```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.impute import SimpleImputer

def load_data(file_path):
    return pd.read_csv(file_path)

def parse_height(height_str):
    if pd.isna(height_str):
        return None, None
    try:
        feet, inches = height_str.split("'")
        feet = int(feet.strip())
        inches = int(inches.replace('"', '').strip())
    except ValueError:
        return None, None
    return feet, inches

def remove_outliers(df, features, method='zscore', threshold=3):
    if method == 'zscore':
        from scipy import stats
        z_scores = np.abs(stats.zscore(df[features]))
        df = df[(z_scores < threshold).all(axis=1)]
    elif method == 'iqr':
        Q1 = df[features].quantile(0.25)
        Q3 = df[features].quantile(0.75)
        IQR = Q3 - Q1
        df = df[~((df[features] < (Q1 - 1.5 * IQR)) | (df[features] >
(Q3 + 1.5 * IQR))).any(axis=1)]
    return df

def preprocess_data(df):
    # Convert height to inches
    # df['Height'] = df['Height_feet'] * 12 + df['Height_inches']

    # One-hot encode gender
    df = pd.get_dummies(df, columns=['Gender'])

    # Handle cup size
    if 'Cup Size' in df.columns:
        df['Cup Size'] = df['Cup Size'].fillna('None')
```

```
        df = pd.get_dummies(df, columns=['Cup Size'])

    # Select features for clustering
    features = ['Height', 'Weight', 'Bust/Chest', 'Waist', 'Hips',
'Body Shape Index']

    # Impute missing values
    imputer = SimpleImputer(strategy='mean')
    df[features] = imputer.fit_transform(df[features])

    # Remove outliers
    df = remove_outliers(df, features)

    return df, features

def cluster_data(df, features, n_clusters=5):
    scaler = StandardScaler()
    scaled_features = scaler.fit_transform(df[features])

    scaled_df = pd.DataFrame(scaled_features, columns=features,
index=df.index)
    important_features = ['Body Shape Index']
    # Increase the importance of selecte features by multiplying , by a
weight
    for feature in important_features:
        if feature in scaled_df.columns:
            scaled_df[feature] *= 3
#     scaled_df['Height_total_inches'] *= 2
#     scaled_df['Waist'] *= 2
#     scaled_df['Body Shape Index'] *= 2

    kmeans = KMeans(n_clusters=n_clusters, n_init=10, random_state=42)
    df['Cluster'] = kmeans.fit_predict(scaled_features)

    return df, kmeans, scaler

def generate_size_chart(df, features, kmeans, scaler):
    cluster_centers = scaler.inverse_transform(kmeans.cluster_centers_)

    important_features = ['Body Shape Index']
    for feature in important_features:
        if feature in features:
            # Calculate the index of the feature
```

```
            feature_idx = features.index(feature)
            # Reverse the multiplication applied during scaling
            cluster_centers[:, feature_idx] /= 3  # Use the same weight
as used in cluster_data

    sorted_indices = np.argsort(cluster_centers[:,
features.index('Height')])
    sorted_centers = cluster_centers[sorted_indices]

    size_chart = pd.DataFrame(sorted_centers, columns=features)
    size_chart['Size'] = ['1', '2', '3', '4', '5']  # Assign sizes
based on cluster centers

    def calculate_confidence(cluster):
        cluster_data = df[df['Cluster'] == cluster]

        # Factor 1: Inverse of coefficient of variation (normalized)
        cv_inv = 1 / (1 + (cluster_data[features].std() /
cluster_data[features].mean()).mean())

        # Factor 2: Cluster size (more samples = higher confidence)
        cluster_size = len(cluster_data) / len(df)

        # Factor 3: Consistency of Body Shape Index
        bsi_consistency = 1 / (1 + cluster_data['Body Shape
Index'].std() / cluster_data['Body Shape Index'].mean())

        # Factor 4: Closeness to cluster center
        distances = np.linalg.norm(cluster_data[features].values -
cluster_centers[cluster], axis=1)
        closeness = 1 / (1 + distances.mean() /
np.linalg.norm(cluster_centers[cluster]))

        # Combine factors
        combined_score = np.mean([cv_inv*2, cluster_size*2,
bsi_consistency*3, closeness*3])

        return combined_score/2
        # # Scale to 0.9-1 range
        # scaled_score = 0.9 + (combined_score * 0.1)

        # return min(scaled_score, 1.0)  # Ensure it doesn't exceed 1.0
```

```
    size_chart['Confidence'] = [calculate_confidence(i) for i in
range(len(size_chart))]

    return size_chart

def main():
    # Load the data
    file_path = '/content/body_m.csv'  # Replace with your actual file
name
    df = load_data(file_path)

    # Parse height
    # df[['Height_feet', 'Height_inches']] = df['Height'].apply(lambda
x: pd.Series(parse_height(x)))

    # Print column names for debugging
    print("Available columns:", df.columns.tolist())

    # Preprocess the data
    df, features = preprocess_data(df)

    # Cluster the data
    df, kmeans, scaler = cluster_data(df, features)

    # Generate size chart
    size_chart = generate_size_chart(df, features, kmeans, scaler)

    print("Generated Size Chart:")
    print(size_chart)

    # Save the size chart to a CSV file
    size_chart.to_csv('generated_size_chart.csv', index=False)
    print("Size chart saved to 'generated_size_chart.csv'")

if __name__ == "__main__":
    main()
```