Задание: Выберите набор данных (датасет) для решения задачи прогнозирования временного ряда. Визуализируйте временной ряд и его основные характеристики. Разделите временной ряд на обучающую и тестовую выборку. Произведите прогнозирование временного ряда с использованием как минимум двух методов. Визуализируйте тестовую выборку и каждый из прогнозов. Оцените качество прогноза в каждом случае с помощью метрик.

In [ ]:

```
import numpy as np
import pandas as pd
from matplotlib import pyplot
import matplotlib.pyplot as plt
```

In [ ]:

```
from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

In [ ]:

```
ts_fb = pd.read_csv('/content/drive/MyDrive/rice_wheat_corn_prices.csv', header=0, index_col=0, parse_dates=True,)
```

In [ ]:

```
ts_fb
```

Out[ ]:

	Price_wheat_ton	Price_rice_ton	Price_corn_ton	Inflation_rate	Price_wheat_ton_infl	Price_rice_ton_infl	Price_corn_ton_infl
Year							
1992-02-01	170.12	278.25	113.62	89.59	322.53	527.53	215.41
1992-03-01	161.44	277.20	117.00	89.59	306.07	525.54	221.82
1992-04-01	153.07	278.00	108.52	89.59	290.21	527.06	205.74
1992-05-01	139.72	274.00	109.64	89.59	264.90	519.48	207.87
1992-06-01	140.36	268.80	110.90	89.59	266.11	509.62	210.26
...	...	...	...	...	...	...	...
2021-08-01	276.18	403.00	256.61	-1.29	272.62	397.80	253.30
2021-09-01	263.60	400.00	235.62	-1.29	260.20	394.84	232.58
2021-10-01	334.50	401.00	239.65	-1.29	330.18	395.83	236.56
2021-11-01	327.82	400.00	248.72	-1.29	323.59	394.84	245.51
2021-12-01	332.06	400.00	264.54	-1.29	327.78	394.84	261.13

359 rows × 7 columns

In [ ]:

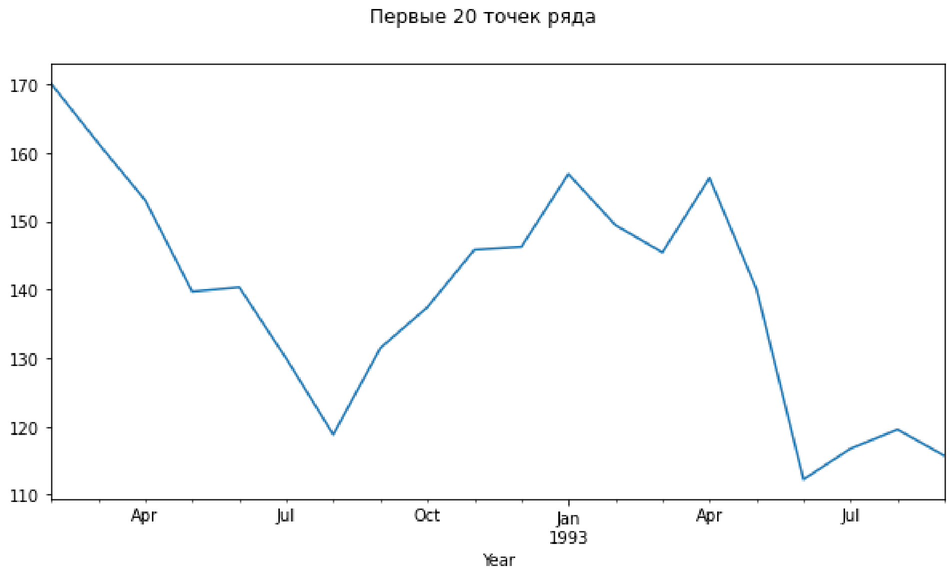
```
ts_fb=ts_fb[['Price_wheat_ton']]
```

In [ ]:

```
fig, ax = pyplot.subplots(1, 1, sharex='col', sharey='row', figsize=(10,5))
fig.suptitle('Временной ряд в виде графика')
ts_fb.plot(ax=ax, legend=True)
pyplot.show()
```

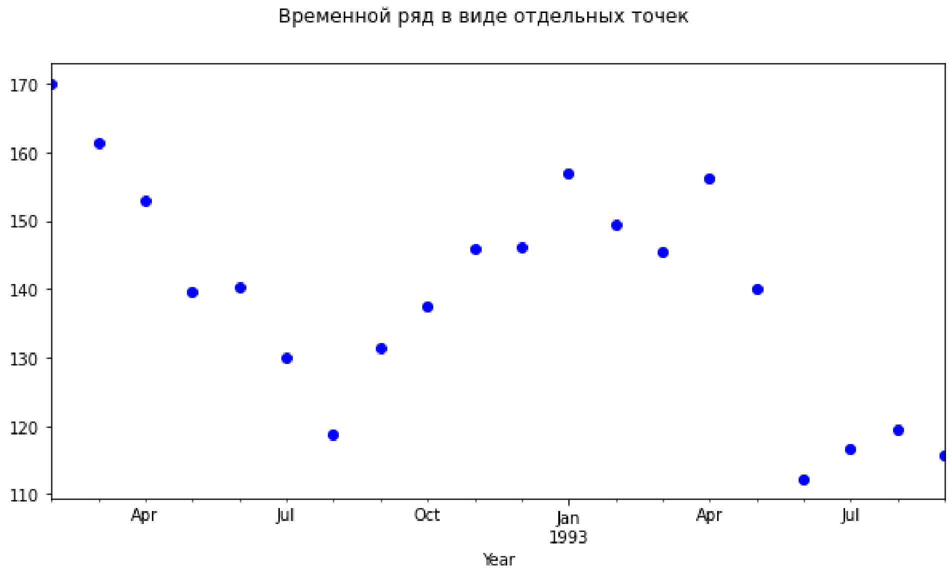
In [ ]:

```
fig, ax = pyplot.subplots(1, 1, sharex='col', sharey='row', figsize=(10,5))
fig.suptitle('Первые 20 точек ряда')
ts_fb[:20].plot(ax=ax, legend=False)
pyplot.show()
```



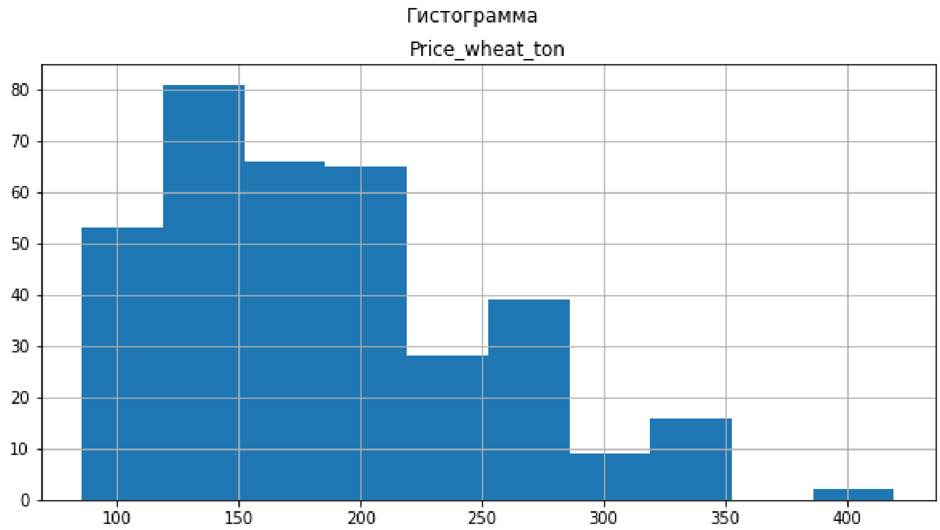
In [ ]:

```
fig, ax = pyplot.subplots(1, 1, sharex='col', sharey='row', figsize=(10,5))
fig.suptitle('Временной ряд в виде отдельных точек')
ts_fb[:20].plot(ax=ax, legend=False, style='bo')
pyplot.show()
```



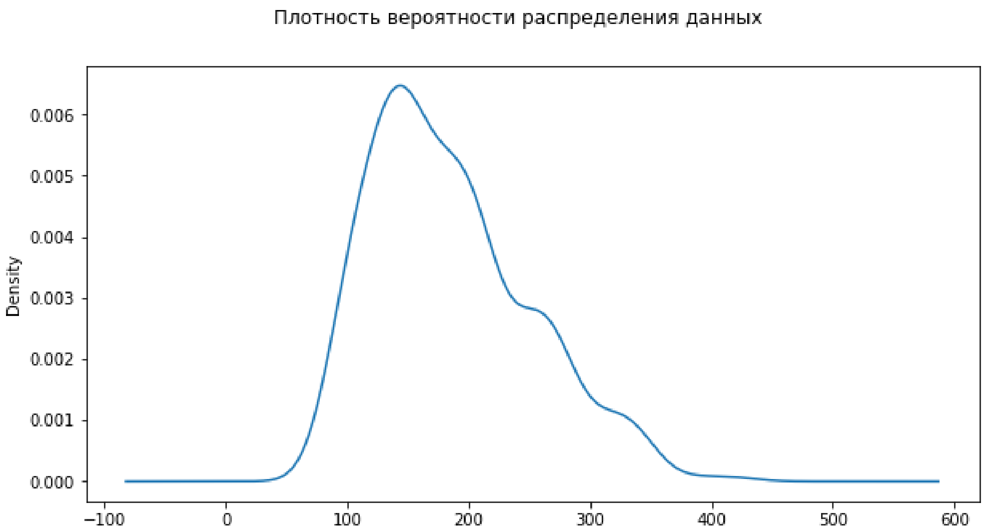
In [ ]:

```
fig, ax = pyplot.subplots(1, 1, sharex='col', sharey='row', figsize=(10,5))
fig.suptitle('Гистограмма')
ts_fb.hist(ax=ax, legend=False)
pyplot.show()
```



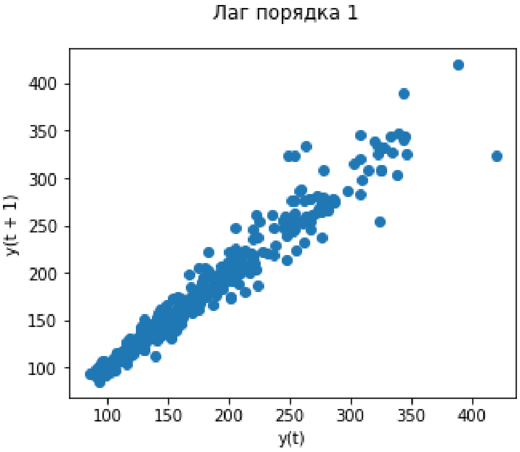
In [ ]:

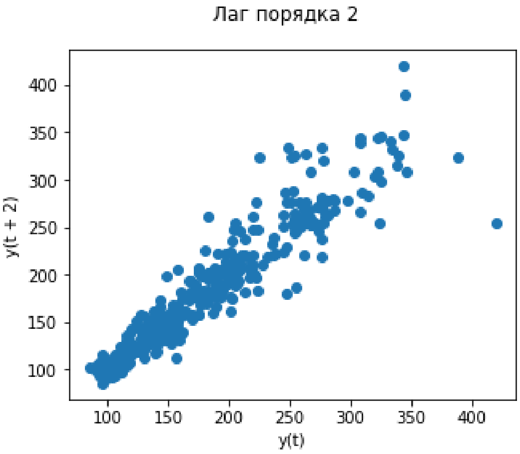
```
fig, ax = pyplot.subplots(1, 1, sharex='col', sharey='row', figsize=(10,5))
fig.suptitle('Плотность вероятности распределения данных')
ts_fb.plot(ax=ax, kind='kde', legend=False)
pyplot.show()
```

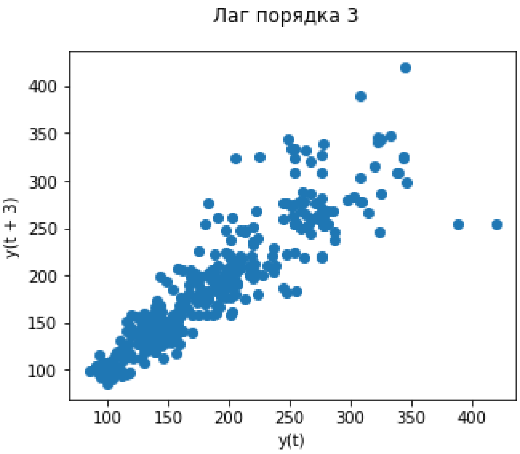


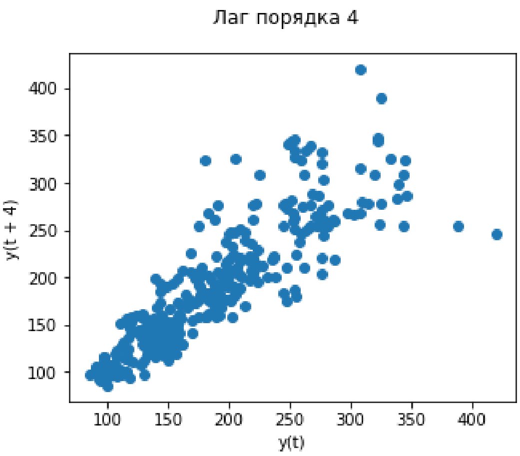
In [ ]:

```
for i in range(1, 5):
    fig, ax = pyplot.subplots(1, 1, sharex='col', sharey='row', figsize=(5,4))
    fig.suptitle(f'Лаг порядка {i}')
    pd.plotting.lag_plot(ts_fb, lag=i, ax=ax)
    pyplot.show()
```









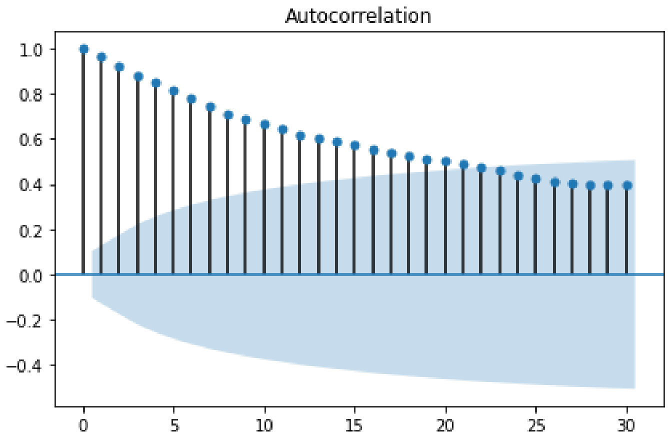
In [ ]:

```
# По оси Y откладывается ковариация
# https://stats.stackexchange.com/questions/357300/what-does-pandas-autocorrelation-graph-show
fig, ax = pyplot.subplots(1, 1, sharex='col', sharey='row', figsize=(10,5))
fig.suptitle('Автокорреляционная диаграмма')
pd.plotting.autocorrelation_plot(ts_fb, ax=ax)
pyplot.show()
```

In [ ]:

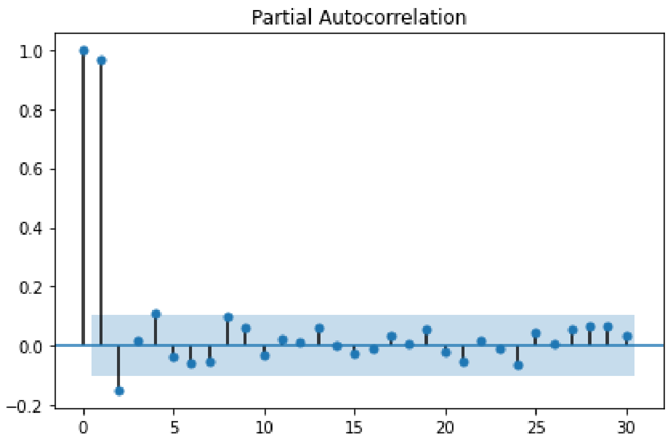
```
from statsmodels.graphics.tsaplots import plot_acf
plot_acf(ts_fb, lags=30)
plt.tight_layout()
```

/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprecated. Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm



In [ ]:

```
from statsmodels.graphics.tsaplots import plot_pacf
plot_pacf(ts_fb, lags=30)
plt.tight_layout()
```

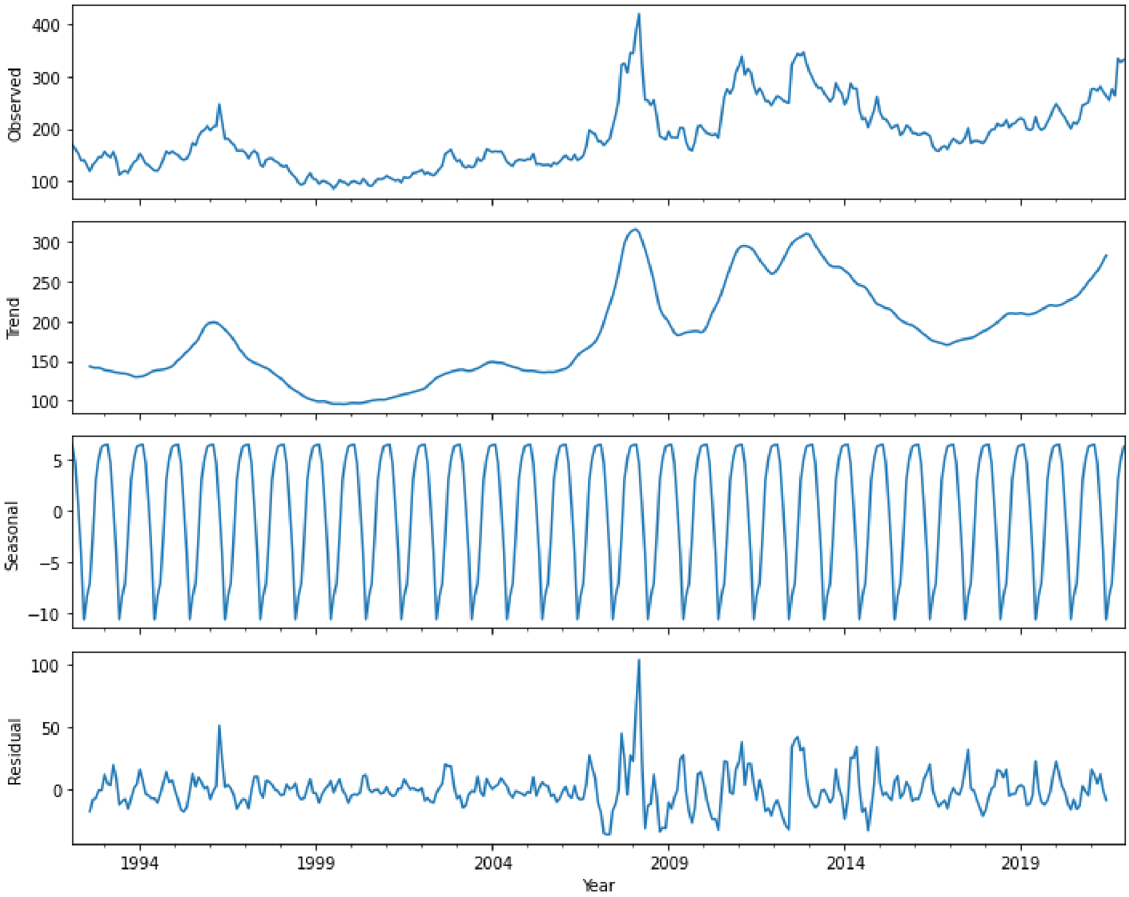


In [ ]:

```
# https://www.statsmodels.org/dev/generated/statsmodels.tsa.seasonal.seasonal_decompose.html
from statsmodels.tsa.seasonal import seasonal_decompose
# Аддитивная модель
def plot_decompose(data=ts_fb['Price_wheat_ton'], model='add'):
    result_add = seasonal_decompose(data, model = 'add')
    fig = result_add.plot()
    fig.set_size_inches((10, 8))
    # Перерисовка
    fig.tight_layout()
    plt.show()
```

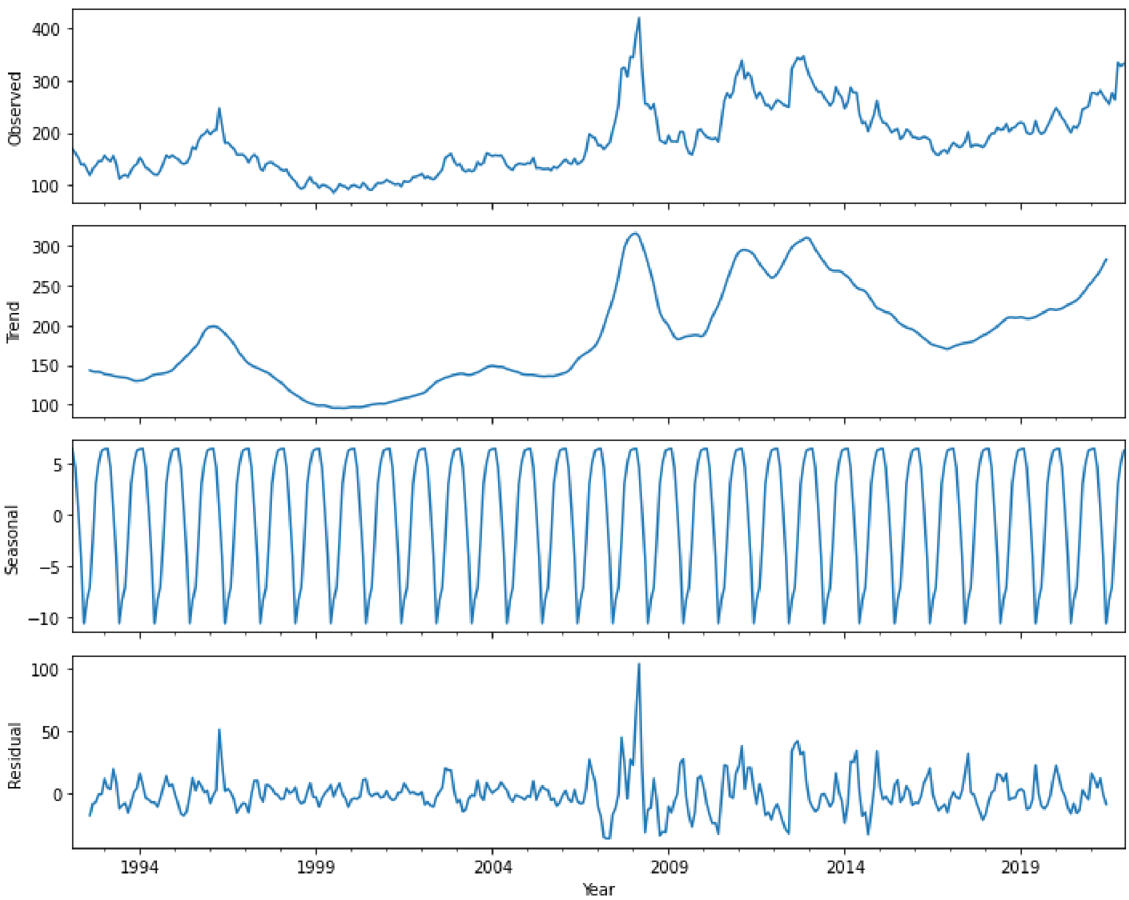
In [ ]:

```
plot_decompose(data=ts_fb['Price_wheat_ton'], model='add')
```



In [ ]:

```
plot_decompose(data=ts_fb['Price_wheat_ton'], model='mul')
```



In [ ]:

```
ts_fb2 = ts_fb.copy()
```

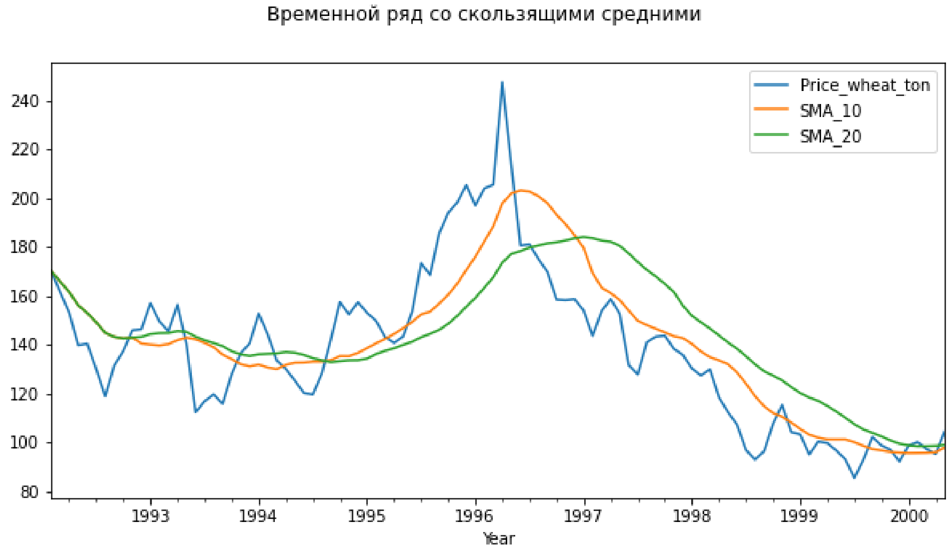
In [ ]:

```
# Простое скользящее среднее (SMA)
```

```
ts_fb2['SMA_10'] = ts_fb2['Price_wheat_ton'].rolling(10, min_periods=1).mean()
ts_fb2['SMA_20'] = ts_fb2['Price_wheat_ton'].rolling(20, min_periods=1).mean()
```

In [ ]:

```
fig, ax = pyplot.subplots(1, 1, sharex='col', sharey='row', figsize=(10,5))
fig.suptitle('Временной ряд со скользящими средними')
ts_fb2[:100].plot(ax=ax, legend=True)
pyplot.show()
```



In [ ]:

```
from sklearn.metrics import mean_squared_error
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from statsmodels.tsa.arima_model import ARIMA
```

In [ ]:

```
# Целочисленная метка шкалы времени
xnum = list(range(ts_fb2.shape[0]))
# Разделение выборки на обучающую и тестовую
Y = ts_fb2['Price_wheat_ton'].values
train_size = int(len(Y) * 0.7)
xnum_train, xnum_test = xnum[0:train_size], xnum[train_size:]
train, test = Y[0:train_size], Y[train_size:]
history_arima = [x for x in train]
history_es = [x for x in train]
```

In [ ]:

```
# Параметры модели (p,d,q)
arima_order = (6,1,0)
# Формирование предсказаний
predictions_arima = list()
for t in range(len(test)):
    model_arima = ARIMA(history_arima, order=arima_order)
    model_arima_fit = model_arima.fit()
    yhat_arima = model_arima_fit.forecast()[0]
    predictions_arima.append(yhat_arima)
    history_arima.append(test[t])
# Вычисление метрики RMSE
error_arima = mean_squared_error(test, predictions_arima, squared=False)
```

In [ ]:

```
# Формирование предсказаний
predictions_es = list()
for t in range(len(test)):
    model_es = ExponentialSmoothing(history_es)
    model_es_fit = model_es.fit()
    yhat_es = model_es_fit.forecast()[0]
    predictions_es.append(yhat_es)
    history_es.append(test[t])
# Вычисление метрики RMSE
error_es = mean_squared_error(test, predictions_es, squared=False)
```

In [ ]:

```
# Ошибка прогноза
np.mean(Y), error_arima, error_es
```

Out[ ]:

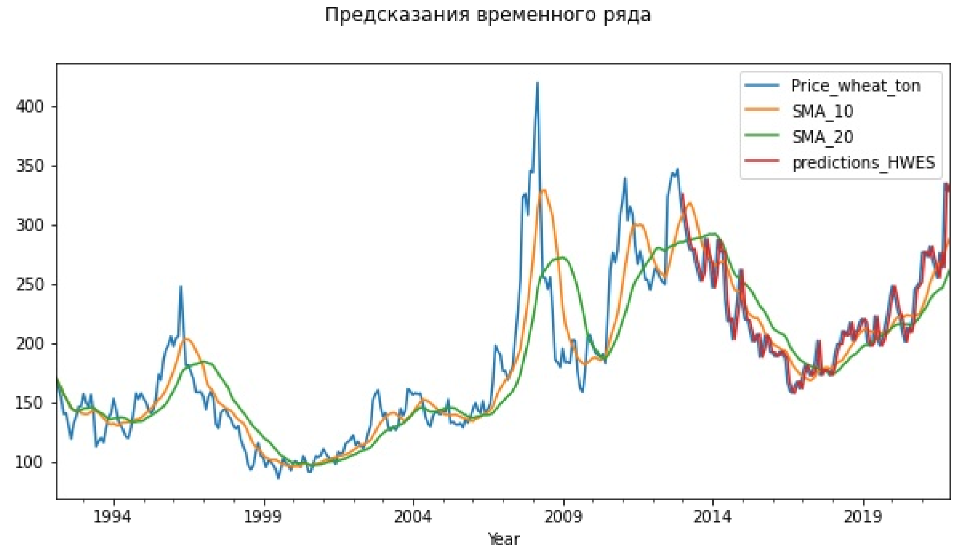
(185.30286908077994, 14.94264993683283, 14.16217857011596)

In [ ]:

```
# Записываем предсказания в DataFrame
ts_fb2['predictions_ARIMA'] = (train_size * [np.NAN]) + list(predictions_arima)
ts_fb2['predictions_HWES'] = (train_size * [np.NAN]) + list(predictions_es)
```

In [ ]:

```
fig, ax = pyplot.subplots(1, 1, sharex='col', sharey='row', figsize=(10,5))
fig.suptitle('Предсказания временного ряда')
ts_fb2.plot(ax=ax, legend=True)
pyplot.show()
```



In [ ]:

```
fig, ax = pyplot.subplots(1, 1, sharex='col', sharey='row', figsize=(10,5))
fig.suptitle('Предсказания временного ряда (тестовая выборка)')
ts_fb2[train_size:].plot(ax=ax, legend=True)
pyplot.show()
```

In [ ]:

```
!pip install gplearn
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting gplearn
  Downloading gplearn-0.4.2-py3-none-any.whl (25 kB)
Requirement already satisfied: joblib>=1.0.0 in /usr/local/lib/python3.7/dist-packages (from gplearn) (1.1.0)
Requirement already satisfied: scikit-learn>=1.0.2 in /usr/local/lib/python3.7/dist-packages (from gplearn) (1.0.2)
Requirement already satisfied: numpy>=1.14.6 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=1.0.2->gplearn) (1.21.6)
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=1.0.2->gplearn) (1.4.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=1.0.2->gplearn) (3.1.0)
Installing collected packages: gplearn
Successfully installed gplearn-0.4.2
```

In [ ]:

```
from gplearn.genetic import SymbolicRegressor
```

In [ ]:

```
function_set = ['add', 'sub', 'mul', 'div', 'sin']
est_gp = SymbolicRegressor(population_size=500, metric='mse',
                          generations=70, stopping_criteria=0.01,
                          init_depth=(4, 10), verbose=1, function_set=function_set,
                          const_range=(-100, 100), random_state=0)
```

In [ ]:

```
est_gp.fit(np.array(xnum_train).reshape(-1, 1), train.reshape(-1, 1))
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
    |   Population Average    |             Best Individual              |
---- ------------------------- ------------------------------------------ ----------
 Gen   Length          Fitness   Length          Fitness      OOB Fitness  Time Left
   0   263.65      3.88413e+54       39          8503.11              N/A      2.45m
   1   159.03      1.13335e+16       43          3956.42              N/A      1.10m
   2    61.32      1.22587e+10       33          3548.79              N/A     39.41s
   3    42.68      5.62364e+06       44          3528.96              N/A     35.00s
   4    42.96      9.87061e+10       29          3386.99              N/A     33.96s
   5    39.36      3.29032e+10       26          3376.33              N/A     32.38s
   6    37.80      1.26406e+06       55          3247.03              N/A     30.47s
   7    36.91      2.91967e+06       32          3205.54              N/A     33.09s
   8    43.67           594323       63          3155.85              N/A     37.48s
   9    50.79           672418       67          2983.11              N/A     32.83s
  10    59.30      1.70349e+06       76          2964.73              N/A     42.10s
  11    65.08           125632       57          2944.03              N/A     40.07s
  12    76.93          33811.1       59          2855.73              N/A     39.09s
  13    77.86      1.91832e+06       87          2805.47              N/A     40.36s
  14    87.73      1.58568e+06       69          2791.53              N/A     46.38s
  15    89.75      2.98576e+10       81          2727.11              N/A     44.44s
  16    92.83            87655      102          2676.76              N/A     42.84s
  17    90.77           211876       81          2664.46              N/A     45.15s
  18    96.83      1.11903e+06       81          2664.46              N/A     39.32s
  19   102.77           706109      102          2651.38              N/A     44.16s
  20    99.93      1.58849e+06      101          2648.07              N/A     38.63s
  21   112.01          22426.4      101          2648.07              N/A     41.58s
  22   116.34          29393.1      100          2646.04              N/A     44.69s
  23   114.55           845989      141          2636.01              N/A     47.62s
  24   127.15           125502      109          2629.95              N/A     49.10s
  25   154.03           529744      291          2613.08              N/A     42.50s
  26   167.09          48729.8      298          2606.89              N/A     48.63s
  27   210.73           131170      115          2602.71              N/A      1.00m
  28   235.10          67084.1      109          2515.77              N/A     59.70s
  29   227.51      1.67267e+06      116          2515.04              N/A     59.48s
  30   158.94          51858.4      116          2509.75              N/A     43.14s
  31   129.67      1.19201e+06      119          2418.86              N/A     38.95s
  32   133.16           780027      139          2367.53              N/A     34.08s
  33   132.34          25554.8      140          2215.15              N/A     33.64s
  34   128.35           164264      157          2188.37              N/A     32.68s
  35   133.68           327949      158          2006.16              N/A     32.80s
  36   139.67          86113.5      158          2006.16              N/A     31.19s
  37   153.02      1.50894e+08      220          1778.69              N/A     31.77s
  38   163.79          68888.2      241          1754.24              N/A     32.42s
  39   193.03           494231      222          1692.89              N/A     34.57s
  40   250.56      1.75656e+08      233          1606.67              N/A     41.38s
  41   250.42          68055.7      233          1513.46              N/A     39.07s
  42   236.00           156490      231          1461.53              N/A     38.68s
  43   234.59      2.60813e+12      220          1441.73              N/A     35.21s
  44   230.28      1.04139e+10      274           1049.3              N/A     33.31s
  45   236.38      2.03518e+06      293          1004.45              N/A     32.68s
  46   264.28           313381      299          996.645              N/A     35.03s
  47   342.69           214382      303          967.557              N/A     40.07s
  48   313.08      1.48745e+08      293          921.204              N/A     35.51s
  49   305.31      1.51403e+08      318          821.456              N/A     34.65s
  50   314.78      1.10119e+06      318          821.456              N/A     33.97s
  51   313.79           979251      385           816.43              N/A     30.46s
  52   328.37           264371      324          801.193              N/A     29.51s
  53   351.21      2.23369e+06      324          801.151              N/A     29.32s
  54   381.20      2.82095e+07      438          791.936              N/A     29.43s
  55   432.66          66809.6      328          791.425              N/A     30.80s
  56   415.93      1.91916e+08      322          782.632              N/A     27.58s
  57   422.16      1.97548e+06      413          761.037              N/A     26.19s
  58   418.50           740163      511          747.482              N/A     24.21s
  59   410.81      1.06146e+06      453          743.036              N/A     21.44s
  60   433.57      2.01478e+08      413           733.55              N/A     20.51s
  61   491.76           909412      413           733.55              N/A     20.31s
  62   493.34       2.2132e+08      403          728.525              N/A     17.70s
  63   436.28      2.76912e+06      555           725.89              N/A     13.35s
  64   440.87      4.06325e+06      445          711.701              N/A     11.43s
  65   450.23      2.95182e+06      455          700.228              N/A     14.87s
  66   500.14           529771      469          700.075              N/A      9.58s
  67   473.03           256860      450          693.784              N/A      4.81s
  68   465.11           131630      523          687.793              N/A      2.41s
  69   478.31      2.76177e+07      523          683.903              N/A      0.00s
```

Out[ ]:

```
SymbolicRegressor(const_range=(-100, 100),
                  function_set=['add', 'sub', 'mul', 'div', 'sin'],
                  generations=70, init_depth=(4, 10), metric='mse',
                  population_size=500, random_state=0, stopping_criteria=0.01,
                  verbose=1)
```

In [ ]:

```
print(est_gp._program)
```

```
sub(sub(7.988, -13.099), sub(add(sub(add(div(sub(div(X0, add(div(add(24.003, 35.637), X0), div(X0, 20.271))), sub(sub(7.988, -13.099), sub(add(add(mul(ad
d(sub(div(X0, 20.271), sin(-14.028)), mul(div(mul(X0, -15.756), sub(29.380, -14.101)), sin(add(div(X0, add(div(sub(X0, 75.483), X0), add(add(sub(div(X0, add
(div(add(24.003, 35.637), X0), div(X0, 20.271))), 7.988), div(X0, 20.271)), mul(div(mul(X0, -15.756), X0), sin(add(div(X0, add(div(X0, add(div(add(24.003, 35.
637), X0), div(X0, 20.271))), div(X0, 20.271))), sub(29.380, -14.101))))))), sub(29.380, -14.101))))), sin(add(div(X0, add(div(X0, add(div(add(24.003, 35.637),
X0), div(X0, 20.271))), div(div(add(X0, X0), add(X0, 73.055)), 20.271))), sub(29.380, -14.101)))), div(X0, 20.271)), mul(div(div(X0, 20.271), add(24.003, 35.63
7)), sin(add(div(X0, add(div(div(X0, 20.271), X0), div(X0, 20.271))), sub(29.380, -14.101))))), sub(div(-13.099, X0), sub(mul(7.988, sin(div(sub(7.988, -13.099)
, add(24.003, 35.637)))), sub(sub(7.988, -13.099), div(X0, 20.271)))))), 20.271), div(sub(7.988, -13.099), sub(X0, 75.483))), sub(X0, X0)), div(X0, 20.271)), s
ub(div(sub(7.988, -13.099), sub(X0, 75.483)), sub(mul(div(sub(X0, X0), X0), sin(sin(-14.028))), sub(sub(7.988, -13.099), sub(add(add(add(div(X0, add(div(X0
, add(div(add(24.003, 35.637), X0), div(X0, 20.271))), div(X0, 20.271))), div(X0, 20.271)), div(X0, 20.271)), mul(div(mul(X0, -15.756), X0), sin(add(div(X0, add
(div(X0, add(div(add(24.003, 35.637), X0), div(X0, 20.271))), mul(div(mul(X0, -15.756), X0), sin(add(div(X0, add(div(X0, add(div(add(24.003, 35.637), X0), div
(X0, 20.271))), div(X0, 20.271))), div(X0, 20.271)))))), sub(29.380, -14.101))))), sub(div(add(24.003, 35.637), X0), sub(mul(div(X0, add(div(add(24.003, 35.63
7), X0), div(X0, 20.271))), sin(div(sub(7.988, -13.099), sub(X0, 75.483)))), sub(sub(7.988, -13.099), sub(add(add(sub(div(X0, add(div(add(24.003, 35.637), X0
), div(X0, 20.271))), 7.988), div(X0, 20.271)), mul(div(mul(X0, -15.756), add(24.003, 35.637)), sin(add(div(X0, add(div(add(div(X0, add(div(X0, add(div(add(div
(X0, add(div(add(24.003, 35.637), X0), div(X0, 20.271))), sub(29.380, -14.101)), X0), div(X0, 20.271))), div(X0, 20.271))), sub(29.380, -14.101)), X0), div(X0,
20.271))), sub(29.380, -14.101))))), sub(div(-13.099, X0), sub(mul(7.988, sin(div(sub(29.380, -14.101), sub(X0, 75.483)))), sub(sub(7.988, -13.099), sub(add(
sub(div(X0, 20.271), sin(-14.028)), mul(div(mul(X0, -15.756), add(24.003, 35.637)), sin(add(div(X0, add(div(sub(X0, 75.483), X0), add(add(sub(div(X0, add(di
v(add(24.003, 35.637), X0), div(X0, 20.271))), 7.988), div(X0, 20.271)), mul(div(mul(X0, -15.756), X0), sin(add(div(X0, add(div(X0, add(div(add(24.003, 35.63
7), X0), div(X0, 20.271))), div(X0, 20.271))), sub(29.380, -14.101))))))), sub(29.380, -14.101))))), sub(add(sin(-70.638), sub(29.380, -14.101)), sub(add(sub(di
v(X0, add(div(add(24.003, 35.637), X0), div(X0, 20.271))), sin(-14.028)), div(X0, 20.271)), add(X0, -12.713)))))))))))))))))
```

In [ ]:

```
import graphviz
dot_data = est_gp._program.export_graphviz()
graph = graphviz.Source(dot_data)
graph
```

Out[ ]:

In [ ]:

```
# Предсказания
y_gp = est_gp.predict(np.array(xnum_test).reshape(-1, 1))
y_gp[:10]
```

Out[ ]:

```
array([343.37133551, 332.01971105, 335.13155802, 299.0643972 ,
       279.19509901, 267.1495002 , 273.22696498, 253.35755502,
       232.9563561 , 234.70538715])
```

In [ ]:

```
# Записываем предсказания в DataFrame
ts_fb2['predictions_GPLEARN'] = (train_size * [np.NAN]) + list(y_gp)
```

In [ ]:

```
fig, ax = pyplot.subplots(1, 1, sharex='col', sharey='row', figsize=(10,5))
fig.suptitle('Предсказания временного ряда (тестовая выборка)')
ts_fb2[train_size:].plot(ax=ax, legend=True)
pyplot.show()
```