In [1]:

```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

%matplotlib inline

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("whitegrid")
sns.set_context("notebook")
#sns.set_context("poster")
```

Feature Selection

The element that has the biggest impact in the quality of your model is data features. You can only include in your model the attributes that you have and if they are not relevant, partially relevant or don't caputre the causality relationships behind the model, or introduce other relationships that correspond to other causes different from the ones that you want to investigate, then you'll have a poor model.

Selecting the relevant features that add to your model is therefore of the utmost importance.

In this notebook we will deal with four approaches:

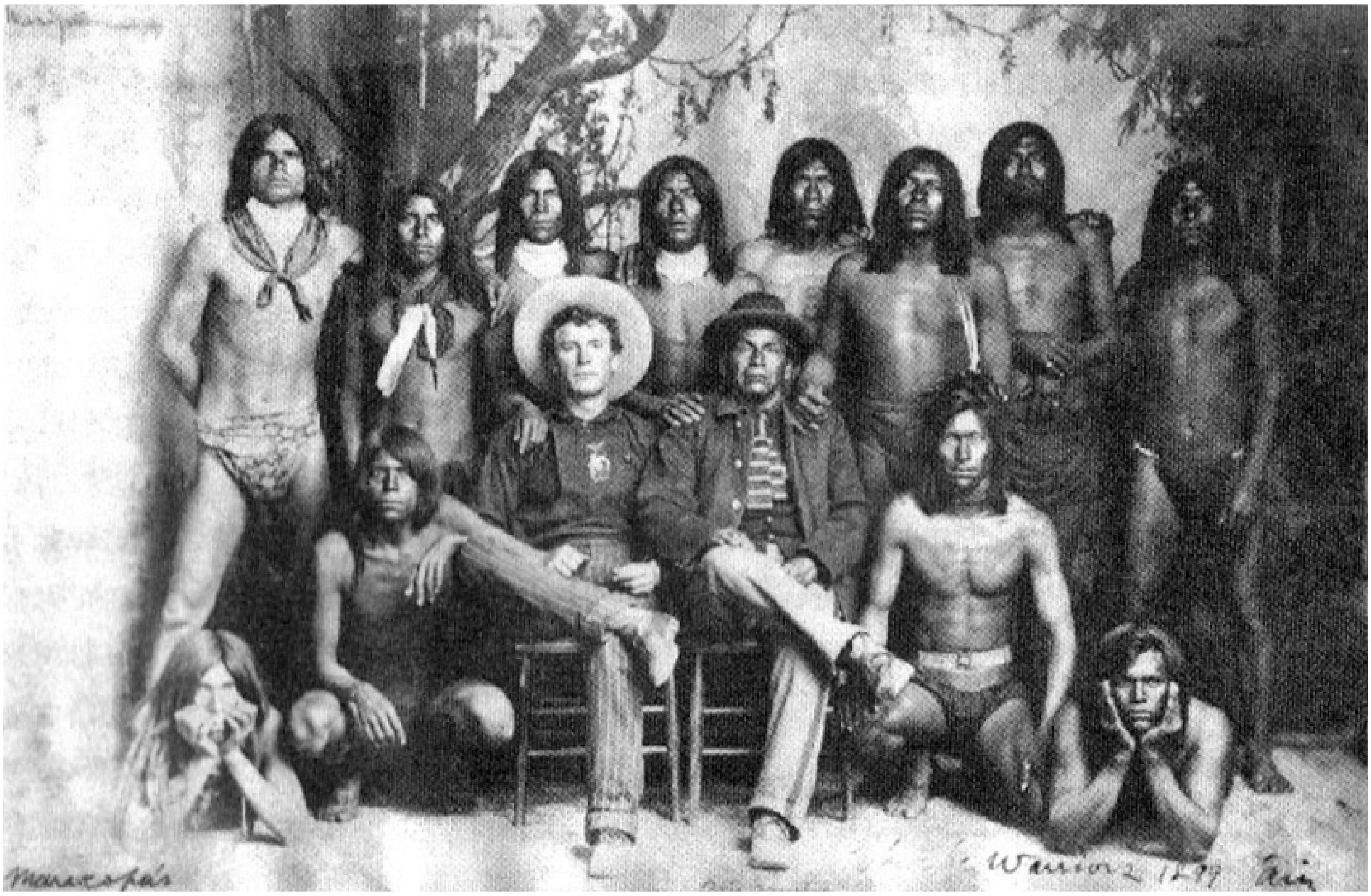
```
    1) Univaritate Selection.
    2) Recursive feature elimination.
    3) PCA - Principal Component Analysis.
    4) Estimating feature importance.
```

Feature selection is a process where you select those features in your data that contribute most to the variable of interest. Irrelevant features decrease the accuracy of many models because you try to adjust on noise, this is particularly important in the case of linear models, such as linear and logistic regressions, where all features are always taken into accout. Three are the main benefits of feature selection:

```
    1) Reduces overfitting. Less redundant data implies less decisions made on noi
se.
    2) Improves accuracy. Less misleading data results in a more accurate model.
    3) Reduces training time. Less data implies faster training.
```

Scikitlearn has a nice and short article on feature selection where you can learn more https://scikit-learn.org/stable/modules/feature_selection.html (https://scikit-learn.org/stable/modules/feature_selection.html)

Again we will use the Pima Indians onset of diabetes dataset.



In this exercise we will use one of the traditional Machine Learning dataset, the Pima Indians diabetes dataset.

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

Content The datasets consists of several medical predictor variables and one target variable, **Outcome**. Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and so on.

> - Pregnancies
> - Glucose
> - BloodPressure
> - SkinThickness
> - Insulin
> - BMI
> - DiabetesPedigreeFunction (scores de likelihood of diabetes based on family history)
> - Age
> - Outcome

In [2]:

```
# Load the Pima indians dataset and separate input and output components

from numpy import set_printoptions
set_printoptions(precision=3)

filename="pima-indians-diabetes.data.csv"
names=["pregnancies", "glucose", "pressure", "skin", "insulin", "bmi", "pedi", "age", "outc
p_indians=pd.read_csv(filename, names=names)
p_indians.head()

# First we separate into input and output components
array=p_indians.values
X=array[:,0:8]
Y=array[:,8]
X
pd.DataFrame(X).head()
```

Out[2]:

	pregnancies	glucose	pressure	skin	insulin	bmi	pedi	age	outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Out[2]:

```
array([[  6.   , 148.   ,  72.   , ...,  33.6  ,   0.627,  50.   ],
       [  1.   ,  85.   ,  66.   , ...,  26.6  ,   0.351,  31.   ],
       [  8.   , 183.   ,  64.   , ...,  23.3  ,   0.672,  32.   ],
       ...,
       [  5.   , 121.   ,  72.   , ...,  26.2  ,   0.245,  30.   ],
       [  1.   , 126.   ,  60.   , ...,  30.1  ,   0.349,  47.   ],
       [  1.   ,  93.   ,  70.   , ...,  30.4  ,   0.315,  23.   ]])
```

Out[2]:

	0	1	2	3	4	5	6	7
0	6.0	148.0	72.0	35.0	0.0	33.6	0.627	50.0
1	1.0	85.0	66.0	29.0	0.0	26.6	0.351	31.0
2	8.0	183.0	64.0	0.0	0.0	23.3	0.672	32.0
3	1.0	89.0	66.0	23.0	94.0	28.1	0.167	21.0
4	0.0	137.0	40.0	35.0	168.0	43.1	2.288	33.0

Univariate Selection

One approach is to use statistical tests for example the Pearson Chi-Squared χ^2 is commonly used to select the most significant features.

We will use the **SelectKBest** class in scikit-learn.

In [3]:

```python
# Univariate selection using Chi-squared
set_printoptions(precision=3)
p_indians.head()

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

# feature selection (we select the 4 best)
test = SelectKBest(score_func=chi2, k=4)
fit = test.fit(X,Y)
print("Scores")

fit.scores_

print("The 4 attributes with the highest scores are: glucose, insulin, bmi and age ")
print()

features=fit.transform(X)
features[0:5,:]
```

Out[3]:

	pregnancies	glucose	pressure	skin	insulin	bmi	pedi	age	outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
Scores
```

Out[3]:

```
array([ 111.52 , 1411.887,   17.605,   53.108, 2175.565,  127.669,
          5.393,  181.304])
```

```
The 4 attributes with the highest scores are: glucose, insulin, bmi and age
```

Out[3]:

```
array([[148. ,   0. ,  33.6,  50. ],
       [ 85. ,   0. ,  26.6,  31. ],
       [183. ,   0. ,  23.3,  32. ],
       [ 89. ,  94. ,  28.1,  21. ],
       [137. , 168. ,  43.1,  33. ]])
```

In [ ]:

Recursive Feature Elimination

This is a very intuitive approach. It consist on recursively removing attributes and building a model with those atrributes remaining. It uses the model accuracy to identify which atrributes or combination of attributes contribute the most.

We will use it with a logistic regression, but the choice of algorithm doesn't matter too much as long as your are consistent.

Recursive Feature Elimination uses the **RFE** class.

In [4]:

```
# Recursive Feature Elimiantion

from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression

p_indians.head()

#Logistic regression
model = LogisticRegression(solver='liblinear')

rfe = RFE(model, 3) #  we want to find the 3 top features
fit = rfe.fit(X, Y)

print(f'Number of features {fit.n_features_:d}')
print(f'Selected features {fit.support_}')
print(f'Ranking of features {fit.ranking_}')
print()
print("Top features seem to be pregnancies, bmi, and pedi(Diabetes Pedigree Function)")
```

Out[4]:

	pregnancies	glucose	pressure	skin	insulin	bmi	pedi	age	outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
Number of features 3
Selected features [ True False False False False  True  True False]
Ranking of features [1 2 3 5 6 1 1 4]

Top features seem to be pregnancies, bmi, and pedi(Diabetes Pedigree Functio
n)
```

In [ ]:

Mission 1

For this and the next mission we will use data from Kaggle In concrete from the World University Rankings Competition https://www.kaggle.com/mylesoneill/world-university-rankings (https://www.kaggle.com/mylesoneill/world-university-rankings)

a) Using the Shanghai rankings find the top 3 most important features to explain them with both univariate and recursive (in recursive because we are using log regression create an output variable of being in the top 50 or not).

b) Same for the Times ranking.

c) Does it change if we choose the top 10 or top 100?

In [5]:

```
# Writefile line is creating a new file called data_cleaning to later import the functions
```

In [6]:

```
%%writefile data_cleaning.py

import pandas as pd
import numpy as np

# Clean shangai dataset function

def shangai_clean(x):
    # Read excel file and sort by total score
    shangai = pd.read_excel("shanghaiData.xlsx").sort_values(by = "total_score", ascending

    # Filter by the latest year
    shangai = shangai[shangai["year"] == shangai["year"].max()]

    # Simplify dataframe with only explanatory variables and drop null values
    shangai.drop(["world_rank", "university_name", "national_rank", "year", "total_score"],

    # Drop null values
    shangai.dropna(inplace = True)

    # Code the top 50 universities
    array_ref = (np.arange(len(shangai)) < x)
    shangai["top_50"] = array_ref
    code = {True:1.0, False:0.0}
    shangai["top_50"] = shangai["top_50"].map(code)

    # Return the array
    return shangai
```

```
Overwriting data_cleaning.py
```

In [7]:

```
%%writefile -a data_cleaning.py

# Clean the times dataset

def times_clean(a):

    # Read the csv file and sort by total score ind descending order
    times = pd.read_csv("timesData.csv").sort_values(by = "total_score", ascending = False)

    # Drop null values from total score
    times["total_score"] = pd.to_numeric(times["total_score"], errors = "coerce")
    times.dropna(inplace = True)

    # Filter by the latest year
    times = times[times["year"] == 2016]

    # Simplify the table by dropping non explanatory variables
    times.drop(["world_rank", "university_name", "country", "year", "total_score"], axis =

    # Convert all other columns to float type
    times["international"] = pd.to_numeric(times["international"], errors = "coerce")
    times["income"] = pd.to_numeric(times["income"], errors = "coerce")
    times["female_male_ratio"] = pd.to_numeric(times["female_male_ratio"].apply(lambda d: d
    times["international_students"] = pd.to_numeric(times["international_students"].apply(l

    student_list = []

    for x in range(len(times)):
        student_value = times["num_students"].iloc[x].replace(",",".")
        student_list.append(student_value)

    times["num_students"] = pd.to_numeric(student_list, errors = "coerce")

    # Drop null values once again after converting all other columns to float
    times.dropna(inplace = True)

    # Add a new column with the actual ranking
    array_ref = (np.arange(len(times)) < a)
    times["top_50"] = array_ref
    code = {True:1.0, False:0.0}
    times["top_50"] = times["top_50"].map(code)

    return times
```

```
Appending to data_cleaning.py
```

In [8]:

```
import data_cleaning as dc
```

In [9]:

```
# Automating column printing function for recursive feature elimination

def col_list(ranking, column_names):
    var_s = ranking
    top_cols = []
    counter = 0
    for x in var_s:
        if var_s[counter] == 1:
            top_cols.append(column_names[counter])
        counter +=1
    return top_cols
```

In [10]:

```
# Automatating column printing function for univariate analysis

def col_dic(scores):
    import math
    indices = np.arange(len(scores))
    first_in = second_in = third_in = 0
    first = second = third = -math.inf
    for x in indices:
        if scores[x] > first:
            second = first
            second_in = first_in
            first = scores[x]
            first_in = x
        elif scores[x] > second:
            third = second
            third_in = second_in
            second = scores[x]
            second_in = x
        elif scores[x] > third:
            third = scores[x]
            third_in = x
    return {"first":first_in, "second":second_in, "third": third_in}
```

In [11]:

```
# Shangai top-50 analysis

shangai_treat = dc.shangai_clean(50)
shan_cols = shangai_treat.columns

array = shangai_treat.values

shan_x = array[:,0:6]
shan_y = array[:,6]

# Run Logistic Regression

model = LogisticRegression(solver='liblinear')
rfe = RFE(model, 3)#  we want to find the 3 top features
rec = rfe.fit(shan_x, shan_y)

# Automate column outputs for RFE

top_cols = col_list(rec.ranking_, shan_cols)

print('\033[1m' + 'Recursive Feature Elimination' '\033[0m')
print()
print(f'Number of features {rec.n_features_:d}')
print(f'Selected features {rec.support_}')
print(f'Ranking of features {rec.ranking_}')
print()
print(f"Top features are: {top_cols[0]}, {top_cols[1]}, {top_cols[2]}")

# Run univariate analysis

test = SelectKBest(score_func=chi2, k=3)
uni = test.fit(shan_x,shan_y)

print()
print('\033[1m' + 'Univariate Analysis' '\033[0m')
uni.scores_

# Automate column outputs for univariate analysis

cols = col_dic(uni.scores_)
print("The top features are: " + shan_cols[cols["first"]] + ", " + shan_cols[cols["second"]

features=uni.transform(shan_x)
features[0:5,:]
```

```
Recursive Feature Elimination

Number of features 3
Selected features [False  True False  True False  True]
Ranking of features [4 1 2 1 3 1]

Top features are: award, ns, pcp

Univariate Analysis
```

Out[11]:

```
array([3533.277, 8078.695, 2861.682, 2551.689,  599.288,  528.423])

The top features are: award, alumni, hici
```

Out[11]:

```
array([[100. , 100. , 100. ],
       [ 40.7,  89.6,  80.1],
       [ 68.2,  80.7,  60.6],
       [ 65.1,  79.4,  66.1],
       [ 77.1,  96.6,  50.8]])
```

In [12]:

```
# Times top-50 analysis

times_treat = dc.times_clean(50)
times_cols = times_treat.columns
array = times_treat.values

times_x = array[:,0:9]
times_y = array[:,9]

# Run Logistic Regression

model = LogisticRegression(solver='liblinear')
rfe = RFE(model, 3)
rec = rfe.fit(times_x, times_y)

# Automate column outputs for RFE

top_cols = col_list(rec.ranking_, times_cols)

print('\033[1m' + 'Recursive Feature Elimination' '\033[0m')
print()
print(f'Number of features {rec.n_features_:d}')
print(f'Selected features {rec.support_}')
print(f'Ranking of features {rec.ranking_}')
print()
print(f"Top features are {top_cols[0]}, {top_cols[1]}, and {top_cols[2]}")

# Run univariate analysis

test = SelectKBest(score_func=chi2, k=3)
uni = test.fit(times_x, times_y)

print()
print('\033[1m' + 'Univariate Analysis' '\033[0m')
uni.scores_

cols = col_dic(uni.scores_)
print("The top features are: " + times_cols[cols["first"]] + ", " + times_cols[cols["second

features=uni.transform(times_x)
features[0:9,:]
```

```
Recursive Feature Elimination

Number of features 3
Selected features [False False  True False False False  True False  True]
Ranking of features [6 5 1 7 2 3 1 4 1]

Top features are research, student_staff_ratio, and female_male_ratio

Univariate Analysis
```

Out[12]:

```
array([591.685,  15.786, 869.926,  44.943,  56.724,  55.647,  41.405,
        60.534,   3.975])
```

```
The top features are: research, teaching, international_students
```

Out[12]:

```
array([[95.6, 97.6, 27. ],
       [86.5, 98.9, 34. ],
       [92.5, 96.2, 22. ],
       [88.2, 96.7, 34. ],
       [89.4, 88.6, 33. ],
       [85.1, 91.9, 27. ],
       [83.3, 88.5, 51. ],
       [77. , 95. , 37. ],
       [85.7, 88.9, 21. ]])
```

In [13]:

```
# c) Does it change if we choose the top 10 or top 100? Since shangai only has 100 rows aft
# the total score, we are going to test this with the times dataset.
```

In [14]:

```
# Shangai top-10 analysis

shangai_treat = dc.shangai_clean(10)
shangai_cols = shangai_treat.columns
shangai_array_10 = shangai_treat.values

shangai_x10 = shangai_array_10[:,0:6]
shangai_y10 = shangai_array_10[:,6]

# Run Logistic Regression

model = LogisticRegression(solver='liblinear')
rfe = RFE(model, 3)
rec = rfe.fit(shangai_x10, shangai_y10)

# Automate column outputs for RFE

top_cols = col_list(rec.ranking_, shangai_cols)

print('\033[1m' + 'Recursive Feature Elimination' '\033[0m')
print()
print(f'Number of features {rec.n_features_:d}')
print(f'Selected features {rec.support_}')
print(f'Ranking of features {rec.ranking_}')
print()
print(f"Top features are {top_cols[0]}, {top_cols[1]}, and {top_cols[2]}")

# Run univariate analysis

test = SelectKBest(score_func=chi2, k=3)
uni = test.fit(shangai_x10, shangai_y10)

print()
print('\033[1m' + 'Univariate Analysis' '\033[0m')
uni.scores_

cols = col_dic(uni.scores_)
print("The top features are: " + shangai_cols[cols["first"]] + ", " + shangai_cols[cols["se

features=uni.transform(shangai_x10)
features[0:6,:]
```

```
Recursive Feature Elimination

Number of features 3
Selected features [ True False  True False  True False]
Ranking of features [1 4 1 3 1 2]

Top features are alumni, hici, and pub

Univariate Analysis
```

Out[14]:

```
array([3839.867, 7499.15 , 1446.554, 1397.945,  168.597,  681.022])
```

```
The top features are: award, alumni, hici
```

Out[14]:

```
array([[100. , 100. , 100. ],
       [ 40.7,  89.6,  80.1],
       [ 68.2,  80.7,  60.6],
       [ 65.1,  79.4,  66.1],
       [ 77.1,  96.6,  50.8],
       [ 53.3,  93.4,  57.1]])
```

In [15]:

```
# Shangai top-100 analysis

shangai_treat = dc.shangai_clean(100)
shangai_cols = shangai_treat.columns
shangai_array_100 = shangai_treat.values

shangai_x100 = shangai_array_100[:,0:6]
shangai_y100 = shangai_array_100[:,6]

# Run Logistic Regression

model = LogisticRegression(solver='liblinear')
rfe = RFE(model, 3)
rec = rfe.fit(shangai_x100, shangai_y100)

# Automate column outputs for RFE

top_cols = col_list(rec.ranking_, shangai_cols)

print('\033[1m' + 'Recursive Feature Elimination' '\033[0m')
print()
print(f'Number of features {rec.n_features_:d}')
print(f'Selected features {rec.support_}')
print(f'Ranking of features {rec.ranking_}')
print()
print(f"Top features are {top_cols[0]}, {top_cols[1]}, and {top_cols[2]}")

# Run univariate analysis

test = SelectKBest(score_func=chi2, k=3)
uni = test.fit(shangai_x100, shangai_y100)

print()
print('\033[1m' + 'Univariate Analysis' '\033[0m')
uni.scores_

cols = col_dic(uni.scores_)
print("The top features are: " + shangai_cols[cols["first"]] + ", " + shangai_cols[cols["se

features=uni.transform(shangai_x100)
features[0:6,:]
```

```
Recursive Feature Elimination

Number of features 3
Selected features [ True  True  True False False False]
Ranking of features [1 1 1 3 4 2]

Top features are alumni, award, and hici

Univariate Analysis
```

Out[15]:

```
array([3589.923, 7203.251, 2934.493, 2328.666,  665.928,  577.754])
```

```
The top features are: award, alumni, hici
```

Out[15]:

```
array([[100. , 100. , 100. ],
       [ 40.7,  89.6,  80.1],
       [ 68.2,  80.7,  60.6],
       [ 65.1,  79.4,  66.1],
       [ 77.1,  96.6,  50.8],
       [ 53.3,  93.4,  57.1]])
```

In [16]:

```
# Times top-10 analysis

times_treat = dc.times_clean(10)
times_cols = times_treat.columns
array_10 = times_treat.values

times_x10 = array_10[:,0:9]
times_y10 = array_10[:,9]

# Run Logistic Regression

model = LogisticRegression(solver='liblinear')
rfe = RFE(model, 3)
rec = rfe.fit(times_x10, times_y10)

# Automate column outputs for RFE

top_cols = col_list(rec.ranking_, times_cols)

print('\033[1m' + 'Recursive Feature Elimination' '\033[0m')
print()
print(f'Number of features {rec.n_features_:d}')
print(f'Selected features {rec.support_}')
print(f'Ranking of features {rec.ranking_}')
print()
print(f"Top features are {top_cols[0]}, {top_cols[1]}, and {top_cols[2]}")

# Run univariate analysis

test = SelectKBest(score_func=chi2, k=3)
uni = test.fit(times_x10, times_y10)

print()
print('\033[1m' + 'Univariate Analysis' '\033[0m')
print()
uni.scores_

cols = col_dic(uni.scores_)
print("The top features are: " + times_cols[cols["first"]] + ", " + times_cols[cols["second

features=uni.transform(times_x10)
features[0:9,:]
```

```
Recursive Feature Elimination

Number of features 3
Selected features [False False  True False False False  True False  True]
Ranking of features [4 5 1 3 6 2 1 7 1]

Top features are research, student_staff_ratio, and female_male_ratio

Univariate Analysis
```

Out[16]:

```
array([287.042,  32.202, 322.269,  26.701,  36.359,  96.145,  38.653,
        58.909,  18.233])

The top features are: research, teaching, num_students
```

Out[16]:

```
array([[95.6  , 97.6  ,  2.243],
       [86.5  , 98.9  , 19.919],
       [92.5  , 96.2  , 15.596],
       [88.2  , 96.7  , 18.812],
       [89.4  , 88.6  , 11.074],
       [85.1  , 91.9  ,  7.929],
       [83.3  , 88.5  , 15.06 ],
       [77.   , 95.   , 18.178],
       [85.7  , 88.9  , 14.221]])
```

In [17]:

```
# Times top-100 analysis

times_treat = dc.times_clean(100)
times_cols = times_treat.columns
array_100 = times_treat.values

times_x100 = array_100[:,0:9]
times_y100 = array_100[:,9]

# Run Logistic Regression

model = LogisticRegression(solver='liblinear')
rfe = RFE(model, 3)
rec = rfe.fit(times_x100, times_y100)

# Automate column outputs for RFE

top_cols = col_list(rec.ranking_, times_cols)

print('\033[1m' + 'Recursive Feature Elimination' '\033[0m')
print()
print(f'Number of features {rec.n_features_:d}')
print(f'Selected features {rec.support_}')
print(f'Ranking of features {rec.ranking_}')
print()
print(f"Top features are {top_cols[0]}, {top_cols[1]}, and {top_cols[2]}")

# Run univariate analysis

test = SelectKBest(score_func=chi2, k=3)
uni = test.fit(times_x100, times_y100)

print()
print('\033[1m' + 'Univariate Analysis' '\033[0m')
print()
uni.scores_

cols = col_dic(uni.scores_)
print("The top features are: " + times_cols[cols["first"]] + ", " + times_cols[cols["second

features=uni.transform(times_x100)
features[0:9,:]
```

```
Recursive Feature Elimination

Number of features 3
Selected features [False False  True False False False  True False  True]
Ranking of features [6 3 1 4 5 7 1 2 1]

Top features are research, student_staff_ratio, and female_male_ratio

Univariate Analysis
```

Out[17]:

```
array([3.567e+02, 1.948e+01, 5.892e+02, 3.776e+01, 5.731e+01, 9.772e+00,
       1.472e+01, 4.798e+01, 4.179e-01])

The top features are: research, teaching, income
```

Out[17]:

```
array([[95.6, 97.6, 97.8],
       [86.5, 98.9, 73.1],
       [92.5, 96.2, 63.3],
       [88.2, 96.7, 55. ],
       [89.4, 88.6, 95.4],
       [85.1, 91.9, 52.1],
       [83.3, 88.5, 53.7],
       [77. , 95. , 80. ],
       [85.7, 88.9, 36.6]])
```

In [18]:

```
print("As we can see for the times datasets, scores and the most important features have ch
print()
print("This is due to the fact that the output variable is different for top-10 and top-100
print()
print("The factors determining whether a university is elite (top-10) can be different from
print("university is very good (top-100).")
print()
print("Hence, different variables should explain the variance in the different output varia
print()
print("Also, there is no reason to believe that this should change for the shangai dataset"
```

```
As we can see for the times datasets, scores and the most important features
have changed for the top-10 or top-100.

This is due to the fact that the output variable is different for top-10 and
top-100.

The factors determining whether a university is elite (top-10) can be differ
ent from the factors determining whether a
university is very good (top-100).

Hence, different variables should explain the variance in the different outp
ut variable.

Also, there is no reason to believe that this should change for the shangai
dataset
```

Principal Component Analysis

Principal Component Analysis is a data reduction technique using linear algebra. The idea here is to "compress" several dimensions into pricipal components.

One problem of PCA is the explainability. Once you compressed the attributes into principal components you can no longer to refer them individually establishing causality links or relationships.

A property of PCA is that you can choose the number of dimensions or principal components. In our example we will select 3 principal components.

For Principal Component Analysis you use the **PCA** class.

In [19]:

```python
from sklearn.decomposition import PCA

p_indians.head()

#PCA
pca = PCA(n_components=3)
pca_fit = pca.fit(X)

print(f"Explained variance: {pca_fit.explained_variance_ratio_}")
print()

np.set_printoptions(formatter={'float': '{: 0.3f}'.format})
print("Principal Components have little resemblance to the source data attributes")
print()
print(pca_fit.components_)
```

Out[19]:

	pregnancies	glucose	pressure	skin	insulin	bmi	pedi	age	outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
Explained variance: [0.889 0.062 0.026]

Principal Components have little resemblance to the source data attributes

[[-0.002  0.098  0.016  0.061  0.993  0.014  0.001 -0.004]
 [-0.023 -0.972 -0.142  0.058  0.095 -0.047 -0.001 -0.140]
 [-0.022  0.143 -0.922 -0.307  0.021 -0.132 -0.001 -0.125]]
```

In [20]:

```python
# First component explains 0.889 of the variance, second explains 0.062, third explains 0.0
# First row is how the first component was formed, second row how the second component was
```

Feature Importance

One of the added features of tree based algorithms is that they can be used to estimate the importance of each feature and use it to refine the model to different levels depending on where we want to situate ourselves in the tension between explainability and accuracy.

In this example we are going to use the ExtraTreesClassifier, but the technique is commonly used in all tree algoritms.

For this example of assessing feature importance with trees we will use the **ExtraTreesClassifier** class.

In [21]:

```
from sklearn.ensemble import ExtraTreesClassifier

p_indians.head()

model = ExtraTreesClassifier(n_estimators=100)
model.fit(X,Y)

print(model.feature_importances_)
```

Out[21]:

	pregnancies	glucose	pressure	skin	insulin	bmi	pedi	age	outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Out[21]:

```
ExtraTreesClassifier(bootstrap=False, ccp_alpha=0.0, class_weight=None,
                     criterion='gini', max_depth=None, max_features='auto',
                     max_leaf_nodes=None, max_samples=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=100,
                     n_jobs=None, oob_score=False, random_state=None, verbos
e=0,
                     warm_start=False)
```

```
[ 0.108  0.228  0.100  0.078  0.077  0.141  0.120  0.147]
```

In [ ]:

Mission 2

a) Using the Shangai Data find the top attributes with a tree classifier for top-10, top-50 and top-100.

b) Same for the Times ranking.

In [24]:

```
# a) Using the Shangai Data find the top attributes with a tree classifier for top-10 and t


array_100 = dc.shangai_clean(100).values
array_50 = dc.shangai_clean(50).values
array_10 = dc.shangai_clean(10).values

x_100 = array_100[:,0:6]
y_100 = array_100[:,6]

x_50 = array_50[:,0:6]
y_50 = array_50[:,6]

x_10 = array_10[:,0:6]
y_10 = array_10[:,6]


# Model with top 100 estimators

print('\033[1m' + 'Shangai dataset with top-100 universities' '\033[0m')

model_100 = ExtraTreesClassifier(n_estimators=100)
model_100.fit(x_100,y_100)

print(model_100.feature_importances_)
indices_100 = model_100.feature_importances_.argsort()[::1][:3]
print()
print("The top 3 features are: " + shan_cols[indices_100[0]] + ", " + shan_cols[indices_100
print()


# Model with top 50 estimators

print('\033[1m' + 'Shangai dataset with top-50 universities' '\033[0m')

model_50 = ExtraTreesClassifier(n_estimators=100)
model_50.fit(x_50,y_50)

print(model_50.feature_importances_)
indices_50 = model_50.feature_importances_.argsort()[::1][:3]
print()
print("The top 3 features are: " + shan_cols[indices_50[0]] + ", " + shan_cols[indices_50[1
print()


# Model with top 10 estimators

print('\033[1m' + 'Shangai dataset with top-10 universities' '\033[0m')

model_10 = ExtraTreesClassifier(n_estimators=100)
model_10.fit(x_10,y_10)

print(model_10.feature_importances_)
indices_10 = model_10.feature_importances_.argsort()[::1][:3]
print()
print("The top 3 features are: " + shan_cols[indices_10[0]] + ", " + shan_cols[indices_10[1
print()
```

Shangai dataset with top-100 universities

Out[24]:

```
ExtraTreesClassifier(bootstrap=False, ccp_alpha=0.0, class_weight=None,
                     criterion='gini', max_depth=None, max_features='auto',
                     max_leaf_nodes=None, max_samples=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=100,
                     n_jobs=None, oob_score=False, random_state=None, verbos
e=0,
                     warm_start=False)
```

```
[ 0.103  0.229  0.242  0.196  0.129  0.101]

The top 3 features are: pcp, alumni, pub
```

Shangai dataset with top-50 universities

Out[24]:

```
ExtraTreesClassifier(bootstrap=False, ccp_alpha=0.0, class_weight=None,
                     criterion='gini', max_depth=None, max_features='auto',
                     max_leaf_nodes=None, max_samples=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=100,
                     n_jobs=None, oob_score=False, random_state=None, verbos
e=0,
                     warm_start=False)
```

```
[ 0.110  0.233  0.191  0.275  0.126  0.064]

The top 3 features are: pcp, alumni, pub
```

Shangai dataset with top-10 universities

Out[24]:

```
ExtraTreesClassifier(bootstrap=False, ccp_alpha=0.0, class_weight=None,
                     criterion='gini', max_depth=None, max_features='auto',
                     max_leaf_nodes=None, max_samples=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=100,
                     n_jobs=None, oob_score=False, random_state=None, verbos
e=0,
                     warm_start=False)
```

```
[ 0.269  0.308  0.092  0.119  0.040  0.172]

The top 3 features are: pub, hici, ns
```

In [23]:

```
# b) Same for the Times ranking.

array_100 = dc.times_clean(100).values
array_50 = dc.times_clean(50).values
array_10 = dc.times_clean(10).values

x_100 = array_100[:,0:9]
y_100 = array_100[:,9]

x_50 = array_50[:,0:9]
y_50 = array_50[:,9]

x_10 = array_10[:,0:9]
y_10 = array_10[:,9]

# Model with top 100 estimators

print('\033[1m' + 'Times dataset with top-100 universities' '\033[0m')

model_100 = ExtraTreesClassifier(n_estimators=100)
model_100.fit(x_100,y_100)

indices_100 = model_100.feature_importances_.argsort()[::1][:3]
print("The top 3 features are: " + times_cols[indices_100[0]] + ", " + times_cols[indices_1
print()

print(model_100.feature_importances_)

# Model with top 50 estimators

print()
print('\033[1m' + 'Times dataset with top-50 universities' '\033[0m')

model_50 = ExtraTreesClassifier(n_estimators=100)
model_50.fit(x_50,y_50)

indices_50 = model_50.feature_importances_.argsort()[::1][:3]
print("The top 3 features are: " + times_cols[indices_50[0]] + ", " + times_cols[indices_50
print()

print(model_50.feature_importances_)

# Model with top 10 estimators

print()
print('\033[1m' + 'Times dataset with top-10 universities' '\033[0m')

model_10 = ExtraTreesClassifier(n_estimators=100)
model_10.fit(x_10,y_10)

indices_10 = model_10.feature_importances_.argsort()[::1][:3]
print("The top 3 features are: " + times_cols[indices_10[0]] + ", " + times_cols[indices_10
print()

print(model_100.feature_importances_)
```

Times dataset with top-100 universities

Out[23]:

```
ExtraTreesClassifier(bootstrap=False, ccp_alpha=0.0, class_weight=None,
                     criterion='gini', max_depth=None, max_features='auto',
                     max_leaf_nodes=None, max_samples=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=100,
                     n_jobs=None, oob_score=False, random_state=None, verbos
e=0,
                     warm_start=False)
```

```
The top 3 features are: num_students, female_male_ratio, student_staff_ratio

[ 0.245  0.051  0.338  0.122  0.053  0.040  0.050  0.055  0.045]
```

Times dataset with top-50 universities

Out[23]:

```
ExtraTreesClassifier(bootstrap=False, ccp_alpha=0.0, class_weight=None,
                     criterion='gini', max_depth=None, max_features='auto',
                     max_leaf_nodes=None, max_samples=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=100,
                     n_jobs=None, oob_score=False, random_state=None, verbos
e=0,
                     warm_start=False)
```

```
The top 3 features are: num_students, student_staff_ratio, female_male_ratio

[ 0.336  0.033  0.410  0.070  0.034  0.020  0.024  0.046  0.026]
```

Times dataset with top-10 universities

Out[23]:

```
ExtraTreesClassifier(bootstrap=False, ccp_alpha=0.0, class_weight=None,
                     criterion='gini', max_depth=None, max_features='auto',
                     max_leaf_nodes=None, max_samples=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=100,
                     n_jobs=None, oob_score=False, random_state=None, verbos
e=0,
                     warm_start=False)
```

```
The top 3 features are: student_staff_ratio, income, num_students

[ 0.245  0.051  0.338  0.122  0.053  0.040  0.050  0.055  0.045]
```

In [ ]:

```

```