Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ:

Прогноз уровня самоубийств на основе деревьев решений и случайных лесов

Студент ИУ5-32М (Группа) Ван Пэй (Подпись, дата) 20.12.2021 Ван Пэй (И.О.Фамилия)

Руководитель _____ (Подпись, дата) _____ (И.О.Фамилия)

Консультант _____ (Подпись, дата) _____ (И.О.Фамилия)

2021 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ
Заведующий кафедрой __ИУ5_____
(Индекс)
_____ _____
(И.О.Фамилия)
« 12 » 20 2020 г.

З А Д А Н И Е
на выполнение научно-исследовательской работы

по теме Прогноз уровня самоубийств на основе деревьев решений и случайных лесов
Студент группы ИУ5-32М
Ван Пэй
(Фамилия, имя, отчество)

Направленность НИР (учебная, исследовательская, практическая, производственная, др.)
Учебная, практическая
Источник тематики (кафедра, предприятие, НИР) Кафедра

График выполнения НИР: 25% к ___ нед., 50% к ___ нед., 75% к __ нед., 100% к ___ нед.

Техническое задание
отслеживание объектов на основе питона
Оформление научно-исследовательской работы:

Расчетно-пояснительная записка на __35_ листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
_____
_____
_____
_____

Дата выдачи задания « __20_ » _____12_____ 20_21_ г.

Руководитель НИР _____ _____
(Подпись, дата) (И.О.Фамилия)

Студент Ван Пэй 20.12.2021 Ван Пэй
(Подпись, дата) (И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

Аннотация

Самоубийство является одной из основных глобальных проблем здравоохранения. Согласно данным Всемирной организации здравоохранения (WTO) о самоубийствах, более 700 000 человек ежегодно умирают от самоубийства.

В этом проекте я буду использовать алгоритмы машинного обучения (дерево решений и случайный лес) для прогнозирования риска самоубийств. Используйте набор данных на Kaggle, который содержит информацию из 101 страны, такую как возрастная группа, пол, валовой внутренний продукт (GDP) на душу населения и индекс развития человеческого потенциала (HDI) за период с 1985 по 2016 год.

Содержание

Аннотация 3
1. Загрузка данных и предварительная обработка 5
1.1 Введение библиотеки и загрузка данных 5
1.2 Обработка данных 6
2. Эксплораторный анализ данных 11
2.1 Заболеваемость самоубийствами в разных странах 11
2.2 Самоубийства в течение времени 12
2.3 Распределение по возрасту и полу 14
2.4 Числовые переменные, коррелирующие с самоубийством 14
3. Подготовка данных для обучения 15
3.1 Создание целевой столбец 15
3.2 Сегментация набора данных 16
3.3 Определение входных и целевых столбцов 18
3.4 Масштабирование цифровых характеристик 19
3.5 Кодирование категориальных столбцов 19
4. Модель 20
4.1 Decision Tree 20
4.1.1 Важность признака 21
4.1.2 Настройка гиперпараметров 22
4.1.3 Хранение модули 26
4.1.4 Прогнозирование на новых входах 26
4.2 Random Forest 27
4.2.1 Важность признака 28
4.2.2 Визуализация 29
4.2.3 Настройка гиперпараметров 29
4.2.4 Хранение модули 31
4.2.5 Прогнозирование на новых входах 31
4.3 Сравнение 32
4.4 Настройка оптимального порога для модели 34
5. Заключение 35

1. Загрузка данных и предварительная обработка

1.1 Введение библиотеки и загрузка данных

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib
import seaborn as sns
import os
import warnings
%matplotlib inline
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
%cd /content/drive/MyDrive/DATASET
```

```
/content/drive/MyDrive/DATASET
```

Установите некоторые параметры по умолчанию:

```
[4] # Установите некоторые параметры по умолчанию
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', 100)
sns.set_style('darkgrid')
matplotlib.rcParams['font.size'] = 12
matplotlib.rcParams['figure.figsize'] = (10, 8)
matplotlib.rcParams['figure.facecolor'] = '#00000000'
warnings.filterwarnings('ignore')
```

Загружаем данные:

```
raw_df = pd.read_csv('master.csv')
raw_df.head()
```

1 to 5 of 5 entries Filter

index	country	year	sex	age	suicides_no	population	suicides/100k pop	country-year	HDI for year	gdp_for_year ($)	gdp_per_capita ($)	generation
0	Albania	1987	male	15-24 years	21	312900	6.71	Albania1987	NaN	2,156,624,900	796	Generation X
1	Albania	1987	male	35-54 years	16	308000	5.19	Albania1987	NaN	2,156,624,900	796	Silent
2	Albania	1987	female	15-24 years	14	289700	4.83	Albania1987	NaN	2,156,624,900	796	Generation X
3	Albania	1987	male	75+ years	1	21800	4.59	Albania1987	NaN	2,156,624,900	796	G.I. Generation
4	Albania	1987	male	25-34 years	9	274300	3.28	Albania1987	NaN	2,156,624,900	796	Boomers

Show 25 per page

Like what you see? Visit the data table notebook to learn more about interactive tables.

1.2 Обработка данных

Давайте посмотрим на типы данных и количество нулевых значений в каждом столбце.

```
raw_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27820 entries, 0 to 27819
Data columns (total 12 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   country             27820 non-null  object
 1   year                27820 non-null  int64
 2   sex                 27820 non-null  object
 3   age                 27820 non-null  object
 4   suicides_no         27820 non-null  int64
 5   population          27820 non-null  int64
 6   suicides/100k pop   27820 non-null  float64
 7   country-year        27820 non-null  object
 8   HDI for year        8364 non-null   float64
 9    gdp_for_year ($)   27820 non-null  object
 10  gdp_per_capita ($)  27820 non-null  int64
 11  generation          27820 non-null  object
dtypes: float64(2), int64(4), object(6)
memory usage: 2.5+ MB
```

В имени столбца gdp_for_year ($) присутствуют ведущие и последующие пробелы.

```
raw_df.rename(columns={'  gdp_for_year  ($)  ':'gdp_for_year  ($)'},  inplace=True)
raw_df.columns
```

```
Index(['country', 'year', 'sex', 'age', 'suicides_no', 'population',
       'suicides/100k pop', 'country-year', 'HDI for year', 'gdp_for_year ($)',
       'gdp_per_capita ($)', 'generation'],
      dtype='object')
```

Имеется 6 числовых столбцов и 6 категориальных столбцов. В столбце ИРЧП за год содержится много пропущенных значений. Теперь давайте разберемся в значении некоторых характеристик в наборе данных.

- population: количество людей соответствующей возрастной группы
- suicides_no: количество самоубийств среди населения определенной возрастной группы
- HDI for year: индекс развития человеческого потенциала (ИРЧП) за год. ИРЧП - это составной индекс, измеряющий средние достижения в трех основных измерениях

человеческого развития - долгой и здоровой жизни, знаниях и достойном уровне жизни.

- gdp_for_year: валовой внутренний продукт (ВВП) за год. ВВП - это общая денежная или рыночная стоимость всех готовых товаров и услуг, произведенных в пределах страны.
- gdp_per_capita: ВВП страны, деленный на общую численность населения.

1.2.1 Очистка данных

Далее мы очистим данные и изучим их для получения быстрых выводов. В HDI for year много пропущенных значений. Давайте начнем с манипуляций с этим столбцом.

```
# Проверьте отсутствующее значение и отбрось
raw_df.loc[raw_df['HDI for year'].isna(), ['country', 'year']].drop_duplicates()
```

	country	year
0	Albania	1987
12	Albania	1988
24	Albania	1989
36	Albania	1992
48	Albania	1993
...	...	...
27688	Uzbekistan	2001
27700	Uzbekistan	2002
27712	Uzbekistan	2003
27724	Uzbekistan	2004
27748	Uzbekistan	2009

1624 rows × 2 columns

Давайте загрузим данные HDI 190 стран с 1990-2019 гг. в формате csv (hdi.csv) во фрейм данных Pandas.

```
hdi_df = pd.read_csv('hdi.csv', index_col='Country')
hdi_df.head()
```

Country	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002
Afghanistan	0.302	0.307	0.316	0.312	0.307	0.331	0.335	0.339	0.344	0.348	0.350	0.353	0.384
Albania	0.650	0.631	0.615	0.618	0.624	0.637	0.646	0.645	0.655	0.665	0.671	0.678	0.684
Algeria	0.572	0.576	0.582	0.586	0.590	0.595	0.602	0.611	0.621	0.629	0.637	0.647	0.657
Andorra	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.813	0.815	0.820
Angola	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.391	0.400	0.410	0.426

```
hdi_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 189 entries, Afghanistan to Zimbabwe
Data columns (total 30 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   1990    144 non-null    float64
 1   1991    144 non-null    float64
 2   1992    144 non-null    float64
 3   1993    144 non-null    float64
 4   1994    144 non-null    float64
 5   1995    148 non-null    float64
 6   1996    148 non-null    float64
 7   1997    148 non-null    float64
 8   1998    148 non-null    float64
 9   1999    151 non-null    float64
 10  2000    174 non-null    float64
 11  2001    174 non-null    float64
 12  2002    175 non-null    float64
 13  2003    176 non-null    float64
 14  2004    178 non-null    float64
 15  2005    185 non-null    float64
 16  2006    186 non-null    float64
 17  2007    186 non-null    float64
 18  2008    186 non-null    float64
 19  2009    186 non-null    float64
 20  2010    188 non-null    float64
 21  2011    188 non-null    float64
 22  2012    188 non-null    float64
 23  2013    188 non-null    float64
 24  2014    188 non-null    float64
 25  2015    188 non-null    float64
 26  2016    188 non-null    float64
 27  2017    189 non-null    float64
 28  2018    189 non-null    float64
 29  2019    189 non-null    float64
dtypes: float64(30)
memory usage: 45.8+ KB
```

Давайте попробуем восполнить недостающую информацию об HDI в raw_df, используя информацию в hdi_df. Хотя это может не полностью восполнить недостающие значения, так как в hdi_df все еще отсутствуют HDI для некоторых стран и лет, это должно помочь сократить довольно много недостающих HDI в raw_df.

```
# Итерация по каждой строке в raw_df и вменение отсутству
for row in raw_df.itertuples():
    # Исключите строки с годом до 1990 года, поскольку с
    if row.year >= 1990:
        try:
            hdi = hdi_df.loc[row.country, str(row.year)]
            raw_df.at[row.Index, 'HDI for year'] = hdi
        except KeyError:
            pass
```

Проверим, уменьшится ли количество отсутствующих значений.

```
raw_df.loc[raw_df['HDI for year'].isna(), ['country', 'year']].drop_duplicates()
```

	country	year
0	Albania	1987
12	Albania	1988
24	Albania	1989
264	Antigua and Barbuda	1985
276	Antigua and Barbuda	1986
...	...	...
27616	Uzbekistan	1995
27628	Uzbekistan	1996
27640	Uzbekistan	1997
27652	Uzbekistan	1998
27664	Uzbekistan	1999

408 rows × 2 columns

Мы видим, что количество стран и лет с отсутствующим HDI сократилось с 1 624 строк до 408 строк.

Из hdi_df мы видим, что HDI в целом увеличивается с годами. Поскольку оставшиеся недостающие HDI - это значения в более ранние годы, разумным подходом будет вменение этих недостающих значений с использованием минимального HDI каждой страны.

```
# Группировка HDI по странам и фильтрация по минимальнос
min_hdi_df = raw_df[['country', 'HDI for year']].groupby('country').agg({"HDI for year": "min"})
min_hdi_df.head()
```

country	HDI for year
Albania	0.615
Antigua and Barbuda	0.759
Argentina	0.694
Armenia	0.617
Aruba	NaN

```
# Преобразование фрейма данных в дикту
min_hdi_dict = min_hdi_df.to_dict()

# Составьте карту отсутствующего значения HDI в соответствии с вхо
mask = raw_df['HDI for year'].isna()
raw_df.loc[mask, 'HDI for year'] = raw_df.loc[mask, 'country'].apply(lambda x: min_hdi_dict['HDI for year'][x])
```

Для оставшихся недостающих значений давайте вменим среднее значение HDI по странам за каждый год.

```
# Вычислите ряд pandas со средним значением HDI для каждого года
mean_hdi = hdi_df.mean()

# Создайте маску bool для строк с отсутствующим HDI (до 1990 года и далее/посл
mask_bf_1990 = (raw_df['HDI for year'].isna()) & (raw_df['year'] < 1990)
mask_aft_1990 = (raw_df['HDI for year'].isna()) & (raw_df['year'] >= 1990)

# Вставьте отсутствующий HDI, используя средний HDI (в/после 1990 года). Если о
raw_df.loc[mask_aft_1990, 'HDI for year'] = raw_df.loc[mask_aft_1990, 'year'].apply(lambda x: mean_hdi[str(x)])
raw_df.loc[mask_bf_1990, 'HDI for year'] = mean_hdi['1990']
```

```
raw_df['HDI for year'].isna().sum()
```

```
0
```

```
raw_df.drop(columns=['country-year'], inplace=True)
raw_df.columns
```

```
Index(['country', 'year', 'sex', 'age', 'suicides_no', 'population',
       'suicides/100k pop', 'HDI for year', 'gdp_for_year ($)',
       'gdp_per_capita ($)', 'generation'],
      dtype='object')
```

2. Эксплораторный анализ данных

Давайте изучим и визуализируем набор данных, чтобы получить некоторые полезные сведения перед обучением наших моделей машинного обучения.

2.1 Заболеваемость самоубийствами в разных странах

Сначала мы визуализируем уровень самоубийств по странам на карте с помощью библиотеки geopandas.

```
import geopandas
```

```
world = geopandas.read_file(geopandas.datasets.get_path('naturalearth_lowres'))
world.head()
```

	pop_est	continent	name	iso_a3	gdp_md_est	geometry
0	920938	Oceania	Fiji	FJI	8374.0	MULTIPOLYGON (((180.00000 -16.06713, 180.00000...
1	53950935	Africa	Tanzania	TZA	150600.0	POLYGON ((33.90371 -0.95000, 34.07262 -1.05982...
2	603253	Africa	W. Sahara	ESH	906.5	POLYGON ((-8.66559 27.65643, -8.66512 27.58948...
3	35623680	North America	Canada	CAN	1674000.0	MULTIPOLYGON (((-122.84000 49.00000, -122.9742...
4	326625791	North America	United States of America	USA	18560000.0	MULTIPOLYGON (((-122.84000 49.00000, -120.0000...

Давайте сгруппируем данные о самоубийствах по странам и рассчитаем агрегированное количество самоубийств на 100 тыс. населения для каждой страны.

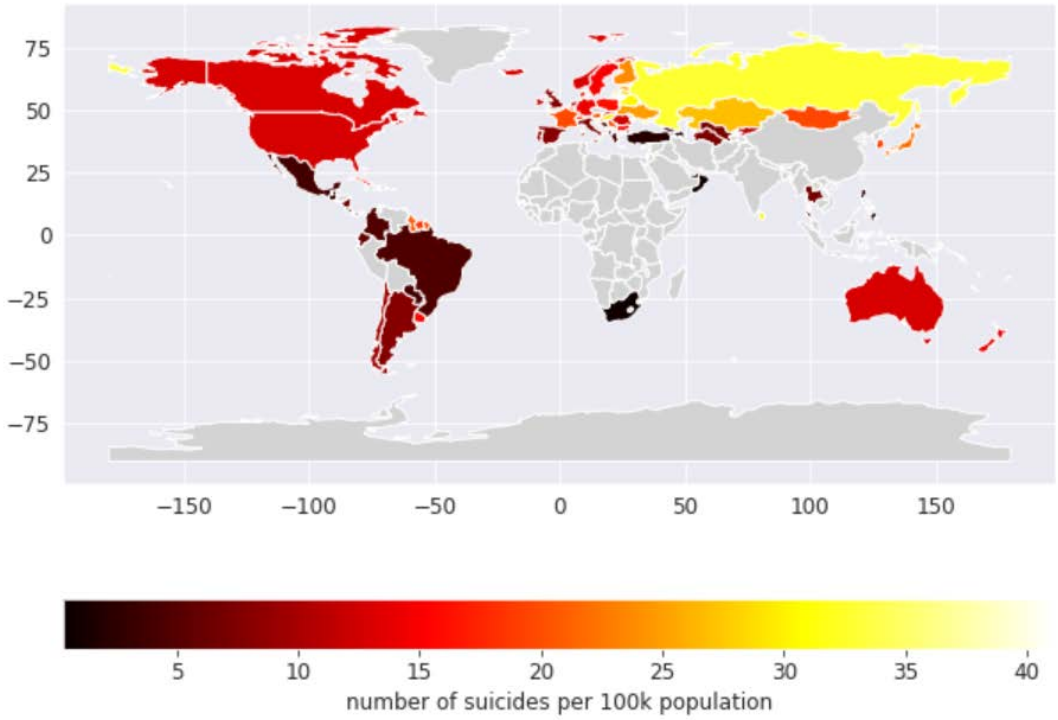
```
# Сгруппируйте данные по странам и найдите сумму количества самоубийст
suicide_by_country_df = raw_df.groupby('country')[['suicides_no', 'population']].sum().reset_index()

# Рассчитайте количество самоубийств/100 тыс. человек населения
suicide_by_country_df['suicides/100k pop'] = 100000 * suicide_by_country_df['suicides_no'] / suicide_by_country_df['population']

# Отсортируйте строки по количеству самоубийств/100 тыс. населения от на
suicide_by_country_df.sort_values(by='suicides/100k pop', ascending=False, inplace=True)
suicide_by_country_df.head()
```

	country	suicides_no	population	suicides/100k pop
52	Lithuania	28039	68085210	41.182219
75	Russian Federation	1209742	3690802620	32.777207
87	Sri Lanka	55641	182525626	30.483939
11	Belarus	59892	197372292	30.344685
40	Hungary	73891	248644256	29.717558

Измените название страны.

```
country_match = {'Russia': 'Russian Federation', 'United States of America': 'United States', 'South Korea': 'Republic of Korea',
                 'Bosnia and Herz.': 'Bosnia and Herzegovina'}
world['name'].replace(country_match, inplace=True)
```

```
# Объедините два кадра данных вместе

table = world.merge(suicide_by_country_df, how='left', left_on=['name'], right_on=['country'])

table.plot(column='suicides/100k pop',
                    cmap='hot',
                    legend=True,
                    legend_kwds={'label': 'number of suicides per 100k population',
                                 'orientation': 'horizontal'},
                    missing_kwds={'color': 'lightgrey'});
```

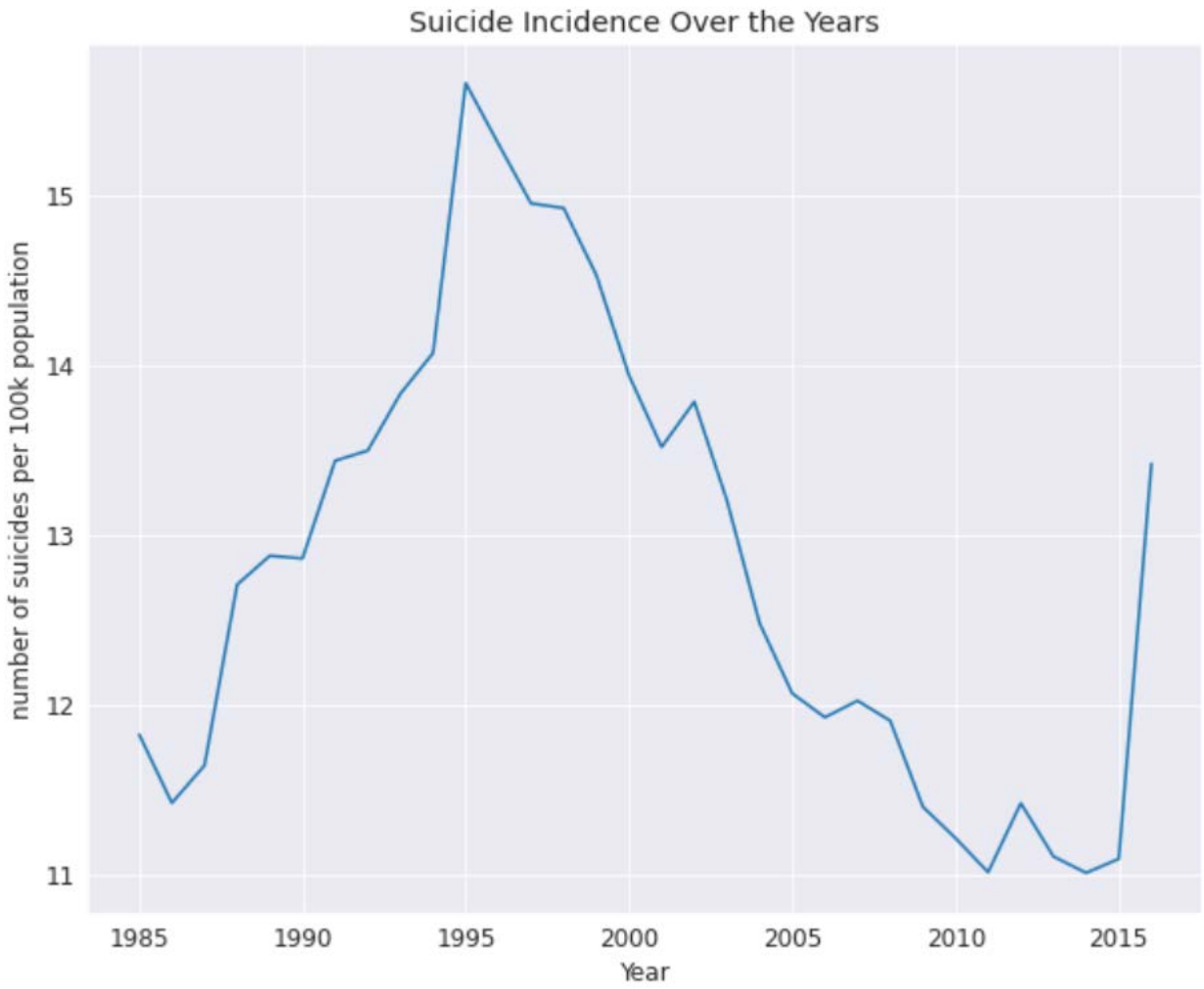


Судя по карте, частота самоубийств выше в Российской Федерации и некоторых европейских странах по сравнению с Австралией и Америкой.

2.2 Самоубийства в течение времени

Давайте посмотрим на частоту самоубийств за последние 20 лет.

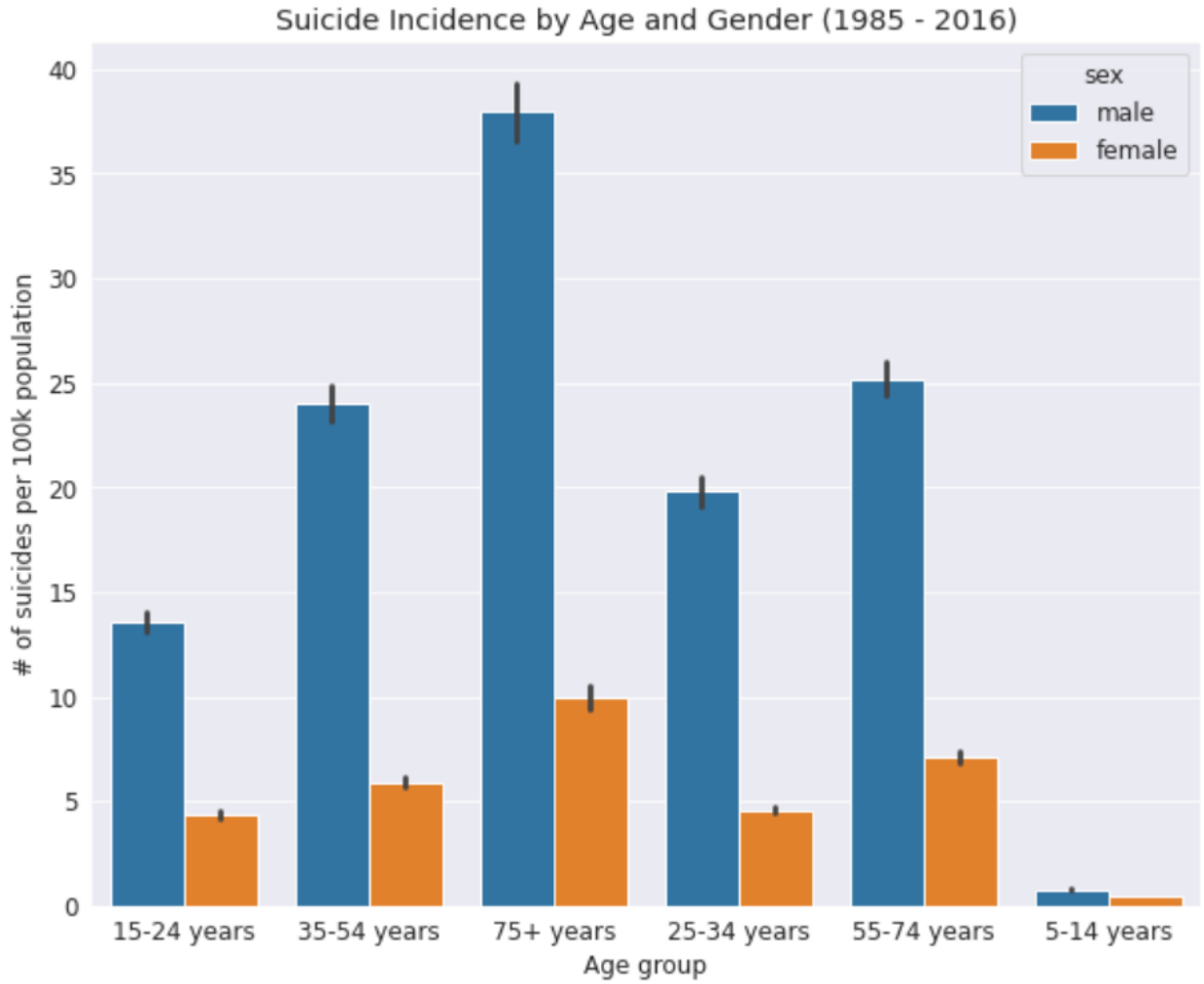
```
# x-axis с 1985 до 2016
x = np.arange(1985, 2017)
y = raw_df.groupby('year')['suicides/100k pop'].mean()

sns.lineplot(x=x, y=y)
plt.xlabel('Year')
plt.ylabel('number of suicides per 100k population')
plt.title('Suicide Incidence Over the Years');
```



Заболеваемость самоубийствами в странах нашей выборки данных достигла пика в 1995 году и с тех пор следовала тенденции к снижению. Однако в 2015 году произошел резкий всплеск числа самоубийств.

2.3 Распределение по возрасту и полу

```
sns.barplot(data=raw_df,  x='age',  y='suicides/100k  pop',  hue='sex')
plt.xlabel('Age  group')
plt.ylabel('#  of  suicides  per  100k  population')
plt.title('Suicide  Incidence  by  Age  and  Gender  (1985  -  2016)');
```



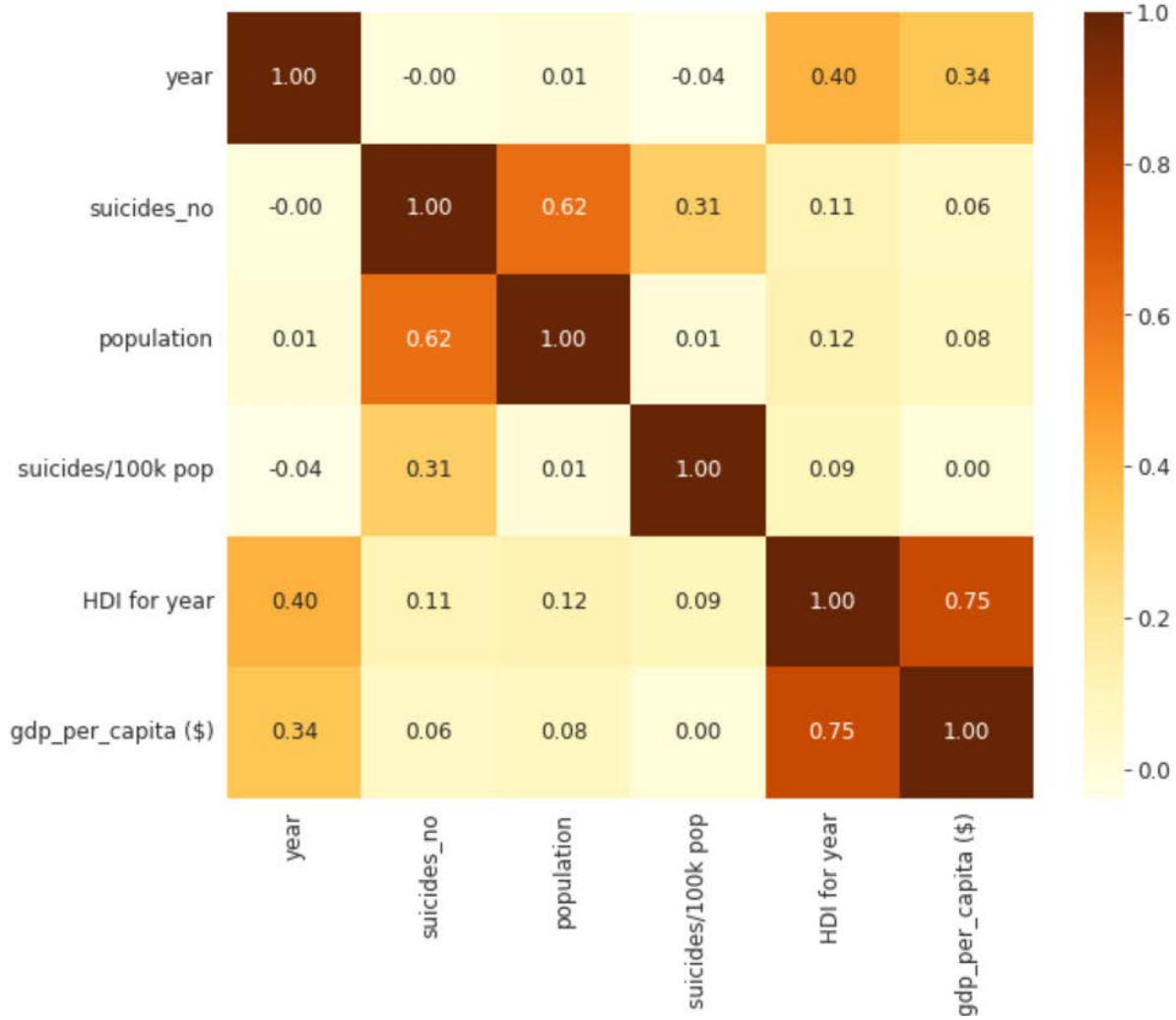
Пожилые люди чаще совершают самоубийства.

Во всех возрастных группах самоубийства почти в 3-4 раза чаще происходят у мужчин, чем у женщин. Это может быть связано с тем, что мужские способы самоубийства часто более жестокие, что повышает вероятность их завершения до того, как кто-то успеет вмешаться.

2.4 Числовые переменные, коррелирующие с самоубийством

После изучения взаимосвязи между категориальными переменными и самоубийством, давайте посмотрим корреляцию между числовыми переменными и самоубийством. Мы можем визуализировать эту взаимосвязь с помощью тепловой карты.

```
sns.heatmap(raw_df.corr(),  annot=True,  cmap='YlOrBr',  fmt='.2f');
```



3. Подготовка данных для обучения

3.1 Создание целевой столбец

Давайте создадим целевой столбец suicide_risk (т.е. высокий / низкий риск), используя информацию из suicides/100k pop.

Если значение самоубийств/100 тыс. человек превышает среднее значение самоубийств/100 тыс. человек, мы классифицируем риск самоубийств как высокий, в противном случае - как низкий.

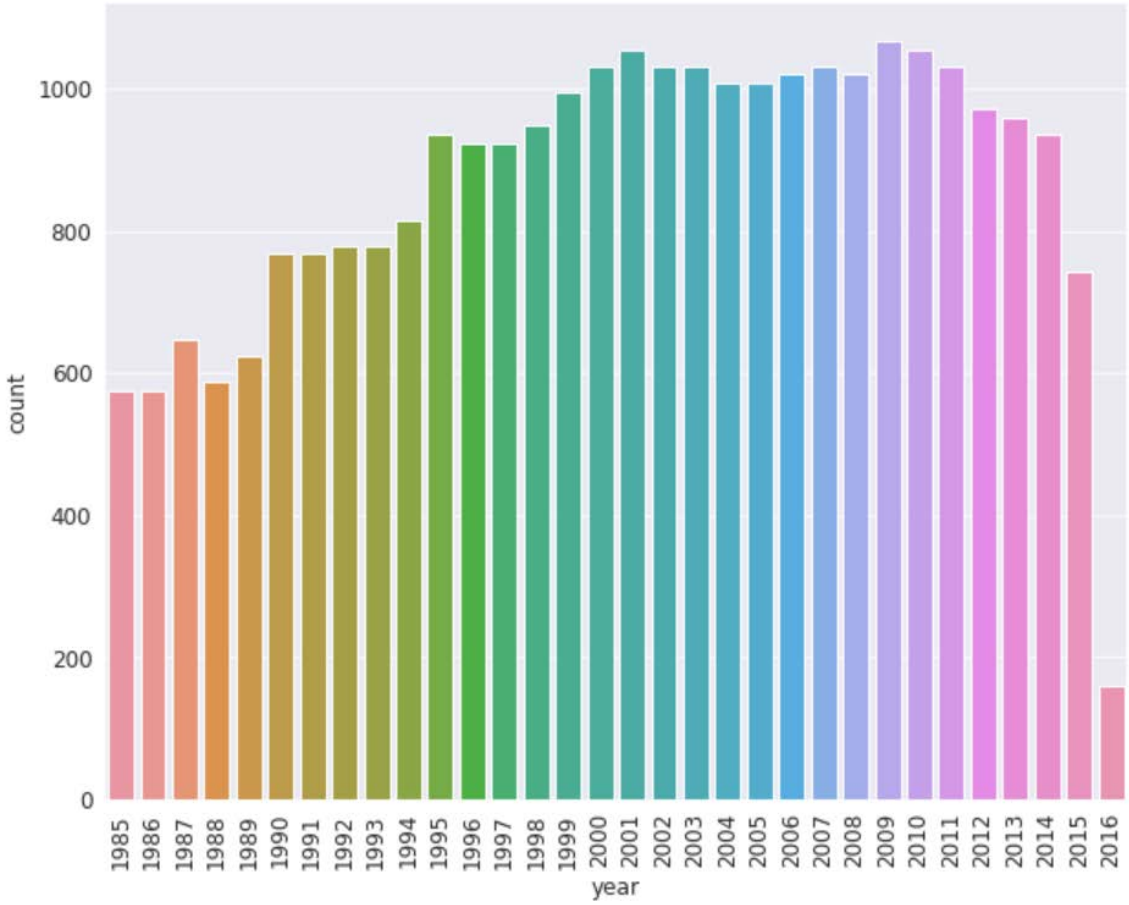
```
raw_df['suicide_risk']  =  'low'
```

```
raw_df.loc[raw_df['suicides/100k  pop']  >  raw_df['suicides/100k  pop'].mean(),  'suicide_risk']  =  'high'
raw_df['suicide_risk'].value_counts()
```

```
low     19061
high     8759
Name: suicide_risk, dtype: int64
```

3.2 Сегментация набора данных

```
sns.countplot(x='year',  data=raw_df)
plt.xticks(rotation='90');
```



Давайте вместо этого посмотрим на кумулятивный процент.

```
# Создать серию, содержащую количество данных за год
year_dt = raw_df.year.value_counts()
year_dt.sort_index(inplace=True)
cum_pct = 100 * year_dt.cumsum() / year_dt.sum()
cum_pct
```

```
1985      2.070453
1986      4.140906
1987      6.470165
1988      8.583753
1989     10.826743
1990     13.587347
1991     16.347951
1992     19.151689
1993     21.955428
1994     24.888569
1995     28.253055
1996     31.574407
1997     34.895758
1998     38.303379
1999     41.883537
2000     45.593098
2001     49.388929
2002     53.098490
2003     56.808052
2004     60.431344
2005     64.054637
2006     67.721064
2007     71.430625
2008     75.097052
2009     78.936017
2010     82.731848
2011     86.441409
2012     89.935298
2013     93.386053
2014     96.750539
2015     99.424874
2016    100.000000
Name: year, dtype: float64
```

Давайте разделим данные на тренировочный/оценочный/тестовый набор, используя 2004 и 2010 год в качестве точек останова.

```
train_df = raw_df.loc[raw_df['year'] <= 2004]
val_df = raw_df.loc[(raw_df['year'] >= 2005) & (raw_df['year'] <=2010)]
test_df = raw_df.loc[raw_df['year'] >= 2011]

print('train_df.shape: ', train_df.shape)
print('val_df.shape: ', val_df.shape)
print('test_df.shape: ', test_df.shape)
```

```
train_df.shape:  (16812, 12)
val_df.shape:  (6204, 12)
test_df.shape:  (4804, 12)
```

3.3 Определение входных и целевых столбцов

Часто не все столбцы в наборе данных полезны для обучения модели. Мы будем игнорировать следующие столбцы:

- year: не имеет значения, поскольку мы прогнозируем риск самоубийства конкретного человека в будущем.
- suicides_no, population: содержит избыточную информацию, как suicides/100k pop
- suicides/100 тыс. населения: содержит избыточную информацию как риск самоубийств
- gdp_for_year ($): содержит избыточную информацию как gdp_per_capita ($)
- generation: содержит избыточную информацию как возраст. Каждое поколение соответствует определенной возрастной группе.

```
input_cols = ['country', 'sex', 'age', 'HDI for year', 'gdp_per_capita ($)']
target_col = 'suicide_risk'
```

```
train_inputs = train_df.loc[:, input_cols]
train_target = train_df.loc[:, target_col]

val_inputs = val_df.loc[:, input_cols]
val_target = val_df.loc[:, target_col]

test_inputs = test_df.loc[:, input_cols]
test_target = test_df.loc[:, target_col]
```

Давайте также определим числовые и категориальные столбцы.

```
numeric_cols = list(train_inputs.select_dtypes(include=np.number).columns)
categorical_cols = list(train_inputs.select_dtypes(include='object').columns)

print('numeric columns: ', numeric_cols)
print('categorical columns: ', categorical_cols)
```

```
numeric columns: ['HDI for year', 'gdp_per_capita ($)']
categorical columns: ['country', 'sex', 'age']
```

3.4 Масштабирование цифровых характеристик

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler().fit(raw_df.loc[:, numeric_cols])
```

```
train_inputs.loc[:, numeric_cols] = scaler.transform(train_inputs.loc[:, numeric_cols])
val_inputs.loc[:, numeric_cols] = scaler.transform(val_inputs.loc[:, numeric_cols])
test_inputs.loc[:, numeric_cols] = scaler.transform(test_inputs.loc[:, numeric_cols])
```

```
train_inputs[numeric_cols].describe()
```

	HDI for year	gdp_per_capita ($)
count	16812.000000	16812.000000
mean	0.543938	0.091012
std	0.198714	0.098749
min	0.000000	0.000000
25%	0.399142	0.016384
50%	0.536481	0.042918
75%	0.695279	0.153068
max	0.967811	0.637489

3.5 Кодирование категориальных столбцов

Здесь используется OneHotEncoder из sklearn.preprocessing.

```
raw_df[categorical_cols].nunique()
```

```
country    101
sex          2
age          6
dtype: int64
```

```
from sklearn.preprocessing import OneHotEncoder

encoder = OneHotEncoder(handle_unknown='ignore').fit(raw_df[categorical_cols])
```

Давайте создадим имена столбцов для каждой отдельной категории с помощью get_feature_names и добавим все столбцы в train_inputs, val_inputs и test_inputs.

```
encoded_cols = list(encoder.get_feature_names(categorical_cols))

train_inputs[encoded_cols] = encoder.transform(train_inputs.loc[:, categorical_cols]).toarray()
val_inputs[encoded_cols] = encoder.transform(val_inputs.loc[:, categorical_cols]).toarray()
test_inputs[encoded_cols] = encoder.transform(test_inputs.loc[:, categorical_cols]).toarray()
```

```
X_train = train_inputs[numeric_cols + encoded_cols]
X_val = val_inputs[numeric_cols + encoded_cols]
X_test = test_inputs[numeric_cols + encoded_cols]
```

4. Модель

4.1 Decision Tree

Давайте обучим классификатор дерева решений для классификации риска самоубийства на высокий или низкий на основе входных данных.

```
from sklearn.metrics import accuracy_score, confusion_matrix

train_pred = tree.predict(X_train)
accuracy_score(train_target, train_pred)
```

```
1.0
```

```
from sklearn.tree import DecisionTreeClassifier

tree = DecisionTreeClassifier(random_state=1)
tree.fit(X_train, train_target)
```

```
DecisionTreeClassifier(random_state=1)
```

Давайте оценим модель, используя валидационный набор.

```
val_pred = tree.predict(X_val)
accuracy_score(val_target, val_pred)
```

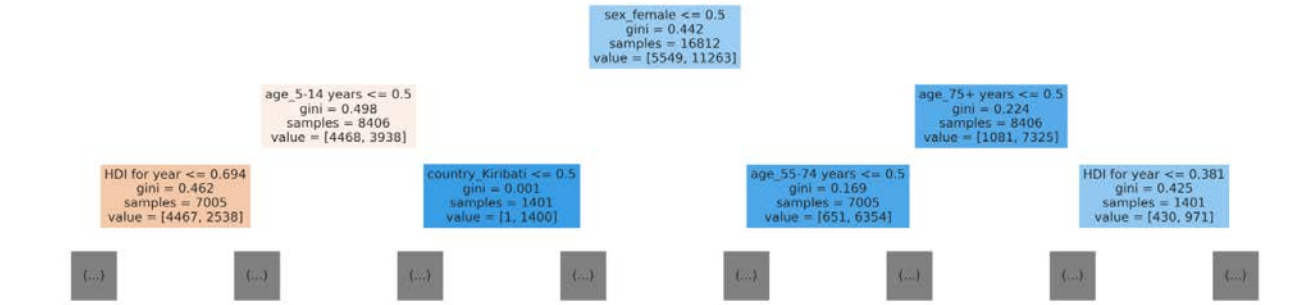
```
0.8813668600902643
```

```
val_target.value_counts()  /  len(val_target)
```

```
low     0.705029
high    0.294971
Name: suicide_risk, dtype: float64
```

Давайте также визуализируем первые несколько слоев дерева решений.

```
from sklearn.tree import plot_tree

plt.figure(figsize=(80, 20))
plot_tree(tree, feature_names=X_train.columns, max_depth=2, filled=True);
```

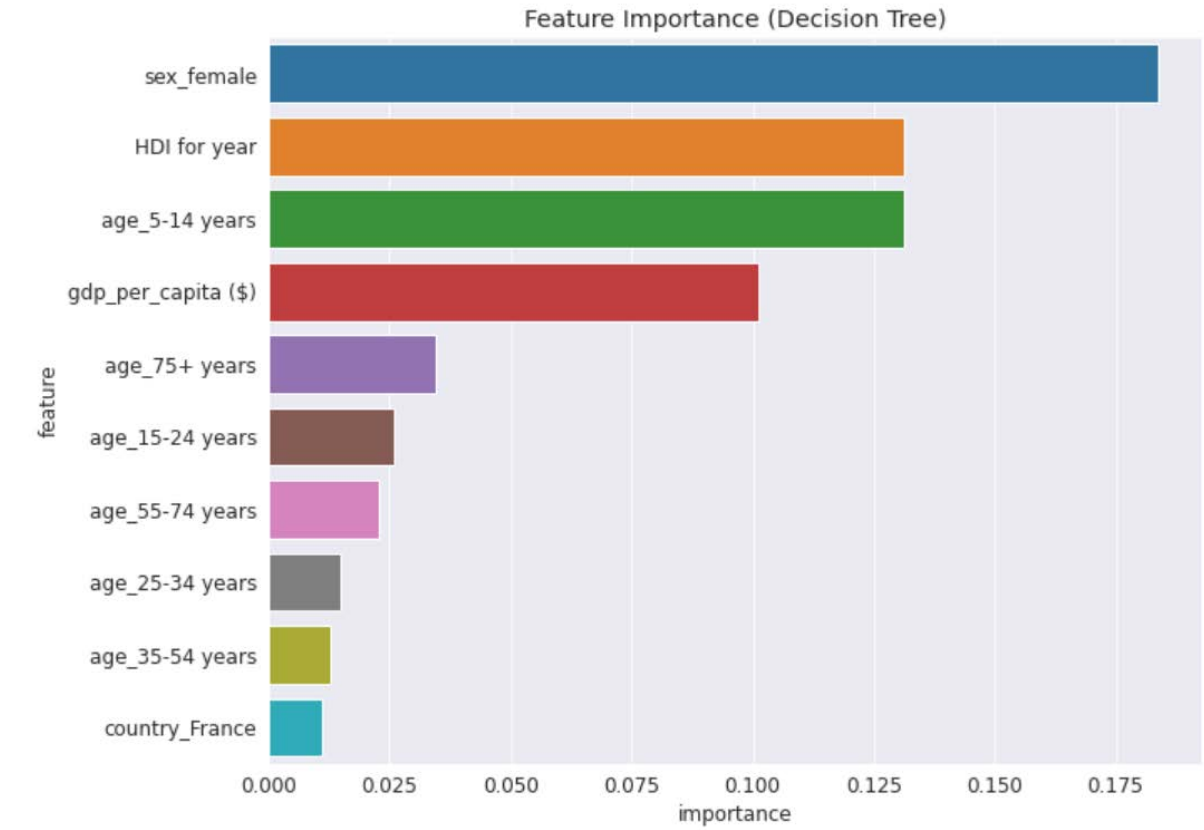


Мы видим, что модель классифицирует входные данные путем принятия ряда решений. Если разделение приводит к увеличению объема информации (измеряется индексом Джини - более низкий индекс Джини указывает на лучшее разделение), то такое разделение будет произведено. В нашей модели дерева решений входные данные сначала были разделены по полу, затем по возрастной группе, HDI за год и стране.

4.1.1 Важность признака

На основе расчета индекса Джини каждому признаку присваивается "важность Джини", которая рассчитывается как (нормализованное) общее снижение индекса Джини, вызванное этим признаком. Давайте посмотрим, какой признак более важен для прогнозирования риска самоубийства.

```
importance_df = pd.DataFrame({
        'feature': X_train.columns,
        'importance': tree.feature_importances_
}).sort_values(by='importance', ascending=False)

sns.barplot(data=importance_df.head(10), x='importance', y='feature')
plt.title('Feature Importance (Decision Tree)');
```



Наиболее важной характеристикой для прогнозирования риска самоубийства является пол_женщины (0,18), за ней следуют HDI за год и возраст_5-14 лет, которые имеют схожую значимость около 0,13. Другие важные характеристики включают gdp_per_capita и другие возрастные группы.

4.1.2 Настройка гиперпараметров

- max_depth: контролирует общую сложность дерева решений (при адекватном предположении, что построенное дерево симметрично)
- min_samples_split: минимальное количество выборок, необходимое для разделения внутреннего узла

1) Настройка максимальной глубины

```
tree.tree_.max_depth
```

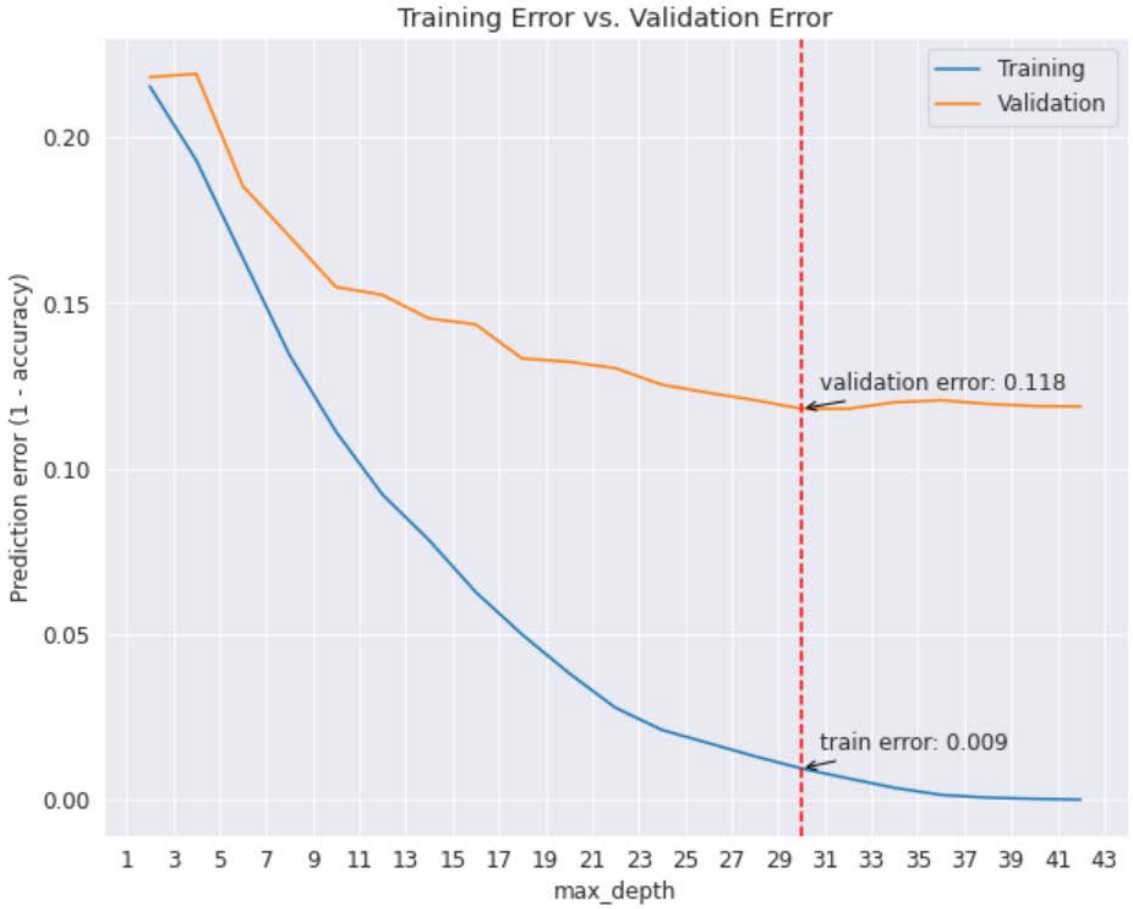
```
42
```

Давайте поэкспериментируем с различными max_depth, используя вспомогательную функцию max_depth_tuning. Мы протестируем max_depth для диапазона от 2 до 42, с шагом 2.

```
def max_depth_tuning(md):
    tree = DecisionTreeClassifier(random_state=1, max_depth=md)
    tree.fit(X_train, train_target)
    train_pred = tree.predict(X_train)
    val_pred = tree.predict(X_val)
    train_error = 1 - accuracy_score(train_target, train_pred)
    val_error = 1 - accuracy_score(val_target, val_pred)
    return {'max_depth': md, 'train_error': train_error, 'val_error': val_error}
```

```
errors_df = pd.DataFrame([max_depth_tuning(md) for md in range(2, 43, 2)])
```

```
fig, ax = plt.subplots()
sns.lineplot(data=errors_df, x='max_depth', y='train_error', ax=ax)
sns.lineplot(data=errors_df, x='max_depth', y='val_error', ax=ax)
plt.ylabel('Prediction error (1 - accuracy)')
plt.xticks(range(1, 44, 2))
plt.title('Training Error vs. Validation Error')
plt.legend(['Training', 'Validation'])

plt.axvline(x=30, color='r', linestyle='--')
ax.annotate('validation error: 0.118',
                        xy=(30, 0.117988),
                        xycoords='data',
                        fontsize=12,
                        xytext=(10, 10),
                        textcoords='offset points',
                        arrowprops=dict(arrowstyle="->", color='black',
                                                connectionstyle="arc3"))
ax.annotate('train error: 0.009',
                        xy=(30, 0.009398),
                        xycoords='data',
                        fontsize=12,
                        xytext=(10, 10),
                        textcoords='offset points',
                        arrowprops=dict(arrowstyle="->", color='black',
                                                connectionstyle="arc3"))
plt.show()
```



Как ошибка обучения, так и ошибка валидации показывают тенденцию к уменьшению с увеличением max_depth. Это не так очевидно из графика, но ошибка валидации является самой низкой, когда max_depth составляет около 30. За пределами max_depth 30 ошибка валидации немного увеличивается.

2) Лучшая комбинация максимальной глубины и минимального разделения выборок

Давайте включим в процесс настройки еще один гиперпараметр min_samples_split. Min_samples_split - это минимальное количество образцов, необходимое для разделения внутреннего узла.

Здесь мы будем использовать функцию GridSearchCV из sklearn.model_selection. Она поможет нам найти оптимальную комбинацию заданных гиперпараметров, которая дает наилучшее предсказание. Сначала укажем тип и диапазон гиперпараметров, которые мы хотим проверить.

```
param_dict = {
        "max_depth": range(2, 40, 2),
        "min_samples_split": range(2, 10, 2)
}
```

Давайте объединим обучающее и проверочное множества перед запуском функции чтобы использовать GridSearchCV.

```
X = pd.concat([X_train, X_val])
Y = pd.concat([train_target, val_target])
print(X.shape, Y.shape)
```

```
(23016, 111) (23016,)
```

```
from sklearn.model_selection import GridSearchCV

tree = DecisionTreeClassifier(random_state=1)
grid = GridSearchCV(tree, param_grid=param_dict)
grid.fit(X, Y)
```

```
GridSearchCV(estimator=DecisionTreeClassifier(random_state=1),
             param_grid={'max_depth': range(2, 40, 2),
                         'min_samples_split': range(2, 10, 2)})
```

Наилучшая комбинация гиперпараметров и оценка точности.

```
print('best params: ', grid.best_params_)
print('accuracy score: ', grid.best_score_)
```

```
best params: {'max_depth': 34, 'min_samples_split': 2}
accuracy score: 0.8560991839832482
```

Оптимальным сочетанием гиперпараметров является max_depth равное 34 и min_samples_split равное 2. Оценим производительность этой модели на тестовом множестве.

```
test_pred = grid.predict(X_test)
accuracy_score(test_target, test_pred)
```

```
0.9094504579517069
```

4.1.3 Хранение модули

```
import joblib
```

```
suicide_risk_dt = {
        'model': grid,
        'scaler': scaler,
        'encoder': encoder,
        'input_cols': input_cols,
        'target_col': target_col,
        'numeric_cols': numeric_cols,
        'categorical_cols': categorical_cols,
        'encoded_cols': encoded_cols
}
```

```
joblib.dump(suicide_risk_dt, 'suicide_risk_dt.joblib')
```

```
['suicide_risk_dt.joblib']
```

4.1.4 Прогнозирование на новых входах

```
suicide_risk_dt_clf = joblib.load('suicide_risk_dt.joblib')
```

```
def predict_input(model, single_input):
        input_df = pd.DataFrame([single_input])
        input_df[model['numeric_cols']] = model['scaler'].transform(input_df[model['numeric_cols']])
        input_df[model['encoded_cols']] = model['encoder'].transform(input_df[model['categorical_cols']]).toarray()
        X_input = input_df[model['numeric_cols'] + model['encoded_cols']]
        pred = model['model'].predict(X_input)[0]
        prob = model['model'].predict_proba(X_input)[0][list(model['model'].classes_).index(pred)]
        return pred, prob
```

```
new_input = {
        'country': 'Republic of Korea',
        'year': 2020,
        'sex': 'male',
        'age': '75+ years',
        'suicides_no': 1400,
        'population': 900000,
        'suicides/100k pop': 155.55,
        'HDI for year': 0.9,
        'gdp_for_year ($)': '1,000,000,000',
        'gdp_per_capita ($)': 29000,
        'generation': 'Silent'
}
```

```
predict_input(suicide_risk_dt_clf, new_input)
```

```
('high', 1.0)
```

Давайте изменим некоторые важные предикторы риска самоубийства (например, пол_женский, HDI за год) и посмотрим, изменится ли прогноз.

```
new_input_2 = {
        'country': 'Singapore',
        'year': 2020,
        'sex': 'female',
        'age': '15-24 years',
        'suicides_no': 14,
        'population': 250000,
        'suicides/100k pop': 5.6,
        'HDI for year': 0.7,
        'gdp_for_year ($)': '300,000,000,000',
        'gdp_per_capita ($)': 80000,
        'generation': 'Millenials'
}
```

```
predict_input(suicide_risk_dt_clf, new_input_2)
```

```
('low', 1.0)
```

4.2 Random Forest

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_jobs=-1, random_state=1)
rf.fit(X_train, train_target)
```

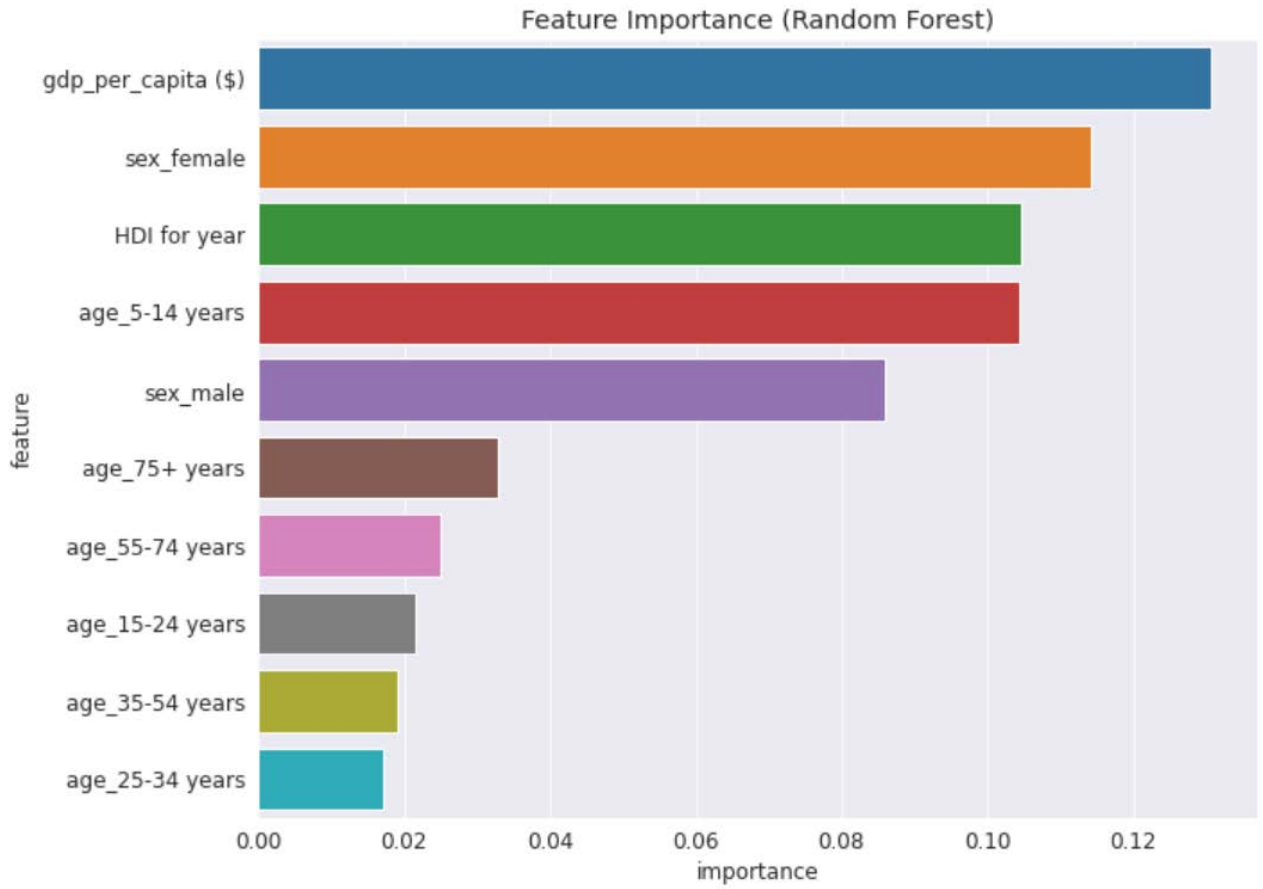
```
RandomForestClassifier(n_jobs=-1, random_state=1)
```

```
train_pred = rf.predict(X_train)
print('training accuracy: ', accuracy_score(train_target, train_pred))

val_pred = rf.predict(X_val)
print('validation accuracy: ', accuracy_score(val_target, val_pred))
```

```
training accuracy:  0.9999405186771354
validation accuracy:  0.902321083172147
```

4.2.1 Важность признака

```
importance_df = pd.DataFrame({
        'feature': X_train.columns,
        'importance': rf.feature_importances_
}).sort_values(by='importance', ascending=False)

sns.barplot(data=importance_df.head(10), x='importance', y='feature')
plt.title('Feature Importance (Random Forest)');
```



gdp_per_capita ($) оказывается самым важным признаком в модели случайного леса (4-й важный признак в модели дерева решений). Другие важные признаки (например,

пол_женский, HDI за год, возраст_5-14 лет) во многом совпадают с результатами модели дерева решений.

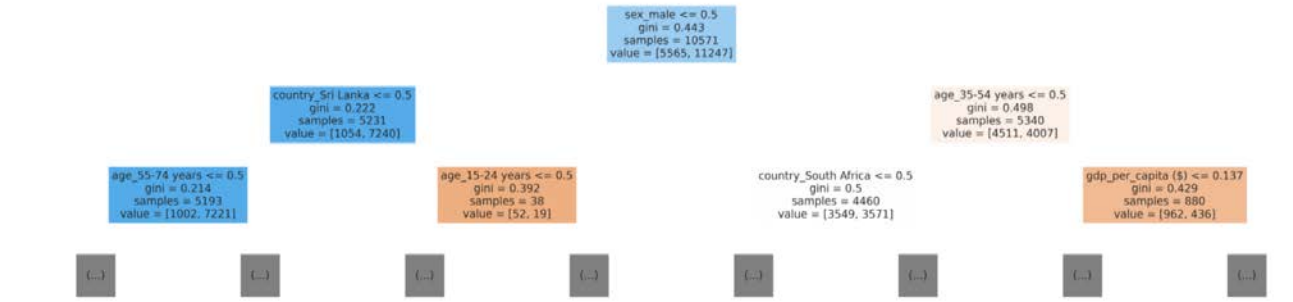
4.2.2 Визуализация

Давайте рассмотрим несколько деревьев решений из случайного леса. Доступ к деревьям решений можно получить через атрибут estimators_.

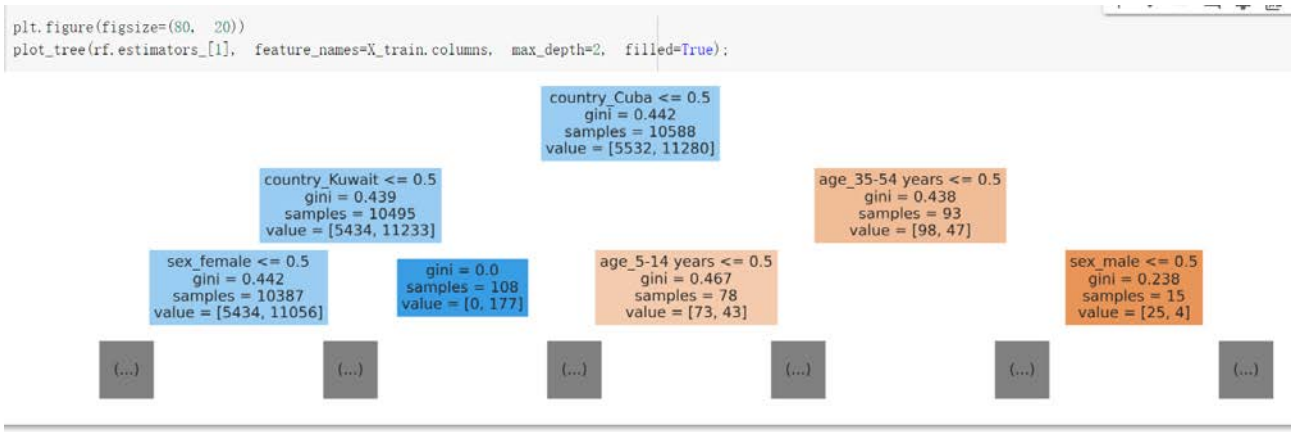
```
len(rf.estimators_)
```

```
100
```

```
plt.figure(figsize=(80, 20))
plot_tree(rf.estimators_[0], feature_names=X_train.columns, max_depth=2, filled=True);
```




```
plt.figure(figsize=(80, 20))
plot_tree(rf.estimators_[1], feature_names=X_train.columns, max_depth=2, filled=True);
```



4.2.3 Настройка гиперпараметров

- n_estimators: количество деревьев в лесу
- max_features: количество признаков, которые необходимо учитывать при поиске наилучшего разделения

Давайте сначала определим вспомогательную функцию.

```
def train_and_evaluate(X_train, train_target, X_val, val_target, **params):
        rf = RandomForestClassifier(n_jobs=-1, random_state=1, **params)
        rf.fit(X_train, train_target)
        train_pred = rf.predict(X_train)
        val_pred = rf.predict(X_val)
        train_error = 1 - accuracy_score(train_target, train_pred)
        val_error = 1 - accuracy_score(val_target, val_pred)
        return {'model': rf, **params, 'train_error': train_error, 'val_error': val_error}
```

Давайте оценим модель случайного леса, используя n_estimators от (10, 100, 500) и max_features от (sqrt, log2). Мы соберем ошибки обучения и ошибки валидации и используем их для выбора оптимальной модели.

```
errors_list = []

for n_estimators in [10, 100, 500]:
    for max_features in ['sqrt', 'log2']:
        outcome = train_and_evaluate(X_train, train_target, X_val, val_target, n_estimators=n_estimators, max_features=max_features)
        errors_list.append(outcome)
```

Преобразуем список errors_list (список словаря) в датафрейм Pandas и отсортируем по ошибкам валидации.

```
errors_df = pd.DataFrame(errors_list).sort_values(by='val_error')
errors_df
```

	model	n_estimators	max_features	train_error	val_error
5	(DecisionTreeClassifier(max_features='log2', r...	500	log2	0.000000	0.092199
3	(DecisionTreeClassifier(max_features='log2', r...	100	log2	0.000059	0.095583
2	(DecisionTreeClassifier(max_features='sqrt', r...	100	sqrt	0.000059	0.097679
4	(DecisionTreeClassifier(max_features='sqrt', r...	500	sqrt	0.000000	0.097840
1	(DecisionTreeClassifier(max_features='log2', r...	10	log2	0.004996	0.108640
0	(DecisionTreeClassifier(max_features='sqrt', r...	10	sqrt	0.004937	0.109284

Давайте оценим точность тестового набора с помощью этой модели.

```
selected_rf = errors_df.iloc[5]['model']
```

```
test_pred = selected_rf.predict(X_test)
accuracy_score(test_target, test_pred)
```

```
0.8786427976686095
```

4.2.4 Хранение модули

```
suicide_risk_rf = {
        'model': selected_rf,
        'scaler': scaler,
        'encoder': encoder,
        'input_cols': input_cols,
        'target_col': target_col,
        'numeric_cols': numeric_cols,
        'categorical_cols': categorical_cols,
        'encoded_cols': encoded_cols
}
```

```
joblib.dump(suicide_risk_rf, 'suicide_risk_rf.joblib')
```

```
['suicide_risk_rf.joblib']
```

4.2.5 Прогнозирование на новых входах

```
suicide_risk_rf_clf = joblib.load('suicide_risk_rf.joblib')
```

```
new_input = {
        'country': 'Albania',
        'year': 2020,
        'sex': 'female',
        'age': '15-24 years',
        'suicides_no': 10,
        'population': 250000,
        'suicides/100k pop': 4,
        'HDI for year': 0.75,
        'gdp_for_year ($)': '11,000,000,000',
        'gdp_per_capita ($)': 4400,
        'generation': 'Millenials'
}
```

```
predict_input(suicide_risk_rf_clf, new_input)
```

```
('low', 1.0)
```

4.3 Сравнение

- AUC, близкий к 1: отличная модель с хорошей мерой разделительной способности
- AUC = 0,5: модель с отсутствием способности к разделению классов (=выполнение случайных догадок)
- AUC, близкий к 0: совершенно неправильная модель с худшей мерой разделимости.

Давайте сравним производительность нашего дерева решений и модели случайного леса с помощью кривой AUC-ROC. Сначала нам нужно рассчитать предсказанные вероятности 1 класса (=низкий риск самоубийства).

```
dt_probs = suicide_risk_dt_clf['model'].predict_proba(X_test)
rf_probs = suicide_risk_rf_clf['model'].predict_proba(X_test)

dt_probs = dt_probs[:, 1]
rf_probs = rf_probs[:, 1]
```

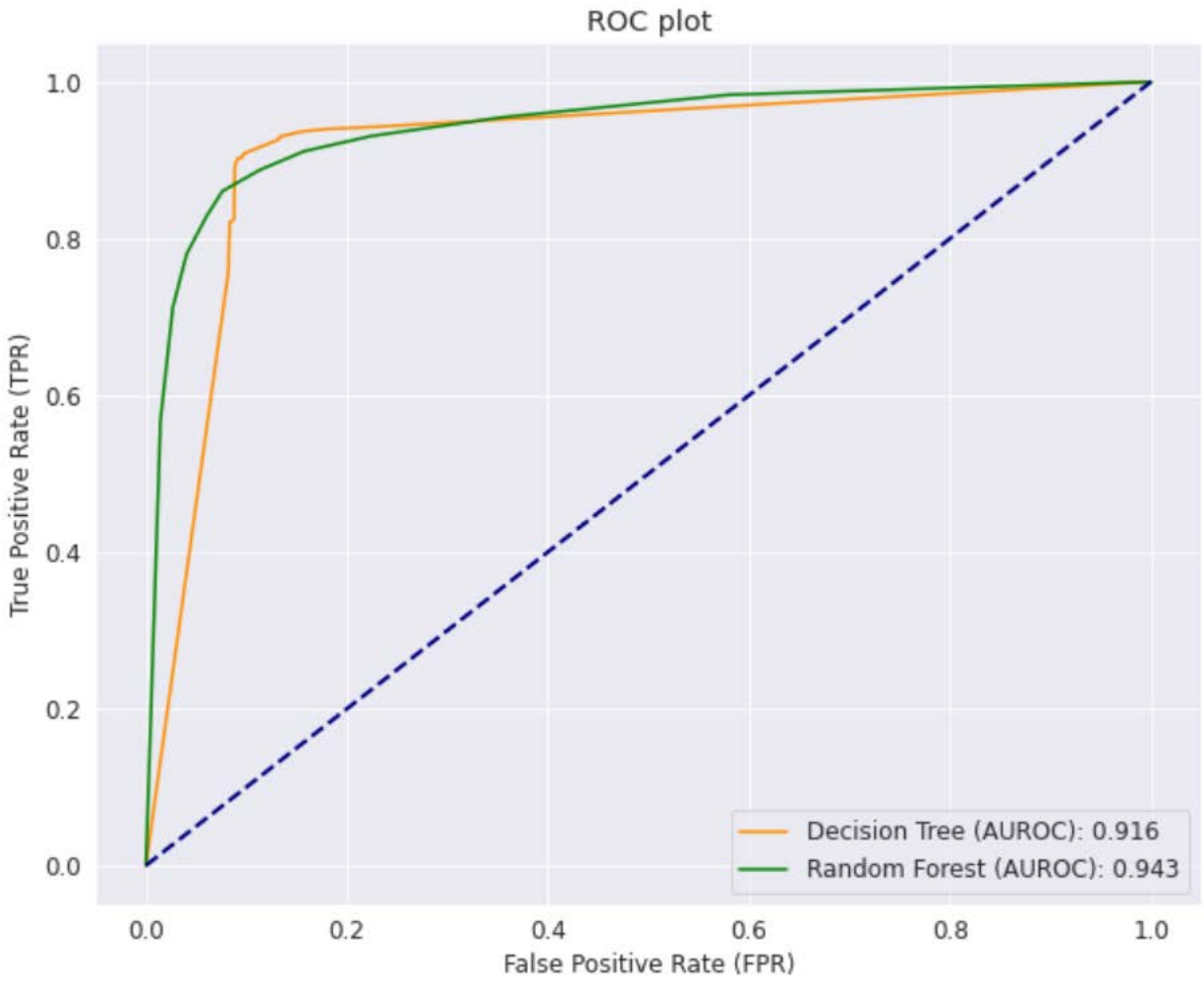
Давайте рассчитаем AUROC для обеих моделей.

```
from sklearn.metrics import roc_curve, roc_auc_score


dt_auc = roc_auc_score(test_target, dt_probs)
rf_auc = roc_auc_score(test_target, rf_probs)

print('Decision Tree (AUROC): {:.3f}'.format(dt_auc))
print('Random Forest (AUROC): {:.3f}'.format(rf_auc))
```

```
Decision Tree (AUROC): 0.916
Random Forest (AUROC): 0.943
```

Модель Random forest обладает лучшей способностью к разделению (AUROC = 0,943), чем модель дерева решений (AUROC = 0,916). Давайте также построим ROC-кривую для обеих моделей.

```
dt_fpr, dt_tpr, dt_t = roc_curve(test_target, dt_probs, pos_label='low')
rf_fpr, rf_tpr, rf_t = roc_curve(test_target, rf_probs, pos_label='low')


plt.plot(dt_fpr, dt_tpr, color='darkorange', label='Decision Tree (AUROC): {:.3f}'.format(dt_auc))
plt.plot(rf_fpr, rf_tpr, color='green', label='Random Forest (AUROC): {:.3f}'.format(rf_auc))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.title('ROC plot')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.legend(loc='lower right')
plt.show();
```



4.4 Настройка оптимального порога для модели

Попробуем найти оптимальный порог с оптимальным балансом между TPR и FPR с помощью метода, называемого 'Youden's J Statistic'.

По сути, J statistic = sensitivity + specificity - 1 = true positive rate - false positive rate.

```
J_rf = rf_tpr - rf_fpr
ix = np.argmax(J_rf)
best_thresh = rf_t[ix]
print('Best Threshold for random forest: %f' % (best_thresh))


J_dt = dt_tpr - dt_fpr
ix = np.argmax(J_dt)
best_thresh = dt_t[ix]
print('Best Threshold for decision tree: %f' % (best_thresh))
```

```
Best Threshold for random forest: 0.600000
Best Threshold for decision tree: 0.666667
```

Используйте лучший порог (вместо порога по умолчанию 0,5) для настройки наших моделей и посмотрите, повысится ли точность модели.

```
# Установите прогноз на "низкий", если оценка вероятности положительного исход
rf_test_pred = np.where(rf_probs >= 0.600, 'low', 'high')
print('Test accuracy (random forest) (threshold = 0.600): ', accuracy_score(test_target, rf_test_pred))
print('Test accuracy (random forest) (default threshold): ', accuracy_score(test_target, suicide_risk_rf_clf['model'].predict(X_test)), '\n')

dt_test_pred = np.where(dt_probs >= 0.666667, 'low', 'high')
print('Test accuracy (decision tree) (threshold = 0.667): ', accuracy_score(test_target, dt_test_pred))
print('Test accuracy (decision tree) (default threshold): ', accuracy_score(test_target, suicide_risk_dt_clf['model'].predict(X_test)))
```

```
Test accuracy (random forest) (threshold = 0.600):  0.8786427976686095
Test accuracy (random forest) (default threshold):  0.8786427976686095

Test accuracy (decision tree) (threshold = 0.667):  0.9057035803497085
Test accuracy (decision tree) (default threshold):  0.9094504579517069
```

Нет большой разницы в точности тестирования обеих моделей до и после настройки порога.

```
] raw_df['suicide_risk'].value_counts()

low     19061
high     8759
Name: suicide_risk, dtype: int64
```

5. Заключение

Мы успешно обучили дерево решений и модель случайного леса для прогнозирования риска самоубийств на основе таких факторов, как возрастная группа, пол, ИРЧП за год (составной индекс продолжительности жизни, образования и дохода на душу населения), ВВП на душу населения и страна.

Важно напомнить, что у нас нет данных о самоубийствах для большинства азиатских и африканских стран в нашем обучающем наборе данных. Следовательно, модели ML не могут быть обобщены за пределами стран, доступных в нашем наборе данных.