
Prapemrosesan & Rekayasa Data - IBDA3111

UTS - Students Performance in Exams

Victor Chendra - **202000338**

Description:

1. Masalah yang akan diselesaikan

- A. How effective is the test preparation course?
- B. Which major factors contribute to test outcomes?
- C. What would be the best way to improve student scores on each test?
- D. What patterns and interactions in the data can you find?

2. Deskripsi singkat dataset

Students Performance in Exams dataset adalah sebuah dataset yang menggambarkan tentang performa ujian siswa. Dalam dataset berisikan tentang nilai-nilai ujian siswa di sekolah. Adapun tabel-tabel yang terdapat di dalam dataset. (exams.csv)

Terdapat 8 kolom, 5 string 3 integers dan 1000 baris data. Dataset ini ingin memberikan gambaran, apakah pengaruh latar belakang orang tua, persiapan ujian, dan lain-lain terhadap kinerja ujian siswa.

3. Sumber dataset

Dataset diperoleh dari kaggle.com

<https://www.kaggle.com/datasets/whenamancodes/students-performance-in-exams>

4. Cara memperoleh dataset

Dataset diperoleh dengan cara mengunduh dari website.

Library

```
In [ ]: # Import library needed
```

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from numpy import mean
from numpy import std
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import OneHotEncoder
from sklearn.feature_selection import VarianceThreshold
from sklearn.feature_selection import RFE
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import Perceptron
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.pipeline import Pipeline
from sklearn import linear_model
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

from matplotlib import pyplot
```

Load and check dataset

```
In [ ]: # Load dataset
df = pd.read_csv('exams.csv', delimiter=',')

# Show dataset
# pd.set_option('display.max_rows', None) # display all rows dataset
display(df.head(5))

# Dataset shape
print(f"Rows : {df.shape[0]}\nColumns: {df.shape[1]}")
```

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	male	group A	high school	standard	completed	67	67	63
1	female	group D	some high school	free/reduced	none	40	59	55
2	male	group E	some college	free/reduced	none	59	60	50
3	male	group B	high school	standard	none	77	78	68
4	male	group E	associate's degree	standard	completed	78	73	68

Rows : 1000

Columns: 8

```
In [ ]: # Check the datatype for all the column values
df.dtypes
```

```
Out[ ]: gender          object
race/ethnicity    object
parental level of education  object
lunch            object
test preparation course   object
math score        int64
reading score      int64
writing score      int64
dtype: object
```

```
In [ ]: # Check unique values columns
df.nunique()
```

```
Out[ ]: gender      2
race/ethnicity  5
parental level of education  6
lunch         2
test preparation course   2
math score     77
reading score   73
writing score   76
dtype: int64
```

```
In [ ]: # Let's check if the data has some missing values
print(f"Check if the data has some missing values")
df.isnull().sum()
```

Check if the data has some missing values

```
Out[ ]: gender          0  
race/ethnicity      0  
parental level of education 0  
lunch              0  
test preparation course 0  
math score         0  
reading score       0  
writing score        0  
dtype: int64
```

↑ There are no missing values in the dataset* ↑

```
In [ ]: # Let's check if the values of the first 5 columns numerical or categorical
```

```
for i in range(5):  
    print(f"{df[df.columns[i]].value_counts()}\n")
```

```
male      517  
female    483  
Name: gender, dtype: int64
```

```
group C    323  
group D    262  
group B    205  
group E    131  
group A     79  
Name: race/ethnicity, dtype: int64
```

```
some college    222  
associate's degree 203  
high school     202  
some high school 191  
bachelor's degree 112  
master's degree    70  
Name: parental level of education, dtype: int64
```

```
standard      652  
free/reduced   348  
Name: lunch, dtype: int64
```

```
none      665  
completed  335  
Name: test preparation course, dtype: int64
```

↑↑ We can see from the output above that the first 5 columns are categorical variables. Then, the rest columns are numerical variables.

```
In [ ]: df.iloc[:, 5:].describe()
```

	math score	reading score	writing score
count	1000.000000	1000.000000	1000.000000
mean	66.396000	69.002000	67.738000
std	15.402871	14.737272	15.600985
min	13.000000	27.000000	23.000000
25%	56.000000	60.000000	58.000000
50%	66.500000	70.000000	68.000000
75%	77.000000	79.000000	79.000000
max	100.000000	100.000000	100.000000

↑↑ Let's see the statistic presentation using .describe()

```
In [ ]: figure, ax = plt.subplots(2,2, constrained_layout=True, figsize=(11, 6))

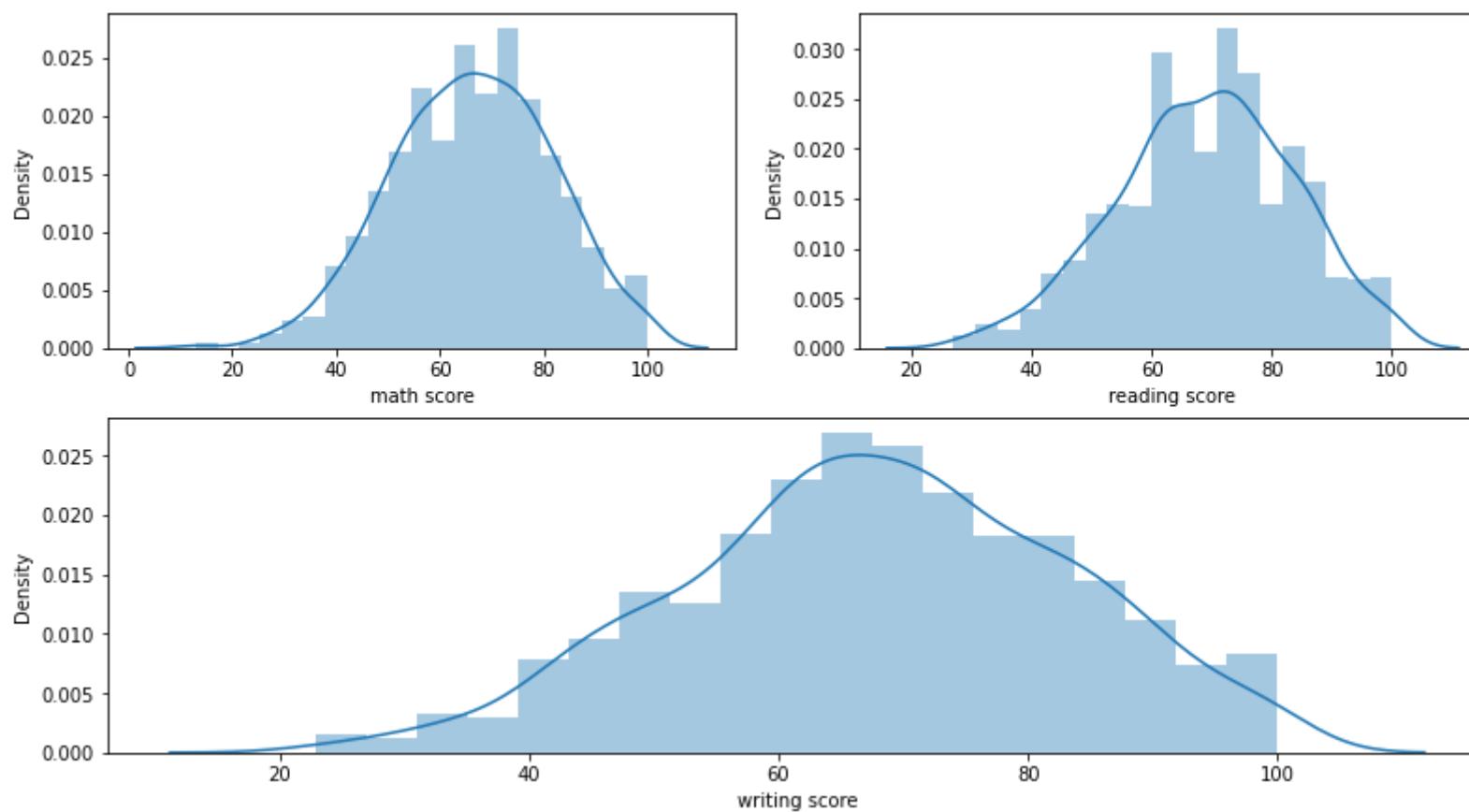
plt.subplot(2, 2, 1)
sns.distplot(df['math score'])

plt.subplot(2, 2, 2)
sns.distplot(df['reading score'])

plt.subplot(2, 2, (3, 4))
sns.distplot(df['writing score'])
```

```
c:\Python 3.10.2\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
    warnings.warn(msg, FutureWarning)
c:\Python 3.10.2\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
    warnings.warn(msg, FutureWarning)
c:\Python 3.10.2\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
    warnings.warn(msg, FutureWarning)
```

```
Out[ ]: <AxesSubplot:xlabel='writing score', ylabel='Density'>
```



↑↑ Let's see the distribution

↑↑ Let's see the distribution of the scores.

1. Data cleaning

1. Delete duplicated rows

```
In [ ]: # Delete duplicated rows
dups = df.duplicated()
print(f"Duplicated rows:")
print(dups.any())
print(df[dups])

print(f"\nShape before delete: {df.shape}")
df.drop_duplicates(inplace=True)
print(f"Shape after delete : {df.shape}\n")
```

```
Duplicated rows:  
True  
   gender race/ethnicity parental level of education      lunch  \  
825    male           group E          associate's degree  standard  
  
   test preparation course  math score  reading score  writing score  
825            completed        100          100          100  
  
Shape before delete: (1000, 8)  
Shape after delete : (999, 8)
```

2. Identifikasi Pencilan lalu diimputasi ke nilai lower bound

```
In [ ]: # This func return upper & lower bound from dataset column
def identify_outlier(df_column):
    Q1, Q3 = np.percentile(df_column, [25, 75])
    IQR = Q3 - Q1
    lower_range = Q1 - (1.5 * IQR)
    upper_range = Q3 + (1.5 * IQR)

    return lower_range, upper_range

# Plot to boxplot so we can see the outlier data
def display_boxplot():
    figure, ax = plt.subplots(2,2, constrained_layout=True, figsize=(15, 5))

    s = 1
    for i in range(len(df.columns)):
        if i == 7:
            plt.subplot(2, 2, (s, s+1))
            plt.title(df.columns[i])
            sns.boxplot(x=df.iloc[:, i])
            s += 1
        continue

        if i >= 5:
            plt.subplot(2, 2, s)
            plt.title(df.columns[i])
            sns.boxplot(x=df.iloc[:, i])
            s += 1

# Take the bounderies value
math_score = identify_outlier(df.iloc[:, 5])
reading_score = identify_outlier(df.iloc[:, 6])
writing_score = identify_outlier(df.iloc[:, 7])

# Capture outlier (lower bound)
outlier_ms = [x for x in df.iloc[:, 5] if x < math_score[0]]
outlier_rs = [x for x in df.iloc[:, 6] if x < reading_score[0]]
outlier_ws = [x for x in df.iloc[:, 7] if x < writing_score[0]]

print(f"Outlier")
print(f"    Math score    = {outlier_ms}")
print(f"    Writing score = {outlier_rs}")
print(f"    Reading score = {outlier_ws}")

display_boxplot()
```

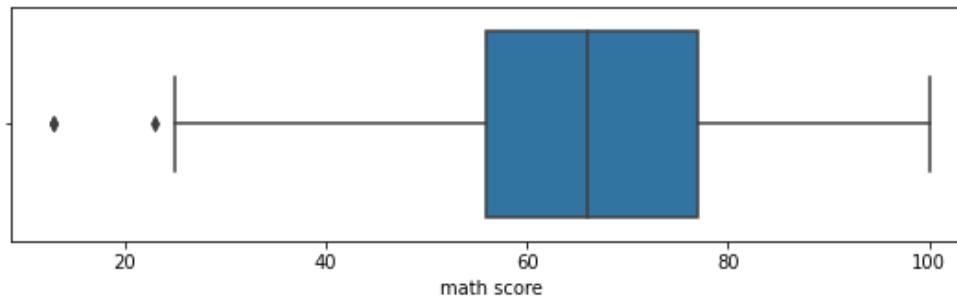
Outlier

Math score = [23, 13, 13]

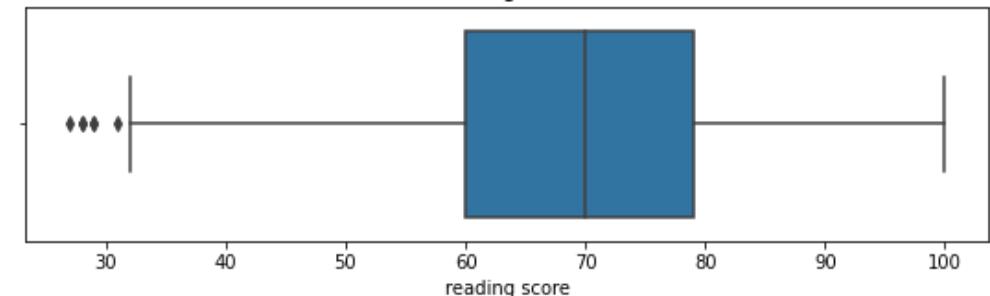
Writing score = [28, 29, 27, 28, 31, 29]

Reading score = [24, 23, 26, 23]

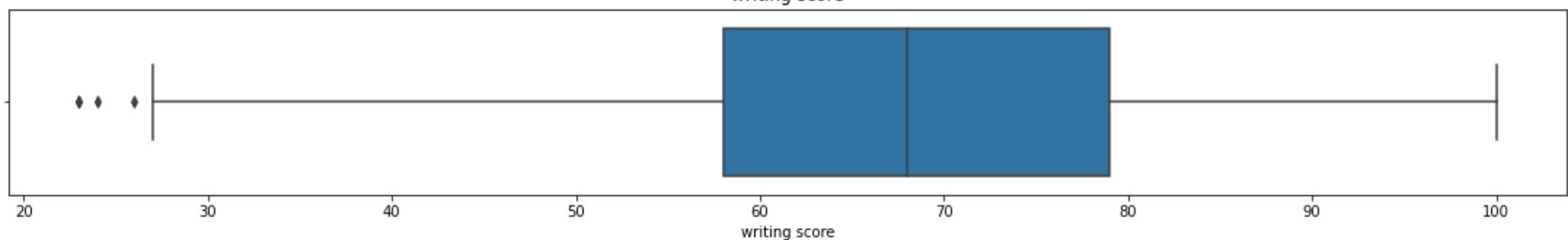
math score



reading score



writing score



↑↑ From boxplot above.. we can see that there is an outlier in our dataset

```
In [ ]: # Function to impute to our dataset
def impute_outlier_to_lower_bound_value(df: pd.DataFrame, bounderies: tuple, n_col: int):
    for i in range(df.shape[0]):
        if df.iloc[i, n_col] <= bounderies[0]:
            df.iloc[i, n_col] = bounderies[0]
    return

impute_outlier_to_lower_bound_value(df, math_score, 5)
impute_outlier_to_lower_bound_value(df, reading_score, 6)
impute_outlier_to_lower_bound_value(df, writing_score, 7)

# Take the bounderies value
math_score = identify_outlier(df.iloc[:, 5])
reading_score = identify_outlier(df.iloc[:, 6])
writing_score = identify_outlier(df.iloc[:, 7])

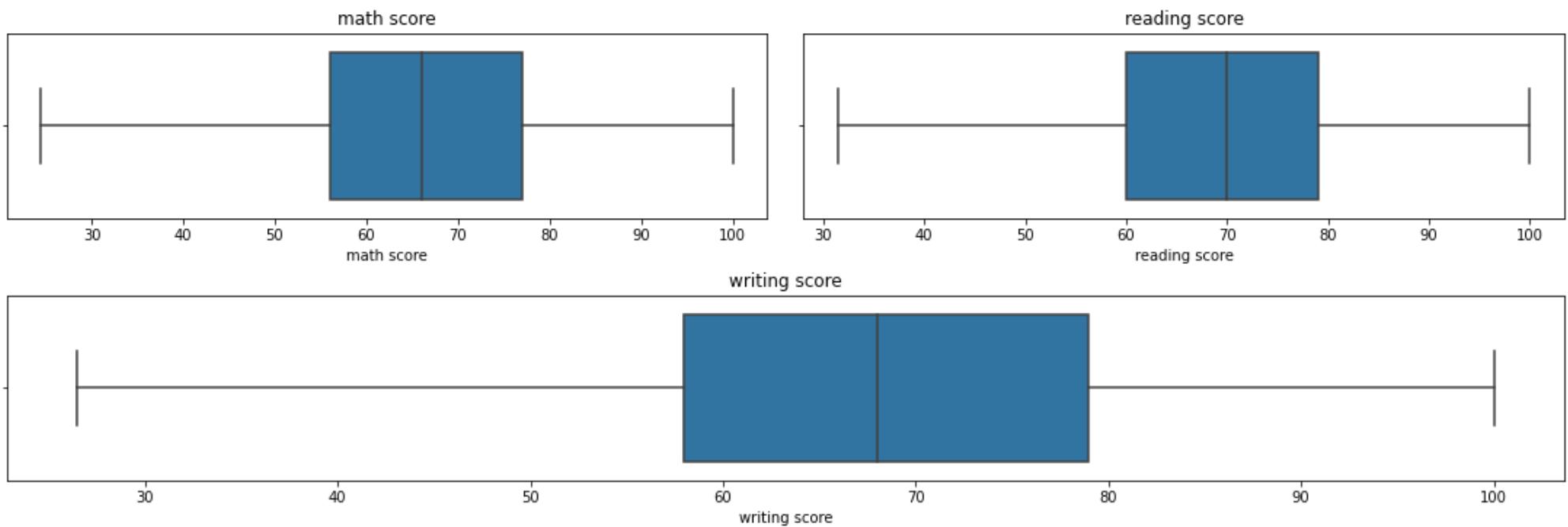
# Capture outlier (lower bound)
outlier_ms = [x for x in df.iloc[:, 5] if x < math_score[0]]
outlier_rs = [x for x in df.iloc[:, 6] if x < reading_score[0]]
outlier_ws = [x for x in df.iloc[:, 7] if x < writing_score[0]]

print(f"Outlier")
print(f"    Math score    = {outlier_ms}")
print(f"    Writing score = {outlier_rs}")
print(f"    Reading score = {outlier_ws}")

display_boxplot()
```

Outlier

```
Math score    = []
Writing score = []
Reading score = []
```



↑↑ Now, we have done imputing the outlier data to it's lower bound value (so no more outlier)

We impute the outlier to it's lower bound because there are just a few outlier and we need it, so we don't delete the outlier data

2. Features selection

1. RFE

↓↓ Adding columns "total" and "average" to the dataset

```
In [ ]: # Display dataset before one hot encoding (after data cleaning)
display(df.head(2))
print(df.shape)

# Adding columns "total" and "average" to the dataset
df['total'] = df['math score'] + df['reading score'] + df['writing score']
df['average'] = df['total'] / 3
df.drop(["total"], axis=1, inplace=True)

print()
display(df.head(2))
print(df.shape)
```

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	male	group A	high school	standard	completed	67.0	67.0	63.0
1	female	group D	some high school	free/reduced	none	40.0	59.0	55.0

(999, 8)

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score	average
0	male	group A	high school	standard	completed	67.0	67.0	63.0	65.666667
1	female	group D	some high school	free/reduced	none	40.0	59.0	55.0	51.333333

(999, 9)

↓↓ One hot encoding

```
In [ ]: # Create a function for one hot encoding (only variable input)
def one_hot_encoding(df):
    encoded = pd.get_dummies(df.iloc[:, :-4])
    ms = df.iloc[:, -4]
    rs = df.iloc[:, -3]
    ws = df.iloc[:, -2]
    avg = df.iloc[:, -1]

    encoded["math score"] = ms
    encoded["reading score"] = rs
    encoded["writing score"] = ws
    encoded["average"] = avg

    return encoded

# ----- Save it for later -----
# Original output
saved_mathscore = df.iloc[:, -4]
save_readingscore = df.iloc[:, -3]
save_writingscore = df.iloc[:, -2]
save_average = df.iloc[:, -1]

# before one hot encoding
display(df.head(3))
df.shape

# Make sure to run it just ONCE otherwise, you need to run it all again from begining
df_encoded = one_hot_encoding(df)

# after one hot encoding
display(df_encoded.head(3))
df_encoded.shape

# Adding new column indicates pass or not the exams
# if it's under 60, it's fail else they pass the test
df_encoded['math_PassStatus'] = np.where(df_encoded['math score']<60, 0, 1)
df_encoded['read_PassStatus'] = np.where(df_encoded['reading score']<60, 0, 1)
df_encoded['write_PassStatus'] = np.where(df_encoded['writing score']<60, 0, 1)
df_encoded['overall_PassStatus'] = np.where(df_encoded['average']<60, 0, 1)

# After added, we drop the column
df_encoded.drop(['math score'], axis=1, inplace=True)
df_encoded.drop(['reading score'], axis=1, inplace=True)
df_encoded.drop(['writing score'], axis=1, inplace=True)
df_encoded.drop(['average'], axis=1, inplace=True)
```

```
display(df_encoded.head(3))  
df_encoded.shape
```

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score	average
0	male	group A	high school	standard	completed	67.0	67.0	63.0	65.666667
1	female	group D	some high school	free/reduced	none	40.0	59.0	55.0	51.333333
2	male	group E	some college	free/reduced	none	59.0	60.0	50.0	56.333333

	gender_female	gender_male	race/ethnicity_group_A	race/ethnicity_group_B	race/ethnicity_group_C	race/ethnicity_group_D	race/ethnicity_group_E	parental level of education_associate's ed degree
0	0	1	1	0	0	0	0	0
1	1	0	0	0	0	1	0	0
2	0	1	0	0	0	0	1	0

3 rows × 21 columns

	gender_female	gender_male	race/ethnicity_group_A	race/ethnicity_group_B	race/ethnicity_group_C	race/ethnicity_group_D	race/ethnicity_group_E	parental level of education_associate's ed degree
0	0	1	1	0	0	0	0	0
1	1	0	0	0	0	1	0	0
2	0	1	0	0	0	0	1	0

3 rows × 21 columns

Out[]: (999, 21)

↑↑ Done One Hot Encoding ✓

```

In [ ]:
"""
This templates function is given from my teacher's jupyter notebook "RFE.ipynb" / from books "Data Preparation for
Machine Learning - Jason Brownlee"
"""

# get a list of models to evaluate
def get_models(X, y, n: int):
    models = dict()
    # lr
    rfe = RFE(estimator=LogisticRegression(), n_features_to_select=n)
    model = DecisionTreeClassifier()
    models['log reg'] = Pipeline(steps=[('s',rfe),('m',model)])
    # cart
    rfe = RFE(estimator=DecisionTreeClassifier(), n_features_to_select=n)
    model = DecisionTreeClassifier()
    models['cart'] = Pipeline(steps=[('s',rfe),('m',model)])
    # perceptron
    rfe = RFE(estimator=Perceptron(), n_features_to_select=n)
    model = DecisionTreeClassifier()
    models['perceptron'] = Pipeline(steps=[('s',rfe),('m',model)])
    # rf
    rfe = RFE(estimator=RandomForestClassifier(), n_features_to_select=n)
    model = DecisionTreeClassifier()
    models['rand forest'] = Pipeline(steps=[('s',rfe),('m',model)])
    # gbm
    rfe = RFE(estimator=GradientBoostingClassifier(), n_features_to_select=n)
    model = DecisionTreeClassifier()
    models['gbm'] = Pipeline(steps=[('s',rfe),('m',model)])
    return models

# evaluate a given model using cross-validation
def evaluate_model(model, X, y):
    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
    scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
    return scores

```

In []:

```
# Split dataset
X = df_encoded.iloc[:, :-1]
y = df_encoded.iloc[:, -1]

# evaluate the models and store results

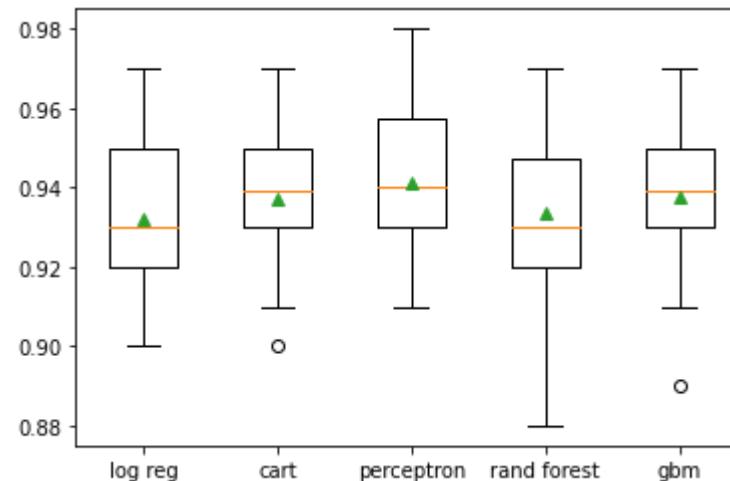
# We select n features to 15 because we think that from 20 features...
# there are redundant features after one hot encoding, like we can drop 1 column but it's still represent the dataset itself
# (Column: gender, race/ethnicity, parental level of education, lunch, test preparation course) = 5 columns
n_features_to_select = 15

models = get_models(X, y, n_features_to_select)

results, names = list(), list()
for name, model in models.items():
    scores = evaluate_model(model, X, y)
    results.append(scores)
    names.append(name)
    print('> %s %.3f (%.3f)' % (name, mean(scores), std(scores)))

# plot model performance for comparison
pyplot.boxplot(results, labels=names, showmeans=True)
pyplot.show()
```

> log reg 0.932 (0.019)
> cart 0.937 (0.017)
> perceptron 0.941 (0.018)
> rand forest 0.934 (0.019)
> gbm 0.938 (0.019)



↑↑ As we can see.. "perceptron" has the highest accuracy, so we take "perception" model as our estimator

```
In [ ]: def get_model_perceptron(X, y, n: int):
    models = dict()
    rfe = RFE(estimator=Perceptron(), n_features_to_select=n)
    model = DecisionTreeClassifier()
    models['perceptron'] = Pipeline(steps=[('s',rfe),('m',model)])
    rfe.fit(X, y)
    return models, rfe.support_, rfe.ranking_

# Ranking the features
percp = get_model_perceptron(X, y, n_features_to_select)
print(f"Selected feat. :{percp[1]}")
print(f"Features rank : {percp[2]}")
```

Selected feat. :[True True False False True True True True False False True False
 True True True True True True True True]
 Features rank : [1 1 5 4 1 1 1 1 2 6 1 3 1 1 1 1 1 1 1 1]

```
In [ ]: selected_fetures = [i for i, s in enumerate(percp[1]) if s]
print(f"Selected features: {selected_fetures}")

df_RFE = df_encoded.iloc[:, selected_fetures]
df_RFE["average"] = y

# Let's see the final dataset after features selection (RFE)
display(df_RFE.head(2))
print(df_RFE.shape)
```

Selected features: [0, 1, 4, 5, 6, 7, 10, 12, 13, 14, 15, 16, 17, 18, 19]

C:\Users\victo\AppData\Local\Temp\ipykernel_4576\2196618856.py:5: SettingWithCopyWarning:
 A value is trying to be set on a copy of a slice from a DataFrame.
 Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
 df_RFE["average"] = y

gender_female	gender_male	race/ethnicity_group_C	race/ethnicity_group_D	race/ethnicity_group_E	parental level of education_associate's degree	parental level of education_master's degree	parental level of education_some high school	lunch_free
0	0	1	0	0	0	0	0	0
1	1	0	0	1	0	0	0	1

(999, 16)

↑↑ Final dataset after features selection (RFE)

2. Features selection chi-square

```
In [ ]: # Display dataset after one hot encoding  
df_encoded.head(2)
```

Out[]:

	gender_female	gender_male	race/ethnicity_group_A	race/ethnicity_group_B	race/ethnicity_group_C	race/ethnicity_group_D	race/ethnicity_group_E	parental level of education_associate's degree
0	0	1	1	0	0	0	0	0
1	1	0	0	0	0	1	0	0

2 rows × 21 columns

```
In [ ]: # Split dataset  
X = df_encoded.iloc[:, :-1]  
y = df_encoded.iloc[:, -1]
```

```
# Features selection based on chi-square score  
fs = SelectKBest(score_func=chi2, k='all')  
fs.fit(X, y)  
X_train_fs = fs.transform(X)
```

```
# Get chi-square score for all columns  
features_score = fs.scores_  
features_score = features_score.tolist()
```

```
print(f"Features score:\n{features_score}")
```

Features score:

```
[1.646292313558209, 1.5410061772260017, 0.10517193147122977, 0.6211477202956669, 8.859720493179172, 8.432822006776183, 3.2824830828256424, 6.214337849241212, 5.446751250342558, 0.09983407905054674, 3.5271481283661714, 2.2988294248913577, 17.45630466477532, 44.049242183903544, 23.547060337939232, 23.342219507272105, 11.72376137658479, 180.74431188914133, 172.26055207393418, 192.1720486724545]
```

↑↑ Done features selection chi-square ✓

```
In [ ]: def selection_sort_and_keep_track_the_original_index(features_score):
    temp = [i for i in range(len(features_score))]
    for i in range(0, len(features_score)-1):
        for j in range(i+1, len(features_score)):
            if features_score[i] > features_score[j]:
                features_score[i], features_score[j] = features_score[j], features_score[i]
                temp[i], temp[j] = temp[j], temp[i]

    return list(zip(temp, features_score))

final_features = selection_sort_and_keep_track_the_original_index(features_score)
print(final_features)

[(9, 0.09983407905054674), (2, 0.10517193147122977), (3, 0.6211477202956669), (1, 1.5410061772260017), (0, 1.646292313558209), (11, 2.2988294248913577), (6, 3.2824830828256424), (10, 3.5271481283661714), (8, 5.446751250342558), (7, 6.214337849241212), (5, 8.432822006776183), (4, 8.859720493179172), (16, 11.72376137658479), (12, 17.45630466477532), (15, 23.342219507272105), (14, 23.547060337939232), (13, 44.049242183903544), (18, 172.26055207393418), (17, 180.74431188914133), (19, 192.1720486724545)]
```

↑↑ We sort the chi-square features score but keep tracking the original index (Don't run it twice)

```
In [ ]: # Next we just need to display the final dataset

# We select best of 15 features out of 20 based on our RFE above
final_features = [g[0] for g in (final_features[5:])]

df_cs = df_encoded.iloc[:, final_features]
df_cs["average"] = y

display(df_cs)
```

C:\Users\victo\AppData\Local\Temp\ipykernel_4576\2541747963.py:5: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.

Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
df_cs["average"] = y
```