Fatec São José dos Campos
Prof. Jessen Vidal

CENTRO PAULA SOUZA

GOVERNO DO ESTADO SÃO PAULO

Projeto #2 de IA: Regressão Linear, Clusterização, Redes Neurais

Carlos Alberto Cardoso
Raphael dos Reis Toledo
Rodrigo Diniz dos Santos
Tiago Henrique da Cruz
Wesley Matheus Nascimento de Almeida

Tecnólogo em Análise e Desenvolvimento de Sistemas

Turma: 6° ADS Inteligência Artificial, Professor: Walmir

Sumário

1. Introdução

Este documento tem como objetivo demonstrar a elaboração de 3 sistemas de inteligência artificial, sendo eles uma Regressão Linear, um clustering com k-means e uma rede neural. A regressão linear tem o objetivo de mostrar a relação da quantidade de arremessos feitos pela porcentagem de acertos desses arremessos feitos por jogadores de basquete da principal liga de basquete profissional da américa do norte a NBA, também considerada a principal liga de basquete do mundo. Para o clustering estamos utilizando os principais status de interesse do público para fazer o agrupamento desses jogadores que foram eleitos os melhores jogadores da semana. Já o sistema de redes neurais foi desenvolvido com o intuito de esclarecer para o usuário se é realmente valido fazer o investimento em um certo tipo de criptomoeda ou se vale a pena fazer o investimento em outro tipo de criptomoeda.

Com este trabalho, o principal foco é demonstrar como sistemas de IA podem ajudar os usuários a tomarem decisões em assuntos do seu cotidiano ou mostrar os dados pesquisados por esses usuários de uma forma mais simplificada e de melhor entendimento.

2. Regressão Linear

2.1 Planejamento

a) **Breve descrição do contexto do problema.**

Algo muito praticado por fãs de modalidades esportivas é o acompanhamento dos resultados obtidos pelos jogadores durante a temporada. Com isso foi decidido buscar uma forma de descobrir qual a provável quantidade de cestas convertidas por um jogador ao longo das partidas para isso foi usada os status dos jogadores eleitos o Melhor Jogador da Semana na NBA afim de atingir o objetivo do projeto. Para isso foi utilizada uma base de dados contendo os jogadores eleitos desde a temporada 1984-85 até a temporada 2017-2018.

b) **Porque está usando Regressão Linear para o problema.**

Modelos de regressão, são modelos matemáticos que relacionam o comportamento de uma variável Y com outra X, em que a Y depende da X para saber seu valor através de uma metodologia estatística que utiliza a relação entre as duas variáveis.

c) **Variáveis do problema (nomes, tipos, domínio de valores).**

FGA: Inteiro - Quantidade de cestas (não lance livre) tentadas

d) **Dentre as variáveis, qual é a Variável a ser Prevista (Classe).**

FGM: Inteiro - Quantidade de cestas (não lance livre) convertidas

2.2 Implementação

O conjunto de dados obtidos originalmente pelo link https://www.kaggle.com/jacobbaruch/nba-player-of-the-week possuía atletas com dados faltantes, para resolver este problema os dados foram tratados por um algoritmo em python para remover os dados defeituosos gerando o arquivo “players_stats_new.xls”.

Figura 2.1 – Algoritmo de Tratamento dos Dados

```
import csv
import os
import pandas
import matplotlib.pyplot as plt

data = csv.reader(open("players_stats.csv","r"))
gerado = csv.writer(open("players_stats_new.csv","w"),lineterminator='\r')
path =  os.path.dirname(os.path.realpath(__file__))
dir = os.listdir(path)
cont = 0
cont = 0

for rows in data:
    if rows[24] == '' or rows[29] == '' or rows[30] == '' or rows[32] == '':
        next
    else:
        gerado.writerow(rows)
```

Após tratar os dados o próximo passo é importar as bibliotecas que serão utilizadas e ler a base de dados na forma de um DataFrame chamado ‘nba’.

Figura 2.2 – Importação de Bibliotecas e Leitura dos Dados

```
In [1]: import pandas as pd
        import seaborn as sns
        import numpy as np
        import matplotlib.pyplot as plt

        from sklearn import linear_model
        from sklearn.metrics import r2_score
        from sklearn.model_selection import train_test_split

In [2]: # dados originais
        data = pd.read_csv('players_stats_new.csv')
        data
```

Out[2]:

	Name	Games Played	MIN	PTS	FGA	FGM	FG%	3PM	3PA	3P%	...	Birth_Place	Birthdate	Collage	Experience	Pos	Team	Age	Height	We
0	AJ Price	26	324	133	137	51	37.2	15	57	26.3	...	us	7-Oct-86	University of Connecticut	5	PG	PHO	29	185.0	8
1	Aaron Brooks	82	1885	954	817	344	42.1	121	313	38.7	...	us	14-Jan-85	University of Oregon	6	PG	CHI	30	180.0	7
2	Aaron Gordon	47	797	243	208	93	44.7	13	48	27.1	...	us	16-Sep-95	University of Arizona	R	PF	ORL	20	202.5	9
3	Adreian Payne	32	740	213	220	91	41.4	1	9	11.1	...	us	19-Feb-91	Michigan State University	R	PF	ATL	24	205.0	10

Com o DataFrame em mãos é preciso separar apenas as varáveis de interesse para o projeto em questão e qual o método de regressão que será utilizado, no caso, Regressão Linear e são configuradas as variáveis de teste e treinamento.

Figura 2.3 – Variáveis de Interesse e Regressão Linear

```
In [5]: # Campos relavantes para o treinamento
        cols = ['FGA']
        # Campo para predição
        cols_target = ['FGM']

        regression = linear_model.LinearRegression()
        x_train, x_test, y_train, y_test = train_test_split(
            data[cols], data[cols_target], test_size=0.2, random_state=4)
```

Com o DataFrame devidamente configurado é calculado o coeficiente de correlação entre as variáveis para saber quais variáveis se relacionam melhor entre si. Um valor próximo de 1 significa uma boa previsão, o que valida a escolhas dos dados em questão que possuem uma correlação de 0,987260.

Figura 2.4 – Coeficiente de Correlação

```
In [4]: # coeficiente de relação (quanto mais próximo de 1.0 ou -1.0, melhor atributo)
        data.corr()
```

Out[4]:

	Games Played	MIN	PTS	FGA	FGM	FG%	3PM	3PA	3P%	FTM	...	BLK	TOV	PF	
Games Played	1.000000	0.868801	0.715884	0.732333	0.727393	0.390008	0.492188	0.511725	0.206241	0.564018	...	0.444156	0.698051	0.855979	0.746
MIN	0.868801	1.000000	0.917161	0.925566	0.920816	0.321441	0.608622	0.624080	0.281137	0.772047	...	0.499550	0.862274	0.866298	0.922
PTS	0.715884	0.917161	1.000000	0.988967	0.989577	0.289329	0.630076	0.637590	0.302480	0.908683	...	0.446003	0.875246	0.740206	0.942
FGA	0.732333	0.925566	0.988967	1.000000	0.987260	0.249937	0.632655	0.650849	0.316259	0.864300	...	0.410698	0.876600	0.747587	0.919
FGM	0.727393	0.920816	0.989577	0.987260	1.000000	0.326304	0.558118	0.565509	0.262973	0.863444	...	0.491525	0.862230	0.764479	0.953
FG%	0.390008	0.321441	0.289329	0.249937	0.326304	1.000000	-0.031659	-0.059201	-0.079547	0.240740	...	0.425336	0.247373	0.397976	0.396
3PM	0.492188	0.608622	0.630076	0.632655	0.558118	-0.031659	1.000000	0.988107	0.556547	0.473934	...	-0.067063	0.519542	0.388288	0.464
3PA	0.511725	0.624080	0.637590	0.650849	0.565509	-0.059201	0.988107	1.000000	0.552525	0.489771	...	-0.075948	0.545271	0.405077	0.464
3P%	0.206241	0.281137	0.302480	0.316259	0.262973	-0.079547	0.556547	0.552525	1.000000	0.203675	...	-0.216722	0.222589	0.088544	0.167
FTM	0.564018	0.772047	0.908683	0.864300	0.863444	0.240740	0.473934	0.489771	0.203675	1.000000	...	0.419897	0.825583	0.605315	0.856

Para exibir a regressão é setado a variável ‘model’ para receber o modelo de Regressão Linear, e realizado o treinamento do modelo através do comando ‘.fit()’.

Figura 2.5 – Aplicação e Treinamento da Regressão Linear

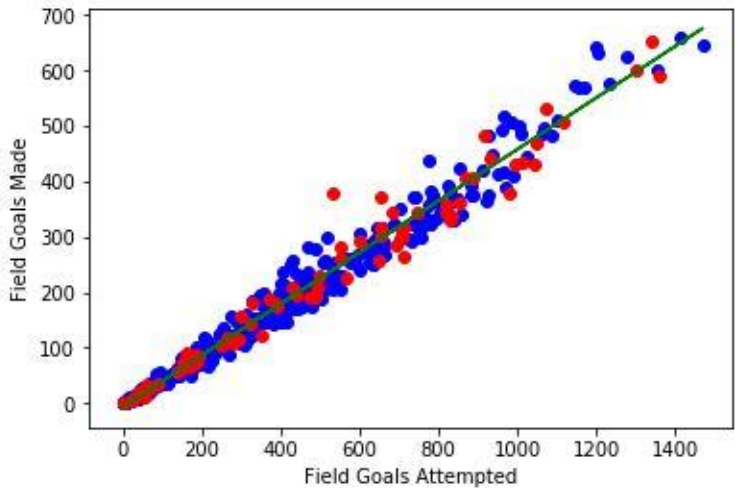
```
In [7]: model = linear_model.LinearRegression()
        model.fit(x_train, y_train)

Out[7]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
                         normalize=False)
```

Com a regressão executada agora resta apenas mostrar graficamente o seu resultado.

Figura 2.6 – Gráfico da Tentativa x Sucesso nos Arremessos

```
In [23]: plt.scatter(x_train['FGA'], y_train, color='blue')
         plt.scatter(x_test['FGA'], y_test, color='red')
         plt.plot(x_train['FGA'], model.predict(x_train), color='green')
         plt.xlabel('Field Goals Attempted')
         plt.ylabel('Field Goals Made')
         plt.show()
```



2.3 Validação e Testes

Agora para fazer uma análise dos resultados obtidos utiliza-se a variável ‘regression’ para executar novamente o treinamento com 80% dos dados e a previsão dos 20% restantes.

Figura 2.7 – Treinamento e Previsão para Análise

```
In [13]: # Executa Treinamento com 80% dos dados disponíveis
         regression.fit(x_train, y_train)

         # Faz previsão dos 20% dos dados que não entraram no treinamento
         output = regression.predict(x_test)
         output

Out[13]: array([[405.68218522],
                [ 72.80750642],
                [ 12.91791986],
                [297.04526076],
                [ 63.98655956],
                [257.58313008],
                [146.62490381],
                [132.69709298],
                [  7.81105589],
                [  5.48975408],
                [ -5.18823422],
                [223.69212373],
                [616.4563891 ],
                [ 82.09271364],
                [ 80.2356722 ],
```

Agora verifica-se a qualidade das previsões obtidas através do R^2.

Figura 2.8 – Análise da Previsão

```
In [14]: # Verifica a qualidade da previsão
         score = r2_score(y_test, output)
         score

Out[14]: 0.9695164460253392
```

Como a qualidade da regressão foi próxima de 1 significa que os resultados obtidos foram de muito boa qualidade, então é feita a predição para uma única entrada, no caso um jogador que faça 201 arremessos.

Figura 2.9 – Previsão para 201 Arremessos

```
In [15]: # Usa o treinamento para fazer uma previsão de um dado novo
         df_new_attempted_info = pd.DataFrame(
             [(201)], columns=['FGA'])

         output2 = regression.predict(df_new_attempted_info)
         output2

Out[15]: array([[87.19957761]])
```

Como pode se ver o resultado obtido foi próximo de um valor real de um jogador validando a qualidade do resultado.

Figura 2.10 – Jogador Real com 201 arremessos

	Name	Games Played	MIN	PTS	FGA	FGM
20	Andre Roberson	67	1286	228	201	92

Como dito anteriormente um coeficiente de correlação próximo de 1 gera previsões de boa qualidade e coeficientes próximo de 0 irá gerar previsões de baixa qualidade. Para corroborar essa ideia foram realizadas previsões com dados com coeficientes bons e ruins.

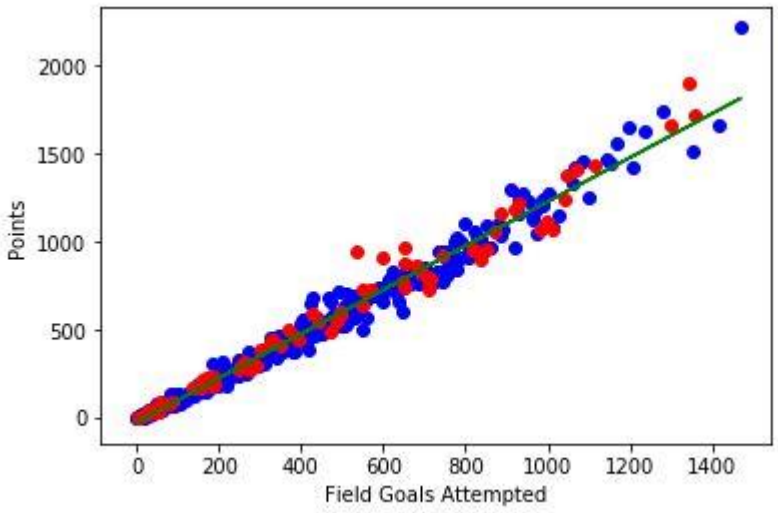
A correlação entre Tentativas de Arremesso (FGA) e quantidade de Pontos (PTS) é de 0.988967 que gera o gráfico a seguir.

Figura 2.11 – Gráfico Tentativa de Arremesso x Quantidade de Pontos

```
In [7]: model = linear_model.LinearRegression()
        model.fit(x_train, y_train)

        y_pred = model.predict(x_test)
        accuracy = model.score(x_test, y_test)

        plt.scatter(x_train['FGA'], y_train, color='blue')
        plt.scatter(x_test['FGA'], y_test, color='red')
        plt.plot(x_train['FGA'], model.predict(x_train), color='green')
        plt.xlabel('Field Goals Attempted')
        plt.ylabel('Points')
        plt.show()
```



Agora é feita a verificação da qualidade das previsões e realizada uma previsão para 201 arremessos para checar com um dado real.

Figura 2.12 – Previsão para 201 Arremessos

```
In [9]: # Verifica a qualidade da previsão
        score = r2_score(y_test, output)
        score
```

```
Out[9]: 0.9732070786718808
```

```
In [10]: # Usa o treinamento para fazer uma previsão de um dado novo
         df_new_attempted_info = pd.DataFrame(
             [(201)], columns=['FGA'])

         output2 = regression.predict(df_new_attempted_info)
         output2
```

```
Out[10]: array([[228.54789686]])
```

Como pode se ver o resultado obtido foi muito próximo de um valor real de um jogador validando a qualidade do resultado.

Figura 2.13 – Jogador Real com 201 Arremessos

	Name	Games Played	MIN	PTS	FGA	FGM
20	Andre Roberson	67	1286	228	201	92

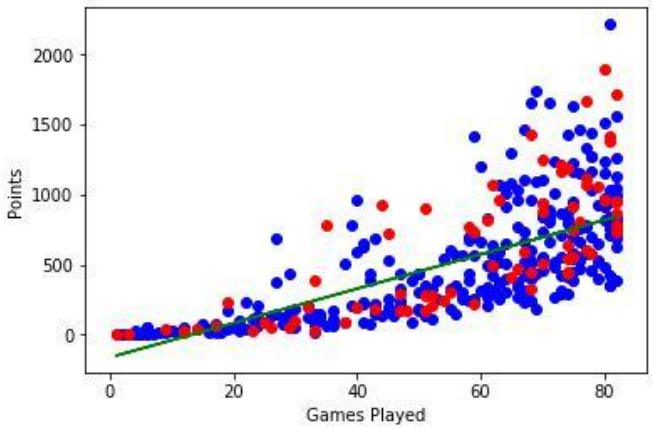
Para mostra que o coeficiente baixo gera previsões ruins foi rodado o programa pra as tupla de dados ‘Games Played’ x ‘PTS’ que possuem um coeficiente de 0,715884.

Figura 2.14 – Gráfico Quantidade de Jogos x Quantidade de Pontos

```
In [7]: model = linear_model.LinearRegression()
        model.fit(x_train, y_train)

        y_pred = model.predict(x_test)
        accuracy = model.score(x_test, y_test)

        plt.scatter(x_train['Games Played'], y_train, color='blue')
        plt.scatter(x_test['Games Played'], y_test, color='red')
        plt.plot(x_train['Games Played'], model.predict(x_train), color='green')
        plt.xlabel('Games Played')
        plt.ylabel('Points')
        plt.show()
```



Agora é feita a verificação da qualidade das previsões e realizada uma previsão para 67 jogos para checar com um dado real.

Figura 2.15 – Previsão para 67 Jogos

```
In [9]: # Verifica a qualidade da previsão
        score = r2_score(y_test, output)
        score

Out[9]: 0.5000342775365391

In [10]: # Usa o treinamento para fazer uma previsão de um dado novo
         df_new_attempted_info = pd.DataFrame(
             [(67)], columns=['Games Played'])

         output2 = regression.predict(df_new_attempted_info)
         output2

Out[10]: array([[658.48126712]])
```

Como pode se ver a qualidade das previsões tem um valor muito baixo e o resultado obtido foi muito superior ao valor real de um jogador, validando a má qualidade do resultado.

Figura 2.16 – Jogador Real com 67 Jogos

	Name	Games Played	MIN	PTS	FGA	FGM
20	Andre Roberson	67	1286	228	201	92

Há a possibilidade de alterar o método de previsão da regressão linear realizando o método dos mínimos quadrados “na mão” ao invés de utilizar bibliotecas prontas. Para isso primeiro é preciso calcular a medida da correlação normalizada entre os dados.

Figura 2.17 – Correlação Normalizada

```
In [7]: #Medida da correlação usando a Correlação Normalizada
        def correlation(x, y):
            x1 = (x - np.mean(x))
            y1 = (y - np.mean(y))

            num = np.sum(x1 * y1)
            div = np.sqrt( np.sum(x1 ** 2) * np.sum(y1 ** 2) )

            return num/div
```

Em seguida é necessário calcular os coeficientes b0 e b1 que auxiliam na previsão do resultado.

Figura 2.18 – Calculo dos Coeficientes b0 e b1

```
In [8]: #Cálculo dos Coeficientes b0 e b1
        b1 = correlation(train[:,0], train[:,1]) * np.std(train[:,1]) / np.std(train[:,0])
        b0 = np.mean(train[:,1]) - (b1 * np.mean(train[:,0]))
        print(b0, b1)

        -4.96161617706494 0.4574613624770486
```

Com isso programado é possível plotar o gráfico utilizando essa nova metodologia. Para verificar se há alguma mudança significativa em relação ao método anterior foi escolhida as tuplas ‘FGA’ x ‘FDM’ e ‘FGA’ x ‘PTS’.

Figura 2.19 – Gráfico da Tentativa x Sucesso nos Arremessos

```
In [10]: #Validação
         pred_made = b0 + (b1 * val[:,0])
         rmse = RMSE(pred_made, val[:,1])

         print(rmse)

         # Plot os pesos preditos de acordo com o set de validação
         plt.plot(val[:,0], val[:,1], linestyle='None', marker='o', markersize=5)
         plt.plot(val[:,0], pred_made)
         plt.title('Linear Regression - Field Goals Made x Attempted')
         plt.xlabel('Field Goals Attempted')
         plt.ylabel('Field Goals Made')

         24.898327235813316

Out[10]: Text(0, 0.5, 'Field Goals Made')
```

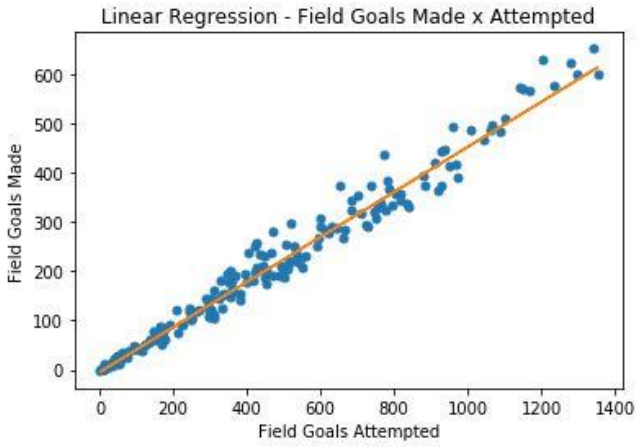


Figura 2.20 – Gráfico Tentativa de Arremesso x Quantidade de Pontos

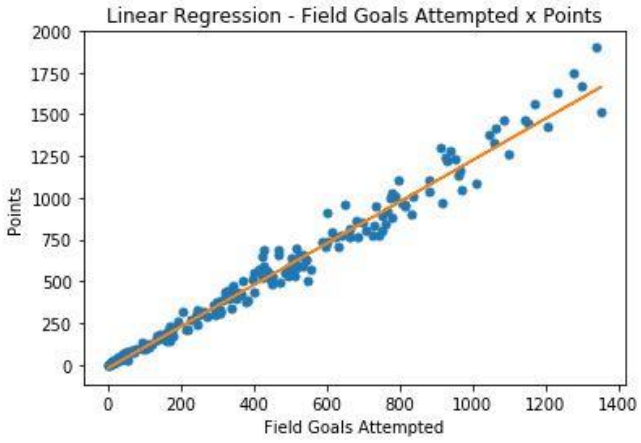
```
In [10]: #Validação
         pred_made = b0 + (b1 * val[:,1])
         rmse = RMSE(pred_made, val[:,0])

         print(rmse)

         # Plot os pesos preditos de acordo com o set de validação
         plt.plot(val[:,1], val[:,0], linestyle='None', marker='o', markersize=5)
         plt.plot(val[:,1], pred_made)
         plt.title('Linear Regression - Field Goals Attempted x Points')
         plt.xlabel('Field Goals Attempted')
         plt.ylabel('Points')

         64.62902439178922

Out[10]: Text(0, 0.5, 'Points')
```



Como pode ser visto nas próximas anteriores, não houve mudanças relevantes em nenhum dos testes. Assim como a previsão para um valor de 201 arremessos apresenta um valor muito próximo ao obtido anteriormente (87.19957761).

Figura 2.9 – Previsão Mínimos Quadrados para 201 Arremessos

```
In [9]: #Calculo da predição dos arremessos acertados pelas tentativas de arremesso
        def compute_single_made(attempt):
            return b0 + b1*attempt

        print(compute_single_made(201))

        86.98811768082183
```

Portanto conclui-se que os dois métodos podem ser utilizados sem medo pois são praticamente equivalentes no sucesso em realizar previsões em regressão linear.

2.4 Sobre Conceitos

a) Para um teste, em específico, explicar a aplicação do conceito de Correlação.

A correlação entre duas variáveis é o fato de que uma delas está de alguma forma relacionada a outra, ou seja, se alterar o valor de uma automaticamente há uma alteração na segunda. E o grau de relacionamento linear entre essas variáveis é dado pelo Coeficiente de Correlação Linear (r) que pode assumir o valor dentro do intervalo [-1,1] sendo quanto mais próximo de -1 ou 1 mais forte é a correlação, e quanto mais próximo de 0 mais fraca.

Como visto no tópico anterior nas figuras 2.12 e 2.13 uma tupla com o coeficiente próximo de 1 gera previsões mais corretas como no caso da tupla 'FGA' x 'PTS' que com um coeficiente de 0.988967, obteve uma previsão de 228.5789686 pontos para 201 arremessos. Resultado extremamente próximo de resultado real de 228 pontos em 201 arremessos.

b) Para um teste, em específico, explicar a aplicação do Gráfico de Dispersão.

O Gráfico de Dispersão é aplicado quando se quer pontuar dados em um eixo vertical e horizontal com a intenção de exibir quanto uma variável é afetada por outra. Se os pontos estão próximos a formar uma linha reta no gráfico de dispersão, as duas variáveis possuem uma alta correlação. Se os marcadores estiverem igualmente distribuídos no gráfico de dispersão, a correlação é baixa, ou zero.

Como visto no tópico anterior na figura 2.11 uma tupla com o coeficiente próximo de 1 no caso da tupla 'FGA' x 'PTS' que com um coeficiente de 0.988967, o gráfico resultante possui seus pontos muito próximos de formar uma linha reta no gráfico, como era esperado.

3. Clustering com K-Means

3.1 Planejamento

a) Breve descrição do contexto do problema.

Algo muito praticado por fãs de modalidades esportivas é o acompanhamento dos resultados obtidos pelos jogadores durante a temporada. Com isso foi decidido buscar uma forma de agrupar os jogadores eleitos o Melhor Jogador da Semana na NBA de acordo com determinados status de interesse no momento. Para isso foi utilizada uma base de dados contendo os jogadores eleitos desde a temporada 1984-85 até a temporada 2017-2018.

b) Por que está usando Clustering para o Problema.

Clustering é o conjunto de técnicas de prospeção de dados (data mining) que visa fazer agrupamentos automáticos de dados segundo o seu grau de semelhança. O que se enquadra perfeitamente na proposta de agrupar perfis de jogadores de acordo com um status de interesse em comum.

c) Variáveis do problema (nomes, tipos, domínio de valores).

Name: String - Nome dos jogadores

Games Played: Inteiro - Quantidade de jogos realizados na temporada

PTS: Inteiro - Quantidade de pontos marcados durante a temporada

FGM: Inteiro - Quantidade de cestas (não lance livre) convertidas

3P%: Inteiro - Porcentagem de arremessos de 3 pontos convertidos

FT%: Inteiro - Porcentagem de lances livres convertidos

REB: Inteiro - Quantidade de rebotes executados na temporada

AST: Inteiro - Quantidade de assistências executadas na temporada

STL: Inteiro - Quantidade de roubadas de bola executadas na temporada

PF: Inteiro - Quantidade de faltas pessoais cometidas durante a temporada

3.2 Implementação

O conjunto de dados obtidos originalmente pelo link https://www.kaggle.com/jacobbaruch/nba-player-of-the-week possuía atletas com dados faltantes, para resolver este problema os dados foram tratados por um algoritmo em python para remover os dados defeituosos gerando o arquivo “players_stats_new.xls”.

Figura 3.1 – Algoritmo de Tratamento dos Dados

```
import csv
import os
import pandas
import matplotlib.pyplot as plt

data = csv.reader(open("players_stats.csv","r"))
gerado = csv.writer(open("players_stats_new.csv","w"),lineterminator='\r')
path =  os.path.dirname(os.path.realpath(__file__))
dir = os.listdir(path)
cont = 0
cont = 0

for rows in data:
    if rows[24] == '' or rows[29] == '' or rows[30] == '' or rows[32] == '':
        next
    else:
        gerado.writerow(rows)
```

Após tratar os dados o próximo passo é importar as bibliotecas que serão utilizadas e ler a base de dados na forma de um DataFrame chamado ‘nba’.

Figura 3.2 – Importação de Bibliotecas e Leitura dos Dados

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
```

In [2]:
```
#Leitura dos dados do dataset
nba =  pd.read_csv('players_stats.csv')
nba.head()
```

Out[2]:

	Name	Games Played	MIN	PTS	FGM	FGA	FG%	3PM	3PA	3P%	...	Age	Birth_Place	Birthdate	Collage	Experience	Height	Pos	Team	Weight	
0	AJ Price	26	324	133	51	137	37.2	15	57	26.3	...	29.0	us	October 7, 1986	University of Connecticut	5	185.0	PG	PHO	81.45	23
1	Aaron Brooks	82	1885	954	344	817	42.1	121	313	38.7	...	30.0	us	January 14, 1985	University of Oregon	6	180.0	PG	CHI	72.45	2
2	Aaron Gordon	47	797	243	93	208	44.7	13	48	27.1	...	20.0	us	September 16, 1995	University of Arizona	R	202.5	PF	ORL	99.00	24
3	Adreian Payne	32	740	213	91	220	41.4	1	9	11.1	...	24.0	us	February 19, 1991	Michigan State University	R	205.0	PF	ATL	106.65	25

Com o DataFrame em mãos é preciso dar uma limpada nos dados que não tem funcionalidade para o projeto em questão, assim são retiradas as variáveis “lixo” deixando apenas as varáveis de interesse.

Figura 3.3 – Variáveis de Interesse

In [3]:
```
#Removendo as variáveis que não tem interesse
nba = nba.drop(['MIN','FGA','FG%','3PM','3PA','FTM','FTA','OREB','DREB','BLK','TOV','EFF','AST/TOV','STL/TOV','Age',
                'Birth_Place','Birthdate','Collage','Experience','Height','Pos','Team','Weight','BMI'], axis=1)
#Variáveis de interesse 'Name','Games Played', 'PTS','FGM', '3P%', 'FT%', 'REB','AST','STL','PF'
nba
```

Out[3]:

	Name	Games Played	PTS	FGM	3P%	FT%	REB	AST	STL	PF
0	AJ Price	26	133	51	26.3	66.7	32	46	7	15
1	Aaron Brooks	82	954	344	38.7	83.3	166	261	54	189
2	Aaron Gordon	47	243	93	27.1	72.1	169	33	21	83
3	Adreian Payne	32	213	91	11.1	65.2	162	30	19	88
4	Al Horford	76	1156	519	30.6	75.9	544	244	68	121

Com o DataFrame devidamente configurado é calculado o coeficiente de correlação entre as variáveis para saber quais variáveis se relacionam melhor entre si.

Figura 3.4 – Coeficiente de Correlação

```
In [4]: # coeficiente de relação (quanto mais próximo de 1.0 ou -1.0, melhor atributo)
        nba.corr()
```

Out[4]:

	Games Played	PTS	FGM	3P%	FT%	REB	AST	STL	PF
Games Played	1.000000	0.727973	0.739667	0.218361	0.341861	0.681647	0.541304	0.684127	0.860430
PTS	0.727973	1.000000	0.990487	0.290644	0.340922	0.696554	0.728515	0.807294	0.753363
FGM	0.739667	0.990487	1.000000	0.254783	0.315404	0.739720	0.704545	0.789740	0.775502
3P%	0.218361	0.290644	0.254783	1.000000	0.317313	-0.061888	0.292613	0.280100	0.099524
FT%	0.341861	0.340922	0.315404	0.317313	1.000000	0.106736	0.267797	0.258761	0.218891
REB	0.681647	0.696554	0.739720	-0.061888	0.106736	1.000000	0.348135	0.548334	0.810851
AST	0.541304	0.728515	0.704545	0.292613	0.267797	0.348135	1.000000	0.766975	0.527894
STL	0.684127	0.807294	0.789740	0.280100	0.258761	0.548334	0.766975	1.000000	0.710023
PF	0.860430	0.753363	0.775502	0.099524	0.218891	0.810851	0.527894	0.710023	1.000000

Para realizar a clusterização dos dados, tem se que utilizar apenas os dados numéricos da base de dados. Para isso é feito uma seleção das colunas que se enquadram nesse quesito dentro do DataFrame ‘Xnba’.

Figura 3.5 – Seleção dos Dados Numéricos

```
In [5]: #Seleção dos dados numericos do dataset contidos nas colunas 1 à 9
        Xnba = nba.iloc[:, 1:10].values
        Xnba
```

```
Out[5]: array([[  26.,  133.,   51., ...,   46.,    7.,   15.],
               [  82.,  954.,  344., ...,  261.,   54.,  189.],
               [  47.,  243.,   93., ...,   33.,   21.,   83.],
               ...,
               [  71., 1143.,  454., ...,  153.,   69.,  175.],
               [  73.,  606.,  240., ...,  178.,   80.,  170.],
               [  16.,   28.,   11., ...,    5.,    2.,    6.]])
```

Agora finalmente é iniciada a clusterização em si, primeiramente precisa setar a quantidade de clusters que serão utilizados e qual o método de inicialização, após isso é aplicado o kmeans ao ‘Xnba’ para realizar a clusterização e é exibido os centroides do clusters.

Figura 3.6 – Clusterização

```
In [6]: # Parametrizando K-Means com 4 Clusters e Metodo de Inicialização Randômica
        kmeans = KMeans(n_clusters = 4, init = 'random')

In [7]: # Aplicando o K-Means
        kmeans.fit(Xnba)

Out[7]: KMeans(algorithm='auto', copy_x=True, init='random', max_iter=300,
            n_clusters=4, n_init=10, n_jobs=None, precompute_distances='auto',
            random_state=None, tol=0.0001, verbose=0)

In [8]: # Listando os Centroids para cada um dos 04 Clusters
        kmeans.cluster_centers_

Out[8]: array([[  27.24022346,  101.88268156,   38.40782123,   20.81620112,
                  63.02569832,   55.60335196,   24.16759777,   11.3575419 ,
                  35.22346369],
               [  71.48245614,  844.69298246,  317.84210526,   29.33947368,
                  75.29912281,  357.71052632,  176.4122807 ,   62.77192982,
                 159.5       ],
               [  62.93288591,  439.69798658,  166.69127517,   26.30671141,
                  72.77248322,  224.08053691,  100.79865772,   37.77852349,
                 115.12080537],
               [  74.47916667, 1374.70833333,  507.08333333,   31.53958333,
                  79.41458333,  466.3125    ,  307.20833333,   87.77083333,
                 168.45833333]])
```

Em seguida é calculada a distância entre os dados de cada linha do DataFrame (instância) e cada um dos 4 clusters criados.

Figura 3.7 – Calculo da Distância das Instâncias aos Centroides

```
In [11]: # Executa K-Means para agrupar os dados e retorna Tabela de Distancias
         # Calcula a distancia entre os dados de cada linha (instância) e cada um dos 4 clusters
         distance = kmeans.fit_transform(Xnba)
         distance

Out[11]: array([[ 398.23735273,  852.61299287, 1428.04813088,   51.96677839],
                [ 576.69008781,  240.31170731,  545.52712425,  958.75398151],
                [ 229.58391882,  690.63662431, 1276.0891348 ,  197.86980565],
                ...,
                [ 927.67286578,  509.83647723,  399.37449369, 1333.9678508 ],
                [ 348.82495388,  290.80056499,  825.30947122,  736.99278775],
                [ 513.55798807,  971.31354677, 1548.79021469,   98.48906168]])
```

Na variável ‘labels’ é guardada a numeração do cluster em que cada instância foi alocada.

Figura 3.8 – Cluster em que a Instância se Encontra

```
In [12]: # Para cada uma das instâncias, a qual dos 4 clusters ela pertence..0, 1, 2 ou 3 ?
         labels = kmeans.labels_
         labels
```

```
Out[12]: array([3, 1, 3, 3, 2, 1, 0, 0, 3, 0, 0, 0, 0, 3, 3, 1, 1, 3, 3, 2, 0, 0,
                0, 0, 3, 0, 3, 2, 0, 2, 1, 0, 3, 3, 0, 1, 3, 0, 1, 0, 1, 0, 3, 0,
                2, 1, 1, 1, 0, 1, 3, 0, 1, 3, 3, 0, 2, 3, 3, 0, 3, 0, 1, 1, 0, 3,
                1, 0, 0, 0, 0, 1, 0, 3, 3, 1, 2, 3, 0, 1, 3, 3, 3, 0, 0, 1, 3, 0,
                1, 1, 3, 2, 0, 1, 3, 1, 0, 0, 3, 3, 0, 1, 0, 3, 3, 1, 1, 3, 2, 2,
                1, 1, 1, 2, 1, 0, 0, 3, 0, 1, 2, 0, 1, 3, 3, 1, 0, 3, 3, 1, 3, 2,
                3, 3, 1, 3, 1, 3, 3, 3, 2, 2, 1, 3, 1, 3, 1, 1, 3, 3, 3, 3, 3, 3,
                0, 1, 1, 1, 3, 1, 3, 0, 3, 3, 2, 2, 1, 3, 2, 3, 3, 1, 1, 0, 3, 0,
                3, 0, 3, 0, 0, 0, 1, 0, 0, 3, 0, 3, 0, 0, 3, 1, 0, 0, 2, 0, 3, 3,
                3, 3, 3, 3, 0, 1, 3, 1, 3, 3, 1, 0, 0, 3, 3, 3, 2, 2, 3, 3, 0, 3,
                3, 0, 1, 3, 3, 0, 3, 2, 0, 2, 0, 0, 3, 0, 2, 3, 3, 3, 3, 0, 3, 3,
                3, 2, 3, 0, 0, 1, 3, 1, 3, 3, 1, 3, 0, 3, 1, 1, 0, 3, 3, 3, 0, 3,
                1, 1, 1, 3, 0, 1, 0, 1, 3, 0, 0, 2, 1, 0, 1, 0, 0, 2, 1, 0, 3, 0,
                3, 1, 2, 0, 0, 2, 2, 0, 0, 3, 0, 3, 3, 0, 0, 2, 3, 3, 0, 2, 1, 3,
                1, 3, 3, 1, 3, 1, 2, 1, 0, 1, 0, 0, 1, 3, 2, 1, 3, 0, 1, 1, 3, 0,
                3, 0, 3, 1, 0, 1, 0, 3, 3, 0, 0, 3, 0, 3, 1, 2, 3, 3, 3, 1, 1, 3,
                0, 3, 0, 1, 0, 1, 0, 2, 3, 0, 1, 3, 1, 0, 0, 0, 0, 3, 1, 0, 2, 3,
                2, 1, 0, 3, 3, 3, 1, 0, 3, 0, 1, 0, 0, 0, 0, 3, 3, 3, 2, 3, 0, 3,
                3, 3, 1, 0, 0, 0, 1, 3, 3, 1, 2, 1, 3, 2, 1, 3, 0, 3, 3, 3, 1, 3,
                3, 0, 3, 0, 3, 0, 3, 0, 0, 3, 0, 3, 1, 3, 0, 2, 0, 3, 0, 1, 0, 0,
                0, 1, 0, 1, 0, 0, 2, 3, 1, 1, 3, 2, 3, 0, 1, 0, 0, 3, 3, 1, 0, 1,
                1, 3, 2, 3, 0, 3, 1, 2, 3, 1, 0, 3, 3, 2, 0, 0, 1, 1, 0, 3, 3, 0,
                1, 3, 1, 2, 1, 3])
```

Para facilitar na exibição visual da clusterização é necessário adicionar uma nova coluna ao DataFrame ‘nba’ uma coluna ‘cluster’ para receber o valor de qual cluster a instância foi classificada.

Figura 3.9 – DataFrame com a Coluna ‘cluster’

```
In [13]: #Adiciona uma coluna 'cluster' ao dataframe para alocar a qual cluster a linha foi clasificada
         nba['cluster'] = labels
         nba
```

Out[13]:

	Name	Games Played	PTS	FGM	3P%	FT%	REB	AST	STL	PF	cluster
0	AJ Price	26	133	51	26.3	66.7	32	46	7	15	3
1	Aaron Brooks	82	954	344	38.7	83.3	166	261	54	189	1
2	Aaron Gordon	47	243	93	27.1	72.1	169	33	21	83	3
3	Adreian Payne	32	213	91	11.1	65.2	162	30	19	88	3
4	Al Horford	76	1156	519	30.6	75.9	544	244	68	121	2
5	Al Jefferson	65	1082	486	40.0	65.5	548	113	47	139	1
6	Alan Anderson	74	545	195	34.8	81.2	204	83	56	148	0
7	Alec Burks	27	374	121	38.2	82.2	114	82	17	64	0
8	Alex Kirk	5	4	1	0.0	100.0	1	1	0	1	3
9	Alex Len	69	432	179	33.3	70.2	454	32	34	213	0
10	Alexey Shved	42	434	133	33.8	80.7	99	106	30	48	0
11	Alexis Ajinca	68	443	181	0.0	81.8	315	47	21	151	0
12	Al-Farouq Aminu	74	412	147	27.4	71.2	342	59	70	137	0

Com todos os passos realizados falta a visualização dessa clusterização, para isso foram escolhidas duas tuplas de variáveis com o valor de suas correlações próximos de 1,00. Que foram Jogos Realizados x Pontos Marcados e Pontos Marcados x Arremessos de 2 e 3 Pontos Convertidos.

Figura 3.10 – Gráfico Jogos Realizados x Pontos Marcados

```
In [15]: # Visualizando os Clusters
x = nba['Games Played']
y = nba['PTS']
plt.scatter(x, y, s = 50, c = nba['cluster'])
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 150, c = 'red')
plt.title('Players Clusters and Centroids')
plt.xlabel('Games Played')
plt.ylabel('Points')
plt.legend()
plt.show()
```

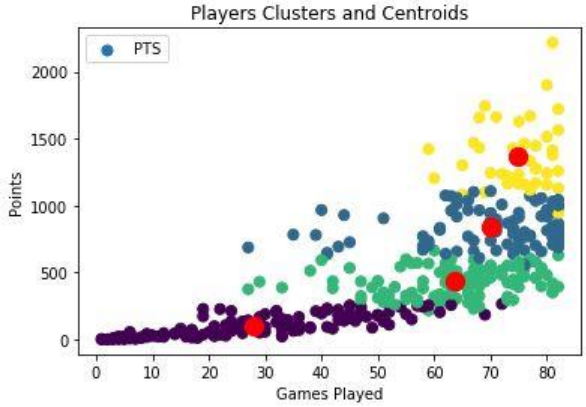
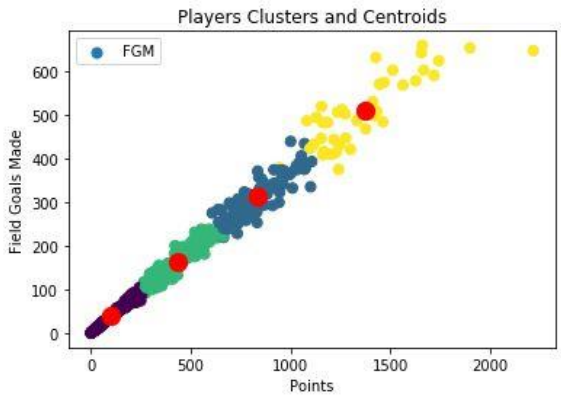


Figura 3.11 – Gráfico Pontos Marcados x Arremessos de 2 e 3 Pontos Convertidos

```
In [16]: # Visualizando os Clusters
x = nba['PTS']
y = nba['FGM']
plt.scatter(x, y, s = 50, c = nba['cluster'])
plt.scatter(kmeans.cluster_centers_[:, 1], kmeans.cluster_centers_[:, 2], s = 150, c = 'red')
plt.title('Players Clusters and Centroids')
plt.xlabel('Points')
plt.ylabel('Field Goals Made')
plt.legend()
plt.show()
```



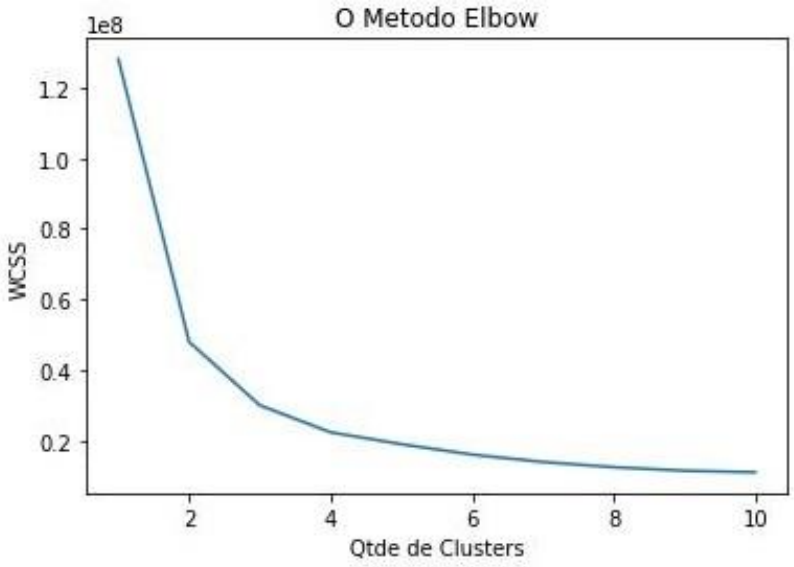
Ao término da clusterização foi executado o método Elbow para determinar a quantidade ideal de clusteres para serem utilizados para uma otimização dos resultados com essa base de dados.

Figura 3.12 – Método Elbow

```
In [18]: # Metododo de Elbow: como encontrar valor ideal de K ?
         # Basicamente o que o método faz é testar a variância dos dados em relação ao número de cluters
         # O que percebemos no Gráfico abaixo?
         # Soma dos erros quadráticos de cada cluster (WCSS) cai e se estabiliza, à medida que aumenta a qtd de clusters
         wcss = []
         maxclusters = 11
         for i in range(1, maxclusters):
          kmeans = KMeans(n_clusters = i, init = 'random')
          kmeans.fit(Xnba)
          print(i,kmeans.inertia_)
          wcss.append(kmeans.inertia_)
         plt.plot(range(1, maxclusters), wcss)
         plt.title('O Metodo Elbow')
         plt.xlabel('Qtde de Clusters')
         plt.ylabel('WCSS') #within cluster sum of squares
         plt.show()
```

Figura 3.13 – Gráfico Elbow

```
1 128061360.45765308
2 48038990.66536994
3 30074951.957812674
4 22439261.90874303
5 19091883.41875273
6 16134522.621431755
7 14047667.101484755
8 12568539.852315769
9 11579517.637209805
10 11130617.707704846
```



3.3 Validação e Testes

Analisando o gráfico Elbow pode-se concluir que a quantidade ideal de clusters para essa base de dados é 3 clusters. Sendo assim o código foi compilado novamente dessa vez com 3 clusters.

Figura 3.14 – Kmeans Iniciado com 3 Clusters

```
In [6]: # Parametrizando K-Means com 3 Clusters e Metodo de Inicialização Randômico
        kmeans = KMeans(n_clusters = 3, init = 'random')
```

Agora com 3 clusters percebe-se uma melhor unidade nas tuplas selecionadas anteriormente.

Figura 3.15 – Gráfico Jogos Realizados x Pontos Marcados

```
In [15]: # Visualizando os Clusters
x = nba['Games Played']
y = nba['PTS']
plt.scatter(x, y, s = 50, c = nba['cluster'])
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 150, c = 'red')
plt.title('Players Clusters and Centroids')
plt.xlabel('Games Played')
plt.ylabel('Points')
plt.legend()
plt.show()
```

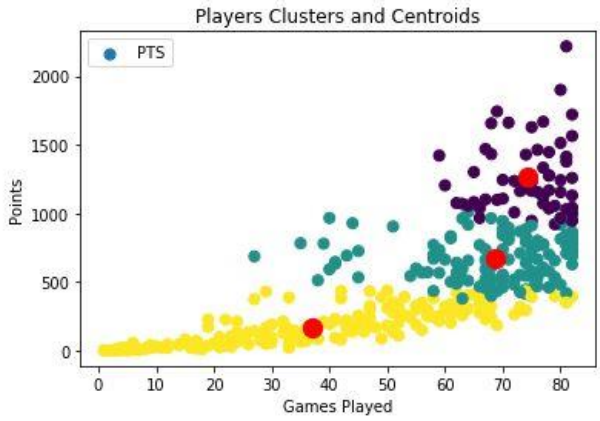
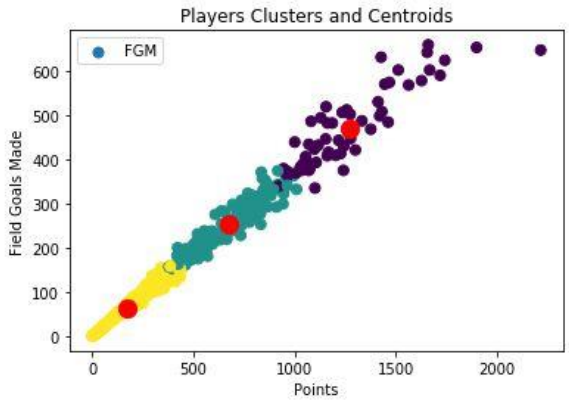


Figura 3.16 – Gráfico Pontos Marcados x Arremessos de 2 e 3 Pontos Convertidos

```
In [20]: # Visualizando os Clusters
x = nba['PTS']
y = nba['FGM']
plt.scatter(x, y, s = 50, c = nba['cluster'])
plt.scatter(kmeans.cluster_centers_[:, 1], kmeans.cluster_centers_[:, 2], s = 150, c = 'red')
plt.title('Players Clusters and Centroids')
plt.xlabel('Points')
plt.ylabel('Field Goals Made')
plt.legend()
plt.show()
```

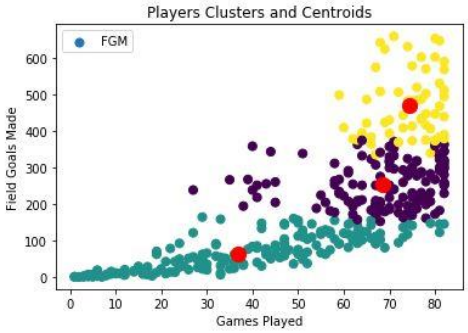


Para verificar como o coeficiente de correlação afeta na clusterização foi rodado alguns testes com outras tuplas de dados da base. Que serão mostrados a seguir.

Uma tupla com um bom coeficiente é a tupla Jogos Realizados x Arremessos de 2 e 3 Pontos Convertidos, o que corrobora com o fato de quanto melhor o o coeficiente melhor a clusterização.

Figura 3.17 - Gráfico Jogos Realizados x Arremessos de 2 e 3 Pontos Convertidos

```
In [19]: # Visualizando os Clusters
x = nba['Games Played']
y = nba['FGM']
plt.scatter(x, y, s = 50, c = nba['cluster'])
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 2], s = 150, c = 'red')
plt.title('Players Clusters and Centroids')
plt.xlabel('Games Played')
plt.ylabel('Field Goals Made')
plt.legend()
plt.show()
```



E o contrário também é comprovado ao se escolher tuplas com um coeficiente de correlação baixo reproduzem uma clusterização precária.

Figura 3.18 - Gráfico Jogos Realizados x Rebotes

```
In [16]: # Visualizando os Clusters
x = nba['Games Played']
y = nba['REB']
plt.scatter(x, y, s = 50, c = nba['cluster'])
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 5], s = 150, c = 'red')
plt.title('Players Clusters and Centroids')
plt.xlabel('Games Played')
plt.ylabel('Rebounds')
plt.legend()
plt.show()
```

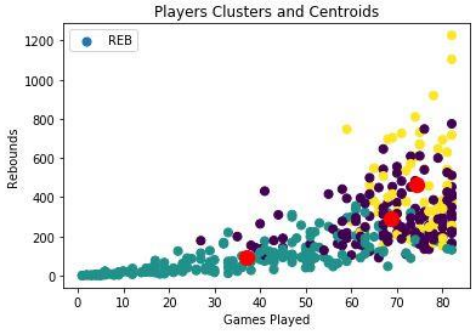


Figura 3.19 - Gráfico Jogos Realizados x Faltas Pessoais Cometidas

In [17]:
```
# Visualizando os Clusters
x = nba['Games Played']
y = nba['PF']
plt.scatter(x, y, s = 50, c = kmeans.labels_)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 8], s = 150, c = 'red')
plt.title('Players Clusters and Centroids')
plt.xlabel('Games Played')
plt.ylabel('Personal Fouls')
plt.legend()
plt.show()
```

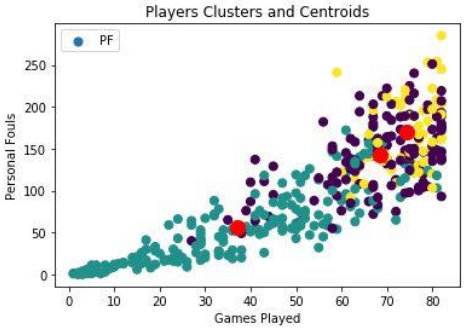


Figura 3.20 - Gráfico Jogos Realizados x Roubadas de Bola

In [18]:
```
# Visualizando os Clusters
x = nba['Games Played']
y = nba['STL']
plt.scatter(x, y, s = 50, c = nba['cluster'])
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 7], s = 150, c = 'red')
plt.title('Players Clusters and Centroids')
plt.xlabel('Games Played')
plt.ylabel('Stolen')
plt.legend()
plt.show()
```

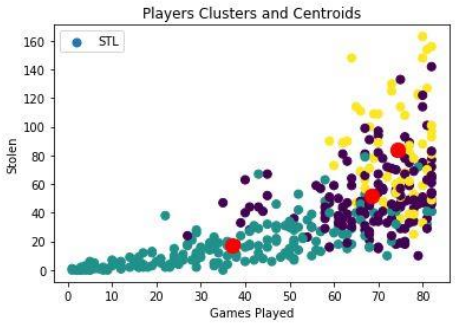
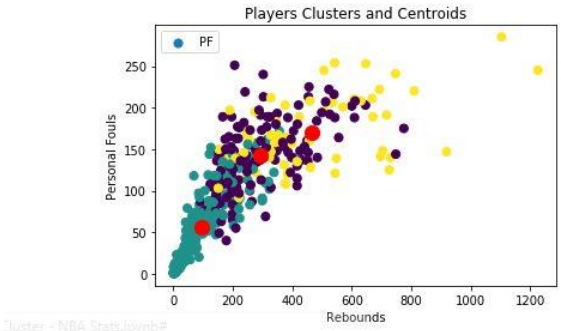


Figura 3.21 - Gráfico Rebotes x Faltas Pessoais Cometidas

```
In [21]: # Visualizando os Clusters
         x = nba['REB']
         y = nba['PF']
         plt.scatter(x, y, s = 50, c = nba['cluster'])
         plt.scatter(kmeans.cluster_centers_[:, 5], kmeans.cluster_centers_[:, 8], s = 150, c = 'red')
         plt.title('Players Clusters and Centroids')
         plt.xlabel('Rebounds')
         plt.ylabel('Personal Fouls')
         plt.legend()
         plt.show()
```



Há a possibilidade de alterar a inicialização do kmeans do método randômico para o método kmeans++ para testar se há mudanças na geração dos grupos. Porém como pode ser visto nas próximas imagens, não houve mudanças relevantes em nenhum dos testes.

Figura 3.22 – Gráfico Jogos Realizados x Pontos Marcados – kmeans++

```
In [15]: # Visualizando os Clusters
         x = nba['Games Played']
         y = nba['PTS']
         plt.scatter(x, y, s = 50, c = nba['cluster'])
         plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 150, c = 'red')
         plt.title('Players Clusters and Centroids')
         plt.xlabel('Games Played')
         plt.ylabel('Points')
         plt.legend()
         plt.show()
```

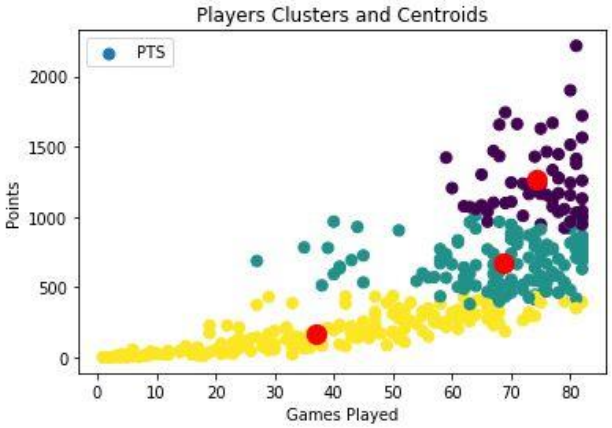
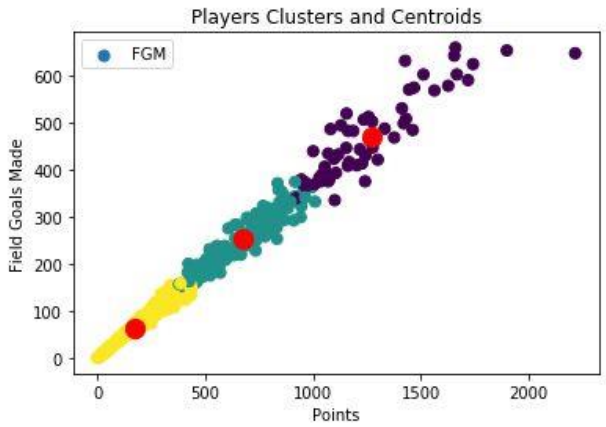


Figura 3.23 – Gráfico Pontos Marcados x Arremessos de 2 e 3 Pontos – kmeans++

```
In [16]: # Visualizando os Clusters
         x = nba['PTS']
         y = nba['FGM']
         plt.scatter(x, y, s = 50, c = nba['cluster'])
         plt.scatter(kmeans.cluster_centers_[:, 1], kmeans.cluster_centers_[:, 2], s = 150, c = 'red')
         plt.title('Players Clusters and Centroids')
         plt.xlabel('Points')
         plt.ylabel('Field Goals Made')
         plt.legend()
         plt.show()
```

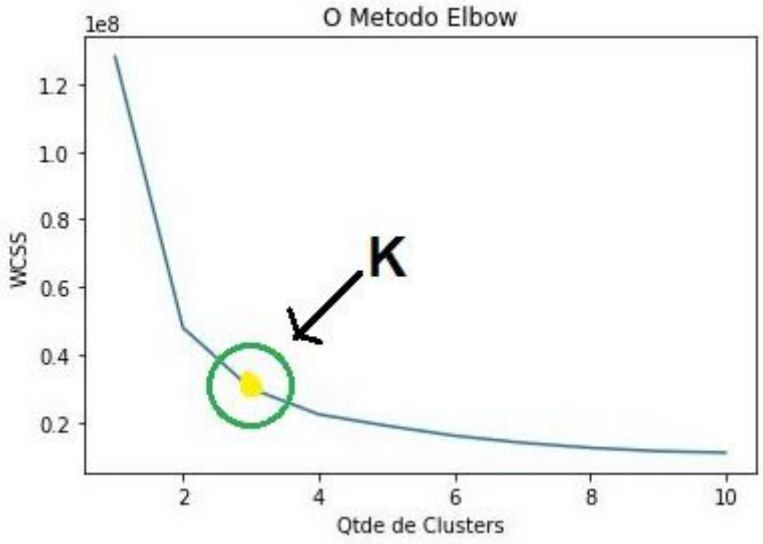


3.4. Sobre Conceitos

a) Explicar a aplicação do conceito de Elbow para a determinação do valor de K (quantidade de grupos).

O K, de K-Means, é a quantidade de centróides que serão criados e ajudará a encontrará a similaridade dos dados. O método Elbow consistem em ir crescendo a quantidade de clusters a partir de 1 e analisando o resultado melhorado a cada incremento. Quando o benefício parar de ser relevante ele entra em um modelo platô, no qual a diferença da distância é quase insignificante, deixando uma aresta que lembra um cotovelo (elbow) em um braço humano. É neste momento que se entende que o algoritmo é relevante com aquela quantidade de K e então ele deve ser usado pra segmentar os dados do gráfico.

Figura 3.24 – Gráfico Elbow



b) Para um teste, em específico, explicar a aplicação do cálculo de médias, usando inicialização randômica.

Utilizando a inicialização randômica o kmeans irá inserir o K centroides de forma aleatória. No funcionamento do algoritmo em sua primeira iteração ele irá calcular a distância média dos pontos em relação aos centroides. Feito isso os centroides mudam de posição indo para o ponto em que é a distância média de todos os pontos que se ligaram a ele. Com essa mudança de posição alguns pontos irão mudar de centroide, com isso este passo é repetido em loop até que no final nenhum ponto mude de centroide. O que significa que os centroides já estão na posição correta.

4. Redes Neurais Artificiais- Muti-layer Perception

4.1 Planejamento

a) Breve descrição do contexto do problema.

Com a revolução que o Bitcoin gerou no mundo financeiro, várias Criptomoedas foram criadas, cada um com sua particularidade. Muitas pessoas desenvolveram um interesse por estas, logo que a chance de lucro pode ser muito grande e satisfatória. O intuito deste trabalho é usar Redes Neurais para encontrar padrões para investimento nesse mercado financeiro inovador.

b) Por que está usando Rede Neural para o Problema.

Porque uma Rede Neural é um método muito eficaz para o reconhecimento de padrões, algo essencial para o investimento em qualquer tipo de mercado existente no mundo atualmente.

c) Variáveis do problema: dados de entrada (nomes, tipos, domínio de valores). É necessário normalizar?

As variáveis são as seguintes: **Simbol**, símbolo da moeda ou acrônimo utilizado para identificação, **Preu**, valor em euros de uma ação ou uma criptomoeda, e **% 1h, % 2h e % 7d**, porcentagem de variação em 1h, 2h ou 7d.

Não houve necessidade de normalização, porém foi realizado a retirada de caracteres desnecessários, como, cifrão ($), o símbolo de porcentagem (%) e virgulas (,).

d) Como o problema foi codificado na Rede Neural, na camada entrada? Justificar.

O problema foi codificado da seguinte maneira: [Preu, % 1h, % 2h, % 7d]. Pois estas são as informações principais que um investidor utiliza antes de decidir se irá investir em um criptomoeda ou não.

e) Quantos neurônios em cada camada: entrada, intermediária e saída? Justificar.

Há 2 neurônios na camada de entrada e saída, e 6 na camada intermediária, pois com esta quantidade foi obtido um bom resultado acompanhado de rapidez no processamento dos dados.

f) Qual a arquitetura da Rede Neural? Justificar.

Foi utilizado uma arquitetura de Perceptron Multicamadas, pois é necessário combinar vários neurônios para realizar tarefas mais complexas, como, encontrar padrões para o investimento em criptomoedas.

g) Qual a função de Ativação para cada camada? Justificar.

Tangente hiperbólica, pois o é a função que teve o melhor nível de acurácia e porque problema não é linearmente divisível.

h) Quais são os valores de saída (camada de saída)? 0, 1, ou outro valor? Justificar.

Os valores de saída são: ['Daytrading', 'Holding', 'Holding ou Daytrading', 'Não invista'], onde **Daytrading**, é a compra e venda de uma moeda no mesmo dia, **Holding**, é a compra de uma moeda e a esperar até que a esta alcance um lucro esperado (geralmente de 50% ou 100%), e **Não invista**, autoexplicativo.

4.2 Implementação

Figura 4.1 – Lendo o Dataset

```
In [1]: import pandas as pd
        dataset = pd.read_csv('Crypto_Currencies.csv')
```

Link do dataset: https://www.kaggle.com/acostasg/crypto-currencies

Significado de cada coluna:

Nom: nome da empresa ou cryptomoneda;

Preu: valor em euros de uma ação ou uma criptomoeda;

Volum: em euros / volume 24 horas, acumulado das transacções diárias em milhões de euros;

Simbol: símbolo da moeda ou acrônimo;

Cap de mercat: valor total de todas as moedas no momento atual;

Oferta circulant: valor na oportunidade de negócio;

% 1h, % 2h e %7d: porcentagem de variação em 1h, 2h ou 7d.

Figura 4.2 – Dataset

```
In [2]: dataset
```

Out[2]:

	Numero	Nom	Simbol	Cap de mercat	Preu	Oferta circulant	Volum 24 hores	% 1h	% 2h	% 7d
0	1	Bitcoin	BTC	$94,426,076,779	$5681.43	16,620,125	$3,867,720,000	0.43%	11.61%	29.76%
1	2	Ethereum	ETH	$31,006,174,813	$326.10	95,083,257	$1,003,690,000	0.14%	6.28%	8.99%
2	3	Ripple	XRP	$9,586,229,114	$0.248345	38,600,451,446*	$380,544,000	0.96%	-4.39%	4.70%
3	4	Bitcoin Cash	BCH	$5,197,422,996	$311.32	16,694,525	$345,263,000	-1.09%	-1.01%	-15.23%
4	5	Litecoin	LTC	$3,086,414,300	$57.85	53,350,082	$510,038,000	1.67%	12.10%	10.82%
5	6	Dash	DASH	$2,288,292,990	$300.36	7,618,400	$64,726,100	-0.22%	2.05%	-2.36%
6	7	NEM	XEM	$1,828,782,000	$0.203198	8,999,999,999*	$6,078,140	-0.99%	-4.06%	-5.45%
7	8	BitConnect	BCC	$1,408,070,870	$196.64	7,160,690	$16,942,200	0.69%	10.68%	39.27%
8	9	NEO	NEO	$1,394,935,000	$27.90	50,000,000*	$84,370,300	-1.27%	-4.39%	-16.82%
9	10	Monero	XMR	$1,328,610,030	$87.34	15,211,391	$43,965,700	-0.42%	-0.71%	-4.84%
10	11	IOTA	MIOTA	$1,178,023,304	$0.423821	2,779,530,283*	$18,520,500	-2.60%	-7.87%	-23.47%

Nesta sessão os dados importantes são tratados para que o cálculo seja realizado, pois no dataset utilizado neste trabalho contém caracteres desnecessários, como, o cifrão ($), o símbolo de porcentagem (%) e virgulas (,).

Figura 4.3 – Selecionando os Dados

Informações necessárias

Logo a baixo será criado uma lista com as informações(Colunas) necessárias para rede neural realizar os calculos, sendo estas:

Simbol: identificação da moedas;

Preu, % 1h, % 2h e %7d: sendo as váriaveis utilizadas neste problema.

In [3]:
```
data = []
for k in range(len(dataset)):
    try:
        data.append([dataset.iloc[k]['Simbol'],
                     float(dataset.iloc[k]['Preu'][1:].replace(',','')),
                    float(dataset.iloc[k]['% 1h'][:len(dataset.iloc[k]['% 1h'])-1]),
                    float(dataset.iloc[k]['% 2h'][:len(dataset.iloc[k]['% 2h'])-1].replace(',','')),
                    float(dataset.iloc[k]['% 7d'][:len(dataset.iloc[k]['% 7d'])-1].replace(',',''))])
    except ValueError:
        continue
```

A função *createTrainData* é responsável por criar um arquivo .csv dos dados de treinamento, sendo que, estes são adicionados manualmente no *dic.*

Figura 4.4 – Dados para Treino

In [4]:
```
def createTrainData():
    dic = {'Preu':[0.000093,0.248345,311.320000,0.203198,196.640000,0.423821,
                   0.027961,10.870000,4.860000,3.790000,3.100000,0.017882,
                   3.650000,0.003574,0.000017,0.118370,0.000057,2.55E+03,7.00E+00],
           '% 1h':[0.55,0.96,-1.09,-0.99,0.69,-2.60,0.44,-0.43,-0.09,-1.61,0.12,0.59,2.47,0.55,1.40,6.93,0.65,0.00,0.15],
           '% 2h':[81.77,-4.39,-1.01,-4.06,10.68,-7.87,22.68,-6.96,-7.06,-6.93,-6.26,-4.67,80.43,90.07,-1.70,101.17,
                   11.55,-88.60,-80.31],
           '% 7d':[-72.35,4.70,-15.23,-5.45,39.27,-23.47,35.14,-8.07,-9.74,-24.78,-17.16,10.67,613.96,30.68,76.92,10.71,
                   -18.02,-66.31,172.18],
           'Investimento':['Holding','Daytrading','Holding','Holding ou Daytrading','Não invista','Holding ou Daytrading',
                           'Não invista','Holding ou Daytrading','Holding ou Daytrading','Holding ou Daytrading',
                           'Holding ou Daytrading','Daytrading','Não invista','Não invista','Não invista', 'Não invista',
                           'Holding','Não invista','Daytrading']
          }
    csv = pd.DataFrame(dic)
    csv.to_csv('trainData.csv')

#createTrainData()
```

Logo abaixo é possível visualizar os dados de treinos gerados anteriormente.

Figura 4.5 – Dados de Treino

```
In [5]: trainData = pd.read_csv('trainData.csv')
        trainData = trainData.drop(['Unnamed: 0'], axis=1)
        trainData
```

Out[5]:

	Preu	% 1h	% 2h	% 7d	Investimento
0	0.000093	0.55	81.77	-72.35	Holding
1	0.248345	0.96	-4.39	4.70	Daytrading
2	311.320000	-1.09	-1.01	-15.23	Holding
3	0.203198	-0.99	-4.06	-5.45	Holding ou Daytrading
4	196.640000	0.69	10.68	39.27	Não invista
5	0.423821	-2.60	-7.87	-23.47	Holding ou Daytrading
6	0.027961	0.44	22.68	35.14	Não invista
7	10.870000	-0.43	-6.96	-8.07	Holding ou Daytrading
8	4.860000	-0.09	-7.06	-9.74	Holding ou Daytrading
9	3.790000	-1.61	-6.93	-24.78	Holding ou Daytrading
10	3.100000	0.12	-6.26	-17.16	Holding ou Daytrading
11	0.017882	0.59	-4.67	10.67	Daytrading
12	3.650000	2.47	80.43	613.96	Não invista
13	0.003574	0.55	90.07	30.68	Não invista
14	0.000017	1.40	-1.70	76.92	Não invista
15	0.118370	6.93	101.17	10.71	Não invista

A partir desta parte do código é onde a rede neural começa a ser construída, sendo que, na primeira sessão são selecionados a função de ativação, o tipo de solucionador, o tipo de parâmetros que serão passados, a quantidade de neurônios e o tipo de gerador randômico desta.

Figura 4.6 – Rede Neural

```
In [6]: from sklearn.neural_network import MLPClassifier

        neuralNetwork = MLPClassifier(activation='tanh', solver='lbfgs', alpha=1e-5,
                            hidden_layer_sizes=(6, 2), random_state=1)
```

```
In [7]: x = []
        y = []
        for k in range(len(trainData)):
            x.append([trainData.iloc[k]['Preu'],trainData.iloc[k]['% 1h'],
                      trainData.iloc[k]['% 2h'],trainData.iloc[k]['% 7d']])
            y.append(trainData.iloc[k]['Investimento'])
```

```
In [8]: neuralNetwork.fit(x, y)
```

```
Out[8]: MLPClassifier(activation='tanh', alpha=1e-05, batch_size='auto', beta_1=0.9,
                      beta_2=0.999, early_stopping=False, epsilon=1e-08,
                      hidden_layer_sizes=(6, 2), learning_rate='constant',
                      learning_rate_init=0.001, max_iter=200, momentum=0.9,
                      n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
                      random_state=1, shuffle=True, solver='lbfgs', tol=0.0001,
                      validation_fraction=0.1, verbose=False, warm_start=False)
```

Logo abaixo é possível ver que o teste do grupo de treinamento deu 100% de acerto.

Figura 4.7 – Teste do Treino

```
In [17]: for k in range(len(y)):
             if y[k] == neuralNetwork.predict(x)[k]: print('True')
             else: print('False')

True
True
True
True
True
True
True
True
True
True
True
True
True
True
True
True
True
True
True
True
```

A partir desde ponto o processamento do dataset é realizado.

Figura 4.8 – Processando o Dataset.

```
In [10]: test = []
         for d in data:
             test.append(d[1:])

In [11]: neuralNetwork.predict(test)

Out[11]: array(['Não invista', 'Não invista', 'Daytrading', ..., 'Não invista',
                'Não invista', 'Holding'], dtype='<U21')

In [12]: results = []
         for k in range(len(neuralNetwork.predict(test))):
             data[k].append(neuralNetwork.predict(test)[k])
             results.append(neuralNetwork.predict(test)[k])
         data = pd.DataFrame(data)
```

Logo abaixo é mostrado o resultado do cálculo da rede neural.

Figura 4.9 – Parte do Dataset com os Resultados.

```
In [18]: data
```

Out[18]:

	0	1	2	3	4	5
0	BTC	5681.430000	0.43	11.61	29.76	Não invista
1	ETH	326.100000	0.14	6.28	8.99	Não invista
2	XRP	0.248345	0.96	-4.39	4.70	Daytrading
3	BCH	311.320000	-1.09	-1.01	-15.23	Holding
4	LTC	57.850000	1.67	12.10	10.82	Não invista
5	DASH	300.360000	-0.22	2.05	-2.36	Holding
6	XEM	0.203198	-0.99	-4.06	-5.45	Holding ou Daytrading
7	BCC	196.640000	0.69	10.68	39.27	Não invista
8	NEO	27.900000	-1.27	-4.39	-16.82	Holding ou Daytrading
9	XMR	87.340000	-0.42	-0.71	-4.84	Holding
10	MIOTA	0.423821	-2.60	-7.87	-23.47	Holding ou Daytrading

4.3 Validação e Testes

Figura 4.10 – Validação dos Testes.

```
In [24]: validation = [[3273.33, -1.56, -7.02, -19.46],[0.307942, -0.98, -6.88, -24.65],[93.07, -1.15, -8.04, -23.62],
                       [92.94, 0.00, -3.37, 88.05],[7.15, -4.42, -29.32, 26.15],[0.359862, -1.66, 25.42, 131.45],
                       [0.000084, -0.80, -13.76, -27.67],[4.99, -0.63, 3.61, 20.11],[0.892281, 6.35, 30.26, 57.98],
                       [0.003856, -1.79, -12.04, -14.42]]

         for k in range(len(validation)):
             print(k,'-',validation[k],'=',neuralNetwork.predict(validation)[k])

         0 - [3273.33, -1.56, -7.02, -19.46] = Holding
         1 - [0.307942, -0.98, -6.88, -24.65] = Holding ou Daytrading
         2 - [93.07, -1.15, -8.04, -23.62] = Holding ou Daytrading
         3 - [92.94, 0.0, -3.37, 88.05] = Não invista
         4 - [7.15, -4.42, -29.32, 26.15] = Daytrading
         5 - [0.359862, -1.66, 25.42, 131.45] = Não invista
         6 - [8.4e-05, -0.8, -13.76, -27.67] = Holding ou Daytrading
         7 - [4.99, -0.63, 3.61, 20.11] = Não invista
         8 - [0.892281, 6.35, 30.26, 57.98] = Não invista
         9 - [0.003856, -1.79, -12.04, -14.42] = Holding ou Daytrading
```

Teste 0 – Pode-se perceber que o resultado esperado é muito provável de estar correto, logo que, estes dados são do Bitcoin no dia 27/11 e todo final de esta moeda tende a cair para no próximo ano ter uma grande valorização.
Teste 1, 2, 6 e 9 – Ambos os testes deram o mesmo resultado, pois estes tiveram uma queda em todas as porcentagens passadas e tem um baixo preço de mercado.
Teste 3 – Mesmo com um valor baixo de mercado, esta moeda teve uma grande valorização nos últimos 7 dias, logo, esta tem grandes chances de ter uma desvalorização em alguns dias.
Teste 4 – Esta moeda tem um valor de mercado muito baixo e desvalorizou nas últimas horas, por isso, realizar Daytrading pode ser uma ótima forma de lucrar com esta.

Teste 5 – Esta moeda tem um valor muito baixo de mercado, porém esta valorizou mais de 100% nos últimos 7 dias, por isso não é um bom investimento no dia de hoje.
Teste 7 e 8 – Estas moedas valorizaram mais de 20% nos últimos 7 dias, por este motivo, estas deveram desvalorizar em alguns dias.

4.4 Sobre conceitos

a) Explicar a aplicação do conceito de Função de Ativação e como interfere no aprendizado da rede.
As funções de ativação permitem que pequenas mudanças nos pesos e bias causem apenas uma pequena alteração no output. Estas verificam se a informação que o neurônio está recebendo é relevante para a informação fornecida ou deve ser ignorada

b) Para um teste, em específico, explicar a aplicação do modelo neural gerado.
Utilizando uma MPL, os sinais, dados inseridos na rede neural, serão multiplicados por um peso, que indica a sua influência na saída da unidade, em seguida é feita a soma ponderada dos sinais que produz um nível de atividade e o resultado desta soma é passado para uma função de ativação, Tangente hiperbólica neste caso, para obter o resultado final.

5. Conclusão

Através da Inteligência Artificial se torna possível fazer com que dados pesquisados por usuários sobre um determinado assunto possam ser exibidos de uma forma mais simplificada, resumida e com fácil entendimento, também ajuda na tomada de decisões do cotidiano por meio do processamento de dados e reconhecimento de padrões utilizados por esses dados informando assim a melhor escolha a ser feita pelo usuário.

Tendo em vista que neste trabalho foram apresentadas e exemplificadas três diferentes técnicas relacionadas a Inteligência Artificial onde são apresentados diferentes tipos de situações e dados buscando diferentes resoluções. Fica claro que pode-se utilizar Inteligência Artificial nas mais variadas situações e necessidades, tornando-se cada vez mais uma referência tecnológica para o presente e futuro.

6. Referências Bibliográficas

https://docs.tibco.com/pub/spotfire_web_player/6.0.0-november-2013/pt-BR/WebHelp/GUID-780960FA-1DCE-4E59-8EB7-54F7144DB362.html

https://www.eecis.udel.edu/~portnoi/classroom/prob_estatistica/2006_2/lecture_slides/aula20.pdf

http://www.diegonogare.net/2015/08/entendendo-como-funciona-o-algoritmo-de-cluster-k-means/

https://imasters.com.br/back-end/data-science-regressoes-com-python