Report on COVID19 outcome

1.1 Problem Statement

The goal of this project is to predict a COVID19 outcome probability based on different input parameters that will be most related to target variable outcome. It is really important to find out whether the patients is death, recovered, treatment etc.

We have a huge dataset of data, where most features are categorical. I think that correct mean encoding should be important. Also the number of columns is quite high so it could be tempting to make some automatically processing for all columns. I personally think that it is important to analyze each variable and it could help to do a better processing

1.2 Methods

We choose 3 models for our classification problem because in our dataset target variable is in categorical format so, when class label is in categorical then this problem is related to classification. Models are given below that we have choose for our prediction.

- Decision Tree
- Logistic Regression
- Random Forest

We will measure the performance of each models on different metrics that shown below:

- **Accuracy**

  Measure to evaluate how accurate model's performance is:

$$\frac{TP + TN}{TP + FP + FN + FP}$$

- **Precision**

Measure to evaluate how accurate model's performance is:

$$\frac{TP}{TP + FP}$$

- **Recall**

Measure to evaluate how accurate model's performance is:

$$\frac{TP}{TP + FN}$$

- **F₁**

Provides information of both sides TN and TP

$$2 * \frac{Precision * Recall}{Precision + Recall}$$

$$where\ TP = True\ Positive$$

$$FP = False\ Positive$$

$$TN = True\ Negative$$

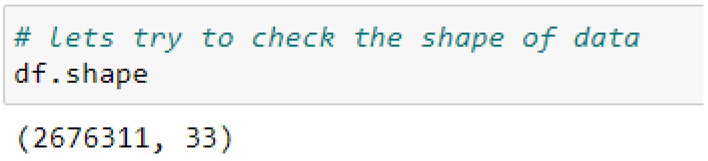
$$FN = False\ Negetive$$

- **Confusion Matrix**
- **Classification Report**

1.3 Dataset

We have a latest data of covid19. This is a huge dataset so, we will explore the dataset to know more about the data shape, descriptive analyses, correlation, missing values, dtypes, column names etc

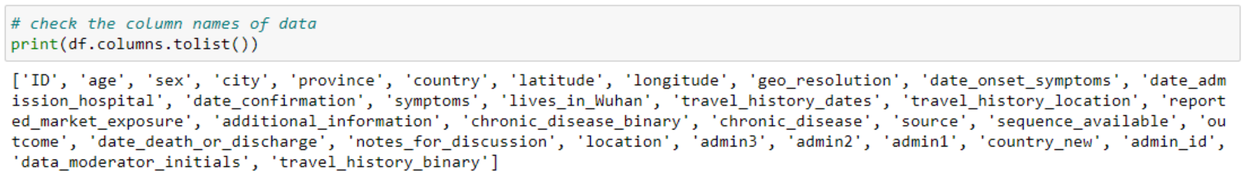
Lets try to look the shape of data:



```
# lets try to check the shape of data
df.shape
```

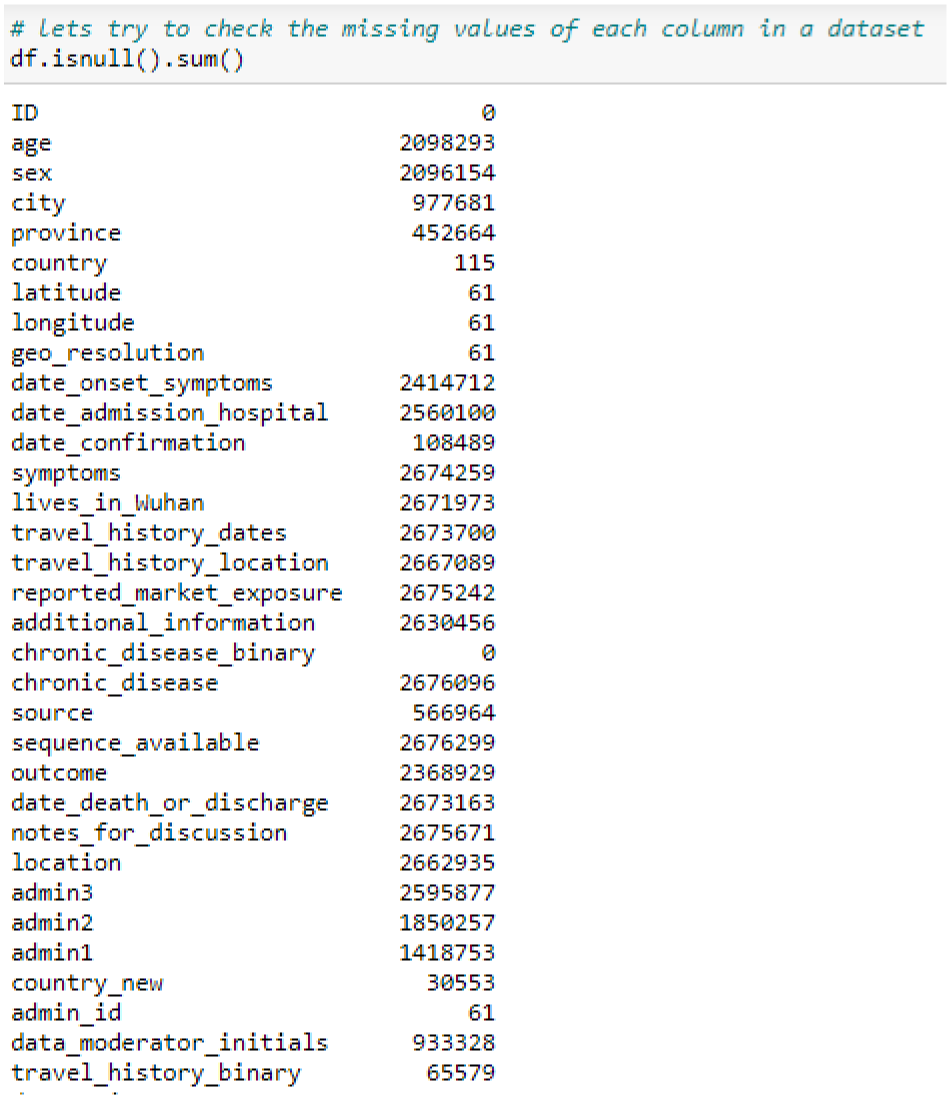
```
(2676311, 33)
```

From above that we have a data in which 2676311 rows and 33 columns.



```
# check the column names of data
print(df.columns.tolist())
```

```
['ID', 'age', 'sex', 'city', 'province', 'country', 'latitude', 'longitude', 'geo_resolution', 'date_onset_symptoms', 'date_adm
ission_hospital', 'date_confirmation', 'symptoms', 'lives_in_Wuhan', 'travel_history_dates', 'travel_history_location', 'report
ed_market_exposure', 'additional_information', 'chronic_disease_binary', 'chronic_disease', 'source', 'sequence_available', 'ou
tcome', 'date_death_or_discharge', 'notes_for_discussion', 'location', 'admin3', 'admin2', 'admin1', 'country_new', 'admin_id',
'data_moderator_initials', 'travel_history_binary']
```



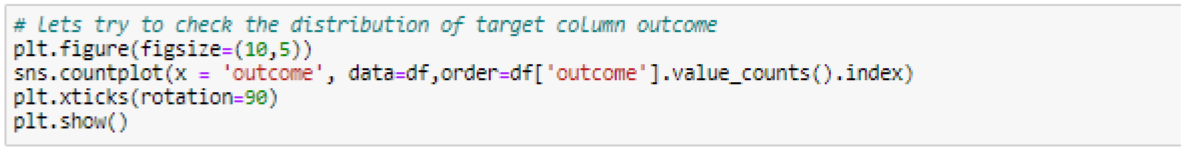
```python
# lets try to check the missing values of each column in a dataset
df.isnull().sum()
```

```
ID                                0
age                         2098293
sex                         2096154
city                         977681
province                     452664
country                         115
latitude                         61
longitude                        61
geo_resolution                   61
date_onset_symptoms         2414712
date_admission_hospital     2560100
date_confirmation            108489
symptoms                    2674259
lives_in_Wuhan              2671973
travel_history_dates        2673700
travel_history_location     2667089
reported_market_exposure    2675242
additional_information      2630456
chronic_disease_binary            0
chronic_disease             2676096
source                       566964
sequence_available          2676299
outcome                     2368929
date_death_or_discharge     2673163
notes_for_discussion        2675671
location                    2662935
admin3                      2595877
admin2                      1850257
admin1                      1418753
country_new                   30553
admin_id                         61
data_moderator_initials      933328
travel_history_binary         65579
```

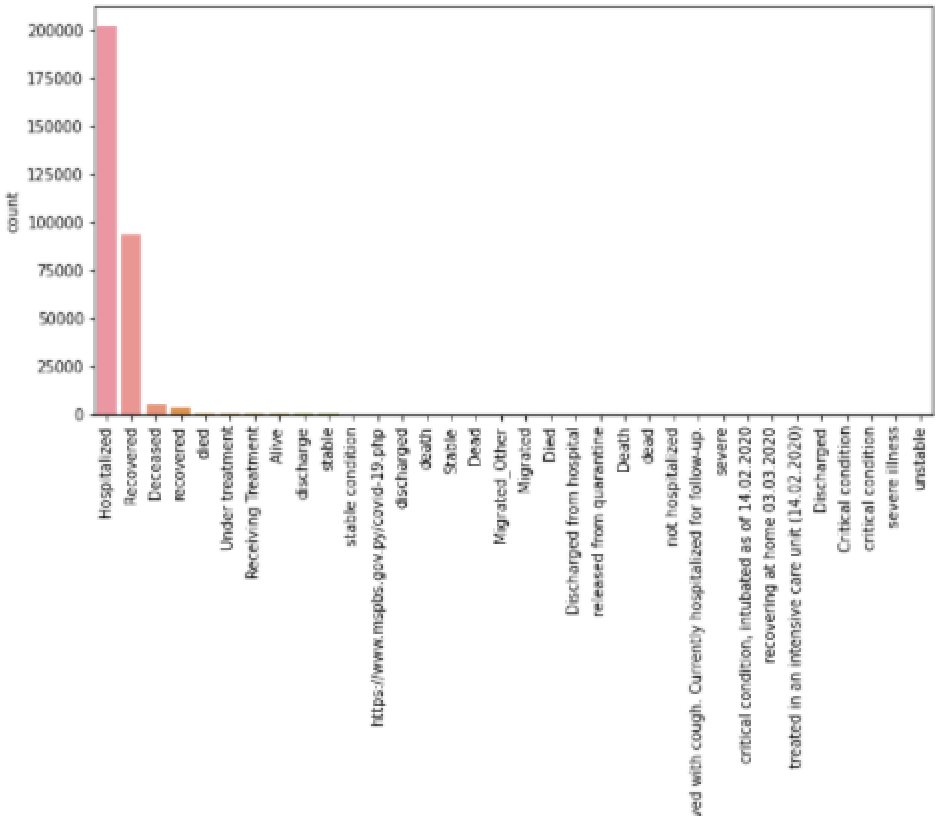
From above fig we can see that too much missing values in every columns. So , we will handles these missing values according our problem.

Data Visualization

In this section, we will visualize all the columns and check the distribution and relationship with others columns and also we will get more anaylisis from the graph.

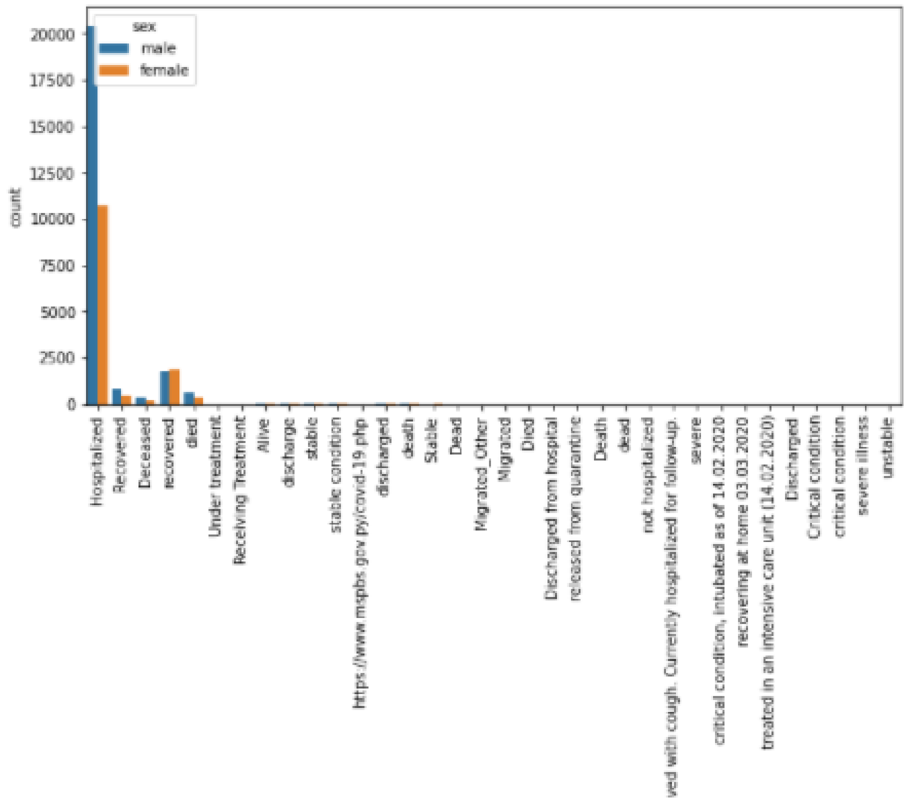


```
# Lets try to check the distribution of target column outcome
plt.figure(figsize=(10,5))
sns.countplot(x = 'outcome', data=df,order=df['outcome'].value_counts().index)
plt.xticks(rotation=90)
plt.show()
```

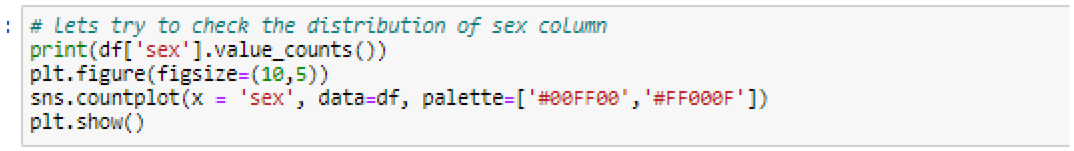


Form above Fig we plot a graph of each category count and we can see that mostly distribution of hospitalized and recovered, other categories are too low distribution.

```
# lets try to check the distribution of target column sex
plt.figure(figsize=(10,5))
sns.countplot(x = 'outcome', data=df,hue='sex',order=df['outcome'].value_counts().index)
plt.xticks(rotation=90)
plt.show()
```

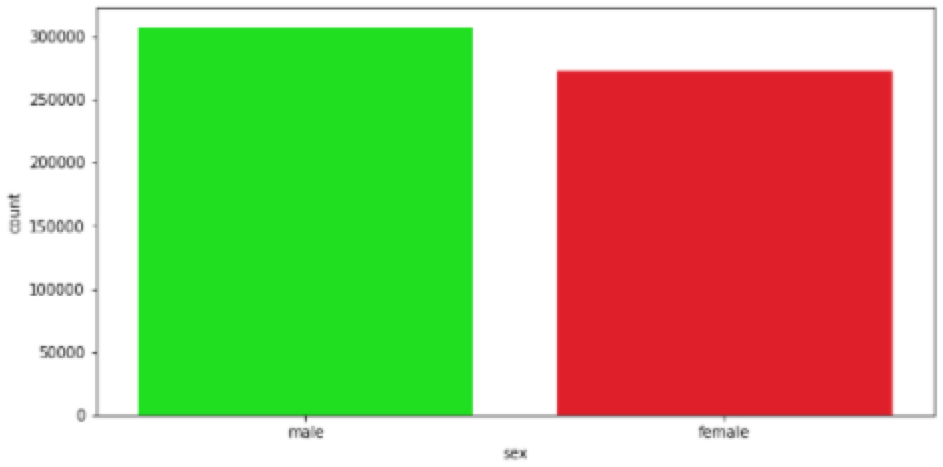


From above graph we check the distribution of outcome against sex column, so we can see that mostly male patients hospitalized as compare to female.

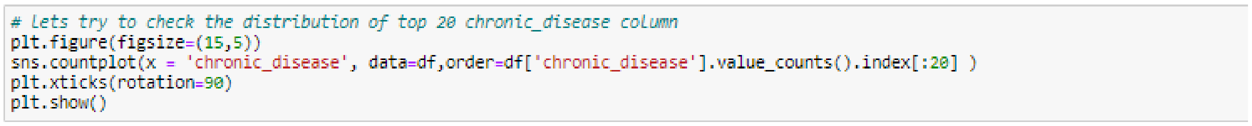


```
# Lets try to check the distribution of sex column
print(df['sex'].value_counts())
plt.figure(figsize=(10,5))
sns.countplot(x = 'sex', data=df, palette=['#00FF00','#FF000F'])
plt.show()
```

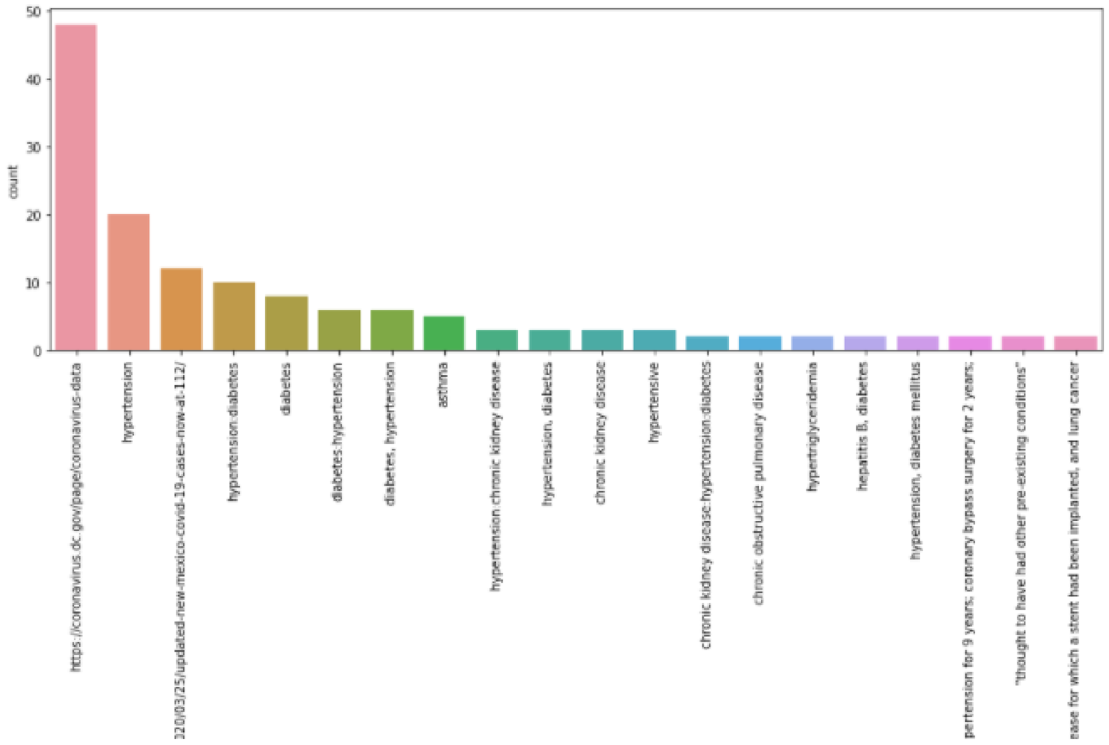
```
male      307188
female    272969
Name: sex, dtype: int64
```



From above graph we can see the distribution of male is greater as compare to female.

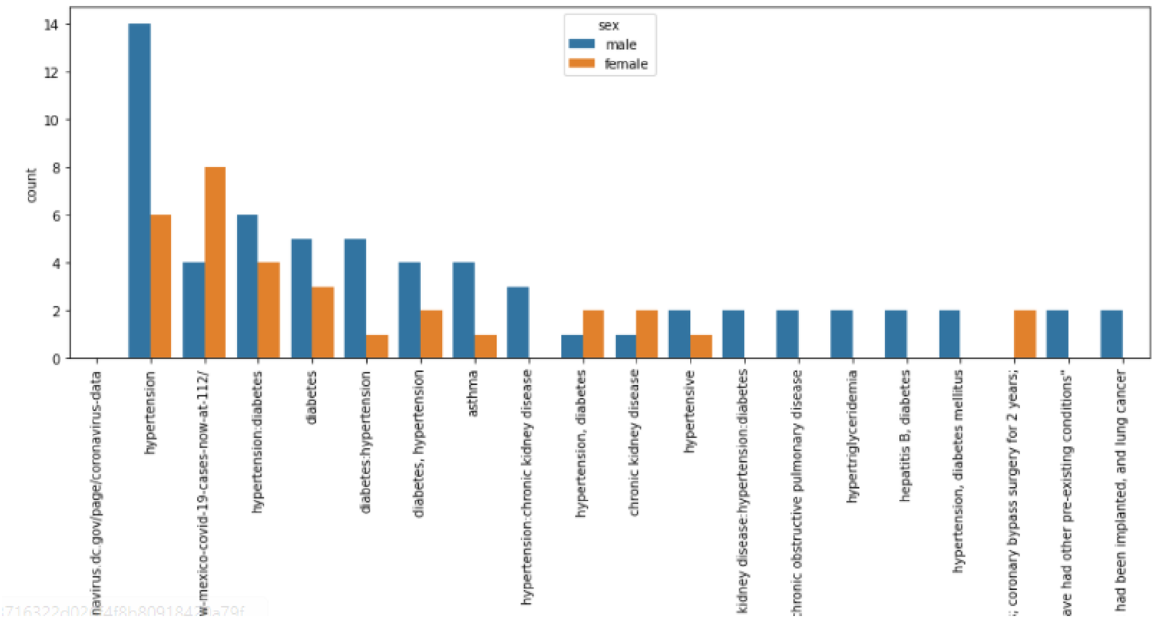


```
# lets try to check the distribution of top 20 chronic_disease column
plt.figure(figsize=(15,5))
sns.countplot(x = 'chronic_disease', data=df,order=df['chronic_disease'].value_counts().index[:20] )
plt.xticks(rotation=90)
plt.show()
```



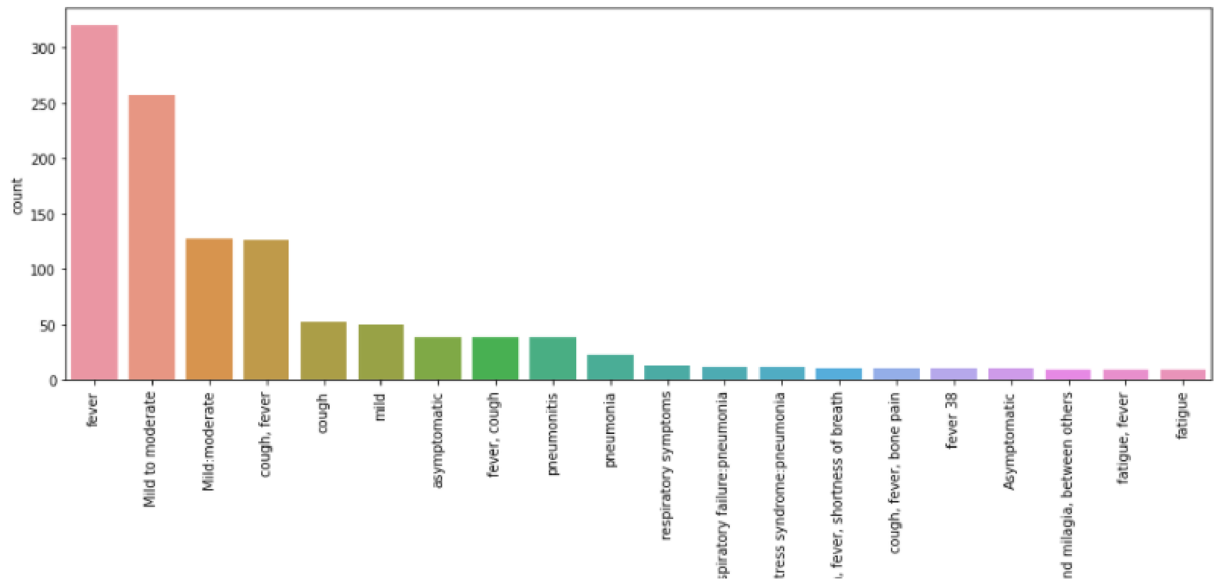
From above graph we plot the top 20 chronic diseases in which also we can see that some invalid values but we also see the distribution of hypertension and others chronic disease in graph.

```
# lets try to check the distribution of top 20 chronic_disease column with sex column
plt.figure(figsize=(15,5))
sns.countplot(x = 'chronic_disease', hue='sex',data=df,order=df['chronic_disease'].value_counts().index[:20] )
plt.xticks(rotation=90)
plt.show()
```



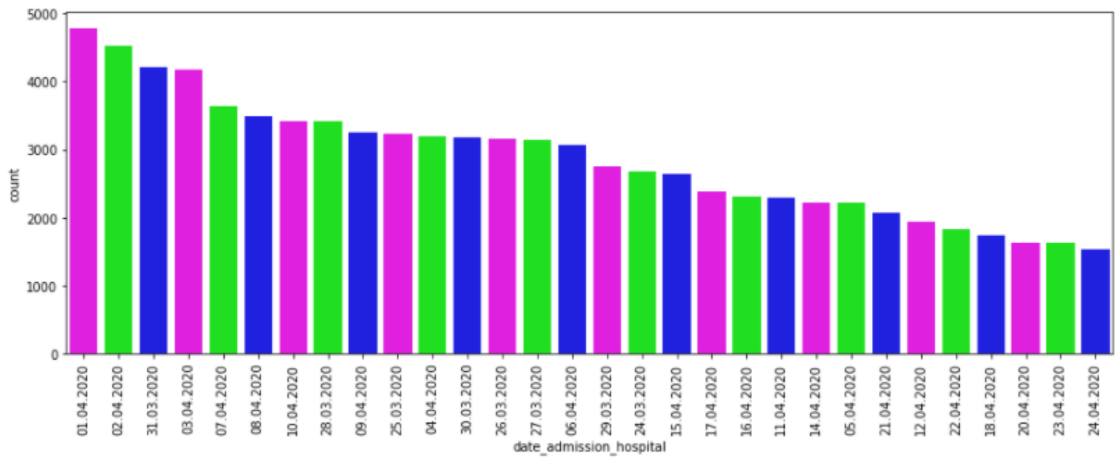
From above we also check the chronic disease against sex so, in this distribution male mostly affected on hypertension chronic disease.

```
# lets try to check the distribution of top 20 symptoms column

plt.figure(figsize=(15,5))
sns.countplot(x = 'symptoms', data=df, order=df['symptoms'].value_counts().index[:20])
plt.xticks(rotation=90)
plt.show()
```



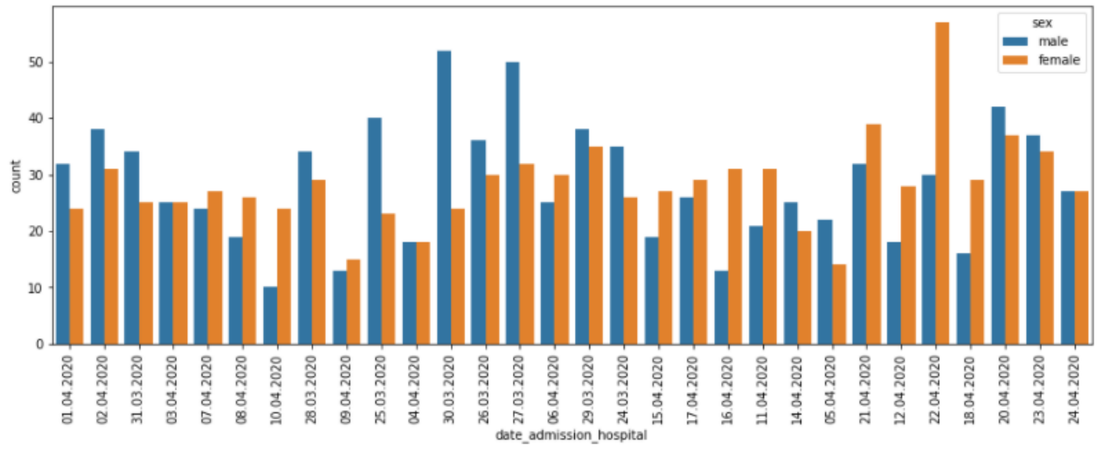
From above graph we check the top 20 distribution of symptoms in covid19 data so, we can see that fever symptom mostly distributed as compare to others symptoms.

```
# lets try to check the distribution of top 30 date_admission_hospital column
plt.figure(figsize=(15,5))
sns.countplot(x = 'date_admission_hospital', data=df, order=df['date_admission_hospital'].value_counts().index[:30], palette = [
plt.xticks(rotation=90)
plt.show()
```



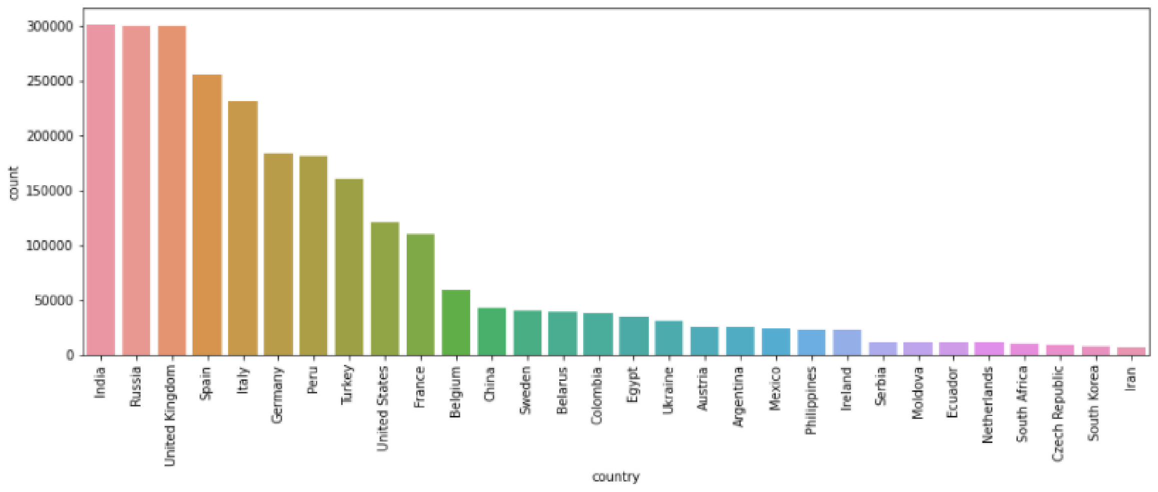
From above graph we plot the top 30 dates and check the distribution of each date so, we can see that patients mostly admitted at 01/04/2020.

```
# lets try to check the distribution of top 30 date_admission_hospital column with sex column
plt.figure(figsize=(15,5))
sns.countplot(x = 'date_admission_hospital', data=df,hue='sex', order=df['date_admission_hospital'].value_counts().index[:30])
plt.xticks(rotation=90)
plt.show()
```



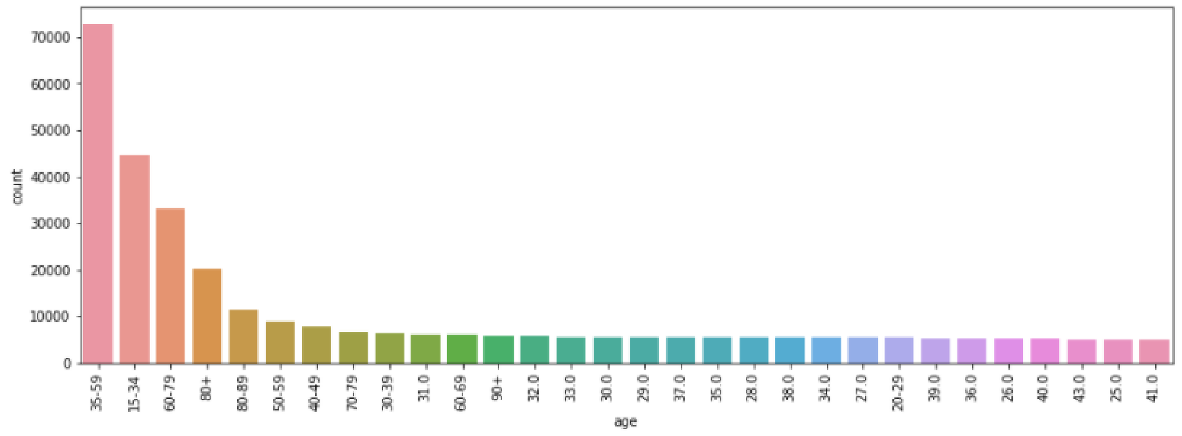
From above we also check which gender admitted mostly so, we can see that mostly male admitted as compare to female.

```python
# lets try to check the distribution of top 30 country column
plt.figure(figsize=(15,5))
sns.countplot(x = 'country', data=df, order=df['country'].value_counts().index[:30])
plt.xticks(rotation=90)
plt.show()
```



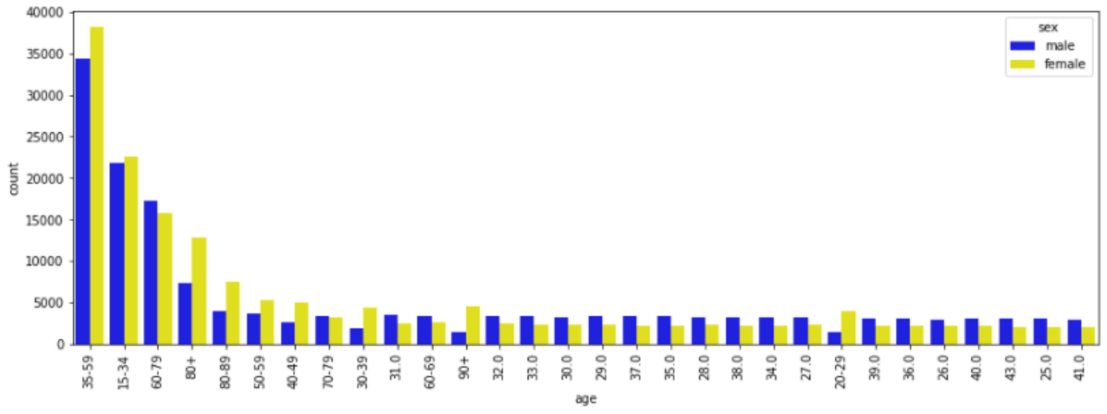
From above graph we plot the top 30 counties that are more effected by covid, India,Russia, UK are more effected as compare to others countries.

```python
# lets try to check the distribution of top 30 age column
plt.figure(figsize=(15,5))
sns.countplot(x = 'age', data=df, order=df['age'].value_counts().index[:30])
plt.xticks(rotation=90)
plt.show()
```



From above graph we can see that mostly effected people age range is 35-59.

```
# lets try to check the distribution of top 30 age column with sex column
plt.figure(figsize=(15,5))
sns.countplot(x = 'age', data=df,hue='sex', order=df['age'].value_counts().index[:30], palette=['#0000FF','#FFFF00'])
plt.xticks(rotation=90)
plt.show()
```



From above graph we check the age against gender. So, we can see that mostly female effected as compare to male.

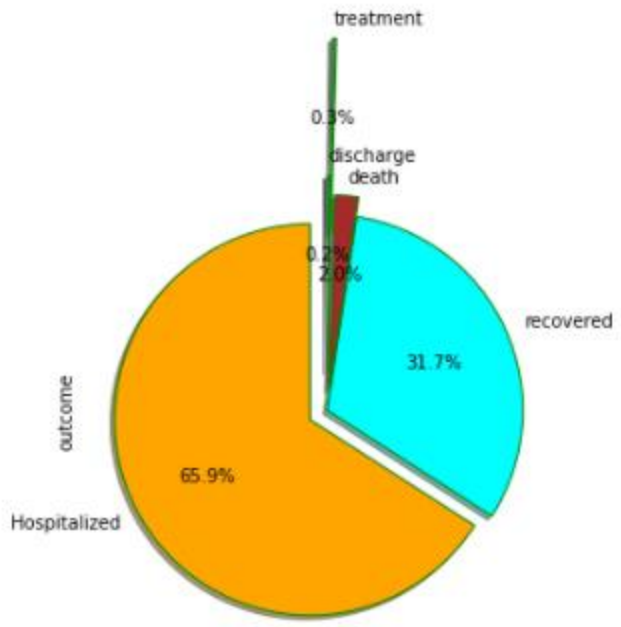
1.4 Feature Engineering/ Preprocessing

In this section, we will preprocessed our data and remove the columns that are not useful for our anylsis and will also handle this missing values but most important to handle the target variable outcome because we have a high cataegories values inside outcome column so, we will reduce the outcome and will extract the best categories for our prediction.

We extracting the data where outcome value are not null and also feature engineering on target column to replace all the categories into specific category that shown in below:

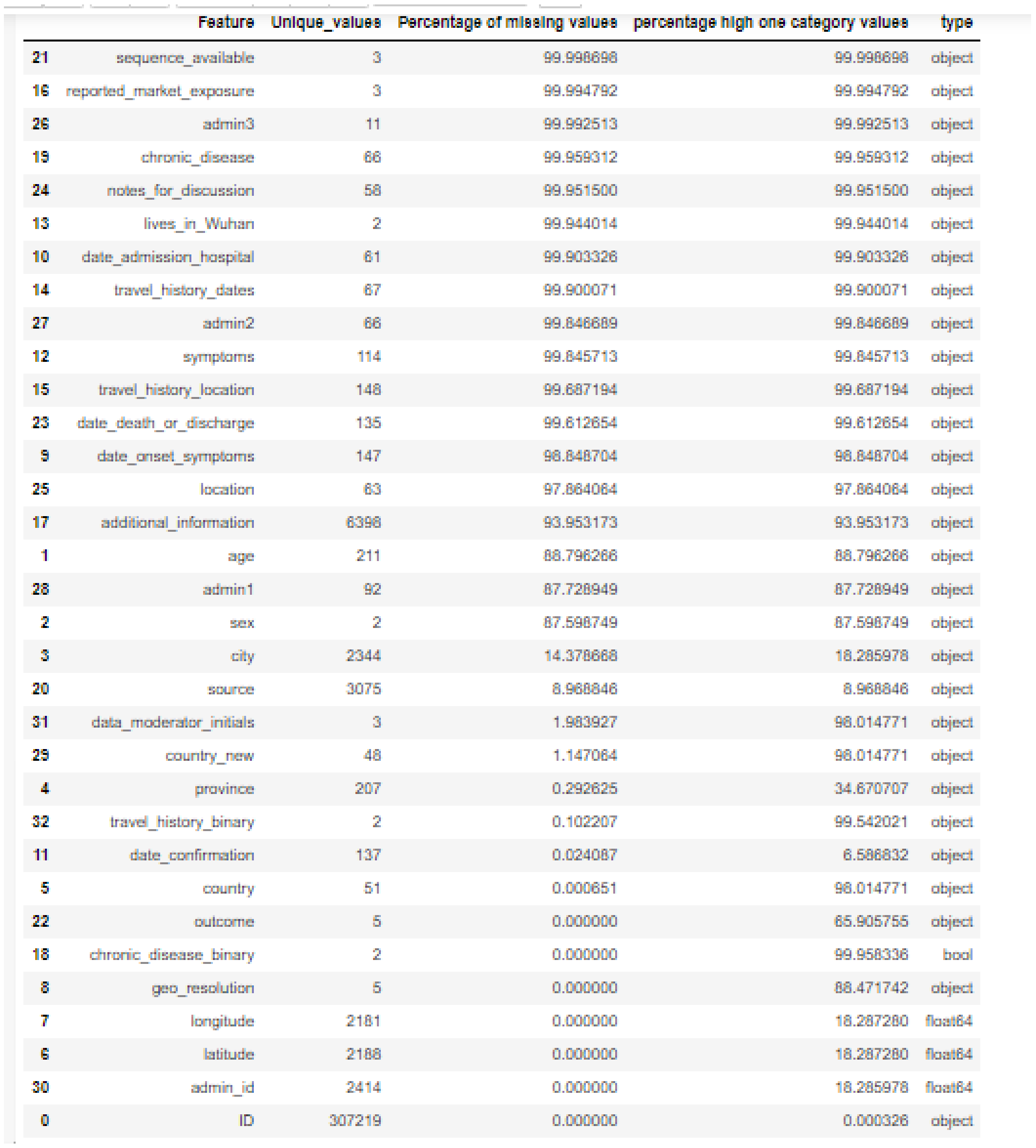
```
# now lets try to check the counts of target variable
df2['outcome'].value_counts()
```

```
Hospitalized    202475
recovered        97315
death             5999
treatment          947
discharge          483
Name: outcome, dtype: int64
```



From above pie chart we can see that mostly patient hospitalized and recovered.

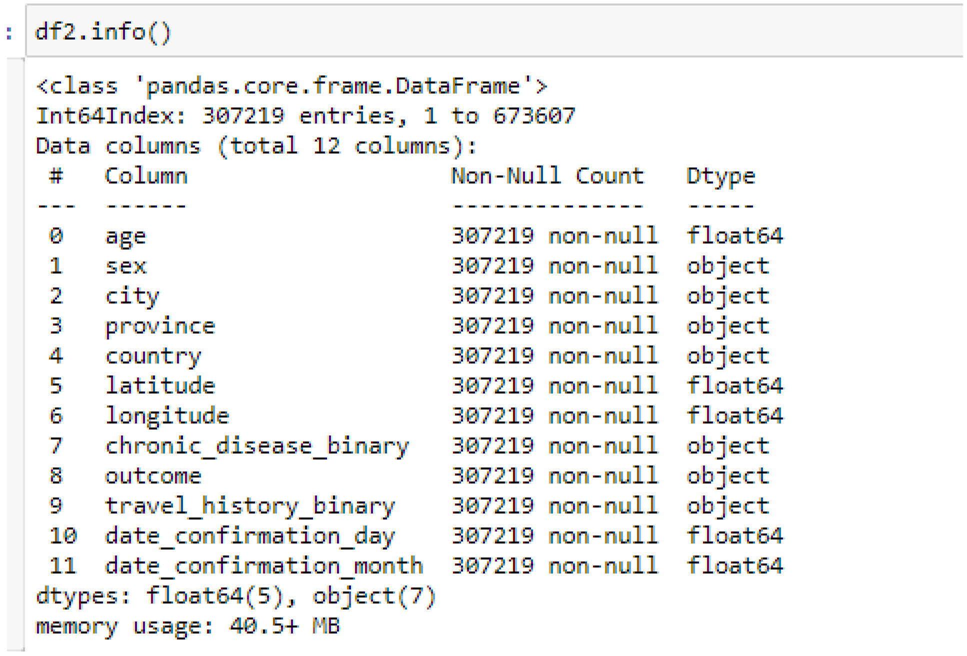
Now, we will remove all the columns where columns missing values greater then 90% and high one category values greater then 90%.



	Feature	Unique_values	Percentage of missing values	percentage high one category values	type
21	sequence_available	3	99.998698	99.998698	object
16	reported_market_exposure	3	99.994792	99.994792	object
26	admin3	11	99.992513	99.992513	object
19	chronic_disease	66	99.959312	99.959312	object
24	notes_for_discussion	58	99.951500	99.951500	object
13	lives_in_Wuhan	2	99.944014	99.944014	object
10	date_admission_hospital	61	99.903326	99.903326	object
14	travel_history_dates	67	99.900071	99.900071	object
27	admin2	66	99.846689	99.846689	object
12	symptoms	114	99.845713	99.845713	object
15	travel_history_location	148	99.687194	99.687194	object
23	date_death_or_discharge	135	99.612654	99.612654	object
9	date_onset_symptoms	147	98.848704	98.848704	object
25	location	63	97.864064	97.864064	object
17	additional_information	6398	93.953173	93.953173	object
1	age	211	88.796266	88.796266	object
28	admin1	92	87.728949	87.728949	object
2	sex	2	87.598749	87.598749	object
3	city	2344	14.378668	18.285978	object
20	source	3075	8.968846	8.968846	object
31	data_moderator_initials	3	1.983927	98.014771	object
29	country_new	48	1.147064	98.014771	object
4	province	207	0.292625	34.670707	object
32	travel_history_binary	2	0.102207	99.542021	object
11	date_confirmation	137	0.024087	6.586832	object
5	country	51	0.000651	98.014771	object
22	outcome	5	0.000000	65.905755	object
18	chronic_disease_binary	2	0.000000	99.958336	bool
8	geo_resolution	5	0.000000	88.471742	object
7	longitude	2181	0.000000	18.287280	float64
6	latitude	2188	0.000000	18.287280	float64
30	admin_id	2414	0.000000	18.285978	float64
0	ID	307219	0.000000	0.000326	object

From above graph we remove the columns where both missing values and high one category values greater then 90%.

So , we fill the missing values of numeric column into mean and categorical column fill with Unknown value.

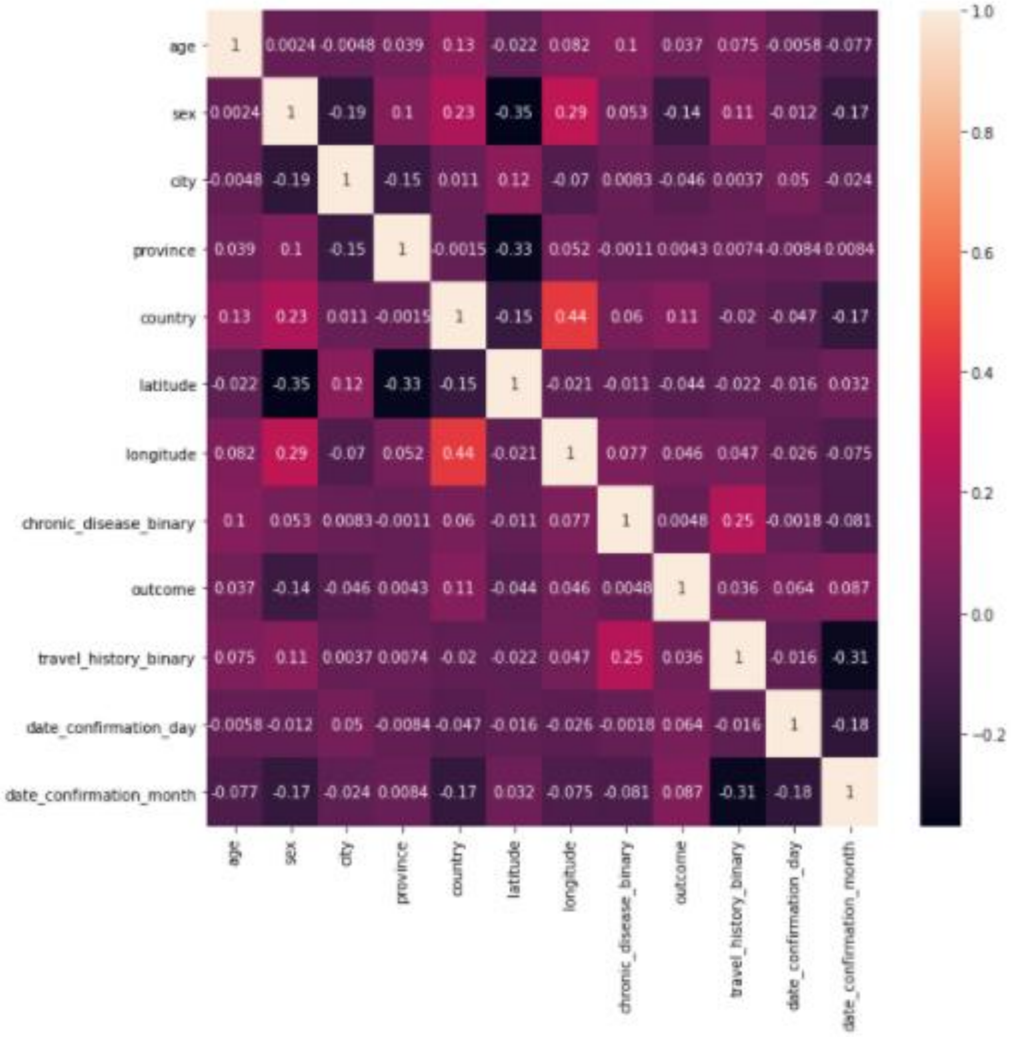


```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 307219 entries, 1 to 673607
Data columns (total 12 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   age                       307219 non-null  float64
 1   sex                       307219 non-null  object
 2   city                      307219 non-null  object
 3   province                  307219 non-null  object
 4   country                   307219 non-null  object
 5   latitude                  307219 non-null  float64
 6   longitude                 307219 non-null  float64
 7   chronic_disease_binary    307219 non-null  object
 8   outcome                   307219 non-null  object
 9   travel_history_binary     307219 non-null  object
 10  date_confirmation_day     307219 non-null  float64
 11  date_confirmation_month   307219 non-null  float64
dtypes: float64(5), object(7)
memory usage: 40.5+ MB
```

From above fig we can see that no missing values are present in our data.

We also check the correlation of each column that shown below:



```
print(corr["outcome"].sort_values(ascending=False))
```

```
outcome                    1.000000
country                    0.106043
date_confirmation_month    0.087160
date_confirmation_day      0.063540
longitude                  0.046404
age                        0.037032
travel_history_binary      0.035702
chronic_disease_binary     0.004778
province                   0.004318
latitude                  -0.043518
city                      -0.045798
sex                       -0.139379
Name: outcome, dtype: float64
```

1.5 Label Encoding

We also label encoding our categorical features in data. Categorical data refers to the information that has specific categories within the dataset. In this malware dataset above, there are many columns are categorical variables.

Machine Learning models are primarily based on mathematical equations. Thus, you can intuitively understand that keeping the categorical data in the equation will cause certain issues since you would only need numbers in the equations

1.6 Split Train Test

We also splitting the 80% data for training and 20% for testing. Every dataset for Machine Learning model must be split into two separate sets – training set and test set.

Usually, the dataset is split into 70:30 ratio or 80:20 ratio. This means that you either take 70% or 80% of the data for training the model while leaving out the rest 30% or 20%. The splitting process varies according to the shape and size of the dataset in question.

1.7 Standard Scaling

We also standard scaling our data. Feature scaling marks the end of the data preprocessing in Machine Learning. It is a method to standardize the independent variables of a dataset within a specific range.

In other words, feature scaling limits the range of variables so that you can compare them on common grounds.

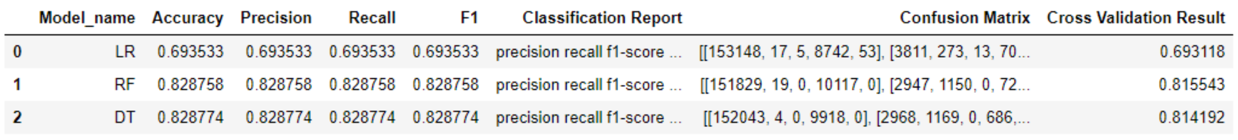
1.8 Cross Validation

We also apply the K Fold cross validation to check the performance on each models. Cross-validation is a statistical method used to estimate the skill of machine learning models.

It is commonly used in applied machine learning to compare and select a model for a given predictive modelling problem because it is easy to understand, easy to implement, and results in skill estimates that generally have a lower bias than other methods.

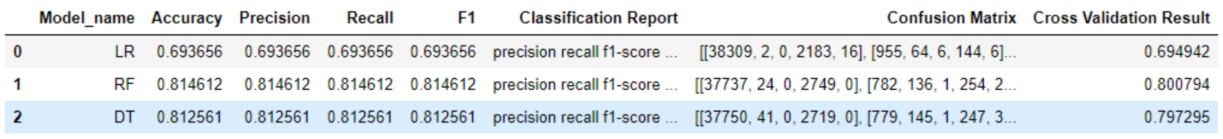
1.9 Results

Now we check the performance of each models on training and testing data.



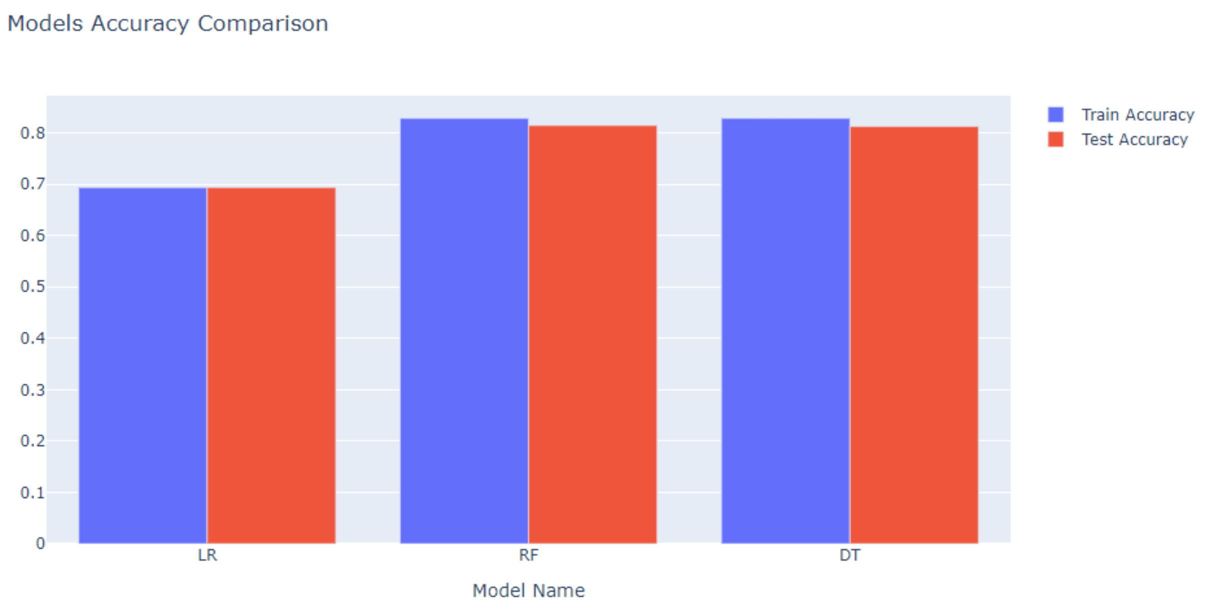
	Model_name	Accuracy	Precision	Recall	F1	Classification Report	Confusion Matrix	Cross Validation Result
0	LR	0.693533	0.693533	0.693533	0.693533	precision recall f1-score ...	[[153148, 17, 5, 8742, 53], [3811, 273, 13, 70...	0.693118
1	RF	0.828758	0.828758	0.828758	0.828758	precision recall f1-score ...	[[151829, 19, 0, 10117, 0], [2947, 1150, 0, 72...	0.815543
2	DT	0.828774	0.828774	0.828774	0.828774	precision recall f1-score ...	[[152043, 4, 0, 9918, 0], [2968, 1169, 0, 686,...	0.814192

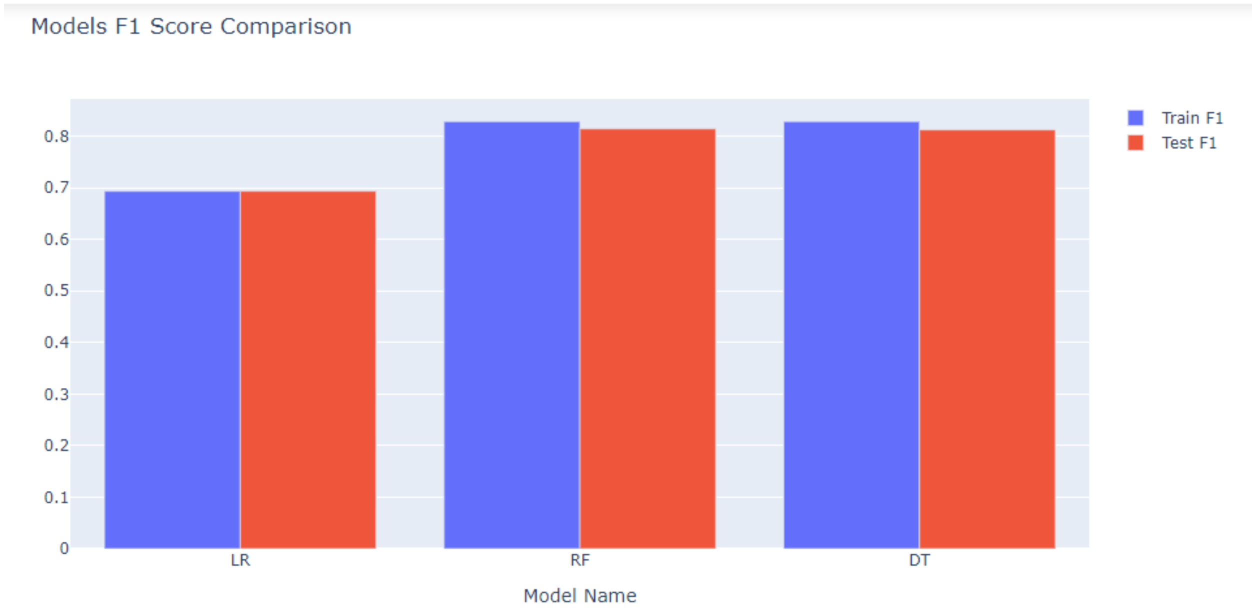
From above fig we can see that Random Forest giving good performance on training data as compare to other classifiers.



	Model_name	Accuracy	Precision	Recall	F1	Classification Report	Confusion Matrix	Cross Validation Result
0	LR	0.693656	0.693656	0.693656	0.693656	precision recall f1-score ...	[[38309, 2, 0, 2183, 16], [955, 64, 6, 144, 6]...	0.694942
1	RF	0.814612	0.814612	0.814612	0.814612	precision recall f1-score ...	[[37737, 24, 0, 2749, 0], [782, 136, 1, 254, 2...	0.800794
2	DT	0.812561	0.812561	0.812561	0.812561	precision recall f1-score ...	[[37750, 41, 0, 2719, 0], [779, 145, 1, 247, 3...	0.797295

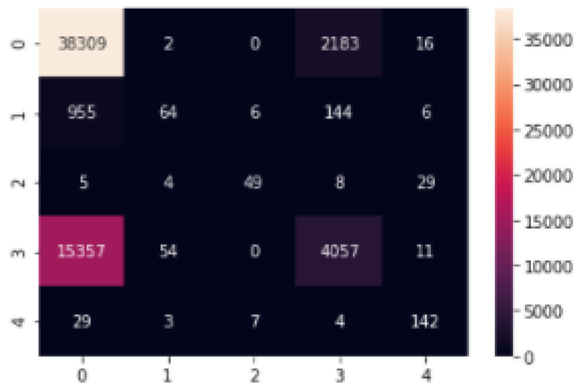
From above fig same Random Forest also giving good performance on testing data as compare to others.





We also check the confusion and classification report of each models that are given below:

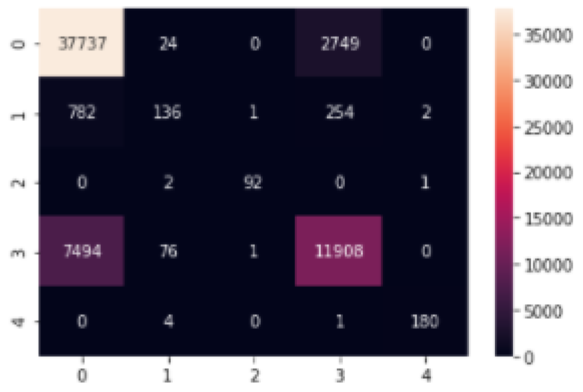
```
LR : Logistic Regression Confusion Matrix
```



```
print("\n=====",results.iloc[:,0][0],": Logistic Regression Classification Report=====\n\n")
print(cr_model0)
```

```
===== LR : Logistic Regression Classification Report=====


              precision    recall  f1-score   support

           0       0.70      0.95      0.81     40510
           1       0.50      0.05      0.10      1175
           2       0.79      0.52      0.62        95
           3       0.63      0.21      0.31     19479
           4       0.70      0.77      0.73       185

    accuracy                           0.69     61444
   macro avg       0.67      0.50      0.51     61444
weighted avg       0.68      0.69      0.64     61444
```

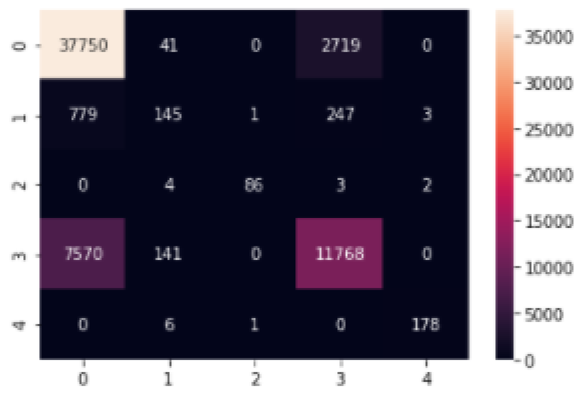
```
RF : Random Forest Confusion Matrix
```



```
print("\n=====",results.iloc[:,0][1],": Random Forest Classification Report=====\n\n")
print(cr_model1)
```

```
===== RF : Random Forest Classification Report=====


              precision    recall  f1-score   support

           0       0.82      0.93      0.87     40510
           1       0.56      0.12      0.19      1175
           2       0.98      0.97      0.97        95
           3       0.80      0.61      0.69     19479
           4       0.98      0.97      0.98       185

    accuracy                           0.81     61444
   macro avg       0.83      0.72      0.74     61444
weighted avg       0.81      0.81      0.80     61444
```

```
DT : Decision Tree Confusion Matrix\n
```



```
print("\n=====",results.iloc[:,0][2],": Decision Tree Classification Report=====\n\n")
print(cr_model2)
```

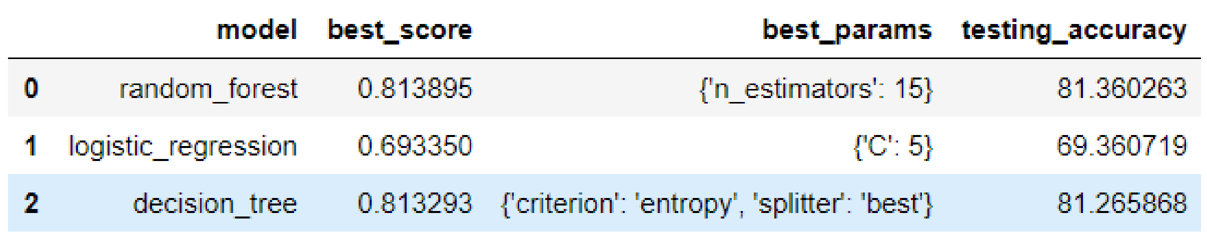
```
===== DT : Decision Tree Classification Report=====


              precision    recall  f1-score   support

           0       0.82      0.93      0.87     40510
           1       0.43      0.12      0.19      1175
           2       0.98      0.91      0.94        95
           3       0.80      0.60      0.69     19479
           4       0.97      0.96      0.97       185

    accuracy                           0.81     61444
   macro avg       0.80      0.71      0.73     61444
weighted avg       0.81      0.81      0.80     61444
```

1.10 Grid Search CV (Hyper parameters Tuning)

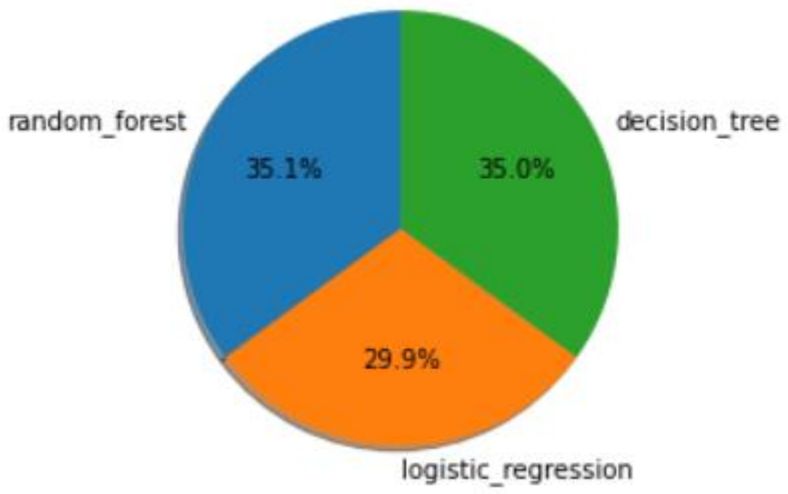
We also apply the Grid Search technique to get the best parameters of each models and best scores of each models. Grid Search CV was the greatest invention of all time. It runs through all the different parameters that is fed into the parameter grid and produces the best combination of parameters, based on a scoring metric of your choice. Obviously, nothing is perfect and Grid Search CV is no exception:

- “best parameters” results are limited
- process is time-consuming

The “best” parameters that Grid Search CV identifies are technically the best that could be produced, but only by the parameters that you included in your parameter grid.



	model	best_score	best_params	testing_accuracy
0	random_forest	0.813895	{'n_estimators': 15}	81.360263
1	logistic_regression	0.693350	{'C': 5}	69.360719
2	decision_tree	0.813293	{'criterion': 'entropy', 'splitter': 'best'}	81.265868



Conclusion

We conclude that random forest giving the good performance on different metrics e.g recall, precision, f1score, cross validation and hyper parameters tuning. So we will choose the Random Forest for our final prediction.