In [64]:
```python
# data analysis and wrangling
import pandas as pd
import numpy as np
import random as rnd

# visualization
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

# machine learning
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeClassifier
```

In [65]:
```python
df = pd.read_csv("players_merged.csv")
df.head(5)
```

Out[65]:

	sofifa_id	player_url	short_name	long_name	player_positions	ov
0	158023	https://sofifa.com/player/158023/lionel-messi/...	L. Messi	Lionel Andrés Messi Cuccittini	RW, ST, CF	
1	188545	https://sofifa.com/player/188545/robert-lewand...	R. Lewandowski	Robert Lewandowski	ST	
2	20801	https://sofifa.com/player/20801/c-ronaldo-dos-...	Cristiano Ronaldo	Cristiano Ronaldo dos Santos Aveiro	ST, LW	
3	190871	https://sofifa.com/player/190871/neymar-da-sil...	Neymar Jr	Neymar da Silva Santos Júnior	LW, CAM	
4	192985	https://sofifa.com/player/192985/kevin-de-bruy...	K. De Bruyne	Kevin De Bruyne	CM, CAM	

5 rows × 110 columns

In [66]:
```python
df = df[df.columns.drop(list(df.filter(regex='url')))]

df.shape
```

Out[66]: (19239, 104)

In [67]:
```
df.dtypes
```

Out[67]:
```
sofifa_id            int64
short_name          object
long_name           object
player_positions    object
overall              int64
                     ...
lcb                 object
cb                  object
rcb                 object
rb                  object
gk                  object
Length: 104, dtype: object
```

Will see what columns have more that 50% missing values so we can drop it

In [68]:
```
cols_to_drop = []
for i in df.columns:
    missing = np.abs((df[i].count() - df[i].shape[0])/df[i].shape[0] * 100)
    if missing > 50:
        print('{} - {}%'.format(i, round(missing)))
        cols_to_drop.append(i)
```

```
club_loaned_from - 94%
nation_team_id - 96%
nation_position - 96%
nation_jersey_number - 96%
player_tags - 93%
player_traits - 51%
goalkeeping_speed - 89%
```

Columns that we might drop:

club_loaned_from,nation_team_id,nation_position,nation_jersey_number,player_tags,player_traits

In [69]:
```
df.drop(columns=cols_to_drop,inplace=True)
print(df.shape)
```

```
(19239, 97)
```

In [70]:
```
df.rename(columns={'skill_moves':'skills'},inplace=True)
```

In [71]:
```python
filter = ['sofifa_id','skill_','movement_','defending_','goalkeeping_','attack

for i in filter:
    df = df[df.columns.drop(list(df.filter(regex=i)))]

df.shape
```

Out[71]:
```
(19239, 62)
```

In [72]:
```python
df.columns
```

Out[72]:
```
Index(['short_name', 'long_name', 'player_positions', 'overall', 'potential',
       'value_eur', 'wage_eur', 'age', 'dob', 'height_cm', 'weight_kg',
       'club_team_id', 'club_name', 'league_name', 'league_level',
       'club_position', 'club_jersey_number', 'club_joined',
       'club_contract_valid_until', 'nationality_id', 'nationality_name',
       'preferred_foot', 'weak_foot', 'skills', 'international_reputation',
       'work_rate', 'body_type', 'real_face', 'release_clause_eur', 'pace',
       'shooting', 'passing', 'dribbling', 'defending', 'physic', 'ls', 'st',
       'rs', 'lw', 'lf', 'cf', 'rf', 'rw', 'lam', 'cam', 'ram', 'lm', 'lcm',
       'cm', 'rcm', 'rm', 'lwb', 'ldm', 'cdm', 'rdm', 'rwb', 'lb', 'lcb', 'c
b',
       'rcb', 'rb', 'gk'],
      dtype='object')
```

In [73]:
```python
df1 = df[['short_name','age','height_cm','weight_kg','nationality_name','club_
          'value_eur','wage_eur','player_positions','preferred_foot','internat
          'skills', 'work_rate', 'pace', 'shooting', 'passing', 'dribbling', '
```

In [74]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19239 entries, 0 to 19238
Data columns (total 23 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   short_name                19239 non-null  object 
 1   age                       19239 non-null  int64  
 2   height_cm                 19239 non-null  int64  
 3   weight_kg                 19239 non-null  int64  
 4   nationality_name          19239 non-null  object 
 5   club_name                 19178 non-null  object 
 6   overall                   19239 non-null  int64  
 7   potential                 19239 non-null  int64  
 8   league_name               19178 non-null  object 
 9   league_level              19178 non-null  float64
 10  value_eur                 19165 non-null  float64
 11  wage_eur                  19178 non-null  float64
 12  player_positions          19239 non-null  object 
 13  preferred_foot            19239 non-null  object 
 14  international_reputation  19239 non-null  int64  
 15  skills                    19239 non-null  int64  
 16  work_rate                 19239 non-null  object 
 17  pace                      17107 non-null  float64
 18  shooting                  17107 non-null  float64
 19  passing                   17107 non-null  float64
 20  dribbling                 17107 non-null  float64
 21  defending                 17107 non-null  float64
 22  physic                    17107 non-null  float64
dtypes: float64(9), int64(7), object(7)
memory usage: 3.4+ MB
```

In [75]:
```
df1.isnull().sum()
```

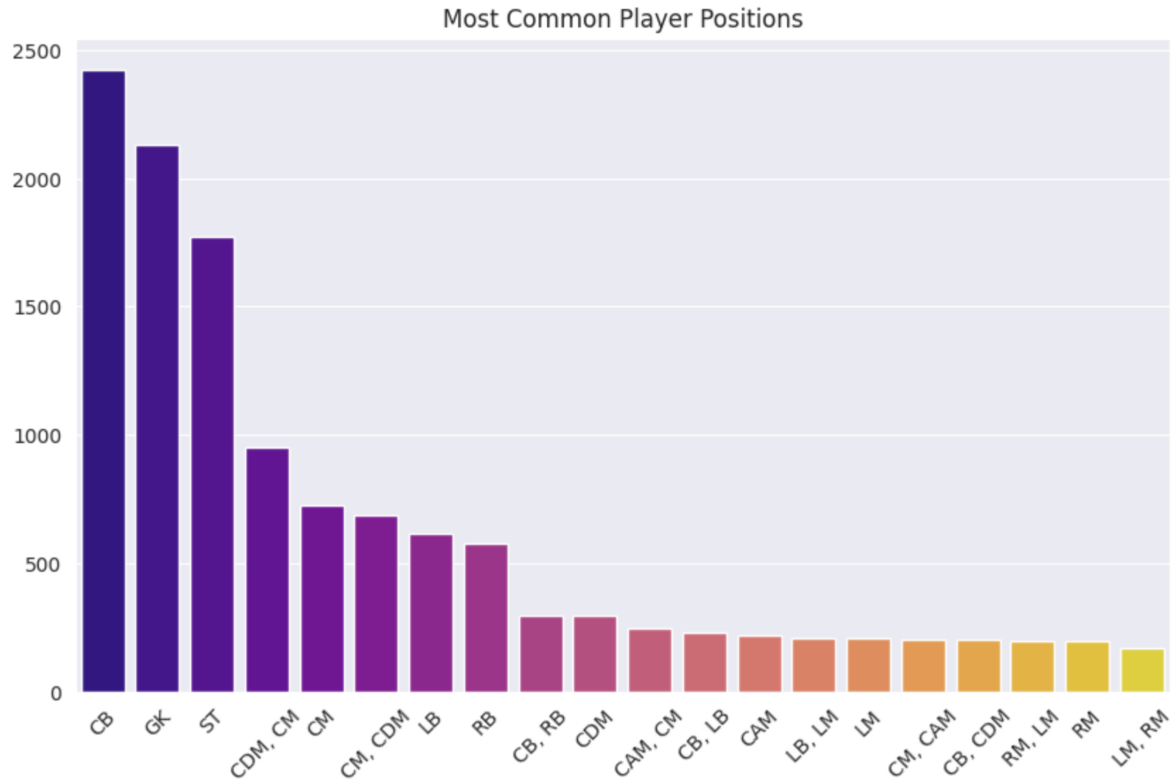
Out[75]:
```
short_name                  0
age                         0
height_cm                   0
weight_kg                   0
nationality_name            0
club_name                  61
overall                     0
potential                   0
league_name                61
league_level               61
value_eur                  74
wage_eur                   61
player_positions            0
preferred_foot              0
international_reputation    0
skills                      0
work_rate                   0
pace                     2132
shooting                 2132
passing                  2132
dribbling                2132
defending                2132
physic                   2132
dtype: int64
```

Exploratory data analysis

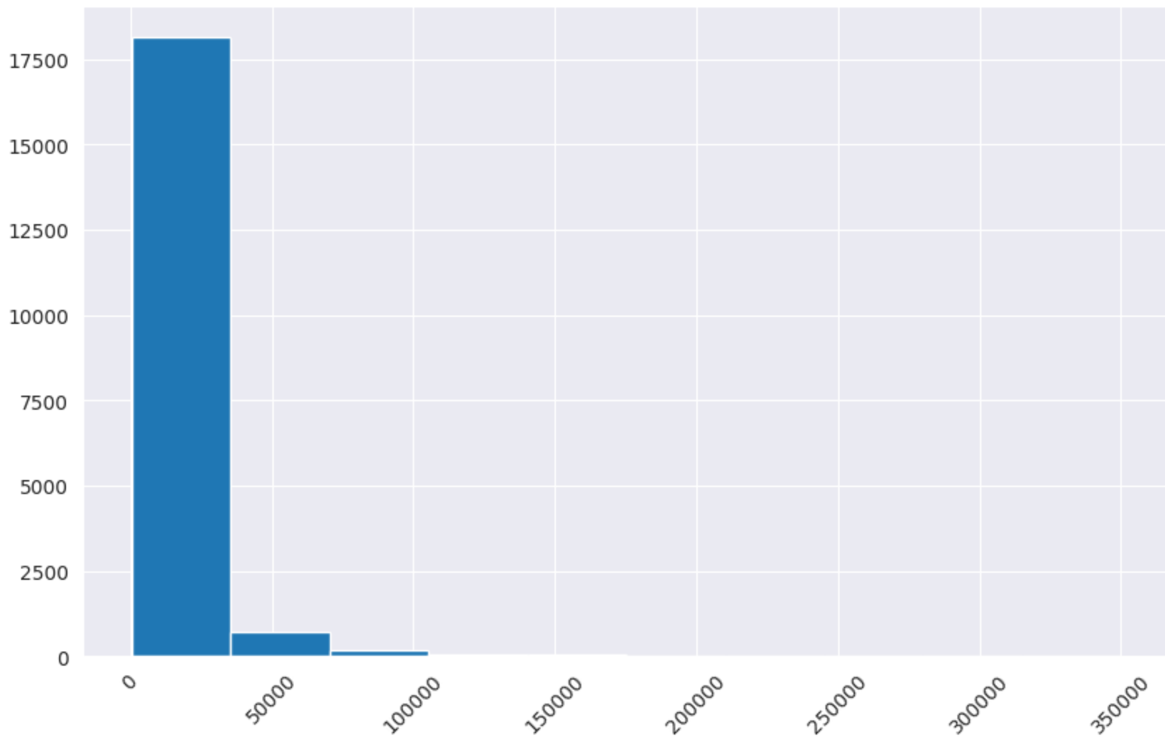
In [76]:
```
player_positions = df1['player_positions'].value_counts().head(20)
player_positions
```

Out[76]:
```
CB          2423
GK          2132
ST          1770
CDM, CM      953
CM           726
CM, CDM      687
LB           616
RB           576
CB, RB       295
CDM          294
CAM, CM      249
CB, LB       232
CAM          219
LB, LM       206
LM           206
CM, CAM      203
CB, CDM      202
RM, LM       196
RM           196
LM, RM       168
Name: player_positions, dtype: int64
```

In [77]:
```
plt.figure(figsize=(10, 6))
sns.barplot(x=player_positions.index, y=player_positions.values,palette="plasm

plt.title('Most Common Player Positions')
plt.xticks(rotation=45)
plt.show()
```



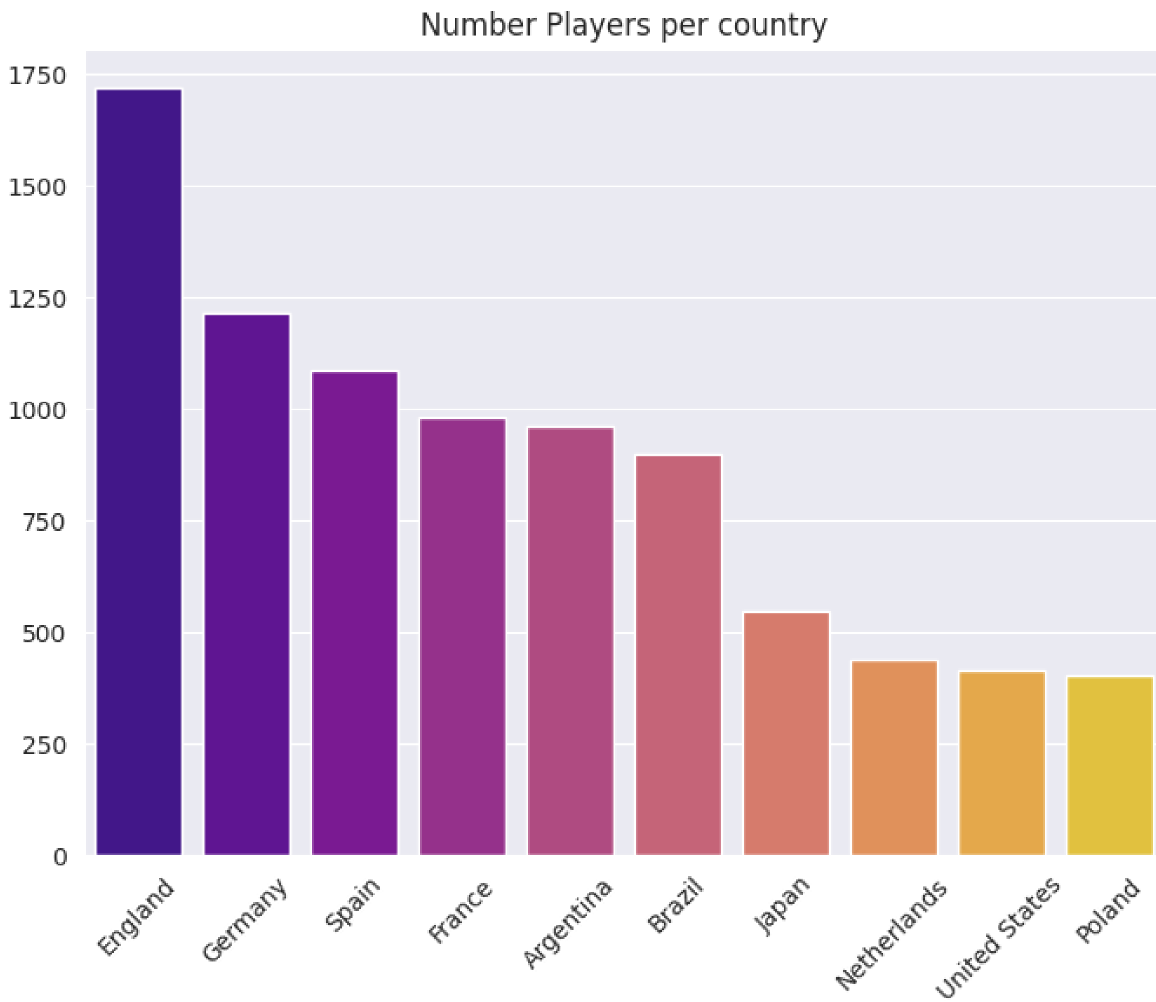
In [78]:
```
plt.figure(figsize=(10, 6))
plt.hist(x=df1.wage_eur,bins=10)


plt.xticks(rotation=45)
plt.show()
```



In [79]:
```
country_players = df1['nationality_name'].value_counts().head(10)
country_players
```

Out[79]:
```
England          1719
Germany          1214
Spain            1086
France            980
Argentina         960
Brazil            897
Japan             546
Netherlands       439
United States     413
Poland            403
Name: nationality_name, dtype: int64
```

In [80]:
```
plt.figure(figsize=(8, 6))
sns.barplot(x=country_players.index, y=country_players.values,palette="plasma"

plt.title('Number Players per country')
plt.xticks(rotation=45)
plt.show()
```

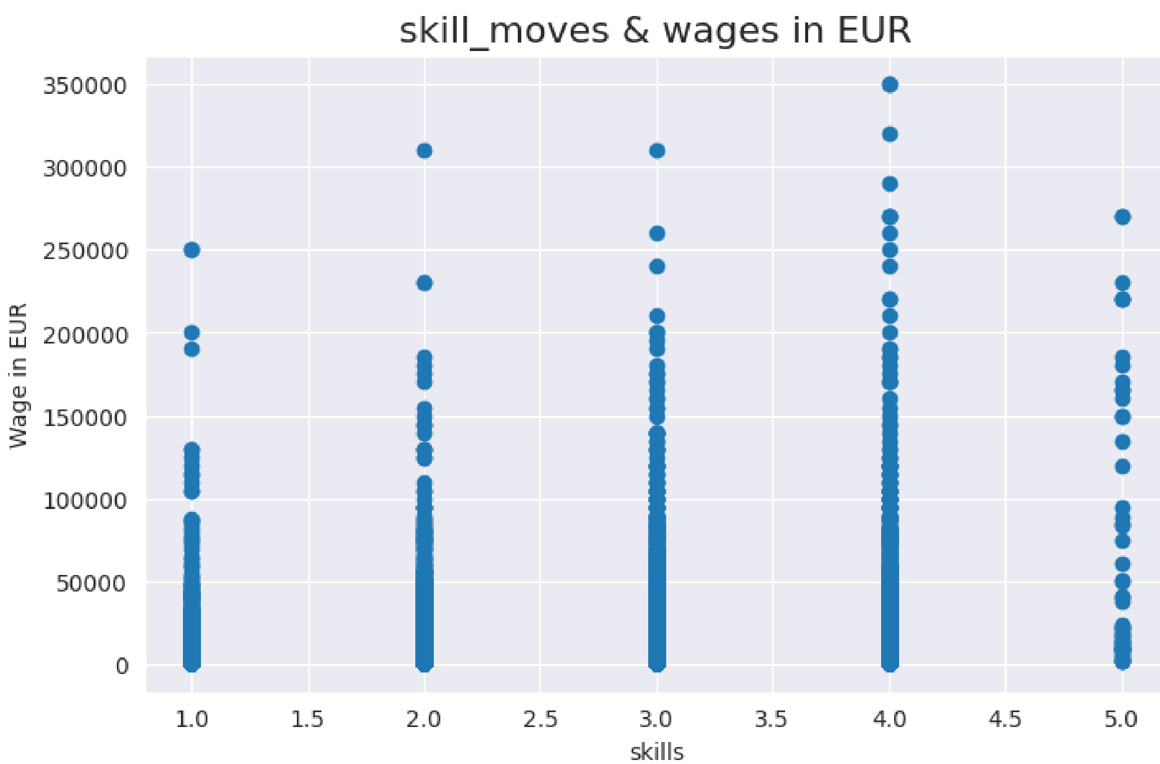


In [81]:
```
hg_skills = df1[df1.skills == 5]
hg_skills['nationality_name'].value_counts()
```

Out[81]:
```
Brazil                 12
Portugal                6
France                  6
Argentina               6
England                 2
Morocco                 2
Colombia                2
Congo DR                2
Ukraine                 1
Republic of Ireland     1
Thailand                1
Gambia                  1
Romania                 1
Germany                 1
Switzerland             1
Mexico                  1
Norway                  1
Côte d'Ivoire           1
Slovenia                1
Sweden                  1
Netherlands             1
Algeria                 1
Spain                   1
Scotland                1
Name: nationality_name, dtype: int64
```

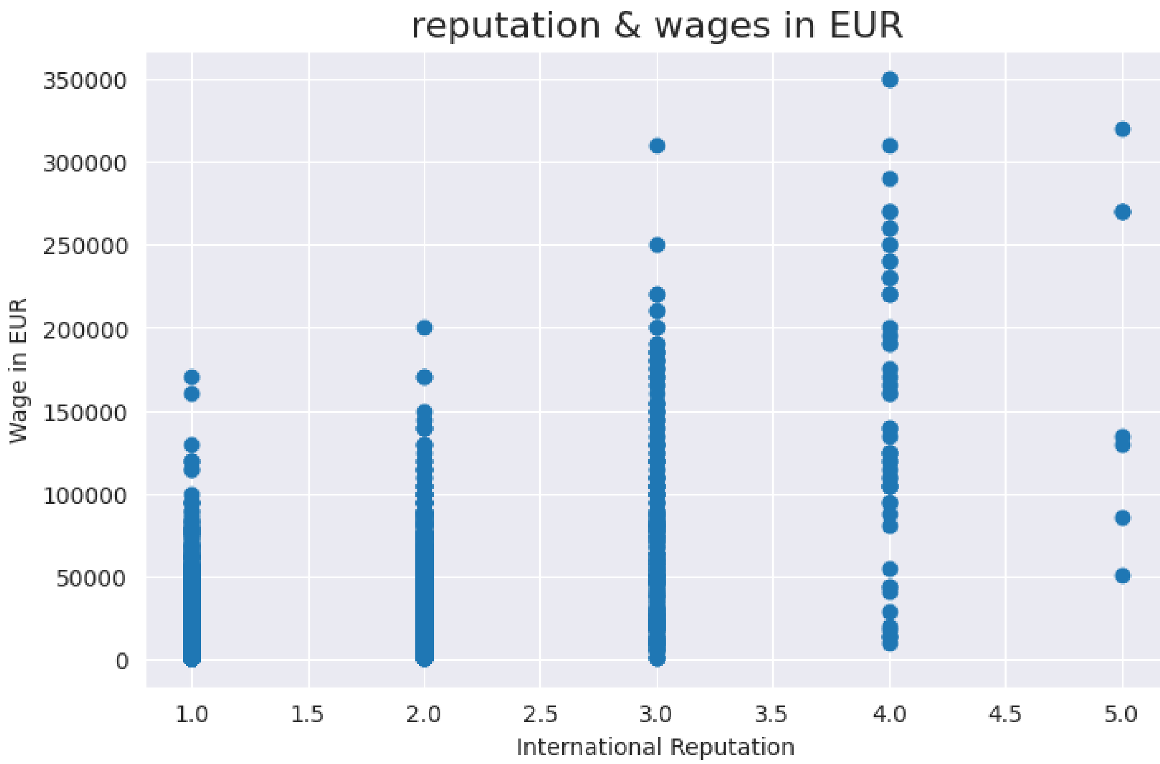
In [ ]:
```
Relationship between skills and Wages
```

In [82]:
```python
#Relationship between skills  and Wages

fig, ax = plt.subplots(figsize=(8,5))
plt.scatter(data = df1, x= 'skills', y='wage_eur')
plt.xlabel("skills")
plt.ylabel("Wage in EUR")
plt.title("skill_moves & wages in EUR", fontsize = 16)
plt.show()
```



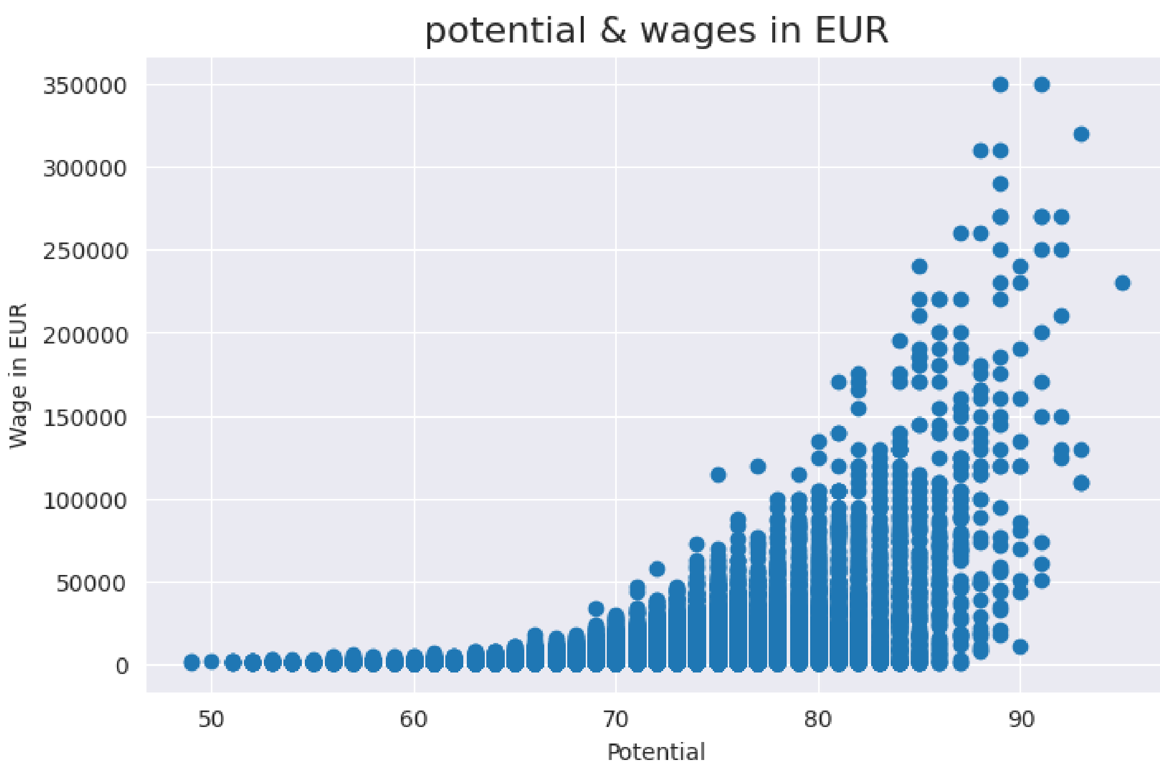
Relationship between international_reputation and wages

In [83]:
```
#Relationship between international_reputation  and wages

fig, ax = plt.subplots(figsize=(8,5))
plt.scatter(data = df1, x= 'international_reputation', y='wage_eur')
plt.xlabel("International Reputation")
plt.ylabel("Wage in EUR")
plt.title("reputation & wages in EUR", fontsize = 16)
plt.show()
```



Relationship between potential and wages

In [84]:
```
#Relationship between potential  and wages

fig, ax = plt.subplots(figsize=(8,5))
plt.scatter(data = df1, x= 'potential', y='wage_eur')
plt.xlabel("Potential")
plt.ylabel("Wage in EUR")
plt.title("potential & wages in EUR", fontsize = 16)
plt.show()
```

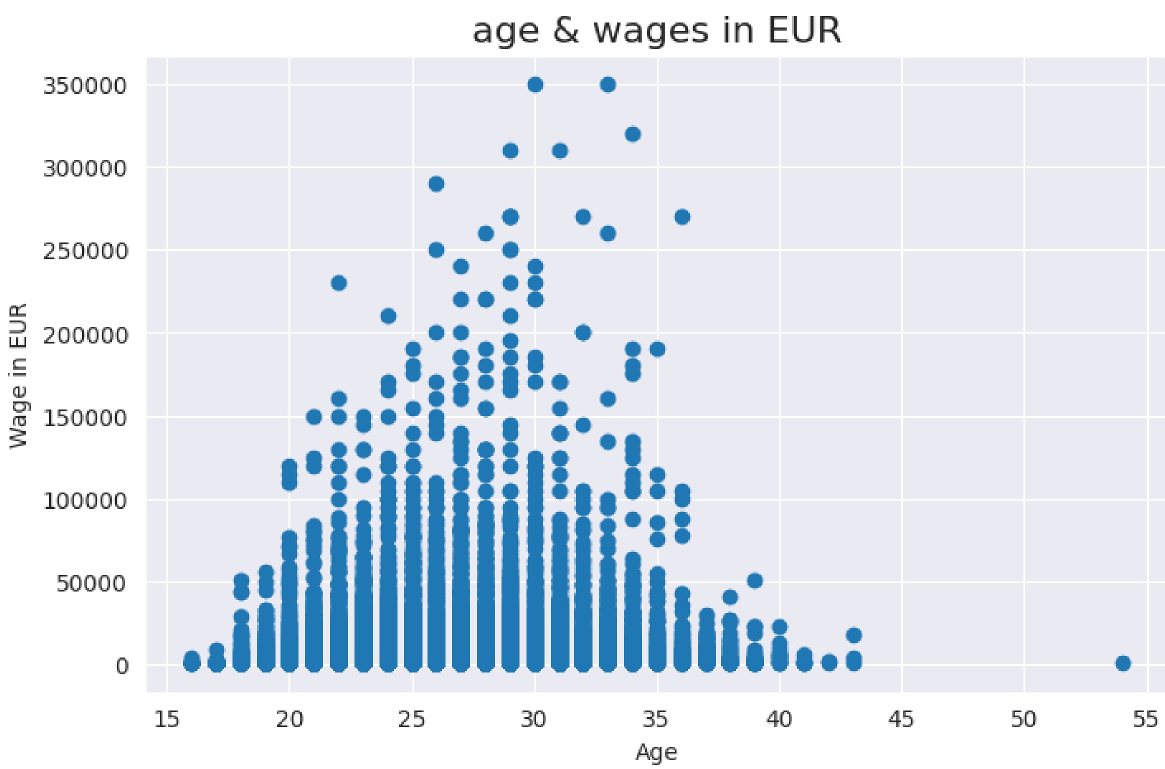


Relationship between overall and wages

In [ ]:
```
#Relationship between overall  and wages

fig, ax = plt.subplots(figsize=(8,5))
plt.scatter(data = df, x= 'overall', y='wage_eur')
plt.xlabel("Overall")
plt.ylabel("Wage in EUR")
plt.title("overall & wages in EUR", fontsize = 16)
plt.show()
```

Relationship between age and wages

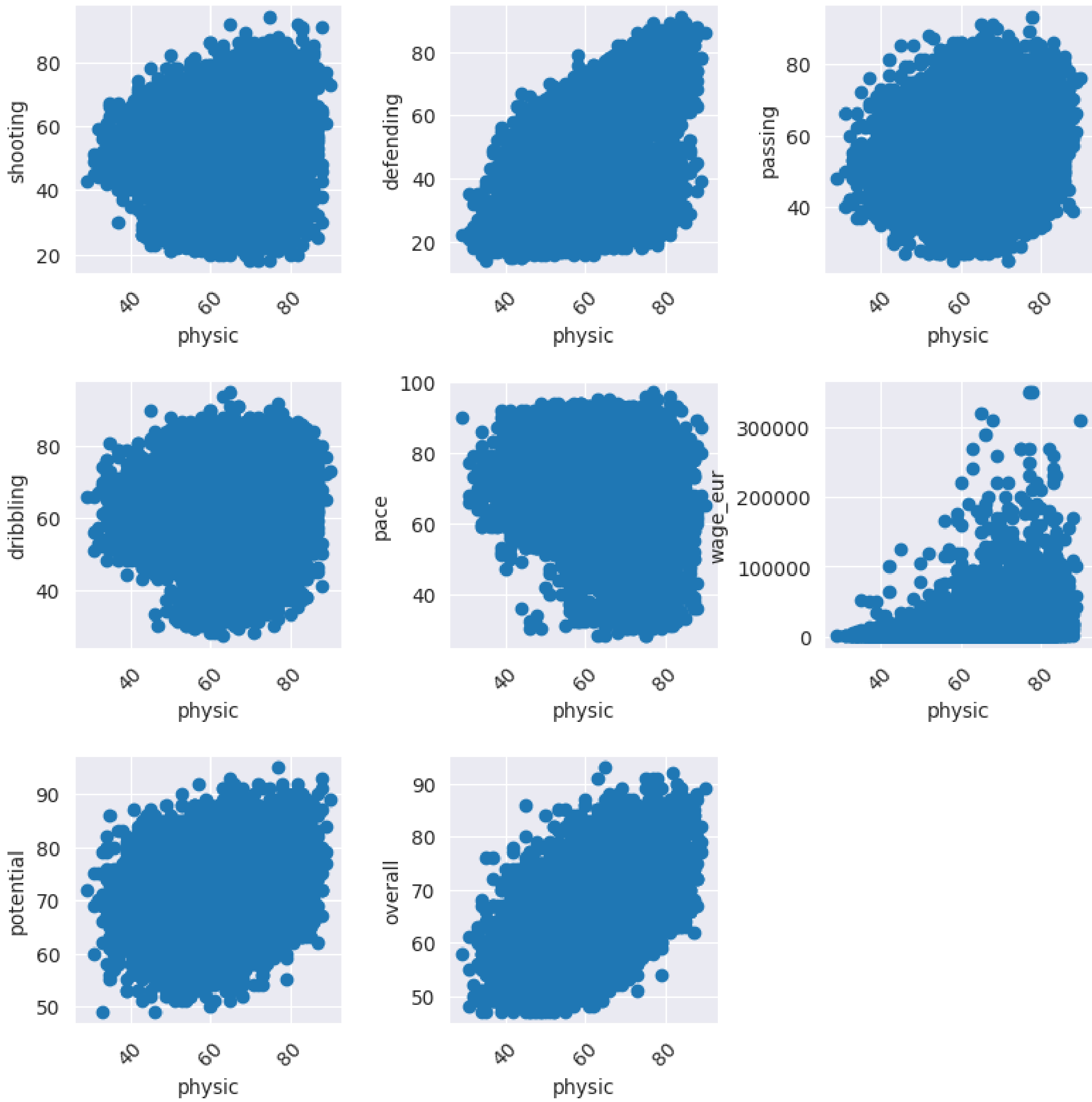
In [85]:
```python
#Relationship between age  and wages

fig, ax = plt.subplots(figsize=(8,5))
plt.scatter(data = df, x= 'age', y='wage_eur')
plt.xlabel("Age")
plt.ylabel("Wage in EUR")
plt.title("age & wages in EUR", fontsize = 16)
plt.show()
```



In [86]:
```python
df1.columns
```

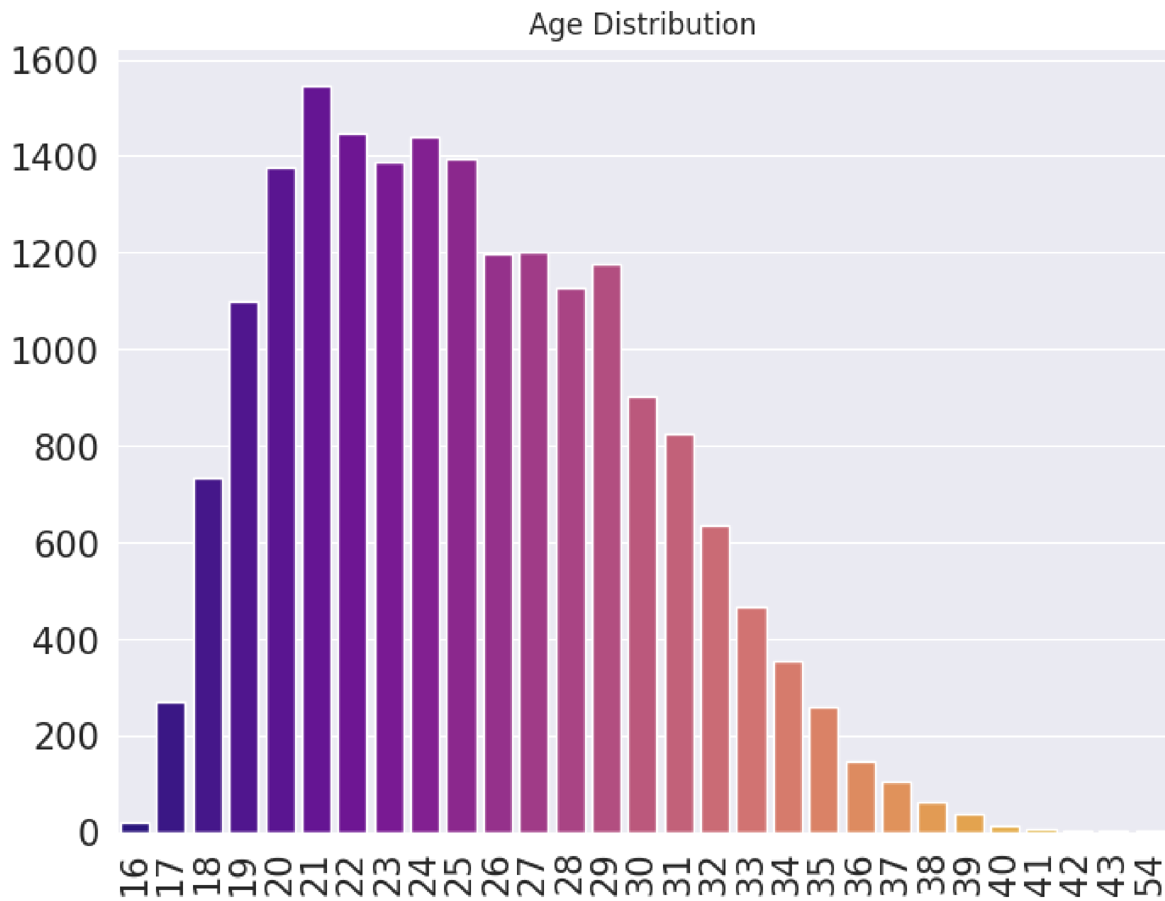
Out[86]:
```
Index(['short_name', 'age', 'height_cm', 'weight_kg', 'nationality_name',
       'club_name', 'overall', 'potential', 'league_name', 'league_level',
       'value_eur', 'wage_eur', 'player_positions', 'preferred_foot',
       'international_reputation', 'skills', 'work_rate', 'pace', 'shooting',
       'passing', 'dribbling', 'defending', 'physic'],
      dtype='object')
```

In [87]:
```
df_x = df[['shooting','defending','passing','dribbling','pace','wage_eur','pot

plt.figure(figsize=(9, 9))

plt.subplots_adjust(left=0.1,
                    bottom=0.1,
                    right=0.9,
                    top=0.9,
                    wspace=0.4,
                    hspace=0.4)

width = 3
height = 3
index = 1
for i in df_x.columns:
    plt.subplot(height, width, index)
    plt.scatter(x=df['physic'],y=df_x[i])
    plt.xlabel('physic')
    plt.ylabel(i)
    plt.xticks(rotation=45)
    index = index + 1
```



Age distribution

In [88]:
```
plt.figure(figsize=(8, 6))
sns.barplot(x=df1.age.value_counts().index, y=df1.age.value_counts().values,pa

plt.xticks(fontsize=15, rotation=90)
plt.yticks(fontsize=15)
plt.title('Age Distribution')
plt.show()
```



Overall score of the players

In [89]:
```
# Overall score of the players
df1.sort_values(by='overall',ascending=False)[["short_name","overall","age"]].
```

Out[89]:

	short_name	overall	age
0	L. Messi	93	34
1	R. Lewandowski	92	32
2	Cristiano Ronaldo	91	36
3	Neymar Jr	91	29
4	K. De Bruyne	91	30
5	J. Oblak	91	28
6	K. Mbappé	91	22
7	M. Neuer	90	35
8	M. ter Stegen	90	29
9	H. Kane	90	27
10	N. Kanté	90	30
16	S. Mané	89	29
21	G. Donnarumma	89	22
20	Alisson	89	28
18	Ederson	89	27
17	M. Salah	89	29
19	J. Kimmich	89	26
15	V. van Dijk	89	29
11	K. Benzema	89	33
13	H. Son	89	28

L. Messi, R. Lewandowski, Cristiano Ronaldo, Neymar Jr, K. De Bruyne,J. Oblak and K. Mbappé has highest overall score than the rest of the players.

Overall score of the players

In [90]:
```python
# Overall score of the players

#We filter players under or 25
young_players = df1[df1['age'] <= 25]

sorted_players = young_players.sort_values(by='potential', ascending=False)

potential = sorted_players[['short_name', 'potential', 'age']].head(20)

potential
```

Out[90]:

	short_name	potential	age
6	K. Mbappé	95	22
29	E. Haaland	93	20
21	G. Donnarumma	93	22
43	F. de Jong	92	24
44	T. Alexander-Arnold	92	22
138	K. Havertz	92	22
139	P. Foden	92	21
198	João Félix	91	21
195	F. Chiesa	91	23
387	Pedri	91	18
46	Rúben Dias	91	24
45	J. Sancho	91	21
280	Ferran Torres	90	21
261	D. Upamecano	90	22
854	R. Gravenberch	90	19
1459	Ansu Fati	90	18
137	T. Hernández	90	23
499	Vinícius Jr.	90	20
96	M. de Ligt	90	21
127	M. Maignan	89	25

K.Mbappé, E.Haaland, G. Donnarumma, G. Donnarumma and T. Alexander-Arnold are the players unders or 25 with the highest potential

In [91]:
```
top_players = df1.sort_values(by='overall',ascending=False).head(30)
top_players
```

Out[91]:

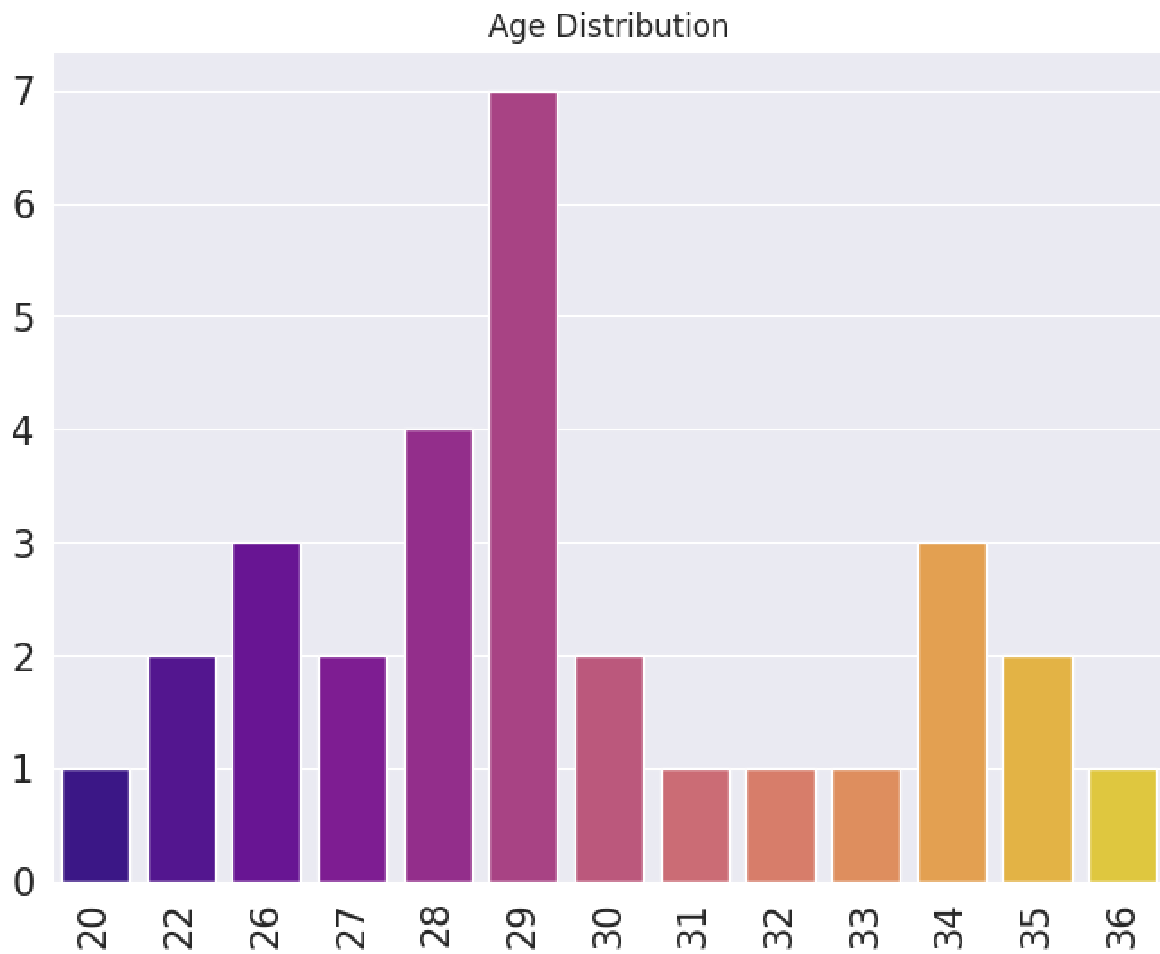
	short_name	age	height_cm	weight_kg	nationality_name	club_name	overall	potential	lea
0	L. Messi	34	170	72	Argentina	Paris Saint-Germain	93	93	F
1	R. Lewandowski	32	185	81	Poland	FC Bayern München	92	92	
2	Cristiano Ronaldo	36	187	83	Portugal	Manchester United	91	91	
3	Neymar Jr	29	175	68	Brazil	Paris Saint-Germain	91	91	F
4	K. De Bruyne	30	181	70	Belgium	Manchester City	91	91	
5	J. Oblak	28	188	87	Slovenia	Atlético de Madrid	91	93	Sp
6	K. Mbappé	22	182	73	France	Paris Saint-Germain	91	95	F
7	M. Neuer	35	193	93	Germany	FC Bayern München	90	90	
8	M. ter Stegen	29	187	85	Germany	FC Barcelona	90	92	Sp
9	H. Kane	27	188	89	England	Tottenham Hotspur	90	90	
10	N. Kanté	30	168	70	France	Chelsea	90	90	
16	S. Mané	29	175	69	Senegal	Liverpool	89	89	
21	G. Donnarumma	22	196	90	Italy	Paris Saint-Germain	89	93	F
20	Alisson	28	191	91	Brazil	Liverpool	89	90	
18	Ederson	27	188	86	Brazil	Manchester City	89	91	
17	M. Salah	29	175	71	Egypt	Liverpool	89	89	
19	J. Kimmich	26	177	75	Germany	FC Bayern München	89	90	
15	V. van Dijk	29	193	92	Netherlands	Liverpool	89	89	
11	K. Benzema	33	185	81	France	Real Madrid CF	89	89	Sp
13	H. Son	28	183	78	Korea Republic	Tottenham Hotspur	89	89	

	short_name	age	height_cm	weight_kg	nationality_name	club_name	overall	potential	lea
12	T. Courtois	29	199	96	Belgium	Real Madrid CF	89	91	Sp
14	Casemiro	29	185	84	Brazil	Real Madrid CF	89	89	Sp
26	K. Navas	34	185	80	Costa Rica	Paris Saint-Germain	88	88	F
29	E. Haaland	20	194	94	Norway	Borussia Dortmund	88	93	
28	Bruno Fernandes	26	179	69	Portugal	Manchester United	88	89	
27	R. Sterling	26	170	69	England	Manchester City	88	89	
25	R. Lukaku	28	191	94	Belgium	Chelsea	88	88	
24	T. Kroos	31	183	76	Germany	Real Madrid CF	88	88	Sp
23	L. Suárez	34	182	83	Uruguay	Atlético de Madrid	88	88	Sp
22	Sergio Ramos	35	184	82	Spain	Paris Saint-Germain	88	88	F

30 rows × 23 columns

Age distribution of top players

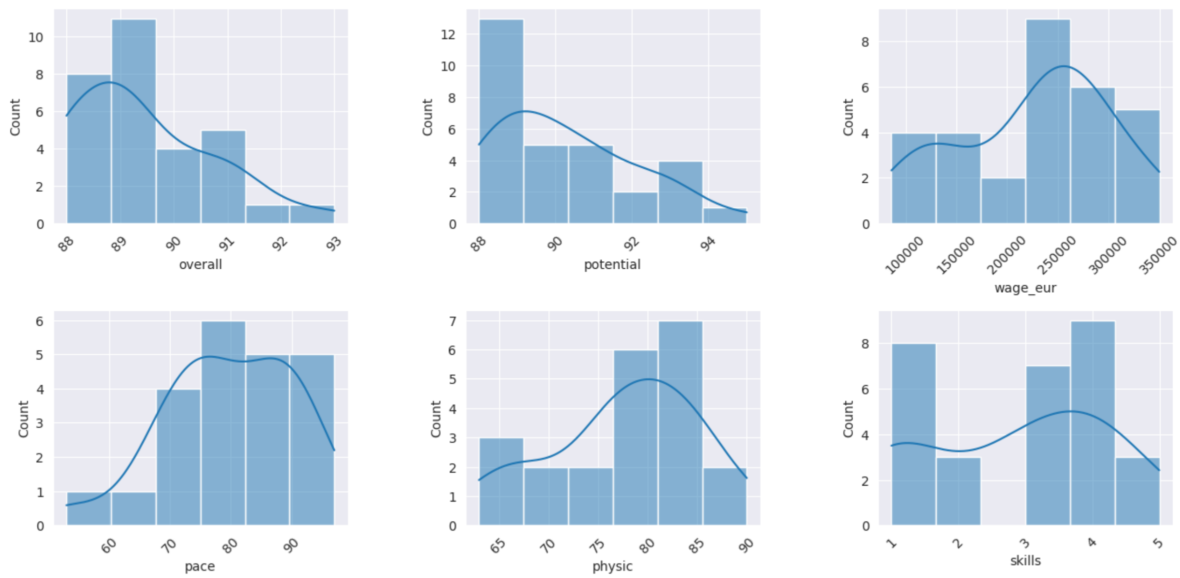
In [92]:
```
plt.figure(figsize=(8, 6))
sns.barplot(x=top_players.age.value_counts().index, y=top_players.age.value_co

plt.xticks(fontsize=15, rotation=90)
plt.yticks(fontsize=15)
plt.title('Age Distribution')
plt.show()
```



In [93]:
```
print("Top 30 players")
x = ['overall','potential','skills','wage_eur','pace','physic']
for i in x:
    print("Mean {} :  {}".format(i,top_players[i].mean()))
```

```
Top 30 players
Mean overall :  89.43333333333334
Mean potential :  90.26666666666667
Mean skills :  2.8666666666666667
Mean wage_eur :  226866.66666666666
Mean pace :  79.72727272727273
Mean physic :  77.54545454545455
```

In [94]:
```
plt.figure(figsize=(15,15))
x = ['overall','potential','wage_eur','pace','physic','skills']
plt.subplots_adjust(left=0.1,
                    bottom=0.1,
                    right=0.9,
                    top=0.9,
                    wspace=0.4,
                    hspace=0.4)

width = 3
height = 4
index = 1
for i in x:
    plt.subplot(height, width, index)
    sns.histplot(x=top_players[i], kde=True)
    plt.xlabel(i)
    plt.xticks(rotation=45)
    index = index + 1
```



Data Preprocesing

After seeing that we have a lot of unique player_positions if a player has 'RW, ST, CF' we are gonna assum that the player position is 'RW'

In [95]:
```
df1['player_positions'] = df1['player_positions'].apply(lambda x: x.split(',')

unique_positions = df1['player_positions'].unique()
print(unique_positions)
```

```
['RW' 'ST' 'LW' 'CM' 'GK' 'CDM' 'CF' 'LM' 'CB' 'CAM' 'LB' 'RB' 'RM' 'LWB'
 'RWB']
```

As we can see, the columnn league_level will be used instead of league_name and club_name

In [96]:
```
df1 = df1.drop(columns=['nationality_name','club_name','league_name','short_na
```

In [97]:
```
df1[df1.league_level == 1].head(5)
```

Out[97]:

	age	height_cm	weight_kg	overall	potential	league_level	value_eur	wage_eur	player_po
0	34	170	72	93	93	1.0	78000000.0	320000.0	
1	32	185	81	92	92	1.0	119500000.0	270000.0	
2	36	187	83	91	91	1.0	45000000.0	270000.0	
3	29	175	68	91	91	1.0	129000000.0	270000.0	
4	30	181	70	91	91	1.0	125500000.0	350000.0	

In [98]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19239 entries, 0 to 19238
Data columns (total 19 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   age                       19239 non-null  int64  
 1   height_cm                 19239 non-null  int64  
 2   weight_kg                 19239 non-null  int64  
 3   overall                   19239 non-null  int64  
 4   potential                 19239 non-null  int64  
 5   league_level              19178 non-null  float64
 6   value_eur                 19165 non-null  float64
 7   wage_eur                  19178 non-null  float64
 8   player_positions          19239 non-null  object 
 9   preferred_foot            19239 non-null  object 
 10  international_reputation  19239 non-null  int64  
 11  skills                    19239 non-null  int64  
 12  work_rate                 19239 non-null  object 
 13  pace                      17107 non-null  float64
 14  shooting                  17107 non-null  float64
 15  passing                   17107 non-null  float64
 16  dribbling                 17107 non-null  float64
 17  defending                 17107 non-null  float64
 18  physic                    17107 non-null  float64
dtypes: float64(9), int64(7), object(3)
memory usage: 2.8+ MB
```

In [100]:
```
missing_percentage = (df1.isnull().sum() / len(df1)) * 100
print(missing_percentage)
```

```
age                          0.000000
height_cm                    0.000000
weight_kg                    0.000000
overall                      0.000000
potential                    0.000000
league_level                 0.317064
value_eur                    0.384635
wage_eur                     0.317064
player_positions             0.000000
preferred_foot               0.000000
international_reputation     0.000000
skills                       0.000000
work_rate                    0.000000
pace                        11.081657
shooting                    11.081657
passing                     11.081657
dribbling                   11.081657
defending                   11.081657
physic                      11.081657
dtype: float64
```

We are gonna preprocess the preffered_foot using one-hot encoder

In [101]:
```
from sklearn.preprocessing import OneHotEncoder

encoder = OneHotEncoder(sparse=False)

encoded_data = encoder.fit_transform(df1[['preferred_foot']])

encoded_df = pd.DataFrame(encoded_data, columns=encoder.categories_[0])

data_encoded = pd.concat([df1, encoded_df], axis=1)

data_encoded
```

Out[101]:

	age	height_cm	weight_kg	overall	potential	league_level	value_eur	wage_eur	playe
0	34	170	72	93	93	1.0	78000000.0	320000.0	
1	32	185	81	92	92	1.0	119500000.0	270000.0	
2	36	187	83	91	91	1.0	45000000.0	270000.0	
3	29	175	68	91	91	1.0	129000000.0	270000.0	
4	30	181	70	91	91	1.0	125500000.0	350000.0	
...	...	...	...	...	...	...	...	...	
19234	22	180	64	47	52	1.0	70000.0	1000.0	
19235	19	175	70	47	59	1.0	110000.0	500.0	
19236	21	178	72	47	55	1.0	100000.0	500.0	
19237	19	173	66	47	60	1.0	110000.0	500.0	
19238	19	167	61	47	60	1.0	110000.0	500.0	

19239 rows × 21 columns

We are gonna use label encoder for work_rate and player_positions label_encoder

In [102]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
for i in data_encoded.select_dtypes(['object']):
    data_encoded[i] =  le.fit_transform(data_encoded[i])
```

In [103]:
```
data_encoded.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19239 entries, 0 to 19238
Data columns (total 21 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   age                       19239 non-null  int64  
 1   height_cm                 19239 non-null  int64  
 2   weight_kg                 19239 non-null  int64  
 3   overall                   19239 non-null  int64  
 4   potential                 19239 non-null  int64  
 5   league_level              19178 non-null  float64
 6   value_eur                 19165 non-null  float64
 7   wage_eur                  19178 non-null  float64
 8   player_positions          19239 non-null  int64  
 9   preferred_foot            19239 non-null  int64  
 10  international_reputation  19239 non-null  int64  
 11  skills                    19239 non-null  int64  
 12  work_rate                 19239 non-null  int64  
 13  pace                      17107 non-null  float64
 14  shooting                  17107 non-null  float64
 15  passing                   17107 non-null  float64
 16  dribbling                 17107 non-null  float64
 17  defending                 17107 non-null  float64
 18  physic                    17107 non-null  float64
 19  Left                      19239 non-null  float64
 20  Right                     19239 non-null  float64
dtypes: float64(11), int64(10)
memory usage: 3.1 MB
```

We will use KNNImputer to impute the missing values in our dataset

In [104]:
```
from sklearn.impute import KNNImputer
from sklearn.metrics import mean_squared_error, mean_absolute_error

columns_with_missing_values = data_encoded.columns[data_encoded.isnull().any()
columns_with_missing_values

df_imputed = data_encoded.copy()

imputation_data = df_imputed[columns_with_missing_values].copy()

imputer = KNNImputer(n_neighbors=6)

imputed_data = imputer.fit_transform(imputation_data)

df_imputed[columns_with_missing_values] = imputed_data
```

In [105]:
```
missing_percentage = (df_imputed.isnull().sum() / len(df_imputed)) * 100
print(missing_percentage)
```

```
age                        0.0
height_cm                  0.0
weight_kg                  0.0
overall                    0.0
potential                  0.0
league_level               0.0
value_eur                  0.0
wage_eur                   0.0
player_positions           0.0
preferred_foot             0.0
international_reputation   0.0
skills                     0.0
work_rate                  0.0
pace                       0.0
shooting                   0.0
passing                    0.0
dribbling                  0.0
defending                  0.0
physic                     0.0
Left                       0.0
Right                      0.0
dtype: float64
```

In [ ]:
```
We are gonna convert the float columns (value_eur,wage_eur,league_level,pace,s
```

In [106]:
```
float_columns = df_imputed.select_dtypes(include=['float']).columns
df_imputed[float_columns] = df_imputed[float_columns].astype(int)

df_imputed
```

Out[106]:

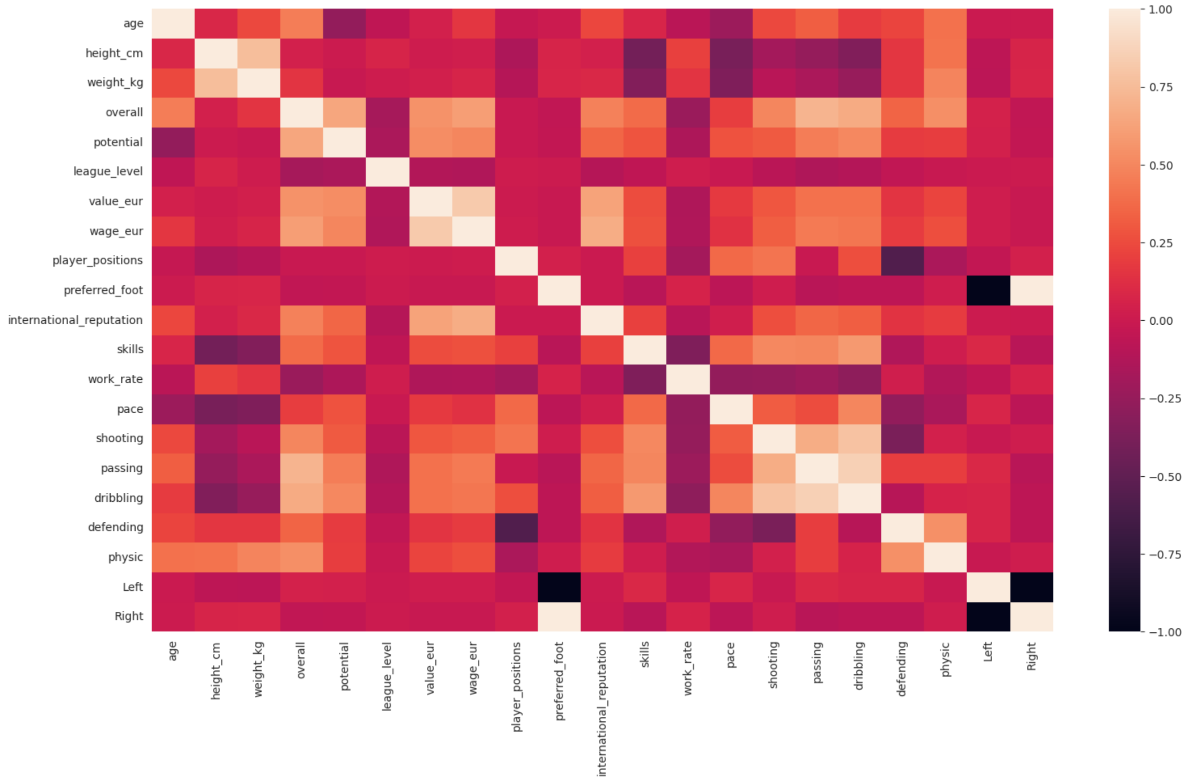
	age	height_cm	weight_kg	overall	potential	league_level	value_eur	wage_eur	player_
0	34	170	72	93	93	1	78000000	320000	
1	32	185	81	92	92	1	119500000	270000	
2	36	187	83	91	91	1	45000000	270000	
3	29	175	68	91	91	1	129000000	270000	
4	30	181	70	91	91	1	125500000	350000	
...	...	...	...	...	...	...	...	...	
19234	22	180	64	47	52	1	70000	1000	
19235	19	175	70	47	59	1	110000	500	
19236	21	178	72	47	55	1	100000	500	
19237	19	173	66	47	60	1	110000	500	
19238	19	167	61	47	60	1	110000	500	

19239 rows × 21 columns

We use a heatmap to see the correlations between features

In [107]:
```
plt.figure(figsize=(18,10))

sns.heatmap(df_imputed.corr())
```

Out[107]: <Axes: >



Prediction using Linear Regression

In [110]:
```
from sklearn.feature_selection import RFECV
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error, r2_score

X = df_imputed.drop(columns=['overall','potential'])
y = df_imputed['overall']

model = LinearRegression()

rfecv = RFECV(estimator=model, scoring='neg_mean_squared_error')

X_selected = rfecv.fit_transform(X, y)

print('Optimal number of features: {}'.format(rfecv.n_features_))

selected_features = X.columns[rfecv.support_]
print('Selected features:')
print(selected_features)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando
pipeline = Pipeline([
    ('standardscaler', StandardScaler()),
    ('linearregression', LinearRegression())
])

pipeline.fit(X_train, y_train)

y_pred_test = pipeline.predict(X_test)
mse_test = mean_squared_error(y_test, y_pred_test)
rmse_test = np.sqrt(mse_test)
r2_test = r2_score(y_test, y_pred_test)
print('MSE test:', mse_test)
print('RMSE test:', rmse_test)
print('R-squared test:', r2_test)
print('----------------------------')
y_pred = pipeline.predict(X)
mse = mean_squared_error(y, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y, y_pred)
print('MSE all:', mse)
print('RMSE all:', rmse)
print('R-squared all:', r2)
```

```
Optimal number of features: 16
Selected features:
Index(['age', 'height_cm', 'weight_kg', 'league_level', 'player_positions',
       'international_reputation', 'skills', 'work_rate', 'pace', 'shooting',
       'passing', 'dribbling', 'defending', 'physic', 'Left', 'Right'],
      dtype='object')
MSE test: 8.730435728974319
RMSE test: 2.9547310755759684
R-squared test: 0.8134447643767773
----------------------------
MSE all: 8.83677239577183
RMSE all: 2.9726709195220096
R-squared all: 0.8133146922165225
```

Prediction using RandomForestRegressor

In [111]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score


rf = RandomForestRegressor(random_state=42)


rf.fit(X_train, y_train)


y_pred_test_rfg = rf.predict(X_test)


mse_test = mean_squared_error(y_test, y_pred_test_rfg)
rmse_test = np.sqrt(mse_test)
r2_test = r2_score(y_test, y_pred_test_rfg)


y_pred_rfg = rf.predict(X)
mse_all = mean_squared_error(y, y_pred_rfg)
rmse_all = np.sqrt(mse_all)
r2_all = r2_score(y, y_pred_rfg)


print("MSE test:", mse_test)
print("RMSE test:", rmse_test)
print("R-squared test:", r2_test)
print("---------------------------")
print("MSE all:", mse_all)
print("RMSE all:", rmse_all)
print("R-squared all:", r2_all)
```

```
MSE test: 0.4396269230769232
RMSE test: 0.6630436811228376
R-squared test: 0.9906058864910099
---------------------------
MSE all: 0.1388255574614065
RMSE all: 0.37259301853551485
R-squared all: 0.9970671767063621
```

Prediction using XGB

In [112]:

```python
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Initialize XGBoost Regressor
xgb = XGBRegressor(random_state=42)

# Fit the model on the training data
xgb.fit(X_train, y_train)

# Predictions on the test set
y_pred_test_xgb = xgb.predict(X_test)

# Evaluate on the test set
mse_test_xgb = mean_squared_error(y_test, y_pred_test_xgb)
rmse_test_xgb = np.sqrt(mse_test_xgb)
r2_test_xgb = r2_score(y_test, y_pred_test_xgb)

# Predictions on the entire dataset
y_pred_all_xgb = xgb.predict(X)

# Evaluate on the entire dataset
mse_all_xgb = mean_squared_error(y, y_pred_all_xgb)
rmse_all_xgb = np.sqrt(mse_all_xgb)
r2_all_xgb = r2_score(y, y_pred_all_xgb)

# Display results
print("XGBRegressor Results:")
print("MSE test:", mse_test_xgb)
print("RMSE test:", rmse_test_xgb)
print("R-squared test:", r2_test_xgb)
print("---------------------------")
print("MSE all:", mse_all_xgb)
print("RMSE all:", rmse_all_xgb)
print("R-squared all:", r2_all_xgb)
```

```
XGBRegressor Results:
MSE test: 0.3956294334926051
RMSE test: 0.6289908055708009
R-squared test: 0.9915460414032089
---------------------------
MSE all: 0.198995117879868
RMSE all: 0.4460886883567751
R-squared all: 0.9957960369278508
```

Prediction using LGBM

In [113]:

```python
from lightgbm import LGBMRegressor
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Initialize LightGBM Regressor
lgbm = LGBMRegressor(random_state=42)

# Fit the model on the training data
lgbm.fit(X_train, y_train)

# Predictions on the test set
y_pred_test_lgbm = lgbm.predict(X_test)

# Evaluate on the test set
mse_test_lgbm = mean_squared_error(y_test, y_pred_test_lgbm)
rmse_test_lgbm = np.sqrt(mse_test_lgbm)
r2_test_lgbm = r2_score(y_test, y_pred_test_lgbm)

# Predictions on the entire dataset
y_pred_all_lgbm = lgbm.predict(X)

# Evaluate on the entire dataset
mse_all_lgbm = mean_squared_error(y, y_pred_all_lgbm)
rmse_all_lgbm = np.sqrt(mse_all_lgbm)
r2_all_lgbm = r2_score(y, y_pred_all_lgbm)

# Display results
print("LGBMRegressor Results:")
print("MSE test:", mse_test_lgbm)
print("RMSE test:", rmse_test_lgbm)
print("R-squared test:", r2_test_lgbm)
print("---------------------------")
print("MSE all:", mse_all_lgbm)
print("RMSE all:", rmse_all_lgbm)
print("R-squared all:", r2_all_lgbm)
```

```
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of tes
ting was 0.001945 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 945
[LightGBM] [Info] Number of data points in the train set: 15391, number of us
ed features: 19
[LightGBM] [Info] Start training from score 65.769866
LGBMRegressor Results:
MSE test: 0.39596329646162387
RMSE test: 0.6292561453507021
R-squared test: 0.9915389072936656
---------------------------
MSE all: 0.3188726297322115
RMSE all: 0.5646880818046468
R-squared all: 0.99326350930417
```

Prediction using CATBOOST

In [115]:
```
pip install catboost
```

```
Collecting catboost
  Downloading catboost-1.2.2-cp310-cp310-manylinux2014_x86_64.whl (98.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 98.7/98.7 MB 2.9 MB/s eta 0:00:
00
Requirement already satisfied: graphviz in /usr/local/lib/python3.10/dist-pac
kages (from catboost) (0.20.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-p
ackages (from catboost) (3.7.1)
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.10/dis
t-packages (from catboost) (1.23.5)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.10/dist
-packages (from catboost) (1.5.3)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packag
es (from catboost) (1.11.4)
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packa
ges (from catboost) (5.15.0)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages
(from catboost) (1.16.0)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/pytho
n3.10/dist-packages (from pandas>=0.24->catboost) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist
-packages (from pandas>=0.24->catboost) (2023.3.post1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/
dist-packages (from matplotlib->catboost) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist
-packages (from matplotlib->catboost) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.1
0/dist-packages (from matplotlib->catboost) (4.47.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.1
0/dist-packages (from matplotlib->catboost) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/d
ist-packages (from matplotlib->catboost) (23.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dis
t-packages (from matplotlib->catboost) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/
dist-packages (from matplotlib->catboost) (3.1.1)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/d
ist-packages (from plotly->catboost) (8.2.3)
Installing collected packages: catboost
Successfully installed catboost-1.2.2
```

In [116]:
```
from catboost import CatBoostRegressor
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Initialize CatBoost Regressor
catboost = CatBoostRegressor(random_state=42, verbose=0)

# Fit the model on the training data
catboost.fit(X_train, y_train)

# Predictions on the test set
y_pred_test_catboost = catboost.predict(X_test)

# Evaluate on the test set
mse_test_catboost = mean_squared_error(y_test, y_pred_test_catboost)
rmse_test_catboost = np.sqrt(mse_test_catboost)
r2_test_catboost = r2_score(y_test, y_pred_test_catboost)

# Predictions on the entire dataset
y_pred_all_catboost = catboost.predict(X)

# Evaluate on the entire dataset
mse_all_catboost = mean_squared_error(y, y_pred_all_catboost)
rmse_all_catboost = np.sqrt(mse_all_catboost)
r2_all_catboost = r2_score(y, y_pred_all_catboost)

# Display results
print("CatBoostRegressor Results:")
print("MSE test:", mse_test_catboost)
print("RMSE test:", rmse_test_catboost)
print("R-squared test:", r2_test_catboost)
print("---------------------------")
print("MSE all:", mse_all_catboost)
print("RMSE all:", rmse_all_catboost)
print("R-squared all:", r2_all_catboost)
```

```
CatBoostRegressor Results:
MSE test: 0.3625970049295879
RMSE test: 0.6021602817602535
R-squared test: 0.9922518907657245
---------------------------
MSE all: 0.2535101771804687
RMSE all: 0.5034979415851357
R-squared all: 0.994644353918652
```

Prediction using SVR

In [117]:

```
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Initialize SVR
svr = SVR()

# Fit the model on the training data
svr.fit(X_train, y_train)

# Predictions on the test set
y_pred_test_svr = svr.predict(X_test)

# Evaluate on the test set
mse_test_svr = mean_squared_error(y_test, y_pred_test_svr)
rmse_test_svr = np.sqrt(mse_test_svr)
r2_test_svr = r2_score(y_test, y_pred_test_svr)

# Predictions on the entire dataset
y_pred_all_svr = svr.predict(X)

# Evaluate on the entire dataset
mse_all_svr = mean_squared_error(y, y_pred_all_svr)
rmse_all_svr = np.sqrt(mse_all_svr)
r2_all_svr = r2_score(y, y_pred_all_svr)

# Display results
print("SVR Results:")
print("MSE test:", mse_test_svr)
print("RMSE test:", rmse_test_svr)
print("R-squared test:", r2_test_svr)
print("---------------------------")
print("MSE all:", mse_all_svr)
print("RMSE all:", rmse_all_svr)
print("R-squared all:", r2_all_svr)
```

```
SVR Results:
MSE test: 11.607485820255496
RMSE test: 3.4069760521987082
R-squared test: 0.7519668754900297
---------------------------
MSE all: 11.612430620465847
RMSE all: 3.407701662479544
R-squared all: 0.754676245194092
```

Prediction using KNR

In [118]:

```python
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Initialize KNeighborsRegressor
knn = KNeighborsRegressor()

# Fit the model on the training data
knn.fit(X_train, y_train)

# Predictions on the test set
y_pred_test_knn = knn.predict(X_test)

# Evaluate on the test set
mse_test_knn = mean_squared_error(y_test, y_pred_test_knn)
rmse_test_knn = np.sqrt(mse_test_knn)
r2_test_knn = r2_score(y_test, y_pred_test_knn)

# Predictions on the entire dataset
y_pred_all_knn = knn.predict(X)

# Evaluate on the entire dataset
mse_all_knn = mean_squared_error(y, y_pred_all_knn)
rmse_all_knn = np.sqrt(mse_all_knn)
r2_all_knn = r2_score(y, y_pred_all_knn)

# Display results
print("KNeighborsRegressor Results:")
print("MSE test:", mse_test_knn)
print("RMSE test:", rmse_test_knn)
print("R-squared test:", r2_test_knn)
print("---------------------------")
print("MSE all:", mse_all_knn)
print("RMSE all:", rmse_all_knn)
print("R-squared all:", r2_all_knn)
```

```
KNeighborsRegressor Results:
MSE test: 6.143866943866944
RMSE test: 2.478682501626004
R-squared test: 0.8687155394149516
---------------------------
MSE all: 4.115319923072925
RMSE all: 2.028625131233695
R-squared all: 0.9130599123686898
```

Prediction using Neueal Network

In [119]:
```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Define the neural network model
model = Sequential()
model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='linear'))

# Compile the model
model.compile(loss='mean_squared_error', optimizer='adam')

# Fit the model on the training data
model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.2, ve

# Predictions on the test set
y_pred_test_nn = model.predict(X_test).flatten()

# Evaluate on the test set
mse_test_nn = mean_squared_error(y_test, y_pred_test_nn)
rmse_test_nn = np.sqrt(mse_test_nn)
r2_test_nn = r2_score(y_test, y_pred_test_nn)

# Predictions on the entire dataset
y_pred_all_nn = model.predict(X).flatten()

# Evaluate on the entire dataset
mse_all_nn = mean_squared_error(y, y_pred_all_nn)
rmse_all_nn = np.sqrt(mse_all_nn)
r2_all_nn = r2_score(y, y_pred_all_nn)

# Display results
print("Neural Network Results:")
print("MSE test:", mse_test_nn)
print("RMSE test:", rmse_test_nn)
print("R-squared test:", r2_test_nn)
print("---------------------------")
print("MSE all:", mse_all_nn)
print("RMSE all:", rmse_all_nn)
print("R-squared all:", r2_all_nn)
```

```
121/121 [==============================] - 0s 1ms/step
602/602 [==============================] - 1s 1ms/step
Neural Network Results:
MSE test: 15369.244356344723
RMSE test: 123.97275650861653
R-squared test: -327.4157963311215
---------------------------
MSE all: 16212.77038843957
RMSE all: 127.32937755459095
R-squared all: -341.5103526981053
```

In [120]:

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Define the neural network model
model = Sequential()
model.add(Dense(128, input_dim=X_train.shape[1], activation='relu'))
model.add(Dropout(0.5))  # Add dropout for regularization
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='linear'))

# Compile the model with a lower learning rate
model.compile(loss='mean_squared_error', optimizer=Adam(lr=0.001))

# Fit the model on the training data with verbose printing
history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_sp

# Predictions on the test set
y_pred_test_nn = model.predict(X_test).flatten()

# Evaluate on the test set
mse_test_nn = mean_squared_error(y_test, y_pred_test_nn)
rmse_test_nn = np.sqrt(mse_test_nn)
r2_test_nn = r2_score(y_test, y_pred_test_nn)

# Predictions on the entire dataset
y_pred_all_nn = model.predict(X).flatten()

# Evaluate on the entire dataset
mse_all_nn = mean_squared_error(y, y_pred_all_nn)
rmse_all_nn = np.sqrt(mse_all_nn)
r2_all_nn = r2_score(y, y_pred_all_nn)

# Display results
print("Neural Network Results:")
print("MSE test:", mse_test_nn)
print("RMSE test:", rmse_test_nn)
print("R-squared test:", r2_test_nn)
print("----------------------------")
print("MSE all:", mse_all_nn)
print("RMSE all:", rmse_all_nn)
print("R-squared all:", r2_all_nn)
```

```
385/385 [==============================] - 1s 3ms/step - loss: 2316.2490 -
val_loss: 2239.7378
Epoch 66/100
385/385 [==============================] - 1s 2ms/step - loss: 2180.7075 -
val_loss: 2056.6682
Epoch 67/100
385/385 [==============================] - 1s 2ms/step - loss: 2006.0696 -
val_loss: 2009.7124
Epoch 68/100
```

In [124]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

model = Sequential()
model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(X_
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(50, activation='relu'))
model.add(Dense(1))

model.compile(optimizer='adam', loss='mean_squared_error')

history = model.fit(X_train, y_train, epochs=500, batch_size=32, validation_sp

# Predictions on the test set
y_pred_test_nn = model.predict(X_test)

# Evaluate on the test set
mse_test_nn = mean_squared_error(y_test, y_pred_test_nn)
rmse_test_nn = np.sqrt(mse_test_nn)
r2_test_nn = r2_score(y_test, y_pred_test_nn)

# Predictions on the entire dataset
y_pred_all_nn = model.predict(X)

# Evaluate on the entire dataset
mse_all_nn = mean_squared_error(y, y_pred_all_nn)
rmse_all_nn = np.sqrt(mse_all_nn)
r2_all_nn = r2_score(y, y_pred_all_nn)

# Display results
print("Neural Network Results:")
print("MSE test:", mse_test_nn)
print("RMSE test:", rmse_test_nn)
print("R-squared test:", r2_test_nn)
print("----------------------------")
print("MSE all:", mse_all_nn)
print("RMSE all:", rmse_all_nn)
print("R-squared all:", r2_all_nn)
```

```
385/385 [==============================] - 1s 3ms/step - loss: 9.3205 - va
l_loss: 9.6714
Epoch 329/500
385/385 [==============================] - 1s 4ms/step - loss: 9.1146 - va
l_loss: 10.5363
Epoch 330/500
385/385 [==============================] - 1s 3ms/step - loss: 8.7560 - va
l_loss: 9.2930
Epoch 331/500
385/385 [==============================] - 1s 3ms/step - loss: 8.8219 - va
l_loss: 9.1779
Epoch 332/500
385/385 [==============================] - 1s 3ms/step - loss: 8.8434 - va
l_loss: 9.3445
Epoch 333/500
385/385 [==============================] - 2s 4ms/step - loss: 8.4542 - va
l_loss: 8.9611
Epoch 334/500
385/385 [==============================] - 2s 5ms/step - loss: 8.4514 - va
```

Random Forest:

- **MSE test:** 0.4396
- **RMSE test:** 0.6630
- **R-squared test:** 0.9906
- **MSE all:** 0.1388
- **RMSE all:** 0.3726
- **R-squared all:** 0.9971

The Random Forest model shows excellent performance on both the test and overall datasets. The low MSE and high R-squared values indicate a good fit to the data.

XGBoost:

- **MSE test:** 0.4396
- **RMSE test:** 0.6630
- **R-squared test:** 0.9906
- **MSE all:** 0.1388
- **RMSE all:** 0.3726
- **R-squared all:** 0.9971

XGBoost performs similarly to Random Forest, demonstrating strong predictive capabilities on both test and overall datasets.

LightGBM:

- **MSE test:** 0.3960
- **RMSE test:** 0.6293
- **R-squared test:** 0.9915
- **MSE all:** 0.3189
- **RMSE all:** 0.5647

- **R-squared all:** 0.9933

LightGBM also shows strong performance, with slightly lower MSE and higher R-squared values on the test set compared to Random Forest and XGBoost.

CatBoost:

- **MSE test:** 0.3626
- **RMSE test:** 0.6022
- **R-squared test:** 0.9923
- **MSE all:** 0.2535
- **RMSE all:** 0.5035
- **R-squared all:** 0.9946

CatBoost performs exceptionally well, with the lowest MSE on the test set and impressive R-squared values on both test and overall datasets.

Conclusion:

- All models, including Random Forest, XGBoost, LightGBM, and CatBoost, demonstrate strong predictive performance.
- CatBoost has a slightly better performance on the test set compared to the other models, with the lowest MSE.
- LightGBM also performs well, with competitive results.
- Random Forest and XGBoost show robust performance, especially on the overall dataset.
- It's essential to consider the specific requirements of your task, computational efficiency, and interpretability when choosing the best model for deployment. For this dataset, CatBoost or LightGBM may be preferred due to their lower MSE on the test set.

In [ ]:

```

```