Assignment 2-A

Implementing PCA from Scratch and Applying it to Car Data

Steps:

1. Data Understanding and Representation:

In this step, we read a CSV file named "Audi.csv" into a pandas DataFrame, print a summary of the DataFrame's information, and display the DataFrame.

The columns and their types are as follows:

#	Column	Non-Null Count	Dtype
0	model	10668 non-null	object
1	year	10668 non-null	int64
2	price	10668 non-null	int64
3	transmission	10668 non-null	object
4	mileage	10668 non-null	int64
5	fuelType	10668 non-null	object
6	tax	10668 non-null	int64
7	mpg	10668 non-null	float64
8	engineSize	10668 non-null	float64

The features with their descriptions and types:

1. model: model of the car (nominal)
2. year: car's registration year (numerical)
3. price: listed price of car (numerical)
4. transmission: type of gearbox in car (nominal)
5. mileage: car's distance used (numerical)
6. fuelType: type of fuel used (nominal)
7. tax: road tax (numerical)
8. mpg: miles per gallon (numerical)
9. engineSize: engine size in litres (numerical)

Then we drop the nominal features which are model ,transmission ,fuelType. Finally, we represent the features in matrix form.

2. Implementing PCA using Covariance Matrices:

In this step we subtract the mean value of each column from the corresponding column in the DataFrame named dataset. Finally, it displays the modified DataFrame. This operation effectively centers the data by removing the mean from each column.
Then, we finally compute the covariance of the centered data.

3. Eigenvalue-Eigenvector Equation:

Here we compute the eigenvalues and eigenvectors of a square matrix (covar) using NumPy. The resulting eigenvalues represent the amount of variance along each principal component, and the corresponding eigenvectors define these components.

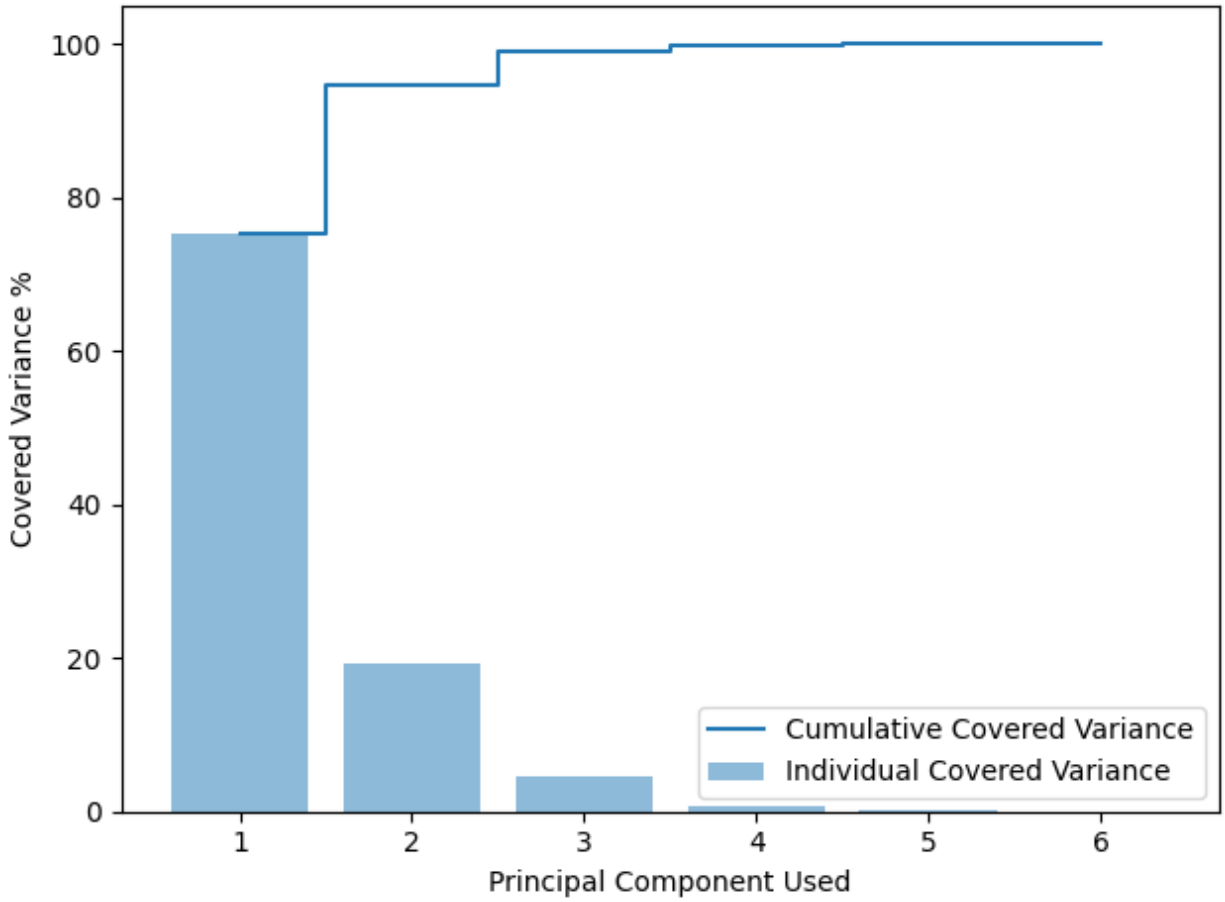
4. Solving for Principal Components:

This Python code defines a function, getKLargestEigenvectors, that takes a pandas DataFrame standardizes its columns, computes the covariance matrix, and then calculates the k largest eigenvalues and corresponding eigenvectors. The function is then called with the entire dataset to obtain all eigenvalues and eigenvectors.

5. Sequential Variance :

This code computes the eigenvalues and eigenvectors of the dataset using the getKLargestEigenvectors function. The var_cover list is created by dividing each eigenvalue by the total sum of eigenvalues and multiplying by 100 to get the percentage of variance covered by each component. The cumulative list is created by taking the cumulative sum of the var_cover list. The bar function is

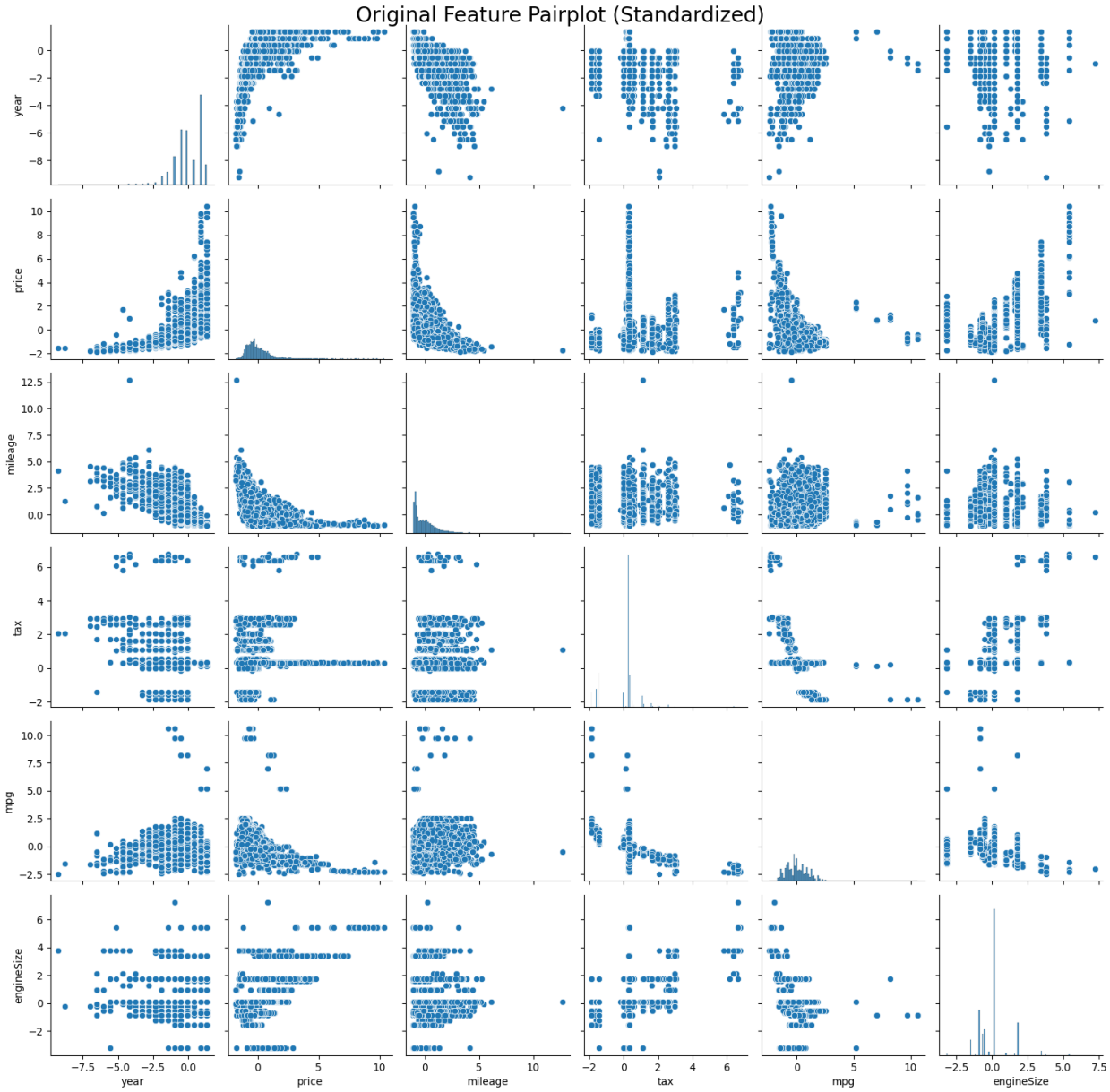
used to plot the individual covered variance for each principal component.



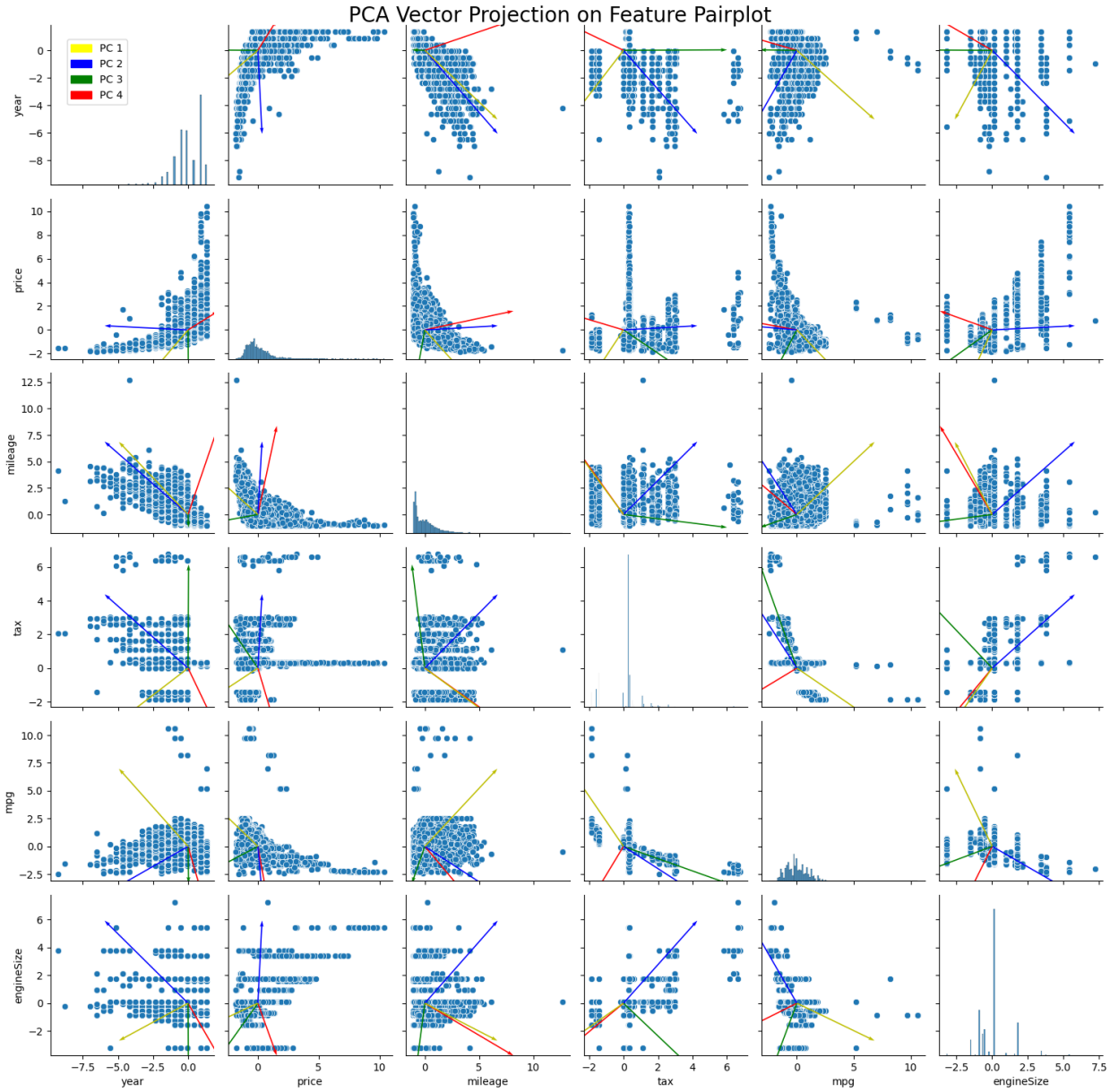
Percentage variance covered by each component: 75.2029582317495 , 19.34939620882094 , 4.619813761720772, 0.6611179152981342, 0.1667138824106361, 5.424387423795344e-15

6. Visualization using Pair Plots:

The code generates a pairplot of the dataset. The dataset is standardized by subtracting the mean and dividing by the standard deviation for each column. The pairplot shows pairwise relationships between the features of the dataset. The diagonal plots show the distribution of each feature.



The quiver function is used to plot the projection of the PCA vectors on the feature pairplot. The color of the arrows corresponds to the principal component (PC) number.



The graph shows that the data is clustered into two groups. The first group is characterized by high values on PC1 and PC2, and corresponds to cars that are newer, have lower mileage, are more expensive, and have larger engines. The second group is characterized by low values on PC1 and PC2, and corresponds to cars that are older, have higher mileage, are less expensive, and have smaller engines.

Here are some specific insights that can be drawn from the graph:

- Newer cars with lower mileage and larger engines tend to be more expensive.

- Older cars with higher mileage and smaller engines tend to be less expensive.
- There is a relatively clear separation between the two groups of cars, suggesting that there are two distinct segments in the car market.
- The first segment is likely targeted at consumers who are looking for newer, more performance-oriented cars.
- The second segment is likely targeted at consumers who are looking for more affordable, fuel-efficient cars.

It is important to note that the PCA projection is a linear transformation of the original data. This means that the relationships between the variables in the PCA space may not be the same as the relationships between the variables in the original data space. However, the PCA projection can still be a useful tool for visualizing high-dimensional data and identifying patterns and trends.

7. Conclusion and Interpretation:

Finally, We observe that 3 Principal Components are sufficient to cover over 99% variance of the centered data. Using these principal components is much more efficient than using the original features, since they are fewer than the original features, we can build complex models faster. From the visualizations we observe that the principal components cover the main trends being followed in the data accurately.

Assignment 2-B

PCA Analysis and Determining Optimal Number of Components

Steps:

1. Exploratory Data Analysis (EDA):

Here we initially load the given data (Hitters.cs). Then we check the null values in the dataset.

```
#   Column      Non-Null Count  Dtype
---  ------     --------------  -----
 0   AtBat      322 non-null    int64
 1   Hits       322 non-null    int64
 2   HmRun      322 non-null    int64
 3   Runs       322 non-null    int64
 4   RBI        322 non-null    int64
 5   Walks      322 non-null    int64
 6   Years      322 non-null    int64
 7   CAtBat     322 non-null    int64
 8   CHits      322 non-null    int64
 9   CHmRun     322 non-null    int64
 10  CRuns      322 non-null    int64
 11  CRBI       322 non-null    int64
 12  CWalks     322 non-null    int64
 13  League     322 non-null    object
 14  Division   322 non-null    object
 15  PutOuts    322 non-null    int64
 16  Assists    322 non-null    int64
 17  Errors     322 non-null    int64
 18  Salary     263 non-null    float64
 19  NewLeague  322 non-null    object
```

Features and their descriptions and types (Salary is the column to be predicted):

1. AtBat: Number of times at bat (numerical)
2. Hits: Number of hits (numerical)
3. HmRun: Number of home runs (numerical)
4. Runs: Number of runs in (numerical)
5. RBI: Number of runs batted (numerical)
6. Walks: Number of walks (numerical)
7. Years: Number of years in the major leagues (numerical)
8. CAtBat: Number of times at bat during his career (numerical)
9. CHits: Number of hits during his career (numerical)
0. CHmRun: Number of home runs during his career (numerical)
1. CRuns: Number of runs during his career (numerical)
2. CRBI: Number of runs batted in during his career (numerical)
3. CWalks: Number of walks during his career (numerical)
4. League: A factor with levels A and N indicating player's league (categorical)
5. Division: A factor with levels E and W indicating player's division (categorical)
6. PutOuts: Number of put outs (numerical)
7. Assists: Number of assists (numerical)
8. Errors: Number of errors (numerical)
9. Salary: Annual salary on opening day in thousands of dollars (numerical)
0. NewLeague: A factor with levels A and N indicating player's league (categorical)

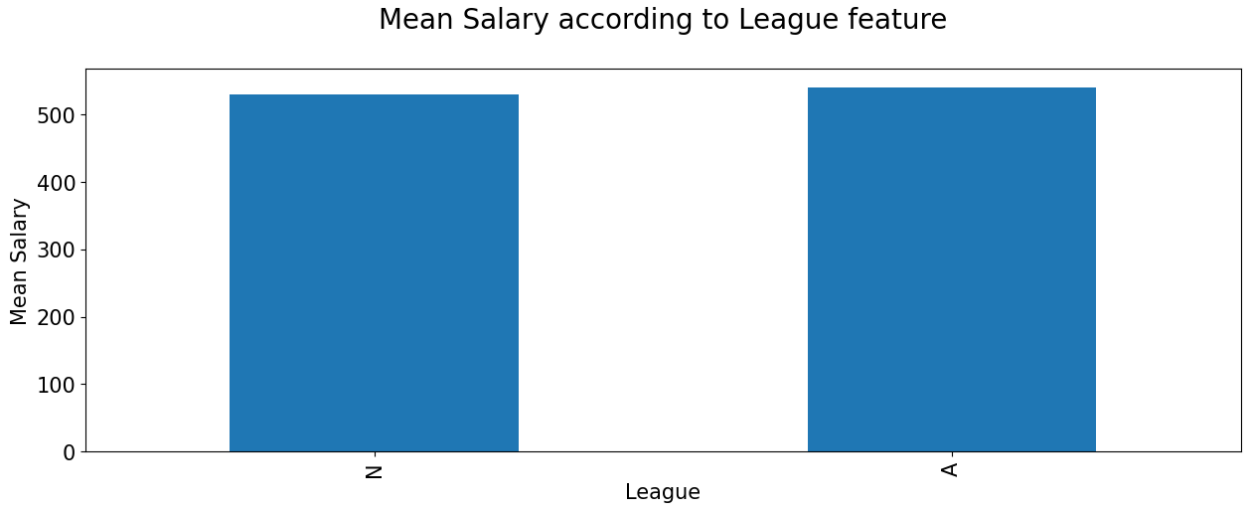
We then handle null value in the salary column with the mean of the salary columns without taking into account the null values.

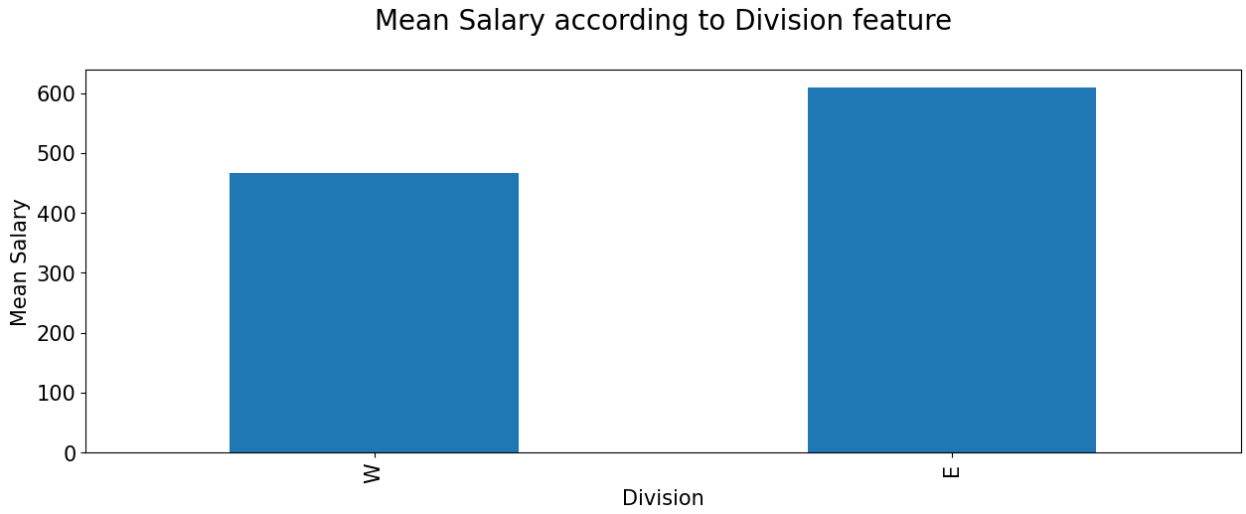
Description of the data:

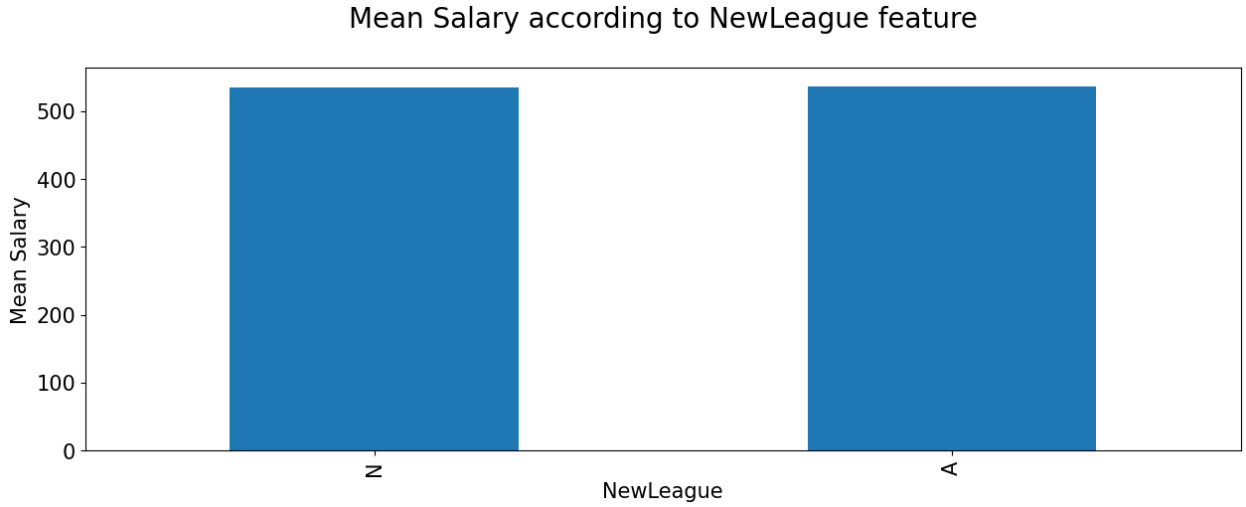
	count	mean	std	min	25%	50%	75%	max
AtBat	322.0	380.928571	153.404981	16.0	255.25	379.500000	512.00	687.0
Hits	322.0	101.024845	46.454741	1.0	64.00	96.000000	137.00	238.0
HmRun	322.0	10.770186	8.709037	0.0	4.00	8.000000	16.00	40.0
Runs	322.0	50.909938	26.024095	0.0	30.25	48.000000	69.00	130.0
RBI	322.0	48.027950	26.166895	0.0	28.00	44.000000	64.75	121.0
Walks	322.0	38.742236	21.639327	0.0	22.00	35.000000	53.00	105.0
Years	322.0	7.444099	4.926087	1.0	4.00	6.000000	11.00	24.0
CAtBat	322.0	2648.683230	2324.205870	19.0	816.75	1928.000000	3924.25	14053.0
CHits	322.0	717.571429	654.472627	4.0	209.00	508.000000	1059.25	4256.0
CHmRun	322.0	69.490683	86.266061	0.0	14.00	37.500000	90.00	548.0
CRuns	322.0	358.795031	334.105886	1.0	100.25	247.000000	526.25	2165.0
CRBI	322.0	330.118012	333.219617	0.0	88.75	220.500000	426.25	1659.0
CWalks	322.0	260.239130	267.058085	0.0	67.25	170.500000	339.25	1566.0
PutOuts	322.0	288.937888	280.704614	0.0	109.25	212.000000	325.00	1378.0
Assists	322.0	106.913043	136.854876	0.0	7.00	39.500000	166.00	492.0
Errors	322.0	8.040373	6.368359	0.0	3.00	6.000000	11.00	32.0
Salary	322.0	535.925882	407.557548	67.5	226.25	535.925882	700.00	2460.0

Exploring Categorical Features:

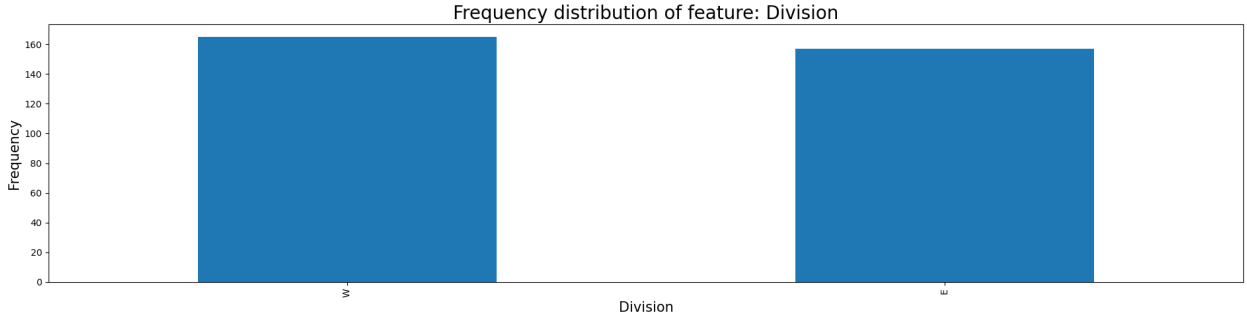
Effect on Target Value:





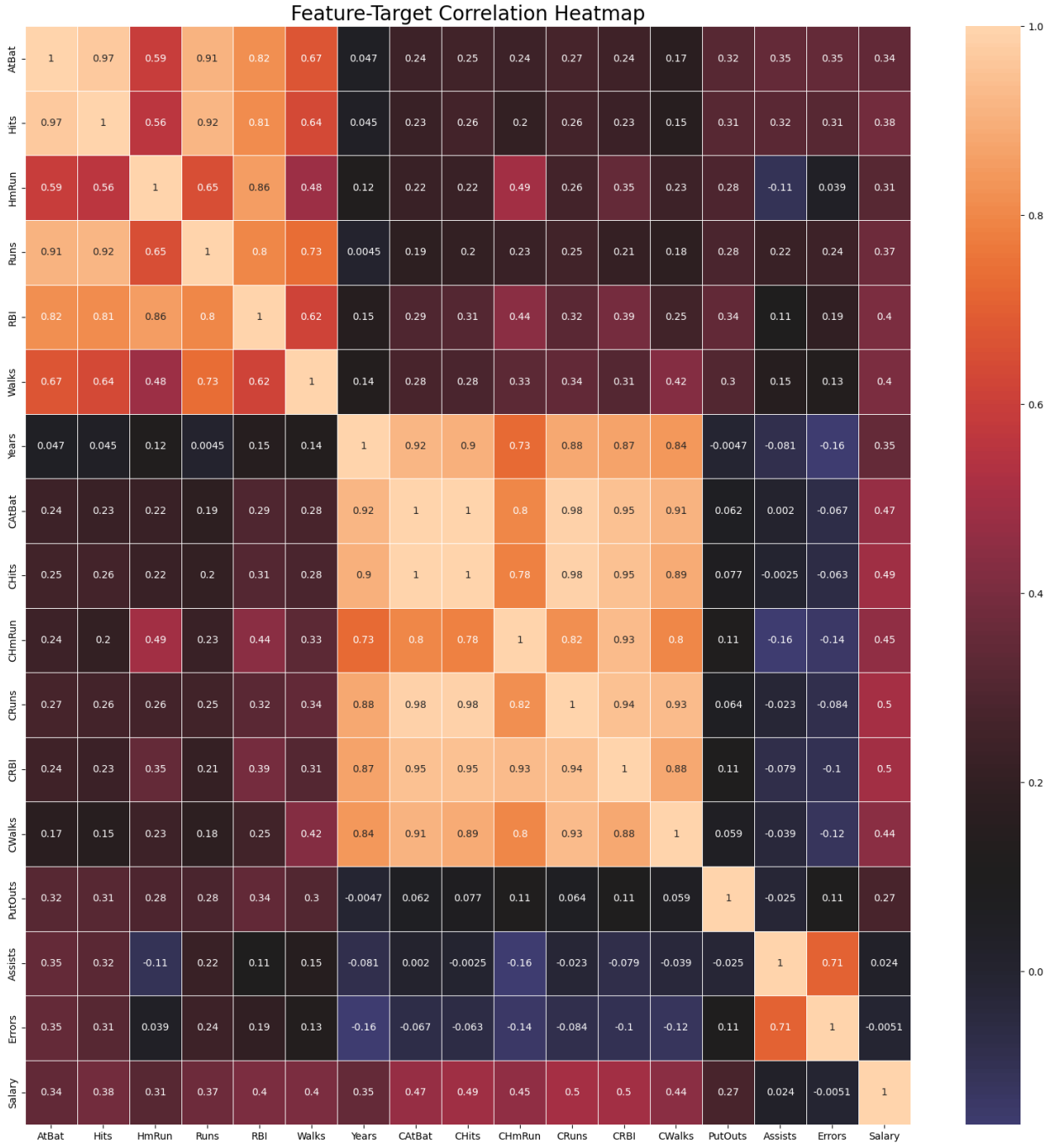


We see that League and NewLeague - Categorical features have negligible effect on the variable we have to predict. Thus, we drop them.



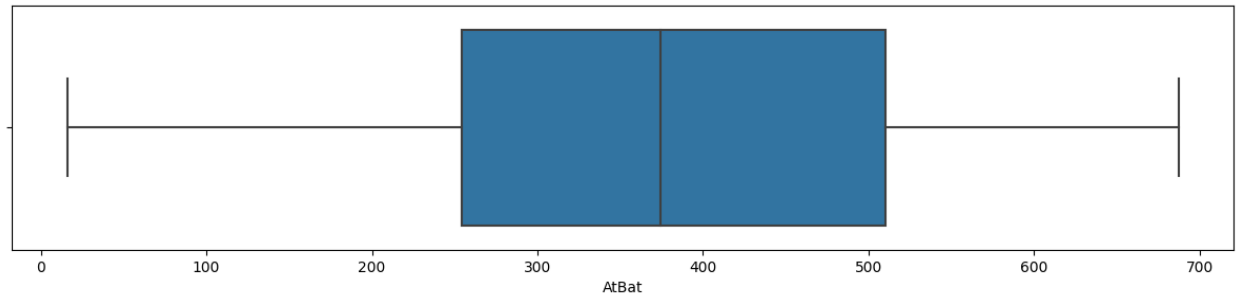
The above graph of frequency for the division feature suggests that this feature should not be removed.

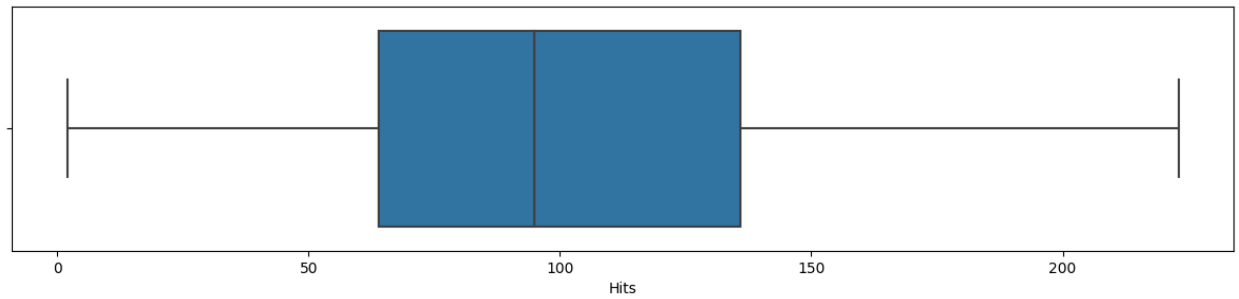
Correlation Heatmap:

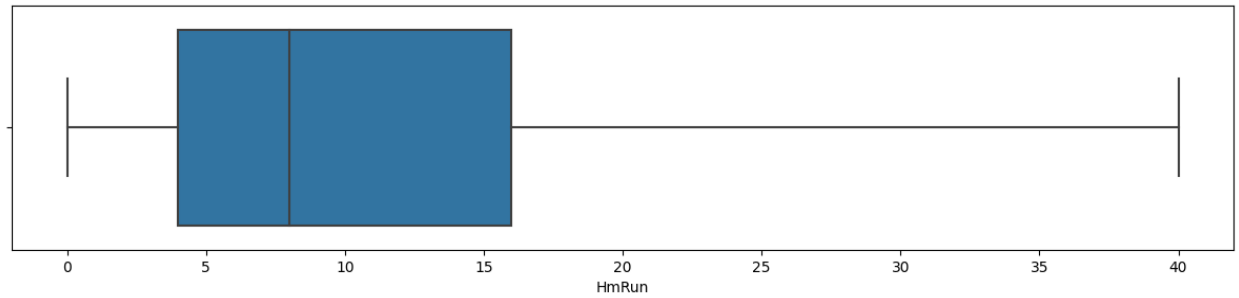


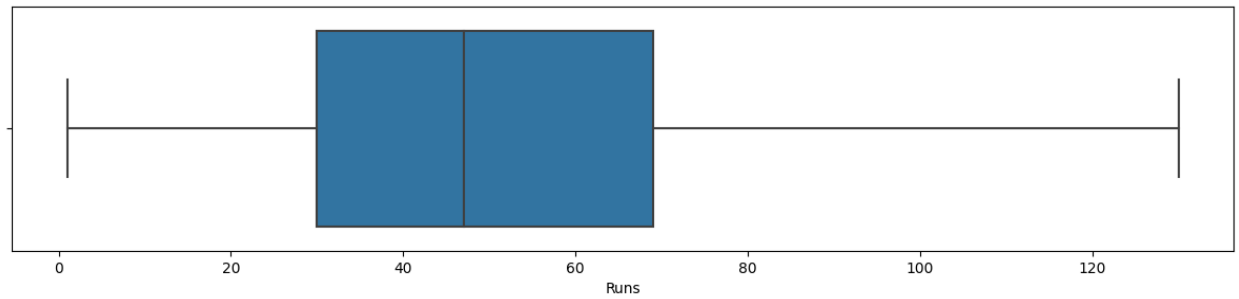
The above heatmap suggest that none o f the features are strongly correlated to the target.

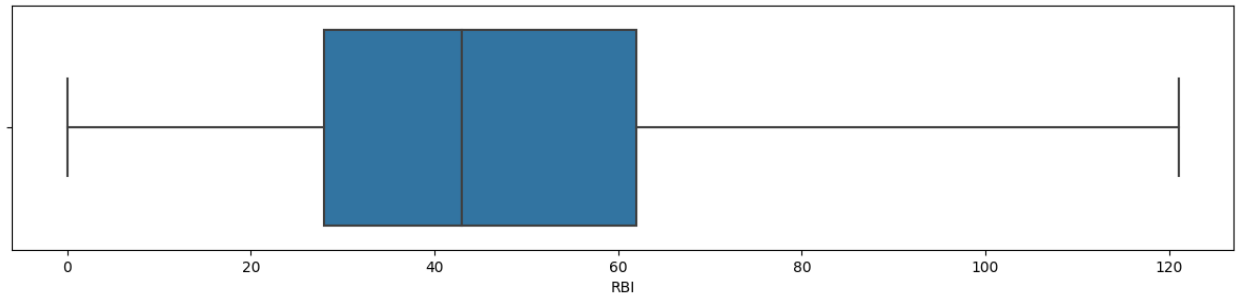
Detecting Outliers:

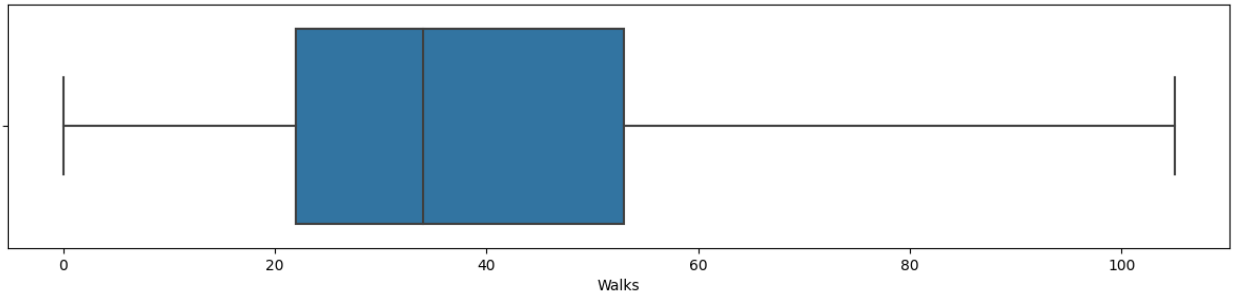


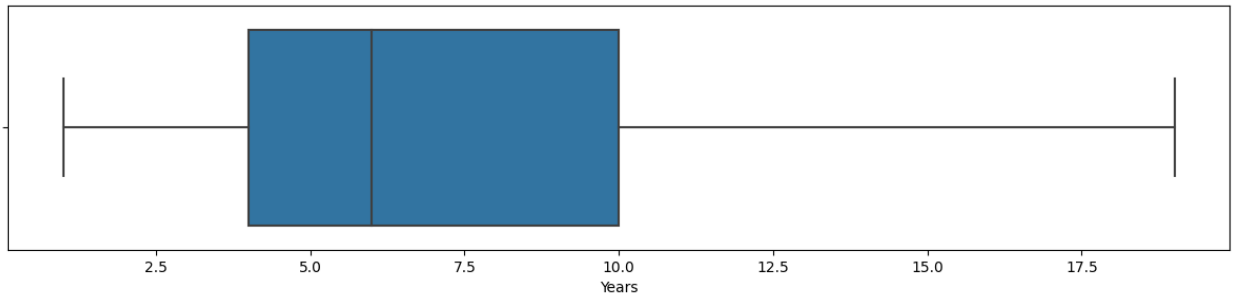


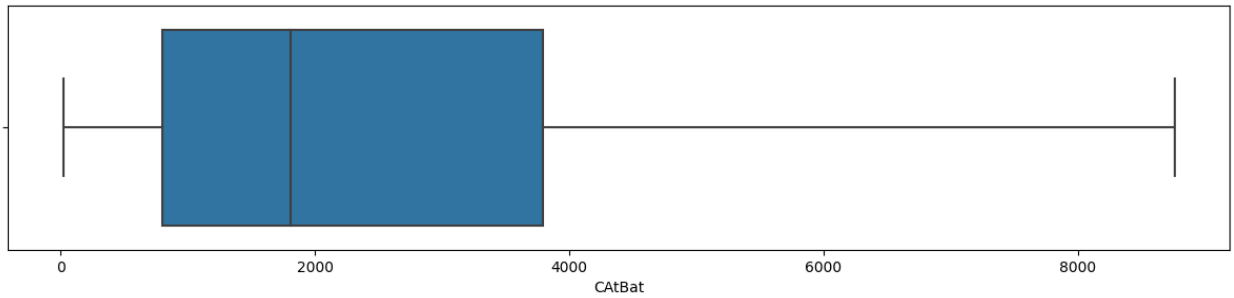


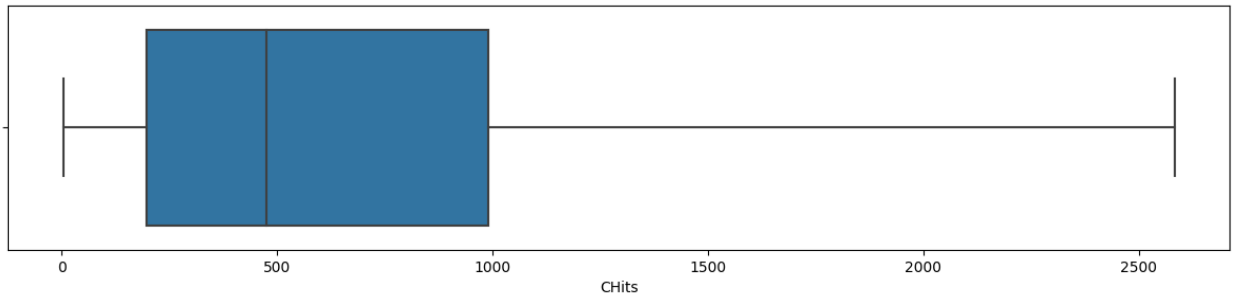


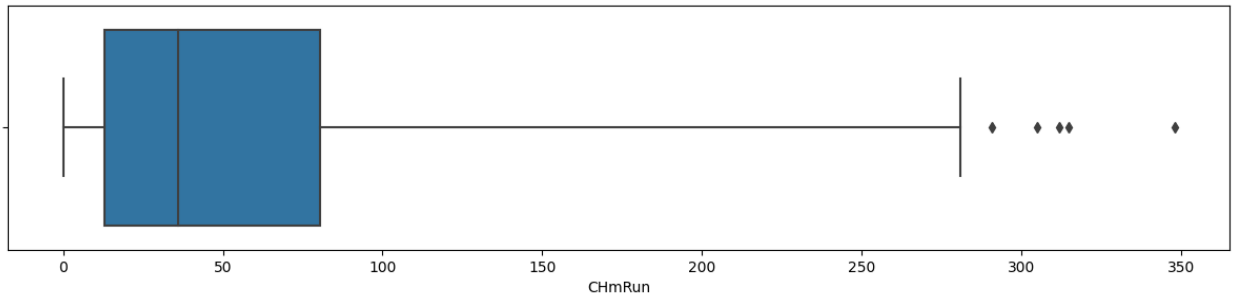


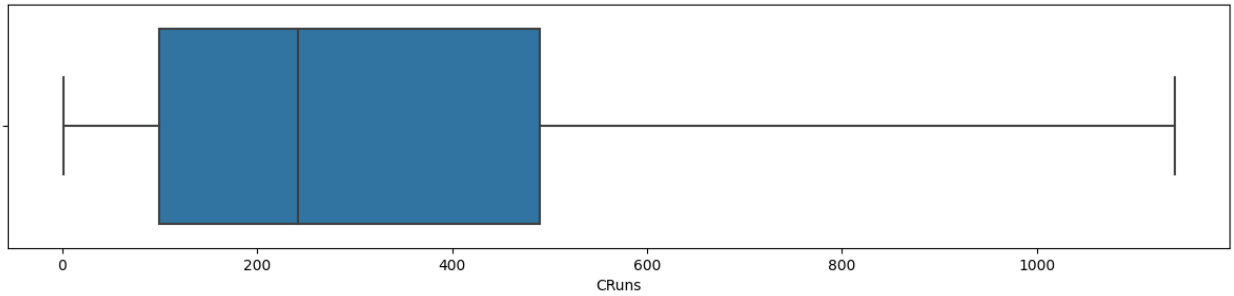

0
20
40
60
80
100
Walks

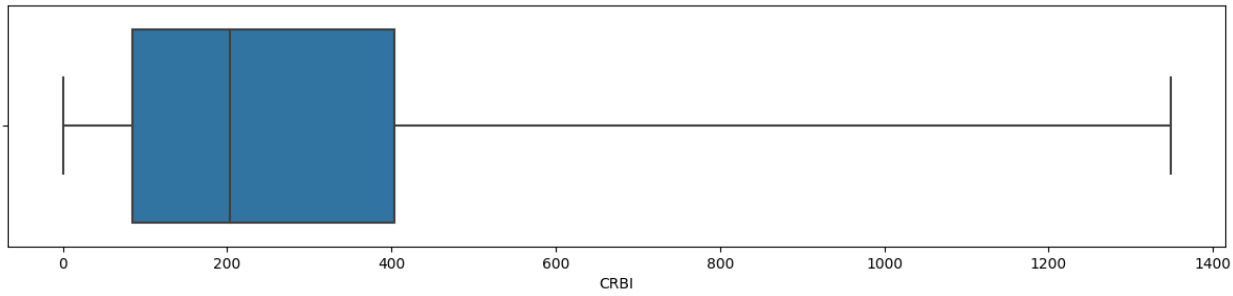

2.5
5.0
7.5
10.0
12.5
15.0
17.5
Years

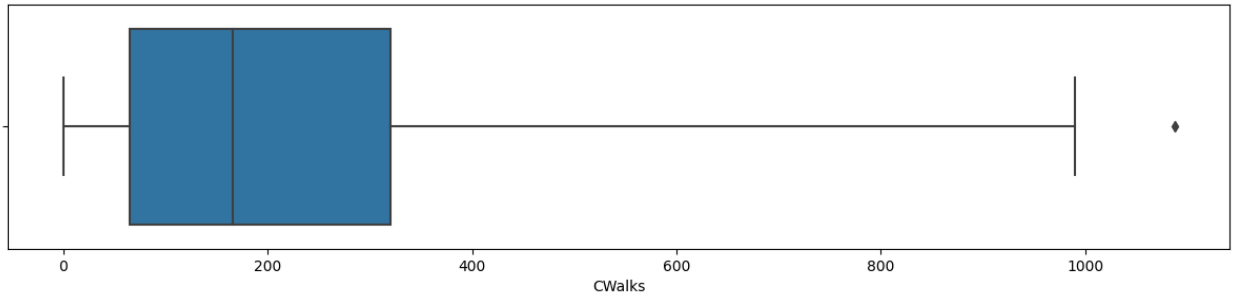

0
2000
4000
6000
8000
CAtBat

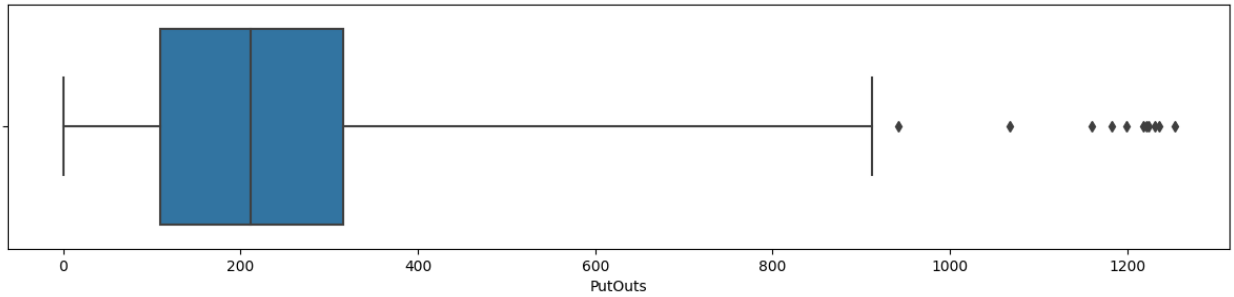

0
500
1000
1500
2000
2500
CHits

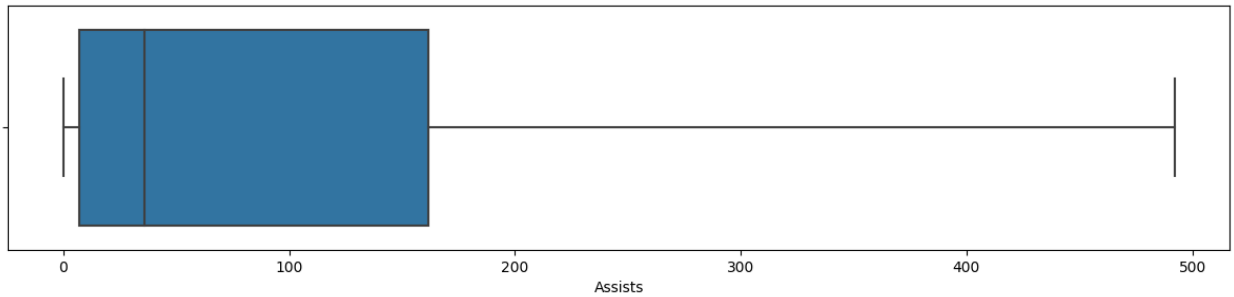

0
50
100
150
200
250
300
350
CHmRun

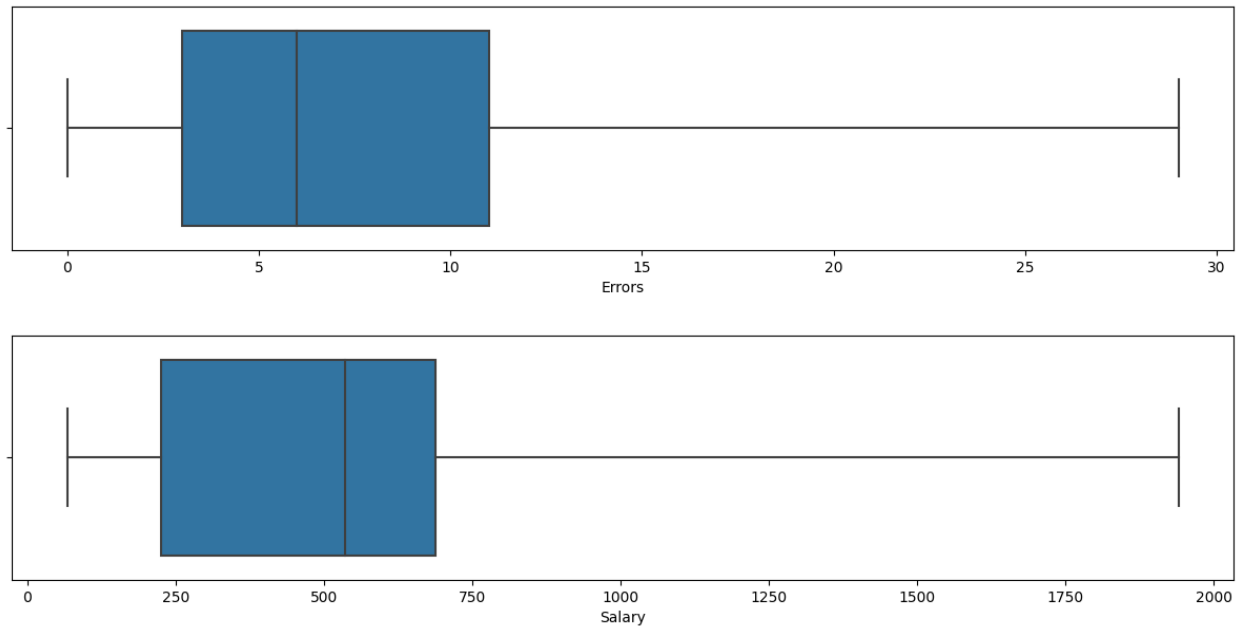

0
200
400
600
800
1000
CRuns


0
200
400
600
800
1000
1200
1400
CRBI


0
200
400
600
800
1000
CWalks


0
200
400
600
800
1000
1200
PutOuts


0
100
200
300
400
500
Assists



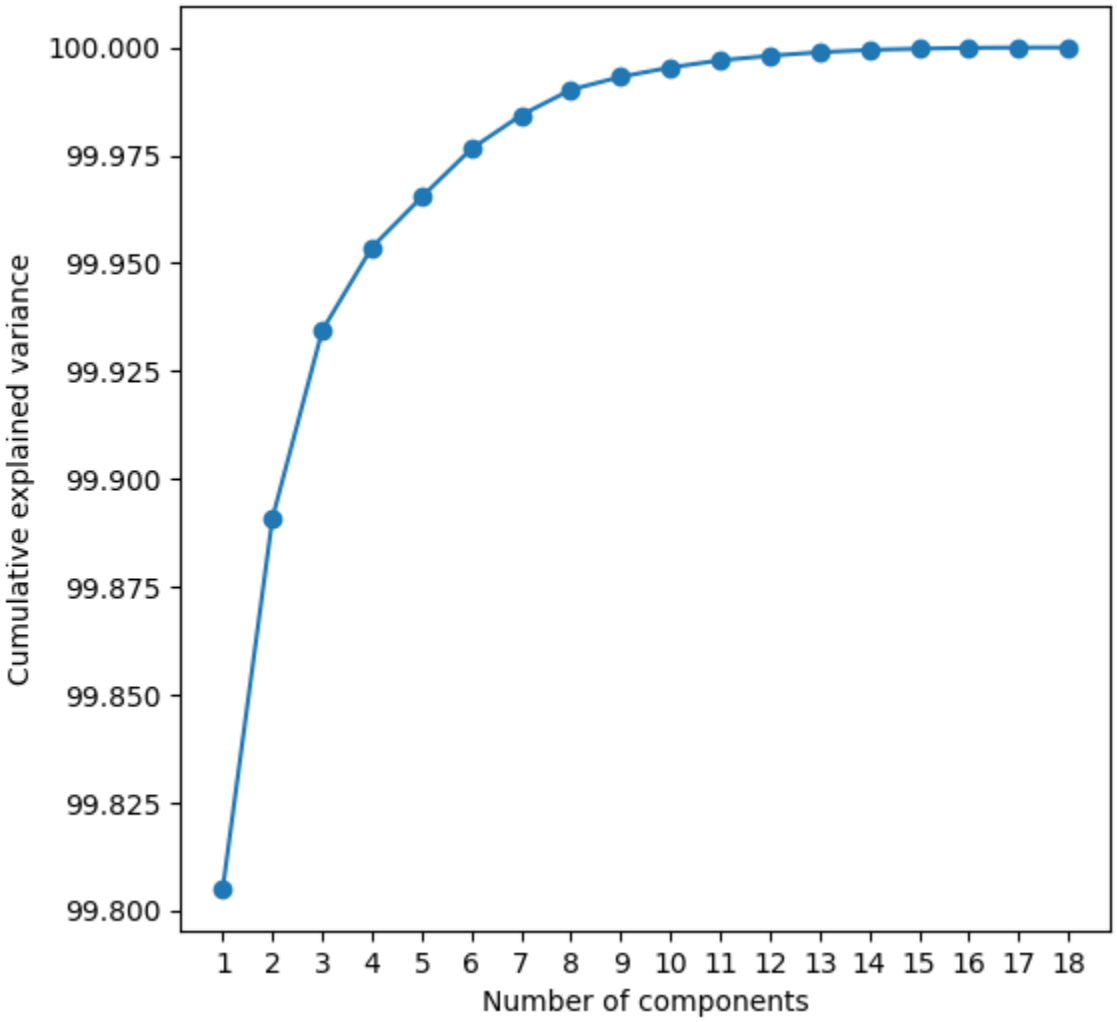
From the above box plot determine the outliers. We remove the outliers and reduce the data by approximately 5 percent.

Now, we convert the division feature into a numerical feature using 1 hot-encoding.
Now we standardize each feature (column) in the DataFrame dataset except for the one named "Salary." It iterates through all columns and for each column that is not named "Salary," it standardizes the data by subtracting the mean and dividing by the standard deviation. This is a common preprocessing step in data analysis to scale features and make them comparable when they have different scales or units.
Finally, we save the preprocessed data.

2. PCA Analysis:

The code reads the CSV file into a pandas DataFrame using the pd.read_csv function.
It then calculates the covariance matrix of the features in the dataset using the np.cov function.
The eigenvalues and eigenvectors of the covariance matrix are then computed using the np.linalg.eigh function.

The code then calculates the percentage of variance covered by each principal component using the formula (i/var_tot)*100, where i is the eigenvalue and var_tot is the total variance.
Finally, the code plots the cumulative explained variance against the number of principal components using the matplotlib.pyplot library.



Number of Principal Components Required for Efficient Prediction:
We observe that after 3-5 Principal Components covered variance increases slowly (begin to stagnate). Thus, 3-5 components are enough for our models.

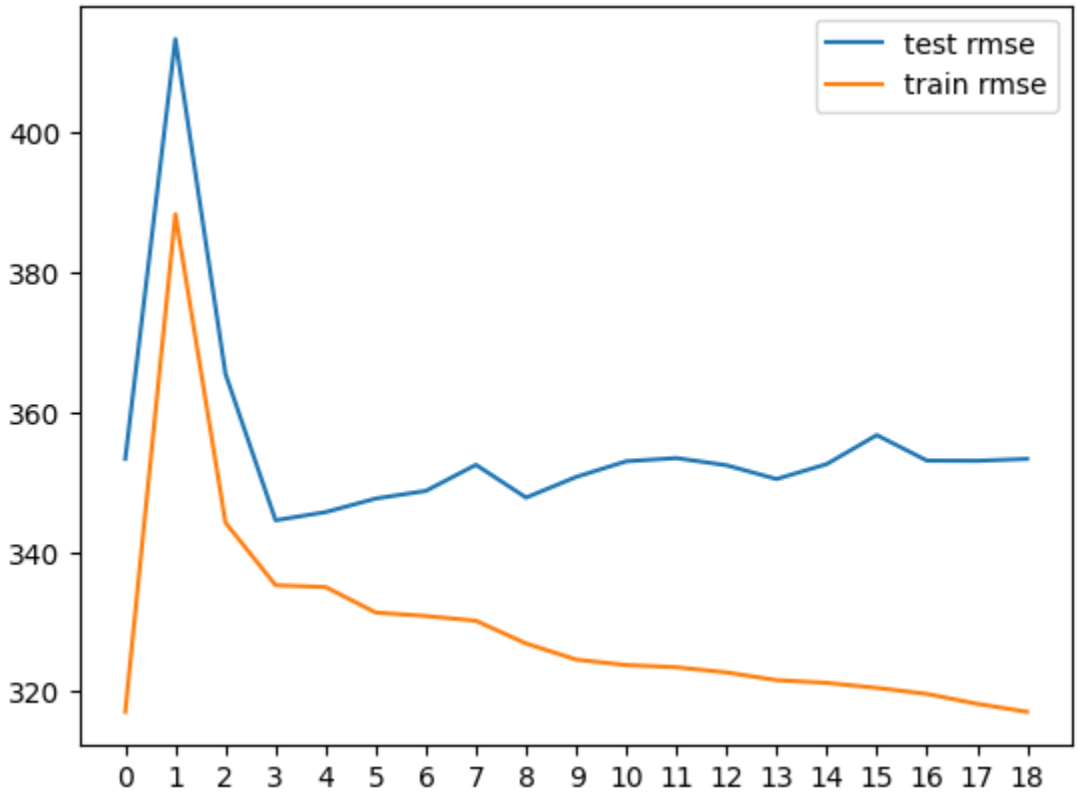
3. Model Training and MSE/RMSE Calculation:

The function first sets the seed for the random number generator using the random_state parameter. It then shuffles the indices of the dataset using the np.random.permutation function.

The function then calculates the number of samples to include in the test set based on the test_size parameter. It then separates the shuffled indices into test and train indices based on the calculated test size. Finally, the function returns the test and train sets based on the shuffled indices.

4. Plotting Number of Components vs RMSE:

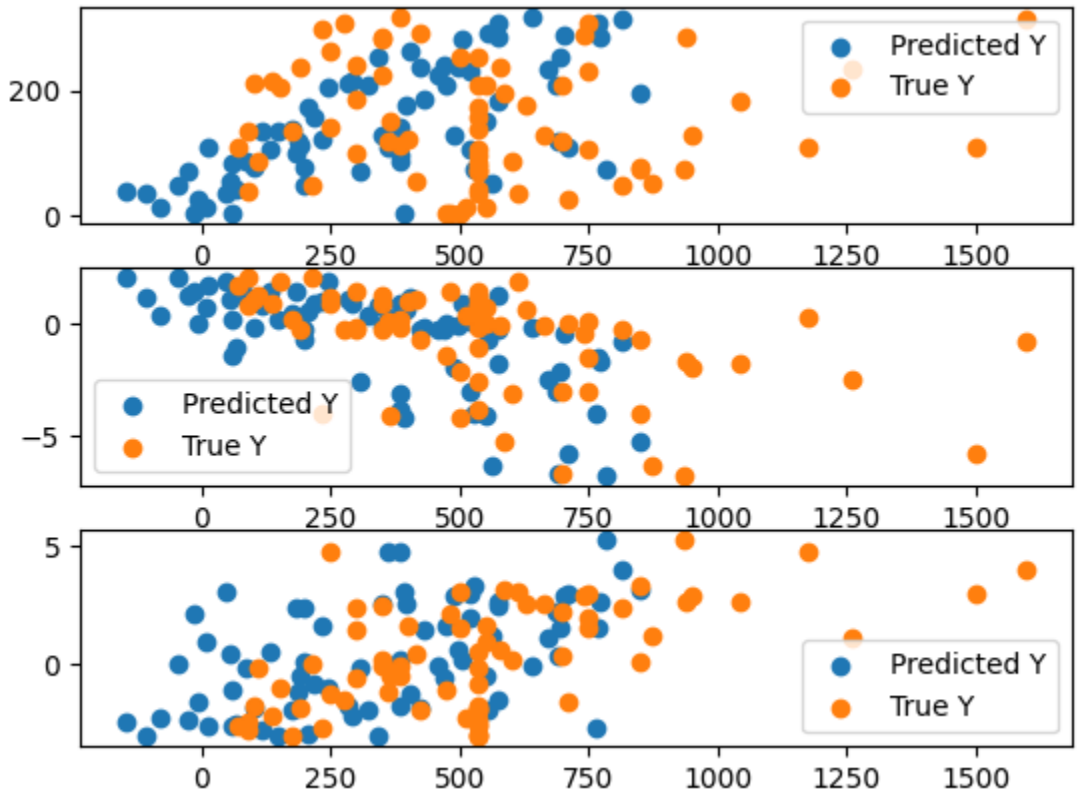
Here we computes the root mean squared error (RMSE) for a linear regression model on the preprocessed dataset. The linear_regressor function computes the RMSE for the test and training sets using 5-fold cross-validation. The X and Y variables contain the feature values and true values for the dataset, respectively. The matrix variable contains the covariance matrix of the dataset. The eig_values and eig_vectors variables contain the eigenvalues and eigenvectors of the covariance matrix, respectively. The a variable contains a range of values to use for the number of principal components. The errs variable contains the RMSE for each number of principal components. The code then plots the RMSE for the test and training sets as a function of the number of principal components. This code is useful for evaluating the performance of a linear regression model with PCA on the preprocessed dataset.



Identify the point where RMSE reaches a minimum:
RMSE (test) reaches a minimum at 3 principal components.

5. Testing the Most Efficient Model:

This code splits the dataset into training and testing sets using the test_train_split function. The X_test and Y_test variables contain the feature values and true values for the test set, respectively. The X_train and Y_train variables contain the feature values and true values for the training set, respectively. The W variable contains the weights learned by the linear regression model. The code then creates a scatter plot for each of the last three features in the test set. The x-axis of each scatter plot shows the predicted values of Y, while the y-axis shows the values of the corresponding feature. This code is useful for visualizing the performance of a linear regression model on the test set.
This code selects a random index i from the test set and prints the feature values, prediction, and true value for that index. The feature values are stored in X_test[i]. The prediction is computed by taking the dot product of X_test[i] and W using the @ operator. The true value is stored in Y_test[i]. This code is useful for evaluating the performance of a linear regression model on the test set.



Predicting the specific point:

Feature values: [1.59890663e+00 -6.40568076e-02 2.09001455e+02]
Prediction: 473.19004805562247
True value: 550.0

6. Conclusion and Analysis:

We observe that the model with 3 principal components is the most efficient model

Significance of Selecting an Appropriate Number of Components for Prediction Efficiency:

Too many or too little components hurt our model's final accuracy. Lower number of components also let our model to be trained faster. Too many components may lead to overfitting while too few lead to underfitting.

Analyze the predicted value from the chosen model and its significance:

The predicted values lie close to the true values on most principal components, although the model is not able to generalise enough for the outliers, also we notice a bias in the model towards the origin on all components.