INTRODUCTION:

A new football club named 'Brussels United FC' has just been inaugurated. This club does not have a team yet. The team is looking to hire players for their roster. Management wants to make such decisions using data-based approach. To start with, a total 15 players are required. Player data for all teams has been acquired from FIFA. This data contains information about the players, the clubs they are currently playing for and various performance measures.

There is a limited budget for hiring players. The team needs 20 possible players to choose from. You have been requested to formulate a report in order to help the management decide regarding potential players.

1. DATA OVERVIEW:

- The data contains details for over 18,000 players playing in various football clubs in Europe.
- It contains information on age, skill rating, wages and player value, etc.
- File provided - Fifa.csv

ID	unique id for every player
Name	name
Age	age
Photo	url to the player's photo
Nationality	nationality
Flag	url to players's country flag
Overall	overall rating
Potential	potential rating
Club	current club
Club Logo	url to club logo
Value	current market value
Wage	current wage
Preferred Foot	left/right
International Reputation	rating on scale of 5
Weak Foot	rating on scale of 5
Skill Moves	rating on scale of 5
Work Rate	attack work rate/defence work rate
Body Type	body type of player
Position	position on the pitch

Jersey Number	jersey number
Joined	joined date
Loaned From	club name if applicable
Contract Valid Until	contract end date
Height	height of the player
Weight	weight of the player
Crossing	rating on scale of 100
Finishing	rating on scale of 100
HeadingAccuracy	rating on scale of 100
ShortPassing	rating on scale of 100
Volleys	rating on scale of 100
Dribbling	rating on scale of 100
Curve	rating on scale of 100
FKAccuracy	rating on scale of 100
LongPassing	rating on scale of 100
BallControl	rating on scale of 100
Acceleration	rating on scale of 100
SprintSpeed	rating on scale of 100
Agility	rating on scale of 100
Reactions	rating on scale of 100\
Balance	rating on scale of 100
ShotPower	rating on scale of 100
Jumping	rating on scale of 100
Stamina	rating on scale of 100
Strength	rating on scale of 100
LongShots	rating on scale of 100
Aggression	rating on scale of 100
Interceptions	rating on scale of 100
Positioning	rating on scale of 100
Vision	rating on scale of 100
Penalties	rating on scale of 100
Composure	rating on scale of 100
Marking	rating on scale of 100
StandingTackle	rating on scale of 100
SlidingTackle	rating on scale of 100
GKDiving	rating on scale of 100
GKHandling	rating on scale of 100
GKKicking	rating on scale of 100

GKPositioning	rating on scale of 100
GKReflexes	rating on scale of 100
Release Clause	release clause value

3. UNDERSTANDING THE DATA:

3. 1. Importing Libraries:

- Employed 'NumPy' and 'pandas' libraries for performing statistical operations and data analysis respectively.
- 'Matplotlib' and 'Seaborn' are other libraries, mainly used for data visualization purpose.
- 'Warnings' library is used to remove the warnings python throws. They are optional as the warnings are not critical.

```
1 import numpy as np
2 import pandas as pd
3 import seaborn as sns
4 import matplotlib.pyplot as plt
5 import warnings
6 warnings.filterwarnings("ignore")
```

4. 2. Read the dataset 'fifa.csv'

- As the name of the dataset suggests, which is **'fifa.csv'**, this file is in .csv format (Comma Separated files).
- It is said to be the most common format of data. **'read_csv()'** is the function used to retrieve data from a CSV file.
- **'head()'** and **'tail()'** are the functions commonly used to display the rows of the dataset.
- By default, the head function displays the first 5 records, whereas, tail displays the last 5 records.

```
df = pd.read_csv("fifa.csv")
df.head()
```

	ID	Name	Age	Photo	Nationality	Flag	Overall	Potential	Club	
0	158023	L. Messi	31	https://cdn.sofifa.org/players/4/19/158023.png	Argentina	https://cdn.sofifa.org/flags/52.png	94	94	FC Barcelona	https://cdn.sofifa.c
1	20801	Cristiano Ronaldo	33	https://cdn.sofifa.org/players/4/19/20801.png	Portugal	https://cdn.sofifa.org/flags/38.png	94	94	Juventus	https://cdn.sofifa
2	190871	Neymar Jr	26	https://cdn.sofifa.org/players/4/19/190871.png	Brazil	https://cdn.sofifa.org/flags/54.png	92	93	Paris Saint-Germain	https://cdn.sofifa
3	193080	De Gea	27	https://cdn.sofifa.org/players/4/19/193080.png	Spain	https://cdn.sofifa.org/flags/45.png	91	93	Manchester United	https://cdn.sofifa
4	192985	K. De Bruyne	27	https://cdn.sofifa.org/players/4/19/192985.png	Belgium	https://cdn.sofifa.org/flags/7.png	91	92	Manchester City	https://cdn.sofifa

3.3 Listing the columns in the dataframe

- We can list the columns of the dataset using the **columns** attribute.
- This gives a clear-cut picture of the features available in the dataframe.
- It is clear from the below code that the dataset is about football players, containing their height, age, name and many more attributes.

```
df.columns
```

```
Index(['ID', 'Name', 'Age', 'Photo', 'Nationality', 'Flag', 'Overall',
       'Potential', 'Club', 'Club Logo', 'Value', 'Wage', 'Preferred Foot',
       'International Reputation', 'Weak Foot', 'Skill Moves', 'Work Rate',
       'Body Type', 'Position', 'Jersey Number', 'Joined', 'Loaned From',
       'Contract Valid Until', 'Height', 'Weight', 'Crossing', 'Finishing',
       'HeadingAccuracy', 'ShortPassing', 'Volleys', 'Dribbling', 'Curve',
       'FKAccuracy', 'LongPassing', 'BallControl', 'Acceleration',
       'SprintSpeed', 'Agility', 'Reactions', 'Balance', 'ShotPower',
       'Jumping', 'Stamina', 'Strength', 'LongShots', 'Aggression',
       'Interceptions', 'Positioning', 'Vision', 'Penalties', 'Composure',
       'Marking', 'StandingTackle', 'SlidingTackle', 'GKDiving', 'GKHandling',
       'GKKicking', 'GKPositioning', 'GKReflexes', 'Release Clause'],
      dtype='object')
```

3.4 Shape of the DataFrame:

```
df.shape
```

```
(18207, 60)
```

The dataframe has over **18000 observations with 60 attributes/variables**.

3.5 Basic Description of the data:

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18207 entries, 0 to 18206
Data columns (total 60 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   ID                         18207 non-null  int64
 1   Name                       18207 non-null  object
 2   Age                        18207 non-null  int64
 3   Photo                      18207 non-null  object
 4   Nationality                18207 non-null  object
 5   Flag                       18207 non-null  object
 6   Overall                    18207 non-null  int64
 7   Potential                  18207 non-null  int64
 8   Club                       17966 non-null  object
 9   Club Logo                  18207 non-null  object
 10  Value                      18207 non-null  object
 11  Wage                       18207 non-null  object
 12  Preferred Foot             18159 non-null  object
 13  International Reputation   18159 non-null  float64
 14  Weak Foot                  18159 non-null  float64
 15  Skill Moves                18159 non-null  float64
 16  Work Rate                  18159 non-null  object
```

- **info ( )** method contains the number of columns, labels, data types, memory usage, range index, and the number of cells in each column (non-null values).
- With the help of info method, we inferred that data types of the values in the dataset are **numeric (int and float) and categorical (object)**.
- Almost all the columns have **missing values**.
- **'Loaned From'** is the column that has the highest number of missing values.
- The type of the csv file is **Pandas, DataFrame.**

4. DATA PREPROCESSING:

The process of preparing your data and making it more **suitable for feeding** it into the ML model is called pre-processing. Pre- processing includes many activities like **Filling missing values, handling outliers, normalizing and transforming data** and so on.

4. 1. Dropping columns

- It is crucial to drop insignificant columns before analysis the data as it many save a lot of time.
- This also makes it easier to focus and wrangle on the remaining columns.
- Upon analysing the entire dataset, we determined that the columns **"Photo", Flag", "Club Logo", "Jersey Number"** doesn't contribute much on our analysis.
- Hence, they are dropped.

```
data = df.drop(["Photo","Flag","Club Logo", "Jersey Number"],axis=1)
data.head()
```

The shape of the dataFrame after dropping the above columns is:

18207 observations with 56 columns.

```
data.shape
```

```
(18207, 56)
```

5. DATA CLEANING:

Data cleaning is the process of handling missing values, incorrectly formatted data and checking for duplicate data.

HANDLING MISSING VALUES:

- **Deletion & Imputation** are the two techniques used to handle missing values.
- **For numerical variables:** If outliers are present, find the **median** and use it to treat the missing values. If no outliers, use the **mean** on the missing values
- **For Categorical variables:** Fill the missing values with **Mode.**

FORMAT CHECKING:

- Once the missing values are handled, it is crucial we categorize the data according to its actual type, in order to get better results.

1. Column name: 'VALUE'

Objective: Convert "Value" column to **Float** after getting rid of currency symbol and suffix.

- The function '**isnull()**' helps to detect if there is any null values in the dataset. The '**sum()**' function displays the total number of null values.

```
In [ ]:  1  data['Value'].isnull().sum()

Out[ ]: 0
```

- From the above image, we could see that the column 'Value' doesn't have any missing values. Hence, we can move on to type conversion.
- The info ( ) clearly stated the 'Value' column is in the type '**string**'
- We are replacing it to float type by getting rid of the **currency symbol and suffix.**

```
data['Value'].replace({'€':'','K':'*10**3','M':'*10**6'},regex=True,inplace=True)
data['Value']=data['Value'].map(eval)
data['Value'].head()
```

```
0    110500000.0
1     77000000.0
2    118500000.0
3     72000000.0
4    102000000.0
Name: Value, dtype: float64
```

- The observations in the column 'Value' has special characters like **euro as €, millions as M and thousands as K**.
- As there are no null values in this column, we are directly getting rid of the euro symbol and replace M and K with 1000000 and 1000 respectively.
- **Replace ( )** method is used to replace the values in the column and **map ()** is used to map the values appropriately.

```
data['Value'].dtype
```

```
dtype('float64')
```

As the above image shows, the data type of the column 'Value' is now float.

2. **Column name : 'WAGE'**

   Objective: Convert "Wage" column to **Float** after getting rid of currency symbol and suffix.

Similar to 'Value', 'Wage' column is first checked for null values and then we replaced the values in the observation.

```
data['Wage'].isnull().sum()
```

```
0
```

```
data['Wage'].replace({'€':'','K':'*10**3','M':'*10**6'},regex=True,inplace=True)
data['Wage']=data['Wage'].map(eval)
data['Wage']
```

```
0         565000
1         405000
2         290000
3         260000
4         355000
```

As the wage column is in integer type, we are converting it using the **astype()** method.

```
data['Wage'] = data['Wage'].astype(float)
data['Wage'].head()
```

```
0    565000.0
1    405000.0
2    290000.0
3    260000.0
4    355000.0
Name: Wage, dtype: float64
```

```
data['Wage'].dtype
```

```
dtype('float64')
```

As the above image shows, the data type of the column 'Wage is now float.

3. Column Name: 'Joined'

Objective: Convert "Joined" column to int with only year.

- Checking for null values in the joined column and it has been discovered that this column has **1553** null values, which is approximately **8.5%** of data missing from the column.

```
data["Joined"].isnull().sum()
```

```
1553
```

```
100*(data["Joined"].isnull().sum()/len(data.index))
```

```
8.529686384357664
```

In order to handle the missing values, we are imputing it with **mode** as the data type of the 'Joined' column is object.

```
data['Joined'].fillna(data['Joined'].mode()[0],inplace=True)
```

```
data["Joined"].isnull().sum()
```

```
0
```

Above image proves that there are no null values in the ‘Joined’ column.

```
data['Joined']=pd.DatetimeIndex(data['Joined']).year
data['Joined'].head()
```

```
0    1970
1    1970
2    1970
3    1970
4    1970
Name: Joined, dtype: int64
```

```
data['Joined'].dtype
```

```
dtype('int64')
```

- For converting the data type of the ‘Joined’ column to int type, pandas has a special method called **DatetimeIndex**.
- This method converts the column into proper Date, Time format.
- The attribute ‘year’ is used to display only the year.
- The resultant value of the ‘Joined’ column is in integer type.

4. Column Name: "Contract Valid Until"

Objective: ‘Convert "Contract Valid Until" column to datetime type**.**

- Similar to previous conversion, we are checking for missing values in this column and it clearly shows that approximately **1.5%** of the data is missing.

```
data['Contract Valid Until'].isnull().sum()
```

```
289
```

```
100*(data["Contract Valid Until"].isnull().sum()/len(data.index))
```

```
1.5873015873015872
```

In order to handle the missing values, we are imputing it with **mode** as the data type of this column is object.

```
data['Contract Valid Until'].fillna(data['Contract Valid Until'].mode()[0],inplace=True)
```

```
data['Contract Valid Until'].isnull().sum()
```

```
0
```

Above image proves that there are **no null values** in this column.

```
data['Contract Valid Until']= pd.to_datetime(data['Contract Valid Until'])
data['Contract Valid Until'].head()
```

```
0   2021-01-01
1   2022-01-01
2   2022-01-01
3   2020-01-01
4   2023-01-01
Name: Contract Valid Until, dtype: datetime64[ns]
```

- For converting the data type of this column to **datetime** type, pandas has a special method called '**to_datetime**'.
- This method converts the column into proper **Date, Time format**.

5. Columns: "Height" and "Weight"

Objective: Convert "Height" and "Weight" column to Float with decimal points.

```
data['Height'].isnull().sum()
```

```
48
```

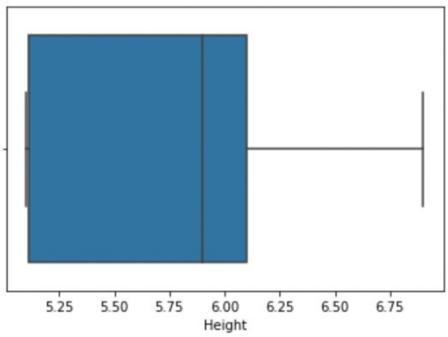
```
data['Weight'].isnull().sum()
```

```
48
```

- The above images show that there are exactly 48 null values in both column.

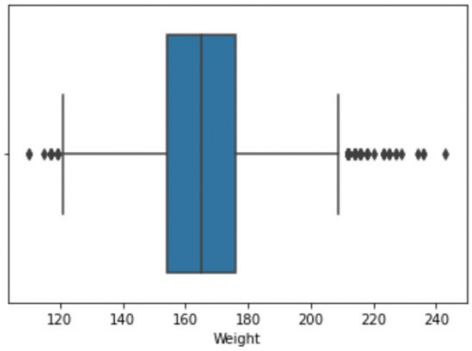
```
1 sns.boxplot(data['Height'])
```

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




```
1 sns.boxplot(data['Weight'])
```

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



- The above box plots concludes that the attribute ‘Height’ has no outliers.
- Thus we are going to do mean imputation on the missing values.
- On the contrary, the box plot of ‘Weight’ column shows that it has a good number of outliers.
- As outliers may affect the mean value, we are going with median imputation.

```
1 Height_mean = data['Height'].mean()
2 Height_mean
```

```
5.797367145768057
```

```
1 data['Height'].fillna(Height_mean,inplace=True)
2 data['Height'].head()
```

```
0    5.70
1    6.20
2    5.90
3    6.40
4    5.11
Name: Height, dtype: float64
```

```
1 data['Height'].isnull().sum()
```

```
0
```

```
1 Weight_median = data['Weight'].median()
2 Weight_median
```

```
165.0
```

```
1 data['Weight'].fillna(Weight_median,inplace=True)
2 data['Weight'].head()
```

```
0    159.0
1    183.0
2    150.0
3    168.0
4    154.0
Name: Weight, dtype: float64
```

```
1 data['Weight'].isnull().sum()
```

```
0
```

The **fillna** ( ) method is used to fill the missing values on both the columns with the specified value.

```
data["Height"] = data["Height"].replace({"'": '.'}, regex=True).astype(float)
data["Height"].head()
```

```
0    5.70
1    6.20
2    5.90
3    6.40
4    5.11
Name: Height, dtype: float64
```

```
data["Height"].dtype
```

```
dtype('float64')
```

```
data["Weight"] = data["Weight"].replace({"lbs": ''}, regex=True).astype(float)
data["Weight"].head()
```

```
0    159.0
1    183.0
2    150.0
3    168.0
4    154.0
Name: Weight, dtype: float64
```

```
data["Weight"].dtype
```

```
dtype('float64')
```

As the height column is in feet and inches, we used to replace function to convert it into decimal format. Similarly for the Weight column, we replaced lbs to decimal format.

6. Column : 'Release Clause'

OBJECTIVE: Convert 'Release Clause' column to Float after getting rid of currency symbol and suffix.

```
data["Release Clause"].isnull().sum()
```

```
1564
```

```
100*(data["Release Clause"].isnull().sum()/len(data.index))
```

```
8.590102707749766
```

- The attribute ‘Release Clause’ has approximately 8.5% of the data missing.
- Before filling the missing values, we are creating a copy of the column and dropping the null values.

```
Dropped_Null_Release_Clause =data['Release Clause'].dropna()
Dropped_Null_Release_Clause.head()
```

```
0    €226.5M
1    €127.1M
2    €228.1M
3    €138.6M
4    €196.4M
Name: Release Clause, dtype: object
```

Outliers are present in Release Clause. Hence, we are going with median imputation.

```
Nonull_Release = Dropped_Null_Release_Clause.replace({'€':'','K':'*10**3','M':'*10**6'},
                                                     regex=True).map(eval).median()
Nonull_Release
```

```
1100000.0
```

```
data['Release Clause'].fillna('1100000.0',inplace=True)
data['Release Clause'].isnull().sum()
```

```
0
```

Finally, replacing object data with float type using the replace method similar to above conversions.

```
data['Release Clause'].replace({'€':'','K':'*10**3','M':'*10**6'},regex=True,inplace=True)
data['Release Clause']=data['Release Clause'].map(eval)
data['Release Clause']
```

```
0        226500000.0
1        127100000.0
2        228100000.0
3        138600000.0
4        196400000.0
```

7. Column - Loaned From

As the column 'Loaned From' has over 93% of the observations missing, we are dropping it as it can be better to drop those cases rather than do imputation and replace them.

```
100* (data['Loaned From'].isnull().sum()/len(data.index))
```

```
93.05761520294392
```

```
data.drop(columns='Loaned From', inplace=True)
data.head()
```

	ID	Name	Age	Nationality	Overall	Potential	Club	Value	Wage	Preferred Foot	...	Composure	Marking	StandingTackle	SlidingTackle
0	158023	L. Messi	31	Argentina	94	94	FC Barcelona	110500000.0	565000.0	Left	...	96.0	33.0	28.0	26.0
1	20801	Cristiano Ronaldo	33	Portugal	94	94	Juventus	77000000.0	405000.0	Right	...	95.0	28.0	31.0	23.0
2	190871	Neymar Jr	26	Brazil	92	93	Paris Saint-Germain	118500000.0	290000.0	Right	...	94.0	27.0	24.0	33.0
3	193080	De Gea	27	Spain	91	93	Manchester United	72000000.0	260000.0	Right	...	68.0	15.0	21.0	13.0
4	192985	K. De Bruyne	27	Belgium	91	92	Manchester City	102000000.0	355000.0	Right	...	88.0	68.0	58.0	51.0

5 rows × 55 columns

8. Other Columns:

- On conducting a self-analysis, we are concluding that majority of the columns has exactly '48 values' missing.
- Hence, we are dropping those values, if and only if, there are all of the same row.

```
data.isnull().sum().sort_values(ascending=False)
```

```
Club                        241
Position                     60
FKAccuracy                   48
SprintSpeed                  48
Volleys                      48
Dribbling                    48
Curve                        48
Vision                       48
LongPassing                  48
BallControl                  48
Acceleration                 48
Agility                      48
HeadingAccuracy              48
Reactions                    48
Balance                      48
ShotPower                    48
Jumping                      48
Stamina                      48
Strength                     48
LongShots                    48
Aggression                   48
ShortPassing                 48
Finishing                    48
Positioning                  48
International Reputation     48
```

Hence, we are dropping the 48 rows with missing values on all the column.

```
1 data.shape
```

```
(18207, 55)
```

```
1 data.dropna(subset=[ 'Stamina'],inplace=True)
```

```
1 data.isnull().sum().sort_values(ascending=False)
```

```
Club                      241
Position                   12
ID                          0
SprintSpeed                 0
Agility                     0
Reactions                   0
Balance                     0
ShotPower                   0
Jumping                     0
Stamina                     0
Strength                    0
LongShots                   0
Aggression                  0
Interceptions               0
Positioning                 0
BallControl                 0
Vision                      0
Penalties                   0
Composure                   0
Marking                     0
StandingTackle              0
SlidingTackle               0
GKDiving                    0
GKHandling                  0
GKKicking                   0
```

On comparing the shape before and after dropping the 48 rows, it proves out hypothesis.

```
1 data.shape
```

```
(18207, 55)
```

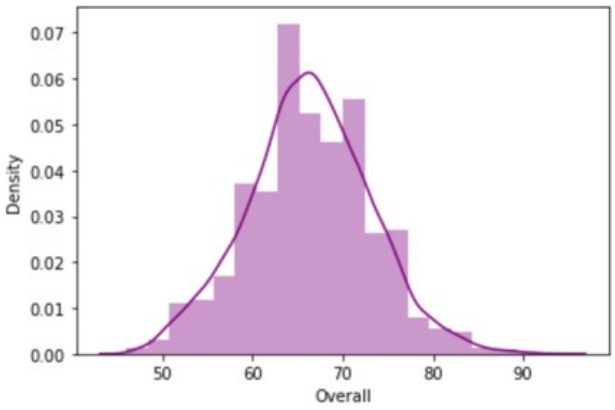
```
1 data.shape
```

```
(18159, 55)
```

6. EXPLORATORY DATA ANALYSIS:

Plot the distribution of Overall rating for all players.

```
sns.distplot(data['Overall'], bins=20, kde=True, color="purple")
plt.show()
```



The plot is more or less normally distributed which is well evident from the shape of the curve (bell shaped) and also the skew value which is 0.083

Moreover, the mean, median and mode values are 66.15,66,66 respectively which also gives a good indication on the normal distribution part.

The Average Overall Score for a Players can be estimated at around 66.

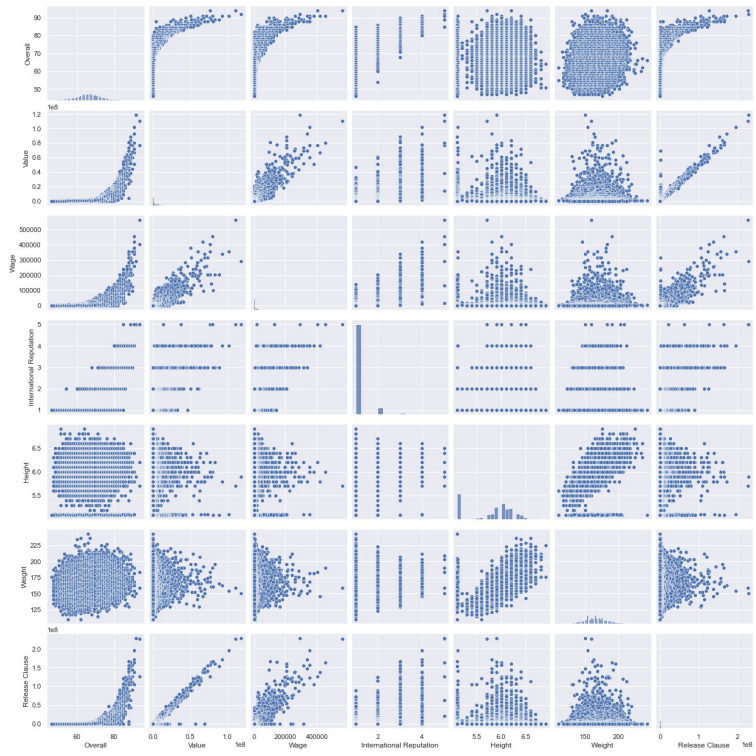
Generate pair plots for the following variables: Overall, Value, Wage, International Reputation, Height, Weight, Release Clause

```
Pair_Plots = data[["Overall", "Value", "Wage", "International Reputation", "Height", "Weight", "Release Clause"]]
Pair_Plots.head()
```

	Overall	Value	Wage	International Reputation	Height	Weight	Release Clause
0	94	110500000.0	565000.0	5.0	5.70	159.0	226500000.0
1	94	77000000.0	405000.0	5.0	6.20	183.0	127100000.0
2	92	118500000.0	290000.0	5.0	5.90	150.0	228100000.0
3	91	72000000.0	260000.0	4.0	6.40	168.0	138600000.0
4	91	102000000.0	355000.0	4.0	5.11	154.0	196400000.0

```
sns.set()
sns.pairplot(Pair_Plots)
plt.show()
```

To generate a pair plot, we first created an object that contains the list of variables for which we need to plot a pair plot. Then using the seaborn library, we plotted a pair plot.



Generate a table containing the top 20 players ranked by Overall score and whose contract expires in 2020.

```
Top_20_Players = data.loc[data["Contract Valid Until"]=="2020"].sort_values(by='Overall',ascending=False).head(20)
Top_20_Players.head()
```

	ID	Name	Age	Nationality	Overall	Potential	Club	Value	Wage	Preferred Foot	...	Composure	Marking	StandingTackle	SlidingTackle	G
3	193080	De Gea	27	Spain	91	93	Manchester United	72000000.0	260000.0	Right	...	68.0	15.0	21.0	13.0	
6	177003	L. Modrić	32	Croatia	91	91	Real Madrid	67000000.0	420000.0	Right	...	84.0	60.0	76.0	73.0	
8	155862	Sergio Ramos	32	Spain	91	91	Real Madrid	51000000.0	380000.0	Right	...	82.0	87.0	92.0	91.0	
5	183277	E. Hazard	27	Belgium	91	91	Chelsea	93000000.0	340000.0	Right	...	91.0	34.0	27.0	22.0	
13	168542	David Silva	32	Spain	90	90	Manchester City	60000000.0	285000.0	Left	...	93.0	59.0	53.0	29.0	

5 rows × 55 columns

Using the loc[] attribute, we grouped the players whose contract will expire in 2020. The above picture shows a sorted dataset, sorted by overall score.

What would the average wage for this set of players be?

The average wage for the above set of players is 205450.

```
Top_20_Players["Wage"].mean()
```

```
205450.0
```

What is the average age?

The average age of the above set of players is 30.65

```
1 Top_20_Players["Age"].mean()
```

```
30.65
```

Is there a correlation between the Overall rating and Value for these players

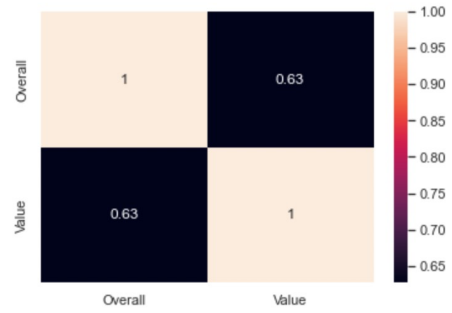
```
1 Corr_Overall_Value = data[["Overall","Value"]]
2 Corr_Overall_Value.corr()
```

	Overall	Value
Overall	1.000000	0.626913
Value	0.626913	1.000000

There is a slight positive correlation between the Overall ratings and Value, since the correlation value is only 0.63.

```
1 sns.heatmap(Corr_Overall_Value.corr(),annot=True)
```

```
<AxesSubplot:>
```



The heatmap give a clear picture of the correlation.

Convert the categorical features to numerical features with suitable encoding techniques

Listing all the categorical columns.

```
data.select_dtypes(object).head()
```

	Name	Nationality	Club	Preferred Foot	Work Rate	Body Type	Position
0	L. Messi	Argentina	FC Barcelona	Left	Medium/ Medium	Messi	RF
1	Cristiano Ronaldo	Portugal	Juventus	Right	High/ Low	C. Ronaldo	ST
2	Neymar Jr	Brazil	Paris Saint-Germain	Right	High/ Medium	Neymar	LW
3	De Gea	Spain	Manchester United	Right	Medium/ Medium	Lean	GK
4	K. De Bruyne	Belgium	Manchester City	Right	High/ High	Normal	RCM

Label Encoding - Column: 'Position'

```
data['Position'].nunique()
```

```
27
```

```
from sklearn.preprocessing import LabelEncoder
labelencoder = LabelEncoder()
Encoded_data = data.copy(deep=True)
Encoded_data['Position'] = labelencoder.fit_transform(Encoded_data['Position'])
Encoded_data.head()
```

As the column Position has 27 unique values and it is a nominal data, we are encoding it using Label encoding technique.

Sklearn library is used for label encoding. An object is created for the label encoder method. We created a copy of the dataframe. Using fit_transform method, we encoded the Position column.

n-1 Encoding - Column: 'Body Type'

```
data['Body Type'].nunique()
```

```
10
```

```
Encoded_data = data.copy(deep=True)
Encoded_data = pd.get_dummies(Encoded_data,columns=['Body Type'], drop_first=True)
Encoded_data.head()
```

Preferred Foot	...	Release Clause	Body Type_C. Ronaldo	Body Type_Courtois	Body Type_Lean	Body Type_Messi	Body Type_Neymar	Body Type_Normal	Body Type_PLAYER_BODY_TYPE_25	Body Type_Shaqiri
Left	...	226500000.0	0	0	0	1	0	0	0	0
Right	...	127100000.0	1	0	0	0	0	0	0	0
Right	...	228100000.0	0	0	0	0	1	0	0	0
Right	...	138600000.0	0	0	1	0	0	0	0	0
Right	...	196400000.0	0	0	0	0	0	1	0	0

Body type column uses the n-1 dummy encoding technique. Get_dummies ( ) method is used for this practise.

Ordinal Encoding - Column: 'Work Rate'

```
data['Work Rate'].unique()
```

```
array(['Medium/ Medium', 'High/ Low', 'High/ Medium', 'High/ High',
       'Medium/ High', 'Medium/ Low', 'Low/ High', 'Low/ Medium',
       'Low/ Low'], dtype=object)
```

```
from sklearn.preprocessing import OrdinalEncoder
ordEncoder = OrdinalEncoder(categories=[['High/ High', 'High/ Medium','High/ Low','Medium/ High',
                                         'Medium/ Medium','Medium/ Low', 'Low/ High', 'Low/ Medium', 'Low/ Low']])
Encoded_data = data.copy(deep=True)
Encoded_data['Ordinal_Work_Rate']= ordEncoder.fit_transform(Encoded_data['Work Rate'].values.reshape(-1,1))
Encoded_data.head()
```

Wage	Preferred Foot	...	Marking	StandingTackle	SlidingTackle	GKDiving	GKHandling	GKKicking	GKPositioning	GKReflexes	Release Clause	Ordinal_Work_Rate
565000.0	Left	...	33.0	28.0	26.0	6.0	11.0	15.0	14.0	8.0	226500000.0	4.0
405000.0	Right	...	28.0	31.0	23.0	7.0	11.0	15.0	14.0	11.0	127100000.0	2.0
290000.0	Right	...	27.0	24.0	33.0	9.0	9.0	15.0	15.0	11.0	228100000.0	1.0
260000.0	Right	...	15.0	21.0	13.0	90.0	85.0	87.0	88.0	94.0	138600000.0	4.0
355000.0	Right	...	68.0	58.0	51.0	15.0	13.0	5.0	10.0	13.0	196400000.0	0.0

Work Rate column follows ordinal encoding.

One Hot Encoding - Column: 'Preferred Foot'

```
data['Preferred Foot'].nunique()
```

```
2
```

```
Encoded_data = data.copy(deep=True)
Encoded_data = pd.get_dummies(Encoded_data,columns=['Preferred Foot'])
Encoded_data.head()
```

alue	Wage	International Reputation	...	StandingTackle	SlidingTackle	GKDiving	GKHandling	GKKicking	GKPositioning	GKReflexes	Release Clause	Preferred Foot_Left	Preferred Foot_Right
00.0	565000.0	5.0	...	28.0	26.0	6.0	11.0	15.0	14.0	8.0	226500000.0	1	0
00.0	405000.0	5.0	...	31.0	23.0	7.0	11.0	15.0	14.0	11.0	127100000.0	0	1
00.0	290000.0	5.0	...	24.0	33.0	9.0	9.0	15.0	15.0	11.0	228100000.0	0	1
00.0	260000.0	4.0	...	21.0	13.0	90.0	85.0	87.0	88.0	94.0	138600000.0	0	1
00.0	355000.0	4.0	...	58.0	51.0	15.0	13.0	5.0	10.0	13.0	196400000.0	0	1

As the Preferred Foot column has 2 unique variables, it is ideal to use one hot encoding technique.

Generate tables containing the top 5 players by Overall rating for each unique position.

We generated a table containing top 5 players grouped by Overall rating for unique position using groupby based on 'Position'. We then applied that on features 'Position' and 'Overall'. Finally we extracted top 5 players from them in ascending order.

```
Top_5_Position = data.groupby(by='Position')['Position','Overall'].head(5)
Top_5_Position.sort_values(['Position','Overall'],ascending=[True,False]).head(10)
```

	Position	Overall
17	CAM	89
31	CAM	88
61	CAM	86
66	CAM	86
74	CAM	86
12	CB	90
42	CB	87
73	CB	86
89	CB	85
102	CB	85

Are there any players appearing in more than one Table. Please point out such players.

```
names = data['Name'].value_counts()
names[names > 1]
```

```
J. Rodríguez    11
Paulinho         8
R. Williams      7
J. Williams      7
J. Valencia      6
                ..
J. Jiménez       2
E. Bajrami       2
L. Maniero       2
B. Moore         2
R. Henry         2
Name: Name, Length: 758, dtype: int64
```

There are 670 player unique name that are appearing in more than once.This is done by applying value_counts on the variable df2['Name'] And then filtering out values more than 1.

What is the average wage one can expect to pay for the top 5 in every position?

First convert the wage to float datatype. Then sort the dataframe df2 by on"'Overall' .Extract top5 from each groups by applying groupby('Position'). Finally apply the mean in top 5 values and extract only the Wage values from it.

```
data = data.sort_values(by='Overall')
top_5 = data.groupby('Position')
top_5.mean()['Wage']
```

```
Position
CAM    10229.645094
CB      7704.724409
CDM     9315.400844
CF     10216.216216
CM      8340.746055
GK      6803.950617
LAM    26142.857143
LB      8726.928896
LCB    11498.456790
LCM    14131.645570
LDM    11860.082305
LF     44666.666667
LM      9656.621005
LS     15260.869565
LW     13068.241470
LWB     9076.923077
RAM    19095.238095
RB      8604.182804
RCB    12688.821752
RCM    14404.092072
RDM    12149.193548
RF     52687.500000
RM      9540.925267
RS     14379.310345
RW     14432.432432
RWB     8597.701149
ST      9938.197026
Name: Wage, dtype: float64
```