HR Analytics Project- Understanding the Attrition in HR

Problem Statement

Every year a lot of companies hire a number of employees. The companies invest time and money in training those employees, not just this but there are training programs within the compnaies for their existing employees as well.

The objective of the model to increase the effectiveness of their employees and reduce the time and money investing in employees.

HR Analytics:

Human resource analytics (HR analytics) is an area in the field of analtyics that refers to applying analytic processes to the human resource department of an organization in the hope of improving employee performance and therefore getting a better return on investment. HR analtyics does not just deal with gathering data on employee efficiency. Instead, it aims to provide insight into each process by gathering data and then using it to make relevant decisions about how to improve these processes.

Attrition in HR

Attrition in human resources refers to the gradual loss of employees overtime. In general, relatively high attrition is problematic for companies. HR professionals often assume a leadership role in designing company compensation programs, work culture, and motivation systems that help the organization retain top employees

How does Attrition affect companies? and how does HR Analytics help in analysing attrition? We will discuss the first question here and for the second question, we will write the code and try to understand the process step by step.

Attrition affecting Companies

A major problem in high employee attrition is its cost to an organization. Job postings, hiring processes, paperwork, and new hire training are some of the common expenses of losing employees and replacing them. Additionally, regular employee turnover prohibits your organization from increasing its collective knowledge base and experience over time. This is especially concerning if your business is customer-facing, as customers often prefer to interact with familiar people. Errors and issues are more likely if you constantly have new workers.

Importing the Libraries

```
import pandas as pd
import numpy as np
import seaborn as sns
from scipy import stats
import warnings
warnings.filterwarnings('ignore')
from matplotlib import pyplot as plt
from scipy.stats import zscore
#data preprocessing
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
#Over Sampling the data using SMOTE
from imblearn.over_sampling import SMOTE
#modelling
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn .ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import cross_val_score
from matplotlib import pyplot
from sklearn.svm import SVC
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
```

EXPLORATORY DATA ANALYSIS

```
In [3]:  1 df.describe()
```

Out[3]:

Education	EmployeeCount	EmployeeNumber	EnvironmentSatisfaction	HourlyRate	JobInvolvement	JobLevel	...	RelationshipSatisfaction	StandardHours
1470.000000	1470.0	1470.000000	1470.000000	1470.000000	1470.000000	1470.000000	...	1470.000000	1470.0
2.912925	1.0	1024.865306	2.721769	65.891156	2.729932	2.063946	...	2.712245	80.0
1.024165	0.0	602.024335	1.093082	20.329428	0.711561	1.106940	...	1.081209	0.0
1.000000	1.0	1.000000	1.000000	30.000000	1.000000	1.000000	...	1.000000	80.0
2.000000	1.0	491.250000	2.000000	48.000000	2.000000	1.000000	...	2.000000	80.0
3.000000	1.0	1020.500000	3.000000	66.000000	3.000000	2.000000	...	3.000000	80.0
4.000000	1.0	1555.750000	4.000000	83.750000	3.000000	3.000000	...	4.000000	80.0
5.000000	1.0	2068.000000	4.000000	100.000000	4.000000	5.000000	...	4.000000	80.0

Employee's average number of years at company is 7.
Mean is not equal to median stating that the data is not normally distributed. Most normally distributes column is Daily rate where mean is almost equal to median

```
Out[4]: Age                         int64
        Attrition                  object
        BusinessTravel             object
        DailyRate                   int64
        Department                 object
        DistanceFromHome            int64
        Education                   int64
        EducationField             object
        EmployeeCount               int64
        EmployeeNumber              int64
        EnvironmentSatisfaction     int64
        Gender                     object
        HourlyRate                  int64
        JobInvolvement              int64
        JobLevel                    int64
        JobRole                    object
        JobSatisfaction             int64
        MaritalStatus              object
        MonthlyIncome               int64
        MonthlyRate                 int64
        NumCompaniesWorked          int64
        Over18                     object
        OverTime                   object
        PercentSalaryHike           int64
        PerformanceRating           int64
        RelationshipSatisfaction    int64
        StandardHours               int64
        StockOptionLevel            int64
        TotalWorkingYears           int64
        TrainingTimesLastYear       int64
        WorkLifeBalance             int64
        YearsAtCompany              int64
```

Numeric variables:

- Related to personal information: age, distance_from_home, employee_number
- Related to income: hourly_rate, daily_rate, monthly_rate, monthly_income, percent_salary_hike

Related to duration in company: years_at_company, years_in_current_role, years_since_last_promotion, years_with_curr_manager, total_working_years

num_companies_worked,standard_hourstraining_times_last_year, employee_count

Categorical variables:

- Binary variables: attrition(target variable), gender, over18, over_time
- Nominal variables: department, education_field, job_role, marital_status
- Ordinal variables:

- Ordinal regarding satisfaction and performance:environment_satisfaction,job_satisfaction, relationship_satisfaction,work_life_balance,job_involvement,performance_rating
- Other ordinal: business_travel, education, job_level, stock_option_level

__Handling null values

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

```
Out[5]: Age                          0
        Attrition                    0
        BusinessTravel               0
        DailyRate                    0
        Department                   0
        DistanceFromHome             0
        Education                    0
        EducationField               0
        EmployeeCount                0
        EmployeeNumber               0
        EnvironmentSatisfaction      0
        Gender                       0
        HourlyRate                   0
        JobInvolvement               0
        JobLevel                     0
        JobRole                      0
        JobSatisfaction              0
        MaritalStatus                0
        MonthlyIncome                0
        MonthlyRate                  0
        NumCompaniesWorked           0
        Over18                       0
        OverTime                     0
        PercentSalaryHike            0
        PerformanceRating            0
        RelationshipSatisfaction     0
        StandardHours                0
```

This dataset has no null values

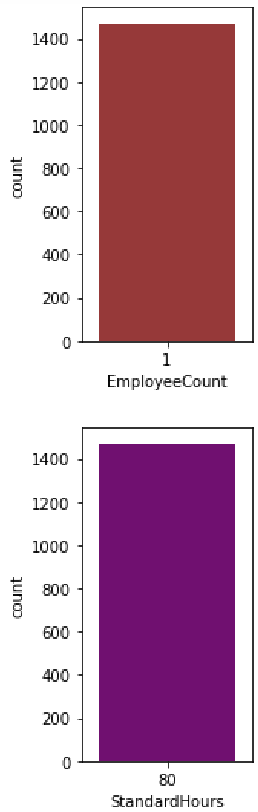
```
In [24]:  for col in df:
              print(col)
              print(df[col].value_counts())
              print()
```

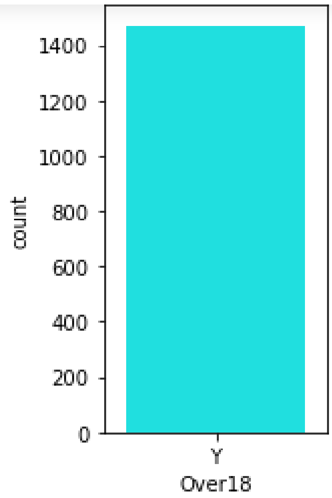
```
1    1470
Name: EmployeeCount, dtype: int64

EmployeeNumber
2048    1
1368    1
1364    1
1363    1
1362    1
        ..
648     1
647     1
645     1
644     1
2046    1
Name: EmployeeNumber, Length: 1470, dtype: int64
```

Displaying value count of unique value in each feature. To identify column having single unique value

UNI VARIATE ANALYSIS





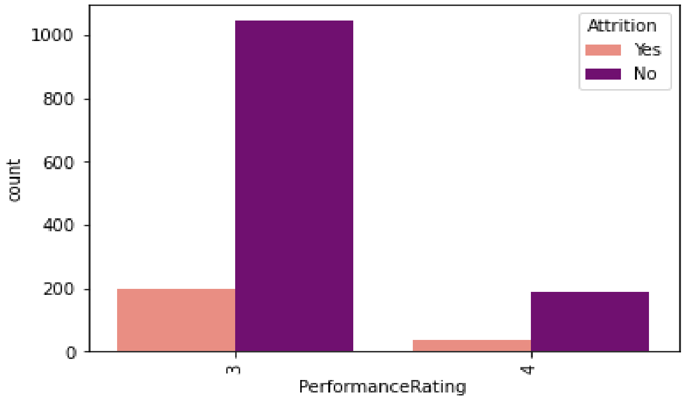
The features standard hours, over18 and employee count has only one value so it won’t create any impact on the target feature Attrition.

All the three columns having single value so I'm going to dropping it from the given dataset

BI VARIATE ANALYSIS (Categorical columns vs Target**)**

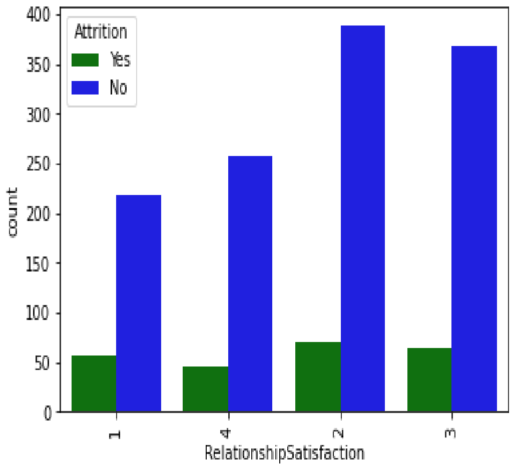
```
l = list(df['PerformanceRating'].unique())
col_l=["salmon","purple"]
chart = sns.countplot(df["PerformanceRating"],palette=col_l,hue=df.Attrition)
chart.set_xticklabels(labels=l, rotation=90)
```

```
[Text(0, 0, '3'), Text(1, 0, '4')]
```

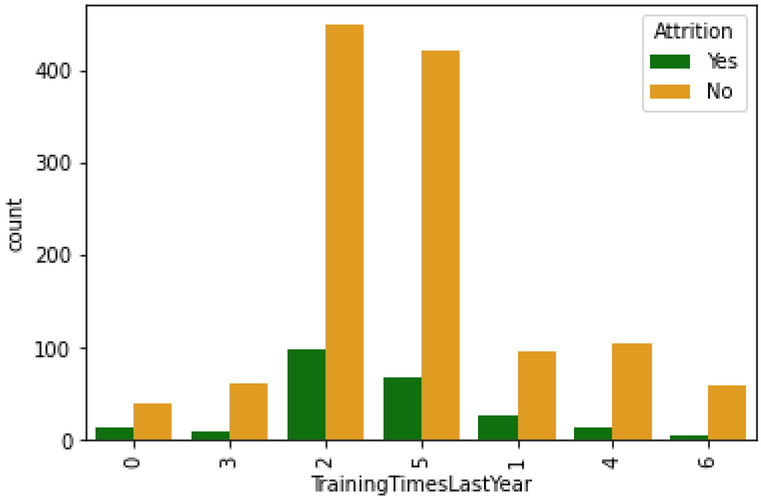



```
the employees who got performance rating as 3 are having less attrition rate.
```

```
[Text(0, 0, '1'), Text(1, 0, '4'), Text(2, 0, '2'), Text(3, 0, '3')]
```

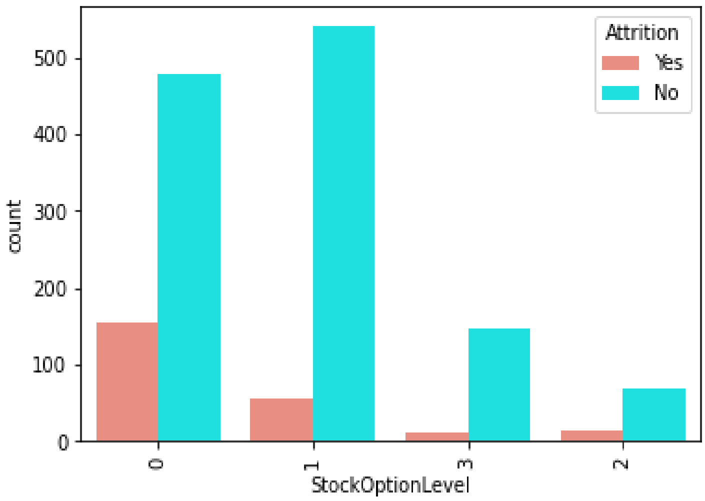


from the above chart its apparent that the employee who having high relationship satisfaction and low relationship satisfaction are having low Attrition rate.so we cannot predict the target coloumn with this value

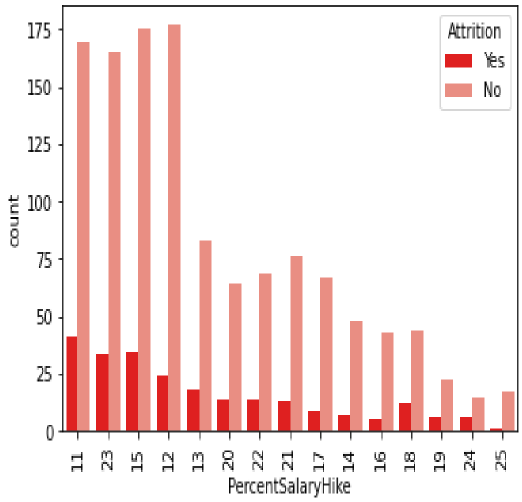


Employee who are trained 2 and 5 times a year are having less attrition rate

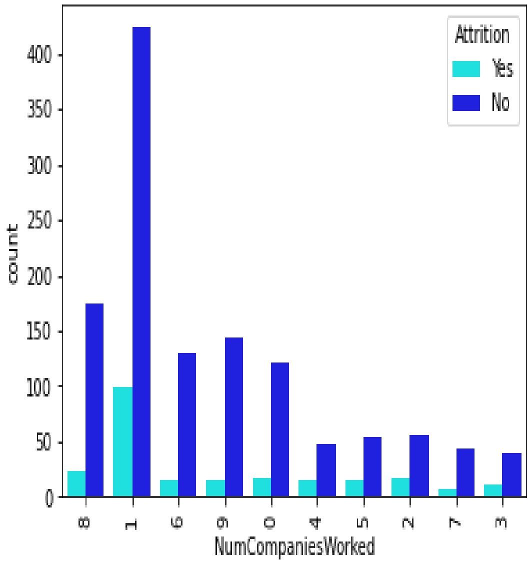
```
]: [Text(0, 0, '0'), Text(1, 0, '1'), Text(2, 0, '3'), Text(3, 0, '2')]
```



the employees who having less stocks are having low attrition rate.



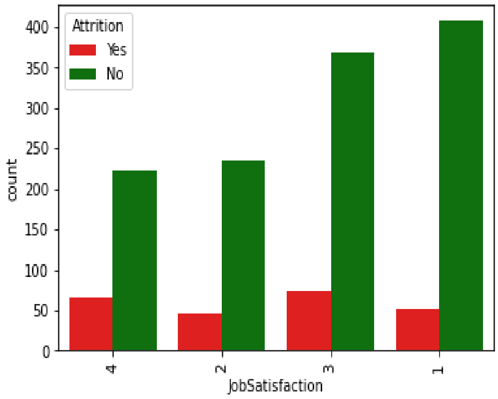
the employees who all are got hike 11 to 15% are having less attrition rate.the employees who got hike between 18 t0 20% having high attrition rate .we can predict the target column using this feature



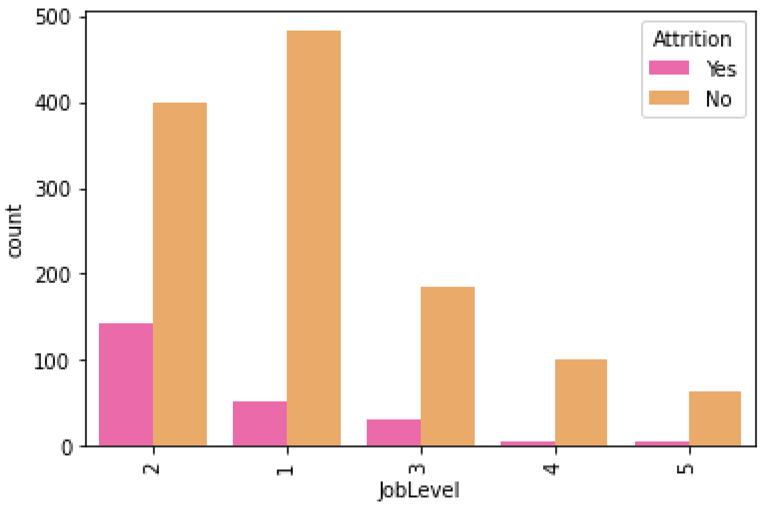
from the above chart its apparent that both the employee who having high satisfaction and low satisfaction are having low Attrition rate.so we cannot predict the target coloumn with this value

from the above chart its apparent that the employees who worked in only one company are having low Attrition rate.

```
[Text(0, 0, '4'), Text(1, 0, '2'), Text(2, 0, '3'), Text(3, 0, '1')]
```



from the above chart its apparent that the employee who having high job satisfaction and low job satisfaction are having low Attrition rate.so we cannot predict the target coloumn with this value

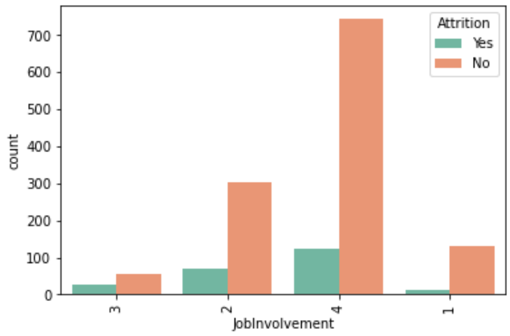


---

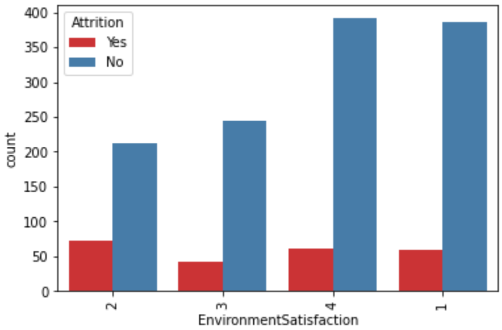
from the above chart its apparent that the entry level employees are having low Attrition rate.

```
33]:  l = list(df['JobInvolvement'].unique())
      chart = sns.countplot(df["JobInvolvement"],palette="Set2",hue=df.Attrition)
      chart.set_xticklabels(labels=l, rotation=90)
```

```
33]: [Text(0, 0, '3'), Text(1, 0, '2'), Text(2, 0, '4'), Text(3, 0, '1')]
```

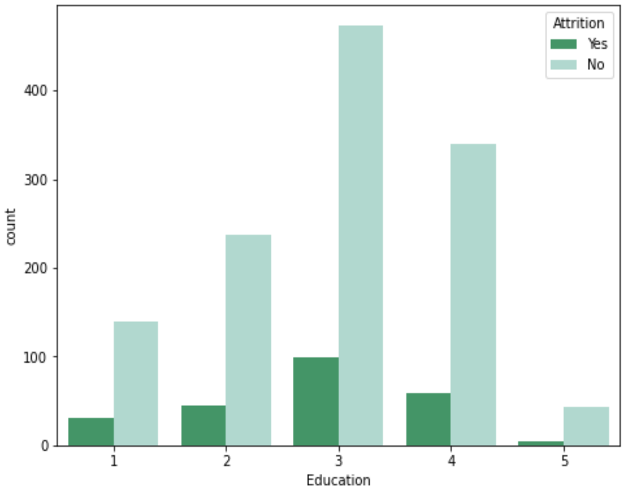


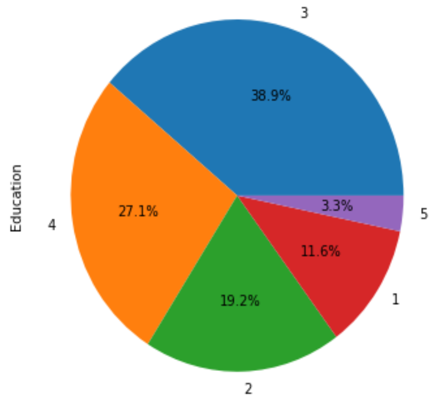
from the above chart its apparent that the employee who all are highly involved in the job having low Attrition rate



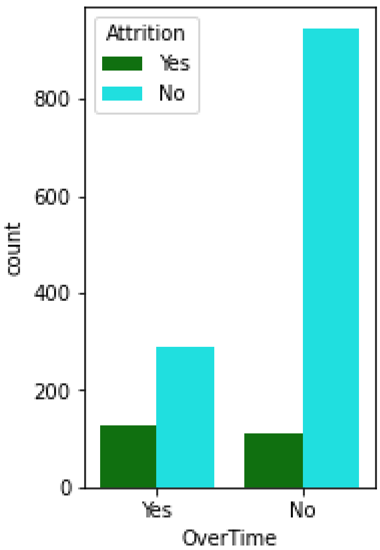
from the above chart its apparent that both the employee who having high satisfaction and low satisfaction are having low Attrition rate.so we cannot predict the target coloumn with this value

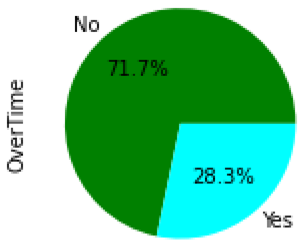
<AxesSubplot:ylabel='Education'>





The employees of 38.9% belongs to education category 3.compare to other category 3 has less percentange of people moving out of the company



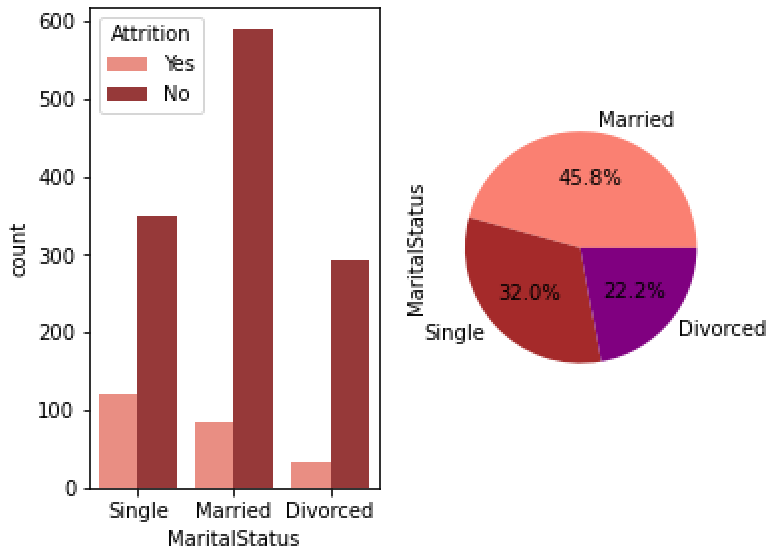


Employees who all are not working overtime has low attrition rate

28.3% employees are willing to work in overtime

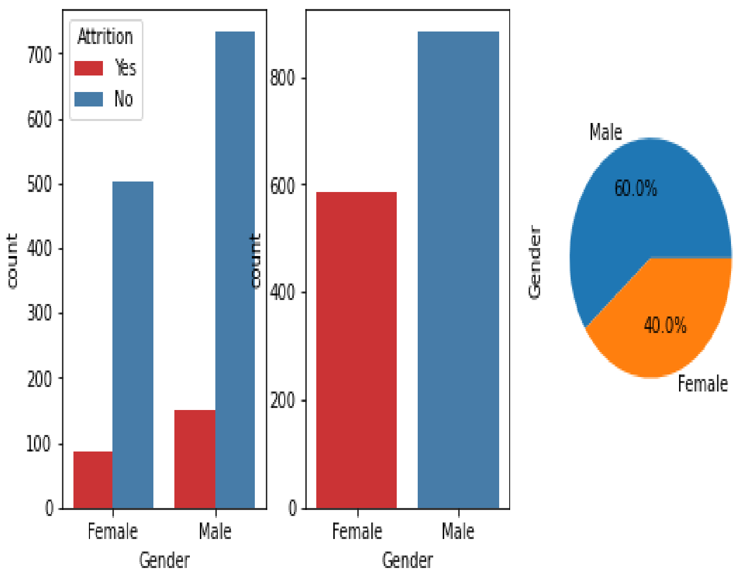
```
Married     673
Single      470
Divorced    327
Name: MaritalStatus, dtype: int64
```

ut[12]: <AxesSubplot:ylabel='MaritalStatus'>

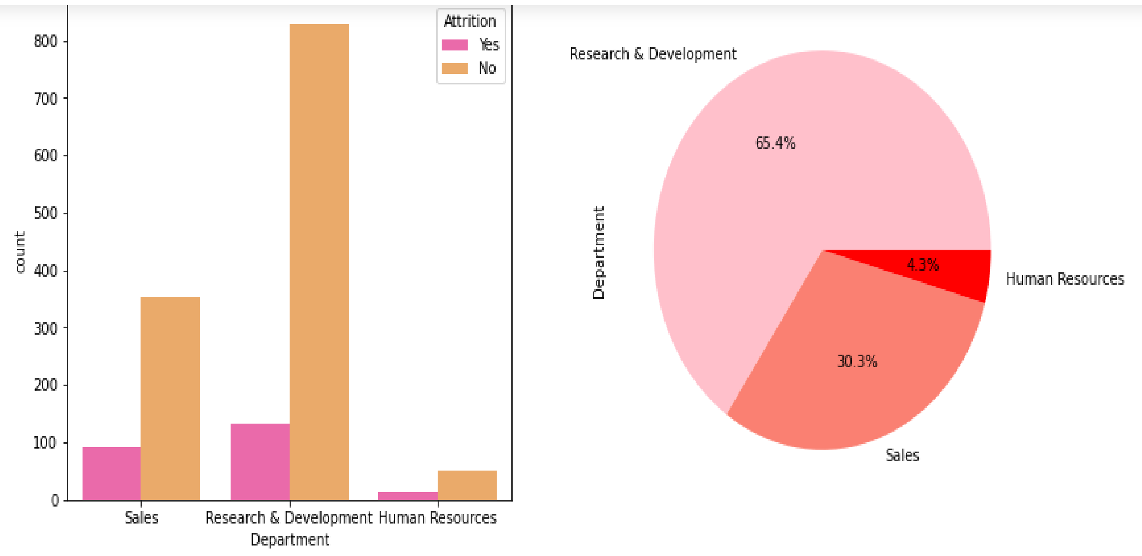


from this above chart its apparent that employees who all are married are having less attrition rate

]: <AxesSubplot:ylabel='Gender'>



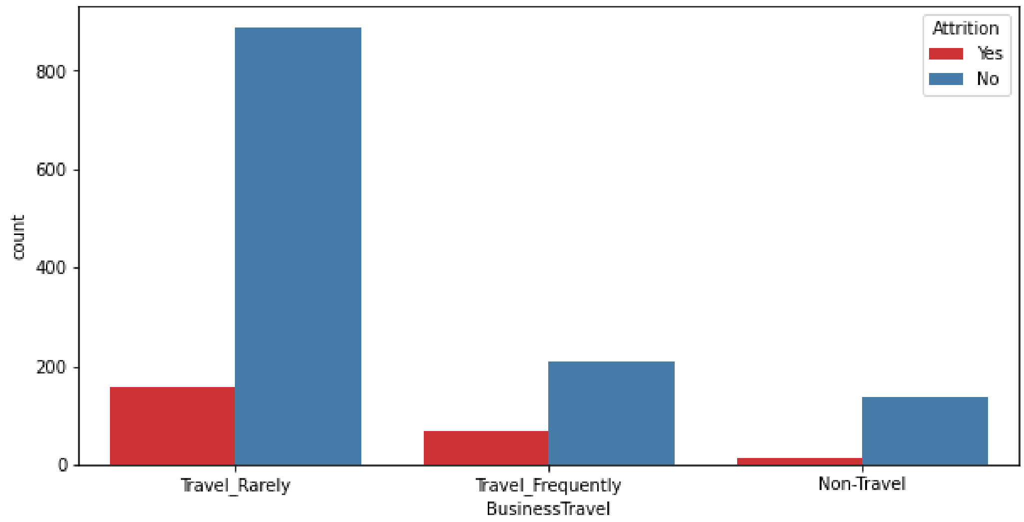
comparing the percentage of attrition out of 588 female only 88 people are quitting it means ((((588-88)/588)*100)) 15% female are leaving but in male out of 882 people more than 150 peple are quitting it means(((882-150)/882)*100)18% male are leaving



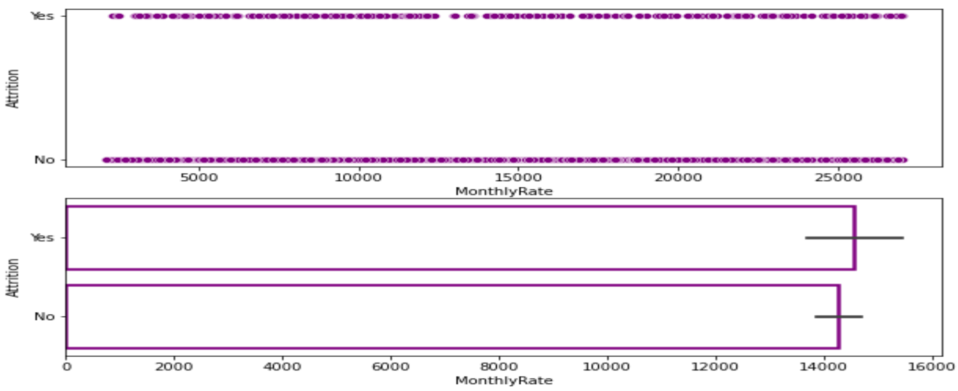
from the above plot its apparent that comparitively the employees who belongs to Research and Developement will like to continue their job.The majority percentage(65.4%) of employee belongs to R&D department

[38]:
```
plt.figure(figsize=(10,5))
sns.countplot(df.BusinessTravel,palette="Set1",hue=df.Attrition)
```

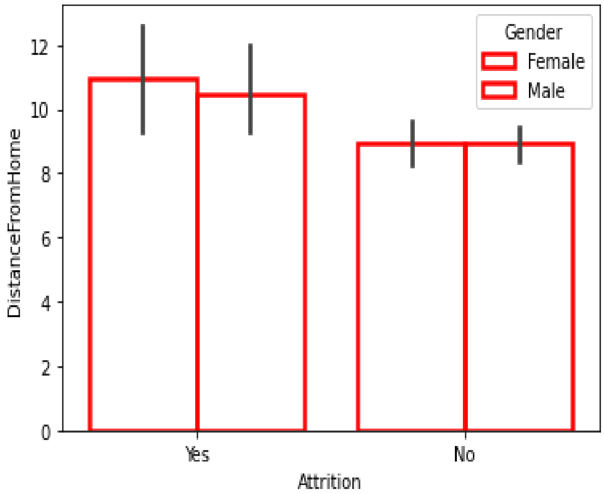
[38]: `<AxesSubplot:xlabel='BusinessTravel', ylabel='count'>`



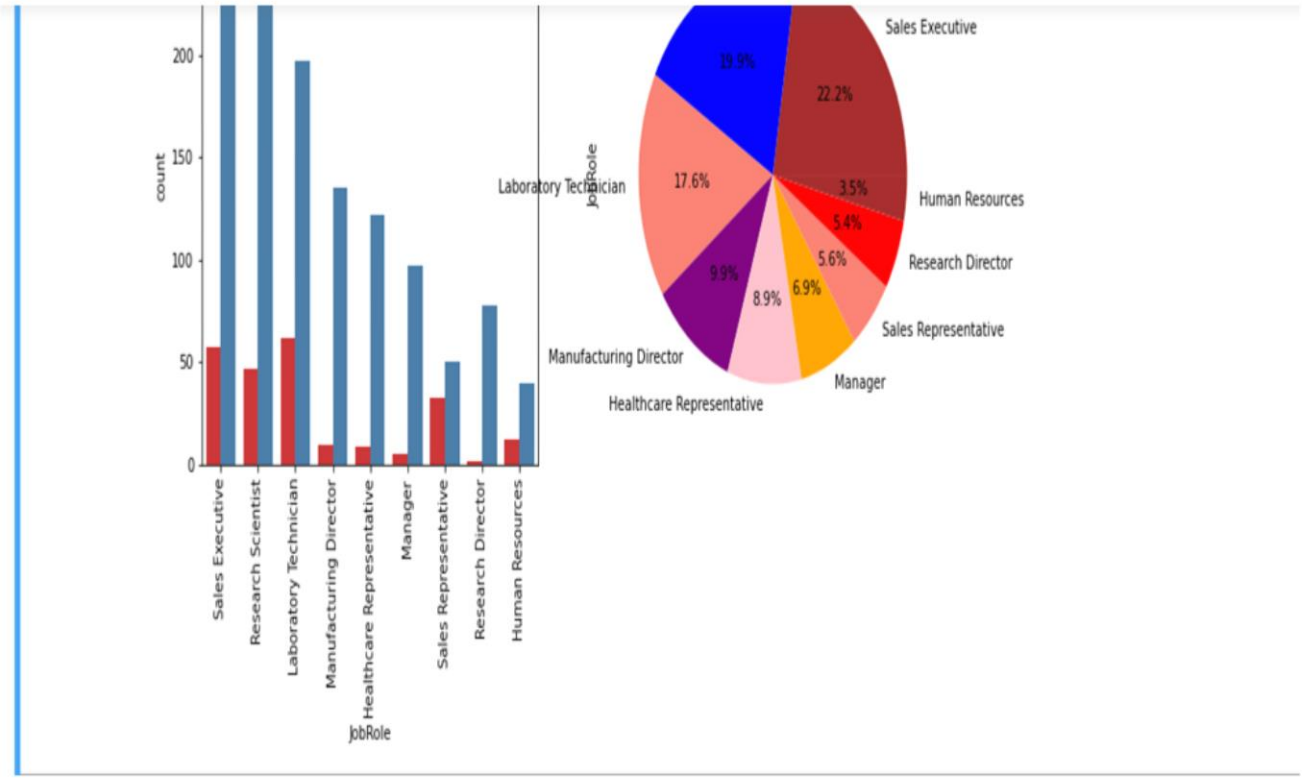
from the above plot its apparent that comparitively the employees who travel rarely will not resign their job



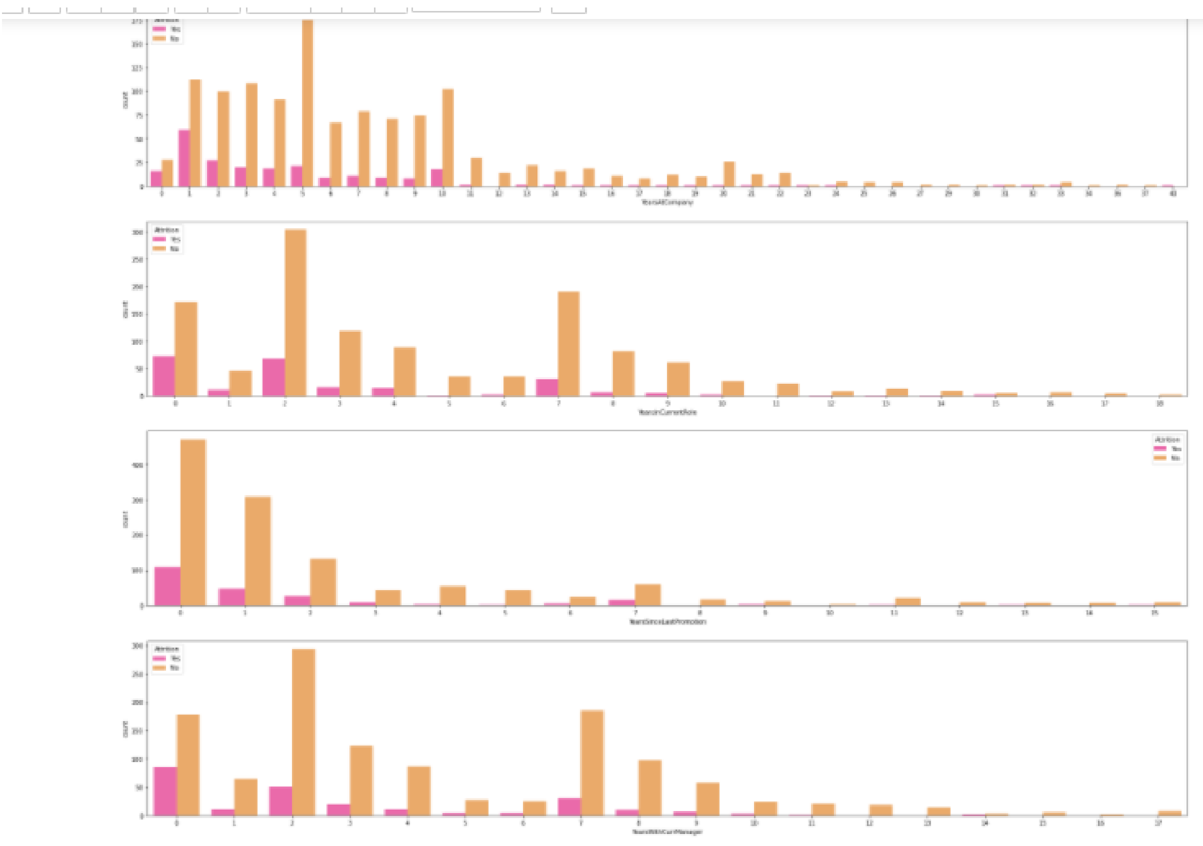
Employees having low monthlyrate are mostly resigning their job



Distance from home is not an important feature to create impact on Attrition feature.Distance from home is not impact on gende

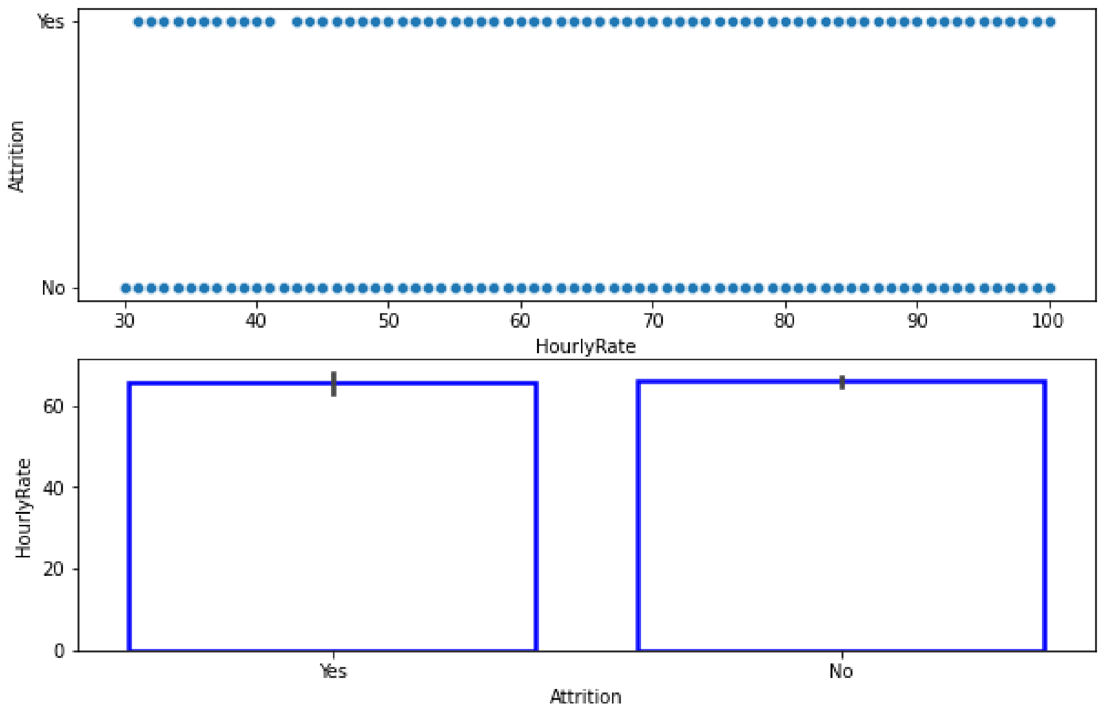


Attrition rate is lower in salesexecutive and research scientist roles



if the employee is in their 5th year with company with same manager are not quitting the job

```
]: <AxesSubplot:xlabel='Attrition', ylabel='HourlyRate'>
```



HourlyRate is not an important feature to create impact on Attrition feature

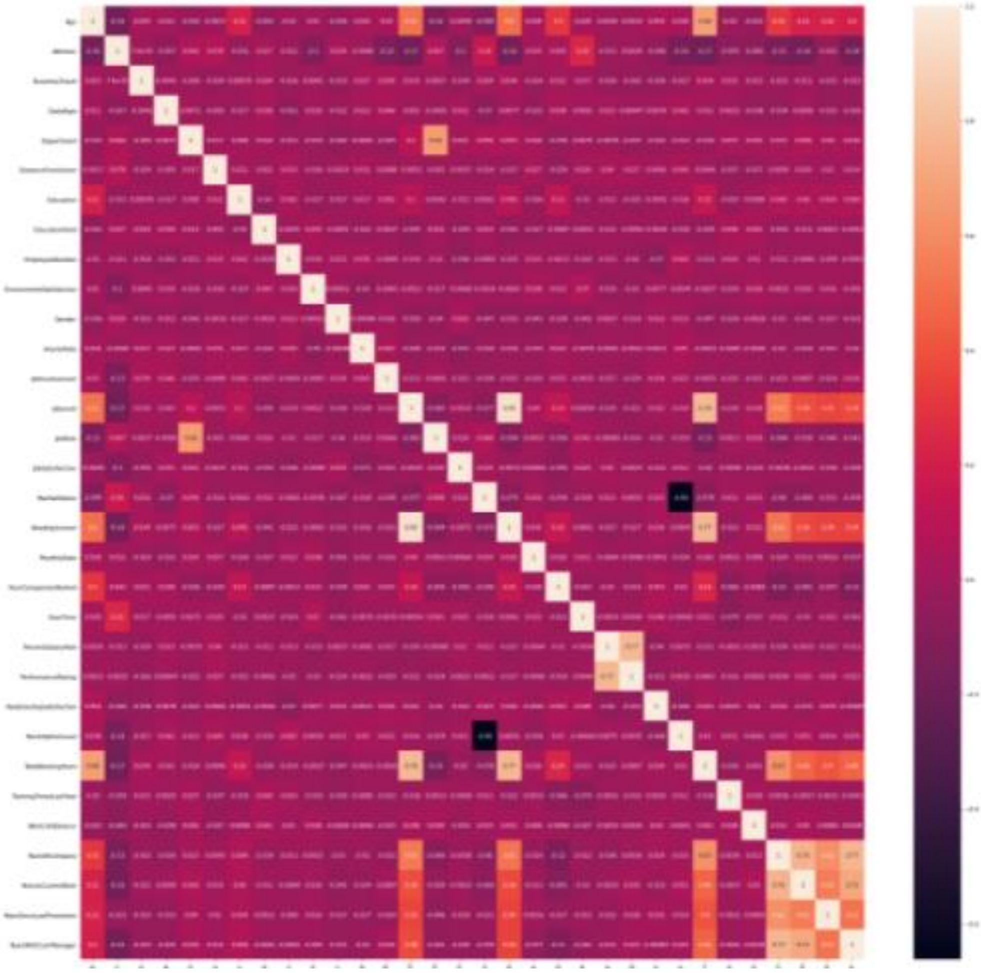
EDA CONCLUSION:

Employees who belongs to below category having less attritionon rate

- travel rarely
- who belongs to R&D department
- who belongs to life science and medical field
- female
- working as sales executive and research scientists
- unmarried
- not working over time
- moderate work life balance
- high job involvement
- working in single company
- performance rating:3
- employees who having 0 stocks
- low monthly income.

we cannot predict using relationship satisfaction,job satisfaction,Environment satisfaction features

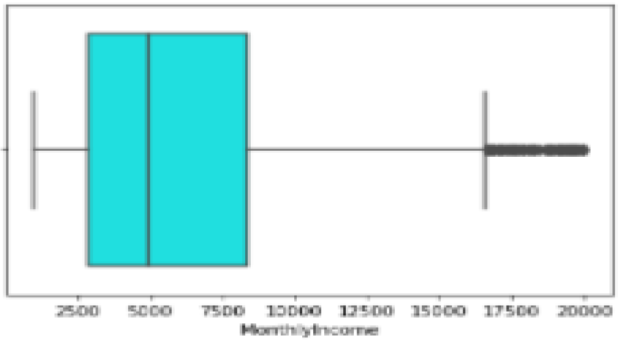
Correlation



Over time feature is highly correlated with attrition

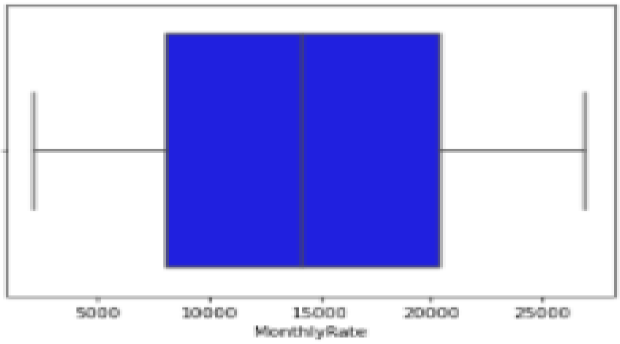
Handling outliers in numerical column:

Out[7]: <AxesSubplot:xlabel='MonthlyIncome'>



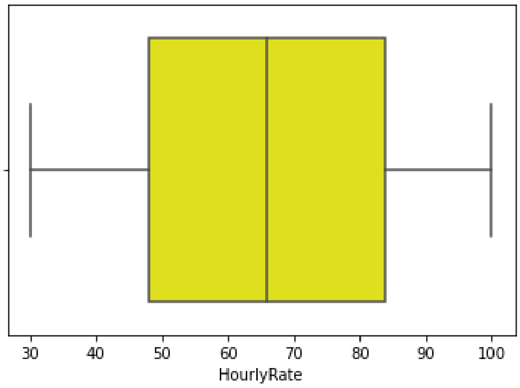
In [8]:
```
sns.boxplot(df['MonthlyRate'],color="blue")
```

Out[8]: <AxesSubplot:xlabel='MonthlyRate'>



In [9]:
```
sns.boxplot(df['DailyRate'],color="green")
```

Out[10]: <AxesSubplot:xlabel='HourlyRate'>



In [11]:
```
z1 = np.abs(stats.zscore(df_new['MonthlyIncome']))
print(z1)
```

```
[0.10834951 0.29171859 0.93765369 ... 0.07669019 0.23647414 0.44597809]
```

In [14]:
```
df_new['MonthlyIncome'] = df_new.MonthlyIncome[(z1<3)]
df_new.shape
```

Out[14]: (1470, 32)

_outliers are removed from numerical data monthly income

Data pre-processing:

The features standard hours, over18 and employee count has only single value so it won't create any impact on the target feature Attrition.

Employee Number feature is just an identifier and it's not required for modelling either. So I'm dropping these features

The features standardhours,over18 and employeecount has only one value so it wont create any impact on the target feature Attrition.

In [4]:
```
cols=['StandardHours','Over18','EmployeeCount']
df_new=df.drop(cols,axis=1)
```

All the three columns having single value so I'm dropping it from the given dataset

In [4]:
```
df_new.shape
```

Out[4]: (1470, 32)

Encoding all categorical column into numerical column using label encoding technique

In [22]:
```
data_clean=df_new
col_encod=['Attrition','BusinessTravel','Department','EducationField','Gender','JobRole','MaritalStatus','OverTime']
```

In [23]:
```
from sklearn import preprocessing
for col in col_encod:
    label = preprocessing.LabelEncoder()
    data_clean[col]= label.fit_transform(df_new[col])
```

In [24]:
```
data_clean.head(5)
```

Out[24]:

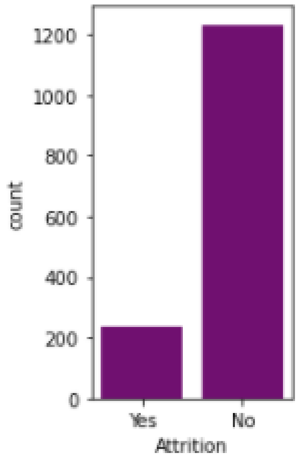
	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeNumber	EnvironmentSatisfaction	...	Perform
0	41	1	2	1102	2	1	2	1	1	2	...	
1	49	0	1	279	1	8	1	1	2	3	...	
2	37	1	2	1373	1	2	2	4	4	4	...	
3	33	0	1	1392	1	3	4	1	5	4	...	
4	27	0	2	591	1	2	1	3	7	1	...	

5 rows × 32 columns

HANDLING CLASS IMBALANCE

Classification problem where the distribution of examples across the known classes is biased or skewed. To avoid this we are using SMOTE technique

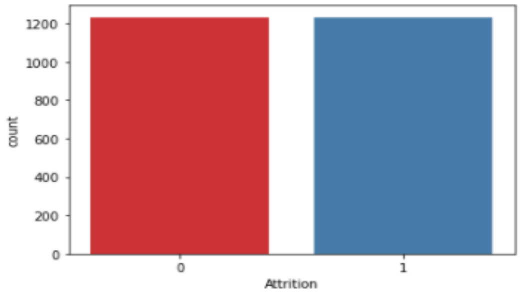
In [112]:
```
plt.figure(figsize=(2,4))
sns.countplot(df.Attrition,color="purple")
```
Out[112]: <AxesSubplot:xlabel='Attrition', ylabel='count'>



the target coloumn attrition has two values 0 and 1.It has class imbalance

SMOTE synthetic over-sampling works to cause the classifier to build larger decision regions that contain nearby minority class points. This will in turn avoid data loss

In [333]:
```
x1=data.drop('Attrition',axis=1)
y1=data['Attrition']
```
In [334]:
```
x1,y1=over.fit_resample(x1,y1)
```
In [335]:
```
sns.countplot(y1,palette="Set1")
```
Out[335]: <AxesSubplot:xlabel='Attrition', ylabel='count'>



Scaling using min max scaler

In [336]:
```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
scaled = scaler.fit_transform(x1)
```

Modelling

It is a binary classification problem so I have modelled using logistic regression and other classification models

___MODELING

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn .ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import cross_val_score
from matplotlib import pyplot
from sklearn.svm import SVC
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
x=scaled
y=y1
```

```
:rain_test_split(x,y,test_size=0.25,random_state=1)

),RandomForestClassifier(),BaggingClassifier(),KNeighborsClassifier(),GradientBoostingClassifier(),DecisionTreeClassifier()]


```

```
------------------------------------------------------------------------
KNeighborsClassifier()

Accuracy score: 0.8

"Confusion Matrix:
" [[196 101]
 [ 21 299]]
classification_report
              precision    recall  f1-score   support

           0       0.90      0.66      0.76       297
           1       0.75      0.93      0.83       320

    accuracy                           0.80       617
   macro avg       0.83      0.80      0.80       617
weighted avg       0.82      0.80      0.80       617

Average accuracy_score 0.8022690437601296
```

```
------------------------------------------------------------------------
BaggingClassifier()

Accuracy score: 0.84

"Confusion Matrix:
" [[262  35]
 [ 63 257]]
classification_report
              precision    recall  f1-score   support

           0       0.81      0.88      0.84       297
           1       0.88      0.80      0.84       320

    accuracy                           0.84       617
   macro avg       0.84      0.84      0.84       617
weighted avg       0.84      0.84      0.84       617

Average accuracy_score 0.8411669367909238
------------------------------------------------------------------------
```

```
RandomForestClassifier()

Accuracy score: 0.9

"Confusion Matrix:
" [[276  21]
 [ 43 277]]
classification_report
              precision    recall  f1-score   support

           0       0.87      0.93      0.90       297
           1       0.93      0.87      0.90       320

    accuracy                           0.90       617
   macro avg       0.90      0.90      0.90       617
weighted avg       0.90      0.90      0.90       617

Average accuracy_score 0.8962722852512156
```

```
LogisticRegression()

Accuracy score: 0.8

"Confusion Matrix:
" [[245  52]
 [ 69 251]]
classification_report
              precision    recall  f1-score   support

           0       0.78      0.82      0.80       297
           1       0.83      0.78      0.81       320

    accuracy                           0.80       617
   macro avg       0.80      0.80      0.80       617
weighted avg       0.81      0.80      0.80       617

Average accuracy_score 0.8038897893030794
-----------------------------------------------------------------
```

```
GradientBoostingClassifier()

Accuracy score: 0.88

"Confusion Matrix:
" [[268  29]
 [ 46 274]]
classification_report
              precision    recall  f1-score   support

           0       0.85      0.90      0.88       297
           1       0.90      0.86      0.88       320

    accuracy                           0.88       617
   macro avg       0.88      0.88      0.88       617
weighted avg       0.88      0.88      0.88       617

Average accuracy_score 0.8784440842787682
-----------------------------------------------------------------
```

```
DecisionTreeClassifier()

Accuracy score: 0.79

"Confusion Matrix:
" [[231  66]
 [ 64 256]]
classification_report
              precision    recall  f1-score   support

           0       0.78      0.78      0.78       297
           1       0.80      0.80      0.80       320

    accuracy                           0.79       617
   macro avg       0.79      0.79      0.79       617
weighted avg       0.79      0.79      0.79       617

Average accuracy_score 0.7893030794165316
```

Random forest classifier has highest accuracy is **0.896272**

Cross Validation:

In order to avoid over fitting, Cross-validation is used to estimate the skill of a machine learning model on unseen data.

```
In [51]:  scorel=[]

In [52]:  lr=LogisticRegression()
          scores=cross_val_score(lr,x,y,cv=5)
          scorel.append(scores)
          scores

Out[52]: array([0.64574899, 0.85395538, 0.83975659, 0.86206897, 0.84178499])

In [53]:  rf=RandomForestClassifier()
          scores=cross_val_score(rf,x,y,cv=5)
          scorel.append(scores)
          scores

Out[53]: array([0.73481781, 0.95537525, 0.93103448, 0.95537525, 0.94523327])

In [54]:  bg=BaggingClassifier()
          scores=cross_val_score(bg,x,y,cv=5)
          scorel.append(scores)
          scores

Out[54]: array([0.70445344, 0.9127789 , 0.91075051, 0.90872211, 0.89655172])

In [55]:  kn=KNeighborsClassifier()
          scores=cross_val_score(kn,x,y,cv=5)
          scorel.append(scores)
          scores

Out[55]: array([0.76923077, 0.831643  , 0.81541582, 0.81541582, 0.82758621])

In [56]:  gb=GradientBoostingClassifier()
          scores=cross_val_score(gb,x,y,cv=5)
          scorel.append(scores)
          scores

Out[56]: array([0.65587045, 0.93711968, 0.90872211, 0.9148073 , 0.9148073 ])
```

Difference of predicted model and crossvalidation score:

- LogisticRegression() difference is0.0378952
- RandomForestClassifier() difference is 0.05058173
- BaggingClassifier() difference is 0.06024702
- KNeighborsClassifier() difference is 0.02531716
- GradientBoostingClassifier() difference is 0.03636322
- DecisionTreeClassifier() difference is 0.05733428

from the observation KNeighborsClassifier model has least difference so I'm selecting KNeighborsClassifier as best model

Hyper Tuning:

__Hyper Tuning

In [226]:

```
from sklearn.model_selection import GridSearchCV,KFold
params = {
    'n_neighbors' : [5,7,9,11,13,15],
               'weights' : ['uniform','distance'],
               'metric' : ['minkowski','euclidean','manhattan'],
                'p':[1,2],'leaf_size':list(range(1,20))

}

gs2 = GridSearchCV(KNeighborsClassifier(), params, verbose = 1, cv=3, n_jobs = -1)
gs2.fit(xtrain, ytrain)
print('Best param:', gs2.best_params_)
```

```
Fitting 3 folds for each of 1368 candidates, totalling 4104 fits
Best param: {'leaf_size': 1, 'metric': 'minkowski', 'n_neighbors': 5, 'p': 1, 'weights': 'distance'}
```

Best parameters: {'leaf_size': 1, 'metric': 'minkowski', 'n_neighbors': 5, 'p': 1, 'weights': 'distance'}

Modelling using best parameter and best model:

```
x_train, x_test, y_train, y_test = train_test_split(x, y,test_size=.25)
model =KNeighborsClassifier(metric='minkowski', n_neighbors=5,weights='distance',p=1,leaf_size=1)
model.fit(x_train,y_train)
model.score(x_test,y_test)
```

```
0.9027552674230146
```

```
y_pred_1 = model.predict(x_test)
```

```
result = confusion_matrix(y_test, y_pred_1)
print("Confusion Matrix:")
print(result)
result1 = classification_report(y_test, y_pred_1)
print("Classification Report:",)
print (result1)
result2 = accuracy_score(y_test,y_pred_1)
print("Accuracy:",result2)
```

```
Confusion Matrix:
[[242  53]
 [  7 315]]
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.82      0.89       295
           1       0.86      0.98      0.91       322

    accuracy                           0.90       617
   macro avg       0.91      0.90      0.90       617
weighted avg       0.91      0.90      0.90       617

Accuracy: 0.9027552674230146
```

Final model after hyper tuning with accuracy **0.9027552674230146**

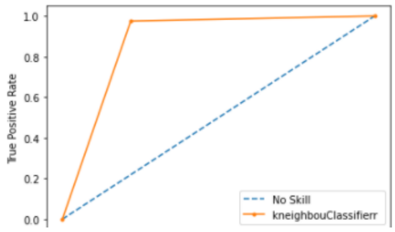
Best model:KNeighbourClassifier Best param: {'leaf_size': 1, 'metric': 'minkowski', 'n_neighbors': 5, 'p': 1, 'weights': 'distance'}

ROC AUC CURVE:

In [68]:

```
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.25,random_state=1)
m=KNeighborsClassifier(metric='minkowski', n_neighbors=5,weights='distance',p=1,leaf_size=1)
m.fit(xtrain,ytrain)
p=m.predict(xtest)
ns_probs = [0 for _ in range(len(ytest))]
m_probs = p
ns_auc = roc_auc_score(ytest, ns_probs)
m_auc = roc_auc_score(ytest, m_probs)
print('No Skill: ROC AUC=%.3f' % (ns_auc))
print('model: ROC AUC=%.3f' % (m_auc))
ns_fpr, ns_tpr,_= roc_curve(ytest, ns_probs)
m_fpr, m_tpr,_=roc_curve(ytest, m_probs)
pyplot.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
pyplot.plot(m_fpr, m_tpr, marker='.', label='kneighbouClassifierr ')
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
pyplot.legend()
pyplot.show()
```

```
No Skill: ROC AUC=0.500
model: ROC AUC=0.878
```



Conclusion:

I have developed a model to predict attrition of an employee with 90.2% accuracy

Saving the model

```
In [398]: from joblib import dump
          dump(model, 'model_hr.joblib')

Out[398]: ['model_hr.joblib']

In [399]: from joblib import load
          loaded = load('model_hr.joblib')
```

Baseball Case Study

Problem Statement:

This dataset utilizes data from 2014 Major League Baseball seasons in order to develop an algoirthm that predicts the number of wins for a given team in the 2015 season based on several different indicators of success. This model is used to select best team based on best input feature. .There are 16 different features that will be used as the inputs to the machine learning and the output will be a value that represents the number of wins.

Input features:

- R: Runs-times reached home plate legally and safely
- AB: At Bats-plate appearances, not including bases on balls, being hit by pitch, sacrifices, interference, or obstruction
- H: Hits- reaching base because of a batted, fair ball without error by the defence
- 2B: Doubles-hits on which the batter reaches second base safely without the contribution of a fielding error
- 3B: Triples- hits on which the batter reaches third base safely without the contribution of a fielding error
- HR: Homeruns
- BB: Walks-times pitching four balls, allowing the batter to take first base
- SO: Strikeouts
- SB: Stolen Bases- number of bases advanced by the runner while the ball is in the possession of the defence
- RA: Runs Allowed
- ER: Earned Runs- number of runs that did not occur as a result of errors or passed balls
- ERA: Earned Run Average (ERA)- total number of earned runs (see "ER" above), multiplied by 9, divided by innings pitched
- SO: Shutouts- number of complete games pitched with no runs allowed

- SV: Saves- number of games where the pitcher enters a game led by the pitcher's team, finishes the game without surrendering the lead, is not the winning pitcher, and either (a) the lead was three runs or fewer when the pitcher entered the game; (b) the potential tying run was on base, at bat, or on deck; or (c) the pitcher pitched three or more innings
- CG: Complete Games-number of games where player was the only pitcher for their team
- E: Errors- the judgment of the official scorer, of a fielder misplaying a ball in a manner that allows a batter or base runner to advance one or more bases

Output: Number of predicted wins (W)

Importing the Libraries

```
import pandas as pd
import numpy as np
import seaborn as sns
from scipy import stats
import warnings
warnings.filterwarnings('ignore')
from matplotlib import pyplot as plt
from scipy.stats import zscore
```

Loading Dataset:

In [3]:
```
df=pd.read_csv("E:\\baseball.csv")
df.head(5)
```

Out[3]:

	W	R	AB	H	2B	3B	HR	BB	SO	SB	RA	ER	ERA	CG	SHO	SV	E
0	95	724	5575	1497	300	42	139	383	973	104	641	601	3.73	2	8	56	88
1	83	696	5467	1349	277	44	156	439	1264	70	700	653	4.07	2	12	45	86
2	81	669	5439	1395	303	29	141	533	1157	86	640	584	3.67	11	10	38	79
3	76	622	5533	1381	260	27	136	404	1231	68	701	643	3.98	7	9	37	101
4	74	689	5605	1515	289	49	151	455	1259	83	803	746	4.64	7	12	35	86

loaded the abalone dataset.

n [201]:
```
df.shape
```
ut[201]: (30, 17)

It has 17 coloumns and 30 rows

n [202]:
```

df.isnull().sum()
```
ut[202]:
```
W      0
R      0
AB     0
H      0
2B     0
3B     0
HR     0
BB     0
SO     0
SB     0
RA     0
ER     0
ERA    0
CG     0
SHO    0
SV     0
E      0
dtype: int64
```

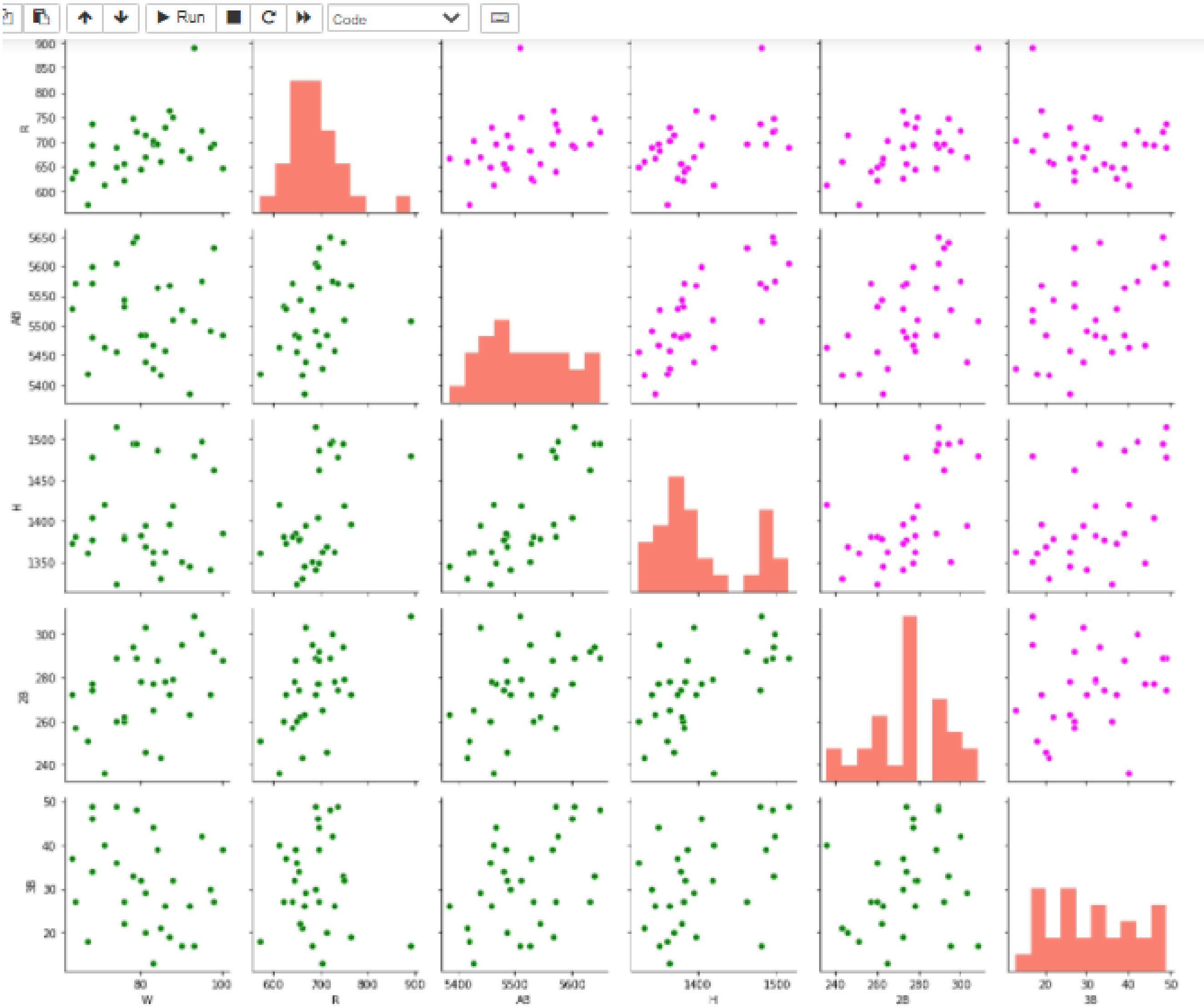
```
From the above observation this dataset has no null values
```

This dataset has no null values and it has 30 rows and 17 columns
CHECKING FOR EMPTY SPACE IN DATASET

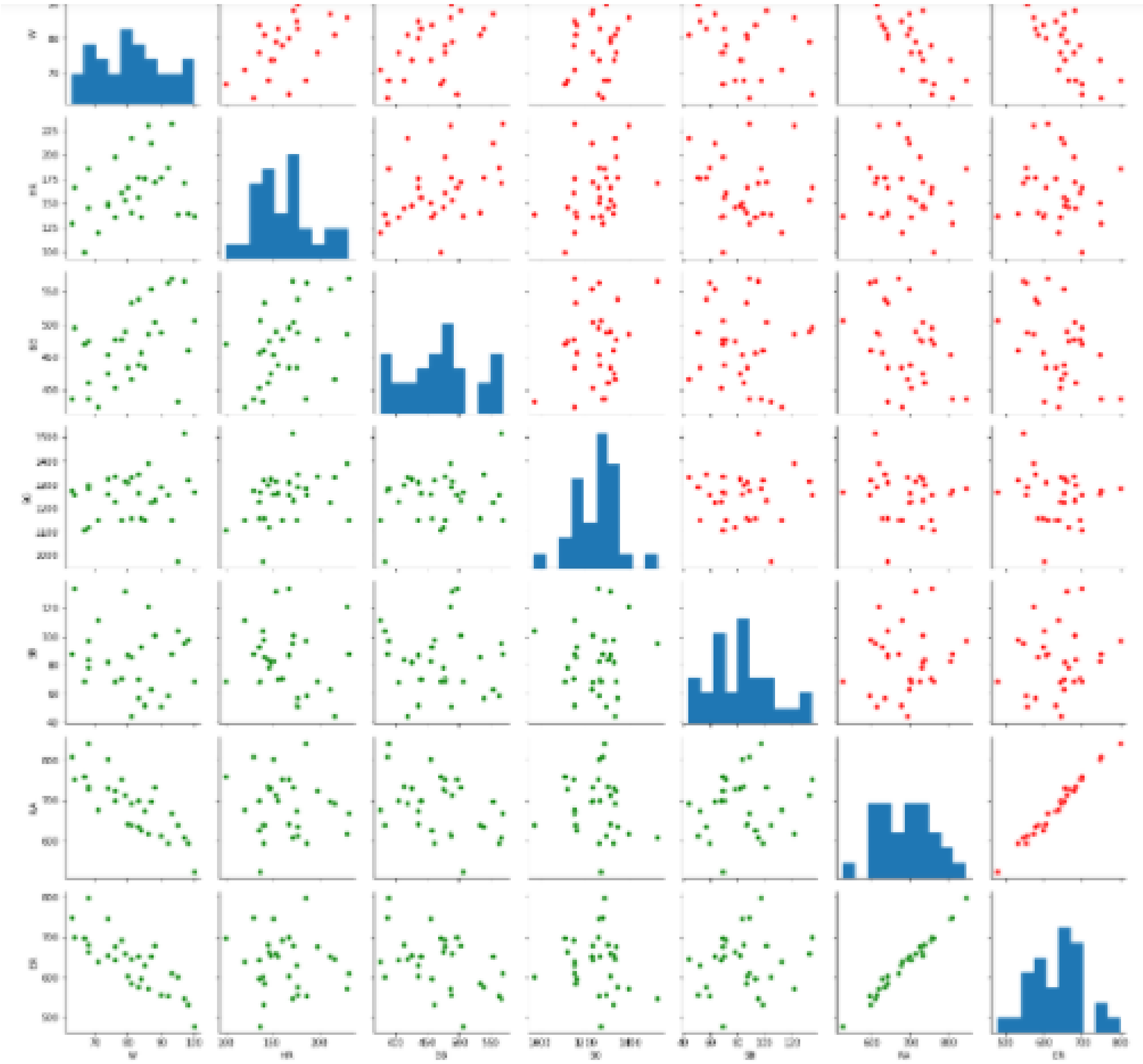
3]:
```
col=['W', 'R', 'AB', 'H', '2B', '3B', 'HR', 'BB', 'SO', 'SB', 'RA', 'ER','ERA', 'CG', 'SHO', 'SV', 'E']
for i in col:
    print(df.loc[df[i]==""])
```

```
Empty DataFrame
Columns: [W, R, AB, H, 2B, 3B, HR, BB, SO, SB, RA, ER, ERA, CG, SHO, SV, E]
Index: []
Empty DataFrame
Columns: [W, R, AB, H, 2B, 3B, HR, BB, SO, SB, RA, ER, ERA, CG, SHO, SV, E]
Index: []
Empty DataFrame
Columns: [W, R, AB, H, 2B, 3B, HR, BB, SO, SB, RA, ER, ERA, CG, SHO, SV, E]
Index: []
Empty DataFrame
Columns: [W, R, AB, H, 2B, 3B, HR, BB, SO, SB, RA, ER, ERA, CG, SHO, SV, E]
Index: []
Empty DataFrame
Columns: [W, R, AB, H, 2B, 3B, HR, BB, SO, SB, RA, ER, ERA, CG, SHO, SV, E]
Index: []
Empty DataFrame
Columns: [W, R, AB, H, 2B, 3B, HR, BB, SO, SB, RA, ER, ERA, CG, SHO, SV, E]
Index: []
Empty DataFrame
Columns: [W, R, AB, H, 2B, 3B, HR, BB, SO, SB, RA, ER, ERA, CG, SHO, SV, E]
Index: []
Empty DataFrame
Columns: [W, R, AB, H, 2B, 3B, HR, BB, SO, SB, RA, ER, ERA, CG, SHO, SV, E]
Index: []
Empty DataFrame
Columns: [W, R, AB, H, 2B, 3B, HR, BB, SO, SB, RA, ER, ERA, CG, SHO, SV, E]
Index: []
Empty DataFrame
Columns: [W, R, AB, H, 2B, 3B, HR, BB, SO, SB, RA, ER, ERA, CG, SHO, SV, E]
Index: []
Empty DataFrame
Columns: [W, R, AB, H, 2B, 3B, HR, BB, SO, SB, RA, ER, ERA, CG, SHO, SV, E]
Index: []
```

This dataset has no empty space as value



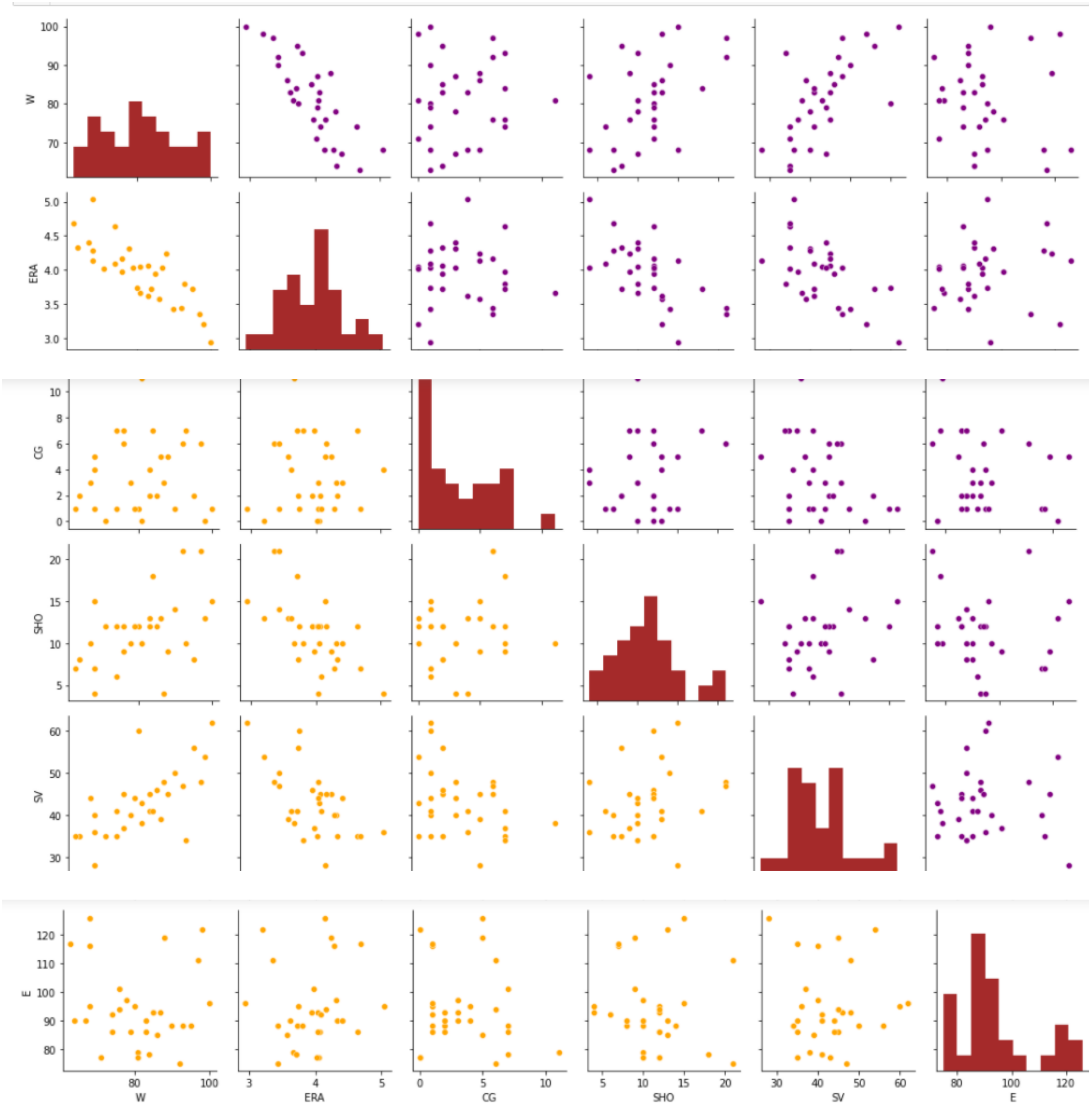
The data points of the features R and AB showing an uphill pattern as you move from left to right so it has positive relationship. The data points of the features R and w showing an uphill pattern as you move from left to right so it has positive relationship.



The data points of ER and RA showing an uphill pattern as you move from left to right so it has positive relationship.

The data points of ER and w showing an downhill pattern as you move from left to right so it has negative relationship.

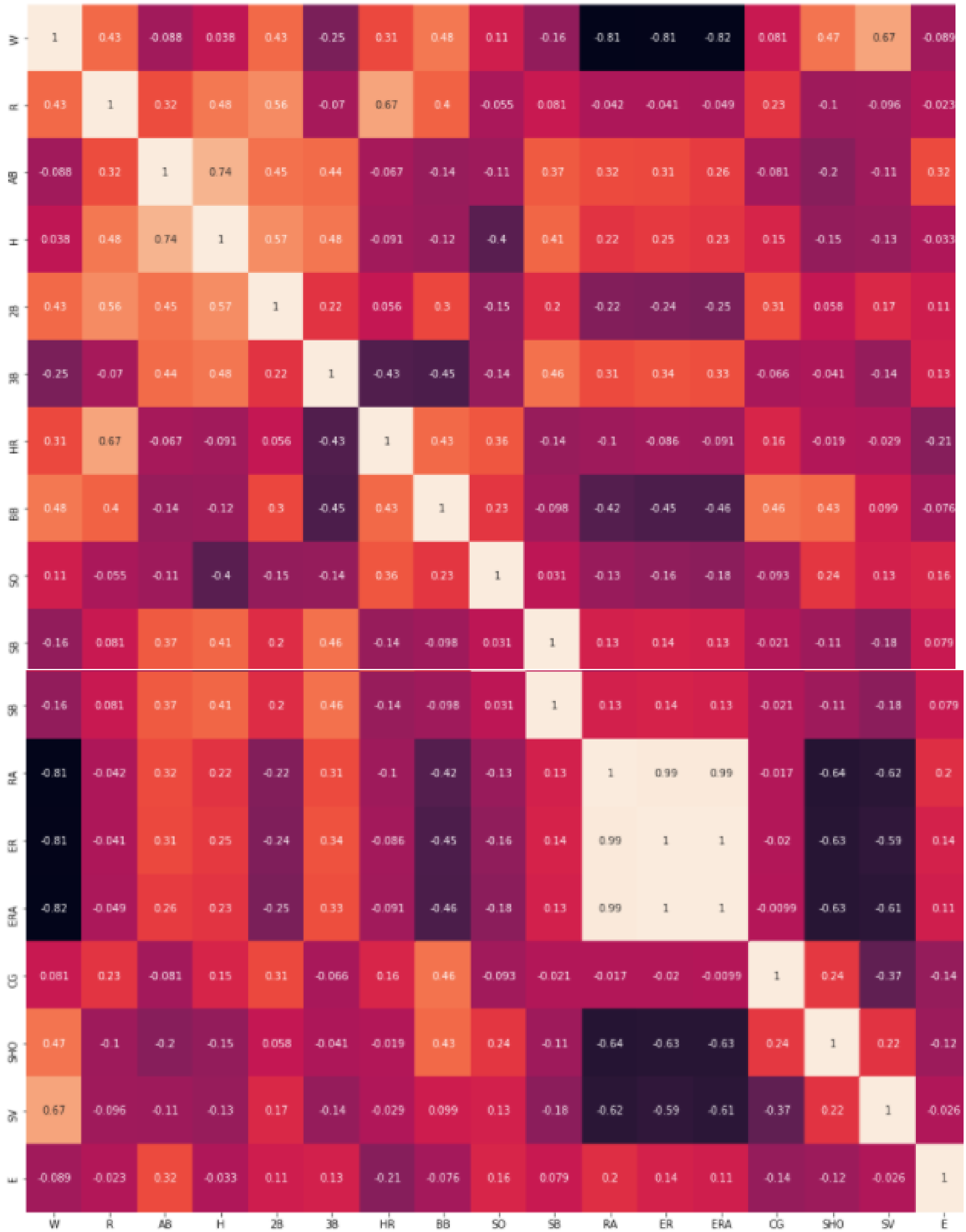
The data points of RA and w showing an downhill pattern as you move from left to right so it has negative relationship.



The data points of the features ERA and w showing an downhill pattern as you move from left to right so it has negative relationship.

The data points of the features SV and W showing an uphill pattern as you move from left to right so it has positive relationship

CORRELATION MATRIX



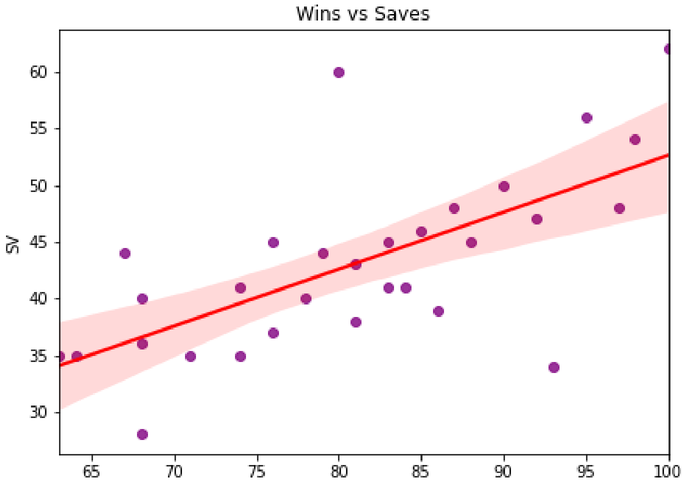
- RA,ER,ERA features are highly negatively correlated with wins.
- RA,ER,ERA features are highly negatively correlated with BB.
- RA,ER,ERA features are highly negatively correlated with sv and sho.

- Sho,bb,2b,SV,HR are positively correlated with win
- ER andERA are posively correlated with RA
- wins feature is highly and positively correlated with saves
- RA,ER,ERA features are highly negatively correlated with saves.
- h,cg having very less correlation with winsRA,ER,ERA features are highly negatively correlated with wins.
- RA,ER,ERA features are highly negatively correlated with BB.
- RA,ER,ERA features are highly negatively correlated with sv and sho.
- Sho,bb,2b,SV,HR are positively correlated with win
- ER andERA are posively correlated with RA
- wins feature is highly and positively correlated with saves
- RA,ER,ERA features are highly negatively correlated with saves.
- h,cg having very less correlation with wins

Finding whether all features which are positively correlated with Wins feature have linear relationship:

```
plt.figure(figsize=(7,5))
plt.title('Wins vs Saves')
sns.set_style="white_grid"
sns.regplot(x="W",y="SV",data=df,scatter_kws = {'color': 'purple'}, line_kws = {'color': 'r'})
plt.show()


```



In this reg plot it is apparent that SV is positively correlated with W data.most of the datapoints are almost close to best fit line it means that the feature SV and W having linear relationship

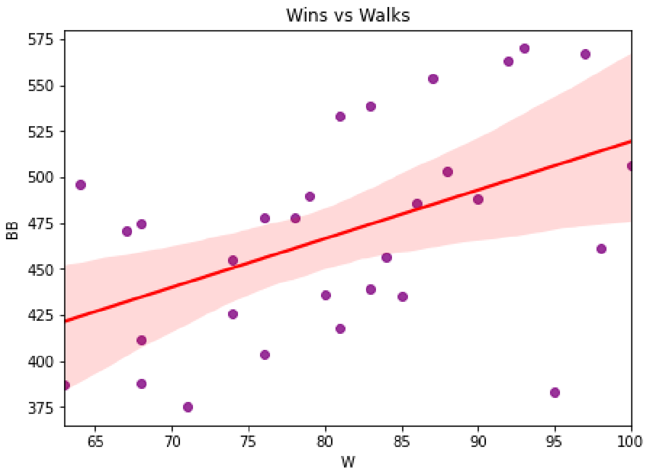
[282]:

```
plt.figure(figsize=(7,5))
plt.title('Wins vs Walks')
sns.set_style="white_grid"
sns.regplot(x="W",y="BB",data=df,scatter_kws = {'color': 'purple'}, line_kws = {'color': 'r'})



```

[282]: <AxesSubplot:title={'center':'Wins vs Walks'}, xlabel='W', ylabel='BB'>

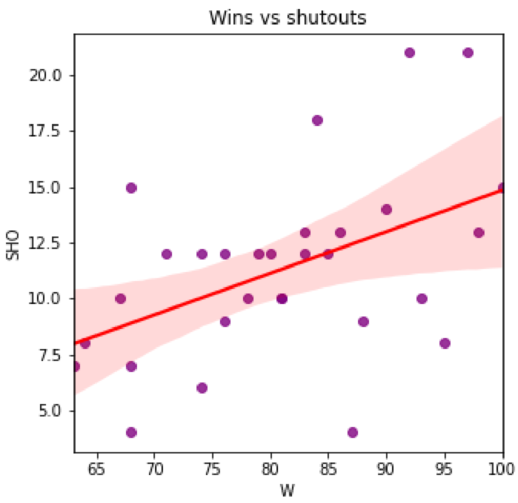


In this reg plot it is apparent that BB is positively correlated with W data. Only few datapoints are close to best fit line it means that the feature BB and W are not having linear relationship. If the value of bb is between 400 to 450 there is more chance to win

:

```
plt.figure(figsize=(5,5))
plt.title('Wins vs shutouts')
sns.set_style="white_grid"
sns.regplot(x="W",y="SHO",data=df,scatter_kws = {'color': 'purple'}, line_kws = {'color': 'r'})
```

: <AxesSubplot:title={'center':'Wins vs shutouts'}, xlabel='W', ylabel='SHO'>

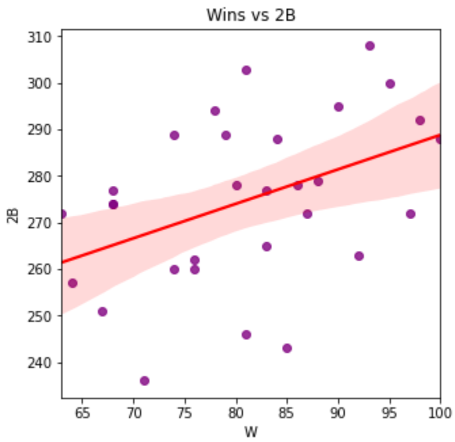


In this reg plot it is apparent that SHO is positively correlated with W data.only few datapoints are close to best fit line it means that the feature SHO and W are not having linear relationship

In [5]:

```
plt.figure(figsize=(5,5))
plt.title('Wins vs 2B')
sns.set_style="white_grid"
sns.regplot(x="W",y="2B",data=df,scatter_kws = {'color': 'purple'}, line_kws = {'color': 'r'})
```

Out[5]: <AxesSubplot:title={'center':'Wins vs 2B'}, xlabel='W', ylabel='2B'>

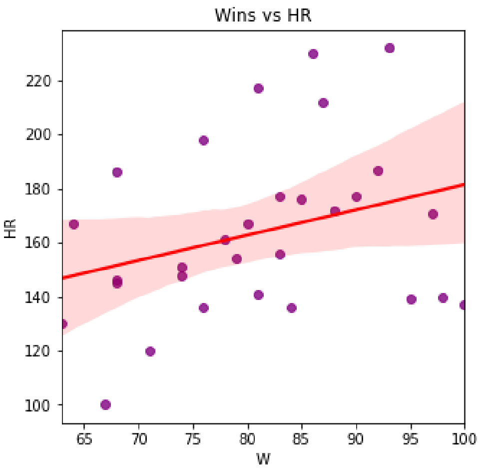


In this reg plot it is apparent that 2B is positively correlated with W data. Only few data points are close to best fit line it means that the feature 2B and W are not having linear relationship

6]:

```
plt.figure(figsize=(5,5))
plt.title('Wins vs HR')
sns.set_style="white_grid"
sns.regplot(x="W",y="HR",data=df,scatter_kws = {'color': 'purple'}, line_kws = {'color': 'r'})
```

6]: <AxesSubplot:title={'center':'Wins vs HR'}, xlabel='W', ylabel='HR'>

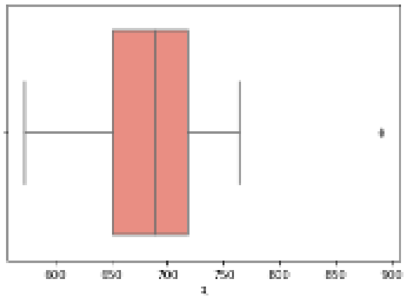


In this reg plot it is apparent that HR is positively correlated with W data. Only few datapoints are close to best fit line it means that the feature HR and W are not having linear relationship

IDENTIFYING OUTLIERS:

In [209]:
```python
sns.boxplot(df['R'],color="salmon")
```

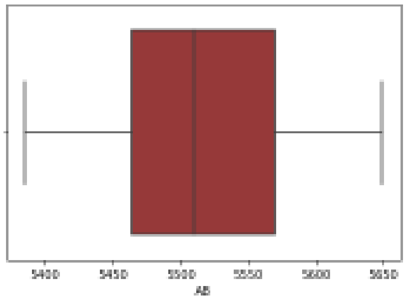
Out[209]: <AxesSubplot:xlabel='R'>



feature R has outlier

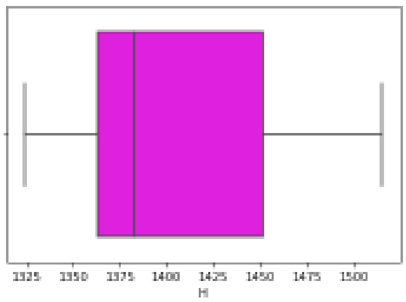
In [210]:
```python
sns.boxplot(df['AB'],color="brown")
```

Out[210]: <AxesSubplot:xlabel='AB'>



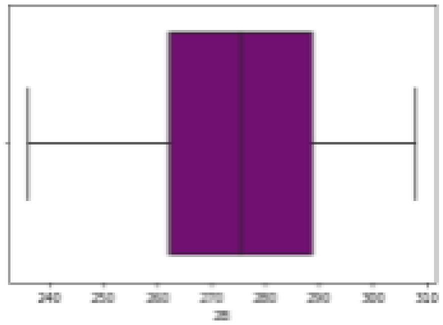
In [211]:
```python
sns.boxplot(df['H'],color=(1,0,1))
```

Out[211]: <AxesSubplot:xlabel='H'>



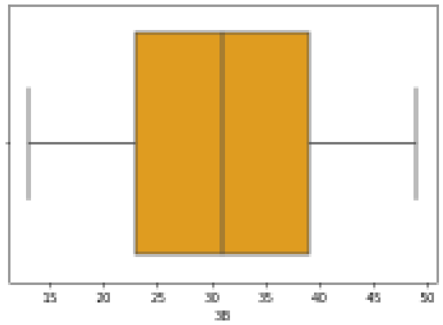
```
In [212]: sns.boxplot(df['2B'],color="Purple")
```

Out[212]: <AxesSubplot:xlabel='2B'>



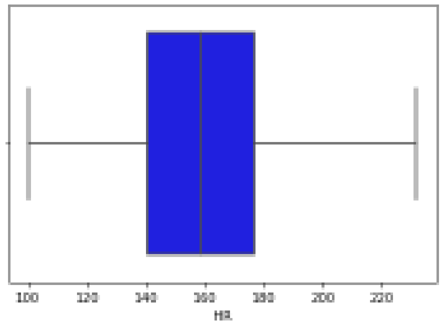

```
In [213]: sns.boxplot(df['3B'],color="orange")
```

Out[213]: <AxesSubplot:xlabel='3B'>




```
In [214]: sns.boxplot(df['HR'],color=(0,0,1))
```

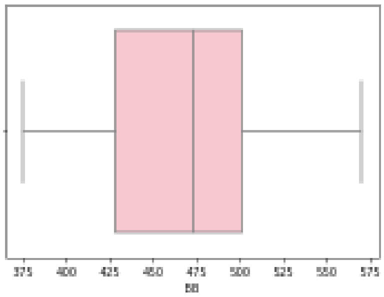
Out[214]: <AxesSubplot:xlabel='HR'>



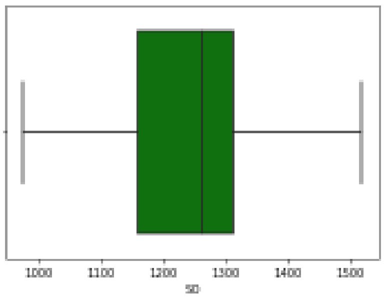

```
In [215]: sns.boxplot(df['BB'],color="pink")
```

Out[215]: <AxesSubplot:xlabel='BB'>

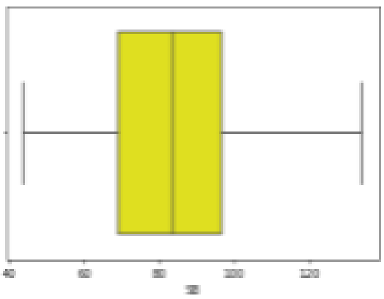
In [215]:
```
1 sns.boxplot(df['BB'],color="pink")
```
Out[215]: <AxesSubplot:xlabel='BB'>



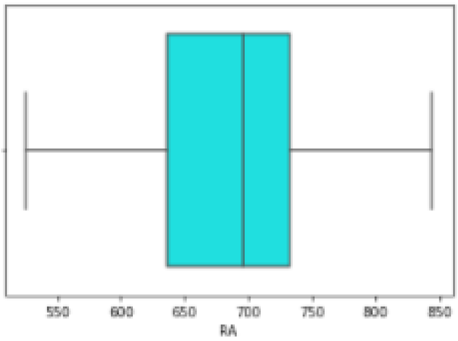
In [221]:
```
1 sns.boxplot(df['SO'],color="green")
```
Out[221]: <AxesSubplot:xlabel='SO'>



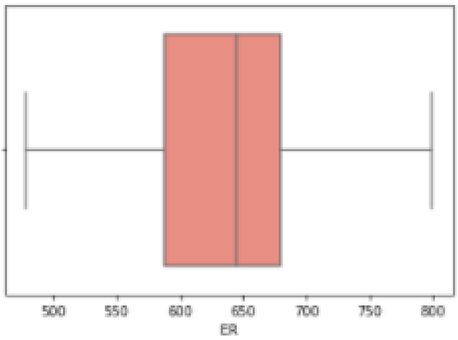
In [217]:
```
1 sns.boxplot(df['SB'],color="yellow")
```
Out[217]: <AxesSubplot:xlabel='SB'>



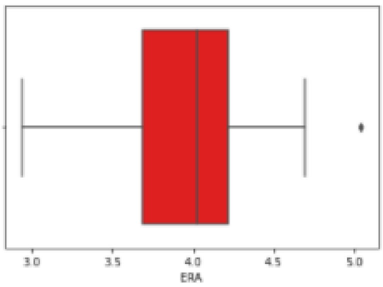
In [218]:
```
1 sns.boxplot(df['RA'],color=(0,1,1))
```
Out[218]: <AxesSubplot:xlabel='RA'>



In [219]:
```
1 sns.boxplot(df['ER'],color="salmon")
```
Out[219]: <AxesSubplot:xlabel='ER'>

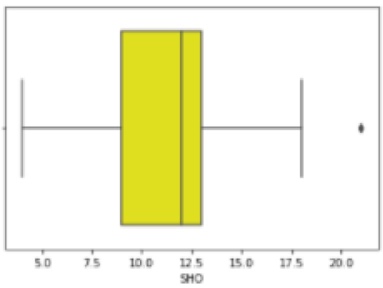


In [220]:
```
1 sns.boxplot(df['ERA'],color="red")
```
Out[220]: <AxesSubplot:xlabel='ERA'>



feature ERA has outlier

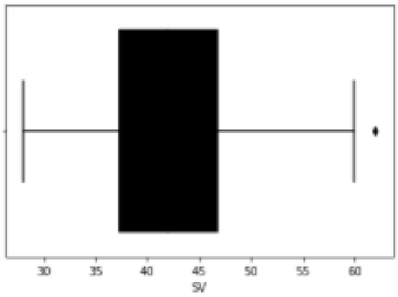
In [143]:
```
1 sns.boxplot(df['SHO'],color=(1,1,0))
```
Out[143]: <AxesSubplot:xlabel='SHO'>



feature SHO has outlier

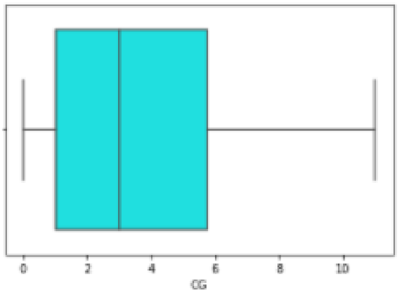
feature SHO has outlier

In [144]:
```
1 sns.boxplot(df['SV'],color=(0,0,0))
```
Out[144]: <AxesSubplot:xlabel='SV'>

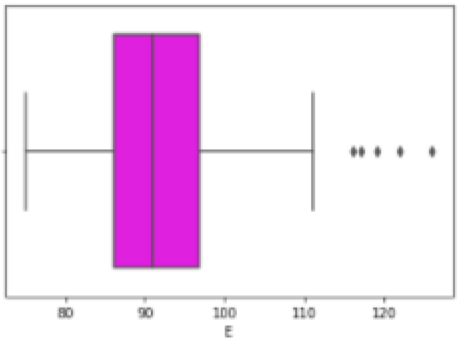


feature SV has outlier

In [145]:
```
1 sns.boxplot(df['CG'],color=(0,1,1))
```
Out[145]: <AxesSubplot:xlabel='CG'>



In [146]:
```
1 sns.boxplot(df['E'],color=(1,0,1))
```
Out[146]: <AxesSubplot:xlabel='E'>



The features R,ERA,SHO,SV,E are having outliers

REMOVING OUTLIERS

In [7]:
```
z = np.abs(zscore(df))
print(np.where(z > 3))
```

```
(array([5], dtype=int64), array([1], dtype=int64))
```

In [8]:
```
data_mod = df[(z<3).all(axis=1)]
data_mod.shape
```

Out[8]: (29, 17)

In [9]:
```
((30-29)/30)*100
```

Out[9]: 3.3333333333333335

After using z score it has only 3.3333% data loss

IQR TO REMOVE OUTLIERS:

In [230]:
```
data=df
Q1=data.quantile(0.25)
Q3=data.quantile(0.75)
IQR=Q3-Q1
df_new=data[~((data<(Q1-1.5*IQR))| (data>(Q1+1.5*IQR))).any(axis=1)]
```

In [26]:
```
df_new.shape
```

Out[26]: (6, 17)

In [27]:
```
((30-6)/30)*100
```

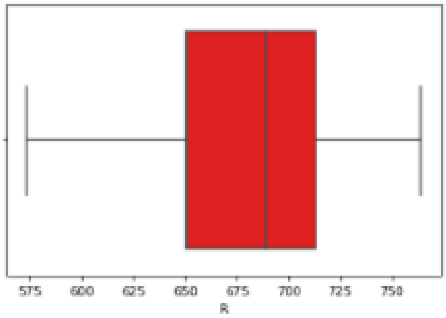
Out[27]: 80.0

80% data loss so I'm choosing zscore method

AFTER USING ZSCORE:

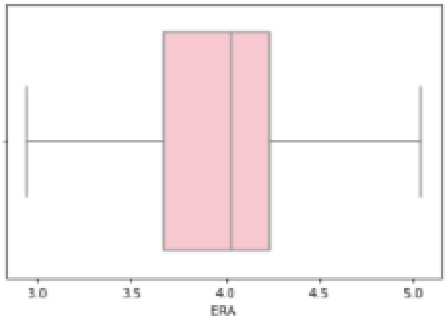
```
In [225]: sns.boxplot(data_mod['R'],color="red")
```

Out[225]: <AxesSubplot:xlabel='R'>



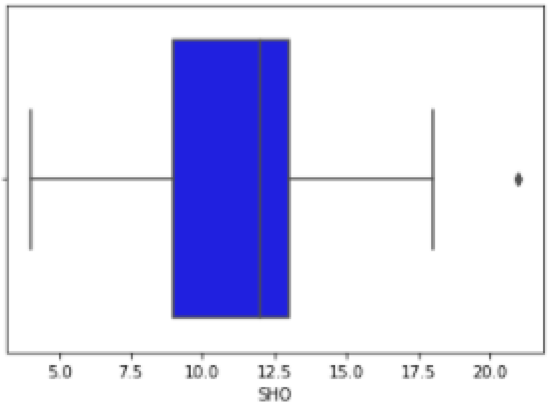

```
In [226]: sns.boxplot(data_mod['ERA'],color="pink")
```

Out[226]: <AxesSubplot:xlabel='ERA'>



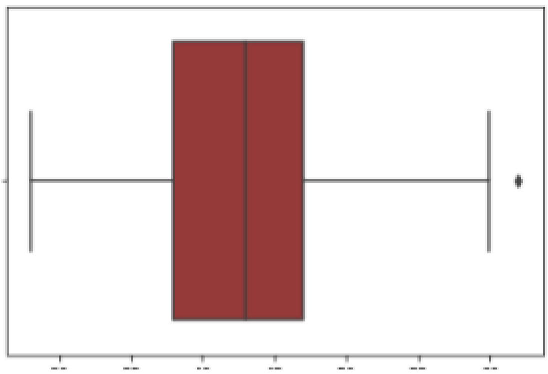

```
In [227]: sns.boxplot(data_mod['SHO'],color="blue")
```

Out[227]: <AxesSubplot:xlabel='SHO'>



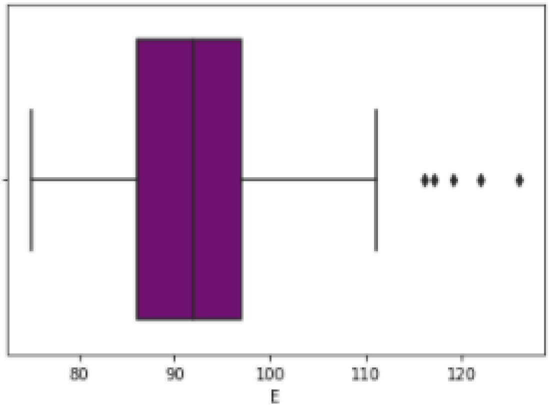

```
In [228]: sns.boxplot(data_mod['SV'],color="brown")
```

Out[228]: <AxesSubplot:xlabel='SV'>



```
In [229]:  1 sns.boxplot(data_mod['E'],color="purple")
```

```
Out[229]: <AxesSubplot:xlabel='E'>
```



After using z_score the outliers in the features R and ERA are removed but the features SV,SHO,E are still having outliers

HANDLING SKEWNESS:

```
In [10]:  1 for col in enumerate(list(data_mod.columns.values)):
          2     print(col[1],"=",data_mod[col[1]].skew())
```

```
W = 0.11901344569985461
R = -0.21536363420992782
AB = 0.16957316834729352
H = 0.7837722117274881
2B = -0.335303936110201
3B = 0.09012434653848651
HR = 0.45086158125803544
BB = 0.15119282971519954
SO = -0.2338149185462262
SB = 0.4949657663368456
RA = 0.018155177145956613
ER = 0.018460990156758887
ERA = 0.016693217783651695
CG = 0.8549795901105167
SHO = 0.5269430585305683
SV = 0.6274804879503074
E = 0.8402711976867623
```

the features H,CG,SHO,SV,E having skewness

To stabilize variance, make the data more normal distribution like, improve the validity of measures of association and to remove skewness I have used power transformation n [18]:

```
col_s=['H','CG','SHO','SV','E']
data_clean=data_mod
from sklearn.preprocessing import power_transform
data_clean[col_s]=power_transform (data_mod[col_s])
```

n [19]:

```
for col in enumerate(list(data_clean.columns.values)):
    print(col[1],"=",data_clean[col[1]].skew())
```

```
W = 0.11901344569985461
R = -0.21536363420992782
AB = 0.16957316834729352
H = 0
2B = -0.335303936110201
3B = 0.09012434653848651
HR = 0.45086158125803544
BB = 0.15119282971519954
SO = -0.2338149185462262
SB = 0.4949657663368456
RA = 0.018155177145956613
ER = 0.018460990156758887
ERA = 0.016693217783651695
CG = -0.045947323970913174
SHO = 0.000529365035686 8707
SV = -0.0009249344497408174
E = 0.06558547868786976
```

Skewness is completely removed

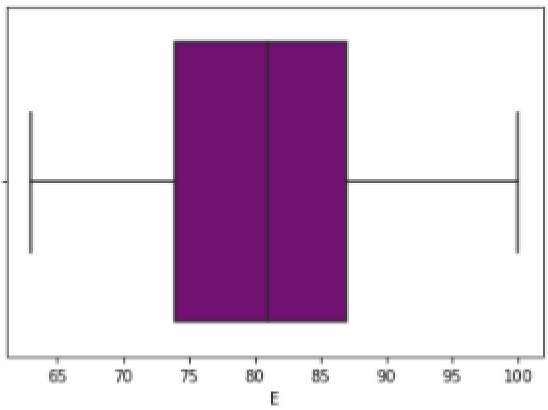
I have used Power transform on skewed data to make it symmetric, and then fit it to a symmetric distribution

:

```
sns.boxplot(data_clean['E'],color="purple")
```

:

```
<AxesSubplot:xlabel='E'>
```

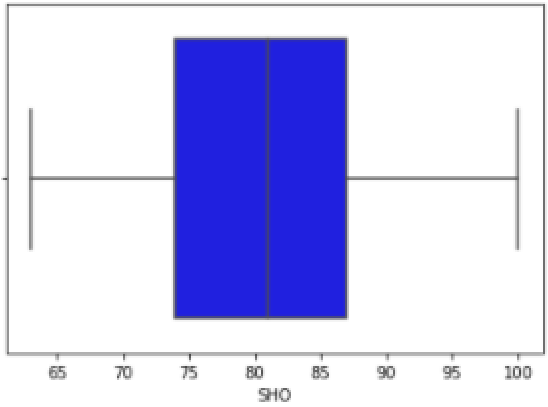


Using zscore the outliers in the feature E is completely removed and the data loss after removing outlier is 3.333333%

In [38]:

```
sns.boxplot(data_clean['SHO'],color="blue")
```

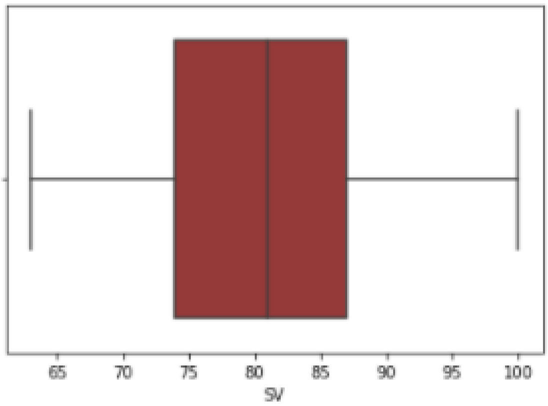
Out[38]: <AxesSubplot:xlabel='SHO'>



In [39]:

```
z3 = np.abs(stats.zscore(data_clean['SV']))
data_clean['SV'] = data_clean[(z3<3)]
sns.boxplot(data_clean['SV'],color="brown")
```

Out[39]: <AxesSubplot:xlabel='SV'>



I have used zscore to remove outliers in the features SV, SHO and E.

SCALING:

__feature scaling transforming un scaled data into scaled data using min max scalining technique

```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
scaled = scaler.fit_transform(x1)
```

MODELLING:

___MODELING

In [108]:

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
import xgboost as xgb
from sklearn.model_selection import train_test_split

```

In [109]:

```
x=scaled
y=y1
```

In [110]:

```
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.30,random_state=7)
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
models=[KNeighborsRegressor(),SVR(),DecisionTreeRegressor(),LinearRegression(),Lasso(),Ridge(),
        RandomForestRegressor(),GradientBoostingRegressor(),xgb.XGBRegressor(objective="reg:squarederror")]
maelist=[]
mselist=[]
rmselist=[]
r2list=[]
```

In [111]:

```
def create_model(model):
    m=model
    m.fit(xtrain,ytrain)
    p=m.predict(xtest)

    mae=mean_absolute_error(p,ytest)
    mse=mean_squared_error(p,ytest)
    rmse=np.sqrt(mean_squared_error(p,ytest))
    r2=r2_score(ytest,p)

    maelist.append(mae)
    mselist.append(mse)
    rmselist.append(rmse)
    r2list.append(r2)
```

```
KNeighborsRegressor()
Mean absolute error 5.266666666666666
Mean squared error 35.76444444444443
Root Mean squared error 5.980338154690287
R2 Score 0.6688477366255146
------------------------------------------------------------------------------------------
SVR()
Mean absolute error 7.641601405900634
Mean squared error 76.28996804828544
Root Mean squared error 8.734412862252702
R2 Score 0.29361140696031995
------------------------------------------------------------------------------------------
DecisionTreeRegressor()
Mean absolute error 1.7777777777777777
Mean squared error 5.111111111111111
Root Mean squared error 2.260776661041756
R2 Score 0.9526748971193416
------------------------------------------------------------------------------------------
LinearRegression()
Mean absolute error 7.894919286223336e-15
Mean squared error 1.1219355096476613e-28
Root Mean squared error 1.0592145720521698e-14
R2 Score 1.0
------------------------------------------------------------------------------------------
Lasso()
Mean absolute error 3.171414003610454
Mean squared error 15.237323453470554
Root Mean squared error 3.9035014350542454
R2 Score 0.8589136717271245
------------------------------------------------------------------------------------------
Ridge()
Mean absolute error 2.8099060243455765
Mean squared error 8.601635421789048
Root Mean squared error 2.9328544835687036
R2 Score 0.9203552275760273
------------------------------------------------------------------------------------------
RandomForestRegressor()
Mean absolute error 3.034444444444443
Mean squared error 12.24765555555555
Root Mean squared error 3.4996650633389974
R2 Score 0.8865957818930041
```

```
------------------------------------------------------------------------------------------
GradientBoostingRegressor()
Mean absolute error 2.3271601042110572
Mean squared error 7.157694017274431
Root Mean squared error 2.6753867042493935
R2 Score 0.9337250553956071
------------------------------------------------------------------------------------------
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.300000012, max_delta_step=0, max_depth=6,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=100, n_jobs=4, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
Mean absolute error 2.9089228312174478
Mean squared error 10.716167470886527
Root Mean squared error 3.2735557839888
R2 Score 0.900776227121421
------------------------------------------------------------------------------------------
Minimum Mean Absolute error is shown by  LinearRegression() 7.894919286223336e-15
Minimum Mean squared error is shown by  LinearRegression() 1.1219355096476613e-28
Minimum Root Mean squared error is shown by  LinearRegression() 1.0592145720521698e-14
Maximun R2 Score is shown by  LinearRegression() 1.0
```

Cross Validation:

In order to avoid over fitting, Cross-validation is used to estimate the skill of a machine learning model on unseen data.

```
n [139]:
from sklearn.model_selection import cross_val_score
k=KNeighborsRegressor()
scores=cross_val_score(k,x,y,scoring='r2',cv=5)
scorel.append(scores)
scores

ut[139]: array([ 0.5065    , -0.22736842,  0.56332915,  0.59501639,  0.90259434])
```

```
n [140]:
from sklearn.model_selection import cross_val_score
svr=SVR()
scores=cross_val_score(svr,x,y,scoring='r2',cv=5)
scorel.append(scores)
scores

ut[140]: array([ 0.19800765, -0.43552919,  0.14123261, -0.19247998,  0.41517121])
```

```
n [141]:
from sklearn.model_selection import cross_val_score
dt=DecisionTreeRegressor()
scores=cross_val_score(dt,x,y,scoring='r2',cv=5)
scorel.append(scores)
scores

ut[141]: array([0.88068182, 0.18421053, 0.87787537, 0.89016393, 0.89976415])
```

```
n [142]:
from sklearn.model_selection import cross_val_score
lr=LinearRegression()
scores=cross_val_score(lr,x,y,scoring='r2',cv=5)
scorel.append(scores)
scores

ut[142]: array([1., 1., 1., 1., 1.])
```

```
n [143]:
from sklearn.model_selection import cross_val_score
l=Lasso()
scores=cross_val_score(l,x,y,scoring='r2',cv=5)
scorel.append(scores)
scores

ut[143]: array([0.88553127, 0.84887851, 0.72013556, 0.64974046, 0.88639659])
```

In [144]:

```
from sklearn.model_selection import cross_val_score
rid=Ridge()
scores=cross_val_score(rid,x,y,scoring='r2',cv=5)
scorel.append(scores)
scores
```

Out[144]:

```
array([0.8984672 , 0.76298028, 0.90904629, 0.92935865, 0.99225316])
```

In [145]:

```
from sklearn.model_selection import cross_val_score
rf=RandomForestRegressor()
scores=cross_val_score(rf,x,y,scoring='r2',cv=5)
scorel.append(scores)
scores
```

Out[145]:

```
array([0.9720642 , 0.95058684, 0.859889  , 0.86196803, 0.9760625 ])
```

In [146]:

```
from sklearn.model_selection import cross_val_score
gb=GradientBoostingRegressor()
scores=cross_val_score(gb,x,y,scoring='r2',cv=5)
scorel.append(scores)
scores
```

Out[146]:

```
array([0.98639233, 0.8546094 , 0.92123603, 0.90768891, 0.98418366])
```

In [147]:

```
from sklearn.model_selection import cross_val_score
xb=xgb.XGBRegressor()
scores=cross_val_score(xb,x,y,scoring='r2',cv=5)
scorel.append(scores)
scores
```

Out[147]:

```
array([0.82751376, 0.94214528, 0.88634985, 0.89240543, 0.94730646])
```

Difference of predicted model and cross validation score:

- KNeighborsRegressor - 0.2337466
- SVR()-0.12155981
- DecisionTreeRegressor()-0.05291075
- LinearRegression() -0
- Lasso - 0.02748291
- Ridge()-0.07189794
- RandomForestRegressor()-0.08946672
- GradientBoostingRegressor()-0.0504586
- XGBRegressor-0.046530

from the observation Linear regression model model has least difference so I'm selecting Linear regression as best model

Hyper Tuning:

___Hyper tuning

In [150]:
```
from sklearn.model_selection import GridSearchCV
parameters = { "normalize":[True, False], "copy_X":[True, False],
               "fit_intercept": [True, False]
               }
grid = GridSearchCV(LinearRegression(), param_grid = parameters, cv = 5, scoring = "r2")
```

In [151]:
```
grid.fit(xtrain,ytrain)

print("Best_parameters",grid.best_params_)

```

```
Best_parameters {'copy_X': True, 'fit_intercept': True, 'normalize': True}
```

Best parameters: **{**'copy_X': True, 'fit_intercept': True, 'normalize': True}
Modelling using best parameter and best model:

In [154]:
```
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.25,random_state=1)
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score,accuracy_score
model=LinearRegression(copy_X=True,fit_intercept=True,normalize=True)
model.fit(xtrain,ytrain)
p=model.predict(xtest)
acc=model.score(xtest,ytest)
mae=mean_absolute_error(p,ytest)
mse=mean_squared_error(p,ytest)
rmse=np.sqrt(mean_squared_error(p,ytest))
r2=r2_score(ytest,p)
print('Accuracy',acc)
print('Mean absolute error',mae)
print('Mean squared error',mse)
print('Root Mean squared error',rmse)
print('r2 score',r2)

```

```
Accuracy 1.0
Mean absolute error 1.0658141036401503e-14
Mean squared error 2.0194839173657902e-28
Root Mean squared error 1.4210854715202004e-14
r2 score 1.0
```

Final model after hyper tuning its retaining 100%accuracy and error values got reduced

Conclusion:

I have developed a model to predict number of wins with 100% accuracy

Saving the model

In [79]:
```
from joblib import dump
dump(model, 'model_baseball.joblib')
```

Out[79]: ['model_baseball.joblib']

In [80]:
```
from joblib import load
loaded = load('model_baseball.joblib')
```