Baseball

March 9, 2023

0.0.1 In this Analysis Linear Regression, Neural Network, XG Boost and Logistc Regression method is used

[1]:
```
import pandas as pd
import numpy as np
from tensorflow.keras import layers
from tensorflow.keras import models
from sklearn.metrics import r2_score, accuracy_score
```

[2]:
```
from sklearn.model_selection import KFold
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error,␣
↪confusion_matrix
import math
from sklearn.model_selection import train_test_split, GridSearchCV
```

[3]:
```
# to ignore warnings
import warnings
warnings.filterwarnings("ignore")
```

0.0.2 With the help of Pandas, read the ".csv" file and performing some task

[4]:
```
data1 = pd.read_csv("baseball.csv")
```

[5]:
```
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 337 entries, 0 to 336
Data columns (total 17 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   @inputs Batting_average  337 non-null    float64
 1   On-base_percentage       337 non-null    float64
 2   Runs                     337 non-null    int64
 3   Hits                     337 non-null    int64
 4   Doubles                  337 non-null    int64
 5   Triples                  337 non-null    int64
 6   HomeRuns                 337 non-null    int64
 7   Runs_batted_in           337 non-null    int64
```

```
 8   Walks                    337 non-null    int64
 9   Strike-Outs              337 non-null    int64
 10  Stolen_bases             337 non-null    int64
 11  Errors                   337 non-null    int64
 12  Free_agency_eligibility  337 non-null    int64
 13  Free_agent               337 non-null    int64
 14  Arbitration_eligibility  337 non-null    int64
 15  Arbitration              337 non-null    int64
 16  @outputs Salary          337 non-null    int64
dtypes: float64(2), int64(15)
memory usage: 44.9 KB
```

0.0.3 Renaming the columns for better understanding

[6]:
```
data1.rename(columns = {'@inputs Batting_average':'Batting_average','@outputs␣
 ↪Salary':'Salary'}, inplace = True)
```

[7]:
```
data11 = data1
```

[8]:
```
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 337 entries, 0 to 336
Data columns (total 17 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Batting_average          337 non-null    float64
 1   On-base_percentage       337 non-null    float64
 2   Runs                     337 non-null    int64
 3   Hits                     337 non-null    int64
 4   Doubles                  337 non-null    int64
 5   Triples                  337 non-null    int64
 6   HomeRuns                 337 non-null    int64
 7   Runs_batted_in           337 non-null    int64
 8   Walks                    337 non-null    int64
 9   Strike-Outs              337 non-null    int64
 10  Stolen_bases             337 non-null    int64
 11  Errors                   337 non-null    int64
 12  Free_agency_eligibility  337 non-null    int64
 13  Free_agent               337 non-null    int64
 14  Arbitration_eligibility  337 non-null    int64
 15  Arbitration              337 non-null    int64
 16  Salary                   337 non-null    int64
dtypes: float64(2), int64(15)
memory usage: 44.9 KB
```

[9]:
```
data1.head(4)
```

[9]:

```
   Batting_average  On-base_percentage  Runs  Hits  Doubles  Triples  \
0            0.271               0.328    74   161       22        6
1            0.264               0.318    24    48        7        0
2            0.251               0.338   101   141       35        3
3            0.224               0.274    28    94       21        1

   HomeRuns  Runs_batted_in  Walks  Strike-Outs  Stolen_bases  Errors  \
0        12              58     49          133            23      17
1         1              22     15           18             0       7
2        32             105     71          104            34       6
3         1              44     27           54             2       7

   Free_agency_eligibility  Free_agent  Arbitration_eligibility  Arbitration  \
0                        1           1                        0            0
1                        0           0                        0            0
2                        0           0                        1            0
3                        1           1                        0            0

   Salary
0     109
1     160
2    2700
3     550
```

data1.tail(4)

[10]:

```
data1.isnull().any()
```

[10]:

```
Batting_average            False
On-base_percentage         False
Runs                       False
Hits                       False
Doubles                    False
Triples                    False
HomeRuns                   False
Runs_batted_in             False
Walks                      False
Strike-Outs                False
Stolen_bases               False
Errors                     False
Free_agency_eligibility    False
Free_agent                 False
Arbitration_eligibility    False
Arbitration                False
Salary                     False
dtype: bool
```

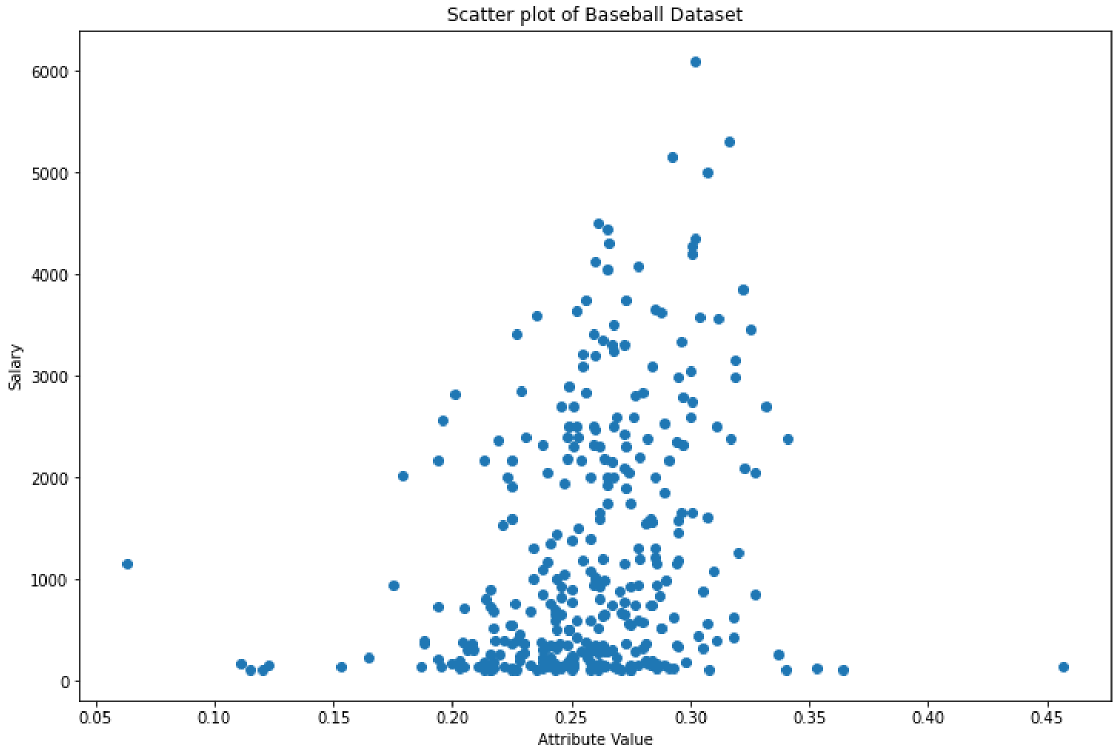
[11]:

```
data1.isnull().sum()
```

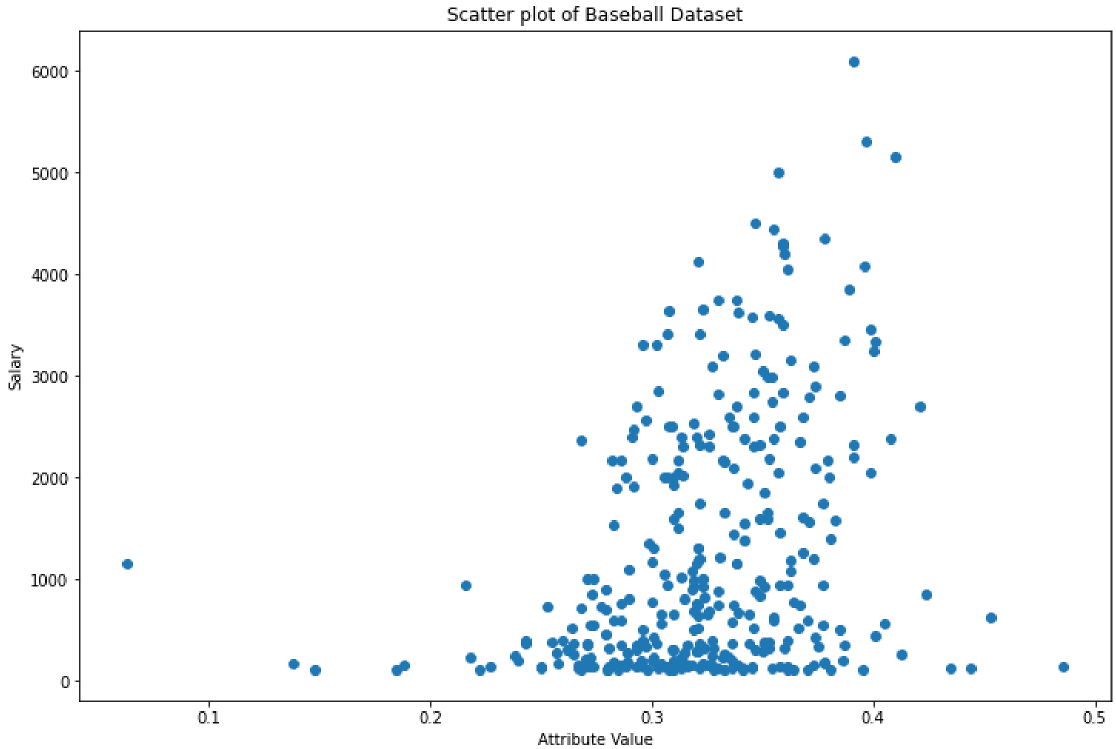
[11]:
```
Batting_average            0
On-base_percentage         0
Runs                       0
Hits                       0
Doubles                    0
Triples                    0
HomeRuns                   0
Runs_batted_in             0
Walks                      0
Strike-Outs                0
Stolen_bases               0
Errors                     0
Free_agency_eligibility    0
Free_agent                 0
Arbitration_eligibility    0
Arbitration                0
Salary                     0
dtype: int64
```

[12]:
```python
import seaborn as sns
import matplotlib.pyplot as plt
```

[13]:
```python
# Create the scatter plot
fig, ax = plt.subplots(figsize=(12,8))
ax.scatter(data1["Batting_average"], data1["Salary"])

ax.set_xlabel("Attribute Value")
ax.set_ylabel("Salary")
ax.set_title("Scatter plot of Baseball Dataset")
plt.show()

#scatterplot is best used to detect Trends, Outilier detection
```

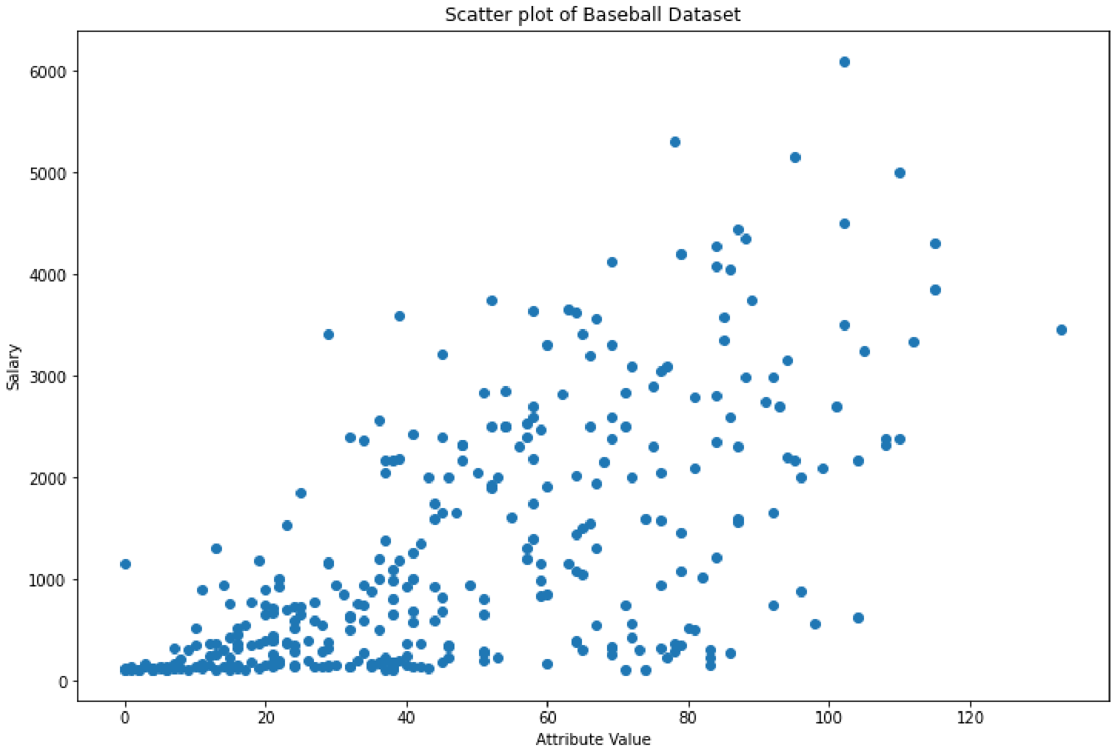


[14]:
```
fig, ax = plt.subplots(figsize=(12,8))
ax.scatter(data1["On-base_percentage"], data1["Salary"])

ax.set_xlabel("Attribute Value")
ax.set_ylabel("Salary")
ax.set_title("Scatter plot of Baseball Dataset")
plt.show()
```

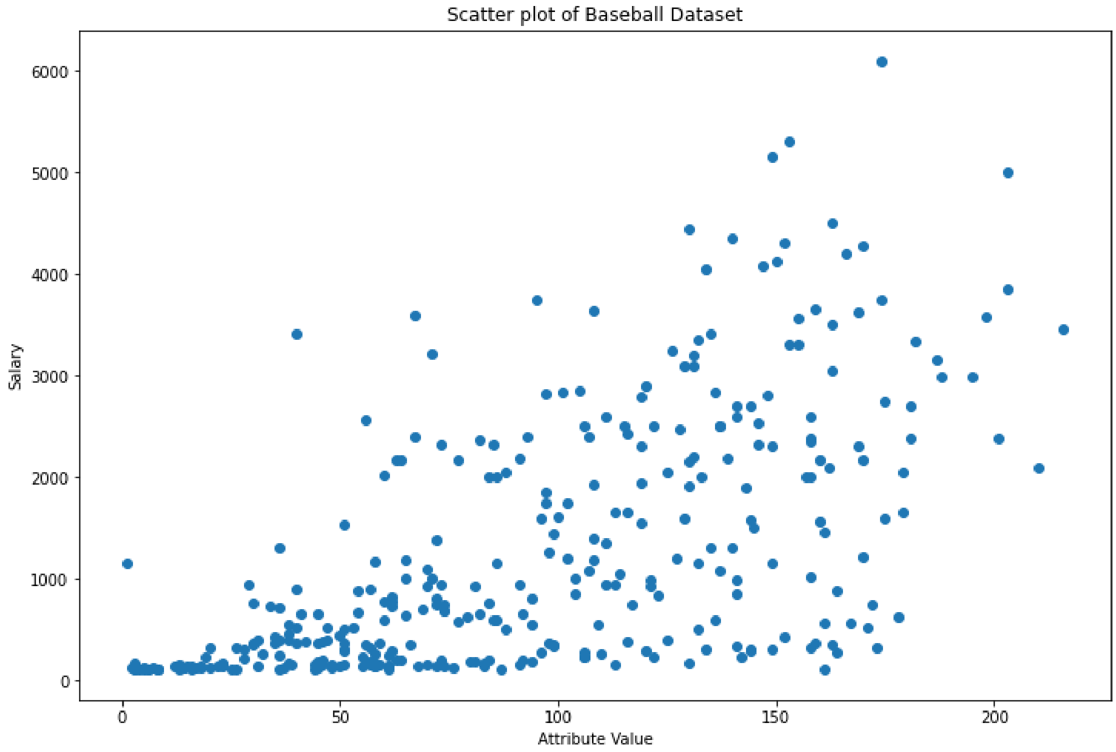


[15]:

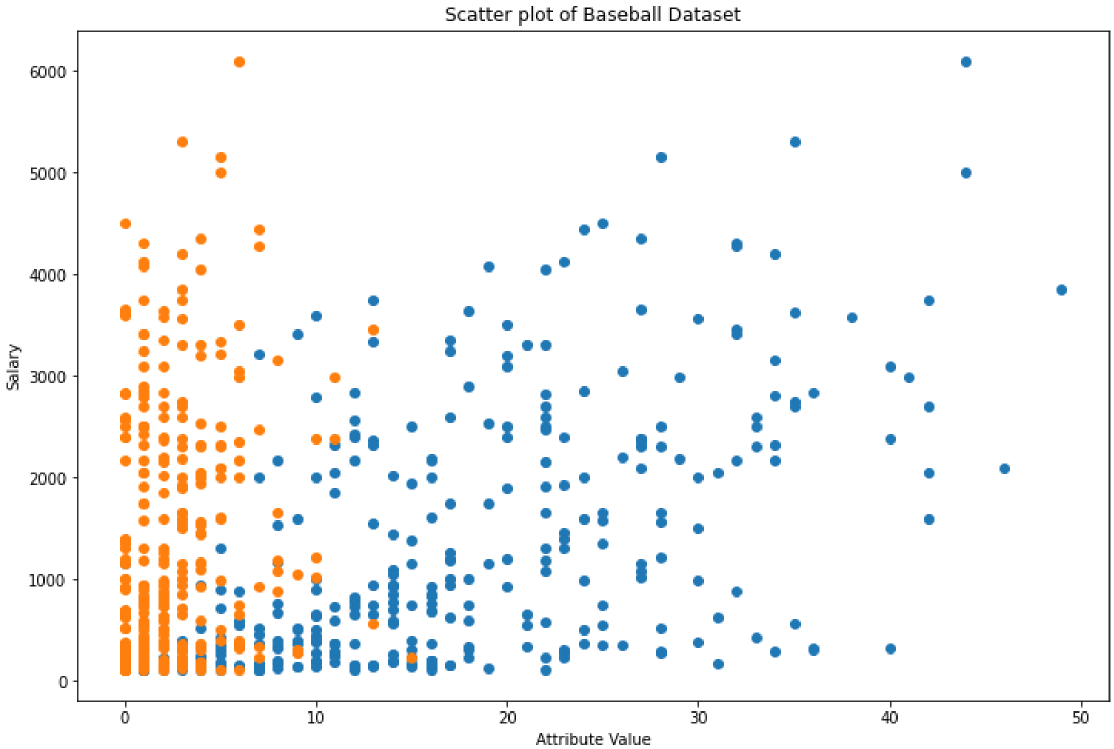
```
fig, ax = plt.subplots(figsize=(12,8))
ax.scatter(data1["Runs"], data1["Salary"])

ax.set_xlabel("Attribute Value")
ax.set_ylabel("Salary")
ax.set_title("Scatter plot of Baseball Dataset")
plt.show()
```



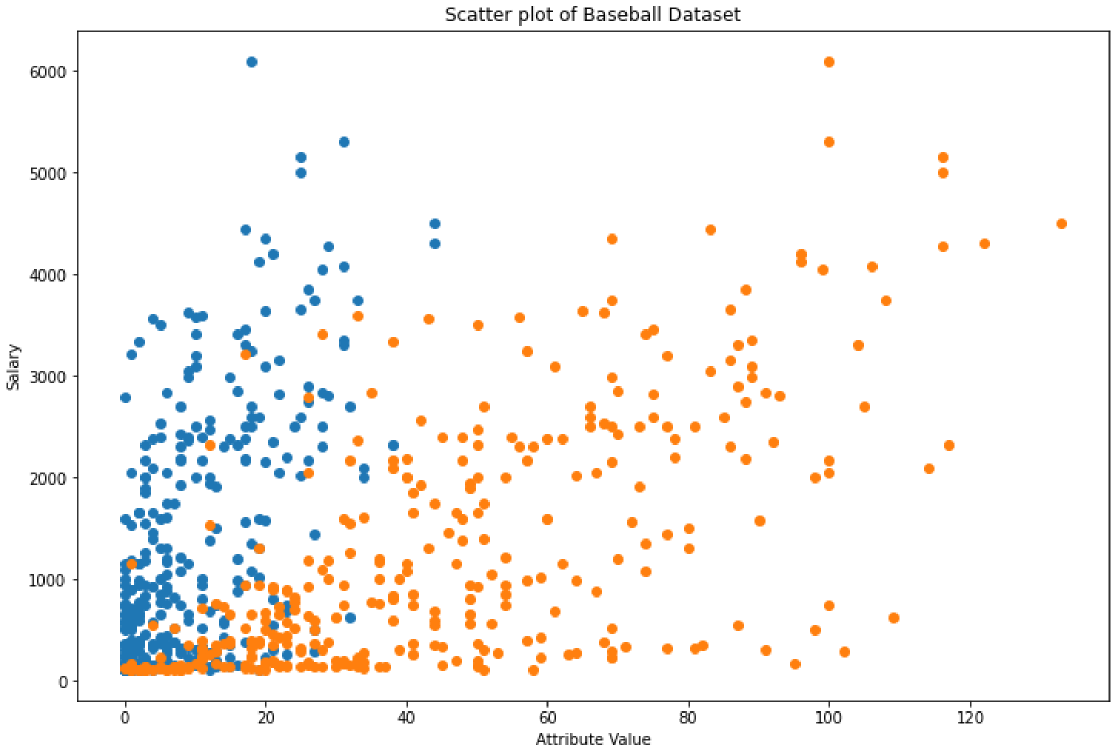
[16]:
```
fig, ax = plt.subplots(figsize=(12,8))
ax.scatter(data1["Hits"], data1["Salary"])

ax.set_xlabel("Attribute Value")
ax.set_ylabel("Salary")
ax.set_title("Scatter plot of Baseball Dataset")
plt.show()
```



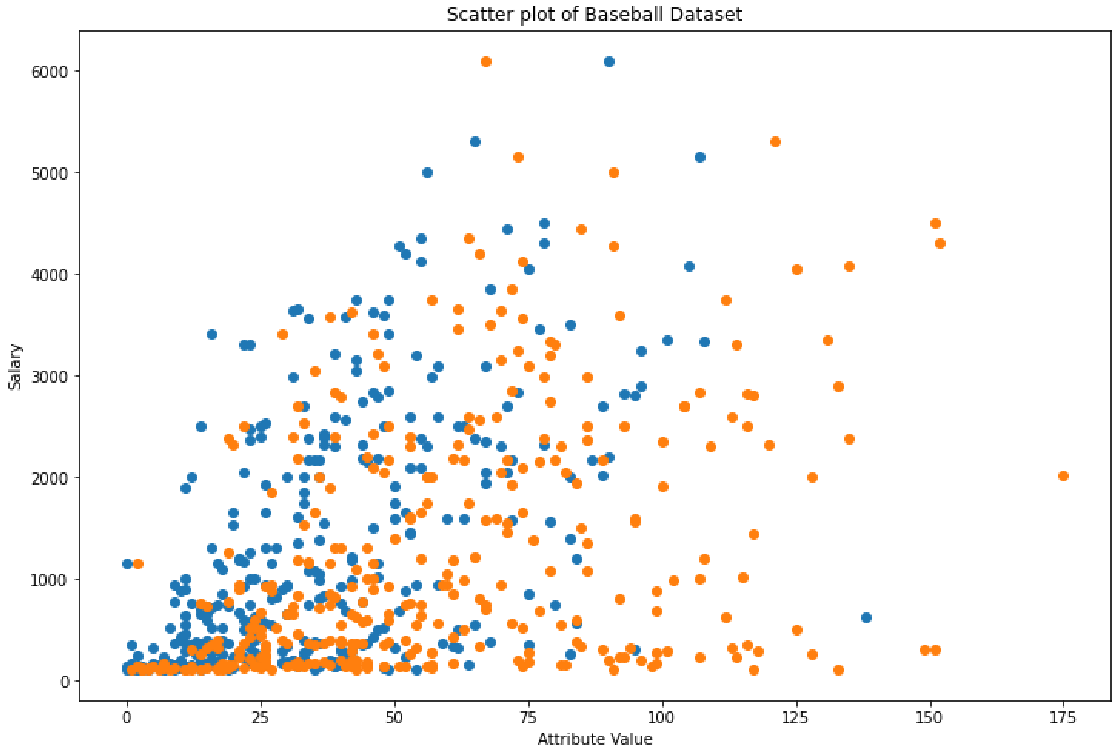
[17]:
```
fig, ax = plt.subplots(figsize=(12,8))
ax.scatter(data1["Doubles"], data1["Salary"])
ax.scatter(data1["Triples"], data1["Salary"])

ax.set_xlabel("Attribute Value")
ax.set_ylabel("Salary")
ax.set_title("Scatter plot of Baseball Dataset")
plt.show()
```



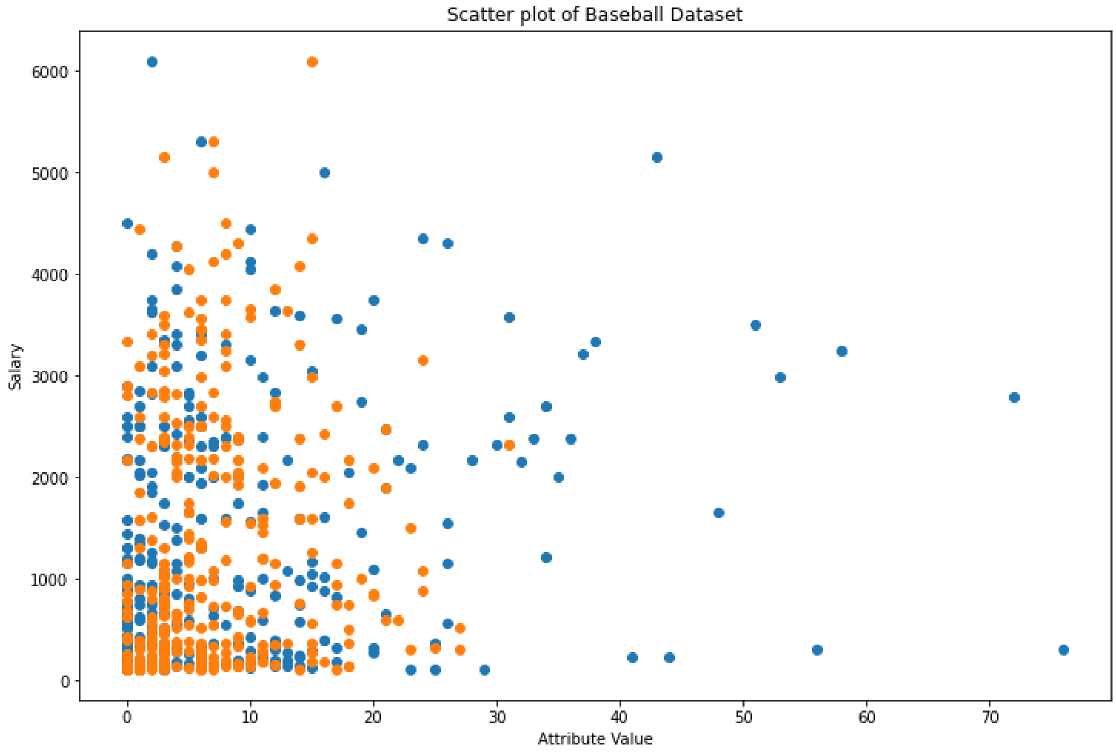
[18]:
```
fig, ax = plt.subplots(figsize=(12,8))

ax.scatter(data1["HomeRuns"], data1["Salary"])
ax.scatter(data1["Runs_batted_in"], data1["Salary"])
ax.set_xlabel("Attribute Value")
ax.set_ylabel("Salary")
ax.set_title("Scatter plot of Baseball Dataset")
plt.show()
```



[19]:
```
fig, ax = plt.subplots(figsize=(12,8))

ax.scatter(data1["Walks"], data1["Salary"])
ax.scatter(data1["Strike-Outs"], data1["Salary"])
ax.set_xlabel("Attribute Value")
ax.set_ylabel("Salary")
ax.set_title("Scatter plot of Baseball Dataset")
plt.show()
```



[20]:
```
fig, ax = plt.subplots(figsize=(12,8))

ax.scatter(data1["Stolen_bases"], data1["Salary"])
ax.scatter(data1["Errors"], data1["Salary"])
ax.set_xlabel("Attribute Value")
ax.set_ylabel("Salary")
ax.set_title("Scatter plot of Baseball Dataset")
plt.show()
```




```
[21]: # Split the dataset into training and testing sets
X = data1.drop("Salary", axis=1) # Independent variables
y = data1["Salary"] # Dependent variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

0.0.4 Building the neural network model

```
[22]: def build_model():
    model = models.Sequential()
    model.add(layers.Dense(64, activation='relu', input_shape=(X.shape[1],)))
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(1))
    model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
    return model
```

0.0.5 Define and perform K-fold cross validation

```
[23]: kf = KFold(n_splits=5)
```

```
[24]: # Perform k-fold cross validation
for train_index, test_index in kf.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
```

```
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
```

0.0.6 With nural network

[25]:
```
# Standardize the training and testing data
X_train_mean = X_train.mean()
X_train_std = X_train.std()
X_train = (X_train - X_train_mean) / X_train_std
X_test = (X_test - X_train_mean) / X_train_std
```

[26]:
```
# Build the model
model = build_model()
```

[27]:
```
history = model.fit(X_train, y_train, epochs=30 , batch_size=5, verbose=0)
```

[28]:
```
r2_scores = []
```

[29]:
```
# Predict the target values for the testing data
y_pred = model.predict(X_test).flatten()
```

```
3/3 [==============================] - 0s 977us/step
```

[30]:
```
r2 = r2_score(y_test, y_pred)
r2_scores.append(r2)
```

[31]:
```
print('Coefficient of determination R-squared (R2):', np.mean(r2_scores))
```

```
Coefficient of determination R-squared (R2): 0.5381060070607421
```

0.0.7 Initialize the Logistic regression and fit the model

[32]:
```
# Create a binary target variableb
data1["HighSalary"] = (data1["Salary"] > data1["Salary"].median()).astype(int)
```

[33]:
```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data1[["Batting_average",␣
↪"On-base_percentage", "Runs", "Hits", "Doubles", "Triples", "HomeRuns",␣
↪"Runs_batted_in", "Walks", "Strike-Outs", "Stolen_bases", "Errors"]],␣
↪data1["HighSalary"], test_size=0.3, random_state=42)
```

[34]:
```
model = LogisticRegression()
```

[35]:
```
model.fit(X_train, y_train)
```

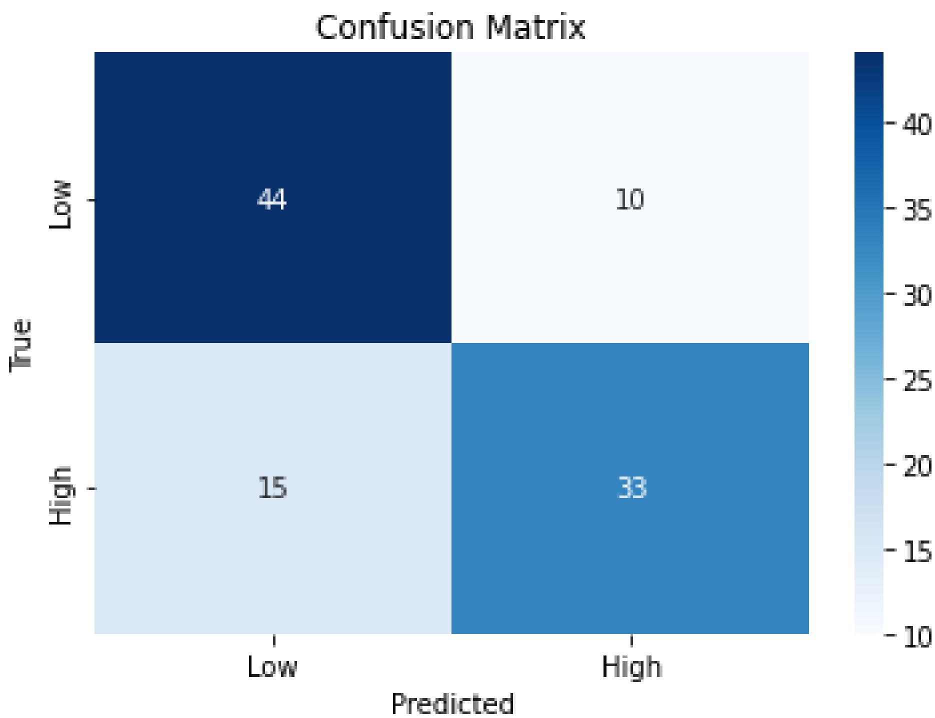
[35]:
```
LogisticRegression()
```

[36]:
```
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

[37]:
```
print("Accuracy: {:.2f}%".format(accuracy*100))
```

```
Accuracy: 75.49%
```

[38]:
```
# Create a confusion matrix
cm = confusion_matrix(y_test, y_pred, labels=[0, 1])

# Plot the confusion matrix
sns.heatmap(cm, annot=True, cmap="Blues", fmt="g", xticklabels=["Low", "High"],␣
↪yticklabels=["Low", "High"])
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix")
plt.show()
```



0.0.8 Intialize the linear regression model and fit the model

[39]:
```
X_train, X_test, y_train, y_test = train_test_split(data11.drop("Salary",␣
↪axis=1), data11["Salary"], test_size=0.2, random_state=42)
```

[40]:
```
model = LinearRegression()
```

[41]:
```
model.fit(X_train, y_train)
```

[41]: LinearRegression()

[42]:
```
# Make predictions on the test data
y_pred = model.predict(X_test)
```

[43]:
```
# Calculating the R2 Score
r2 = r2_score(y_test, y_pred)
print("Coefficient of determination R-squared (R2): ", r2)
```

```
Coefficient of determination R-squared (R2):  0.7858719726763712
```

0.0.9 By using XG Boost algorithm

[44]:
```
import xgboost as xgb
```

[45]:
```
# Define the XGBoost model
# xgb_model = xgb.XGBRegressor(objective='reg:squarederror')
xgb_model = xgb.XGBRegressor()
```

[46]:
```
param_grid = {
    'learning_rate': [0.01, 0.05, 0.1],
    'max_depth': [3, 5, 7],
    'n_estimators': [50, 100, 200]
}
```

[47]:
```
# Setting up cross-validation
xgb_cv = GridSearchCV(xgb_model, param_grid, cv=5)
```

[48]:
```
xgb_model.fit(X_train, y_train)
```

[48]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=None, …)
```

[49]:
```
# Predict the target values for the testing data
y_pred = xgb_model.predict(X_test)
```

[50]:
```
r2_scores = []
```

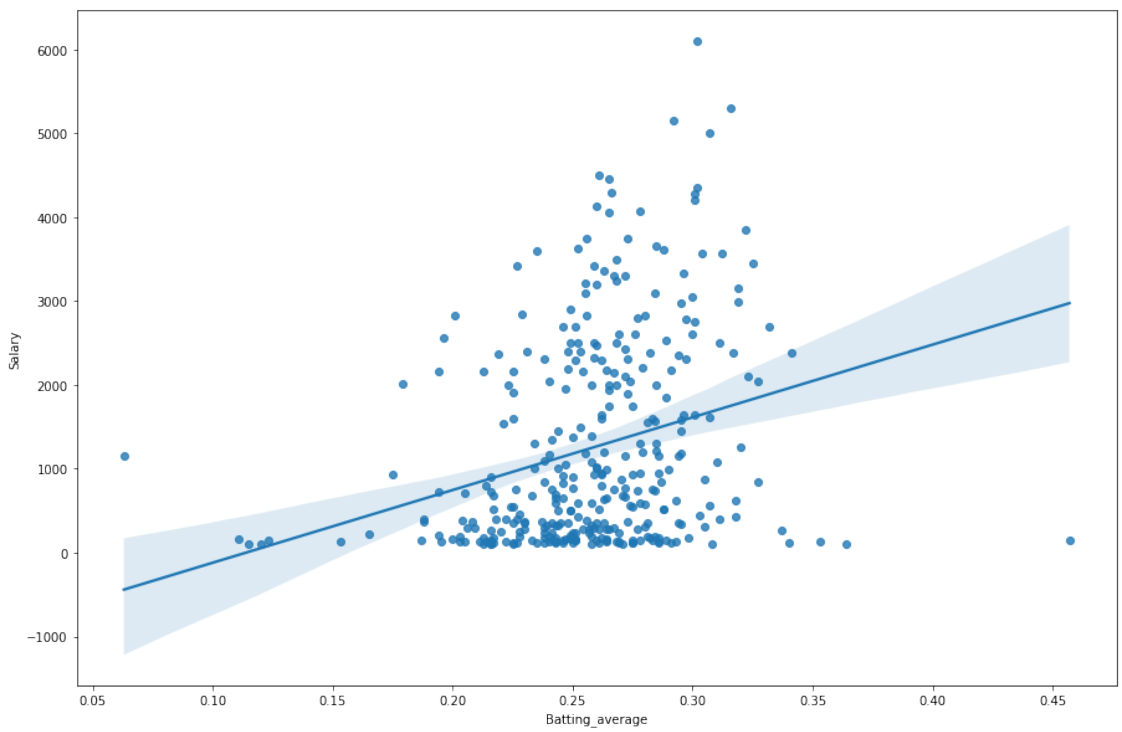
[51]:
```
# Evaluate the model on the testing data
r2 = r2_score(y_test, y_pred)
```

[52]:
```
r2_scores.append(r2)
```

[53]:
```
# Compute the mean evaluation metrics over all the folds
print('Coefficient of determination R-squared (R2):', np.mean(r2_scores))
```

```
Coefficient of determination R-squared (R2): 0.7914602652460396
```

[54]:
```
# Selecting the relevant columns for the regression
x = data1['Batting_average']
y = data1['Salary']

#enlarging the figure for better visualization
fig, ax = plt.subplots(figsize=(15, 10))

# Fit a regression model
sns.regplot(x, y, ax=ax)

# Show the plot
plt.show()
```



By Sushan Shankar