

## Загрузки данных и библиотек

Начнём, пожалуй, с загрузки двух основных библиотек, которые нам потребуются: `pandas` и `numpy`.

Мы импортируем, к примеру `pandas`, указывая `pd`, как псевдоним. Точно также с `numpy`, будем обращаться к ней по псевдониму `nd`.

```
import pandas as pd
import numpy as np
```

Чтобы минимизировать количество кода, запустим выполним часть заранее подготовленного кода из файла *f2forecast.py*. Это позволит нам одной командой загрузить все необходимые библиотеки и некоторые вспомогательные функции (функция графика сезонности, метрики и функция для построения набора комбинаций переменных), которые помогут нам облегчить и ускорить работу. И надо учесть, что без этого не будет работать часть методов.

```
| %run f2forecast.py
```

```
from pandas.plotting import scatter_matrix
# import time

from adtk.data import validate_data_series
from adtk.visualization import plot
from adtk.detector import ThresholdAD
from adtk.detector import OutlierDetector

from sklearn.neighbors import LocalOutlierFactor
# from sklearn.metrics import r2_score

from scipy.stats import variation

import seaborn as sns

import matplotlib.pyplot as plt

import statsmodels.api as sm
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.api import ExponentialSmoothing
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.stats.outliers_influence import variance_inflation_factor

# from dateutil.parser import parse

# import itertools
from itertools import compress, product

# import pandas as pm
from pandas.api import auto_arima

import warnings
warnings.filterwarnings("ignore") # specify to ignore warning messages
```

```
# функция графика сезонности
def sesonal(data, s):
    plt.figure(figsize=(19,8), dpi= 80)
    for i, v in enumerate(data.index.year.unique()):
        plt.plot(list(range(1,len(data[data.index.year==v])+1)), data[data.index.year==v][data.columns[0]].values, label=v)
    plt.title("Сезонность по периодам")
    plt.legend(loc="best")
    plt.show()

def metrics(real, forecast):

    if type(real)==pd.core.frame.DataFrame:
        real=real[real.columns[0]].values

    print("Тест на стационарность:")
    dftest = adfuller(real-forecast, autolag='AIC')
    print("\tТ-статистика = {:.3f}".format(dftest[0]))
    print("\tP-значение = {:.3f}".format(dftest[1]))
    print("Критические значения:")
    for k, v in dftest[4].items():
        print("\t({}): {} - Данные {} стационарны с вероятностью {}% процентов".format(k, v, "не" if v<dftest[0] else "", 100-int(v)))

    #real=np.array(real[real.columns[0]].values)
    forecast=np.array(forecast)
    print("MAO:", round(abs(real-forecast).mean(),4))
    print("MSE:", round(((real-forecast)**2).mean(),4))
    print("MAPE:", round((abs(real-forecast)/real).mean(),4))
    print("MPE:", round(((real-forecast)/real).mean(),4))
    print("Стандартная ошибка:", round(((real-forecast)**2).mean())**0.5,4))

def metrics_short(real, forecast):
    real=np.array(real[real.columns[0]].values)
    forecast=np.array(forecast)
    print("MAO:", round(abs(real-forecast).mean(),4))
    print("MSE:", round(((real-forecast)**2).mean(),4))
    print("MAPE:", round((abs(real-forecast)/real).mean(),4))
    print("MPE:", round(((real-forecast)/real).mean(),4))
    print("Стандартная ошибка:", round(((real-forecast)**2).mean())**0.5,4))
```

```
print("\t{}: {} - Данные {} стационарны с вероятностью {}% процентов".format(k, v, "не" if v<dfstest[0] else "", 100-int(k[: -1])))
```

```
def h_map(data, level):
    corr = data.corr()
    plt.figure(figsize=(14, 14))
    sns.heatmap(corr[(corr >= level) | (corr <= -level)],
                cmap="RdBu_r", vmax=1.0, vmin=-1.0, linewidths=0.1,
                annot=True, annot_kws={"size": 8}, square=True)
    plt.show()

#небольшая функция для построения набора комбинаций переменных
def combinations(items):
    return list( set(compress(items,mask)) for mask in product("[0,1]"*len(items)) )

def get_factors(data, Y, columns):

    # колонки, которые показали свою значимость в процессе отбора критериев
    # переменная spisCol хранит варианты комбинаций все переменных
    spisCol=combinations(columns)

    print('Количество комбинаций ', len(spisCol))

    #добавим константу в набор данных, нужна для расчета регрессии
    data=sm.add_constant(data)

    #сохраним в этом списке данные лучших моделей
    arr_res=[]

    #пробежимся циклом по всем вариантам комбинаций
    for c in spisCol:
        perem=list(c)
        flag=True

        if len(perem)==0: continue

        if not('const' in c):
            perem.append('const')

        # если больше одного критерия, рассчитаем VIF
        if len(perem)>1:
            vif = [variance_inflation_factor(data[perem].values, i) for i in range(data[perem].shape[1])]
        else:
            vif=[]

        # если больше одного критерия, рассчитаем VIF
        if len(perem)>1:
            vif = [variance_inflation_factor(data[perem].values, i) for i in range(data[perem].shape[1])]
        else:
            vif=[]

        #проверим список VIF, если хоть одна переменная больше 1000 (очень большое значение, на самом деле),
        #то в модели присутствует мультиколлинераность
        for vv in vif:
            if vv>1000:
                flag=False

        #рассчитаем саму модель
        reg = sm.OLS(Y, data[perem])
        res=reg.fit()

        #отбросим нулевую гипотезу для всех регрессоров конкретной модели
        for val in res.values:
            if val<2 and val>-2:
                flag=False
                break
        for val in res.pvalues:
            if val>0.05:
                flag=False
                break
        #если нулевую гипотезу отбросили и VIF в норме, сохраним результаты
        if flag:
            re=np.array(res.fittedvalues.copy())
            MSE=((np.array(Y)-re)**2).sum()/len(re)

            MAPE=(abs((np.array(Y)-re)/np.array(Y))).sum()/len(re)

            arr_res.append([round(MSE,4), res.rsquared, perem])

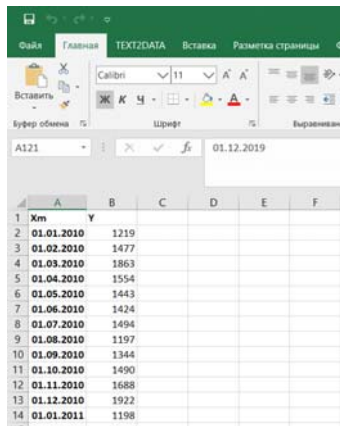
    #отсортируем и выведем результаты
    arr_res.sort()
    df_model=pd.DataFrame(arr_res, columns=['MSE', 'r2', 'Переменные'])
    print('Результаты перебора в порядке возрастания MSE:')
    print(df_model)
    return df_model
```

Теперь займёмся с загрузкой файла с данными.

Датасет связан с тарифом на перевозку зерна, его погрузку по всей железной дороги и по годам.

Здесь я переключаюсь на файл для того, чтобы показать каким образом должны быть обязательно организованы данные, чтобы не возникало проблем при работе. Должно быть 2 столбца. Столбец X<sub>m</sub>, это значение даты по порядку. Будем использовать их в качестве индекса. Если месяца, то соответственно число, месяц и год. И если кварталы, то это опять же число, месяц и год, но с интервалом раз в три квартала. Могут быть недели, но тогда ставим соответственно начала дату, начала недели и второй столбец, который мы обозначили, как у, это непосредственно те значения, которые мы будем прогнозировать. И того должно быть 2 столбца, обязательно с названиями латинских букв X<sub>m</sub> и у. X<sub>m</sub> хранит даты через равные промежутки времени, а столбец у соответственно говоря хранит сами значения, а лист может быть любым названием. Если лист

всего один, то загружать можно, не указывая название листа, если листов несколько, то мы можем указывать в явном виде с какого листа грузить.



Xm	Y
01.01.2010	1219
01.02.2010	1477
01.03.2010	1863
01.04.2010	1554
01.05.2010	1443
01.06.2010	1424
01.07.2010	1494
01.08.2010	1197
01.09.2010	1344
01.10.2010	1490
01.11.2010	1688
01.12.2010	1922
01.01.2011	1198

Когда мы грузим, мы храним таблицу в некоей переменной `df`. К `pd` обращаемся к методу библиотеки `pandas`. К методу `read_excel()`, то есть прочитать файл, где указываем как раз название файла и если вы будете грузить некий другой файл, то вот это как раз необходимо будет изменить название, указывая в обязательном порядке индекс `col`. Это как раз `Xm` столбец, в котором хранятся наши с вами даты, это необходимо для того, чтобы `excel` упорядочил непосредственно по индексу и могли обращаться к значениям по дате. Если у нас лист всего один, то мы можем не указывать название листа и загрузить просто в переменную `df`.

```
df=pd.read_excel(r'D://Магистратура//3 семестр//Арабов Муллошараф Курбонович//forecast_dataset.xlsx', index_col='Xm')
```

Если же листов несколько, то мы можем модифицировать функцию загрузки, передать ещё один параметр `sheet_name`. Параметр названия листа - `Data`.

```
df=pd.read_excel(r'D://Магистратура//3 семестр//Арабов Муллошараф Курбонович//forecast_dataset.xlsx', index_col='Xm', sheet_name='Data')
```

```
sheet_name='Data')
```

Теперь непосредственно посмотрим, изучим нашу переменную, мы можем набирать в юпитере любую переменную и запускать ячейку. Она будет пытаться вывести полностью, но так как у нас очень большое количество значений, то может вывести только начало и конец.



Xm	Y
2010-01-01	1219.0
2010-02-01	1477.0
2010-03-01	1863.0
2010-04-01	1554.0
2010-05-01	1443.0
...	...
2019-08-01	2192.3
2019-09-01	2333.4
2019-10-01	2606.7
2019-11-01	2527.5
2019-12-01	2491.0

120 rows x 1 columns

Либо можем достаточно компактно вызывать, например только первые 5 строк нашей переменной `df.head()`, либо последние 5 значений, это метод `df.tail()`. И мы с вами видим, что уже есть таблица со значениями, которые начинаются с января 2010 года и заканчиваются на декабре 2019 года.

```
df.head()
```

	Y
Xm	
2010-01-01	1219.0
2010-02-01	1477.0
2010-03-01	1863.0
2010-04-01	1554.0
2010-05-01	1443.0

```
df.tail()
```

	Y
Xm	
2019-08-01	2192.3
2019-09-01	2333.4
2019-10-01	2606.7
2019-11-01	2527.5
2019-12-01	2491.0

Теперь нам необходимо с вами волигировать, проверить на всякий случай наши значения на предмет отсутствия пропусков и корректности. Для этого у нас есть функция `validate_series` и фактически мы переприсвоим значение `df`, то есть мы говорим, что наша переменная ничто иное, как результат работы функции волигирования, той же самой переменной `df`.

```
df = validate_series(df)
```

Мы видим, что выполнение произошло без ошибок, без сообщений. Выведем результат командой `print()`. Команда `print()` выводит на экран. Переменную можно было просто вывести, выведем как командой. И вот мы с вами видим, что вот те самые наши значения, с ними всё в порядке, замечаний нет, переменные остались теми же.

```
print(df)
```

```

Y
Xm
2010-01-01    1219.0
2010-02-01    1477.0
2010-03-01    1863.0
2010-04-01    1554.0
2010-05-01    1443.0
...
2019-08-01    2192.3
2019-09-01    2333.4
2019-10-01    2606.7
2019-11-01    2527.5
2019-12-01    2491.0

[120 rows x 1 columns]
```

И ещё хочется сделать, это изучить временной ряд в виде графика. Для этого мы используем функцию `plot()`, куда передаём нашу переменную `df` и получаем график с 2010 по декабрь 2019 года. Мы видим в ряду есть колебания. Наверняка есть какие-то закономерности, но мы как раз изучим это сейчас, и эта тема будет называться «выявление корректировка аномалий». И отдельный блок будет посвящён как раз выявлению сезонности.

```
plot(df)
```

```
[<AxesSubplot:>]
```



## Очищение временного ряда от аномалий

В данном случае мы постараемся выявить ряд аномалий, скорректировать их. Это необходимо для того, чтобы уменьшить шумы на входе, чтобы было меньше мусора, то есть мы будем выбирать сами единичные несистемные позиции, которые выбиваются из общей тенденции.

То есть в первую очередь посмотрим на характеристики нашего ряда, для этого воспользуемся функцией `describe()`.

```
: df.describe()
:
```

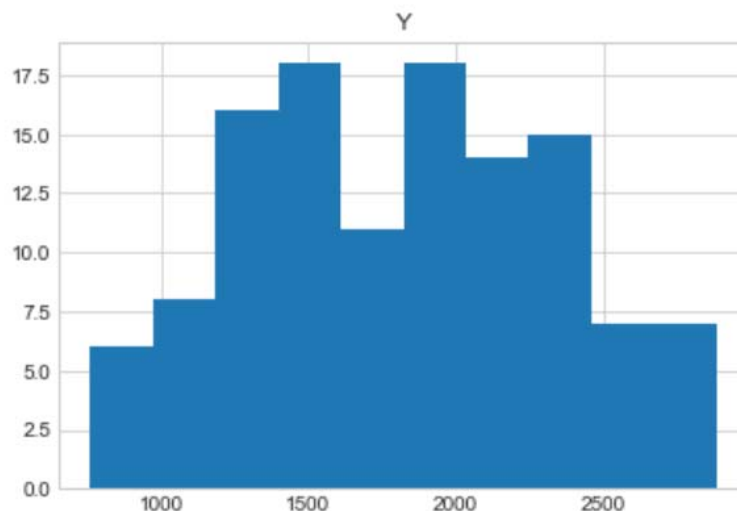
	Y
count	120.000000
mean	1804.510000
std	526.755368
min	762.000000
25%	1400.250000
50%	1825.500000
75%	2215.475000
max	2883.400000

Она позволит нам взглянуть на ключевые характеристики. `count` – количество, `mean` – среднее, `min` и `max` – соответственно минимальные и максимальные значения. В этих 2 значениях (2883 и 762) соответственно где-то в районе крайних, мы будем искать те самые выбросы. Но важно посмотреть, насколько ещё нормально распределены данные, нет ли каких-то критических отклонений?! Здесь нам поможет стандартное отклонение. Оно позволяет смотреть на ширину колокола.

Попробуем построить с вами гистограмму нашего ряда.

```
Ввод [13]: df.hist()
```

```
Out[13]: array([[<AxesSubplot:title={ 'center': 'Y' }>]], dtype=object)
```



Это частота встречаемости тех или иных значений. Например, мы видим, что значение в районе полутора тысяч – 2000 встречались чаще, чем крайние значения. В идеале должно быть колокол. В реальной жизни ровного колокола чаще всего мы не встречаем, то по крайней мере не должно быть каких-то единичных огромных отклонений, слева-справа, здесь у нас всё как раз хорошо. На подобное отклонение указывает большие различия между верхним 70% квантилем и максимальных и минимальных значений и 25% квантилем. Также важным признаком является близость медианы и среднего. Здесь они у нас совпадают.

Давайте попробуем отфильтровать значение, которое приближается к минимуму и максимуму, потому что, собственно говоря, именно по ним в первую очередь и оцениваем выбросы, то есть это всё-таки выброс с точки зрения неких максимальных и минимальных значений.

```
Ввод [65]: threshold_ad = ThresholdAD(high=2750, low=850)
anomalies = threshold_ad.detect(df)
plot(df, anomaly=anomalies, ts_linewidth=1, ts_markersize=3, anomaly_markersize=5, anomaly_color='red', anomaly_tag="marker");
```



Соответственно, давайте укажем всё что больше, например 2750, это верхний интервал и всё что, например меньше 850, даже расширим базовые границы. Попросим отобразить на графике. И вот мы с вами видим, что 2013 году у нас есть 2 точки и ещё 4 точки это 2017-2018 год, который как раз выбиваются, с одной стороны, ниже минимума и выше максимума. Скорее всего именно здесь стоит поискать аномалии, но нам важно экспертное мнение, определяется это именно по мнению эксперта. Действительно ли это события были аномальными, либо на самом деле они лежали в некой общей последовательности? Но если мы говорим про отгрузку зерна, то в 2013 году у нас был эрбанового зерна, и мы вот недогрузили его. И если мы говорим на пике, то это некий большой огромный урожай, успешно отработало сельское хозяйство как раз по зерну в 2017-2018 году, плюс стимулирование тарифов на перевозку, напомним, что мы работаем с вами с рядом, это погрузка зерна по всей железной дороге.

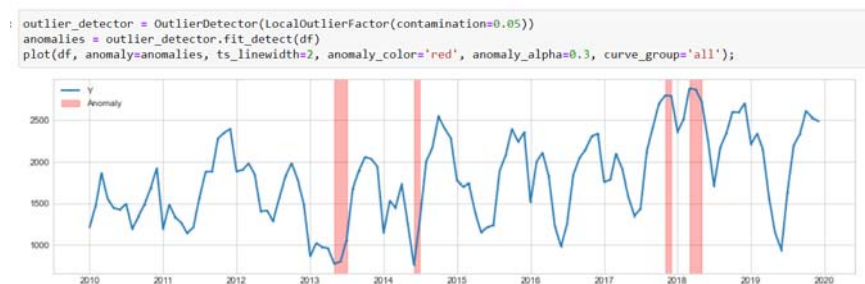
В какие периоды были отмечены аномалии?

```
anomalies[anomalies.Y]
```

	Y
Xm	
2013-05-01	True
2013-06-01	True
2014-06-01	True
2017-11-01	True
2017-12-01	True
2018-03-01	True
2018-04-01	True

Вот можно посмотреть даты, индексы, для которых были замечены аномалии, но также можно попытаться оценивать не только min и max, но и некие отклонения, связанных с именем скорости изменений выхода ему сезонности.

Здесь нам поможет немножко другой детектор, который называется outlier\_detector, он уже как раз отбирает факторы, в том что на основе кластеризации, группируя значения по кучкам максимально близким с точки зрения сезонности и значений, пытается найти те, которые выбиваются единичные из общей массы и на них этот детектор указывает, как на аномалии. Чувствительность детектора можно менять, изменяя переменную contamination. Например, поставлю одну сотую, фактически у нас не будет наблюдаться аномалий, но только две это 2013 и 2014 годы. Если же поставлю, например там две десятых, то у нас чуть-ли не треть графика будет закрашена, как аномалии. Здесь экспертной необходимо поиграться, попытаться посмотреть при каких значениях детектора с одной стороны он не закрашивает нормальное значение, а с другой стороны позволяет нам не критично посмотреть с точки зрения подобных выбросов.



Пока берём на заметку всё тот же 2013-2014 год и также 2018 и 2019, которые выбиваются из общей аномалии. Но важным тестом является на мой взгляд ещё это как раз корректировкой сезонности. У нас чаще всего бывают проблемы и когда мы строим периоды, месяца к месяцу, очень хорошо видно, как они выбиваются или не выбиваются из общей массы.

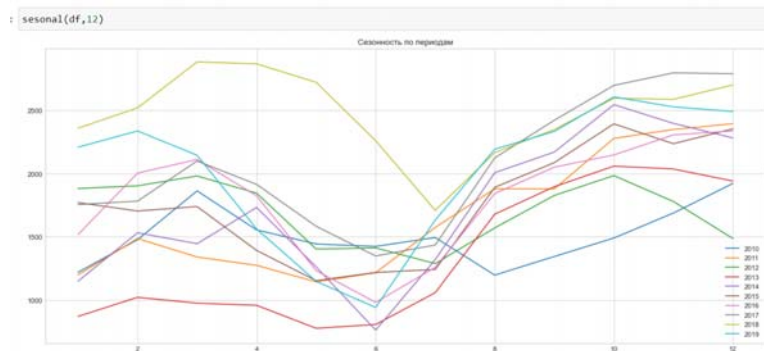
```
Ввод [18]: anomalies[anomalies]
```

```
Out[18]: Xm
2013-05-01    True
2013-06-01    True
2014-06-01    True
2017-11-01    True
2018-03-01    True
2018-04-01    True
dtype: bool
```

Есть у нас описано же функция sesonal, которая позволяет как раз построить подобный график. Если будем строить по кварталам, то измените 12 периодов на 4 и вы получите соответственно кварталный график. В данном случае у нас по месяцам, соответственно это год году. А на что

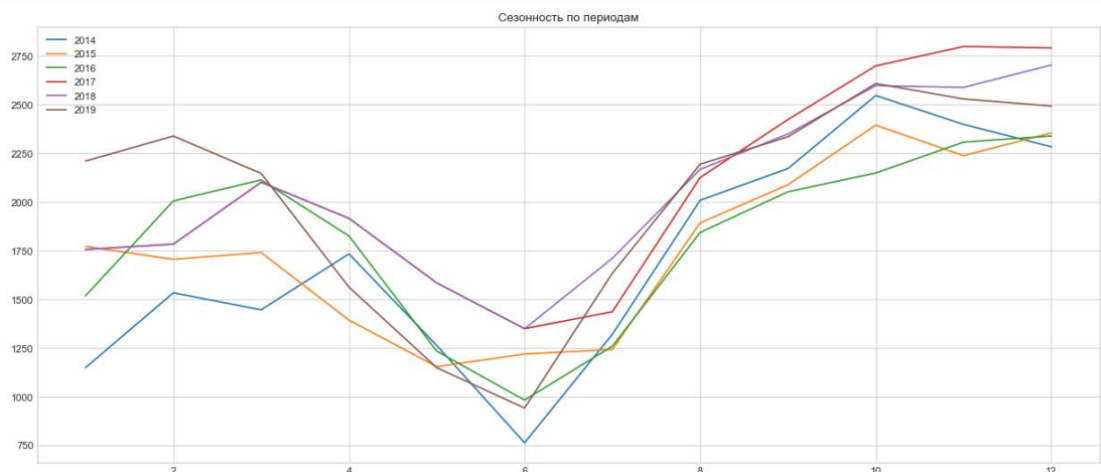


нужно обращать внимание? Из общей массы выбивается, например красный, 2013 год как раз очень плохой, низкой полугодие, наоборот, очень хорошее полугодие 2018 года. Я бы наверное заложил такую гипотезу, изучая сам ряд, что скорее всего нам стоит отбросить данные до 2014 года, как несопоставимое, но мы проверим её чуть позже, но такую гипотезу я хотел бы озвучить.



Ну например, если построю уже подобный график не по всему ряду, а начиная с 2014 года, то я получу гораздо более, наверное. Сейчас посмотрим, вменяемую картину по сезонности. Действительно, здесь необходимо нам в любом случае уже будет корректировать, на мой взгляд всё начало 2018 года выбивается из общей массы и должно вернуться где-то к 2017 году, потому что второе полугодие практически полностью совпадает, даже чуть не добираем до 2017 года, то есть он является наиболее близким. Если мы работаем со стационарными рядами, то есть у которых нет сезонности и нет тренда, то мы могли бы сглаживать, например через скользящую среднюю и выбрать 2 крайних значения, складывать между собой и делить на 2, но в сезонных рядах лучше постараться найти аналогичные периоды в прошлом. Для того чтобы увидеть аномалии, я предполагаю также взглянуть ещё и на декомпозицию ряда, где мы постараемся извлечь из него тренд и сезонность.

Ввод [77]: `sesonal(df['2014':],12)`



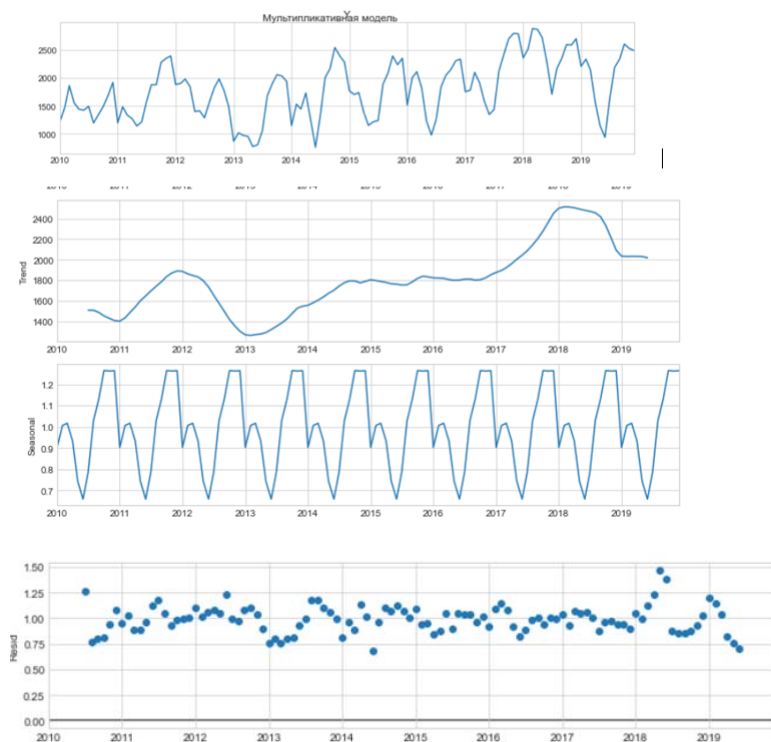
И в первую очередь сделаем это с использованием мультипликативной декомпозиции. Если будете строить по кварталам данный метод, то обратите внимание, что нам необходимо изменить частоту будет с 12 на 4 и мы используем мультипликативную составляющую. Что это значит мультипликативное? Когда у нас следующий тренд такие как этот ряд, такие как тренд сезонности и ошибка перемножаются между собой. Давайте посмотрим на наш ряд, что мы видим первый график – это, собственно говоря, основная модель, второй – тренд. Обращаю ваше внимание, что так как тренд считается концентрированной скользящей средней, то для начала и конца он не просчитан и не отрисован, но мы видим как раз с вами, что тренд очень серьезно ломается в 2013-



ом году и ошибка нарастает как раз в 2018-ом году, это вот те периоды, которые нуждаются на мой взгляд в корректировке. То есть мы ещё раз обращаем внимание на то, как ломается-не ломается тренд, то есть можем мы использовать тренд, как неизменный, либо будет несколько другим. Либо мы с вами говорим про нарастающую ошибку, то есть в данном случае мы видим, что где наибольший разброс ошибки, там скорее всего имеют место быть аномалией. Вот 2018 год он как раз у нас попадает под подобные аномалии.

```
from statsmodels.tsa.tsatools import freq_to_period
# Декомпозиция
result_M = seasonal_decompose(df.Y, model='multiplicative', period=12)

# Построение графика
plt.rcParams.update({'figure.figsize': (10,10)})
result_M.plot().supertitle('Мультипликативная модель')
Text(0.5, 0.98, 'Мультипликативная модель')
```



Попробуем взглянуть на наши коэффициенты сезонности.

```
result_M.seasonal['2011']
```

Xm	Value
2011-01-01	0.902632
2011-02-01	1.006176
2011-03-01	1.016755
2011-04-01	0.933020
2011-05-01	0.743877
2011-06-01	0.659420
2011-07-01	0.786730
2011-08-01	1.028969
2011-09-01	1.131295
2011-10-01	1.264530
2011-11-01	1.262591
2011-12-01	1.264004

```
Freq: MS, Name: seasonal, dtype: float64
```

```
Ввод [21]: result_M.seasonal['2012']
```

```
Out[21]: Xm
```

Xm	Value
2012-01-01	0.902632
2012-02-01	1.006176
2012-03-01	1.016755
2012-04-01	0.933020
2012-05-01	0.743877
2012-06-01	0.659420
2012-07-01	0.786730
2012-08-01	1.028969
2012-09-01	1.131295
2012-10-01	1.264530
2012-11-01	1.262591
2012-12-01	1.264004

```
Freq: MS, Name: seasonal, dtype: float64
```

Они будут абсолютно одинаковые для каждого из годов. Ну например коэффициент сезонности 2011 года. Можем также построить и для 2012 года. Они для данной модели абсолютно одинаковы. О чём они говорят? О том что в первом месяце года мы грузим примерно на 10% меньше среднего. И самая большая погрузка у нас получается в октябре на 26% больше среднего по году. Коэффициент сезонности ничто иное, как колебание вокруг среднего.

Но и для каких-то периодов, для кого-то года можно посмотреть значения тренда.

```
result_M.trend['2011']
```

```
Xm
2011-01-01    1402.666667
2011-02-01    1434.458333
2011-03-01    1485.208333
2011-04-01    1540.333333
2011-05-01    1600.708333
2011-06-01    1647.916667
2011-07-01    1696.083333
2011-08-01    1741.958333
2011-09-01    1786.041667
2011-10-01    1836.541667
2011-11-01    1871.166667
2011-12-01    1890.166667
Freq: MS, Name: trend, dtype: float64
```

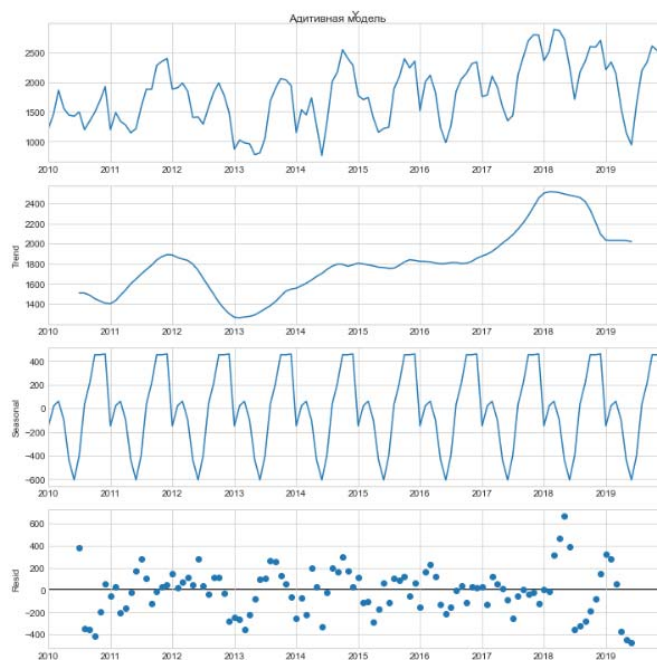
Это вот, собственно говоря, вот это наш график, что с ним происходит, как он меняется. В первую очередь, конечно, наиболее интересно коэффициенты сезонности. Почему? Потому что, например прогнозируя даже объём на год целиком по тем коэффициентам, мы потом сможем разложить его по периодам. (см на 2 график)

Можно посмотреть на модель декомпозиции аддитивную, они будут более чем похожи между собой и опять же наш основной ряд. Тренд, обратите внимание сколько он чётко совпадает и действительно есть также выброс в 2012 году, есть выбросы в 2017 году и точно также идёт разбежка по ошибкам.

```
# Decompose
result_A = seasonal_decompose(df.Y, model='additive', period=12)

# Plot
plt.rcParams.update({'figure.figsize': (10,10)})
result_A.plot().suptitle('Аддитивная модель')

Text(0.5, 0.98, 'Аддитивная модель')
```



Место, на что нужно обратить внимание здесь, что сезонные коэффициенты они уже активны, то если в мультипликативной модели мы умножаем на средний коэффициент, то здесь насколько тысяч тонн в данном случае, с тех единиц, которых, мы как раз и оцениваем, мы погружаем с вами больше.

```
result_A.seasonal['2011']
```

Xm	
2011-01-01	-149.517400
2011-02-01	20.922415
2011-03-01	59.600656
2011-04-01	-100.216474
2011-05-01	-436.639622
2011-06-01	-607.482677
2011-07-01	-399.882215
2011-08-01	33.754823
2011-09-01	213.667323
2011-10-01	453.059915
2011-11-01	452.780748
2011-12-01	459.952508

Freq: MS, Name: seasonal, dtype: float64

Аналогичным образом можно посмотреть на тренд, также будет аддитивной, то есть он достаточно похож, то есть принципы те же, но в одном случае тренд будет умножаться на коэффициент сезонности, во втором случае к тренду будет добавляться значение сезонности, либо вычитаться в зависимости от знака. И того делаем выводы, изучая наш ряд, я отбросил бы данные до 2014 года, в связи со сломом тренда и в связи с тем, что там были некоторые события, которые не повторялись в прошлом и в корректировке нуждается всё начало 2018 года, то есть мы за основу возьмём с вами как раз 2017 год, именно по аналогии подставим значения 2018 год.

```
result_A.trend['2011']
```

Xm	
2011-01-01	1402.666667
2011-02-01	1434.458333
2011-03-01	1485.208333
2011-04-01	1540.333333
2011-05-01	1600.708333
2011-06-01	1647.916667
2011-07-01	1696.083333
2011-08-01	1741.958333
2011-09-01	1786.041667
2011-10-01	1836.541667
2011-11-01	1871.166667
2011-12-01	1890.166667

Freq: MS, Name: trend, dtype: float64

Как подставить значения?

```
df.loc['2018-01-01']=df.loc['2017-01-01']
```

Ну достаточно просто. Мы говорим, что df.loc это обращение к конкретному периоду, 2018 года 01.01 = 2017 год 01.01.

И таким же образом мы присваиваем значения для остальных периодов со 2 по 6, хотя опять же нам никто не мешает присвоить период периоду, в данном случае надо будет отбросить как раз вот тот самый loc.

```

: df.loc['2018-02-01']=df.loc['2017-02-01']
df.loc['2018-03-01']=df.loc['2017-03-01']
df.loc['2018-04-01']=df.loc['2017-04-01']
df.loc['2018-05-01']=df.loc['2017-05-01']
df.loc['2018-06-01']=df.loc['2017-06-01']

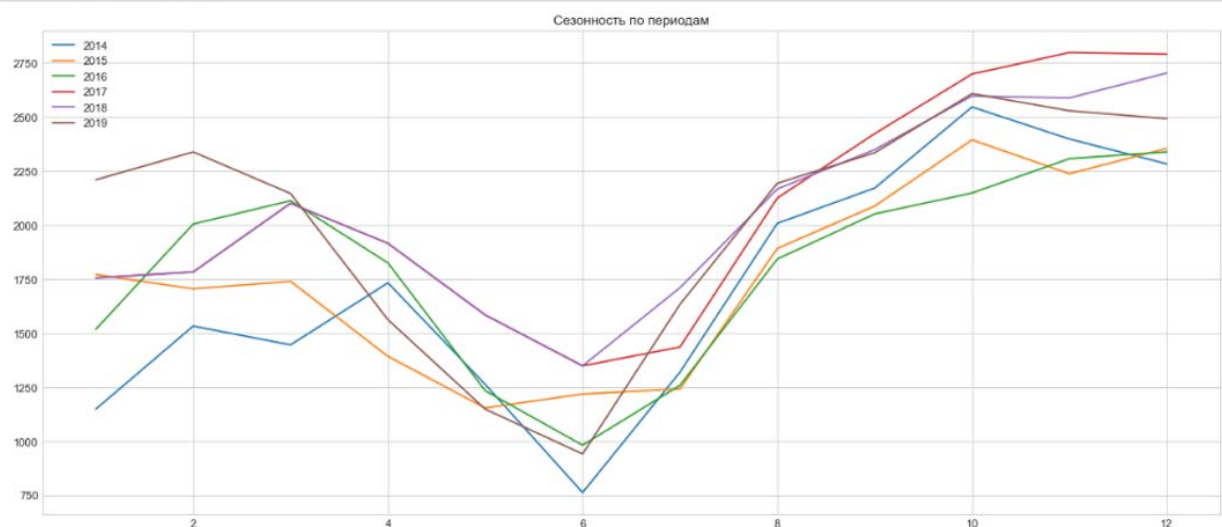
```

Давайте взглянем на получившийся результат, в первую очередь на сезонные декомпозиции. Мы с вами видим, что в целом результаты идут достаточно близко, выбивается там 2014 год, но не будем его уже корректировать, оставить как есть. В остальном там соблюдается сезонность, и мы можем использовать ряд.

```

sesonal(df['2014:'],12)

```



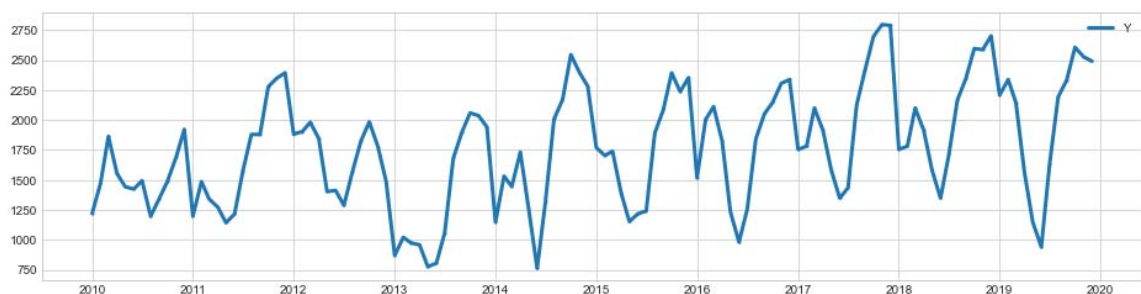
Давайте выведем его целиком, чтобы оценить после корректировки и приступим уже в начале к тому, что разделим данные на тренировочные и тестовые и дальше уже перейдем к прогнозированию методом Хольта Винтерса.

```

plot(df, ts_linewidth=3)

```

[<AxesSubplot:>]



## Тестовые и тренировочные наборы данных

Сейчас мы научимся с вами делить данные на тестовые и тренировочные. Для чего они нужны? Очень часто модель переобучается на тех данных, которые есть, она хорошо описывает данные, но обладает слабой прогностической способностью и в данном случае мы будем тестировать модель пряча от неё часть данных уже известных для нас, то есть dataset мы будем делить на тестовую и тренировочную часть, как правило, тестовая часть это некий кусочек, там до 20% от всего набора данных, на котором как раз и будем проверять нашу модель.

Давайте для этого создадим переменную train, которую мы присвоим значения нашей модели, начиная с 2014 года модели фактических данных, которые у нас есть. С 2014 по 2018 годы, то есть мы не будем использовать. Мы обсуждали это чуть раньше. Почему? Потому что несопоставимые данные с 2010 по 2013 годы, мы начинаем использовать данные с 2014 года и по ... и далее, а вот данные как раз 2019 года, 7 месяцев, которые у нас есть, мы будем с вами использовать, как тестовые и на них будем проверять, какая из моделей точнее предсказывает. Именно эту модель, которая обладает лучшей прогностической способностью, мы будем использовать. Конечно, перед применением, мы выбранную модель обучим на всём известном наборе данных и её же будем использовать как раз для прогнозирования будущих периодов.

Давайте выведем значение первых пять значений нашей модели, на нашей переменной train. Это будет наш тренировочный набор данных.

```
train=df['2014':'2018']  
train.head()
```

Y	
Xm	
2014-01-01	1148.0
2014-02-01	1532.0
2014-03-01	1445.0
2014-04-01	1732.0
2014-05-01	1262.0

И тестовый набор данных это ничто иное, как наши данные за 2019 год. Давайте опять же выведем и их посмотрим, всё ли так?!

```
test=df['2019']  
test.head()
```

Y	
Xm	
2019-01-01	2208.5
2019-02-01	2337.1
2019-03-01	2145.0
2019-04-01	1561.6
2019-05-01	1147.7

И так, с 2014 года начинается и заканчивается 2019, но первые 5 значений мы вывели. Имея на руках уже как раз тренировочные и тестовые данные, дальше мы приступаем непосредственно к прогнозированию.

## Модель Хольта Винтерса

Сейчас мы рассмотрим с вами одну из замечательных и самую популярную модель Хольта Винтерса.

Она представлена системой из четырёх уравнений, где сам прогноз ничто иное, как основной ряд данных. Новая сглаженная величина плюс тренд, умноженное на коэффициент сезонности, ничто иное, как смешанная модель, где аддитивный тренд в классическом варианте и мультипликативная сезонность. И каждый раз пересчитывается отдельно сглаженная величина, тренд и сезонность, что позволяет этой модели адаптироваться под процессы происходящее в ряду. Например, сменяется тренд и вам не надо пересчитывать всю модель, она в каких-то интервалах может сама обучаться и адаптироваться под изменение. Причём переменная сглаживания  $\alpha$ ,  $\beta$  и  $\gamma$  как раз и регулируют модель, будет модель больше ли обращать внимание на последние данные, соответственно учиться, либо наоборот не обращать внимание, оставаться стабильной, для неё более выгодно в плане точности, чем меняться под последние некие там выбросы, изменения, случайные вариации нашего ряда. На самом деле нам не надо выстраивать такую систему уравнений.

Благо уже тип библиотеки написанные и нам остаётся только обучить и натренировать модель. Сама модель будет храниться в переменной `fit1`.

```
fit1 = ExponentialSmoothing(train, seasonal_periods=12, trend='add', seasonal='mul').fit()
```

Почему модель? Потому что здесь будет и некоторые функции можем вызвать и храниться значения. `ExponentialSmoothing` это собственно говоря и есть сама наша модель, но не специальное сглаживание, а именно модель Хольта Винтерса, где мы изначально обучим её на тренировочных данных, а потом проверим уже на тестовых данных. Сезонность 12 периодов, она может находить и сама, но лучше указать, для того чтобы она не путалась, если по кварталам будем менять на 4 и трендовая составляющая – тренд аддитивный, сезонность – мультипликативная. То есть мы с вами будем строить в данном случае классическую модель, где на сезонность умножаем и тренд добавляем. Но мы видим, что можем с вами построить ещё 3 вида моделей. Как с мультипликативным трендом и сезонностью, с аддитивным трендом и сезонностью, так и наоборот, мультипликативным трендом и аддитивной сезонностью.

Давайте натренируем нашу модель.

```
fit1.params
```

```
{'smoothing_level': 0.43059111003131184,
'smoothing_trend': 0.00010012598597901177,
'smoothing_seasonal': 9.961094143486319e-05,
'damping_trend': nan,
'initial_level': 1766.812069492951,
'initial_trend': 7.322892300879126,
'initial_seasons': array([0.89297386, 0.96861345, 1.05289051, 0.96431581, 0.74762966,
0.6340379 , 0.75730707, 1.08084187, 1.19074019, 1.3261292 ,
1.31524387, 1.32590762]),
'use_boxcox': False,
'lamda': None,
'remove_bias': False}
```

Если мы захотим посмотреть на её параметры, для нас это не совсем важное, обязательное, но может быть любопытно, но мы можем увидеть с какими параметрами тренировалась, то есть как

подобрала эти вот постоянные сглаживания. Они подбираются фактически с элементами перебора, для того чтобы минимизировать среднеквадратическую ошибку. И так мы видим, что уровень сглаживания основного ряда небольшой, а тренда очень маленький, это значит, что в модели большей степени используют первоначальные значения и недообучается в дальнейшем, а вот сезонность сильно упала, в расчёт новых коэффициента сезонности почти всё даётся прошлому значению сезонности, а очень маленькую часть на новые фактическая, поэтому под сезонность наша модель обучается медленно. Стартовый уровень основных данных мы уже видим – 1766. Стартовый тренд – 7. И здесь коэффициенты сезонности, мультипликативные коэффициенты сезонности, который инициировала модель, но они каждый период будут меняться, поэтому, например, к 2018 году, коэффициент сезонности могут быть несколько другими, как раз уже на основе новых данных.

Чтобы обратиться к смоделируемым значениям, мы можем обратиться к внутренней переменной `fittedvalues`, где мы видим, что с индексом даты у нас есть значение, которое смоделировала бы модель исходя опять же из этих коэффициентов.

```
fit1.fittedvalues
```

Xm	
2014-01-01	1584.256147
2014-02-01	1521.763901
2014-03-01	1666.649017
2014-04-01	1446.063496
2014-05-01	1222.044342
2014-06-01	1055.596273
2014-07-01	1115.345600
2014-08-01	1724.885989
2014-09-01	2042.802696
2014-10-01	2345.771671
2014-11-01	2421.229120
2014-12-01	2440.050150
2015-01-01	1603.983447
2015-02-01	1824.768816
2015-03-01	1934.690806
2015-04-01	1701.866380
2015-05-01	1221.440502
2015-06-01	1015.460263
2015-07-01	1322.646905
2015-08-01	1845.418140

И если нам необходимо сделать прогноз, то мы вызываем уже внутренний метод `forecast`.

Например 12, это будет на 12 периода вперёд.

```
fit1.forecast(12)
```

2019-01-01	1809.399280
2019-02-01	1969.772892
2019-03-01	2148.827507
2019-04-01	1975.120677
2019-05-01	1536.756174
2019-06-01	1307.883068
2019-07-01	1567.727937
2019-08-01	2245.386250
2019-09-01	2482.376849
2019-10-01	2774.307727
2019-11-01	2761.138892
2019-12-01	2793.204190

Freq: MS, dtype: float64



Если на 4, то построим на 4 квартала вперёд. Правда на 4 месяца вперёд, прогноз.

```
fit1.forecast(4)
```

```
2019-01-01    1809.399280
2019-02-01    1969.772892
2019-03-01    2148.827507
2019-04-01    1975.120677
Freq: MS, dtype: float64
```

И если мы, например хотим на 24, то мы можем построить и на 24.

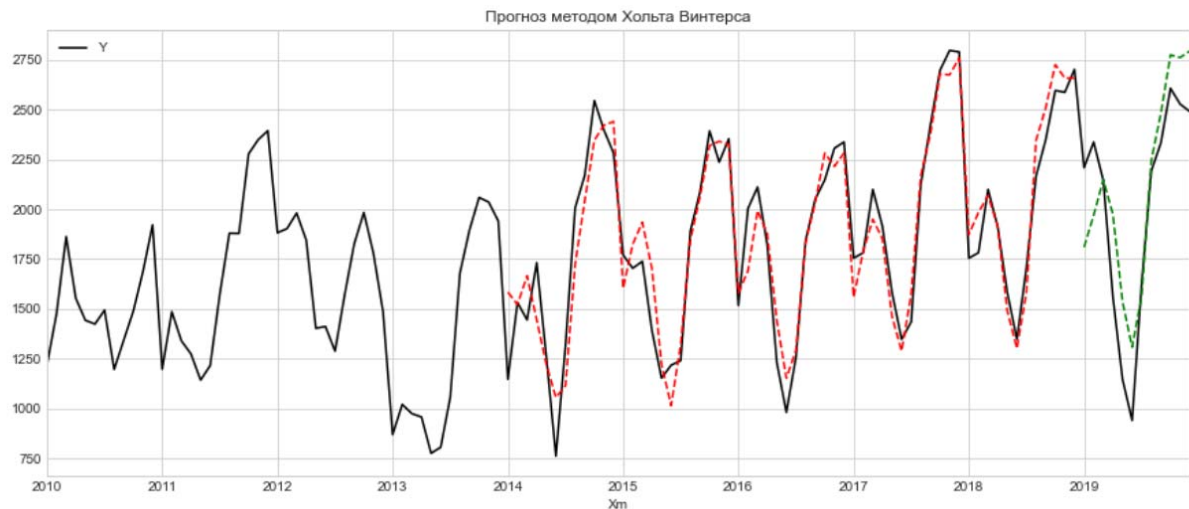
```
fit1.forecast(24)
```

```
2019-01-01    1809.399280
2019-02-01    1969.772892
2019-03-01    2148.827507
2019-04-01    1975.120677
2019-05-01    1536.756174
2019-06-01    1307.883068
2019-07-01    1567.727937
2019-08-01    2245.386250
2019-09-01    2482.376849
2019-10-01    2774.307727
2019-11-01    2761.138892
2019-12-01    2793.204190
2020-01-01    1887.668041
2020-02-01    2054.672856
2020-03-01    2241.113504
2020-04-01    2059.643959
2020-05-01    1602.286401
2020-06-01    1363.456228
2020-07-01    1634.107104
2020-08-01    2340.123873
2020-09-01    2586.746644
2020-10-01    2890.544293
2020-11-01    2876.421214
2020-12-01    2909.420940
Freq: MS, dtype: float64
```

Хотя я бы не рекомендовал модель Винтерса, для прогнозирования длинных горизонтов, потому что к сожалению она будет опираться на изначальные данные и например, если у вас тренд в конце получился сильно отрицательной, а сама модель начнёт прогнозировать не совсем адекватные значения, то есть будет большое падение на таком горизонте и наоборот, если тренд будет высоким, то модель будет переоценивать фактически значения.

Давайте попробуем отобразить результаты на графике, это не сложно, ах это собственно наш график и переменная `df.plot` это как раз отображаем основной ряд. Причём я буду весь этот ряд данных, которые есть у нас. `fit1.fittedvalues` - это данные смоделированы. Мы вот видим они красным, в данном случае обведены и мы например можем менять цвета, как менять линии стили, которыми проведены данные кривые для большей наглядности, если у вас возникает необходимость или потребность. `fit1.forecast` – это прогноз на 12 месяцев, но мы помним, что у нас только 7 месяцев, поэтому можем поменять, чтобы были совсем сопоставимые периоды и вот мы видим, что чёрное это фактический ряд, красное это смоделированное значение на тренировочных данных, а зелёное – это предсказанное значение на тестовых данных.

```
ax = df.plot(figsize=(15,6), color='black', title="Прогноз методом Хольта Винтерса" )
fit1.fittedvalues.plot(ax=ax, style='--', color='red')
fit1.forecast(12).plot(ax=ax, style='--', color='green')
plt.show()
```



Мы можем, конечно, на глаз оценить как-то модель, но хорошо или плохо получилось, то вопросы всегда в сравнении, и мы можем с вами натренировать ещё 3 модели, например в переменной fit2 мы сохраним с вами модель, которая будет, к примеру, полностью мультипликативной.

```
fit2 = ExponentialSmoothing(train, seasonal_periods=12, trend='mul', seasonal='mul').fit()
```

В переменной fit3 сохраню аддитивную модель, где аддитивный тренд и аддитивная сезонность.

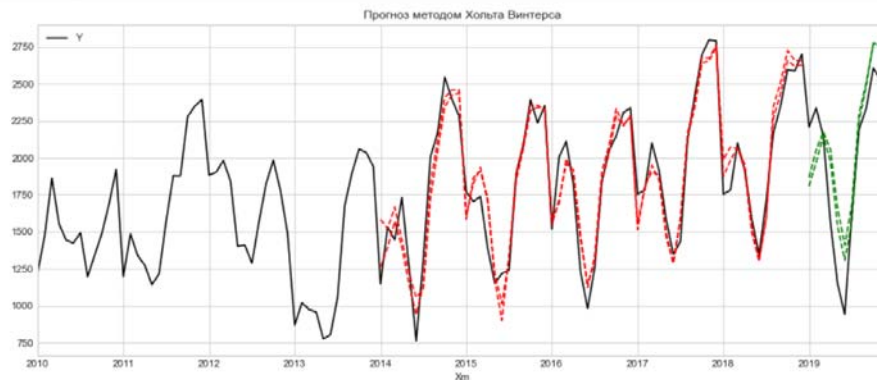
```
fit3 = ExponentialSmoothing(train, seasonal_periods=12, trend='add', seasonal='add').fit()
```

И в переменной fit4 сохраню модель обратную оригинальной, где у меня будет мультипликативный тренд и аддитивная сезонность.

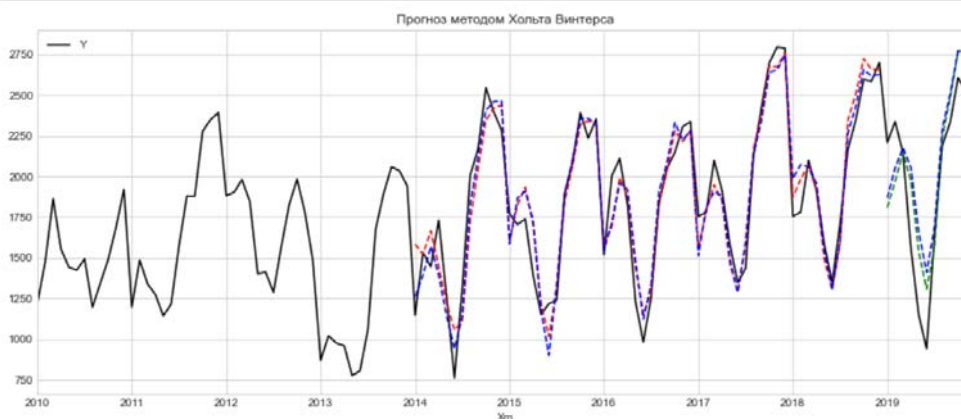
```
fit4 = ExponentialSmoothing(train, seasonal_periods=12, trend='mul', seasonal='add').fit()
```

Для каждой из них я могу построить или мог бы отобразить и даже все на графике. Ну например я могу там добавить модель, например обратную модель 4 на этот же график, для этого просто изменю название переменной, в которой буду обращаться. Вот мы видим, что очень близко ложатся модели между собой, но давайте не зелёным отобразим модель, а другим цветом, синим и прогноз для переменной 3, чтобы нагляднее было. вот можно пытаться рассмотреть кто попал точнее, кто попал хуже, вроде бы в данном случае модель 4 лучше, но я предлагаю перейти к следующему разделу и изучить ни что иное, как метрики точности, и уже изучив метрики точности, адекватностей моделей, мы с вами вернёмся и напишем точность нашей модели, выберем ту, которая лучше прогнозирует на тренировочных данных и затем ту, которая лучше прогнозирует на тестовых данных.

```
Ввод [41]: ax = df.plot(figsize=(15,6), color='black', title="Прогноз методом Хольта Винтерса" )
fit1.fittedvalues.plot(ax=ax, style='--', color='red')
fit1.forecast(12).plot(ax=ax, style='--', color='green')
fit4.fittedvalues.plot(ax=ax, style='--', color='red')
fit4.forecast(12).plot(ax=ax, style='--', color='green')
plt.show()
```



```
: ax = df.plot(figsize=(15,6), color='black', title="Прогноз методом Хольта Винтерса" )
fit1.fittedvalues.plot(ax=ax, style='--', color='red')
fit1.forecast(12).plot(ax=ax, style='--', color='green')
fit4.fittedvalues.plot(ax=ax, style='--', color='blue')
fit4.forecast(12).plot(ax=ax, style='--', color='blue')
plt.show()
```



## Стандартная метрика точности прогнозирования временных рядов

В этот раз мы изучим с вами метрики точности модели и научимся оценивать их на адекватность. Что нам важно знать для каждой модели? Но в первую очередь это MAD (среднее абсолютное отклонение) – это факт минус прогноз по модулю в сумме, делённая на количество ошибок, то есть мы говорим о том, что мы ошибаемся в среднем на столько-то тысяч тонн, килограмм, штук и т.д., в большую или в меньшую сторону, мы не знаем в какую. Но эта ошибка может быть не очень показательна. Почему? Потому что например если мы грузим в миллионных тонн, то 10000 тонн может быть крайне незначительным отклонением, если мы грузим с вами в тех десятках тысяч тонн, то 10000 тонн это крайне большая средняя абсолютная ошибка, поэтому в данном случае чаще оперируют средней абсолютной ошибкой в процентах – это Mapэ, ни что иное как факт минус прогноз по модулю, делённая на факт, и всё это в сумме делённая на количество ошибок. Тут настолько по отношению и факту мы отклоняем в большую и в меньшую сторону знака, опять же мы с вами отбрасываем и не уточняем это самая средняя плюс-минус 10% ошибка, про которую чаще всего говорят в прогнозировании. Но когда сравнивают модели между собой в первую очередь смотрят на среднеквадратическую ошибку, либо MSE, это ничто иное, как факт минус прогноз, возведённая в квадрат, то есть мы подсвечиваем, возведя в квадрат, крупные большие ошибки, делим на количество ошибок. Получается очень большая величина, сама по себе она ничего не значит, то если мы извлечём из неё корень, то получим величину близкую к

стандартному отклонению. Если помните, мы разбирались со стандартным отклонением при описании ряда, собственно говоря, мы можем перейти от точных прогнозов, к интервальным прогнозам. Они могут быть более интересны. Почему? Потому что прогнозируемая величина плюс-минус одна стандартная ошибка, либо корень из MSE, нам даст интервал, в которой попадём с 68% вероятностью, 2 ошибки в каждую сторону мы получим с вами, так называемый 95% доверительный интервал, то есть интервал, в котором попадём с вероятностью 95%. Средняя процентная ошибка, МПЕ это без модуля уже, факт минус прогноз, делённая на факт, суммируем и делим на количество ошибок. И получим таким образом некую величину, которая должна стремиться к нулю, ну или может быть как отрицательная, так и положительная. То есть, если эта величина отрицательная, то мы говорим, что у нас прогноз системно больше, чем факт и модель является переоценивающей. И наоборот, если мы получаем при положительной, то в среднем предсказываем меньше, чем есть по факту, то есть мы получили с вами недооценивающую модель. Ну и последнее, чем должны с вами разобраться, это тест на стационарность Дики-Фуллнера. О чём мы говорим? Что ошибки, которые остались, то есть прогноз минус факт, мы должны получить с вами случайную величину, то есть колебаний не должны иметь никаких закономерностей не нести никакую информацию и ни сезонность. Если этого не произошло, и информационно составляющая осталась, то есть ни белый шум — это нестационарный ряд, то мы не можем использовать модель, потому что она не смогла извлечь всю информацию из ряда и используя эту модель, можем подвести компанию под кризис, когда коэффициент на выброс модели и очень сильно будем ошибаться. Поэтому крайне важно проверять остатки на стационарность, и мы можем использовать только те модели, где остатки стационарны.

Нам не надо считать сам тест, он будет рассчитан для нас в функции, и я предлагаю перейти как раз к расчёту точности наших моделей. Теперь перейдём в наш файл, напомним, что `train` это наш тренировочный набор данных. В данном случае он чёрный, начиная с 2014 года. `fit1` хранится модели с аддитивным трендом плюс с пликативной сезонностью. `Fittedvalues` – это как раз предсказанные значения. Функция `metrics` это уже созданная описанная специальная для этого курса функция, которая посчитает основные метрики для нас с вами. Так что мы получили?

```
metrics(train, fit1.fittedvalues)
```

Тест на стационарность:  
 Т-статистика = -4.417  
 Р-значение = 0.000  
 Критические значения :  
   1%: -3.5745892596209488 - Данные стационарны с вероятностью 99% процентов  
   5%: -2.9239543084490744 - Данные стационарны с вероятностью 95% процентов  
  10%: -2.6000391840277777 - Данные стационарны с вероятностью 90% процентов  
 MAD: 119.6107  
 MSE: 22840.1586  
 MAPE: 0.0749  
 MPE: -0.0124  
 Стандартная ошибка: 151.1296

В первую очередь получили тест на стационарность. Можем почитать, что дана стационарность с вероятностью 99%, то если `t` статистика выходит за интервал плюс-минус в данном случае там 3.5, то есть меньший 3.5, то мы подтверждаем, что ряд стационарен и `p` значение должно быть меньше 500. Их в нашем случае уверенно мы утверждаем, что остатки стационарны и модели извлекла все значения из ряда. МАД это 119, это значит со средним ошибаемся с 119000 тонн, мы не знаем в большую или в меньшую сторону, средняя квадратическая ошибка 22000, но пока это ничего не значит и именно на нём будет смотреть в дальнейшем, сравнивая модели между собой по точности. MAPE примерно здесь в долях и единицах, это соответственно примерно 7,4%, то есть плюс-минус ошибаемся. Немного нехорошо, всё нужно смотреть в сравнении. МПИ пол процента мы переоцениваем, то есть предсказываем чуть больше и стандартного отклонения в 151 единиц, это значит, что если мы предскажем некое значение в полторы тысячи тонн, то факта будет лежать с вероятностью 7%, в интервале предсказанная величина плюс-минус 151. Если мы говорим про 95% интервал, то это будет полторы тысячи тонн плюс-минус уже 302. Каждую сторону единиц

мы получаем 95% интервал. И как правило интервальные ошибки более информативные, чем точные.

Давайте попробуем с вами оценить точность на тренировочных данных остальных моделей, меняя просто название переменной, можем копировать необходимое значение, и мы видим сейчас например, что модель 2 у нас получилась несколько хуже буквально чуть-чуть.

```
metrics(train, fit2.fittedvalues)

Тест на стационарность:
  Т-статистика = -4.447
  Р-значение = 0.000
Критические значения :
  1%: -3.5745892596209488 - Данные стационарны с вероятностью 99% процентов
  5%: -2.9239543084490744 - Данные стационарны с вероятностью 95% процентов
  10%: -2.6000391840277777 - Данные стационарны с вероятностью 90% процентов
MAD: 120.1126
MSE: 23081.7233
MAPE: 0.0753
MPE: -0.0147
Стандартная ошибка: 151.9267
```

Смотрю на среднее квадратическое отклонение в первую очередь. Модель 3 у нас получается лучше, чем на тренировочных данных, чем модель 1 и 2. Я вижу это по среднеквадратическому отклонению. Уменьшаются МАПИ и МЭД.

```
: metrics(train, fit3.fittedvalues)

Тест на стационарность:
  Т-статистика = -6.918
  Р-значение = 0.000
Критические значения :
  1%: -3.5463945337644063 - Данные стационарны с вероятностью 99% процентов
  5%: -2.911939409384601 - Данные стационарны с вероятностью 95% процентов
  10%: -2.5936515282964665 - Данные стационарны с вероятностью 90% процентов
MAD: 118.3745
MSE: 22156.7854
MAPE: 0.0734
MPE: -0.0116
Стандартная ошибка: 148.8516
```

Давайте взглянем на модель 4.

```
: metrics(train, fit4.fittedvalues)

Тест на стационарность:
  Т-статистика = -6.888
  Р-значение = 0.000
Критические значения :
  1%: -3.5463945337644063 - Данные стационарны с вероятностью 99% процентов
  5%: -2.911939409384601 - Данные стационарны с вероятностью 95% процентов
  10%: -2.5936515282964665 - Данные стационарны с вероятностью 90% процентов
MAD: 117.5729
MSE: 21137.0976
MAPE: 0.0717
MPE: -0.0057
Стандартная ошибка: 145.386
```

Модель 4 у нас получается лучше предыдущих. Модели в данном случае у нас получились крайне одинаковыми. И того объявляем победителем модель 4.

Но давайте попробуем посмотреть, как эти модели справляются с прогнозированием наших тестовых данных. Стиль заменим уже на ряд test и будем обращаться не к данным, которые смоделировали, а собственно говоря, строить прогноз forecast на величину длины len, собственно говоря, самого теста.

Можем сейчас показать чуть ниже, что такое len() , например test, оно вернёт нам количество периодов ряду тесту, то есть это такая автоматизация небольшая, то если у нас вдруг измениться и будет развиваться курс и увеличиться количество значений в тесте, то соответственно нам не надо будет каждый раз менять эту строку.

```
: len(test)
```

```
: 12
```

И так смотрим, что у нас получилось. Мы видим, что данные уже нестационарные, но это нормально для тестовых данных, основная модель у нас получилась стационарная и можно использовать любую из них без ограничения, если бы не так, мы бы отбросили дальше, даже не пошли с ней, мы смотрим, что нестационарное, но в общем не критично. Почему? Потому что у нас 12 значений и здесь словить наличие закономерности достаточно тяжело, гораздо интересно для нас как раз MSE, MAPE, MЭД и МПИ, то есть стандартная ошибка. Мы видим, что результаты существенно ухудшились по отношению к модели на тестовых данных, но, собственно говоря, это говорит либо об изменении условий, то есть возможные закономерности какие-то были сломаны, либо говорит о том, что модель не очень подходит.

```
: metrics(test, fit1.forecast(len(test)))
```

Тест на стационарность:

Т-статистика = -68.513

Р-значение = 0.000

Критические значения :

1%: -4.9386902332361515 - Данные стационарны с вероятностью 99% процентов

5%: -3.477582857142857 - Данные стационарны с вероятностью 95% процентов

10%: -2.8438679591836733 - Данные стационарны с вероятностью 90% процентов

MAD: 242.5334

MSE: 79558.3895

MAPE: 0.145

MPE: -0.0821

Стандартная ошибка: 282.061

Но давайте попробуем оценить тоже самое для модели 2 и 3.

```
In [392]: metrics(test, fit2.forecast(len(test)))
```

Тест на стационарность:

Т-статистика = -2.203

Р-значение = 0.205

Критические значения :

1%: -4.9386902332361515 - Данные не стационарны с вероятностью 99% процентов

5%: -3.477582857142857 - Данные не стационарны с вероятностью 95% процентов

10%: -2.8438679591836733 - Данные не стационарны с вероятностью 90% процентов

MAD: 261.4398

MSE: 88733.671

MAPE: 0.1538

MPE: -0.0947

Стандартная ошибка: 297.882

```
In [393]: metrics(test, fit3.forecast(len(test)))
```

Тест на стационарность:

Т-статистика = -1.473

Р-значение = 0.547

Критические значения :

1%: -4.9386902332361515 - Данные не стационарны с вероятностью 99% процентов

5%: -3.477582857142857 - Данные не стационарны с вероятностью 95% процентов

10%: -2.8438679591836733 - Данные не стационарны с вероятностью 90% процентов

MAD: 244.4204

MSE: 85630.1037

MAPE: 0.1545

MPE: -0.1057

Стандартная ошибка: 292.6262

Здесь меняем названия моделей. Мы видим, что модель 2 проигрывает модели 1 по среднему квадратическому отклонению. Модель 3, побеждает модель 2 и соответственно, хотя лучше было как раз у нас на тренировочных данных.



Но давайте посмотрим, что происходит с моделью 4?!

```
|: metrics(test, fit4.forecast(len(test)))
```

```
Тест на стационарность:  
Т-статистика = -1.490  
Р-значение = 0.539  
Критические значения :  
1%: -4.9386902332361515 - Данные не стационарны с вероятностью 99% процентов  
5%: -3.477582857142857 - Данные не стационарны с вероятностью 95% процентов  
10%: -2.8438679591836733 - Данные не стационарны с вероятностью 90% процентов  
MAD: 263.1262  
MSE: 94612.078  
MAPE: 0.1651  
MPE: -0.1192  
Стандартная ошибка: 307.5908
```

Модель 4 у нас получается уже хуже чем все предыдущие и вот здесь как раз интересный момент, что не всегда та модель, которая хорошо предсказывала на тестовых данных она будет так же хорошо предсказывать, вернее на хороших тренировочных данных будет так же хорошо предсказывать на тестовых данных.

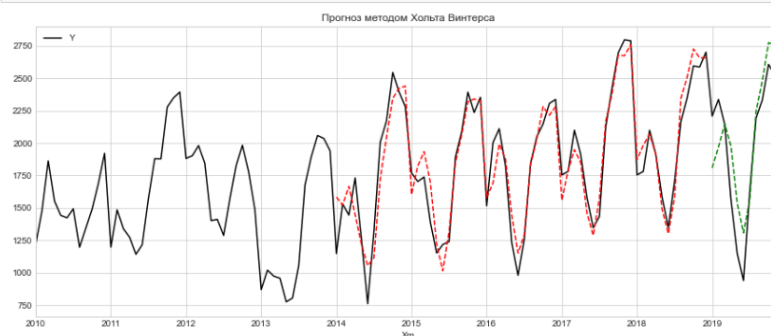
Давайте натренируем сейчас модель уже на полном ряду и лучшую модель это у нас модель с аддитивным трендом и мультипликативной сезонностью и как раз построим график и предскажем значение 2014 года, но понятно, что оценить её точность уже никак мы не сможем.

```
fit = ExponentialSmoothing(df["2014":], seasonal_periods=12, trend='add', seasonal='mul').fit()
```

Trend заменяем на df, начиная с 2014 года и до конца, мы подсказываем все данные, которые у нас есть.

Тренируем модель и давайте посмотрим код графика.

```
ax = df.plot(figsize=(15,6), color='black', title="Прогноз методом Хольта Винтерса" )  
fit1.fittedvalues.plot(ax=ax, style='--', color='red')  
fit1.forecast(12).plot(ax=ax, style='--', color='green')  
plt.show()
```



Построим уже график непосредственно для модели натренированной на полном ряду и посмотрим, как она описывает фактический ряд и спрогнозируем на 12 месяцев вперед относительно седьмого месяца уже 2019 года, которую у нас есть. Вот наша модель, которая получилась, модель 1 с аддитивными трендами мультипликативной сезонностью и зелёный это прогноз на 12 месяцев вперед относительно июля 2019 года. И если бы эта модель победила, мы пока ещё не знаем, потому что у нас впереди ещё модель Сарима, то, собственно говоря, именно этот прогноз конечно с ручными корректировками эксперта, мы брали бы в работу.

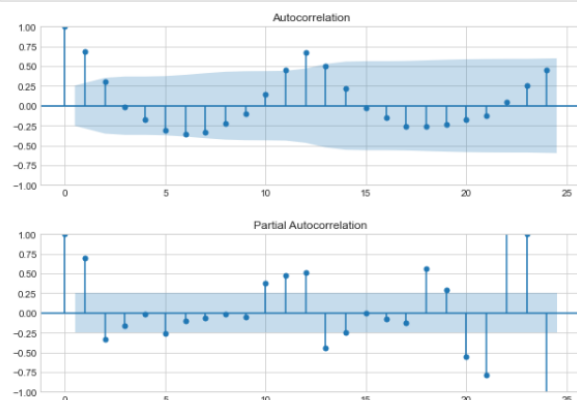


## Модель Sarima

Ещё одна популярная модель, это развитие модели Сарима и Арима. Модель Арима отличается тем, что добавлена сезонная компонента.

Модель Сарима в первую очередь подразумевает изучению вообще структур самого ряда и здесь важнейшим инструментом является автокорреляция ряда самого с собой.

```
plt.rcParams.update({'figure.figsize': (10,3)})  
plot_acf(train.Y, lags=24)  
plot_pacf(train.Y, lags=24)  
plt.show()
```



Ну вот, в частности, мы только что построили с вами как раз тафты корреляционной графики. И о чём это говорит? На самом деле величина столбика это у нас коэффициент корреляции данных нулю периода с данными периода +1, то есть грубо говоря мы сами утверждаем если видим по крайней мере по данному графику, что следующее значение имеет тесную связь с предыдущим, как правило это характеристика именно наличие в ряду тренда. Мы видели сами, что в нашем ряду содержится тренд. Таким образом, если, условно говоря, значения выхода зону значимости залито здесь синим цветом, мы говорим, что есть некая связь с отставанием настолько то периодов. Например, вот мы видим, что есть связь с отставанием на 12 периодов, то есть в нашем ряду всё-таки есть статическая значимая сезонность. И даже если вы не применяете метод Сарима, то я бы очень рекомендовал диагностировать ряд, непосредственно используя как раз вот диагностические графики автокорреляций. Ниже график автокорреляции в данном случае можно сказать такой же. Мы видим, что первой разницы у нас хорошо убирают тренд и мы видим значимые закономерности, раз в 24, 23 и 21 периодов есть значимые закономерности, может быть они как-то указывают на сезонную составляющую.

На самом деле, изучение этого графика позволяет определить ордер, то есть вот собственно говоря значение каждый из элемента модели Сарима, но как правило, достаточно тонкая и сложная работа, поэтому мы с вами её упростим и будем использовать ничто иное, как авто подбор значения нашей модели, а именно мы запускаем функцию `auto_arima` подбор.

```

: model = auto_arma(train, seasonal=True, m=12, trace=True, suppress_warnings=True, error_action='ignore', stepwise=True)
model

Performing stepwise search to minimize aic
ARIMA(2,0,2)(1,0,1)[12] intercept : AIC=853.317, Time=0.76 sec
ARIMA(0,0,0)(0,0,0)[12] intercept : AIC=916.842, Time=0.02 sec
ARIMA(1,0,0)(1,0,0)[12] intercept : AIC=835.186, Time=0.28 sec
ARIMA(0,0,1)(0,0,1)[12] intercept : AIC=865.723, Time=0.16 sec
ARIMA(0,0,0)(0,0,0)[12] intercept : AIC=1080.778, Time=0.02 sec
ARIMA(1,0,0)(0,0,0)[12] intercept : AIC=876.843, Time=0.03 sec
ARIMA(1,0,0)(2,0,0)[12] intercept : AIC=inf, Time=0.90 sec
ARIMA(1,0,0)(1,0,1)[12] intercept : AIC=833.097, Time=0.35 sec
ARIMA(1,0,0)(0,0,1)[12] intercept : AIC=859.885, Time=0.15 sec
ARIMA(1,0,0)(2,0,1)[12] intercept : AIC=inf, Time=1.07 sec
ARIMA(1,0,0)(1,0,2)[12] intercept : AIC=837.460, Time=0.94 sec
ARIMA(1,0,0)(0,0,2)[12] intercept : AIC=inf, Time=0.76 sec
ARIMA(1,0,0)(2,0,2)[12] intercept : AIC=inf, Time=1.19 sec
ARIMA(0,0,0)(1,0,1)[12] intercept : AIC=inf, Time=0.29 sec
ARIMA(2,0,0)(1,0,1)[12] intercept : AIC=845.347, Time=0.49 sec
ARIMA(1,0,1)(1,0,1)[12] intercept : AIC=848.917, Time=0.44 sec
ARIMA(0,0,1)(1,0,1)[12] intercept : AIC=842.580, Time=0.38 sec
ARIMA(2,0,1)(1,0,1)[12] intercept : AIC=inf, Time=0.45 sec
ARIMA(1,0,0)(1,0,1)[12] intercept : AIC=inf, Time=0.26 sec

Best model: ARIMA(1,0,0)(1,0,1)[12] intercept
Total fit time: 8.972 seconds

:
  ARIMA
  ARIMA(1,0,0)(1,0,1)[12] intercept

```

Мы передаём на в ход наш тренировочный набор. Мы говорим, что у нас есть сезонность, период 12 месяцев, если были бы кварталы, то мы бы указали с вами 4. Мы хотим видеть трассировку данных, подавлять сообщение об ошибках, если возникает ошибка, продолжаем игнорировать и считаем по этапу.

Давайте попробуем подобрать значение нашей модели. Так считаются параметры непосредственно нашей модели. Мы видим, что она перебирает не все подряд в данном случае, она там использует определённую логику, оптимизирует и выбирает наиболее оптимальные значения, которые необходимо посчитать. Ориентируется как раз на показатели BIC. Чем меньше они, тем лучше, тем точнее модель, то есть это фактически начисление штрафов за ошибки, в частности за большие ошибки. И так мы видим, что ордер первую на основную модель собственно говоря Арима это 1,0,0, и 1, то есть мы используем авторегрессивную часть, мы не используем сглаживание и разницы и в сезонной компоненте аналогичная история, мы используем авторегрессивную часть на глубину 2, то есть не только от последнего и на двух предыдущих значений зависит и также не используем скользящую среднюю и первой разницы.

```

mod = sm.tsa.statespace.SARIMAX(train,
                                order=(1, 0, 0),
                                seasonal_order=(2, 0, 0, 12))

results = mod.fit()

print(results.summary().tables[1])

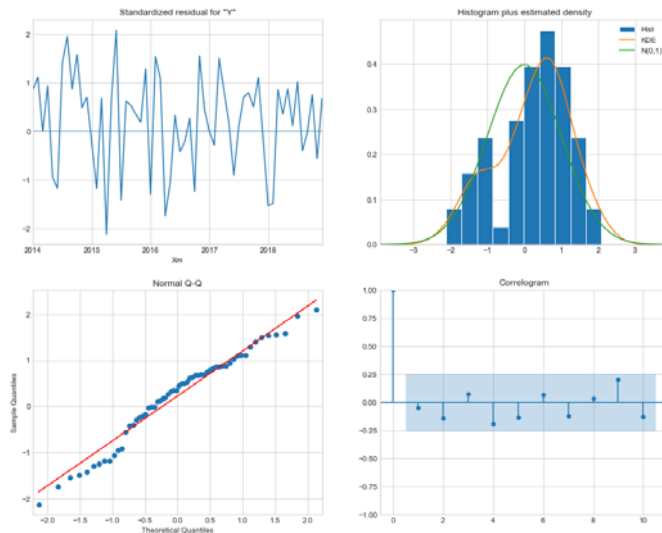
```

	coef	std err	z	P> z	[0.025	0.975]
ar.L1	0.9163	0.054	16.820	0.000	0.809	1.023
ar.S.L12	0.2815	0.082	3.449	0.001	0.122	0.441
ar.S.L24	0.6452	0.093	6.913	0.000	0.462	0.828
sigma2	3.323e+04	8915.545	3.728	0.000	1.58e+04	5.07e+04

Давайте попробуем посмотреть, что у нас получается. Если мы построим саму модель, то есть в данном случае mod переменная будет хранить модель SARIMAX с теми значениями, которые мы подобрали. Если бы у нас здесь получились другие значения чуть позже покажу, мы бы с вами использовали их, но здесь мы воспроизводим для перовой части и для сезонной части. Давайте смоделируем значение и посмотрим, что у нас с вами получается. И так мы получили с вами обученную модель, для нас с меньшей степени значимое её коэффициенты, для нас скорее важнее обученная модель, которая хранится в переменной results. Соответственно, если мы меняем order,

нам необходимо заново будут переучить всю модель. Как нам получить доступ, собственно говоря, к информации этой модели?

В первую очередь здесь есть очень интересная возможность, посмотреть по-другому на остатки данной модели.



Здесь они уже защиты дополнительной функцией, но, в частности, это стандартизированная разница, скажем так нормированная, но важнее и любопытнее это распределение ошибок. В идеале оно должно быть нормальным, а колокообразным, здесь у нас есть определённый выброс и перекося, но это ни настолько критично пока в нашем случае, где в данном случае мы ещё видим на графике опять же нормальность остатков, где скажем так, теоретическое значение должно быть ошибки, это фактически красная линия, в котором должны попасть и это те значения ошибок, которое мы попали, мы не сильно отклоняемся и мы можем утверждать, что в целом если брать меньшую сторону, чуть ошибаемся, отрицательные остатки переоцениваем как раз, они распределены нормально, а также автокорреляцию ряда можно применять не только к самому ряду, но и к остаткам. Опять же если мы не видим значимые закономерности, можем утверждать, что ряд стационарен.

Давайте посмотрим на метрики нашего ряда, но для этого нам необходимо будет получить с вами ничто иное, как предсказание модели. Давайте сохраним её, ну то есть значение модели для соответствующего ряда.

```
predict=results.get_prediction()
predict.predicted_mean[:10]
```

Xm	
2014-01-01	0.000000
2014-02-01	1098.569813
2014-03-01	1445.385324
2014-04-01	1369.491697
2014-05-01	1624.989000
2014-06-01	1212.197791
2014-07-01	781.368610
2014-08-01	1265.966585
2014-09-01	1842.852125
2014-10-01	1958.742588

Freq: MS, Name: predicted\_mean, dtype: float64

давайте сохраним в `predict=results` - это предикт переменная. `results` - это наша переменная, которая хранит уже обученную модель. `get_prediction`, то есть получаем предсказание. И давайте сразу же выведем на самом деле переменную `predict`, но надо обратиться к её отдельной составляющей, это в среднем. То есть она выводит, как и интервал может выводить, но нас интересует как раз среднее значение, точные значения предсказания. Давайте выведем не все, а

выведем интервал 10 предсказаний. Обратите внимание, что первое значение = 0, это связано с тем, что мы используем сглаживание, скользящую среднюю в том числе регрессию и не можем предсказать, значение выпадает. Соответственно, когда мы будем оценивать точность, чтобы не было перекосов, давайте будем исключать первое значение, как из данных, с которыми мы сравниваем, так и непосредственно самого предсказания.

Соответственно мы можем построить предсказание, уже начиная только с какого-то периода, то есть указав как значение predict.

```
predict=results.get_prediction(start='2014-02-01')
metrics(train['2014-02-01'], predict.predicted_mean)
```

Тест на стационарность:  
T-статистика = -5.455  
P-значение = 0.000

Критические значения :  
1%: -3.5506699942762414 - Данные стационарны с вероятностью 99% процентов  
5%: -2.913766394626147 - Данные стационарны с вероятностью 95% процентов  
10%: -2.5946240473991997 - Данные стационарны с вероятностью 90% процентов

MAD: 201.5655  
MSE: 66325.8754  
MAPE: 0.121  
MPE: 0.02  
Стандартная ошибка: 257.5381

Предскажем модель и стартуем. И здесь дальше начинается интересная вещь, мы преобразуем текстовое значение в дату, используя ещё одну функцию библиотеки pandas, то есть pandas преобразовать строку в дату и соответственно, когда будем вводить метрики, мы говорим, что нас интересует не все данные train, а нас интересует данная, начинающие со второго периода, потому что первый как раз является нулевым. Давайте посмотрим, что у нас получилось? Мы получили с вами стационарные остатки точности 99%, мы получили с вами МЭД равным двумстам одному. MSE очень высокое. Обратите внимание, что в модели Хульта Винтера лучше было 22800 с копейками, а здесь 66000 с копейками, что тут же сказывается и на всех остальных метриках нашей модели, то есть как МЭПИ мы в среднем ошибаемся, плюс-минус 12%, так и на стандартной ошибке.

Давайте попробуем сейчас предсказать и посмотреть уже как раз будущие периоды. Ну например можем предсказать там, давайте замахнёмся далеко до 2021 года и здесь мы можем использовать уже дату не, там, месяца, а именно указывать непосредственно по годам.

```
: predict=results.get_prediction(start='2019', end='2021')
```

Отобразим точно так же на графике, не путать вот это вот первое значение, которое выпадает в результатах и тут же хочу сказать, что мы можем обращаться не только с точки зрения предсказания. Мы можем обращаться к нашим результатам, через переменную точно также fittedvalues, то есть мы получаем точно такие же значения, но уже сохранённые, то есть можно через get\_prediction и можно через fittedvalues. Мы видим, что неплохо в общем описываем ряд, но промахиваться будем в том числе, скорее всего и по тестовой выборке.



Давайте попробуем посмотреть, насколько точно мы с вами умеем предсказывать уже тестовые значения, которые пока спрятаны от нашей модели и посмотрим метрику.

```
predict=results.get_prediction(start='2019-01-01', end='2019-07-01')
metrics(test[5:], predict.predicted_mean)
```

Тест на стационарность:  
 Т-статистика = -2.510  
 Р-значение = 0.113  
 Критические значения :  
 1%: -5.354256481481482 - Данные не стационарны с вероятностью 99% процентов  
 5%: -3.6462381481481483 - Данные не стационарны с вероятностью 95% процентов  
 10%: -2.901197777777778 - Данные не стационарны с вероятностью 90% процентов  
 MAD: 647.9496  
 MSE: 559683.2623  
 MAPE: 0.3697  
 MPE: -0.0244  
 Стандартная ошибка: 748.1198

И так мы строим прогноз 2019 года с 1 по 7 месяц и сравниваем с тестовыми данными. В данном случае мы получаем среднеквадратическую ошибку - 559000. И если сравнивать из всех моделей, которые мы построили, я бы порекомендовал конечно же использовать модель классическую, Хольта Винтерса, на тестовых данных он даёт лучший результат. Но это не значит, что мы не можем улучшить наш прогноз.

Давайте попробуем модель изменить и например добавить сюда ещё 1 параметр - train.

```
In[66]: del = auto_arima(train, seasonal=True, m=12, trace=True, suppress_warnings=True, error_action='ignore', stepwise=True, trend='t')
del
```

Performing stepwise search to minimize aic

Model	AIC	Time
ARIMA(2,0,2)(1,0,1)[12] Intercept	AIC=inf	Time=0.83 sec
ARIMA(0,0,0)(0,0,0)[12] Intercept	AIC=990.835	Time=0.83 sec
ARIMA(1,0,0)(1,0,0)[12] Intercept	AIC=845.799	Time=0.26 sec
ARIMA(0,0,1)(0,0,1)[12] Intercept	AIC=inf	Time=0.30 sec
ARIMA(0,0,0)(0,0,0)[12]	AIC=990.835	Time=0.83 sec
ARIMA(1,0,0)(0,0,0)[12] Intercept	AIC=883.833	Time=0.83 sec
ARIMA(1,0,0)(2,0,0)[12] Intercept	AIC=inf	Time=1.06 sec
ARIMA(1,0,0)(1,0,1)[12] Intercept	AIC=818.533	Time=0.45 sec
ARIMA(1,0,0)(0,0,1)[12] Intercept	AIC=868.488	Time=0.15 sec
ARIMA(1,0,0)(2,0,1)[12] Intercept	AIC=833.545	Time=1.22 sec
ARIMA(1,0,0)(1,0,2)[12] Intercept	AIC=829.153	Time=1.07 sec
ARIMA(1,0,0)(0,0,2)[12] Intercept	AIC=inf	Time=0.75 sec
ARIMA(1,0,0)(2,0,2)[12] Intercept	AIC=832.836	Time=1.19 sec
ARIMA(0,0,0)(1,0,1)[12] Intercept	AIC=831.142	Time=0.44 sec
ARIMA(2,0,0)(1,0,1)[12] Intercept	AIC=820.465	Time=0.40 sec
ARIMA(1,0,1)(1,0,1)[12] Intercept	AIC=820.471	Time=0.35 sec
ARIMA(0,0,1)(1,0,1)[12] Intercept	AIC=857.011	Time=0.29 sec
ARIMA(2,0,1)(1,0,1)[12] Intercept	AIC=inf	Time=nan sec
ARIMA(1,0,0)(1,0,1)[12]	AIC=818.533	Time=0.28 sec

Best model: ARIMA(1,0,0)(1,0,1)[12]  
 Total fit time: 9.653 seconds

```
Out[66]: ARIMA(1,0,0)(1,0,1)[12]
```

Он у нас не указан, то есть если посмотрите на подобранные модели, у нас идёт там none тренд, но это не совсем так. Тренд можно использовать через всякие разные сглаживания, но можно промоделировать отдельно, например, тренд мы будем использовать линейный, он обладает лучшей предсказательной способностью. Тренд кодируется буквой, чтоб всё помещалось,

перенесём буквой t, это как раз наш линейный тренд и подбора счёт модели занимает какое-то время. Через 9 секунд, добавление тренда резко осложняет расчёты, мы смогли получить уже немножко другую модель. Она учитывала, что есть линейный тренд, несколько изменился, брать внимание именно сезонный ордер. Основной ордер остался таким же.

И давайте попробуем поработать уже непосредственно с нашей моделью с новым ордер, для этого мы меняем ордер сезонный на тот, который приведён в подборе модели.

```
mod = sm.tsa.statespace.SARIMAX(train,
                                order=(1, 0, 0),
                                seasonal_order=(1, 0, 1, 12))

results = mod.fit()

print(results.summary().tables[1])
```

	coef	std err	z	P> z	[0.025	0.975]
ar.L1	0.9220	0.064	14.320	0.000	0.796	1.048
ar.S.L12	0.9984	0.007	144.961	0.000	0.985	1.012
ma.S.L12	-0.8894	0.237	-3.758	0.000	-1.353	-0.426
sigma2	3.14e+04	7.82e+06	4.02e+09	0.000	3.14e+04	3.14e+04

И давайте попробуем посмотреть и на наши остатки, то есть, но не сильно изменилась, хотя чуть-чуть стала лучше выглядеть, хотя не факт. Распределения остатков в целом примерно так же, коррелограмма показывает, что остатки случайные.

```
predict=results.get_prediction(start='2014-02-01')
metrics(train['2014-02-01':], predict.predicted_mean)
```

Тест на стационарность:  
T-статистика = -5.560  
P-значение = 0.000

Критические значения :  
1%: -3.5506699942762414 - Данные стационарны с вероятностью 99% процентов  
5%: -2.913766394626147 - Данные стационарны с вероятностью 95% процентов  
10%: -2.5946240473991997 - Данные стационарны с вероятностью 90% процентов

MAD: 199.9203  
MSE: 67230.3984  
MAPE: 0.1202  
MPE: 0.0207  
Стандартная ошибка: 259.2883

Давайте посмотрим на показатели точности самой модели. И мы видим, что на тренировочных данных она не сильно показывает выше точно 67000, у нас была здесь 66325, то есть даже стала хуже точность нашей модели.

Но для чистоты эксперимента давайте проверим наши результаты непосредственно на тестовой выборке, и мы получаем с вами несколько лучший, существенно лучший результат по отношению к Ариме без трендовой компоненты, но тем не менее эта модель уступает модели Хольта Винтерса. Именно модель Хольта Винтерса мы берём за основу по результатам нашего моделирования на данном примере.