Bike rental service. Analysis of demand factors using ML

В этом проекте мы будем анализировать данные сервиса проката велосипедов в Корее с помощью методов машинного обучения, чтобы понять, от чего зависит спрос на эту услугу.

1. Постановка цели и задач

Data
Работаем с данными сервиса проката велосипедов в Корее за год.

Column Name	Description
Date	дата
Rented Bike Count	*сколько велосипедов было взято в прокат, целевая переменная*
Hour	час дня
Temperature	температура воздуха в градусах Цельсия
Humidity	влажность воздуха
Wind Speed	скорость ветра в м/с
Visibility	мера различимости объектов на расстоянии в 10 метров
Dew point temperature	температура, зарегистрированная в начале дня, в градусах Цельсия
Solar Radiation	интенсивность солнечного света
Rainfall	количество осадков в мм
Snowfall	количество выпавшего снега в мм
Seasons	время года
Holiday	является ли день праздничным
Functioning Day	маркер, работал ли сервис проката в указанное время

Цель:

> *Изучить данные и выявить факторы влияющие на спрос велосипедов.*

Задачи:

1. Провести предобработку данных: проверить данные на наличие выбросов, ошибочных значений, пропусков, дубликатов и некорректных типов.
2. Провести EDA: реализовать все уровни анализа (одномерные/многомерные) с использованием визуализаций, изучить распределения и взаимосвязь признаков.
3. Подготовить данные для построения модели (кодирование признаков, масштабирование, разбиение выборки на обучающую и тестовую).
4. Реализовать базовую регрессионную модель прогнозирования количества велосипедов, взятых в прокат.
5. При помощи инструментов Feature Selection и подбора гиперпараметров подобрать наилучшую прогнозную модель по adjusted R2 (основная метрика) и RMSE.

2. Загрузка и подготовка данных

In [2]:
```
# Импорт необходимых библиотек
# При необходимости можно установить пакеты из requirements.txt
# pip install -r requirements.txt
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import phik
from sklearn import metrics
from sklearn.metrics import mean_squared_error, r2_score, make_scorer
from sklearn.model_selection import train_test_split, cross_val_score, KFold, cross_validate, Stra
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LinearRegression, ElasticNet
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.compose import make_column_transformer, make_column_selector
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.datasets import make_regression
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
# Загрузка данных, используем кодировку Windows-1251, что может быть полезно для данных на Кирилли

url = open('data_url.txt', 'r').readline()
df = pd.read_csv(url, encoding='cp1251')
df
```

Out[3]:

	Date	Rented Bike Count	Hour	Temperature(°C)	Humidity(%)	Wind speed (m/s)	Visibility (10m)	Dew point temperature(°C)
0	01/12/2017	254	0	-5.2	37	2.2	2000	-17.6
1	01/12/2017	204	1	-5.5	38	0.8	2000	-17.6
2	01/12/2017	173	2	-6.0	39	1.0	2000	-17.7
3	01/12/2017	107	3	-6.2	40	0.9	2000	-17.6
4	01/12/2017	78	4	-6.0	36	2.3	2000	-18.6
...	...	...	...	...	...	...	...	...
8755	30/11/2018	1003	19	4.2	34	2.6	1894	-10.3
8756	30/11/2018	764	20	3.4	37	2.3	2000	-9.9
8757	30/11/2018	694	21	2.6	39	0.3	1968	-9.9
8758	30/11/2018	712	22	2.1	41	1.0	1859	-9.8
8759	30/11/2018	584	23	1.9	43	1.3	1909	-9.3

8760 rows × 14 columns

In [4]:
```
# Выведем значения в момент, когда прокат не работал
print(f'За время наблюдений сервис проката не работал {(df["Functioning Day"] == "No").sum()} часо

df_filtered = df[['Rented Bike Count', 'Functioning Day']]

filtered_result = df_filtered.loc[df_filtered['Functioning Day'] == 'No']
filtered_result['Rented Bike Count'].unique()
print(filtered_result)
```

```
За время наблюдений сервис проката не работал 295 часов
      Rented Bike Count Functioning Day
3144                  0              No
3145                  0              No
3146                  0              No
3147                  0              No
3148                  0              No
...                 ...             ...
8251                  0              No
8252                  0              No
8253                  0              No
8254                  0              No
8255                  0              No

[295 rows x 2 columns]
```

In [5]:
```
# Целевая переменная в данные часы равна нулю.<br>
# Удалим эти значения, т.к. они не информативны и приводят к искажению результатов анализа.
# Нулевое значение целевой переменной приведет к смещению средней в меньшую сторону.

df = df[df['Functioning Day'] != 'No']
df = df.drop('Functioning Day', axis=1)
df
```

Out[5]:

	Date	Rented Bike Count	Hour	Temperature(°C)	Humidity(%)	Wind speed (m/s)	Visibility (10m)	Dew point temperature(°C)
0	01/12/2017	254	0	-5.2	37	2.2	2000	-17.6
1	01/12/2017	204	1	-5.5	38	0.8	2000	-17.6
2	01/12/2017	173	2	-6.0	39	1.0	2000	-17.7
3	01/12/2017	107	3	-6.2	40	0.9	2000	-17.6
4	01/12/2017	78	4	-6.0	36	2.3	2000	-18.6
...	...	...	...	...	...	...	...	...
8755	30/11/2018	1003	19	4.2	34	2.6	1894	-10.3
8756	30/11/2018	764	20	3.4	37	2.3	2000	-9.9
8757	30/11/2018	694	21	2.6	39	0.3	1968	-9.9
8758	30/11/2018	712	22	2.1	41	1.0	1859	-9.8
8759	30/11/2018	584	23	1.9	43	1.3	1909	-9.3

8465 rows × 13 columns

In [6]:
```
# Преобразуем временной ряд Date - разобьём его на отдельные признаки (год, месяц, день, день неде
df['Date'] = pd.to_datetime(df['Date'], format='%d/%m/%Y')
df['year'] = df['Date'].dt.year
df['month'] = df['Date'].dt.month
df['day'] = df['Date'].dt.day
df['day_of_week'] = df['Date'].dt.dayofweek
df = df.drop('Date', axis=1)
df
```

Out[6]:

	Rented Bike Count	Hour	Temperature(°C)	Humidity(%)	Wind speed (m/s)	Visibility (10m)	Dew point temperature(°C)	Solar Radiation (MJ/m2)
0	254	0	-5.2	37	2.2	2000	-17.6	0.0
1	204	1	-5.5	38	0.8	2000	-17.6	0.0
2	173	2	-6.0	39	1.0	2000	-17.7	0.0
3	107	3	-6.2	40	0.9	2000	-17.6	0.0
4	78	4	-6.0	36	2.3	2000	-18.6	0.0
...	...	...	...	...	...	...	...	...
8755	1003	19	4.2	34	2.6	1894	-10.3	0.0
8756	764	20	3.4	37	2.3	2000	-9.9	0.0
8757	694	21	2.6	39	0.3	1968	-9.9	0.0
8758	712	22	2.1	41	1.0	1859	-9.8	0.0
8759	584	23	1.9	43	1.3	1909	-9.3	0.0

8465 rows × 16 columns

Данные загружены, удалены лишние строки, временные переменные приведены к соответствующему формату

3. Exploratory Data Analysis

In [7]:
```
df.describe()
```

Out[7]:

	Rented Bike Count	Hour	Temperature(°C)	Humidity(%)	Wind speed (m/s)	Visibility (10m)	Dew point temperature(°C)
count	8465.000000	8465.000000	8465.000000	8465.000000	8465.000000	8465.000000	8465.000000
mean	729.156999	11.507029	12.771057	58.147194	1.725883	1433.873479	3.944997
std	642.351166	6.920899	12.104375	20.484839	1.034281	609.051229	13.242399
min	2.000000	0.000000	-17.800000	0.000000	0.000000	27.000000	-30.600000
25%	214.000000	6.000000	3.000000	42.000000	0.900000	935.000000	-5.100000
50%	542.000000	12.000000	13.500000	57.000000	1.500000	1690.000000	4.700000
75%	1084.000000	18.000000	22.700000	74.000000	2.300000	2000.000000	15.200000
max	3556.000000	23.000000	39.400000	98.000000	7.400000	2000.000000	27.200000

In [8]:
```
# Обратим внимание, что по году минимальное значение 2017, максимальное 2018.
# Этот признак не информативен, поэтому его можно удалить и не учитывать в анализе.
df = df.drop('year', axis=1)
```

In [9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 8465 entries, 0 to 8759
Data columns (total 15 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Rented Bike Count          8465 non-null   int64
 1   Hour                       8465 non-null   int64
 2   Temperature(°C)            8465 non-null   float64
 3   Humidity(%)                8465 non-null   int64
 4   Wind speed (m/s)           8465 non-null   float64
 5   Visibility (10m)           8465 non-null   int64
 6   Dew point temperature(°C)  8465 non-null   float64
 7   Solar Radiation (MJ/m2)    8465 non-null   float64
 8   Rainfall(mm)               8465 non-null   float64
 9   Snowfall (cm)              8465 non-null   float64
 10  Seasons                    8465 non-null   object
 11  Holiday                    8465 non-null   object
 12  month                      8465 non-null   int32
 13  day                        8465 non-null   int32
 14  day_of_week                8465 non-null   int32
dtypes: float64(6), int32(3), int64(4), object(2)
memory usage: 958.9+ KB
```

In [10]:
```
# Проверим датасет на наличие дубликатов
print(df[df.duplicated()])
```

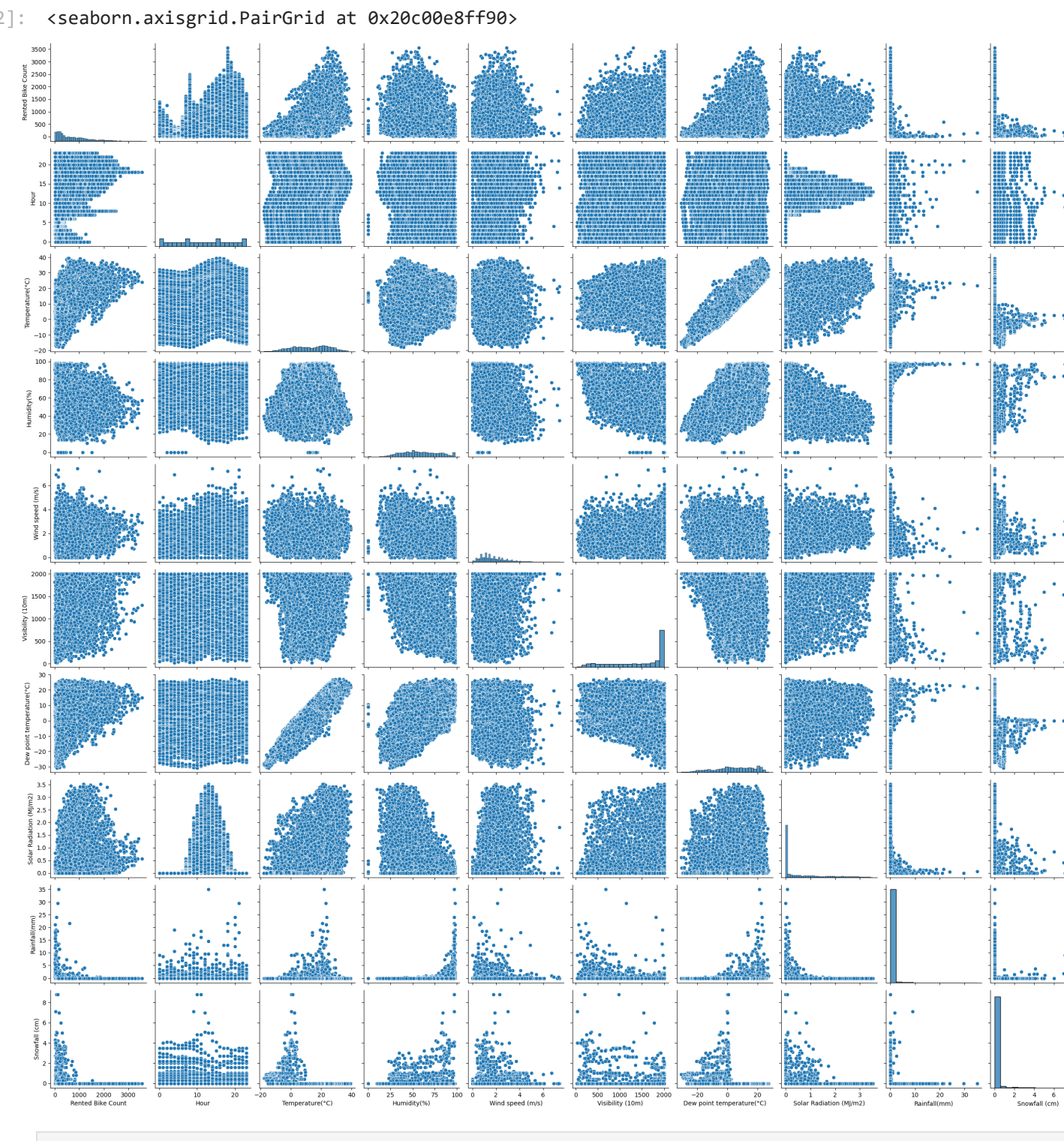
```
Empty DataFrame
Columns: [Rented Bike Count, Hour, Temperature(°C), Humidity(%), Wind speed (m/s), Visibility (10
m), Dew point temperature(°C), Solar Radiation (MJ/m2), Rainfall(mm), Snowfall (cm), Seasons, Holi
day, month, day, day_of_week]
Index: []
```

In [11]:
```
# Пропуски и дубликаты отсутствуют, однако имеются категориальные признаки, требующие кодировки пе
df.columns
```

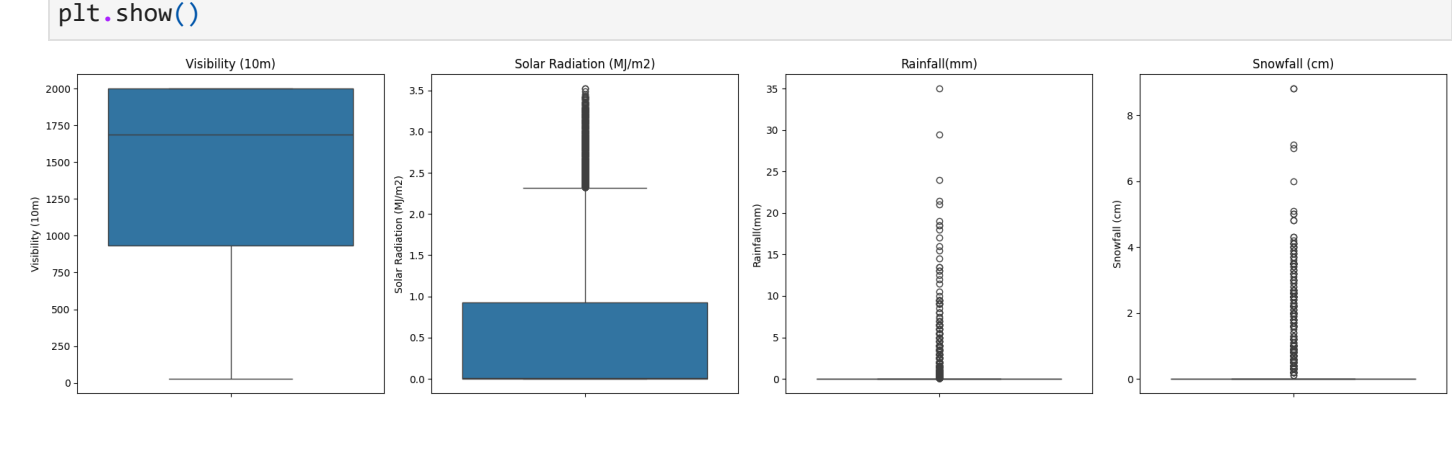
Out[11]:
```
Index(['Rented Bike Count', 'Hour', 'Temperature(°C)', 'Humidity(%)',
       'Wind speed (m/s)', 'Visibility (10m)', 'Dew point temperature(°C)',
       'Solar Radiation (MJ/m2)', 'Rainfall(mm)', 'Snowfall (cm)', 'Seasons',
       'Holiday', 'month', 'day', 'day_of_week'],
      dtype='object')
```

In [12]:
```
df_count_var = df[['Rented Bike Count', 'Hour', 'Temperature(°C)', 'Humidity(%)',
       'Wind speed (m/s)', 'Visibility (10m)', 'Dew point temperature(°C)',
       'Solar Radiation (MJ/m2)', 'Rainfall(mm)', 'Snowfall (cm)', 'Seasons',
       'Holiday']]

sns.pairplot(df_count_var)
```

Out[12]:
```
<seaborn.axisgrid.PairGrid at 0x20c00e8ff90>
```



In [13]:
```
# Можно обратить внимание, что распределения признаков погодных условий имеют тяжелый хвост.
# Это может быть связано с выбросами, посмотрим поподробнее на эти переменные.
selected_features = ['Visibility (10m)', 'Solar Radiation (MJ/m2)', 'Rainfall(mm)', 'Snowfall (cm)
num_selected_features = len(selected_features)
fig, axes = plt.subplots(1, num_selected_features, figsize=(num_selected_features * 5, 5))
for i, feature in enumerate(selected_features):
    sns.boxplot(data=df, y=feature, ax=axes[i])
    axes[i].set_title(feature)
plt.tight_layout()
plt.show()
```



In [14]:
```
# Отметим, что признаки Rainfall и Snowfall сосредоточены у нуля.
# Заменим эти переменные одним признаком "есть осадки" / "нет осадков" (Precipitation).

df['Precipitation'] = ((df['Rainfall(mm)'] > 0) | (df['Snowfall (cm)'] > 0)).astype(int) # Провери
df = df.drop(['Rainfall(mm)', 'Snowfall (cm)'], axis=1)                                    # весомо
df
```

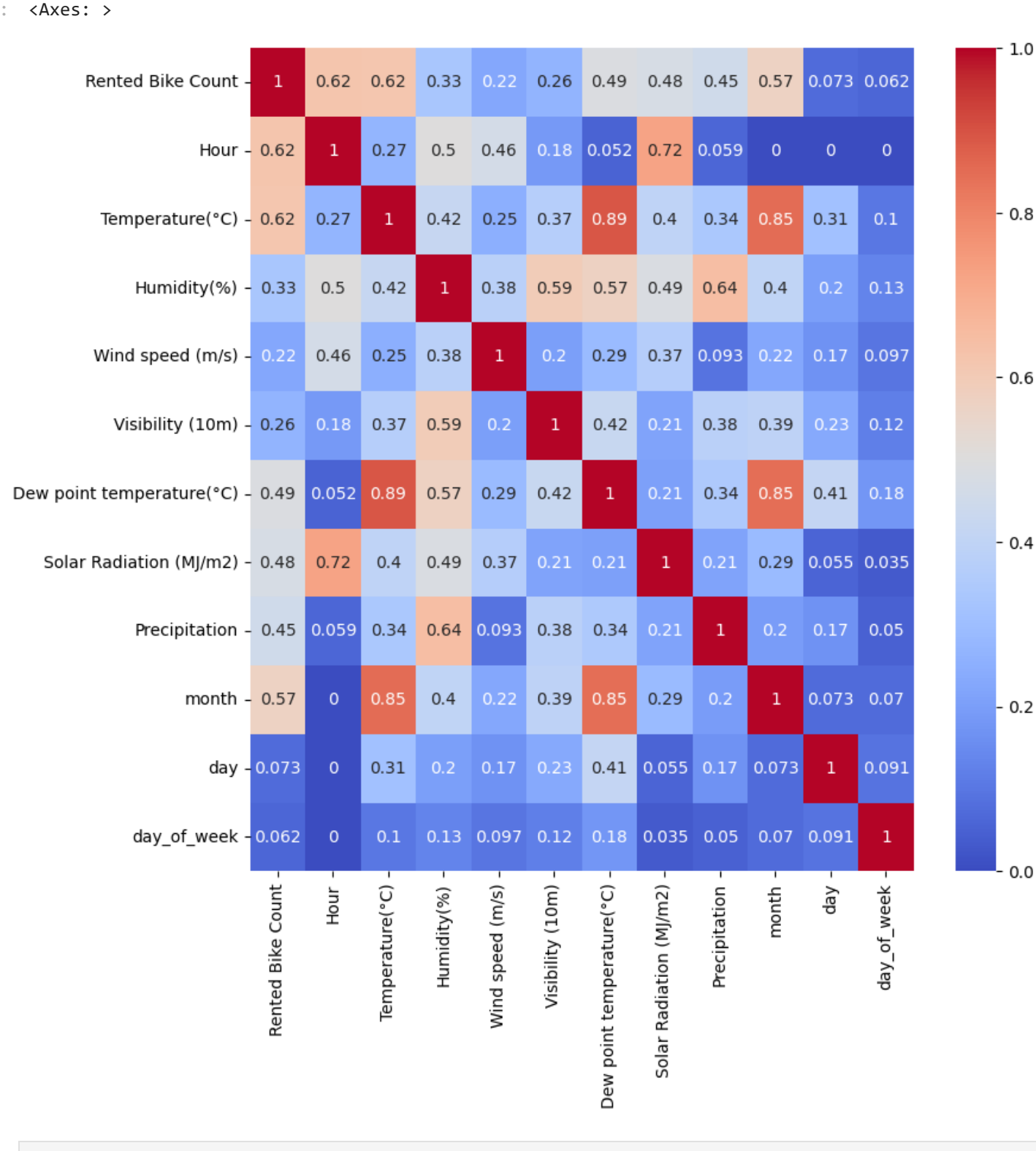
Out[14]:

	Rented Bike Count	Hour	Temperature(°C)	Humidity(%)	Wind speed (m/s)	Visibility (10m)	Dew point temperature(°C)	Solar Radiation (MJ/m2)	Seasons
0	254	0	-5.2	37	2.2	2000	-17.6	0.0	Winter
1	204	1	-5.5	38	0.8	2000	-17.6	0.0	Winter
2	173	2	-6.0	39	1.0	2000	-17.7	0.0	Winter
3	107	3	-6.2	40	0.9	2000	-17.6	0.0	Winter
4	78	4	-6.0	36	2.3	2000	-18.6	0.0	Winter
...	...	...	...	...	...	...	...	...	...
8755	1003	19	4.2	34	2.6	1894	-10.3	0.0	Autumn
8756	764	20	3.4	37	2.3	2000	-9.9	0.0	Autumn
8757	694	21	2.6	39	0.3	1968	-9.9	0.0	Autumn
8758	712	22	2.1	41	1.0	1859	-9.8	0.0	Autumn
8759	584	23	1.9	43	1.3	1909	-9.3	0.0	Autumn

8465 rows × 14 columns

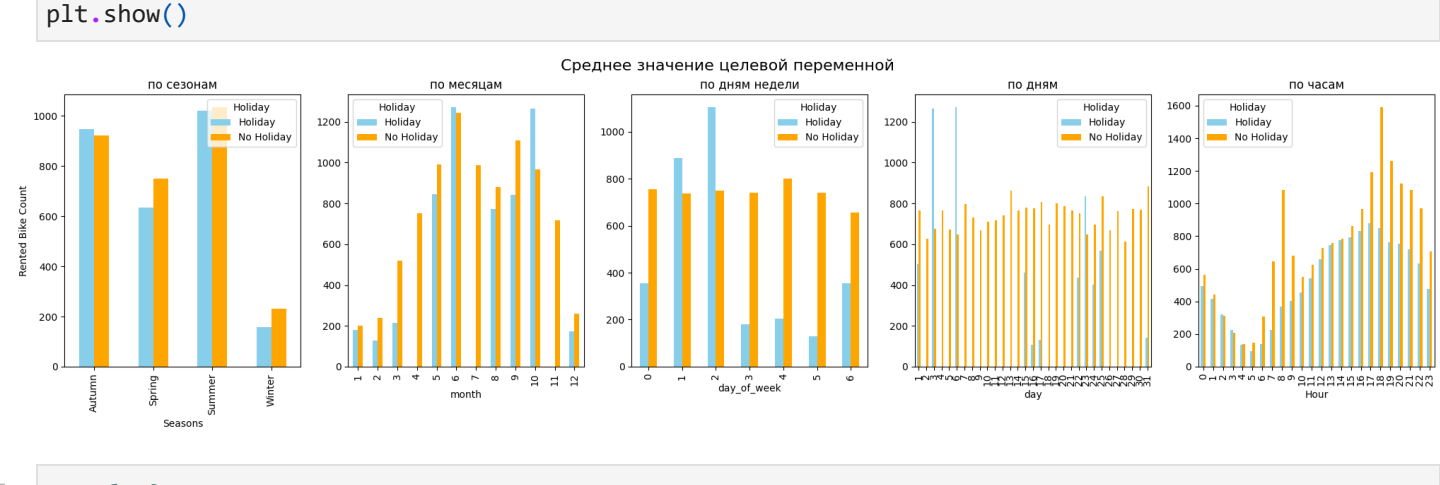
In [15]:
```
# Построим корреляционную матрицу и проанализируем зависимости между факторами

k = df[['Rented Bike Count', 'Hour', 'Temperature(°C)', 'Humidity(%)',
       'Wind speed (m/s)', 'Visibility (10m)', 'Dew point temperature(°C)',
       'Solar Radiation (MJ/m2)', 'Precipitation', 'month', 'day', 'day_of_week']].phik_matrix(int
       'Wind speed (m/s)', 'Visibility (10m)', 'Dew point temperature(°C)',
       'Solar Radiation (MJ/m2)', 'Precipitation', 'month', 'day', 'day_of_week'])
plt.figure(figsize=(9, 9))
sns.heatmap(k, annot=True, cmap='coolwarm')
```

Out[15]:
```
<Axes: >
```



In [ ]:
```
# Отметим корреляцию:
# 1. Высокая: температура в начале дня, температура, месяц.
# 2. Высокая: час дня, интенсивность солнечного света.
# 3. Средняя: кол-ва велосипедов взятых в прокат (целевая переменная), температура, час дня.
# Вывод: наблюдаем мультиколлинеарность, при построении модели необходимо будет предусмотреть регу
```

In [16]:
```
# Распределение среднего значения целевой переменной по временным признакам

by_seasons = df.groupby(['Seasons', 'Holiday'])['Rented Bike Count'].mean().unstack()
by_month = df.groupby(['month', 'Holiday'])['Rented Bike Count'].mean().unstack()
by_day_of_week = df.groupby(['day_of_week', 'Holiday'])['Rented Bike Count'].mean().unstack()
by_day = df.groupby(['day', 'Holiday'])['Rented Bike Count'].mean().unstack()
by_hour = df.groupby(['Hour', 'Holiday'])['Rented Bike Count'].mean().unstack()

fig, axs = plt.subplots(1, 5, figsize=(25, 5))

by_seasons.plot(kind='bar', ax=axs[0], color=['skyblue', 'orange'])
axs[0].set_title('по сезонам')
axs[0].set_ylabel('Rented Bike Count')

by_month.plot(kind='bar', ax=axs[1], color=['skyblue', 'orange'])
axs[1].set_title('по месяцам')

by_day_of_week.plot(kind='bar', ax=axs[2], color=['skyblue', 'orange'])
axs[2].set_title('по дням недели')

by_day.plot(kind='bar', ax=axs[3], color=['skyblue', 'orange'])
axs[3].set_title('по дням')

by_hour.plot(kind='bar', ax=axs[4], color=['skyblue', 'orange'])
axs[4].set_title('по часам')

fig.suptitle('Среднее значение целевой переменной', fontsize=16)
plt.show()
```



In [17]:
```
# Наблюдаем:
# 1. Наибсший спрос на прокат летом и осенью, достаточно высокий весной, низкий зимой.
# 2. Стабильный высокий спрос в непраздничные дни, низкий спрос - по будням в праздничные дни.
# 3. Распределение по дням в месяце равномерное, корреляция с целевой переменной очень слабая - мо
# 4. Повышенный спрос в рабочие дни в утренние и вечерние часы - арендой пользуются офисные сотру

df = df.drop('day', axis=1)
```

4. Построение моделей

In [18]:
```
# Разбиваем датасет на тренировочную и тестовую выборки.

X = df[['Hour', 'Temperature(°C)', 'Humidity(%)',
       'Wind speed (m/s)', 'Visibility (10m)', 'Dew point temperature(°C)',
       'Solar Radiation (MJ/m2)', 'Seasons', 'Holiday', 'month', 'day_of_week',
       'Precipitation']]
y = df['Rented Bike Count']

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

print('Train:', X_train.shape, y_train.shape)
print('Test:', X_test.shape, y_test.shape)
```

```
Train: (6348, 12) (6348,)
Test: (2117, 12) (2117,)
```

In [19]:
```
# Кодирование категориальных признаков

X_train_cod = pd.get_dummies(X_train, columns=['Seasons', 'Holiday'], drop_first=True)
X_test_cod = pd.get_dummies(X_test, columns=['Seasons', 'Holiday'], drop_first=True)
```

In [21]:
```
# Стандартизация

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_cod)
X_test_scaled = scaler.transform(X_test_cod)
```

4.1 Построение базовой модели линейной регрессии

In [22]:
```
model = LinearRegression()
model.fit(X_train_scaled, y_train)

y_train_predict = model.predict(X_train_scaled)
y_test_predict = model.predict(X_test_scaled)

r2_train = r2_score(y_train, y_train_predict)
r2_test = r2_score(y_test, y_test_predict)

print(f"Train R^2: {r2_train}")
print(f"Test R^2: {r2_test}")

n_train = X_train.shape[0]
n_test = X_test.shape[0]
k = X_train.shape[1]

adjusted_r2_train = 1 - ((1 - r2_train) * (n_train - 1)) / (n_train - k - 1)
adjusted_r2_test = 1 - ((1 - r2_test) * (n_test - 1)) / (n_test - k - 1)

print(f"Adjusted train R^2: {adjusted_r2_train}")
print(f"Adjusted test R^2: {adjusted_r2_test}")
```

```
Train R^2: 0.5414604791232176
Test R^2: 0.5492994301163697
Adjusted train R^2: 0.5405918959739641
Adjusted test R^2: 0.5467288945466913
```

In [23]:
```
# Качество модели недостаточное. Применим Pipeline и Grid Search.
```

4.2 Полиномиальная регрессия с применением Pipeline и Grid Search

In [24]:
```
# Применяем стандартизацию, кодирование (OneHotEncoder) и Grid Search подбор степени полинома и ко

preprocessor = make_column_transformer(
    (make_pipeline(StandardScaler()), make_column_selector(dtype_include=np.number)),
    (make_pipeline(OneHotEncoder(drop='first')), make_column_selector(dtype_include=object))
)

model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('poly', PolynomialFeatures()),
    ('elasticnet', ElasticNet())

])

param_grid = {
    'poly__degree': [1, 2, 3],
    'elasticnet__alpha': [0.01, 0.1, 0.25, 0.5, 0.75, 1],
    'elasticnet__l1_ratio': [0, 0.25, 0.5, 0.75, 1]

}

grid_search = GridSearchCV(model, param_grid, cv=5, scoring='r2', error_score="raise")
grid_search.fit(X_train, y_train)


best_model = grid_search.best_estimator_
print("Best Parameters:", grid_search.best_params_)
print("Лучшая метрика (R^2) на обучающем наборе данных:", grid_search.best_score_)
```

```
Best Parameters: {'elasticnet__alpha': 0.1, 'elasticnet__l1_ratio': 1, 'poly__degree': 3}
Лучшая метрика (R^2) на обучающем наборе данных: 0.8256248531283547
```

In [25]:
```
y_train_pred = best_model.predict(X_train)
train_r2 = r2_score(y_train, y_train_pred)
adjusted_train_r2 = 1 - ((1 - train_r2) * (n_train - 1)) / (n_train - k - 1)
train_rmse = np.sqrt(mean_squared_error(y_train, y_train_pred))

cross_val_r2 = cross_val_score(best_model, X_train, y_train, cv=7, scoring='r2')
adjusted_cross_val_r2 = 1 - ((1 - np.mean(cross_val_r2)) * (n_train - 1)) / (n_train - k - 1)
cross_val_rmse = np.sqrt(-cross_val_score(best_model, X_train, y_train, cv=7, scoring='neg_mean_sq

print("Adjusted R2 на тренировочной выборке:", adjusted_train_r2)
print("RMSE на тренировочной выборке:", train_rmse)
print("Adjusted R2 на кросс-валидации:", cross_val_r2.mean())
print("RMSE на кросс-валидации::", cross_val_rmse.mean())
```

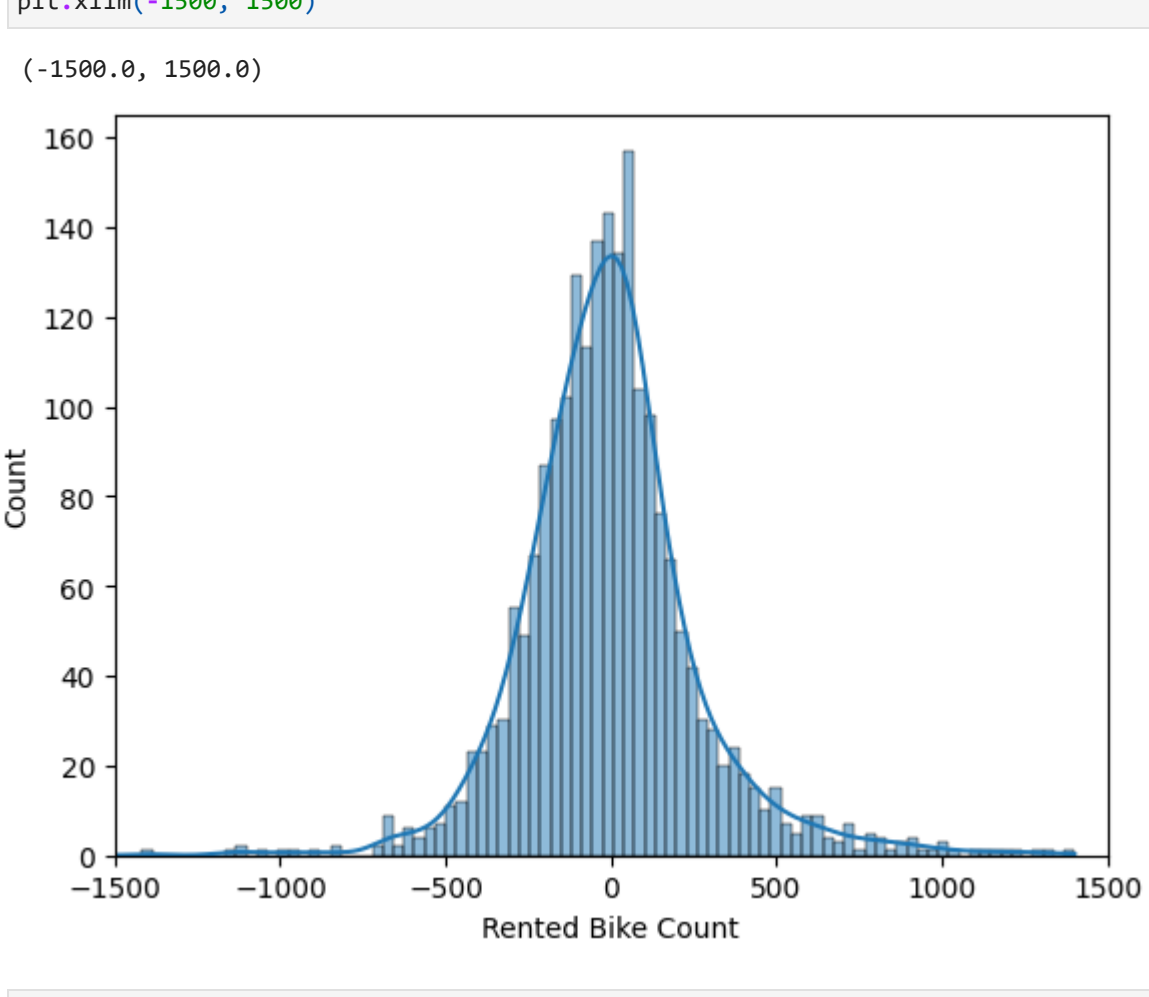
```
Adjusted R2 на тренировочной выборке: 0.8480049012864939
RMSE на тренировочной выборке: 251.93880488668086
Adjusted R2 на кросс-валидации: 0.8265823389967439
RMSE на кросс-валидации:: 268.60793015596613
```

In [26]:
```
y_pred_test = grid_search.predict(X_test)
r2_test = r2_score(y_test, y_pred_test)
adjusted_r2_test = 1 - ((1 - r2_test) * (n_train - 1)) / (n_train - k - 1)
print("Adjusted R2 на тестовой выборке:", adjusted_r2_test)
```

```
Adjusted R2 на тестовой выборке: 0.8212090403021672
```

In [27]:
```
# Проанализируем остатки, т.е. ошибки модели.
sns.histplot((y_test-y_pred_test), bins=100, kde=True)
plt.xlim(-1500, 1500)
```

Out[27]:
```
(-1500.0, 1500.0)
```



In [28]:
```
# Остатки распределены нормально, что говорит о хорошем качестве построения модели.
# Попробуем повысить метрику с помощью ансамбля Gradient Boosting.
```

4.3 Ансамбль GradientBoosting

In [29]:
```
# Используем тот же preprocessor, применим Grid Search для поиска лучших параметров градиентного б

gb_regressor = GradientBoostingRegressor(random_state=42)

model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('gb', gb_regressor)
])

param_grid = {
    'gb__n_estimators': [100, 200, 300],
    'gb__learning_rate': [0.01, 0.1, 0.2]
}
grid_search_gb = GridSearchCV(estimator=model, param_grid=param_grid, scoring='r2', cv=5)
grid_search_gb.fit(X_train, y_train)

best_params_gb = grid_search_gb.best_params_
best_r2_gb = grid_search_gb.best_score_

print("Best Parameters:", best_params_gb)
print("Best R-squared:", best_r2_gb)
```

```
Best Parameters: {'gb__learning_rate': 0.2, 'gb__n_estimators': 300}
Best R-squared: 0.9258454979834078
```

In [30]:
```
best_model_gb = grid_search_gb.best_estimator_
y_train_pred_gb = best_model_gb.predict(X_train)
train_r2_gb = r2_score(y_train, y_train_pred_gb)
adjusted_train_r2_gb = 1 - ((1 - train_r2_gb) * (n_train - 1)) / (n_train - k - 1)
train_rmse = np.sqrt(mean_squared_error(y_train, y_train_pred_gb))

cross_val_r2 = cross_val_score(best_model_gb, X_train, y_train, cv=7, scoring='r2')
adjusted_cross_val_r2 = 1 - ((1 - np.mean(cross_val_r2)) * (n_train - 1)) / (n_train - k - 1)
cross_val_rmse = np.sqrt(-cross_val_score(best_model_gb, X_train, y_train, cv=7, scoring='neg_mean

print("Adjusted R2 на тренировочной выборке:", adjusted_train_r2)
print("RMSE на тренировочной выборке:", train_rmse)
print("Adjusted R2 на кросс-валидации:", cross_val_r2.mean())
print("RMSE на кросс-валидации::", cross_val_rmse.mean())
```

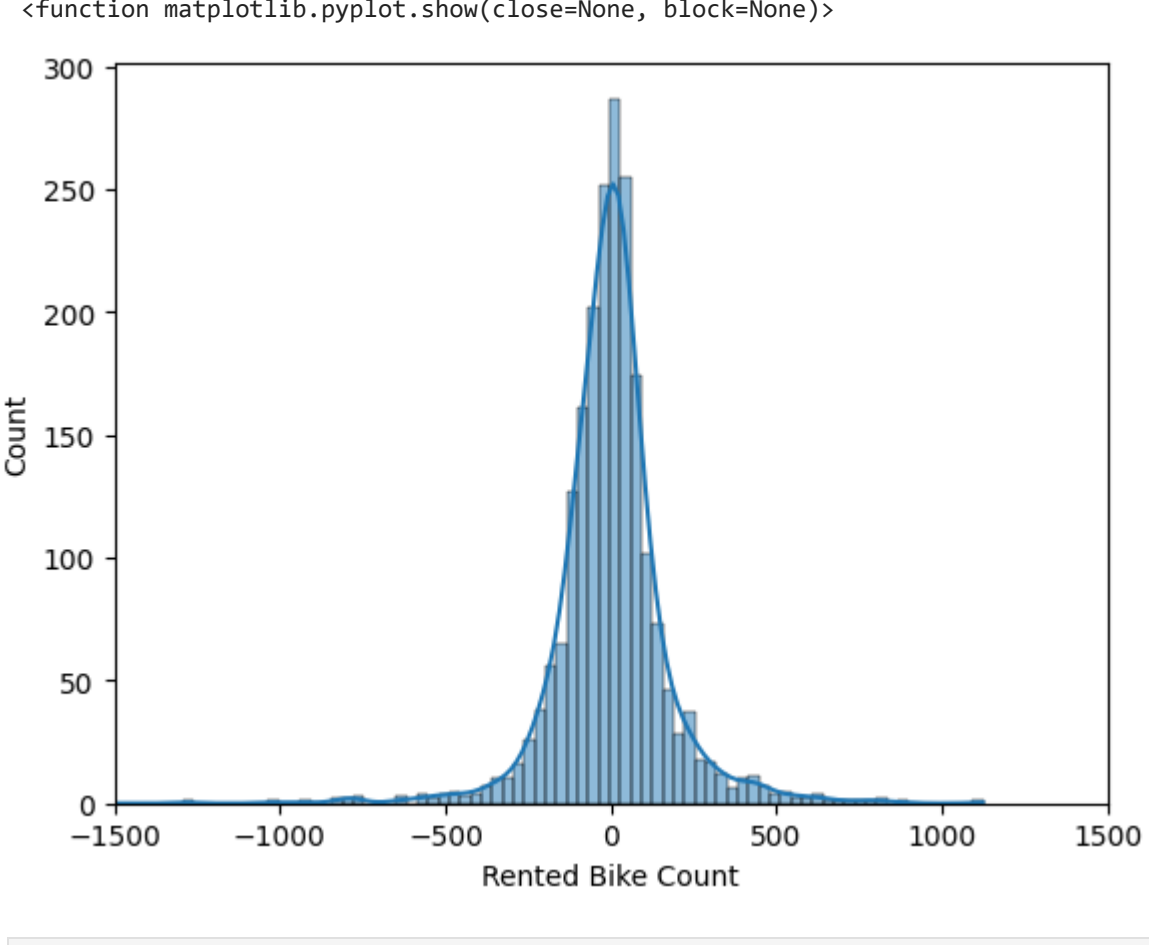
```
Adjusted R2 на тренировочной выборке: 0.8480049012864939
RMSE на тренировочной выборке: 137.44408661302612
Adjusted R2 на кросс-валидации: 0.9272212098201982
RMSE на кросс-валидации:: 174.10138639215168
```

In [31]:
```
y_pred_test_gb = grid_search_gb.predict(X_test)
r2_test_gb = r2_score(y_test, y_pred_test_gb)
adjusted_r2_test = 1 - ((1 - r2_test_gb) * (n_train - 1)) / (n_train - k - 1)
print("Adjusted R2 на тестовой выборке:", adjusted_r2_test)
```

```
Adjusted R2 на тестовой выборке: 0.9172202411776642
```

In [33]:
```
sns.histplot((y_test-y_pred_test_gb), bins=100, kde=True)
plt.xlim(-1500, 1500)
plt.show
```

Out[33]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



In [32]:
```
# Остатки по итоговой модели распределены нормально.
# Метрика оценки качества:
# 1) полиномиальная регрессия 3 степени adj.R2 = 0.821
# 2) ансамбль GradientBoosting adj.R2 = 0.917
```