[1]:
```
# 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
```

+ Code + Markdown

[2]:
```
df = pd.read_csv("/kaggle/input/gender-classification-dataset/gender_classification_v7.csv")
```

[3]:
```
df.shape
```

[3]:
```
(5001, 8)
```

[4]:
```
df.head()
```

[4]:

	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

[5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5001 entries, 0 to 5000
Data columns (total 8 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   long_hair                 5001 non-null   int64
 1   forehead_width_cm         5001 non-null   float64
 2   forehead_height_cm        5001 non-null   float64
 3   nose_wide                 5001 non-null   int64
 4   nose_long                 5001 non-null   int64
 5   lips_thin                 5001 non-null   int64
 6   distance_nose_to_lip_long  5001 non-null   int64
 7   gender                    5001 non-null   object
dtypes: float64(2), int64(5), object(1)
memory usage: 312.7+ KB
```

[6]:
```
df.describe()
```

[6]:

	long_hair	forehead_width_cm	forehead_height_cm	nose_wide	nose_long	lips_thin	distance_nose_to_lip_long
count	5001.000000	5001.000000	5001.000000	5001.000000	5001.000000	5001.000000	5001.000000
mean	0.869626	13.181484	5.946311	0.493901	0.507898	0.493101	0.498900
std	0.336748	1.107128	0.541268	0.500013	0.499988	0.500002	0.500049
min	0.000000	11.400000	5.100000	0.000000	0.000000	0.000000	0.000000
25%	1.000000	12.200000	5.500000	0.000000	0.000000	0.000000	0.000000
50%	1.000000	13.100000	5.900000	0.000000	1.000000	0.000000	0.000000
75%	1.000000	14.000000	6.400000	1.000000	1.000000	1.000000	1.000000
max	1.000000	15.500000	7.100000	1.000000	1.000000	1.000000	1.000000

[7]:
```
df.isnull().sum()
```

[7]:
```
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
```

[8]:
```
df.describe()
```

[8]:

	long_hair	forehead_width_cm	forehead_height_cm	nose_wide	nose_long	lips_thin	distance_nose_to_lip_long
count	5001.000000	5001.000000	5001.000000	5001.000000	5001.000000	5001.000000	5001.000000
mean	0.869626	13.181484	5.946311	0.493901	0.507898	0.493101	0.498900
std	0.336748	1.107128	0.541268	0.500013	0.499988	0.500002	0.500049
min	0.000000	11.400000	5.100000	0.000000	0.000000	0.000000	0.000000
25%	1.000000	12.200000	5.500000	0.000000	0.000000	0.000000	0.000000
50%	1.000000	13.100000	5.900000	0.000000	1.000000	0.000000	0.000000
75%	1.000000	14.000000	6.400000	1.000000	1.000000	1.000000	1.000000
max	1.000000	15.500000	7.100000	1.000000	1.000000	1.000000	1.000000

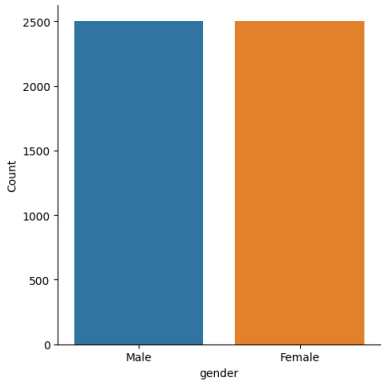
Libraries

[9]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

[10]:
```
plt.figure(figsize=(5,5))
sns.catplot(x='gender', data = df, kind = 'count')
plt.xlabel('gender')
plt.ylabel('Count')
plt.show()
```

```
<Figure size 500x500 with 0 Axes>
```

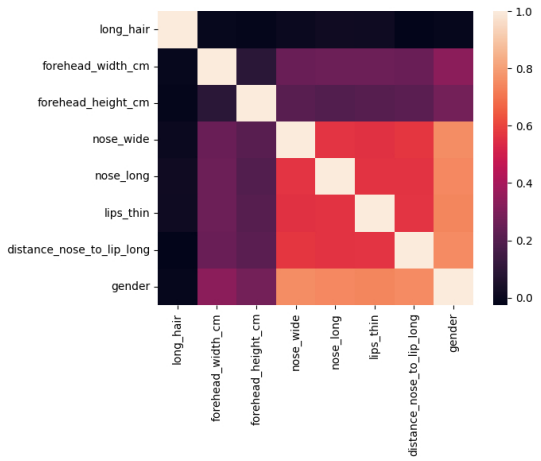


[11]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

[12]:
```
temp_df = df.copy()
temp_df['gender'] = le.fit_transform(temp_df['gender'])
```

[13]:
```
corr = temp_df.corr()
sns.heatmap(corr)
```

[13…
```
<AxesSubplot: >
```



[14]:
```
target = temp_df['gender']
temp_df.drop(columns=['gender'],inplace=True)
```

[15]:
```
temp_df.head()
```

[15…

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

[16]:
```
x_t,x_te,y_t,y_te= train_test_split(temp_df,target,train_size=0.8,random_state=42)
```

Linear Regression

[17]:
```
from sklearn.linear_model import LinearRegression
linearr = LinearRegression()
reg = linearr.fit(x_t,y_t)
```

[18]:
```
linearscore=reg.score(x_te,y_te)
linearscore
```

[18…
```
0.8323674916343946
```

[19]:
```
X = df.drop("gender", axis="columns")
```

[20]:
```
Y = df['gender']
```

[21]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
```

Logistic regression

[23]:
```
logisticr = LogisticRegression()
lr = logisticr.fit(X_train,Y_train)
```

[24]:
```
X_test_prediction = lr.predict(X_test)
X_test_prediction
```

[24…
```
array(['Male', 'Female', 'Female', ..., 'Female', 'Male', 'Female'],
      dtype=object)
```

[25]:
```
accuracy_score(X_test_prediction, Y_test)
```

[25…
```
0.9600399600399601
```

[26]:
```
lrscore = lr.score(X_test,Y_test)
lrscore
```

[26…
```
0.9600399600399601
```

Decision Tree

[27]:
```
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X_train,Y_train)
accuracyscoretree = clf.score(X_test,Y_test)
accuracyscoretree
```

[27…
```
0.9530469530469531
```

Random Forest Classifier

[28]:
```
randomclf = RandomForestClassifier()
randomclf.fit(X_train,Y_train)
```

[28…
```
RandomForestClassifier()
```

[29]:
```
accuracyscoreforest = randomclf.score(X_test,Y_test)
accuracyscoreforest
```

[29…
```
0.958041958041958
```

KNN Model

[30]:
```
knn=KNeighborsClassifier()
knn.fit(X_train,Y_train)
```

[30…
```
KNeighborsClassifier()
```

[31]:
```
neigh = KNeighborsClassifier(n_neighbors=3)
```

```
neigh.fit(X_train, Y_train)
```

[31…]
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

[32]:
```
knnaccuracy = neigh.score(X_test,Y_test)
knnaccuracy
```

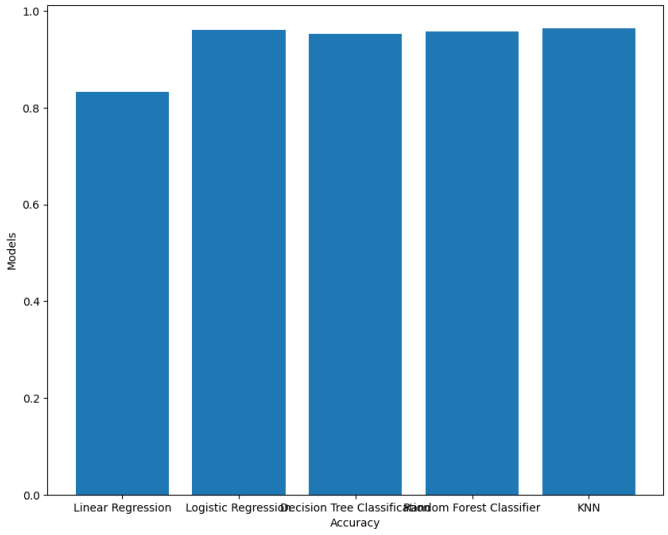
[32…] 0.964035964035964

[33]:
```
neigh1 = KNeighborsClassifier(n_neighbors=6)
neigh1.fit(X_train, Y_train)
knnaccuracy1 = neigh.score(X_test,Y_test)
knnaccuracy1
```

[33…] 0.964035964035964

[34]:
```
names=['Linear Regression','Logistic Regression' ,'Decision Tree Classification','Random Forest Classifier','KNN']
acc=[linearscore,lrscore,accuracyscoretree,accuracyscoreforest,knnaccuracy]
plt.figure(figsize=(10,8))
graph = plt.bar(names,acc)
plt.xlabel('Accuracy')
plt.ylabel('Models')
```

[34…] Text(0, 0.5, 'Models')



Hyper-Parameter

[35]:
```
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV
```

[36]:
```
grid={
    'n_neighbors':[3,5,7],
    'weights':['uniform','distance'],
    'algorithm':['auto','ball_tree','kd_tree'],
    'metric':['manhattan','euclidian','minkowski']
}
```

```
gr = GridSearchCV(estimator=knn,param_grid=grid).fit(X_train,Y_train)
gr = pd.DataFrame(gr.cv_results_)
gr.T
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/model_selection/_validation.py:378: FitFailedWarning:
90 fits failed out of a total of 270.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
90 fits failed with the following error:
Traceback (most recent call last):
  File "/opt/conda/lib/python3.10/site-packages/sklearn/model_selection/_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/opt/conda/lib/python3.10/site-packages/sklearn/neighbors/_classification.py", line 213, in fit
    self._validate_params()
  File "/opt/conda/lib/python3.10/site-packages/sklearn/base.py", line 600, in _validate_params
    validate_parameter_constraints(
  File "/opt/conda/lib/python3.10/site-packages/sklearn/utils/_param_validation.py", line 97, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'metric' parameter of KNeighborsClassifier must be a str among {'correlation', 'braycurtis', 'l2', 'haversine', 'nan_euclidean', 'precomputed', 'l1', 'rogerstanimoto', 'pyfunc', 'p', 'russellrao', 'infinity', 'matching', 'dice', 'sokalsneath', 'seuclidean', 'chebyshev', 'euclidean', 'wminkowski', 'cosine', 'cityblock', 'canberra', 'hamming', 'sqeuclidean', 'minkowski', 'jaccard', 'kulsinski', 'sokalmichener', 'yule', 'mahalanobis', 'manhattan'} or a callable. Got 'euclidian' instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/opt/conda/lib/python3.10/site-packages/sklearn/model_selection/_search.py:952: UserWarning: One or more of the test scores are non-finite: [0.96875 0.96775 0.96975 0.969   0.972   0.96975     nan     nan     nan
     nan     nan     nan 0.9665  0.9665  0.968   0.96725 0.9685  0.96875
 0.96825 0.96725 0.97    0.9695  0.971   0.96925     nan     nan     nan
     nan     nan     nan 0.96625 0.96625 0.968   0.96775 0.9685  0.96875
 0.96875 0.96775 0.96975 0.969   0.972   0.96975     nan     nan     nan
     nan     nan     nan 0.9665  0.9665  0.968   0.96725 0.9685  0.96875]
  warnings.warn(
```

[37…]

	0	1	2	3	4	5	6	7	8	9	…	44	45	46	47	48	49	50	51	52	53
mean_fit_time	0.008956	0.007527	0.007332	0.007479	0.007452	0.007373	0.000844	0.000726	0.000731	0.000725	…	0.000711	0.000732	0.000714	0.000719	0.007274	0.007299	0.007434	0.007226	0.007251	0.007169
std_fit_time	0.002257	0.000105	0.000084	0.000159	0.000046	0.000174	0.000111	0.00001	0.000012	0.000007	…	0.000011	0.00003	0.000014	0.000013	0.000259	0.000254	0.000205	0.000054	0.000086	0.000125
mean_score_time	0.033271	0.012647	0.031974	0.014769	0.034629	0.015653	0.0	0.0	0.0	0.0	…	0.0	0.0	0.0	0.0	0.029817	0.010839	0.030712	0.012189	0.031547	0.013215

std_score_time	0.001477	0.00061	0.000321	0.000505	0.000632	0.000323	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.000775	0.000288	0.000956	0.000367	0.000405	0.00023
param_algorithm	auto	auto	auto	auto	auto	auto	auto	auto	auto	auto	...	kd_tree	kd_tree	kd_tree	kd_tree	kd_tree	kd_tree	kd_tree	kd_tree	kd_tree	kd_tree
param_metric	manhattan	manhattan	manhattan	manhattan	manhattan	manhattan	euclidian	euclidian	euclidian	euclidian	...	euclidian	euclidian	euclidian	euclidian	minkowski	minkowski	minkowski	minkowski	minkowski	minkowski
param_n_neighbors	3	3	5	5	7	7	3	3	5	5	...	5	5	7	7	3	3	5	5	7	7
param_weights	uniform	distance	uniform	distance	uniform	distance	uniform	distance	uniform	distance	...	uniform	distance	uniform	distance	uniform	distance	uniform	distance	uniform	distance
params	{'algorithm': 'auto', 'metric': 'manhattan', '...	{'algorithm': 'auto', 'metric': 'manhattan', '...	{'algorithm': 'auto', 'metric': 'manhattan', '...	{'algorithm': 'auto', 'metric': 'manhattan', '...	{'algorithm': 'auto', 'metric': 'manhattan', '...	{'algorithm': 'auto', 'metric': 'manhattan', '...	{'algorithm': 'auto', 'metric': 'euclidian', '...	{'algorithm': 'auto', 'metric': 'euclidian', '...	{'algorithm': 'auto', 'metric': 'euclidian', '...	{'algorithm': 'auto', 'metric': 'euclidian', '...	...	{'algorithm': 'kd_tree', 'metric': 'euclidian'...	{'algorithm': 'kd_tree', 'metric': 'euclidian'...	{'algorithm': 'kd_tree', 'metric': 'euclidian'...	{'algorithm': 'kd_tree', 'metric': 'euclidian'...	{'algorithm': 'kd_tree', 'metric': 'minkowski'...	{'algorithm': 'kd_tree', 'metric': 'minkowski'...	{'algorithm': 'kd_tree', 'metric': 'minkowski'...	{'algorithm': 'kd_tree', 'metric': 'minkowski'...	{'algorithm': 'kd_tree', 'metric': 'minkowski'...	{'algorithm': 'kd_tree', 'metric': 'minkowski'...
split0_test_score	0.96125	0.96375	0.96125	0.96125	0.96125	0.96125	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	0.9575	0.96125	0.96125	0.96	0.9575	0.96
split1_test_score	0.975	0.97375	0.9775	0.97875	0.98375	0.9775	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	0.97625	0.97625	0.98	0.97875	0.98	0.9775
split2_test_score	0.96	0.96	0.965	0.96375	0.9675	0.9675	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	0.95875	0.96	0.95625	0.9575	0.96125	0.96375
split3_test_score	0.9725	0.96625	0.97125	0.96875	0.975	0.97125	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	0.9725	0.96875	0.97375	0.97125	0.9725	0.97
split4_test_score	0.975	0.975	0.97375	0.9725	0.9725	0.97125	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	0.9675	0.96625	0.96875	0.96875	0.97125	0.9725
mean_test_score	0.96875	0.96775	0.96975	0.969	0.972	0.96975	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	0.9665	0.9665	0.968	0.96725	0.9685	0.96875
std_test_score	0.006708	0.005777	0.005884	0.006245	0.007525	0.005327	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	0.007391	0.005831	0.0085	0.007722	0.008116	0.006225
rank_test_score	13	25	5	11	1	5	37	37	37	37	...	37	37	37	37	31	33	22	28	18	13

17 rows × 54 columns

+ Code + Markdown

[ ]: