# New Scan 5:47 PM

Scanned on: 17:47 November 10, 2022 UTC


Overall Similarity Score | Results Found | Total Words in Text

| | |
|---|---|
| Identical Words | 43 |
| Words with Minor Changes | 8 |
| Papaphrased Words | 0 |
| Omitted Words | 0 |

New Scan 5:47 PM

Scanned on: 17:47 November 10, 2022 UTC

# Results

Sources that matched your submitted document.

| | |
|---|---|
| Copyleaks Internal Database | 3% |
| Copyleaks Internal Database | 3% |
| Copyleaks Internal Database | 3% |
| Copyleaks Internal Database | 3% |
| Copyleaks Internal Database | 2% |
| Copyleaks Internal Database | 2% |

**IDENTICAL**

Identical matches are one to one exact wording in the text.

**MINOR CHANGES**

Nearly identical with different form, ie "slow" becomes "slowly".

**PARAPHRASED**

Close meaning but different words used to convey the same message.

Unsure about your report?

The results have been found after comparing your submitted text to online sources, open databases and the Copyleaks internal database. For any questions about the report contact us on support@copyleaks.com

Learn more about different kinds of plagiarism here

Scanned on: 17:47 November 10, 2022 UTC

# Scanned Text

Your text is highlighted according to the matched content in the results above.

IDENTICAL MINOR CHANGES PARAPHRASED

```
import numpy as np
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as plt
from matplotlib import gridspec
from IPython.display import display, HTML
from math import pi
plt.style.use('fivethirtyeight')


%matplotlib inline


from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn import linear_model, tree, svm
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from google.colab import drive


drive.mount('/content/drive')
DATA_PATH1 = '/content/drive/MyDrive/project/fifa_data_2015_to_2020.csv'
df= pd.read_csv(DATA_PATH1, sep=',')
display(df.head(5))
df.columns
def get_year(df,year):
temp_data = df.query('year==@year').reset_index()
return temp_data
df_2015 = get_year(df,2015)
df_2016 = get_year(df,2016)
df_2017 = get_year(df,2017)
df_2018 = get_year(df,2018)
df_2019 = get_year(df,2019)
df_2020 = get_year(df,2020)
l_df=[df_2015,df_2016,df_2017,df_2018,df_2019,df_2020]
Overall rating distribution by year
plt.figure(figsize = (20,7))
sb.countplot(data = df, x = 'overall', hue='year', palette = sb.cubehelix_palette(6, start=5, rot=0, dark=0.2,
light=.8, reverse=False))
plt.title('Overall rating distribution by year')
```

```
plt.legend(loc=1);
Comparison of the top 5 players from 2015 to 2020
l=[]
for i in l_df:
temp = i.head(5)[['short_name','overall','year']]
l.append(temp)
fig = plt.figure(figsize=(22,12))
plt.suptitle('Comparaison of the top 5 players from 2015 to 2020',fontsize=22)
plt.tight_layout(pad=0.5, w_pad=0.5, h_pad=0.5)
plt.subplot(231)
sb.barplot(data=l[0], x='short_name', y='overall', palette='Reds_r')
plt.ylim(85, 100)
plt.xlabel('Player Name')
plt.title('Top 5 most rated players in FIFA 2015')
plt.subplot(232)
sb.barplot(data=l[1], x='short_name', y='overall', palette='Reds_r')
plt.ylim(85, 100)
plt.xlabel('Player Name')
plt.title('Top 5 most rated players in FIFA 2016')
plt.subplot(233)
sb.barplot(data=l[2], x='short_name', y='overall', palette='Reds_r')
plt.ylim(85, 100)
plt.xlabel('Player Name')
plt.title('Top 5 most rated players in FIFA 2017')
plt.subplot(234)
sb.barplot(data=l[3], x='short_name', y='overall', palette='Reds_r')
plt.ylim(85, 100)
plt.xlabel('Player Name')
plt.title('Top 5 most rated players in FIFA 2018')
plt.subplot(235)
sb.barplot(data=l[4], x='short_name', y='overall', palette='Reds_r')
plt.ylim(85, 100)
plt.xlabel('Player Name')
plt.title('Top 5 most rated players in FIFA 2019')
plt.subplot(236)
sb.barplot(data=l[5], x='short_name', y='overall', palette='Reds_r')
plt.ylim(85, 100)
plt.xlabel('Player Name')
plt.title('Top 5 most rated players in FIFA 2020');
Best overall rating for each position by year
def get_player_noGK(df):
idx_df_gk = list(df.query('pace==0').index)
df_no_GK = df.drop(idx_df_gk)
return df_no_GK
def get_best_by_position(df):
lis=[]
temp = df.groupby(['team_position'])[['overall']].max()
for i in list(temp.index):
overall = temp.loc[i][0]
best_i = df.query('team_position==@i & overall==@overall').iloc[0]
lis.append(best_i)
best_pos = pd.DataFrame(lis)
return best_pos
lis=[]
lis.append(get_best_by_position(df_2015))
lis.append(get_best_by_position(df_2016))
lis.append(get_best_by_position(df_2017))
lis.append(get_best_by_position(df_2018))
lis.append(get_best_by_position(df_2019))
lis.append(get_best_by_position(df_2020))
best_pos = pd.concat(lis, ignore_index=True)
plt.figure(figsize=(15,6))
plt.title('Best overall rating for each position by year')
sb.barplot(data=best_pos, x='team_position', y='overall', hue='year')
plt.legend(loc='upper right', bbox_to_anchor=(1.1, 1))
plt.xlabel('Team position')
```

```
plt.ylabel('Overall')
plt.ylim(60,100);
Best Players in every position in each year
gb = best_pos.groupby('team_position')
l = [gb.get_group(x) for x in gb.groups]
for i in range(15):
display(l[i].loc[:,['short_name','team_position','overall','club','nationality','year']])
# Player overall rating prediction
# Construction of the features dataframe for the regression model


years=[2019,2018,2017,2016,2015]
features_selected = ['age','wage_eur', 'value_eur','international_reputation', 'pace','skill_moves' ,'shooting',
'passing', 'dribbling',
'defending', 'physic','skill_oa', 'attack_oa', 'movements_oa', 'power_oa', 'mentality_oa', 'gk_oa',
'defending_oa', 'trait_coef','year']


ols_xdata = df.query('year in @years')[features_selected].copy()
ols_ydata = df.query('year in @years')['overall'].copy()


ols_xdata.loc[ols_xdata.value_eur==0,'value_eur'] = ols_xdata['value_eur'].mean()
ols_xdata.loc[ols_xdata.wage_eur==0,'wage_eur'] = ols_xdata['wage_eur'].mean()


ols_xdata['isGK'] = (ols_xdata['pace']==0).astype(int)
cols = list(ols_xdata.columns)
ols_xdata
# Adding 2nd degree polynomial features to make the regression model more accurate


polynomial = PolynomialFeatures(2, include_bias=False)
ols_xdata = polynomial.fit_transform(ols_xdata)
# Train/Test data split


#xtrain, xtest, ytrain, ytest = train_test_split( ols_xdata, ols_ydata, test_size=0.2, shuffle=True,
random_state=42)
xtrain,ytrain=ols_xdata,ols_ydata


display((xtrain.shape, ytrain.shape))


x_2020 = df.query('year ==2020')[features_selected].copy()
x_2020['isGK'] = (x_2020['pace']==0).astype(int)
x_2020 = polynomial.fit_transform(x_2020)
y_2020 = np.array(df.query('year==2020')['overall'].copy()).flatten()
regression = linear_model.LinearRegression()
regression.fit(xtrain, ytrain)
ypred = regression.predict(x_2020)
predicted_2020 = np.round(regression.predict(x_2020)).astype(int)
print('Mean squared error: %.2f' % mean_squared_error(y_2020, predicted_2020))
print('Coefficient of determination: %.2f'% r2_score(y_2020, predicted_2020))
# Choosing a player from the dataset


def get_player(df,features_selected):
player= df.sample(1)
player_x = player[features_selected].copy()
player_x['isGK'] = (player_x['pace']==0).astype(int)
player_x= polynomial.fit_transform(player_x)
player_y = np.array(player['overall'].copy())
return player_x,player_y
#Predicting the overall rating of the random player
```

```
player_x,player_y = get_player(df, features_selected)
estimated_y = regression.predict(player_x)
print('Actual value y=',int(player_y), 'Predicted value=', int(np.round(estimated_y)))
plt.figure(figsize=(10,10))
sb.scatterplot(predicted_2020, y_2020, alpha=0.15)
plt.plot(y_2020, y_2020, color='r', alpha=0.5)
plt.legend(['Correct pred','True-Vs-Pred scatter'])
plt.title('Prediction of overall ratings')
plt.xlabel(' Overall Prediction')
plt.ylabel(' Overall True');
```