Прогнозирование заказов такси

Введение

Описание исследования

В рамках данного исследования будет проведен анализ исторических данных о заказах такси в аэропортах. Цель состоит в разработке модели прогнозирования количества заказов такси на следующий час, что позволит компании «Чётенькое такси» привлекать больше водителей в периоды пиковой нагрузки. Для достижения этой цели будут использованы методы временных рядов и машинного обучения.

Цель исследования

Основной целью исследования является создание модели, которая будет предсказывать количество заказов такси на следующий час с максимальной точностью. Для этого потребуется разработать и протестировать различные модели прогнозирования, настроить их гиперпараметры и выбрать наиболее эффективную на основе метрики RMSE (Root Mean Squared Error) на тестовой выборке.

Этапы работы

- Загрузка данных и ресемплирование: Импортировать данные из файла taxi.csv и выполнить ресемплирование по часам, чтобы обеспечить единый временной интервал для анализа.
- Анализ данных: Провести первичный анализ данных, включая исследование тенденций, сезонности и других характеристик временного ряда.
- Обучение моделей: Построить и обучить различные модели прогнозирования с различными гиперпараметрами. Обеспечить, чтобы тестовая выборка составила 10% от общего объема данных.
- Оценка моделей: Проверить производительность моделей на тестовой выборке и выбрать ту, которая обеспечивает наилучшее значение метрики RMSE (не больше 48).

Теперь, когда задача ясна как никогда, а план готов, можно приступать.

Подготовка

Используемые библиотеки

```
!pip install scikit-learn==1.4.0 -q
!pip install matplotlib==3.8.4 -q
!pip install numpy==1.21.4 -q

import pandas as pd
import numpy as np
```

```
import lightgbm as lgb
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV,
train_test_split, TimeSeriesSplit
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder,
StandardScaler, MinMaxScaler, TargetEncoder
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error,
root_mean_squared_error
from statsmodels.tsa.seasonal import seasonal_decompose
```

Загрузка исходных данных

```
# Загрузка данных с разбором даты
taxi = pd.read_csv('/datasets/taxi.csv', index_col='datetime',
parse_dates=['datetime'])

taxi.sort_index(inplace = True)
```

Первичное ознакомление с данными

```
display(taxi.head(), taxi.tail(), taxi.info())
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 26496 entries, 2018-03-01 00:00:00 to 2018-08-31
23:50:00
Data columns (total 1 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   num_orders  26496 non-null  int64
dtypes: int64(1)
memory usage: 414.0 KB
```

```
                     num_orders
datetime
2018-03-01 00:00:00           9
2018-03-01 00:10:00          14
2018-03-01 00:20:00          28
2018-03-01 00:30:00          20
2018-03-01 00:40:00          32
```

```
                     num_orders
datetime
2018-08-31 23:10:00          32
2018-08-31 23:20:00          24
2018-08-31 23:30:00          27
```

```
2018-08-31 23:40:00           39
2018-08-31 23:50:00           53

None
```

Имеются данные с марта по сентябрь 2018 года, в данных не пропусков.

Анализ

Анализ временных рядов представляет собой важный инструмент в статистике и машинном обучении, предназначенный для изучения данных, изменяющихся во времени. Временные ряды, как правило, содержат последовательность наблюдений, собранных на регулярных интервалах времени, таких как дни, месяцы или часы. Этот тип анализа помогает выявить структурные закономерности, тренды, сезонные колебания и циклические эффекты, которые могут быть использованы для прогнозирования будущих значений.

Предобработка данных

Перед началом анализа обработаем данные.

```
taxi.index.duplicated().sum()

0
```

Даты не повторяются, дубликатов нет, как и пропусков, короткая получилась обработка.

Анализ временного ряда

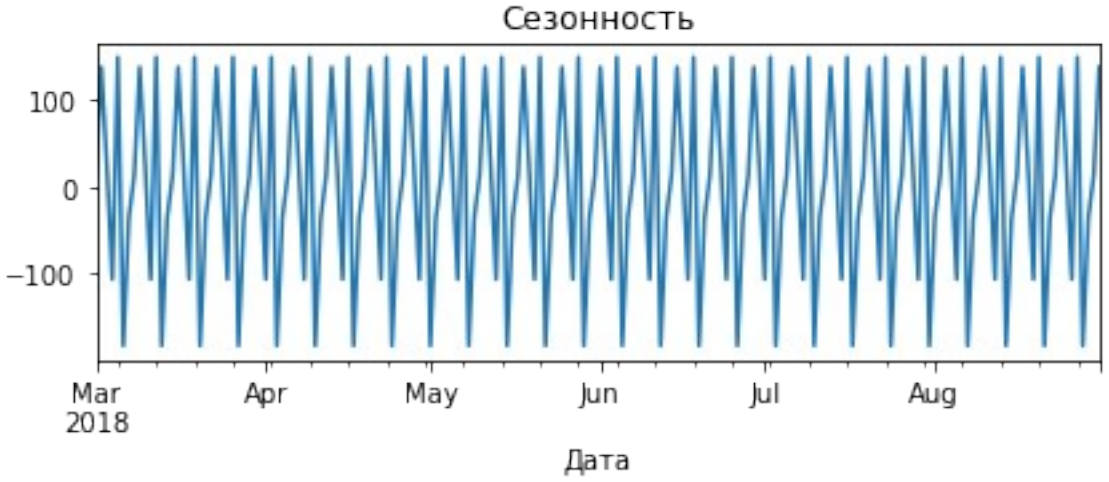
```
taxi.resample('1H').sum().describe()

        num_orders
count  4416.000000
mean     84.422781
std      45.023853
min       0.000000
25%      54.000000
50%      78.000000
75%     107.000000
max     462.000000
```

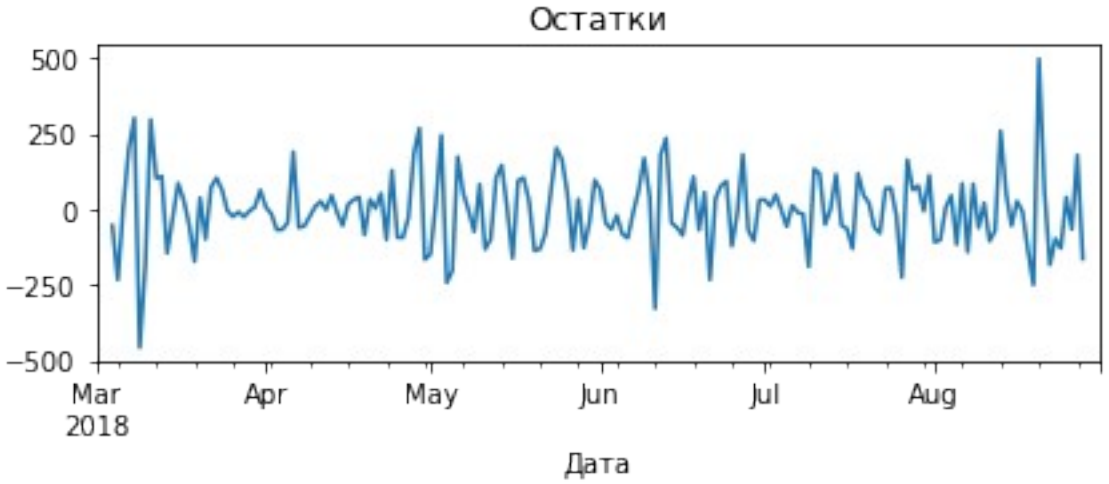
В среднем ~80 заказов в час, но бывает и ноль, стандартное отклонение в 45 заказов.

```
taxi.resample('1D').sum().plot()
plt.title('Динамика изменения количества заказов по дням')
plt.xlabel('Дата');
```

В количестве заказов явный тренд на увеличение. Разложим ряд на тренд, сезонность и остатки.

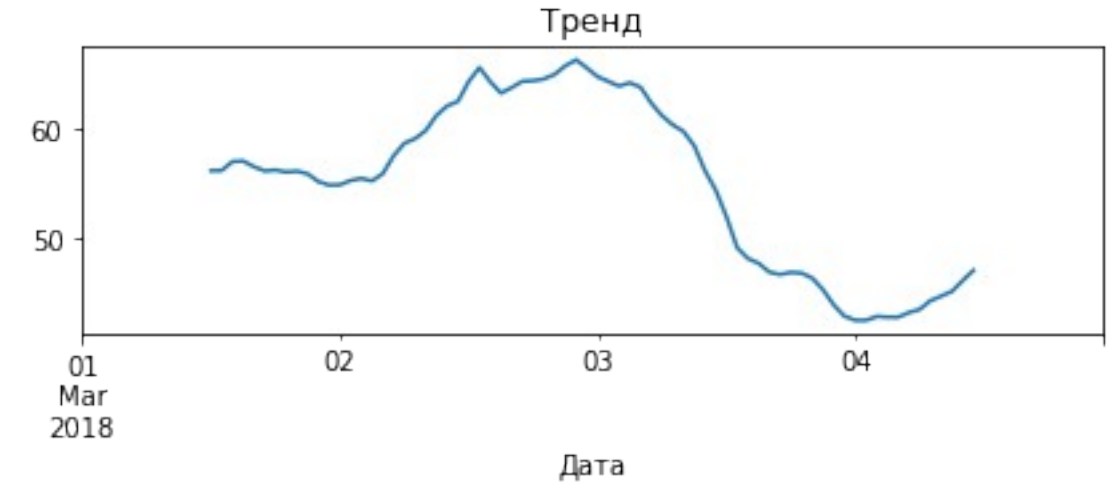
```
decomposed = seasonal_decompose(taxi.resample('1D').sum())
plt.figure(figsize=(6, 8))
plt.subplot(311)
decomposed.trend.plot(ax=plt.gca())
plt.title('Тренд')
plt.xlabel('Дата')
plt.subplot(312)
decomposed.seasonal.plot(ax=plt.gca())
plt.title('Сезонность')
plt.xlabel('Дата')
plt.subplot(313)
decomposed.resid.plot(ax=plt.gca())
plt.title('Остатки')
plt.xlabel('Дата')
plt.tight_layout()
```

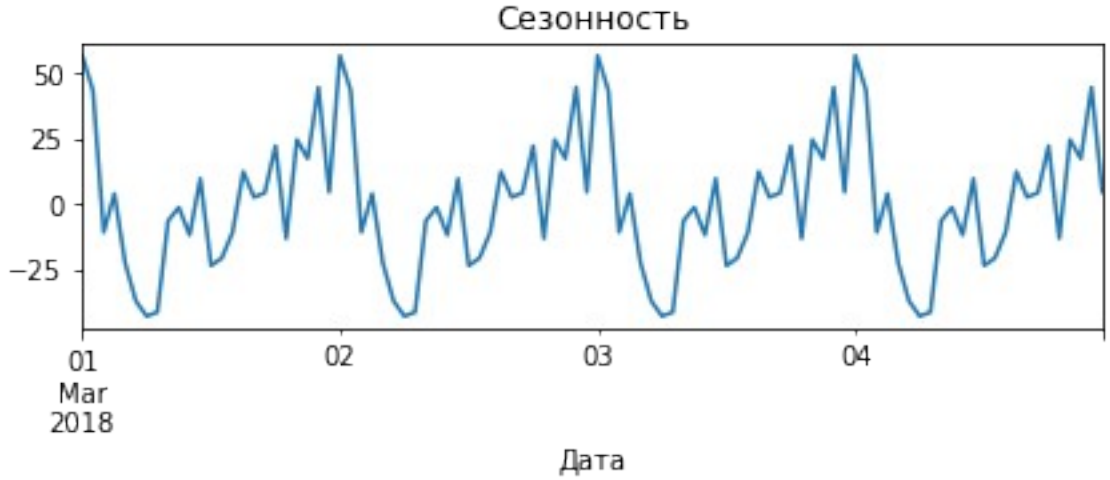


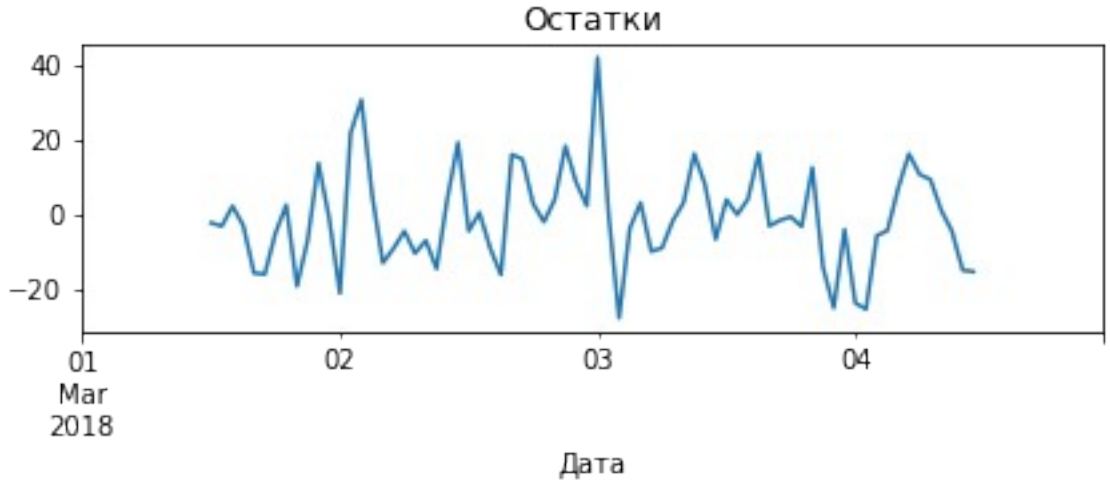


Данные имеют явный линейный тренд роста числа закзов такси по дням и недельную сезонность. В остатках нет закономерностей. Рассмотрим более короткий промежуток по часам.

```
decomposed =seasonal_decompose(taxi.resample('1H').sum()[0:96])
plt.figure(figsize=(6, 8))
plt.subplot(311)
decomposed.trend.plot(ax=plt.gca())
plt.title('Тренд')
plt.xlabel('Дата')
plt.subplot(312)
decomposed.seasonal.plot(ax=plt.gca())
plt.title('Сезонность')
plt.xlabel('Дата')
plt.subplot(313)
decomposed.resid.plot(ax=plt.gca())
plt.title('Остатки')
plt.xlabel('Дата')
plt.tight_layout()
```







Также имеется суточная сезонность на количество заказов. Данные нестационарны.

Обучение

В обучении моделей временных рядов важным этапом является создание новых признаков, которые могут улучшить точность прогнозов. В этом процессе мы начинаем с генерации новых признаков на основе исторических данных. Эти признаки помогают моделям лучше захватывать закономерности и тренды в данных, что в свою очередь способствует более точным прогнозам.

Создание признаков

```
taxi_res = taxi.resample('1H').sum()

def make_features(data, max_lag, rolling_mean_size):
    data['month'] = data.index.month
    data['day'] = data.index.day
    data['dayofweek'] = data.index.dayofweek

    for lag in range(1, max_lag + 1):
        data['lag_{}'.format(lag)] = data['num_orders'].shift(lag)

    data['rolling_mean'] =
data['num_orders'].rolling(rolling_mean_size,closed = 'left').mean()

make_features(taxi_res, 192, 24)

taxi_res.dropna(inplace = True)

# Оптимизация числовых типов данных
def optimize_memory_usage(df: pd.DataFrame, print_size: bool=True) ->
pd.DataFrame:
    '''
    Function optimizes memory usage in dataframe
    df: pd.DataFrame - data table
    print_size: bool - display of optimization results
    return pd.DataFrame - amount of optimized memory
    '''
    numerics = ['int16', 'int32', 'int64', 'float16', 'float32',
'float64'] # Типы, которые будем проверять на оптимизацию
    # Размер занимаемой памяти до оптимизации (в Мб)
    before_size = df.memory_usage().sum() / 1024**2
    for column in df.columns:
        column_type = df[column].dtypes
        if column_type in numerics:
            column_min = df[column].min()
            column_max = df[column].max()
            if str(column_type).startswith('int'):
                if column_min > np.iinfo(np.int8).min and column_max <
np.iinfo(np.int8).max:
                    df[column] = df[column].astype(np.int8)
                elif column_min > np.iinfo(np.int16).min and
```

```
column_max < np.iinfo(np.int16).max:
                    df[column] = df[column].astype(np.int16)
                elif column_min > np.iinfo(np.int32).min and
column_max < np.iinfo(np.int32).max:
                    df[column] = df[column].astype(np.int32)
                elif column_min > np.iinfo(np.int64).min and
column_max < np.iinfo(np.int64).max:
                    df[column] = df[column].astype(np.int64)
            else:
                if column_min > np.finfo(np.float32).min and
column_max < np.finfo(np.float32).max:
                    df[column] = df[column].astype(np.float32)
                else:
                    df[column] = df[column].astype(np.float64)
    # Размер занимаемой памяти после оптимизации (в Мб)
    after_size = df.memory_usage().sum() / 1024**2
    if print_size: print('Размер использования памяти: до {:5.2f} Mb -
после {:5.2f} Mb ({:.1f}%)'
                         .format(before_size, after_size, 100 *
(before_size - after_size) / before_size))
    return df

taxi_res = optimize_memory_usage(taxi_res)
```

```
Размер использования памяти: до  6.38 Mb - после  3.16 Mb (50.4%)
```

```
train, test = train_test_split(taxi_res, shuffle=False, test_size=0.1,
random_state = 42)
```

Работа с нестационарными данными в временных рядах требует особого подхода, так как стандартные модели, предполагающие стационарность, могут не давать хороших результатов. Нестационарные данные часто характеризуются изменением статистических свойств (среднего, дисперсии и т.д.) со временем. Для таких данных можно использовать модели и методы, которые могут справляться с такими изменениями.

Линейная регрессия

Линейная регрессия может использоваться с нестационарными данными, но ее эффективность и точность могут быть ограничены.

```
RANDOM_STATE = 42

#Создаём пайплайн для подготовки признаков
data_preprocessor = ColumnTransformer(
    [
        ('num', 'passthrough', list[train.columns])
    ],
    remainder='passthrough'
)
```

```
#Финашльный пайплайн
pipe_final = Pipeline([
    ('preprocessor', data_preprocessor),
    ('models', LinearRegression())
])

## TimeSeriesSplit для кросс-валидации
tscv = TimeSeriesSplit(n_splits=5)

#Параметры пайплайна
param_grid = {
       'preprocessor__num': [StandardScaler(), MinMaxScaler(),
'passthrough']
}

Linear = RandomizedSearchCV(
    pipe_final,
    param_grid,
    random_state = RANDOM_STATE,
    cv=tscv,
    n_iter = 3,
    scoring='neg_root_mean_squared_error',
    n_jobs=-1
)

Linear.fit(train.drop('num_orders', axis = 1).dropna(),
train['num_orders'])

print('Лучшие параметры:', Linear.best_estimator_)
print ('RMSE модели регрессии на кросс-валидации:', -
Linear.best_score_)
```

```
Лучшие параметры: Pipeline(steps=[('preprocessor',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('num',
'passthrough',

list[Index(['num_orders', 'month', 'day', 'dayofweek', 'lag_1',
'lag_2', 'lag_3',
       'lag_4', 'lag_5', 'lag_6',
       ...
       'lag_184', 'lag_185', 'lag_186', 'lag_187', 'lag_188',
'lag_189',
       'lag_190', 'lag_191', 'lag_192', 'rolling_mean'],
      dtype='object', length=197)])])),
                ('models', LinearRegression())])
RMSE модели регрессии на кросс-валидации: 22.9043888092041
```

Модель довольно сильно ошибается, однако с увеличением количества лагов модель становится все точне. При оценке модели по необходимо понимать, что модель хуже

прогнозирует сезонные колебания количества заказов зависящие от времени суток или дня недели.

Градиентный бустинг LightGBM

Градиентный бустинг, включая LightGBM (Light Gradient Boosting Machine), может быть использован для прогнозирования временных рядов, хотя он не предназначен специально для работы с временными рядами.

```
lgb_model = lgb.LGBMRegressor()

#Финашльный пайплайн
pipe_final = Pipeline([
    ('models', lgb_model)
])

#Параметры пайплайна
param_grid = {
    'models__boosting_type': ['gbdt'],  # Тип бустинга
    'models__objective': ['regression'],  # Целевая функция для задачи регрессии
    'models__metric': ['mae'],  # Метрика для оценки модели (корень из средней квадратичной ошибки)
    'models__num_leaves': range(2,16),  # Количество листьев в дереве
    'models__max_depth': range(2,16),# Максимальная глубина дерева
}


GBM = RandomizedSearchCV(
    pipe_final,
    param_grid,
    random_state = RANDOM_STATE,
    cv=tscv,
    scoring='neg_root_mean_squared_error',
    n_jobs=-1
)

GBM.fit(train.drop('num_orders', axis = 1).dropna(),
train['num_orders'])

print('Лучшие параметры:', GBM.best_estimator_)
print ('Метрика модели регрессии на кросс-валидации:', -
GBM.best_score_)

Лучшие параметры: Pipeline(steps=[('models',
                 LGBMRegressor(max_depth=3, metric='mae',
num_leaves=6,
                               objective='regression'))])
Метрика модели регрессии на кросс-валидации: 22.589224728604982
```

Метрики идентичны простой линейной регрессии, вероятно модель не смогла уловить необходимые закономерности тренда и сезонности данных из-за недостатоной сложности самой модели или данных.

SARIMAX

SARIMAX (Seasonal AutoRegressive Integrated Moving Average with eXogenous regressors): Модель временных рядов, которая включает в себя сезонные компоненты и экзогенные переменные. SARIMAX полезен для учета сезонных изменений и трендов в данных, что делает его подходящим для анализа сложных временных рядов.

```
model = sm.tsa.SARIMAX(
    train['num_orders'].iloc[:-len(train)//10],  # 1/10 часть тренировочной выборки используем для валидации
    exog=train.drop('num_orders', axis=1).iloc[:-len(train)//10],
    trend='t',
    method = 'nm'
)

results = model.fit()
```

```
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =          199     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  4.37367D+00    |proj g|=  1.35496D-01

 This problem is unconstrained.


At iterate    5    f=  4.37289D+00    |proj g|=  7.99885D-02

At iterate   10    f=  4.37267D+00    |proj g|=  3.33623D-01

At iterate   15    f=  4.37256D+00    |proj g|=  1.16361D-02

At iterate   20    f=  4.37252D+00    |proj g|=  2.59580D-01

At iterate   25    f=  4.37249D+00    |proj g|=  2.30163D-01

At iterate   30    f=  4.37243D+00    |proj g|=  1.41825D-01

At iterate   35    f=  4.37239D+00    |proj g|=  8.85376D-02

At iterate   40    f=  4.37235D+00    |proj g|=  1.09745D-01

At iterate   45    f=  4.37233D+00    |proj g|=  3.54757D-02
```

```
/opt/conda/lib/python3.9/site-packages/statsmodels/base/model.py:604:
ConvergenceWarning: Maximum Likelihood optimization failed to
converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "


At iterate   50    f=  4.37229D+00    |proj g|=  5.85762D-02

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
  199     50     63      1     0     0   5.858D-02   4.372D+00
  F =   4.3722907675345786

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT
```

```
# Прогнозирование
forecast = results.get_forecast(steps=len(train["num_orders"].iloc[-
len(train)//10:]), exog = train.drop('num_orders',axis=1).iloc[-
len(train)//10:])
predicted_values = forecast.predicted_mean
print(f'Метрика RMSE на валидационной выборке:
{root_mean_squared_error(train["num_orders"].iloc[-len(train)//10:],
predicted_values)}')
```

```
Метрика RMSE на валидационной выборке: 28.94583134594145
```

Дообучим модель на всех тренировочных данных.

```
model = sm.tsa.SARIMAX(
    train['num_orders'],
    exog=train.drop('num_orders', axis=1),
    trend='t')

results = model.fit()
```

```
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
```

```
 N =          199     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  4.43564D+00    |proj g|=  2.03595D-01

 This problem is unconstrained.

At iterate    5    f=  4.42731D+00    |proj g|=  1.62602D-01

At iterate   10    f=  4.42596D+00    |proj g|=  1.92417D-02

At iterate   15    f=  4.42587D+00    |proj g|=  7.43266D-02

At iterate   20    f=  4.42577D+00    |proj g|=  1.77392D-02

At iterate   25    f=  4.42576D+00    |proj g|=  1.87847D-02

At iterate   30    f=  4.42575D+00    |proj g|=  1.85038D-02

At iterate   35    f=  4.42574D+00    |proj g|=  1.61358D-02

At iterate   40    f=  4.42574D+00    |proj g|=  7.06429D-03

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
  199     42     59      1     0     0   3.775D-02   4.426D+00
  F =   4.4257328370395141

CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
```

Модель имеет RMSE выше чем RMSE линейной регрессии, это может быть связано с тем, что SARIMAX Требует тщательной настройки параметров, таких как порядок авторегрессии (p), интеграции (d), скользящего среднего (q) и сезонных компонентов. Неправильный выбор параметров может ухудшить качество прогнозов. Подобрать корректные параметры для модели оказалось проблематично из-за долгого обучения, и "смерти ядра".

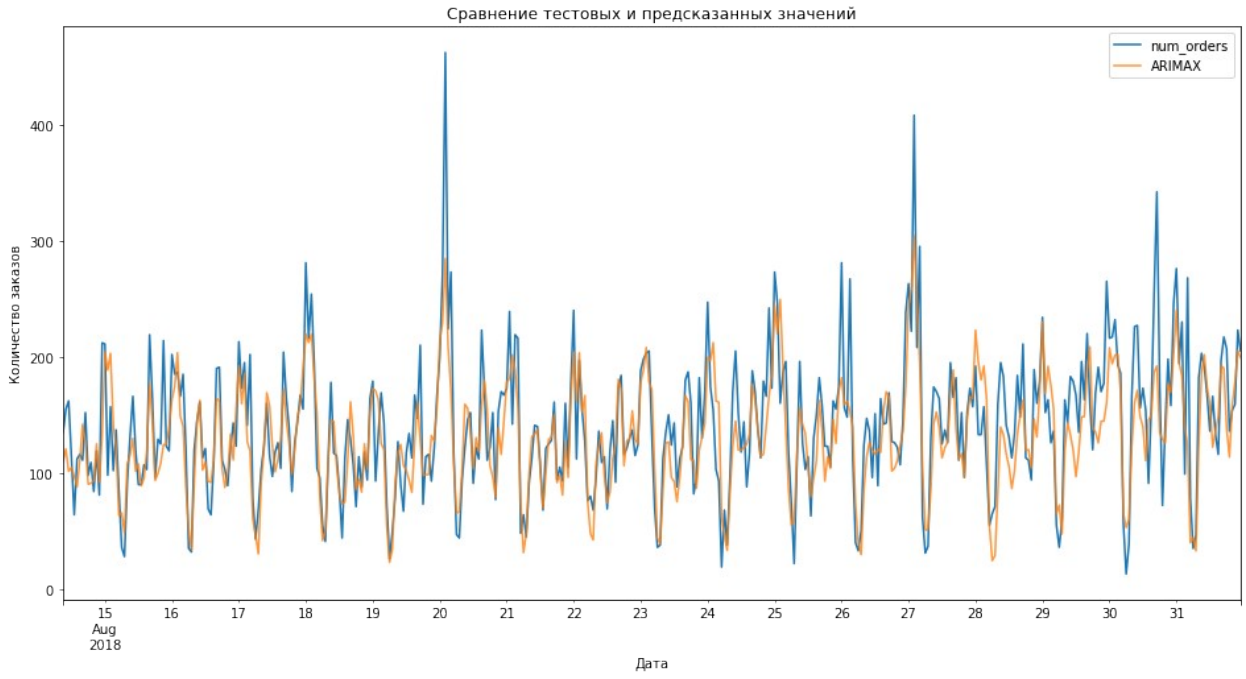
Тестирование

RMSE модели SARIMAX выше RMSE других моделей, однако эта модель способна лучше справлятся с сезонностью в данных и имеет широкий выбор гиперпарметров для настройки, а так как RMSE модели удовлетворяет условиям задачи, для дальнейшего тестирования выберем модель SARIMAX, кажущуюся наиболее перспективной.

```
# Прогнозирование
forecast_test = results.get_forecast(steps=len(test),
                                     exog =
test.drop('num_orders',axis=1))

predicted_values_test = forecast_test.predicted_mean

print(f'RMSE SARIMAX на тестовой выборке:
{root_mean_squared_error(test["num_orders"], predicted_values_test)}')
```

```
RMSE SARIMAX на тестовой выборке: 35.510491627964484
```

```
test_res = pd.DataFrame(test["num_orders"])
test_res['ARIMAX'] = predicted_values_test

test_res['num_orders'].plot(figsize=(16,8))
test_res['ARIMAX'].plot(alpha=0.8)
plt.xlabel('Дата')
plt.ylabel('Количество заказов')
plt.title('Сравнение тестовых и предсказанных значений')
plt.legend();
```



На графике видно что модель не смогла до конца уловить недельную сезонность в данных.

Итоги

Для обучения моделей были предоставленны данные о количестве заказов такси каждые 10 минут, с февраля по сентябрь 2018 года. Данные были ресемплированы чтобы отображать количество заказов в час. Для улучшения качества прогнозирования и учета различных аспектов временных рядов, были добавлены несколько новых признаков:

- Месяц
- День месяца
- День недели
- Отставание от lag_1 до lag_192 (До определенного количества лагов метрики моделей растут.)
- Скользящее среднее за 24 часа

Данные, на которых обучались модели, включают тренды и сезонные компоненты. Количество заказов варьируется в зависимости от времени суток и дня недели. В целом, наблюдается рост количества заказов с течением времени, что может быть связано либо с притоком новых клиентов, либо с сезонными особенностями.

Были обучены и протестированы три модели для прогнозирования: линейная регрессия, градиентный бустинг и SARIMAX. Оценка их производительности на валидации показала следующие результаты:

RMSE линейной регрессии: 22.9

RMSE градиентного бустинга: 22.59

RMSE SARIMAX: 28.94

На тестовой выборке модель SARIMAX показала RMSE равный 35.51.

Сопоставимые оценки RMSE на валидации у трех разных моделей (линейная регрессия, градиентный бустинг и SARIMAX) могут указывать на несколько вещей:

1. Нехватка информации в данных: Если данные содержат шум, выбросы или недостаточно признаков для учета всех факторов, модели могут показывать схожие результаты. Даже если одна модель более сложная и имеет дополнительные возможности, ее эффективность может быть ограничена качеством данных.
2. Неучет ключевых факторов: Все три модели могут испытывать трудности с учетом ключевых факторов, таких как недельная сезонность и праздничные периоды.
3. Ограниченные гиперпараметры: Модели могут быть недостаточно настроены. Например, для градиентного бустинга и SARIMAX требуется тщательная настройка гиперпараметров. Если

параметры не были оптимизированы, это может привести к снижению различий в производительности между моделями.

Модель SARIMAX, будучи специализированной для учета сезонности и трендов, более эффективно справляется с такими компонентами по сравнению с линейной регрессией и градиентным бустингом, которые могут не учитывать все сезонные и трендовые особенности данных, поэтому для дальнейшего развития и масштабирования я предлагаю модель SARIMAX. Новые данные и тонкая настройка гиперпараметров модели помогут достичь лучших результатов.