NBA Data Analysis

Nome: Giovanni
Cognome: Imbesi
Matricola: 1000006253
Corso: Social Media Management
Docente: Antonino Furnari
Anno Accademico: 2021/2022

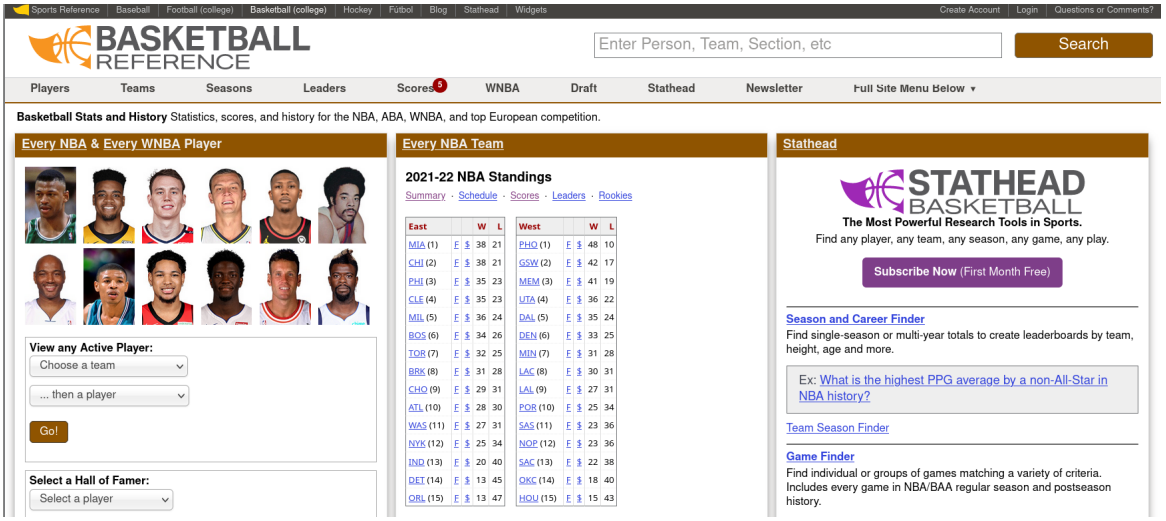
1. Introduzione

La National Basket Association è il campionato di basket americano, nonchè il più importante e conosciuto campionato di basket al mondo. L'analisi delle statistiche e dei dati è sempre stata una componente essenziale all'interno di questo sport, ma ad oggi permea ogni suo aspetto, tanto che ogni franchigia possiede un team di analisti esperti nella valutazione dei giocatori.

L'idea alla base di questo progetto consiste nell'analizzare due task tramite gli strumenti della Regressione.
In particolare, il primo task consiste nel predire quella che è la media dei minuti giocati da ogni giocatore a partire dalle statistiche individuali, attraverso l'utilizzo di un regressore lineare multinomiale.
Il secondo invece riguarda la classificazione: dato un giocatore, si cercherà di predire la posizione in campo mediante l'utilizzo di un gressore logistico.

1.1 Data Source

Esistono decine di siti web che mettono ad disposizione, anche gratuitamente, migliaia di dati relativi l'NBA. In questa sede saranno utilizzati i database di Basketball-reference, il quale non solo mette a disposizione numerose statistiche su ogni giocatore o squadra, ma permette di conoscere i risultati, le news ed in generale ogni dettaglio che ruota attorno al mondo NBA.



Una volta ottenuti i dati si può procedere all'analisi: per i nostri scopi si è optato per dei semplici file csv. Il primo 'nbaStats" fa riferimento alle statistiche degli ultimi 10 anni, mentre il secondo 'nbaOldStats" include tutte le statistiche dei giocatori in attività dalla fine degli anni 80' agli anni 90'. Quest'ultimo verrà utilizzato come parametro di confronto nel task di classificazione.
È importante sottolineare che ogni parametro è rappresentato dalla media delle statistiche che il singolo giocatore ha ottenuto in ogni partita.

1.2 Librerie

Di seguito vengono elencate le librerie utilizzate all'interno del notebook. In generale utilizzeremo pandas per la creazione e gestione dei dataFrame e sklearn per utilizzare tutti gli strumenti della regressione.

In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure as fg
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import r2_score
import seaborn as sns
from sklearn.feature_selection import RFE
from sklearn.preprocessing import PolynomialFeatures
```

2. Average Minutes Prediction

2.1 Presentazione ed Analisi dei dati

Per prima cosa importiamo i dati e creiamo un dataframe per il file nbaStats.csv.

In [2]:

```
df= pd.read_csv("nbaStats.csv")
df = df.fillna(0.0)
df.head()
```

Out[2]:

	Rk	Player	Pos	Age	Tm	G	GS	MP	FG	FGA	...	FT%	ORB	DRB	TR
0	1	Precious Achiuwa\achiupr01	C	22	TOR	48	23	23.1	3.1	7.3	...	0.579	2.2	4.7	6
1	2	Steven Adams\adamsst01	C	28	MEM	56	55	26.1	2.8	5.0	...	0.556	4.5	5.1	9
2	3	Bam Adebayo\adebaba01	C	24	MIA	34	34	33.4	7.1	13.5	...	0.751	2.9	7.4	10
3	4	Santi Aldama\aldamsa01	PF	21	MEM	27	0	10.3	1.4	3.6	...	0.600	0.9	1.5	2
4	5	LaMarcus Aldridge\aldrila01	C	36	BRK	39	11	22.9	5.8	10.4	...	0.855	1.6	4.1	5

5 rows × 30 columns

In [3]:

```
df.describe()
```

Out[3]:

	Rk	Age	G	GS	MP	FG	F
count	6470.000000	6470.000000	6470.000000	6470.000000	6470.000000	6470.000000	6470.000
mean	3101.707419	26.256878	42.524420	19.471561	19.125286	2.974482	6.651
std	1854.872475	4.151838	25.643409	25.507143	9.334368	2.124724	4.478
min	1.000000	19.000000	1.000000	0.000000	0.000000	0.000000	0.000

	Rk	Age	G	GS	MP	FG	F
25%	1476.250000	23.000000	19.250000	0.000000	11.700000	1.300000	3.300
50%	3093.500000	26.000000	44.000000	6.000000	18.700000	2.500000	5.600
75%	4710.750000	29.000000	66.000000	34.000000	26.800000	4.100000	9.200
max	6328.000000	43.000000	83.000000	82.000000	42.000000	11.200000	24.500

Dovendo predirre la media dei minuti giocati, andremo a porre i corrispondenti valori in una variabile y, mentre in X conserveremo i valori relativi alle altre statistiche ad eccezione di alcune poco rilevanti per i nostri scopi.

In [4]:
```
y=df['MP']
X=df.drop(['Rk','Player','Pos','MP','Tm'], axis=1)
```

2.2 Feature Selection

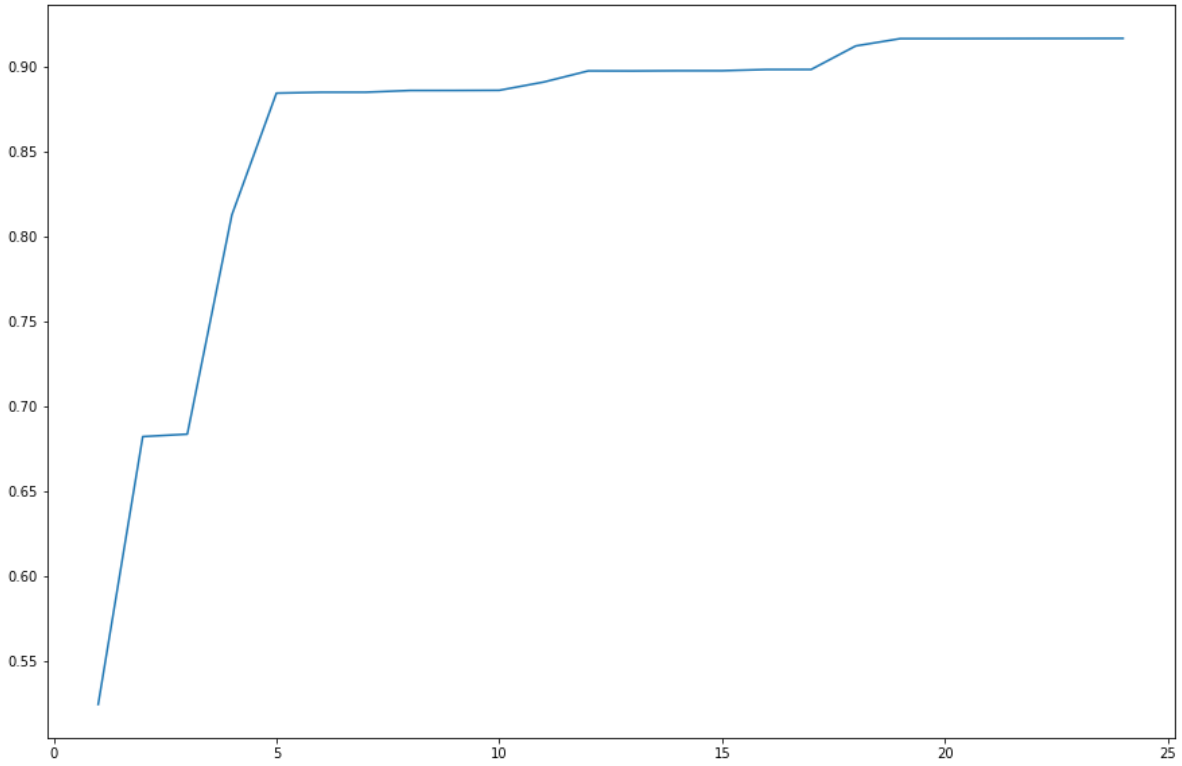
Per allenare il nostro regressore dobbiamo per prima cosa scegliere quelle che sono le feature discriminanti. Per farlo useremo il principio del recursive feature elimination(RFE), il quale permette di individuare il numero e quali sono le feature con maggior potere predittivo.

In [5]:
```
nof_list=np.arange(1,25)
high_score=0
nof=0
score_list =[]
for n in range(len(nof_list)):
    X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0
    model = LinearRegression()
    rfe = RFE(model, n_features_to_select=nof_list[n])
    X_train_rfe = rfe.fit_transform(X_train,y_train)
    X_test_rfe = rfe.transform(X_test)
    model.fit(X_train_rfe,y_train)
    score = model.score(X_test_rfe,y_test)
    score_list.append(score)
    print("Score con ",nof_list[n],"feature: ",score)
    if(score>high_score):
        high_score = score
        nof = nof_list[n]

plt.figure(figsize=(15,10))
plt.plot(nof_list, score_list)
```

```
Score con  1 feature:  0.5247426104195702
Score con  2 feature:  0.6824049355152861
Score con  3 feature:  0.6838360398561816
Score con  4 feature:  0.8129229384595629
Score con  5 feature:  0.8846115388981514
Score con  6 feature:  0.8851241892399774
Score con  7 feature:  0.8851427565811785
Score con  8 feature:  0.8861703850014968
Score con  9 feature:  0.8861814149976832
Score con  10 feature:  0.8862676848560889
Score con  11 feature:  0.8911487352547361
```

```
Score con  12 feature:  0.897716998575518
Score con  13 feature:  0.8976787773283685
Score con  14 feature:  0.8978117825905823
Score con  15 feature:  0.8978093729120955
Score con  16 feature:  0.8985156683571134
Score con  17 feature:  0.8985339744621247
Score con  18 feature:  0.9123928334447079
Score con  19 feature:  0.9167051099196502
Score con  20 feature:  0.9167054689824123
Score con  21 feature:  0.9167383701355346
Score con  22 feature:  0.9167794141256331
Score con  23 feature:  0.9168044583663804
Score con  24 feature:  0.916864067044914
```

Out[5]:
```
[<matplotlib.lines.Line2D at 0x7f1017ff0e80>]
```



Dal grafico è possibile notare chè all'aumentare del numero di feature, la precisione nella predizione aumenta. Allo stesso tempo però, è possibile che un numero elevato di attributi vada a generare dell'overfitting. Per tale motivo è necessario effettuare un trade-off tra error rate e flessibilità. Una buona soluzione potrebbe essere quella di selezionare n=5 feature, ovvero il punto in cui la curva flette maggiormente.

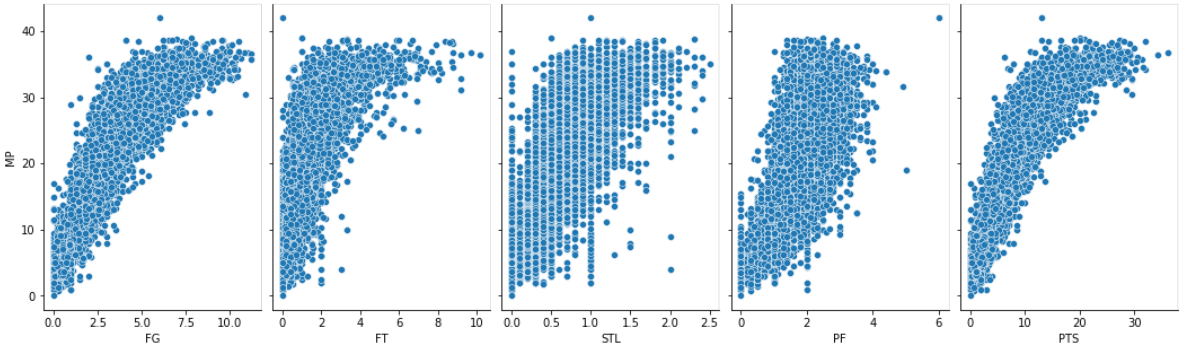
In [6]:
```
cols = list(X.columns)
model = LinearRegression()

rfe = RFE(model, n_features_to_select=5)
X_rfe = rfe.fit_transform(X,y)
model.fit(X_rfe,y)
temp = pd.Series(rfe.support_,index = cols)
selected_features_rfe = temp[temp==True].index
X=df[selected_features_rfe]
print(selected_features_rfe)
```

```
Index(['FG', 'FT', 'STL', 'PF', 'PTS'], dtype='object')
```

Le feature selezionate sono quelle col maggior potere predittivo. In ordine abbiamo:
-FG: indica il rapporto tra il numero di tiri segnati ed il numero totale di tiri
-FT: indica il numero totale di tiri liberi
-STL: indica il numero di palle rubate
-PF: indica il numero di falli che il giocatore ha commesso
-PTS: indica il numero di punti segnati

Quì di seguito è possibile notare le relazioni tra variabili indipendenti e quella dipendente.

In [7]:
```
g=sns.pairplot(data=df,
               y_vars=['MP'],
               x_vars=(['FG', 'FT', 'STL', 'PF', 'PTS']))
g.fig.set_size_inches(15,5)
```



2.3 Allenamento e Validazione del regressore

Una volta definite le feature, possiamo allenare il nostro regressore lineare, osservando i valori dell'intercetta e dei coefficienti

In [8]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.30, random_state=0)

lin_reg = LinearRegression()
model = lin_reg.fit(X_train, y_train)
print(f'intercetta = {model.intercept_}')
print(f'coefficienti= {model.coef_}')
```

```
intercetta = 3.1957394469958533
coefficienti= [-1.28562319 -2.12531823  4.73360671  3.03108414  1.82267711]
```

Analizziamo i dati ottenuti: per farlo useremo due metriche, l'r2 score ed il Mean Absolute Error

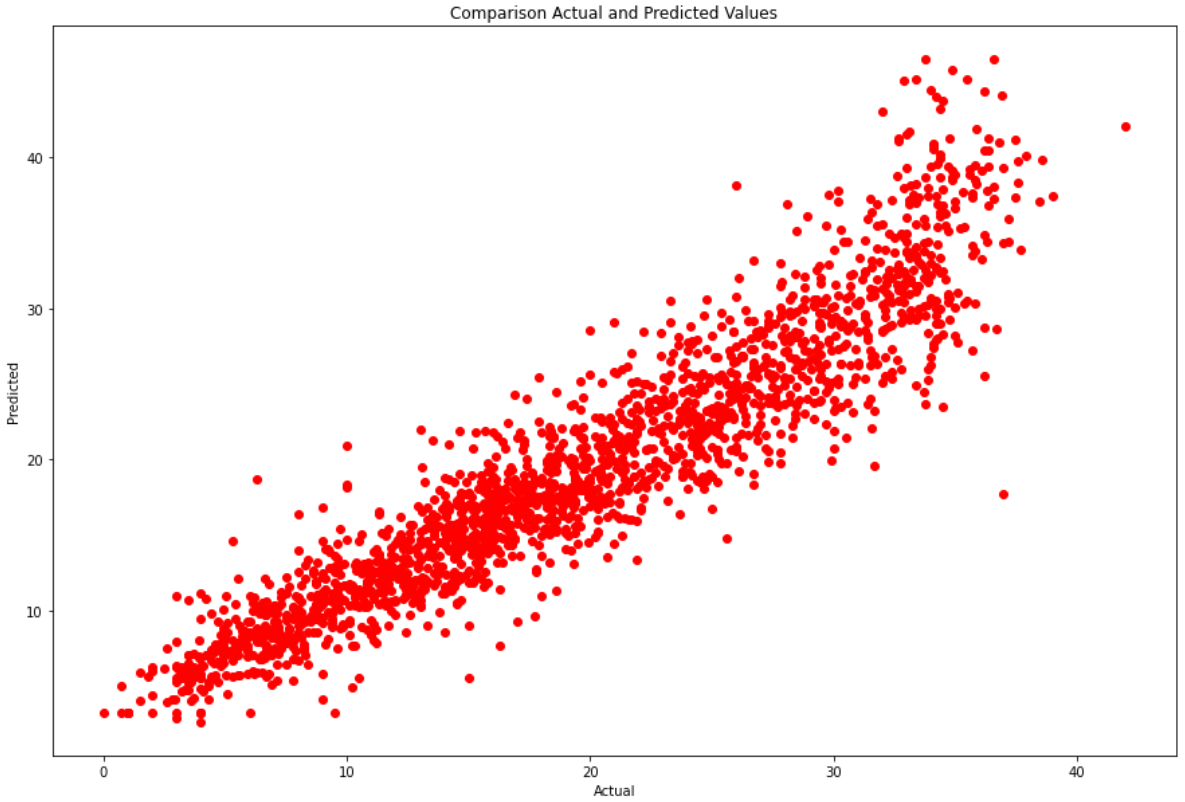
In [9]:

```
y_test_predicted=model.predict(X_test)
MAE_linearReg=mean_absolute_error(y_test, y_test_predicted)
print("MAE", MAE_linearReg)
print("r2_score",r2_score(y_test, y_test_predicted))
```

```
MAE 2.508927686029837
r2_score 0.8749705777346773
```

In [10]:

```
plt.figure(figsize=(15,10))
plt.scatter(y_test, y_test_predicted, color='red')
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Comparison Actual and Predicted Values")
plt.show()
```



3. Approccio Polinomiale

3.1 Best Degree Value

Nonostante i valori ottenuti siano buoni, è forse possibile fare di meglio tramite un regressore Polinomiale. Per verificarlo dobbiamo prima di tutto definire il numero di termini aggiuntivi.

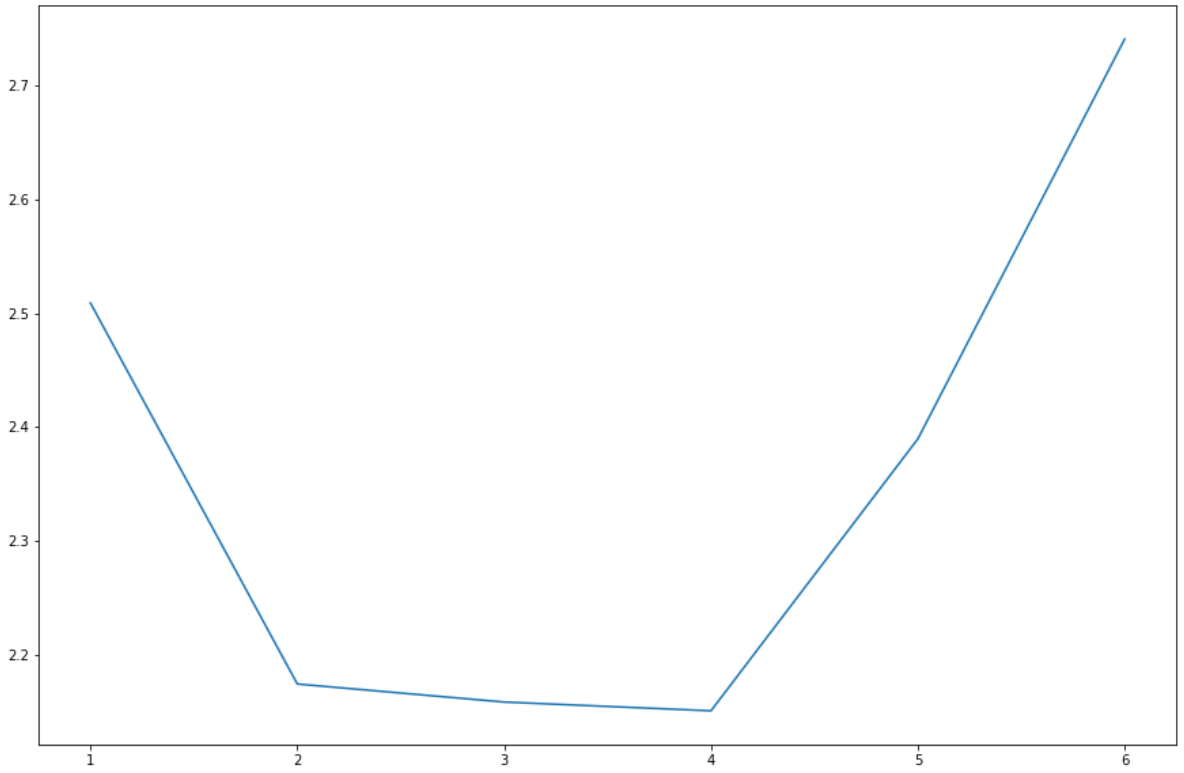
In [11]:

```
score_list =[]
alpha_value=[1,2,3,4,5,6]
for n in alpha_value:
    X=df[selected_features_rfe]
    poly_reg=PolynomialFeatures(degree = n)
    X= poly_reg.fit_transform(X)
    X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0
    model = LinearRegression()
    model.fit(X_train,y_train)
    y_pred=model.predict(X_test)
    score=mean_absolute_error(y_test,y_pred)
    score_list.append(score)
    print("Score con alpha=",n,": ",score)


plt.figure(figsize=(15,10))
plt.plot(alpha_value,score_list)
```

```
Score con alpha= 1 :  2.5089276860298337
Score con alpha= 2 :  2.173925718572096
Score con alpha= 3 :  2.158059528769445
Score con alpha= 4 :  2.1503566742031017
Score con alpha= 5 :  2.3894370159492215
Score con alpha= 6 :  2.7410477572552083
```

Out[11]:

```
[<matplotlib.lines.Line2D at 0x7f1018b56230>]
```



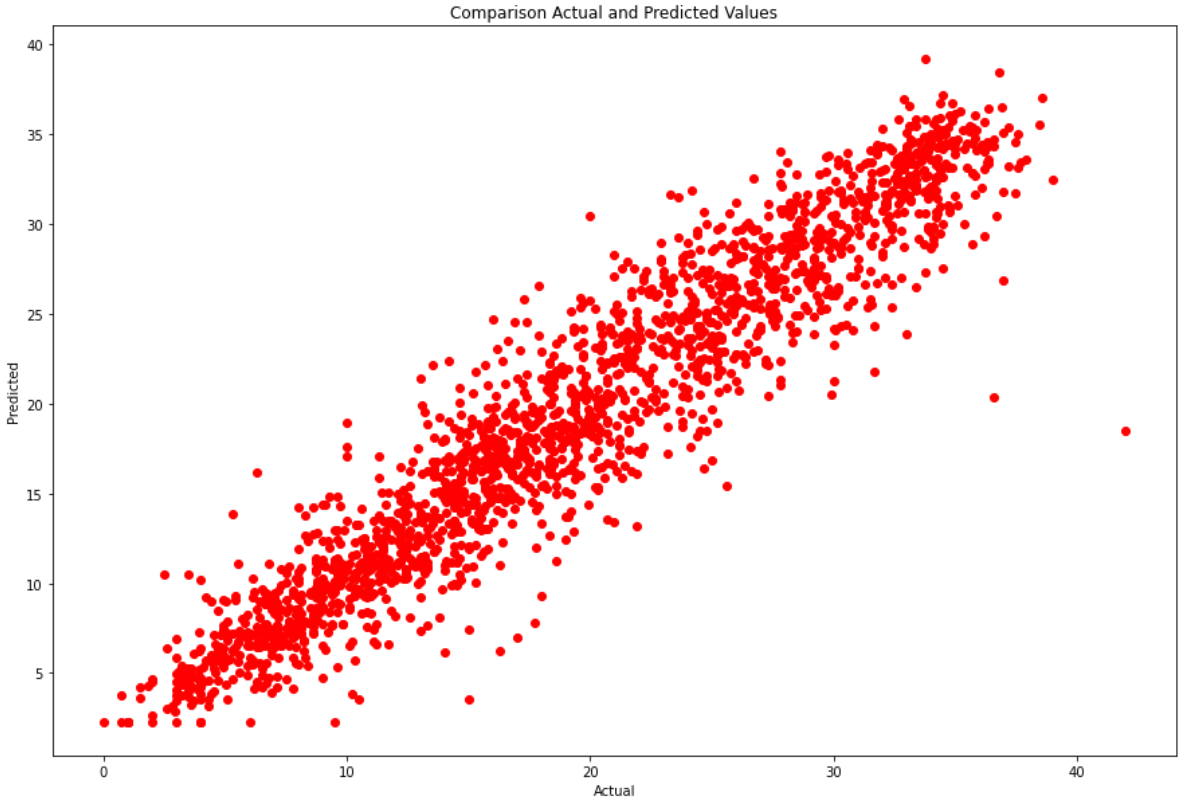
Il valore alpha=4 è sicuramente il migliore. È ora possibile allenare il regressore Polinomiale.

3.2 Training & Result

In [12]:

```
X=df[selected_features_rfe]
poly_reg=PolynomialFeatures(degree = 4)
X= poly_reg.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.3,
model = LinearRegression()
model.fit(X_train,y_train)
y_test_predicted=model.predict(X_test)


print("MAE", mean_absolute_error(y_test,y_test_predicted))
print("r2_score",r2_score(y_test, y_test_predicted))


plt.figure(figsize=(15,10))
plt.scatter(y_test, y_test_predicted, color='red')
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Comparison Actual and Predicted Values")
plt.show()
```

```
MAE 2.1503566742031017
r2_score 0.9038818763584229
```



I risultati ottenuti sono sicuramente migliori rispetto ai precedenti ma è importante sottolineare che l'aggiunta dei termini di grado superiore ha prodotto un modello più complesso e costoso dal punto di vista computazionale, nonchè più esposto all'overfitting.

4. Position Prediction

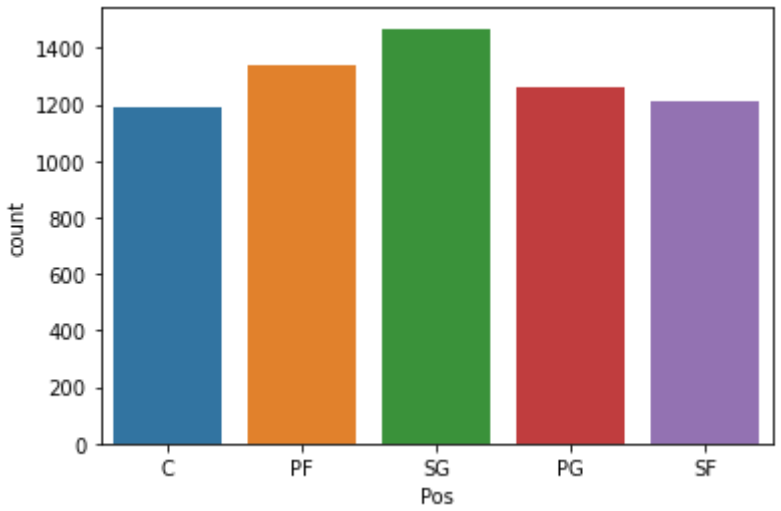
4.1 Distribuzione delle classi

Ogni giocatore può appartenere ad una delle seguenti classi:
-C: Center
-PF: Point Forward
-PG: Point Guard
-SF: Shooting Guard
-SF: Small Forward

Verifichiamo inizialmente la distribuzione delle posizioni.

In [13]:
```
print(df['Pos'].value_counts())
sns.countplot(x='Pos', data=df)
plt.figure(figsize=(15,10))
plt.show()
```

```
SG    1468
PF    1338
PG    1262
SF    1213
C     1189
Name: Pos, dtype: int64
```




```
<Figure size 1080x720 with 0 Axes>
```

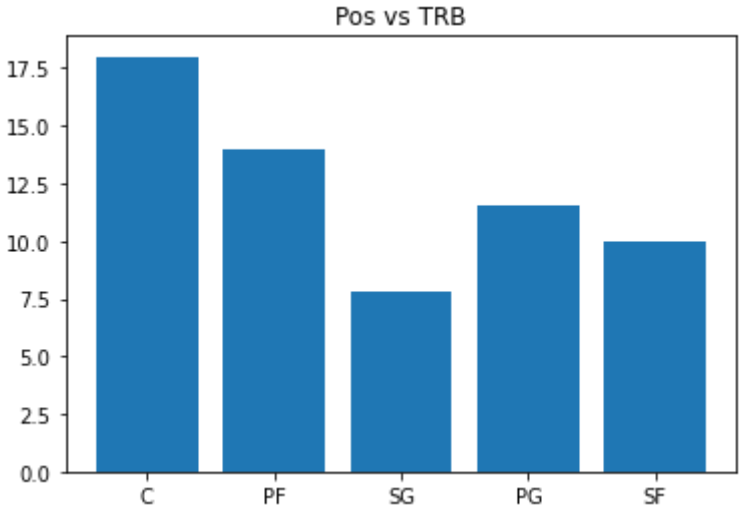
La distribuzione delle classi è più o meno uniforme, non siamo quindi in presenza di un dataset particolarmente sbilanciato

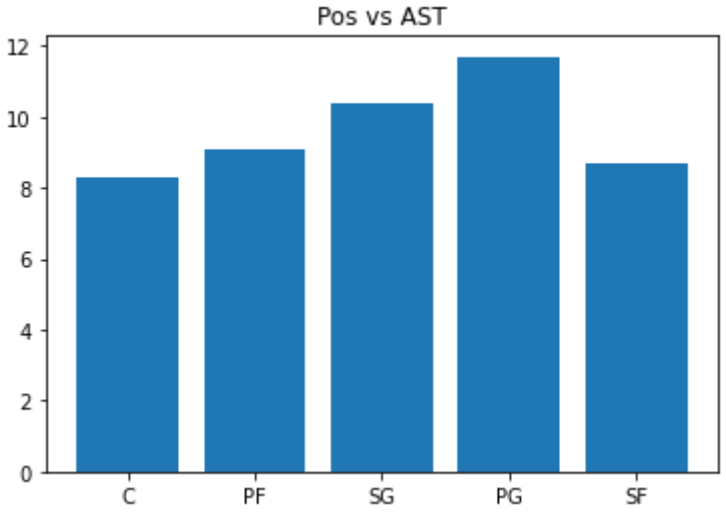
4.2 Scelta delle Feature

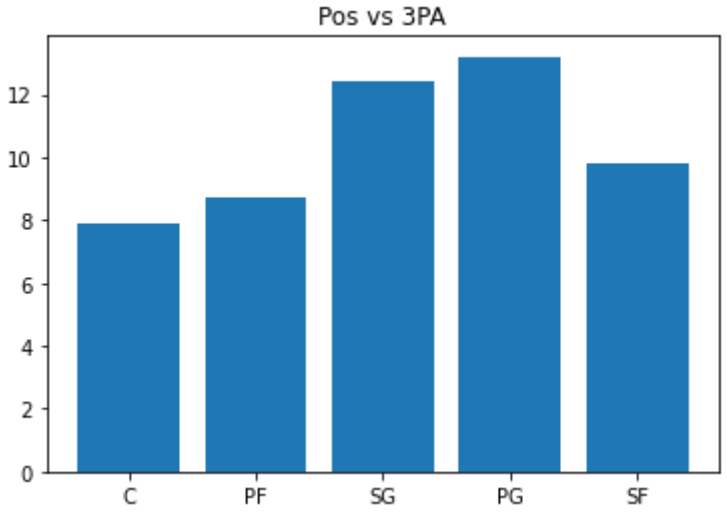
Il primo passo per costruire il modello è ovviamente quello di scegliere le feature più discriminative. I dati ci dicono che generalmente i giocatori che ricoprono la posizione di Centro(quindi di classe C) raccolgono un numero di rimbalzi maggiore rispetto a quelli che ricoprono la posizione di Point Guard,i quali invece registrano un maggior numero di assist.
Di seguito possiamo osservare alcune di queste relazioni.

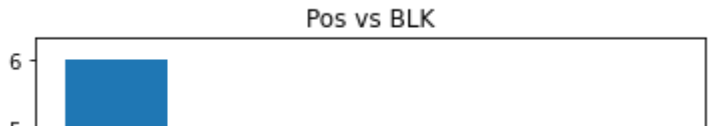
In [14]:

```
plt.bar(df['Pos'], df['TRB'])
plt.title("Pos vs TRB")
plt.show()

plt.bar(df['Pos'], df['AST'])
plt.title("Pos vs AST")
plt.show()

plt.bar(df['Pos'], df['3PA'])
plt.title("Pos vs 3PA")
plt.show()

plt.bar(df['Pos'], df['BLK'])
plt.title("Pos vs BLK")
plt.show()
```









Sulla base di queste ed altre considerazioni, sono state selezionate le seguenti feature:

-MP: indica in media il numero di minuti giocati a partita

-3PA: indica il numero di tiri da 3 tentati

-FT: indica il numero di tiri liberi effettuati

-TRB: indica il numero totale di rimbalzi

-ORB: indica il numero di rimbalzi offensivi

-BLK: indica il numero di stoppate effettuate

-AST: indica il numero di assist

-STL: indica il numero di palle rubate

4.3 Training del Regressore Logistico

In [15]:

```
logisticPredictors=['MP','3PA','FT','TRB','ORB','AST','BLK','STL']
X_log=df[logisticPredictors]

y_log=df['Pos']

X_log_train, X_log_test, y_log_train, y_log_test = train_test_split(
    X_log, y_log, test_size=0.20, random_state=0)
logistic_Model= LogisticRegression(random_state=0, multi_class='multinomia
                                   solver='newton-cg').fit(X_log_train, y_
preds = logistic_Model.predict(X_log_test)
```

Una volta allenato e testato il modello possiamo osservarne i risultati analizzando la matrice di confusione ad esso associata.

In [16]:

```
confmtrx = np.array(confusion_matrix(y_log_test, preds))
pd.DataFrame(confmtrx, index=['C','PF', 'PG','SG','SF'],
columns=['predicted_C', 'predicted_PF', 'predicted_PG', 'predicted_SG', 'p
```

Out[16]:

	predicted_C	predicted_PF	predicted_PG	predicted_SG	predicted_SF
C	172	51	1	3	3
PF	64	126	7	41	18
PG	1	2	190	2	51
SG	5	70	7	92	69
SF	2	17	45	54	201

Grazie al metodo classification_report è possibile estrarre alcune metriche utili per valutare il classificatore.

In [17]:

```
class_report=classification_report(y_log_test, preds)
print(class_report)
```

```
              precision    recall  f1-score   support

           C       0.70      0.75      0.73       230
          PF       0.47      0.49      0.48       256
          PG       0.76      0.77      0.77       246
          SF       0.48      0.38      0.42       243
          SG       0.59      0.63      0.61       319

    accuracy                           0.60      1294
   macro avg       0.60      0.60      0.60      1294
weighted avg       0.60      0.60      0.60      1294
```

La matrice di confusione e le matriche associate ci dicono che il modello ha avuto difficoltà nella classificazione, in particolare per i giocatori che ricoprono la posizione di PF(Point Forward) e SF(Small Forward).
Supponiamo invece di voler allenare un regressore logistico sul database nbaOldStats.

5. Old Statistics Analysis

In [18]:

```
dfOld= pd.read_csv("nbaOldStats.csv")
dfOld = dfOld.fillna(0.0)
print(len(dfOld))
dfOld.head()
```

```
5798
```

Out[18]:

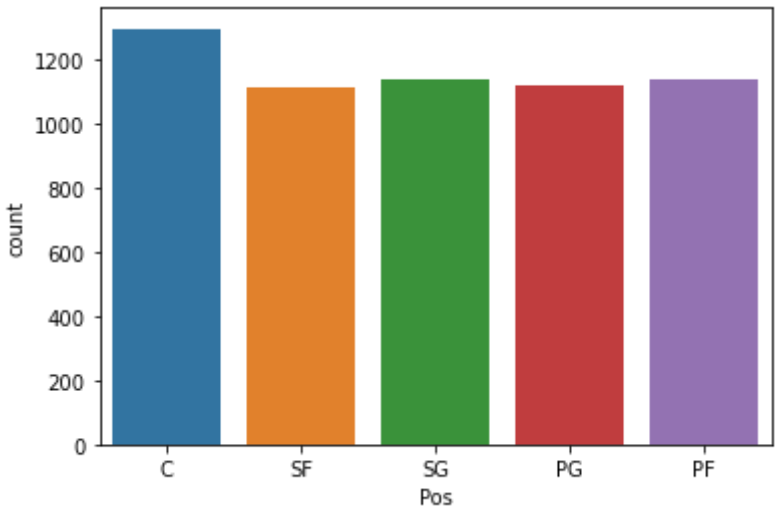
	Rk	Player	Pos	Age	Tm	G	GS	MP	FG	FGA	...	FT%	ORB	DRB	TR
0	1	Kareem Abdul-Jabbar*\abdulka01	C	37	LAL	79	79	33.3	9.2	15.3	...	0.732	2.1	5.8	7.
1	2	Alvan Adams\adamsal01	C	30	PHO	82	69	26.0	5.8	11.2	...	0.883	1.9	4.2	6.
2	3	Mark Aguirre\aguirma01	SF	25	DAL	80	79	33.7	9.9	19.6	...	0.759	2.4	3.6	6.
3	4	Danny Ainge\aingeda01	SG	25	BOS	75	73	34.2	5.6	10.6	...	0.868	1.0	2.6	3.

	Rk	Player	Pos	Age	Tm	G	GS	MP	FG	FGA	...	FT%	ORB	DRB	TR
4	5	Chuck Aleksinas\aleksch01	C	25	GSW	74	4	15.1	2.2	4.6	...	0.733	1.2	2.5	3.

In [19]:

```
print(dfOld['Pos'].value_counts())
sns.countplot(x='Pos', data=dfOld)
plt.show()
```

```
C     1295
SG    1135
PF    1134
PG    1120
SF    1114
Name: Pos, dtype: int64
```



In [20]:

```
X_log=dfOld[logisticPredictors]

y_log=dfOld['Pos']

X_log_train, X_log_test, y_log_train, y_log_test = train_test_split(
    X_log, y_log, test_size=0.20, random_state=0)

logistic_Model= LogisticRegression(random_state=0, multi_class='multinomia
                                   solver='newton-cg').fit(X_log_train, y_

old_preds = logistic_Model.predict(X_log_test)
confmtrx = np.array(confusion_matrix(y_log_test, old_preds))


pd.DataFrame(confmtrx, index=['C','PF', 'PG','SG','SF'],
columns=['predicted_C', 'predicted_PF', 'predicted_PG', 'predicted_SG', 'p
```

Out[20]:

	predicted_C	predicted_PF	predicted_PG	predicted_SG	predicted_SF
C	207	36	0	13	6
PF	97	102	0	30	6
PG	1	1	192	1	21
SG	13	31	6	112	61
SF	7	1	34	34	148

In [21]:

```
class_report=classification_report(y_log_test, old_preds)
print(class_report)
```

```
              precision    recall  f1-score   support

           C       0.64      0.79      0.71       262
          PF       0.60      0.43      0.50       235
          PG       0.83      0.89      0.86       216
          SF       0.59      0.50      0.54       223
          SG       0.61      0.66      0.64       224

    accuracy                           0.66      1160
   macro avg       0.65      0.66      0.65      1160
weighted avg       0.65      0.66      0.65      1160
```

6. Conclusioni

Possiamo osservare come in generale i risultati ottenuti siano migliori rispetto ai precedenti. Sebbene il problema di predire la posizione di un giocatore, a partire dalle sue statistiche, sia già in partenza molto complesso, nell'NBA moderna è ancora più difficile rispetto al passato. Tra gli anni 80' e 90' infatti, i sistemi di gioco erano molto più statici ed i giocatori avevano un ruolo più definito rispetto ad oggi, ed è per questo che è possibile notare un miglioramento, seppur contenuto, nella classificazione.