Лабораторна робота №5.
Аналіз демографічних даних

У завданні Вам пропонується навчити і налаштувати дерева і (при бажанні) випадковий ліс в завданні класифікації на даних Adult сховища UCI.

Підключаємо необхідні бібліотеки

In [1]:
```
%matplotlib inline
from matplotlib import pyplot as plt
plt.rcParams['figure.figsize'] = (10, 8)
import seaborn as sns
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import collections
from sklearn.model_selection import GridSearchCV
from sklearn import preprocessing
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprecated. Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

Спочатку налаштуємо доступ до даних на google drive (якщо ви відкриваєте блокнот в google colab, а не на PC) шляхом монтування google drive

In [2]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

Перевіримо шлях до папки з матеріалами лабораторної роботи на google drive. Якщо у вас шлях відрізняється то відредагуйте

In [3]:
```
!ls gdrive/'My Drive'/TEACHING/IntroDataScience/intro_to_data_science/Lab_5_6/da
```

```
adult.data.csv  adult_test.csv  adult_train.csv  telecom_churn.csv
```

Перемістимо матеріали лабораторної роботи з google drive на віртуальну машину google colab

In [4]:
```
!cp -a gdrive/'My Drive'/TEACHING/IntroDataScience/intro_to_data_science/Lab_5_6/
!ls
```

```
data    lab0_intro_decision_tree.ipynb  lab5_decision_trees.ipynb
gdrive  lab0_intro_decision_tree.pdf    sample_data
```

Опис набору даних

[Набір] (http://archive.ics.uci.edu/ml/machine-learning-databases/adult (http://archive.ics.uci.edu/ml/machine-learning-databases/adult)) даних UCI Adult (качати не треба, все є в репозиторії): класифікація людей за допомогою демографічних даних для прогнозування чи заробляє людина більш \ $ 50 000 в рік.

Опис ознак:

Age - вік, кількісний ознака

Workclass - тип роботодавця, кількісний ознака

fnlwgt - підсумковий вага об'єкта, кількісний ознака

Education - рівень освіти, якісної ознаки

Education_Num - кількість років навчання, кількісний ознака

Martial_Status - сімейний стан, категоріальний ознака

Occupation - професія, категоріальний ознака

Relationship - тип сімейних відносин, категоріальний ознака

Race - раса, категоріальний ознака

Sex - підлогу, якісна ознака

Capital_Gain - приріст капіталу, кількісний ознака

Capital_Loss - втрати капіталу, кількісний ознака

Hours_per_week - кількість годин роботи в тиждень, кількісний ознака

Country - країна, категоріальний ознака

Цільова змінна: **Target** - рівень заробітку, категоріальна (бінарна) ознака

Зчитуємо навчальну і тестову вибірки.

In [6]:
```
data_train = pd.read_csv('data/adult_train.csv', sep=';')
```

In [7]:
```
data_train.tail()
```

Out[7]:

	Age	Workclass	fnlwgt	Education	Education_Num	Martial_Status	Occupation	Relationsh
32556	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	W
32557	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husba
32558	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarri
32559	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-ch
32560	52	Self-emp-inc	287927	HS-grad	9	Married-civ-spouse	Exec-managerial	W

In [8]:
```
data_test = pd.read_csv('data/adult_test.csv', sep=';')
```

In [9]:
```
data_test.tail()
```

Out[9]:

	Age	Workclass	fnlwgt	Education	Education_Num	Martial_Status	Occupation	Relation
16277	39	Private	215419.0	Bachelors	13.0	Divorced	Prof-specialty	Not-in-fa
16278	64	NaN	321403.0	HS-grad	9.0	Widowed	NaN	Other-rel
16279	38	Private	374983.0	Bachelors	13.0	Married-civ-spouse	Prof-specialty	Husb
16280	44	Private	83891.0	Bachelors	13.0	Divorced	Adm-clerical	Own-
16281	35	Self-emp-inc	182148.0	Bachelors	13.0	Married-civ-spouse	Exec-managerial	Husb

In [10]:
```
# необхідно прибрати рядки з неправильними мітками в тестовій вибірці
data_test = data_test[(data_test['Target'] == ' >50K.')
                      | (data_test['Target']==' <=50K.')]

# перекодуємо target в числове поле
data_train.at[data_train['Target'] == ' <=50K', 'Target'] = 0
data_train.at[data_train['Target'] == ' >50K', 'Target'] = 1

data_test.at[data_test['Target'] == ' <=50K.', 'Target'] = 0
data_test.at[data_test['Target'] == ' >50K.', 'Target'] = 1
```

Первинний аналіз даних.

In [11]:
```
data_test.describe(include='all').T
```

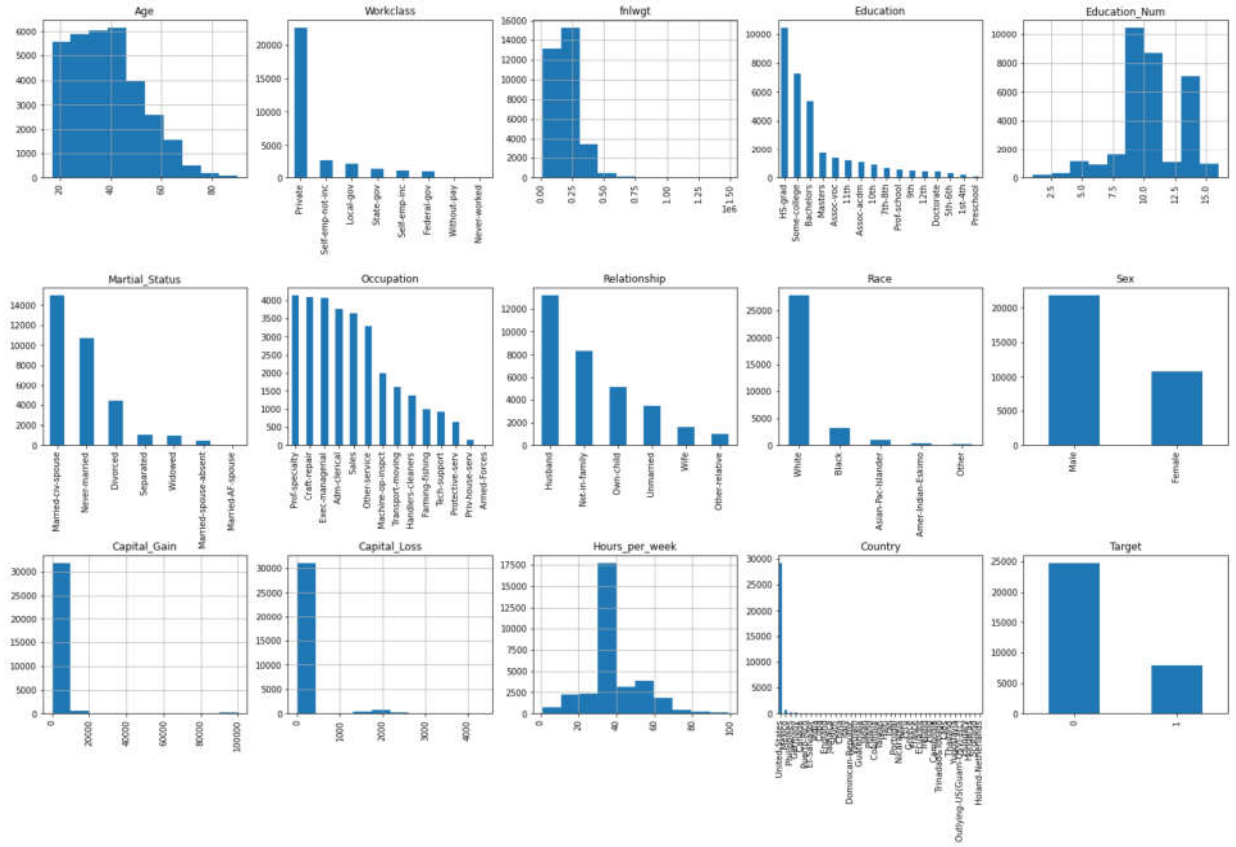
Out[11]:

	count	unique	top	freq	mean	std	min	25%	50%	75%
Age	16281	73	35	461	NaN	NaN	NaN	NaN	NaN	NaN
Workclass	15318	8	Private	11210	NaN	NaN	NaN	NaN	NaN	NaN
fnlwgt	16281	NaN	NaN	NaN	189436	105715	13492	116736	177831	23838
Education	16281	16	HS-grad	5283	NaN	NaN	NaN	NaN	NaN	NaN
Education_Num	16281	NaN	NaN	NaN	10.0729	2.56755	1	9	10	12
Martial_Status	16281	7	Married-civ-spouse	7403	NaN	NaN	NaN	NaN	NaN	NaN
Occupation	15315	14	Prof-specialty	2032	NaN	NaN	NaN	NaN	NaN	NaN
Relationship	16281	6	Husband	6523	NaN	NaN	NaN	NaN	NaN	NaN
Race	16281	5	White	13946	NaN	NaN	NaN	NaN	NaN	NaN
Sex	16281	2	Male	10860	NaN	NaN	NaN	NaN	NaN	NaN
Capital_Gain	16281	NaN	NaN	NaN	1081.91	7583.94	0	0	0	0
Capital_Loss	16281	NaN	NaN	NaN	87.8993	403.105	0	0	0	0
Hours_per_week	16281	NaN	NaN	NaN	40.3922	12.4793	1	40	40	45
Country	16007	40	United-States	14662	NaN	NaN	NaN	NaN	NaN	NaN
Target	16281	2	0	12435	NaN	NaN	NaN	NaN	NaN	NaN

In [12]:
```
data_train['Target'].value_counts()
```

Out[12]:
```
0    24720
1     7841
Name: Target, dtype: int64
```

In [13]:
```
fig = plt.figure(figsize=(25, 15))
cols = 5
rows = np.ceil(float(data_train.shape[1]) / cols)
for i, column in enumerate(data_train.columns):
    ax = fig.add_subplot(rows, cols, i + 1)
    ax.set_title(column)
    if data_train.dtypes[column] == np.object:
        data_train[column].value_counts().plot(kind="bar", axes=ax)
    else:
        data_train[column].hist(axes=ax)
        plt.xticks(rotation="vertical")
plt.subplots_adjust(hspace=0.7, wspace=0.2)
```



Перевіряємо типи даних

In [14]:
```
data_train.dtypes
```

Out[14]:
```
Age                int64
Workclass         object
fnlwgt             int64
Education         object
Education_Num      int64
Martial_Status    object
Occupation        object
Relationship      object
Race              object
Sex               object
Capital_Gain       int64
Capital_Loss       int64
Hours_per_week     int64
Country           object
Target            object
dtype: object
```

In [15]:
```
data_test.dtypes
```

Out[15]:
```
Age                object
Workclass          object
fnlwgt            float64
Education          object
Education_Num     float64
Martial_Status     object
Occupation         object
Relationship       object
Race               object
Sex                object
Capital_Gain      float64
Capital_Loss      float64
Hours_per_week    float64
Country            object
Target             object
dtype: object
```

З'ясувалося, що в тесті вік віднесений до типу object, необхідно це виправити.

In [16]:
```
data_test['Age'] = data_test['Age'].astype(int)
```

Також наведемо показники типу float в int для відповідності train і test вибірок.

In [17]:
```
data_test['fnlwgt'] = data_test['fnlwgt'].astype(int)
data_test['Education_Num'] = data_test['Education_Num'].astype(int)
data_test['Capital_Gain'] = data_test['Capital_Gain'].astype(int)
data_test['Capital_Loss'] = data_test['Capital_Loss'].astype(int)
data_test['Hours_per_week'] = data_test['Hours_per_week'].astype(int)
```

Заповнимо пропуски в кількісних полях медіанним значенням, а в категоріальних - значенням, що найчастіше зустрічається

In [18]:
```
# виділимо в вибірках категоріальні і числові поля

categorical_columns_train = [c for c in data_train.columns
                             if data_train[c].dtype.name == 'object']
numerical_columns_train = [c for c in data_train.columns
                           if data_train[c].dtype.name != 'object']

categorical_columns_test = [c for c in data_test.columns
                            if data_test[c].dtype.name == 'object']
numerical_columns_test = [c for c in data_test.columns
                          if data_test[c].dtype.name != 'object']

print('categorical_columns_test:', categorical_columns_test)
print('categorical_columns_train:', categorical_columns_train)
print('numerical_columns_test:', numerical_columns_test)
print('numerical_columns_train:', numerical_columns_train)
```

```
categorical_columns_test: ['Workclass', 'Education', 'Martial_Status', 'Occupat
ion', 'Relationship', 'Race', 'Sex', 'Country', 'Target']
categorical_columns_train: ['Workclass', 'Education', 'Martial_Status', 'Occupa
tion', 'Relationship', 'Race', 'Sex', 'Country', 'Target']
numerical_columns_test: ['Age', 'fnlwgt', 'Education_Num', 'Capital_Gain', 'Cap
ital_Loss', 'Hours_per_week']
numerical_columns_train: ['Age', 'fnlwgt', 'Education_Num', 'Capital_Gain', 'Ca
pital_Loss', 'Hours_per_week']
```

In [19]:
```
# заповнимо пропуски

for c in categorical_columns_train:
    data_train[c] = data_train[c].fillna(data_train[c].mode())
for c in categorical_columns_test:
    data_test[c] = data_test[c].fillna(data_train[c].mode())

for c in numerical_columns_train:
    data_train[c] = data_train[c].fillna(data_train[c].median())
for c in numerical_columns_test:
    data_test[c] = data_test[c].fillna(data_train[c].median())
```

Кодуємо категоріальні ознаки 'Workclass', 'Education', 'Martial_Status', 'Occupation', 'Relationship', 'Race', 'Sex', 'Country'. Це можна зробити за допомогою методу `pandas get_dummies` .

In [20]:
```
data_train = pd.concat([data_train, pd.get_dummies(data_train['Workclass'],
                                                   prefix="Workclass"),
                        pd.get_dummies(data_train['Education'], prefix="Education"
                        pd.get_dummies(data_train['Martial_Status'], prefix="Marti
                        pd.get_dummies(data_train['Occupation'], prefix="Occupatio
                        pd.get_dummies(data_train['Relationship'], prefix="Relatio
                        pd.get_dummies(data_train['Race'], prefix="Race"),
                        pd.get_dummies(data_train['Sex'], prefix="Sex"),
                        pd.get_dummies(data_train['Country'], prefix="Country")],
                       axis=1)

data_test = pd.concat([data_test, pd.get_dummies(data_test['Workclass'], prefix='
                       pd.get_dummies(data_test['Education'], prefix="Education"),
                       pd.get_dummies(data_test['Martial_Status'], prefix="Martia
                       pd.get_dummies(data_test['Occupation'], prefix="Occupation'
                       pd.get_dummies(data_test['Relationship'], prefix="Relations
                       pd.get_dummies(data_test['Race'], prefix="Race"),
                       pd.get_dummies(data_test['Sex'], prefix="Sex"),
                       pd.get_dummies(data_test['Country'], prefix="Country")],
                      axis=1)
```

In [21]:
```
data_train.drop(['Workclass', 'Education', 'Martial_Status',
                 'Occupation', 'Relationship', 'Race', 'Sex', 'Country'],
                axis=1, inplace=True)
data_test.drop(['Workclass', 'Education', 'Martial_Status', 'Occupation',
                'Relationship', 'Race', 'Sex', 'Country'],
               axis=1, inplace=True)
```

In [22]:
```
data_test.describe(include='all').T
```

Out[22]:

	count	mean	std	min	25%	50%	75%	
Age	16281.0	38.767459	13.849187	17.0	28.0	37.0	48.0	
fnlwgt	16281.0	189435.677784	105714.907671	13492.0	116736.0	177831.0	238384.0	1
Education_Num	16281.0	10.072907	2.567545	1.0	9.0	10.0	12.0	
Capital_Gain	16281.0	1081.905104	7583.935968	0.0	0.0	0.0	0.0	
Capital_Loss	16281.0	87.899269	403.105286	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	
Country_ Thailand	16281.0	0.000737	0.027140	0.0	0.0	0.0	0.0	
Country_ Trinadad&Tobago	16281.0	0.000491	0.022162	0.0	0.0	0.0	0.0	
Country_ United-States	16281.0	0.900559	0.299262	0.0	1.0	1.0	1.0	
Country_ Vietnam	16281.0	0.001167	0.034143	0.0	0.0	0.0	0.0	
Country_ Yugoslavia	16281.0	0.000430	0.020731	0.0	0.0	0.0	0.0	

105 rows × 8 columns

In [23]:
```
set(data_train.columns) - set(data_test.columns)
```

Out[23]:
```
{'Country_ Holand-Netherlands'}
```

In [24]:
```
data_train.shape, data_test.shape
```

Out[24]:
```
((32561, 106), (16281, 105))
```

У тестовій вибірці не виявилося Голландії. Заведемо необхідну ознаку з нулів.

In [25]:
```
data_test['Country_ Holand-Netherlands'] = np.zeros([data_test.shape[0], 1])
```

In [26]:
```
set(data_train.columns) - set(data_test.columns)
```

Out[26]:
```
set()
```

In [27]:
```
data_train.head(2)
```

Out[27]:

	Age	fnlwgt	Education_Num	Capital_Gain	Capital_Loss	Hours_per_week	Target	Workclass_ Federal-gov
0	39	77516	13	2174	0	40	0	0
1	50	83311	13	0	0	13	0	0

2 rows × 106 columns

In [28]:
```
data_test.head(2)
```

Out[28]:

	Age	fnlwgt	Education_Num	Capital_Gain	Capital_Loss	Hours_per_week	Target	Workclass_ Federal-gov
1	25	226802	7	0	0	40	0	0
2	38	89814	9	0	0	50	0	0

2 rows × 106 columns

In [29]:
```
X_train=data_train.drop(['Target'], axis=1)
y_train = data_train['Target']

X_test=data_test.drop(['Target'], axis=1)
y_test = data_test['Target']
```

Дерево рішень без налаштування параметрів

Навчіть на наявній вибірці дерево рішень (`DecisionTreeClassifier`) максимальної глибини 3 і отримайте якість на тесті. Використовуйте параметр `random_state = 17` для відтворюваності результатів.

In [ ]:
```
tree = # Ваш код тут
tree.fit # Ваш код тут
```

Зробіть за допомогою отриманої моделі прогноз для тестової вибірки.

In [ ]:
```
tree_predictions = tree.predict # Ваш код тут
```

In [ ]:
```
accuracy_score # Ваш код тут
```

Питання 6. Яка частка правильних відповідей дерева рішень на тестовій вибірці при максимальній глибині дерева = 3 і random_state = 17?

Дерево рішень з налаштуванням параметрів

Навчіть на наявній вибірці дерево рішень (`DecisionTreeClassifier` , знову `random_state` = 17). Максимальну глибину налаштуйте на крос-валідації за допомогою `GridSearchCV` . Проведіть 5-кратну крос-валідацію

In [ ]:
```
tree_params = {'max_depth': range(2,11)}

locally_best_tree = GridSearchCV # Ваш код тут

locally_best_tree.fit # Ваш код тут
```

In [ ]:
```
print("Best params:", locally_best_tree.best_params_)
print("Best cross validaton score", locally_best_tree.best_score_)
```

Навчіть на наявній вибірці дерево рішень максимальної глибини 9 (це краще значення `max_depth` в моєму випадку) та оцініть частку правильних відповідей на тесті. Використовуйте параметр random_state = 17 для відтворюваності результатів.

In [ ]:
```
tuned_tree = # Ваш код тут
tuned_tree.fit # Ваш код тут
tuned_tree_predictions = tuned_tree.predict # Ваш код тут
accuracy_score # Ваш код тут
```

Питання 7. Яка частка правильних відповідей дерева рішень на тестовій вибірці при максимальній глибині дерева = 9 і random_state = 17?

2.3. Випадковий ліс без налаштування параметрів (опціонально) ¶

Навчіть на наявній вибірці випадковий ліс (`RandomForestClassifier`), число дерев зробіть рівним ста, а `random_state` = 17.

In [ ]:
```
rf = # Ваш код тут
rf.fit # Ваш код тут
```

Зробіть за допомогою отриманої моделі прогноз для тестової вибірки.

In [ ]:
```
forest_predictions = rf.predict # Ваш код тут
```

In [ ]:
```
accuracy_score # Ваш код тут
```

2.4. Випадковий ліс з налаштуванням параметрів

Навчіть на наявній вибірці випадковий ліс (`RandomForestClassifier`). Максимальну глибину і максимальне число ознак для кожного дерева налаштуйте за допомогою GridSearchCV.

In [ ]:
```
forest_params = {'max_depth': range(10, 21),
                 'max_features': range(5, 105, 10)}

locally_best_forest = GridSearchCV # Ваш код тут

locally_best_forest.fit # Ваш код тут
```

In [ ]:
```
print("Best params:", locally_best_forest.best_params_)
print("Best cross validaton score", locally_best_forest.best_score_)
```

Зробіть за допомогою отриманої моделі прогноз для тестової вибірки.

In [ ]:
```
tuned_forest_predictions = locally_best_forest.predict # Ваш код тут
accuracy_score # Ваш код тут
```