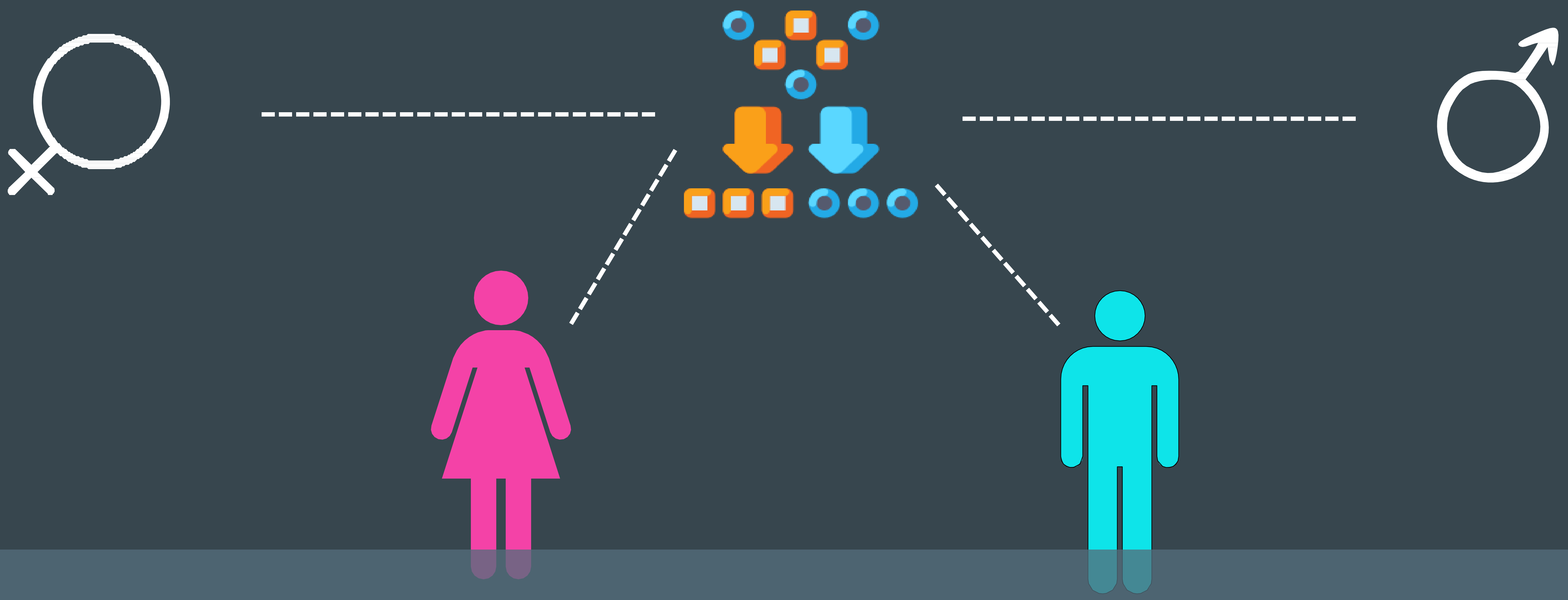
GENDER CLASSIFICATION



GROUP 5: AMOGH JAGINI
JYOTI NAIN
B. SREE VANI

Bhavan's Vivekananda College
Sainikpuri
Autonomous College

BACKGROUND

A survey was conducted to record various gender attributes such as long hair, forehead width in cm, forehead height in cm, nose wide, nose long, lips thin, distance from nose to lips.

OBJECTIVE

On the basis of these various factors, our objective is to determine the person's gender by their characteristics.

THE PATH

We followed numerous ML Algorithms to fit a predictive model to this data inorder to determine Gender.

ABOUT THE DATA

- long*hair* - This column contains 0's and 1's where 1 is "long hair" and 0 is "not long hair".
- foreheadwidthcm - This column is in CM's. This is the width of the forehead.
- foreheadheight*cm* - This is the height of the forehead and it's in Cm's.
- nosewide - This column contains 0's and 1's where 1 is "wide nose" and 0 is "not wide nose".
- nose*long* - This column contains 0's and 1's where 1 is "Long nose" and 0 is "not long nose".
- lipsthin - This column contains 0's and 1's where 1 represents the "thin lips" while 0 is "Not thin lips".
- distance*nose*to*lip*long - This column contains 0's and 1's where 1 represents the "long distance between nose and lips" while 0 is "short distance between nose and lips".
- gender - This is either "Male" or "Female".

The data around this population had 8 attributes/features and 5001 observations.

Categorical Columns

- LONG HAIR
- NOSE WIDE
- NOSE LONG
- LIPS THIN
- DISTANCE FROM NOSE TO LIP LONG
- GENDER

Numerical Columns

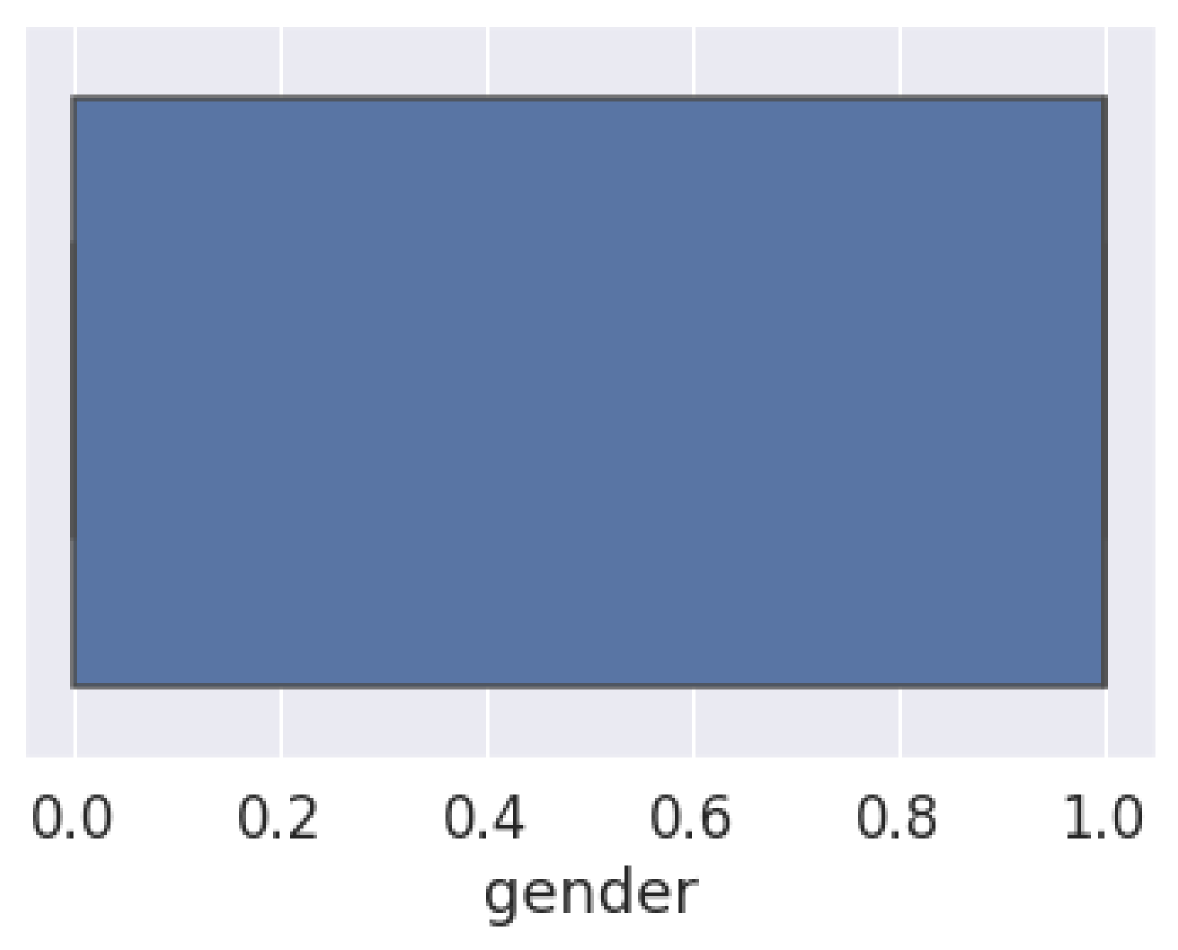
- FOREHEAD HEIGHT IN CM
- FOREHEAD WIDTH IN CM

	long_hair	forehead_width_cm	forehead_height_cm	nose_wide	nose_long	lips_thin	distance_nose_to_lip_long	gender
0	1	11.8	6.1	1	0	1	1	Male
1	0	14.0	5.4	0	0	1	0	Female
2	0	11.8	6.3	1	1	1	1	Male
3	0	14.4	6.1	0	1	1	1	Male
4	1	13.5	5.9	0	0	0	0	Female

DATA CLEANING

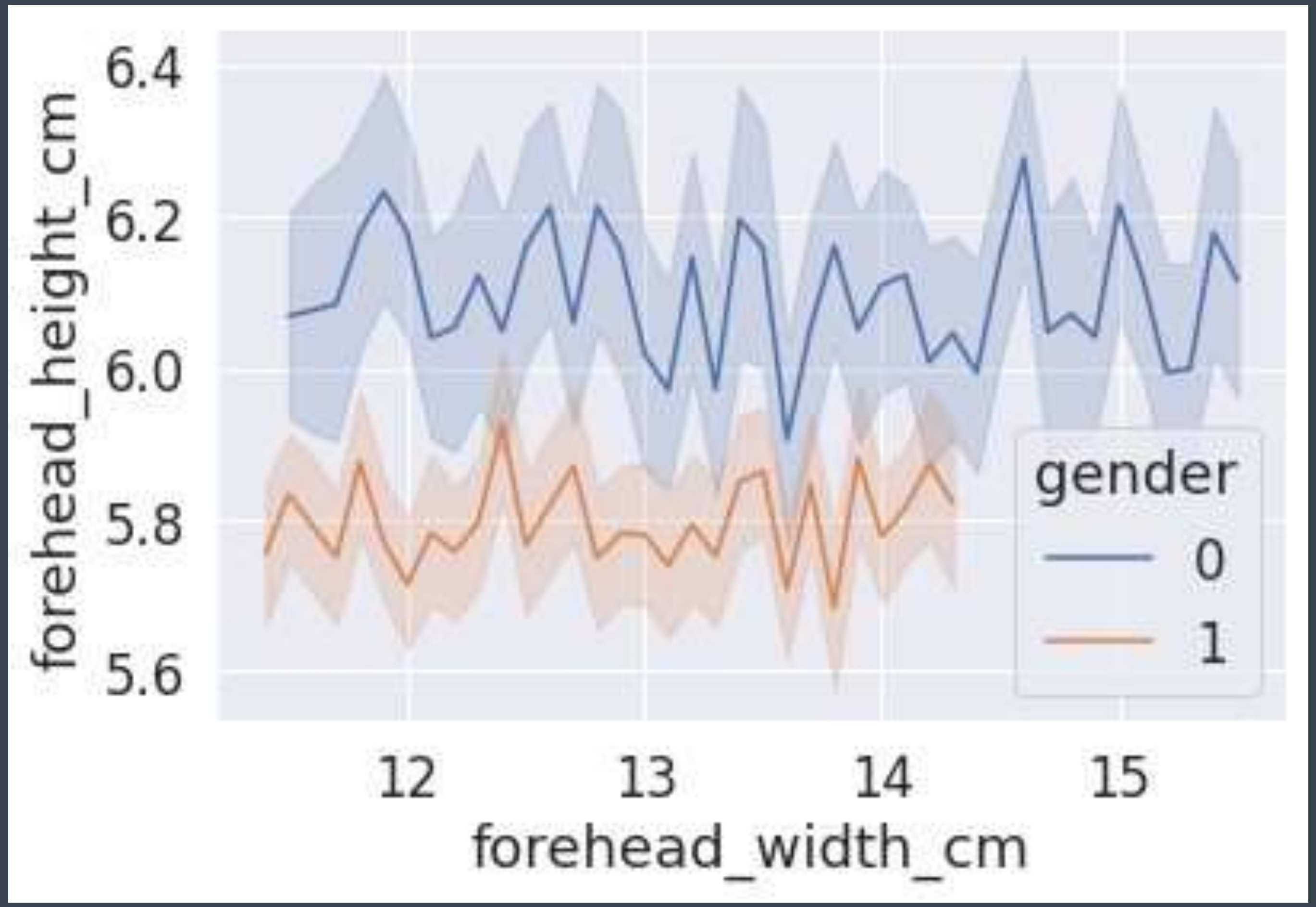
- There are no null values present.
- Replacing male with 0 and female with 1

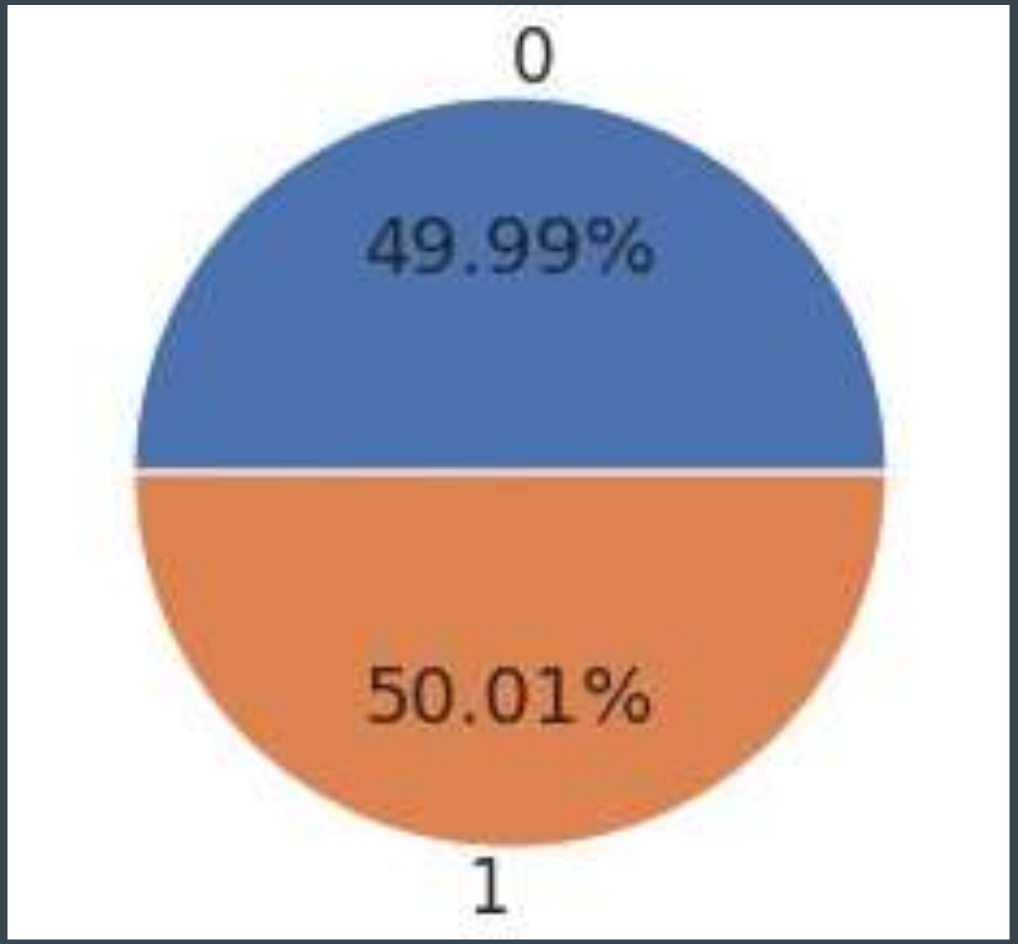
```
long_hair                    0
forehead_width_cm            0
forehead_height_cm           0
nose_wide                    0
nose_long                    0
lips_thin                    0
distance_nose_to_lip_long    0
gender                       0
dtype: int64
```

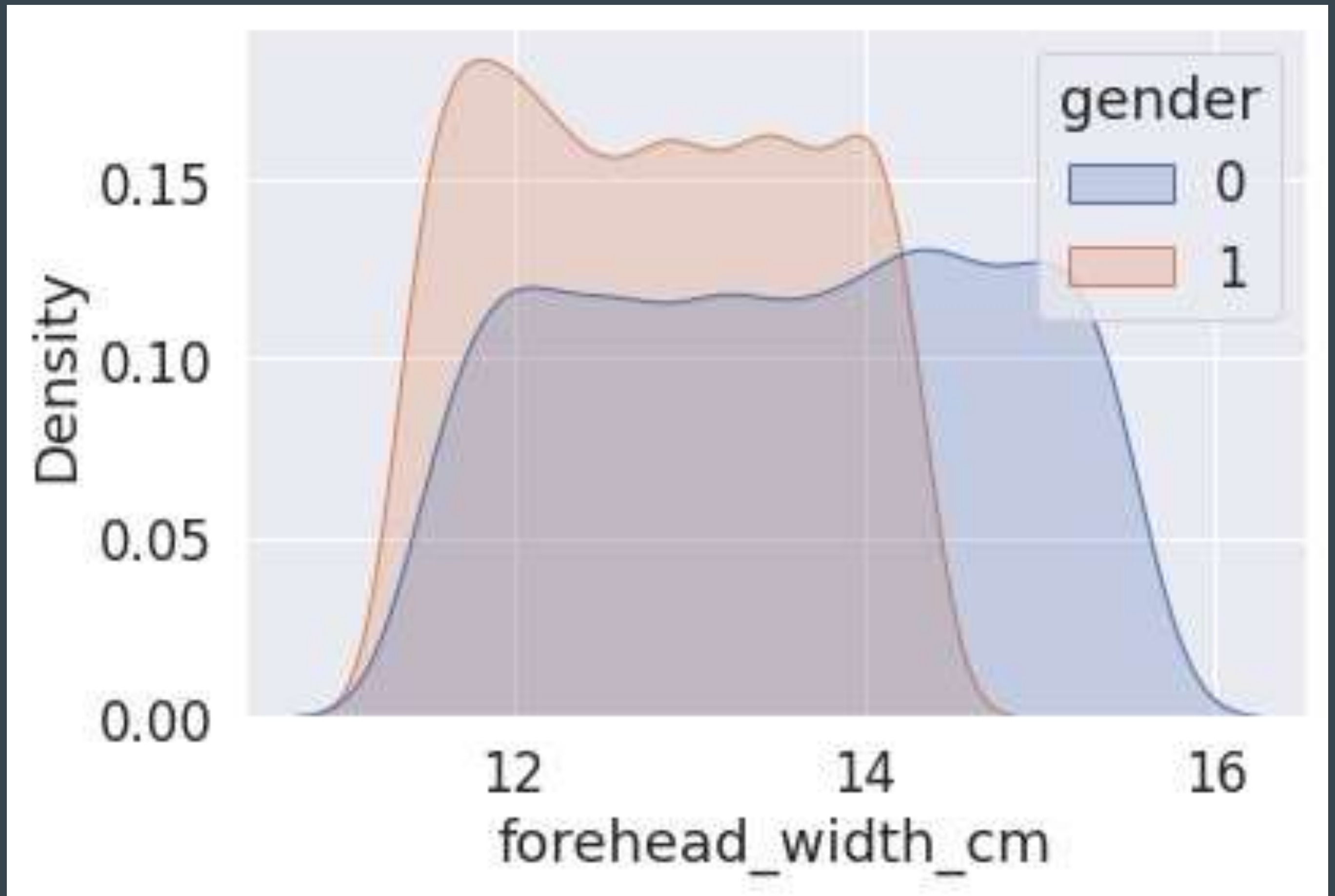


EXPLORATORY DATA ANALYSIS

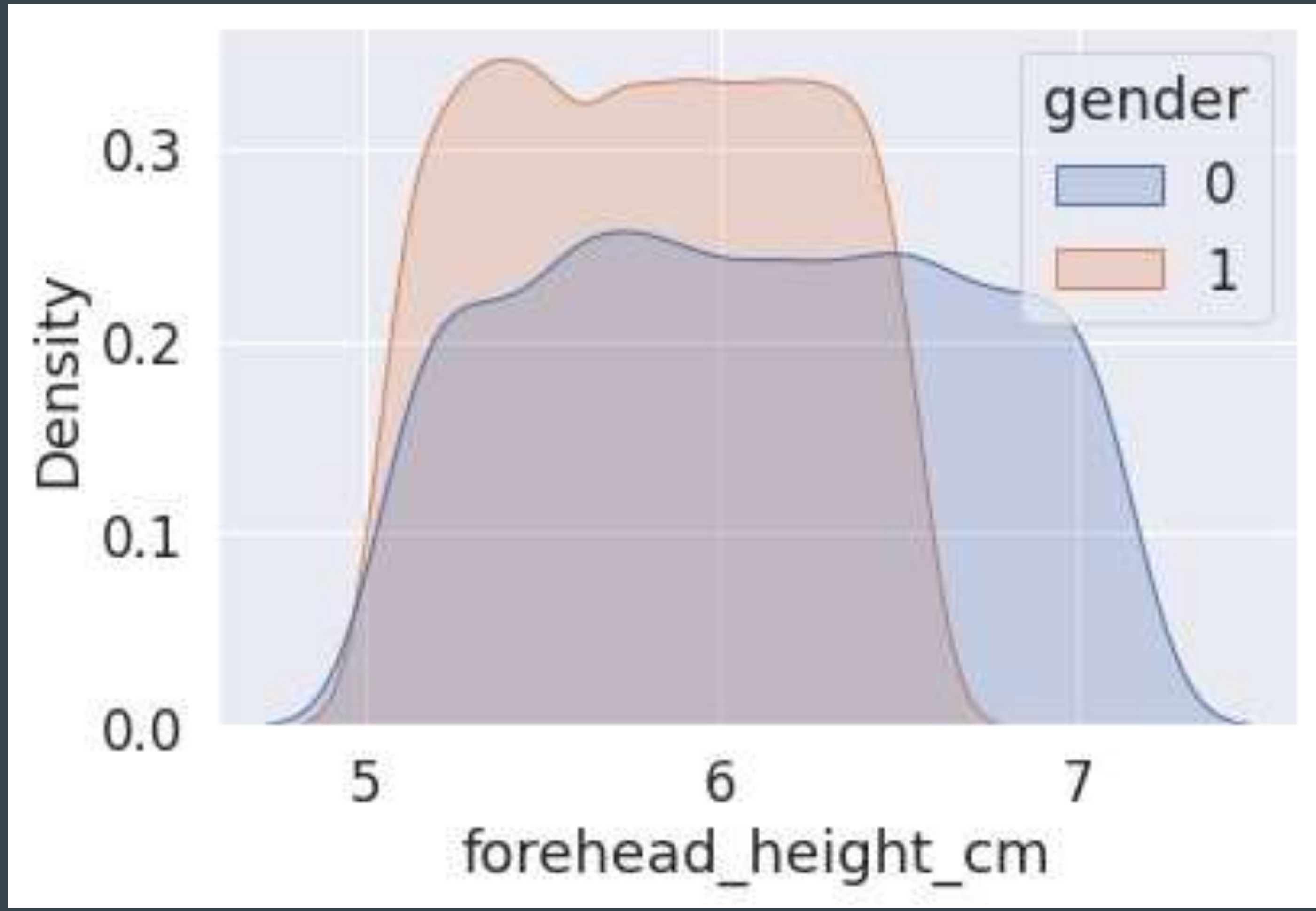
- These line plot show the gender based on several different variables of the dataset.
- The pie chart shows number of male and female from the given dataset.



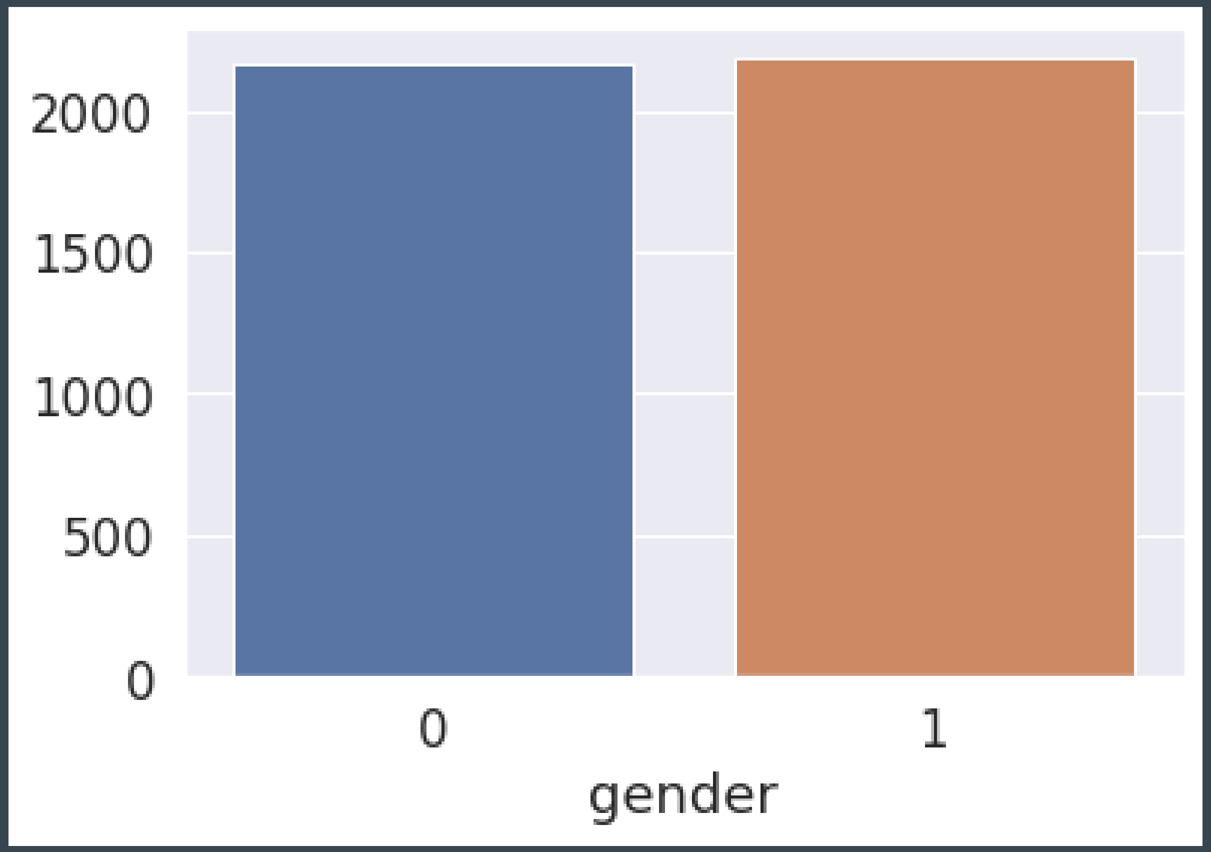




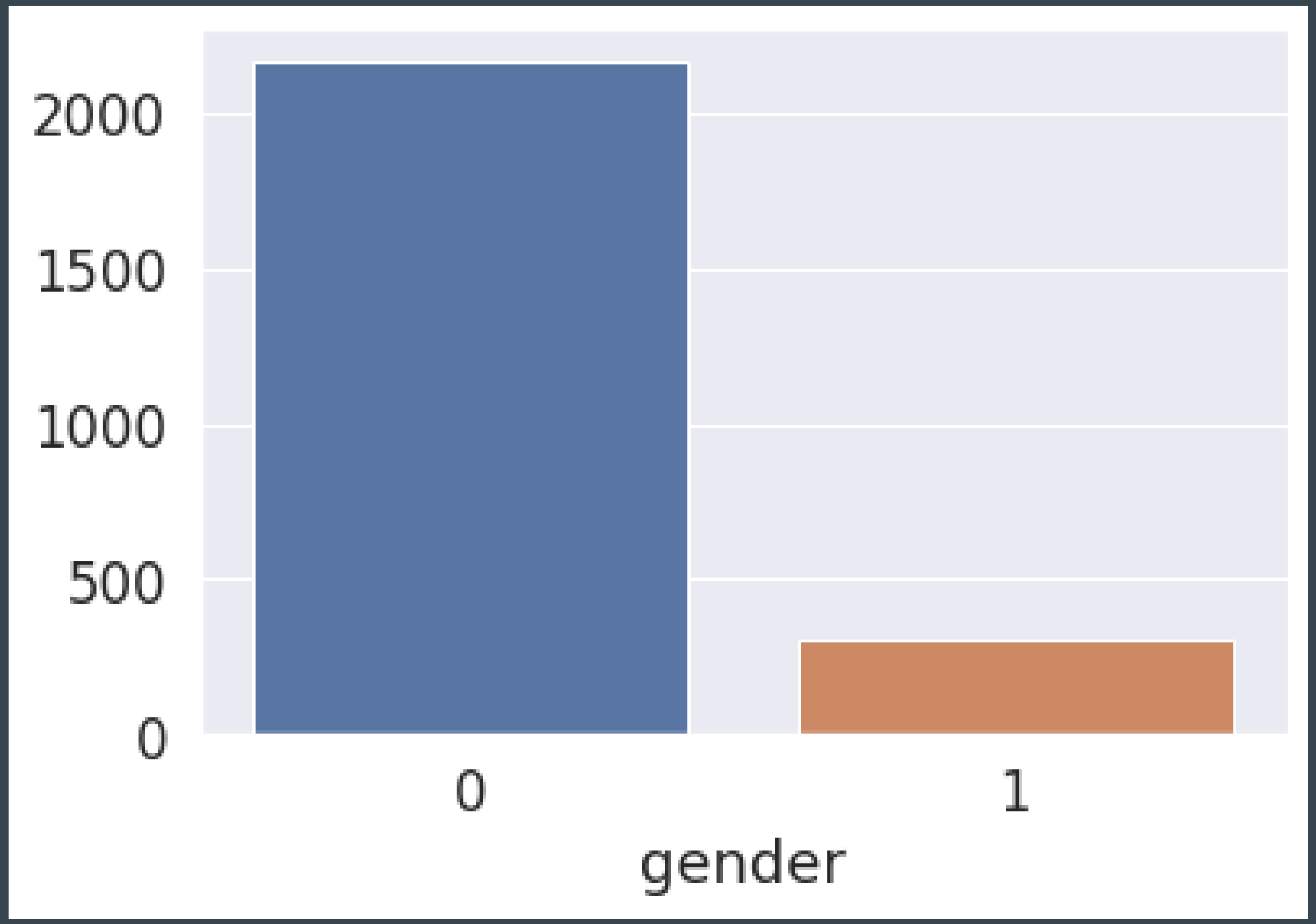
- These kde plot show the gender based on several different features like forehead width in cm, forehead height in cm



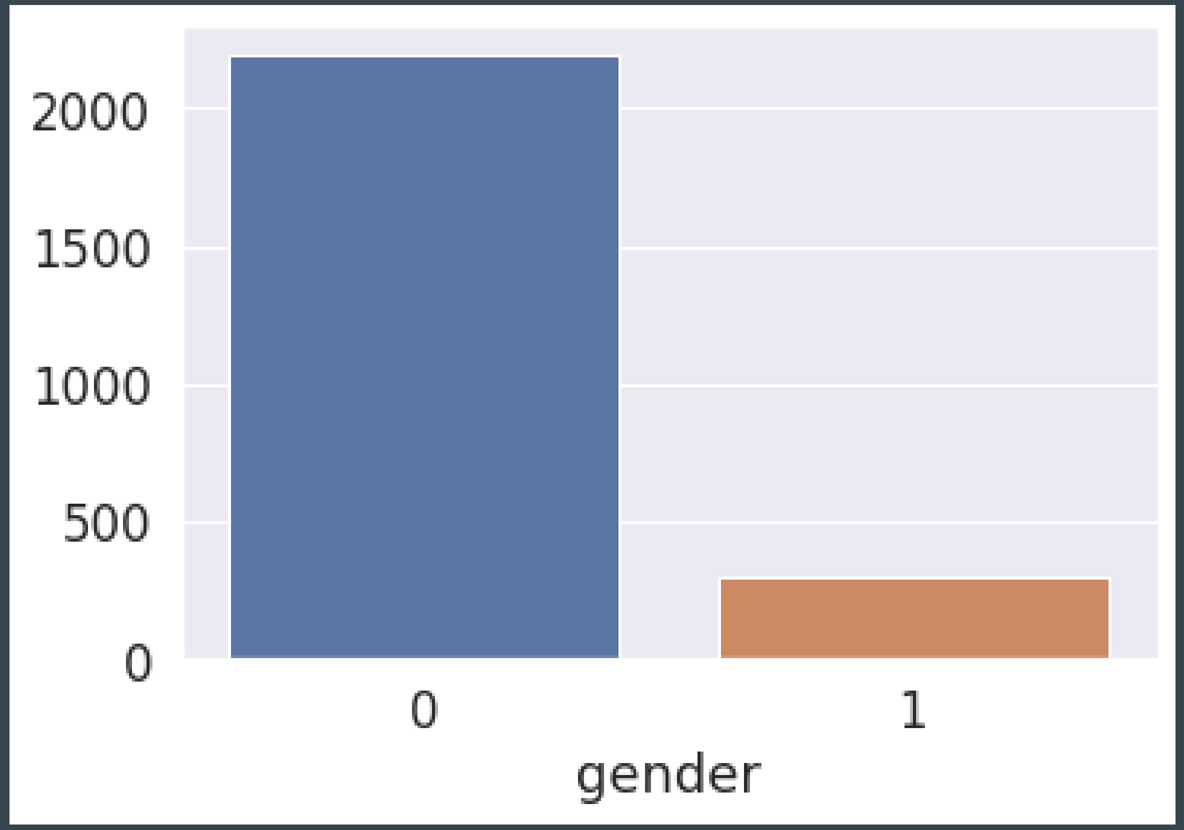
These bar plot show the gender based on several different features like long hair, lips thin, distance from nose to lip long



gender vs long hair

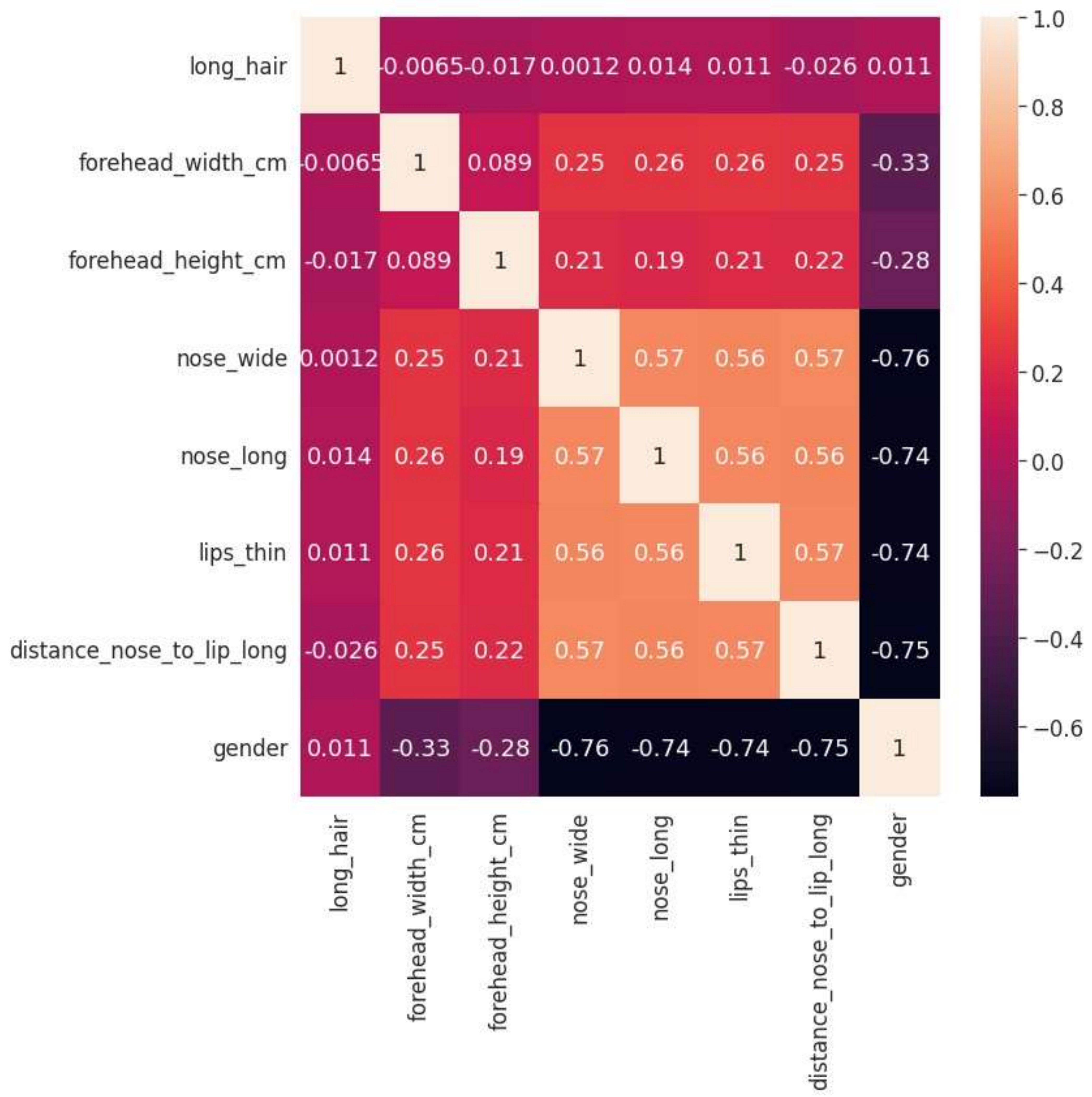


gender vs distance from nose to lip long



gender vs lips thin

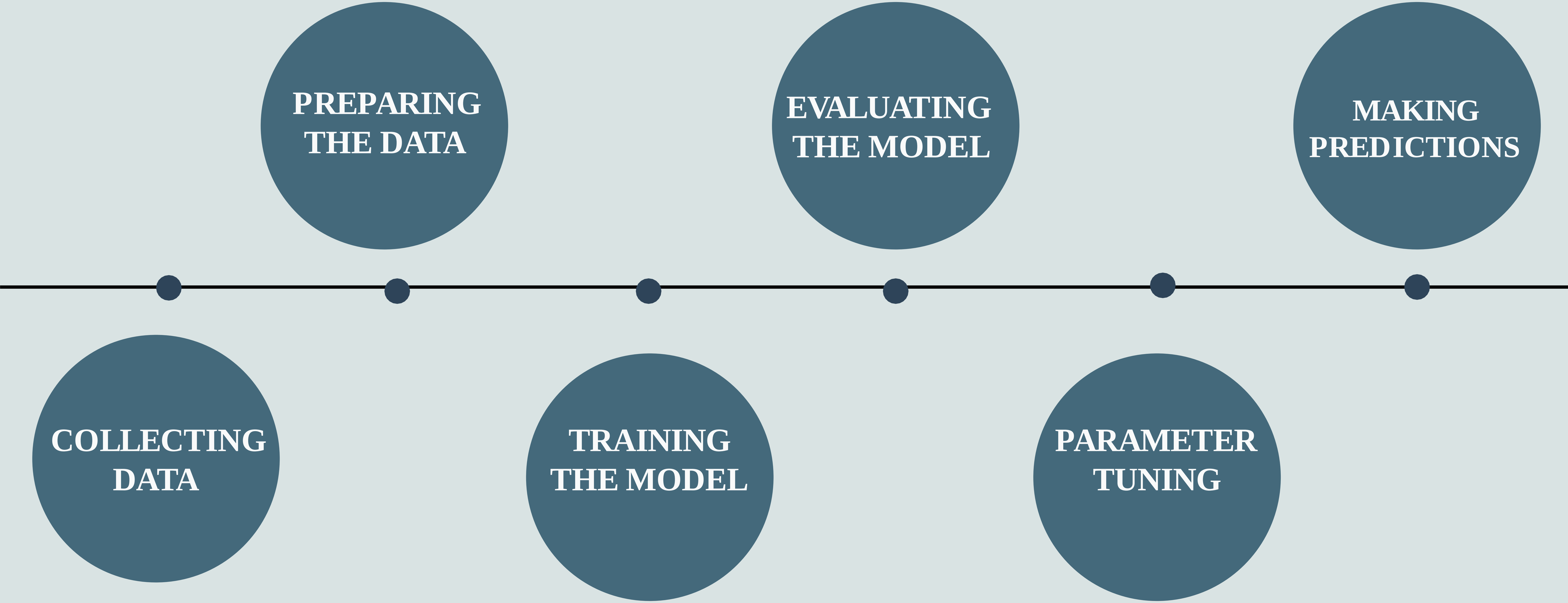
CORRELATION PLOT



EXPLORATORY DATA ANALYSIS INSIGHTS

- We replaced male with 0 and female with 1.
- Male's are having more forehead width, forehead height, nose wide, nose long, thin lips, distance from nose to lips.
- Female are having longer hair.
- There are no outliser and missing values in this dataset.

Steps involved in Analysis of Machine Learning Algorithm



LOGISTIC REGRESSION

ACCURACY VALUES

70 - 30 is giving the best ACCURACY value compared to other train-test splits.

TRAIN-TEST SPLIT	ACCURACY
90-10	0.96600
80-20	0.96650
75-25	0.96587
70-30	0.96772
60-40	0.96701

KNN (K NEAREST NEIGHBORS)

ACCURACY VALUES

75 - 25 is giving the best ACCURACY value compared to other train-test splits.

TRAIN-TEST SPLIT	ACCURACY
90-10	0.97022
80-20	0.96950
75-25	0.97227
70-30	0.971151
60-40	0.97034

SVM (SUPPORT VECTOR MACHINE)

ACCURACY VALUES

90 - 10 is giving the best ACCURACY value compared to other train-test splits.

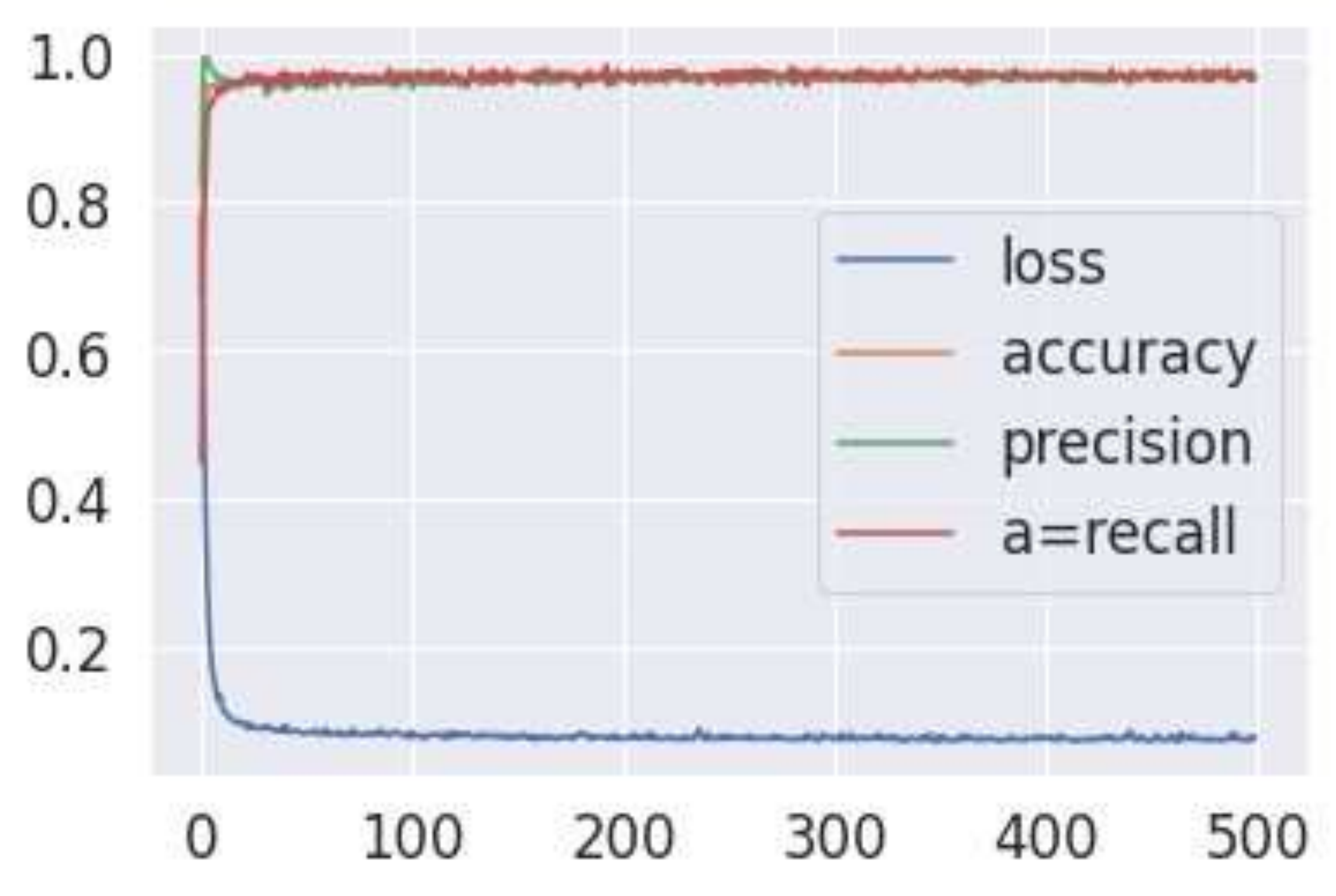
TRAIN-TEST SPLIT	ACCURACY
90-10	0.96956
80-20	0.96825
75-25	0.96800
70-30	0.96915
60-40	0.96867

NEURAL NETWORK

ADAM

Train-Test Proportion	Architecture	Optimizer	Epochs	Accuracy
70--30	7--1	Adam	50	0.9532
70--30	7--1	Adam	100	0.9529
70--30	7--1	Adam	200	0.9617
70--30	7--1	Adam	300	0.9637
70--30	7--1	Adam	400	0.9640
70--30	7--1	Adam	500	0.9652
70--30	5--2--1	Adam	50	0.9563
70--30	5--2--1	Adam	100	0.9537
70--30	5--2--1	Adam	200	0.9669
70--30	5--2--1	Adam	300	0.9637
70--30	5--2--1	Adam	400	0.9629
70--30	5--2--1	Adam	500	0.9669
70--30	7--5--3--1	Adam	50	0.4967
70--30	7--5--3--1	Adam	100	0.4967
70--30	7--5--3--1	Adam	200	0.4967
70--30	7--5--3--1	Adam	300	0.4967
70--30	7--5--3--1	Adam	400	0.4967
70--30	7--5--3--1	Adam	500	0.4967
70--30	7--5--3--2--1	Adam	50	0.4967
70--30	7--5--3--2--1	Adam	100	0.4967
70--30	7--5--3--2--1	Adam	200	0.4967
70--30	7--5--3--2--1	Adam	300	0.4967
70--30	7--5--3--2--1	Adam	400	0.4967
70--30	7--5--3--2--1	Adam	500	0.4967
70--30	5--3--2--1	Adam	50	0.4967
70--30	5--3--2--1	Adam	100	0.4967
70--30	5--3--2--1	Adam	200	0.4967
70--30	5--3--2--1	Adam	300	0.4967
70--30	5--3--2--1	Adam	400	0.4967
70--30	5--3--2--1	Adam	500	0.4967
70--30	5--3--1	Adam	50	0.9586
70--30	5--3--1	Adam	100	0.9532
70--30	5--3--1	Adam	200	0.9632
70--30	5--3--1	Adam	300	0.9634
70--30	5--3--1	Adam	400	0.9634
70--30	5--3--1	Adam	500	0.9654
70--30	7--5--2--1	Adam	50	0.4967
70--30	7--5--2--1	Adam	100	0.4967
70--30	7--5--2--1	Adam	200	0.4967
70--30	7--5--2--1	Adam	300	0.4967
70--30	7--5--2--1	Adam	400	0.4967
70--30	7--5--2--1	Adam	500	0.4967

Train-Test Proportion	Architecture	Optimizer	Epochs	Accuracy
80--20	7--1	Adam	50	0.9503
80--20	7--1	Adam	100	0.9513
80--20	7--1	Adam	200	0.9545
80--20	7--1	Adam	300	0.9615
80--20	7--1	Adam	400	0.9590
80--20	7--1	Adam	500	0.9610
80--20	5--2--1	Adam	50	0.9533
80--20	5--2--1	Adam	100	0.9535
80--20	5--2--1	Adam	200	0.9620
80--20	5--2--1	Adam	300	0.9638
80--20	5--2--1	Adam	400	0.9650
80--20	5--2--1	Adam	500	0.9648
80--20	7--5--3--1	Adam	50	0.4959
80--20	7--5--3--1	Adam	100	0.4959
80--20	7--5--3--1	Adam	200	0.4959
80--20	7--5--3--1	Adam	300	0.4959
80--20	7--5--3--1	Adam	400	0.4959
80--20	7--5--3--1	Adam	500	0.4959
80--20	7--5--3--2--1	Adam	50	0.4959
80--20	7--5--3--2--1	Adam	100	0.4959
80--20	7--5--3--2--1	Adam	200	0.4959
80--20	7--5--3--2--1	Adam	300	0.4959
80--20	7--5--3--2--1	Adam	400	0.4959
80--20	7--5--3--2--1	Adam	500	0.4959
80--20	5--3--2--1	Adam	50	0.4959
80--20	5--3--2--1	Adam	100	0.4959
80--20	5--3--2--1	Adam	200	0.4959
80--20	5--3--2--1	Adam	300	0.4959
80--20	5--3--2--1	Adam	400	0.4959
80--20	5--3--2--1	Adam	500	0.4959
80--20	5--3--1	Adam	50	0.9553
80--20	5--3--1	Adam	100	0.9600
80--20	5--3--1	Adam	200	0.9563
80--20	5--3--1	Adam	300	0.9630
80--20	5--3--1	Adam	400	0.9613
80--20	5--3--1	Adam	500	0.9645
80--20	7--5--2--1	Adam	50	0.4959
80--20	7--5--2--1	Adam	100	0.4959
80--20	7--5--2--1	Adam	200	0.4959
80--20	7--5--2--1	Adam	300	0.4959
80--20	7--5--2--1	Adam	400	0.4959
80--20	7--5--2--1	Adam	500	0.4959


1.0
0.8
0.6
0.4
0.2
0
100
200
300
400
500
loss
accuracy
precision
a=recall

Epochs ---->

SGD

Train-Test Proportion	Architecture	Optimizer	Epochs	Accuracy
70--30	7--1	SGD	50	0.4967
70--30	7--1	SGD	100	0.4967
70--30	7--1	SGD	200	0.4967
70--30	7--1	SGD	300	0.4967
70--30	7--1	SGD	400	0.4967
70--30	7--1	SGD	500	0.4967
70--30	5--2--1	SGD	50	0.5033
70--30	5--2--1	SGD	100	0.5033
70--30	5--2--1	SGD	200	0.5033
70--30	5--2--1	SGD	300	0.5033
70--30	5--2--1	SGD	400	0.5033
70--30	5--2--1	SGD	500	0.5033
70--30	7--5--3--1	SGD	50	0.5033
70--30	7--5--3--1	SGD	100	0.5033
70--30	7--5--3--1	SGD	200	0.5033
70--30	7--5--3--1	SGD	300	0.5033
70--30	7--5--3--1	SGD	400	0.5033
70--30	7--5--3--1	SGD	500	0.5033
70--30	7--5--3--2--1	SGD	50	0.5033
70--30	7--5--3--2--1	SGD	100	0.5033
70--30	7--5--3--2--1	SGD	200	0.5033
70--30	7--5--3--2--1	SGD	300	0.5033
70--30	7--5--3--2--1	SGD	400	0.5033
70--30	7--5--3--2--1	SGD	500	0.5033
70--30	5--3--2--1	SGD	50	0.5033
70--30	5--3--2--1	SGD	100	0.5033
70--30	5--3--2--1	SGD	200	0.5033
70--30	5--3--2--1	SGD	300	0.5033
70--30	5--3--2--1	SGD	400	0.5033
70--30	5--3--2--1	SGD	500	0.5033
70--30	5--3--1	SGD	50	0.4370
70--30	5--3--1	SGD	100	0.4370
70--30	5--3--1	SGD	200	0.4370
70--30	5--3--1	SGD	300	0.4370
70--30	5--3--1	SGD	400	0.4370
70--30	5--3--1	SGD	500	0.4370
70--30	7--5--2--1	SGD	50	0.4979
70--30	7--5--2--1	SGD	100	0.4979
70--30	7--5--2--1	SGD	200	0.4979
70--30	7--5--2--1	SGD	300	0.4979
70--30	7--5--2--1	SGD	400	0.4979
70--30	7--5--2--1	SGD	500	0.4979

Train-Test Proportion	Architecture	Optimizer	Epochs	Accuracy
80--20	7--1	SGD	50	0.4959
80--20	7--1	SGD	100	0.4959
80--20	7--1	SGD	200	0.4959
80--20	7--1	SGD	300	0.4959
80--20	7--1	SGD	400	0.4959
80--20	7--1	SGD	500	0.4959
80--20	5--2--1	SGD	50	0.5041
80--20	5--2--1	SGD	100	0.5041
80--20	5--2--1	SGD	200	0.5041
80--20	5--2--1	SGD	300	0.5041
80--20	5--2--1	SGD	400	0.5041
80--20	5--2--1	SGD	500	0.5041
80--20	7--5--3--1	SGD	50	0.5041
80--20	7--5--3--1	SGD	100	0.5041
80--20	7--5--3--1	SGD	200	0.5041
80--20	7--5--3--1	SGD	300	0.5041
80--20	7--5--3--1	SGD	400	0.5041
80--20	7--5--3--1	SGD	500	0.5041
80--20	7--5--3--2--1	SGD	50	0.5041
80--20	7--5--3--2--1	SGD	100	0.5041
80--20	7--5--3--2--1	SGD	200	0.5041
80--20	7--5--3--2--1	SGD	300	0.5041
80--20	7--5--3--2--1	SGD	400	0.5041
80--20	7--5--3--2--1	SGD	500	0.5041
80--20	5--3--2--1	SGD	50	0.5041
80--20	5--3--2--1	SGD	100	0.5041
80--20	5--3--2--1	SGD	200	0.5041
80--20	5--3--2--1	SGD	300	0.5041
80--20	5--3--2--1	SGD	400	0.5041
80--20	5--3--2--1	SGD	500	0.5041
80--20	5--3--1	SGD	50	0.4366
80--20	5--3--1	SGD	100	0.4366
80--20	5--3--1	SGD	200	0.4366
80--20	5--3--1	SGD	300	0.4366
80--20	5--3--1	SGD	400	0.4366
80--20	5--3--1	SGD	500	0.4366
70--30	7--5--2--1	SGD	50	0.4969
70--30	7--5--2--1	SGD	100	0.4969
70--30	7--5--2--1	SGD	200	0.4969
70--30	7--5--2--1	SGD	300	0.4969
70--30	7--5--2--1	SGD	400	0.4969
70--30	7--5--2--1	SGD	500	0.4969

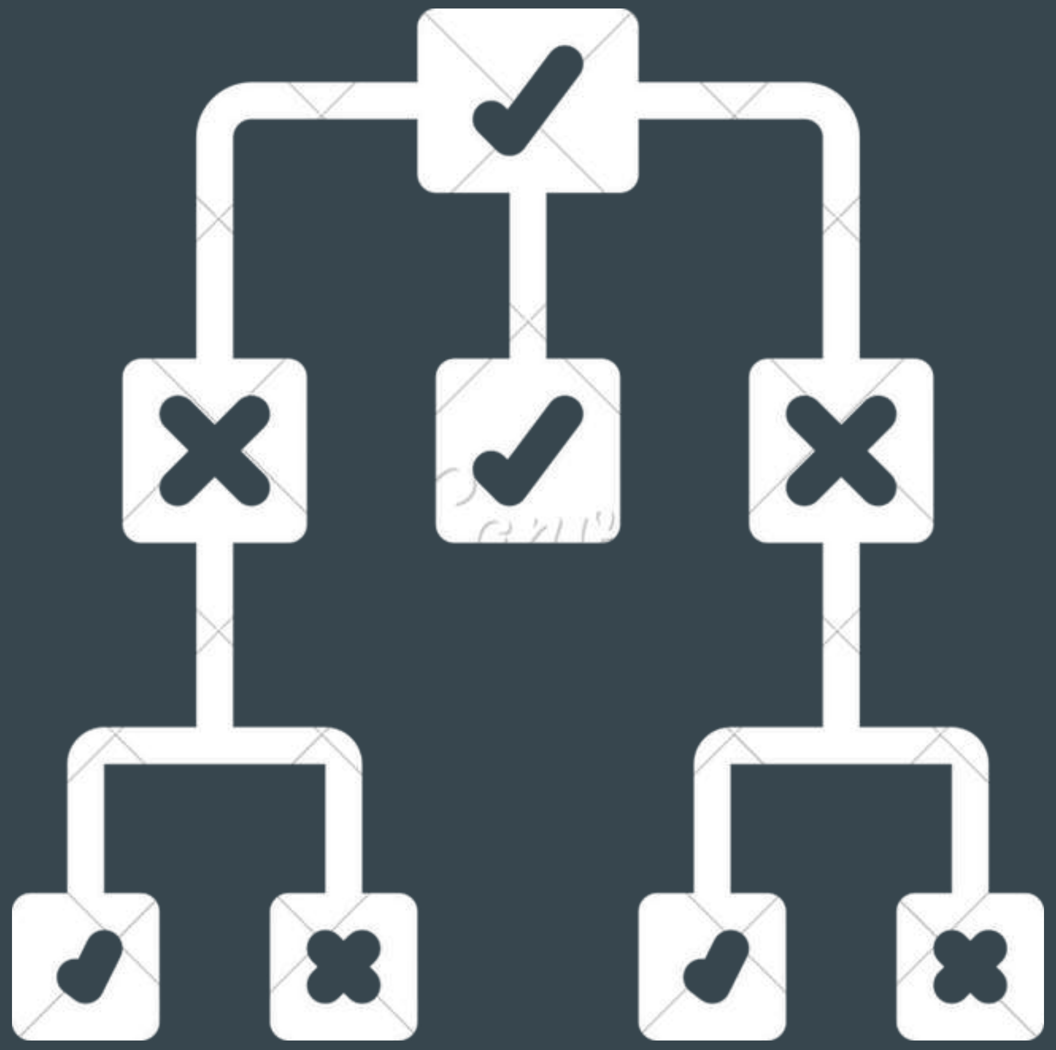
BAGGING

BAGGING CLASSIFIER → 0.96207

DECISION TREE → 0.97804

RANDOM FOREST → 0.97205

Train-Test Proportions	Bagging Classifier	Decision Tree	Random Forest
80--20	0.96703	0.97102	0.97602
70--30	0.96402	0.97102	0.97068
90--10	0.96207	0.97804	0.97205
60--40	0.96901	0.96851	0.97051
75--25	0.96962	0.97122	0.97761

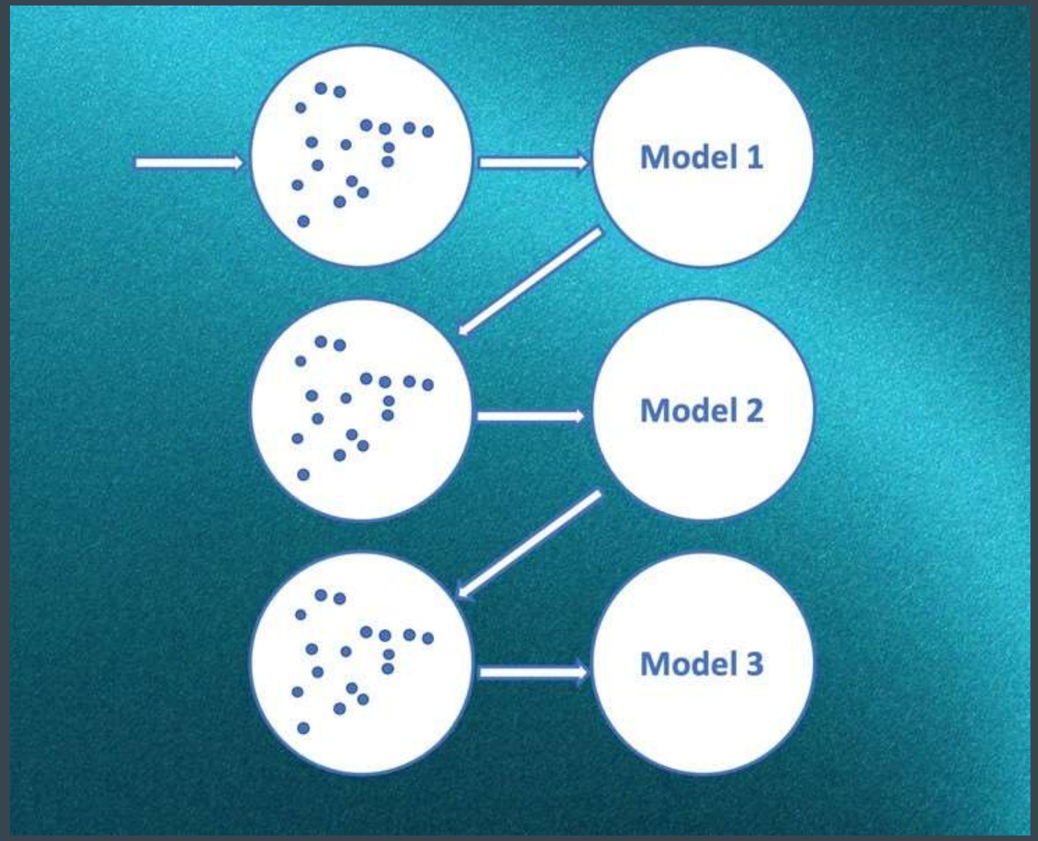


BOOSTING

ADABOOST → 0.98081

GRADIENT BOOST → 0.97761

XGBOOST → 0.98403



Train-Test Proportions	AdaBoost	GBM	XGBoost
80--20	0.98003	0.98003	0.98101
70--30	0.98001	0.97602	0.97601
90--10	0.97601	0.96868	0.98403
60--40	0.97701	0.97301	0.97701
75--25	0.98081	0.97761	0.98081

SUMMARY

Boosting is giving the highest accuracy comapred to other algorithms.

ALGORITHM	ACCURACY VALUES
LOGISTIC REGRESSION	0.96772
NEURAL NETWORK	0.96690
BAGGING	0.97804
BOOSTING	0.98403
KNN	0.97227
SVM	0.96956

CONCLUSION

XG Boost yields the highest accuracy value.

The best accuracy value is observed as 0.98081.

90-10 train-test split worked best for our dataset.

https://github.com/SreeVani23/UNP-2ND-PROJECT/blob/main/HDS_Project_G5_P2_gender_classification_dataset%20%20%20%20%20.ipynb

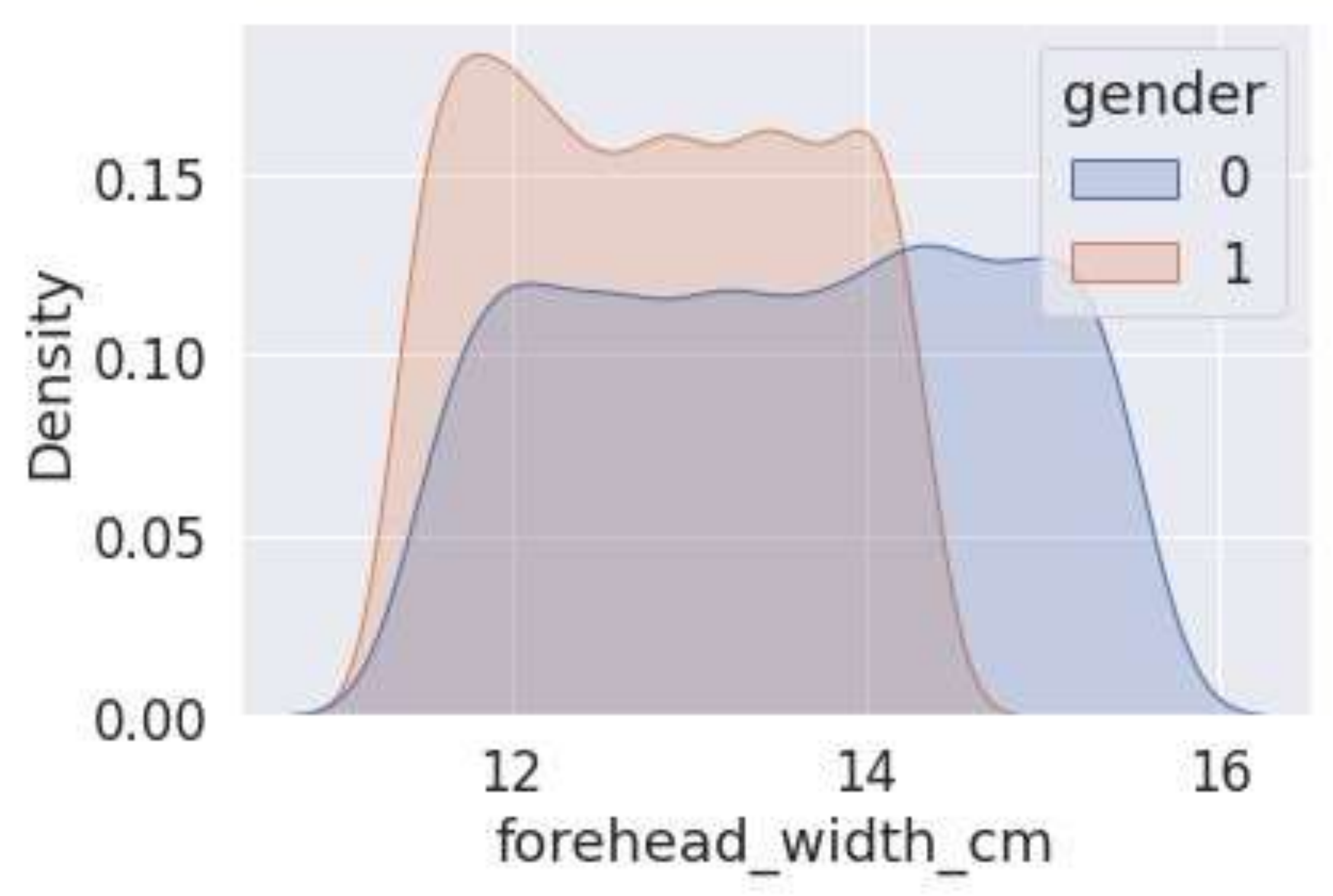
Google Colaboratory
google.com

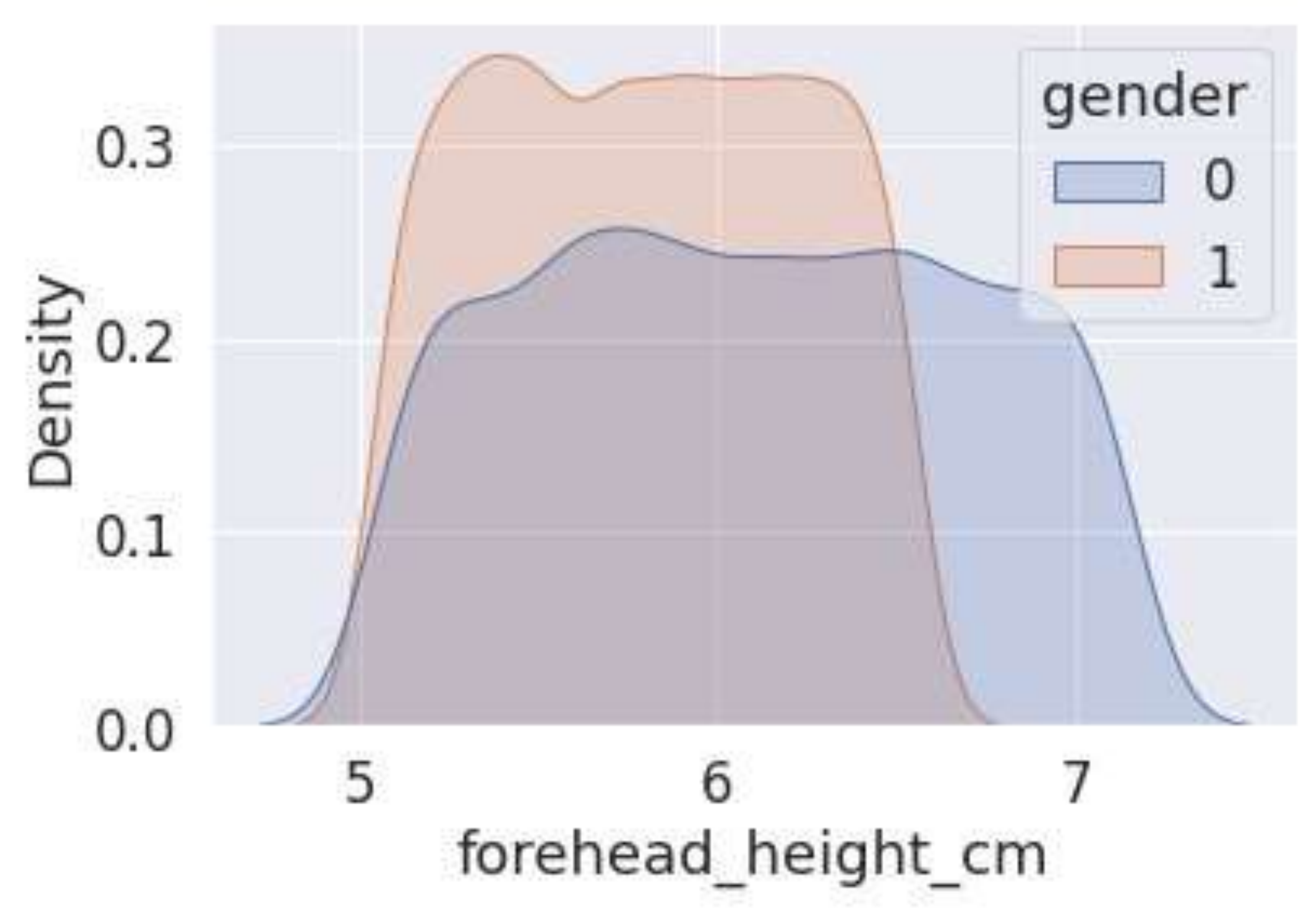
THANK YOU

BY: AMOGH JAGINI
B. SREE VANI
JYOTI NAIN

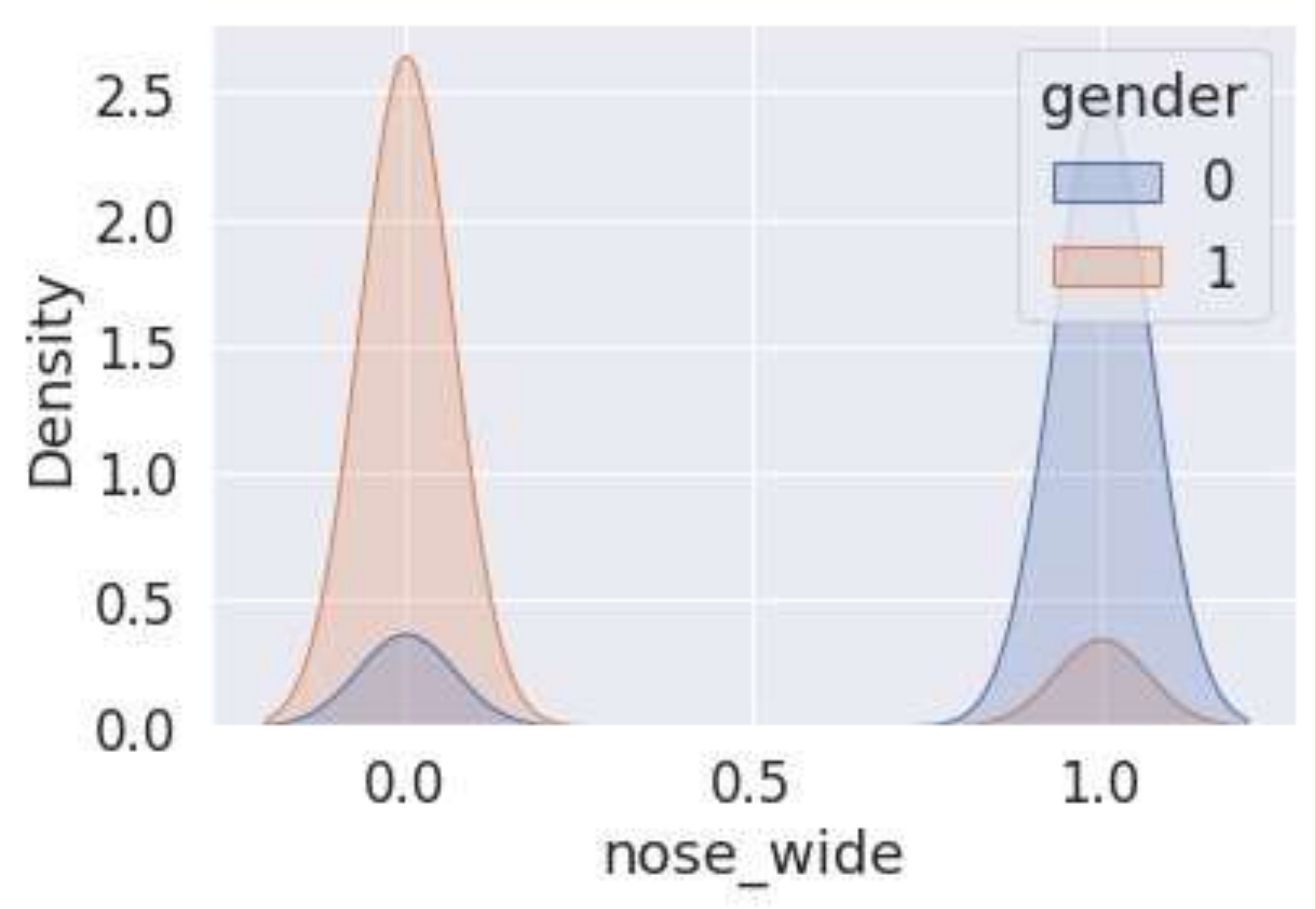
APPENDIX

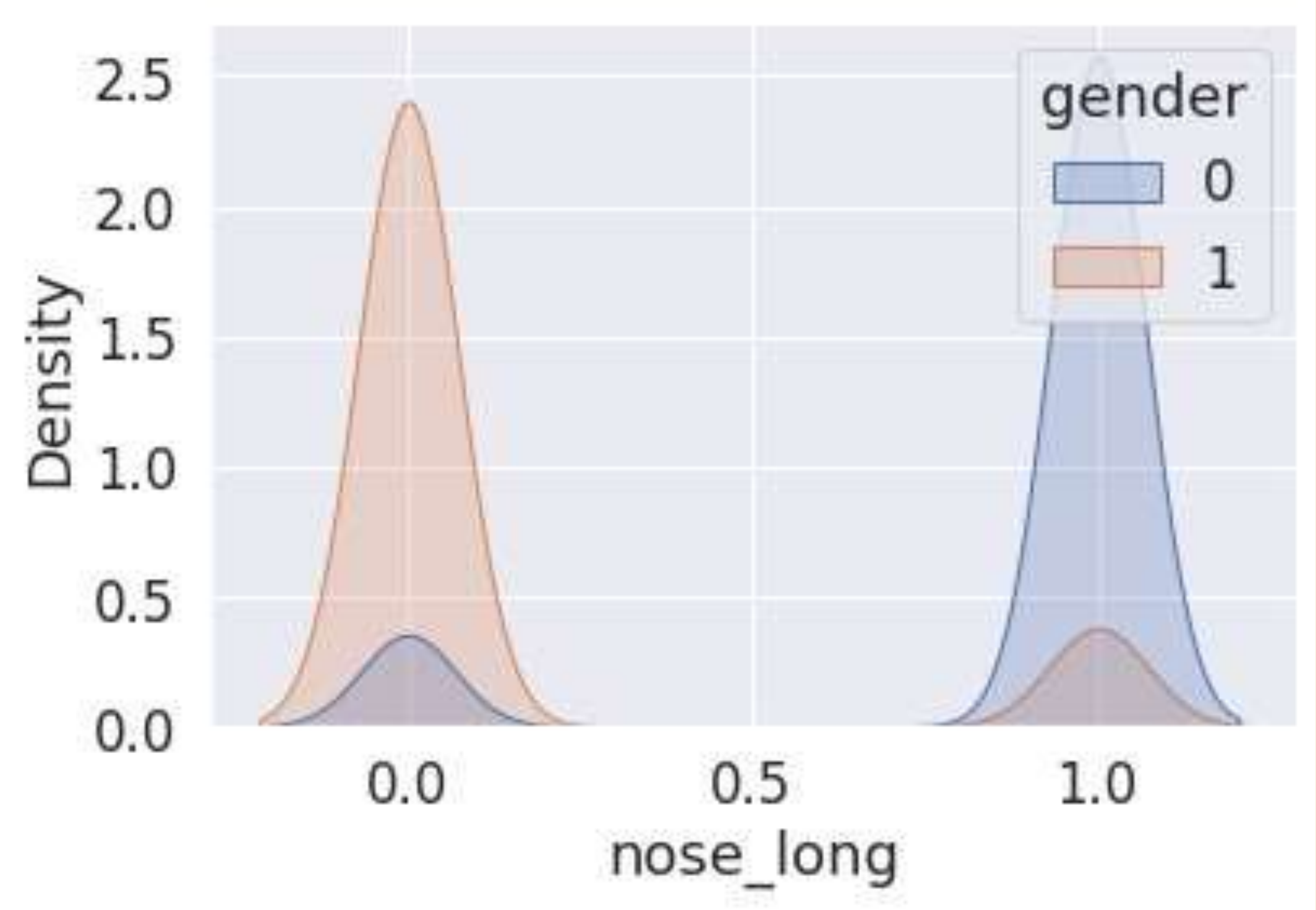
EDA



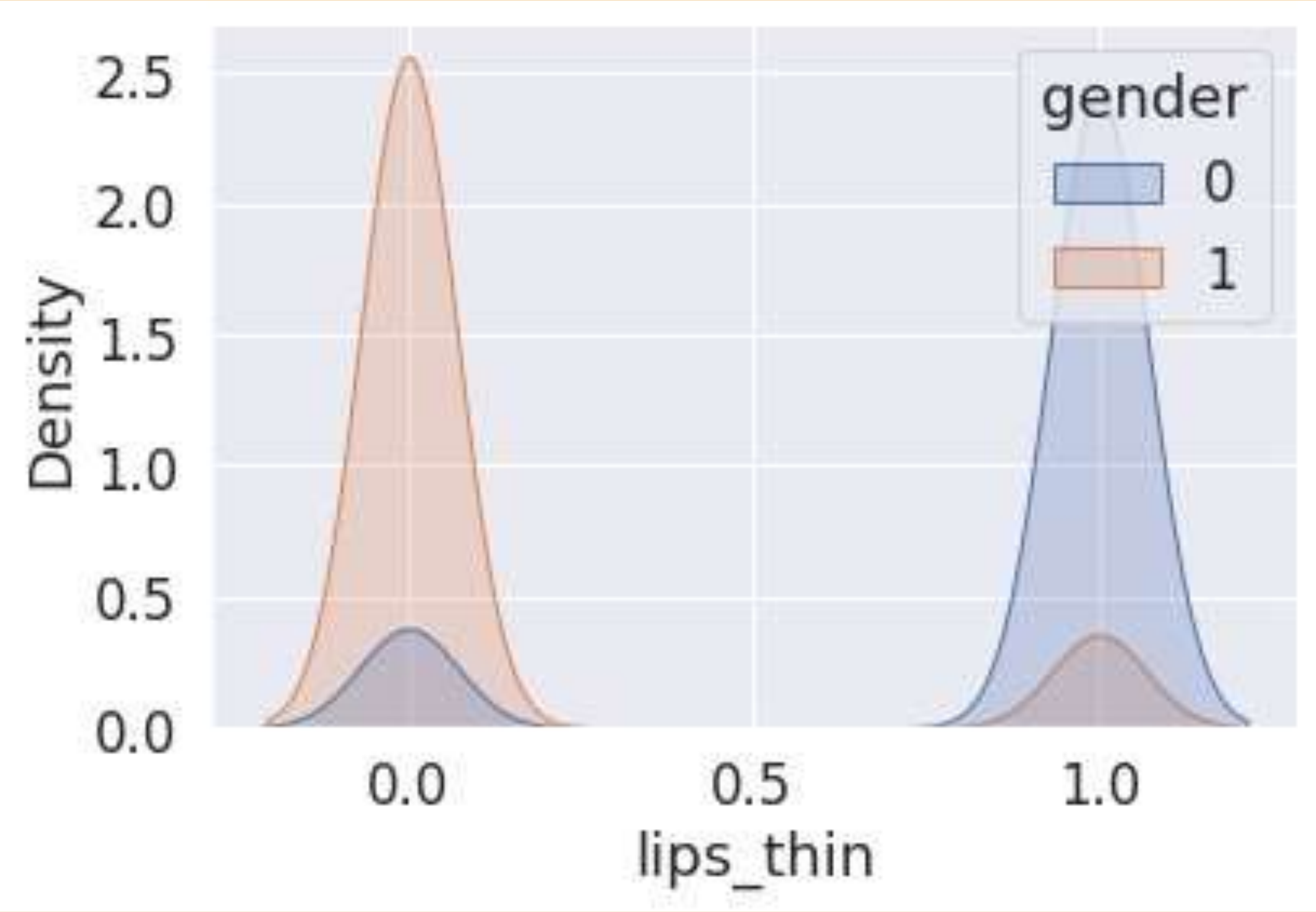


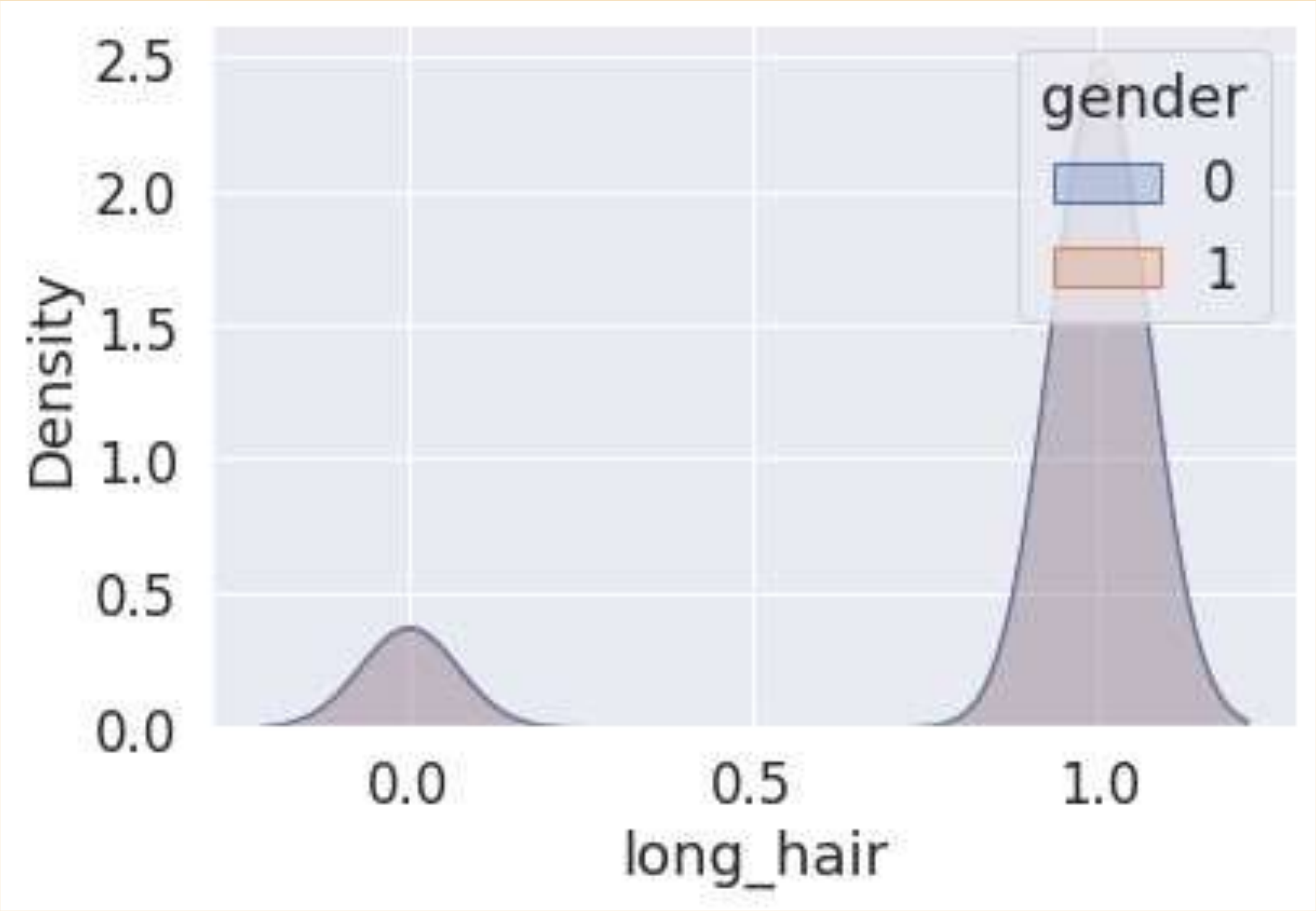
kde plot for forehead width in cm and forehead height in cm with respective to gender.



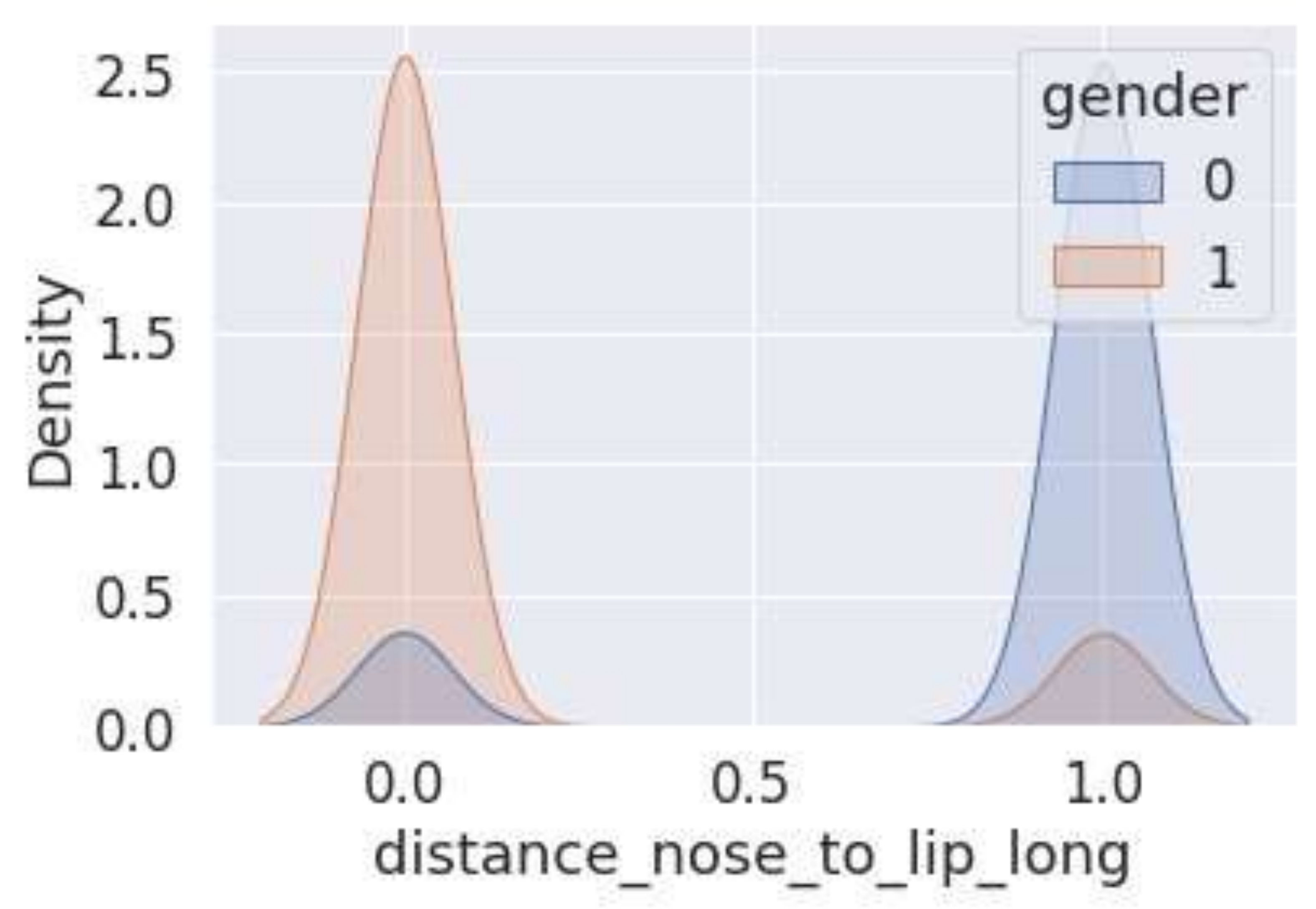


kde plot for nose wide and nose long with respective to gender.



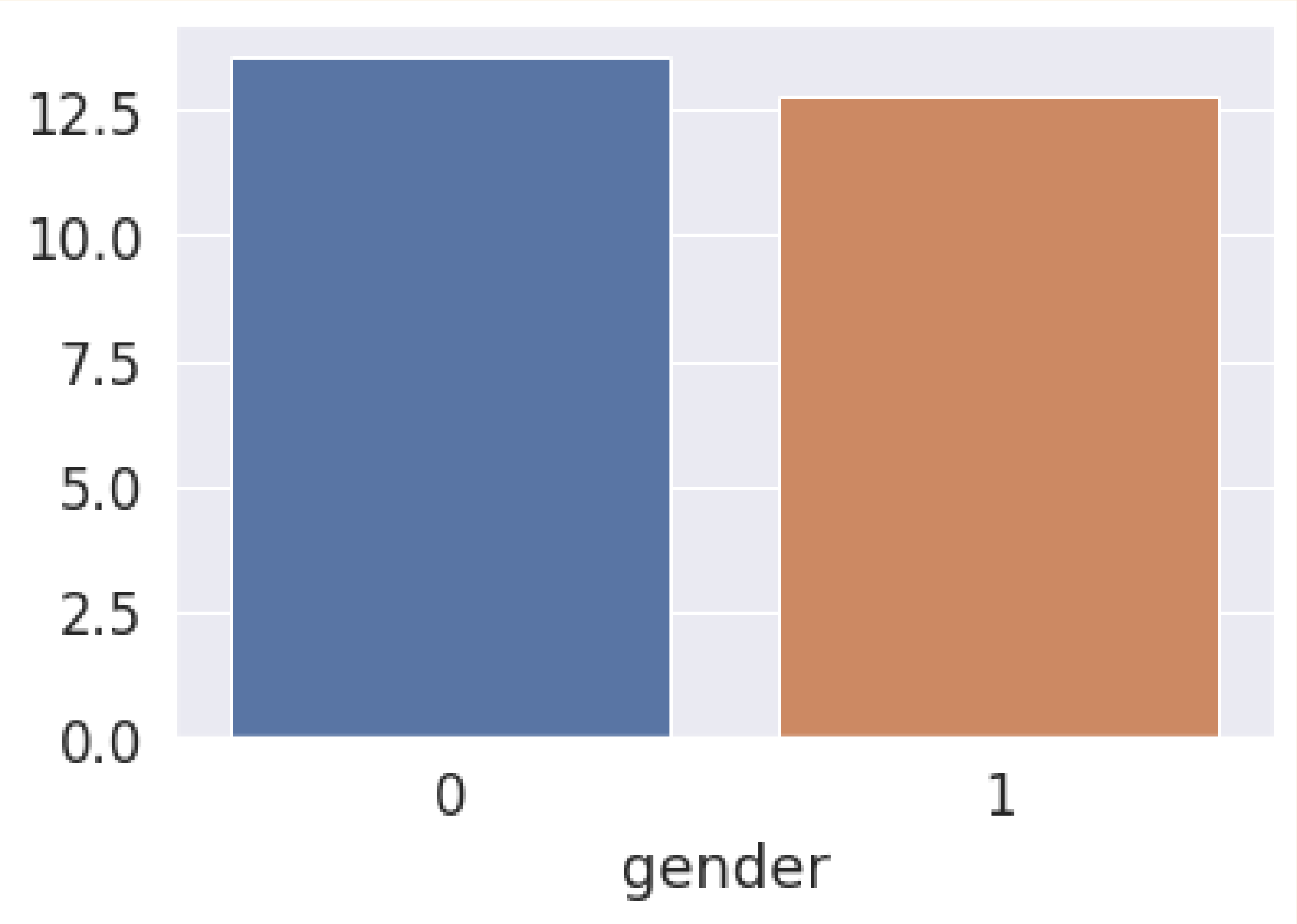


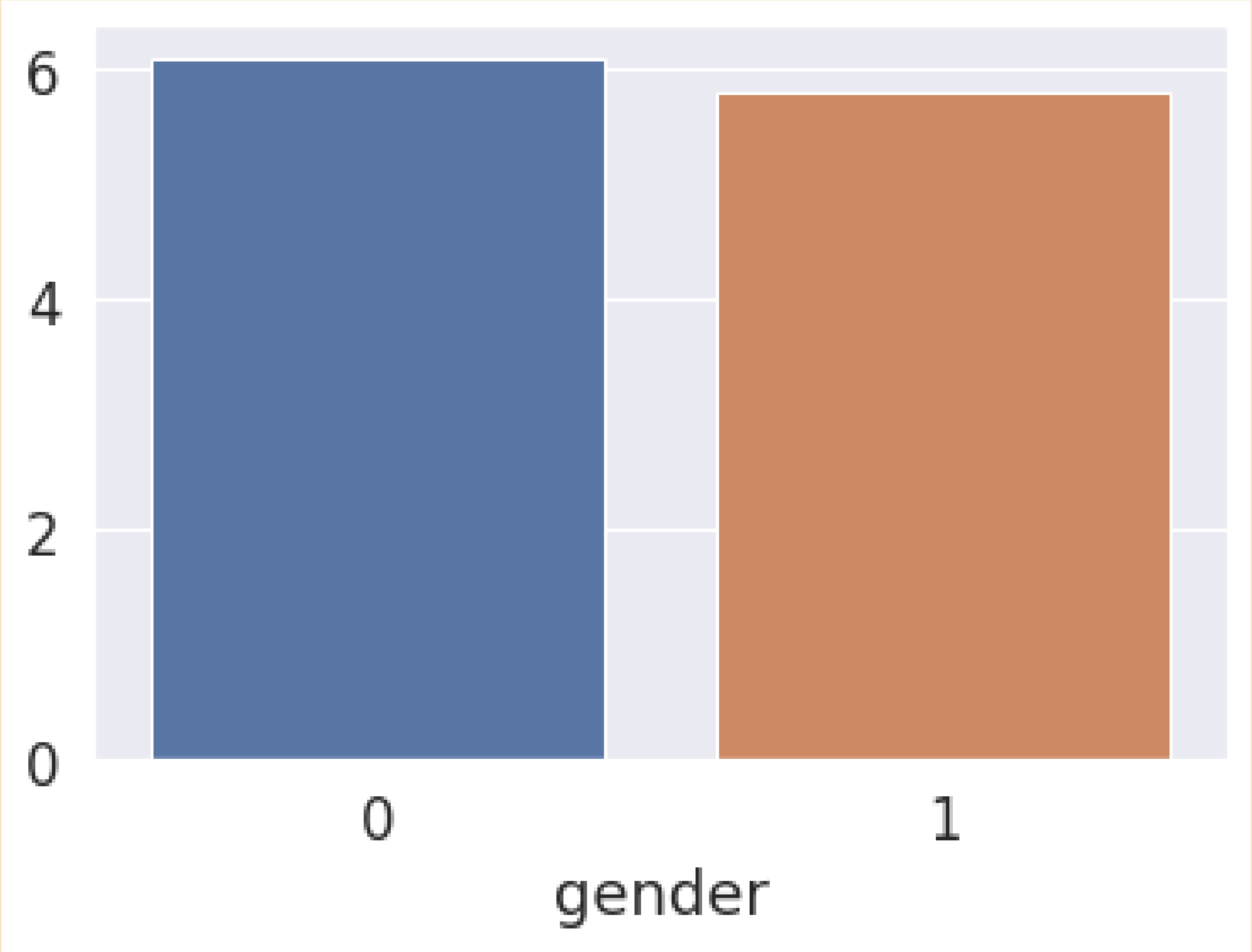
kde plot for lips thin and long hair with respective to gender.



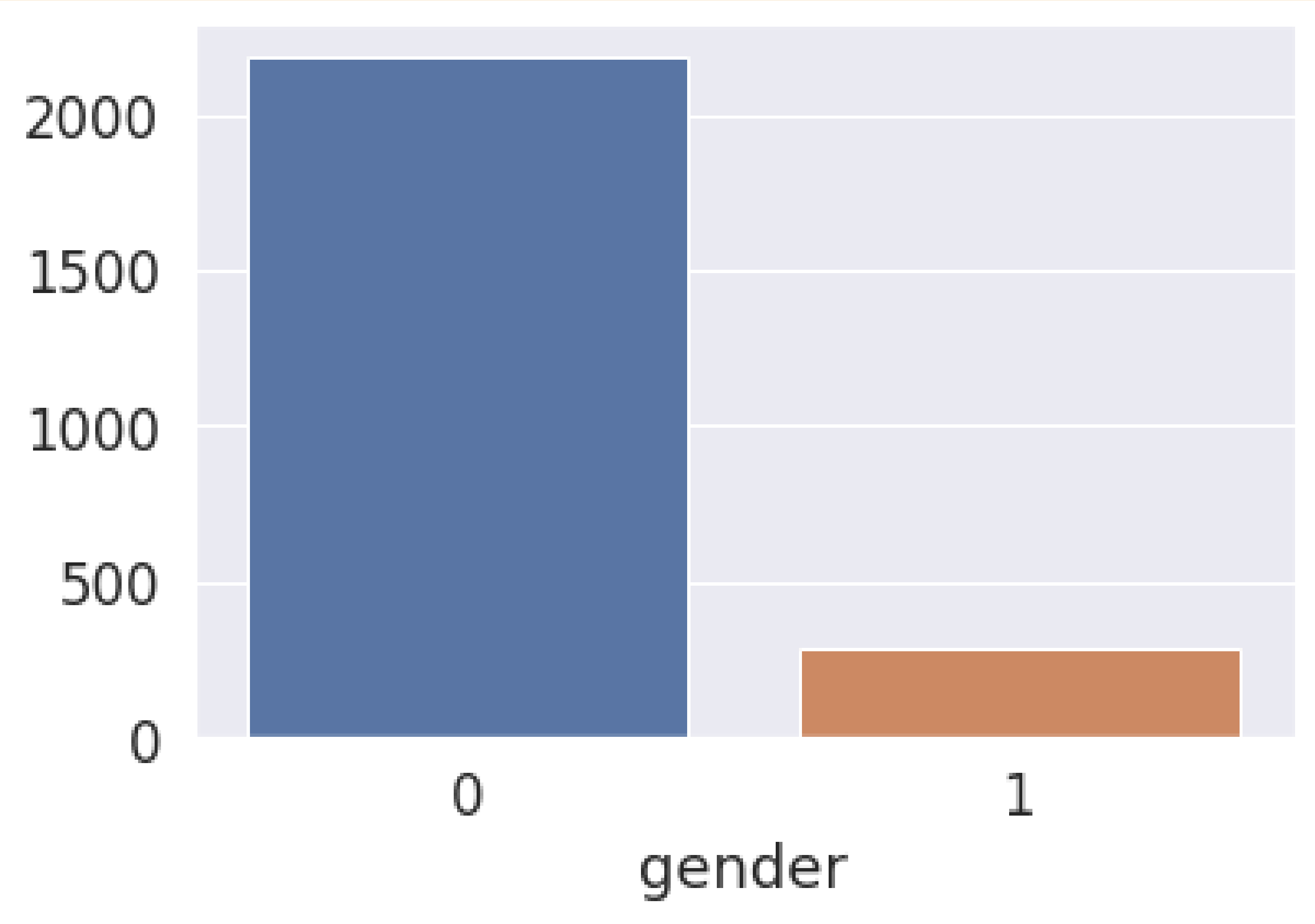
kde plot for distance from nose to lip long with respective to gender.

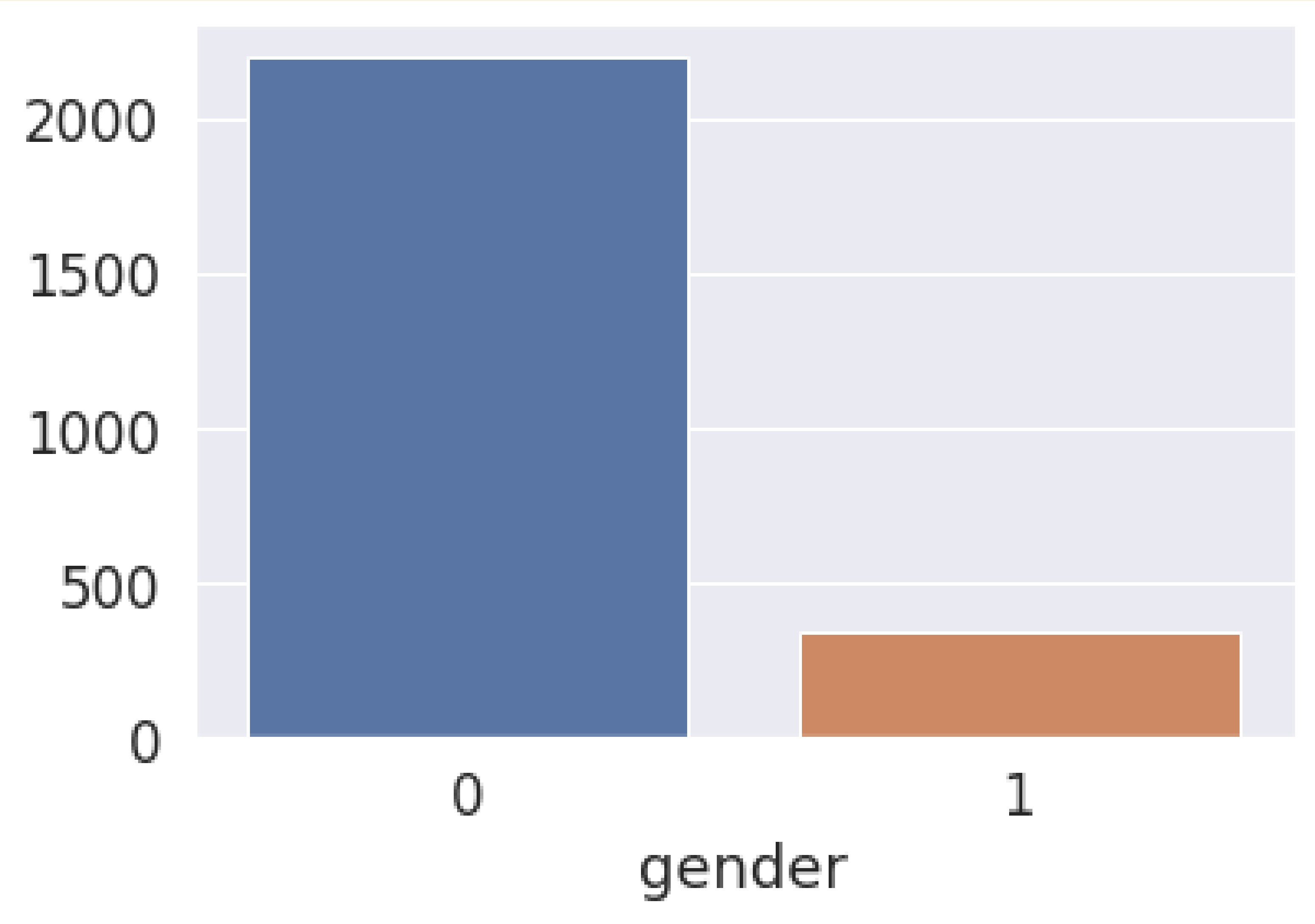
BAR PLOT



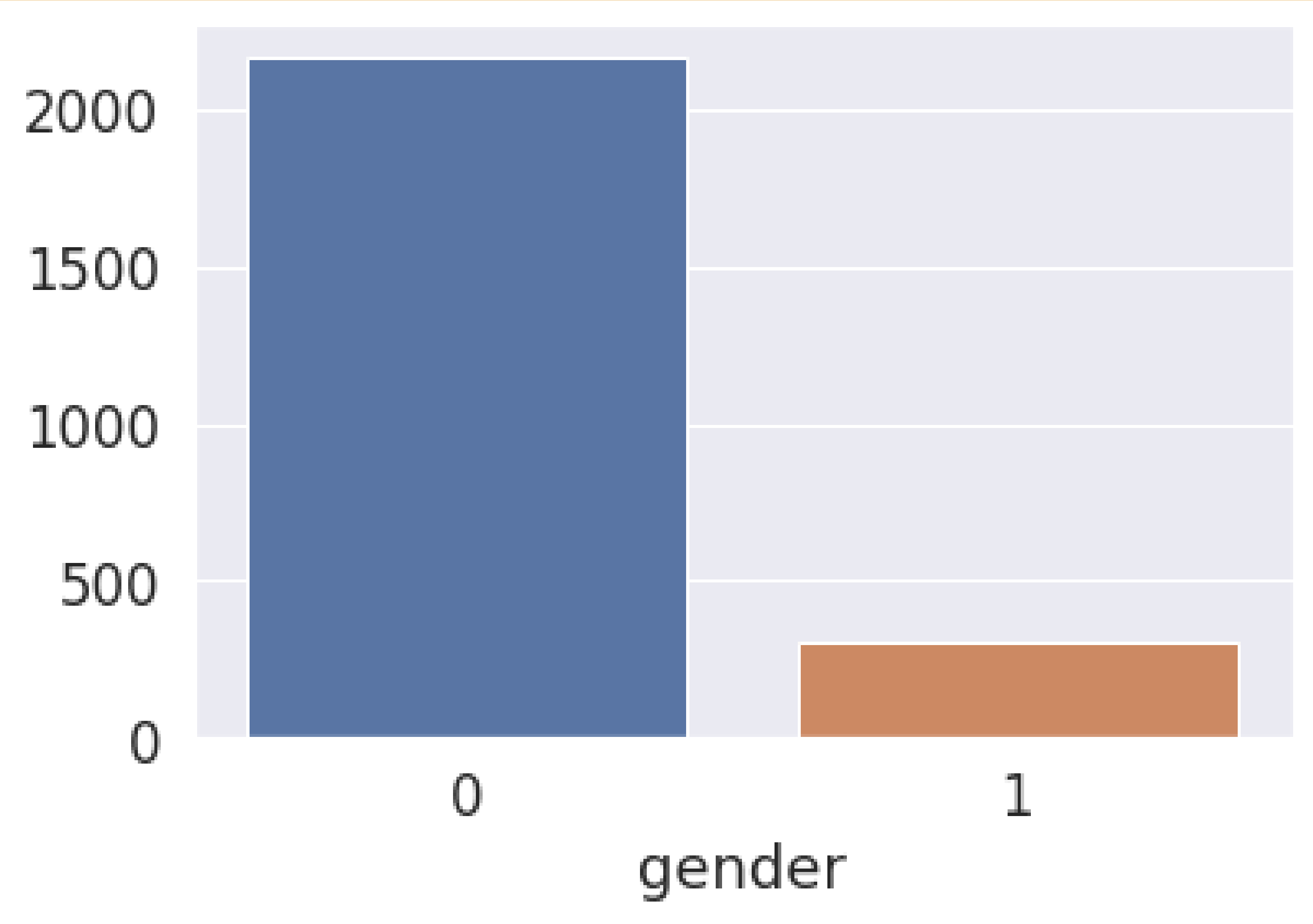


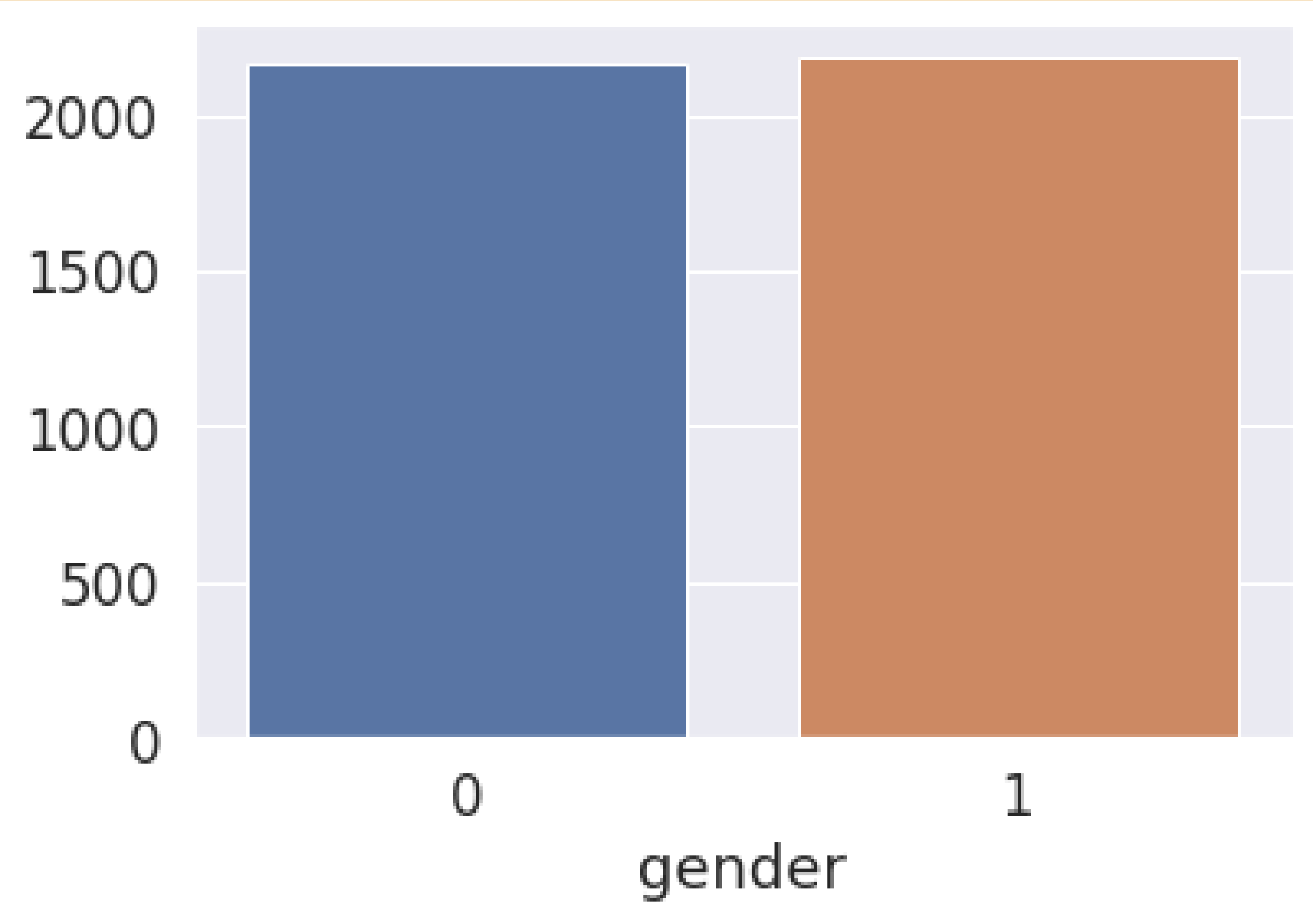
Bar plot for forehead width in cm and forehead height in cm with respective to gender.



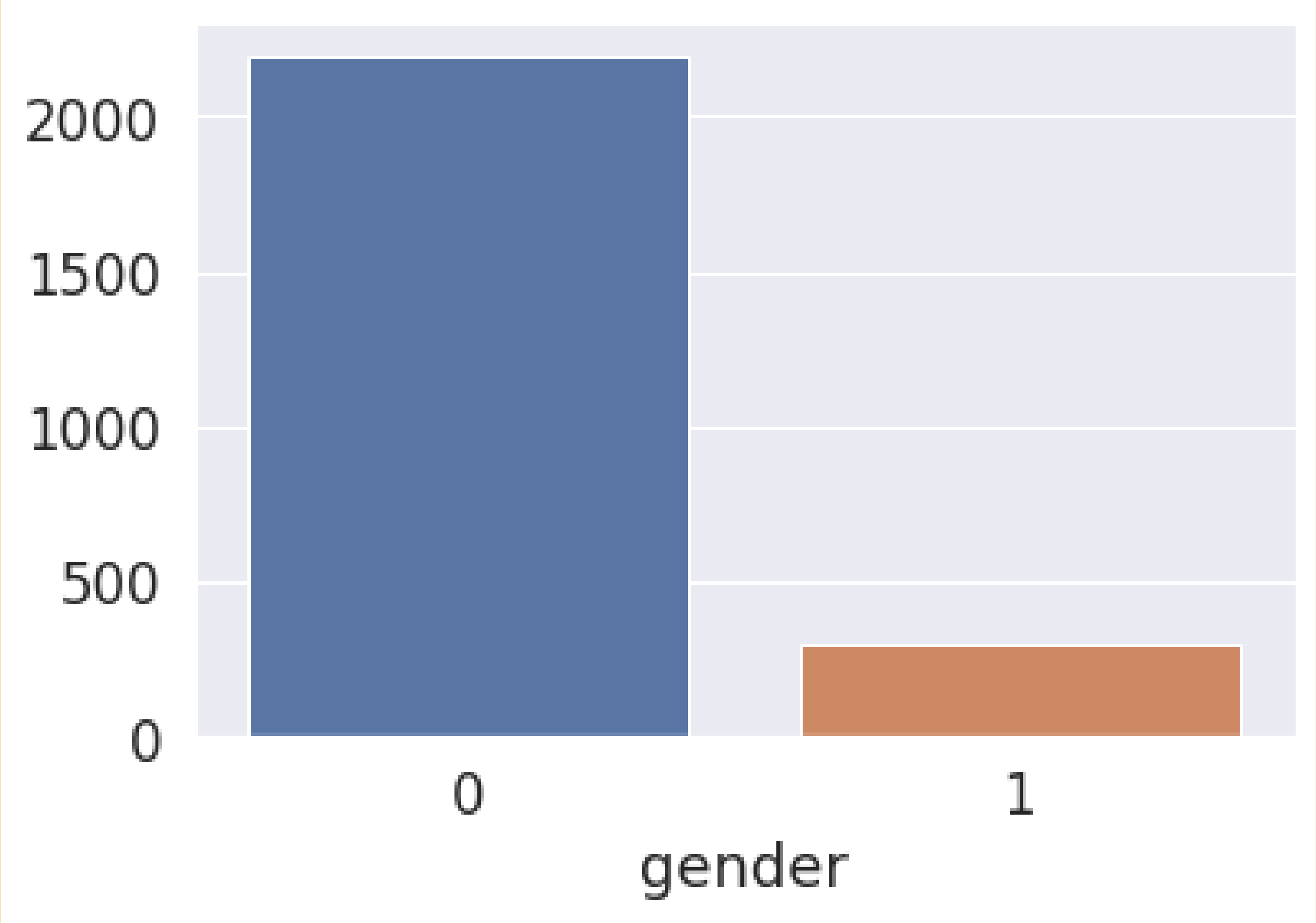


Bar plot for nose wide and nose long with respective to gender.



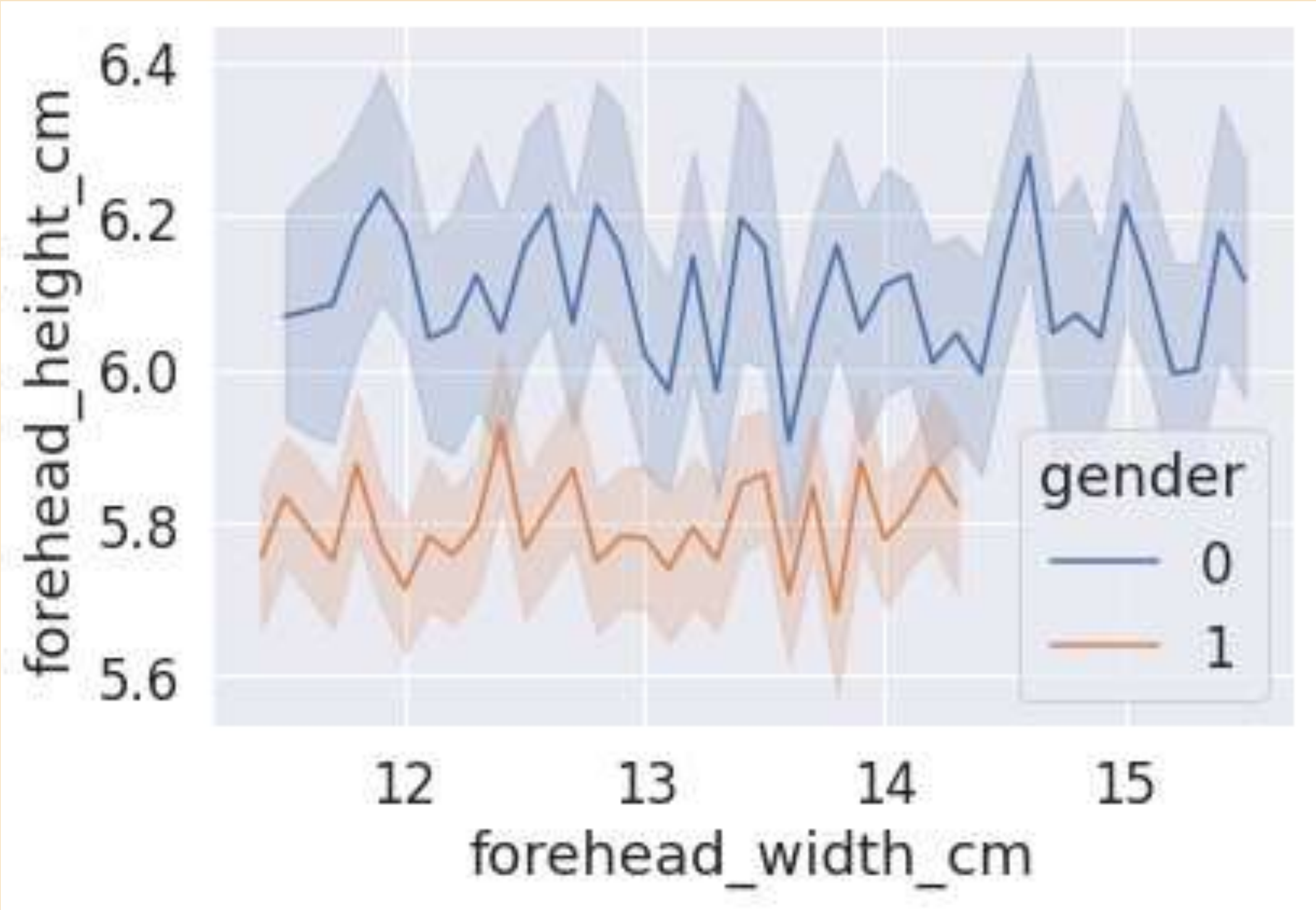


Bar plot for lips thin and long hair with respective to gender.

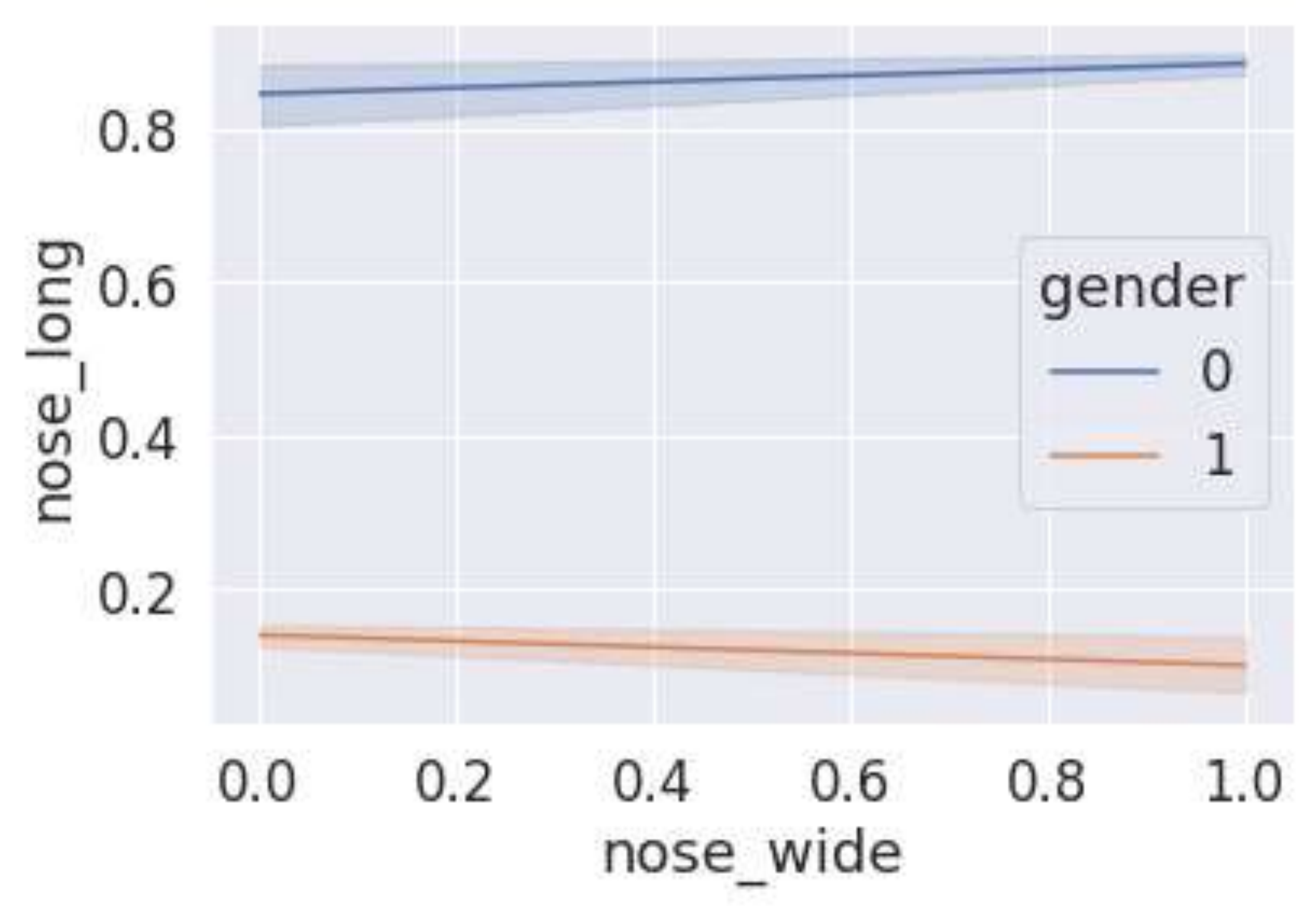


Bar plot for distance from nose to lip long with respective to gender.

LINE PLOT



Line plot for forehead width in cm and forehead height in cm with respective to gender.



Line plot for nose wide and nose long with respective to gender.

FITTING LOGISTIC REGRESSION

```
[ ] X = data.drop("gender",axis=1)
    y = data["gender"]
```

MODEL-1

```
[ ] X_train, X_test, y_train, y_test = train_test_split(X, y,train_size = 0.2, random_state=55)

[ ] from sklearn.linear_model import LogisticRegression     #applying logistic regression

    logreg= LogisticRegression(C=1e9)
    logreg.fit(X_train,y_train)

    LogisticRegression(C=1000000000.0)

[ ] y_pred = logreg.predict(X_test)
    y_pred

    array([0, 1, 1, ..., 0, 1, 0])

[ ] from sklearn.metrics import accuracy_score
    accuracy_score(y_test,y_pred)

    0.9665083729067733
```

MODEL-2

```
[ ] X_train, X_test, y_train, y_test = train_test_split(X, y,train_size = 0.3, random_state=55)

[ ] from sklearn.linear_model import LogisticRegression     #applying logistic regression

    logreg= LogisticRegression(C=1e9)
    logreg.fit(X_train,y_train)

    LogisticRegression(C=1000000000.0)

[ ] y_pred = logreg.predict(X_test)
    y_pred

    array([0, 1, 1, ..., 0, 1, 0])

[ ] from sklearn.metrics import accuracy_score
    accuracy_score(y_test,y_pred)

    0.967723507569266
```



```
[ ] X_train, X_test, y_train, y_test = train_test_split(X, y,train_size = 0.4, random_state=55)

[ ] logreg= LogisticRegression(C=1e9)
    logreg.fit(X_train,y_train)

    LogisticRegression(C=1000000000.0)

[ ] y_pred = logreg.predict(X_test)
    y_pred

    array([0, 1, 1, ..., 1, 0, 0])

[ ] accuracy_score(y_test,y_pred)

    0.9670109963345551
```

MODEL-3

MODEL-4

```
[ ] X_train, X_test, y_train, y_test = train_test_split(X, y,train_size = 0.1, random_state=55)

[ ] logreg= LogisticRegression(C=1e9)
    logreg.fit(X_train,y_train)

    LogisticRegression(C=1000000000.0)

[ ] y_pred = logreg.predict(X_test)
    y_pred

    array([0, 1, 1, ..., 1, 1, 1])

[ ] accuracy_score(y_test,y_pred)

    0.9660075538769163
```

```
[ ] X_train, X_test, y_train, y_test = train_test_split(X, y,train_size = 0.25, random_state=55)

[ ] logreg= LogisticRegression(C=1e9)
    logreg.fit(X_train,y_train)

    LogisticRegression(C=1000000000.0)

[ ] y_pred = logreg.predict(X_test)
    y_pred

    array([0, 1, 1, ..., 1, 0, 1])

[ ] accuracy_score(y_test,y_pred)

    0.9658757664622767
```

MODEL-5

K-FOLD CROSS VALIDATION

```
[ ] from sklearn.linear_model import LogisticRegression
    from sklearn.model_selection import cross_val_score,KFold,ShuffleSplit

    logreg=LogisticRegression()
    cv = KFold(n_splits=5, random_state=0, shuffle=True)
```

```
[ ] scores = cross_val_score(logreg, X, y, scoring='neg_mean_absolute_error')
```

```
[ ] from numpy import mean
    print(mean(scores))

    -0.03179220779220779
```

```
[ ] print("Avg accuracy: {}".format(scores.mean()))

    Avg accuracy: -0.03179220779220779
```

NEURAL NETWORK

ADAM

```
[ ] import tensorflow as tf
    tf.random.set_seed(55)
    model= tf.keras.Sequential([

                                tf.keras.layers.Dense(5, activation='relu'),
                                tf.keras.layers.Dense(2, activation='relu'),
                                tf.keras.layers.Dense(1, activation='sigmoid')
    ])
```

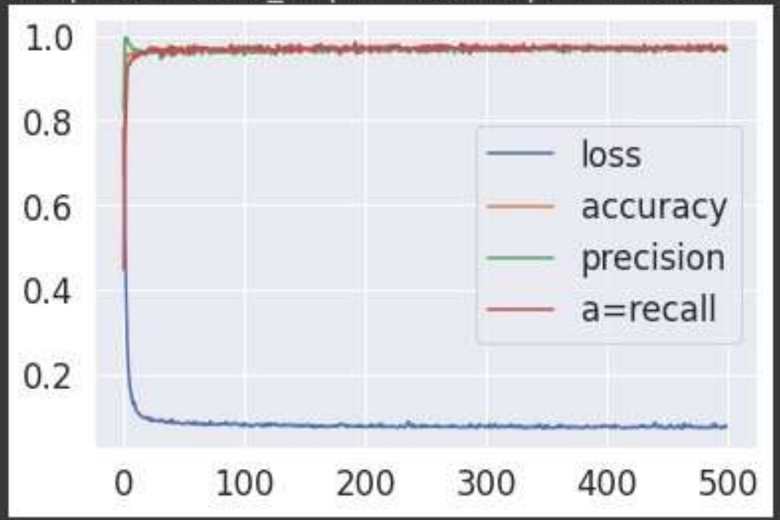
```
[ ] model.compile(loss= tf.keras.losses.binary_crossentropy,
                  optimizer= tf.keras.optimizers.Adam(lr=0.01),
                  metrics= [tf.keras.metrics.BinaryAccuracy(name='accuracy'),
                            tf.keras.metrics.Precision(name='precision'),
                            tf.keras.metrics.Recall(name='a=recall')
                  ]
                  )
```

```
[ ] history= model.fit(X_train, y_train, epochs= 500, verbose=0)
```



```
[ ] pd.DataFrame(history.history).plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f946897fad0>
```




```
[ ] model.evaluate(X_test, y_test)
```

```
110/110 [==============================] - 1s 2ms/step - loss: 0.0768 - accuracy: 0.9669 - precision: 0.9593 - a=recall: 0.
[0.07677220553159714,
 0.9668666124343872,
 0.9592529535293579,
 0.9746981263160706]
```

SGD

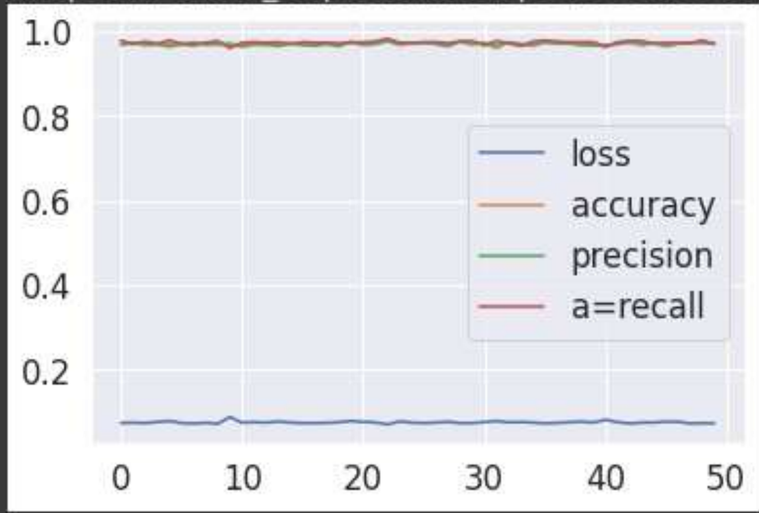
```
[ ] import tensorflow as tf
    tf.random.set_seed(55)
    model1= tf.keras.Sequential([
                                tf.keras.layers.Dense(5, activation='relu'),
                                tf.keras.layers.Dense(2, activation='relu'),
                                tf.keras.layers.Dense(1, activation='sigmoid')
    ])
```

```
[ ] model1.compile(loss= tf.keras.losses.binary_crossentropy,
                   optimizer= tf.keras.optimizers.SGD(lr=0.01),
                   metrics= [tf.keras.metrics.BinaryAccuracy(name='accuracy'),
                             tf.keras.metrics.Precision(name='precision'),
                             tf.keras.metrics.Recall(name='a=recall')
                   ]
                   )
```

```
[ ] history= model.fit(X_train, y_train, epochs= 50, verbose=0)
```

```
[ ] pd.DataFrame(history.history).plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f94691a9f10>
```




```
[ ] model1.evaluate(X_test, y_test)
```

```
110/110 [==============================] - 1s 2ms/step - loss: 1.5209 - accuracy: 0.5033 - precision: 0.0000e+00 - a=recal
[1.5209084749221802, 0.503284752368927, 0.0, 0.0]
```

BAGGING

BAGGING CLASSIFIER

```
[ ]  from sklearn import datasets
     from sklearn.model_selection import train_test_split
     from sklearn.metrics import accuracy_score
     from sklearn.model_selection import cross_val_score
     from sklearn.ensemble import BaggingClassifier
```

```
[ ]  clf = BaggingClassifier(n_estimators = 100, random_state = 42)
```

```
[ ]  a = clf.fit(X_train, y_train)
```

```
[ ]  y_pred = a.predict(X_test)
```

```
[ ]  accuracy = accuracy_score(y_test,y_pred)
     accuracy

     0.9696243005595524
```

DECISION TREE

```
[ ]  from sklearn.tree import DecisionTreeClassifier
```

```
[ ]  tree = DecisionTreeClassifier(max_depth=7, random_state=50)
```

```
[ ]  b = tree.fit(X_train, y_train)
```

```
[ ]  y_pred = b.predict(X_test)
```

```
[ ]  accuracy = accuracy_score(y_test,y_pred)
     accuracy

     0.9712230215827338
```

RANDOM FOREST

```
[ ] from sklearn.ensemble import RandomForestClassifier

[ ] rf = RandomForestClassifier(n_estimators =100, random_state = 50)

[ ] c = rf.fit(X_train, y_train)

[ ] y_pred = c.predict(X_test)

[ ] accuracy = accuracy_score(y_test,y_pred)
    accuracy

    0.9776179056754596
```

BOOSTING

Ada Boost

```
[ ] from sklearn.ensemble import AdaBoostClassifier

    adaboost = AdaBoostClassifier(n_estimators=500, learning_rate=0.01, random_state=50)

[ ] d = adaboost.fit(X_train, y_train)

[ ] y_pred = d.predict(X_test)

[ ] accuracy = accuracy_score(y_test,y_pred)
    accuracy

    0.9808153477218226
```

GB

```
[ ] from sklearn.ensemble import GradientBoostingClassifier

    grad_boost= GradientBoostingClassifier(max_depth=5,learning_rate=0.01,random_state=0,n_estimators=1000)
```

```
[ ] e = grad_boost.fit(X_train, y_train)
```

```
[ ] y_pred = e.predict(X_test)
```

```
[ ] accuracy = accuracy_score(y_test,e.predict(X_test))
    accuracy

    0.9776179056754596
```

XGBoost

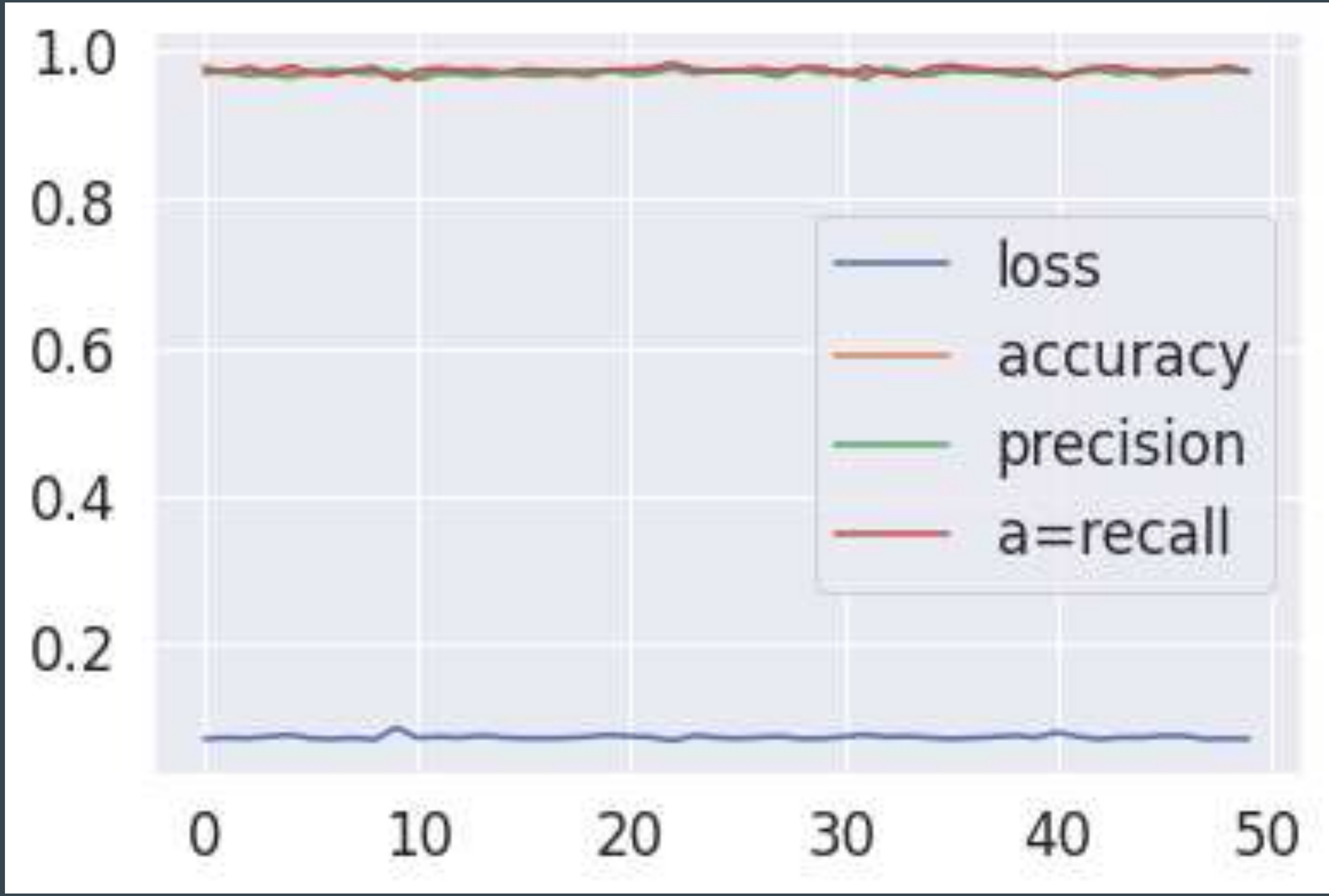
```
[188] import xgboost as xgb
      from xgboost import XGBClassifier
      xgb_boost=xgb.XGBClassifier(random_state=50,learning_rate=0.01,n_estimators=700)

[193] f = xgb_boost.fit(X_train, y_train)

[194] y_pred = f.predict(X_test)

[195] accuracy = accuracy_score(y_test,y_pred)
      accuracy

      0.9808153477218226
```



Epochs ---->