Points Per Game Change Log

17/02/25

- For this model I am going to use KNN
- I am going to build an NBA player comparison model
- I am going to obtain the data from a web scraper on (https://www.basketball-reference.com/leagues/NBA_2025_per_game.html) this site.
- My aim is for me to be able to give the model stats from a player and it to be able to predict how many points per game they will average.
- I chose a KNN Regressor as I am trying to predict a specific value rather than trying to categorise a value, I want the model to predict an exact points per game that the player might get.
- My model is currently not working the best due to its inability to deal with outliers.
- This has now been fixed by using log transformation.
- Changed the amount of neighbours used to ttry and increase the r2 score.
- Final model r2 accuracy score is 80%.

```
import requests

url =
'https://www.basketball-reference.com/leagues/NBA_2015_per_game.html'

headers = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/115.0.0.0
Safari/537.36",
    "Accept-Language": "en-US,en;q=0.9",
}

response = requests.get(url, headers=headers)

response.encoding = "utf-8"

with open("old-data.html", "w", encoding="utf-8") as file:
    file.write(response.text)
```

- My first step is to extract the raw html from the web page so that I can then extract the necessary table from there. I am using headers here so that the bot appears as a real user and not a bot as some sites will block it from accessing the site if it detects bot behaviour.

- ChatGPT gave me the response.encoding line, I need to use this here as when I was importing the data through html without utf 8 encoding, it was unable to recognise certain player names as they have symbols from other languages in them.

```
from bs4 import BeautifulSoup
import pandas as pd
```

```
# HTML file containing the table
html_file = "old-data.html"

# Load the HTML file with BeautifulSoup
with open(html_file, "r", encoding="utf-8") as file:
    soup = BeautifulSoup(file, "lxml")  # You can also try 'lxml' if
needed

# Locate the <table> with id 'per_game_stats'
table = soup.find("table", {"id": "per_game_stats"})

if table:
    print("Found <table id='per_game_stats'>.")

    # Parse the table using pandas
    try:
        df = pd.read_html(str(table))[0]
        # Save the DataFrame to a CSV file
        df.to_csv("15-nba.csv", index=False)
        print("Player stats saved to '15-nba.csv'")
    except ValueError as e:
        print(f"Error parsing the table: {e}")
else:
    print("No <table> with id 'per_game_stats' found.")
```

```
Found <table id='per_game_stats'>.
Player stats saved to '15-nba.csv'
```

```
/var/folders/jp/fbwfkpy50vs_7m6dlkzl468h0000gn/T/
ipykernel_57364/3989037152.py:19: FutureWarning: Passing literal html
to 'read_html' is deprecated and will be removed in a future version.
To read from a literal string, wrap it in a 'StringIO' object.
  df = pd.read_html(str(table))[0]
```

- I am following a similar template to what I did to obtain data for my 4th year project which can be seen here (https://github.com/DoyleAaron/4th-year-final-project), it is a great method of getting data but there can be a lot of data cleaning to ensure it is ready to be used in a machine learning model.
- Now I need to go back and get a few more years worth of data so that I have a substantial amount of data to build my model from. This is done manually by going through and changing the url for each year and saving it to a new CSV file.
- I have now gone back and obtained the last 10 years worth of per game data for all NBA players, I now need to combine and tidy the data so that I can use it in my model.

```
csv_files = ["15-nba.csv", "16-nba.csv", "17-nba.csv", "18-nba.csv",
"19-nba.csv", "20-nba.csv", "21-nba.csv", "22-nba.csv", "23-nba.csv",
"24-nba.csv", "25-nba.csv"]

df = pd.concat([pd.read_csv(file) for file in csv_files],
ignore_index=True)

df.to_csv("all-season-data.csv", index=False)

df = df.dropna()
#  Here I am dropping the rows with NaN values - AD
```

- As all of the data follows the same format as it is coming from the same website I can use pd.concat which takes in the array of csv files that I entered and combines them into one big csv file.

```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import numpy as np
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import train_test_split

df = pd.read_csv("all-season-data.csv")

print(df.head())

# I am just importing the data and printing the first 5 rows to ensure
that the data is loaded correctly. - AD

    Rk              Player   Age Team Pos     G    GS    MP   FG   FGA
...  \
0  1.0  Russell Westbrook  26.0  OKC  PG  67.0  67.0  34.4  9.4  22.0
...
1  2.0       James Harden  25.0  HOU  SG  81.0  81.0  36.8  8.0  18.1
...
2  3.0       Kevin Durant  26.0  OKC  SF  27.0  27.0  33.8  8.8  17.3
...
3  4.0       LeBron James  30.0  CLE  SF  69.0  69.0  36.1  9.0  18.5
...
4  5.0      Anthony Davis  21.0  NOP  PF  68.0  68.0  36.1  9.4  17.6
...

   ORB  DRB   TRB  AST  STL  BLK  TOV   PF   PTS
Awards
0  1.9  5.4   7.3  8.6  2.1  0.2  4.4  2.7  28.1          MVP-
4,AS,NBA2
1  0.9  4.7   5.7  7.0  1.9  0.7  4.0  2.6  27.4          MVP-
2,AS,NBA1
2  0.6  6.0   6.6  4.1  0.9  0.9  2.7  1.5  25.4
```

```
AS
3  0.7  5.3   6.0  7.4  1.6  0.7  3.9  2.0  25.3  MVP-3,DPOY-
13,AS,NBA1
4  2.5  7.7  10.2  2.2  1.5  2.9  1.4  2.1  24.4   MVP-5,DPOY-
4,AS,NBA1

[5 rows x 31 columns]

X = df[["eFG%", "AST", "STL", "BLK", "FG%", "3P%", "FT%", "TOV", "2P
%"]]
y = df["PTS"]

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

X_train = X_train.dropna()
y_train = y_train.loc[X_train.index]
X_test = X_test.dropna()
y_test = y_test.loc[X_test.index]
```

- In this code block I am just outlining what my X and Y values are and then splitting the training and test data in a 80/20 split.
- After some issues I ran into with null values I found online to drop the null values directly from the training set as when I tried to drop null values normally from the overall dataset it wasn't working as expected.

```
knn = KNeighborsRegressor(n_neighbors=1)

knn.fit(X_train, y_train)

KNeighborsRegressor(n_neighbors=1)
```

- I am using a Regressor rather than a classifier because I want to try and predict

```
y_pred = knn.predict(X_test)

from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score

# Calculate error metrics
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print results
print("MAE:", mae)
print("MSE:", mse)
print("R2:", r2)
```

```
MAE: 2.7499617444529454
MSE: 14.730573833205815
R2: 0.5992720372697735
```

- As you can see, from a couple of training metrics that I have here my Mean Squared Error is quite high which would indicate that my model is struggling with larger differences. I have ran into a similar issue before and I fixed it by using log transformation on my data so I am going to try that and see how it affects the results.
- The mean absolute error tracks the average difference between the predictions and the actual values so it is a good indicator of how well your model is working.

```
df["log-PTS"] = np.log1p(df["PTS"])

X = df[["eFG%", "AST", "STL", "BLK", "FG%", "3P%", "FT%", "TOV", "2P
%"]]
y = df["log-PTS"]

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

X_train = X_train.dropna()
y_train = y_train.loc[X_train.index]
X_test = X_test.dropna()
y_test = y_test.loc[X_test.index]

newknn = KNeighborsRegressor(n_neighbors=1)

newknn.fit(X_train, y_train)

KNeighborsRegressor(n_neighbors=1)

y_pred = newknn.predict(X_test)

from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2_score = r2_score(y_test, y_pred)

print("MAE:", mae)
print("MSE:", mse)
print("R2:", r2_score)

MAE: 0.28072583136554236
MSE: 0.12764682964651433
R2: 0.6577841608009756
```

- As we can see now in comparison, the Mean Absolute Error and the Mean Squared Error have absolutely plummeted.

- This is down to the Log transformation that I applied. What log transformation does is transform the data from say 30 to 3.0, this is a massive help as it is essentially the same value however it removes the big disparity between it as it is only going to be out by decimal places rather than full integer values.
- We are now at 69% on the r2 score which is my main focus, I want to see if I can improve this at all.

```
from sklearn.model_selection import GridSearchCV
param_grid = {"n_neighbors": range(1, 20)}
knn = KNeighborsRegressor()
grid = GridSearchCV(knn, param_grid, cv=5, scoring="r2")
grid.fit(X_train, y_train)
print("Best k:", grid.best_params_)

Best k: {'n_neighbors': 12}
```

- ChatGPT gave me this code block. What this is doing is going through 20 amounts of neighbours and seeing which is most effective for my model.

```
df["log-PTS"] = np.log1p(df["PTS"])

X = df[["eFG%", "AST", "STL", "BLK", "FG%", "3P%", "FT%", "TOV", "2P
%"]]
y = df["log-PTS"]

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

X_train = X_train.dropna()
y_train = y_train.loc[X_train.index]
X_test = X_test.dropna()
y_test = y_test.loc[X_test.index]

updatedKNN = KNeighborsRegressor(n_neighbors=12)

updatedKNN.fit(X_train, y_train)

KNeighborsRegressor(n_neighbors=12)

y_pred = updatedKNN.predict(X_test)

from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2_score = r2_score(y_test, y_pred)

print("MAE:", mae)
print("MSE:", mse)
print("R2:", r2_score)
```

```
MAE: 0.21738889535253583
MSE: 0.07650380504019764
R2: 0.7948964818299761
```

- After updating the amount of neighbours, we have seen a massive rise in the r2 score to just under 80% which for a model like this is a great score.
- Upping the amount of neighbours boosted the r2 score and overall made the model better as now it is checking more data around it to make its prediction therefore making the model more accurate and precise in its predictions.

```
# Sample data for a player
columns = ["eFG%", "AST", "STL", "BLK", "FG%", "3P%", "FT%", "TOV",
"2P%"]
player_features = pd.DataFrame([[0.550, 9.5, 1.3, 1.0, 0.480, 0.350,
0.800, 2.5, 0.510]], columns=columns)

predicted_log_ppg = updatedKNN.predict(player_features)

# Convert back to real PPG
predicted_ppg = np.expm1(predicted_log_ppg)

print("Predicted PPG:", predicted_ppg[0])

Predicted PPG: 15.862178737991444
```

- This is just a random sample input generated by ChatGPT that takes in some synthetic sample data and then based off of those statistics predicts how many points per game this player would score in the NBA.