A Project Report

on

Cloth Size Prediction

Submitted by,

Kotthagattu Meher Sai
Integrated M.Tech CS(3RD YEAR)
University of Hyderabad

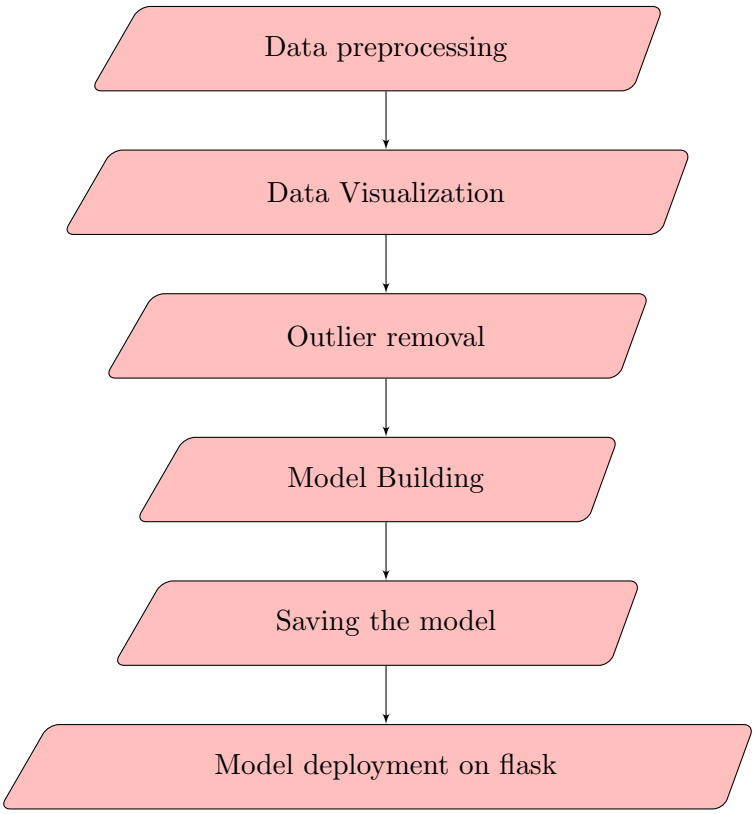
Abstract:

In the realm of online shopping, customers often face the challenge of accurately determining their ideal cloth size, resulting in the purchase of mismatched items. This project aims to address this issue by developing an automated cloth size recommendation system to help customers in making informed purchasing decisions.

Introduction:

With the ever-increasing popularity of online shopping, customers are often unable to try on garments before making a purchase. As a result, determining the correct cloth size becomes crucial to avoid dissatisfaction and returns. This project presents an innovative approach to resolve this challenge by utilizing user-specific data, including weight, height, and age, to predict the most suitable cloth size accurately.

Project Workflow:



Data Collection and preprocessing:

To train the predictive model effectively, a diverse and representative dataset is collected from Kaggle consisting of anonymized user information, including weight, height, age, and associated cloth sizes.
There is exactly 119735 of data in which the cloth sizes are XS, S, M, L, XL, XXL, XXXL.

Python libraries used in the project are **Pandas, Numpy , Matplotlib , scikit-learn.**

Dealing with null values:

There were 257 null values in **age** column, 330 null values in **height** column.

To remove null values from the dataset we filled the null values using **median** of respective columns as **median** is a accurate measure of the data rather than the **mean**.

Target column encoding:

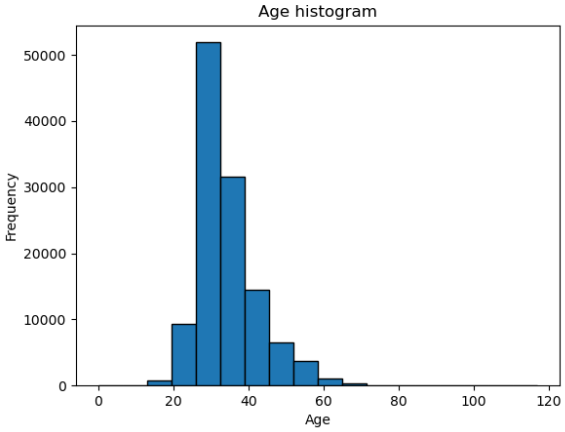
As the **size** column has 7 categories they can be encoded as:

- 'XXS':0
- 'S':1
- 'M':2
- 'L':3
- 'XL':4
- 'XXL':5
- 'XXXL':6

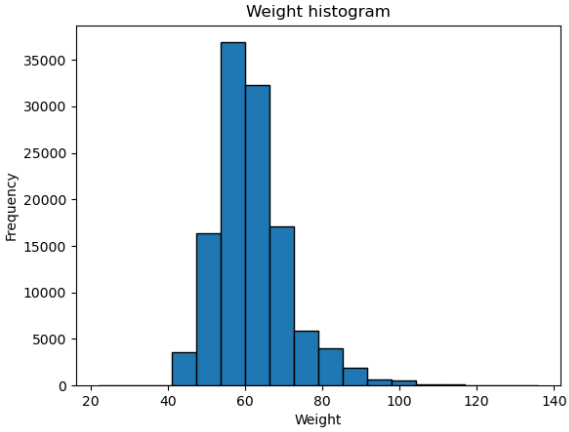
Data Visualization:

Data Visualization helps in understanding the data patterns and gives insights about the data.
We have plotted graphs for univariate, bivariate, multivariate analysis using **Matplotlib** library.

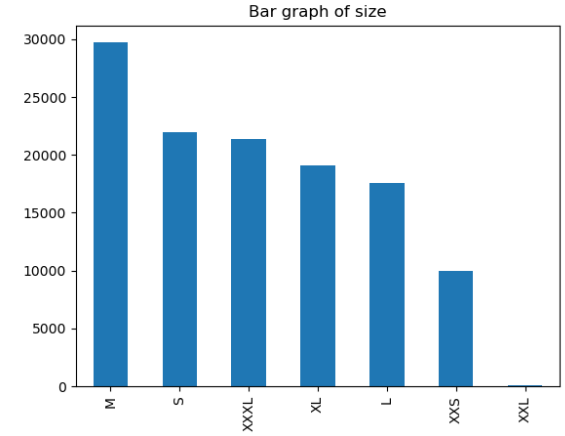
Univariate analysis:



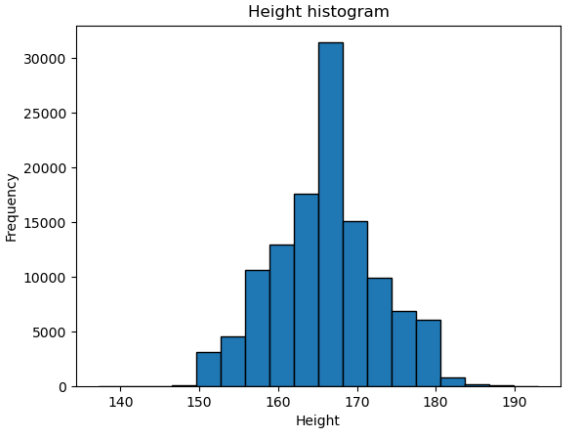
(a) Figure 1



(b) Figure 2



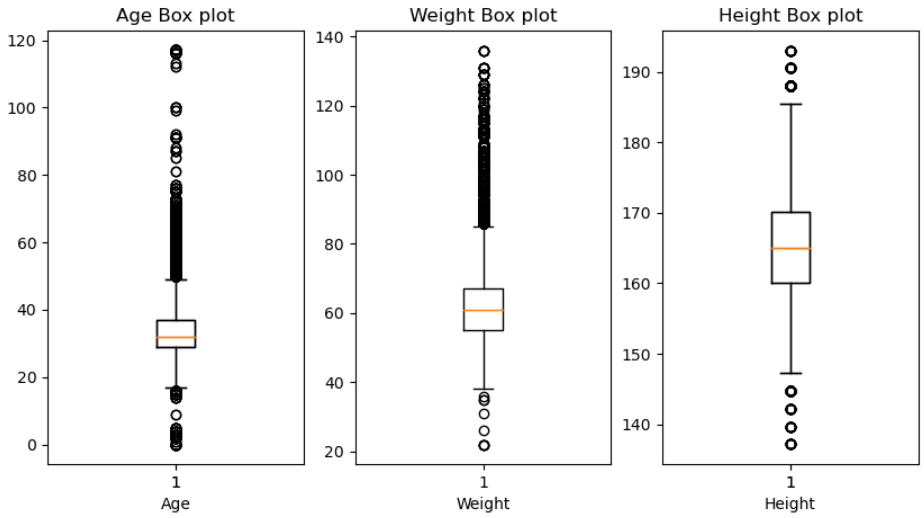
(c) Figure 3



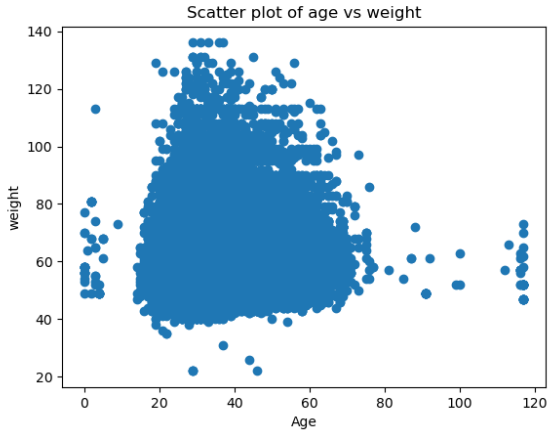
(d) Figure 4

Figure 1: Univariate analysis

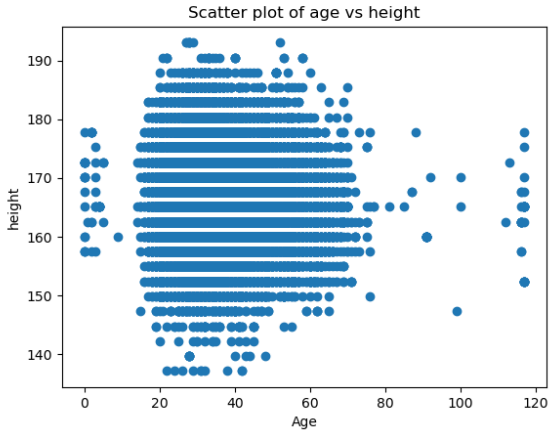
Bivariate analysis:



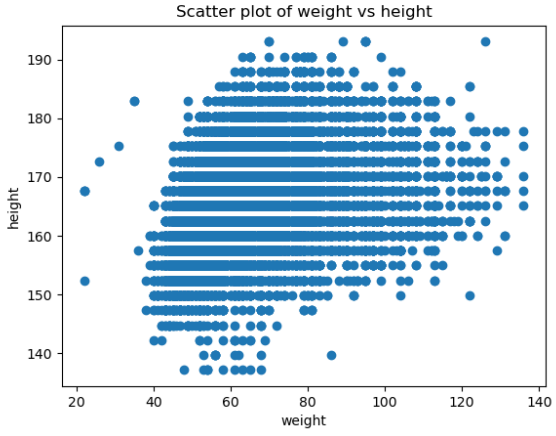
(a) Box plots



(b) age vs weight



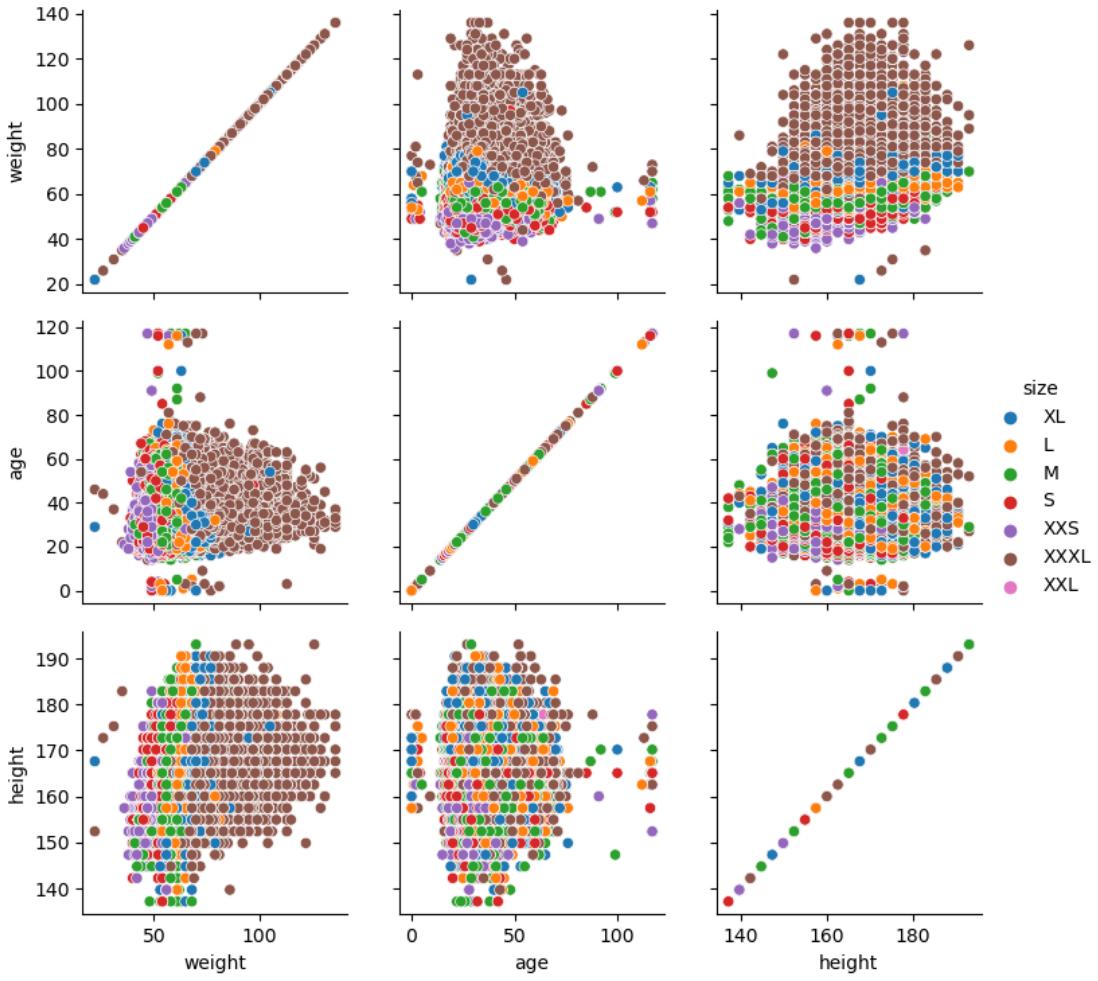
(c) age vs height



(d) weight vs height

Figure 2: Bivariate Analysis

Multivariate analysis



(a) Pair plot

Figure 3: Multivariate analysis

Handling outliers using z-score

There are outliers in **age**,**weight**,**height** column which can result in inaccurate results. Hence removing the outliers is must and we remove them using the z-score calculation on the data.

1. For every subset data of particular size category , we calculate the mean of the data and standard deviation and perform the below operation.

$$zscore = \frac{ndf - ndf.mean()}{ndf.std()} \tag{1}$$

Here, ndf represents the subset of data for the current size category.

2. Outlier Removal:

   - We iterate over every feature of the subset of data of particular size category and apply the outlier removal condition.

$$(temp\_frame[x][feature] > -3) \ \& \ (temp\_frame[x][feature] < 3) \tag{2}$$

   Here, x represents the index of the size category, and $feature$ represents age,height, or weight.

   Outliers are removed by filtering the values that fall outside the range of -3 to 3 standard deviations from the mean.

Training Process:

As the problem is on classification, we can use models like Logistic Regression, Support Vector Machines, K Neighbors Classifier.

After training on these models the accuracy is between 50%-60%. Hence, using ensemble learning algorithms like Decision Trees, Random Forest Classifier, GradientBoost, CatBoost would be right choice.
Training on these algorithms on the default hyperparameters didnt give good accuracy.

Hyperparameter Tuning using GridSearchCV:
To tune the hyperparameters, we setup a grid of parameters with fixed range of values and GridSearchCV will train the model across all possible hyperparameters and finally gives the hyperparameters for which the model does best on training data.

- Random Forest classifier

  We define the following grid:

```
rf_grid = {"n_estimators": np.arange(10,100,10),
           "max_depth": [None,3,5,10],
           "min_samples_split": np.arange(2,14,2),
           "min_samples_leaf": np.arange(1,15,2)}
```

  Create a GridSearchCV object for RandomForestClassifier and fit the training data.

```
rf_gs = GridSearchCV(RandomForestClassifier(),
                           param_grid=rf_grid,
                           cv=5,
                           verbose=3)
rf_gs.fit(X_train,y_train)
```

  The best hyperparameters are

```
{'max_depth': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 90}
```

Results on these hyperparameters

AL : Random Forest Classifier (20%)
Training Model Performance Check
Accuracy Score : 1.0000
F1 Score : 1.0000
Precision Score : 1.0000
Recall Score : 1.0000

Testing Model Performance Check
Accuracy Score : 0.9959
F1 Score : 0.9956
Precision Score : 0.9959
Recall Score : 0.9959

- Decision Tree Classifier

  The best hyperparameters are

```
{'criterion': 'entropy',
 'max_depth': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2}
```

Results on these hyperparameters

AL : Decision Forest Classifier (20%)
Training Model Performance Check
Accuracy Score : 1.0000
F1 Score : 1.0000
Precision Score : 1.0000
Recall Score : 1.0000

Testing Model Performance Check
Accuracy Score : 0.9996
F1 Score : 0.9996
Precision Score : 0.9996
Recall Score : 0.9996

- Gradient Boost Classifier

  The best hyperparameters are

```
{'learning_rate': 0.1,
 'max_depth': 5,
 'min_samples_leaf': 2,
 'min_samples_split': 5,
 'n_estimators': 100}
```

Results on these hyperparameters

```
AL : Gradient Boost Classifier (20%)
Training Model Performance Check
Accuracy Score : 0.9990
F1 Score : 0.9990
Precision Score : 0.9990
Recall Score : 0.9990

Testing Model Performance Check
Accuracy Score : 0.9988
F1 Score : 0.9988
Precision Score : 0.9988
Recall Score : 0.9988
```

- Cat Boost Classifier

  We used **RandomizedSearchCV** which will randomly sample hyperparameter combinations from the specified distributions and takes lesser time than **GridSearchCV**.

  We define the following grid:

```
cb_grid = {
    'iterations': stats.randint(100, 1000),  # Randomly sample
       between 100 and 1000 iterations
    'learning_rate': stats.uniform(0.01, 0.2),  # Randomly sample
       between 0.01 and 0.2 for learning rate
    'depth': stats.randint(3, 10),  # Randomly sample between 3 and
       10 for tree depth
    'l2_leaf_reg': stats.uniform(1, 10),  # Randomly sample between
       1 and 10 for L2 regularization
    'border_count': stats.randint(32, 255)  # Randomly sample
       between 32 and 255 for border count
```

```
catboost_classifier = CatBoostClassifier()

randomized_search_cb = RandomizedSearchCV(
    catboost_classifier,
    param_distributions=cb_grid,
    n_iter=50,  # Number of random combinations to try
    cv=5,
    scoring='accuracy',
    n_jobs=-1,
    verbose=True
)
```

The best hyperparameters are

```
{'border_count': 180,
 'depth': 8,
 'iterations': 952,
```

```
'l2_leaf_reg': 1.2987757642851208,
'learning_rate': 0.16990696372755265}
```

Results on these hyperparameters

AL : Cat Boost Classifier (20%)
Training Model Performance Check
Accuracy Score : 0.9999
F1 Score : 0.9991
Precision Score : 0.9991
Recall Score : 0.9991

Testing Model Performance Check
Accuracy Score : 0.9991
F1 Score : 0.9988
Precision Score : 0.9988
Recall Score : 0.9988

Choosing the model:

Based on the above results , **Decision tree classifier** will be the right choice as this model has excellent performance on both training and testing sets, with very high accuracy, precision, recall, and F1 score. It seems to generalize well on unseen data.

About the algorithm:

A Decision Tree classifier is a simple yet powerful supervised learning algorithm used for classification tasks. It partitions the feature space into regions based on the input features and assigns a class label to each region.

1. Overview of the model:
    - A Decision Tree is a tree-like structure where each internal node represents a feature, each branch represents a decision based on the feature's value, and each leaf node represents a class label.
    - It recursively partitions the feature space based on the selected features and thresholds to minimize impurity or maximize information gain at each step.
    - Decision Trees are easy to understand and interpret, making them useful for extracting insights from the data.
2. Splitting Criteria:
    - Decision Trees use various metrics to decide how to split the data at each node. Common impurity metrics include **Gini impurity** and **entropy**.
3. Decision Tree Algorithm Steps:
    - Select the best feature and threshold that provides the best split (maximizes information gain or minimizes impurity) for the current set of data.

- Divide the data based on the selected feature and threshold into two subsets (left and right nodes) for binary classification.
- Recursively repeat the above steps for each subset until a stopping condition is met (e.g., maximum depth of the tree or minimum samples per leaf).

Mathematical Formulas:

- Gini Impurity:
  - For a binary classification problem, the Gini impurity (G) for a node with samples from two classes (p and (1-p)) can be calculated as:
$$G = 1 - (p^2 + (1-p)^2)$$
- Entropy:
  - Entropy (H) is another metric used for impurity measurement. For a binary classification problem with two classes (p and (1-p)), the entropy is calculated as:
$$H = -p\log_2(p) - (1-p)\log_2(1-p)$$
- Information Gain:
  - Information gain measures the reduction in impurity achieved by a particular split. For a parent node with impurity I(p), and left and right child nodes with impurities I(left) and I(right), the information gain (IG) is given by:
$$IG = I(p) - \left(\frac{n_{left}}{n_{total}} \cdot I(left) + \frac{n_{right}}{n_{total}} \cdot I(right)\right)$$
  - * n_{left} and n_{right} are the number of samples in the left and right child nodes, respectively.
  - * n_{total} is the total number of samples in the parent node.

Saving the model to a pkl file:

To save the model, we use the **pickle** module.

```
import pickle

pickle.dump(dt,open("decisionTree_Model.pkl","wb"))
```

Also saving parameters **mean** and **standard deviation** of the same training set(random_ state is used) is required to apply z-score on new data and make predictions.

```
data_temp =pd.read_csv("final_test.csv")

data_temp['age'] = data_temp['age'].fillna(data_temp['age'].median()
   )
data_temp['height'] = data_temp['height'].fillna(data_temp['height'
   ].median())

X1 = data_temp.drop('size',axis=1)
y1 = data_temp['size']
X_train1,X_test1,y_train1,y_test1 = train_test_split(X1,y1,test_size
   =0.2,random_state=42)
```

```
mean_age = X_train1['age'].mean()
std_age = X_train1['age'].std()

mean_weight = X_train1['weight'].mean()
std_weight = X_train1['weight'].std()

mean_height = X_train1['height'].mean()
std_height = X_train1['height'].std()
```

We save the parameters in a dictionary and save it to **preprocessing _ params.pkl** file.

```
preprocessing_params = {
    'mean_age': mean_age,
    'std_age': std_age,
    'mean_weight': mean_weight,
    'std_weight': std_weight,
    'mean_height': mean_height,
    'std_height': std_height
}

pickle.dump(preprocessing_params, open('preprocessing_params.pkl','
   wb'))
```

Making predictions on data:

```
age_norm = (50 - mean_age) / std_age
weight_norm = (60 - mean_weight) / std_weight
height_norm = (155 - mean_height) / std_height

# Make prediction using the loaded model
input_data = pd.DataFrame({'age': [age_norm],
                           'height': [height_norm],
                               'weight': [weight_norm],
                               })
prediction = model.predict(input_data)
prediction
```

Model deployment on Flask Server :

The folder structure for the flask app is:

```
Cloth_size_prediction/
├── templates/
│   ├── base.html
│   ├── index.html
│   └── index.html
├── app.py
├── final_test.csv
├── decisionTree_Model.pkl
├── preprocessing_params.pkl
└── main.py
```

The app.py contains the flask deployment code which is as follows:

```
import pandas as pd
import numpy as np
import pickle
from flask import Flask, request, jsonify, render_template



# Load the trained model
model = pickle.load(open("decisionTree_Model.pkl","rb"))

# Load the preprocessing parameters
preprocessing_params = pickle.load(open('preprocessing_params.pkl','
   rb'))

# Extract the mean and standard deviation values
mean_age = preprocessing_params['mean_age']
std_age = preprocessing_params['std_age']
mean_weight = preprocessing_params['mean_weight']
std_weight = preprocessing_params['std_weight']
mean_height = preprocessing_params['mean_height']
std_height = preprocessing_params['std_height']

# Define Flask app
app = Flask(__name__,template_folder='templates')

# Home page
@app.route('/')
def home():
    return render_template('index.html')

# Prediction page
@app.route('/predict', methods=['POST'])
def predict():
    # Get input values from the form
```

```
    age = float(request.form['age'])
    weight = float(request.form['weight'])
    height = float(request.form['height'])

    # Perform z-score scaling on the input data
    age_norm = (age - mean_age) / std_age
    weight_norm = (weight - mean_weight) / std_weight
    height_norm = (height - mean_height) / std_height
    input_data = pd.DataFrame({'age': [age_norm],
                           'height': [height_norm],
                               'weight': [weight_norm],
                               })
    # Make prediction using the loaded model
    prediction = model.predict(input_data)
    category = {'XL':4, 'L':3, 'M':2, 'S':1, 'XXS':0, 'XXXL':6, 'XXL
       ':5}
    def get_key(val):
        for key, value in category.items():
            if val == value:
                return key
        return "key doesn't exist"
    # Get the predicted size
    size = get_key(prediction[0])

    # Render the prediction result template with the predicted size
    return render_template('result.html', size=size)

# Run the Flask app
if __name__ == '__main__':
    app.run(debug=True)
```

By running the above app.py file the flask server is launched.

The template folder contains the front-end web design html files which are:
base.html

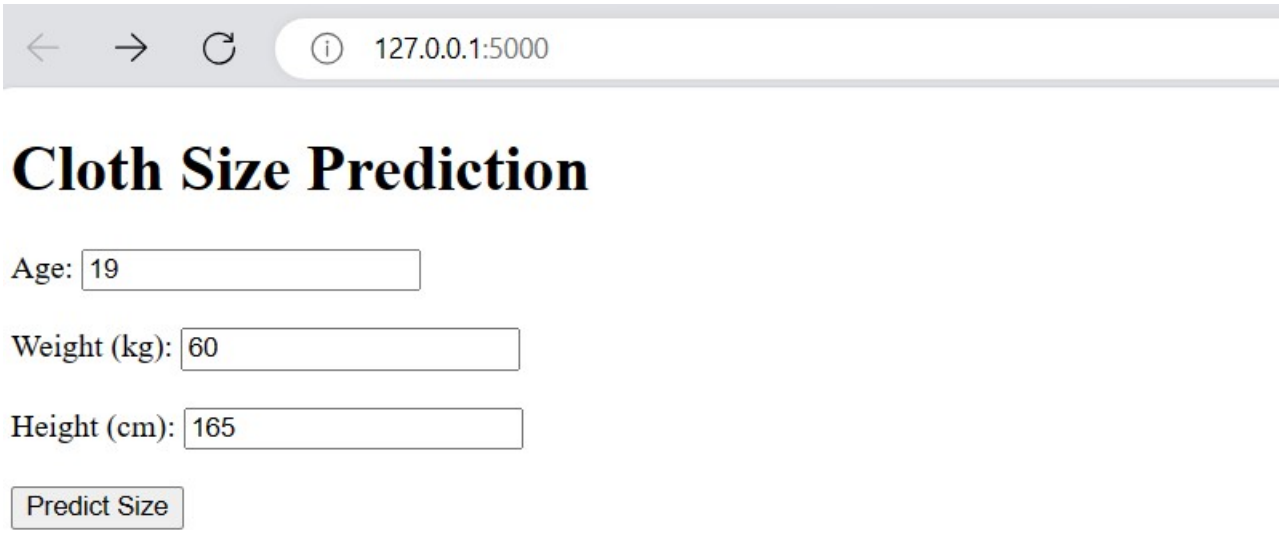
```
<!DOCTYPE html>
<html>
<head>
    <title>{% block title %}{% endblock %}</title>
</head>
<body>
    {% block content %}{% endblock %}
</body>
</html>
```

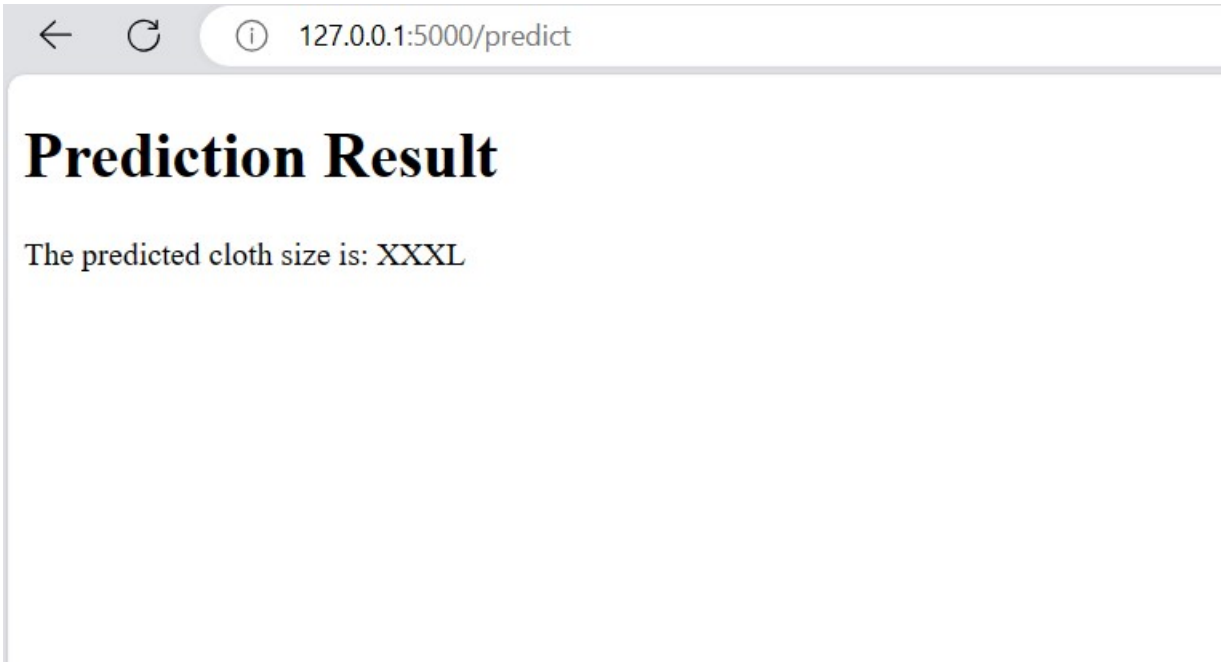
index.html

```
{% extends 'base.html' %}

{% block content %}
    <h1>Cloth Size Prediction</h1>
    <form action="/predict" method="post">
        <label for="age">Age:</label>
        <input type="text" id="age" name="age" required><br><br>
        <label for="weight">Weight (kg):</label>
        <input type="text" id="weight" name="weight" required><br><br>
        <label for="height">Height (cm):</label>
        <input type="text" id="height" name="height" required><br><br>
        <input type="submit" value="Predict Size">
    </form>
{% endblock %}
```

result.html

```
{% extends 'base.html' %}

{% block content %}
    <h1>Prediction Result</h1>
    <p>The predicted cloth size is: {{ size }}</p>
{% endblock %}
```


127.0.0.1:5000
Cloth Size Prediction
Age: 19
Weight (kg): 60
Height (cm): 165
Predict Size


127.0.0.1:5000/predict
Prediction Result
The predicted cloth size is: XXXL

Final code of the project: (main.py)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,precision_score,
   recall_score, f1_score
from sklearn.tree import DecisionTreeClassifier

# Loading the data
data = pd.read_csv("final_test.csv")

# Data description
print(data.info())

# Count of null values
print("Number␣of␣null␣values")
print(data.isna().sum())

# Filling missing values
data['age'] = data['age'].fillna(data['age'].median())
data['height'] = data['height'].fillna(data['height'].median())

# Descriptive statistics
print("Descriptive␣Statistics")
print(data.describe())

# Age bar plot of 5000 data

data['age'].plot.hist(bins=18,edgecolor='black')
plt.title("Age␣histogram")
plt.xlabel("Age");

# Weight bar plot
data['weight'].plot.hist(bins=18,edgecolor='black')
plt.title("Weight␣histogram")
plt.xlabel("Weight");

# Height histogram plot

data["height"].plot.hist(bins=18,edgecolor="black")
plt.title("Height␣histogram")
plt.xlabel("Height");
```

```
plt.savefig("heighthist.png")

# Bar graph of size
data['size'].value_counts().plot(kind="bar")
plt.title("Bar graph of size")

# Box plot of age,weight,height data
fig, ax = plt.subplots(nrows=1, ncols=3)
fig.set_size_inches(10, 5)


box1 =ax[0].boxplot(data['age'])
ax[0].boxplot(data['age'])
ax[0].set_xlabel('Age')
ax[0].set_title('Age Box plot')
outliers1 = [flier.get_ydata() for flier in box1['fliers']]


box2 =ax[1].boxplot(data['weight'])
ax[1].set_xlabel('Weight')
ax[1].set_title('Weight Box plot')
outliers2 = [flier.get_ydata() for flier in box2['fliers']]

box3 =ax[2].boxplot(data['height'])
ax[2].boxplot(data['height'])
ax[2].set_xlabel('Height')
ax[2].set_title('Height Box plot');
outliers3 = [flier.get_ydata() for flier in box3['fliers']]
plt.show()

# Outliers of data
print("outliers of age data")
print(np.unique(outliers1[0]))
print("outliers of weight data")
print(np.unique(outliers2[0]))
print("outliers of height data")
print(np.unique(outliers3[0]))

# Scatter plot of age vs weight
plt.scatter(data['age'],data['weight'])
plt.title("Scatter plot of age vs weight")
plt.xlabel('Age')
plt.ylabel('weight');

# Scatter plot of age vs height
```

```
plt.scatter(data['age'],data['height'])
plt.title("Scatter plot of age vs height")
plt.xlabel('Age')
plt.ylabel('height');

# Scatter plot of weight vs height

plt.scatter(data['weight'],data['height'])
plt.title("Scatter plot of weight vs height")
plt.xlabel('weight')
plt.ylabel('height');

# Multivariate plot

g = sns.PairGrid(data,hue="size")
g.map(sns.scatterplot)
g.add_legend()

# Outlier removal
temp_frame = []
sizes = []
for size_type in data['size'].unique():
    sizes.append(size_type)
    ndf = data[['age','height','weight']][data['size'] == size_type]
    zscore = ((ndf - ndf.mean())/ndf.std())
    temp_frame.append(zscore)

for x in range(len(temp_frame)):
    temp_frame[x]['age'] = temp_frame[x]['age'][(temp_frame[x]['age'
       ]>-3) & (temp_frame[x]['age']<3)]
    temp_frame[x]['height'] = temp_frame[x]['height'][(temp_frame[x
       ]['height']>-3) & (temp_frame[x]['height']<3)]
    temp_frame[x]['weight'] = temp_frame[x]['weight'][(temp_frame[x
       ]['weight']>-3) & (temp_frame[x]['weight']<3)]

for x in range(len(sizes)):
    temp_frame[x]['size'] = sizes[x]
data = pd.concat(temp_frame)

# Filling missing values in the new data of z-scores
data["age"] = data["age"].fillna(data['age'].median())
data["height"] = data["height"].fillna(data['height'].median())
data["weight"] = data["weight"].fillna(data['weight'].median())


# Splitting the data into X and y
```

```
X = data.drop('size',axis=1)
y = data['size']
category = {'XL':4, 'L':3, 'M':2, 'S':1, 'XXS':0, 'XXXL':6, 'XXL':5}
y_num = y.map(category)

# Splitting into training and testing data(80:20)

X_train,X_test,y_train,y_test = train_test_split(X,y_num,test_size
    =0.2,random_state=42)


# Training and evaluating the model

dt = DecisionTreeClassifier(criterion='entropy',max_depth=None,
    min_samples_leaf=1,min_samples_split=2)
dt.fit(X_train,y_train)
training_prediction = dt.predict(X_train)
testing_prediction = dt.predict(X_test)

# Training Metrics
training_accuracy = accuracy_score(y_train, training_prediction)
training_f1 = f1_score(y_train, training_prediction, average = '
    weighted')
training_precision = precision_score(y_train, training_prediction,
    average = 'weighted')
training_recall = recall_score(y_train, training_prediction, average
     = 'weighted')

# Testing Metrics

testing_accuracy = accuracy_score(y_test, testing_prediction)
testing_f1 = f1_score(y_test, testing_prediction, average = '
    weighted')
testing_precision = precision_score(y_test, testing_prediction,
    average = 'weighted')
testing_recall = recall_score(y_test, testing_prediction, average =
    'weighted')


print('AL : Decision Tree Classifier (20%)')
print('\n')

print('Training Model Performance Check')
print('Accuracy Score : {:.4f}'.format(training_accuracy))
print('F1 Score : {:.4f}'.format(training_f1))
print('Precision Score : {:.4f}'.format(training_precision))
```

```
print('Recall Score : {:.4f}'.format(training_recall))

print('\n')
print('Testing Model Performance Check')
print('Accuracy Score : {:.4f}'.format(testing_accuracy))
print('F1 Score : {:.4f}'.format(testing_f1))
print('Precision Score : {:.4f}'.format(testing_precision))
print('Recall Score : {:.4f}'.format(testing_recall))
```

Conclusion:

In this project, we have developed a decision tree classifier for predicting the cloth size. We utilized a Kaggle dataset for our project. Prior to training the machine learning model, we had to do lot of data preprocessing steps like filling missing values, removing the outliers with z-score criteria. Then it was about choosing a classification algorithm which can give good accuracy. Hence, we choosed hyperparameter tuning process through GridSearchCV to find best hyperparameters for the algorithms Random Forest, Decision Tree,Gradient boost, Cat boost. On evaluating these models on training and test data, Decision Tree classifier generalizes well on unseen data with very high accuracy of 99.96% , precision of 0.9996, recall of 0.9996 and F1-score of 0.9996 on testing data. Then we have deployed the project on to the flask server which will benefit the user with interacting with the prediction model and choose right cloth sizes!
While the decision tree classifier exhibited excellent performance, there are potential avenues for future work. Exploring alternative classification algorithms, collecting additional data to enrich the dataset, and incorporating more features could further improve the model's accuracy and predictive power.

In conclusion, we have developed a good prediction model for cloth size prediction with remarkable accuracy and generalization capabilities.The combination of effective data preprocessing, model selection, hyperparameter tuning, and deployment enabled us to build a reliable and practical solution for predicting cloth sizes. This experience has enriched our understanding of machine learning techniques and their applications in real-world scenarios.