In [1]:

```
# Load libraries
import numpy as np
import pandas as pd
import sys
import os


import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import display
%matplotlib inline

import plotly.offline as py
import plotly.graph_objs as go
import plotly.tools as tls
py.init_notebook_mode()

import warnings
warnings.filterwarnings('ignore')

from pandas import set_option
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split,KFold,StratifiedKFold,
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, f1_score
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import AdaBoostClassifier,GradientBoostingClassifier,
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
from xgboost import XGBClassifier
from tabulate import tabulate
```

In [2]:

```
data = pd.read_csv("diabetes.csv")
```

In [3]:

```
data.shape
```

Out[3]:

```
(101766, 51)
```

In [4]:
```
data.head()
```

Out[4]:

	id	encounter_id	patient_nbr	race	gender	age	weight	admission_type_id	d
0	1	2278392	8222157	Caucasian	Female	[0-10)	?	6	
1	2	149190	55629189	Caucasian	Female	[10-20)	?	1	
2	3	64410	86047875	AfricanAmerican	Female	[20-30)	?	1	
3	4	500364	82442376	Caucasian	Male	[30-40)	?	1	
4	5	16680	42519267	Caucasian	Male	[40-50)	?	1	

5 rows × 51 columns

In [5]:
```
data = data.replace("?", np.NaN, )
```

Exploratory Data Analysis

In [6]:
```
data.isnull().sum()
```

Out[6]:
```
id                          0
encounter_id                0
patient_nbr                 0
race                     2273
gender                      0
age                         0
weight                  98569
admission_type_id           0
discharge_disposition_id    0
admission_source_id         0
time_in_hospital            0
payer_code              40256
medical_specialty       49949
num_lab_procedures          0
num_procedures              0
num_medications             0
number_outpatient           0
number_emergency            0
number_inpatient            0
```

In [7]:
```
#Replacing missing race with previous value - Forward fill

data = data.where(~data.race.isnull(), data.fillna(axis=0, method='ffill'))
```

In [8]:
```
data['weight'] = data['weight'].fillna(data['weight'].mode()[0])
data = data.where(~data.payer_code.isnull(), data.fillna(axis=0, method='ff
data = data.where(~data.medical_specialty.isnull(), data.fillna(axis=0, met


data = data.where(~data.diag_1.isnull(), data.fillna(axis=0, method='ffill'
data = data.where(~data.diag_2.isnull(), data.fillna(axis=0, method='ffill'
data = data.where(~data.diag_3.isnull(), data.fillna(axis=0, method='ffill'
```

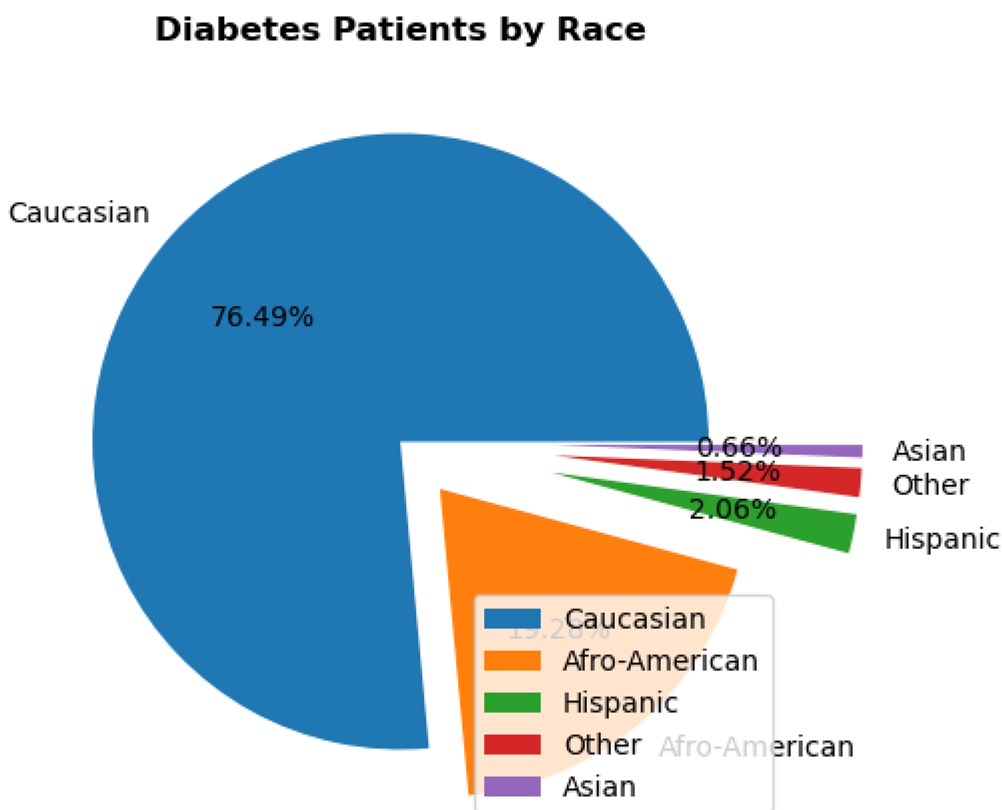
In [9]:
```
df = data.groupby(["race"]).size().sort_values(ascending = False)
```

In [10]:
```
df
```

Out[10]:
```
race
Caucasian          77840
AfricanAmerican    19622
Hispanic            2094
Other               1542
Asian                668
dtype: int64
```

In [11]:
```
Caucasian = data.loc[data["race"]=="Caucasian"].count()[0]
Afro_American = data.loc[data["race"]=="AfricanAmerican"].count()[0]
Hispanic = data.loc[data["race"]=="Hispanic"].count()[0]
Other = data.loc[data["race"]=="Other"].count()[0]
Asian = data.loc[data["race"]=="Asian"].count()[0]
```

In [12]:
```
plt.figure(figsize = [5,5], dpi = 100)
labels = ["Caucasian", "Afro-American", "Hispanic", "Other", "Asian"]
explode = [0,0.2,0.5,0.5,0.5]

plt.pie([Caucasian, Afro_American, Hispanic, Other, Asian], labels = labels
plt.title("Diabetes Patients by Race", fontdict = {"fontweight": "bold"})

plt.legend()
plt.show()
```



Caucasian are largest group of diabetic patients diagnosed, followed by Afro-American.

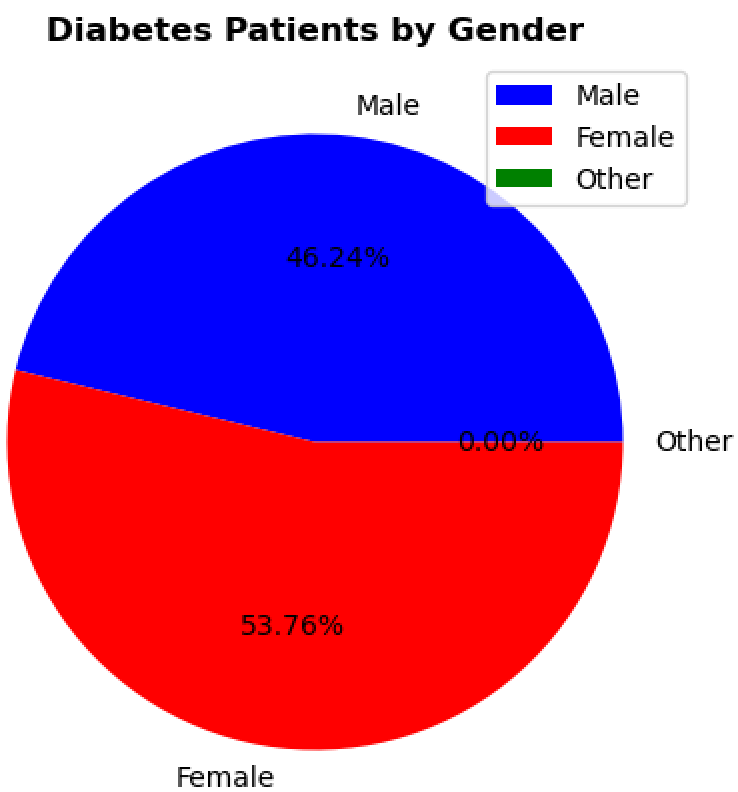
In [13]:
```
df = data.groupby(["gender"]).size().sort_values(ascending = False)
```

In [14]:
```
df
```

Out[14]:
```
gender
Female             54708
Male               47055
Unknown/Invalid        3
dtype: int64
```

In [15]:
```
Male = data.loc[data["gender"]=="Male"].count()[0]
Female = data.loc[data["gender"]=="Female"].count()[0]
Other = data.loc[data["gender"]=="Other"].count()[0]
```

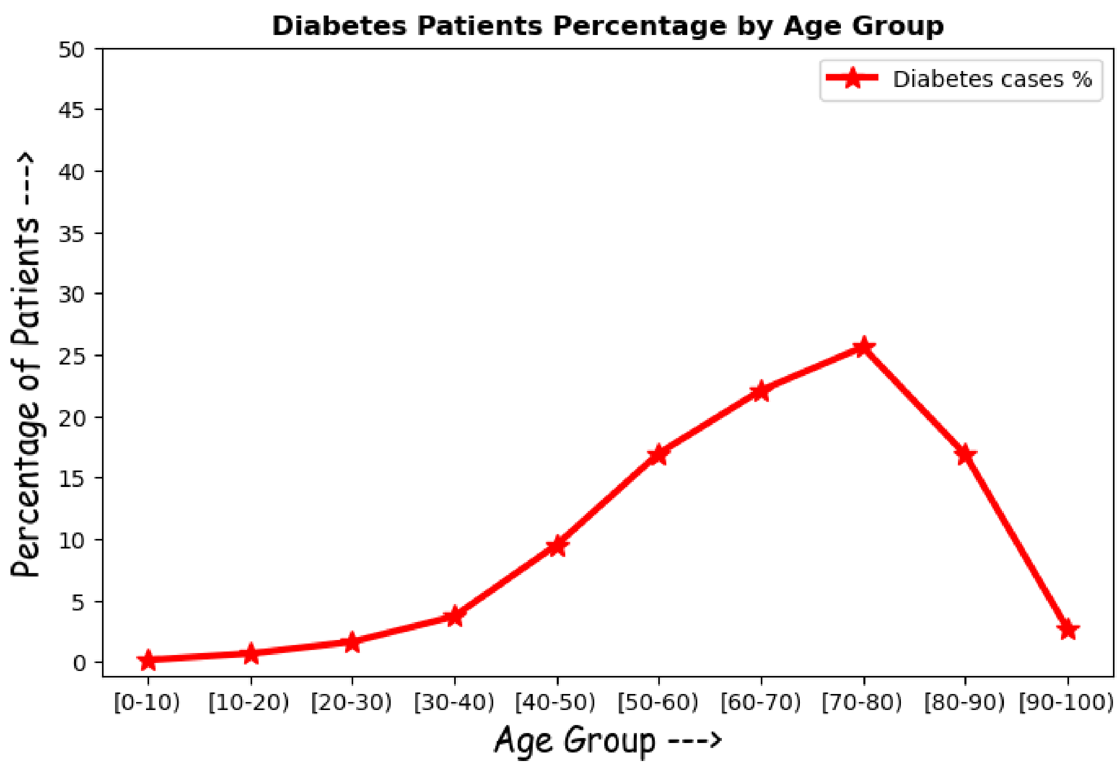
In [16]:
```
plt.figure(figsize = [5,5], dpi = 100)
labels = ["Male", "Female", "Other"]
colors = ["Blue", "Red", "Green"]

plt.pie([Male, Female, Other], colors = colors, labels = labels, autopct =
plt.title("Diabetes Patients by Gender", fontdict = {"fontweight": "bold"})

plt.legend()
plt.show()
```



Females are marginally more in number than Males. Others are negligible.

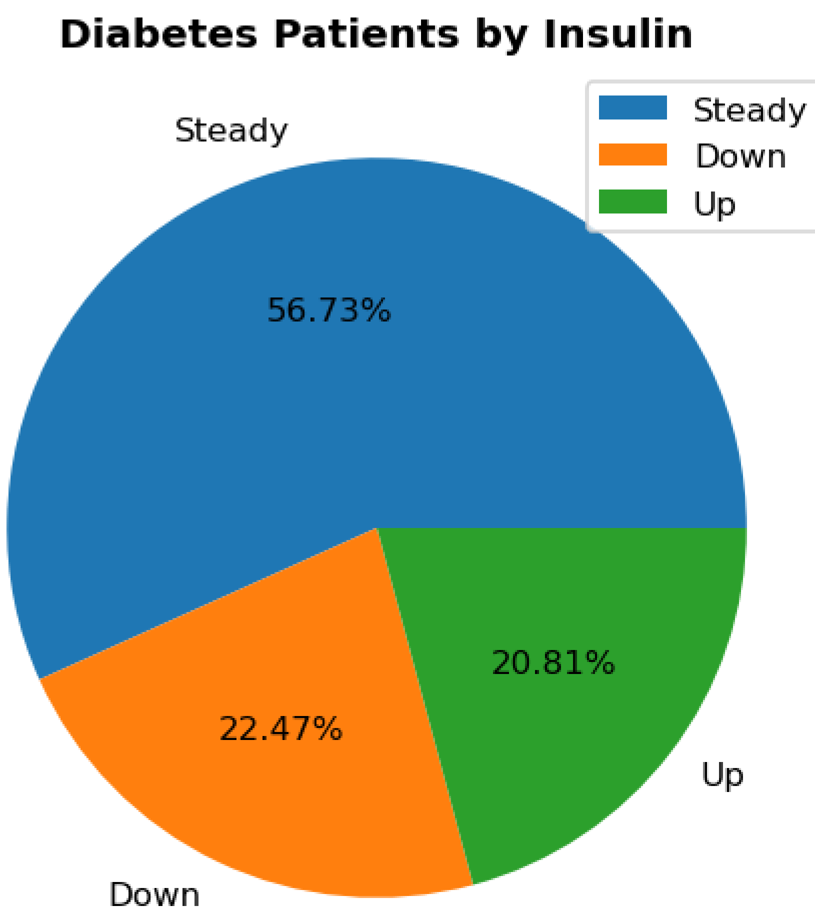
In [17]:

```
df = data.groupby(["age"]).size()
df1 = pd.DataFrame(df)
df1.columns = ["Count"]
df1["pct"] = (df1["Count"]/(df1["Count"].sum()))*100
df1["pct"]


plt.figure(figsize = [8,5], dpi = 100)

plt.xlabel("Age Group --->", fontdict = {"fontname": "Comic Sans MS", "font
plt.ylabel("Percentage of Patients --->", fontdict = {"fontname": "Comic Sa

plt.plot(df1["pct"], label = "Diabetes cases %", color = "red", linewidth =
plt.title("Diabetes Patients Percentage by Age Group", fontdict = {"fontwei

plt.yticks([0,5,10,15,20,25,30,35,40,45,50])

plt.legend()
plt.show()
```



Patients in age group [70-80] forms largest percentage of patients i.e. around 25%. While [0-10] form smallest percentage.

In [18]:
```
df = data.groupby(["insulin"]).size().sort_values(ascending = False)

Steady = data.loc[data["insulin"]=="Steady"].count()[0]
Down = data.loc[data["insulin"]=="Down"].count()[0]
Up = data.loc[data["insulin"]=="Up"].count()[0]

plt.figure(figsize = [5,5], dpi = 120)
labels = ["Steady", "Down", "Up"]

plt.pie([Steady, Down, Up], labels = labels, autopct = "%0.2f%%")
plt.title("Diabetes Patients by Insulin", fontdict = {"fontweight": "bold"}

plt.legend()
plt.show()
```



Drop some unwanted features

In [19]:
```
cols = ['id','weight', 'encounter_id', 'patient_nbr','admission_type_id', '
data = data.drop(columns = cols)
```

In [20]:
```
data.describe()
```

Out[20]:

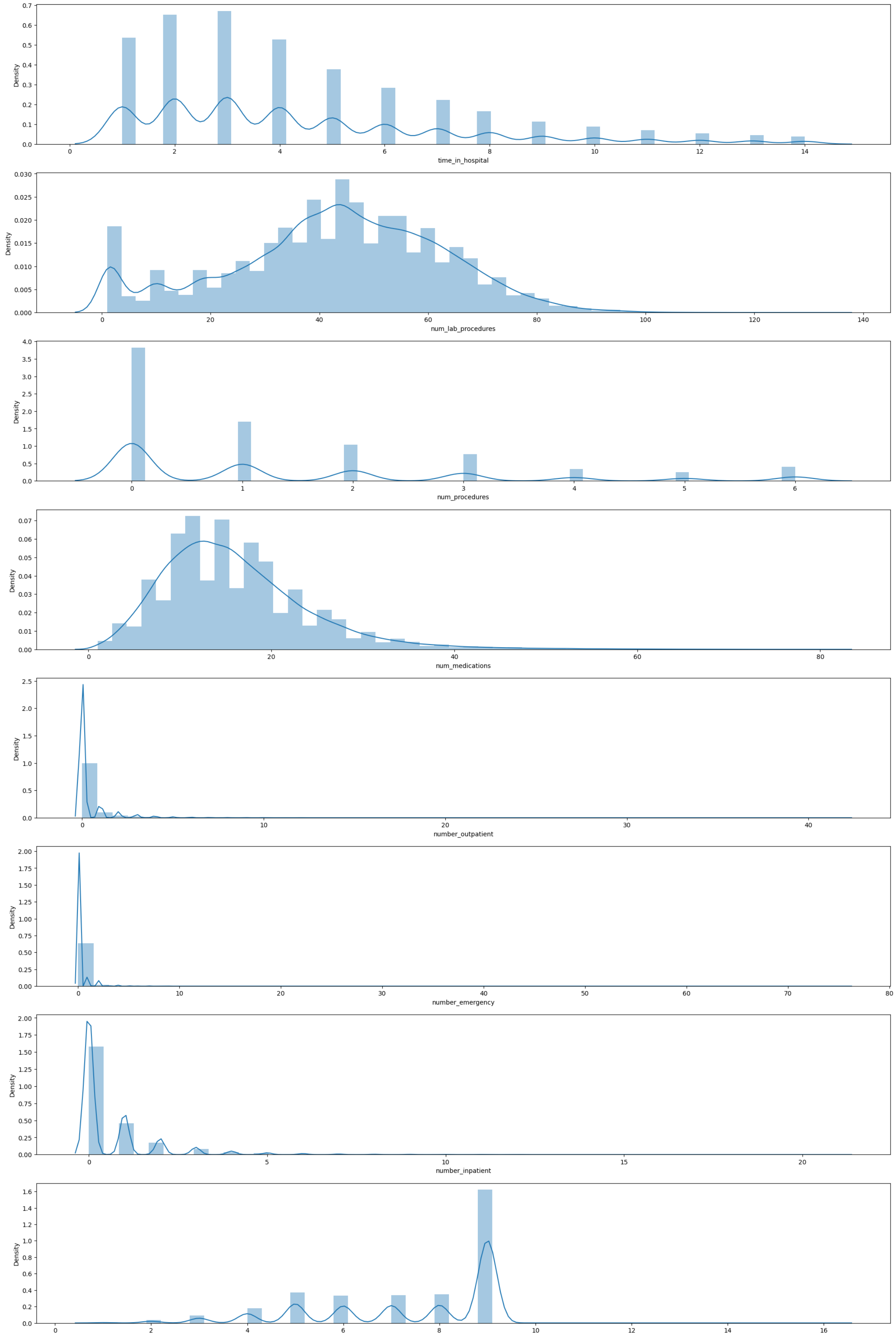
	time_in_hospital	num_lab_procedures	num_procedures	num_medications	number_o
count	101766.000000	101766.000000	101766.000000	101766.000000	10176
mean	4.395987	43.095641	1.339730	16.021844	
std	2.985108	19.674362	1.705807	8.127566	
min	1.000000	1.000000	0.000000	1.000000	
25%	2.000000	31.000000	0.000000	10.000000	
50%	4.000000	44.000000	1.000000	15.000000	
75%	6.000000	57.000000	2.000000	20.000000	
max	14.000000	132.000000	6.000000	81.000000	4

Feature Engineering

Feature engineering is one of the most crucial parts of building a good machine learning model. If we have useful features, the model will perform better. There are many situations where you can avoid large, complicated models and use simple models with crucially engineered features. We must keep in mind that feature engineering is something that is done in the best possible manner only when you have some knowledge about the domain of the problem and depends a lot on the data in concern. However, there are some general techniques that you can try to create features from almost all kinds of numerical and categorical variables. Feature engineering is not just about creating new features from data but also includes different types of normalization and transformations.

In [21]:

```
# Numerical features
num_feats=[col for col in data.columns if data[col].dtypes != 'object']

# Plot distribution of numerical columns
fig=plt.figure(figsize=(20,30))
for i, col in enumerate(num_feats):
    plt.subplot(len(num_feats),1,1*i+1)
    sns.distplot(data[col])

fig.tight_layout()
plt.show()
```

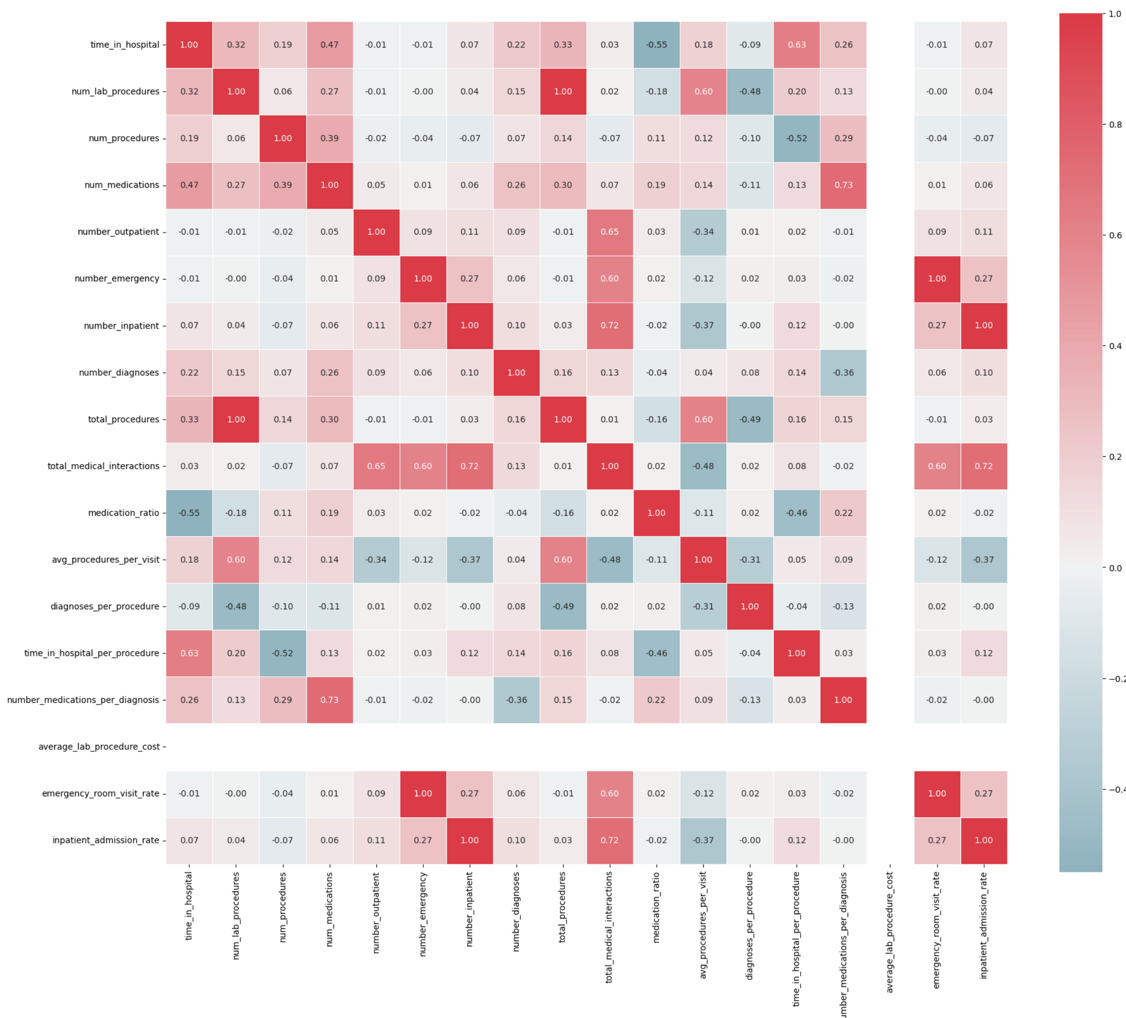

Density
time_in_hospital
Density
num_lab_procedures
Density
num_procedures
Density
num_medications
Density
number_outpatient
Density
number_emergency
Density
number_inpatient
Density
number_diagnoses

In [22]:
```
data['total_procedures'] = data['num_procedures'] + data['num_lab_procedure
data['total_medical_interactions'] = data['number_outpatient'] + data['numb
data['medication_ratio'] = data['num_medications'] / data['time_in_hospita
data['avg_procedures_per_visit'] = data['total_procedures'] / (data['numbe
data['diagnoses_per_procedure'] = data['number_diagnoses'] / data['total_pr

data["time_in_hospital_per_procedure"] = data["time_in_hospital"] / data["
data["number_medications_per_diagnosis"] = data["num_medications"] / data["
data["average_lab_procedure_cost"] = data["num_lab_procedures"].mean()
data["emergency_room_visit_rate"] = data["number_emergency"] / data.shape[
data["inpatient_admission_rate"] = data["number_inpatient"] / data.shape[0]
```

Type *Markdown* and LaTeX: α^2

In [23]:

```python
# Correlation
def HeatMap(df,x=True):
    correlations = df.corr()
    ## Create color map ranging between two colors
    cmap = sns.diverging_palette(220, 10, as_cmap=True)
    fig, ax = plt.subplots(figsize=(20, 20))
    fig = sns.heatmap(correlations, cmap=cmap, vmax=1.0, center=0, fmt='.2f
    fig.set_xticklabels(fig.get_xticklabels(), rotation = 90, fontsize = 10
    fig.set_yticklabels(fig.get_yticklabels(), rotation = 0, fontsize = 10)
    plt.tight_layout()
    plt.show()

HeatMap(data,x=True)
```



Cleaning Data

In [24]:
```
# convert range to interger value in age column
data['age'] = data.age.str.extract('(\d+)-(\d+)').astype('int').mean(axis=1

# replace '?' into None
data = data.replace(to_replace ="?",value ="None")
```

In [25]:
```
from sklearn.preprocessing import LabelEncoder
# get only categorical columns list
cat_feats= [col for col in data.columns if data[col].dtypes == 'object']

# encode the categorical features
encoder = LabelEncoder()
data[cat_feats] = data[cat_feats].apply(encoder.fit_transform)
```

Model Development

Spliting Model

In [26]:
```
data = data[~data.isin([np.inf, -np.inf]).any(1)]
```

In [27]:
```
from sklearn.model_selection import train_test_split

y = data['diabetesMed']
X = data.drop(columns = 'diabetesMed')

# split data into train and validation set
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.2,
```

In [34]:
```
y.value_counts()[1]/X.shape[0]
```

Out[34]:
```
0.7813190280845692
```

In [29]:
```
X.shape
```

Out[29]:
```
(22183, 52)
```

Baseline Model

In [ ]:
```
```

In [34]:

```
def BasedModel():
    basedModels = []
    basedModels.append(('LR'   , LogisticRegression()))
    basedModels.append(('LDA'  , LinearDiscriminantAnalysis()))
    basedModels.append(('KNN'  , KNeighborsClassifier()))
    basedModels.append(('RF'   , RandomForestClassifier()))
    basedModels.append(('NB'   , GaussianNB()))
    basedModels.append(('AB'   , AdaBoostClassifier()))
    basedModels.append(('GBM'  , GradientBoostingClassifier()))
    basedModels.append(('ET'   , ExtraTreesClassifier()))
    basedModels.append(('XG'   , XGBClassifier()))
    basedModels.append(('LG'   , LGBMClassifier()))
    basedModels.append(('CAT'   , CatBoostClassifier(silent=True)))
    return basedModels
```

In [42]:

```
def BasedLine(X_train, y_train, X_valid, y_valid, models):
    # Test options and evaluation metric
    scoring = 'accuracy'
    results, results_weigh = [],[]
    names = []
    scores, scores_weigh = [],[]
    data = []
    for name, model in models:
        model.fit(X_train, y_train)

        cv_results = cross_validate(model, X_train, y_train, scoring=['f1_w
        cv_weigh = cv_results["test_f1_weighted"].mean()
        cv_non = cv_results["test_f1"].mean()
        score_non = f1_score(model.predict(X_valid), y_valid)
        score_weigh = f1_score(model.predict(X_valid), y_valid,  average='w

        results.append(cv_non)
        results_weigh.append(cv_weigh)
        names.append(name)
        scores.append(score_non)
        scores_weigh.append(score_weigh)
        data.append([name,cv_non, score_non, cv_weigh,score_weigh])
    print(tabulate(data, headers=["Model", "CV F1 Score", "Model F1 Score",

    return names, results, scores
```

In [43]:

```
models = BasedModel()
names,results, scores = BasedLine(X_train, y_train,X_valid, y_valid, models
```

```
| Model    |   CV F1 Score |   Model F1 Score |   CV F1 Weighted |   Model
F1 Weighted |
|---------+---------------+------------------+------------------+--------
-------------|
| LR       |      0.879916 |         0.881181 |         0.761395 |
0.827321 |
| LDA      |      0.979951 |         0.980225 |         0.967617 |
0.96943  |
| KNN      |      0.84877  |         0.848686 |         0.695025 |
0.79241  |
| RF       |      0.99881  |         0.998986 |         0.998144 |
0.99842  |
| NB       |      0.999964 |         1        |         0.999944 |
1        |
| AB       |      0.999892 |         1        |         0.999831 |
1        |
| GBM      |      0.999171 |         0.999421 |         0.998706 |
0.999098 |
| ET       |      0.999712 |         0.999566 |         0.999549 |
0.999323 |
| XG       |      0.999387 |         0.999421 |         0.999043 |
0.999098 |
| LG       |      0.999676 |         0.999421 |         0.999493 |
0.999098 |
| CAT      |      0.999892 |         0.99971  |         0.999831 |
0.999549 |
```

In [ ]: