Table of contents

OBJECTIVE

this dataset includes FIFA 2019 players attributes like Finishing, Heading, Accuracy, ShortPassing, Volleys, Dribbling, Curve, FKAccuracy, LongPassing, BallControl, Acceleration, SprintSpeed, Agility, Reactions, Balance, ShotPower and many more. The daset has 87 columns and over 18000 rows. The attempt in this analysis is to find how finishing is related to other attributes of the players. Finishing is where you are working up close to the goal and most of the chances are from balls crossed (passed) in. Finishing in plain and simple terms, you finish (score a goal).

PACKAGES

In [105]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

DATA STRUCTURE

In [106]:
```
df = pd.read_csv(r'D:\fifa.csv')
```

As can be seen below the dataset is quite large in volume and contains many columns that are irrelevant to our analysis. As I proceed further, I will drop them on detection.

In [107]:
```
df.head()
```

Out[107]:

	Unnamed: 0	ID	Name	Age	Wage	Overall	Potential	Club	
0	0	158023	L. Messi	31	€565K	94	94	FC Barcelona	https://cdn.sofifa.org/te
1	1	20801	Cristiano Ronaldo	33	€405K	94	94	Juventus	https://cdn.sofifa.org/t
2	2	190871	Neymar Jr	26	€290K	92	93	Paris Saint-Germain	https://cdn.sofifa.org/t
3	3	193080	De Gea	27	€260K	91	93	Manchester United	https://cdn.sofifa.org/t
4	4	192985	K. De Bruyne	27	€355K	91	92	Manchester City	https://cdn.sofifa.org/t

5 rows × 87 columns

DATA PREPROCESSING

Firstly, I drop the rows having null values.

In [108]:
```
df.isnull().sum()
```

Out[108]:
```
Unnamed: 0          0
ID                  0
Name                0
Age                 0
Wage                0
                 ...
GKHandling         48
GKKicking          48
GKPositioning      48
GKReflexes         48
Release Clause   1564
Length: 87, dtype: int64
```

In [109]:
```
df.dropna()
```

Out[109]:

	Unnamed: 0	ID	Name	Age	Wage	Overall	Potential	Club	Club Logo	Value	...	Composure	Marl

0 rows × 87 columns

In [110]:
```
print(df.columns)
```

```
Index(['Unnamed: 0', 'ID', 'Name', 'Age', 'Wage', 'Overall', 'Potential',
       'Club', 'Club Logo', 'Value', 'Unnamed: 10', 'Special',
       'Preferred Foot', 'International Reputation', 'Weak Foot',
       'Skill Moves', 'Work Rate', 'Body Type', 'Real Face', 'Position',
       'Jersey Number', 'Joined', 'Loaned From', 'Contract Valid Until',
       'Height', 'Weight', 'LS', 'ST', 'RS', 'LW', 'LF', 'CF', 'RF', 'RW',
       'LAM', 'CAM', 'RAM', 'LM', 'LCM', 'CM', 'RCM', 'RM', 'LWB', 'LDM',
       'CDM', 'RDM', 'RWB', 'LB', 'LCB', 'CB', 'RCB', 'RB', 'Crossing',
       'Finishing', 'HeadingAccuracy', 'ShortPassing', 'Volleys', 'Dribblin
g',
       'Curve', 'FKAccuracy', 'LongPassing', 'BallControl', 'Acceleration',
       'SprintSpeed', 'Agility', 'Reactions', 'Balance', 'ShotPower',
       'Jumping', 'Stamina', 'Strength', 'LongShots', 'Aggression',
       'Interceptions', 'Positioning', 'Vision', 'Penalties', 'Composure',
       'Marking', 'StandingTackle', 'SlidingTackle', 'GKDiving', 'GKHandlin
g',
       'GKKicking', 'GKPositioning', 'GKReflexes', 'Release Clause'],
      dtype='object')
```

As is evident from above that not all the variables go into determining the finishing potential of a player, hence irrelevant. As per common knowledge and prior research, the finishing of a player can potentially depend on his attributes like Strength, Stamina, Acceleration, Sprint speed, Agility etc. SO, here I will explain the importance of each such variable that will form a part of independent variables in this analysis.

1. Acceleration - is what helps players get past their opponents quickly. Even if they aren't as fast as the player they are trying to beat at full speed, if they accelerate faster then they will be able to beat that player.
2. SprintSpeed - describes how fast a player can run over a certain distance. However, it is also used to express something quick, explosive and abrupt in action.
3. Agility - is proposed as a rapid whole body movement with change of velocity or direction in response to a stimulus. By working on agility and improving the balance and coordination, soccer players will be able to move faster and change directions more quickly while maintaining control
4. Reactions - Reaction time is duration between applications of a stimulus to onset of response. A person can have great reaction time, but it doesn't help if his or her body can't do anything about the stimulus.
5. Balance - refers to the tactical balance in the team's playing system in each positional area. It is important for a team to achieve a good playing system balance in order for it to effectively achieve each of its tactical objectives in each positional area.
6. ShotPower - is hitting the ball in an attempt to score a goal.
7. Jumping - Jumping is the player's ability and quality for jumping from the surface for headers. The higher the value is, the higher the player can jump.
8. Stamina - Stamina determines the rate at which a player will tire during a game. It evaluates how tired your player gets as the match approaches half time or full time.
9. Strength - Strength is about the quality or state of being physically strong. The higher the value, the more likely the player will win a physical challenge.
10. LongShots - This attribute measures the accuracy of shots from outside the penalty area.This is a great attribute for midfielders to have.
11. Aggression - The aggression level of a player measures the frequency and the aggression of jostling, tackling and slide tackling. It is the attribute which determines the player's power of will or commitment to a match.

After looking at the definition of independent variables, it has become quite evident why they are relevant in determinig the target variable. Considering the same, I am dropping rest of the irrelevant columns as it's also hindering visualization.

In [111]:
```
df.drop(['ID', 'Name', 'Age', 'Overall', 'Potential', 'Club', 'Club Logo', 'Va
lue', 'Wage', 'Special',
       'Preferred Foot', 'International Reputation', 'Weak Foot',
       'Skill Moves', 'Work Rate', 'Body Type', 'Real Face', 'Position',
       'Jersey Number', 'Joined', 'Loaned From', 'Contract Valid Until',
       'Height', 'Weight', 'LS', 'ST', 'RS', 'LW', 'LF', 'CF', 'RF', 'RW',
       'LAM', 'CAM', 'RAM', 'LM', 'LCM', 'CM', 'RCM', 'RM', 'LWB', 'LDM',
       'CDM', 'RDM', 'RWB', 'LB', 'LCB', 'CB', 'RCB', 'RB', 'Crossing','Headin
gAccuracy', 'ShortPassing', 'Volleys', 'Dribbling',
       'Curve', 'FKAccuracy', 'LongPassing', 'BallControl', 'Interceptions',
'Positioning', 'Vision', 'Penalties', 'Composure',
       'Marking', 'StandingTackle', 'SlidingTackle', 'GKDiving', 'GKHandling',
       'GKKicking', 'GKPositioning', 'GKReflexes', 'Release Clause'], axis=1,
inplace=True)
```

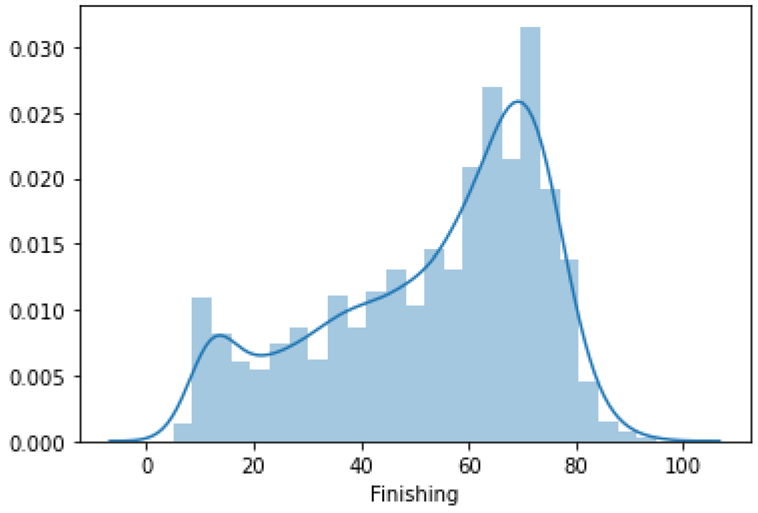
The volume is so large that there is definitely going to occur error when a regression will be run. so, I have deciced to consider only first 5000 rows which is quite sufficient for carrying out analysis. An eyeball over the data tells us that the variation in values of data remains same overall.

```
In [112]: df = df.iloc[0:5000, :]
```

A further plotting of one of the variables under consideration tells about the existence of outliers within the dataframe. Below, it can be seen Finishing has no considerable outliers. So, I will be moving to the next variable.

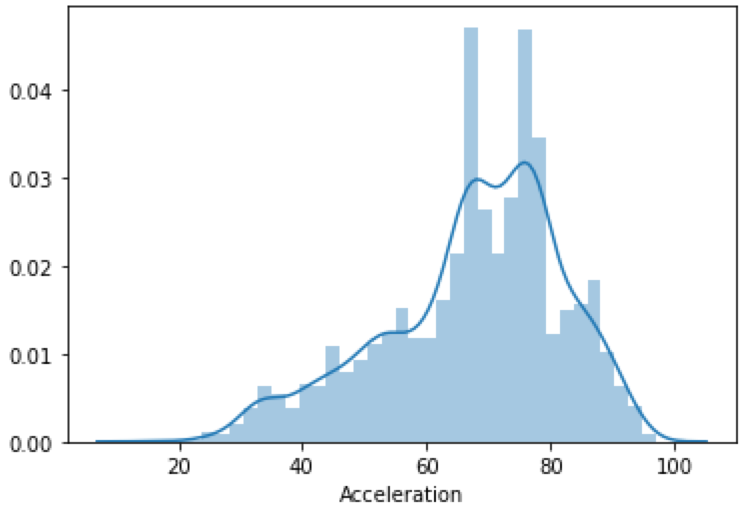
```
In [113]: import seaborn as sns
          sns.distplot(df['Finishing'])

Out[113]: <matplotlib.axes._subplots.AxesSubplot at 0x14f7d804850>
```



```
In [114]: sns.distplot(df['Acceleration'])

Out[114]: <matplotlib.axes._subplots.AxesSubplot at 0x14f004804f0>
```



However, distribution of acceleration shows that towards the left end, there are outliers that might affect our results drastically. So, 5% of data from both the ends is being dropped. Below is my new dataset.

In [115]:
```
#to remove outliers
q = df['Acceleration'].quantile(0.95)
df = df[df['Acceleration']<q]
df.describe(include='all')
```

Out[115]:

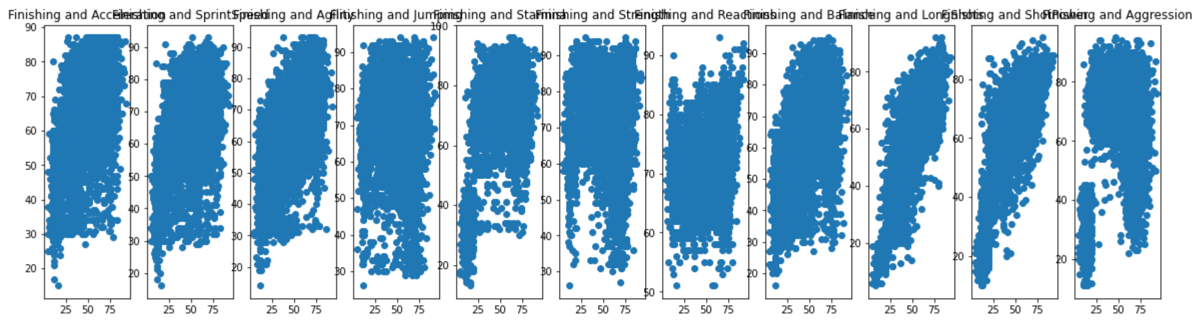
	Unnamed: 0	Unnamed: 10	Finishing	Acceleration	SprintSpeed	Agility	Reaction
count	4703.000000	0.0	4703.000000	4703.000000	4703.000000	4703.000000	4703.00000
mean	2530.919413	NaN	52.481820	66.057410	66.387838	66.477142	71.25536
std	1435.318176	NaN	20.452486	13.770916	13.541550	13.791833	5.56822
min	3.000000	NaN	5.000000	15.000000	16.000000	14.000000	51.00000
25%	1290.500000	NaN	38.000000	58.000000	59.000000	59.000000	68.00000
50%	2545.000000	NaN	58.000000	69.000000	69.000000	69.000000	71.00000
75%	3773.500000	NaN	69.000000	77.000000	76.000000	76.000000	75.00000
max	4999.000000	NaN	94.000000	87.000000	93.000000	93.000000	93.00000

A quick glance at the summary above tells us that there is considerable variation among players with respect to their attributes that can not be let go of. From finishing ranging between 5 and 94, to agility ranging between 14 and 93 and likewise. To be more sure, below I am plotting scatter plot of each of the determinant variable with the target variable to have a clearer picture.

DATA VISUALIZATION AND ANALYSIS

In [116]:
```
f, (ax1, ax2, ax3, ax4, ax5, ax6, ax7, ax8, ax9, ax10, ax11) = plt.subplots(1,
11, figsize =(20,5))
ax1.scatter(df['Finishing'], df['Acceleration'])
ax1.set_title('Finishing and Acceleration')
ax2.scatter(df['Finishing'], df['SprintSpeed'])
ax2.set_title('Finishing and SprintSpeed')
ax3.scatter(df['Finishing'], df['Agility'])
ax3.set_title('Finishing and Agility')
ax4.scatter(df['Finishing'], df['Jumping'])
ax4.set_title('Finishing and Jumping')
ax5.scatter(df['Finishing'], df['Stamina'])
ax5.set_title('Finishing and Stamina')
ax6.scatter(df['Finishing'], df['Strength'])
ax6.set_title('Finishing and Strength')
ax7.scatter(df['Finishing'], df['Reactions'])
ax7.set_title('Finishing and Reactions')
ax8.scatter(df['Finishing'], df['Balance'])
ax8.set_title('Finishing and Balance')
ax9.scatter(df['Finishing'], df['LongShots'])
ax9.set_title('Finishing and LongShots')
ax10.scatter(df['Finishing'], df['ShotPower'])
ax10.set_title('Finishing and ShotPower')
ax11.scatter(df['Finishing'], df['Aggression'])
ax11.set_title('Finishing and Aggression')
```

Out[116]:
```
Text(0.5, 1.0, 'Finishing and Aggression')
```



Great! the above depiction clearly shows a strong relationship between the target and each of the determinant variable. So, this is a good motivation to move further. At this point of time, my data is cleaned and ready for further processing. Since, there is no strong polynomial sort of relationship visible in any of the figures, I have decided to fit multiple linear regression model. Multiple linear regression model is different from simple linear regression model, there is no need to check for CLRM assumptions prior. Once the model is fitted, its robustness is checked and if the results are not favourable, changing the model might be considered. That's how it works. Let's begin with the analysis.

In [117]:
```
df.columns.values
```

Out[117]:
```
array(['Unnamed: 0', 'Unnamed: 10', 'Finishing', 'Acceleration',
       'SprintSpeed', 'Agility', 'Reactions', 'Balance', 'ShotPower',
       'Jumping', 'Stamina', 'Strength', 'LongShots', 'Aggression'],
      dtype=object)
```

In [118]:
```
#defining dependent and independent variables
x = df[['Acceleration', 'SprintSpeed',
       'Agility', 'Reactions', 'Balance', 'ShotPower', 'Jumping',
       'Stamina', 'Strength', 'LongShots', 'Aggression']]
y = df[['Finishing']]
```

In [119]:
```
#splitting the data for training and testing in the ratio of 0.2
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, rand
om_state=0)
```

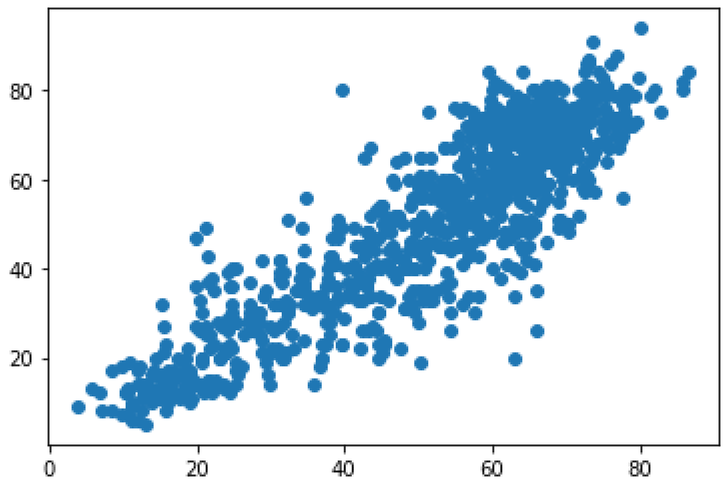
In [120]:
```
#running the regression
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(x_train, y_train)
```

Out[120]:
```
LinearRegression()
```

In [121]:
```
y_pred = regressor.predict(x_test)
np.set_printoptions(precision=2)
```

In [122]:
```
#plotting actual and predicted values of the target variable in the test datas
et
plt.scatter(y_pred, y_test)
```

Out[122]:
```
<matplotlib.collections.PathCollection at 0x14f00b02340>
```



Woah! great correlation between both. That means predicted values are not very different from actual values which is good thing, but here is when we need to be cautious. So, let's have a look at other statistics.

In [123]:
```
regressor.coef_
```

Out[123]:
```
array([[ 0.06,  0.09,  0.15,  0.15, -0.04,  0.28, -0.03, -0.02,  0.12,
         0.63, -0.2 ]])
```

```
In [124]: regressor.intercept_

Out[124]: array([-21.67])
```

```
In [125]: regressor.score(x_train, y_train)  #r squared

Out[125]: 0.7878720268215855
```

adjusted r-squared formula

$$R^2(adj.) = 1 - (1 - R^2) * \frac{n-1}{n-p-1}$$

```
In [126]: x.shape

Out[126]: (4703, 11)
```

```
In [127]: r2 = regressor.score(x,y)
          n = 4703
          p = 11
          adjusted_r2 = 1-(1 - r2)*(n-1)/(n-p-1)
          adjusted_r2

          0.787302408234665
```

All but Jumping, strength and longshots have negative effect on finishing. This raises my eyesbrows. Moreover, the value of negative intercept term is quite larger in comparison with rest of the coefficients. However, the R squared and adjusted R squared values are satisfactory. Next, I am going to check for multicollinearity as this might be the cause of dissatisfying results.

Multicollinearity can be checked by either looking at the strength of correlation among the independent variables. I f they are highly correlated, the one with higher p-value can be dropped or they can be combined into one by observing the pattern or the way they are related. The second way is to determine their Variance Inflation Factor and the one with higher values can be considered for dropping from the list. Now, it has always been a debate to figure out what should be a good cut-off for the VIF, generally a value of 10 is considered by most of the analysts. Among both, I have opted the second route as follows.

ROBUSTNESS CHECK OF THE MODEL

In [128]:
```
#checking multicollinearity
from statsmodels.stats.outliers_influence import variance_inflation_factor
variables = df[['Acceleration', 'SprintSpeed',
       'Agility', 'Reactions', 'Balance', 'ShotPower', 'Jumping',
       'Stamina', 'Strength', 'LongShots', 'Aggression']]
vif = pd.DataFrame()
vif["VIF"] = [variance_inflation_factor(variables.values, i) for i in range(va
riables.shape[1])]
vif["features"] = variables.columns
```

In [129]:
```
vif
```

Out[129]:

	VIF	features
0	96.270501	Acceleration
1	70.533787	SprintSpeed
2	68.353375	Agility
3	62.262170	Reactions
4	26.839429	Balance
5	58.361058	ShotPower
6	23.317580	Jumping
7	54.691099	Stamina
8	28.706644	Strength
9	16.761007	LongShots
10	14.578956	Aggression

As expected and from common knowledge, multicollinearity has to exist among these indicators. It can be seen that variables like acceleration, sprint speed, agility, Stamina, reactions, shot power suffer from high multicollinearity. So, I decided to drop them altogether.

In [130]:
```
df.drop(['Acceleration', 'SprintSpeed', 'Agility', 'Stamina', 'Reactions', 'Sh
otPower'], axis=1, inplace=True)
```

In [131]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
variables = df[['Balance', 'Jumping', 'Strength', 'LongShots', 'Aggression']]
vif = pd.DataFrame()
vif["VIF"] = [variance_inflation_factor(variables.values, i) for i in range(va
riables.shape[1])]
vif["features"] = variables.columns
```

In [132]:
```
vif
```

Out[132]:

	VIF	features
0	8.438189	Balance
1	11.461269	Jumping
2	9.170667	Strength
3	5.366452	LongShots
4	7.009692	Aggression

This is the beauty of using software tools, add and delete as many rows and columns as and whenever you want. So, after dropping the variables with high multicollinearity,the VIF of remaining variables has improved for good. Now, I am going to perform a second round of model fitting with improved dataset.

In [133]:
```
#defining variables
x2 = df[['Balance', 'Jumping', 'Strength', 'LongShots', 'Aggression']]
y2 = df[['Finishing']]
```

In [134]:
```
#test and train split
x2_train, x2_test, y2_train, y2_test =  train_test_split(x2, y2, test_size=0.2
, random_state=0)
```

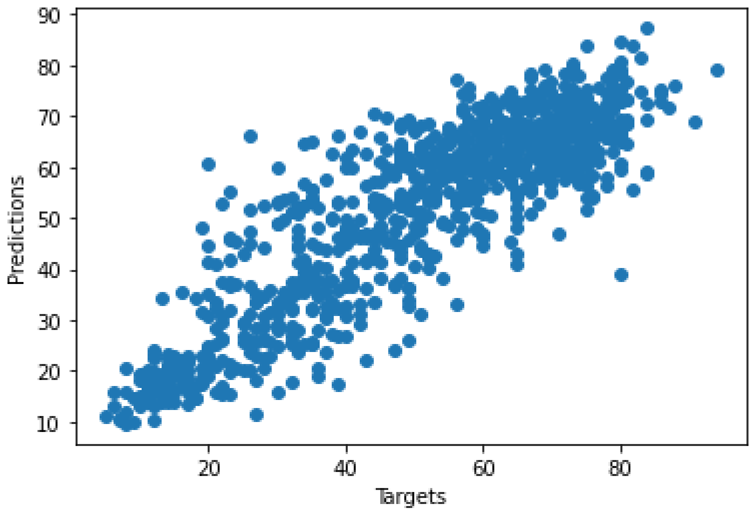
In [135]:
```
#running the regression
regressor2 = LinearRegression()
regressor2.fit(x2_train, y2_train)
```

Out[135]:
```
LinearRegression()
```

In [136]:
```
y_predictions  = regressor2.predict(x2_test)
np.set_printoptions(precision=2)
```

```
In [137]: #plotting predicted and actual values of the test dataset
          plt.scatter(y2_test, y_predictions)
          plt.xlabel('Targets')
          plt.ylabel('Predictions')

Out[137]: Text(0, 0.5, 'Predictions')
```



Once, again it can be seen that the predicted values are quite closer to the actual values. Now, let's have a look at other statistics.

```
In [138]: regressor2.coef_

Out[138]: array([[ 0.12,  0.01,  0.15,  0.89, -0.16]])

In [139]: regressor2.intercept_

Out[139]: array([-6.63])

In [140]: regressor2.score(x2_train, y2_train)

Out[140]: 0.7638265598627446

In [141]: x2.shape

Out[141]: (4703, 5)

In [142]: #adjusted r_squared
          r2_squared = regressor2.score(x,y)
          n = 4703
          p = 11
          adjusted_regressor2 = 1-(1 - r2_squared)*(n-1)/(n-p-1)
          adjusted_regressor2

          0.7635748415339
```

FINAL RESULTS AND FINDINGS

It's pretty clear from above that the results from second round of model fitting is nicer and more reliable than the previous one. The coefficients are mostly positive and hold more weight in determing the target variable as compared to earlier ones. The intercept has also reduced drastically in terms of its absolute value. The only variable having negative impact on finishing is aggression, which implies keeping rest of the attributes intact, an increase in aggression is going to impact the finishing ability negatively. Now, this doesn't sound unconvincing and makes sense. The attribute having highest positive impact on finishing a goal is Longshots while Jumping having the least impact. The values of R-squared and adjusted R-squared are high enough to claim that the target variable is significantly explained through the determinant variables.

Next, I am going to take a closer look at predicted and actual values and see how are their residuals behaving.

In [143]:
```
df_pf = pd.DataFrame(y_predictions, columns=['Predictions'])
df_pf.head()
```

Out[143]:

	Predictions
0	48.880650
1	54.691940
2	41.484911
3	28.879966
4	36.799936

In [144]:
```
# resetting the index values as it may create hindrance in calculating residua
ls
y2_test = y2_test.reset_index(drop=True)
y2_test.head()
```

Out[144]:

	Finishing
0	30.0
1	58.0
2	20.0
3	30.0
4	26.0

```
In [145]: df_pf['Targets'] = y2_test
          df_pf.head()
```

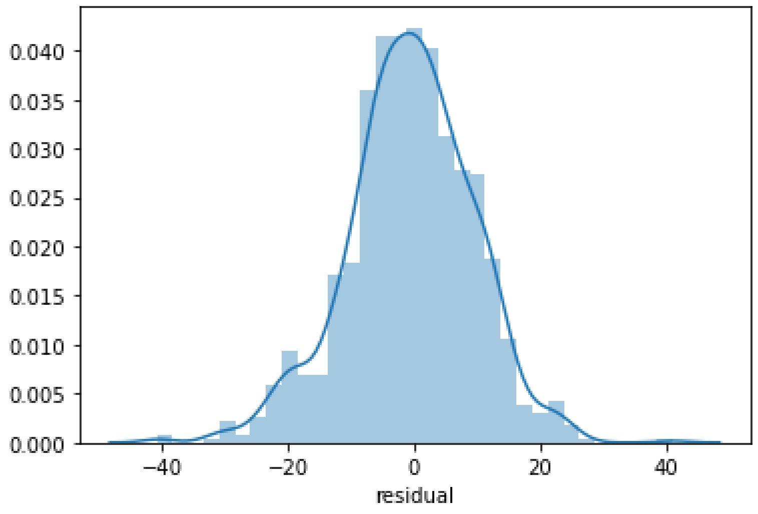
Out[145]:

	Predictions	Targets
0	48.880650	30.0
1	54.691940	58.0
2	41.484911	20.0
3	28.879966	30.0
4	36.799936	26.0

```
In [146]: df_pf['residual'] = df_pf['Targets'] - df_pf['Predictions']
```

```
In [147]: sns.distplot(df_pf['residual'])  #wow normally distributed
```

```
Out[147]: <matplotlib.axes._subplots.AxesSubplot at 0x14f00d0da90>
```



Great! this is a very fine result. The residuals are normally distributed which is a very desirable thing in most of the model fittings.

```
In [148]: df_pf['Difference%'] = np.absolute(df_pf['residual']/df_pf['Targets']*100)
```

In [149]:
```
df_pf.describe()
```

Out[149]:
```
       Predictions     Targets    residual  Difference%
count   941.000000  941.000000  941.000000   941.000000
mean     52.278550   51.693943   -0.584608    20.040182
std      18.064492   20.672313   10.052715    23.904937
min       9.573999    5.000000  -40.842975     0.015456
25%      38.846961   35.000000   -6.388593     5.897705
50%      58.425311   56.000000   -0.284793    12.915152
75%      66.088762   69.000000    6.008926    23.784211
max      87.291152   94.000000   41.100014   204.214877
```

Here, I have created a dataframe consisting predicted values, target values, their difference and the percentage of difference. so, the summary tells us that the difference % ranges between 0.015 and 204 with an average of 20. Overall, the differences ar not drastically large which is a good thing and the results are in compliance with our prior assumptions. However, there's always room left for improvements and certainly something more can be done with the dataset but here too I ended up getting nice results.