Faculty of Engineering
Cairo University

Machine learning Project

Binary Prediction of Poisonous Mushrooms

Brief Problem Description

Building machine learning model to predict whether a mushroom is edible or poisonous based on its physical characteristics.

Problem Motivation

Accurately determining whether a mushroom is edible or poisonous is critical for both foragers and food safety professionals. Misidentification can lead to severe health consequences, including poisoning and even death. Traditional identification methods rely on expert knowledge, which is not always accessible to the general public.

Dataset

https://www.kaggle.com/competitions/playground-series-s4e8/data

Evaluation Metrics

Accuracy, Precision, Recall, F1 Score

Matthews correlation coefficient (MCC): is a metric used to evaluate the quality of binary classifications, especially when the classes are imbalanced.

$$\mathrm{MCC} = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

Project Pipeline

Data loading

Train.csv file shape: (3116945, 22)

Test.csv file shape: (2077964, 21)

Data Splitting

Split train to train and validation (80 , 20) with stratification on the target feature (“class”)

Data cleaning

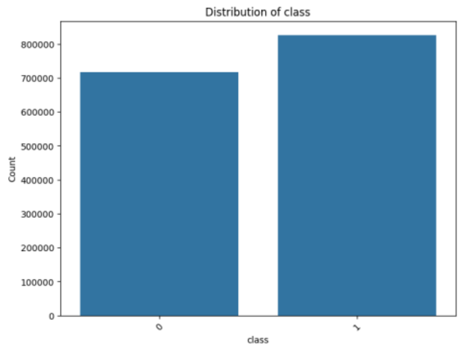
1. Dropping features with null values > 40%   -> 6 features been dropped
2. The data had some noise in the categorical features values -> we took the noise values and combine them as ‘other’ values and then to NaN -> so we can fill them with meaningful values
3. Taking version of train dataset and dropping rows with null so we can train our target encoder -> we used target encoder because the values of the categorical features was nominal and we have more than 7 unique values on average for each categorical feature so we cannot use either OHE or Label-Encoder

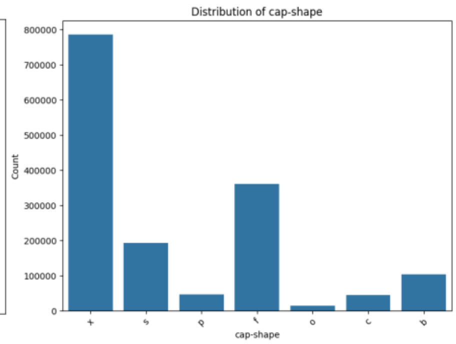
4. Transform the categorical features of train , test and val with the target encoder
5. Impute the missing values (and noise) with Simple imputer using mean (failed to use KNN imputer took a lot of time and Ram crash)
6. Now we have cleaned data

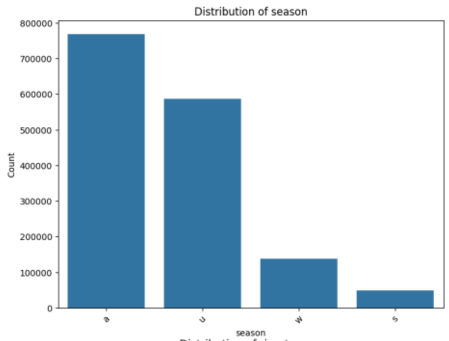
Data visualization

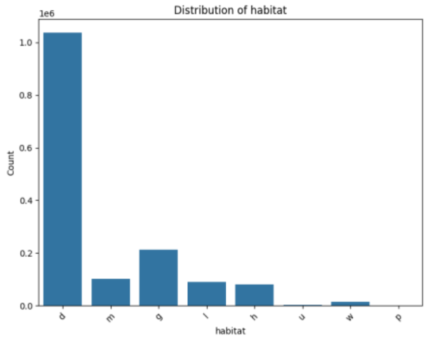
1. Univariant Analysis

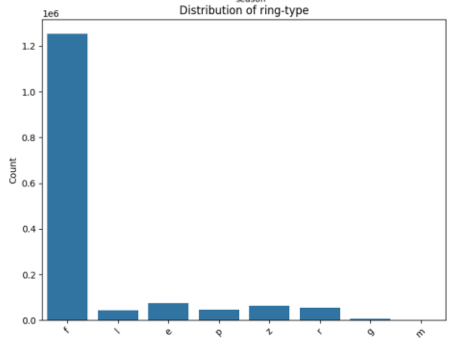
    - Categorical features

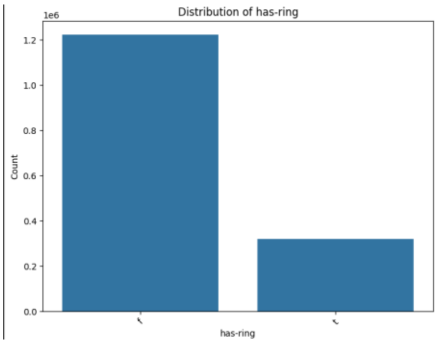


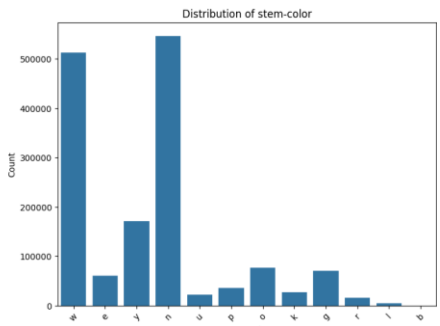


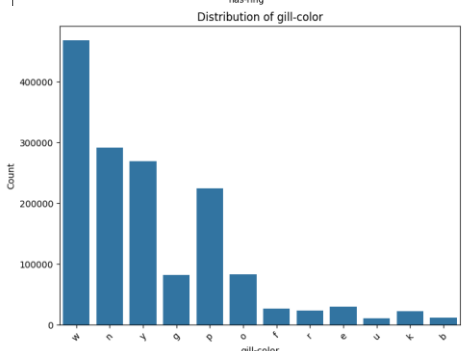


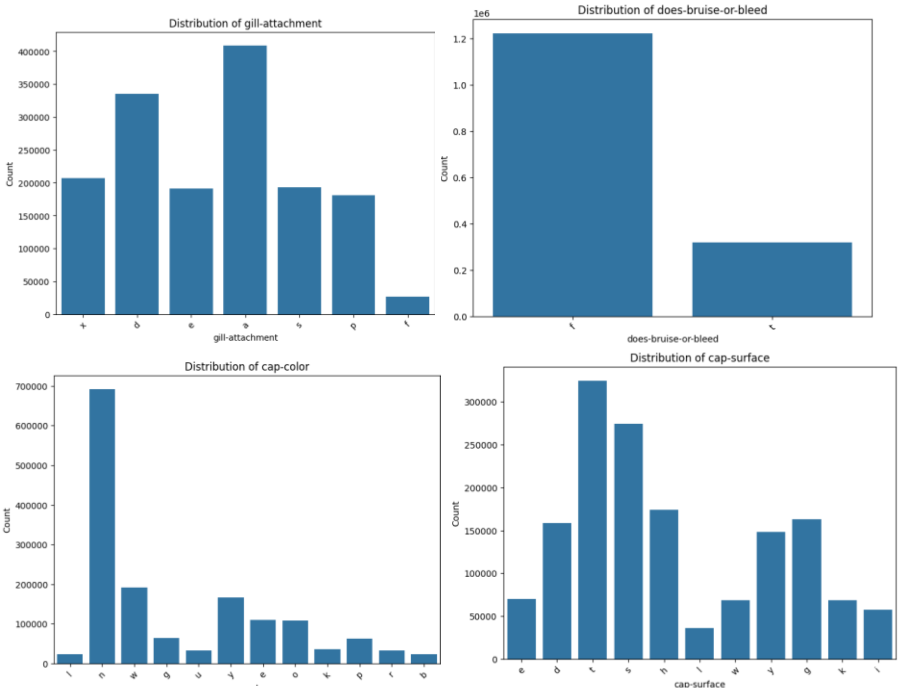




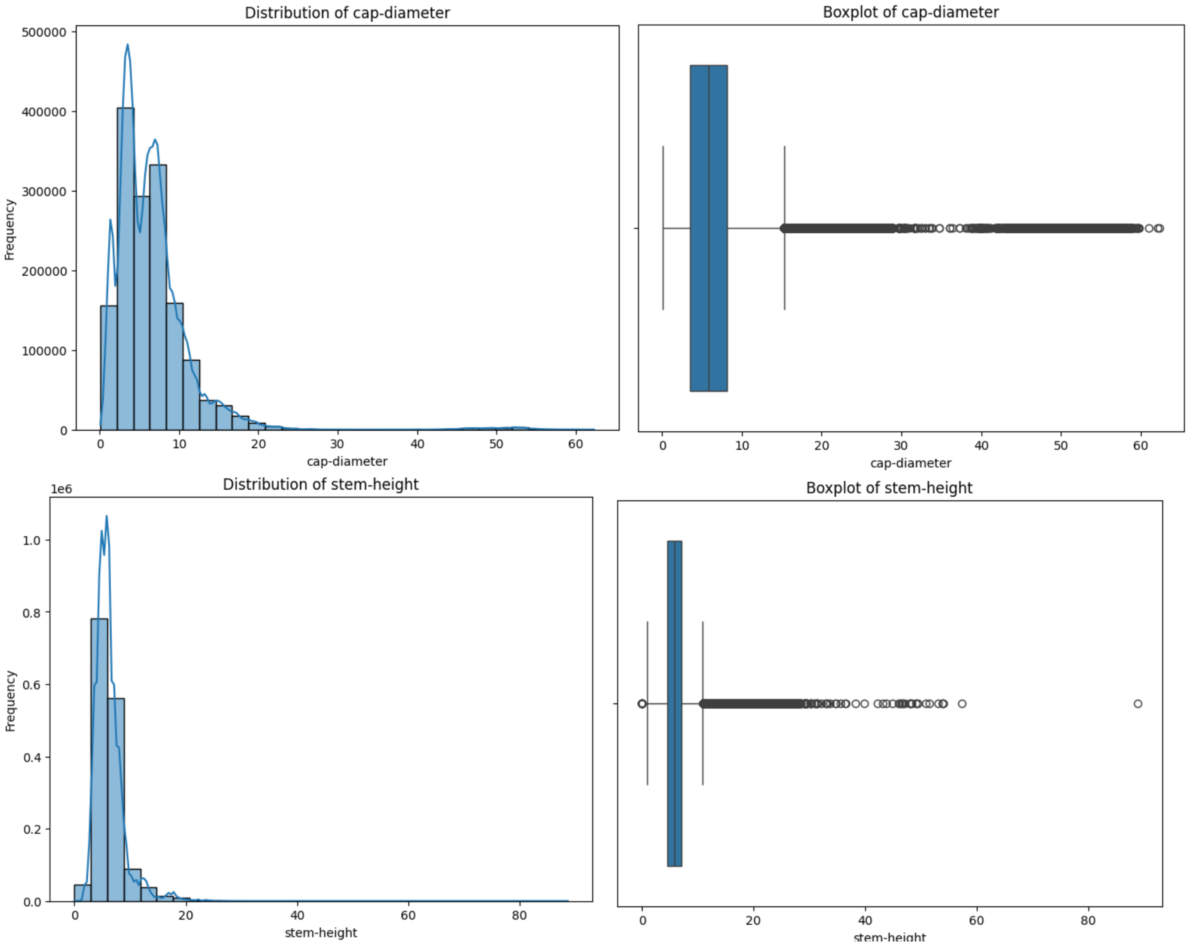


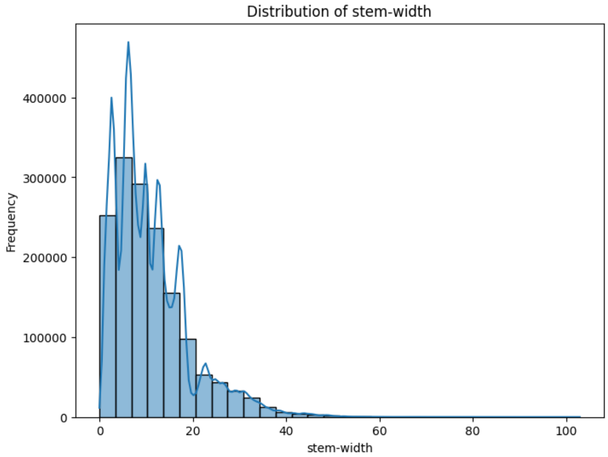


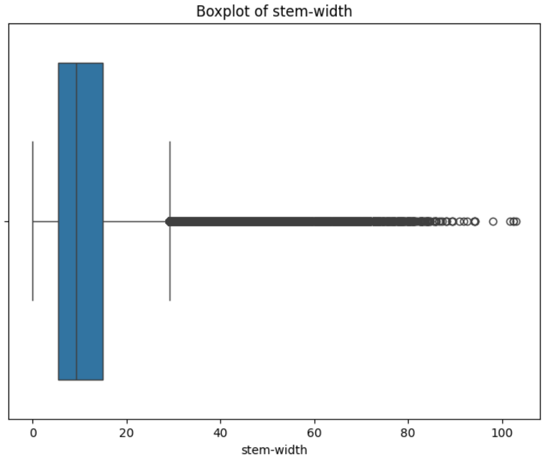




- Numerical features

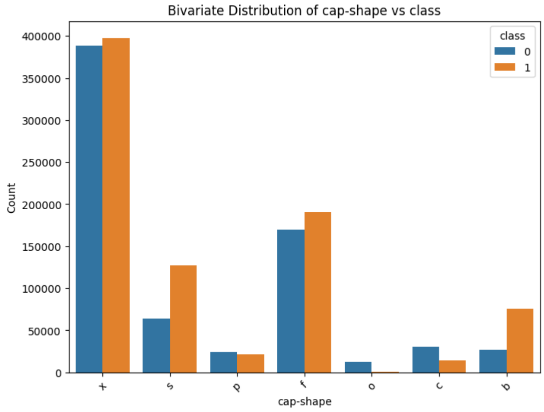




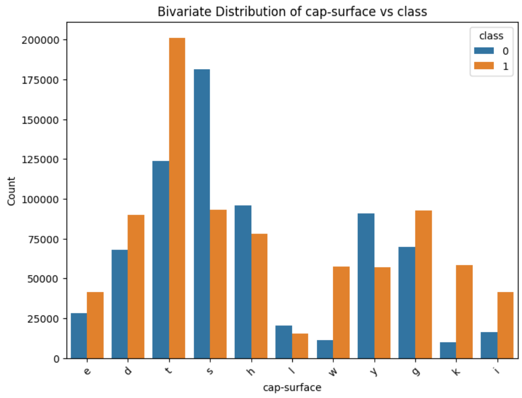


2. Bivariant Analysis

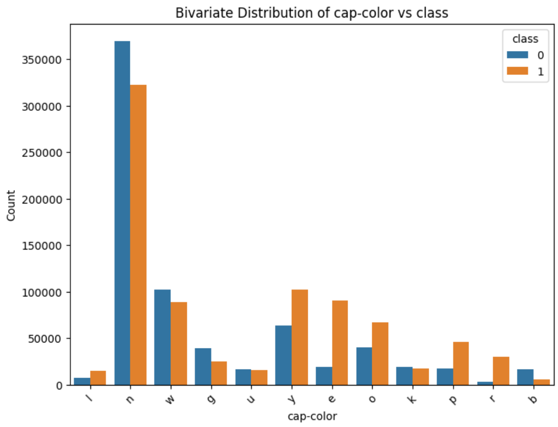
- Categorical features with the target & Chi-Squared test



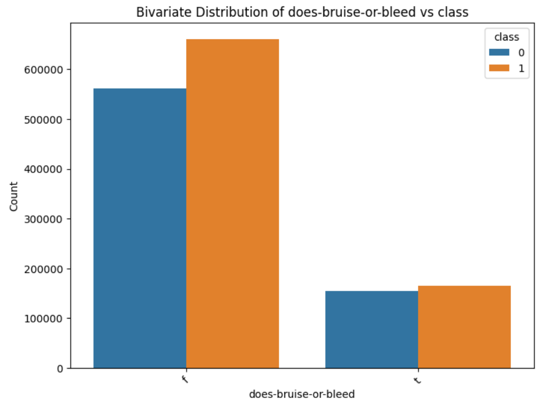

```
Chi-squared test for cap-shape vs class:
Chi2: 53708.778989029895, p-value: 0.0
```



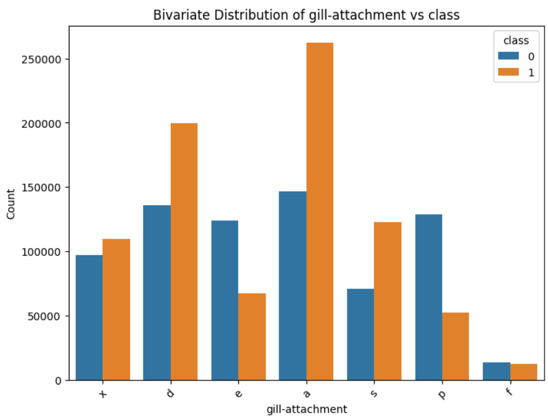

```
Chi-squared test for cap-surface vs class:
Chi2: 134646.02529185047, p-value: 0.0
```




```
Chi-squared test for cap-color vs class:
Chi2: 103406.20421175637, p-value: 0.0
```

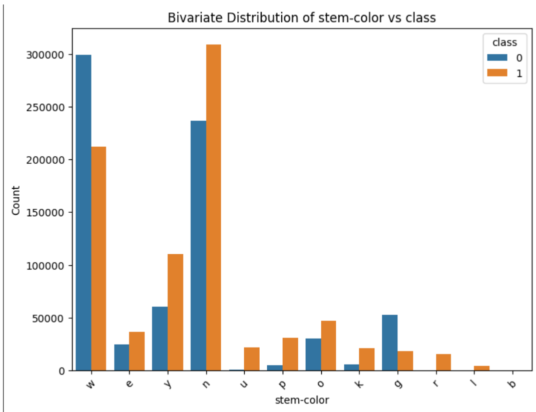



```
Chi-squared test for does-bruise-or-bleed vs class:
Chi2: 530.9023652932098, p-value: 1.7973583732328518e-117
```

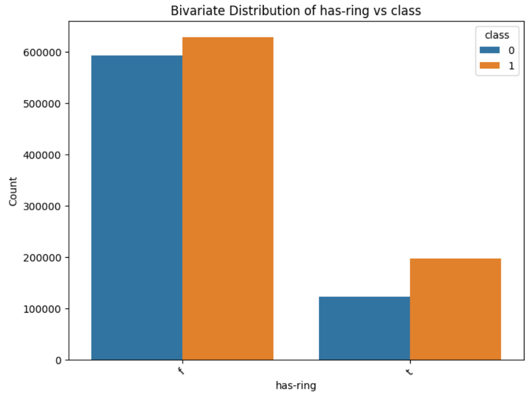



```
Chi-squared test for gill-attachment vs class:
Chi2: 101288.28880970505, p-value: 0.0
```

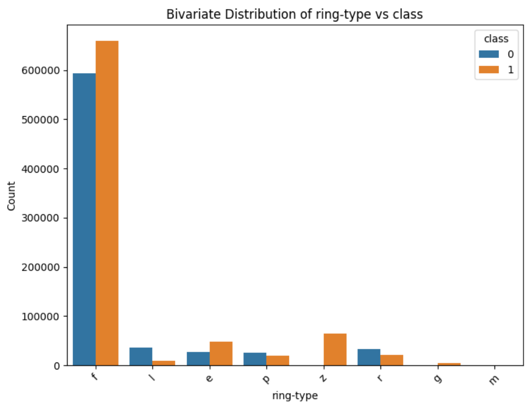

```
Chi-squared test for gill-color vs class:
Chi2: 57583.14891315284, p-value: 0.0
```



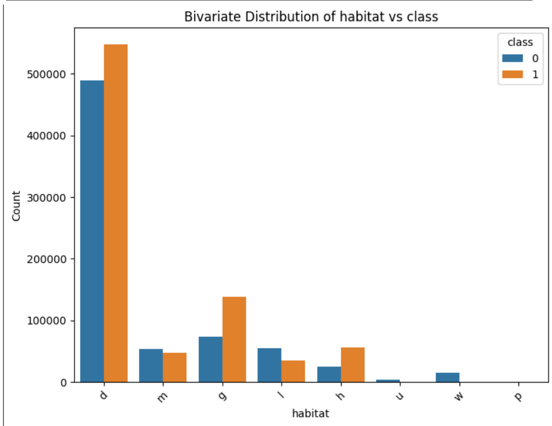

```
Chi-squared test for stem-color vs class:
Chi2: 122847.51974723193, p-value: 0.0
```



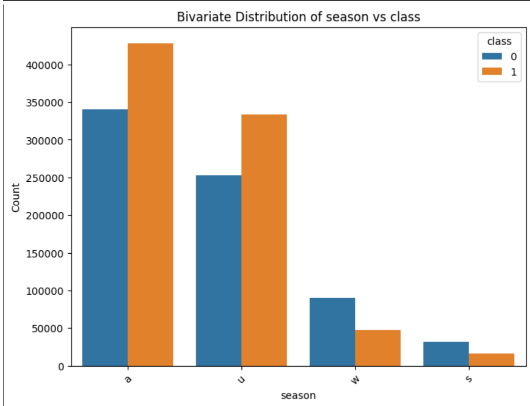

```
Chi-squared test for has-ring vs class:
Chi2: 10323.127226260543, p-value: 0.0
```




```
Chi-squared test for ring-type vs class:
Chi2: 90238.65840724252, p-value: 0.0
```

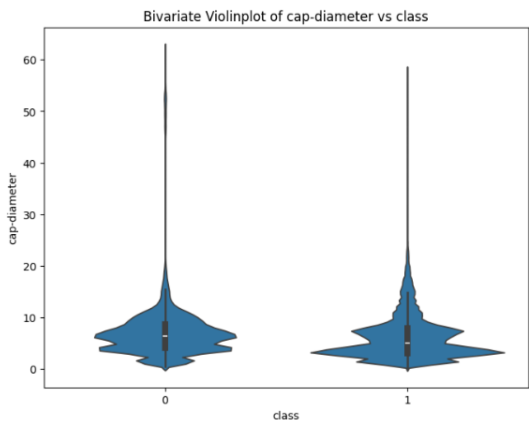



```
Chi-squared test for habitat vs class:
Chi2: 50474.75065423056, p-value: 0.0
```

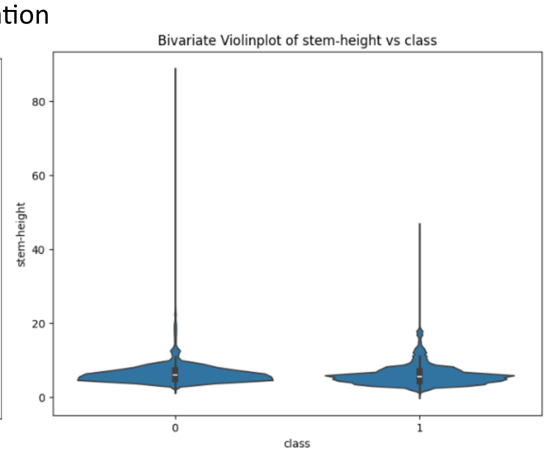



```
Chi-squared test for season vs class:
Chi2: 31984.322245109684, p-value: 0.0
```

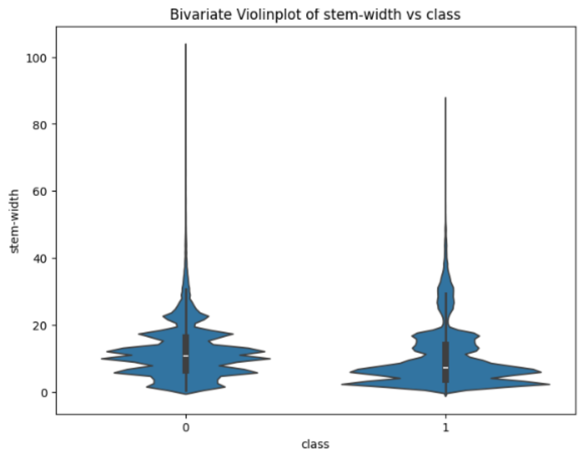
- Numerical features with the target & Correlation




```
Correlation between cap-diameter and class: -0.11529953305141552
```

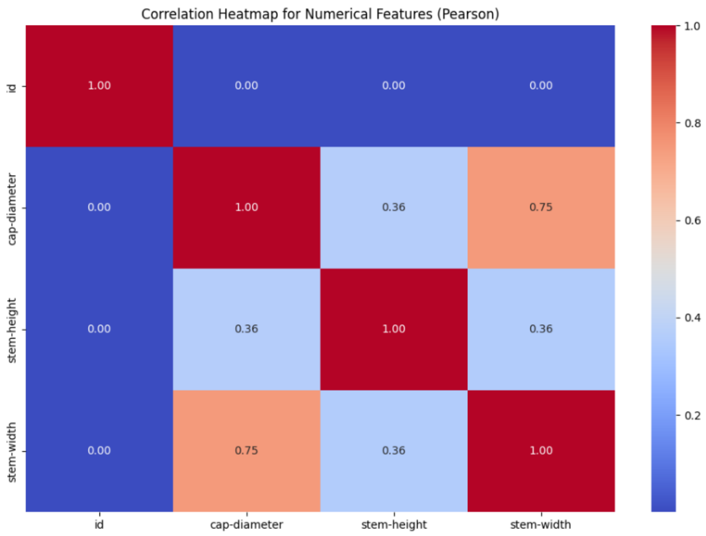


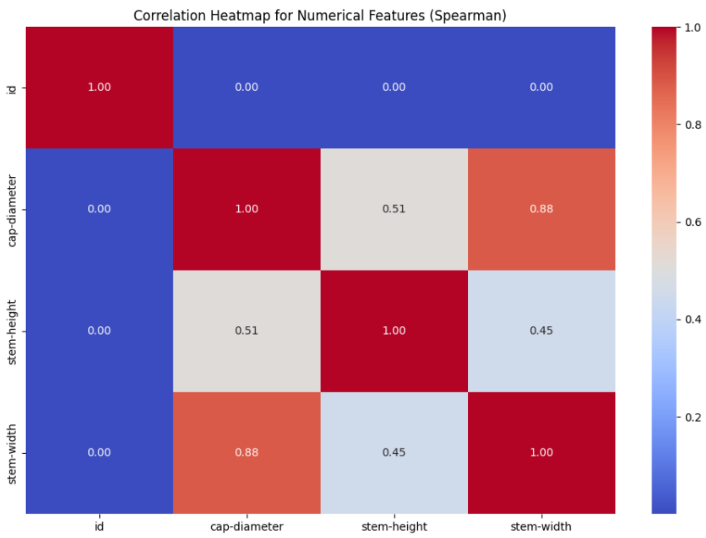

```
Correlation between stem-height and class: -0.08317931898170804
```

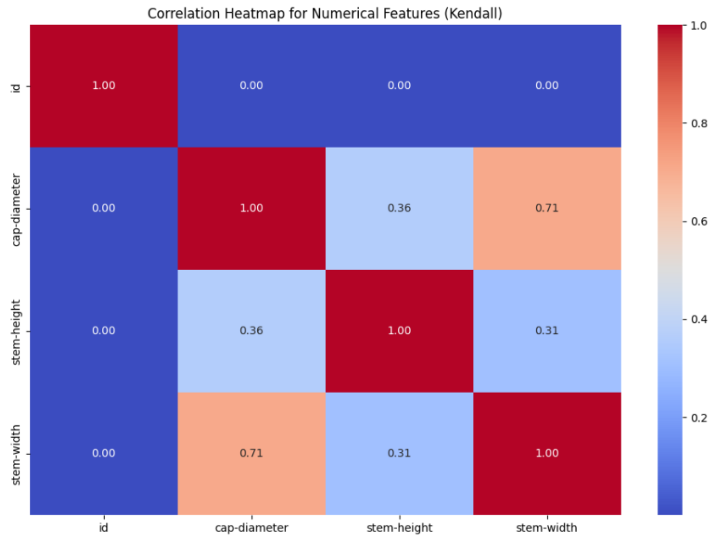



```
Correlation between stem-width and class: -0.11107333862332487
```

- Multicollinearity







Feature selection

Form the visuals we have only one problem

stem-width & cap-diameter is high correlated -> will drop one and keep the highest correlated with the target

```
Pearson correlation between 'stem-width' and 'class': -0.16934260357984848
Spearman correlation between 'stem-width' and 'class': -0.22160116090702292
Kendall correlation between 'stem-width' and 'class': -0.18099223947462714

Pearson correlation between 'cap-diameter' and 'class': -0.1622962730474319
Spearman correlation between 'cap-diameter' and 'class': -0.19821371816220826
Kendall correlation between 'cap-diameter' and 'class': -0.16192168554973005
```

We dropped cap-diameter

Outliers removing

We remove the outliers using IQR method

Modeling

logReg

We apply Standard Scaler first on the data

Parm : (random_state=42, max_iter=1000 ,n_jobs=-1 ,C=1,class_weight=None)

```
=== Logistic Regression ===
---- Train Metrics ----
[[754808 316765]
 [319804 943523]]
              precision    recall  f1-score   support

           0       0.70      0.70      0.70   1071573
           1       0.75      0.75      0.75   1263327

    accuracy                           0.73   2334900
   macro avg       0.73      0.73      0.73   2334900
weighted avg       0.73      0.73      0.73   2334900

Train Accuracy: 0.7274
Train MCC: 0.4512
---- Validation Metrics ----
[[202443  79867]
 [ 98546 242533]]
              precision    recall  f1-score   support

           0       0.67      0.72      0.69    282310
           1       0.75      0.71      0.73    341079

    accuracy                           0.71    623389
   macro avg       0.71      0.71      0.71    623389
weighted avg       0.72      0.71      0.71    623389

Validation Accuracy: 0.7138
Validation MCC: 0.4265
```

Note: high bias & small (Acceptable)variance

Parm: (random_state=42, max_iter=1000 ,n_jobs=-1,solver='lbfgs',C=10,class_weight=None)

```
=== Logistic Regression (C=10 , new solver algorithm) ===
---- Train Metrics ----
[[754808 316765]
 [319804 943523]]
              precision    recall  f1-score   support

           0       0.70      0.70      0.70   1071573
           1       0.75      0.75      0.75   1263327

    accuracy                           0.73   2334900
   macro avg       0.73      0.73      0.73   2334900
weighted avg       0.73      0.73      0.73   2334900

Train Accuracy: 0.7274
Train MCC: 0.4512
---- Validation Metrics ----
[[202442  79868]
 [ 98546 242533]]
              precision    recall  f1-score   support

           0       0.67      0.72      0.69    282310
           1       0.75      0.71      0.73    341079

    accuracy                           0.71    623389
   macro avg       0.71      0.71      0.71    623389
weighted avg       0.72      0.71      0.71    623389

Validation Accuracy: 0.7138
Validation MCC: 0.4265
```

Note: still high bias & small (Acceptable) variance

Action: try polynomial features to increase the model complexity

```
poly_logreg_pipeline = Pipeline([
    ("poly", PolynomialFeatures(degree=2, interaction_only=False, include_bias=False)),
    ("scaler", StandardScaler()),
    ("logreg", LogisticRegression(max_iter=1000,  random_state=42))
])
```

```
=== Logistic Regression with Polynomial Features ===
---- Train Metrics ----
[[ 846059  225514]
 [ 202853 1060474]]
              precision    recall  f1-score   support

           0       0.81      0.79      0.80   1071573
           1       0.82      0.84      0.83   1263327

    accuracy                           0.82   2334900
   macro avg       0.82      0.81      0.81   2334900
weighted avg       0.82      0.82      0.82   2334900

Train Accuracy: 0.8165
Train MCC: 0.6301
---- Validation Metrics ----
[[214695  67615]
 [ 55411 285668]]
              precision    recall  f1-score   support

           0       0.79      0.76      0.78    282310
           1       0.81      0.84      0.82    341079

    accuracy                           0.80    623389
   macro avg       0.80      0.80      0.80    623389
weighted avg       0.80      0.80      0.80    623389

Validation Accuracy: 0.8026
Validation MCC: 0.6007
```

Note: improved but still high bias

Action: increase the polynomial degree -> Data is huge and takes a lot of time

SVM

Parm: SVC(kernel='linear')

Takes forever

Action: perform on sample of the data

Parm: SVC(kernel='linear') with 10k row only

```
=== SVM (Balanced Sample) ===
---- Train Metrics ----
[[3370 1219]
 [1400 4011]]
              precision    recall  f1-score   support

           0       0.71      0.73      0.72      4589
           1       0.77      0.74      0.75      5411

    accuracy                           0.74     10000
   macro avg       0.74      0.74      0.74     10000
weighted avg       0.74      0.74      0.74     10000

Train Accuracy: 0.7381
Train MCC: 0.4745
---- Validation Metrics ----
[[210982  71328]
 [104306 236773]]
              precision    recall  f1-score   support

           0       0.67      0.75      0.71    282310
           1       0.77      0.69      0.73    341079

    accuracy                           0.72    623389
   macro avg       0.72      0.72      0.72    623389
weighted avg       0.72      0.72      0.72    623389

Validation Accuracy: 0.7183
Validation MCC: 0.4396
```

Note: high Bias , (Acceptable) Variance

Action : more complex kernel

Parm: SVC(kernel='rbf')

```
=== SVM (Balanced Sample (rbf) ) ===
---- Train Metrics ----
[[4256  333]
 [ 279 5132]]
              precision    recall  f1-score   support

           0       0.94      0.93      0.93      4589
           1       0.94      0.95      0.94      5411

    accuracy                           0.94     10000
   macro avg       0.94      0.94      0.94     10000
weighted avg       0.94      0.94      0.94     10000

Train Accuracy: 0.9388
Train MCC: 0.8767
---- Validation Metrics ----
[[250070  32240]
 [ 17811 323268]]
              precision    recall  f1-score   support

           0       0.93      0.89      0.91    282310
           1       0.91      0.95      0.93    341079

    accuracy                           0.92    623389
   macro avg       0.92      0.92      0.92    623389
weighted avg       0.92      0.92      0.92    623389

Validation Accuracy: 0.9197
Validation MCC: 0.8382
```

Note: improvement on bias and (Acceptable) variance

Action: increase the size of the sample 5 times

```
=== SVM (Balanced Sample (rbf & increased samples)) ===
---- Train Metrics ----
[[21712  1235]
 [ 1002 26051]]
              precision    recall  f1-score   support

           0       0.96      0.95      0.95     22947
           1       0.95      0.96      0.96     27053

    accuracy                           0.96     50000
   macro avg       0.96      0.95      0.95     50000
weighted avg       0.96      0.96      0.96     50000

Train Accuracy: 0.9553
Train MCC: 0.9099
---- Validation Metrics ----
[[259352  22958]
 [ 12530 328549]]
              precision    recall  f1-score   support

           0       0.95      0.92      0.94    282310
           1       0.93      0.96      0.95    341079

    accuracy                           0.94    623389
   macro avg       0.94      0.94      0.94    623389
weighted avg       0.94      0.94      0.94    623389

Validation Accuracy: 0.9431
Validation MCC: 0.8853
```

Note: improvement on bias and variance but takes long time

Action : increase the sample size more -> Huge amount of time

Action : try to use tree based model

Decision Tree

Parm: DecisionTreeClassifier(random_state=42)

```
=== Decision Tree ===
---- Train Metrics ----
[[1071450     123]
 [    780 1262547]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00   1071573
           1       1.00      1.00      1.00   1263327

    accuracy                           1.00   2334900
   macro avg       1.00      1.00      1.00   2334900
weighted avg       1.00      1.00      1.00   2334900

Train Accuracy: 0.9996
Train MCC: 0.9992
---- Validation Metrics ----
[[275625   6685]
 [  6571 334508]]
              precision    recall  f1-score   support

           0       0.98      0.98      0.98    282310
           1       0.98      0.98      0.98    341079

    accuracy                           0.98    623389
   macro avg       0.98      0.98      0.98    623389
weighted avg       0.98      0.98      0.98    623389

Validation Accuracy: 0.9787
Validation MCC: 0.9571
```

Note : gives good results -> low bias , high variance (overfitting)

Action: try to reduce overfitting -> (tune max_depth , ccp_alpha)

Parm: DecisionTreeClassifier(random_state=42,max_depth=14)

```
=== Decision Tree ===
---- Train Metrics ----
[[1057976   13597]
 [  20048 1243279]]
              precision    recall  f1-score   support

           0       0.98      0.99      0.98   1071573
           1       0.99      0.98      0.99   1263327

    accuracy                           0.99   2334900
   macro avg       0.99      0.99      0.99   2334900
weighted avg       0.99      0.99      0.99   2334900

Train Accuracy: 0.9856
Train MCC: 0.9710
---- Validation Metrics ----
[[277913   4397]
 [  6176 334903]]
              precision    recall  f1-score   support

           0       0.98      0.98      0.98    282310
           1       0.99      0.98      0.98    341079

    accuracy                           0.98    623389
   macro avg       0.98      0.98      0.98    623389
weighted avg       0.98      0.98      0.98    623389

Validation Accuracy: 0.9830
Validation MCC: 0.9658
```

Note : reduce the overfitting

Parm: DecisionTreeClassifier(random_state=42,ccp_alpha=0.001)

ccp_alpha: Controls complexity through cost-complexity pruning-> Higher values prune more of the tree.

```
=== Decision Tree ===
---- Train Metrics ----
[[1021272   50301]
 [  56021 1207306]]
              precision    recall  f1-score   support

           0       0.95      0.95      0.95   1071573
           1       0.96      0.96      0.96   1263327

    accuracy                           0.95   2334900
   macro avg       0.95      0.95      0.95   2334900
weighted avg       0.95      0.95      0.95   2334900

Train Accuracy: 0.9545
Train MCC: 0.9084
---- Validation Metrics ----
[[267647  14663]
 [ 18967 322112]]
              precision    recall  f1-score   support

           0       0.93      0.95      0.94    282310
           1       0.96      0.94      0.95    341079

    accuracy                           0.95    623389
   macro avg       0.95      0.95      0.95    623389
weighted avg       0.95      0.95      0.95    623389

Validation Accuracy: 0.9461
Validation MCC: 0.8914
```

Note: reduce overfitting but increase bias

Parm: DecisionTreeClassifier(random_state=42,ccp_alpha=0.0001)

```
=== Decision Tree ===
---- Train Metrics ----
[[1045527   26046]
 [  25901 1237426]]
              precision    recall  f1-score   support

           0       0.98      0.98      0.98   1071573
           1       0.98      0.98      0.98   1263327

    accuracy                           0.98   2334900
   macro avg       0.98      0.98      0.98   2334900
weighted avg       0.98      0.98      0.98   2334900

Train Accuracy: 0.9778
Train MCC: 0.9552
---- Validation Metrics ----
[[274568   7742]
 [  7783 333296]]
              precision    recall  f1-score   support

           0       0.97      0.97      0.97    282310
           1       0.98      0.98      0.98    341079

    accuracy                           0.98    623389
   macro avg       0.97      0.97      0.97    623389
weighted avg       0.98      0.98      0.98    623389

Validation Accuracy: 0.9751
Validation MCC: 0.9497
```

Note: reduce overfitting , (Acceptable) bias -> but we can increase bias more

Action : more complex -> Random forest

Random Forest

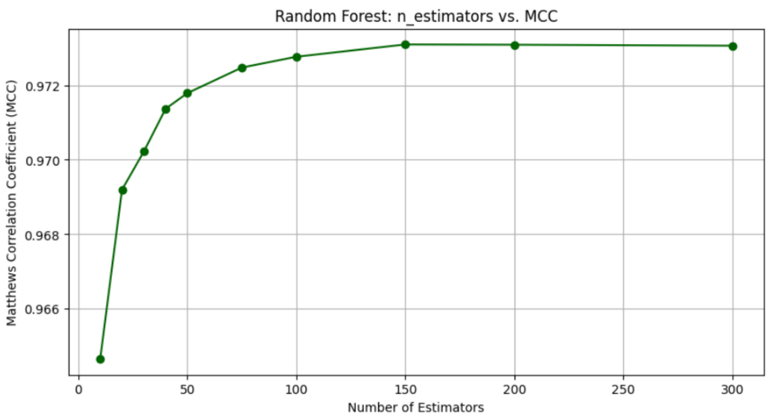
Parm: RandomForestClassifier(random_state=42) -> default parm

```
=== Random Forest ===
---- Train Metrics ----
[[1071149     424]
 [    518 1262809]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00   1071573
           1       1.00      1.00      1.00   1263327

    accuracy                           1.00   2334900
   macro avg       1.00      1.00      1.00   2334900
weighted avg       1.00      1.00      1.00   2334900

Train Accuracy: 0.9996
Train MCC: 0.9992
---- Validation Metrics ----
[[278978   3332]
 [  3561 337518]]
              precision    recall  f1-score   support

           0       0.99      0.99      0.99    282310
           1       0.99      0.99      0.99    341079

    accuracy                           0.99    623389
   macro avg       0.99      0.99      0.99    623389
weighted avg       0.99      0.99      0.99    623389

Validation Accuracy: 0.9889
Validation MCC: 0.9777
```

Note: low bias & small (Acceptable) variance

Action : increase the number of estimators (study the best number of estimators first)

Working on sample of the data 200K row



Parm: RandomForestClassifier(n_estimators=150, random_state=42)

```
=== Random Forest (double estimators) ===
---- Train Metrics ----
[[1071163     410]
 [    494 1262833]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00   1071573
           1       1.00      1.00      1.00   1263327

    accuracy                           1.00   2334900
   macro avg       1.00      1.00      1.00   2334900
weighted avg       1.00      1.00      1.00   2334900

Train Accuracy: 0.9996
Train MCC: 0.9992
---- Validation Metrics ----
[[278996   3314]
 [  3508 337571]]
              precision    recall  f1-score   support

           0       0.99      0.99      0.99    282310
           1       0.99      0.99      0.99    341079

    accuracy                           0.99    623389
   macro avg       0.99      0.99      0.99    623389
weighted avg       0.99      0.99      0.99    623389

Validation Accuracy: 0.9891
Validation MCC: 0.9779
```

Note: small change

Action : try Ensemble Models

Bagging

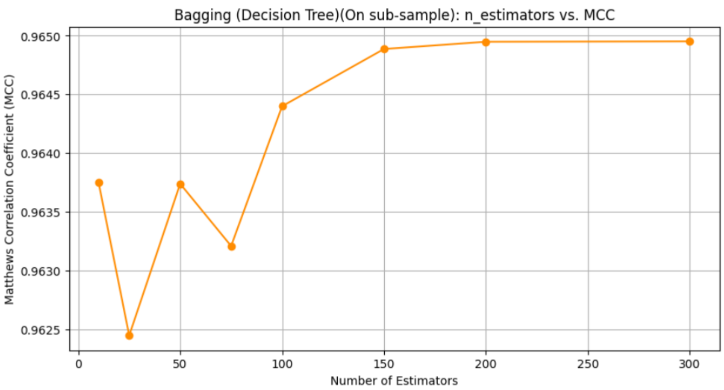
Parm : BaggingClassifier(estimator =DecisionTreeClassifier(), n_estimators=50, random_state=42,n_jobs=-1)

```
=== Bagging (Decision Tree) ===
---- Train Metrics ----
[[1071056     517]
 [    729 1262598]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00   1071573
           1       1.00      1.00      1.00   1263327

    accuracy                           1.00   2334900
   macro avg       1.00      1.00      1.00   2334900
weighted avg       1.00      1.00      1.00   2334900

Train Accuracy: 0.9995
Train MCC: 0.9989
---- Validation Metrics ----
[[278523   3787]
 [  4254 336825]]
              precision    recall  f1-score   support

           0       0.98      0.99      0.99    282310
           1       0.99      0.99      0.99    341079

    accuracy                           0.99    623389
   macro avg       0.99      0.99      0.99    623389
weighted avg       0.99      0.99      0.99    623389

Validation Accuracy: 0.9871
Validation MCC: 0.9740
```

Note : converges really will with small number of estimators relative to the other models

Action : increase the number of estimator (study the best number first)

On sample of the data only 200K row



Parm: BaggingClassifier(estimator =DecisionTreeClassifier(), n_estimators=200, random_state=42,n_jobs=-1)

```
=== Bagging (Decision Tree)(200 estimators) ===
---- Train Metrics ----
[[1071157     416]
 [    488 1262839]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00   1071573
           1       1.00      1.00      1.00   1263327

    accuracy                           1.00   2334900
   macro avg       1.00      1.00      1.00   2334900
weighted avg       1.00      1.00      1.00   2334900

Train Accuracy: 0.9996
Train MCC: 0.9992
---- Validation Metrics ----
[[278501   3809]
 [  4074 337005]]
              precision    recall  f1-score   support

           0       0.99      0.99      0.99    282310
           1       0.99      0.99      0.99    341079

    accuracy                           0.99    623389
   macro avg       0.99      0.99      0.99    623389
weighted avg       0.99      0.99      0.99    623389

Validation Accuracy: 0.9874
Validation MCC: 0.9745
```

Note : small change

Adaboost

Parm: default parm

Note: bad performance

Action: increase the number of estimators

Parm: AdaBoostClassifier(n_estimators=100,random_state=42)

```
=== AdaBoost ===
---- Train Metrics ----
[[ 817145  254428]
 [ 258688 1004639]]
              precision    recall  f1-score   support

           0       0.76      0.76      0.76   1071573
           1       0.80      0.80      0.80   1263327

    accuracy                           0.78   2334900
   macro avg       0.78      0.78      0.78   2334900
weighted avg       0.78      0.78      0.78   2334900

Train Accuracy: 0.7802
Train MCC: 0.5576
---- Validation Metrics ----
[[215320  66990]
 [ 72270 268809]]
              precision    recall  f1-score   support

           0       0.75      0.76      0.76    282310
           1       0.80      0.79      0.79    341079

    accuracy                           0.78    623389
   macro avg       0.77      0.78      0.77    623389
weighted avg       0.78      0.78      0.78    623389

Validation Accuracy: 0.7766
Validation MCC: 0.5500
```

Note: high bias

Action : increase the number of estimator more -> huge amount of time needed

XGBoost

Parm: XGBClassifier(n_estimators=100, learning_rate=0.1, use_label_encoder=False, eval_metric='logloss', random_state=42)

```
=== XGBoost ===
---- Train Metrics ----
[[1049511   22062]
 [  30124 1233203]]
              precision    recall  f1-score   support

           0       0.97      0.98      0.98   1071573
           1       0.98      0.98      0.98   1263327

    accuracy                           0.98   2334900
   macro avg       0.98      0.98      0.98   2334900
weighted avg       0.98      0.98      0.98   2334900

Train Accuracy: 0.9776
Train MCC: 0.9550
---- Validation Metrics ----
[[274710   7600]
 [  8112 332967]]
              precision    recall  f1-score   support

           0       0.97      0.97      0.97    282310
           1       0.98      0.98      0.98    341079

    accuracy                           0.97    623389
   macro avg       0.97      0.97      0.97    623389
weighted avg       0.97      0.97      0.97    623389

Validation Accuracy: 0.9748
Validation MCC: 0.9491
```

Note: very low variance , needs to decrease bias && it was fast

Action : increase the number of estimator directly

Parm: xgb_model = XGBClassifier(objective='binary:logistic', use_label_encoder=False, n_estimators=500)

```
=== XGBoost ===
---- Train Metrics ----
[[1061342   10231]
 [  13669 1249658]]
              precision    recall  f1-score   support

           0       0.99      0.99      0.99   1071573
           1       0.99      0.99      0.99   1263327

    accuracy                           0.99   2334900
   macro avg       0.99      0.99      0.99   2334900
weighted avg       0.99      0.99      0.99   2334900

Train Accuracy: 0.9898
Train MCC: 0.9794
---- Validation Metrics ----
[[279049   3261]
 [  3719 337360]]
              precision    recall  f1-score   support

           0       0.99      0.99      0.99    282310
           1       0.99      0.99      0.99    341079

    accuracy                           0.99    623389
   macro avg       0.99      0.99      0.99    623389
weighted avg       0.99      0.99      0.99    623389

Validation Accuracy: 0.9888
Validation MCC: 0.9774
```

Note:very low variance , very low bias

Parm: feval=mcc_eval, # custom MCC metric with 1000 estimator

```
=== XGBoost custom MCC  ===
---- Train Metrics ----
[[1062312    9261]
 [  12465 1250862]]
              precision    recall  f1-score   support

           0       0.99      0.99      0.99   1071573
           1       0.99      0.99      0.99   1263327

    accuracy                           0.99   2334900
   macro avg       0.99      0.99      0.99   2334900
weighted avg       0.99      0.99      0.99   2334900

Train Accuracy: 0.9907
Train MCC: 0.9813
---- Validation Metrics ----
[[279121   3189]
 [  3501 337578]]
              precision    recall  f1-score   support

           0       0.99      0.99      0.99    282310
           1       0.99      0.99      0.99    341079

    accuracy                           0.99    623389
   macro avg       0.99      0.99      0.99    623389
weighted avg       0.99      0.99      0.99    623389

Validation Accuracy: 0.9893
Validation MCC: 0.9783
```

Note: low variance , very low bias

Best model we can use is XGBoost -> lets tune some of his parameters using StratifiedKFold CV and GridsearchCV

```
xgb = XGBClassifier(
    objective='binary:logistic',
    use_label_encoder=False,
    eval_metric='logloss',
    verbosity=0,
    n_jobs=-1
)
param_grid = {
    'max_depth': [6,10, 14],
    'min_child_weight': [5, 7],
    'gamma': [1e-6, 1e-4],
    'subsample': [0.7,0.8],
    'colsample_bytree': [0.6],
    'reg_alpha': [0.1],
    'n_estimators': [50]
}
cv = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)
mcc_scorer = make_scorer(matthews_corrcoef)
grid_search = GridSearchCV(
    estimator=xgb,
    param_grid=param_grid,
    scoring=mcc_scorer,
    n_jobs=-1,
    cv=cv,
    verbose=2
)
```

```
Best parameters: {'colsample_bytree': 0.6, 'gamma': 0.0001, 'max_depth': 14, 'min_child_weight': 5, 'n_estimators': 50, 'reg_alpha': 0.1, 'subsample': 0.8}
Best cross-validation score: 0.9797767095341166
```

Test Scores

submission_CV_best_all_W_outliers.csv Complete (after deadline) · 6h ago	**0.97998**	**0.98013**	☐
submission_rf_150_estimator_all_W_outliers.csv Complete (after deadline) · 8h ago	**0.97917**	**0.97902**	☐
submission_rf_100_estimator_W_outliers.csv Complete (after deadline) · 20h ago	**0.97892**	**0.97884**	☐
submission.csv Complete (after deadline) · 2d ago · xgb_1000_estimator.csv XGboost with 80% of the train with no outliers	**0.97863**	**0.97864**	☐
submission_xgb_500_estimator_all_W_outliers.csv Complete (after deadline) · 9h ago	**0.97853**	**0.97854**	☐
submission_xgb_500_estimator_W_outliers.csv Complete (after deadline) · 9h ago	**0.97843**	**0.97842**	☐
submission_rf_200_estimator.csv Complete (after deadline) · 20h ago · random forest with 200 estimators	**0.97779**	**0.97767**	☐
submission_bagging_200_estimator.csv Complete (after deadline) · 9h ago	**0.97454**	**0.97475**	☐

With less than 0.5% difference between the first score on the lead board (using AutoML)

Conclusion

- Tree based model was more powerful in our problem
- Xgboost is robust to overfitting relative to the models we used
- It always best practise to test the number of estimators relative to the evaluation matric to choose the best number of estimator that will lead to the best generalization
- Outliers can be not removed when you use tree based model and it can help the model performance
- Tree based model is much faster than logreg , svm model
- After getting the best model performance do intensive parm. tuning using gridSearchCV to get even more better result on your model
- Train the model on the whole dataset train , validation before submitting the test file on the competition it will give more better results
- Endcoding the features with the correct way (target encoder in our problem) can prevent you from getting really high dim data
- Splitting the data with stratification will help the model performance when dealing with imbalanced data