This is part of a three part hand-in to the M1-exam.

Link to this colab is https://colab.research.google.com/drive/1i7VYq8SgB5dFA8jNcMaVJBBHqsD43exI

Preprocessing

Importing libraries and fetching data

```
## Get data
!wget -c https://github.com/bande15/SDS_M1_EXAM/raw/master/combine_data.csv -O df.csv

## Mute Warnings
from warnings import simplefilter
simplefilter(action='ignore', category=(FutureWarning,DeprecationWarning, Warning))

#Import General Libraries
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import xgboost as xgb
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier

#Install and import special libraries
!pip install catboost
from catboost import CatBoostClassifier
```

Reading and processing data; The analysis is trying to identify and predict position from physical stats, so all oth
before missing values are dropped. Data is then converted to metric.

```
df = pd.read_csv('df.csv')
df_pre = df.drop(['Player','Pfr_ID','Year','AV'], 1)
df_pre = df_pre.dropna(how='any' , axis=0)
df = df.drop(['Player','Pfr_ID','Year','AV','Round','Pick','Team'], 1)
a = len(df)
df = df.dropna(how='any' , axis=0)
b = a - len(df)
a = len(df)
print(b , 'observations dropped due to missing values.', a ,'values left.' )
del a , b
```

```
3333 observations dropped due to missing values. 2885 values left.
```

We wil now count occurrences for each unique positon before and after observations with missing data is droppe

```
df_old = pd.read_csv('df.csv')
a = df_old.Pos.value_counts()
b = df.Pos.value_counts()
print(a , b)
```

```
WR      857
CB      630
RB      540
DE      487
DT      463
OT      460
OLB     424
OG      365
QB      350
TE      337
ILB     276
FS      229
SS      213
C       171
P       120
FB      117
K        85
S        27
EDGE     23
LS       20
G        14
NT        3
LB        3
DB        2
OL        2
Name: Pos, dtype: int64 CB      311
DE      279
WR      275
OT      273
DT      253
RB      245
OLB     240
OG      224
TE      194
ILB     130
FS      123
C       115
SS      107
FB       77
QB       12
G         8
EDGE      8
S         7
LS        2
LB        1
OL        1
Name: Pos, dtype: int64
```

From the above lists of observations per position it becomes clear that some positions have very few or no cohe only does this prevent a 'coherent' analysis, but the option of simply scrappinf the observations, or konverting mi yield truthful results. This is a essential fundamentiil problem with the following analysis, and it will be discussed handout. The short version is, excludidng positions will not only prevent the models from predicting those positio associate some combination of stats with other positions, this yielding false results.

```python
# Metric Conversion
def to_cm(F):
    return F*2.54
def to_kg(F):
    return F*0.45359237
```

```
df[['Ht','Vertical','BroadJump']] = df[['Ht','Vertical','BroadJump']].apply(to_cm)
df[['Wt']] = df[['Wt']].apply(to_kg)
df_pre[['Ht','Vertical','BroadJump']] = df_pre[['Ht','Vertical','BroadJump']].apply(to_cm)
df_pre[['Wt']] = df_pre[['Wt']].apply(to_kg)
df.info()
df.head()
```

With the newly cleaned and metrified dataframe, we can now try to make some explorative data analysis.

First off, we can look at the different teams, asking the question:

How different are the teams?

This can be examined by grouping for the 'Team' variable, and taking the mean of relevant variables.

```
df_pre.groupby('Team')['Ht','Wt','Forty','Vertical','BenchReps','BroadJump','Cone','Shuttle','Rou
```

From above output it looks like the different teams *generally* have similar statistics. This might imply that the tea
physical stats in their player picks. Thus it also becomes interesting to look at the different positions.

Similar to the above code, we can group by the position variable, 'Pos', and take the mean of the remaining rele

```
df_pre.groupby('Pos')['Ht','Wt','Forty','Vertical','BenchReps','BroadJump','Cone','Shuttle','Rour
```

From above output it becomes very clear, that players playing different positions also have different physical trai
extreme examples is the positions CB and OG. Looking at those two specific positions, it becomes clear that CB
lighter, can't take as many bench reps, but they can jump higher.

Those results points towards the following hypothesis:

Player positions implies different skills, thus requiring different physical attributes.

Unsupervised learning

Focusing on above hypothesis, we can try to use unsupervised learning to see, if the data itself is capable of dis
players' positions based on their physical stats.

Thus our first step will be to make a new dataframe, df_u, based on the original dataframe, df, as defined earlier
be transforming the 'Pos' variable into dummy variables using the get_dummies function, specifying the drop_firs

```
df_u = pd.get_dummies(df, drop_first=True)
df_u.head()
```

To avoid overspecification we reduce the dataset to 'fundamentals' by excluding stats that could be derived. Tha
be picked first for a specific position becaseu of physical stats, so including both physical stats and position, may
overspecification. Before running unsupervised machinelearning, we reduce the data to fundamental physical st

We will now be scaling our variables, by using the StandardScaler function. Scaling is important when using dist
KMeans is. It can also be a good idea to scale your data, if the different variables have large differences in their

After scaling we use the elbow method on the inertias plot, to determine a rightful amount of clusters.

```
ks = range(1, 15)
inertias = []
scale = StandardScaler()
sample = df_u
scale.fit(sample)
samples = scale.transform(sample)

for i in ks:
    model = KMeans(n_clusters=i)
    model.fit(samples)
    inertias.append(model.inertia_)


plt.figure(figsize=(20,12))
plt.plot(ks, inertias, '-o')
plt.xlabel('number of clusters, i')
plt.ylabel('inertia')
plt.xticks(ks)
plt.show()
```

Looking at above plot we determine k clusters = 2, thus we want to use a total of 2 clusters.

```
model = KMeans(n_clusters=2)
model.fit(samples)
labels = model.predict(samples)
```

Now we want to use the PCA algorithm on our data. Principal components analysis is a hepful technique in orde dimension reduction. Thus it helps us by showing how many PCA features we will include in further analysis.

```
#Fitting the PCA algorithm with our Data
pca = PCA().fit(samples)
#Plotting the Cumulative Summation of the Explained Variance
plt.figure()
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of Components')
plt.ylabel('Variance (%)') #for each component
plt.title('PCA component selection')
plt.show()
```

Looking at above plot of the explained variance, we choose 17 principal components. In comparison there are 2 a few of them being practically identical.

Using 17 principal components, the clusters can now be made and plotted.

```
pca = PCA(n_components=17)
sample = pca.fit_transform(samples)
xs = sample[:,0]
ys = sample[:,1]
plt.scatter(xs, ys, c=labels)
plt.axis('equal')
plt.show()
```

Looking at above plot it seems like the algorithm is capable of splitting the observations into two well-defined an

Following this, it could be very interesting to see, how the different player positions are spread throughout the clu

This can be examined using the crosstab function. In the following we have set normalize='index'. What this doe the different rows, thus summarizing the two clusters for any given position equals 1.

```
lab = pd.DataFrame(labels)
lab.columns = ['Clusters']
df_up = df_u.join(lab)
pd.crosstab(df.Pos,df_up.Clusters,normalize='index')
```

Even though most roles are represented more in one cluster than the other, even at a ratio of 2:1, it doesn't look particularly good at defining the different positions, thus the clusters are not particularly informative.

Supervised learning

From unsupervised learning, it did not appear like the clusters were capable of splitting the players into different position. In the following code, we will try to use different supervised ML algorithms to see if this works any bette position a player has, based on how their physical traits are.

Thus the questions becomes: can we instead predict position by using some supervised machine learning mode

To run a supervised data analysis, the data is first converted to numeric, then scaled, and split into test and train variables seperated from the datasets.

```
supervised = df
supervised.head()
```

Changing the position variable into a numerical variable

We want to use the position variable (Pos) as our target variable. Because the variable is categorical it is first ne numerical variable.

```
posdict = {}
a = supervised.Pos.unique()
b = 0
while b < len(a):
  posdict[a[b]] = b
  b += 1
supervised['PosN'] = supervised['Pos'].map(posdict)
```

Splitting data into test- and train-sets

We now want to split the X- and y-variables into a train- and test-set. We do this by using the train_test_split fun sklearn.model_selection. Before doing this, we want to use a scale (the StandardScaler() function).

Afterwards we split the data into a training and a test set. For this purpose we use a test size of 25%, and we sp value (42), so we can easily replicate our results.

```
encoder = LabelEncoder()
scaler = StandardScaler()
dataset = scaler.fit_transform(supervised.drop(['Pos','PosN'], 1))
target = encoder.fit_transform(supervised['PosN'])
train_data, test_data, train_target, test_target = train_test_split(dataset, target, test_size=0.
```

Now that the data has been split into a training and a test set, and it is scaled, we can use different models in or the models perform.

Linear Regression

```
# K-fold cross-validation
model1 = LinearRegression()
scores1 = cross_val_score(model1, train_data, train_target, cv = 5)
print("An average cross validation score of {}".format(np.mean(scores1)))

# Model training
model1.fit(train_data, train_target)

# Model performance
print('Model performance:', model1.score(test_data, test_target))
```

Unsurprisingly the model has a very poor performance, and it will not be used any further.

Logistic Regression

```
# K-fold cross-validation
model2 = LogisticRegression()
scores2 = cross_val_score(model2, train_data, train_target, cv = 5)
print("An average cross validation score of {}".format(np.mean(scores2)))

# Model training
model2.fit(train_data, train_target)

# Model performance
print('Model performance:', model2.score(test_data, test_target))
```

LogisticRegression is much better at predicting positions, but it still only gets roughly half correct.

Decision Tree model

```
model3 = DecisionTreeClassifier()
scores3 = cross_val_score(model3, train_data, train_target, cv = 5)
print("An average cross validation score of {}".format(np.mean(scores3)))

# Model training
model3.fit(train_data, train_target)

# Model performance
print('Model performance:', model3.score(test_data, test_target))
```

DecisionTree is slightly worse than LogisticRegression, but still much better than LinearRegression.

CatBoost

```
model4 = CatBoostClassifier(logging_level='Silent')
scores4 = cross_val_score(model4, train_data, train_target, cv = 5)
```

```
print("An average cross validation score of {}".format(np.mean(scores4)))

# Model training
model4.fit(train_data, train_target)

# Model performance
print('Model performance:', model4.score(test_data, test_target))
```

CatBoostClassifier is so far the best model with a performance of roughly 0.57.

XGBoost

```
model5 = xgb.XGBClassifier()
scores5 = cross_val_score(model5, train_data, train_target, cv = 5)
print("An average cross validation score of {}".format(np.mean(scores5)))

# Model training
model5.fit(train_data, train_target)

# Model performance
print('Model performance:', model5.score(test_data, test_target))
```

The XGBoostClassifier appears to be slightly worse compared to CatBoostClassifier. It performs better than Log DecisionTree though.

Predicting the models

We call the .predict function on test_data in the models (except LinearRegression, as it was absolutely garbage) predictions in y_pred2, y_pred3, y_pred4 and y_pred5.

```
y_pred1 = model1.predict(test_data)
y_pred2 = model2.predict(test_data)
y_pred3 = model3.predict(test_data)
y_pred4 = model4.predict(test_data)
y_pred5 = model5.predict(test_data)
```

Evaluating the models using classification_report and confusion_matrix

Using the confusion matrixes, correctly predicted positions can be seen on the diagonal. Thus a perfect model w on the diagonal and nowhere else. Looking at the CatBoostClassifier it becomes clear, that the model is the bes models, at predicting which positions a player is drafted to. In certain positions, however, it looks like the model l predicting them.

Overall it looks like the CatBoost model is good at predicting which position a player has - and in situations wher predict right, it generally looks like the model predicts 2-3 different positions. This behaviour might be explained, requiring almost identical physical traits, which also makes it hard for the model to distinguish the positions from into consideration, it does appear that the players' physical traits are deterministic to which positions the players

Logistic Regression

```
print(classification_report(test_target, y_pred2))
print(confusion_matrix(test_target,y_pred2))
```

Decision Tree

```
print(classification_report(test_target,y_pred3))
print(confusion_matrix(test_target,y_pred3))
```

CatBoostClassifier

```
print(classification_report(test_target,y_pred4))
print(confusion_matrix(test_target,y_pred4))
```

XGBoostClassifier

```
print(classification_report(test_target,y_pred5))
print(confusion_matrix(test_target,y_pred5))
```

Discussion

The critical flaw in this analysis is the missing data. We cannot create a coherent analysis of only some positions
are not independent, but defined by their relation to ther positions. Therefore trying to define the positions based
without the other positions are ultimately futile. Consider the following example: If weight was very heavily correl
LS (Long snapper) and DT (Defensive tackle), then a high 'Wt' value might indicate **either** DT or LS, but if the LS
due to low number of observations, the calculated correlation with weight and the DT positions will be unnaturall
nuber of discussion twists the entire analysis, and including positions with low number of observations undermin
In other words, a coherent analysis is not possible from the given dataset.

If, for the sake of the exercise, the data is treated as valid, we can see from the confusion matrices that roughly
are correct, and the rest of the 'mispredicted' are not randomly distributed, but clustered in positions similar to th

Conklusion

The models have not achived a theoretical coherensy, but have nevertheless managed to predict the correct pos
chance.