**۱- توضیح کد مدل رگرسیون خطی:** برای اینکار کلاسی تحت عنوان LinearRegression در linear_regression.py طراحی شده است. به طوری که داریم:

```
class LinearRegression:
    def __init__(self, train_data, train_labels, test_data, test_labels):
        ones_vector = np.ones((train_data.shape[0], 1))
        self.X = np.hstack((ones_vector, train_data))

        ones_vector = np.ones((test_data.shape[0], 1))
        self.X_test = np.hstack((ones_vector, test_data))

        self.y = train_labels
        self.y_test = test_labels

        self.beta = np.zeros((self.X.shape[1], 1))
```

همانطور که می‌دانیم، مدل رگرسیون خطی معادل است با:

$$output^{(i)} = \beta_0 + \beta_1 x_1^{(i)} + \beta_2 x_2^{(i)} \quad (۱)$$

به عبارت دیگر داریم:

$$\mathbf{output} = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} \\ \vdots & \vdots & \vdots \\ 1 & x_1^{(n)} & x_2^{(n)} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix} \quad (۲)$$

لذا برای دست کردن ماتریس X، در ابتدا یک بردار تمام یک به اندازه تعداد سطرهای داده آموزشی می‌سازیم و سپس بردار $\mathbf{x_1}$ و $\mathbf{x_2}$ را به ستون های آن با np.hstack اضافه میکنیم. به طور مشابه این کار برای X_test نیز انجام میگیرد. سپس، برچسب داده های آموزشی در y و برچسب داده های تست در y_test ذخیره شده و در انتها بردار بتا نیز به صورت تمام صفر ساخته می‌شود.

```
def loss(self, mode='train'):
    if mode == 'train':
        output = self.X @ self.beta
        return np.sum((np.power(output - self.y, 2))) / 2 * output.shape[0]
    else:
        output = self.X_test @ self.beta
        return np.sum((np.power(output - self.y_test, 2))) / 2 * output.shape[0]

def predict(self, mode='train'):
    if mode == 'train':
        return self.X @ self.beta
    else:
        return self.X_test @ self.beta
```

برای تابع خطا نیز یک ورودی mode دریافت می‌شود. در صورتی که معادل با train باشد خطا بر داده آموزشی و در غیر این صورت بر داده تست محاسبه می‌شود. با توجه به این که خطا SSE می‌باشد داریم:

$$SSE = \frac{1}{2n}\sum_{i=1}^{n}(output_i - y_i)^2 \quad (۳)$$

به عبارت برداری داریم:

$$SSE = \frac{1}{2n}(\boldsymbol{output} - \boldsymbol{y})^T(\boldsymbol{output} - \boldsymbol{y}) \quad (۴)$$

که در آن output خروجی مدل ما و y برچسب داده می‌باشد. علت تقسیم بر 2 نیز ساده‌سازی پس از مشتق جزئی و تقسیم بر n نیز کوچکتر شدن حاصل پس از مشتق جزئی می‌باشد سبب می‌شود آهسته تر به سمت مینیمم حرکت کرده و لذا خطا به روش بهینه تر مینیمم گردد.

تابع predict نیز برای دریافت خروجی output از خارج مدل می‌باشد. که اگر mode برابر train باشد داده آموزشی و در غیر این صورت از داده تست استفاده می‌گردد.

```
def reset_beta(self):
    self.beta = np.zeros((self.X.shape[1], 1))
```

همانطور که در ادامه مشاهده می‌کنیم، پس از اجرای gradient descent، وزن ها مقادیر مختلفی می‌گیرند و منطقی نیست که برای شروع stochastic gradient descent همچنان این مقادیر را داشته باشند. لذا از این تابع استفاده می‌شود که وزن ها هنگام شروع مقادیر صفر داشته باشند.

```
def gradient_descent(self, iterations=1000, learning_rate=0.01):
    for iteration in range(iterations):
        output = self.X @ self.beta
        grad_beta = self.X.T @ (output - self.y) / self.X.shape[0]
        self.beta -= learning_rate * grad_beta
```

برای پیاده سازی Gradient Descent به نحوه فوق این کار صورت میگیرد. به طوری که در یک تعداد پیمایش که مقدار پیشفرض ۱۰۰۰ داده شده و از ورودی نیز میتوان دریافت کرد، ابتدا خروجی مدل محاسبه می‌گردد. با توجه به رابطه (۱) و (۳) و مشتق زنجیره ای داریم:

$$\frac{\partial SSE}{\partial \beta_0} = \frac{\partial SSE}{\partial output_i} \times \frac{\partial output_i}{\partial \beta_0} = \frac{output_i - y_i}{n} \quad (۵)$$

$$\frac{\partial SSE}{\partial \beta_1} = \frac{\partial SSE}{\partial output_i} \times \frac{\partial output_i}{\partial \beta_1} = \frac{output_i - y_i}{n}\ {x_1}^{(i)} \quad (۶)$$

$$\frac{\partial SSE}{\partial \beta_2} = \frac{\partial SSE}{\partial output_i} \times \frac{\partial output_i}{\partial \beta_2} = \frac{output_i - y_i}{n}\ {x_2}^{(i)} \quad (۷)$$

در کد جهت تسهیل در سرعت اجرا، به سرعت Vectorized این کار صورت گرفته است که محاسبات به صورت موازی انجام بگیرند. گویی که داریم:

$$\frac{\partial SSE}{\partial \boldsymbol{\beta}} = \frac{\partial SSE}{\partial \boldsymbol{output}} \times \frac{\partial \boldsymbol{output}}{\partial \boldsymbol{\beta}} = \mathrm{X}^{\boldsymbol{T}} \frac{\boldsymbol{output} - \boldsymbol{y}}{n} \quad (۸)$$

لذا در هر پیمایش ابتدا خروجی محاسبه می‌شود. سپس به کمک رابطه (۸) گرادیان تابع خطا نسبت به بردار بتا محاسبه و بردار بتا بروز میشود. برای بروز رسانی از اندازه قدم یا learning rate ۰.۰۱ استفاده شده است. لازم به ذکر است که مقادیر ۰.۰۱ و ۱۰۰۰ به صورت آزمایشی با آزمون و خطا بدست آمده اند.

```python
def stochastic_gradient_descent(self, iterations=1000, learning_rate=0.01, batch_size=200):
    Xy = np.hstack((self.X, self.y))
    for iteration in range(iterations):
        for start_index in range(0, self.X.shape[0], batch_size):
            stop_index = start_index + batch_size
            X_batch, y_batch = Xy[start_index:stop_index, :-1], Xy[start_index:stop_index, -1:]

            output = X_batch @ self.beta
            grad_beta = X_batch.T @ (output - y_batch) / batch_size
            self.beta -= learning_rate * grad_beta
```

پیاده سازی stochastic gradient descent نیز مشابه قبل است. با این تفاوت که با توجه به سایت realpython، در هر پیمایش گرادیان به جای کل تابع یک پیمایش دیگری به اندازه batch_size / size انجام میشود و هربار به جای کل داده روی batch_size تا داده این کار صورت میگیرد. برای این کار همانطور که مشاهده می‌شود ابتدا برچسب داده ها به خود داده اضافه و ماتریس Xy تشکیل می‌شود. سپس در هر پیمایش batch این دو از ماتریس Xy استخراج و محاسبه روی آن صورت می‌گیرد.

**اجرای موارد خواسته شده در سوال:**

برای این کار از Jupyter notebook استفاده شده است. در فایل Q1.ipynb داریم:

**Read data**

```
In [2]: data = np.load('data.npz')
```

**get X, y, X_test, and y_test**

```
In [3]: x1 = data['x1'].reshape(-1, 1)
        x2 = data['x2'].reshape(-1, 1)
        X = np.hstack((x1, x2))

        x1_test = data['x1_test'].reshape(-1, 1)
        x2_test = data['x2_test'].reshape(-1, 1)
        X_test = np.hstack((x1_test, x2_test))

        y = data['y'].reshape(-1, 1)
        y_test = data['y_test'].reshape(-1, 1)
```

**Initializing our model**

```
In [4]: model = LinearRegression(X, y, X_test, y_test)
```

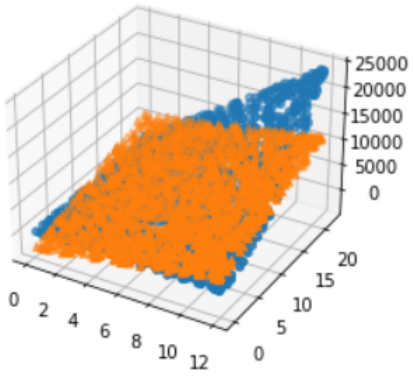
همانطور که از عنوان هر cell پیداست، ابتدا داده های مورد نظر خوانده و سپس X و X_test و برچسب های متناظر استخراج میشود. لازم به ذکر است که ماتریس های X و X_test در اینجا تنها حاوی داده ها می باشند و بردار ۱ در ابتدا را ندارند. سپس مدل مورد نظر ما با توجه به کد توضیح داده شده ساخته می‌شود.

### Using Gradient Descent

```
In [5]: model.gradient_descent()

        gd_predicted_y = model.predict(mode='test')

        gd_train_loss = model.loss(mode='train')
        gd_test_loss = model.loss(mode='test')

        gd_train_loss, gd_test_loss
```

```
Out[5]: (78344996283152.6, 15058939105016.86)
```

```
In [6]: fig = plt.figure()
        ax = fig.add_subplot(projection='3d')
        ax.scatter(x1_test, x2_test, y_test, marker='o')
        ax.scatter(x1_test, x2_test, gd_predicted_y, marker='^');
```




هنگام استفاده از Gradient Descent، مشاهده می‌شود که میزان خطا در train به مراتب بیشتر از test است. علت آن نیز این است که تعداد داده های آموزشی چهار برابر داده تست می‌باشد. لذا با توجه به رابطه (۳) هنگام محاسبه خطا فاصله Residual های بیشتری با یکدیگر جمع شده و به طبع مقدار بیشتری نیز اختیار می‌کند.

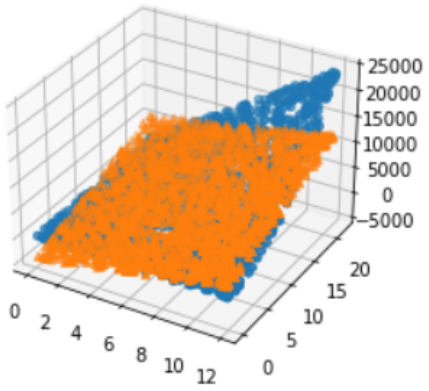
پس از رسم نمودار به صورت سه بعدی نیز مشاهده می‌شود که داده آموزشی یک منحنی مشابه یک صفحه کج شده می‌باشد (نقاط آبی رنگ). اما با توجه به استفاده از رگرسیون خطی، حاصل مدل ما یک صفحه می‌باشد که سعی شده است تا حد امکان بر این منحنی تطبیق شود (نقاط نارنجی رنگ).

## Using Stochastic Gradient Descent

In [7]:
```
# reset previous weights
model.reset_beta()

model.stochastic_gradient_descent()

sgd_predicted_y = model.predict(mode='test')

sgd_train_loss = model.loss(mode='train')
sgd_test_loss = model.loss(mode='test')

sgd_train_loss, sgd_test_loss
```

Out[7]: (73506803886759.47, 13260649293648.326)

In [8]:
```
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
ax.scatter(x1_test, x2_test, y_test, marker='o')
ax.scatter(x1_test, x2_test, sgd_predicted_y, marker='^');
```




هنگام استفاده از Stochastic Gradient Descent نیز مشاهده می‌شود نتایج تا حدودی بهتر از Gradient Descent می‌باشد. چراکه با انجام این کار، به صورت یک خط راست به سمت مینیمم حرکت نمی‌کنیم و با حرکت کج و معوج به سوی آن می‌رویم. لذا امکان فرار کردن از مینیمم محلی را به ما می‌دهد و در نتیجه نیز این امر قابل مشاهده است.

نمودار سه بعدی نیز مشابه قبل یک صفحه نارنجی رنگی است که تا حد امکان میخواهد بر روی منحنی آبی رنگ منطبق گردد.

## ۲- موارد خواسته شده:

**a.** به کمک کتابخانه پانداس و تابع read_csv میتوانیم فایل csv حاوی ۱۹۰۲۰ ردیف و ۹۰ ستون را تبدیل به یک دیتافریم و مشاهده کنیم:

### Read data

```
In [2]: df = pd.read_csv('players.csv')
        df
```

Out[2]:

| | ID | Name | FullName | Age | Height | Weight | PhotoUrl | Nationality | Overall | Potential | ... | LMRating | CMRating |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 158023 | L. Messi | Lionel Messi | 33 | 170 | 72 | https://cdn.sofifa.com/players/158/023/21_60.png | Argentina | 93 | 93 | ... | 93 | 90 |
| **1** | 20801 | Cristiano Ronaldo | C. Ronaldo dos Santos Aveiro | 35 | 187 | 83 | https://cdn.sofifa.com/players/020/801/21_60.png | Portugal | 92 | 92 | ... | 91 | 84 |
| **2** | 200389 | J. Oblak | Jan Oblak | 27 | 188 | 87 | https://cdn.sofifa.com/players/200/389/21_60.png | Slovenia | 91 | 93 | ... | 38 | 41 |
| **3** | 192985 | K. De Bruyne | Kevin De Bruyne | 29 | 181 | 70 | https://cdn.sofifa.com/players/192/985/21_60.png | Belgium | 91 | 91 | ... | 91 | 91 |
| **4** | 190871 | Neymar Jr | Neymar da Silva Santos Jr. | 28 | 175 | 68 | https://cdn.sofifa.com/players/190/871/21_60.png | Brazil | 91 | 91 | ... | 91 | 86 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **19015** | 257371 | M. Nzongong | Mike Nzongong | 19 | 179 | 74 | https://cdn.sofifa.com/players/257/371/21_60.png | Republic of Ireland | 49 | 60 | ... | 51 | 47 |
| **19016** | 259160 | L. Bell | Lewis Bell | 17 | 181 | 70 | https://cdn.sofifa.com/players/259/160/21_60.png | England | 49 | 67 | ... | 51 | 44 |
| **19017** | 259157 | Y. Arai | Yasin Arai | 16 | 176 | 70 | https://cdn.sofifa.com/players/259/157/21_60.png | England | 49 | 74 | ... | 53 | 47 |
| **19018** | 253763 | R. Dinanga | Ricardo Dinanga | 18 | 174 | 73 | https://cdn.sofifa.com/players/253/763/21_60.png | Republic of Ireland | 49 | 59 | ... | 49 | 43 |
| **19019** | 241493 | S. Cartwright | Samuel Cartwright | 19 | 185 | 75 | https://cdn.sofifa.com/players/241/493/21_60.png | England | 49 | 65 | ... | 38 | 36 |

19020 rows × 90 columns

برای مشاهده ردیف های ابتدایی میتوانیم از تابع head استفاده کنیم که ۵ ردیف ابتدایی را برمیگرداند. در صورت استفاده از پارامتر میتوانیم این عدد را به تعداد دلخواه عوض کنیم:

### a. showing first/last 5 rows

```
In [3]: df.head()
```

Out[3]:

| | ID | Name | FullName | Age | Height | Weight | PhotoUrl | Nationality | Overall | Potential | ... | LMRating | CMRating | RMRa |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 158023 | L. Messi | Lionel Messi | 33 | 170 | 72 | https://cdn.sofifa.com/players/158/023/21_60.png | Argentina | 93 | 93 | ... | 93 | 90 | |
| **1** | 20801 | Cristiano Ronaldo | C. Ronaldo dos Santos Aveiro | 35 | 187 | 83 | https://cdn.sofifa.com/players/020/801/21_60.png | Portugal | 92 | 92 | ... | 91 | 84 | |
| **2** | 200389 | J. Oblak | Jan Oblak | 27 | 188 | 87 | https://cdn.sofifa.com/players/200/389/21_60.png | Slovenia | 91 | 93 | ... | 38 | 41 | |
| **3** | 192985 | K. De Bruyne | Kevin De Bruyne | 29 | 181 | 70 | https://cdn.sofifa.com/players/192/985/21_60.png | Belgium | 91 | 91 | ... | 91 | 91 | |
| **4** | 190871 | Neymar Jr | Neymar da Silva Santos Jr. | 28 | 175 | 68 | https://cdn.sofifa.com/players/190/871/21_60.png | Brazil | 91 | 91 | ... | 91 | 86 | |

5 rows × 90 columns

جهت مشاهده ردیف های انتها نیز از تابعی مشابه با head تحت عنوان tail استفاده می‌کنیم:

```
In [4]: df.tail()
```

Out[4]:

| | ID | Name | FullName | Age | Height | Weight | PhotoUrl | Nationality | Overall | Potential | ... | LMRating | CMRating |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **19015** | 257371 | M. Nzongong | Mike Nzongong | 19 | 179 | 74 | https://cdn.sofifa.com/players/257/371/21_60.png | Republic of Ireland | 49 | 60 | ... | 51 | 47 |
| **19016** | 259160 | L. Bell | Lewis Bell | 17 | 181 | 70 | https://cdn.sofifa.com/players/259/160/21_60.png | England | 49 | 67 | ... | 51 | 44 |
| **19017** | 259157 | Y. Arai | Yasin Arai | 16 | 176 | 70 | https://cdn.sofifa.com/players/259/157/21_60.png | England | 49 | 74 | ... | 53 | 47 |
| **19018** | 253763 | R. Dinanga | Ricardo Dinanga | 18 | 174 | 73 | https://cdn.sofifa.com/players/253/763/21_60.png | Republic of Ireland | 49 | 59 | ... | 49 | 43 |
| **19019** | 241493 | S. Cartwright | Samuel Cartwright | 19 | 185 | 75 | https://cdn.sofifa.com/players/241/493/21_60.png | England | 49 | 65 | ... | 38 | 36 |

5 rows × 90 columns

**b**. با توجه به این صفحه از سایت stackoverflow، ابتدا به کمک تابع isnull هر درایه که پر نشده باشد False و شده باشدTrue میشود. سپس به کمک any(axis=1). میگوییم در هر axis=1 که ردیف می‌باشد اگر حداقل یک True باشد نتیجه برای آن ردیف را True برگرداند. سپس با قرار دادن خروجی این نتیجه به ورودی دیتافریم اصلی ردیف هایی را میتوانیم دریافت کنیم که حداقل یک missing values دارند که ۱۸۱۱۸ ردیف می‌باشند:

```
In [5]: df[df.isnull().any(axis=1)]
```

Out[5]:

| | ID | Name | FullName | Age | Height | Weight | PhotoUrl | Nationality | Overall | Potential | ... | LMRating | CMRating |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **4** | 190871 | Neymar Jr | Neymar da Silva Santos Jr. | 28 | 175 | 68 | https://cdn.sofifa.com/players/190/871/21_60.png | Brazil | 91 | 91 | ... | 91 | 86 |
| **6** | 208722 | S. Mané | Sadio Mané | 28 | 175 | 69 | https://cdn.sofifa.com/players/208/722/21_60.png | Senegal | 90 | 90 | ... | 90 | 84 |
| **11** | 212831 | Alisson | Alisson Ramses Becker | 27 | 191 | 91 | https://cdn.sofifa.com/players/212/831/21_60.png | Brazil | 90 | 91 | ... | 41 | 43 |
| **14** | 200145 | Casemiro | Carlos Henrique Venancio Casimiro | 28 | 185 | 84 | https://cdn.sofifa.com/players/200/145/21_60.png | Brazil | 89 | 89 | ... | 77 | 84 |
| **16** | 165153 | K. Benzema | Karim Benzema | 32 | 185 | 81 | https://cdn.sofifa.com/players/165/153/21_60.png | France | 89 | 89 | ... | 87 | 83 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **19015** | 257371 | M. Nzongong | Mike Nzongong | 19 | 179 | 74 | https://cdn.sofifa.com/players/257/371/21_60.png | Republic of Ireland | 49 | 60 | ... | 51 | 47 |
| **19016** | 259160 | L. Bell | Lewis Bell | 17 | 181 | 70 | https://cdn.sofifa.com/players/259/160/21_60.png | England | 49 | 67 | ... | 51 | 44 |
| **19017** | 259157 | Y. Arai | Yasin Arai | 16 | 176 | 70 | https://cdn.sofifa.com/players/259/157/21_60.png | England | 49 | 74 | ... | 53 | 47 |
| **19018** | 253763 | R. Dinanga | Ricardo Dinanga | 18 | 174 | 73 | https://cdn.sofifa.com/players/253/763/21_60.png | Republic of Ireland | 49 | 59 | ... | 49 | 43 |
| **19019** | 241493 | S. Cartwright | Samuel Cartwright | 19 | 185 | 75 | https://cdn.sofifa.com/players/241/493/21_60.png | England | 49 | 65 | ... | 38 | 36 |

18118 rows × 90 columns

برای این که بفهمیم این مقادیر گم شده متعلق به چه ستون هایی می‌باشند و هرکدام چند مقدار گم شده دارند، میتوانیم پس از isnull مقدار های هر ستون را به کمک sum(). جمع بزنیم که خروجی یک Series می باشد. سپس میگوییم تنها آن هایی که sum() بیشتر از صفر دارند برگردانده شوند و داریم:

```
In [6]: df.isnull().sum()[df.isna().sum() > 0]

Out[6]: ClubPosition          234
        ContractUntil         234
        ClubNumber            234
        NationalPosition    17895
        NationalNumber      17895
        dtype: int64
```

**c.** به کمک توابع پیشفرض پانداس به راحتی این کار قابل انجام است.

## c. Mean, min and max of player's weight

```
In [7]: df['Weight'].mean()

Out[7]: 75.05241850683491

In [8]: df['Weight'].min()

Out[8]: 50

In [9]: df['Weight'].max()

Out[9]: 110
```

**d.** با توجه به stackoverflow، ابتدا روی ستون Nationality تابع value_counts فراخوانی می‌شود که نشان بدهد از هر ملیت چندتا داریم. سپس به کمک تابع min، مینیمم این تعداد را دریافت میکنیم. بار دیگر روی خروجی تابع values_counts آن هایی که تعدادشان برابر با مینیمم هستند را استخراج میکنیم که کشور های زیر بدست می‌آیند:

```
In [10]: min_count = df['Nationality'].value_counts().min()
         df['Nationality'].value_counts()[df['Nationality'].value_counts() == min_count]
```

```
Out[10]: Indonesia              1
         Korea DPR              1
         Saint Lucia            1
         Hong Kong              1
         Malta                  1
         Singapore              1
         South Sudan            1
         Guatemala              1
         Barbados               1
         Chad                   1
         Suriname               1
         Papua New Guinea       1
         Andorra                1
         Rwanda                 1
         Guam                   1
         Bermuda                1
         Aruba                  1
         Tanzania               1
         Malawi                 1
         São Tomé & Príncipe    1
         New Caledonia          1
         Puerto Rico            1
         Malaysia               1
         Name: Nationality, dtype: int64
```

مشابه همین کار را برای ماکسیمم نیز میتوان انجام داد:

```
In [11]: max_count = df['Nationality'].value_counts().max()
         df['Nationality'].value_counts()[df['Nationality'].value_counts() == max_count]
```

```
Out[11]: England    1706
         Name: Nationality, dtype: int64
```

**e.** به کمک عملگرد & میتوان این کار را انجام داد:

## e. Finding players who have Growth < 3 and Potential < 84

```
In [12]: df[(df['Growth'] < 3) & (df['Potential'] < 84)]
```

Out[12]:

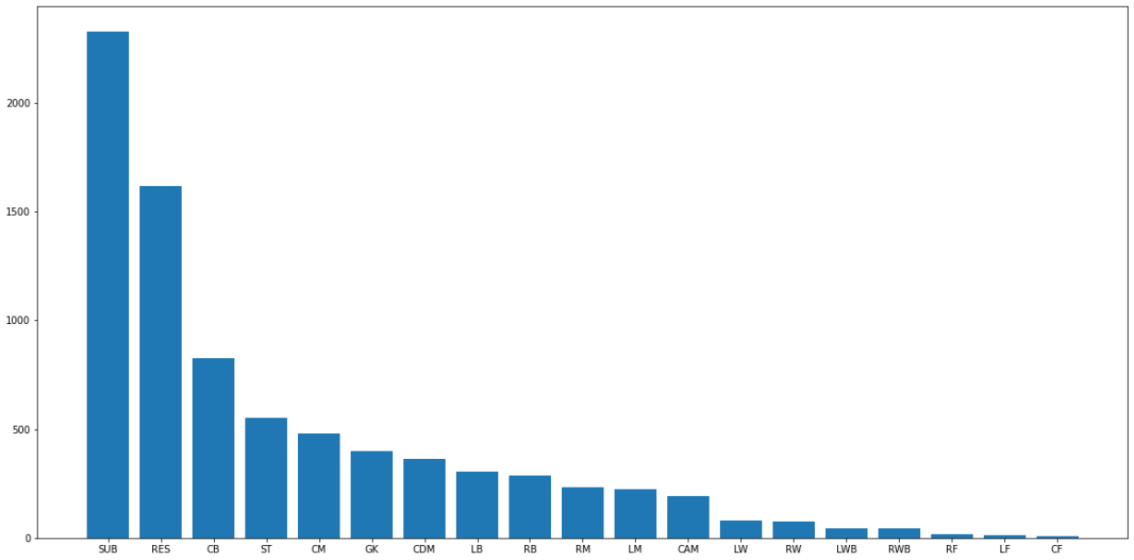
| | ID | Name | FullName | Age | Height | Weight | PhotoUrl | Nationality | Overall | Potential | ... | LMRating | CMRating |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **139** | 169195 | Renato Augusto | Renato Augusto | 32 | 186 | 86 | https://cdn.sofifa.com/players/169/195/21_60.png | Brazil | 83 | 83 | ... | 82 | 83 |
| **140** | 169416 | C. Vela | Carlos Vela | 31 | 177 | 77 | https://cdn.sofifa.com/players/169/416/21_60.png | Mexico | 83 | 83 | ... | 83 | 78 |
| **143** | 215333 | D. Zapata | Duván Zapata | 29 | 189 | 88 | https://cdn.sofifa.com/players/215/333/21_60.png | Colombia | 83 | 83 | ... | 75 | 68 |
| **144** | 216547 | Rafa | Rafael A. Ferreira Silva | 27 | 172 | 66 | https://cdn.sofifa.com/players/216/547/21_60.png | Portugal | 83 | 83 | ... | 83 | 77 |
| **147** | 220407 | M. Dúbravka | Martin Dúbravka | 31 | 190 | 80 | https://cdn.sofifa.com/players/220/407/21_60.png | Slovakia | 83 | 83 | ... | 37 | 41 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **18684** | 253744 | Xu Wu | Wu Xu | 27 | 184 | 78 | https://cdn.sofifa.com/players/253/744/21_60.png | China PR | 52 | 54 | ... | 49 | 50 |
| **18742** | 256758 | O. Alıcı | Ömer Alıcı | 34 | 185 | 83 | https://cdn.sofifa.com/players/256/758/21_60.png | Turkey | 51 | 51 | ... | 18 | 20 |
| **18888** | 157190 | J. Russell | John Russell | 35 | 172 | 70 | https://cdn.sofifa.com/players/157/190/21_60.png | Republic of Ireland | 50 | 50 | ... | 49 | 50 |
| **18968** | 243073 | Teng Shangkun | Shangkun Teng | 29 | 186 | 78 | https://cdn.sofifa.com/players/243/073/21_60.png | China PR | 50 | 51 | ... | 23 | 25 |
| **18986** | 233720 | Li Hao | Hao Li | 28 | 173 | 63 | https://cdn.sofifa.com/players/233/720/21_60.png | China PR | 50 | 51 | ... | 50 | 48 |

8261 rows × 90 columns

**f.** ابتدا ClubPosition این افراد بدست می آوریم و به کمک value_counts مشابه بخش d، تعداد هر موقعیت را استخراج میکنیم. در انتها به کمک matplotlib یک bar chart رسم میکنیم که محور افقی برابر با series key و محور عمودی مقادیر آن به صورت لیست است:

### f. Bar chart of previous section based on Club position

In [13]:
```
extracted_players_positions = df[(df['Growth'] < 3) & (df['Potential'] < 84)]['ClubPosition']
positions_count = extracted_players_positions.value_counts()

fig = plt.figure(figsize=(20, 10))
ax = fig.add_subplot()

ax.bar(positions_count.keys(), positions_count.to_list());
```




**g.** با توجه به این که بازیکن خلاف بازیکن f معادل با بازیکن آینده دار است، ابتدا بازیکن های آینده دار را استخراج میکنیم. سپس در ستون Club به کمک value_counts تعداد هر club را ارزیابی میکنیم و ماکسیمم آن را بدست می آوریم. سپس Club ای را بر میگردانیم که value_counts آن برابر با ماکسیمم باشد:

### g. Club that has the most promising players and their numbers

In [14]:
```
promising_players = df[(df.Growth > 4) & (df.Potential > 84)]

max_count = promising_players['Club'].value_counts().max()
promising_players['Club'].value_counts()[promising_players['Club'].value_counts() == max_count]
```

Out[14]:
```
Sporting CP    10
Name: Club, dtype: int64
```

**.h**

### h. Number of players whom their contract is until 2021 and are not in national team

```
In [15]: len(df[(df['ContractUntil'] == 2021) & (df['NationalTeam'] == 'Not in team')])
Out[15]: 6727
```

**i.** با توجه به این که در صورت سوال مشخص نشده است چه چیزی گزارش شود، به کمک تابع describe تمامی مقادیر نیاز را میتوانیم مشاهده کنیم:

### i. Report of ValueEUR for players whom their age is below 24 and are in Chelsea club

```
In [16]: df[(df['Age'] < 24) & (df['Club'] == 'Chelsea')][['ValueEUR']].describe()
```

Out[16]:

| | ValueEUR |
|---|---|
| count | 1.200000e+01 |
| mean | 2.800000e+07 |
| std | 3.315794e+07 |
| min | 5.000000e+05 |
| 25% | 4.700000e+06 |
| 50% | 2.225000e+07 |
| 75% | 3.625000e+07 |
| max | 1.210000e+08 |

**.j**

### j. Positions, WageEUR and Club report of E. Hazard

```
In [17]: df[df['Name'] == 'E. Hazard'][['Name','FullName', 'ClubPosition', 'WageEUR', 'Club']]
```

Out[17]:

| | Name | FullName | ClubPosition | WageEUR | Club |
|---|---|---|---|---|---|
| 22 | E. Hazard | Eden Hazard | SUB | 350000 | Real Madrid |