Projet P6
US ML TEMPLATE - FROM (almost) SCRATCH TO PREDICTION

```
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             32561 non-null  int64
 1   workClass       32561 non-null  object
 2   fnlwgt          32561 non-null  int64
 3   education       32561 non-null  object
 4   education-num   32561 non-null  int64
 5   marital-status  32561 non-null  object
 6   occupation      32561 non-null  object
 7   relationship    32561 non-null  object
 8   race            32561 non-null  object
 9   sex             32561 non-null  object
 10  capital-gain    32561 non-null  int64
 11  capital-loss    32561 non-null  int64
 12  hours-per-week  32561 non-null  int64
```

Ophélie Sabanowski
Florent Vanhollebeke

JANUARY 2022

The data

Data Set Characteristics:	Multivariate	**Number of Instances:**	48842	**Area:**	Social
Attribute Characteristics:	Categorical, Integer	**Number of Attributes:**	14	**Date Donated**	1996-05-01
Associated Tasks:	Classification	**Missing Values?**	Yes	**Number of Web Hits:**	2362543

Predict whether income exceeds $50K/yr based on census data.

- adult.data
  32537 rows × 15 columns
- adult.test
  16282 rows × 15 columns

Following the study of the data, we decided to delete the columns :

- education: by keeping education num, the information seemed sufficient to us
- fnlwgt: after research this data would correspond to the number of individuals with the same characteristics, we did not consider this information relevant

```
X_test.replace(' *','', regex=True, inplace=True)
```

Treatment of periods, question marks and spaces

```
X_test = X_test[(X_test["occupation"] != '?')]
X_test = X_test[(X_test["workClass"] != '?')]
X_test = X_test[(X_test['native-country'] != '?')]
X_test
```

```
[4]: df.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 32537 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             32537 non-null  int64
 1   workClass       32537 non-null  object
 2   fnlwgt          32537 non-null  int64
 3   education       32537 non-null  object
 4   education-num   32537 non-null  int64
 5   marital-status  32537 non-null  object
 6   occupation      32537 non-null  object
 7   relationship    32537 non-null  object
 8   race            32537 non-null  object
 9   sex             32537 non-null  object
 10  capital-gain    32537 non-null  int64
 11  capital-loss    32537 non-null  int64
 12  hours-per-week  32537 non-null  int64
 13  native-country  32537 non-null  object
 14  income          32537 non-null  object
dtypes: int64(6), object(9)
memory usage: 4.0+ MB
```

Data analysis with info and describe functions for numeric data.

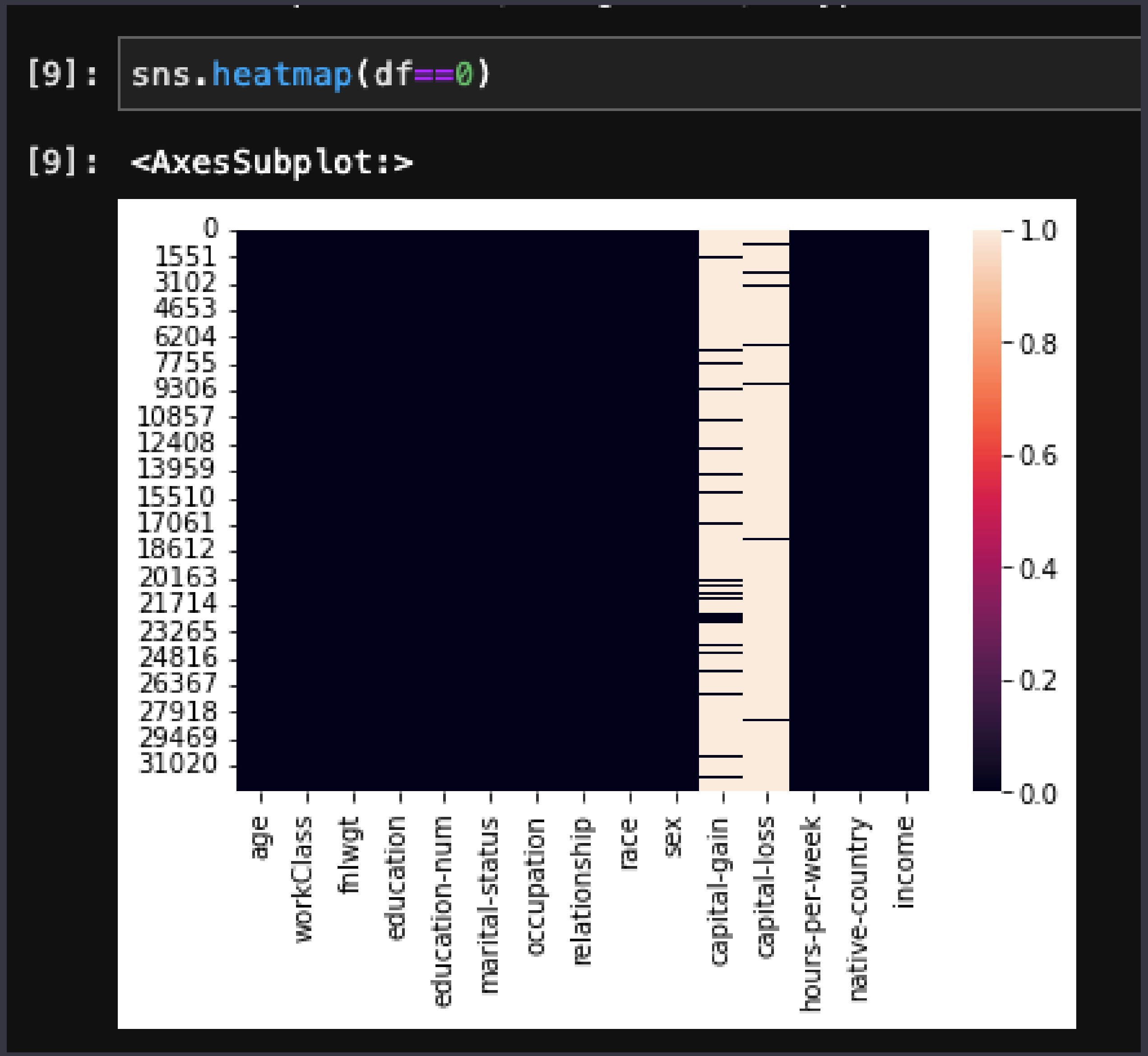
We notice that there are 9 categorical columns.

We notice that on the columns capital_gain and capital_loss, there are more than 75% of zero values.

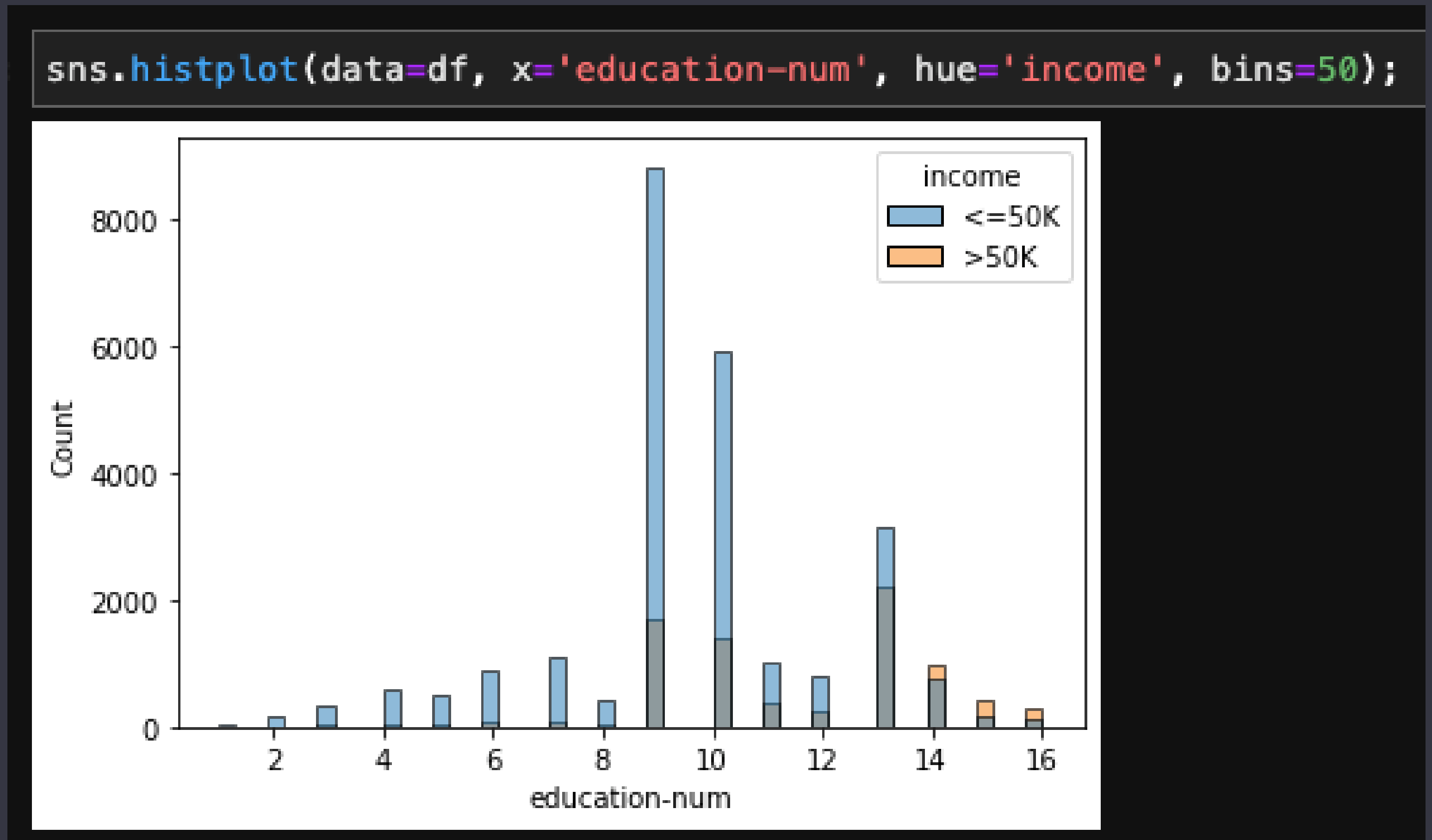
```
[5]: df.describe()
```

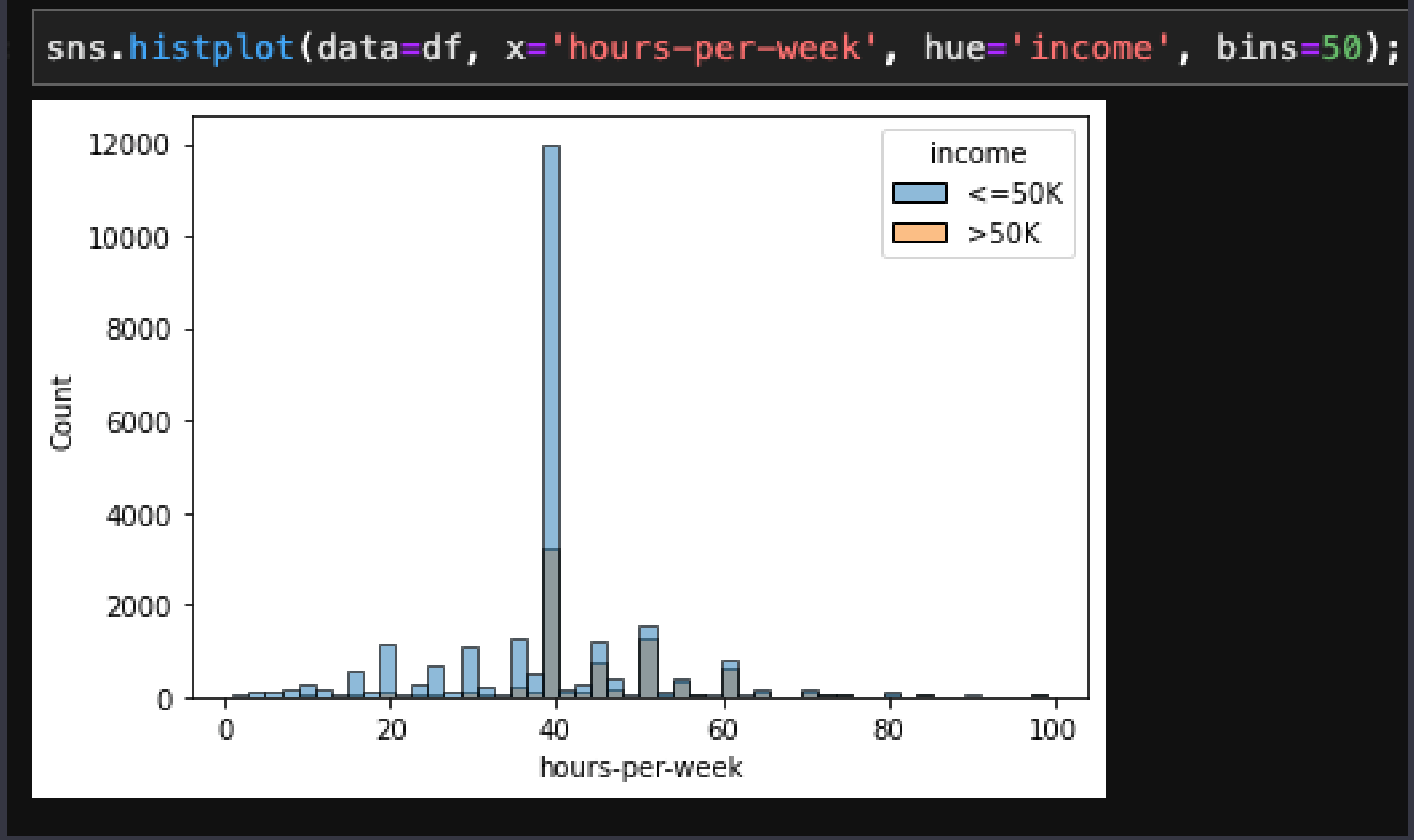
[5]:

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week
count	32561.000000	3.256100e+04	32561.000000	32561.000000	32561.000000	32561.000000
mean	38.581647	1.897784e+05	10.080679	1077.648844	87.303830	40.437456
std	13.640433	1.055500e+05	2.572720	7385.292085	402.960219	12.347429
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.000000
25%	28.000000	1.178270e+05	9.000000	0.000000	0.000000	40.000000
50%	37.000000	1.783560e+05	10.000000	0.000000	0.000000	40.000000
75%	48.000000	2.370510e+05	12.000000	0.000000	0.000000	45.000000
max	90.000000	1.484705e+06	16.000000	99999.000000	4356.000000	99.000000

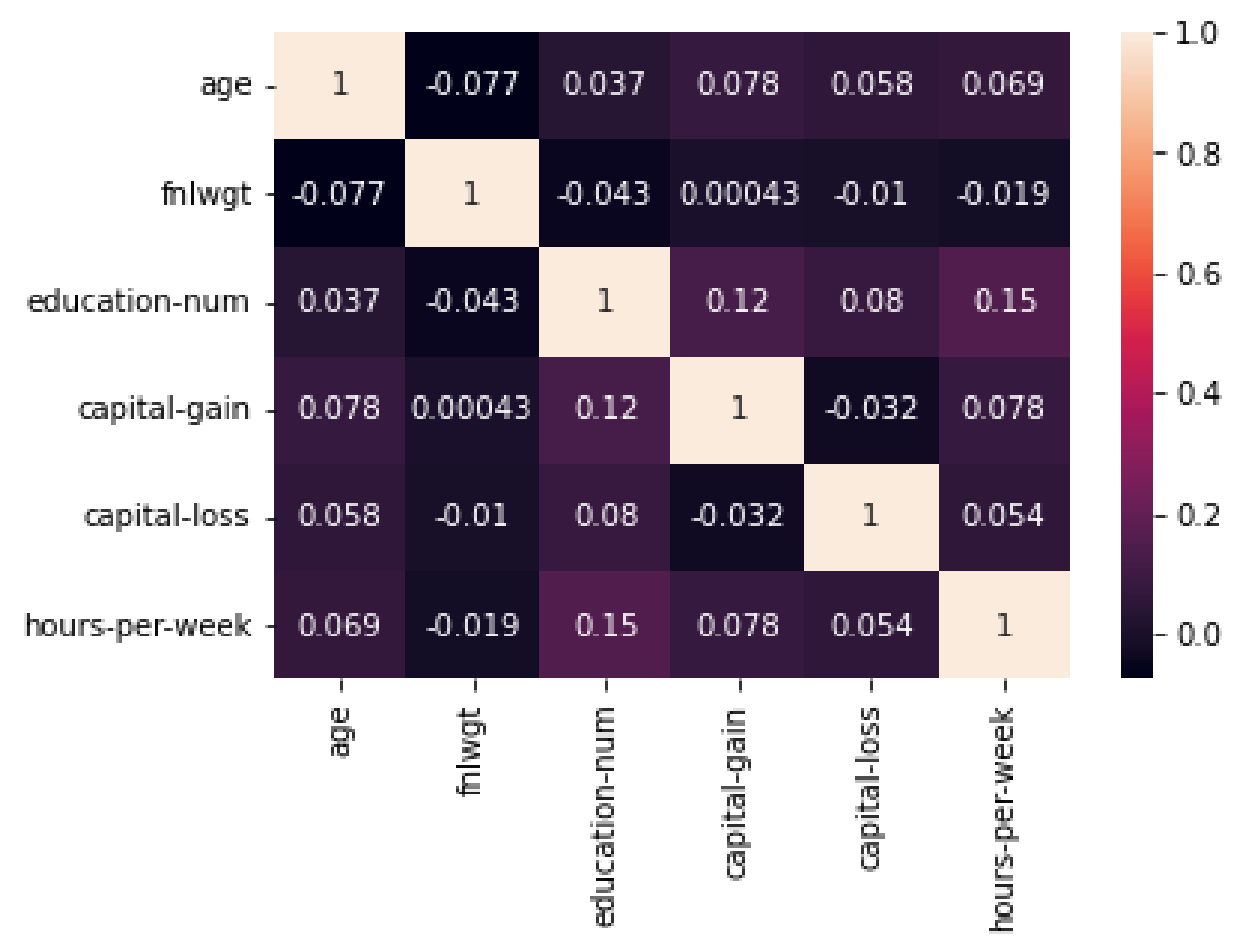


We looked at the data for a few columns graphically.
The values of capital_gain and capital_loss have a lot of 0 values but we decide to keep the information.
We note that the level of education has an impact on earnings.





```
[11]: sns.heatmap(df.corr(), annot=True);
```



Here we have represented a heatmap showing the correlations.
We do not see any explanatory columns.

```python
sns.pairplot(data=df, hue='income');
```

The "capital gain" variable seems rather correlated with age, the number of years of study and the number of hours worked per week.

```python
def regroupe(row):
    if row['new_race'] == 'Amer':
        return 'Others_race'
    elif row['new_race'] == 'Asian':
        return 'Asian'
    elif row['new_race'] == 'Black':
        return 'Black'
    elif row['new_race'] == 'Other':
        return 'Others_race'
    elif row['new_race'] == 'White':
        return 'White'
    else:
        return row['new_race']


df['new_race2'] = df.apply(regroupe, axis=1)
```

```python
df.new_race2.value_counts()
```

```
White          27816
Black           3124
Asian           1039
Others_race      582
Name: new_race2, dtype: int64
```

We have grouped by similar category the variables on the columns race, marital status, workclass and native country.

```python
df = pd.concat((df.drop('sex', axis=1), pd.get_dummies(df.sex, drop_first=True)), axis=1)
df
```

	age	workClass	education-num	occupation	relationship	capital-gain	capital-loss	hours-per-week	native-country	income	new_race2	marital-status2	Male
0	39	Public	13	Adm-clerical	Not-in-family	2174	0	40	United-States	<=50K	White	Never-married	1
1	50	Self-emp	13	Exec-managerial	Husband	0	0	13	United-States	<=50K	White	Married-civ-spouse	1
2	38	Private	9	Handlers-cleaners	Not-in-family	0	0	40	United-States	<=50K	White	Alone	1
3	53	Private	7	Handlers-cleaners	Husband	0	0	40	United-States	<=50K	Black	Married-civ-spouse	1
4	28	Private	13	Prof-specialty	Wife	0	0	40	Others_country	<=50K	Black	Married-civ-spouse	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
32556	27	Private	12	Tech-support	Wife	0	0	38	United-States	<=50K	White	Married-civ-spouse	0
32557	40	Private	9	Machine-op-inspct	Husband	0	0	40	United-States	>50K	White	Married-civ-spouse	1
32558	58	Private	9	Adm-clerical	Unmarried	0	0	40	United-States	<=50K	White	Alone	0
32559	22	Private	9	Adm-clerical	Own-child	0	0	20	United-States	<=50K	White	Never-married	1
32560	52	Self-emp	9	Exec-managerial	Wife	15024	0	40	United-States	>50K	White	Married-civ-spouse	0

30162 rows × 13 columns

Then creation of dummies on our categorical columns.

```python
df = pd.concat((df.drop('new_race2', axis=1), pd.get_dummies(df['new_race2'], drop_first=True)), axis=1)
df = pd.concat((df.drop('workClass', axis=1), pd.get_dummies(df['workClass'], drop_first=True)), axis=1)
df =  pd.concat((df.drop('occupation', axis=1), pd.get_dummies(df['occupation'], drop_first=True)), axis=1)
df =  pd.concat((df.drop('relationship', axis=1), pd.get_dummies(df['relationship'], drop_first=True)), axis=1)
df =  pd.concat((df.drop('native-country', axis=1), pd.get_dummies(df['native-country'], drop_first=True)), axis=1)
df =  pd.concat((df.drop('marital-status2', axis=1), pd.get_dummies(df['marital-status2'], drop_first=True)), axis=1)
df
```

We performed the same data processing on X_test

Creation of y_test

```
y_test = X_test.loc[:,'income']
y_test = pd.DataFrame(y_test)
y_test
```

	income
1	<=50K.
2	<=50K.
3	>50K.
4	>50K.
6	<=50K.
...	...
16276	<=50K.
16277	<=50K.
16279	<=50K.
16280	<=50K.
16281	>50K.

15060 rows × 1 columns

```
X_test = X_test.drop('income', axis=1)
X_test
```

Shapes display

```
X_train.shape, y_train.shape
```

```
((30139, 34), (30139,))
```

```
X_test.shape, y_test.shape
```

```
((15060, 35), (15060,))
```

Standardization of X

```
# Standardisation des données

sc = StandardScaler()
X_train_sc = sc.fit_transform(X_train)
X_test_sc = sc.transform(X_test)
X_train_sc = pd.DataFrame(X_train_sc)
X_test_sc = pd.DataFrame(X_test_sc)
X_test_sc
```

	0	1	2	3	4	5	6	
0	-1.023647	-1.225149	-0.147502	-0.218673	-0.078031	0.692725	3.114927	-0.132
1	-0.033639	-0.440434	-0.147502	-0.218673	0.756794	0.692725	-0.321035	-0.132
2	-0.795183	0.736639	-0.147502	-0.218673	-0.078031	0.692725	-0.321035	-0.132
3	0.423288	-0.048076	0.890157	-0.218673	-0.078031	0.692725	3.114927	-0.132
4	-0.338257	-1.617506	-0.147502	-0.218673	-0.912857	0.692725	-0.321035	-0.132
...	...	...	...	...	...	...	...	
15055	-0.414411	1.128996	-0.147502	-0.218673	-0.078031	0.692725	-0.321035	-0.132
15056	0.042516	1.128996	-0.147502	-0.218673	-0.411961	-1.443574	-0.321035	-0.132
15057	-0.033639	1.128996	-0.147502	-0.218673	0.756794	0.692725	-0.321035	-0.132
15058	0.423288	1.128996	0.588766	-0.218673	-0.078031	0.692725	-0.321035	-0.132
15059	-0.262102	1.128996	-0.147502	-0.218673	1.591619	0.692725	-0.321035	-0.132

15060 rows × 34 columns

```python
# Réalisation d'une régression logistique pour classification
reglog = LogisticRegression()
reglog.fit(X_train_sc, y_train)
```

```
LogisticRegression()
```

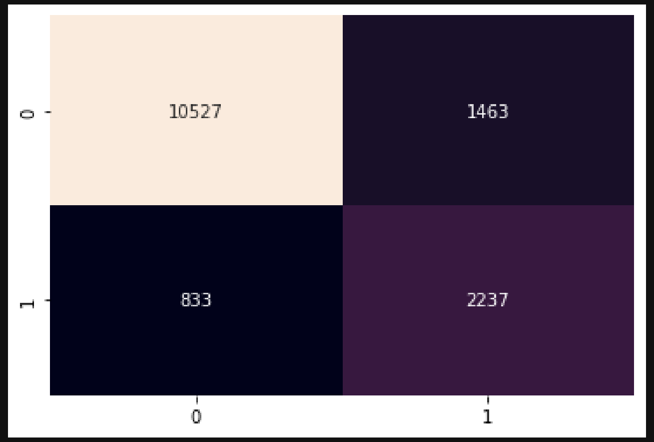
```python
# prédictions sur le test set
y_pred = reglog.predict(X_test_sc)
pd.Series(y_pred).value_counts()
```

```
<=50K     11990
>50K       3070
dtype: int64
```

```python
# Part de la variance expliquée par le modèle
print(reglog.score(X_train_sc,y_train), reglog.score(X_test_sc,y_test))
```

```
0.8466770629417034 0.847543160690571
```

```python
# Il y a 10527+2237 observations correctement expliquées par le modèle, le reste est mal prédit.
sns.heatmap(confusion_matrix(reglog.predict(X_test_sc),y_test), annot=True, fmt='d', cbar=False);
```




```python
from sklearn.metrics import confusion_matrix
pd.DataFrame(confusion_matrix(y_test, y_pred),
             columns=[f'predit {k}' for k in reglog.classes_],
             index = [f'vrai {k}' for k in reglog.classes_])
```

	predit <=50K	predit >50K
vrai <=50K	10527	833
vrai >50K	1463	2237

First we perform a logistic regression.
We visualize the predictions via heatmap.
We are refining with RFE.
With the remaining columns, we arrive at a score of 0.847 on the test.

```python
# On peut aussi utiliser RFE pour sélection les variables : disons qu'on veut en garder en 25
rfe = RFE(estimator=LogisticRegression(solver='lbfgs', max_iter=500), n_features_to_select=25)
rfe = rfe.fit(X_train_sc, y_train)
col_rfe = X_train_sc.columns[rfe.support_]
```

```python
#colonnes supprimées
X_train_sc.columns[~rfe.support_]
```

```
Int64Index([6, 8, 9, 12, 13, 24, 25, 28, 33], dtype='int64')
```

```python
#On crée un jeu de données RFE avec uniquement les variables sélectionnées via RFE
X_train_rfe = X_train_sc[col_rfe]
X_test_rfe = X_test_sc[col_rfe]
```

```python
reglog = LogisticRegression(solver='lbfgs', max_iter=500)
reglog.fit(X_train_rfe,y_train)
print(reglog.score(X_train_rfe,y_train), reglog.score(X_test_rfe,y_test))
```

```
0.8464779853346163 0.8476095617529881
```

Nous constations qu'il n'y a pas d'amélioration en utilisant la méthode RFE après la première régression logistique.

On tente une pénalisation avec penalty = "l1"

```
]: reglog2 = LogisticRegression(solver = 'saga', penalty='l1', max_iter = 5000)
```

```
]: reglog2.fit(X_train_sc, y_train)
```

```
]: LogisticRegression(max_iter=5000, penalty='l1', solver='saga')
```

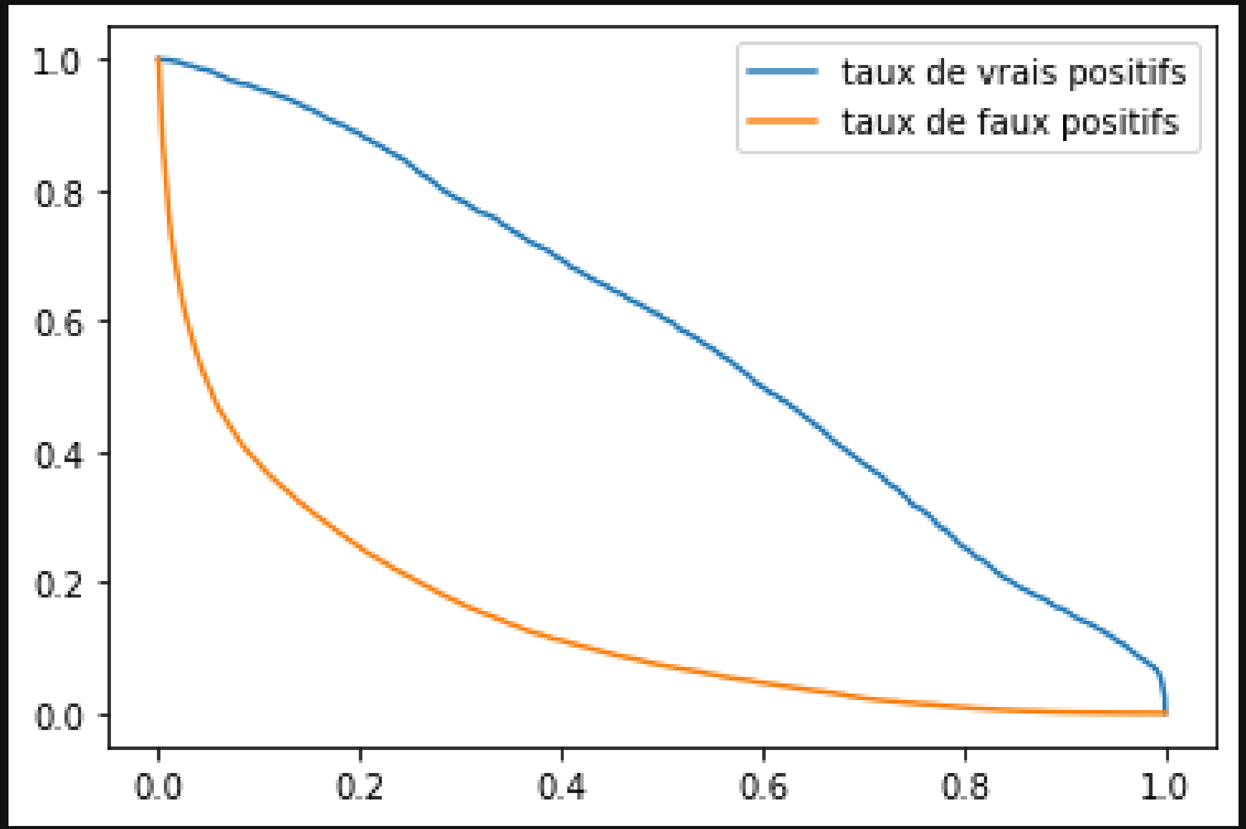
```
]: accuracy_score(reglog2.predict(X_test_sc),y_test)
```

```
]: 0.847675962815405
```

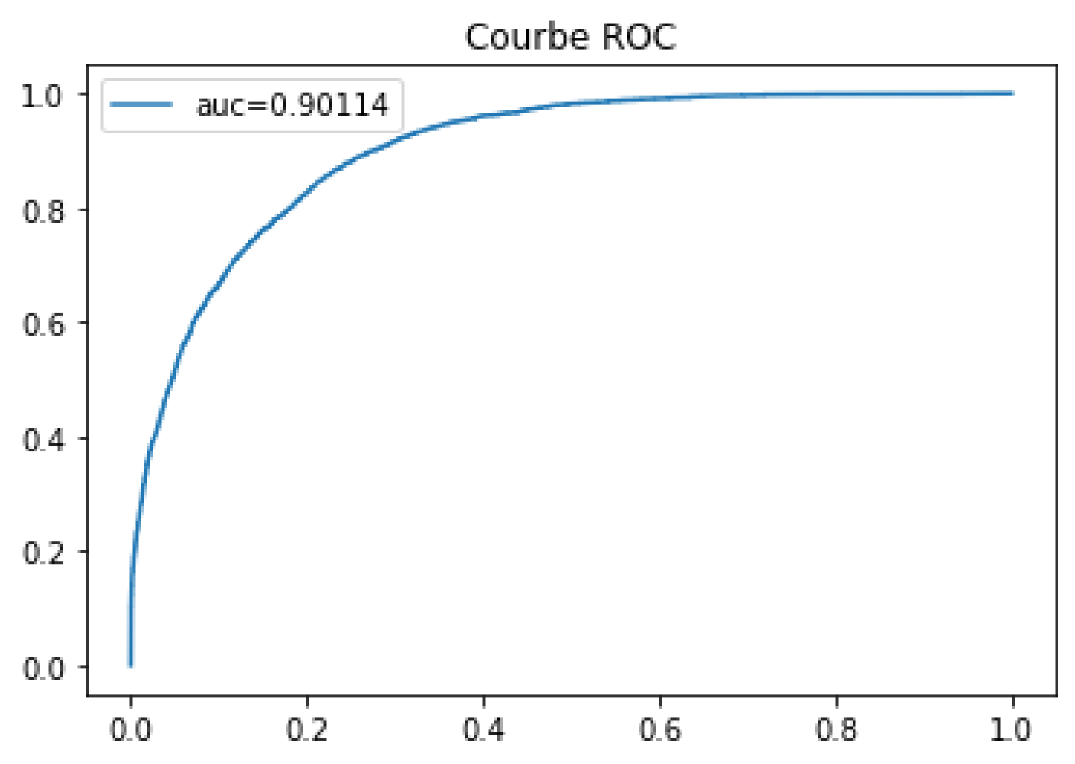
Le score est légèrement supérieur à celui de la régression logistique sans pénalité et avec le solver par défaut.

```python
proba1 = reglog.predict_proba(X_test_sc)[:,1]
fpr, tpr, seuils = roc_curve(y_test, proba1, pos_label='>50K', drop_intermediate=False)
```

```python
plt.plot(seuils[1:], tpr[1:], label='taux de vrais positifs')
plt.plot(seuils[1:], fpr[1:], label='taux de faux positifs')
plt.legend();
```




```python
score_auc = auc(fpr[1:], tpr[1:])
fig, ax = plt.subplots()
ax.plot(fpr[1:], tpr[1:], label='auc=%1.5f' %score_auc)
ax.set_title("Courbe ROC")
ax.legend();
```



Le résultat de l'AUC est de 0,90. C'est correct.

We then display the curve of true positives and false positives as well as the ROC curve which gives a result of 0.90.

```python
model = SVC()

param_grid = {
    'kernel' : ['linear', 'poly', 'rbf'],
    'gamma' : ['auto', 'scale', 0.1, 0.001, 0.0001]
}
grid = GridSearchCV(model, param_grid, n_jobs=-1)

%time grid.fit(X_train_sc, y_train)
print(grid.best_params_)
```

```
CPU times: user 24.2 s, sys: 744 ms, total: 24.9 s
Wall time: 9min 35s
{'gamma': 'auto', 'kernel': 'rbf'}
```

```python
param_grid = {
    'C' : [ 10**k for k in range(-3,3)],
    'kernel' : ['rbf'],
    'gamma' : ['auto']
}
grid = GridSearchCV(model, param_grid, n_jobs=-1)

%time grid.fit(X_train_sc, y_train)
print(grid.best_params_)
```

```
CPU times: user 23.5 s, sys: 371 ms, total: 23.9 s
Wall time: 5min 13s
{'C': 1, 'gamma': 'auto', 'kernel': 'rbf'}
```

We are now trying a more precisely SVC SVM model.

With GridSearch, we find first parameter gamma "auto" and kernel "rbf".

We try again with different values of "C" and we find 1.

```
param_grid = {
    'max_depth': [50, 100, 150],
    'n_estimators': [100, 500, 1000, 1500],
    'random_state' :[0]
}

grid_search_rfc = GridSearchCV(RandomForestClassifier(), param_grid = param_grid, cv = 3, n_jobs = -1, verbose = 2)
grid_search_rfc.fit(X_train_sc, y_train)
print(classification_report(y_test, grid_search_rfc.predict(X_test_sc), zero_division=0))
```

```
Fitting 3 folds for each of 12 candidates, totalling 36 fits
[CV] END .....max_depth=50, n_estimators=100, random_state=0; total time=   2.4s
[CV] END ....max_depth=50, n_estimators=1500, random_state=0; total time=  52.9s
/opt/homebrew/anaconda3/lib/python3.8/site-packages/joblib/externals/loky/process_executor.py:702: UserWarning: A wo
tor. This can be caused by a too short worker timeout or by a memory leak.
  warnings.warn(
[CV] END .....max_depth=50, n_estimators=100, random_state=0; total time=   2.5s
[CV] END ....max_depth=50, n_estimators=1500, random_state=0; total time=  53.0s
[CV] END .....max_depth=50, n_estimators=500, random_state=0; total time=  13.4s
[CV] END ....max_depth=50, n_estimators=1500, random_state=0; total time=  56.8s
[CV] END .....max_depth=50, n_estimators=500, random_state=0; total time=  13.6s
[CV] END ....max_depth=100, n_estimators=100, random_state=0; total time=   3.3s
[CV] END ....max_depth=100, n_estimators=100, random_state=0; total time=   3.2s
[CV] END ....max_depth=100, n_estimators=500, random_state=0; total time=  19.7s
[CV] END ...max_depth=100, n_estimators=1500, random_state=0; total time=  53.7s
[CV] END ....max_depth=50, n_estimators=1000, random_state=0; total time=  28.4s
[CV] END ....max_depth=100, n_estimators=500, random_state=0; total time=  21.3s
[CV] END ...max_depth=100, n_estimators=1500, random_state=0; total time=  52.4s
[CV] END ....max_depth=150, n_estimators=100, random_state=0; total time=   4.1s
[CV] END ....max_depth=150, n_estimators=100, random_state=0; total time=   3.6s
[CV] END ....max_depth=150, n_estimators=100, random_state=0; total time=   3.4s
[CV] END ....max_depth=150, n_estimators=500, random_state=0; total time=  16.5s
[CV] END ...max_depth=150, n_estimators=1500, random_state=0; total time=  38.2s
[CV] END ....max_depth=150, n_estimators=500, random_state=0; total time=  17.0s
[CV] END ...max_depth=150, n_estimators=1500, random_state=0; total time=  36.4s
              precision    recall  f1-score   support

       <=50K       0.88      0.92      0.90     11360
        >50K       0.70      0.62      0.66      3700

    accuracy                           0.84     15060
   macro avg       0.79      0.77      0.78     15060
weighted avg       0.84      0.84      0.84     15060
```

We test RandomForestClassifier with always different parameters.
Finally we find an accuracy of 0.84
with
max_depth = 50 and n_estimators = 1500.

```
[88]: print(f"Meileurs paramètres sur le jeu d'entraînement {grid_search_rfc.best_params_}")

print("Accuracy de chacun des modèles :")
for res, params in zip(grid_search_rfc.cv_results_['params'], grid_search_rfc.cv_results_['mean_test_score']):
    print(f"pour les paramètres {res}, la précision du modèle est {params}")
```

```
Meileurs paramètres sur le jeu d'entraînement {'max_depth': 50, 'n_estimators': 1500, 'random_state': 0}
Accuracy de chacun des modèles :
pour les paramètres {'max_depth': 50, 'n_estimators': 100, 'random_state': 0}, la précision du modèle est 0.84312695425402
pour les paramètres {'max_depth': 50, 'n_estimators': 500, 'random_state': 0}, la précision du modèle est 0.8430937372238857
pour les paramètres {'max_depth': 50, 'n_estimators': 1000, 'random_state': 0}, la précision du modèle est 0.8436246317600924
pour les paramètres {'max_depth': 50, 'n_estimators': 1500, 'random_state': 0}, la précision du modèle est 0.8437573347532177
pour les paramètres {'max_depth': 100, 'n_estimators': 100, 'random_state': 0}, la précision du modèle est 0.8428283378427318
pour les paramètres {'max_depth': 100, 'n_estimators': 500, 'random_state': 0}, la précision du modèle est 0.8430605664294268
pour les paramètres {'max_depth': 100, 'n_estimators': 1000, 'random_state': 0}, la précision du modèle est 0.843558270355885
pour les paramètres {'max_depth': 100, 'n_estimators': 1500, 'random_state': 0}, la précision du modèle est 0.8435582505405957
```

```
[89]: param_grid = {
          'max_depth': [5, 20, 50],
          'n_estimators': [ 1000, 1500, 2000, 2500],
          'random_state' :[0]
      }

      grid_search_rfc = GridSearchCV(RandomForestClassifier(), param_grid = param_grid, cv = 3, n_jobs = -1, verbose = 2)
      grid_search_rfc.fit(X_train_sc, y_train)
      print(classification_report(y_test, grid_search_rfc.predict(X_test_sc), zero_division=0))
```

	precision	recall	f1-score	support
<=50K	0.88	0.94	0.91	11360
>50K	0.77	0.61	0.68	3700
accuracy			0.86	15060
macro avg	0.83	0.78	0.80	15060
weighted avg	0.86	0.86	0.85	15060

We are fine-tuning the various parameters again.
Our accuracy increases to 0.86
with
max_depth = 20 and n_estimator = 2500.

```
[93]: print(f"Meilleurs paramètres sur le jeu d'entraînement {grid_search_rfc.best_params_}")

      print("Accuracy de chacun des modèles :")
      for res, params in zip(grid_search_rfc.cv_results_['params'], grid_search_rfc.cv_results_['mean_test_score']):
          print(f"pour les paramètres {res}, la précision du modèle est {params}")

      Meilleurs paramètres sur le jeu d'entraînement {'max_depth': 20, 'n_estimators': 2500, 'random_state': 0}
      Accuracy de chacun des modèles :
      pour les paramètres {'max_depth': 5, 'n_estimators': 1000, 'random_state': 0}, la précision du modèle est 0.8442881797512009
      pour les paramètres {'max_depth': 5, 'n_estimators': 1500, 'random_state': 0}, la précision du modèle est 0.844487237543437
```

Finally we end with a cross validation.
The best model is therefore
the RandomForestClassifier
with a score of 0.8587.

Validation croisée

```
models = {
    'rfc': RandomForestClassifier(max_depth= 20, n_estimators= 2500),
    'reglog2': LogisticRegression(solver = 'saga', penalty='l1',max_iter=5000),
    'svc': SVC(gamma='auto', C=1, kernel='rbf')
}

for k,v in models.items():
    model = v
    scores = cross_val_score(model, X_train_sc, y_train, cv=5)
    print(f'scores de validation croisée du modèle {k} : {scores} / score moyen : {np.mean(scores)}')
```

```
scores de validation croisée du modèle rfc : [0.85434638 0.85749834 0.86015262 0.86214333 0.85963166] / score moyen : 0.8587544668764944
scores de validation croisée du modèle reglog2 : [0.84207034 0.83825481 0.84771068 0.85102853 0.85150158] / score moyen : 0.8461131885061792
scores de validation croisée du modèle svc : [0.8450564  0.84389516 0.84654944 0.84887193 0.84685582] / score moyen : 0.8462457483681319
```

Our difficulties:

- Data cleaning.
- Choice of parameters, we had to select values because the models took too long to run.
- Visualization of the ROC curve.

Finally we studied three different models.

However, we are satisfied with the work done. Thank you :-)