What Causes DIABETES ?

Let’s Explore

```
import pandas
import numpy
import matplotlib.pyplot as mat
import seaborn as sns
from matplotlib.gridspec import GridSpec
from scipy.stats import pearsonr
from scipy.stats import chi2_contingency

path =
r'/kaggle/input/heart-attack-in-youth-vs-adult-in-germany/heart_attack
_germany.csv'
data = pandas.read_csv(path)
```

Understanding the Dataset

```
# Rows and Columns Information
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 275644 entries, 0 to 275643
Data columns (total 22 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   State                     275644 non-null  object
 1   Age_Group                 275644 non-null  object
 2   Heart_Attack_Incidence    275644 non-null  int64
 3   Year                      275644 non-null  int64
 4   Gender                    275644 non-null  object
 5   BMI                       275644 non-null  float64
 6   Smoking_Status            275644 non-null  object
 7   Alcohol_Consumption       275644 non-null  float64
 8   Physical_Activity_Level   275644 non-null  object
 9   Diet_Quality              275644 non-null  object
 10  Family_History            275644 non-null  int64
 11  Hypertension              275644 non-null  int64
 12  Cholesterol_Level         275644 non-null  float64
 13  Diabetes                  275644 non-null  int64
 14  Urban_Rural               275644 non-null  object
 15  Socioeconomic_Status      275644 non-null  object
 16  Air_Pollution_Index       275644 non-null  float64
 17  Stress_Level              275644 non-null  object
 18  Healthcare_Access         275644 non-null  object
 19  Education_Level           275644 non-null  object
 20  Employment_Status         275644 non-null  object
 21  Region_Heart_Attack_Rate  275644 non-null  float64
dtypes: float64(5), int64(5), object(12)
memory usage: 46.3+ MB
```

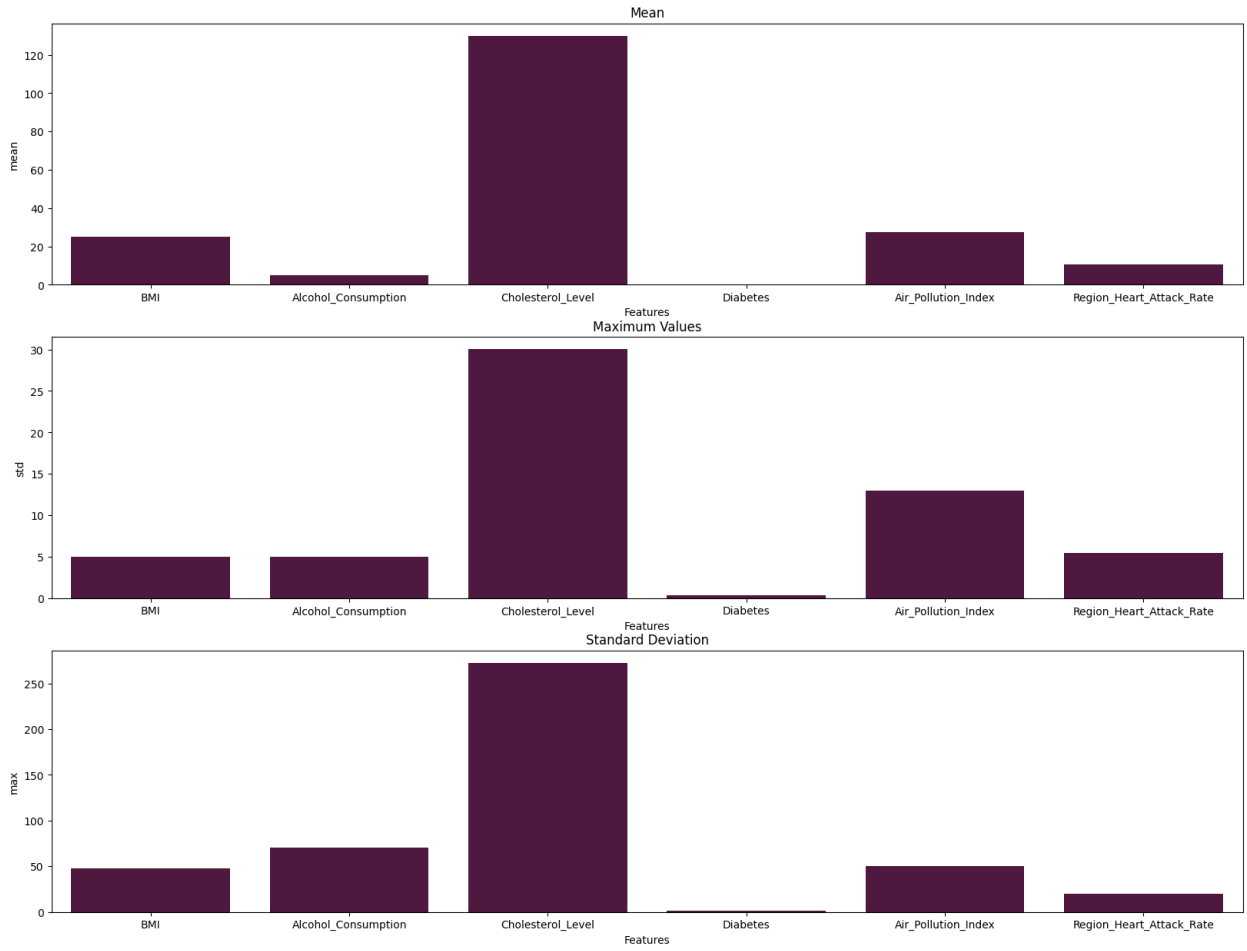
```
# Central Tendencies and other info
data.describe()
```

```
       Heart_Attack_Incidence           Year            BMI  \
count           275644.000000  275644.000000  275644.000000
mean                 0.150070    2018.997319      24.992669
std                  0.357141       2.582667       4.996535
min                  0.000000    2015.000000       1.400000
25%                  0.000000    2017.000000      21.600000
50%                  0.000000    2019.000000      25.000000
75%                  0.000000    2021.000000      28.400000
max                  1.000000    2023.000000      47.900000

       Alcohol_Consumption  Family_History   Hypertension
Cholesterol_Level  \
count        275644.000000   275644.000000  275644.000000
275644.000000
mean              4.985734        0.300540       0.400564
130.034229
std               5.001789        0.458494       0.490014
30.009242
min               0.000000        0.000000       0.000000          -
7.900000
25%               1.400000        0.000000       0.000000
109.800000
50%               3.500000        0.000000       0.000000
130.100000
75%               6.900000        1.000000       1.000000
150.300000
max              70.000000        1.000000       1.000000
272.400000

            Diabetes  Air_Pollution_Index  Region_Heart_Attack_Rate
count  275644.000000        275644.000000             275644.000000
mean        0.199525            27.486225                 10.489019
std         0.399644            13.001527                  5.483277
min         0.000000             5.000000                  1.000000
25%         0.000000            16.220000                  5.750000
50%         0.000000            27.430000                 10.490000
75%         0.000000            38.780000                 15.240000
max         1.000000            50.000000                 20.000000
```

```
# Plotting mean
mean = data.drop(columns=['Year', 'Hypertension', 'Family_History',
'Heart_Attack_Incidence']).describe().loc[['mean']].T
highest_value = data.drop(columns=['Year', 'Hypertension',
'Family_History', 'Heart_Attack_Incidence']).describe().loc[['max']].T
std = data.drop(columns=['Year', 'Hypertension', 'Family_History',
'Heart_Attack_Incidence']).describe().loc[['std']].T

# Adding the columns to plot
mean['Features'] = mean.index.tolist()
highest_value['Features'] = highest_value.index.tolist()
```

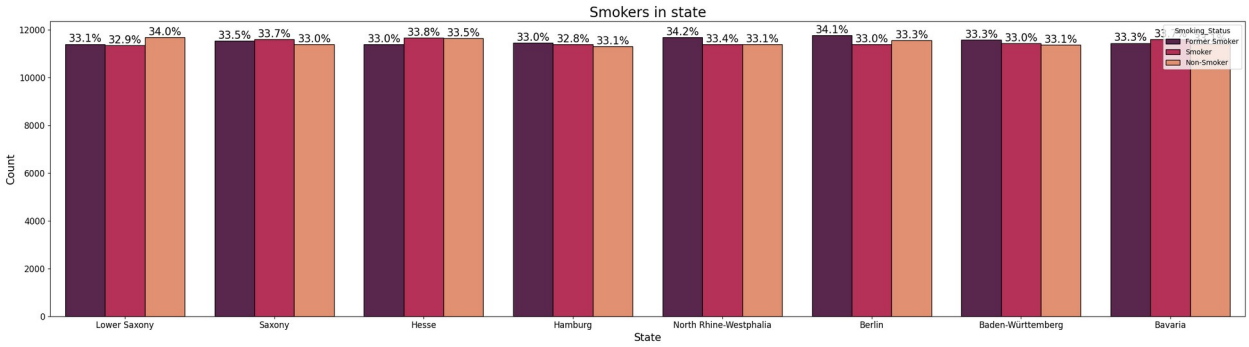
```
std['Features'] = std.index.tolist()

fig, grid = mat.subplots(3, 1, figsize=(20, 15))

sns.barplot(
    mean,
    x='Features',
    y='mean',
    color='xkcd:plum',
    ax=grid[0]
)
grid[0].set_title('Mean')

sns.barplot(
    highest_value,
    x='Features',
    y='max',
    color='xkcd:plum',
    ax=grid[2]
)
grid[2].set_title('Standard Deviation')

sns.barplot(
    std,
    x='Features',
    y='std',
    color='xkcd:plum',
    ax=grid[1]
)
grid[1].set_title('Maximum Values')
```

```
Text(0.5, 1.0, 'Maximum Values')
```




```
mat.figure(figsize=(25, 7))

# Create the countplot
ax = sns.countplot(x='State', hue='Smoking_Status', data=data,
palette='rocket', edgecolor='black')

# Prepare data for annotations
state_counts = data.groupby(['State',
'Smoking_Status']).size().reset_index(name='Count')
total_counts = data['State'].value_counts()

# Annotate bars with percentages
for container in ax.containers:

    heights = [bar.get_height() for bar in container]
    sorted_heights = sorted(heights, reverse=True)

    for bar in container:
        count = bar.get_height()
        if count > 0:
```

```
            label = ax.get_xticklabels()[int(bar.get_x())].get_text()
            percentage = (count / total_counts[label]) * 100

            ax.text(
                bar.get_x() + bar.get_width() / 2,
                bar.get_height() + 1,
                f"{percentage:.1f}%",
                ha='center',
                va='bottom',
                fontsize='15',
                color='black',
            )

# Customize the plot
mat.title('Smokers in state', fontsize=20)
mat.ylabel('Count', fontsize=15)
mat.xlabel('State', fontsize=15)
mat.xticks(fontsize=12)
mat.yticks(fontsize=12)
mat.tight_layout()
mat.show()
```



Visualising - Distribution Plots

```
data.head()
```

	State	Age_Group	Heart_Attack_Incidence	Year	Gender	BMI
0	Lower Saxony	Youth	0	2018	Other	25.6
1	Saxony	Adult	0	2021	Female	36.7
2	Hesse	Youth	1	2022	Female	28.6
3	Lower Saxony	Adult	0	2015	Male	27.6
4	Hamburg	Adult	0	2015	Female	15.2

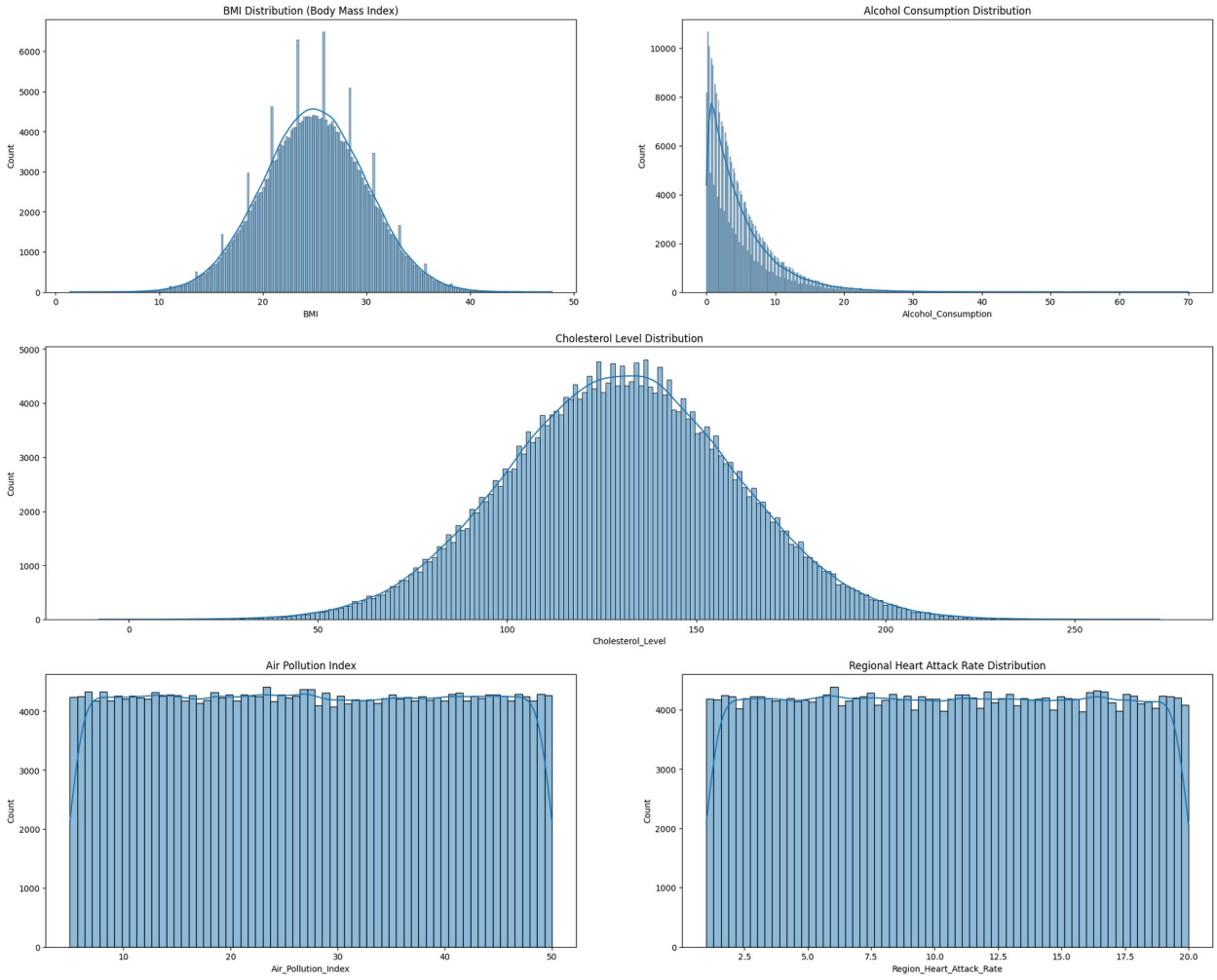
```
  Smoking_Status  Alcohol_Consumption Physical_Activity_Level
Diet_Quality  \
0  Former Smoker                  4.2                Moderate
Average
1         Smoker                  2.4                     Low
Poor
2         Smoker                 29.5                    High
Poor
3     Non-Smoker                  4.2                Moderate
Poor
4         Smoker                  4.3                Moderate
Good

   ...  Cholesterol_Level  Diabetes  Urban_Rural  Socioeconomic_Status
\
0  ...              154.4         0        Rural                   Low

1  ...               75.0         1        Rural                   Low

2  ...              121.9         0        Urban                Middle

3  ...              152.3         0        Urban                   Low

4  ...              130.3         0        Urban                  High


  Air_Pollution_Index Stress_Level  Healthcare_Access Education_Level
\
0               31.58     Moderate           Moderate         Primary

1               46.22         High               Easy         Primary

2               15.69         High               Hard       Secondary

3               26.50         High               Hard        Tertiary

4               11.21         High           Moderate        Tertiary


  Employment_Status Region_Heart_Attack_Rate
0           Retired                     1.92
1        Unemployed                    14.16
2           Student                     3.49
3           Student                     3.24
4          Employed                     9.98

[5 rows x 22 columns]
```

```
dist_columns = ['BMI', 'Alcohol_Consumption', 'Cholesterol_Level',
'Air_Pollution_Index', 'Region_Heart_Attack_Rate']
```

```python
figure = mat.figure(figsize=(25, 20))

grid = GridSpec(3, 2, figure=figure)

first_graph = figure.add_subplot(grid[0, 0])
second_graph = figure.add_subplot(grid[0, 1])
third_graph = figure.add_subplot(grid[1, :])
fourth_graph = figure.add_subplot(grid[2, 0])
fifth_graph = figure.add_subplot(grid[2, 1])

sns.histplot(data['BMI'], ax=first_graph, kde=True)
first_graph.set_title('BMI Distribution (Body Mass Index)')

sns.histplot(data['Alcohol_Consumption'], ax=second_graph, kde=True)
second_graph.set_title('Alcohol Consumption Distribution')

sns.histplot(data['Cholesterol_Level'], ax=third_graph, kde=True)
third_graph.set_title('Cholesterol Level Distribution')

sns.histplot(data['Air_Pollution_Index'], ax=fourth_graph, kde=True)
fourth_graph.set_title('Air Pollution Index')

sns.histplot(data['Region_Heart_Attack_Rate'], ax=fifth_graph,
kde=True)
fifth_graph.set_title('Regional Heart Attack Rate Distribution')
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
```

```
instead.
  with pd.option_context('mode.use_inf_as_na', True):

Text(0.5, 1.0, 'Regional Heart Attack Rate Distribution')
```



Note:

- BMI and Cholesterol Level both are normal distribution
- Alcohol consumption is right skewed distribution
- Remaining two does not have a specific pattern

Visual - Bar Charts

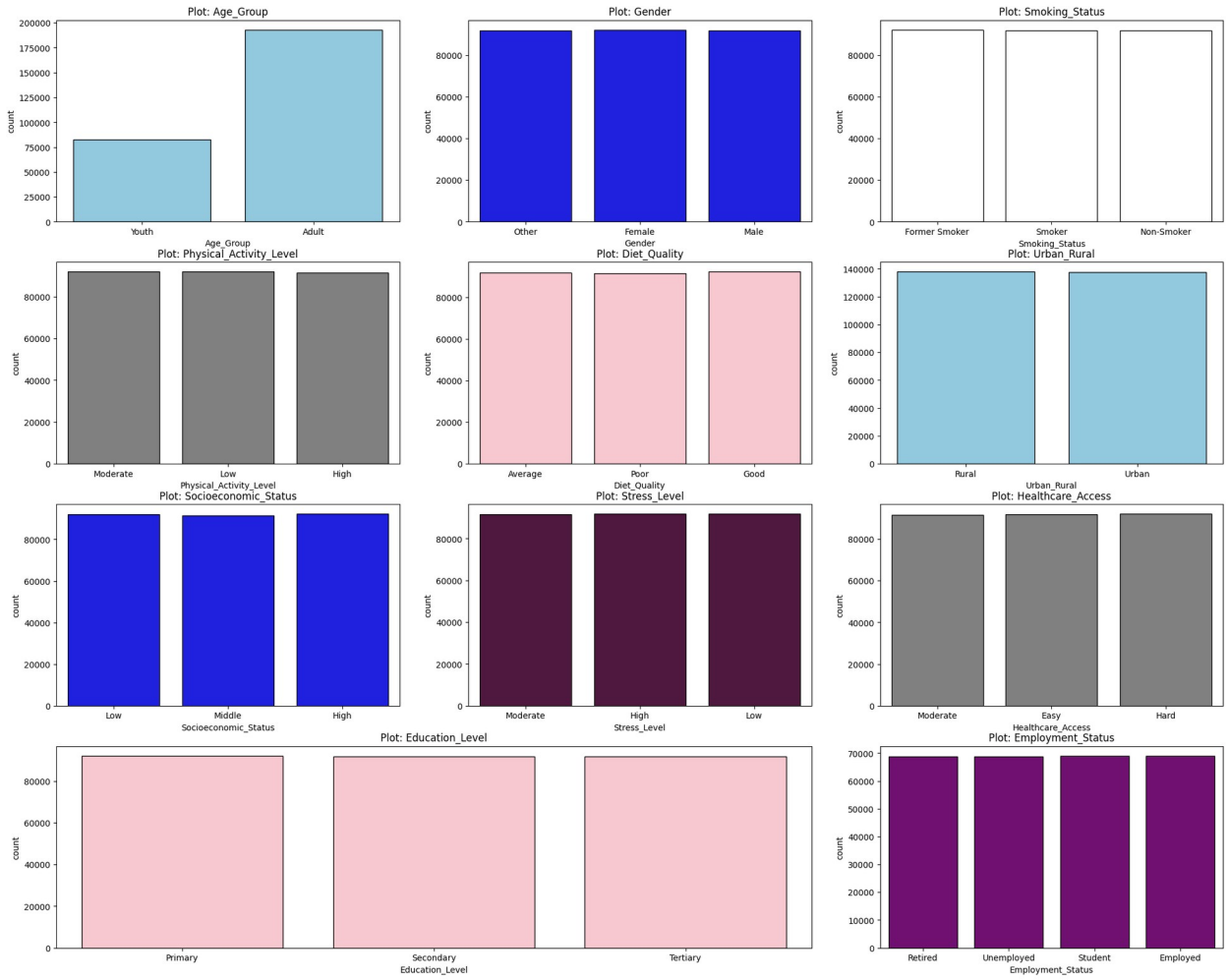
```
data.head()
```

```
          State Age_Group  Heart_Attack_Incidence  Year  Gender   BMI
\
0  Lower Saxony     Youth                       0  2018   Other  25.6

1        Saxony     Adult                       0  2021  Female  36.7
```

```
2         Hesse     Youth                        1  2022  Female  28.6

3  Lower Saxony     Adult                        0  2015    Male  27.6

4       Hamburg     Adult                        0  2015  Female  15.2


  Smoking_Status  Alcohol_Consumption Physical_Activity_Level
Diet_Quality  \
0  Former Smoker                  4.2                Moderate
Average
1         Smoker                  2.4                     Low
Poor
2         Smoker                 29.5                    High
Poor
3     Non-Smoker                  4.2                Moderate
Poor
4         Smoker                  4.3                Moderate
Good

   ...  Cholesterol_Level  Diabetes  Urban_Rural  Socioeconomic_Status
\
0  ...              154.4         0        Rural                   Low

1  ...               75.0         1        Rural                   Low

2  ...              121.9         0        Urban                Middle

3  ...              152.3         0        Urban                   Low

4  ...              130.3         0        Urban                  High


  Air_Pollution_Index Stress_Level  Healthcare_Access Education_Level
\
0               31.58     Moderate           Moderate         Primary

1               46.22         High               Easy         Primary

2               15.69         High               Hard       Secondary

3               26.50         High               Hard        Tertiary

4               11.21         High           Moderate        Tertiary


  Employment_Status Region_Heart_Attack_Rate
0           Retired                     1.92
1        Unemployed                    14.16
2           Student                     3.49
```

```
3           Student                         3.24
4          Employed                         9.98

[5 rows x 22 columns]
```

```
# Counts
cat_columns =
data.select_dtypes(include='object').drop(columns='State',
axis=1).columns

figure = mat.figure(figsize=(25, 20))

mat.tight_layout(pad=3)
grid = GridSpec(4, 3)

g1 = figure.add_subplot(grid[0, 0])
g2 = figure.add_subplot(grid[0, 1])
g3 = figure.add_subplot(grid[0, 2])
g4 = figure.add_subplot(grid[1, 0])
g5 = figure.add_subplot(grid[1, 1])
g6 = figure.add_subplot(grid[1, 2])
g7 = figure.add_subplot(grid[2, 0])
g8 = figure.add_subplot(grid[2, 1])
g9 = figure.add_subplot(grid[2, 2])
g10 = figure.add_subplot(grid[3, :2])
g11 = figure.add_subplot(grid[3, 2])

axises = [g1, g2, g3, g4, g5, g6, g7, g8, g9, g10, g11]
colors = ['skyblue', 'blue', 'white', 'grey', 'pink','skyblue',
'blue', 'xkcd:plum', 'grey', 'pink', 'purple']

for i, ax in enumerate(axises):
    if i < len(cat_columns):
        ax.set_title(f"Plot: {cat_columns[i]}")
        sns.countplot(x=data[cat_columns[i]], ax=ax, color=colors[i],
edgecolor='black')
    else:
        ax.axis('off')
```



- Almost all the features have the same distribution of data to the categories we have in each features.
- We have more Adults than Young. Meaning the data is mostly dominated by adults.

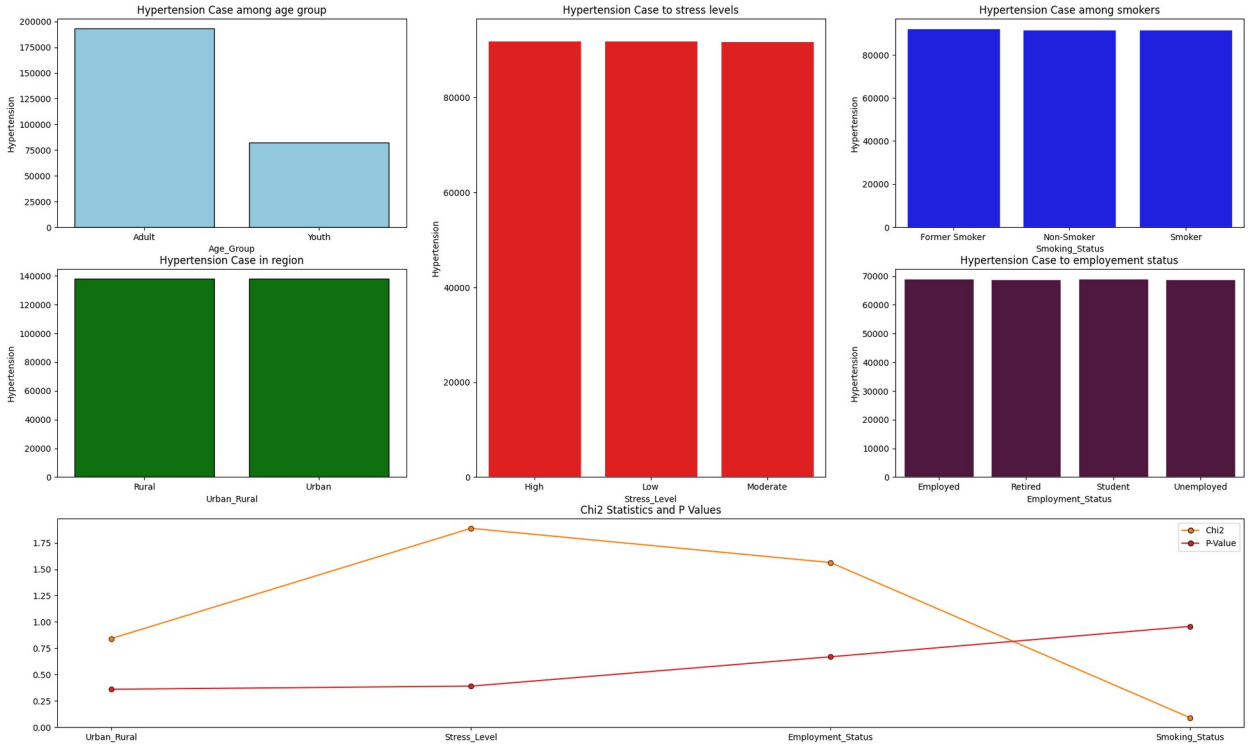
- Hypertension Cases

```
# To save from futurewarming

# Grouping
age_Hypertension_case = pandas.DataFrame(data.groupby('Age_Group')
['Hypertension'].count()).reset_index()
Smoking_Hypertension_case =
pandas.DataFrame(data.groupby('Smoking_Status')
['Hypertension'].count()).reset_index()
Urban_rural_ht_case = pandas.DataFrame(data.groupby('Urban_Rural')
['Hypertension'].count()).reset_index()
Stress_hypertension_case =
pandas.DataFrame(data.groupby('Stress_Level')
['Hypertension'].count()).reset_index()
employee_hypertension_case =
```

```
pandas.DataFrame(data.groupby('Employment_Status')
['Hypertension'].count()).reset_index()

# Tight Layout
mat.tight_layout(pad=5)

# Grid Plotting
figure = mat.figure(figsize=(25, 15))
gridspec = GridSpec(3, 3, figure=figure)

# Subplots
ax1 = figure.add_subplot(gridspec[0, 0])
ax2 = figure.add_subplot(gridspec[0, 2])
ax3 = figure.add_subplot(gridspec[0:2, 1])
ax4 = figure.add_subplot(gridspec[1, 0])
ax5 = figure.add_subplot(gridspec[1, 2])
ax6 = figure.add_subplot(gridspec[2, :])

sns.barplot(age_Hypertension_case, x='Age_Group', y='Hypertension',
            ax=ax1,
            color='skyblue',
            edgecolor='black')
ax1.set_title('Hypertension Case among age group')

sns.barplot(Smoking_Hypertension_case, x='Smoking_Status',
y='Hypertension', ax=ax2, color='blue', edgecolor='white')
ax2.set_title('Hypertension Case among smokers')

sns.barplot(Urban_rural_ht_case, x='Urban_Rural', y='Hypertension',
ax=ax4, color='green', edgecolor='black')
ax4.set_title('Hypertension Case in region')

sns.barplot(Stress_hypertension_case, x='Stress_Level',
y='Hypertension', ax=ax3, color='red', edgecolor='white')
ax3.set_title('Hypertension Case to stress levels')

sns.barplot(employee_hypertension_case, x='Employment_Status',
y='Hypertension', ax=ax5, color='xkcd:plum', edgecolor='white')
ax5.set_title('Hypertension Case to employement status')


# Chi2 Statistics
columns = ['Urban_Rural', 'Stress_Level', 'Employment_Status',
'Smoking_Status']

chi2s = []
p_values = []
for index in range(len(columns)):

    # Contingency Table
```

```
    contingency_table = pandas.crosstab(data[columns[index]], 
data['Hypertension'])

    # Chi2
    chi2, p_value, dof, expected = chi2_contingency(contingency_table)
    chi2s.append(chi2)
    p_values.append(p_value)


line = sns.lineplot(x=columns, y=chi2s, mec='0.1', marker='o', 
linestyle='-', color='C1', ax=ax6)
line = sns.lineplot(x=columns, y=p_values, mec='0.1', marker='o', 
linestyle='-', color='C3', ax=ax6)
ax6.set_title('Chi2 Statistics and P Values')
ax6.legend([mat.Line2D([0], [0], color=color, linestyle='-', 
marker='o', mec='0.1') for color in ['C1', 'C3']], ['Chi2', 'P-
Value'], loc='upper right')
line.lines[0].set_markevery(1)

mat.show()
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119: 
FutureWarning: use_inf_as_na option is deprecated and will be removed 
in a future version. Convert inf values to NaN before operating 
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119: 
FutureWarning: use_inf_as_na option is deprecated and will be removed 
in a future version. Convert inf values to NaN before operating 
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119: 
FutureWarning: use_inf_as_na option is deprecated and will be removed 
in a future version. Convert inf values to NaN before operating 
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119: 
FutureWarning: use_inf_as_na option is deprecated and will be removed 
in a future version. Convert inf values to NaN before operating 
instead.
  with pd.option_context('mode.use_inf_as_na', True):

<Figure size 640x480 with 0 Axes>
```



- Hypertension Case are more in adults.
- Remaining plot shows equal distribution of hypertension cases. (Which will be understood using chisquare test for categorical relationship)

- Chi-Square Test Inference

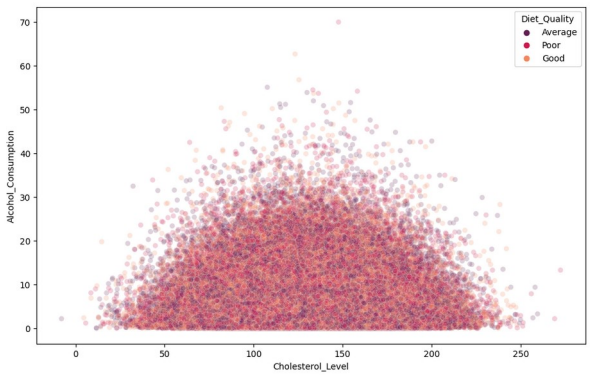
- Could not find any correaltion or association or nature of dependence of the feature like ('Urban_Rural', 'Stress_Level', 'Employment_Status', 'Smoking_Status') with Hypertension.
- That means people have equal amount of hypertension cases in each categories in repsective features

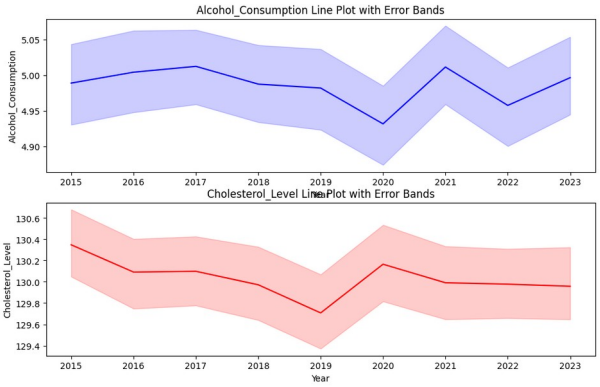
Visual - Line plots

```
# Alcohol and Cholesterol
figure = mat.figure(figsize=(25, 7))
gridspace = GridSpec(2, 2, figure=figure)

# Building axis
ax1 = figure.add_subplot(gridspace[0, 1])
ax2 = figure.add_subplot(gridspace[1, 1])
ax3 = figure.add_subplot(gridspace[:, 0])

columns = ['Alcohol_Consumption', 'Cholesterol_Level']
color = ['blue', 'red']


for i, ax in enumerate([ax1, ax2, ax3], start=0):
```

```python
    if i <= 1:
        sns.lineplot(
            data=data,
            x='Year',
            y=columns[i],
            ax=ax,
            color=color[i]
        )
        ax.set_title(f'{columns[i]} Line Plot with Error Bands')
    else:
        sns.scatterplot(
            data=data,
            x=columns[i-1],
            y=columns[i-2],
            hue='Diet_Quality',
            palette='rocket',
            ax=ax,
            color='blue',
            alpha=0.2
        )

# Pearson Correlation Value

stats, p_value = pearsonr(x=data['Alcohol_Consumption'],
y=data['Cholesterol_Level'])
print('Pearson Correlation Value -', stats,'\nP_value -', p_value)
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):

Pearson Correlation Value - -0.0015933780013966865
P_value - 0.4028461115798291
```





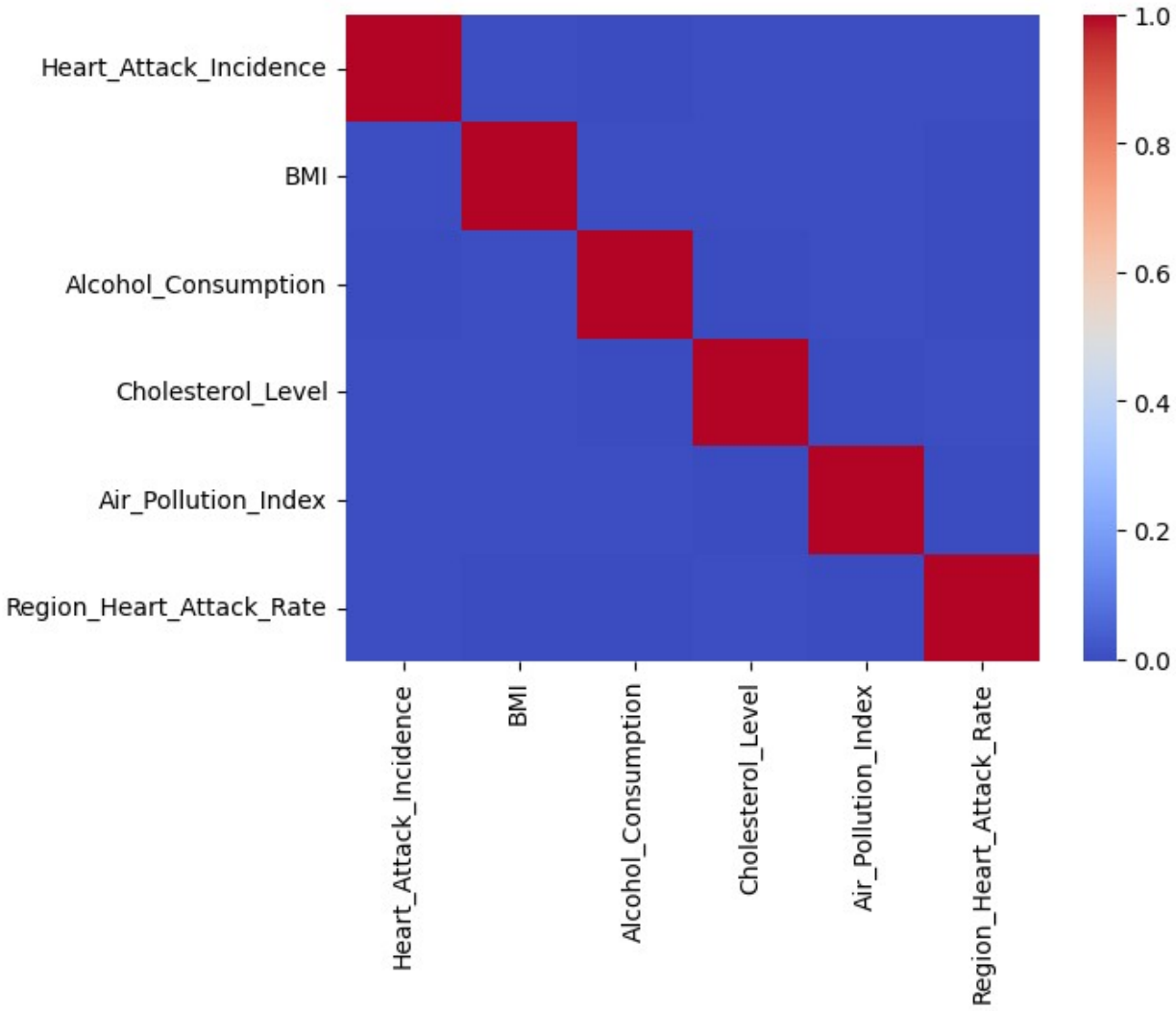
- Both are related, are they ?
- Since the correlation as well as the scatter plot do not show significant patterns or any signs of relationship.
- Conclusion is there is no correlation between the Alcohol and cholesterol.
- We also have p_value to be more than 0.05, failing to reject the null hypothesis. Which shows that we do not have enough evidence to support the linear relationship and thus the occurance of the datapoints are random.

```
data.select_dtypes(include=['int', 'float']).head()
```

```
   Heart_Attack_Incidence  Year   BMI  Alcohol_Consumption
Family_History  \
0                       0  2018  25.6                  4.2
1
1                       0  2021  36.7                  2.4
0
2                       1  2022  28.6                 29.5
0
3                       0  2015  27.6                  4.2
0
4                       0  2015  15.2                  4.3
0

   Hypertension  Cholesterol_Level  Diabetes  Air_Pollution_Index  \
0             0              154.4         0                31.58
1             1               75.0         1                46.22
2             0              121.9         0                15.69
3             1              152.3         0                26.50
4             1              130.3         0                11.21

   Region_Heart_Attack_Rate
0                      1.92
1                     14.16
2                      3.49
3                      3.24
4                      9.98
```

```
continous_variable = data.select_dtypes(include=['int',
'float']).drop(columns=['Year', 'Family_History', 'Hypertension',
'Diabetes'])
corr_matrix = continous_variable.corr()

sns.heatmap(
    corr_matrix,
    cmap='coolwarm'
)
```

```
<Axes: >
```



- Cannot see a pattern in the correlation matrix. None of them have correlation between the continous variables

Machine Learning

- Data Preprocessing
- Data Transformation
- Feature Selection
- Model Training
- Model Prediction
- Model Evaluation

```
# Importing the modules from the libraries
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, f1_score,
accuracy_score, recall_score, precision_score
from imblearn.over_sampling import SMOTE

from sklearn.ensemble import RandomForestClassifier
```

- Data Preprocessing and Data Transformation

```
categorical = data.select_dtypes(include='object')
numerical = data.select_dtypes(include=['float',
'int']).drop(columns='Year')

# Encoded Categorical Data
encoder = OneHotEncoder(sparse_output=False)
encoded_data = encoder.fit_transform(categorical)

encoded_df = pandas.DataFrame(
    encoded_data,
columns=encoder.get_feature_names_out(categorical.columns)
)

# Scaled Numerical Data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(numerical)

scaled_df = pandas.DataFrame(scaled_data, columns=numerical.columns)


# Merging the dataset
data_m = pandas.concat([encoded_df, numerical], axis=1)
```

- Feature Selection

```
# Def function for the recursive feature elimination
def feature_selection(model, feature_to_choose, X, y):
```

```
    rfe = RFE(estimator=model, n_features_to_select=feature_to_choose)

    rfe.fit(X, y)
    return rfe

data_m.head()
```

```
   State_Baden-Württemberg  State_Bavaria  State_Berlin  State_Hamburg
\
0                      0.0            0.0           0.0            0.0

1                      0.0            0.0           0.0            0.0

2                      0.0            0.0           0.0            0.0

3                      0.0            0.0           0.0            0.0

4                      0.0            0.0           0.0            1.0


   State_Hesse  State_Lower Saxony  State_North Rhine-Westphalia  \
0          0.0                 1.0                           0.0
1          0.0                 0.0                           0.0
2          1.0                 0.0                           0.0
3          0.0                 1.0                           0.0
4          0.0                 0.0                           0.0

   State_Saxony  Age_Group_Adult  Age_Group_Youth  ...  \
0           0.0              0.0              1.0  ...
1           1.0              1.0              0.0  ...
2           0.0              0.0              1.0  ...
3           0.0              1.0              0.0  ...
4           0.0              1.0              0.0  ...

   Employment_Status_Unemployed  Heart_Attack_Incidence   BMI  \
0                           0.0                       0  25.6
1                           1.0                       0  36.7
2                           0.0                       1  28.6
3                           0.0                       0  27.6
4                           0.0                       0  15.2

   Alcohol_Consumption  Family_History  Hypertension
Cholesterol_Level  \
0                  4.2               1             0
154.4
1                  2.4               0             1
75.0
2                 29.5               0             0
121.9
3                  4.2               0             1
152.3
```

```
4                   4.3                0              1
130.3

   Diabetes  Air_Pollution_Index  Region_Heart_Attack_Rate
0         0                31.58                      1.92
1         1                46.22                     14.16
2         0                15.69                      3.49
3         0                26.50                      3.24
4         0                11.21                      9.98

[5 rows x 49 columns]
```

```
# Data Assignment
X = data_m.drop(columns=['Diabetes'])
y = data_m['Diabetes']

# Oversampling method to balance out the feature
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

# Train Test Split
X_train, X_test, y_train, y_test = train_test_split(X_resampled,
y_resampled, random_state=42, test_size=0.2)


# Model Initialization
logistic_model = LogisticRegression(max_iter=500)
rfe_linear = feature_selection(model=logistic_model,
feature_to_choose=5, X=X_resampled, y=y_resampled)
```

- Logistic Regression

```
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.fit_transform(X_test)

logistic_model.fit(X_train_scaled, y_train)

predict = logistic_model.predict(X_test_scaled)
predict_training = logistic_model.predict(X_train_scaled)

class_report = classification_report(y_test, predict,
output_dict=True)
class_report_training = classification_report(y_train,
predict_training, output_dict=True)

class_report_df = pandas.DataFrame(class_report)
class_report_training_df = pandas.DataFrame(class_report_training)

# Test Set Classification Report
print('Classification Report for Test Set:')
class_report_df
```

```
Classification Report for Test Set:

                        0             1  accuracy     macro avg
weighted avg
precision        0.632894      0.637647  0.635233      0.635270
0.635271
recall           0.643225      0.627248  0.635233      0.635236
0.635233
f1-score         0.638017      0.632405  0.635233      0.635211
0.635210
support     44109.000000  44150.000000  0.635233  88259.000000
88259.000000
```

```
# Training Set Classification Report
print('Classification Report for Training Set:')
class_report_training_df
```

```
Classification Report for Training Set:

                         0              1  accuracy      macro avg  \
precision         0.634055       0.637993  0.635994       0.636024
recall            0.643429       0.628558  0.635994       0.635994
f1-score          0.638707       0.633240  0.635994       0.635974
support     176537.000000  176496.000000  0.635994  353033.000000

             weighted avg
precision        0.636023
recall           0.635994
f1-score         0.635974
support     353033.000000
```

- Random Forest Classifier

```
# Training the random forest classifier
forest_classifier = RandomForestClassifier(
    random_state=42,
    n_estimators=200,
    criterion='gini',
    max_depth=5
)

X_trainr, X_testr, y_trainr, y_testr = train_test_split(X_resampled,
y_resampled, random_state=29, test_size=0.2)

X_trainr_scaled = scaler.fit_transform(X_trainr)
X_test_scaled = scaler.fit_transform(X_testr)

# Fitting the data to the model
forest_classifier.fit(X_trainr_scaled, y_trainr)
```

```
# Predicting the model - Test Dataset
predict = forest_classifier.predict(X_test_scaled)

# Training Dataset
predict_training = forest_classifier.predict(X_train_scaled)

# Evalutation
classification_test_random_forest =
pandas.DataFrame(classification_report(y_testr, predict,
output_dict=True))
classification_training_random_forest =
pandas.DataFrame(classification_report(y_trainr, predict_training,
output_dict=True))

# Test Set Score
print('Classification Test Set Random Forest :')
classification_test_random_forest
```

```
Classification Test Set Random Forest :
```

```
                      0             1  accuracy     macro avg
weighted avg
precision      0.777911      0.502494  0.505093      0.640202
0.640244
recall         0.014680      0.995807  0.505093      0.505243
0.505093
f1-score       0.028815      0.667939  0.505093      0.348377
0.348279
support    44143.000000  44116.000000  0.505093  88259.000000
88259.000000
```

```
print('Classification Training Set Random Forest :')
classification_training_random_forest
```

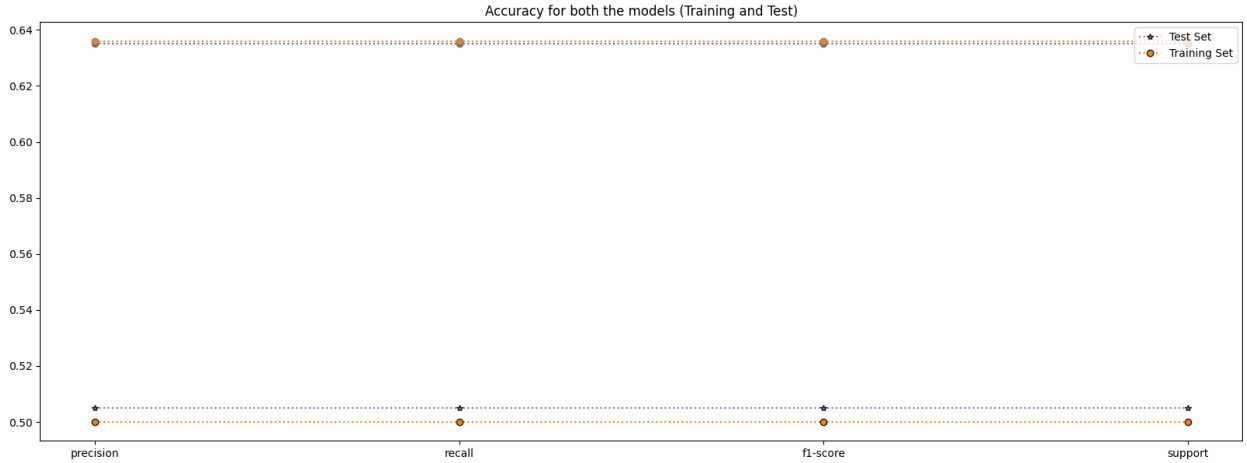
```
Classification Training Set Random Forest :
```

```
                       0              1  accuracy      macro avg  \
precision       0.503145       0.500048  0.500058       0.501596
recall          0.003173       0.996867  0.500058       0.500020
f1-score        0.006306       0.666012  0.500058       0.336159
support    176503.000000  176530.000000  0.500058  353033.000000

            weighted avg
precision       0.501596
recall          0.500058
f1-score        0.336184
support    353033.000000
```

```
mat.figure(figsize=(20, 7))
mat.tight_layout(pad=5)
```

```
# Random Forest Regression
mat.plot(classification_test_random_forest['accuracy'], 'C4*:',
markevery=1, mec='0.1')
mat.plot(classification_training_random_forest['accuracy'], 'C1o:',
markevery=1, mec='0.1')

# Logistic Regression
mat.plot(class_report_df['accuracy'], 'C5o:', markevery=1, mec='0.5')
mat.plot(class_report_training_df['accuracy'], 'C1o:', markevery=1,
mec='0.5')

mat.title('Accuracy for both the models (Training and Test)')
mat.legend(['Test Set', 'Training Set'], loc='upper right')
```

```
<matplotlib.legend.Legend at 0x7e68fafbbe20>
```



- Logistic Regression has beaten the most robust classification model (Random Forest Classifier)
- The accuracy score from logistic regression can be observed 23.17% higher than the score calculated by random forest classifier

What I tried ?

- Tried using parameter tunning, however, it has not been shown here. But I used several combination with the help of Grid Search CV to validate the best performing combination of parameters.
- Had to use the oversampling method to equalize the values of our target variable as it was imbalanced.
- The score we used to get was biased towards 0 which we do not want our model to be. It should be unbiased. Here is where imblearn's oversampling method comes into play which increases the datapoint making them equally or evenly distributed.

What can be done to increase the score?

- Several more models, can be trained and used to predict the diabetes feature, which is the actual feature taken as y.

- Neural Network using sigmoid activation function as this problem is classification problem.
- Decision tree with more parameters given to it, to get the most out of these tree-based models.
- The same thing also can be done with this random forest to enhance the accuracy score.