# Part 2B

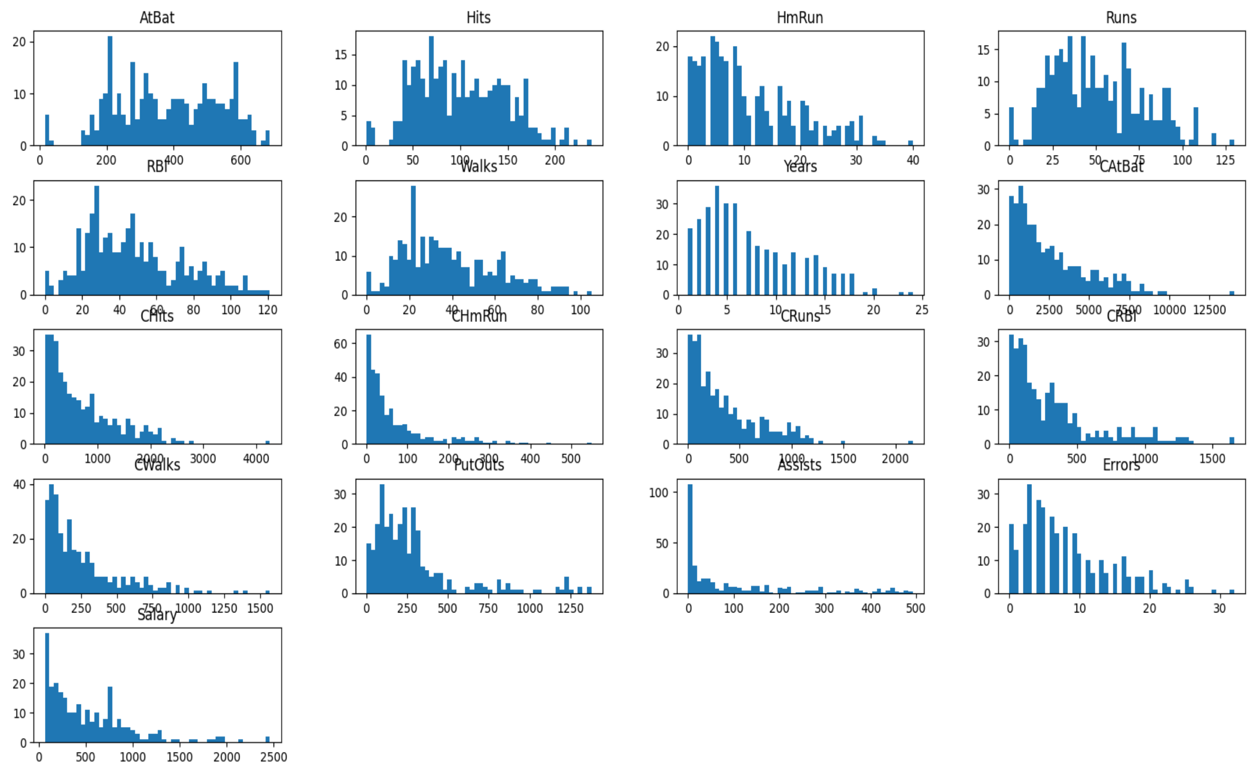
The assignment required us to apply regression model after applying PCA and choosing a model with those components which gave us the best result.

## STEP 1: Exploratory Data Analysis

Loading the dataset and handling NULL values:

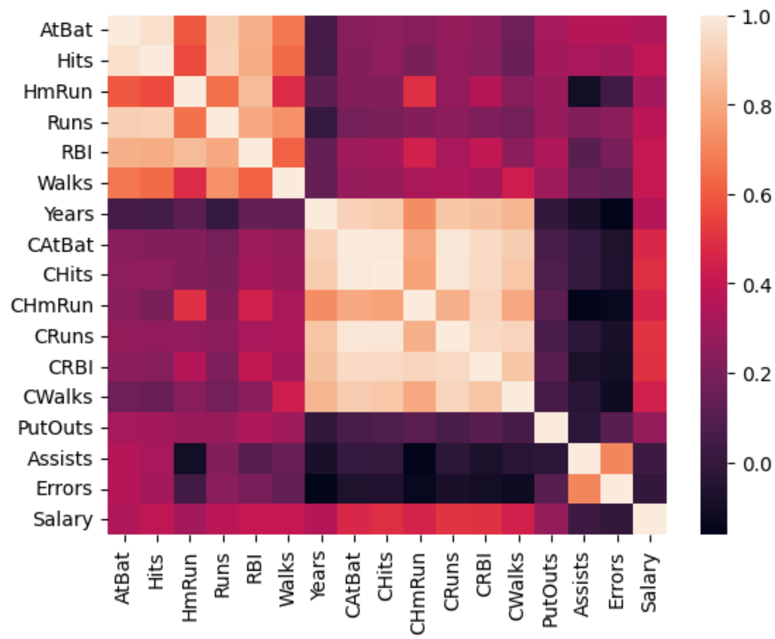
3 categorical columns were found in this dataset. The NULL values found in the ‘salary’ column which is our target variable here, the mean of the column used to replace such values.

The following image shows the distributions of individual numeric features in form of histograms.




We standardizing the data of the input columns and separate the input features and the output feature.

Following is the heatmap of the features




## STEP 2: PCA Analysis

We use the function written in the first part of assignment to apply PCA.

Following is the result with 4 principal components.

```
def covariance(x):
    return np.cov(x.T)

cov_mat = covariance(X)

from numpy.linalg import eig

eig_vals, eig_vecs = np.linalg.eig(cov_mat)
sorted_indices = np.argsort(eig_vals)[::-1]
eig_vals = eig_vals[sorted_indices]
eig_vecs = eig_vecs[:, sorted_indices]
# print('Eigenvalues \n', eig_vals)
# print('Eigenvectors \n', eig_vecs)

# Select top k eigenvectors
k =4
W = eig_vecs[:k, :] # Projection matrix

eig_vals_total = sum(eig_vals)
explained_variance = [(i / eig_vals_total) for i in eig_vals]
explained_variance = np.round(explained_variance, 2)
cum_explained_variance = np.cumsum(explained_variance)
```

```
X_proj = X.dot(W.T) ### Projected Data
X_proj
```

## STEP 3: Plotting number of components vs rmse

We used a custom function for splitting the training and testing datasets. Provided below is the code for that.

```
def custom_train_test_split(X, y, test_size, random_state):
    if random_state is not None:
        random.seed(random_state)
    num_samples = len(X)
    num_test = int(test_size * num_samples)
    indices = list(range(num_samples))
    random.shuffle(indices)
    test_indices = indices[:num_test]
    train_indices = indices[num_test:]
    X_train = [X[i] for i in train_indices]
    Y_train = [y[i] for i in train_indices]
    X_test = [X[i] for i in test_indices]
    Y_test = [y[i] for i in test_indices]
    X_train = np.array(X_train)
    X_test = np.array(X_test)
    Y_train = np.array(Y_train)
    Y_test = np.array(Y_test)
    return X_train, X_test, Y_train, Y_test

def predict_Y( bias ,weights , features):
  return bias + np.dot(features, weights)

def get_cost(Y,Y_hat):
  rmse = np.sqrt(((Y - Y_hat) ** 2).mean())
  return rmse

def update_theta(x , y , y_hat , b_0 , theta_o , learning_rate):
  grad_b = (np.sum(y_hat-y))/len(y)
  grad_w = (np.dot((y_hat-y),x))/len(y)
  b_1 = b_0 - learning_rate*grad_b
  theta_1 = theta_o - learning_rate*grad_w
  return b_1 , theta_1
```

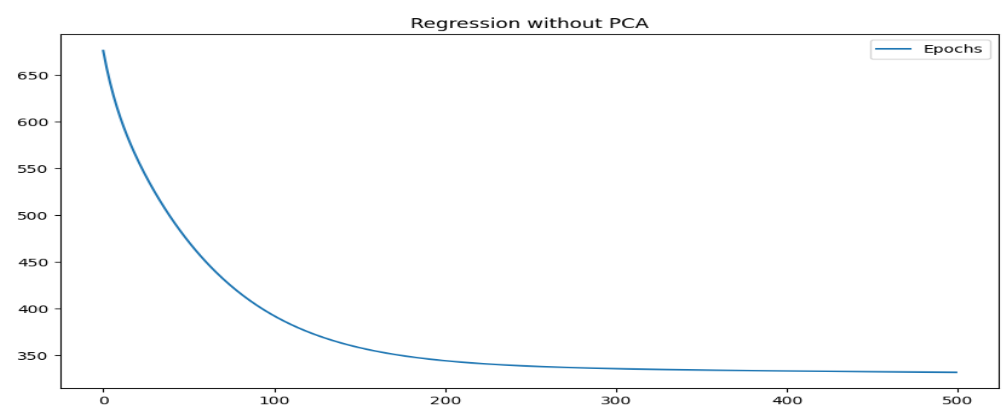
For comparison of models built with and without using PCA on data we first ran a linear regression algorithm with 16 features and stored the testing errors we got with that model.

The code for gradient descent algorithm used here is given below.

```
def run_gradient_descent(X,Y,alpha,num_iterations):
  b=random.random()
  theta=np.random.rand(X.shape[1])
  J = []
  for each_iter in range(num_iterations):

    Y_hat = predict_Y(b,theta,X)
    prev_b = b
    prev_theta = theta
    b,theta = update_theta(X,Y,Y_hat,prev_b,prev_theta,alpha)
    J.append(get_cost(Y,Y_hat))


  print("Final Estimate of b and theta : ",b,theta)
  return b,theta,J
```

Following are the results for that (we will need them when we want to compare with models built with different number of PCA components):

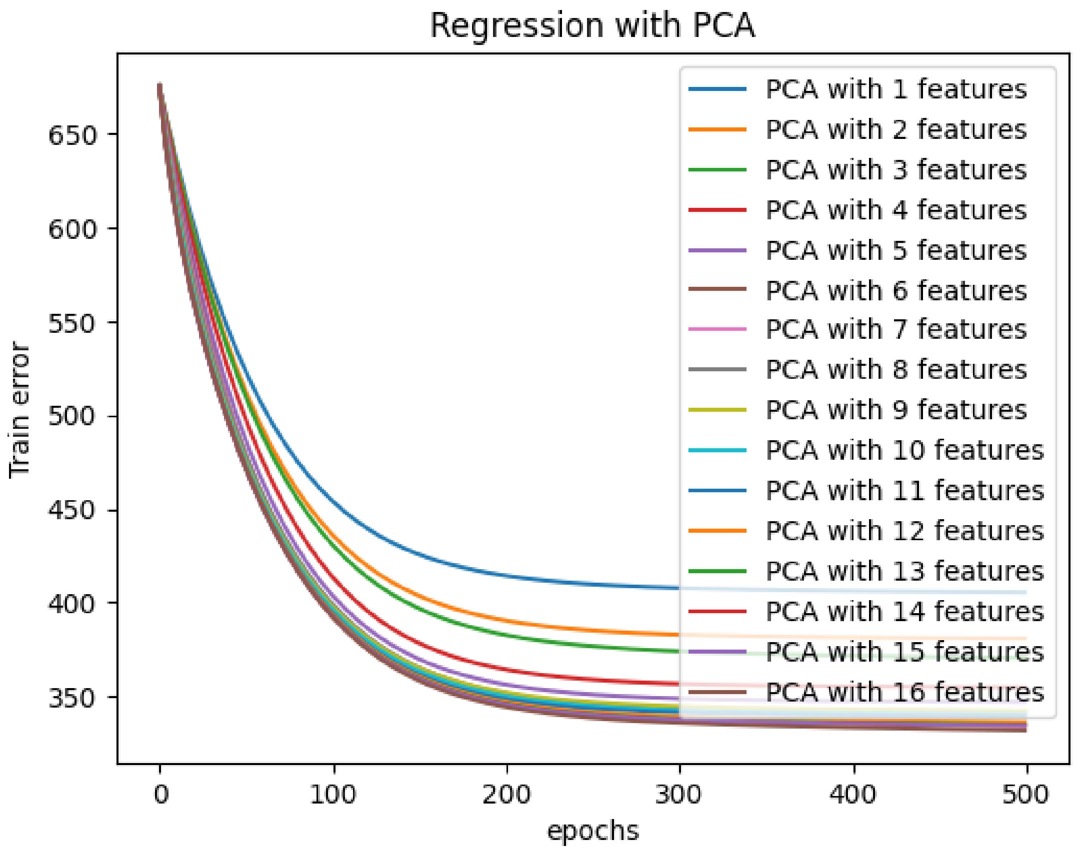
```
Final Estimate of b and theta :  532.2867483775136 [ 18.35550592  33.14322171
-23.94180993  35.64632964  66.17946863
  66.33658261  32.59701984  -2.46896789 -43.49828337 -31.05795737
 -49.31163819  47.3135223   56.61599876 -27.40959488  -3.1901271
  52.72960953]


Minimum Cost Function Value: 331.94233250498314
Test errors: 277.3728953580285
```



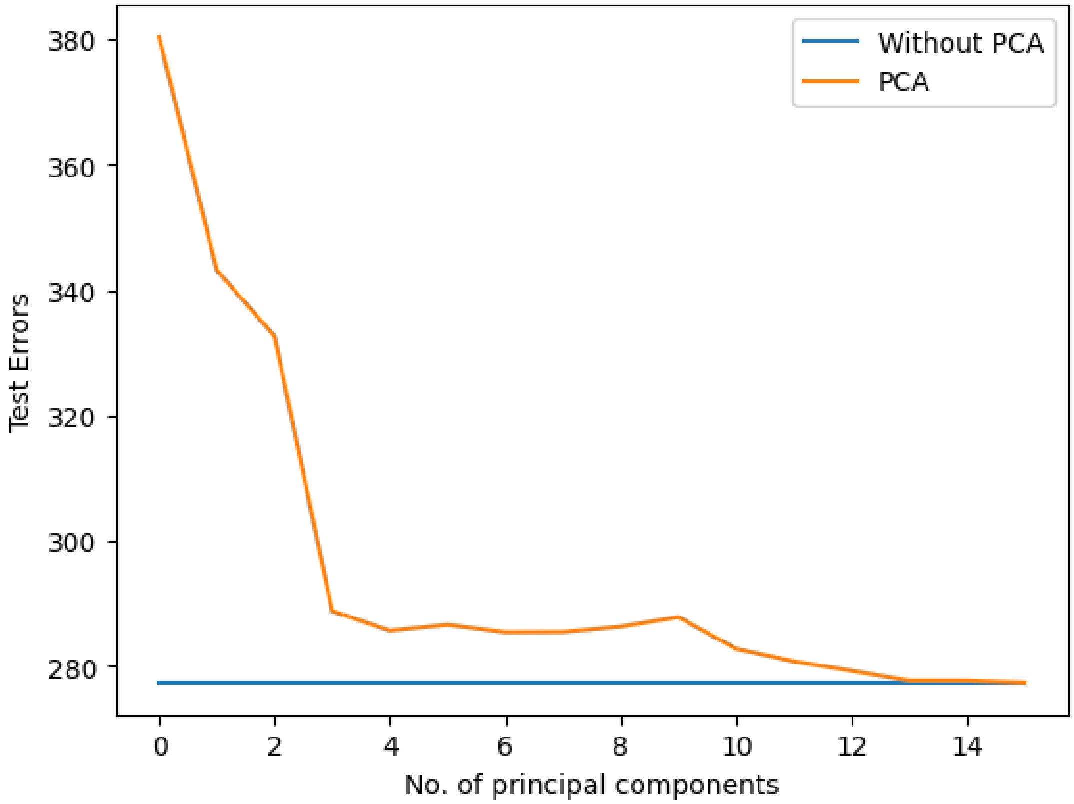

With different number of PCA components we ran the same regression model and got the following results.

```
Minimum Cost Function Value: 331.9189849201732
```

```
Test errors: 277.3817341183595
```




Hence, plotting the graph of testing errors vs the number of components for conclusion purposes: We are able to observe how, with different numbers of principal components, what is the Error Value of the Cost Function.

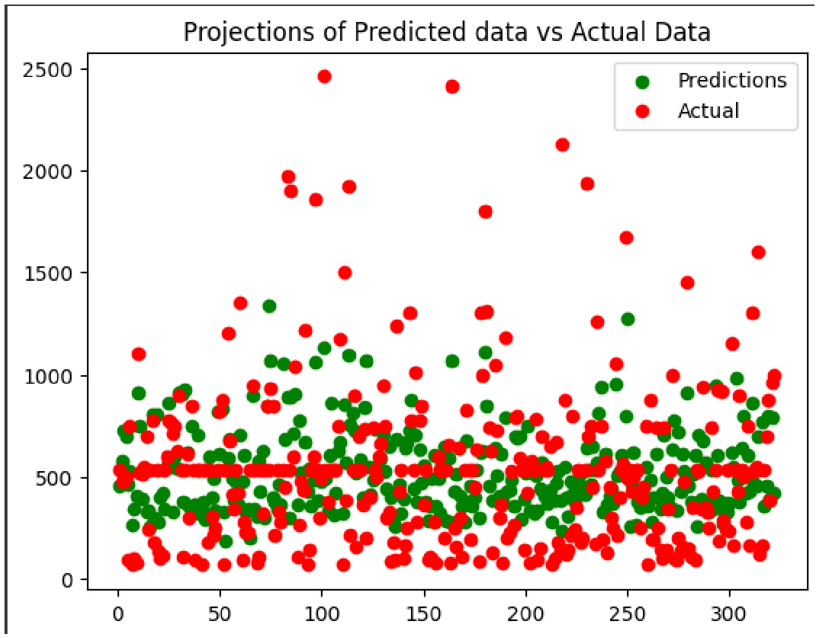



As we can see, the best reduction in testing errors we can get is with 4 principal components.

Therefore, the Best estimate of weights and bias would be:

```
The best estimate of Bias b : 531.4777759499187
The best estimate of Weights Thetas : [  63.09206595  -92.62463963 -106.42246516  150.01733765]
```

For this selected model that is considered the most optimum, we can see the projection of our predicted values vs the actual data -




Looking at the graph of Training Error vs No. of Principal Components, we could have selected the model that included 10 principal components, but we chose the model with 4 parameters in order to actually select components that could give the same information as the actual dataset and also reduce the number of components up to an extent.