project machine learning model for conflicting variant genomic classification

under supervision /Dr Mohamed el sayeh

Student name :sarah youssef thomas

ID:211002134

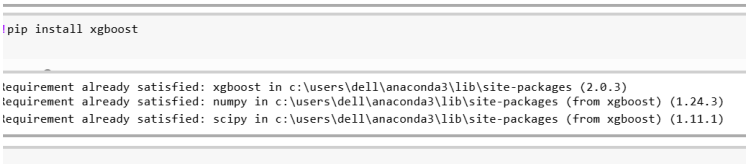
INTRODUCTION

ClinVar is a public database that provides information about human genetic variants. These variants are typically classified by clinical laboratories on a categorical scale that includes benign, likely benign, uncertain significance, likely pathogenic, and pathogenic. When different laboratories assign conflicting classifications to the same variant, it can create confusion for clinicians and researchers trying to determine the variant's impact on a patient's disease. our ,objective is to create a classification machine learning model to predict whether or not a variant has conflicting classifications or not

Work pipeline

1.first we upload the necessary libraries for our analysis

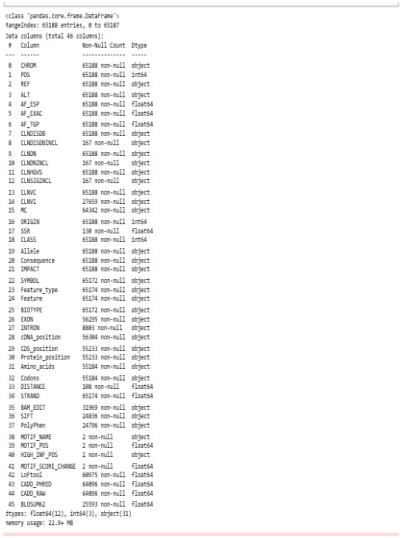
```
#Importing neccesary libaries
# import necessary libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
%matplotlib inline
from scipy import stats
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import stats
from scipy.stats import boxcox
from scipy.stats.mstats import winsorize
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import Lasso
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import confusion_matrix, r2_score, mean_squared_error
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
```



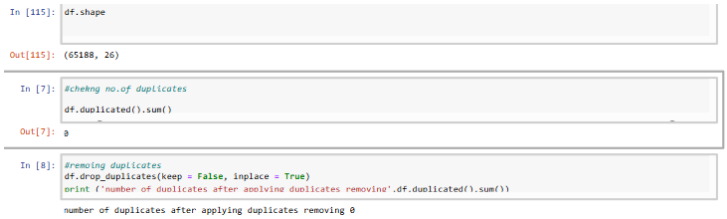
```
!pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\dell\anaconda3\lib\site-packages (2.0.3)
Requirement already satisfied: numpy in c:\users\dell\anaconda3\lib\site-packages (from xgboost) (1.24.3)
Requirement already satisfied: scipy in c:\users\dell\anaconda3\lib\site-packages (from xgboost) (1.11.1)
```

2. then we loaded the data set which is contain 46 column and 65187 entries
from exploring data we noticed that we have columns with high percentage of nulls and columns with very low percentage of null so removing the nulls will from all data set will not be applicable



3.we started checked data shape and duplications which was = zero



4. we creates a new DataFrame, `var_df`, which summarizes the characteristics of each column in an existing DataFrame, `df`. The new DataFrame includes columns for the variable name, data type, missing percentage, a flag indicating whether the variable is numeric or categorical, and the count of unique values. Initially, `var_df` is created with specified columns but no rows. The missing percentages for each column in `df` are calculated and sorted in descending order. The code then iterates through each column in `df`, determines its data type, calculates its missing percentage, and counts its unique values. Based on the data type, the variable is flagged as either numeric or categorical. These details are stored in `var_df` using `pd.concat` to append a new row for each column. Finally, the code outputs the structure and content of `var_df`, revealing that the DataFrame contains 46 entries with five columns: 'variable_name', 'data_type', 'missing_percentage', 'flag', and 'unique_values_count'. The data types in `var_df` consist of 'float64' for 'missing_percentage' and 'object' for the other columns. The summary shows a diverse range of missing percentages and unique values for each variable, providing a comprehensive overview of the dataset's structure and characteristics.

From this we found that we have a columns that have percentage of nulls greater than 40% precent so we talked decision to remove it

```
# create new dataframe with data type, nulls, & unique values
var_df = pd.DataFrame(columns=['variable_name', 'data_type', 'missing_percentage', 'flag', 'unique_values_count'])

missing_percentages = df.isnull().mean() * 100
missing_percentages = missing_percentages.sort_values(ascending=False)

# create variables and flag as numeric or categorial
for col in df.columns:
    data_type = df[col].dtype
    missing_percentage = missing_percentages[col]
    unique_values_count = df[col].nunique()
    if data_type == 'int64' or data_type == 'float64':
        flag = 'numeric'
    else:
        flag = 'categorical'


 # concat values obtained into a new dataframe called 'var_df'
    var_df = pd.concat([var_df, pd.DataFrame({'variable_name': [col], 'data_type': [data_type], 'missing_percentage': [missing

var_df.info()
print(var_df)
```

```
# on the original data set
threshold = 50

missing_percentages = df.isnull().mean() * 100
columns_to_drop = missing_percentages[missing_percentages > threshold].index.tolist()

df = df.drop(columns=columns_to_drop)

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 65188 entries, 0 to 65187
Data columns (total 31 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   CHROM             65188 non-null  object
 1   POS               65188 non-null  int64
 2   REF               65188 non-null  object
 3   ALT               65188 non-null  object
 4   AF_ESP            65188 non-null  float64
 5   AF_EXAC           65188 non-null  float64
 6   AF_TGP            65188 non-null  float64
 7   CLNDISDB          65188 non-null  object
 8   CLNDN             65188 non-null  object
 9   CLNHGVS           65188 non-null  object
 10  CLNVC             65188 non-null  object
 11  MC                64342 non-null  object
 12  ORIGIN            65188 non-null  int64
 13  CLASS             65188 non-null  int64
 14  Allele            65188 non-null  object
 15  Consequence       65188 non-null  object
 16  IMPACT            65188 non-null  object
 17  SYMBOL            65172 non-null  object
 18  Feature_type      65174 non-null  object
 19  Feature           65174 non-null  object
 20  BIOTYPE           65172 non-null  object
 21  EXON              56295 non-null  object
 22  cDNA_position     56304 non-null  object
 23  CDS_position      55233 non-null  object
 24  Protein_position  55233 non-null  object
 25  Amino_acids       55184 non-null  object
 26  Codons            55184 non-null  object
 27  STRAND            65174 non-null  float64
 28  LoFtool           60975 non-null  float64
 29  CADD_PHRED        64096 non-null  float64
 30  CADD_RAW          64096 non-null  float64
dtypes: float64(7), int64(3), object(21)
memory usage: 15.4+ MB
```

5.we did a heatmap to understand the correlation between the numerical values based on it and the provided information about column we took decision to remove the following columns ‘'EXON', 'CLNDISDB','Feature', 'MC','CADD_RAW’as EXON = contains dates, not performing time series, not relevant, CLNDISB = Provides MedGen database identifiers, not relevant, Feature: Value included in consequence column, MC ‘identifier; 'CADD_RAW: directly related to CADD_PHRED - only CADD_PHRED is needed with respect to genetic mutations, it uses a scale that is easier to work with.

EXON = contains dates, not performing time series, not relevant, CLNDISB = Provides MedGen database identifiers, not relevant, Feature: Value included in consequence column, MC 'identifier; 'CADD_RAW: directly related to CADD_PHRED - only CADD_PHRED is needed with respect to genetic mutations, it uses a scale that is easier to work with.

```
[14]: df.drop(['EXON', 'CLNDISDB','Feature',
               'MC','CADD_RAW',], axis=1, inplace=True)
```

6. then we check the nummerical and categorical values and we used still missing function to identify if we having missings

```
7]: # Iterate over the columns of the DataFrame and isolate missing columns
for column in df.columns:
    if df[column].isnull().any():
        still_missing[column] = df[column]

print("Columns with missing values:")
still_missing.info()
```

```
Columns with missing values:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 65188 entries, 0 to 65187
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   SYMBOL            65172 non-null  object
 1   Feature_type      65174 non-null  object
 2   BIOTYPE           65172 non-null  object
 3   cDNA_position     56304 non-null  object
 4   CDS_position      55233 non-null  object
 5   Protein_position  55233 non-null  object
 6   Amino_acids       55184 non-null  object
 7   Codons            55184 non-null  object
 8   STRAND            65174 non-null  float64
 9   LoFtool           60975 non-null  float64
 10  CADD_PHRED        64096 non-null  float64
dtypes: float64(3), object(8)
memory usage: 5.5+ MB
```

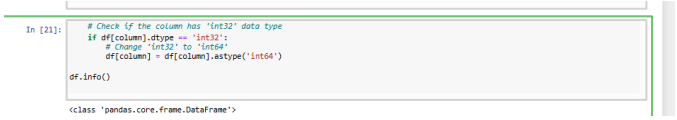
Then we applied still missing function to fill our missing depending on data type and we updated our data with it

```
for column in still_missing.columns:
    if still_missing[column].dtype == 'object':
        # Fill missing values in object columns with forward fill (ffill)
        still_missing[column] = still_missing[column].fillna(method='ffill')
    elif still_missing[column].dtype == 'float64':
        # Interpolate missing values in float64 columns
        still_missing[column] = still_missing[column].interpolate()

# Calculate the mean of the 'LoFtool' column
try:
    loftool_mean = still_missing['LoFtool'].fillna(method='ffill')
  # Rest of your code using loftool_mean
except KeyError:
    print("'LoFtool' column not found. Skipping mean calculation.")
  # Handle the case where the column is missing
# Check the updated DataFrame information
still_missing.info()
```

```
df.update(still_missing)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 65188 entries, 0 to 65187
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   CHROM             65188 non-null  object
 1   POS               65188 non-null  int64
 2   REF               65188 non-null  object
 3   ALT               65188 non-null  object
 4   AF_ESP            65188 non-null  float64
 5   AF_EXAC           65188 non-null  float64
 6   AF_TGP            65188 non-null  float64
 7   CLNDN             65188 non-null  object
 8   CLNHGVS           65188 non-null  object
 9   CLNVC             65188 non-null  object
 10  ORIGIN            65188 non-null  int64
 11  CLASS             65188 non-null  int64
 12  Allele            65188 non-null  object
 13  Consequence       65188 non-null  object
 14  IMPACT            65188 non-null  object
 15  SYMBOL            65188 non-null  object
 16  Feature_type      65188 non-null  object
 17  BIOTYPE           65188 non-null  object
 18  cDNA_position     65188 non-null  object
 19  CDS_position      65188 non-null  object
 20  Protein_position  65188 non-null  object
 21  Amino_acids       65188 non-null  object
```

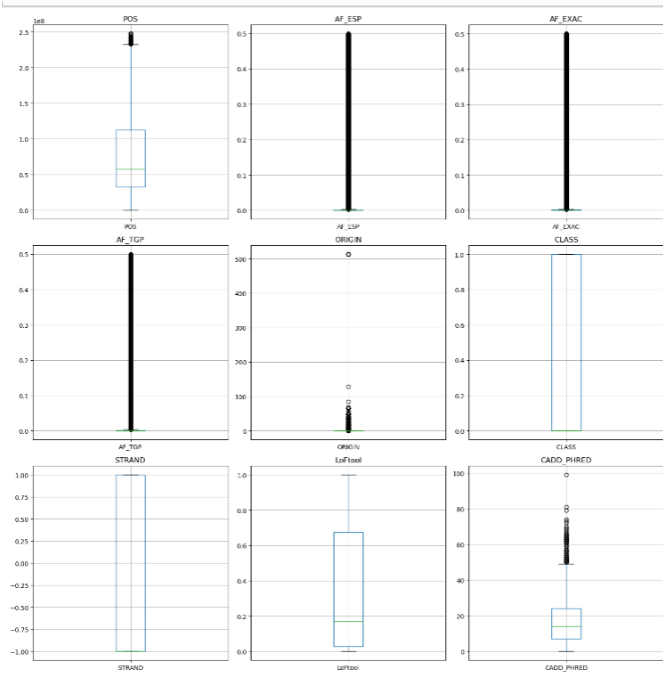
. we also applied transformation of data type from ‘int32’ to ‘int64’

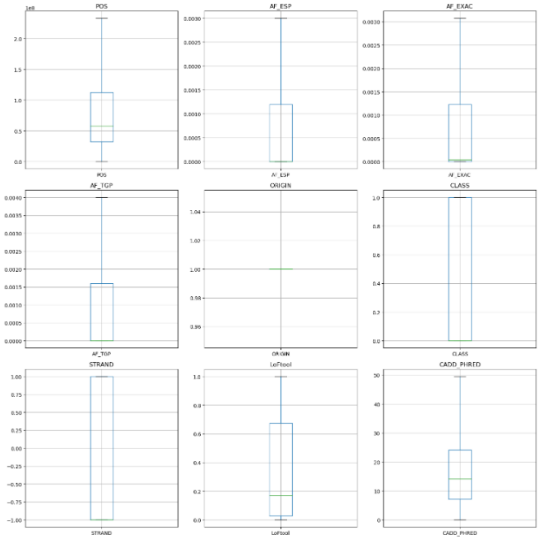


Then we started handling outliers by clapping using IQR

```python
# Function to cap outliers using IQR
def cap_outliers_iqr(df):
    df_capped = df.copy()
    Q1 = df.quantile(0.25)
    Q3 = df.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Cap the outliers for each column
    for column in df.columns:
        df_capped[column] = np.where(df[column] < lower_bound[column], lower_bound[column], df[column])
        df_capped[column] = np.where(df_capped[column] > upper_bound[column], upper_bound[column], df_capped[column])

    return df_capped


# Separate numerical columns
numerical_columns = df.select_dtypes(include=['number'])

# Cap the outliers
df_capped_numerical = cap_outliers_iqr(numerical_columns)

# Fill missing values with the mean of the columns (if there are any missing values)
df_capped_numerical = df_capped_numerical.fillna(df_capped_numerical.mean())

# Update the original DataFrame with capped numerical columns
df_updated = df.copy()
df_updated[numerical_columns.columns] = df_capped_numerical

# Calculate the number of rows and columns for subplots
num_columns = 3
num_rows = (len(numerical_columns.columns) - 1) // num_columns + 1

# Create subplots
fig, axes = plt.subplots(num_rows, num_columns, figsize=(15, 5 * num_rows))

# Iterate through columns and plot boxplots
for i, column in enumerate(numerical_columns.columns):
    row = i // num_columns
    col = i % num_columns
    ax = axes[row][col] if num_rows > 1 else axes[col]  # Adjust if there's only one row
    df_updated.boxplot(column=column, ax=ax)
    ax.set_title(column)

# Remove any empty subplots
for j in range(len(numerical_columns.columns), num_rows * num_columns):
    fig.delaxes(axes.flatten()[j])

# Adjust layout and display the plots
plt.tight_layout()
plt.show()
```

Here is our values before and after removing outliers





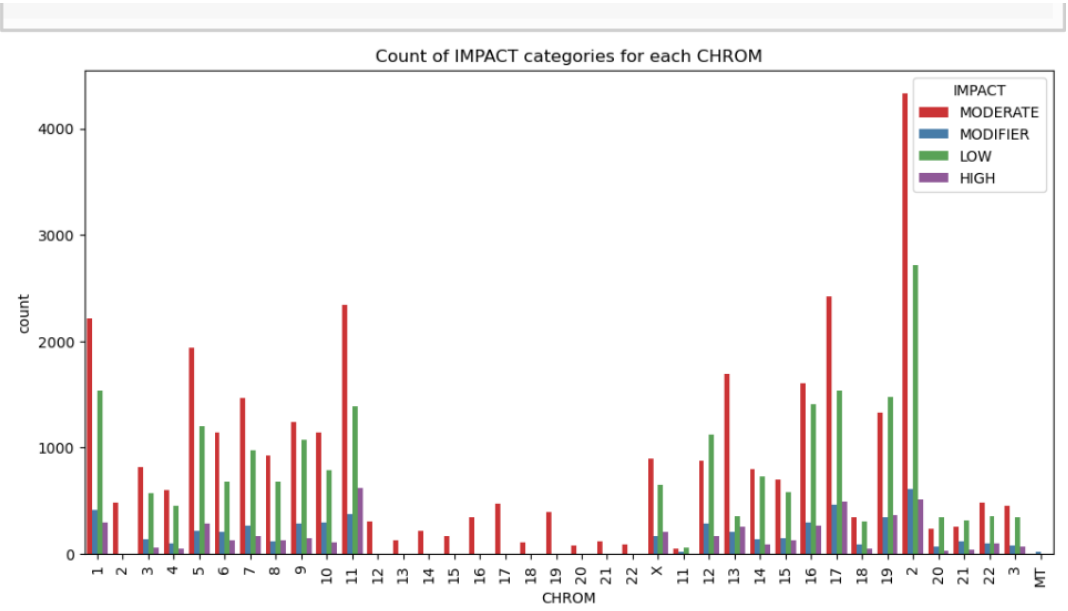
Now we have our data cleaned and with no outliers

```
5]: df_updated.info()
```

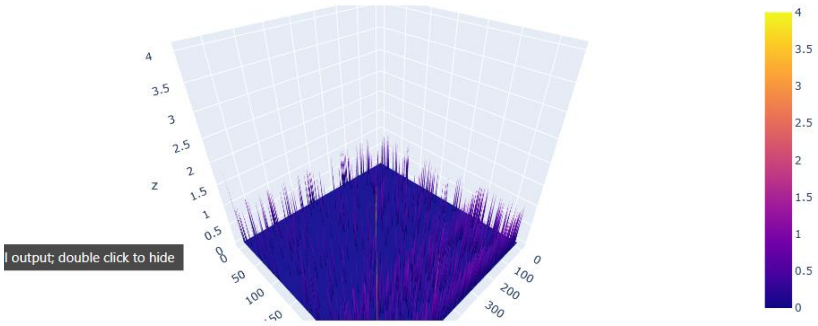
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 65188 entries, 0 to 65187
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   CHROM             65188 non-null  object
 1   POS               65188 non-null  float64
 2   REF               65188 non-null  object
 3   ALT               65188 non-null  object
 4   AF_ESP            65188 non-null  float64
 5   AF_EXAC           65188 non-null  float64
 6   AF_TGP            65188 non-null  float64
 7   CLNDN             65188 non-null  object
 8   CLNHGVS           65188 non-null  object
 9   CLNVC             65188 non-null  object
 10  ORIGIN            65188 non-null  float64
 11  CLASS             65188 non-null  float64
 12  Allele            65188 non-null  object
 13  Consequence       65188 non-null  object
 14  IMPACT            65188 non-null  object
 15  SYMBOL            65188 non-null  object
 16  Feature_type      65188 non-null  object
 17  BIOTYPE           65188 non-null  object
 18  cDNA_position     65188 non-null  object
 19  CDS_position      65188 non-null  object
 20  Protein_position  65188 non-null  object
 21  Amino_acids       65188 non-null  object
 22  Codons            65188 non-null  object
 23  STRAND            65188 non-null  float64
 24  LoFtool           65188 non-null  float64
 25  CADD_PHRED        65188 non-null  float64
dtypes: float64(9), object(17)
memory usage: 12.9+ MB
```

EDA Exploratory data analysis

. 1.there are about 4,000 chromosomes that have a moderate impact identifier for the consequence type.

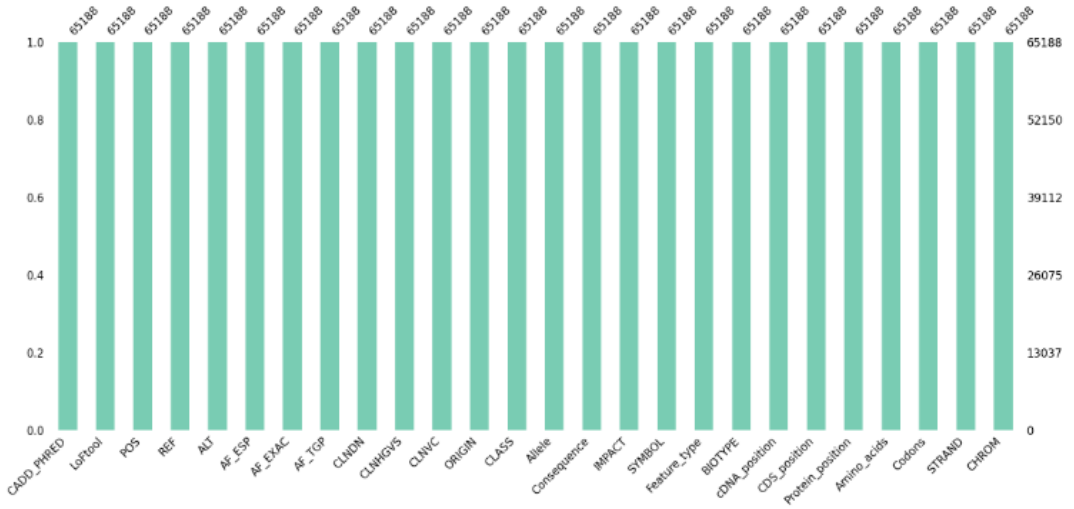


2.we also did a visualization to demonstrated correlation between the 'POS', 'CADD_PHRED columns which show datapoints frequency

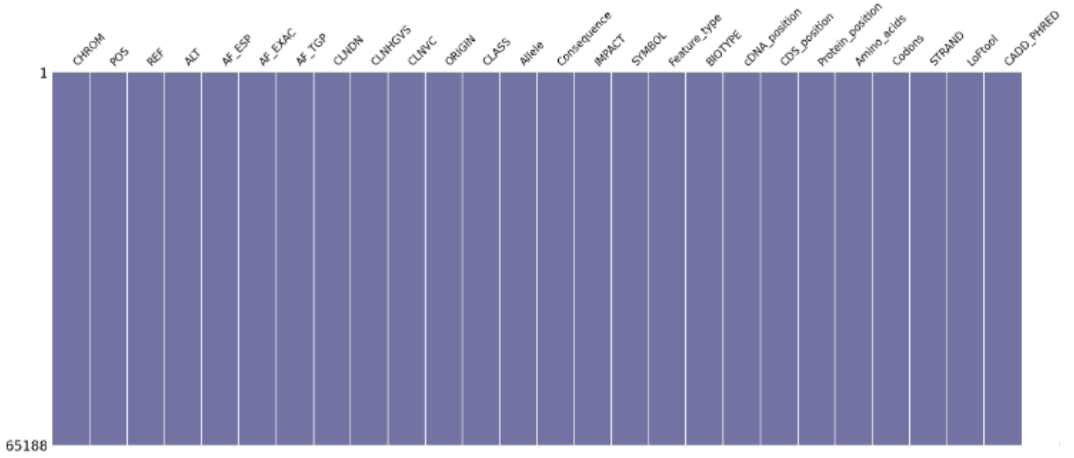


3, this 2 visualization to confirm that we don’t have missing values

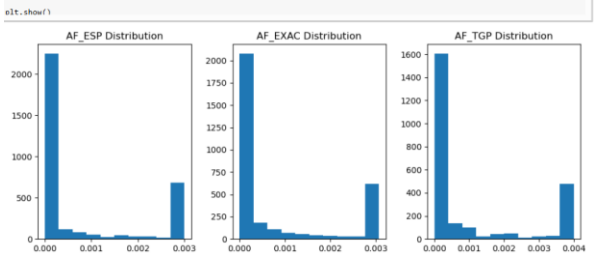
```
import missingno as msno
msno.bar(df_updated,color='#79ccb3',sort='descending')
plt.show()
```




```
msno.matrix(df_updated,color=(0.45,0.45,0.64),figsize=(27, 10), width_ratios=(10, 0))
plt.show()
```

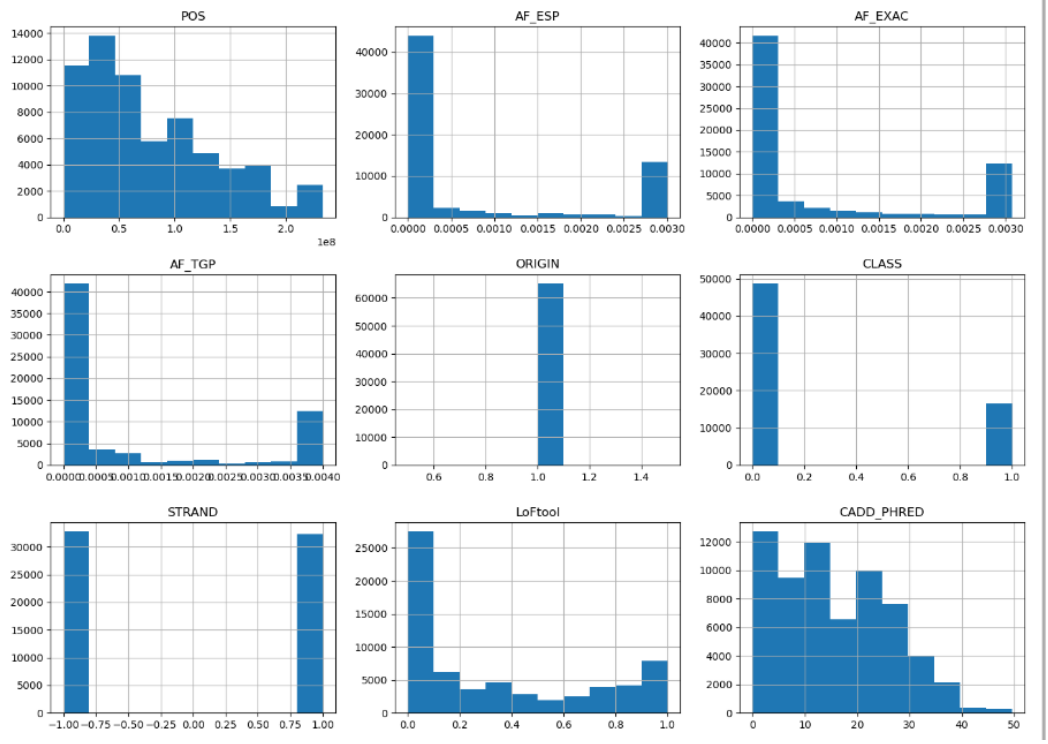


4, the distribution shows that the values of allele frequency are concentrated in region from 0.00 to 0.004 'AF_ESP, 'AF_EXAC', AF_TGP'

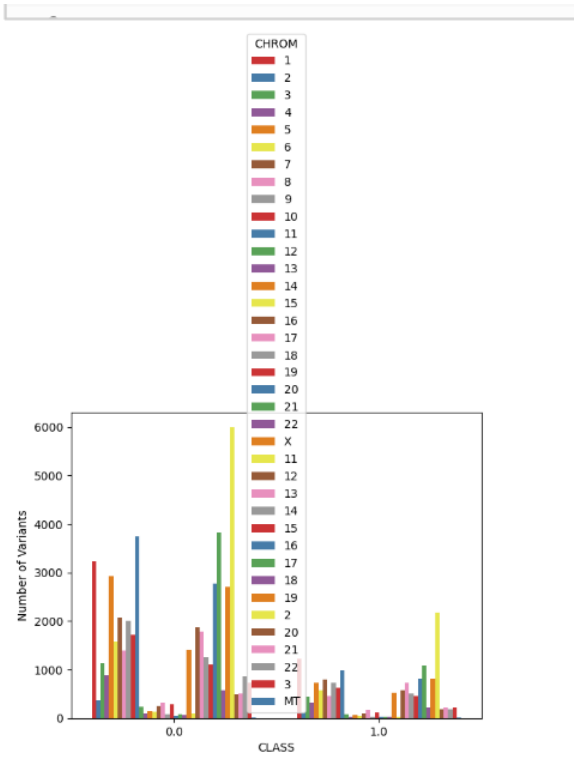


5. a histogram that shows the distribution of all numerical columns .

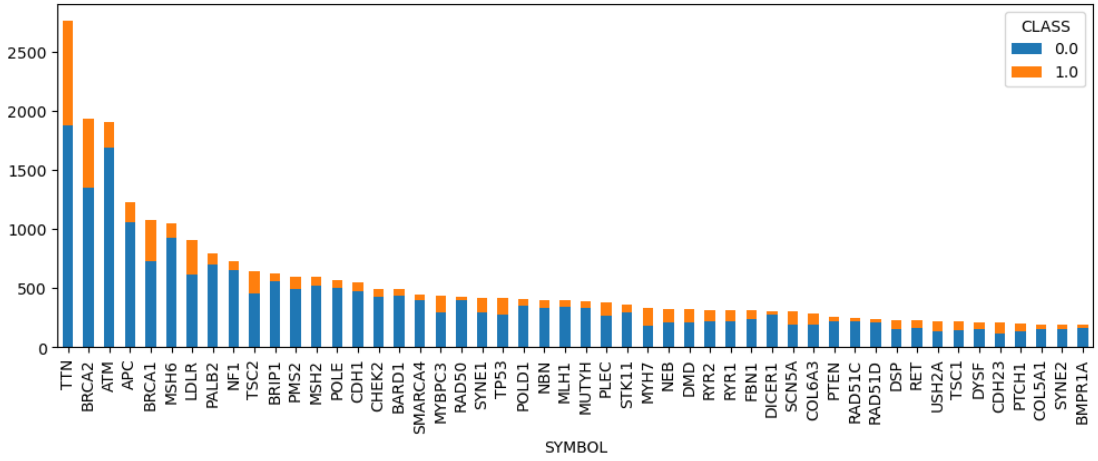
```
[33]: # Plot histograms of all columns with larger size
df_updated.hist(figsize=(14, 10))
plt.tight_layout()
plt.show()
```



6.number of variant in chromosomes with respect to class from that we can identify that class 0 have higher number of variants



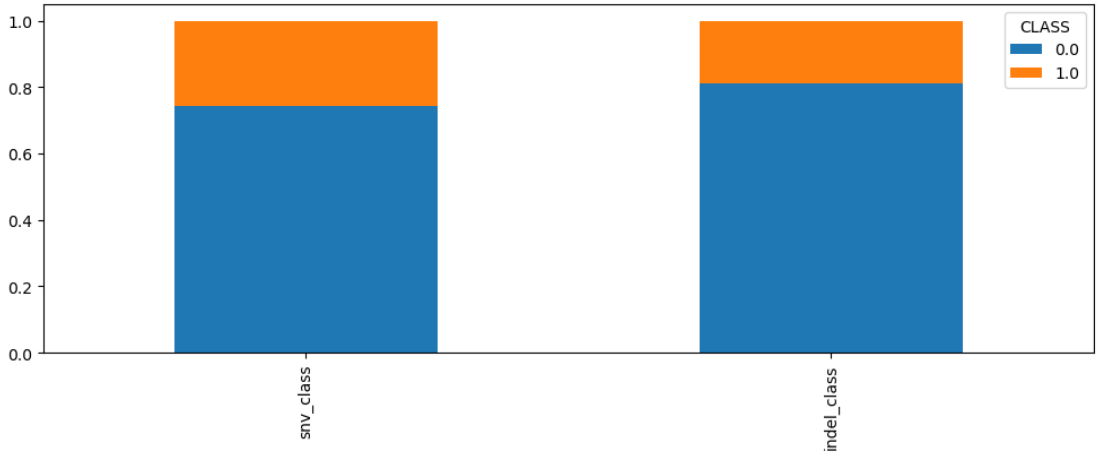
7, the distribution of genes in each class we can identify that gene TNN is the most common specially in class 0



8. we also did that could to identify types of alterations between the reference and alternative alleles

```
In [36]: snvs = df_updated.loc[(df_updated.REF.str.len()==1) & (df_updated.ALT.str.len()==1)]
         indels = df_updated.loc[(df_updated.REF.str.len()>1) | (df_updated.ALT.str.len()>1)]

In [37]: len(df_updated) == (len(snvs) + len(indels))

Out[37]: True

In [38]: snp_indel = pd.concat([snvs.CLASS.value_counts(normalize=True).rename('snv_class'),
                                indels.CLASS.value_counts(normalize=True).rename('indel_class')],
                               axis=1).T
```

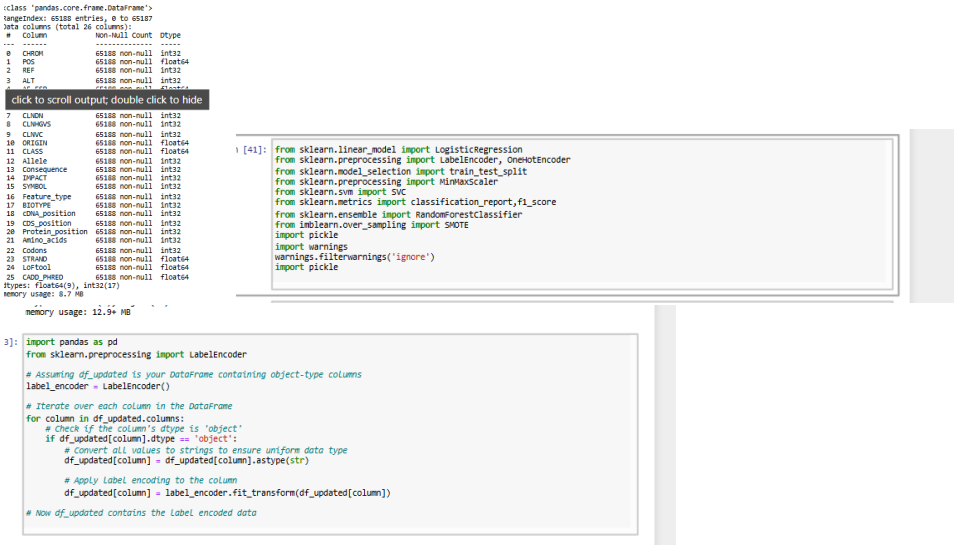
And we visualized it as following inference that indels induced alteration is more frequent in class o while single nucleotide polymorphism is more in class 1



9.Finally a pairplot for the distribution of the numerical values in data

model preprocessing and training

first we imported the necessary libraries and encoded the categorical values using label encoder



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 65188 entries, 0 to 65187
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   CHROM             65188 non-null  int32
 1   POS               65188 non-null  float64
 2   REF               65188 non-null  int32
 3   ALT               65188 non-null  int32
 4   AF_ESP            65188 non-null  float64
 5   AF_EXAC           65188 non-null  float64
 6   AF_TGP            65188 non-null  float64
 7   CLNDN             65188 non-null  int32
 8   CLNHGVS           65188 non-null  int32
 9   CLNVC             65188 non-null  int32
 10  ORIGIN            65188 non-null  float64
 11  CLASS             65188 non-null  float64
 12  Allele            65188 non-null  int32
 13  Consequence       65188 non-null  int32
 14  IMPACT            65188 non-null  int32
 15  SYMBOL            65188 non-null  int32
 16  Feature_type      65188 non-null  int32
 17  BIOTYPE           65188 non-null  int32
 18  cDNA_position     65188 non-null  int32
 19  CDS_position      65188 non-null  int32
 20  Protein_position  65188 non-null  int32
 21  Amino_acids       65188 non-null  int32
 22  Codons            65188 non-null  int32
 23  STRAND            65188 non-null  float64
 24  LoFtool           65188 non-null  float64
 25  CADD_PHRED        65188 non-null  float64
dtypes: float64(9), int32(17)
memory usage: 8.7 MB
```

click to scroll output; double click to hide

```
memory usage: 12.9+ MB
```

```
[41]: from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report,f1_score
from sklearn.ensemble import RandomForestClassifier
from imblearn.over_sampling import SMOTE
import pickle
import warnings
warnings.filterwarnings('ignore')
import pickle
```

```
3]: import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Assuming df_updated is your DataFrame containing object-type columns
label_encoder = LabelEncoder()

# Iterate over each column in the DataFrame
for column in df_updated.columns:
    # Check if the column's dtype is 'object'
    if df_updated[column].dtype == 'object':
        # Convert all values to strings to ensure uniform data type
        df_updated[column] = df_updated[column].astype(str)

        # Apply label encoding to the column
        df_updated[column] = label_encoder.fit_transform(df_updated[column])

# Now df_updated contains the label encoded data
```

Then we created 2 new features:

1.mean of 'LoFtool' and 'CADD_PHRED' as 'Pathogenicity_Score_Mean'

2. mean of 'AF_ESP', 'AF_EXAC', and 'AF_TGP' as Allele_Freq_Mean

```
]: #creation of new compostions
# Calculate mean of 'AF_ESP', 'AF_EXAC', and 'AF_TGP'
df_updated['Allele_Freq_Mean'] = df_updated[['AF_ESP', 'AF_EXAC', 'AF_TGP']].mean(axis=1)

# Calculate mean of 'LoFtool' and 'CADD_PHRED'
df_updated['Pathogenicity_Score_Mean'] = df_updated[['LoFtool', 'CADD_PHRED']].mean(axis=1)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 65188 entries, 0 to 65187
Data columns (total 28 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CHROM                   65188 non-null  int32
 1   POS                     65188 non-null  float64
 2   REF                     65188 non-null  int32
 3   ALT                     65188 non-null  int32
 4   AF_ESP                  65188 non-null  float64
 5   AF_EXAC                 65188 non-null  float64
 6   AF_TGP                  65188 non-null  float64
 7   CLNDN                   65188 non-null  int32
 8   CLNHGVS                 65188 non-null  int32
 9   CLNVC                   65188 non-null  int32
 10  ORIGIN                  65188 non-null  float64
 11  CLASS                   65188 non-null  float64
 12  Allele                  65188 non-null  int32
 13  Consequence             65188 non-null  int32
 14  IMPACT                  65188 non-null  int32
 15  SYMBOL                  65188 non-null  int32
 16  Feature_type            65188 non-null  int32
 17  BIOTYPE                 65188 non-null  int32
 18  cDNA_position           65188 non-null  int32
 19  CDS_position            65188 non-null  int32
 20  Protein_position        65188 non-null  int32
 21  Amino_acids             65188 non-null  int32
 22  Codons                  65188 non-null  int32
 23  STRAND                  65188 non-null  float64
 24  LoFtool                 65188 non-null  float64
 25  CADD_PHRED              65188 non-null  float64
 26  Allele_Freq_Mean        65188 non-null  float64
 27  Pathogenicity_Score_Mean  65188 non-null  float64
dtypes: float64(11), int32(17)
memory usage: 9.7 MB
```

Then we applied standard scaler for all our data points as following based on iteration for data types

```
scaler = StandardScaler()

# Iterate over columns in your DataFrame
for column in df_updated.columns:
    # Check if column data type is int32 or float64
    if df_updated[column].dtype == 'int32' or df_updated[column].dtype == 'float64':
        # Fit and transform the data using StandardScaler
        df_updated[column] = scaler.fit_transform(df_updated[[column]])
```

We specified the variables to be X and y and split it into train and test and checked the shape

Define model variables and spliting into test and train

In [49]:
```
X=df_updated.drop(['CLASS'] , axis=1)
y=df_updated['CLASS']
```

In [50]:
```
#spliting model variables
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=42)
```

In [51]:
```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(45631, 27)
(19557, 27)
(45631,)
(19557,)
```

As the target data for classification project have to be discrete values so we applied discretization to ensure smooth data training

As the target data for classification project have to be discrete values so we applied discretization

[77]:
```
num_bins = 1
bin_edges = np.linspace(np.min(y_train), np.max(y_train), num_bins + 1)

# Discretize the target variable into bins
y_train_discrete = np.digitize(y_train, bin_edges)

# Print the unique values in the discretized target variable
print(np.unique(y_train_discrete))
y_test_discrete = np.digitize(y_test, bin_edges)

# Print the unique values in the discretized test target variable
print(np.unique(y_test_discrete))
```

Then we imported the training models as following

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from imblearn.over_sampling import SMOTE
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, AdaBoostClassifier
```

We applied smote method to ensure that the categories in our data set are balanced as class o in target have higher number of values

```
smote_factor=SMOTE(random_state=11)
x_smote, y_smote = smote_factor.fit_resample(X_train, y_train_discrete)
print("Shape before the Oversampling : ",X_train.shape)
print("Shape after the Oversampling : ",x_smote.shape)
x_smote_test, y_smote_test = smote_factor.fit_resample(X_test,y_test_discrete)
print("Shape before the Oversampling : ",X_test.shape)
print("Shape after the Oversampling : ",x_smote_test.shape)
```

```
Shape before the Oversampling :  (45631, 27)
Shape after the Oversampling :  (68278, 27)
Shape before the Oversampling :  (19557, 27)
Shape after the Oversampling :  (29230, 27)
```

Now we will started models training and the evolution criteria based on accuracy and classification report values and according to it

The Random Forest model shows the highest accuracy at 0.80, with a macro average f1-score also at 0.80. It exhibits strong performance in both classes, with a precision of 0.77 and 0.84 for classes 1 and 2, respectively. Additionally, it maintains a good balance between recall and precision, evidenced by high f1-scores of 0.81 for class 1 and 0.79 for class 2. This indicates that the model is proficient in both correctly identifying true positives and minimizing false positives across both classes, leading to an overall robust performance.

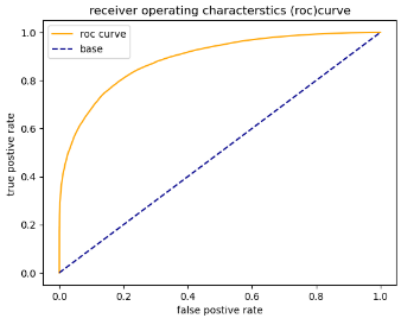
The Gradient Boosting model follows closely with an accuracy of 0.79 and a macro average f1-score of 0.79. It demonstrates high precision (0.80 for class 1 and 0.78 for class 2) and recall (0.78 for class 1 and 0.80 for class 2), resulting in balanced f1-scores of 0.79 for both classes. Gradient Boosting is effective in handling both minority and majority classes well, ensuring that the model is not biased towards any particular class while maintaining a high level of predictive performance.

Model	Class	Precision	Recall	F1-Score	Support	Accuracy	Macro Avg Precision	Macro Avg Recall	Macro Avg F1-Score	Support	Accuracy	Macro Avg Precision	Macro Avg Recall	Macro Avg F1-Score	Weighted Avg Precision	Weighted Avg Recall	Weighted Avg F1-Score
Logistic Regression	1	0.59	0.44	0.51	14615	0.57	0.57	0.57	0.51	14615	0.57	0.57	0.57	0.56	0.57	0.57	0.56
	2	0.55	0.69	0.61	14615				0.61	14615							
Support Vector Machine	1	0.71	0.69	0.70	14615	0.70	0.70	0.70	0.70	14615	0.70	0.70	0.70	0.70	0.70	0.70	0.70
	2	0.70	0.72	0.71	14615				0.71	14615							
Random Forest	1	0.77	0.86	0.81	14615	0.80	0.81	0.80	0.81	14615	0.80	0.81	0.80	0.80	0.81	0.80	0.80
	2	0.84	0.74	0.79	14615				0.79	14615							
KNN	1	0.61	0.63	0.62	14615	0.61	0.61	0.61	0.62	14615	0.61	0.61	0.61	0.61	0.61	0.61	0.61
	2	0.62	0.59	0.60	14615				0.60	14615							
Decision Tree	1	0.72	0.77	0.74	14615	0.74	0.74	0.74	0.74	14615	0.74	0.74	0.74	0.74	0.74	0.74	0.74
	2	0.75	0.71	0.73	14615				0.73	14615							
GradientBoosting	1	0.80	0.78	0.79	14615	0.79	0.79	0.79	0.79	14615	0.79	0.79	0.79	0.79	0.79	0.79	0.79
	2	0.78	0.80	0.79	14615				0.79	14615							
AdaBoost	1	0.76	0.73	0.74	14615	0.75	0.75	0.75	0.74	14615	0.75	0.75	0.75	0.75	0.75	0.75	0.75
	2	0.74	0.77	0.76	14615				0.76	14615							

```
Classification Report of performance for Logistic Regression
--------------------------------------------------------
              precision    recall  f1-score   support

           1       0.59      0.44      0.51     14615
           2       0.55      0.69      0.61     14615

    accuracy                           0.57     29230
   macro avg       0.57      0.57      0.56     29230
weighted avg       0.57      0.57      0.56     29230

--------------------------------------------------------
Classification Report of performance for Support Vector Machine
--------------------------------------------------------
              precision    recall  f1-score   support

           1       0.71      0.69      0.70     14615
           2       0.70      0.72      0.71     14615

    accuracy                           0.70     29230
   macro avg       0.70      0.70      0.70     29230
weighted avg       0.70      0.70      0.70     29230

--------------------------------------------------------
Classification Report of performance for Random Forest
--------------------------------------------------------
              precision    recall  f1-score   support

           1       0.77      0.86      0.81     14615
           2       0.84      0.74      0.79     14615

    accuracy                           0.80     29230
   macro avg       0.81      0.80      0.80     29230
weighted avg       0.81      0.80      0.80     29230

--------------------------------------------------------
Classification Report of performance for KNN
--------------------------------------------------------
              precision    recall  f1-score   support

           1       0.61      0.63      0.62     14615
           2       0.62      0.59      0.60     14615

    accuracy                           0.61     29230
   macro avg       0.61      0.61      0.61     29230
weighted avg       0.61      0.61      0.61     29230

--------------------------------------------------------
Classification Report of performance for Decision Tree
--------------------------------------------------------
              precision    recall  f1-score   support

           1       0.72      0.77      0.74     14615
           2       0.75      0.71      0.73     14615

    accuracy                           0.74     29230
   macro avg       0.74      0.74      0.74     29230
weighted avg       0.74      0.74      0.74     29230

--------------------------------------------------------
Classification Report of performance for GradientBoosting
--------------------------------------------------------
              precision    recall  f1-score   support

           1       0.80      0.78      0.79     14615
           2       0.78      0.80      0.79     14615

    accuracy                           0.79     29230
   macro avg       0.79      0.79      0.79     29230
weighted avg       0.79      0.79      0.79     29230

--------------------------------------------------------
Classification Report of performance for AdaBoost
--------------------------------------------------------
              precision    recall  f1-score   support

           1       0.76      0.73      0.74     14615
           2       0.74      0.77      0.76     14615

    accuracy                           0.75     29230
   macro avg       0.75      0.75      0.75     29230
weighted avg       0.75      0.75      0.75     29230
```

We start working on GradientBoostingClassifier and random forest classifier to choose the most suitable one so we did AUC /ROC TEST from to the AUC score we found that they have the same auc_score so we choose the random forest it have higher accuracy score ;80% and better classification report values

```
]: y_probs_1 = clf.predict_proba(x_smote_test)
   y_probs_1

]: array([[0.58862   , 0.41138   ],
          [0.78670661, 0.21329339],
          [0.30254206, 0.69745794],
          ...,
          [0.27253989, 0.72746011],
          [0.22982773, 0.77017227],
          [0.08053208, 0.91946792]])

]: y_probs_postive_1 = y_probs[ :, 1]
   y_probs_postive_1[:10]

]: array([0.45, 0.1 , 0.47, 0.17, 0.76, 0.66, 0.68, 0.18, 0.39, 0.71])

]: fpr, tpr, thresholds = roc_curve(y_smote_test, y_probs_postive_1, pos_label=2)

]:

]:     plt.plot(fpr,tpr, color='orange',label='roc curve')
       #plot line with no predictive power (baseline)
       plt.plot([0,1],[0,1],color='darkblue',linestyle='--',label='base')
       #customization of the plot
       plt.xlabel('false postive rate')
       plt.ylabel('true postive rate')
       plt.title('receiver operating characterstics (roc)curve')
       plt.legend()
       plt.show()
   plot_roc_AUC(fpr,tpr)
```




```
]: from sklearn.metrics import roc_auc_score
   roc_auc_score( y_smote_test,y_probs_postive)

]: 0.8899886785107882
```

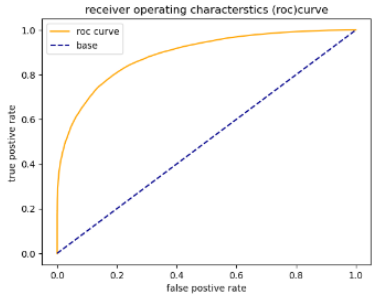
```
from sklearn.metrics import roc_curve
y_probs = rf.predict_proba(x_smote_test)
y_probs
```

```
array([[0.55, 0.45],
       [0.9 , 0.1 ],
       [0.53, 0.47],
       ...,
       [0.  , 1.  ],
       [0.35, 0.65],
       [0.09, 0.91]])
```

```
y_probs_postive = y_probs[ :, 1]
y_probs_postive[:10]
```

```
array([0.45, 0.1 , 0.47, 0.17, 0.76, 0.66, 0.68, 0.18, 0.39, 0.71])
```

```
    plt.plot(fpr,tpr, color='orange',label='roc curve')
    #plot line with no predictive power (baseline )
    plt.plot([0,1],[0,1],color='darkblue',linestyle='--',label='base')
    #customization of the plot
    plt.xlabel('false postive rate')
    plt.ylabel('true postive rate ')
    plt.title('receiver operating characterstics (roc)curve')
    plt.legend()
    plt.show()
plot_roc_AUC(fpr,tpr)
```

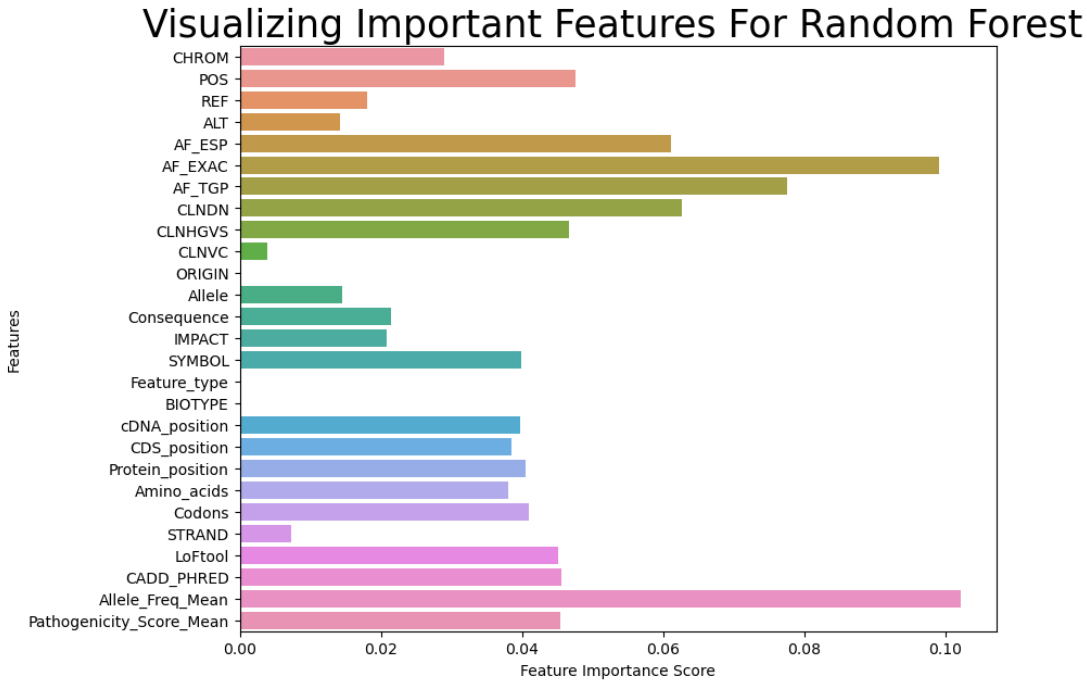



```
from sklearn.metrics import roc_auc_score
roc_auc_score( y_smote_test,y_probs_postive)
```

```
0.8899886785107882
```

.features importance discovery

We applied features important discovery to identify which feature is the most important for our model in prediction the target variable we found that our new feature which is mean_allele_frequency is the heights features



Then we set a subset that contain only the heights 20 ranking features I order to increase model accuracy

```
data = pd.DataFrame(df_updated)
drop_cols = ['ORIGIN', 'Feature_type', 'BIOTYPE', 'CLNVC', 'STRAND', 'Allele', 'IMPACT']
supsted_data= data.drop(columns=drop_cols)

# Print the new DataFrame
supsted_data
#training moodle based on new df
X=supsted_data.drop(['CLASS'] , axis=1)
y=supsted_data['CLASS']
#spliting model variables
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=42)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
print(np.unique(y_train))
num_bins = 1
bin_edges = np.linspace(np.min(y_train), np.max(y_train), num_bins + 1)

# Discretize the target variable into bins
y_train_discrete = np.digitize(y_train, bin_edges)

# Print the unique values in the discretized target variable
print(np.unique(y_train_discrete))
y_test_discrete = np.digitize(y_test, bin_edges)

# Print the unique values in the discretized test target variable
print(np.unique(y_test_discrete))
smote_factor=SMOTE(random_state=11)
x_smote, y_smote = smote_factor.fit_resample(X_train, y_train_discrete)
print("Shape before the Oversampling : ",X_train.shape)
print("Shape after the Oversampling : ",x_smote.shape)
x_smote_test, y_smote_test = smote_factor.fit_resample(X_test,y_test_discrete)
print("Shape before the Oversampling : ",X_test.shape)
print("Shape after the Oversampling : ",x_smote_test.shape)
```

It give us model accuracy =80,0% but the original accuracy for model wa s 80,3% so we choose to complete with the original data set
Note ;working with subset will not effect the process badly as the accurac y difference is small

```
rf = RandomForestClassifier()
rf.fit(x_smote,y_smote)
rf.score(x_smote_test, y_smote_test )
```

```
0.8006158056790968
```

```
data = pd.DataFrame(df_updated)
drop_cols = ['ORIGIN', 'Feature_type', 'BIOTYPE', 'CLNVC', 'STRAND', 'Allele', 'IMPACT']
supsted_data= data.drop(columns=drop_cols)

# Print the new DataFrame
supsted_data
#training moodle based on new df
X=supsted_data.drop(['CLASS'] , axis=1)
y=supsted_data['CLASS']
#spliting model variables
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=42)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
print(np.unique(y_train))
num_bins = 1
bin_edges = np.linspace(np.min(y_train), np.max(y_train), num_bins + 1)

# Discretize the target variable into bins
y_train_discrete = np.digitize(y_train, bin_edges)

# Print the unique values in the discretized target variable
print(np.unique(y_train_discrete))
y_test_discrete = np.digitize(y_test, bin_edges)

# Print the unique values in the discretized test target variable
print(np.unique(y_test_discrete))
smote_factor=SMOTE(random_state=11)
x_smote, y_smote = smote_factor.fit_resample(X_train, y_train_discrete)
print("Shape before the Oversampling : ",X_train.shape)
print("Shape after the Oversampling : ",x_smote.shape)
x_smote_test, y_smote_test = smote_factor.fit_resample(X_test,y_test_discrete)
print("Shape before the Oversampling : ",X_test.shape)
print("Shape after the Oversampling : ",x_smote_test.shape)
```

```
(45631, 20)
```

Hyper parameters tunning

We used 2 methods to search for best parameters tunning .interestingly ,we found that both give us parameter tunning that decreased the accuracy of our model so we confirmed that the best parameters in our case is the default parameters

First we set our param _grid the for the function RandomizedSearchCV it give ide the following parameters

```
random_search = RandomizedSearchCV(RandomForestClassifier(),
                                   param_grid)
random_search.fit(x_smote,y_smote)
print(random_search.best_estimator_)
```

```
RandomForestClassifier(max_depth=9, max_features=None, max_leaf_nodes=9,
                       n_estimators=50)
```

```
model_random = RandomForestClassifier(max_depth=6,
                                      max_features=None,
                                      max_leaf_nodes=9,
                                      n_estimators=25)
model_random.fit(x_smote,y_smote)
y_pred = model_random .predict(x_smote_test)
print(classification_report(y_pred, y_smote_test))
print(f"Accuracy Score for randomforest model after paramters tunning is : ",model_random.score(x_smote_test, y_smote_test)*
```

```
              precision    recall  f1-score   support

           1       0.65      0.66      0.66     14411
           2       0.66      0.66      0.66     14819

    accuracy                           0.66     29230
   macro avg       0.66      0.66      0.66     29230
weighted avg       0.66      0.66      0.66     29230

Accuracy Score for randomforest model after paramters tunning is :  65.7475196715703 %
```

Second using gridsearchcv model

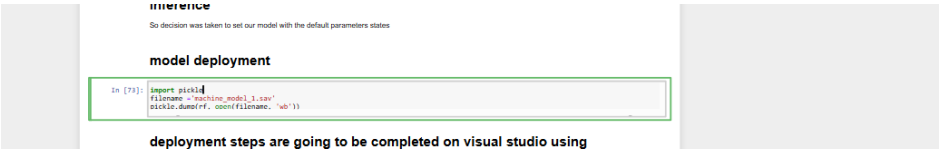
```
9]: ##Create a GridSearchCV object with parallel processing
grid_search = GridSearchCV(RandomForestClassifier(), param_grid, refit=True, verbose=2, n_jobs=4)

# Perform the grid search
grid_search.fit(x_smote,y_smote)

# Print the best parameters and the corresponding score
print("Best Parameters:", grid_search.best_params_)
print("Best Score:", grid_search.best_score_)
```

```
Fitting 5 folds for each of 108 candidates, totalling 540 fits
Best Parameters: {'max_depth': 6, 'max_features': None, 'max_leaf_nodes': 9, 'n_estimators': 25}
Best Score: 0.6526992597521525
```

So decision was taken to set our model with the default parameters states

Model deployment on web interface

First we downloaded our model in form of .sav file using pickle library



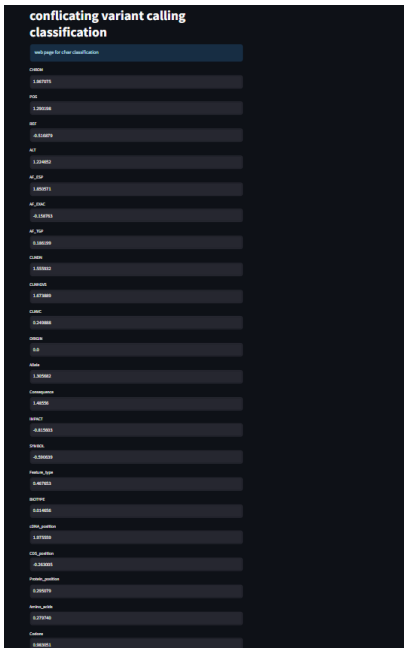
The in visual studio we use stream lit to create gui that takes info according to the features and display if that a case of conflicting variant classification or not

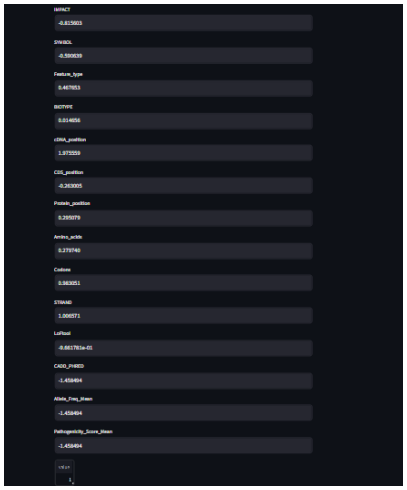
We installed stream lit created the columns inform of data frame and induced the model using the previously downloaded .sav file

```
Amino_acids=st.text_input('Amino_acids')
Codons = st.text_input('Codons')
dna_strand = st.text_input('STRAND')
Loss_of_Function_tolerance= st.text_input('LoFtool')
CADD_PHRED= st.text_input('CADD_PHRED')
Allele_Freq_Mean= st.text_input('Allele_Freq_Mean')
Pathogenicity_Score_Mean= st.text_input('Pathogenicity_Score_Mean')
#creating dataform
df=pd.DataFrame({'CHROM':[chromosome],'POS':[variant_postion_onchromosome],
'REF':[refernce_form], 'ALT':[alternative_form],
'AF_ESP':[Allele_frequencies_ESP],'AF_EXAC':[Allele_frequencies_EXAC],'AF_TGP':[Allele_frequencies_TGP],
'CLNDN':[Allele_frequencies_genome_pro],'CLNHGVS':[top_level_expression],'CLNVC':[Variant_Type],'ORIGIN':[Allele_origin],
'Allele':[Allele],'Consequence':[Consequence],
'IMPACT':[IMPACT] , 'SYMBOL':[gene_SYMBOL],'Feature_type':[Feature_type],'BIOTYPE':[BIOTYPE] , 'cDNA_position':[cDNA_position]
,'CDS_position':[CDS_position],
'Protein_position':[Protein_position] , 'Amino_acids':[Amino_acids] ,'Codons':[Codons] ,'STRAND':[dna_strand] , 'LoFtool':[Loss_of_Function_tolerance] ,
'CADD_PHRED':[CADD_PHRED], 'Allele_Freq_Mean':[Allele_Freq_Mean] , 'Pathogenicity_Score_Mean':[Pathogenicity_Score_Mean]},index=[0])
#load our previously implementated model
model=pickle.load(open(r"C:\Users\Dell\Downloads\machine_model_1.sav",'rb'))
Confirmation=st.sidebar.button('confirm')
if Confirmation:
    result=model.predict(df)
```

```
import streamlit as st
import pickle
import pandas as pd
#setting the interface
st.title('conflicating variant calling classification ')
st.info('web page for cfvar classification')
st.sidebar.header('Features')
#assigned faeatures columns to our streamlit app
chromosome=st.text_input('CHROM')
variant_postion_onchromosome=st.text_input('POS')
refernce_form=st.text_input('REF')
alternative_form= st.text_input('ALT')
Allele_frequencies_ESP = st.text_input('AF_ESP')
Allele_frequencies_EXAC=st.text_input('AF_EXAC')
Allele_frequencies_TGP=st.text_input('AF_TGP')
Allele_frequencies_genome_pro=st.text_input('CLNDN')
top_level_expression = st.text_input('CLNHGVS')
Variant_Type=st.text_input('CLNVC')
Allele_origin=st.text_input('ORIGIN')
Allele =st.text_input('Allele')
Consequence=st.text_input('Consequence')
IMPACT=st.text_input('IMPACT')
gene_SYMBOL=st.text_input('SYMBOL')
Feature_type=st.text_input('Feature_type')
```

The resulted interface





Conclusion

In our classification approach we started from data cleaning (handling missing and remove outliers),EDA which help us to uncover patterns , scaling and encoding categorical values ,handling unbalanced data , discretization of target data and ended with random forest model with accuracy 80% we surprisingly found that the most important feature is a composite feature ;Allele_Freq_Mean so further addition of composite features may help us to reduce data features and increase model performance .after searching about TTNgene which shows the highest frequency in both target categories . We found that The TTN gene is one of the largest genes in the human genome, encoding the titin protein, which is crucial for muscle function. Due to its large size, it contains numerous exons and introns, making it susceptible to a high degree of variation it have high allelic variability .thus, validation of model efficiency could be induced using allelic condition of this gene