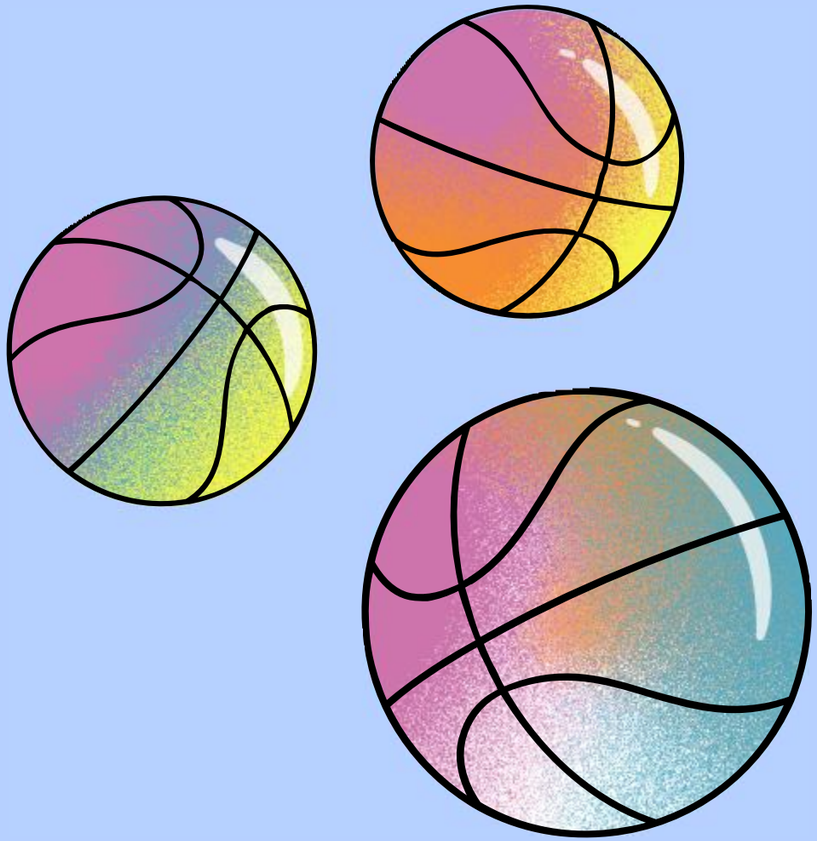
NBA STATS

KASDDUDE



MEET OUR TEAM

kasdDUDE

Adeline Sanusie

Immanuel

Frigas Hanifyan

Muhamad Akmal

Data Preprocessing

pada dataset:

- mvps
- nicknames
- player_mvp_stats
- players
- salaries
- teams

1	2	3	4
Mengecek informasi dataframe secara menyeluruh	Mengecek dan menangani missing value	Mengecek dan menangani duplikasi data	Mengecek dan menangani ambiguitas data

Mengecek informasi dataframe secara menyeluruh

```
display_both_head_and_tail(mvps_df)
```

	Rank	Player	Age	Tm	First	Pts Won	Pts Max	Share	G	MP	...	TRB	AST	STL	BLK	FG%	3P%	FT%	WS	W
0	1	Michael Jordan	27	CHI	77	891	960	928.00	82	37.0	...	6.0	5.5	2.7	1.0	539.00	312.00	851.00	20.3	3
1	2	Magic Johnson	31	LAL	10	497	960	518.00	79	37.1	...	7.0	12.5	1.3	0.2	477.00	0.32	906.00	15.4	2
2	3	David Robinson	25	SAS	6	476	960	496.00	82	37.7	...	13.0	2.5	1.5	3.9	552.00	143.00	762.00	17.0	2
3	4	Charles Barkley	27	PHI	2	222	960	231.00	67	37.3	...	10.1	4.2	1.6	0.5	0.57	284.00	722.00	13.4	2
4	5	Karl Malone	27	UTA	0	142	960	148.00	82	40.3	...	11.8	3.3	1.1	1.0	527.00	286.00	0.77	15.5	2
5	6	Clyde Drexler	28	POR	1	75	960	78.00	82	34.8	...	6.7	6.0	1.8	0.7	482.00	319.00	794.00	12.4	2
6	7	Kevin Johnson	24	PHO	0	32	960	33.00	77	36.0	...	3.5	10.1	2.1	0.1	516.00	205.00	843.00	12.7	0
7	8	Dominique Wilkins	31	ATL	0	29	960	0.03	81	38.0	...	9.0	3.3	1.5	0.8	0.47	341.00	829.00	11.4	1
8	9T	Larry Bird	34	BOS	0	25	960	26.00	60	38.0	...	8.5	7.2	1.8	1.0	454.00	389.00	891.00	6.6	0
9	9T	Terry Porter	27	POR	0	25	960	26.00	81	32.9	...	3.5	8.0	2.0	0.1	515.00	415.00	823.00	13.0	2
10	11	Patrick Ewing	28	NYK	0	20	960	21.00	81	38.3	...	11.2	3.0	1.0	3.2	514.00	0.00	745.00	10.0	1
11	12	John Stockton	28	UTA	0	15	960	16.00	82	37.8	...	2.9	14.2	2.9	0.2	507.00	345.00	836.00	14.0	2
12	13	Isiah Thomas	29	DET	0	11	960	11.00	48	34.5	...	3.3	9.3	1.6	0.2	435.00	292.00	782.00	3.4	9
13	14	Robert Parish	37	BOS	0	10	960	0.01	81	30.1	...	10.6	0.8	0.8	1.3	598.00	0.00	767.00	10.0	1
14	15	Joe Dumars	27	DET	0	8	960	8.00	80	38.1	...	2.3	5.5	1.1	0.1	481.00	311.00	0.89	9.9	1
459	1	Nikola Jokić	25	DEN	91	971	1010	961.00	72	34.6	...	10.8	8.3	1.3	0.7	566.00	388.00	868.00	15.6	3
460	2	Joel Embiid	26	PHI	1	586	1010	0.58	51	31.1	...	10.6	2.8	1.0	1.4	513.00	377.00	859.00	8.8	2
461	3	Stephen Curry	32	GSW	5	453	1010	449.00	63	34.2	...	5.5	5.8	1.2	0.1	482.00	421.00	916.00	9.0	2
462	4	Giannis Antetokounmpo	26	MIL	1	348	1010	345.00	61	33.0	...	11.0	5.9	1.2	1.2	569.00	303.00	685.00	10.2	2
463	5	Chris Paul	35	PHO	2	139	1010	138.00	70	31.4	...	4.5	8.9	1.4	0.3	499.00	395.00	934.00	9.2	2
464	6	Luka Don?i?	21	DAL	0	42	1010	42.00	66	34.3	...	8.0	8.6	1.0	0.5	479.00	0.35	0.73	7.7	1
465	7	Damian Lillard	30	POR	0	38	1010	38.00	67	35.8	...	4.2	7.5	0.9	0.3	451.00	391.00	928.00	10.4	2
466	8	Julius Randle	26	NYK	0	20	1010	0.02	71	37.6	...	10.2	6.0	0.9	0.3	456.00	411.00	811.00	7.8	0
467	9	Derrick Rose	32	TOT	1	10	1010	0.01	50	25.6	...	2.6	4.2	1.0	0.4	0.47	388.00	866.00	3.1	1
468	10	Rudy Gobert	28	UTA	0	8	1010	8.00	71	30.8	...	13.5	1.3	0.6	2.7	675.00	0.00	623.00	11.3	2
469	11	Russell Westbrook	32	WAS	0	5	1010	5.00	65	36.4	...	11.5	11.7	1.4	0.4	439.00	315.00	656.00	3.7	7
470	12	Ben Simmons	24	PHI	0	3	1010	3.00	58	32.4	...	7.2	6.9	1.6	0.6	557.00	0.30	613.00	6.0	1
471	13T	James Harden	31	TOT	0	1	1010	1.00	44	36.6	...	7.9	10.8	1.2	0.8	466.00	362.00	861.00	7.0	2
472	13T	LeBron James	36	LAL	0	1	1010	1.00	45	33.4	...	7.7	7.8	1.1	0.6	513.00	365.00	698.00	5.6	1
473	13T	Kawhi Leonard	29	LAC	0	1	1010	1.00	52	34.1	...	6.5	5.2	1.6	0.4	512.00	398.00	885.00	8.8	2

```
mvps_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 474 entries, 0 to 473
Data columns (total 21 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Rank     474 non-null    object
 1   Player   474 non-null    object
 2   Age      474 non-null    int64
 3   Tm       474 non-null    object
 4   First    474 non-null    int64
 5   Pts Won  474 non-null    int64
 6   Pts Max  474 non-null    int64
 7   Share    474 non-null    float64
 8   G        474 non-null    int64
 9   MP       474 non-null    float64
 10  PTS      474 non-null    float64
 11  TRB      474 non-null    float64
 12  AST      474 non-null    float64
 13  STL      474 non-null    float64
 14  BLK      474 non-null    float64
 15  FG%      474 non-null    float64
 16  3P%      474 non-null    float64
 17  FT%      474 non-null    float64
 18  WS       474 non-null    float64
 19  WS/48    474 non-null    float64
 20  Year     474 non-null    int64
dtypes: float64(12), int64(6), object(3)
memory usage: 77.9+ KB
```

Mengecek dan menangani missing value

```
print("Tidak ditemukan missing value pada dataset mvps")
cek_missing_values(mvps_df)
```

```
Tidak ditemukan missing value pada dataset mvps
```

	Total	Percent

Mengecek dan menangani duplikasi data

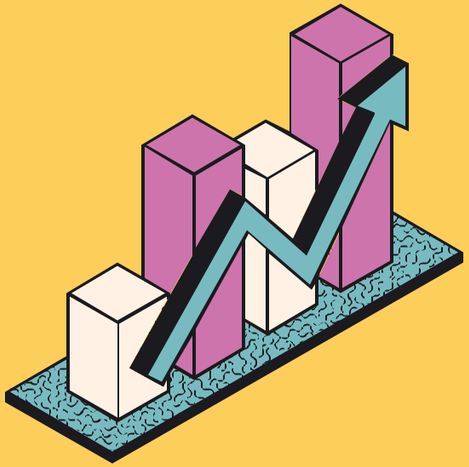
```
print("Jumlah duplikasi data pada dataset mvps:", mvps_df.duplicated().sum())
```

```
Jumlah duplikasi data pada dataset mvps: 0
```

EKSPLORASI

SEDERHANA

1A



Faktor Utama Seorang Pemain Menjadi MVP (secara statistik)

Mendefinisikan pemain dengan peringkat 1 adalah MVP pada tahun itu

```
mvps_df.loc[mvps_df["Rank"] == "1", "is_mvp"] = 1
mvps_df.loc[mvps_df["Rank"] != "1", "is_mvp"] = 0
display(mvps_df)
```

Mendapatkan semua fitur yang memiliki index korelasi >0.1 dengan MVP

```python
from scipy.stats import pearsonr

for i in lst_column :
    if i != "is_mvp" :
        corr, _ = pearsonr(mvps_and_teams[i], mvps_and_teams["is_mvp"])
        if corr > 0.1 :
            print(i + " : " + str(corr))
```

```
First : 0.9217981829015227
Pts Won : 0.7120673799647391
Share : 0.6708995401575344
PTS : 0.25120414400397706
WS : 0.39224615089855785
WS/48 : 0.2779454293078135
W : 0.2507622586487429
W/L% : 0.11227508768540638
GB : 0.15018439064370775
SRS : 0.2557466449068879
```

Didapatkan 10 fitur yang berkorelasi serta nilai korelasinya

Perbedaan MVP player dan non-MVP player

Pada gambar di bawah, dapat dilihat bahwa MVP player memiliki nilai yang baik pada seluruh attribut

Average of Non-MVP Indicator

Pts Won	First	Share	PTS	WS
138,71	2	107,16	21,97	10,57

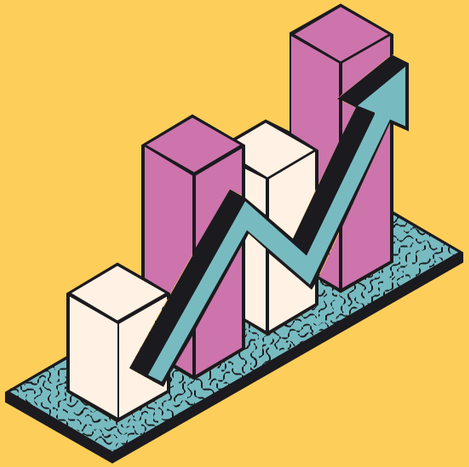
WS/48	W	W/L%	GB	SRS
172,03	50,36	536,5	484,85	3,8

Average of MVP Indicator

Pts Won	First	Share	PTS	WS
1.045,97	87,16	815,06	27,16	15,85

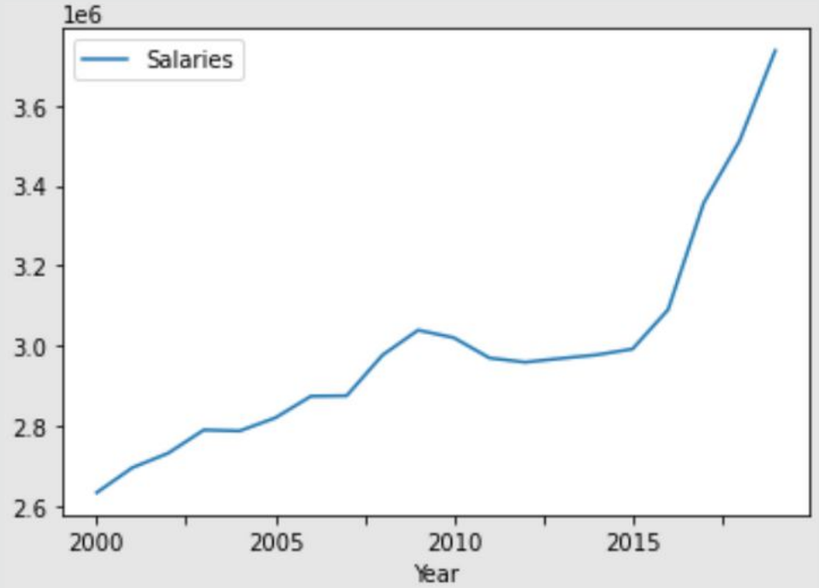
WS/48	W	W/L%	GB	SRS
254,11	59,97	649,42	1.041,5	7,01

1B



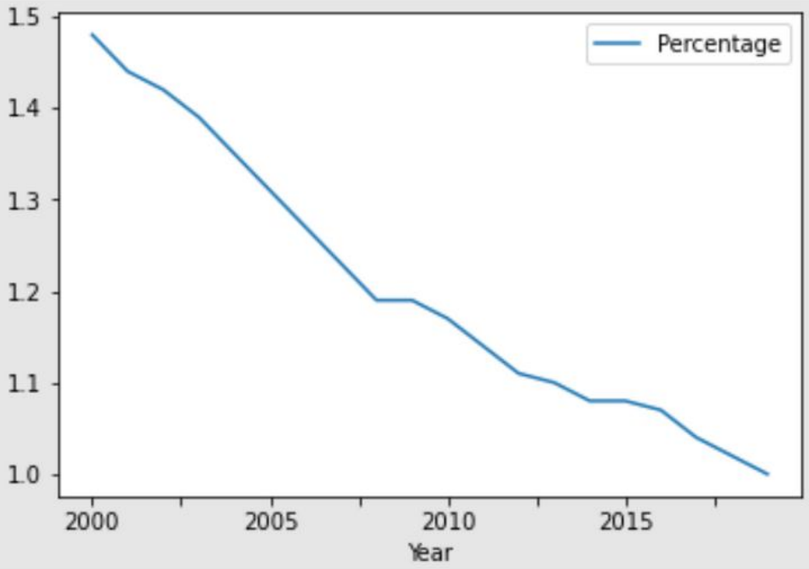
Kenaikan Rata-Rata Gaji Pemain dari tahun 2000-2019

```
df_salaries_year = salaries_df.groupby("Year")["Salaries"].mean().reset_index()
df_salaries_year = df_salaries_year.astype({"Year":'category'})
```



Terdapat perumbuhan gaji dari 2000 - 2019 (naik secara signifikan pada tahun 2015 - 2019). Namun, seperti yang kita ketahui nilai dari mata uang mengalami inflasi dari setiap tahunnya sehingga kita perlu membandingkan nilai pada tahun selain 2019 dengan tahun 2019.

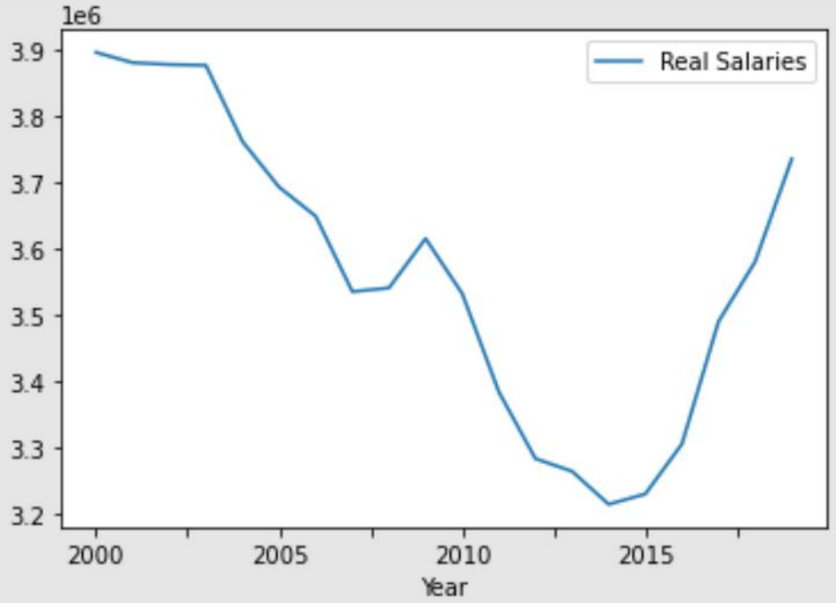
```
inflation_df = pd.read_excel("datasets/persentase_dolar_2000_2019.xlsx")
inflation_df = inflation_df.astype({"Year":'category'})
```



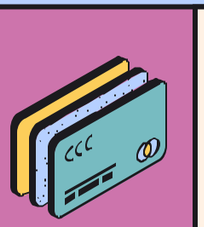
Terdapat penurunan nilai sehingga secara garis besar setaip tahun mengalami penurunan nilai mata uang (inflasi). Dengan demikian kita perlu mengetahui nilai asli salaries sesuai dengan tahun 2019.

Membandingkan nilai salaries yang sudah tidak terikat dengan inflasi

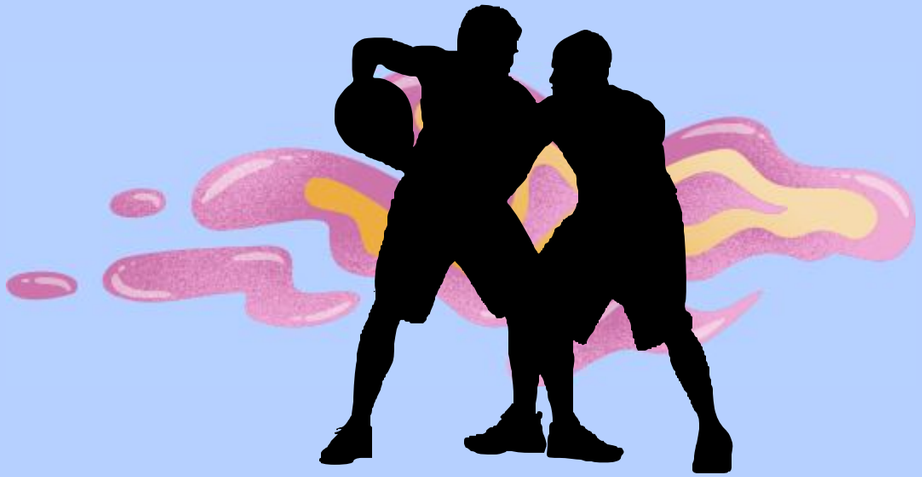
```
salaries_df_update = df_salaries_year.merge(inflation_df, on="Year").reset_index()
salaries_df_update["Real Salaries"] = salaries_df_update["Salaries"] * salaries_df_update["Percentage"]
```



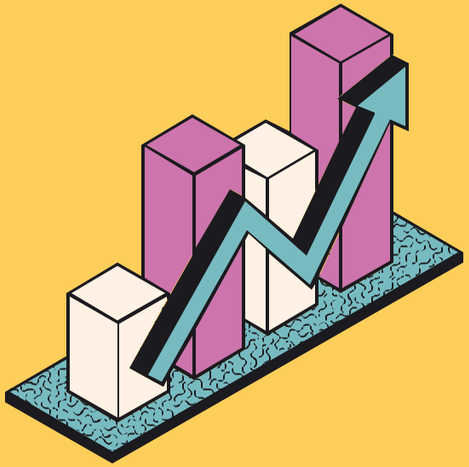
Rata-rata gaji pemain pada tahun 2019 lebih kecil dibanding tahun 2000. Lalu, terjadi penurunan rata-rata gaji pemain dari rentang tahun 2000 - 2014 dan rata-rata gaji pemain mulai kembali naik pada rentang 2014-2019.



Dengan demikian, walaupun secara nominal mata uang gaji pemain mengalami peningkatan pada tahun 2000-2019. Tetapi apabila berbicara mengenai nilai mata uang, dengan mempertimbangkan nilai inflasi yang terjadi terhadap US Dolar maka sejatinya nilai dari gaji yang diberikan mengalami penurunan



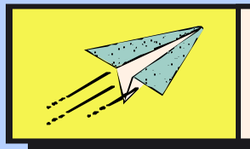
1C



Korelasi antar Kemenangan Suatu Tim dengan Suatu Variabel tertentu

Mengubah kolom-kolom pada dataset player menjadi numerik, kecuali kolom nama tim

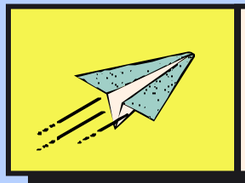
```
temp_player_df = players_df.copy()
temp_player_df = temp_player_df.drop(["Rk", "Player", "Pos"],axis=1)
lst_column_temp_player = temp_player_df.columns


for i in lst_column_temp_player :
    if i != "Tm" :
        temp_player_df[i] = pd.to_numeric(temp_player_df[i], errors='coerce')
```



Mencari rata-rata seluruh data dari setiap Tim pada setiap tahun

```
data_player_groupby_team_and_year = temp_player_df.groupby(['Tm','Year']).mean().reset_index()
display(data_player_groupby_team_and_year.head())
```

Tm	Year	Age	G	GS	MP	FG	FGA	FG%	3P	...	FT%	ORB
ATL	1991	27.357143	59.928571	29.285714	19.442857	3.200000	6.935714	381.340714	0.242857	...	599.428571	1.17142
ATL	1992	26.600000	54.600000	27.333333	21.120000	3.773333	8.126667	465.800000	0.333333	...	529.430000	1.32000
ATL	1993	27.058824	49.235294	24.117647	18.529412	2.976471	6.558824	313.574118	0.347059	...	421.231176	1.19411
ATL	1994	28.133333	54.933333	27.333333	20.113333	3.406667	7.460000	399.367333	0.246667	...	531.097333	1.23333
ATL	1995	28.777778	45.833333	22.777778	19.144444	2.827778	6.544444	328.299444	0.438889	...	537.312222	1.15000



Menggabungkan data rata-rata performa pemain tim dengan data performa tim secara keseluruhan

```
new_data = pd.merge(data_player_groupby_team_and_year, teams_df)
```

Team	Year	Age	G	GS	MP	FG	FGA	FG%	3P	3PA	3P%
Atlanta Hawks	2003	27.380952	39.380952	19.523810	16.942857	2.223810	5.290476	350.415238	0.314286	0.947619	153.492857
Atlanta Hawks	2004	26.695652	34.391304	17.826087	21.360870	3.017391	7.165217	352.464348	0.347826	1.169565	104.145217
Atlanta Hawks	2005	28.000000	43.200000	20.500000	19.405000	2.780000	6.280000	408.148500	0.310000	0.955000	208.009000
Atlanta Hawks	2006	24.133333	53.200000	27.333333	19.820000	2.833333	6.260000	398.900000	0.433333	1.166667	169.285333
Atlanta Hawks	2007	24.631579	44.473684	21.578947	19.447368	2.678947	6.136842	355.178947	0.352632	1.078947	182.907895
Atlanta Hawks	2008	25.125000	51.750000	25.625000	20.450000	2.906250	6.562500	377.923125	0.468750	1.293750	199.881250

Mencari korelasi pearson-r antara kemenangan dengan data-data lainnya

```
lst_column = new_data.columns
for i in lst_column :
    if i != "Team" and i != "Year" and i != "W":
        corr, _ = pearsonr(new_data[i], new_data["W"])
        if corr > 0.1 :
            print(i + " : " + str(corr))
```

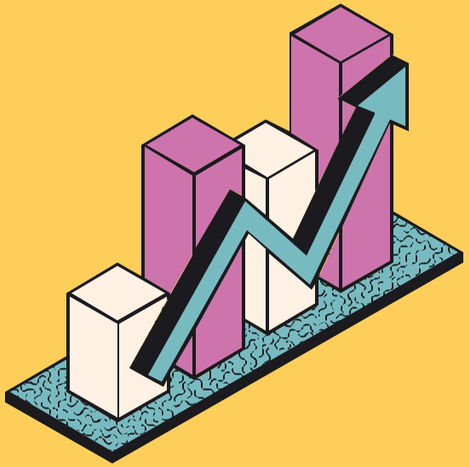
```
Age : 0.44280206964911945
G : 0.39962093994939296
GS : 0.3651750357548144
FG% : 0.12783300867285427
3P% : 0.12018030872938029
2P% : 0.10292221361507647
eFG% : 0.11459375970752983
W/L% : 0.592452417360374
PS/G : 0.3280766228384861
SRS : 0.932136748388367
```

Kami menyimpulkan bahwa variable yang dapat dijadikan acuan adalah:

- PS/G (passing per game)
- G (Game played)
- GS (Game started)
- FG% (Field Goal Percentage)
- 3P% (3 Point Field Goal Percentage)
- 2P% (2 Point Field Goal Percentage)
- eFG% (Effective Field Goal Percentage)

SRS tidak termasuk karena data tersebut memang dihitung dari data kemenangannya.

1D



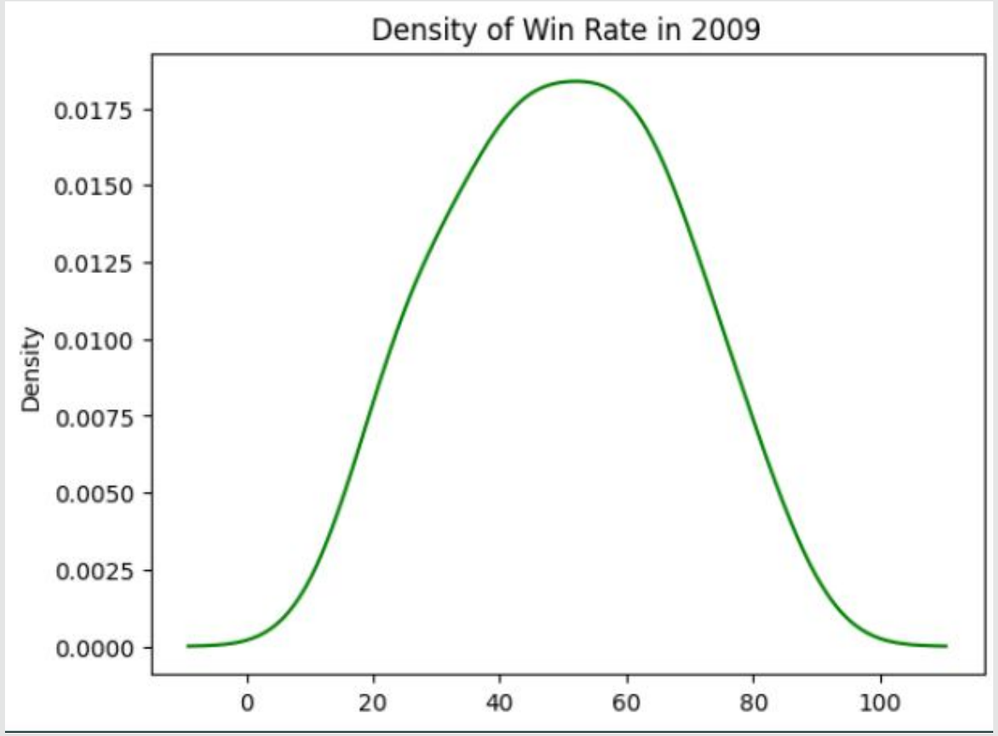
Tahun Paling Kompetitif untuk Liga NBA

Menghitung nilai skewness dari W/L% pada setiap tahun

```
from scipy.stats import skew

lst_year = new_data["Year"].unique()

lst_skew_score = []

for i in lst_year :
    temp_df = new_data.loc[new_data["Year"] == i]
    skew_score = skew(temp_df["W/L%"])
    lst_skew_score.append(skew_score)



for i in range(len(lst_skew_score)) :
    lst_skew_score[i] = abs(lst_skew_score[i])
min_score = min(lst_skew_score)
print(lst_skew_score)

idx = lst_skew_score.index(min_score)

print("Tahun paling kompetitif adalah " + str(lst_year[idx]))
```

Dengan melihat nilai skewness, apabila nilai skew adalah 0, maka suatu tahun memiliki distribusi W/L% yang merata. Dengan demikian, kami mengambil nilai skewness yang paling mendekati 0 untuk mendapatkan W/L% yang paling terdistribusi dengan baik

Didapatakan pada tahun 2009, nilai skewness paling mendekati 0
Berikut grafik W/L% pada tahun tersebut

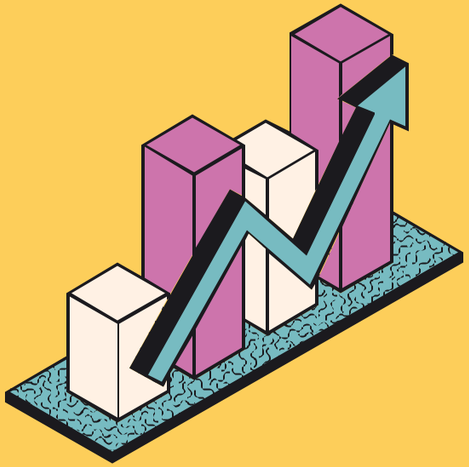


Berdasarakan eksplorasi tersebut, didapatkan bahwa 2009 merupakan tahun yang paling kompetitif. Hal ini didasarkan dari persebaran W/L%nya pada tahun tersebut

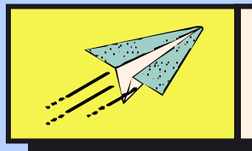
EKSPLORASI

SENDIRI

1E



Menentukan Golden Age Seorang Pemain



Mencari rata-rata tertinggi game score dari umur pemain

Menghitung rata-rata game score dari semua pemain

```
players_df["Avg GameScore"] = players_df["PTS"] + (0.4 * players_df["FG"]) - (0.7 * players_df["FGA"])
- (0.4*(players_df["FTA"] - players_df["FT"])) + (0.7 * players_df["ORB"]) + (0.3 * players_df["DRB"])
+ players_df["STL"] + (0.7* players_df["AST"]) + (0.7* players_df["BLK"]) - (0.4* players_df["PF"]) - players_df["TOV"]
```

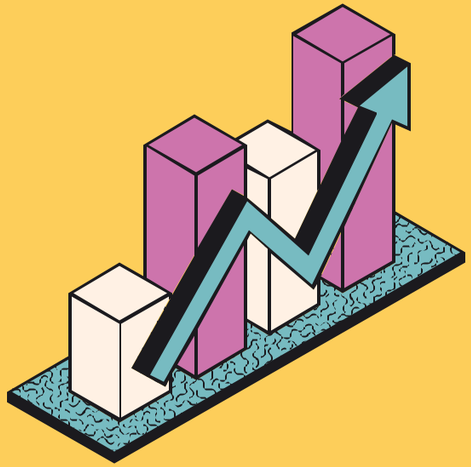
Mengambil umur pemain dengan rata-rata game score tertinggi

```
avg_age_and_gamescore = players_df.groupby("Age")["Avg GameScore"].mean().reset_index()

display(avg_age_and_gamescore[avg_age_and_gamescore["Avg GameScore"] == avg_age_and_gamescore["Avg GameScore"].max()])
```

	Age	Avg GameScore
10	28	6.945052

Golden age pemain basket ada pada umur 28

1F

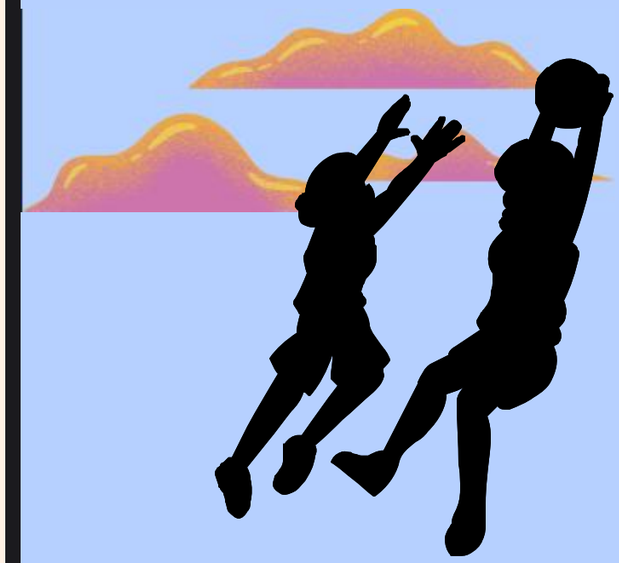


Mengetahui tingkat hubungan antara performa pemain (GameScore) pada suatu musim dengan gaji di musim berikutnya

Mencari korelasi gamescore dengan salaries di musim berikutnya

Membandingkan gamescore pada tahun 2018 dengan salaries di 2019

```
import seaborn as sns

temp_player_2018 = players_df.loc[players_df['Year'] == 2018]
temp_salaries_2019 = salaries_df.loc[salaries_df['Year'] == 2019]

temp_player_2018 = temp_player_2018.loc[:, ['Player', 'Avg GameScore']]
temp_salaries_2019 = temp_salaries_2019.loc[:, ['Name', 'Salaries']]

temp_salaries_2019.rename(columns={'Name' : 'Player'}, inplace=True)

data_2019 = pd.merge(temp_player_2018, temp_salaries_2019, on=["Player"])
data_2019 = data_2019.loc[:, ['Avg GameScore', 'Salaries']]
data_2019 = data_2019.sort_values(by='Salaries')
display(data_2019.corr())
```



Hasil Eksplorasi

Korelasi antara Gamescore 2018 dengan salaries 2019

	Avg GameScore	Salaries
Avg GameScore	1.000000	0.701726
Salaries	0.701726	1.000000

Dengan melihat hasil korelasi, yaitu 70% maka terdapat hubungan antara avg gamescore dan salaries.

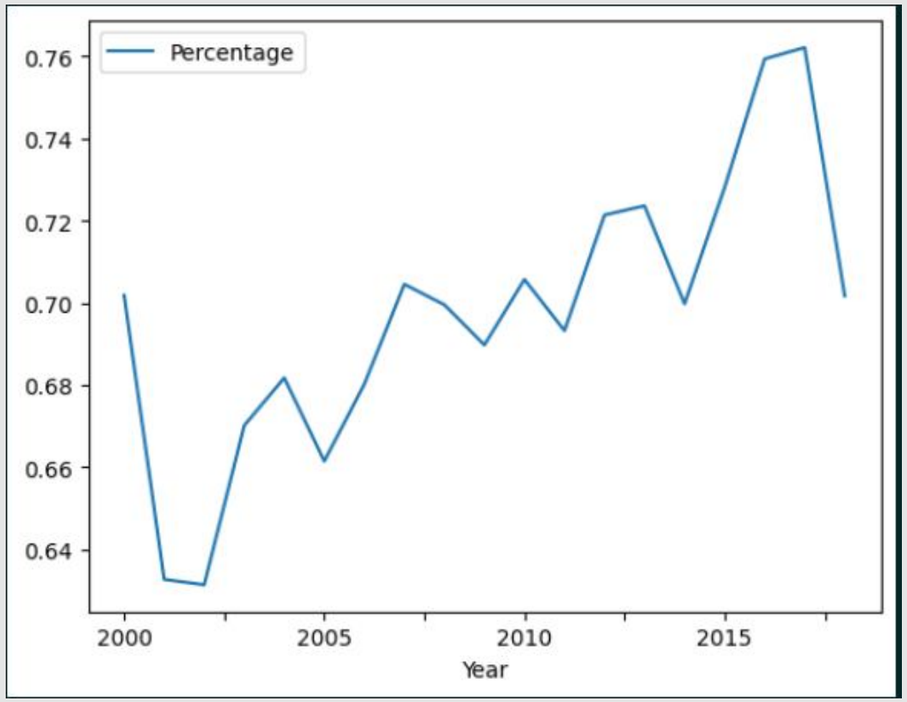
Korelasi antara Gamescore 2017 dengan salaries 2018

	Avg GameScore	Salaries
Avg GameScore	1.000000	0.762129
Salaries	0.762129	1.000000

Dengan melihat hasil korelasi, yaitu 76% maka terdapat hubungan antara avg gamescore pada 2017 dengan salaries di 2018.

Hasil Eksplorasi

Grafik Hubungan gamescore dengan salaires antara tahun 2000-2019

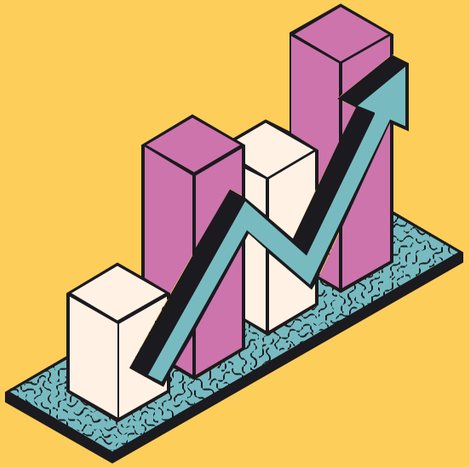


Dari grafik korelasi pada tahun 2000-2019 yang memiliki tingkat hubungan lebih dari 60%, maka terdapat pengaruh antara gamescore di musim tersebut dengan salaries di musim berikutnya

MEMBUAT

MODEL

2A



Model untuk Memprediksi Posisi Seorang Pemain

Diketahui terdapat lima posisi utama dalam permainan basket, yaitu Center (C), Power Forward (PF), Small Forward (SF), Point Guard (PG) dan Shooting Guard (SG). Dengan demikian, dalam pembuatan model kami memutuskan untuk membuang posisi selain kelima posisi tersebut. Hal- hal tersebut terdapat pada dalam dataset Player. Selain itu, berdasarkan jurnal yang kami dapatkan yang berjudul "BODY HEIGHT, BODY MASS, BODY MASS INDEX OF ELITE BASKETBALL PLAYERS IN RELATION TO THE PLAYING POSITION AND THEIR IMPORTANCE FOR SUCCESS IN THE GAME ", terdapat pengaruh tinggi dan berat badan pemain dalam menentukan posisi ideal pemain, sehingga kami menambahkan dataset baru yang berisikan data tersebut

Disclaimer : Dataset Player yang diambil hanya pada rentan 1996-2019 disebabkan pada data pendukung hanya terdapat sampai tahun 2019

Menggabungkan dataset player dengan data tinggi dan berat badan pemain

```
players_df = pd.merge(data_df, players_df, on=["Player", "Year"])

display(players_df)
```

	Player	Year	player_height	player_weight	Rk	Pos	Age	Tm	G	GS	...	FT%	ORB	DRB	TRB	AST	STL	BLK	TOV	PF	PTS
0	Dwayne Schintzius	1996	215.90	117.933920	342	C	27	IND	33	8	...	619	0.7	1.7	2.4	0.4	0.3	0.4	0.6	1.6	3.4
1	Ed O'Bannon	1996	203.20	100.697424	272	SG	23	NJN	64	29	...	713	1	1.6	2.6	1	0.7	0.2	1	1.5	6.2
2	Ed Pinckney	1996	205.74	108.862080	296	PF	32	TOT	74	47	...	0.76	2.6	3.6	6.2	1	0.9	0.4	1	2.1	6.5
3	Ed Pinckney	1996	205.74	108.862080	296	PF	32	TOR	47	24	...	758	2.4	3.6	6	1.1	0.7	0.4	1.1	2.1	7
4	Ed Pinckney	1996	205.74	108.862080	296	PF	32	PHI	27	23	...	764	2.7	3.8	6.5	0.8	1.2	0.4	0.9	2	5.6
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9782	Matthew Dellavedova	2019	190.50	90.718400	133	PG	28	MIL	12	0	...	1	0	0.8	0.8	2.4	0.2	0	0.9	0.8	1.7
9783	Matthew Dellavedova	2019	190.50	90.718400	133	PG	28	CLE	36	0	...	792	0.2	1.7	1.9	4.2	0.3	0.1	1.6	1.8	7.3
9784	Maurice Harkless	2019	200.66	99.790240	208	SF	25	POR	60	53	...	671	1.3	3.2	4.5	1.2	1.1	0.9	0.8	2.7	7.7
9785	Maxi Kleber	2019	208.28	108.862080	283	PF	27	DAL	71	18	...	784	1.3	3.4	4.6	1	0.5	1.1	0.8	2	6.8
9786	Meyers Leonard	2019	213.36	117.933920	302	C	26	POR	61	2	...	843	0.8	3	3.8	1.2	0.2	0.1	0.7	1.7	5.9

Melakukan drop fitur-fitur yang tidak diperlukan

```
players_df = players_df.drop(["Rk", "Player", "Tm", "Year"], axis=1)

display(players_df.columns)

lst_column = players_df.columns

for i in lst_column :
    if i != "Pos":
        players_df[i] = pd.to_numeric(players_df[i])

display(players_df.info())

lst_pos = players_df["Pos"].unique()
```

Preprocessing Dataset Player
Membuang posisi pemain selain PG, SG, PF, SF, dan C dan melakukan encoding

```
players_df = players_df.loc[(players_df["Pos"] == "SG") | (players_df["Pos"] == "PG") | (players_df["Pos"] == "SF") | (players_df["Pos"] == "PF")|
```

```
players_df["Pos"] = players_df["Pos"].replace(['PF', 'PG', 'C', 'SF', 'SG'], [0,1,2,3,4])
```

Pemisahan kolom-kolom menjadi feature (X) dan target (y)

```
X = players_df.drop('Pos', axis=1)
y = players_df['Pos'].to_frame()
```

Membagi data menjadi train, test, and test validation

```python
from sklearn.model_selection import train_test_split

# Membagi dataset menjadi training set dan test set
# dengan jumlah test set adalah sebanyak 20% dari data keseluruhan
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```python
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Membagi dataset train menjadi training set dan test set
# dengan jumlah test set adalah sebanyak 20% dari data train

X_train_temp, X_test_temp, y_train_temp, y_test_temp = train_test_split(X_train, y_train, test_size=0.2, random_state=42)
```

Melakukan GridCV untuk mendapatkan model terbaik di antara moodel-model clasiffier, yaitu RFC, LogisticRegression, SVC, MLPC, KNN, dan GaussianNB

```
# define the models and the parameter grids
models = [RandomForestClassifier(), LogisticRegression(),SVC(), MLPClassifier(), KNeighborsClassifier(), GaussianNB() ]
param_grids = [{
    'n_estimators': [10, 50, 100, 200],
    'max_depth': [None, 5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
},
                {'C': [0.1, 1, 10], 'penalty': ['l1', 'l2']},
                {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf']},
                {'hidden_layer_sizes': [(10,), (50,), (100,)], 'learning_rate_init': [0.1, 0.01, 0.001], 'activation' : ['relu', 'logistic']},
                {'n_neighbors': [1,2,3,4,5, 6, 7, 8, 9], 'metric': ['euclidean', 'manhattan']},
                {'var_smoothing': [1e-9, 1e-8, 1e-7, 1e-6, 1e-5, 1e-4]}]
```

Berikut hasil dari akurasi keenam model tersebut berikut dengan parameter terbaiknya

```
Model 0:
Best parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 200}
Best score: 0.7941794664510914
Model 1:
Best parameters: {'C': 10, 'penalty': 'l2'}
Best score: 0.7689571544058206
Model 2:
Best parameters: {'C': 10, 'kernel': 'linear'}
Best score: 0.7710590137429264
Model 3:
Best parameters: {'activation': 'logistic', 'hidden_layer_sizes': (10,), 'learning_rate_init': 0.01}
Best score: 0.770250606305578
Model 4:
Best parameters: {'metric': 'manhattan', 'n_neighbors': 9}
Best score: 0.6596604688763137
Model 5:
Best parameters: {'var_smoothing': 0.0001}
Best score: 0.6674211802748585
```

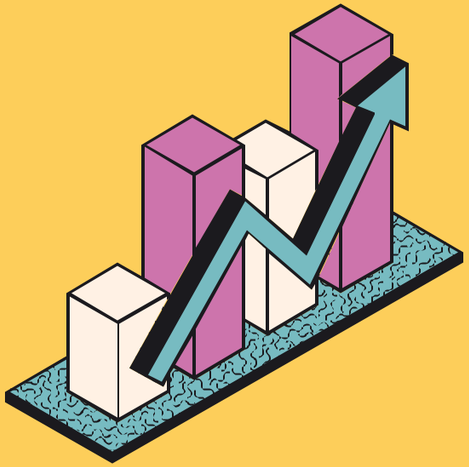
Diketahui, model terbaik yang didapatkan adalah model dengan menggunakan Random Forest Classification. Dengan demikian kita gunakan RFC untuk melakukan validasi data terakhir

Berikut mertric yang didapatkan dari hasil validation menggunakan model RFC yang didapatkan dari GridCV

```
Nilai rata-rata akurasi: 0.8009307135470527
F1 Macro Average: 0.8002120561607138
F1 Micro Average: 0.8009307135470526
Precision Macro Average: 0.8004874644204358
Precision Micro Average: 0.8009307135470527
Recall Macro Average: 0.8004429459228561
Recall Micro Average: 0.8009307135470527
```

Dari hasil ini didapatkan bahwa hasil akurasi pada validation tidak berbeda jauh dengan data testing., sehingga menandakan bahwa model kita merupakan model yang baik karena memiliki *error variance* yang rendah dan tidak overfit terhadap data testing non-final.

2B



Model untuk Memprediksi Gaji Pemain

Preprocessing Dataset Inflasi
memisahkan kolom Yearmon (string) menjadi 3 kolom: Year, Month, Date

```
us_inflation_2b_cleaned = cleaning_inflationDf_separateDateColumns(us_inflation_2b_cleaned)
```

	Yearmon	CPI
0	01-01-1913	9.8
1	01-02-1913	9.8
2	01-03-1913	9.8
3	01-04-1913	9.8
4	01-05-1913	9.7

⇨

	Yearmon	CPI	Year	Month	Day
0	01-01-1913	9.800	1913	1	1
1	01-02-1913	9.800	1913	2	1
2	01-03-1913	9.800	1913	3	1
3	01-04-1913	9.800	1913	4	1
4	01-05-1913	9.700	1913	5	1
...	...	...	...	...	...

Preprocessing Dataset Inflasi
menggabungkan row-row pada tahun yang sama menjadi satu row per tahun berdasarkan median

```
us_inflation_2b_cleaned = cleaning_inflationDf_getYearByMedian(us_inflation_2b_cleaned)
```

	Yearmon	CPI	Year	Month	Day
0	01-01-1913	9.800	1913	1	1
1	01-02-1913	9.800	1913	2	1
2	01-03-1913	9.800	1913	3	1
3	01-04-1913	9.800	1913	4	1
4	01-05-1913	9.700	1913	5	1
...	...	...	...	...	...

	Year	CPI
0	1913	9.85
1	1914	10.00
2	1915	10.10
3	1916	10.80
4	1917	12.90
5	1918	14.90

Pada kasus ini kami memutuskan mengambil median dari inflasi karena CPI merupakan perbandingan inflasi pada tahun ini dengan inflasi pada suatu tahun yang dijadikan pivot. Karena CPI merupakan ratio, maka kami merasa bahwa mengambil median lebih tepat dibandingkan mengambil mean-nya.

```
combined_df = us_inflation_2b_cleaned.merge(teams_2b, left_on=["Year"], right_on=["Year"])
combined_df = mvp_stats_2b.merge(combined_df, left_on=["Team", "Year"], right_on=["Team", "Year"])
combined_df = salaries_2b.merge(combined_df, left_on=["Name", "Year"], right_on=["Player", "Year"])
combined_df = cleaning_anyDf_removeDuplicatedColumns(combined_df)
```

Menggabungkan dataset mengenai inflasi, gaji, dan mvp_stats menjadi satu, lalu menghapus kolom duplikat. Kolom-kolom duplikat ini bisa muncul akibat adanya informasi yang sama pada file dataset yang berbeda.

Missing values pada target feature

Kami memutuskan untuk mendrop missing values pada target feature karena target feature merupakan ground truth. Oleh karena itu, kami tidak setuju apabila kita melakukan imputasi untuk missing values pada kolom ini. Apabila kita melakukan imputasi terhadap missing values untuk kolom ini, maka kolom ini tidak sepenuhnya menjadi ground truth lagi karena diisikan dengan data-data sintesis yang tidak diketahui kebenarannya.

```
print("Banyak missing values pada kolom Salaries: ", combined_df.Salaries.isna().sum())
combined_df = combined_df[combined_df.Salaries.notna()]
combined_df.isna().sum()
```

Membagi dataset menjadi X (feature) dan y (target)

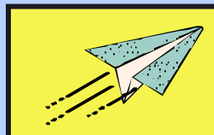
```
x_2b, y_2b = combined_df.drop("Salaries", axis=1), combined_df[["Salaries"]]
```

Drop feature yang tidak diperlukan

```
x_2b.drop("Team", axis=1, inplace=True)
x_2b.drop("Player", axis=1, inplace=True)
x_2b.drop("Salaries_Rank", axis=1, inplace=True)
```

Pada dataset ini, kita sudah memiliki gabungan seluruh feature mengenai statistik performa team dan statistik performa pemain. Oleh karena itu, nama tim dan nama pemain sudah tidak lagi berkaitan dengan gaji yang diperoleh. Nama tim sudah dapat digantikan dengan data mengenai performa tim, dan nama pemain dapat digantikan dengan data mengenai performa pemain.

Sementara itu, kita melakukan drop kolom Rank karena kolom tersebut merupakan Ranking gaji seorang pemain pada tahun itu. Oleh karena itu, kami merasa informasi ini tidak valid untuk dijadikan dasar dalam memprediksi gaji pemain.



Membagi dataset menjadi training, testing, dan final testing

```
x_2b_train, x2b_final_test, y_2b_train, y_2b_final_test = train_test_split(
    x_2b, y_2b, test_size=0.2, random_state=random_state
)
del x_2b
del y_2b

x_2b_train, x2b_test, y_2b_train, y_2b_test = train_test_split(
    x_2b_train, y_2b_train, test_size=0.2, random_state=random_state,
)
```

Kami memutuskan untuk membagi dataset menjadi training, testing, dan final testing. Dataset final testing diperlukan karena kami sangat disarankan untuk memasuki fase testing (final testing) hanya sekali saja, yakni ketika kami sudah benar-benar yakin dengan model yang kami buat. Oleh karena itu, selain final testing, kami juga membuat testing yang belum bersifat final.

```python
class OrderedMultiClassOneHotEncoder:
    def __init__(self, target_col="Pos",
                 base_classes=('C', 'PF', 'SF', 'PG', 'SG'),
                 separator="-"):
        self.target_col = target_col
        self.base_classes = base_classes
        self.separator = separator

    def fit(self, x, y=0):
        return self

    def _get_col_name(self, class_name):
        return f"{self.target_col}_{class_name}"

    def transform(self, x, y=0):
        column_values = x[self.target_col]
        x = x.drop(self.target_col, axis=1)
        for base_class in self.base_classes:
            x[self._get_col_name(base_class)] = 0

        for ind, val in enumerate(column_values):
            classes =  val.split(self.separator)
            weights = [i for i in range(len(classes), 0, -1)]  # len, len-1, len-2, ..., 2, 1
            sum_ = sum(weights)

            for class_, weight in zip(classes, weights):
                class_ = self._get_col_name(class_)
                col = x.columns.get_loc(class_)
                x.iloc[ind, col] = weight / sum_
```

```python
label_encoder_params = [{
    'encoding': [
        OrderedMultiClassOneHotEncoder("Pos", ['C', 'PF', 'SF', 'PG', 'SG'], "-")
    ]
}]
```

Kami membuat algoritma encoding yang mirip dengan one-hot untuk melakukan encode kolom Pos. Apabila Pos='C', hasil encoding akan bernilai 1 pada kolom C dan 0 pada kolom lainnya. Apabila Pos='SF-SG', maka kolom SF akan bernilai 2/3, kolom SG bernilai 1/3, dan kolom lainnya bernilai 0.

Sebelumnya kami juga sudah mencoba membandingkan tingkat akurasinya ketika menggunakan label encoding. Modifikasi one-hot encoding ini memberikan performa yang lebih baik dibandingkan label encoding

```python
scaler1 = preprocessing.MinMaxScaler()
scaler2 = preprocessing.StandardScaler()
scaler3 = preprocessing.RobustScaler()

scaler_params = [{
    'scaler': [None, scaler1, scaler2, scaler3],
}]
```

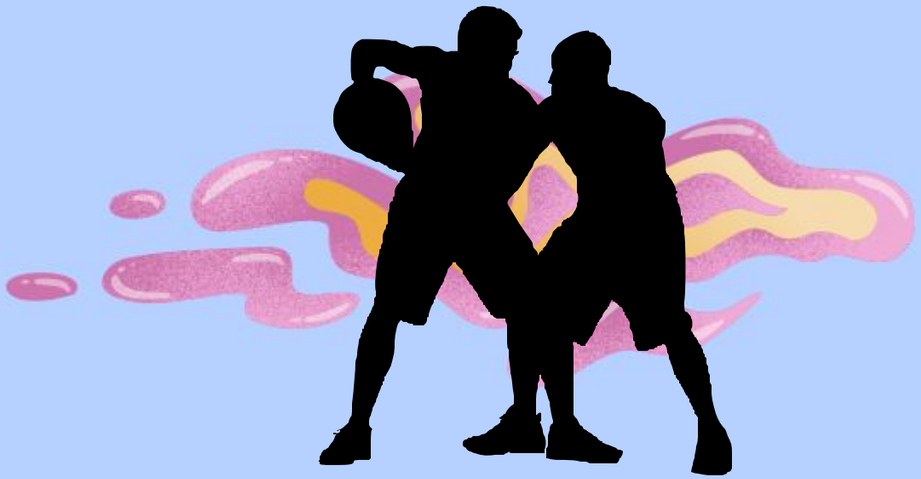
Kami memutuskan untuk mencoba 3 scaler berbeda untuk dimasukkan ke dalam GridSearchCV: MinMax, Standard, Robust. Selain itu, kami juga mencoba mengukur performa model tanpa scaling.

```python
imputer1 = SimpleImputer()
imputer2 = KNNImputer()

imputer_params = [{
    'imputer': [imputer2, imputer1],
}]
```

Untuk menangani missing values pada data, kami memilih SimpleImputer dan KNNImputer untuk dicoba menggunakan GridSearchCV

```python
feature_selector_params = [{
    'feature_selector': [None]
}, {
    'feature_selector': [SelectFromModel(estimator=Lasso())],
    'feature_selector__estimator__alpha': [0.01]
}, {
    'feature_selector': [SelectKBest()],
    'feature_selector__k': [6, 7],
}, {
    'feature_selector': [PCA()],
    'feature_selector__n_components': ['mle']
}]
```

Untuk feature reduction, kami memilih ANOVA, Lasso, dan PCA untuk dicoba oleh GridSearchCV



```
predictor_params = [{
    'predictor': [
        RandomForestRegressor(random_state=random_state),
    ],
    'predictor__min_samples_split': [5,],
}, {
    'predictor': [
        GradientBoostingRegressor(random_state=random_state),
    ],
    'predictor__n_estimators': [100, 110, 120, 130],
}]
```

Kami telah mencoba berbagai macam model, beberapa di antaranya adalah LinearRegression, SVR, Neural Network, dan KNN. Hasilnya, Gradient Boosting dan Random Forest merupakan 2 model dengan akurasi yang terbaik.

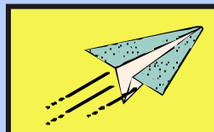
Deklarasi Pipeline dan GridSearchCV

```
pipeline = Pipeline(
    steps=[
        ('encoding', None),
        ('df_captor', DataFrameCaptor()),
        ('scaler', None),
        ('imputer', None),
        ('feature_selector', None),
        ('predictor', None),
    ]
)
grid_search_cv = GridSearchCV(
    estimator=pipeline,
    param_grid=grid_params,
    cv=5,
    scoring='neg_mean_squared_error',
    n_jobs=-2,
    verbose=10,
    error_score='raise',
)

grid_search_cv.fit(x_2b_train, y_2b_train.Salaries)
```

```python
y_predict = best_pipeline.predict(x_test)
mae = metrics.mean_absolute_error(y_test, y_predict)

r2 = metrics.r2_score(y_test, y_predict)
salaries_mean = y_2b_test.Salaries.mean()

print("R2 score: ", r2)
print("Rata-rata error absolut: ", mae)
print("Rata-rata error relatif: ", mae / salaries_mean)
```

```
R2 score:  0.644720852169788
Rata-rata error absolut:  2071723.280294468
Rata-rata error relatif:  0.44974900918922966
```

Untuk testing yang bersifat non-final, model terbaik pada GridSearchCV menghasilkan akurasi dengan R2 score sebesar 0.64472.

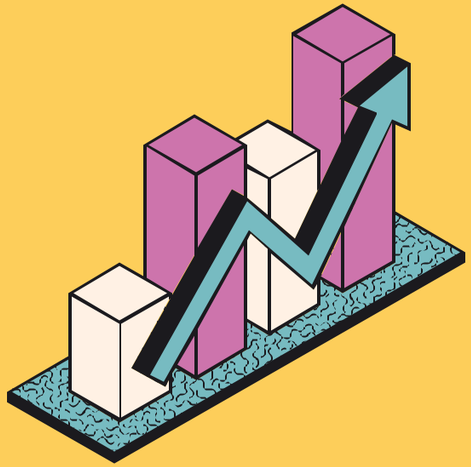


Akurasi Model pada Final Testing

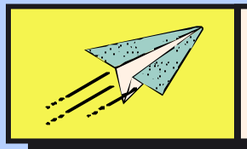
```
R2 score:  0.648963927785881
Rata-rata error absolut:  2078200.1344884185
Rata-rata error relatif:  0.4511550650964541
```

Setelah yakin dengan model yang kami buat, kami memasuki tahap final-testing. Hasilnya didapatkan bahwa akurasi pada final-testing ini tidak berbeda jauh dengan testing non-final, yakni R2 score bernilai 0.64896. Hal ini menandakan bahwa model kami merupakan model yang cukup baik karena memiliki *error variance* yang rendah.

2C



Cluster dari Dataset & Penjelasan Karakteristik dari Cluster yang Terbentuk



Dataframe pw_avg_gamescore

```
display(pw_avg_gamescore_df.columns)
display(pw_avg_gamescore_df.head())
```

✓ 0.4s

```
Index(['Rk', 'Player', 'Pos', 'Age', 'Tm', 'G', 'GS', 'MP', 'FG', 'FGA', 'FG%',
       '3P', '3PA', '3P%', '2P', '2PA', '2P%', 'eFG%', 'FT', 'FTA', 'FT%',
       'ORB', 'DRB', 'TRB', 'AST', 'STL', 'BLK', 'TOV', 'PF', 'PTS', 'Year',
       'Avg GameScore'],
      dtype='object')
```

	Rk	Player	Pos	Age	Tm	G	GS	MP	FG	FGA	...	DRB	TRB	AST	STL	BLK	TOV	PF	PTS	Year	Avg GameScore
0	1	Alaa Abdelnaby	PF	22	POR	43	0	6.7	1.3	2.7	...	1.4	2.1	0.3	0.1	0.3	0.5	0.9	3.1	1991	2.07
1	2	Mahmoud Abdul-Rauf	PG	21	DEN	67	19	22.5	6.2	15.1	...	1.3	1.8	3.1	0.8	0.1	1.6	2.2	14.1	1991	7.23
2	3	Mark Acres	C	28	ORL	68	0	19.3	1.6	3.1	...	3.2	5.3	0.4	0.4	0.4	0.6	3.2	4.2	1991	3.98
3	4	Michael Adams	PG	28	DEN	66	66	35.5	8.5	21.5	...	3.0	3.9	10.5	2.2	0.1	3.6	2.5	26.5	1991	21.00
4	5	Mark Aguirre	SF	31	DET	78	13	25.7	5.4	11.7	...	3.1	4.8	1.8	0.6	0.3	1.6	2.7	14.2	1991	9.28

5 rows × 32 columns

Dataframe final_salaries_df

```
display(final_salaries_df.columns)
display(final_salaries_df.head())
```

✓ 0.3s

```
Index(['Name', 'Year', 'Salaries', 'Rank'], dtype='object')
```

	Name	Year	Salaries	Rank
0	Shaquille O'Neal	2000	25370160.0	1
1	Kevin Garnett	2000	24872880.0	2
2	Alonzo Mourning	2000	22205920.0	3
3	Juwan Howard	2000	22200000.0	4
4	Scottie Pippen	2000	21896600.0	5

Mengambil player pada tahun 2000-2019 & Menggabungkan Dataframe

```
# ambil player dari tahun 2000-2019
pw_avg_gamescore_df = pw_avg_gamescore_df.loc[(pw_avg_gamescore_df['Year'] >= 2000) & (pw_avg_gamescore_df['Year'] <= 2019)]

# merge player_with_avg_gamescore dan final_salaries
combined_data_df = pw_avg_gamescore_df.merge(final_salaries_df, left_on=["Player", "Year"],
                                             right_on=["Name", "Year"])
combined_data_df
```

	Rk	Player	Pos	Age	Tm	G	GS	MP	FG	FGA	...	STL	BLK	TOV	PF	PTS	Year	Avg GameScore	Name	Salaries	Rank
0	1	Tariq Abdul-Wahad	SG	25	TOT	61	56	25.9	4.5	10.6	...	1.0	0.5	1.7	2.4	11.4	2000	7.39	Tariq Abdul-Wahad	3700000.0	195
1	1	Tariq Abdul-Wahad	SG	25	ORL	46	46	26.2	4.8	11.2	...	1.2	0.3	1.9	2.5	12.2	2000	7.83	Tariq Abdul-Wahad	3700000.0	195
2	1	Tariq Abdul-Wahad	SG	25	DEN	15	10	24.9	3.4	8.7	...	0.4	0.8	1.3	2.1	8.9	2000	5.59	Tariq Abdul-Wahad	3700000.0	195
3	2	Shareef Abdur-Rahim	SF	23	VAN	82	82	39.3	7.2	15.6	...	1.1	1.1	3.0	3.0	20.3	2000	15.83	Shareef Abdur-Rahim	3700000.0	157
4	3	Cory Alexander	PG	26	DEN	29	2	11.3	1.0	3.4	...	0.8	0.1	1.0	1.3	2.8	2000	2.06	Cory Alexander	3700000.0	283
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
10310	528	Tyler Zeller	C	29	ATL	2	0	5.5	0.0	1.0	...	0.0	0.0	0.0	2.0	0.0	2019	0.15	Tyler Zeller	1933941.0	339
10311	528	Tyler Zeller	C	29	MEM	4	1	20.5	4.0	7.0	...	0.3	0.8	1.0	4.0	11.5	2019	8.92	Tyler Zeller	1933941.0	339
10312	530	Ivica Zubac	C	21	TOT	59	37	17.6	3.6	6.4	...	0.2	0.9	1.2	2.3	8.9	2019	7.77	Ivica Zubac	1544951.0	389
10313	530	Ivica Zubac	C	21	LAL	33	12	15.6	3.4	5.8	...	0.1	0.8	1.0	2.2	8.5	2019	7.13	Ivica Zubac	1544951.0	389
10314	530	Ivica Zubac	C	21	LAC	26	25	20.2	3.8	7.2	...	0.4	0.9	1.4	2.5	9.4	2019	8.52	Ivica Zubac	1544951.0	389

10315 rows × 35 columns

Normalisasi menggunakan MinMaxScaler

```
df_copy = combined_data_df.drop("Player", axis=1)
df_copy = df_copy.drop("Name", axis=1)
df_copy = df_copy.drop("Tm", axis=1)
df_copy = df_copy.drop("Pos", axis=1)
df_copy = df_copy.drop("Age", axis=1)
df_copy = df_copy.drop("Year", axis=1)

scaler = MinMaxScaler()
scaler_transform = scaler.fit_transform(df_copy)

normalized_df = pd.DataFrame(scaler_transform, columns = df_copy.columns)
normalized_df["Pos"] = combined_data_df["Pos"]
normalized_df["Age"] = combined_data_df["Age"]
normalized_df["Year"] = combined_data_df["Year"]
normalized_df
```

Melakukan PCA setelah Normalisasi untuk mengurangi dimensi dari suatu data dengan cara menemukan kombinasi terbaik dari fitur-fitur yang ada.

In [150]:
```
array_pos = normalized_df['Pos'].unique()

le = preprocessing.LabelEncoder()
le.fit(array_pos)

le.classes_

normalized_df.Pos = le.transform(normalized_df['Pos'])
normalized_df['Pos']
```

```
0        11
1        11
2        11
3         8
4         5
        ..
10310     0
10311     0
10312     0
10313     0
10314     0
Name: Pos, Length: 10315, dtype: int32
```

Memilih n_components=2

```
In [151]: pca = PCA(n_components=2)

          pca.fit(normalized_df)
          trans_pca = pca.transform(normalized_df)
          trans_pca.shape

          np.cumsum((pca.explained_variance_ratio_))
```

```
array([0.48624091, 0.75629203])
```

Didapatkan hasil Silhouette Score setelah menggunakan PCA

```
Saat K = 2 , silhouette scorenya adalah 0.420060303004141
Saat K = 3 , silhouette scorenya adalah 0.3925270927132345
Saat K = 4 , silhouette scorenya adalah 0.3706422366870174
Saat K = 5 , silhouette scorenya adalah 0.3690310638667751
Saat K = 6 , silhouette scorenya adalah 0.3572247123721965
Saat K = 7 , silhouette scorenya adalah 0.34687505811632235
Saat K = 8 , silhouette scorenya adalah 0.34354004410731026
Saat K = 9 , silhouette scorenya adalah 0.3419346989810902
Setelah dilakukan silhoutte analysis, nilai K yang paling optimal yaitu K = 2
```

Mencoba clustering tanpa melakukan PCA

Mengambil column yang kami nilai memiliki korelasi untuk melakukan clustering
Kami mengambil kolumn "PTS" dan "FG" dalam mencari nilai K dan Silhouette Score

```
attribute_choose = normalized_df[["PTS", "FG"]]
attribute1 = normalized_df["PTS"]
attribute2 = normalized_df["FG"]
new_attribute = ["PTS", "FG"]
```

1) K-Means Clustering

Kami melakukan clustering dengan menggunakan K-Means karena lebih cepat dan memiliki hasil yang homogen

```
k = [2, 3, 4, 5, 6, 7, 8, 9]
silhoutte_scr = []
k_scr, max_scr = 0, 0

for i in k:
    kmeans = KMeans(n_clusters=i, random_state=42).fit(attribute_choose)
    labels = kmeans.labels_
    silhoutte_scr = silhouette_score(attribute_choose, labels)

    # menyimpan silhouette score tertinggi
    if silhoutte_scr>max_scr:
        max_scr = silhoutte_scr
        k_scr = i
    print("Saat K =",i, ", silhouette scorenya adalah", silhoutte_scr)
print("Setelah dilakukan silhoutte analysis, nilai K yang paling optimal yaitu K =", k_scr)
```

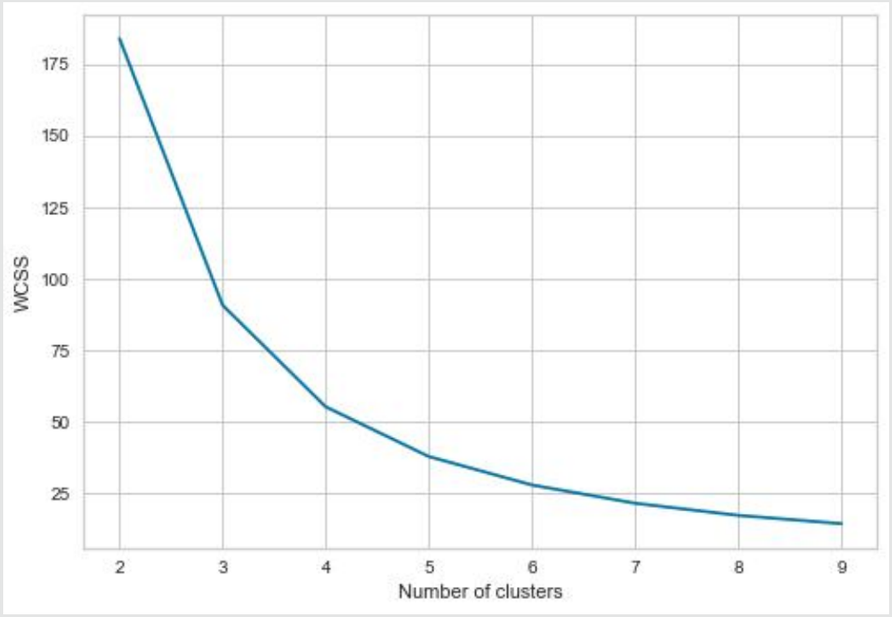
```
Saat K = 2 , silhouette scorenya adalah 0.6258189671275907
Saat K = 3 , silhouette scorenya adalah 0.5725823706005236
Saat K = 4 , silhouette scorenya adalah 0.542732610463241
Saat K = 5 , silhouette scorenya adalah 0.5202813150940414
Saat K = 6 , silhouette scorenya adalah 0.5124256318861243
Saat K = 7 , silhouette scorenya adalah 0.49699376226393016
Saat K = 8 , silhouette scorenya adalah 0.4934959688076004
Saat K = 9 , silhouette scorenya adalah 0.4758358338887536
Setelah dilakukan silhoutte analysis, nilai K yang paling optimal yaitu K = 2
```

Silhoutte score tanpa melakukan PCA memiliki nilai yang lebih tinggi.

2) Elbow Method

Selain menggunakan Metrik Silhoutte Coefficient, kami menggunakan Elbow Method untuk memastikan didapat K yang optimal

```
# List untuk menyimpan within-cluster sums of square untuk setiap jumlah cluster
wcss = []

for i in k:
    kmeans = KMeans(n_clusters=i, random_state=42).fit(attribute_choose)
    wcss.append(kmeans.inertia_)
plt.plot(k, wcss)
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

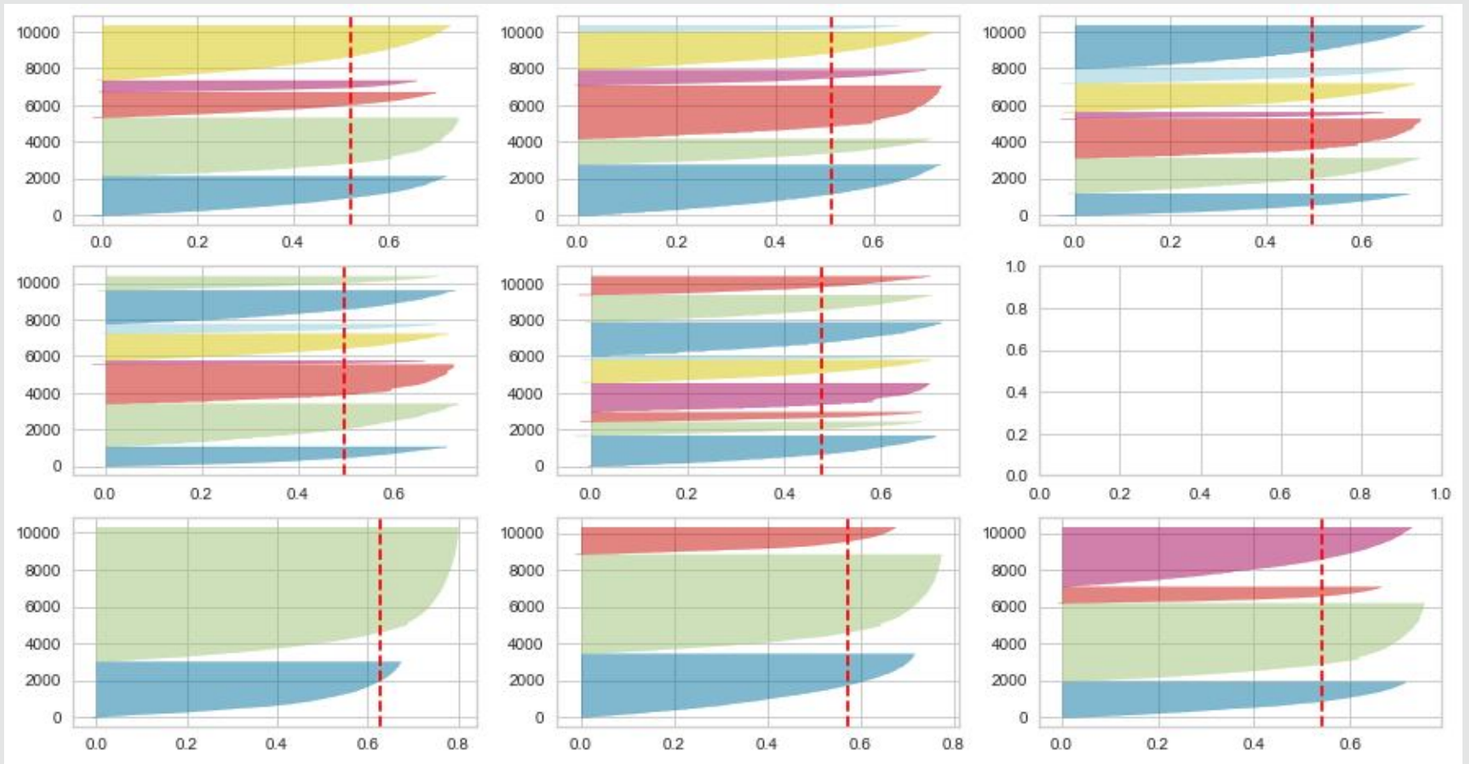


Hasil Elbow Method menunjukan jumlah cluster yang optimal berada pada K=3

3) Silhouette Visualizer

Silhouette Visualizer mempermudah kami untuk melihat secara visual titik data yang tergabung dalam cluster yang sesuai atau tidak sesuai

```
fig, ax = plt.subplots(3, 3, figsize=(15,8))
for i in k:
    '''
    Create KMeans instance for different number of clusters
    '''
    km = KMeans(n_clusters=i, init='k-means++', n_init=10, max_iter=100, random_state=42)
    q, mod = divmod(i-2, 3)
    '''
    Create SilhouetteVisualizer instance with KMeans instance
    Fit the visualizer
    '''
    visualizer = SilhouetteVisualizer(km, colors='yellowbrick', ax=ax[q-1][mod])
    visualizer.fit(attribute_choose)
```



Saat k=2 dan k=3 terlihat bahwa ukuran cluster berbeda (cluster dapat dibedakan) dan terpisah dengan baik

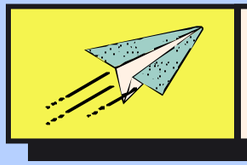
4) Silhouette Method

Selain menggunakan ketiga cara di atas, kami juga menggunakan Silhouette Method.

```
sil = []

for i in k:
  kmeans = KMeans(n_clusters=i, random_state=42).fit(attribute_choose)
  labels = kmeans.labels_
  score = silhouette_score(attribute_choose, labels, metric='euclidean')
  sil.append(score)
best_k = sil.index(max(sil)) + 2
print("Setelah diterapkan Silhouette Method, nilai K yang paling optimal yaitu K=", best_k)
```

```
Setelah diterapkan Silhouette Method, nilai K yang paling optimal yaitu K= 2
```

KESIMPULAN
Dari keempat cara (Metrik Silhoutte Coefficient, Elbow Method, Silhouette Visualizer, dan Silhouette Method), kami memilih K=2.



Menerapkan K-Means Clustering dengan K=2

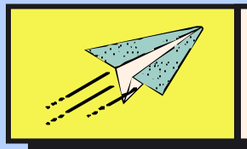
```
count_kmeans = KMeans(n_clusters=2, random_state=42)
count_kmeans.fit(attribute_choose)
predict_kmeans = count_kmeans.predict(attribute_choose)
```

```
# tampilkan attribute_choose dan cluster
combined_data_df3['cluster'] = predict_kmeans
combined_data_df3
```

Hasil clustering

	FG	PTS	cluster
0	4.5	11.4	0
1	4.8	12.2	0
2	3.4	8.9	1
3	7.2	20.3	0
4	1.0	2.8	1
...	...	...	...
10310	0.0	0.0	1
10311	4.0	11.5	0
10312	3.6	8.9	1
10313	3.4	8.5	1
10314	3.8	9.4	1

10315 rows × 3 columns



Membubuhkan hasil cluster kedalam DataFrame asli

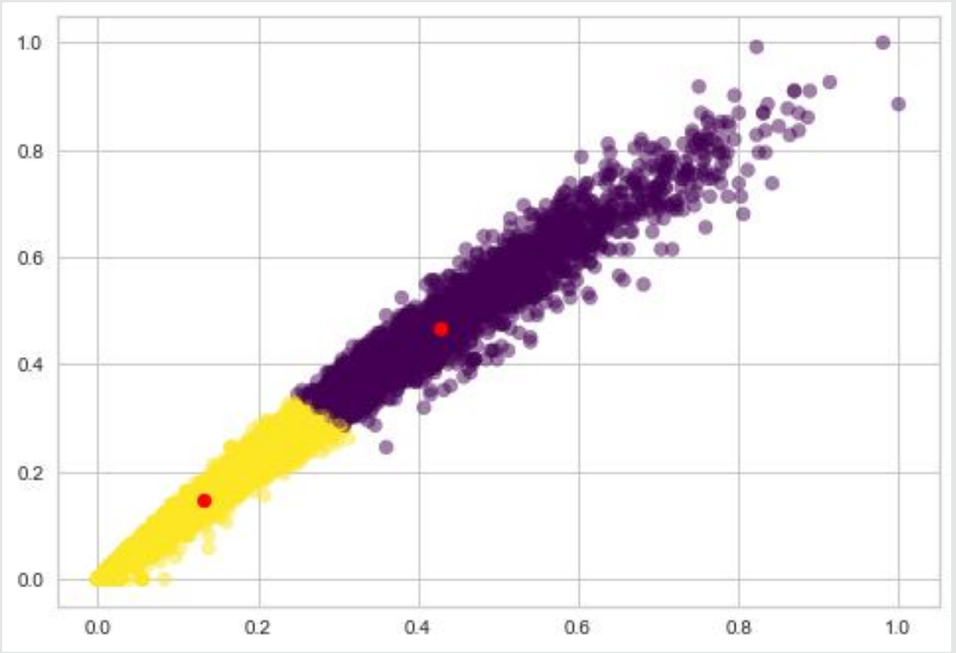
```
df_copy['cluster'] = combined_data_df3['cluster'].values
df_copy
```

	Rk	G	GS	MP	FG	FGA	FG%	3P	3PA	3P%	...	AST	STL	BLK	TOV	PF	PTS	Avg GameScore	Salaries	Rank	cluster
0	1	61	56	25.9	4.5	10.6	424.00	0.0	0.4	0.13	...	1.6	1.0	0.5	1.7	2.4	11.4	7.39	3700000.0	195	0
1	1	46	46	26.2	4.8	11.2	433.00	0.0	0.5	95.00	...	1.6	1.2	0.3	1.9	2.5	12.2	7.83	3700000.0	195	0
2	1	15	10	24.9	3.4	8.7	389.00	0.1	0.1	0.50	...	1.7	0.4	0.8	1.3	2.1	8.9	5.59	3700000.0	195	1
3	2	82	82	39.3	7.2	15.6	465.00	0.4	1.2	302.00	...	3.3	1.1	1.1	3.0	3.0	20.3	15.83	3700000.0	157	0
4	3	29	2	11.3	1.0	3.4	286.00	0.3	1.2	257.00	...	2.0	0.8	0.1	1.0	1.3	2.8	2.06	3700000.0	283	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
10310	528	2	0	5.5	0.0	1.0	0.00	0.0	0.5	0.00	...	0.5	0.0	0.0	0.0	2.0	0.0	0.15	1933941.0	339	1
10311	528	4	1	20.5	4.0	7.0	571.00	0.0	0.0	0.00	...	0.8	0.3	0.8	1.0	4.0	11.5	8.92	1933941.0	339	0
10312	530	59	37	17.6	3.6	6.4	559.00	0.0	0.0	0.00	...	1.1	0.2	0.9	1.2	2.3	8.9	7.77	1544951.0	389	1
10313	530	33	12	15.6	3.4	5.8	0.58	0.0	0.0	0.00	...	0.8	0.1	0.8	1.0	2.2	8.5	7.13	1544951.0	389	1
10314	530	26	25	20.2	3.8	7.2	538.00	0.0	0.0	0.00	...	1.5	0.4	0.9	1.4	2.5	9.4	8.52	1544951.0	389	1

10315 rows × 30 columns

Visualisasi Cluster

```
kmeans = KMeans(n_clusters=2, random_state=42).fit(normalized_df.loc[:,new_attribute])
centroids = kmeans.cluster_centers_

plt.scatter(attribute1, attribute2, c= kmeans.labels_.astype(float), s=50, alpha=0.5, cmap='viridis')
plt.scatter(centroids[:, 0], centroids[:, 1], c='red', s=50)
plt.show()
```

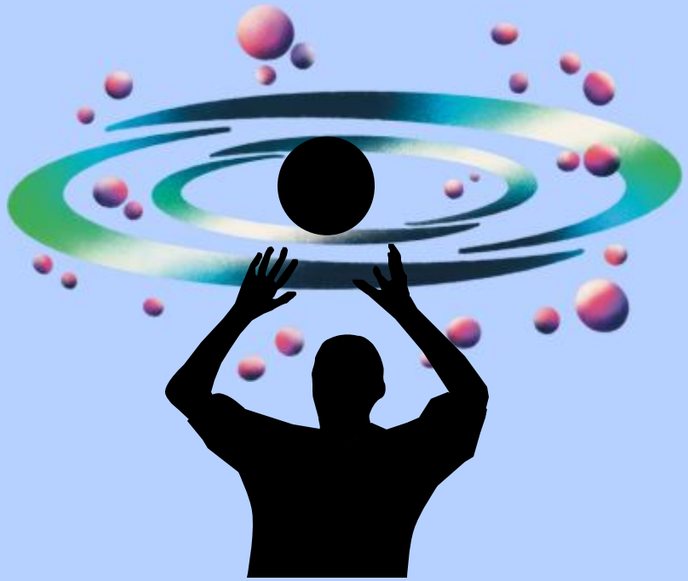


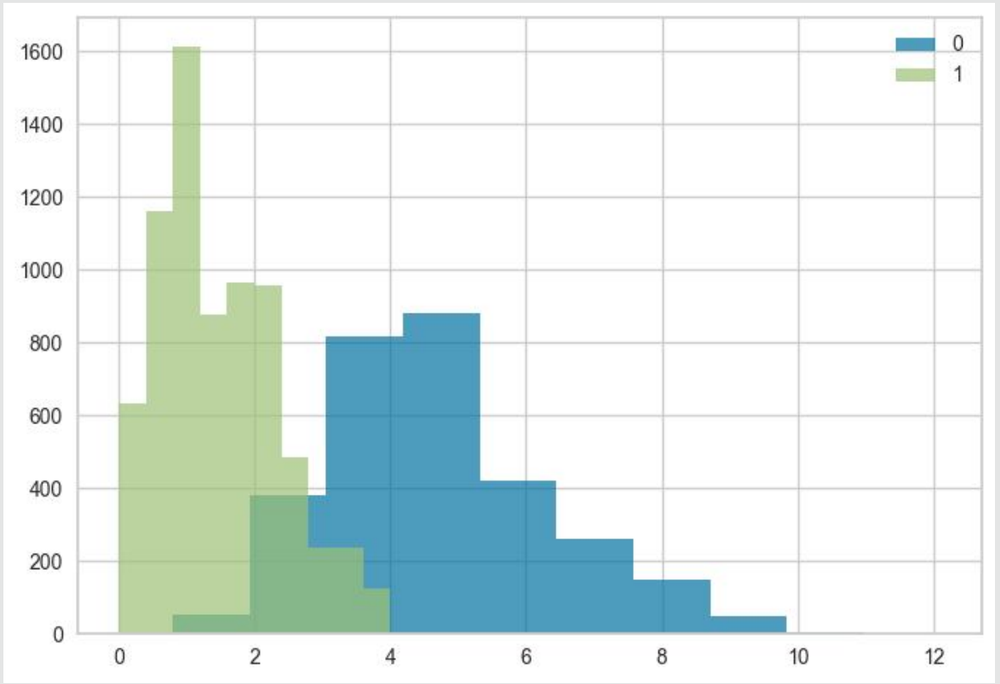
Interpretasi Cluster

```
keep_cluster, keep_cluster2 = [], []

# menyimpan pengelompokkan cluster 0 dan 1
for i in range(len(predict_kmeans)) :
    if predict_kmeans[i] == 0 :
        keep_cluster.append(i)
    else:
        keep_cluster2.append(i)

# untuk cluster 0 dan 1
cluster_zero, cluster_one = combined_data_df.iloc[keep_cluster], combined_data_df.iloc[keep_cluster2]
```

Interpretasi hasil cluster berdasarkan 2P

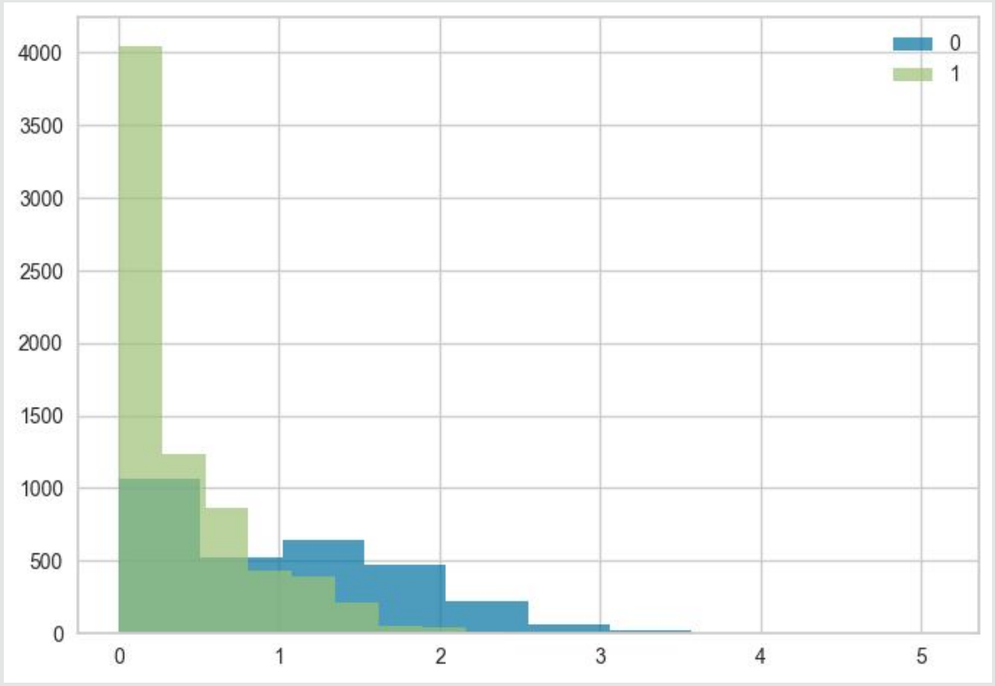
```
df_copy[df_copy['cluster'] == 0]['2P'].hist(bins=10, alpha=0.7)
df_copy[df_copy['cluster'] == 1]['2P'].hist(bins=10, alpha=0.7)

plt.legend(['0', '1'])
plt.show()
```





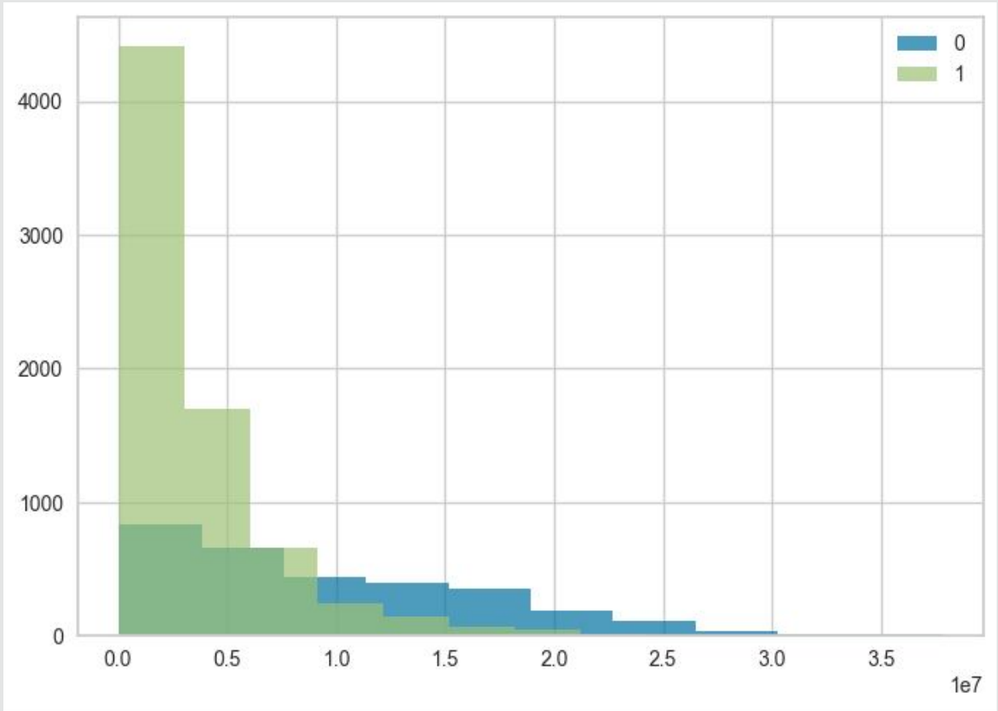
Interpretasi hasil cluster berdasarkan 3P

```
df_copy[df_copy['cluster'] == 0]['3P'].hist(bins=10, alpha=0.7)
df_copy[df_copy['cluster'] == 1]['3P'].hist(bins=10, alpha=0.7)

plt.legend(['0', '1'])
plt.show()
```



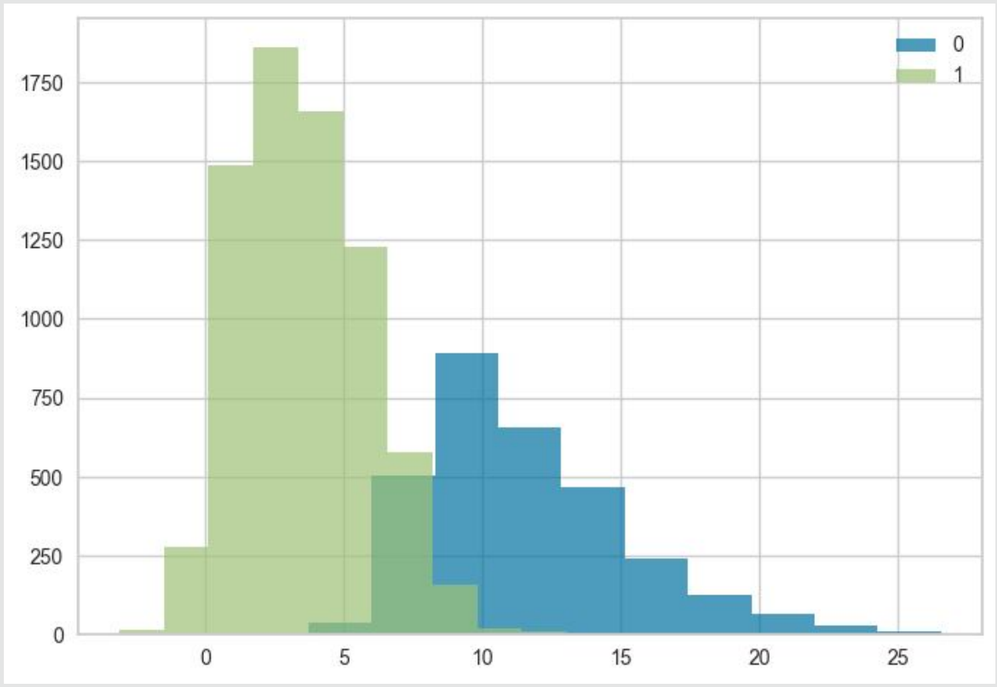
Interpretasi hasil cluster berdasarkan Salaries

```
df_copy[df_copy['cluster'] == 0]['Salaries'].hist(bins=10, alpha=0.7)
df_copy[df_copy['cluster'] == 1]['Salaries'].hist(bins=10, alpha=0.7)

plt.legend(['0', '1'])
plt.show()
```



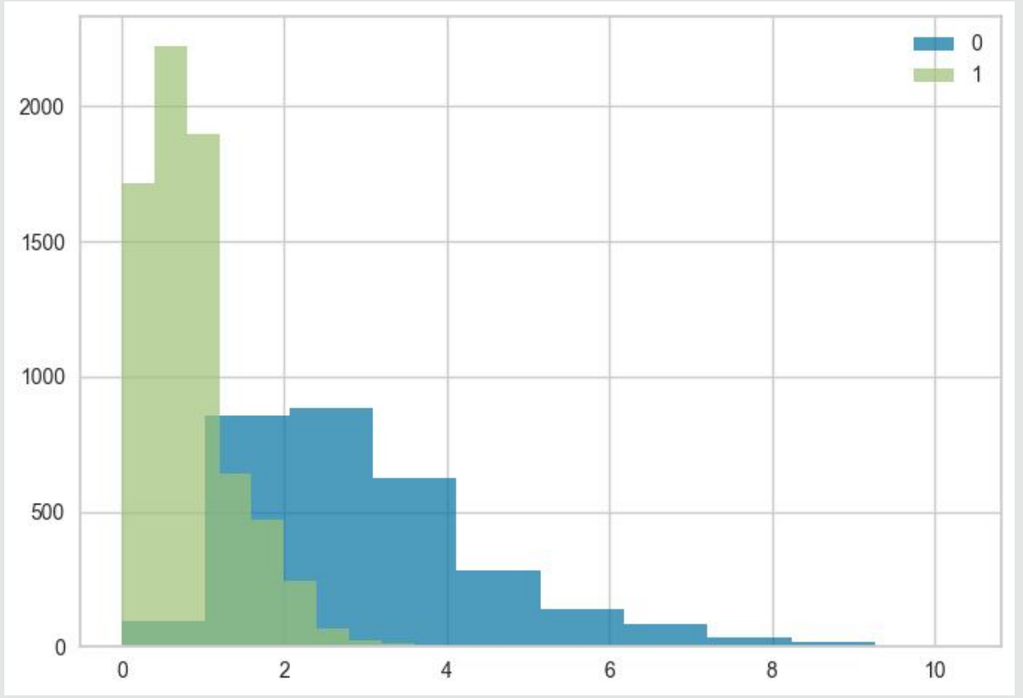
Interpretasi hasil cluster berdasarkan Avg GameScore

```
df_copy[df_copy['cluster'] == 0]['Avg GameScore'].hist(bins=10, alpha=0.7)
df_copy[df_copy['cluster'] == 1]['Avg GameScore'].hist(bins=10, alpha=0.7)

plt.legend(['0', '1'])
plt.show()
```

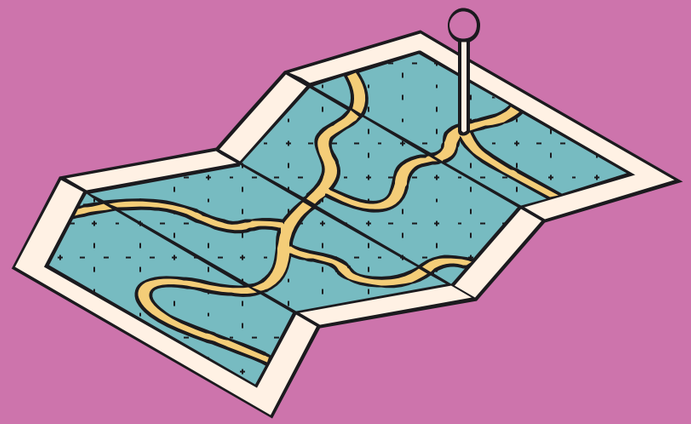


Interpretasi hasil cluster berdasarkan Free Throw (FT)

```
df_copy[df_copy['cluster'] == 0]['FT'].hist(bins=10, alpha=0.7)
df_copy[df_copy['cluster'] == 1]['FT'].hist(bins=10, alpha=0.7)

plt.legend(['0', '1'])
plt.show()
```



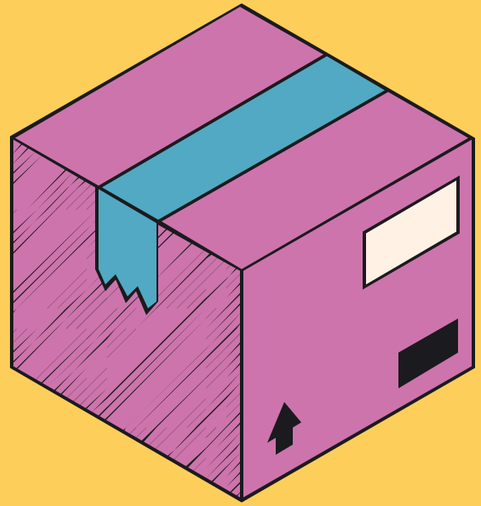
SUMBER



1B

Dataset harga persentase dolar tahun 2019

https://www.in2013dollars.com/us/inflation/2003

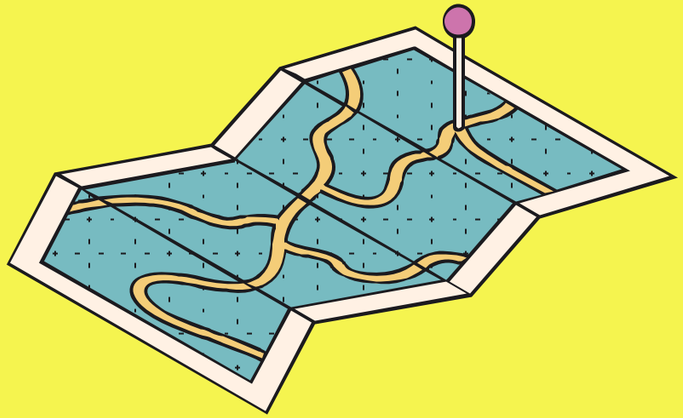


1E

Game Score

https://www.basketball-reference.com/about/glossary.html#pts

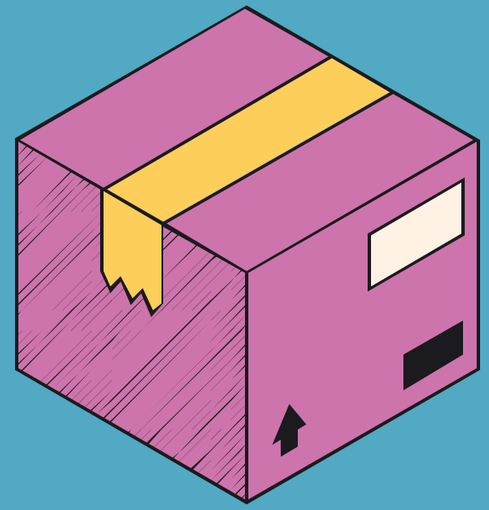
SUMBER



2A

Dataset tinggi dan berat pemain NBA

https://www.kaggle.com/code/justinas/nba-height-and-weight-analysis/ data?select=all_seasons.csv



2A

Jurnal pengaruh tinggi dan berat badan pemain terhadap posisi

https://www.researchgate.net/publication/357164353_BODY_HEIGHT_BODY_MASS_BODY_MASS_INDEX_OF_ELITE_BASKETBALL_PLAYERS_IN_RELATION_TO_THE_PLAYING_POSITION_AND_THEIR_IMPORTANCE_FOR_SUCCESS_IN_THE_GAME