# [DOCTOR FEE PREDICTION]

Milestone 2

12 May 2024

Machine Learning

Prof. Dina khatab


## Team

AI_R_1
Nour sameh 2021170885
Nour madkour 2021170884
Rawan badr 2021170863
Menna maged 2021170875
Hoor hesham 2021170861

## Data Overview

### Data Description

- **Doctor Name**: The name of the doctor.
- **City**: The city where the doctor is located.
- **Specialization**: The area of specialization or medical expertise of the doctor.
- **Doctor Qualification**: The qualifications or degrees held by the doctor.
- **Experience (Years)**: The number of years of experience the doctor has.
- **Total Reviews**: The total number of reviews received by the doctor.
- **Patient Satisfaction Rate (%age)**: The percentage of patients satisfied with the doctor's services.
- **Avg Time to Patients (mins)**: The average time taken by the doctor to attend to patients.
- **Wait Time (mins)**: The average wait time for patients before being attended to by the doctor.
- **Hospital Address**: The address of the hospital where the doctor practices.
- **Doctors Link**: A link to the doctor's profile or information.
- **Fee (PKR)**: The consultation fee charged by the doctor .

```
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   Doctor Name                      2386 non-null   object
 1   City                             2386 non-null   object
 2   Specialization                   2386 non-null   object
 3   Doctor Qualification             2386 non-null   object
 4   Experience(Years)                2386 non-null   float64
 5   Total_Reviews                    2386 non-null   int64
 6   Patient Satisfaction Rate(%age)  2386 non-null   int64
 7   Avg Time to Patients(mins)       2386 non-null   int64
 8   Wait Time(mins)                  2386 non-null   int64
 9   Hospital Address                 2386 non-null   object
 10  Doctors Link                     2386 non-null   object
 11  Fee(PKR)                         2386 non-null   int64
dtypes: float64(1), int64(5), object(6)
memory usage: 223.8+ KB
```

1.No null values but there may be missing values
2.data need to be encoded "Doctor Name" , "City", "Specialization" ,
"Doctor Qualification" ,"Hospital Address", "Doctors Link"

3. dropped duplicated values ->(13 rows )
4. Statistics for categorical columns.

| | count | unique | top | freq |
|---|---|---|---|---|
| Doctor Name | 2373 | 2190 | Dr. Muhammad Amjad | 4 |
| City | 2373 | 117 | LAHORE | 151 |
| Specialization | 2373 | 150 | General Physician | 406 |
| Doctor Qualification | 2373 | 1041 | MBBS | 332 |
| Hospital Address | 2373 | 1178 | No Address Available | 552 |
| Doctors Link | 2373 | 1606 | No Link Available | 645 |

found out that most freq in these 2 cols 'Hospital Address' & 'Doctors Link ' No Address Available & No Link Available So we may drop them and may not
5. Renamed columns to manipulate on the easily

```python
#0
def rename_cols(df):
  df.rename(columns={'Fee(PKR)': 'Fee'}, inplace=True)
  df.rename(columns={'Patient Satisfaction Rate(%age)': 'Patient_Satisfaction_Rate'}, inplace=True)
  df.rename(columns={'Experience(Years)': 'Experience_Years'}, inplace=True)
  df.rename(columns={'Avg Time to Patients(mins)': 'Avg_time_per_Patient'}, inplace=True)
  df.rename(columns={'Wait Time(mins)': 'Wait_Time'}, inplace=True)
```

**Table of content**

# 1.Splitting Data

```
[ ]  X = df.iloc[:, :-1]
     y = df.iloc[:, -1]
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)
```

# 2.Doctor Name and Feature engineering (title)

```
def extract_titles_and_clean_name(name):
    # Define a regex pattern for titles
    title_regex = r'(Dr\.|Prof\.|Mr\.|Ms\.|Colonel|Assoc\. Prof\. Dr\.|Asst\. Prof\. Dr\.|Prof\. Dr\.)'

    # List of accepted titles
    accepted_titles = ["Dr", "Asst Prof Dr", "Prof, Dr", "Assoc Prof Dr"]

    # Find all titles in the name
    titles = re.findall(title_regex, name)

    # Clean the name by removing the extracted titles
    cleaned_name = re.sub(title_regex, '', name).strip()

    # Convert titles to a cleaned string without periods
    title_str = ', '.join(titles).replace('.', '').strip()

    # Check if the concatenated title string is in the list of accepted titles
    if title_str not in accepted_titles:
        title_str = 'others'

    return title_str, cleaned_name


X_train[['Titles', 'Cleaned Name']] = X_train['Doctor Name'].apply(lambda x: pd.Series(extract_titles_and_clean_name(x)))

X_train['Doctor Name'] = X_train['Cleaned Name']
X_train.drop('Cleaned Name', axis=1, inplace=True)

X_train.head()
```

Cleaning Doctor name column and creating title column contain titles after cleaning
*If not from these added to unique value ‘others’

```
title_counts = X_train['Titles'].value_counts()
title_counts
```

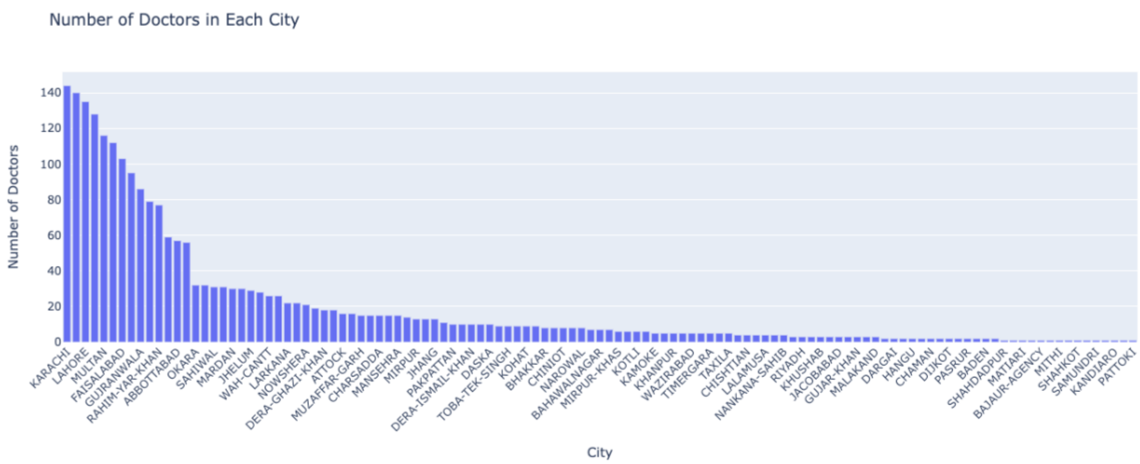
```
Titles
Dr               1512
Asst Prof Dr      201
Prof, Dr          124
Assoc Prof Dr      60
others              1
Name: count, dtype: int64
```

Most from 2-4 repitition
the duplicated names maybe Similarity of names we will see that when see more relation with other columns like 'Link' , ‘City’, ‘Specialization’(refer to 1.2 in Dr names number(5))

5. 1st refered to dropped rows of dr that have same ['Doctor Name', 'Specialization','City'] and also 'Doctors Link' as it is wierd to have dr have same all things(8

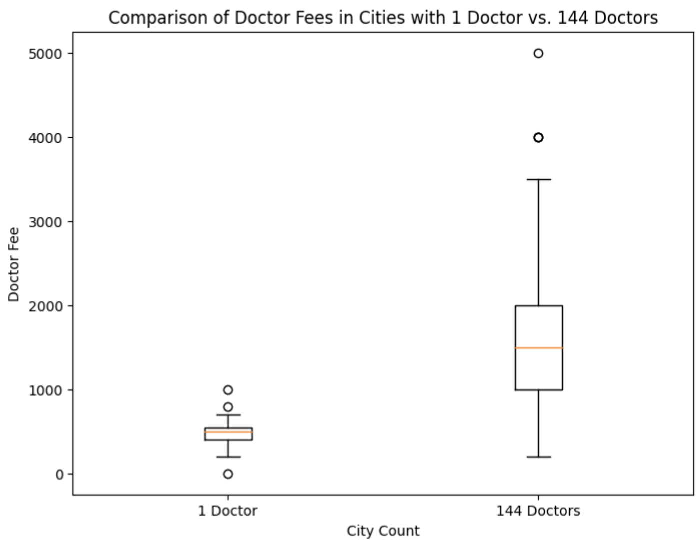
## 1.3 Analysis

1




In conclusion, the analysis reveals that Karachi has the highest number of doctors compared to other cities in the dataset. However, it's worth noting that several cities have only one doctor listed. This discrepancy in the distribution of doctors across cities might indicate variations in healthcare accessibility and resource allocation.

2.




```
Summary Statistics for Fees in Cities with 1 Doctor:
count      15.000000
mean      493.333333
std       240.436112
min         0.000000
25%       400.000000
50%       500.000000
75%       550.000000
max      1000.000000
Name: Fee, dtype: float64

Summary Statistics for Fees in Cities with 144 Doctors:
count     144.000000
mean     1524.305556
std       816.460597
min       200.000000
25%      1000.000000
50%      1500.000000
75%      2000.000000
max      5000.000000
Name: Fee, dtype: float64
```

Cities with 1 doctor have a fee range from 0 to 1000, whereas cities with 144 doctors have a wider fee range from $200 to 5000 This indicates greater variability in the fees charged by doctors in cities with a higher concentration of doctors.

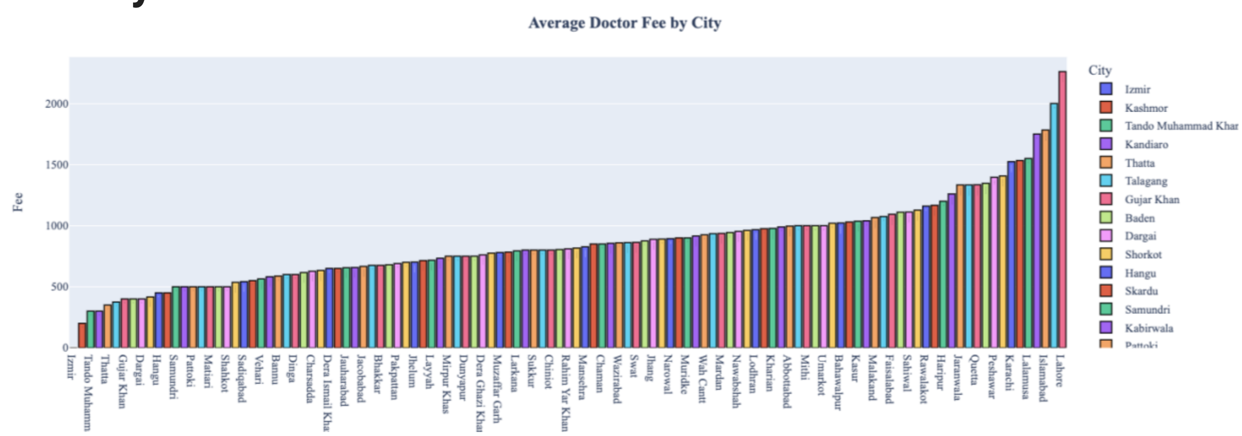
avg comparing
In cities with only 1 doctor, the mean fee charged is significantly lower at approximately 493.33 compared to cities with 144 doctors, where the mean fee is substantially higher at approximately 1524.31. This suggests that there is a noticeable difference in the average fees depending on the number of doctors in a city.

# 2.City

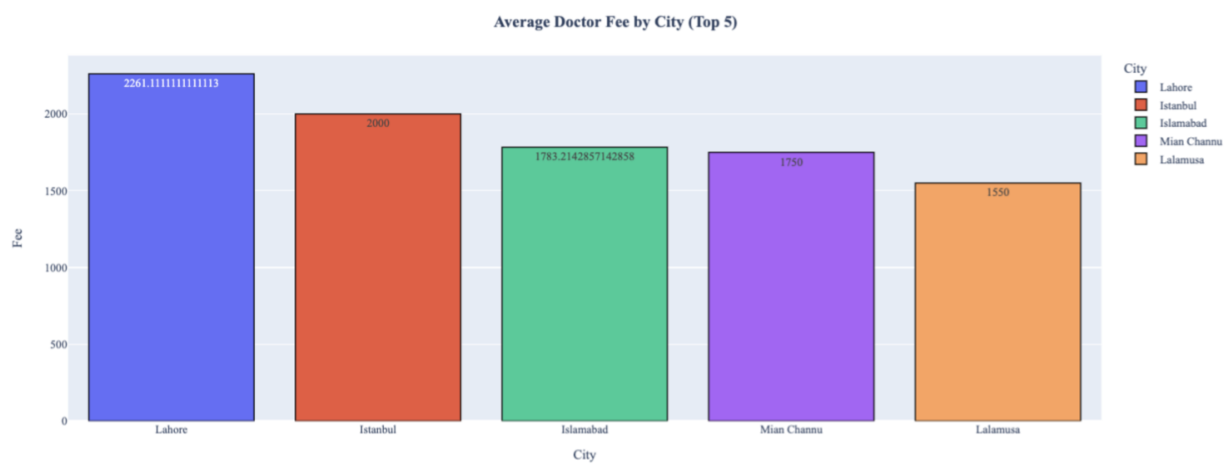
## 2.1 Cleaning

1.Removed '-' and made city lower case in order to see if there is any repeated city

## 2.2 Analysis




2. the avg fees are in LAHORE city lets dive into it and see fees of drs in this city , whilr in IZMIR is the lowes avg tends to 0



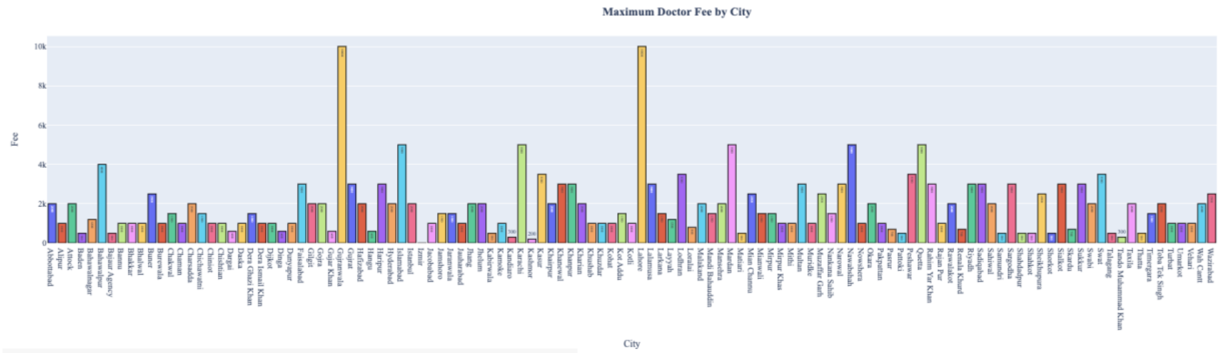
3.The fees vary considerable, with a standard deviation of approximately 1369.37, suggesting a wide range of fee amounts.

4. The city with the minimum average fees is Izmir with an average fee of $0.00.

| | Doctor Name | City | Specialization | Doctor Qualification | Experience_Years | Total_Reviews | Patient_Satisfaction_Rate | Avg_time_per_Patient | Wait_Time | Hospital Address | Doctors Link | Fee | Titles |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2025 | Hayati Turker | Izmir | Eye Surgeon | M.D. | 18.0 | 0 | 94 | 14 | 11 | Netgoz Eye Hospital, YalÄ± Mahallesi, Izmir | No Link Available | 0 | Dr |

5. this suggest that this dr giving charity for ppl as it is free for (Eye Surgeon)

**`Patient_Satisfaction_Rate is 94`**




```
Summary Statistics for Fees in GUJRANWALA:
count       86.000000
mean      1259.534884
std       1146.132019
min         20.000000
25%        700.000000
50%       1000.000000
75%       1500.000000
max      10000.000000
Name: Fee, dtype: float64
```

a wide range of fees charged by doctors in the city, with a minimum fee of 20 and a maximum fee of 10,000. The mean fee of approximately 1259.53 indicates the average cost of medical services

(IQR) of 800 (from 700 to 1500) suggests that the middle 50% of fees are relatively consistent, with the median fee (1000) falling within this range. This indicates that while there is considerable variability in fees, **a significant portion of doctors charge fees within a certain range**

## 2.3 Feature Engineering (Region)

Used folium library to see correlation between cities and desided to devide it to 6 regions

```
df['Region'].value_counts()
```

```
Region
Punjab Region           1420
KPK Region               389
Sindh Region             329
Balochistan Region       119
Kashmir Region            24
International Region      12
Name: count, dtype: int64
```

# 3.Specialization

## 2.1 Cleaning

```
[ ] df[['Specialization']].describe().T
```

| | count | unique | top | freq |
|---|---|---|---|---|
| Specialization | 2293 | 140 | General Physician | 406 |

1.Processed specialization to remove redundant values

```
from tabulate import tabulate
```

Captured more typos and mapped the corrected once

```
specialization_mapping = {
    "Pediatrician,Pediatric": "Pediatrician",
    "Lung Specialist": "Pulmonologist",
    "Eye Surgeon,Eye Specialist": "Ophthalmologist",
    "Sexologist": "Andrologist",
    "Cosmetic Surgeon,Dermatologist": "Cosmetic Dermatologist",
    "Internal Medicine Specialist,General Physician,Infectious
Diseases": "Infectious Disease Specialist",
}
```

```
df["Specialization"].nunique()
```

```
104
```

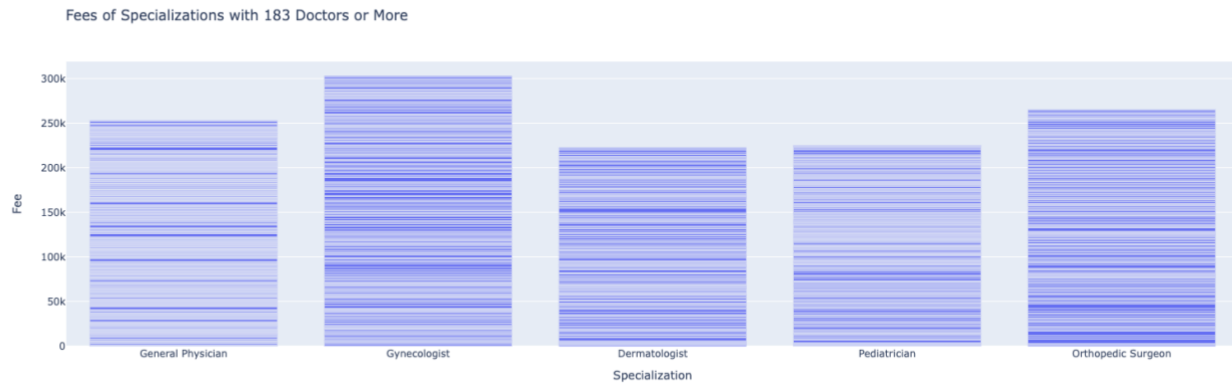
Went from 140 to 128 and that's good

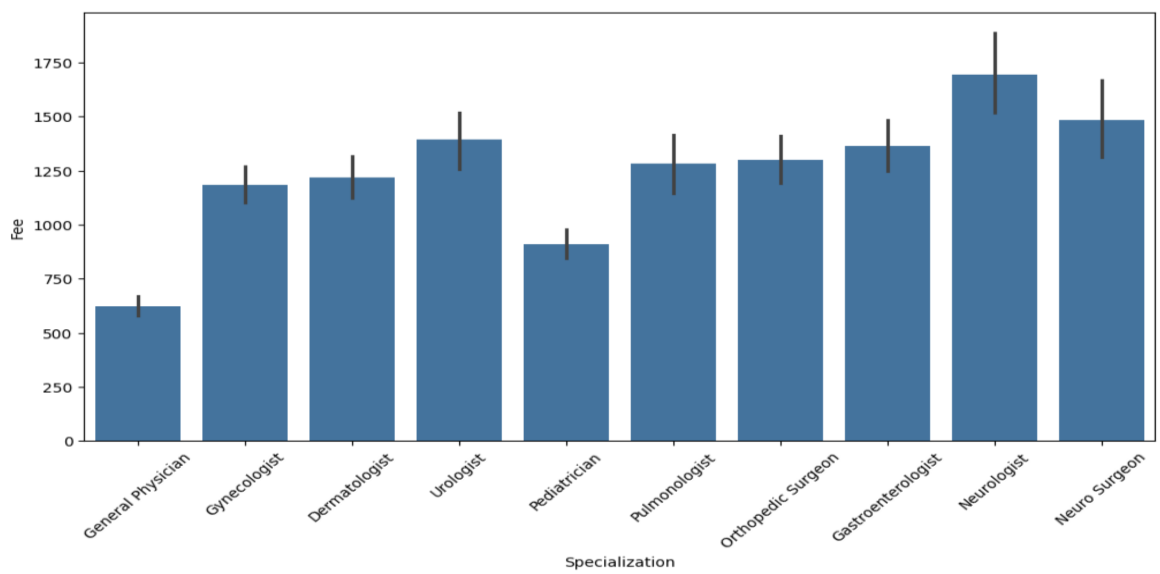
```
#clean
def process_specialization(entry):
  entry = entry.replace('/', ',')

  specialties = [s.strip() for s in entry.split(',')]
  unique_specialties = []
  for specialty in specialties:
      if specialty not in unique_specialties:
          unique_specialties.append(specialty)
  # Join back into a string
  unique_specialties_str = ','.join(unique_specialties)
  return unique_specialties_str
df['Specialization'] = df['Specialization'].apply(process_specialization)
```

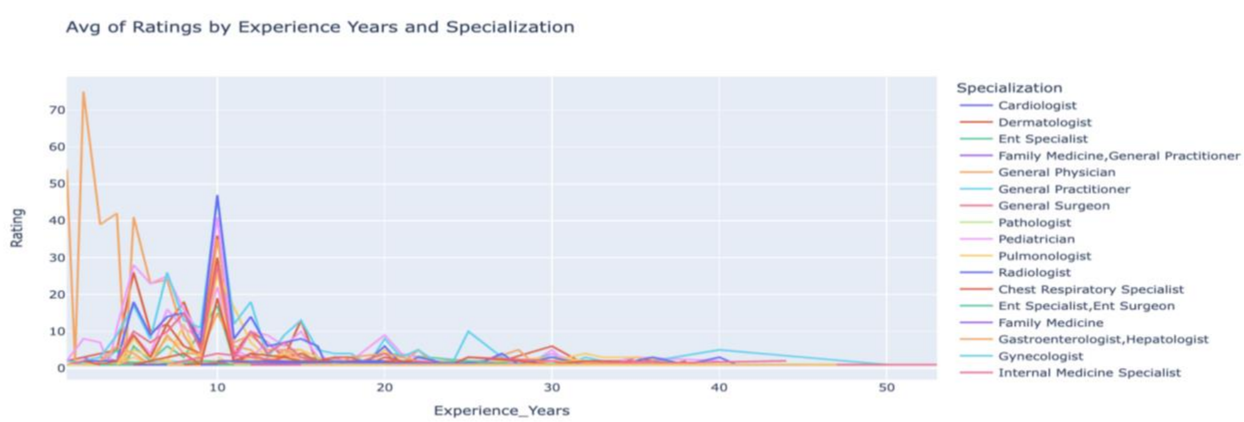
Replace any'/' with (,) then split specialization using(,)

```
#map
def map_specialization(specialization):
    for key, value in specialization_mapping.items():
        if key in specialization:
            return value
    return specialization
df['Specialization'] = df['Specialization'].apply(map_specialization)
```

## 2.2 Analysis

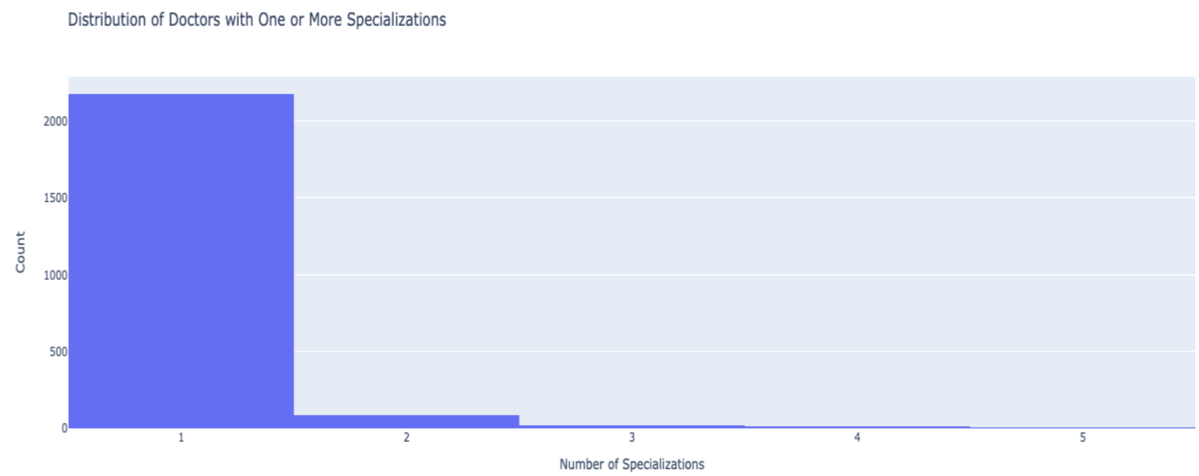



```
Drs with less experiene have highest rating and dr that are 47
yrs are not given a rate
```

## 3.3 Feature Engineering (Specialization Count)




1 . Most drs have 1 or 2 specialization
Number of Dr with multiple specializations: 118
3. Maximum number of specializations: 5
Minimum number of specializations:1

```
def calc_Specialization_count(df):
  df['Specialization Count'] = df['Specialization'].str.count(',') + 1

calc_Specialization_count(X_train)
```

```
max_specialization_count = X_train['Specialization Count'].max()
min_specialization_count = X_train['Specialization Count'].min()

print("Maximum number of specializations:", max_specialization_count)
print("Minimum number of specializations:", min_specialization_count)
```

```
Maximum number of specializations: 5
Minimum number of specializations: 1
```

# 4. Doctor Qualification

| | count | unique | top | freq |
|---|---|---|---|---|
| **Doctor Qualification** | 2293 | 1014 | MBBS | 332 |

# 4.1 cleaning

```
Doctor Qualification
MBBS                                                              332
MBBS, FCPS                                                        129
 MBBS                                                              78
MBBS,FCPS                                                          68
MBBS, FCPS (Gastroenterology)                                      40
MBBS, FCPS (Obstetrics &amp; Gynecology)                           39
MBBS, FCPS (Orthopedic Surgery)                                    38
MBBS, FCPS (Dermatology)                                           33
MBBS, FCPS (Urology)                                               28
MBBS, FCPS (Pulmonology)                                           24
MBBS, FCPS (Neurology)                                             24
MBBS, FCPS (Neuro Surgery)                                         22
MBBS, FCPS (Pediatrics)                                            18
MBBS, MCPS                                                         18
MBBS, FCPS (Nephrology)                                            17
MBBS, FCPS (Orthopaedic Surgery)                                   15
 MBBS , FCPS                                                       14
 MBBS , FCPS (Obstetrics &amp; Gynecology)                         14
MD                                                                 14
MBBS, MCPS, FCPS                                                   12
MBBS, DTCD                                                         10
MBBS, MS (Urology)                                                 10
MBBS , FCPS (Orthopedic Surgery)                                   10
MBBS, MD                                                           10
MBBS, DCH                                                           9
MBBS, FCPS (Obstetrics &amp; Gynaecology)                           9
 MBBS , FCPS (Obstetrics &amp; Gynaecology)                         9
MBBS, FCPS, MCPS                                                    8
MBBS, MS (Neurosurgery)                                             8
MBBS, FCPS (Neurosurgery)                                           8
 MBBS , FCPS (Orthopedic Surgery)                                   7
MBBS, MCPS (Obstetrics &amp; Gynaecology)                           7
MBBS, FCPS (Ophthalmology)                                          7
MBBS, MCPS (Pediatrics)                                             7
 MBBS , FCPS (Pediatrics)                                           7
MBBS, FCPS (Gastroentrology)                                        6
MBBS, MCPS (Dermatology)                                            6
 MBBS , MS (Neurosurgery)                                           6
MBBS, RMP                                                           6
MBBS, FCPS (Medicine)                                               6
MBBS, DOMS                                                          5
MBBS, FCPS (Paediatrics)                                            5
```

there are typo error so will remove it

## Tried nltk gave us 845 unique so prefered to do it manually

```
"""
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import CountVectorizer

# Download NLTK resources (if not already downloaded)
#nltk.download('punkt')
#nltk.download('stopwords')
#nltk.download('wordnet')

# Initialize Lemmatizer and CountVectorizer
lemmatizer = WordNetLemmatizer()
vectorizer = CountVectorizer()

# Tokenization and Lemmatization
def preprocess_text(text):
    tokens = word_tokenize(text.lower())  # Tokenization and convert to lowercase
    tokens = [lemmatizer.lemmatize(token) for token in tokens]  # Lemmatization
    return ' '.join(tokens)

# Remove stopwords and perform lemmatization
stop_words = set(stopwords.words('english'))

# Apply preprocessing to each value in the 'Doctor Qualification' column
df['Cleaned Qualifications'] = df['Doctor Qualification'].apply(preprocess_text)

# Vectorization using Bag of Words
X = vectorizer.fit_transform(df['Cleaned Qualifications'])

# X now contains the vectorized representation of 'Doctor Qualification' column
"""
```

```python
def clean_qualifications(df):
    # Combine and update all replacements into a single dictionary
    replacements = {
        r'\bPhD\b': 'PHD', r'\bM\.D\.\b': 'MD', r'\bD\.M\.S\b': 'DMS',
        r'\bB\.Sc\.\b': 'BSC', r'\bM\.S\.\b': 'MS', r'\bM\.Phil\b': 'MPHIL',
        r'\bG\.A\.M\.S\b': 'GAMS', r'\(D\.H\.B\)': 'DHB', r'\(D\.Ac\)': 'PHD',
        r'Ophtamology': 'Ophthalmology', r'Gastroentrology': 'Gastroenterology',
        r'OtoRhinoLaryngology': 'Otorhinolaryngology', r'Paediatrics': 'Pediatrics',
        r'Pulmonology': 'Pulmonary', r'ENT': 'Otolaryngology', r'OrthopedicSurgery': 'Orthopedic Surgery',
        r'NeuroSurgery': 'Neurosurgery', r'Medicine': 'Internal Medicine',
        r'OBSTETRICS&GYNAECOLOGY': 'Obstetrics&Gynecology', r'Gynecology&amp;Obstetrics': 'Gynecology and Obstetrics',
        r'Genecology&amp;Obstetrics': 'Gynecology and Obstetrics', r'OtorhinolaryngologicENT': 'Otorhinolaryngologic,ENT',
        r'MasterOfSurgery': 'Master of Surgery', r'MD\d*': 'MD', r'MDGastroenterology': 'MD,Gastroenterology',
        r'FCPSPediatrics': 'FCPS,Pediatrics', r'MBBSMD': 'MBBS,MD', r'FRCSOrthopedics': 'FRCS,Orthopedics',
        r'MCPSGynae/Obs': 'MCPS(Gynecology/Obs)', r'MD-RMP': 'MD, RMP', r'Masters\(NeuroSurgeon\)': 'Masters, Neurosurgery',
        r'\(|\)': '', r'[^a-zA-Z,]': '', r'Ophthalmologist': 'Ophthalmology', r'GASTROENTEROLOGY': 'Gastroenterology',
        r'MCPS,': 'MCPS', r'M\.D': 'MD', 'MD 1': 'MD'
    }

    # Apply all replacements
    df['Doctor Qualification'] = df['Doctor Qualification'].replace(replacements, regex=True)

    # Additional replacements to handle specific concatenations
    concatenations = {
        r'FCPSOBSTETRICSampGYNAECOLOGY': 'FCPS,Obstetrics&Gynecology',
        r'FCPSOtolaryngology': 'FCPS,Otolaryngology',
        r'MCPSFCPS': 'MCPS,FCPS'
    }
    df['Doctor Qualification'] = df['Doctor Qualification'].replace(concatenations, regex=True)

    # Remove all unnecessary spaces, then remove spaces around commas
    df['Doctor Qualification'] = df['Doctor Qualification'].str.replace(r'\s+', '')
    df['Doctor Qualification'] = df['Doctor Qualification'].str.replace(r'\s*,\s*', ',', regex=True)
    df['Doctor Qualification'] = df['Doctor Qualification'].str.replace(r'\([^)]*\)', '', regex=True)


    # Enhanced cleaning function
    def enhance_cleaning(qualification):
        # Replace HTML entities and correct specific cases
        qualification = qualification.replace('&amp;', '&')
        qualification = re.sub(r'(?<!\w)([A-Z]+)(?!\w)', lambda x: x.group(1), qualification)
        qualification = qualification.replace('DiplomainTBandChestDiseases', 'DTBCD')

        # Split, sort, and remove duplicates
        parts = sorted(set(qualification.split(',')))  # Remove duplicates and sort
        return ','.join(parts)

    # Apply the enhanced cleaning function
    df['Doctor Qualification'] = df['Doctor Qualification'].apply(enhance_cleaning)

    return df
```

**826 and that's better than nltk**

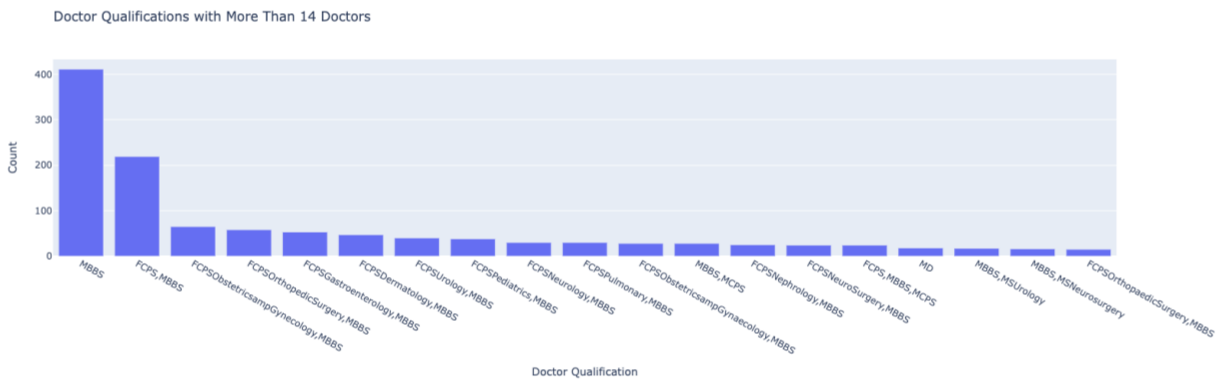
## 4.2 Analysis

```
#drs that dont have mbbs
#conclusion : 89 rows
```

```
Qualifications of rows where 'MBBS' is not present:
MD
MD,MSMasterofSurgery
DiplomaAnaesthsia,DiplomaChildHealthfromInstituteofHealthampManagementScienceIslamabad,MD
DHMS,DIPSexology,InternationalAffiliateMemberAmericanPsychologicalAssociationDivisionUSA,MScPsychology
DHMS
CISUK,CLDPUAE,CMTOSriLanka,CTOFUK,DIPSexology,MPHILPsychology,RHMPPak
CRSM,MD,MemberofAmericanSocietyforReproductiveInternalMedicine,PGFM
CertifiedSexologist,DiplomaPsychosexualampRelationshipTherapist,DiplomateSexologist,PHDHumanSexology,RHMP
MD
FCPS,PHDGastro
DoctorofInternalMedicineMD
MDBasicMedicalQualification
MD
MD
Doctorate
CertifiedUrodynamicist,MD,MSUrology
MSneurosurgeon
MD
MD,MSC
MCPS
MCPSPediatrics,MD
MD,MasterofSurgeryinOrthopedics
FCPSPediatrics,MD
FRACSOrthopedicSurgery,FRCSTraumaampOrth,MBCHBBachelorofInternalMedicineampBachelorofSurgery
DiplomainDiabetes,MD
DAC,DHMS,DIPSexology
DiplomaInDermatology,DoctorofInternalMedicineMD,MemberofMedicalDermatologysocietyUSAMMDSUSA
FCPSNeuroSurgery,MD
FACCUSA,FACPUSA,FASIMUSA,FRCPEdinburgh,FRCPLONDON,MRCPUK
MCPSPediatrics,MD
MDBASICMEDICALQUALIFICATION
MD
MD
MD,MastersNeuroSurgeon
FCPS,MD
MDBasicMedicalQualification
MD
FCPS,FRCS,MCPS
CRSMSexualampReproductiveInternalMedicine,FCPSUrology,MCPSGenSurgery,MD,MemberofPakistansocietyforAndrologyampSexualInternalMedicinePSA
MD
DiplomateAmericanBoardHairRestorationSurgery,MBB,MCPSDermatology
MD
DHMS
FCPSSURGERY,FCPSUROLOGY
FCPSUROLOGY,MD,MRCS
DIPLASTHUK,DPDUK,MDEurope,MRCGPUK
MD
MDGeneralPhysician
FCPSPediatrics,MCPSPediatrics,MD
MD
MD
DiplomaDermatology,MD,MasterofScienceinPublicHealth
FCPSII,MD
CertifiedUrodynamicist,MD,MSUrology
FCPSNeurology,FRCP,MD
MDRussia
FellowshipinVitreoretinalDiseasesandSurgery,Ophthalmology
```

- 96.2% of the drs have MBBs Qualification
- 3.8 % only dont have MBBs.

most of drs have MBBS (Bachelor of Medicine, Bachelor of Surgery)

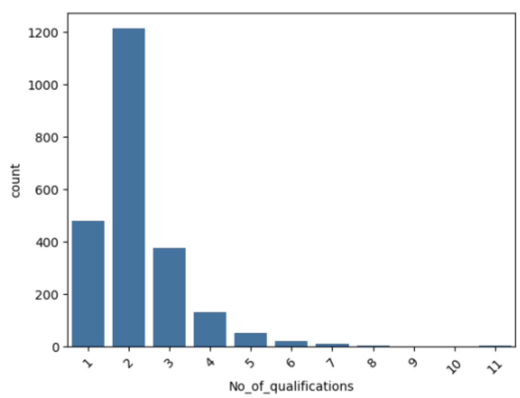


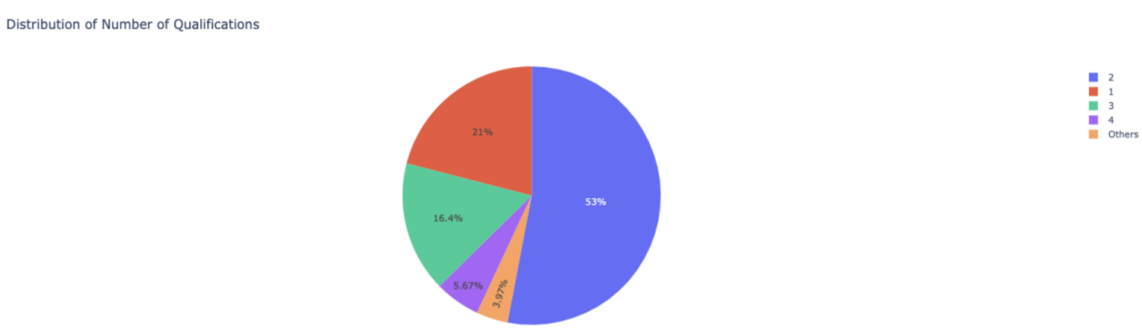

Most drs take this qualifications

```
Maximum number of qualifications: 10
Minimum number of qualifications: 1
```

## 4.3 Feature Engineering (Number of Qualification)

```
No_of_qualifications
2      999
1      390
3      317
4      112
5       47
6       20
7        6
8        5
10       1
9        1
```

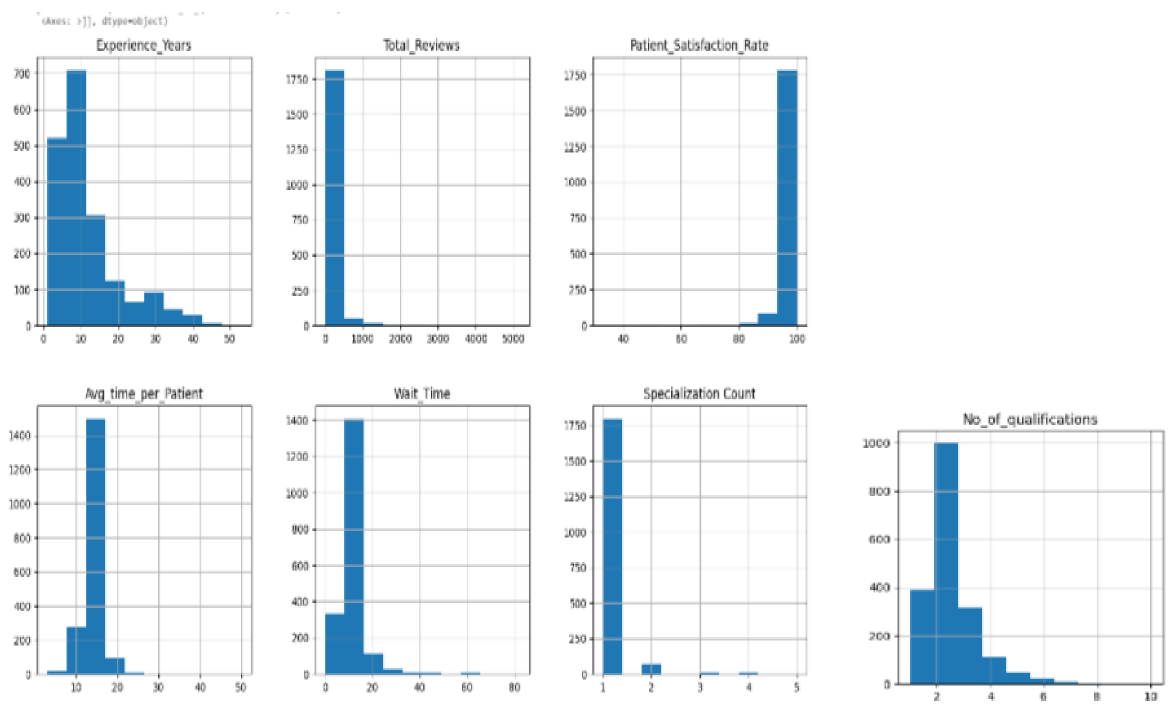


## 5.1 Numerical Values

```
num_cols = X_train.select_dtypes(include=['int','float']).columns
num_cols
```

```
Index(['Experience_Years', 'Total_Reviews', 'Patient_Satisfaction_Rate',
       'Avg_time_per_Patient', 'Wait_Time', 'Specialization Count',
       'No_of_qualifications'],
      dtype='object')
```




we concluded that

1. experience is somehow normally distributed
2. total_reviews is right skewed and the max is > 2000 & min is almost 0
3. try log to cancel skewness df['total_rooms'] = np.log(df['total_rooms'] +1)

## /////////// lesa btt3ml

## 5.2 Hospital Address

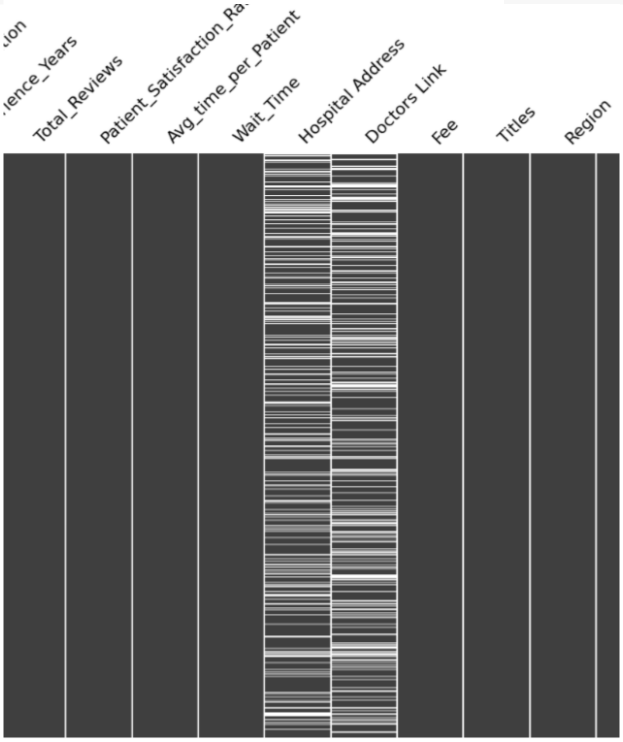
## feature engnieering has_hospital address

Those who have address seems to be easier to patient to go to so avg fees of drs are more

```
print(filtered_doctors[['Doctor Name']])

                Doctor Name
1             Haris Shakeel
8               Awais Ahmad
27          Anwisha Samreen
44                Saad Arif
48              Komal Azhar
...                     ...
2329             Maria Ijaz
2344             Asim Niaz
2365          Umber Mushtaq
2367  Hamraz Khan Yousaf Zai
2379         Muhammad Ikrama

[254 rows x 1 columns]
```

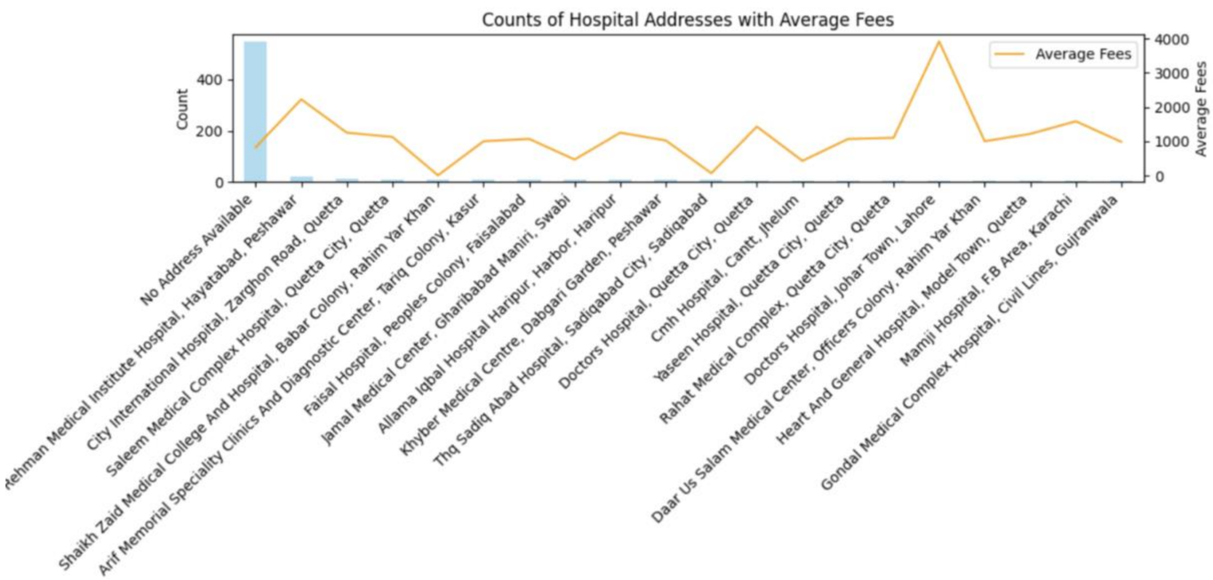
Drs that don’t have both link nor hospital address `254`

`Wanted to see relation on msno`



## 5.1 Analysis

Highest avg fees in Doctors Hospital, Johar Town,Lahore hase the highest avg fees , it appeared 6 times in df




Doctors with the highest fees at Doctors Hospital, Johar Town, Lahore :

| | Doctor Name | Fee |
|---|---|---|
| 46 | Ghazanfar Ali Shah | 5000 |
| 96 | Qurat Ul Ain Sajida | 3500 |
| 839 | Syed Shahzad Hussain Shah | 3000 |
| 987 | Tariq Sohail | 4000 |
| 1767 | Muhammad Bilal | 3000 |
| 1959 | Khurshid Alam | 5000 |

## 6.Doctors Link

| | count | unique | top | freq |
|---|---|---|---|---|
| Doctors Link | 2293 | 1567 | No Link Available | 645 |

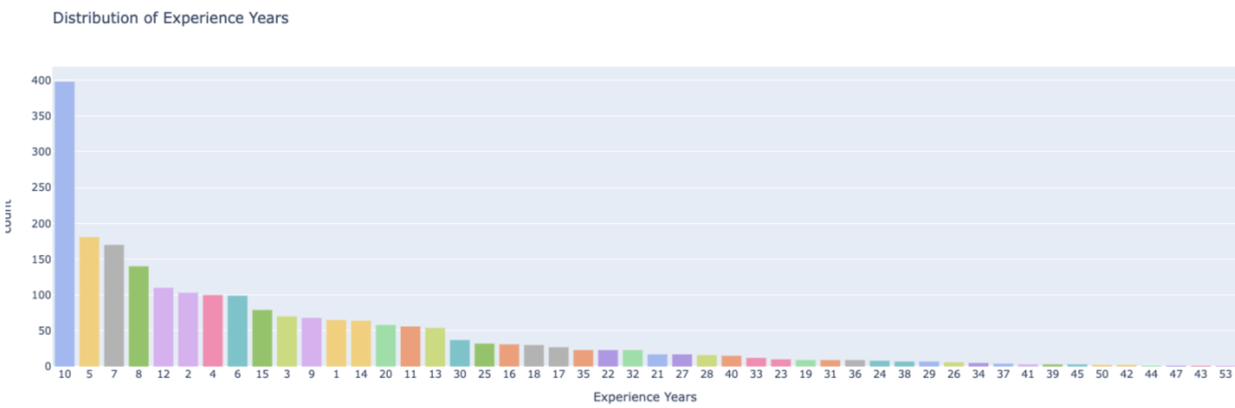
Replaced no link av by homepage `df['Doctors Link'] = df['Doctors Link'].replace('No Link Available', 'https://instacare.pk/')`

# 7. Experience_Years & Feature Engineering (Experience_Group)

```
X_train[['Experience_Years']].describe().T
```

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **Experience_Years** | 1898.0 | 11.753161 | 8.76123 | 1.0 | 6.0 | 10.0 | 14.0 | 53.0 |

most drs have experience 10 yrs

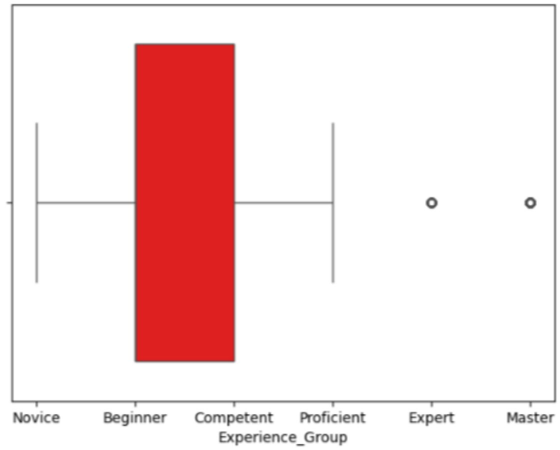


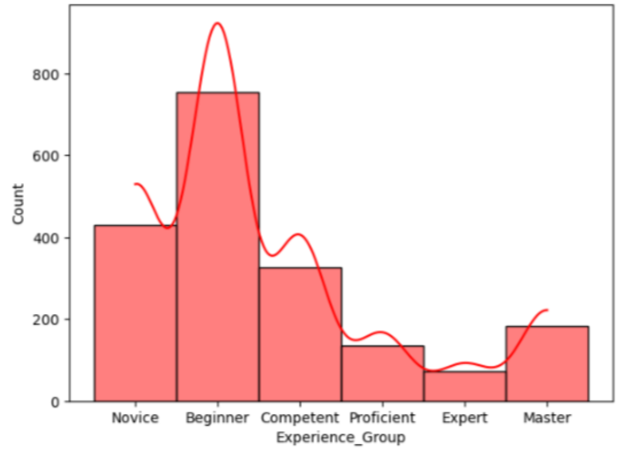

concluded that there are drs that have experience yrs 1.5 and 4.5 so will round them

```
def binning_Experience(df):
  df['Experience_Years'] = df['Experience_Years'].round()
  bins = [0, 5, 10, 15, 20, 25, float('inf')]
  labels = ['Novice', 'Beginner', 'Competent', 'Proficient', 'Expert', 'Master']
  df['Experience_Group'] = pd.cut(df['Experience_Years'], bins=bins, labels=labels, right=True)

binning_Experience(X_train)
```

Apply round to years, then based on the number of years, we categorize them into one of these labels.

```
Minimum Novice
Maximum Master
```

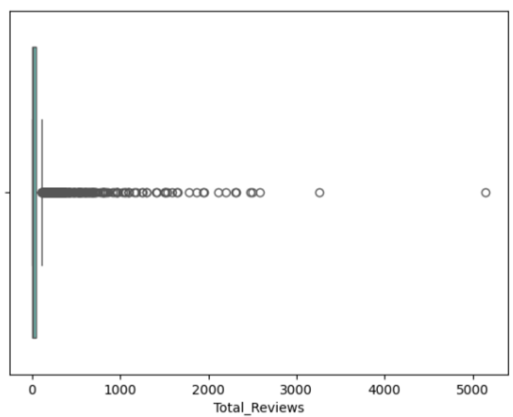


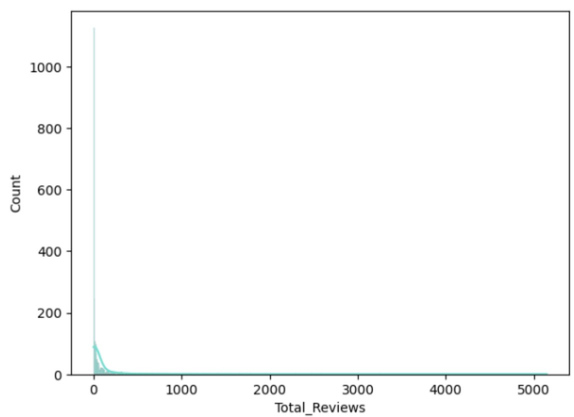




## ///lesaa

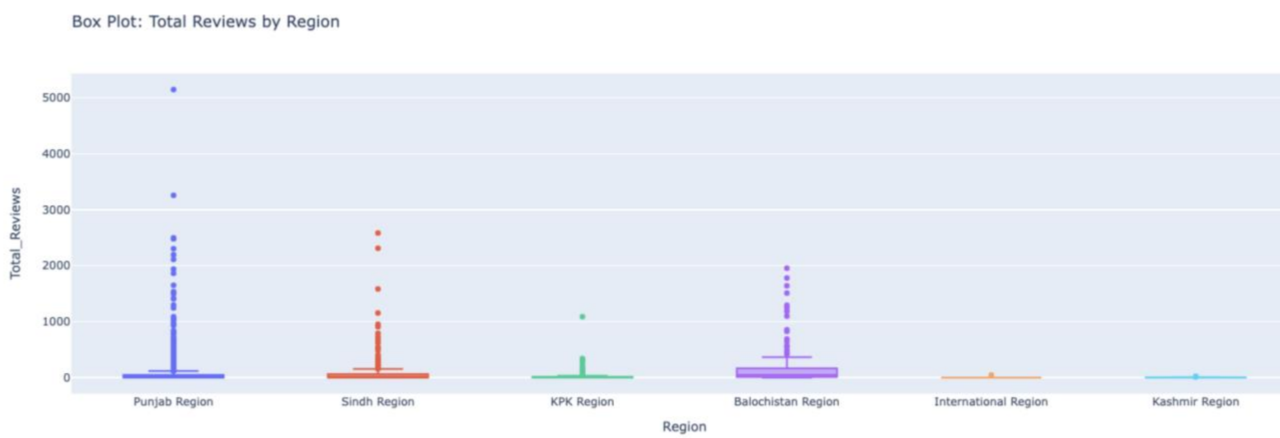
## 8.Total_Reviews

```
Minimum 0
Maximum 5147
```



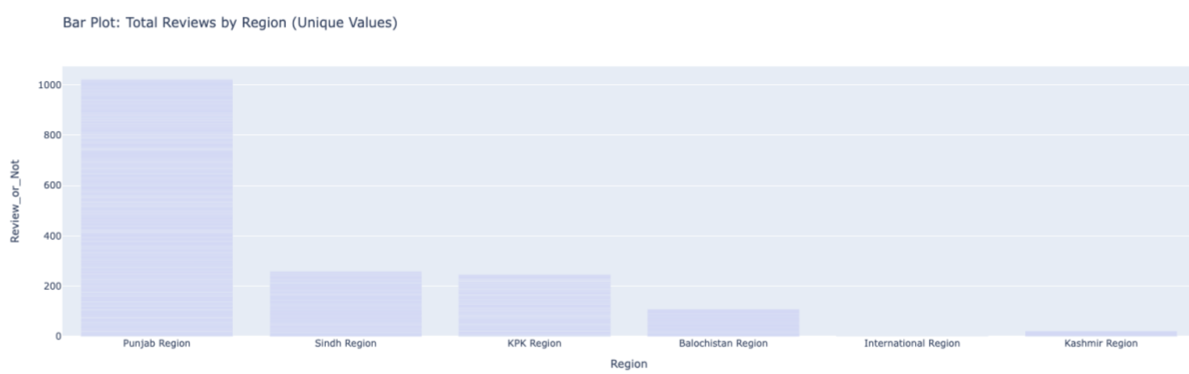

High diversion
most of reviews are 0

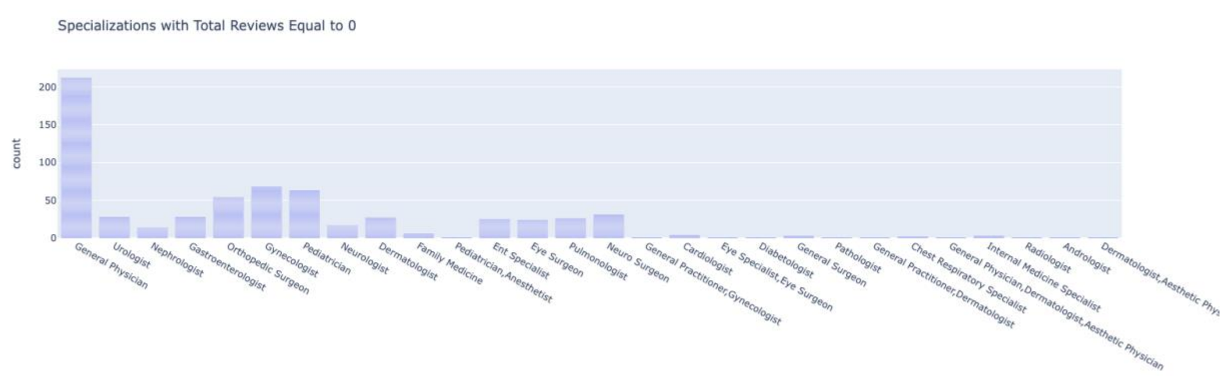


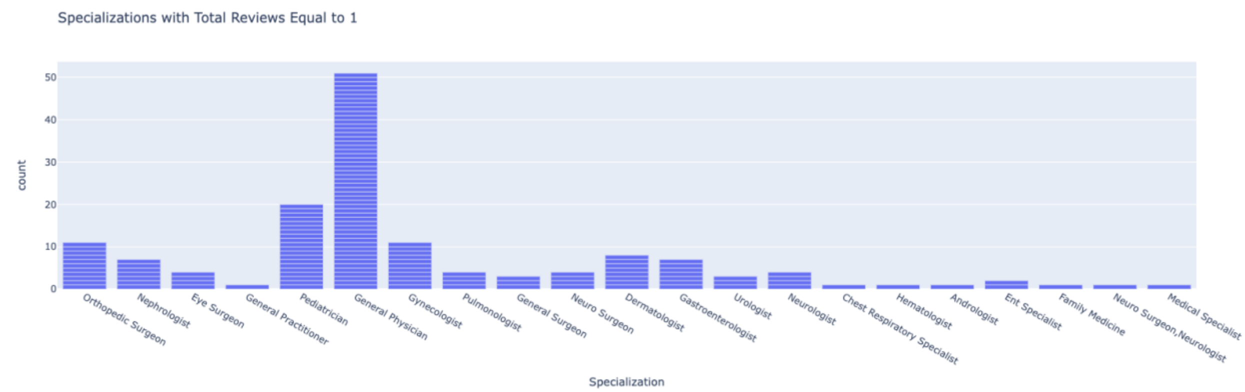

# ////lesa

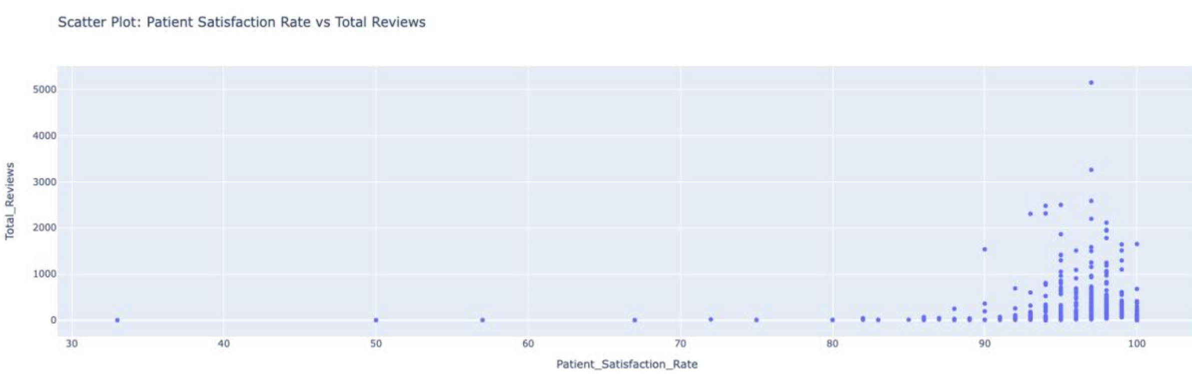
# 8.1Total_Reviews  feature engnieering

```
df['Review_or_Not'] = (df['Total_Reviews'] > 0).astype(int)
```





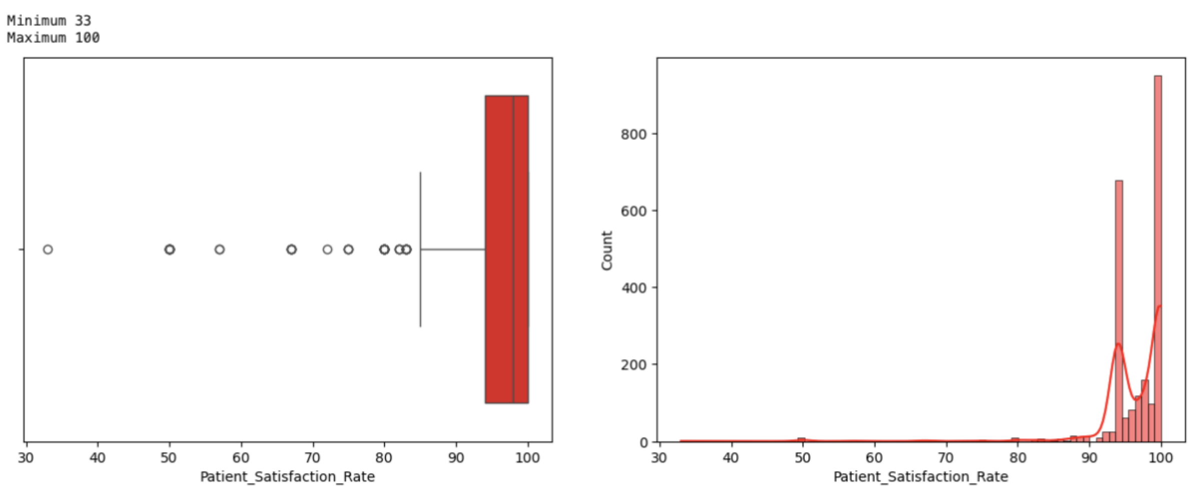

As patient satsfactiion rate increase total reviews increase too

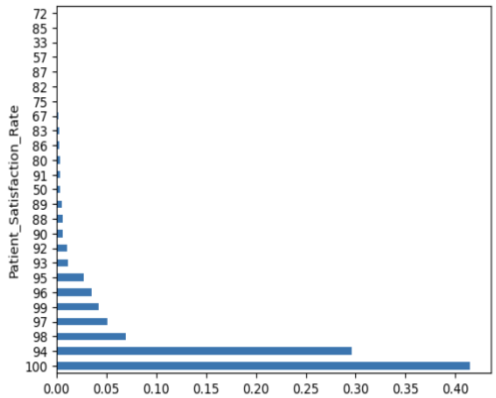
## ////lesaaa

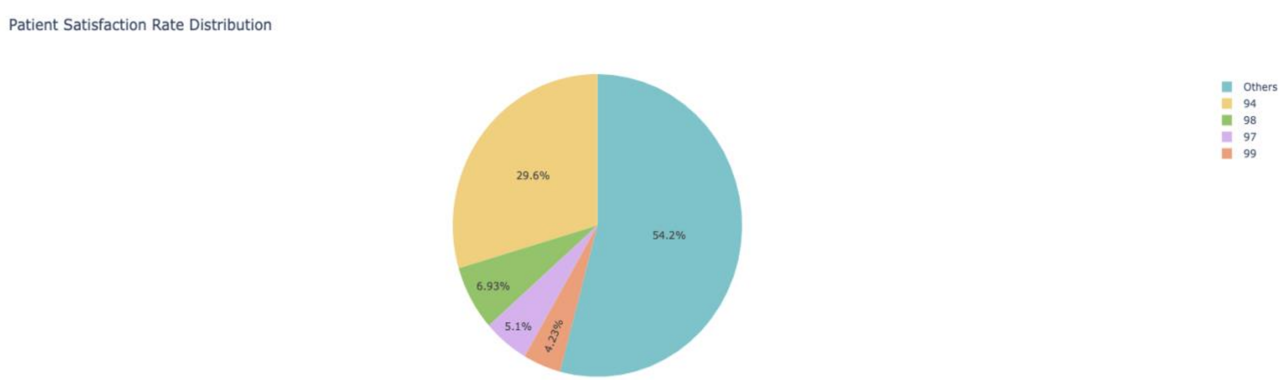
## 9.Patient Satisfaction Rate(%age)

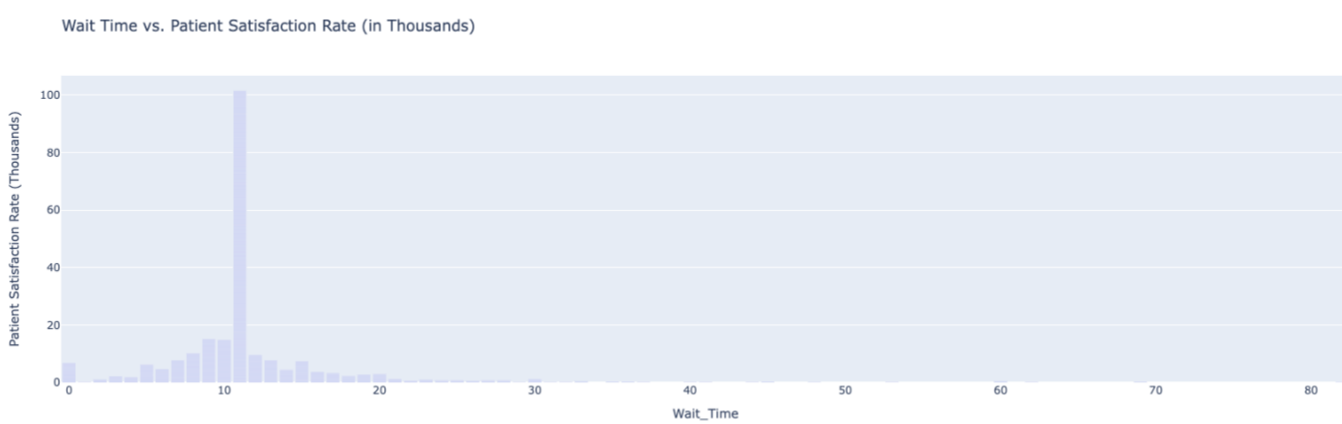
Number of unique values: 25

---





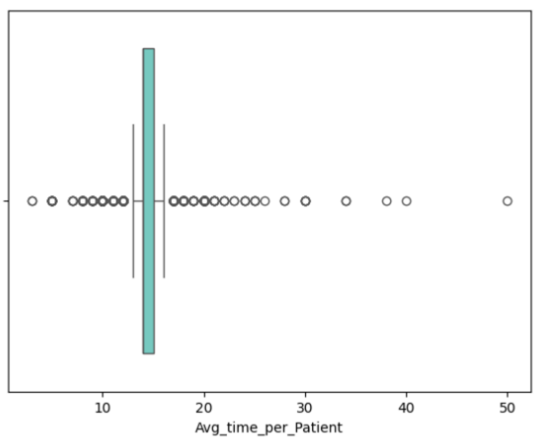



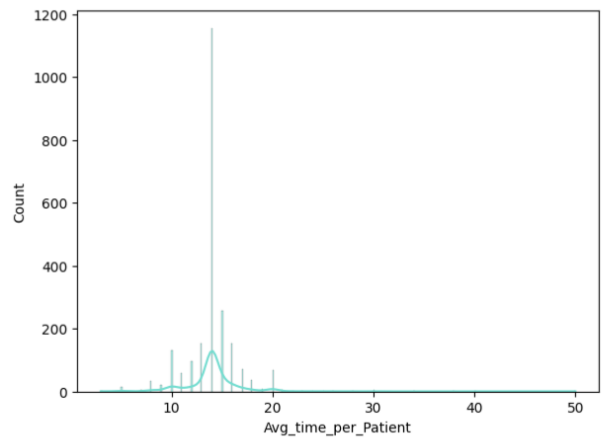

if the waiting time for the dr but the dr is good so it is worth the wait so it doesnt affect

# /// lesaaa

# 10.Avg Time to Patients(min)

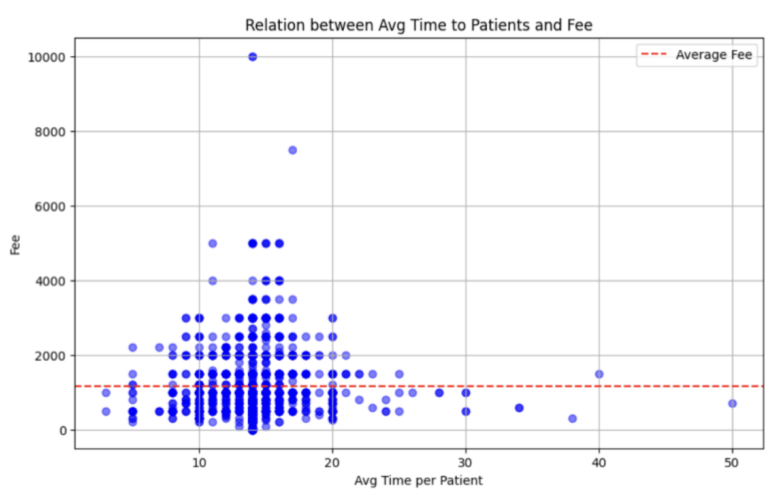
```
Minimum 3
Maximum 50
```




---

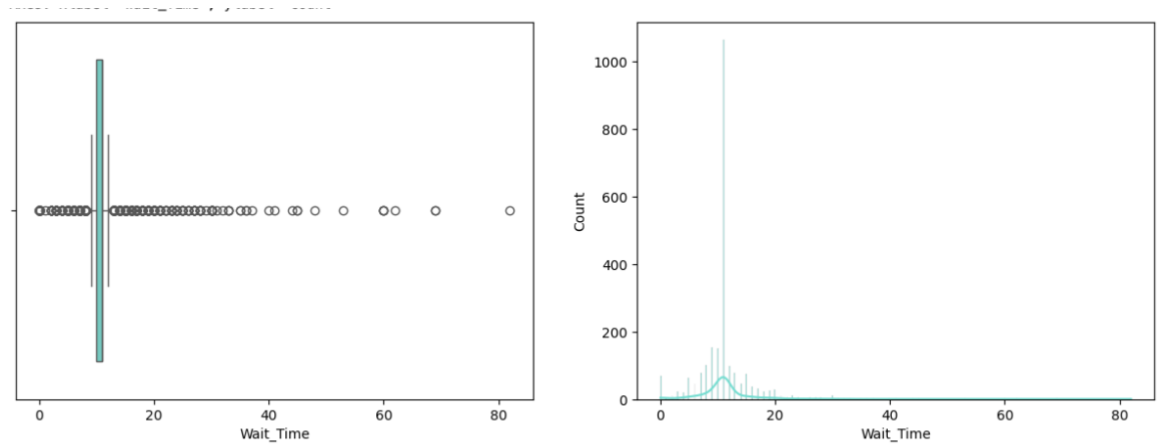
the majority of the patients spend of average less than 15 mins



# 11.Wait Time(mins)

max val =  82

min val =  0




## Feature Engineering (Total Time)

```
df['Total Time'] = df['Avg_time_per_Patient'] + df['Wait_Time']
```

# 13 . Log Transformation with outliers

(before)

```
for col in num_cols:
    lower_limit, upper_limit = calculate_outliers(X_train[col])
    lower_outliers = len(X_train[X_train[col] < lower_limit])
    upper_outliers = len(X_train[X_train[col] > upper_limit])
    total_outliers = lower_outliers + upper_outliers
    outlier_percentage = (total_outliers / X_train.shape[0]) * 100

    print(f"Total outliers in column before log transformation {col}: {total_outliers}, Percentage: {outlier_percentage}%")
    print('-'*100)
```

```
Total outliers in column before log transformation Experience_Years: 175, Percentage: 9.220231822971549%
----------------------------------------------------------------------------------------------------
Total outliers in column before log transformation Total_Reviews: 282, Percentage: 14.857744994731295%
----------------------------------------------------------------------------------------------------
Total outliers in column before log transformation Patient_Satisfaction_Rate: 26, Percentage: 1.36986301369863%
----------------------------------------------------------------------------------------------------
Total outliers in column before log transformation Avg_time_per_Patient: 457, Percentage: 24.077976817702844%
----------------------------------------------------------------------------------------------------
Total outliers in column before log transformation Wait_Time: 676, Percentage: 35.61643835616438%
----------------------------------------------------------------------------------------------------
Total outliers in column before log transformation Specialization Count: 101, Percentage: 5.321390937829294%
----------------------------------------------------------------------------------------------------
Total outliers in column before log transformation No_of_qualifications: 80, Percentage: 4.214963119072708%
----------------------------------------------------------------------------------------------------
Total outliers in column before log transformation Total Time: 483, Percentage: 25.447839831401474%
----------------------------------------------------------------------------------------------------
```

(after)

```
#9
def apply_log_transform(df, columns):
    for col in columns:
        df[col] = np.log1p(df[col])

# columns_to_transform = ['Experience_Years', 'Patient_Satisfaction_Rate',
#                         'Avg_time_per_Patient', 'Wait_Time', 'Total Time','Total_Reviews']

apply_log_transform(X_train, num_cols)
```

```
for col in num_cols:
    lower_limit, upper_limit = calculate_outliers(X_train[col])
    lower_outliers = len(X_train[X_train[col] < lower_limit])
    upper_outliers = len(X_train[X_train[col] > upper_limit])
    total_outliers = lower_outliers + upper_outliers
    outlier_percentage = (total_outliers / df.shape[0]) * 100

    print(f"Total outliers in column after log transformation {col}: {total_outliers}, Percentage: {outlier_percentage}%")
    print('-'*100)
```

```
Total outliers in column after log transformation Experience_Years: 61, Percentage: 2.5705857564264645%
----------------------------------------------------------------------------------------------------
Total outliers in column after log transformation Total_Reviews: 0, Percentage: 0.0%
----------------------------------------------------------------------------------------------------
Total outliers in column after log transformation Patient_Satisfaction_Rate: 27, Percentage: 1.1378002528445006%
----------------------------------------------------------------------------------------------------
Total outliers in column after log transformation Avg_time_per_Patient: 457, Percentage: 19.258322798145814%
----------------------------------------------------------------------------------------------------
Total outliers in column after log transformation Wait_Time: 676, Percentage: 28.487147071217866%
----------------------------------------------------------------------------------------------------
Total outliers in column after log transformation Specialization Count: 101, Percentage: 4.256215760640539%
----------------------------------------------------------------------------------------------------
Total outliers in column after log transformation No_of_qualifications: 33, Percentage: 1.390644753476612%
----------------------------------------------------------------------------------------------------
Total outliers in column after log transformation Total Time: 540, Percentage: 22.756005056890015%
----------------------------------------------------------------------------------------------------
```

All outliers decreased after applying the log transformation.

# 12. Scaling Data

```
num_cols = ['Experience_Years', 'Total_Reviews', 'Patient_Satisfaction_Rate','Avg_time_per_Patient', 'Wait_Time',
            'Specialization Count','No_of_qualifications', 'Total Time']

scaler1 = StandardScaler()

# Fit the scaler on training data
X_train_scaled = scaler1.fit_transform(X_train[num_cols])
```

# 13.Encoding Data

```
categorical_columns = X_train.select_dtypes(include=['object','category']).columns
categorical_columns
```

```
Index(['Doctor Name', 'City', 'Specialization', 'Doctor Qualification',
       'Hospital Address', 'Doctors Link', 'Titles', 'Region',
       'Experience_Group'],
      dtype='object')
```

Categorical cols that should to encoding in training data

## * Specialization:

We Extracted top 20 from Specialization column and contain the rest in other column

```
top_20_specializations = df['Specialization'].value_counts().head(20)
X_train['Specialization'] = X_train['Specialization'].apply(lambda x: x if x in top_20_specializations.index else 'Others')
```

Then we use one hot encoder for it

```
onehot_encoded = pd.get_dummies(X_train['Specialization']).astype(int)

df_encoded = pd.concat([X_train, onehot_encoded], axis=1)
df_encoded
```

## * Doctor Qualification, Experience Group , title , city and region

We used Target encoder for all of them

```
qualification_encoder = ce.TargetEncoder(cols=['Doctor Qualification'])

X_train['Doctor Qualification'] = qualification_encoder.fit_transform(X_train['Doctor Qualification'], y_train['Fee Category'])
```

```
Experience_Group_encoder = ce.TargetEncoder(cols=['Experience_Group'])

X_train['Experience_Group'] = Experience_Group_encoder.fit_transform(X_train['Experience_Group'], y_train['Fee Category'])
```

```
titles_encoder = ce.TargetEncoder(cols=['Titles'])

X_train['Titles_encoded'] = titles_encoder.fit_transform(X_train['Titles'], y_train['Fee Category'])

X_train['Titles'] = X_train['Titles_encoded']
X_train.drop(columns=['Titles_encoded'], inplace=True)
```

```
city_encoder = ce.TargetEncoder(cols=['City'])

X_train['City'] = city_encoder.fit_transform(X_train['City'], y_train['Fee Category'])
```

```
Region_encoder = ce.TargetEncoder(cols=['Region'])

X_train['Region'] = Region_encoder.fit_transform(X_train['Region'], y_train['Fee Category'])
```

## *Hospital Address and Doctor Link

```
X_train['hospital_encoded'] = X_train['Hospital Address'].apply(lambda x: 0 if x == 'No Address Available' else 1)
```

There is some rows with no address so we put 1 if there is address and if No Address Available we put 0

```
X_train['Doctors Link_encoded'] = X_train['Doctors Link'].apply(lambda x: 0 if x == 'No Link Available' else 1)
```

If no link available 0 else 1

*Target Column(Fee Category)

Encoding target column using ordinal encoder

## Fee Category

```
[ ]  y_train = pd.DataFrame(y_train, columns=['Fee Category'])
```

```
fee_encoder = OrdinalEncoder()

y_train['Fee Category Encoded'] = fee_encoder.fit_transform(y_train[['Fee Category']])

y_train
```

| | Fee Category | Fee Category Encoded |
|---|---|---|
| 1754 | Cheap | 0.0 |
| 1843 | Expensive | 1.0 |
| 1029 | Expensive | 1.0 |

# 14.preprocessing for X_test

```
#Doctor name cleaning
X_test[['Titles', 'Cleaned Name']] = X_test['Doctor Name'].apply(lambda x: pd.Series(extract_titles_and_clean_name(x)))

X_test['Doctor Name'] = X_test['Cleaned Name']
X_test.drop(columns='Cleaned Name', inplace = True)

X_test["Titles"].unique()
```

```
array(['Prof, Dr', 'Dr', 'Asst Prof Dr', 'Assoc Prof Dr', 'others'],
      dtype=object)
```

```
#City
X_test['City'] = X_test['City'].str.replace('-', ' ').str.title()
X_test['City'].unique()
```

```
array(['Peshawar', 'Khanpur', 'Faisalabad', 'Hyderabad', 'Burewala',
       'Okara', 'Quetta', 'Swabi', 'Bahawalpur', 'Bhalwal', 'Jhelum',
       'Hafizabad', 'Gujranwala', 'Dera Ghazi Khan', 'Lodhran', 'Gilgit',
       'Jaranwala', 'Charsadda', 'Lahore', 'Istanbul', 'Gujrat',
       'Rahim Yar Khan', 'Pasrur', 'Islamabad', 'Abbottabad', 'Jamshoro',
       'Multan', 'Karachi', 'Sahiwal', 'Mansehra', 'Kharian', 'Nowshera',
       'Shahdadpur', 'Sargodha', 'Gujar Khan', 'Mardan', 'Kasur', 'Kohat',
       'Mirpur Khas', 'Swat', 'Muzaffar Garh', 'Daska', 'Khuzdar',
       'Mandi Bahauddin', 'Bhakkar', 'Jacobabad', 'Haripur', 'Layyah',
       'Attock', 'Jhang', 'Mirpur', 'Dunyapur', 'Malakand', 'Wah Cantt',
       'Vehari', 'Timergara', 'Sialkot', 'Dijkot', 'Sheikhupura',
       'Talagang', 'Sukkur', 'Sadiqabad', 'Nankana Sahib', 'Mianwali',
       'Shorkot', 'Wazirabad', 'Bannu', 'Khanewal', 'Khairpur',
       'Pakpattan', 'Toba Tek Singh', 'Chichawatni', 'Jauharabad',
       'Dera Ismail Khan', 'Muridke', 'Kot Addu', 'Turbat', 'Narowal',
       'Bahawalnagar', 'Rawalakot', 'Chiniot', 'Gojra', 'Chakwal',
       'Dinga', 'Alipur'], dtype=object)
```

```
#Specialization
all_specialization_preprocessing(X_test)
```

```
X_test = clean_qualifications(X_test)
```

```
numerical_columns=['Experience_Years', 'Total_Reviews', 'Patient_Satisfaction_Rate', 'Avg_time_per_Patient',
                   'Wait_Time','No_of_qualifications','Specialization Count','Total Time']
```

We use the same preprocessing we used in x_train but based on training data

```python
def all_Feature_engineering(df):
  assign_region_clean_city(df)
  calc_Specialization_count(df)
  calc_No_of_qualifications(df)
  binning_Experience(df)
  calc_Total_time(df)

all_Feature_engineering(X_test)
```

```python
cols_to_scale = ['Experience_Years', 'Total_Reviews', 'Patient_Satisfaction_Rate','Avg_time_per_Patient',
                 'Wait_Time', 'Specialization Count','No_of_qualifications', 'Total Time']
```

```python
X_test.columns
```

```
Index(['Doctor Name', 'City', 'Specialization', 'Doctor Qualification',
       'Experience_Years', 'Total_Reviews', 'Patient_Satisfaction_Rate',
       'Avg_time_per_Patient', 'Wait_Time', 'Hospital Address', 'Doctors Link',
       'Titles', 'Region', 'Specialization Count', 'No_of_qualifications',
       'Experience_Group', 'Total Time'],
      dtype='object')
```

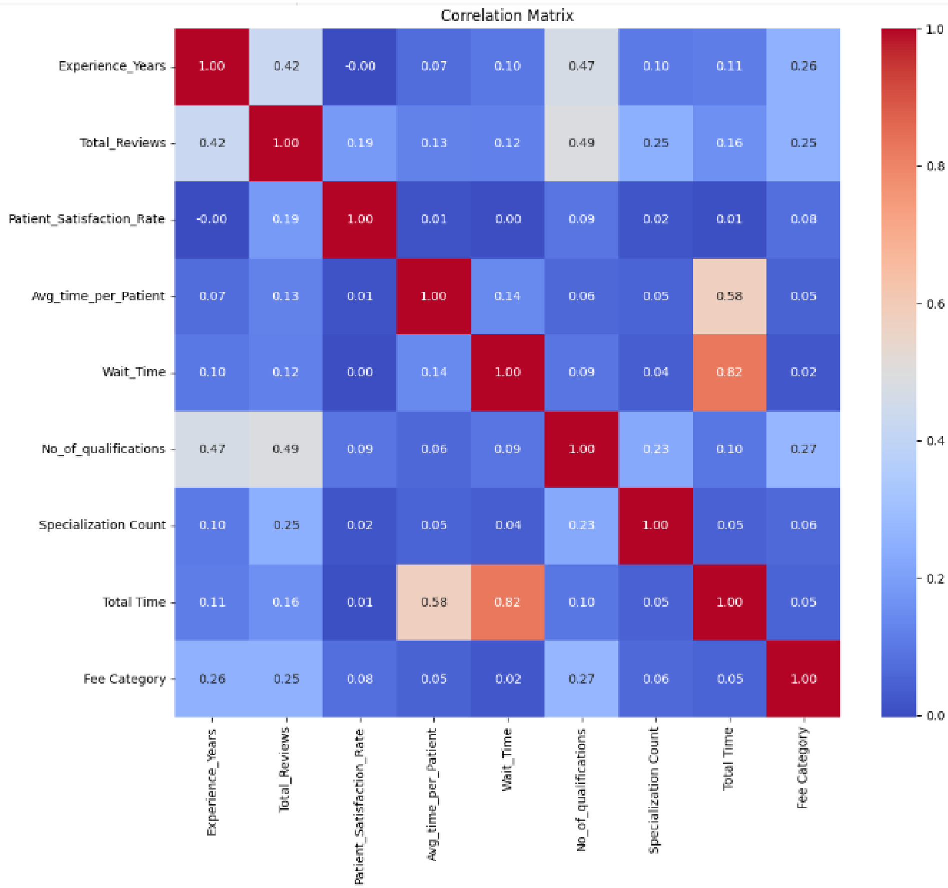
```python
num_cols = ['Experience_Years', 'Total_Reviews', 'Patient_Satisfaction_Rate','Avg_time_per_Patient', 'Wait_Time',
            'Specialization Count','No_of_qualifications', 'Total Time']

X_test_scaled = scaler1.transform(X_test[num_cols])
```

Encoding test data based on training

```python
X_test['Specialization'] = X_test['Specialization'].apply(lambda x: x if x in top_20_specializations.index else 'Others')
spec_onehot_encoded = pd.get_dummies(X_test['Specialization']).astype(int)
X_test = pd.concat([X_test, spec_onehot_encoded], axis=1)
# df['Fee Category'] = fee_encoder.transform(df[['Fee Category']])
X_test['Doctor Qualification'] = qualification_encoder.transform(X_test['Doctor Qualification'])
X_test['Experience_Group'] = Experience_Group_encoder.transform(X_test['Experience_Group'])
X_test['Titles'] = titles_encoder.transform(X_test['Titles'])
X_test['Hospital Address'] = X_test['Hospital Address'].apply(lambda x: 0 if x == 'No Address Available' else 1)
X_test['Doctors Link'] = X_test['Doctors Link'].apply(lambda x: 0 if x == 'No Link Available' else 1)
X_test['Region'] = Region_encoder.transform(X_test['Region'])
X_test['City'] = city_encoder.transform(X_test['City'])
```

```python
y_test = pd.DataFrame(y_test, columns=['Fee Category'])

y_test['Fee Category'] = fee_encoder.transform(y_test[['Fee Category']])
```

# 14. Feature Selection



will drop reviews or not ass it gives same corr as total reviews

2If two variables are correlated, we can predict one from the other. Therefore, if two features are correlated, the model only needs one, as the second does not add additional information, Patient_Satisfaction_Rate

```
Cardinality ratio for 'Doctor Name' column: 0.9297863061491496
Cardinality ratio for 'City' column: 0.0510248582642826
Cardinality ratio for 'Specialization' column: 0.0453554295682512
Cardinality ratio for 'Doctor Qualification' column: 0.36022677714784124
Cardinality ratio for 'Hospital Address' column: 0.5063235935455734
Cardinality ratio for 'Doctors Link' column: 0.6833842128216311
Cardinality ratio for 'Titles' column: 0.0017444395987788923
Cardinality ratio for 'Region' column: 0.0026166593981683385
Cardinality ratio for 'Experience_Group' column: 0.0030527692978630614
```

1. **High Cardinality Ratios (> 0.5)**:
    - Columns like 'Doctor Name' and 'Hospital Address' have a high proportion of unique values relative to the dataset size.
    - High cardinality ratios may pose challenges for certain machine learning models, particularly those sensitive to high dimensionality.
2. **Moderate Cardinality Ratios (0.3 - 0.5)**:
    - The 'Doctor Qualification' column falls into this category, indicating a moderate number of unique values compared to the dataset size.
    - These columns may still be useful for encoding into numerical features or for grouping similar categories.
3. **Low Cardinality Ratios (< 0.1)**:

- Columns such as 'City', 'Specialization', 'Titles', 'Region', 'Experience_Group', and 'Satisfaction_Category' exhibit low cardinality ratios.
- Such columns are suitable candidates for various encoding techniques, including one-hot encoding, label encoding, or target encoding.

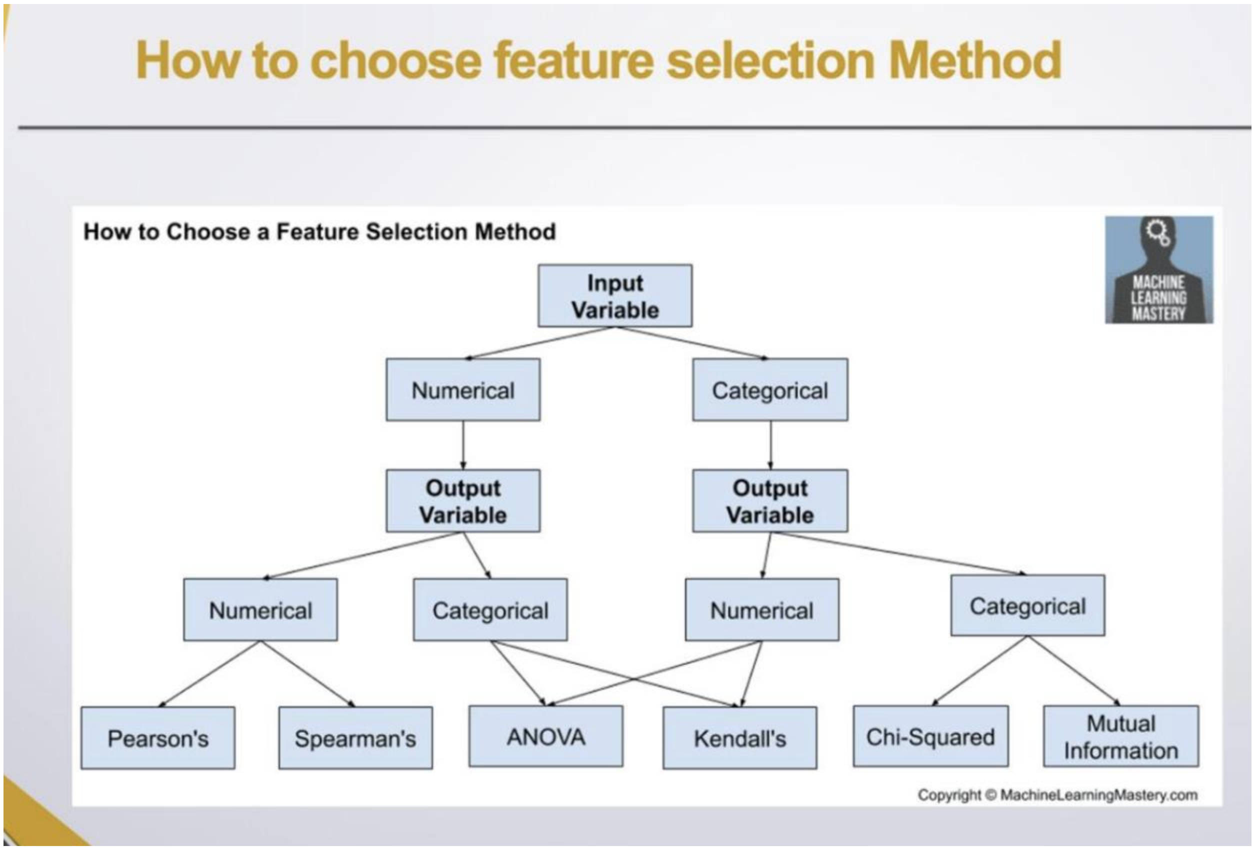
Surly we will drop Doctor Name as CR is almost 1 & Hospital Address, Doctors Link

Concatenated avg time per patient and wait time to total time so will drop too

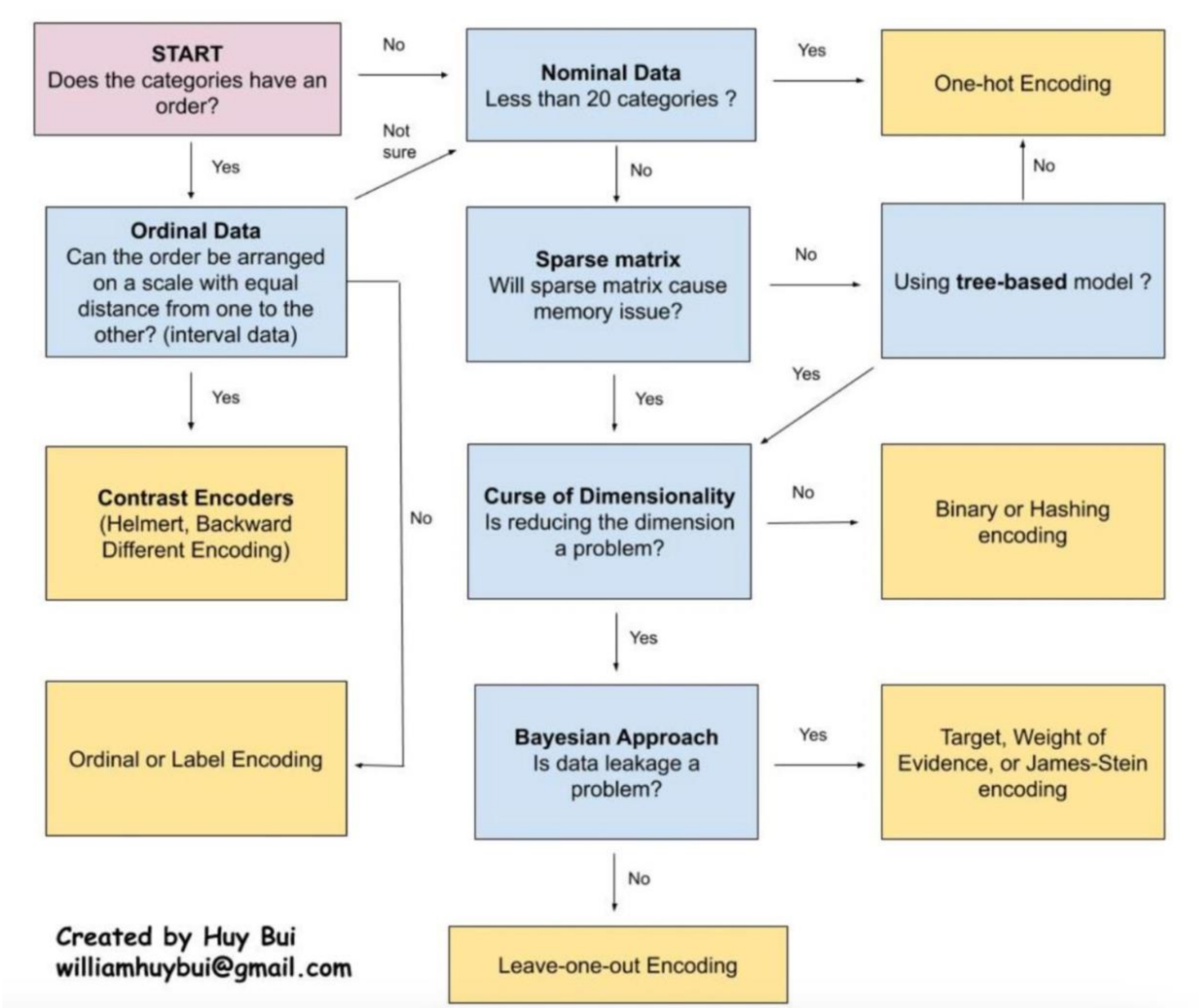
ideally, we want the skewness to be as close to zero as possible, indicating that the data is symmetrically distributed.

If the dataset follows the theoretical distribution perfectly, the points on the Q-Q plot will fall along a straight line. Deviations from the straight line indicate departures from the theoretical distribution.

- The x-axis represents the quantiles of the theoretical distribution.
- The y-axis represents the quantiles of the dataset's distribution




## 14 Feature encoding


START
Does the categories have an order?
No
Not sure
Yes
Nominal Data
Less than 20 categories ?
Yes
One-hot Encoding
No
Ordinal Data
Can the order be arranged on a scale with equal distance from one to the other? (interval data)
Sparse matrix
Will sparse matrix cause memory issue?
No
Using tree-based model ?
No
Yes
Yes
Yes
Contrast Encoders
(Helmert, Backward Different Encoding)
No
Curse of Dimensionality
Is reducing the dimension a problem?
No
Binary or Hashing encoding
Yes
Ordinal or Label Encoding
Bayesian Approach
Is data leakage a problem?
Yes
Target, Weight of Evidence, or James-Stein encoding
No
Created by Huy Bui
williamhuybui@gmail.com
Leave-one-out Encoding