

```
from IPython.utils.text import DollarFormatter
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

```
# Load the dataset
d = pd.read_csv('/content/players.csv')
d.head()
len(d)
```

1968

```
d.isnull().sum()
missing_percentage = d.isnull().mean() * 100
# Filter columns with more than 30% missing values
columns_with_missing_30 = missing_percentage[missing_percentage > 30]
# Display the columns with more than 30% missing values
print("Columns with more than 30% missing values:")
print(columns_with_missing_30)
```

```
Columns with more than 30% missing values:
loaned_from      100.000000
nation_jersey_number  73.069106
dtype: float64
```

```
missing_percentage = d.isnull().mean() * 100
threshold = 30
# Filter columns with more than the threshold percentage of missing values
columns_to_drop = missing_percentage[missing_percentage >
threshold].index
# Drop the columns
d_cleaned = d.drop(columns=columns_to_drop)
# Display the resulting dataframe after removing columns with too many missing values
print(f"Columns dropped: {list(columns_to_drop)}")
print(f"Remaining columns: {d_cleaned.columns}")
```

```
Columns dropped: ['loaned_from', 'nation_jersey_number']
Remaining columns: Index(['ID ', 'age', 'height_cm', 'weight_kg', 'overall', 'pace', 'shooting',
'passing', 'dribbling', 'defending', 'physic'],
dtype='object')
```

```
# Mean imputation for numerical columns
d['shooting'] = d['shooting'].fillna(d['shooting'].mean())
print(d['shooting'])
# Median imputation for numerical columns
d['passing'] = d['passing'].fillna(d['passing'].median(), inplace=True)
print(d['passing'])
```

```
0      92.000000
1      93.000000
2      85.000000
3      64.511615
4      83.000000
...
1963    74.000000
1964    69.000000
1965    65.000000
1966    68.000000
1967    32.000000
Name: shooting, Length: 1968, dtype: float64
0      92.0
1      82.0
2      87.0
3      71.0
4      86.0
...
1963    70.0
1964    74.0
1965    72.0
1966    73.0
1967    57.0
Name: passing, Length: 1968, dtype: float64
```

```
data=d.copy()
def detect_outliers_iqr(column):
    Q1 = column.quantile(0.25)
    Q3 = column.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    # Return a boolean series indicating if the value is an outlier
```

```

return (column < lower_bound) | (column > upper_bound)
# Apply the IQR method to each *numerical* feature in the DataFrame
numerical_cols = data.select_dtypes(include=np.number).columns
for col in numerical_cols:
    data[f'{col}_Outlier'] = detect_outliers_iqr(data[col])
# Filter the DataFrame to display rows where any outlier flag is True
outliers_only = data[data.filter(like='_Outlier').any(axis=1)]
# Display the filtered DataFrame with outliers
outliers_only

```

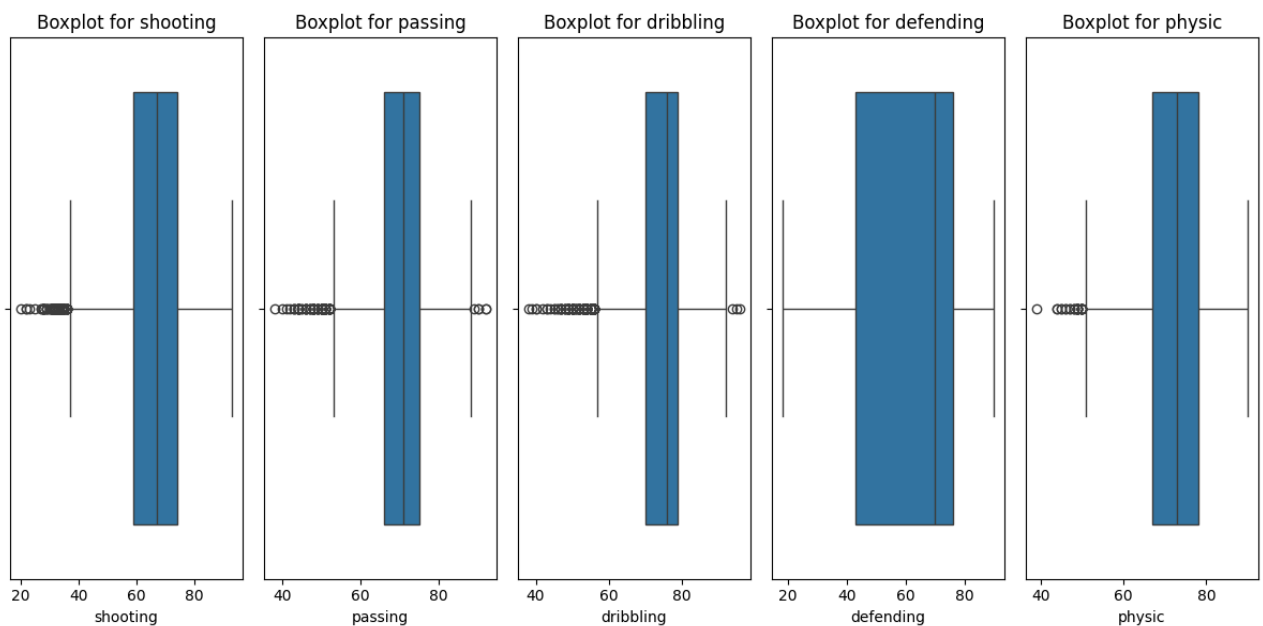
	ID	age	height_cm	weight_kg	overall	loaned_from	nation_jersey_number	pace	shooting	passing	...	weight_kg_Outlier
0	1	32	170	72	94	NaN	NaN	87.0	92.000000	92.0	...	False
1	2	34	187	83	93	NaN	7.0	90.0	93.000000	82.0	...	False
2	3	27	175	68	92	NaN	10.0	91.0	85.000000	87.0	...	False
3	4	26	188	87	91	NaN	1.0	NaN	64.511615	71.0	...	False
4	5	28	175	74	91	NaN	10.0	91.0	83.000000	86.0	...	False
...	...	...	...	...	...	...	...	...	...	...	...	...
1953	1954	32	176	69	75	NaN	NaN	73.0	70.000000	73.0	...	False
1957	1958	31	191	85	75	NaN	NaN	52.0	29.000000	48.0	...	False
1958	1959	30	189	90	75	NaN	NaN	56.0	38.000000	50.0	...	False
1960	1961	31	193	84	75	NaN	NaN	45.0	33.000000	53.0	...	False
1967	1968	32	184	87	75	NaN	6.0	51.0	32.000000	57.0	...	False

286 rows × 26 columns

```

plt.figure(figsize=(12, 6))
for i, col in enumerate(d.columns[8:13]): # Only plotting 5 features
    plt.subplot(1, 5, i+1) # Create subplots for each feature
    sns.boxplot(x=d[col])
    plt.title(f"Boxplot for {col}")
plt.tight_layout()
plt.show()

```



```

data=d.copy()
def replace_outliers_with_nan(column):
    Q1 = column.quantile(0.25)
    Q3 = column.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    # Replace outliers with NaN
    return column.where((column >= lower_bound) & (column <= upper_bound), np.nan)
# Replace outliers with NaN for numerical features only
numerical_cols = data.select_dtypes(include=np.number).columns
for col in numerical_cols:
    data[col] = replace_outliers_with_nan(data[col])
null_counts = data.isnull().sum()

```

```
print("Sum of null values in each column:")
print(null_counts)
```

```
Sum of null values in each column:
ID          0
age         3
height_cm   3
weight_kg    8
overall     55
loaned_from 1968
nation_jersey_number 1438
pace        244
shooting     88
passing      90
dribbling    313
defending    203
physic       231
dtype: int64
```

```
data=d.copy()
def remove_outliers_iqr(data):
    # Calculate Q1 (25th percentile) and Q3 (75th percentile) for each feature
    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3 - Q1

    # Define lower and upper bounds
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

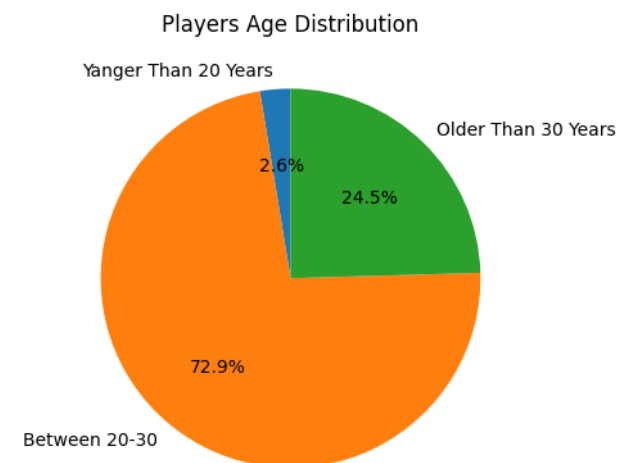
    # Filter the dataset to remove rows with outliers in any column
    data_cleaned = data[~((data < lower_bound) | (data >
upper_bound)).any(axis=1)]

    return data_cleaned
cleaned_data = remove_outliers_iqr(data)
print("Original Data:")
print(len(data))
print("\nCleaned Data (without outliers):")
print(len(cleaned_data))
```

```
Original Data:
1968
```

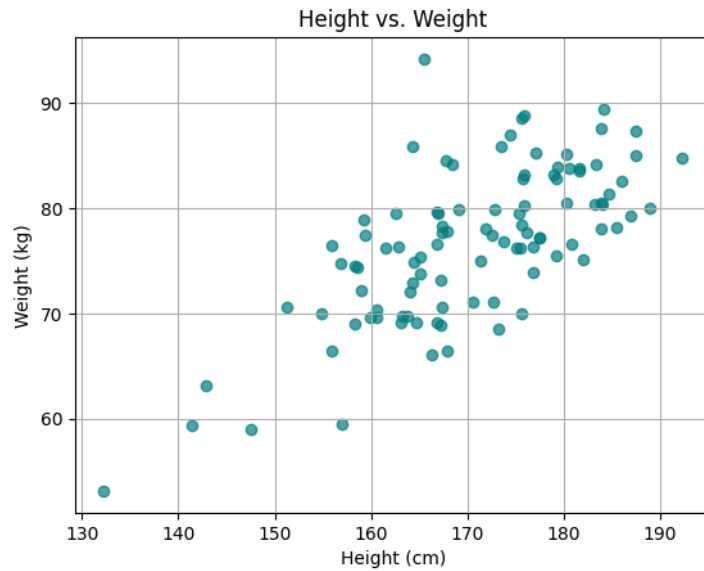
```
Cleaned Data (without outliers):
1682
```

```
labels = ['Yanger Than 20 Years', 'Between 20-30', 'Older Than 30 Years ']
# Assuming 'data' refers to the 'age' column from the DataFrame 'd' for these categories
sizes = [
    sum(1 for x in d['age'] if x <= 20),
    sum(1 for x in d['age'] if x > 20 and x <= 30),
    sum(1 for x in d['age'] if x > 30)
]
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90)
plt.title('Players Age Distribution') # Updated title to be more descriptive
plt.show()
```

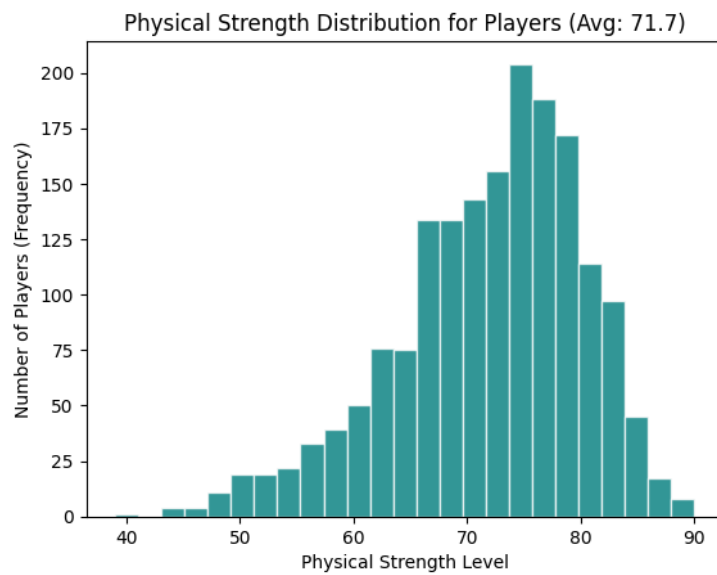


```
# Generate random data
height = np.random.normal(170, 10, 100)
weight = height * 0.45 + np.random.normal(0, 5, 100)
# Scatter plot
plt.scatter(height, weight, color='teal', alpha=0.7)
```

```
plt.title("Height vs. Weight")
plt.xlabel("Height (cm)")
plt.ylabel("Weight (kg)")
plt.grid(True)
plt.show()
```



```
plt.hist(physical_strength, bins=25, color='teal', edgecolor='white', alpha=0.8, label='Player Distribution')
plt.title(f'Physical Strength Distribution for Players (Avg: {mean_val:.1f})')
plt.xlabel('Physical Strength Level')
plt.ylabel('Number of Players (Frequency)')
plt.show()
```



```
columns = ['age', 'weight_kg', 'pace', 'shooting', 'physic']
data = data[columns]
corr = data.corr()
# Plot Heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap of Player Attributes')
plt.show()
```

