[4]:
```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

```
/kaggle/input/gender-classification-dataset/gender_classification_v7.csv
```

[5]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

import warnings
warnings.filterwarnings('ignore')
```

[6]:
```
df=pd.read_csv('/kaggle/input/gender-classification-dataset/gender_classification_v7.csv')
```

[7]:
```
df.head()
```

[7]:

	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

[8]:
```
df.shape
```

[8]:
```
(5001, 8)
```

NO NULL VALUES

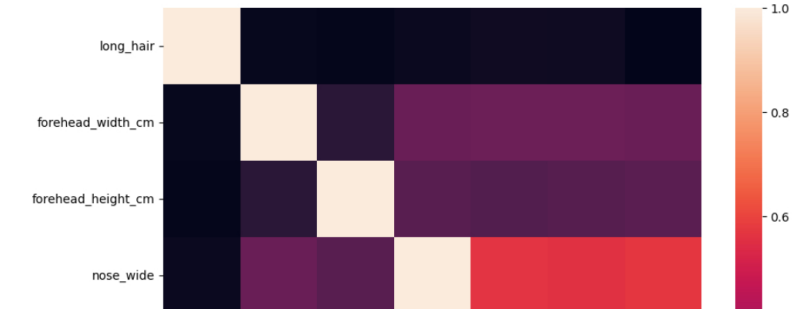
[9]:
```
total_null = df.isnull().sum().sort_values(ascending = False)
percent = ((df.isnull().sum()/df.isnull().count())*100).sort_values(ascending = False)
print("Total records = ", df.shape[0])

missing_data = pd.concat([total_null,percent.round(2)],axis=1,keys=['Total Missing','In Percent'])
missing_data.head(16)
```

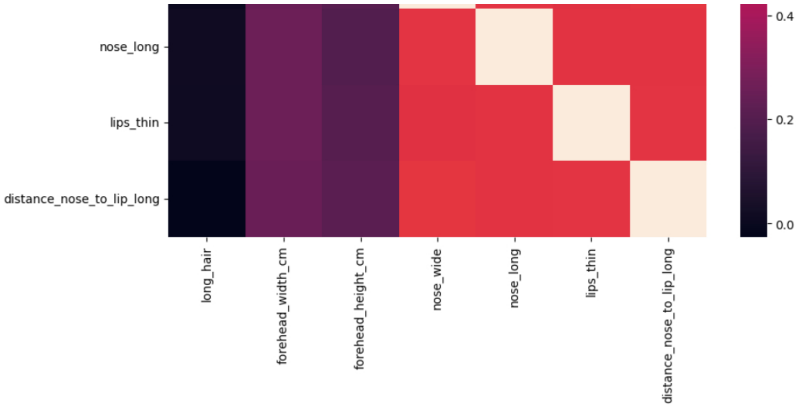
```
Total records =  5001
```

[9]:

	Total Missing	In Percent
long_hair	0	0.0
forehead_width_cm	0	0.0
forehead_height_cm	0	0.0
nose_wide	0	0.0
nose_long	0	0.0
lips_thin	0	0.0
distance_nose_to_lip_long	0	0.0
gender	0	0.0

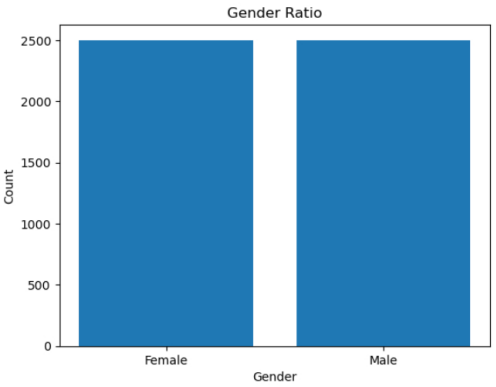
[10]:
```
import seaborn as sns
corr = df.corr()
plt.figure(figsize=(10,8))
sns.heatmap(corr)
plt.show()
```





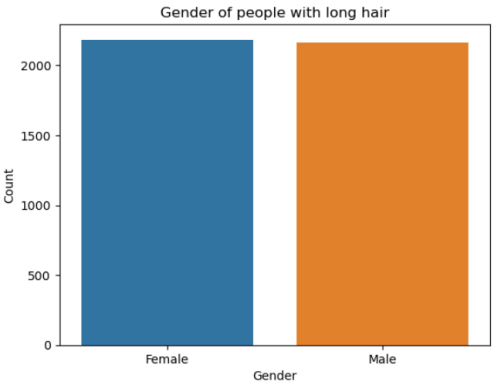
[11]:

```
gender_count = df["gender"].value_counts()

plt.bar(gender_count.index, gender_count.values)
plt.title("Gender Ratio")
plt.xlabel("Gender")
plt.ylabel("Count")
plt.show()
```



[12]:

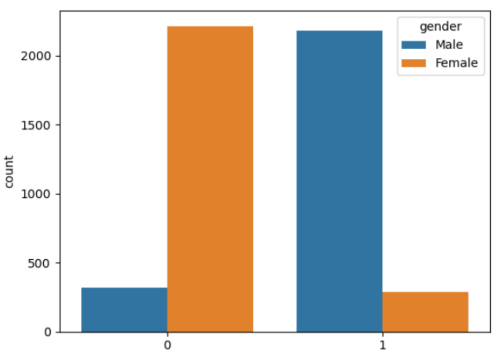
```
long_hair = df[df["long_hair"] == 1]


gender_count = long_hair["gender"].value_counts()


sns.barplot(x=gender_count.index, y=gender_count.values)
plt.title("Gender of people with long hair")
plt.xlabel("Gender")
plt.ylabel("Count")
plt.show()
```



[13]:

```
sns.countplot(x='nose_wide', hue='gender', data=df)
```

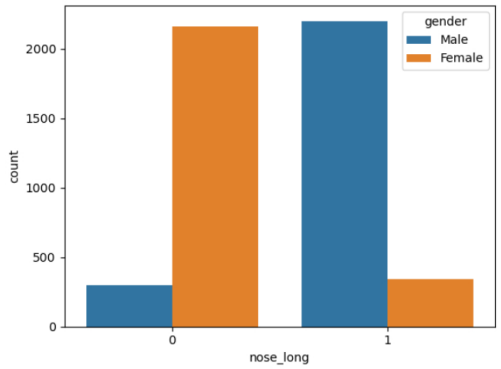
[13]: <AxesSubplot:xlabel='nose_wide', ylabel='count'>



nose_wide

[14]:
```
sns.countplot(x='nose_long', hue='gender', data=df)
```

[14]:
```
<AxesSubplot:xlabel='nose_long', ylabel='count'>
```



Linear Regression

[15]:
```
from sklearn.linear_model import LinearRegression
```

[16]:
```
features1=df.copy()
```

[17]:
```
features1['gender'] = (features1['gender'] == 'Male').astype(int)
```

[18]:
```
X_train, X_test, Y_train, Y_test = train_test_split(features1.drop('gender', axis=1), features1['gender'], test_size=0.2)
lr = LinearRegression()
lr.fit(X_train, Y_train)
Y_pred = lr.predict(X_test)
Y_pred_binary = []

for pred in Y_pred:
    if pred >= 0.5:
        Y_pred_binary.append(1)
    else:
        Y_pred_binary.append(0)

accuracy1 = accuracy_score(Y_test,Y_pred_binary)
accuracy1*100
```

[18]:
```
96.7032967032967
```

Logistic Regression

[19]:
```
from sklearn.linear_model import LogisticRegression
```

[20]:
```
features2=df.copy()
```

[25]:
```
X_train, X_test, Y_train, Y_test = train_test_split(features2.drop('gender', axis=1), features2['gender'], test_size=0.2)
```

[26]:
```
lor = LogisticRegression()
lor.fit(X_train,Y_train)
```

[26]:
```
LogisticRegression()
```

[27]:
```
Y_pred = lor.predict(X_test)
```

[29]:
```
accuracy2 = accuracy_score(Y_test , Y_pred)
accuracy2*100
```

[29]:
```
95.90409590409591
```

Decision Tree

[30]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
```

[31]:
```
features3=df.copy()
```

[32]:
```
X_train, X_test, Y_train, Y_test = train_test_split(features3.drop('gender', axis=1), features3['gender'], test_size=0.2)
```

[33]:
```
dectree = DecisionTreeClassifier()
dectree.fit(X_train,Y_train)
```

[33]:
```
DecisionTreeClassifier()
```

[34]:
```
Y_pred = dectree.predict(X_test)
accuracy3 = accuracy_score(Y_test,Y_pred)
accuracy3*100
```

[34]:
```
95.7042957042957
```

Random Forest

[35]:
```
from sklearn.ensemble import RandomForestClassifier
```

[36]:
```
features4 = df.copy()
```

[54]:
```
x_train, x_test, y_train, y_test = train_test_split(features4.drop('gender', axis=1), features4['gender'], test_size=0.2)

ranfo = RandomForestClassifier()
ranfo.fit(x_train, y_train)

y_pred = ranfo.predict(x_test)

accuracy4 = accuracy_score(y_test, y_pred)
accuracy4*100
```

[54]:
```
96.40359640359641
```

Random Forest using GridSearchCV

[55]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
```

[62]:
```
param_grid1 = {
    'n_estimators': [50, 100, 200],
    'max_depth': [5, 10, 15],
    'min_samples_split': [2, 5, 10]
}
```

[63]:
```
grid_search = GridSearchCV(estimator=ranfo, param_grid=param_grid1, cv=5)
grid_search.fit(x_train, y_train)
```

[63]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(),
             param_grid={'max_depth': [5, 10, 15],
                         'min_samples_split': [2, 5, 10],
                         'n_estimators': [50, 100, 200]})
```

[66]:
```
print('Best Hyperparameters:', grid_search.best_params_)
y_pred = grid_search.predict(x_test)
accuracy_5 = accuracy_score(y_test, y_pred)
accuracy_5*100
```

```
Best Hyperparameters: {'max_depth': 5, 'min_samples_split': 5, 'n_estimators': 200}
```

[66]:
```
96.90309690309691
```

KNN

[70]:
```
features6=df.copy()
```

[71]:
```
from sklearn.neighbors import KNeighborsClassifier
```

[73]:
```
X = features6.drop('gender',axis=1)
Y= features6['gender']
```

[74]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
```

[75]:
```
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train,Y_train)
```

[75]:
```
KNeighborsClassifier(n_neighbors=3)
```

[76]:
```
Y_pred = knn.predict(X_test)
```

[77]:
```
accuracy6 = accuracy_score(Y_test,Y_pred)
accuracy6*100
```

[77]:
```
96.40359640359641
```

KNN using GridSearchCV

[78]:
```
features7=df.copy()
```

[79]:
```
X = features7.drop('gender',axis=1)
Y = features7['gender']
```

[80]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
```

[82]:
```
param_dist = {'n_neighbors': range(1, 21),
              'weights': ['uniform', 'distance'],
              'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
              'p': [1, 2]}
```

[83]:
```
grid_search = GridSearchCV(knn,param_dist, cv=5,n_jobs=-1)
grid_search.fit(X_train, Y_train)
```

[83]:
```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(n_neighbors=3), n_jobs=-1,
             param_grid={'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
                         'n_neighbors': range(1, 21), 'p': [1, 2],
                         'weights': ['uniform', 'distance']})
```

[86]:
```
print(grid_search.best_params_)
```

```
{'algorithm': 'auto', 'n_neighbors': 16, 'p': 2, 'weights': 'uniform'}
```

[87]:
```
Y_pred = grid_search.predict(X_test)
```

[96]:
```
accuracy7 = accuracy_score(Y_test,Y_pred)
accuracy7t = grid_search.best_score_
accuracy7*100
```

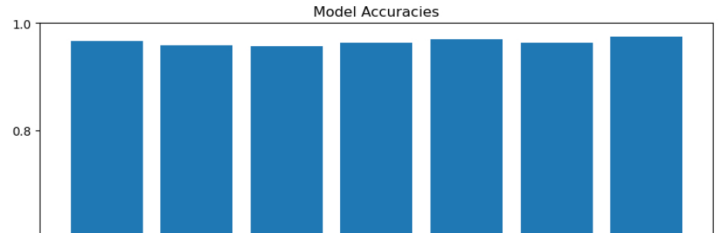
[96]:
```
96.7032967032967
```

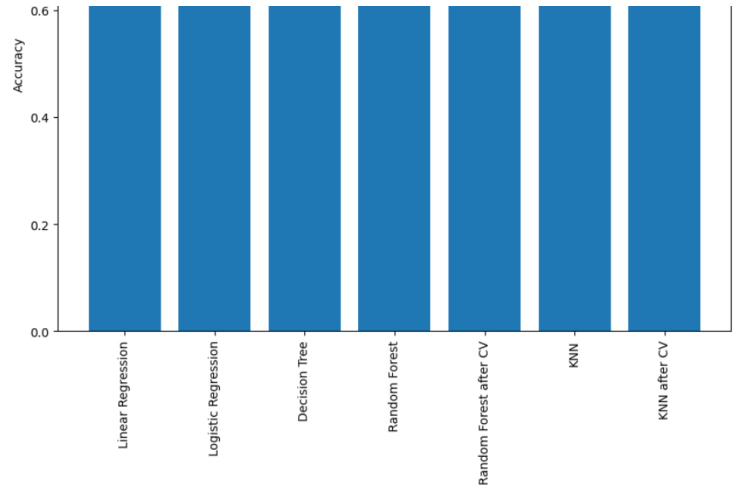
[97]:
```
accuracy7t
```

[97]:
```
0.975
```

Graphs

```
Algomodels = ['Linear Regression','Logistic Regression','Decision Tree','Random Forest','Random Forest after CV','KNN','KNN after CV']
Accuracies = [accuracy1,accuracy2,accuracy3,accuracy4,accuracy_5,accuracy6,accuracy7t]
plt.figure(figsize=(10,8))
plt.bar(Algomodels,Accuracies)
plt.title('Model Accuracies')
plt.ylabel('Accuracy')
plt.ylim(0,1)
plt.xticks(rotation=90)
plt.show()
```





+ Code + Markdown