008176	-0.011731	0.03427
Initial_admin_Numeric	0.003870	0.022238	0.029144	-0.030211	0.077742	0.41857
Full_meals_eaten_z	0.000519	0.003624	-0.032895	0.022596	0.056894	0.30914
VitD_levels_z	0.009497	0.465956	0.269577	0.048511	-0.031316	0.24948
VitD_supp	0.004807	0.033239	0.005262	0.010221	-0.059888	-0.39894
Soft_drink_Numeric	-0.006661	-0.000435	-0.001022	0.015815	0.043754	0.15161
HighBlood_Numeric	0.004446	0.253339	-0.444176	0.000645	0.037322	-0.05921
Stroke_Numeric	0.002574	0.005632	-0.042068	0.014251	0.019473	-0.00740
Complication_risk_Numeric	-0.012760	0.047321	-0.024953	-0.007704	0.030753	-0.14622
Overweight	-0.004221	-0.009209	-0.031270	0.010292	0.023171	0.02964
Arthritis_Numeric	0.014169	0.025174	0.001331	-0.004182	0.041588	-0.02692
Diabetes_Numeric	0.003059	-0.018072	-0.013821	0.031552	-0.045202	0.19892
Hyperlipidemia_Numeric	-0.016919	0.007645	0.015285	-0.013031	-0.027616	-0.02877
BackPain_Numeric	0.013058	0.028699	-0.004391	-0.007488	-0.056813	-0.05231
Allergic_rhinitis_Numeric	-0.004781	0.021191	-0.018176	0.018652	-0.080610	-0.16228
Reflux_esophagitis_Numeric	-0.006299	0.005009	0.022739	-0.011452	0.049785	-0.14271
Asthma_Numeric	0.010613	0.003846	-0.022848	0.020275	0.069501	0.28531
Services_Numeric	0.022174	0.006317	-0.020594	0.022008	-0.106947	0.03742
Survey_Timely_Admission_z	-0.454228	-0.025166	-0.003597	0.294320	-0.008408	0.00472
Survey_Timely_Treatment_z	-0.427845	-0.024588	-0.005156	0.291532	-0.004641	-0.01031
Survey_Timely_Visit_z	-0.394958	-0.026328	-0.000804	0.293049	-0.018232	0.00127

	PC1	PC2	PC3	PC4	PC5	PC
Survey_Reliability_z	-0.152017	0.059208	0.021453	-0.551324	0.002419	-0.01762
Survey_Options_z	0.189757	-0.072596	-0.015157	0.576573	-0.008731	0.02131
Survey_Hours_of_Treatment_z	-0.409917	0.026758	0.015015	-0.160866	0.020499	0.02438
Survey_Courteous_Staff_z	-0.356317	0.038849	0.003297	-0.168022	0.002844	0.00715
Survey_Doctor_Active_Listening_z	-0.311975	0.027545	0.011643	-0.164199	0.002358	0.00922

40 rows × 40 columns

Identify Significant Features

In [63]:
```
loadings['PC1'] = loadings['PC1'].abs()
loadings_sorted = loadings.sort_values(by = 'PC1', ascending=False)
loadings_sorted
```

Out[63]:

	PC1	PC2	PC3	PC4	PC5	PC6
Survey_Timely_Admission_z	0.454228	-0.025166	-0.003597	0.294320	-0.008408	0.004728
Survey_Timely_Treatment_z	0.427845	-0.024588	-0.005156	0.291532	-0.004641	-0.010318
Survey_Hours_of_Treatment_z	0.409917	0.026758	0.015015	-0.160866	0.020499	0.024384
Survey_Timely_Visit_z	0.394958	-0.026328	-0.000804	0.293049	-0.018232	0.001273
Survey_Courteous_Staff_z	0.356317	0.038849	0.003297	-0.168022	0.002844	0.007156
Survey_Doctor_Active_Listening_z	0.311975	0.027545	0.011643	-0.164199	0.002358	0.009229
Survey_Options_z	0.189757	-0.072596	-0.015157	0.576573	-0.008731	0.021317
Survey_Reliability_z	0.152017	0.059208	0.021453	-0.551324	0.002419	-0.017627
Services_Numeric	0.022174	0.006317	-0.020594	0.022008	-0.106947	0.037425
Employment_Numeric	0.020735	0.003402	0.005931	0.018041	0.013927	-0.014712
Initial_days	0.020673	0.370826	0.248350	0.066446	0.017133	-0.255415
TotalCharge_z	0.018538	0.603843	0.353786	0.079501	-0.016730	0.023702
Hyperlipidemia_Numeric	0.016919	0.007645	0.015285	-0.013031	-0.027616	-0.028779
Arthritis_Numeric	0.014169	0.025174	0.001331	-0.004182	0.041588	-0.026924
BackPain_Numeric	0.013058	0.028699	-0.004391	-0.007488	-0.056813	-0.052316
Complication_risk_Numeric	0.012760	0.047321	-0.024953	-0.007704	0.030753	-0.146223
Asthma_Numeric	0.010613	0.003846	-0.022848	0.020275	0.069501	0.285315
Population_z	0.010413	0.016551	0.017589	0.029349	0.282061	-0.348041
VitD_levels_z	0.009497	0.465956	0.269577	0.048511	-0.031316	0.249489
Doc_visits	0.006999	0.002303	-0.014318	-0.008176	-0.011731	0.034276
Zip	0.006772	0.007689	-0.001431	0.008393	0.631159	0.029780
Soft_drink_Numeric	0.006661	-0.000435	-0.001022	0.015815	0.043754	0.151619
Reflux_esophagitis_Numeric	0.006299	0.005009	0.022739	-0.011452	0.049785	-0.142717
Gender_Numeric	0.006277	-0.022550	-0.004350	-0.048169	-0.032301	0.166296
VitD_supp	0.004807	0.033239	0.005262	0.010221	-0.059888	-0.398944
Allergic_rhinitis_Numeric	0.004781	0.021191	-0.018176	0.018652	-0.080610	-0.162289
HighBlood_Numeric	0.004446	0.253339	-0.444176	0.000645	0.037322	-0.059218
Overweight	0.004221	-0.009209	-0.031270	0.010292	0.023171	0.029641
Initial_admin_Numeric	0.003870	0.022238	0.029144	-0.030211	0.077742	0.418576
Education_z	0.003771	-0.006874	0.017639	0.022411	0.004581	-0.069123
Diabetes_Numeric	0.003059	-0.018072	-0.013821	0.031552	-0.045202	0.198925
Area_Numeric	0.003038	0.028842	0.008182	0.001300	0.095280	-0.101747
Stroke_Numeric	0.002574	0.005632	-0.042068	0.014251	0.019473	-0.007406
Additional_charges_z	0.002217	0.360473	-0.603529	0.022431	-0.004164	-0.007403
Income_z	0.002100	-0.003073	-0.002831	-0.018009	0.084955	-0.073121

	PC1	PC2	PC3	PC4	PC5	PC6
Children_z	0.001905	0.002385	-0.010280	0.025239	0.084415	0.113063
State_Numeric	0.000749	-0.010256	-0.000288	-0.016894	-0.664622	-0.022542
Marital_Numeric	0.000725	-0.008129	-0.006751	-0.019424	0.010482	0.173454
Full_meals_eaten_z	0.000519	0.003624	-0.032895	0.022596	0.056894	0.309148
Age	0.000371	0.254303	-0.411300	0.030313	-0.043297	0.056900

40 rows × 40 columns

In [64]:
```
loadings['PC2'] = loadings['PC2'].abs()
loadings_sorted = loadings.sort_values(by = 'PC2', ascending=False)
loadings_sorted
```

Out[64]:

	PC1	PC2	PC3	PC4	PC5	PC6
TotalCharge_z	0.018538	0.603843	0.353786	0.079501	-0.016730	0.023702
VitD_levels_z	0.009497	0.465956	0.269577	0.048511	-0.031316	0.249489
Initial_days	0.020673	0.370826	0.248350	0.066446	0.017133	-0.255415
Additional_charges_z	0.002217	0.360473	-0.603529	0.022431	-0.004164	-0.007403
Age	0.000371	0.254303	-0.411300	0.030313	-0.043297	0.056900
HighBlood_Numeric	0.004446	0.253339	-0.444176	0.000645	0.037322	-0.059218
Survey_Options_z	0.189757	0.072596	-0.015157	0.576573	-0.008731	0.021317
Survey_Reliability_z	0.152017	0.059208	0.021453	-0.551324	0.002419	-0.017627
Complication_risk_Numeric	0.012760	0.047321	-0.024953	-0.007704	0.030753	-0.146223
Survey_Courteous_Staff_z	0.356317	0.038849	0.003297	-0.168022	0.002844	0.007156
VitD_supp	0.004807	0.033239	0.005262	0.010221	-0.059888	-0.398944
Area_Numeric	0.003038	0.028842	0.008182	0.001300	0.095280	-0.101747
BackPain_Numeric	0.013058	0.028699	-0.004391	-0.007488	-0.056813	-0.052316
Survey_Doctor_Active_Listening_z	0.311975	0.027545	0.011643	-0.164199	0.002358	0.009229
Survey_Hours_of_Treatment_z	0.409917	0.026758	0.015015	-0.160866	0.020499	0.024384
Survey_Timely_Visit_z	0.394958	0.026328	-0.000804	0.293049	-0.018232	0.001273
Arthritis_Numeric	0.014169	0.025174	0.001331	-0.004182	0.041588	-0.026924
Survey_Timely_Admission_z	0.454228	0.025166	-0.003597	0.294320	-0.008408	0.004728
Survey_Timely_Treatment_z	0.427845	0.024588	-0.005156	0.291532	-0.004641	-0.010318
Gender_Numeric	0.006277	0.022550	-0.004350	-0.048169	-0.032301	0.166296
Initial_admin_Numeric	0.003870	0.022238	0.029144	-0.030211	0.077742	0.418576
Allergic_rhinitis_Numeric	0.004781	0.021191	-0.018176	0.018652	-0.080610	-0.162289
Diabetes_Numeric	0.003059	0.018072	-0.013821	0.031552	-0.045202	0.198925
Population_z	0.010413	0.016551	0.017589	0.029349	0.282061	-0.348041
State_Numeric	0.000749	0.010256	-0.000288	-0.016894	-0.664622	-0.022542
Overweight	0.004221	0.009209	-0.031270	0.010292	0.023171	0.029641
Marital_Numeric	0.000725	0.008129	-0.006751	-0.019424	0.010482	0.173454
Zip	0.006772	0.007689	-0.001431	0.008393	0.631159	0.029780
Hyperlipidemia_Numeric	0.016919	0.007645	0.015285	-0.013031	-0.027616	-0.028779
Education_z	0.003771	0.006874	0.017639	0.022411	0.004581	-0.069123
Services_Numeric	0.022174	0.006317	-0.020594	0.022008	-0.106947	0.037425
Stroke_Numeric	0.002574	0.005632	-0.042068	0.014251	0.019473	-0.007406
Reflux_esophagitis_Numeric	0.006299	0.005009	0.022739	-0.011452	0.049785	-0.142717
Asthma_Numeric	0.010613	0.003846	-0.022848	0.020275	0.069501	0.285315
Full_meals_eaten_z	0.000519	0.003624	-0.032895	0.022596	0.056894	0.309148

	PC1	PC2	PC3	PC4	PC5	PC6
Employment_Numeric	0.020735	0.003402	0.005931	0.018041	0.013927	-0.014712
Income_z	0.002100	0.003073	-0.002831	-0.018009	0.084955	-0.073121
Children_z	0.001905	0.002385	-0.010280	0.025239	0.084415	0.113063
Doc_visits	0.006999	0.002303	-0.014318	-0.008176	-0.011731	0.034276
Soft_drink_Numeric	0.006661	0.000435	-0.001022	0.015815	0.043754	0.151619

40 rows × 40 columns

In [65]:
```
loadings['PC3'] = loadings['PC3'].abs()
loadings_sorted = loadings.sort_values(by = 'PC3', ascending=False)
loadings_sorted
```

Out[65]:

	PC1	PC2	PC3	PC4	PC5	PC6
Additional_charges_z	0.002217	0.360473	0.603529	0.022431	-0.004164	-0.007403
HighBlood_Numeric	0.004446	0.253339	0.444176	0.000645	0.037322	-0.059218
Age	0.000371	0.254303	0.411300	0.030313	-0.043297	0.056900
TotalCharge_z	0.018538	0.603843	0.353786	0.079501	-0.016730	0.023702
VitD_levels_z	0.009497	0.465956	0.269577	0.048511	-0.031316	0.249489
Initial_days	0.020673	0.370826	0.248350	0.066446	0.017133	-0.255415
Stroke_Numeric	0.002574	0.005632	0.042068	0.014251	0.019473	-0.007406
Full_meals_eaten_z	0.000519	0.003624	0.032895	0.022596	0.056894	0.309148
Overweight	0.004221	0.009209	0.031270	0.010292	0.023171	0.029641
Initial_admin_Numeric	0.003870	0.022238	0.029144	-0.030211	0.077742	0.418576
Complication_risk_Numeric	0.012760	0.047321	0.024953	-0.007704	0.030753	-0.146223
Asthma_Numeric	0.010613	0.003846	0.022848	0.020275	0.069501	0.285315
Reflux_esophagitis_Numeric	0.006299	0.005009	0.022739	-0.011452	0.049785	-0.142717
Survey_Reliability_z	0.152017	0.059208	0.021453	-0.551324	0.002419	-0.017627
Services_Numeric	0.022174	0.006317	0.020594	0.022008	-0.106947	0.037425
Allergic_rhinitis_Numeric	0.004781	0.021191	0.018176	0.018652	-0.080610	-0.162289
Education_z	0.003771	0.006874	0.017639	0.022411	0.004581	-0.069123
Population_z	0.010413	0.016551	0.017589	0.029349	0.282061	-0.348041
Hyperlipidemia_Numeric	0.016919	0.007645	0.015285	-0.013031	-0.027616	-0.028779
Survey_Options_z	0.189757	0.072596	0.015157	0.576573	-0.008731	0.021317
Survey_Hours_of_Treatment_z	0.409917	0.026758	0.015015	-0.160866	0.020499	0.024384
Doc_visits	0.006999	0.002303	0.014318	-0.008176	-0.011731	0.034276
Diabetes_Numeric	0.003059	0.018072	0.013821	0.031552	-0.045202	0.198925
Survey_Doctor_Active_Listening_z	0.311975	0.027545	0.011643	-0.164199	0.002358	0.009229
Children_z	0.001905	0.002385	0.010280	0.025239	0.084415	0.113063
Area_Numeric	0.003038	0.028842	0.008182	0.001300	0.095280	-0.101747
Marital_Numeric	0.000725	0.008129	0.006751	-0.019424	0.010482	0.173454
Employment_Numeric	0.020735	0.003402	0.005931	0.018041	0.013927	-0.014712
VitD_supp	0.004807	0.033239	0.005262	0.010221	-0.059888	-0.398944
Survey_Timely_Treatment_z	0.427845	0.024588	0.005156	0.291532	-0.004641	-0.010318
BackPain_Numeric	0.013058	0.028699	0.004391	-0.007488	-0.056813	-0.052316
Gender_Numeric	0.006277	0.022550	0.004350	-0.048169	-0.032301	0.166296
Survey_Timely_Admission_z	0.454228	0.025166	0.003597	0.294320	-0.008408	0.004728
Survey_Courteous_Staff_z	0.356317	0.038849	0.003297	-0.168022	0.002844	0.007156
Income_z	0.002100	0.003073	0.002831	-0.018009	0.084955	-0.073121

	PC1	PC2	PC3	PC4	PC5	PC6
Zip	0.006772	0.007689	0.001431	0.008393	0.631159	0.029780
Arthritis_Numeric	0.014169	0.025174	0.001331	-0.004182	0.041588	-0.026924
Soft_drink_Numeric	0.006661	0.000435	0.001022	0.015815	0.043754	0.151619
Survey_Timely_Visit_z	0.394958	0.026328	0.000804	0.293049	-0.018232	0.001273
State_Numeric	0.000749	0.010256	0.000288	-0.016894	-0.664622	-0.022542

40 rows × 40 columns

In [66]:
```
loadings['PC4'] = loadings['PC4'].abs()
loadings_sorted = loadings.sort_values(by = 'PC4', ascending=False)
loadings_sorted
```

Out[66]:

	PC1	PC2	PC3	PC4	PC5	PC6
Survey_Options_z	0.189757	0.072596	0.015157	0.576573	-0.008731	0.021317
Survey_Reliability_z	0.152017	0.059208	0.021453	0.551324	0.002419	-0.017627
Survey_Timely_Admission_z	0.454228	0.025166	0.003597	0.294320	-0.008408	0.004728
Survey_Timely_Visit_z	0.394958	0.026328	0.000804	0.293049	-0.018232	0.001273
Survey_Timely_Treatment_z	0.427845	0.024588	0.005156	0.291532	-0.004641	-0.010318
Survey_Courteous_Staff_z	0.356317	0.038849	0.003297	0.168022	0.002844	0.007156
Survey_Doctor_Active_Listening_z	0.311975	0.027545	0.011643	0.164199	0.002358	0.009229
Survey_Hours_of_Treatment_z	0.409917	0.026758	0.015015	0.160866	0.020499	0.024384
TotalCharge_z	0.018538	0.603843	0.353786	0.079501	-0.016730	0.023702
Initial_days	0.020673	0.370826	0.248350	0.066446	0.017133	-0.255415
VitD_levels_z	0.009497	0.465956	0.269577	0.048511	-0.031316	0.249489
Gender_Numeric	0.006277	0.022550	0.004350	0.048169	-0.032301	0.166296
Diabetes_Numeric	0.003059	0.018072	0.013821	0.031552	-0.045202	0.198925
Age	0.000371	0.254303	0.411300	0.030313	-0.043297	0.056900
Initial_admin_Numeric	0.003870	0.022238	0.029144	0.030211	0.077742	0.418576
Population_z	0.010413	0.016551	0.017589	0.029349	0.282061	-0.348041
Children_z	0.001905	0.002385	0.010280	0.025239	0.084415	0.113063
Full_meals_eaten_z	0.000519	0.003624	0.032895	0.022596	0.056894	0.309148
Additional_charges_z	0.002217	0.360473	0.603529	0.022431	-0.004164	-0.007403
Education_z	0.003771	0.006874	0.017639	0.022411	0.004581	-0.069123
Services_Numeric	0.022174	0.006317	0.020594	0.022008	-0.106947	0.037425
Asthma_Numeric	0.010613	0.003846	0.022848	0.020275	0.069501	0.285315
Marital_Numeric	0.000725	0.008129	0.006751	0.019424	0.010482	0.173454
Allergic_rhinitis_Numeric	0.004781	0.021191	0.018176	0.018652	-0.080610	-0.162289
Employment_Numeric	0.020735	0.003402	0.005931	0.018041	0.013927	-0.014712
Income_z	0.002100	0.003073	0.002831	0.018009	0.084955	-0.073121
State_Numeric	0.000749	0.010256	0.000288	0.016894	-0.664622	-0.022542
Soft_drink_Numeric	0.006661	0.000435	0.001022	0.015815	0.043754	0.151619
Stroke_Numeric	0.002574	0.005632	0.042068	0.014251	0.019473	-0.007406
Hyperlipidemia_Numeric	0.016919	0.007645	0.015285	0.013031	-0.027616	-0.028779
Reflux_esophagitis_Numeric	0.006299	0.005009	0.022739	0.011452	0.049785	-0.142717
Overweight	0.004221	0.009209	0.031270	0.010292	0.023171	0.029641
VitD_supp	0.004807	0.033239	0.005262	0.010221	-0.059888	-0.398944
Zip	0.006772	0.007689	0.001431	0.008393	0.631159	0.029780
Doc_visits	0.006999	0.002303	0.014318	0.008176	-0.011731	0.034276

	PC1	PC2	PC3	PC4	PC5	PC6
Complication_risk_Numeric	0.012760	0.047321	0.024953	0.007704	0.030753	-0.146223
BackPain_Numeric	0.013058	0.028699	0.004391	0.007488	-0.056813	-0.052316
Arthritis_Numeric	0.014169	0.025174	0.001331	0.004182	0.041588	-0.026924
Area_Numeric	0.003038	0.028842	0.008182	0.001300	0.095280	-0.101747
HighBlood_Numeric	0.004446	0.253339	0.444176	0.000645	0.037322	-0.059218

40 rows × 40 columns