Project: Date-A-Scientist

Machine Learning Fundamentals

Jessy Metaxas

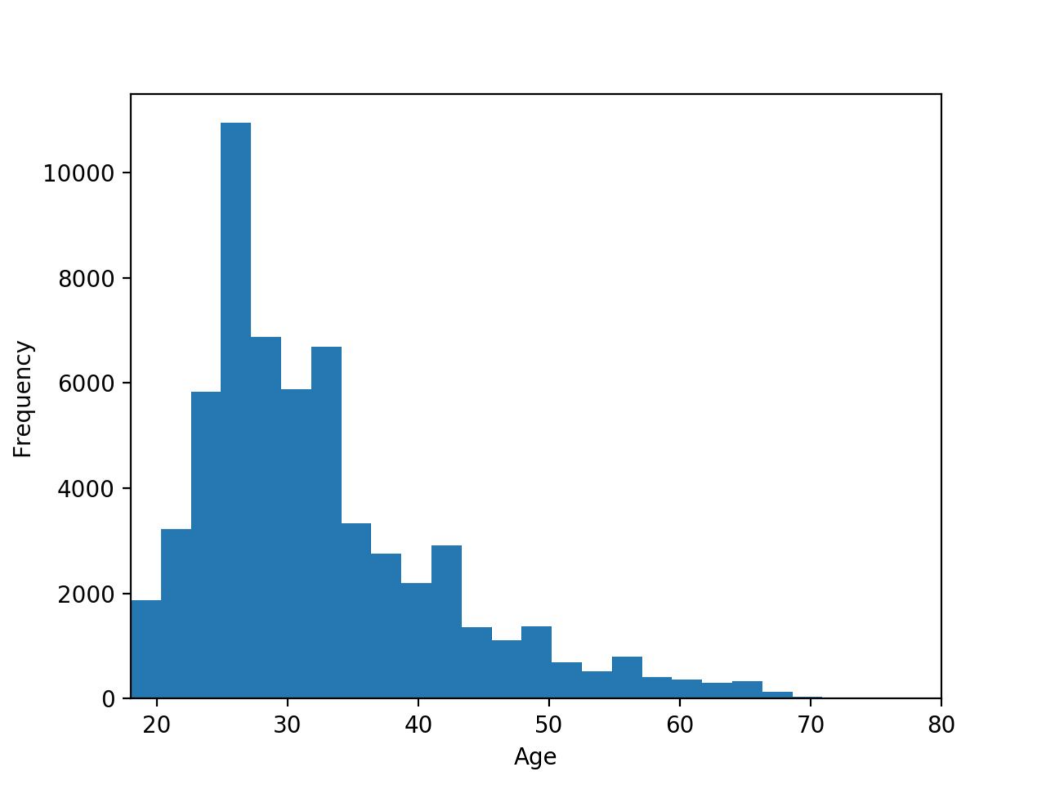
11/10/2018

Table of Contents

Exploration of the dataset

The *profile.csv* file is a tabular dataset containing 59,946 records and 31 columns (with most of them being composed of text data and only a few of them being composed of numerical data such as **age**, **height** and **income**).

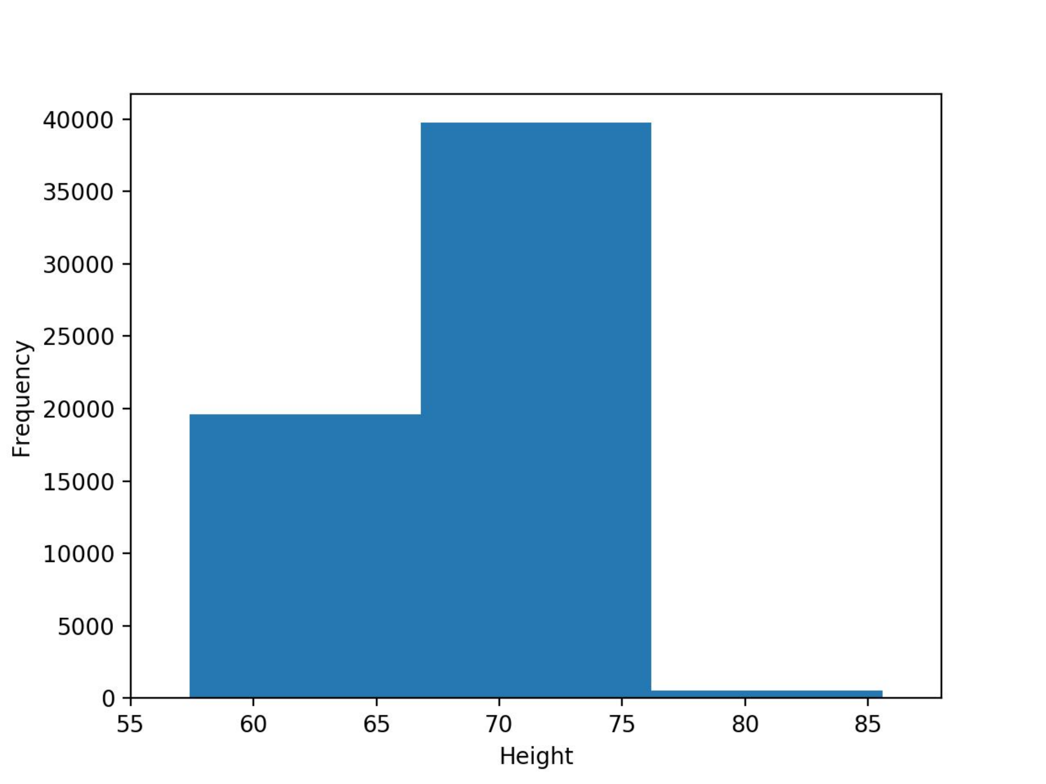
Distribution of ages



This histogram shows that the majority of the users are rather young. The primary age group of users seems to be between 18 and 30.

```
plt.hist(df.age, bins=40)
plt.xlabel("Age")
plt.ylabel("Frequency")
plt.xlim(18, 80)
plt.show()
```

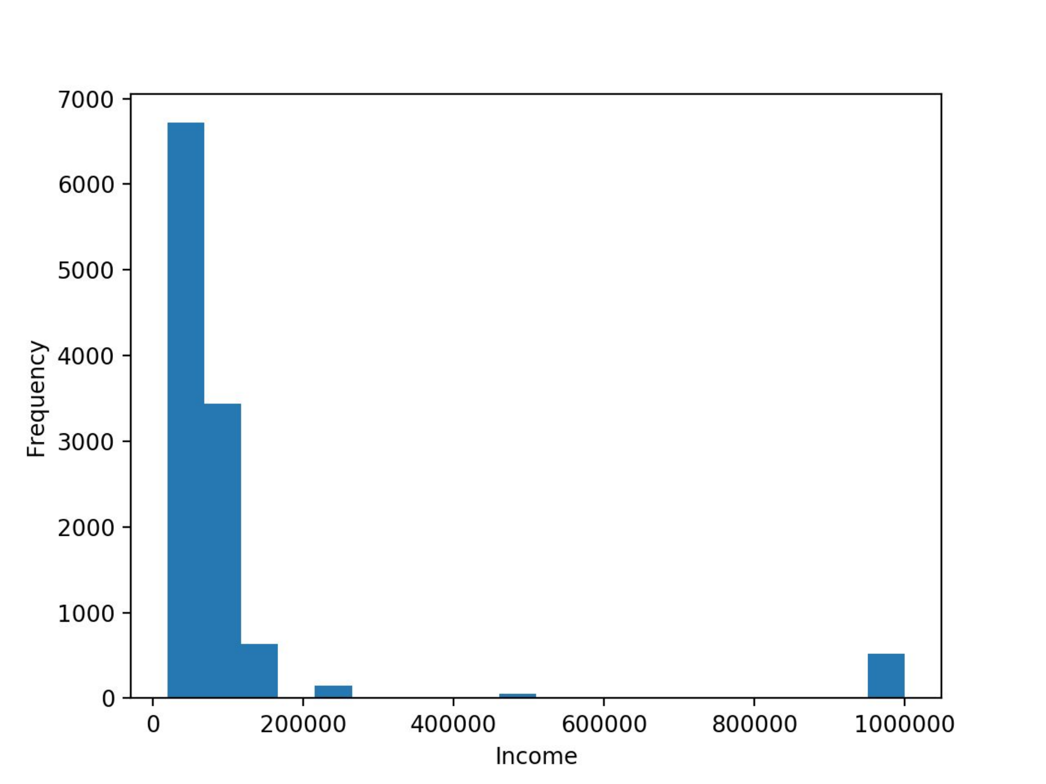
Distribution of heights



The distribution of heights shows that the users seems to be split into two main different groups (perhaps this might reflect the distribution of men and women among users).

```
df = df.dropna(subset=["height"])
plt.hist(df.height)
plt.xlabel("Height")
plt.ylabel("Frequency")
plt.xlim(55, 88)
plt.show()
```

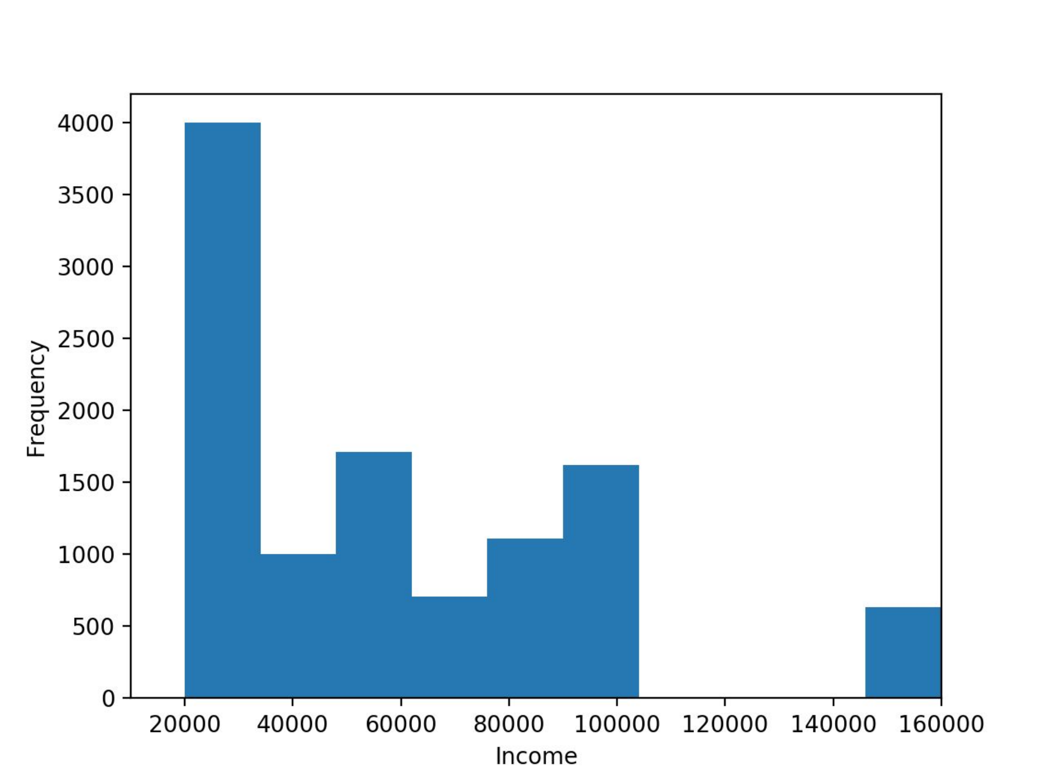
Distribution of incomes



The distribution of incomes shows that the vast majority of users seems to be earning less than 100,000$. However, a small group of users reported revenues >= 1,000,000$.

```
df = df.dropna(subset=["height"])
plt.hist(df.height)
plt.xlabel("Height")
plt.ylabel("Frequency")
plt.xlim(55, 88)
plt.show()
```

Distribution of incomes (between 0 and 160,000$)

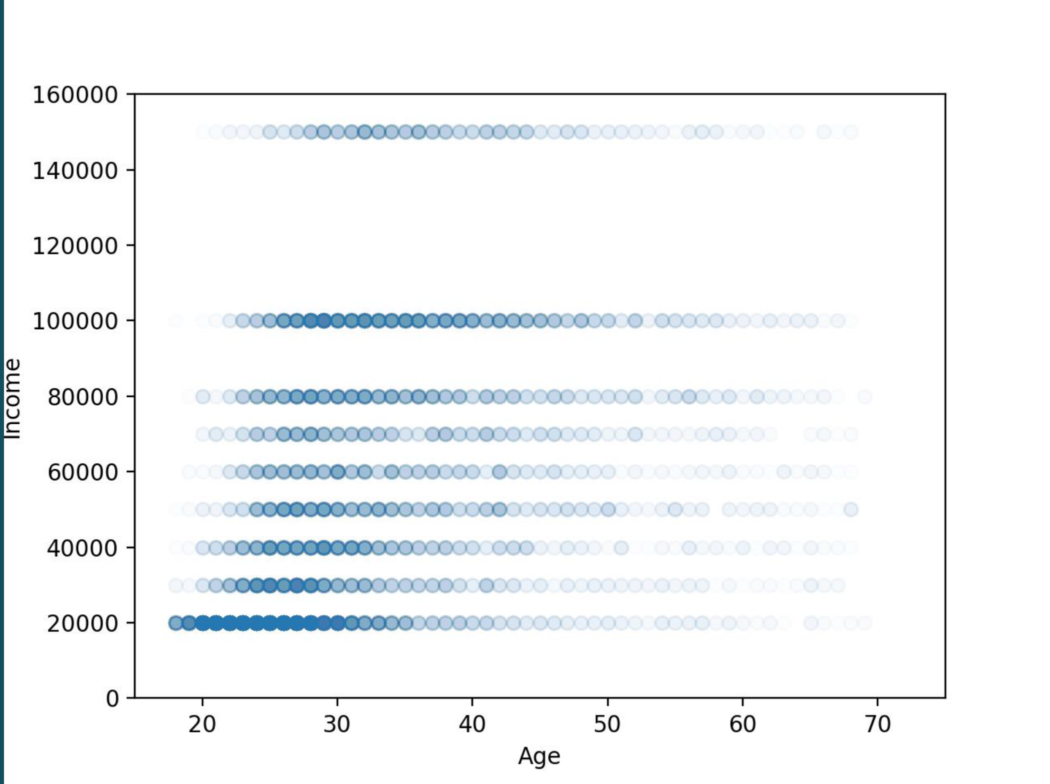


If we only focus on what seems to be the majority income class, 3 groups seems to be emerging: The users earning less than 30,000$, those earning more than 80,000$ and those earning between 30,000 and 80,000$.

```
df['income'] = df['income'].replace(-1, np.nan, regex=True)
df = df.dropna(subset=["income"])

plt.hist(df.income, bins=70)
plt.xlabel("Income")
plt.ylabel("Frequency")
plt.xlim(10000, 160000)
plt.show()
```

Relationship between age and income



If we plot ***age*** against ***income*** (by focusing on the values between 0 and 160,000$), we can see a certain trend that might show some relationship between those 2 features.

```
df['income'] = df['income'].replace(-1,
np.nan, regex=True)
df = df.dropna(subset=["income"])

plt.scatter(df.age, df.income,
alpha=0.01)
plt.xlabel("Age")
plt.ylabel("Income")
plt.axis([15, 75, 0, 160000])
plt.show()
```

Questions to answer

Following this first exploration of the dataset, there are some questions we might try to answer:

- Is it possible to guess the **age** of the users?
- Can we predict their **income** level?

After augmenting the dataset, we'll see that it might also be possible to try to answer different kind of questions:

- Is it possible to guess whether someone is a **male** or a **female**?
- Can we guess whether someone has a **"fit" body type**?
- Whether a user has **graduated**?

Augmenting the dataset

Since most of the features of the dataset is composed of multiple-choice text data (sex, education, job, etc.), it seemed relevant to convert them into numerical data in order to be able to **exploit as much information as possible**.

Thus, the column <u>*sex*</u> which is composed of the values **m** and **f**, became ***sex_code*** with, respectively, the values 0 and 1.

```
df["sex_code"] = df.sex.map({
    "m": 0, "f": 1
})
```

The column *education*, which is composed of various text data such as "*graduated from college/university*", "*graduated from ph.d program*" or "*working on masters program*", was converted into 2 different features :

- ***has_graduated*** (the value of **1** being assigned if the word "**graduated**" explicitly appears in the answer)
- ***is_studying*** (the value of **1** being assigned if the word "**working**" explicitly appears in the answer)

```
df['has_graduated'] = df.apply(lambda row: 1 if (
       "graduated" in row.education
) else 0, axis=1)
```

A third feature from the *education* column was also generated : ***has_high_academic_degree*** (with the value of **1** being assigned when someone explicitly reports having graduated from either a master program, ph.d program, law school, med school or space camp).

```
df['has_high_academic_degree'] = df.apply(lambda row: 1 if (
        row.education == 'graduated from masters program' or
        row.education == 'graduated from ph.d program' or
        row.education == 'graduated from law school' or
        row.education == 'graduated from med school' or
        row.education == 'graduated from space camp'
) else 0, axis=1)
```

3 features were created from the *income* column:

- ***high_income*** (the value of **1** being assigned when income >= 80,000)
- ***middle_income*** (same as above when income between 30,001 and 79, 999)
- ***low_income*** (same as above when income <= 30,000)

```
df["low_income"] = df.apply(lambda row: 1 if (
       0 <= row['income'] <= 30000
) else 0, axis=1)
```

The ***age_code*** feature, representing two age groups (**0** for “*under 30*” and **1** for “*over 30*”), was created from the *age* column.

7 features were created from the *job* column :

- ***stem_career*** (the value of 1 being assigned when someone explicitly reports working in the “science / tech / engineering” or “computer / hardware / software” field)
- ***health_career*** (same as above if field is “medicine / health”)
- ***law_career*** (same as above if field is “law / legal services”)
- ***artistic_career*** (same as above if field is “artistic / musical / writer”)
- ***education_career*** (same as above if field is “education / academia”)
- ***business_career*** (same as above if field is “sales / marketing / biz dev”)
- ***financial_career*** (same as above if field is “banking / financial / real estate”)

The column _orientation_ was split into 2 different features:

- ***is_straight*** (the value of **1** being assigned if the answer provided for *orientation* is “straight”)
- ***is_gay_bi*** (same as above if the answer is either “gay” or “bisexual”)

The column *body_type* (which is composed of multiple-choice text data allowing a user to pick what describes best his/her body type) was split into 4 different features:

- ***has_chubby_body_type*** (the value of **1** being assigned if the answer provided for *body_type* is either “overweight”, “full figured”, “curvy” or “a little extra”)
- ***has_fit_body_type*** (same as above if the answer is either “fit”, “athletic”, or “jacked”)
- ***has_thin_body_type*** (same as above if the answer is either “skinny”, “thin”, or “used up”)
- ***has_average_body_type*** (same as above if the answer is “average”)

The same process was used to generate 5 features from the *diet* column :

- ***eats_anything***
- ***eats_vegetarian***
- ***eats_vegan***
- ***eats_kosher***
- ***eats_halal***

With, for example, the value of **1** being assigned to ***eats_vegetarian*** if the user explicitly reports eating “mostly vegetarian”, “vegetarian”, “strictly vegetarian”.

8 features were generated from the *ethnicity* column :

- ***is_white***
- ***is_asian***
- ***is_latin***
- ***is_black***
- ***Is_islander***
- ***Is_native***
- ***Is_middle_eastern***
- ***is_indian***

With, for example, the value of **1** being assigned to ***is_white*** whenever the word "white" appears in the answer. Since someone could be of "mixed race" heritage, a given user - white and asian, for example - could have the value of **1** assigned to both ***is_white*** and ***is_asian***.

Following the same process, 8 features were generated from the *religion* column:

- ***is_agnostic***
- ***is_catholic***
- ***is_atheist***
- ***is_non_catholic_christian***
- ***is_jewish***
- ***is_buddhist***
- ***is_hindu***
- ***is_muslim***

With, for example, the value of **1** being assigned to ***is_catholic*** whenever the word “catholicism” appeared in the answer.

The column *offspring* (which is composed of multiple-choice text data allowing users to indicate whether or not they have children and whether or not they want more) was split into 4 different features:

- ***has_kids***
- ***has_no_kids***
- ***wants_kids***
- ***doesnt_want_kids***

With, for example, the value of **1** being assigned to ***has_kids*** whenever the words “has kids” or “has a kid” explicitly appear in the answer. And the value of **1** being assigned to ***doesnt_want_kids*** whenever the words “doesn’t want kids”, “doesn’t want any”, “but doesn’t want more” appear in the answer.

Someone indicating that they “have kids, but doesn’t want more” would have the value of **1** assigned to both ***has_kids*** and ***doesnt_want_kids***.

Why having two separated features such as ***has_kids / has_no_kids*** or ***wants_kids / doesnt_want_kids***, instead of a single one for each of them (like ***has_kids_code***, where **1** would mean “*Current user has one or more kids*” and where **0** would mean “*Current user doesn’t have kids*”)?

- Combining the features ***has_kids*** and ***has_no_kids*** into a single feature named ***has_kids_code*** could only make sense when dropping from the dataset every rows having a “NaN” value in the offspring column (35,561 rows). Keeping the “NaN” values and filling them with 0 (in order to keep as much as rows as possible) wouldn’t be relevant, since a “NaN” value gives no indication whether someone really has or doesn’t have kids.

- By keeping the features separated, a **1** value for ***has_kids*** can be understood as “*has explicitly reported having kids*” and a **0** value as “*has not explicitly reported having kids*”.

Following the same logic, 6 more features were created from the *pets* column:

- ***has_cats***
- ***has_dogs***
- ***likes_cats***
- ***likes_dogs***
- ***dislikes_cats***
- ***dislikes_dogs***

With, for example, the value of **1** being assigned to ***has_cats*** whenever the word “has cats” explicitly appears in the answer.

3 new features were created from the merging of the 10 essays columns (essay0, essay1, etc.):

- ***essay_len*** (total character length of the 10 essays)
- ***essay_count_words*** (total number of words in the 10 essays)
- ***essay_words_mean_length*** (average word length of the 10 essays)

Following the example given in the project instruction, each of the *drink*, *drugs* and *smokes* columns were converted into 3 differents features:

- ***drinks_code***
- ***drugs_code***
- ***smokes_code***

```
df["smokes_code"] = df.smokes.map({
    "no": 0,
    "trying to quit": 1,
    "when drinking": 2,
    "sometimes": 3,
    "yes": 4
})
```

Regression approaches

1) Can we guess the age of the users?

To identify, the features with the best correlations, we first remove any rows where *income* = -1 (thus reducing the dataset to 11,504 rows), then we use the `.corrwith()` method on the ***age*** column.

```
print(df.corrwith(df['age']))
```

The following features are those that have the most positive and negative linear relationship with the ***age*** feature :

➔ high_income, middle_income, low_income
➔ has_high_academic_degree, has_graduated, is_studying
➔ drinks_code, drugs_code, smokes_code
➔ has_kids, has_no_kids, wants_kids, doesnt_want_kids

1) Can we guess the age of the users?

➔ **Multiple Linear Regression**

R^2	**0.3303186518537492**
Execution time	0.25 seconds

➔ **K Nearest Neighbors Regression (k = 41)**

R^2	**0.326663**
Execution time	0.70 seconds

1) Can we guess the age of the users?

In this context, the **Multi Linear Regression** model return the best coefficient of determination R^2 of the prediction (although the difference with the KNN Regression model is not very significant).

The **MLR** model is also the fastest of the two models and the most simple to implement (since in the case of the KNN Regression model, we first had to compute the best K in order to find the best R^2).

However, with a R^2 of 0.33, it's not possible to consider that the model is good enough to guess accurately the age of the users based on a few key features.

2) Can we predict the income level of the users?

To identify, the features with the best correlations, we first remove any rows where *income* = -1, then we use the `.corrwith()` method on the ***income*** column.

```
print(df.corrwith(df['income']))
```

However, not a single feature seems to share a strong positive/negative relationship with the ***income*** column.

We decide anyway to use them all to train our model.

2) Can we predict the income level of the users?

➔ **Multiple Linear Regression**

R^2	**0.03556759534188347**
Execution time	0.28 seconds

➔ **K Nearest Neighbors Regression (k = 26)**

R^2	**0.041667**
Execution time	3.12 seconds

2) Can we predict the income level of the users?

In this context, both the **MLR** and **KNN Regression** models completely fail to predict accurately the income level of the users.

Instead of trying to predict the income level by using regression approaches, could it be possible to get better result by trying to guess a *user's income class* with classification models?

Classification approaches

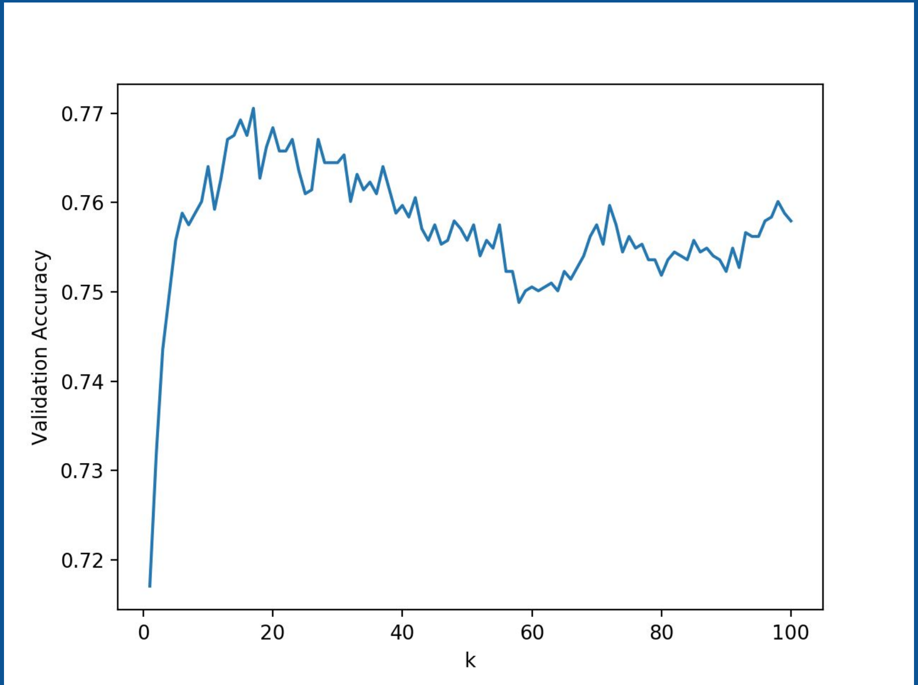
1) Can we guess if a user is a low income earner?

The regression approaches failed to effectively predict the level income of the users. Thus, instead of trying to directly guess the income level, we'll see if - when using classification models - we get better results by trying to guess whether a user is a **low income** earner (<= 30,000$ / year).

```
guess = ['low_income']
features = [
    'age', 'sex_code', 'height',
    'is_straight', 'is_gay_bi',
    'has_high_academic_degree', 'has_graduated', 'is_studying',
    'has_chubby_body_type', 'has_fit_body_type',
    'stem_career', 'education_career', 'financial_career',
    'drinks_code', 'drugs_code', 'smokes_code',
]
```

1) Can we guess if a user is a low income earner?

➔ K Nearest Neighbors



Best k = 17

1) Can we guess if a user is a low income earner?

➔ **K Nearest Neighbors (k = 17)**

Accuracy	**0.770535**
Execution time	0.87 seconds

class	precision	recall	f1-score
1	0.69	0.62	**0.65**

With class 1 = "is a low income earner"

1) Can we guess if a user is a low income earner?

➔ **Support Vector Machine (kernel=RBF, c=4, gamma=8)**

Accuracy	**0.7566275532377227**
Execution time	6.11 seconds

class	precision	recall	f1-score
1	0.67	0.59	**0.63**

1) Can we guess if a user is a low income earner?

➔ **Naive Bayes**

Accuracy	**0.7548891786179922**
Execution time	0.22 seconds

class	precision	recall	f1-score
1	0.67	0.58	**0.62**

1) Can we guess if a user is a low income earner?

Classifier	Class	Accuracy	F1-score	Execution Time
KNN	1	**0.770535**	**0.65**	0.87s
SVM	1	0.756627	0.63	6.11s
NB	1	0.754889	0.62	**0.22s**

- In this context, the **K Nearest Neighbors** classifier return the best accuracy and F1-score.
- The Support Vector Machine classifier is the slowest of the 3 models, while being less accurate than the KNN classifier (7x slower than the KNN model).
- The Naive Bayes classifier is the fastest of the 3 models, but it also returns the lowest F1 score.

1) Can we guess if a user is a low income earner?

Although being far from perfect, with an accuracy of **0.77** and a F1-score of **0.65** (precision:0.69, recall: 0.62), the **KNN** classifier seems to be quite efficient at predicting whether a user is a *low income* earner.

2) Can we guess if a user is under or over 30?

The regression approaches weren't very effective at predicting the age of the users. We'll see if - when using classification models - we achieve better results by trying to guess whether a user is **under 30** (0) or **over 30** (1).

```
conditions = [
    (df['age'] >= 0) & (df['age'] <= 30),
    (df['age'] > 30)
]
choices = [0, 1]
df["age_code"] = np.select(conditions, choices)
```

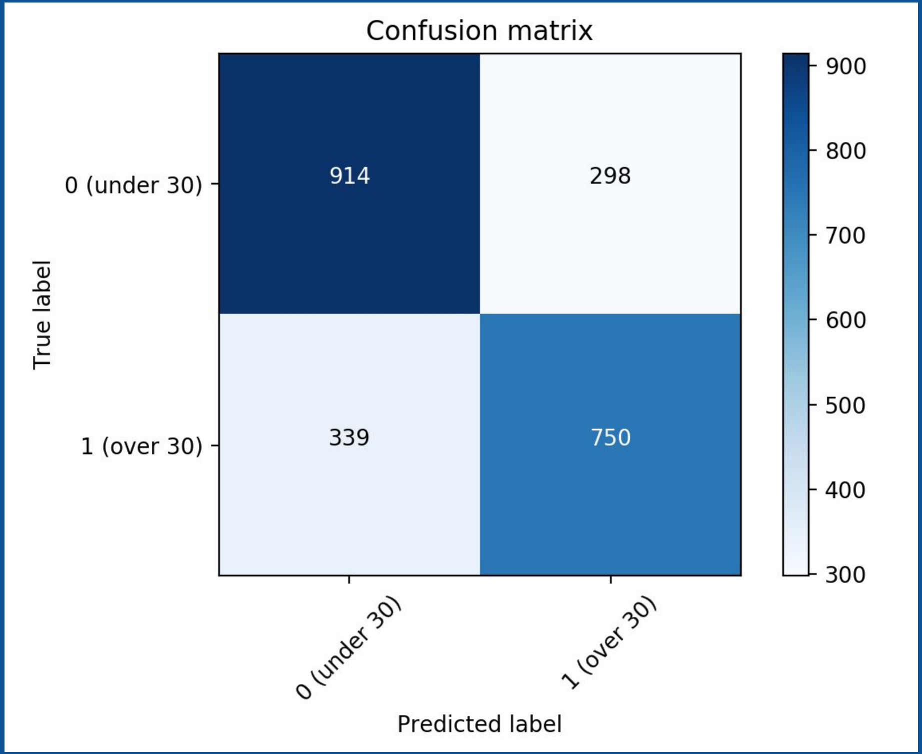
2) Can we guess if a user is under or over 30?

```
guess = ['age_code']
features = [
    'income', 'high_income', 'middle_income', 'low_income',
    'has_high_academic_degree', 'has_graduated', 'is_studying',
    'is_agnostic', 'is_catholic', 'is_atheist', 'is_non_catholic_christian',
    'is_jewish', 'is_buddhist', 'is_hindu', 'is_muslim',
    'has_fit_body_type', 'has_chubby_body_type', 'has_thin_body_type',
'has_average_body_type',
    'eats_anything', 'eats_vegetarian', 'eats_vegan',
    'drinks_code', 'drugs_code', 'smokes_code',
    'has_kids', 'has_no_kids', 'wants_kids', 'doesnt_want_kids',
    'has_cats', 'has_dogs',
    'stem_career', 'health_career', 'law_career', 'artistic_career',
    'education_career', 'business_career', 'financial_career',
    'essay_len', 'essay_count_words', 'essay_words_mean_length',
]
```

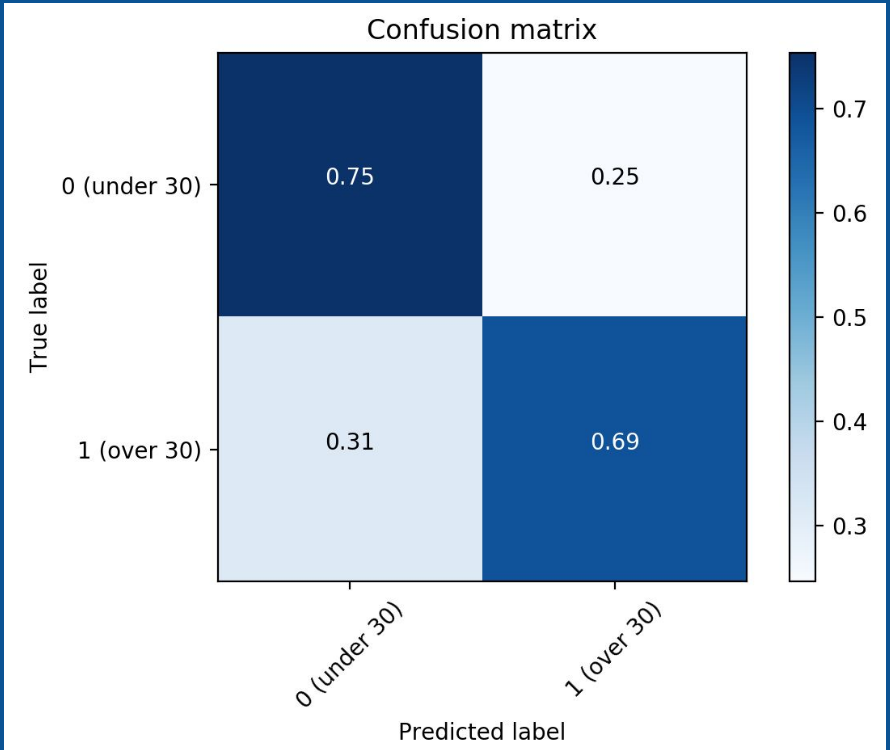
2) Can we guess if a user is under or over 30?

Classifier	Class	Accuracy	F1-score	Execution Time
KNN (k = 44)	0	0.709691	0.73	3.99s
	1		0.69	
SVM (linear) c=4, g=8	0	**0.723163**	**0.74**	11.68s
	1		**0.7**	
NB	0	0.701434	0.7	**0.21s**
	1		0.7	

2) Can we guess if a user is under or over 30?



Confusion matrix, without normalization



Normalized confusion matrix

2) Can we guess if a user is under or over 30?

With a global accuracy of **0.72** and a F1-score of **0.74** and **0.7** for respectively the 0 (*under 30*) and 1 (*over 30*) groups, the **SVM** classifier seems to be effective enough to predict a user's age group.

However, it should be noted that, when working on the full dataset (i.e. by keeping any rows where *income* = -1), the accuracy falls to 0.66, and the F1-score falls to 0.73 (*under 30*) and 0.53 (*over 30*) (with a far worse execution time of 383.44 seconds). It might indicate that the prediction of the *over 30* group relies much more on the income data.

3) Can we guess whether someone is a male or a female?

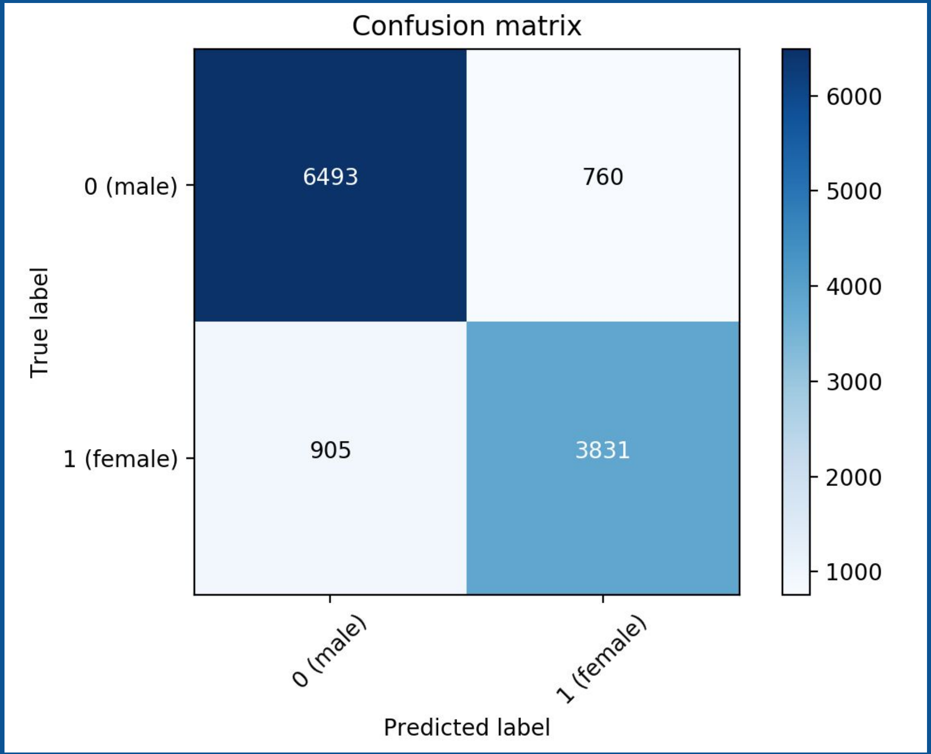
```
guess = ['sex_code']
features = [
    'height',
    'has_fit_body_type', 'has_chubby_body_type', 'has_thin_body_type',
'has_average_body_type',
    'eats_anything', 'eats_vegetarian', 'eats_vegan',
    'stem_career', 'health_career', 'education_career',
]
```

- *working on the full dataset*
- sex_code = 0 (**male**) or 1 (**female**)

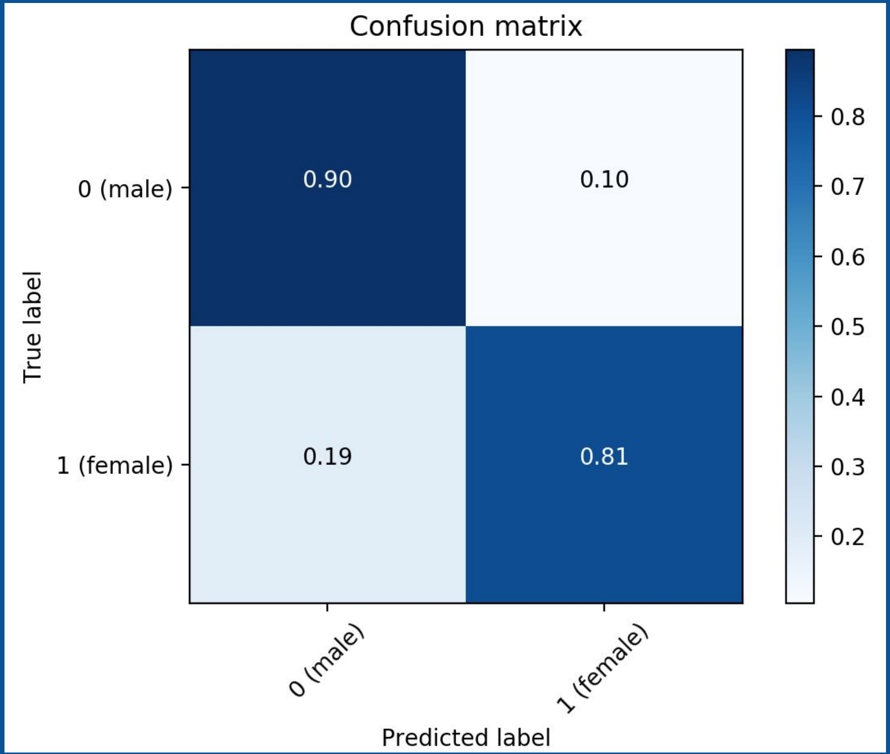
3) Can we guess whether someone is a male or a female?

Classifier	Class	Accuracy	F1-score	Execution Time
KNN (k = 28)	0	0.857703	0.88	5.77s
	1		**0.82**	
SVM (linear) c=4, g=8	0	**0.861122**	**0.89**	39.05s
	1		**0.82**	
NB	0	0.686379	0.77	**0.83s**
	1		0.48	

3) Can we guess whether someone is a male or a female?



Confusion matrix, without normalization



Normalized confusion matrix

3) Can we guess whether someone is a male or a female?

With a global accuracy of **0.86** and a F1-score of **0.89** and **0.82** for respectively the 0 (*male*) and 1 (*female*) groups, the **SVM** classifier seems to be really effective at predicting whether a user is a male or a female. It is however 7x slower than the KNN classifier which offers almost identical results.

4) Can we guess whether a user has a “fit” body type?

```
guess = ['has_fit_body_type']
features = [
    'sex_code', 'age', 'height',
    'is_straight', 'is_gay_bi',
    'eats_anything', 'eats_vegetarian', 'eats_vegan', 'eats_kosher', 'eats_halal',
    'drinks_code', 'drugs_code', 'smokes_code',
    'high_income', 'low_income',
    'has_high_academic_degree', 'has_graduated', 'is_studying',
]
```

With **has_fit_body_type** = 1 when a user reports having a *fit*, *muscular* or *jacked* body type.

4) Can we guess whether a user has a "fit" body type?

Classifier	Class	Accuracy	F1-score	Execution Time
KNN (k=48)	1	**0.651456**	**0.55**	1.17s
SVM (linear, c=4, g=8)	1	0.623641	0.53	9.14s
NB	1	0.641894	0.55	**0.16s**

Despite returning a relatively good accuracy rate of **0.65**, the F1-score (**0.55**) is not high enough to consider the KNN classifier sufficiently reliable to guess whether someone has a "fit" body type.

5) Can we guess whether a user has graduated?

```
guess = ['has_graduated']
features = [
    'age',
    'income',
]
```

With **has_graduated** = 1 when a user reports having graduated.

5) Can we guess whether a user has graduated?

Classifier	Class	Accuracy	F1-score	Execution Time
KNN (k=22)	1	**0.701434**	0.78	0.4s
SVM (rbf, c=4, g=8)	1	0.694915	**0.79**	5.06s
NB	1	0.625814	0.77	**0.22s**

With an accuracy of **0.7** and a F1-score of **0.78**, the **KNN** classifier seems to be effective at predicting whether a user has graduated (with just the *age* and *income* features). The SVM provides similar results with a far slower execution time (and thus a greater difficulty to find the best C and gamma values).

Conclusion

Can we guess...	Best Model	Accuracy	F1-Score	Result
the age of the users?	MLR	0.33	-	No
the income level of the users?	KNN R	0.04	-	No, not at all
if a user is a low income earner?	KNN	0.77	0.65	Pretty much yes
if a user is under or over 30?	SVM	0.72	0.74, 0.7	Yes
whether a user is a male or a female?	SVM	0.86	0.89, 0.82	Yes
whether a user has a “fit” body type?	KNN	0.65	0.55	Not really
whether a user has graduated?	KNN	0.70	0.78	Yes

- ➔ Given the current structure of the dataset, it wasn’t possible to use **regression** approaches to effectively predict the *age* or *income level* of the users.

- ➔ However, the categorization of these numerical data into different groups/classes has made it possible to use several types of **classification** approaches with much better results (although still being perfectible).

- ➔ It was only possible to work and use **income** data effectively by reducing the dataset to **11,504** records (out of 59,946).

- ➔ In order to be able to use the full dataset - and since the income feature proved to be important to answer accurately to different kind of questions (age class, graduation) - perhaps it would have been necessary to require users to provide this information (*however, not really conceivable for a dating site*).

- ➔ Guessing whether a user has a **fit / muscular body type** was the less performing question (considering only the classification approaches).

- ➔ Asking, for example, *“How many times a week do you workout?”* would probably have greatly helped to increase the accuracy and F1-score for this *body type* question.

- Since these data are coming from a dating website, perhaps a next step could be to get and focus on the total **number of interactions** one user has with the others.

- The goal would be then to try to find a model, based on some key features, that would predict how “popular” a user might be.

The End