[2]:
```
df=pd.read_csv('Mall_Customers.csv')
df.head()
```

[2]:

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

[3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CustomerID              200 non-null    int64
 1   Genre                   200 non-null    object
 2   Age                     200 non-null    int64
 3   Annual Income (k$)      200 non-null    int64
 4   Spending Score (1-100)  200 non-null    int64
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
```

[4]:
```
df.shape
```

[4]:
```
(200, 5)
```

[5]:
```
df.isnull().sum()
```

[5]:
```
CustomerID              0
Genre                   0
Age                     0
Annual Income (k$)      0
Spending Score (1-100)  0
dtype: int64
```

[6]:
```
def calculate_outliers_percentage(df, columns):
    outliers = {}
    for col in columns:
        z = np.abs(stats.zscore(df[col]))
        outlier_count = (z > 3).sum()
        outliers[col] = (outlier_count / len(df[col])) * 100
    return pd.DataFrame(outliers, index=['Outlier Percentage %']).round(3)
```

[9]:
```
import scipy.stats as stats # For Checking Outliers
```

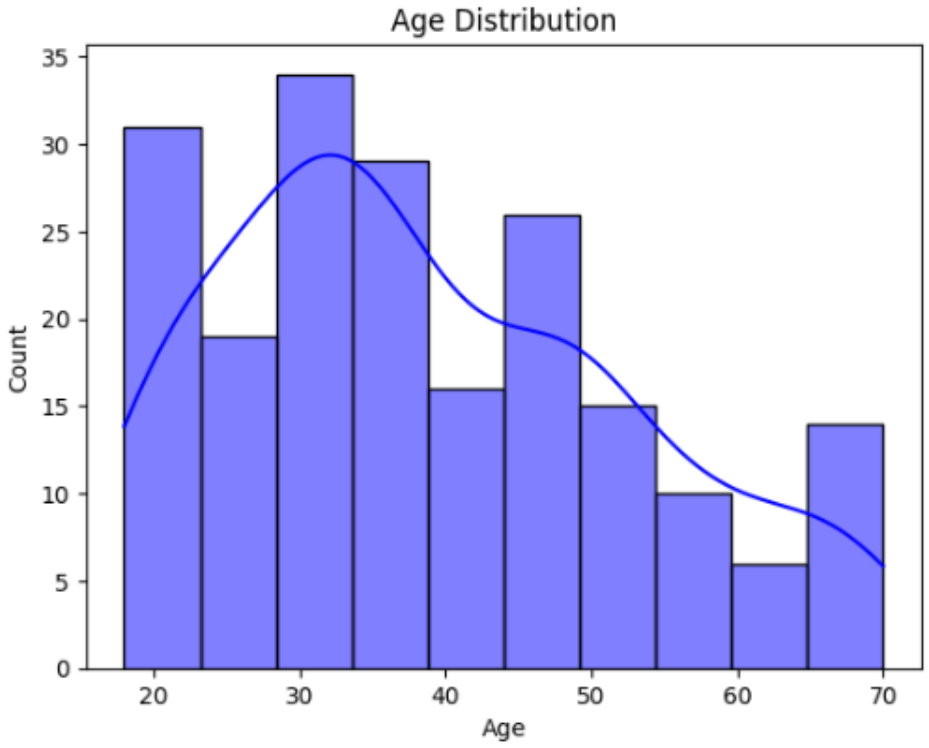
[10]:
```
columns = ['CustomerID','Age','Annual Income (k$)','Spending Score (1-100)']
calculate_outliers_percentage(df, columns)
```

[10]:

	CustomerID	Age	Annual Income (k$)	Spending Score (1-100)
Outlier Percentage %	0.0	0.0	0.0	0.0

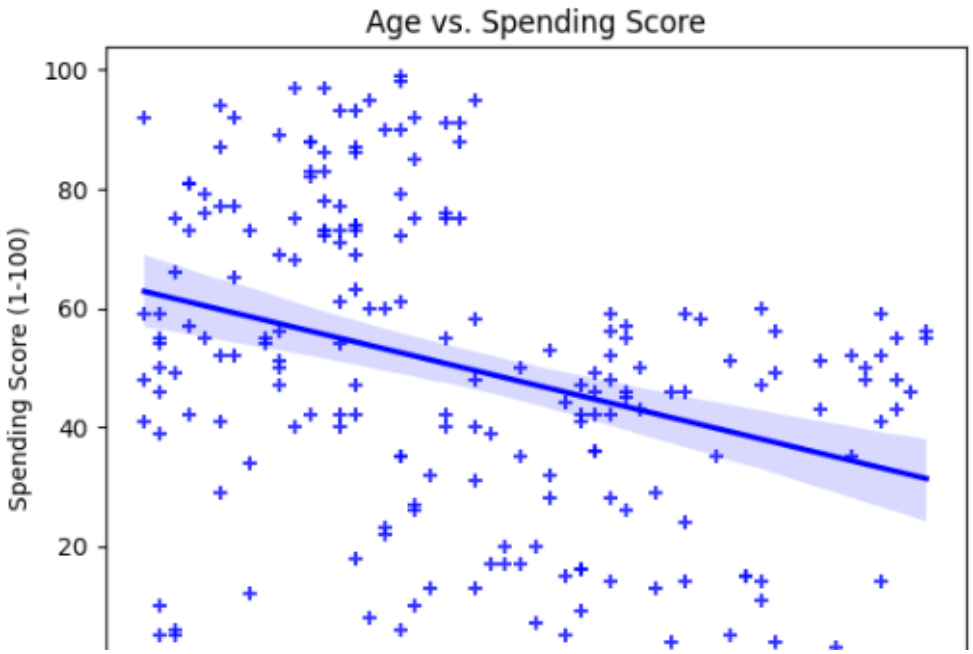
[12]:
```
sns.histplot(df['Age'], bins=10, kde=True, color='b')

plt.xlabel('Age')
plt.ylabel('Count')
plt.title('Age Distribution')
plt.show()
```



[13]:
```
sns.regplot(x='Age', y='Spending Score (1-100)', data=df, color='b', marker='+')


plt.xlabel('Age')
plt.ylabel('Spending Score (1-100)')
plt.title('Age vs. Spending Score')
plt.show()
```



[16]:
```
from sklearn.cluster import KMeans
```

[15]:
```
X = df.iloc[: , [2,4]].values
```

[18]:
```
#Using the elbow method to find the optimal number of clusters
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++',n_init=10)
    kmeans.fit(X)
    print('Cost_Function=',kmeans.inertia_,'with', i, 'Clusters')
    wcss.append(kmeans.inertia_)
```

```
Cost_Function= 171535.50000000003 with 1 Clusters
Cost_Function= 75949.15601023019 with 2 Clusters
Cost_Function= 45840.67661610866 with 3 Clusters
Cost_Function= 28165.583566629342 with 4 Clusters
Cost_Function= 23830.9603937729 with 5 Clusters
Cost_Function= 19489.643884468667 with 6 Clusters
Cost_Function= 15523.684014328752 with 7 Clusters
Cost_Function= 12997.449288119285 with 8 Clusters
Cost_Function= 11448.046985329489 with 9 Clusters
Cost_Function= 10165.683591704306 with 10 Clusters
```

[20]:
```
plt.figure(figsize=(10,5))
sns.lineplot(wcss,marker='o',color='red')
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
#From the previous Gragh the optimal # of Clusters is 4
```