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

ФАКУЛЬТЕТ «Радиотехнический»

КАФЕДРА ИУ-5 «Системы обработки информации и управления»

Лабораторная работа №2-3 по курсу

Технологии машинного обучения

Темы работ: "Обработка пропусков в данных, кодирование категориальных признаков, масштабирование данных."

"Подготовка обучающей и тестовой выборки, кросс-валидация и подбор гиперпараметров на примере метода ближайших соседей."

Выполнил: Лисин А. В.
Группа: РТ5-61Б

Дата выполнения: «__» __________ 2021 г.

Подпись: __________

Проверил:

Дата проверки: «__» __________ 2021 г.

Подпись: __________

Москва, 2021 г.

Содержание

Описание задания

Цель лабораторной работы №2: изучение способов предварительной обработки данных для дальнейшего формирования моделей.

Задание:

- Выбрать набор данных (датасет), содержащий категориальные признаки и пропуски в данных. Для выполнения следующих пунктов можно использовать несколько различных наборов данных (один для обработки пропусков, другой для категориальных признаков и т.д.)
- Для выбранного датасета (датасетов) на основе материалов лекции решить следующие задачи:
  - обработку пропусков в данных;
  - кодирование категориальных признаков;
  - масштабирование данных.

Цель лабораторной работы №3: изучение способов подготовки выборки и подбора гиперпараметров на примере метода ближайших соседей.

Задание:

- Выберите набор данных (датасет) для решения задачи классификации или регрессии.
- С использованием метода train_test_split разделите выборку на обучающую и тестовую.
- Обучите модель ближайших соседей для произвольно заданного гиперпараметра K. Оцените качество модели с помощью подходящих для задачи метрик.
- Произведите подбор гиперпараметра K с использованием GridSearchCV и/или RandomizedSearchCV и кросс-валидации, оцените качество оптимальной модели. Желательно использование нескольких стратегий кросс-валидации.
- Сравните метрики качества исходной и оптимальной моделей.

Ход выполнения работы

Лабораторная работа 2-3 Лисин РТ5-61Б

In [2]:
```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
sns.set(style="ticks")
```

In [3]:
```python
data = pd.read_csv('moves.csv', sep = ",")
```

Данные датасета:

1. Имя (Name)
2. Тип (Type)
3. Категория (Cat.)
4. Сила (Power)
5. Точность (Acc.)
6. Очки силы (сколько требуется потратить на способность)
7. CD прокачки техники (TM)
8. Эффект (Effect)
9. Вероятность действия (Prob. (%))

In [4]:
```python
data.isnull().sum()
```

Out[4]:
```
Name           0
Type           0
Cat.           0
Power        259
Acc.         160
PP             1
TM           506
Effect        32
Prob. (%)    468
dtype: int64
```

In [5]:
```python
data.shape
```

Out[5]:
```
(607, 9)
```

In [6]:
```python
data.dtypes
```

Out[6]:
```
Name          object
Type          object
Cat.          object
Power        float64
Acc.          object
PP           float64
TM            object
Effect        object
Prob. (%)    float64
dtype: object
```

In [7]:
```python
data.head()
```

Out[7]:

	Name	Type	Cat.	Power	Acc.	PP	TM	Effect	Prob. (%)
0	Absorb	GRASS	Special	20.0	100	25.0	NaN	User recovers half the HP inflicted on opponent.	NaN
1	Acid	POISON	Special	40.0	100	30.0	NaN	May lower opponent's Special Defense.	10.0
2	Acid Armor	POISON	Status	NaN	NaN	40.0	NaN	Sharply raises user's Defense.	NaN
3	Acid Spray	POISON	Special	40.0	100	20.0	NaN	Sharply lowers opponent's Special Defense.	100.0
4	Acrobatics	FLYING	Physical	55.0	100	15.0	TM62	Stronger when the user does not have a held item.	NaN

In [8]:
```python
data.isnull().sum()
```

Out[8]:
```
Name           0
Type           0
Cat.           0
Power        259
Acc.         160
PP             1
TM           506
Effect        32
Prob. (%)    468
dtype: int64
```

Преобразуем датасет к рабочему виду

In [9]:
```python
#Логично, что если в таблице нет данных о точности способности, значит она работает гарантированно
data['Acc.'][data['Acc.'].isnull()] = 100
#Аналогично с вероятностью действия способности
data['Prob. (%)'][data['Prob. (%)'].isnull()] = 100
#Также логично, что если в таблице нет данных о силе способности, то она не наносит урон, а значит равна нулю
data['Power'][data['Power'].isnull()] = 0
```

```
<ipython-input-9-fed872aca1b6>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  data['Acc.'][data['Acc.'].isnull()] = 100
<ipython-input-9-fed872aca1b6>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  data['Prob. (%)'][data['Prob. (%)'].isnull()] = 100
<ipython-input-9-fed872aca1b6>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  data['Power'][data['Power'].isnull()] = 0
```

In [10]:
```python
data.shape
```

Out[10]:
```
(607, 9)
```

In [11]:
```python
#Почему то в некоторых строках значение "точности" равнялось бесконечности, что не имеет никакого смысла.
for i in range(len(data['Acc.'])):
    if(data['Acc.'][i] == '∞'):
        data['Acc.'][i] = 100
```

```
<ipython-input-11-a6b7b03823ce>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  data['Acc.'][i] = 100
```

In [12]:
```python
data.isnull().sum()
```

Out[12]:
```
Name           0
Type           0
Cat.           0
Power          0
Acc.           0
PP             1
TM           506
Effect        32
Prob. (%)      0
dtype: int64
```

In [13]:
```python
data['Acc.'] = data['Acc.'].astype(str).astype(int)
```

In [14]:
```python
data[data['PP'].isnull()].head()
```

Out[14]:

	Name	Type	Cat.	Power	Acc.	PP	TM	Effect	Prob. (%)
516	Struggle	NORMAL	Physical	50.0	100	NaN	NaN	Only usable when all PP are gone. Hurts the user.	100.0

In [15]:
```python
#Единственное пустое значение для PP у способности, которая работает, когда значение PP кончилось
#Мы могли бы сделать его значение равно нулю, но это неправильно, потому что способность вместо этого тратит хі
#и поэтому она будет портить статистику, легче будет просто ее удалить
data = data.drop(516, 0)
```

In [16]:
```python
data.isnull().sum()
```

Out[16]:
```
Name           0
Type           0
Cat.           0
Power          0
Acc.           0
PP             0
TM           505
Effect        32
Prob. (%)      0
dtype: int64
```

In [17]:
```python
#TM имеет слишком много пропусков и не имеет значения, поэтому можем спокойно его удалить
#Также удалим Effect потому что это описания и после этого этапа они аткже не нужны
data = data.dropna(axis=1, how='any')
```

In [18]:
```python
#Столбец Name так же не имеет значения, поэтому удалим его
data = data.drop('Name', 1)
```

In [19]:
```python
data.isnull().sum()
```

Out[19]:
```
Type         0
Cat.         0
Power        0
Acc.         0
PP           0
Prob. (%)    0
dtype: int64
```

In [20]:
```python
data['PP'] = data['PP'].astype(int)
data['Power'] = data['Power'].astype(int)
data['Prob. (%)'] = data['Prob. (%)'].astype(int)
```

In [21]:
```python
data.dtypes
```

Out[21]:
```
Type         object
Cat.         object
Power         int64
Acc.          int64
PP            int64
Prob. (%)     int64
dtype: object
```

Кодирование категориальных признаков

In [22]:
```python
data = pd.get_dummies(data)
```

In [23]:
```python
data.head()
```

Out[23]:

	Power	Acc.	PP	Prob. (%)	Type_BUG	Type_DARK	Type_DRAGON	Type_ELECTRIC	Type_FAIRY	Type_FIGHTING	...	Type_ICE	Type_NO
0	20	100	25	100	0	0	0	0	0	0	...	0	
1	40	100	30	10	0	0	0	0	0	0	...	0	
2	0	100	40	100	0	0	0	0	0	0	...	0	
3	40	100	20	100	0	0	0	0	0	0	...	0	
4	55	100	15	100	0	0	0	0	0	0	...	0	

5 rows × 25 columns

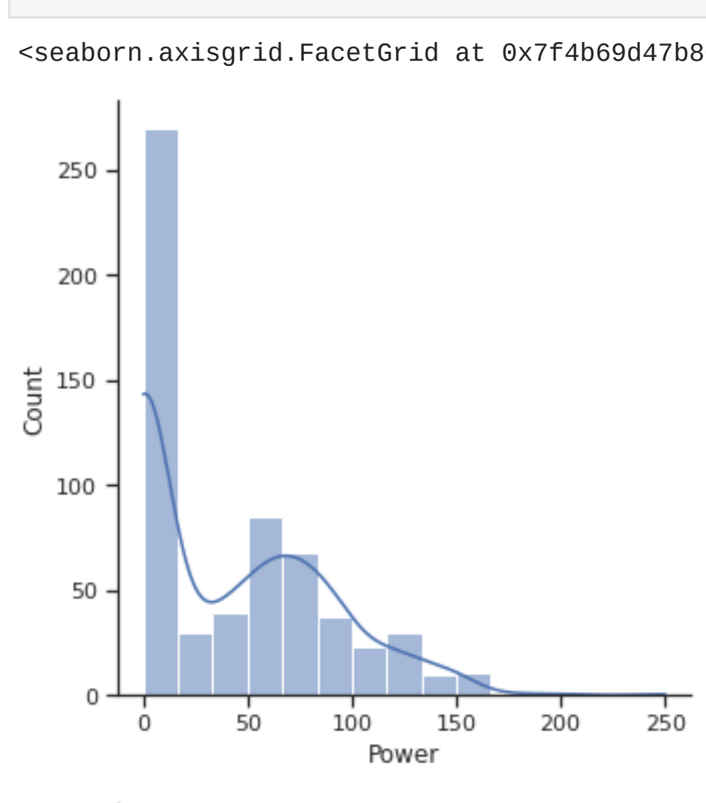
Масштабирование данных

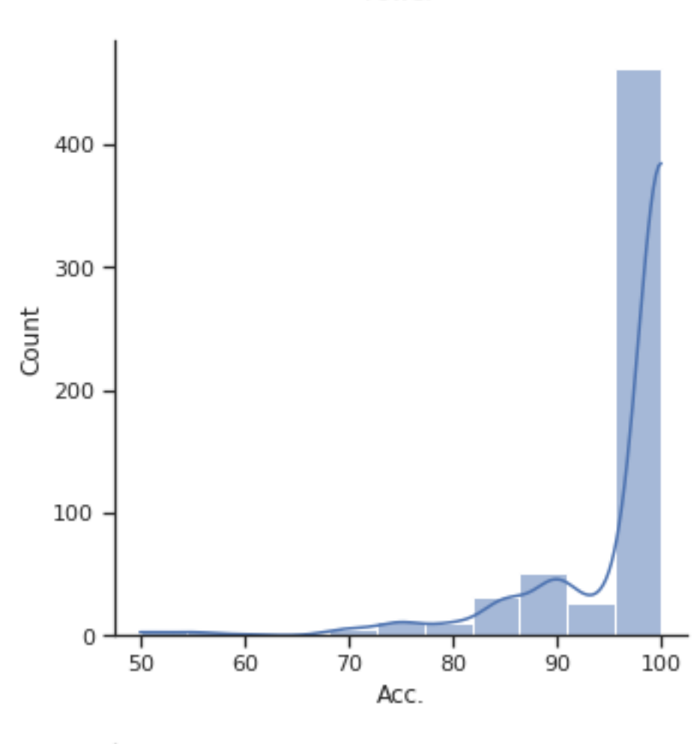
In [24]:
```python
from sklearn.preprocessing import MinMaxScaler, StandardScaler, Normalizer
```

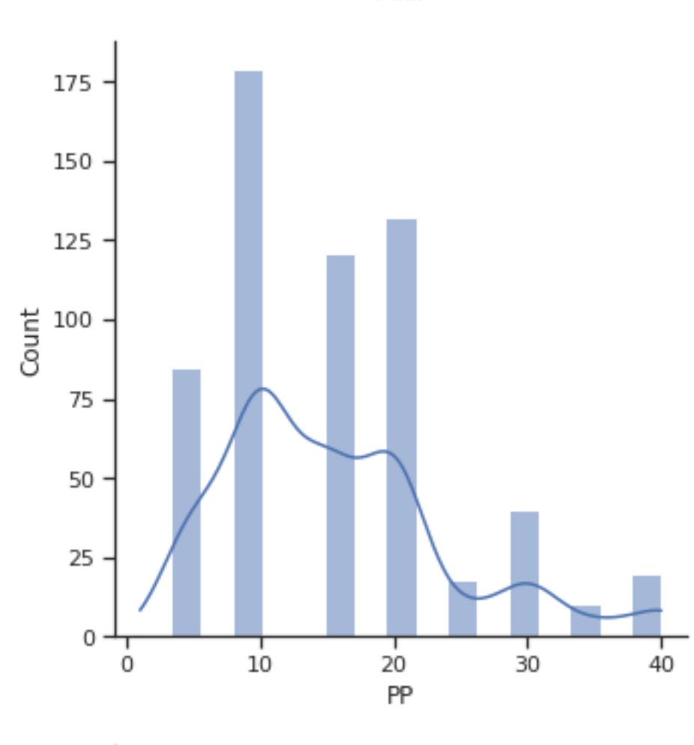
In [25]:
```python
data_unscaled = data.copy()
```

In [26]:
```python
sns.displot(data=data, x="Power", kde = True)
sns.displot(data=data, x="Acc.", kde = True)
sns.displot(data=data, x="PP", kde = True)
sns.displot(data=data, x="Prob. (%)", kde = True)
```

Out[26]:
```
<seaborn.axisgrid.FacetGrid at 0x7f4b69d47b80>
```







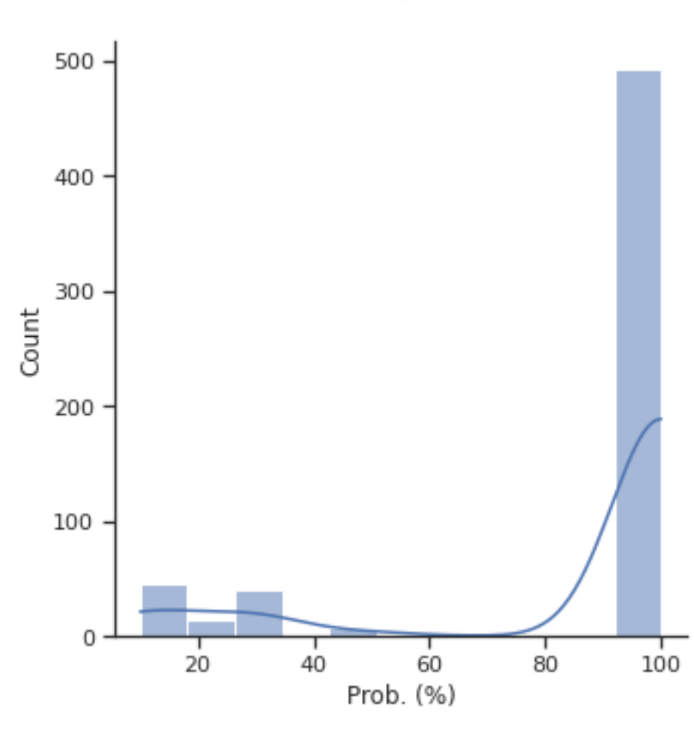
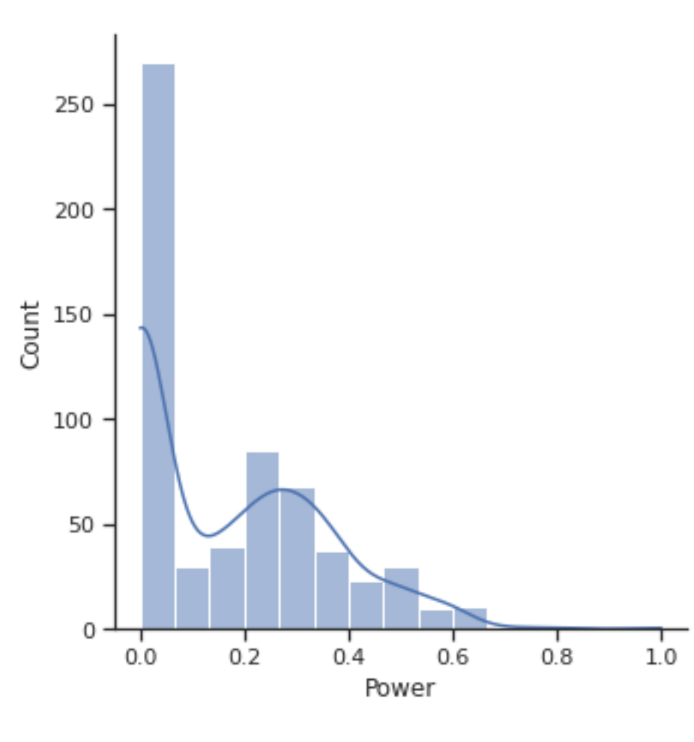


График Power соответствует нормальному распределению, за исключением перекоса на нулевом значении, поэтому применим MinMaxScaler

In [27]:
```python
from sklearn.preprocessing import MinMaxScaler, StandardScaler
mms = MinMaxScaler()
Power = mms.fit_transform(data[['Power']])
data["Power"] = Power
```

In [28]:
```python
sns.displot(data=data, x="Power", kde = True)
```

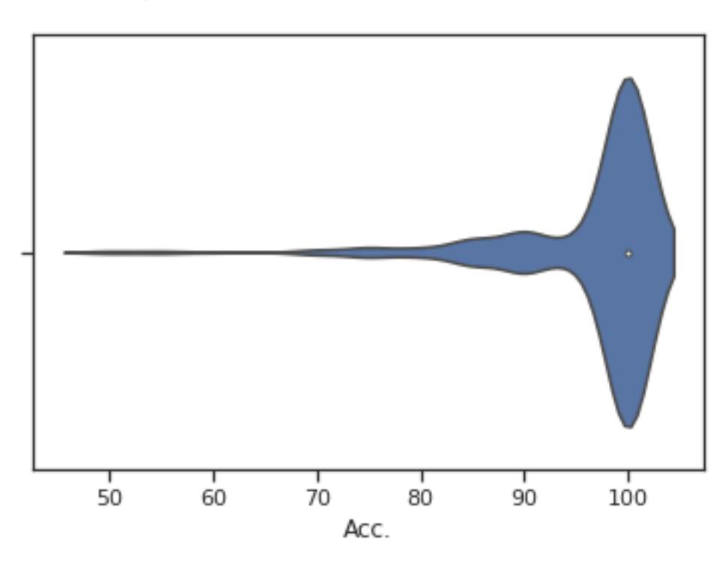
Out[28]:
```
<seaborn.axisgrid.FacetGrid at 0x7f4b69b7d370>
```



Рассмотрим Acc.

In [29]:
```python
sns.violinplot(data=data, x="Acc.", kde = True)
```

Out[29]:
```
<AxesSubplot:xlabel='Acc.'>
```



Видим "хвост", поэтому используем Z-оценку

In [30]:
```python
ss = StandardScaler()
Accuracy = ss.fit_transform(data[['Acc.']])
data["Acc."] = Accuracy
```

In [31]:
```python
sns.violinplot(data=data, x="Acc.", kde = True)
```

Out[31]:
```
<AxesSubplot:xlabel='Acc.'>
```

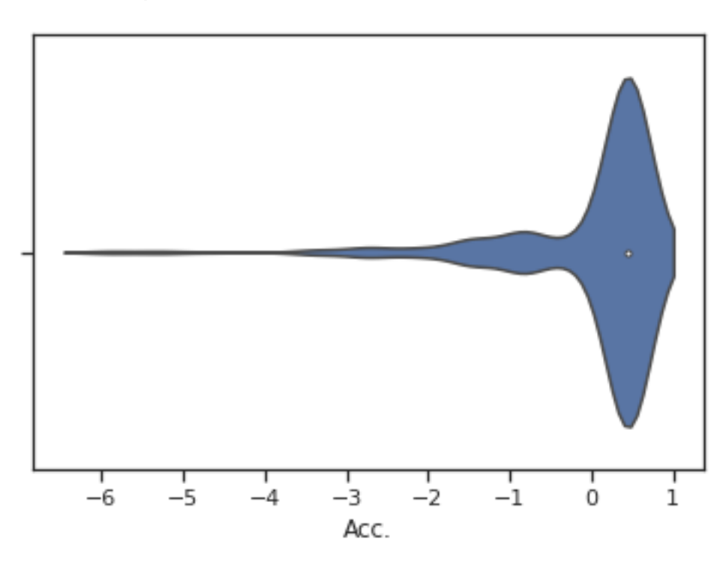
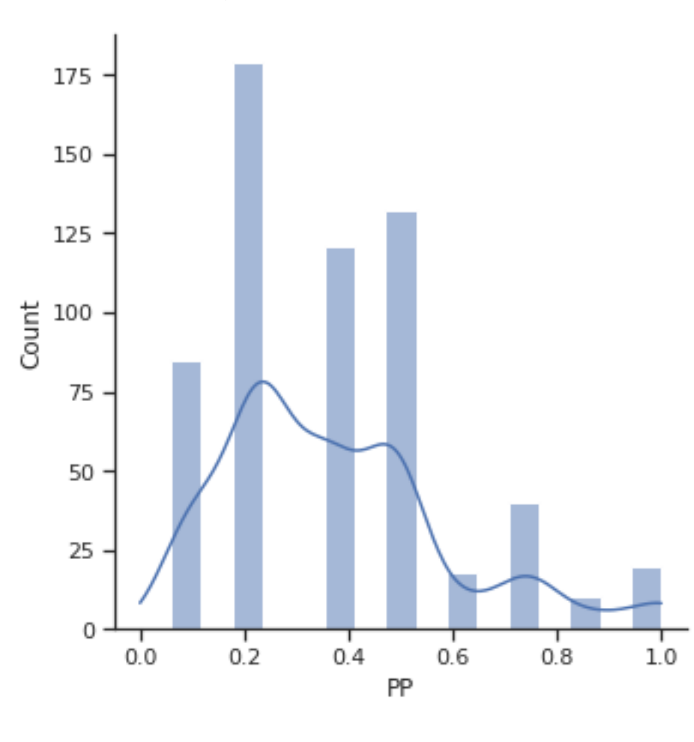


График PP соответствует нормальному распределению, поэтому применим MinMaxScaler

In [32]:
```python
mms = MinMaxScaler()
PP = mms.fit_transform(data[['PP']])
data["PP"] = PP
```

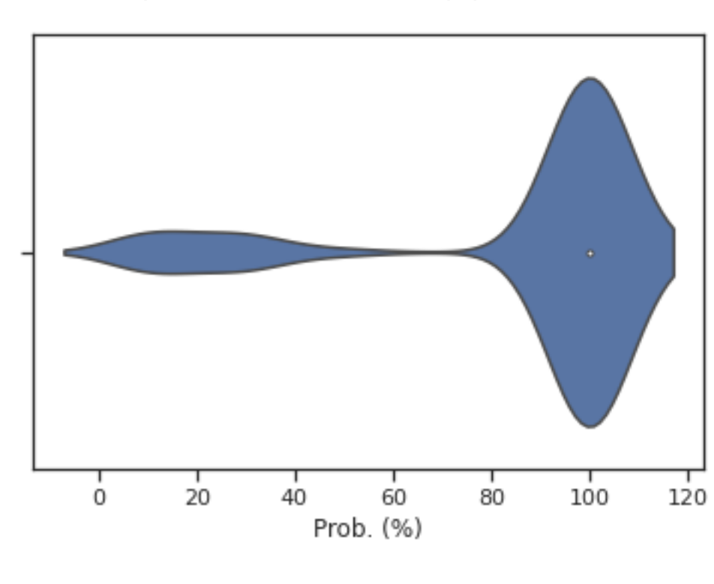
In [33]:
```python
sns.displot(data=data, x="PP", kde = True)
```

Out[33]:
```
<seaborn.axisgrid.FacetGrid at 0x7f4b69d87340>
```



In [34]:
```python
sns.violinplot(data=data, x="Prob. (%)", kde = True)
```

Out[34]:
```
<AxesSubplot:xlabel='Prob. (%)'>
```

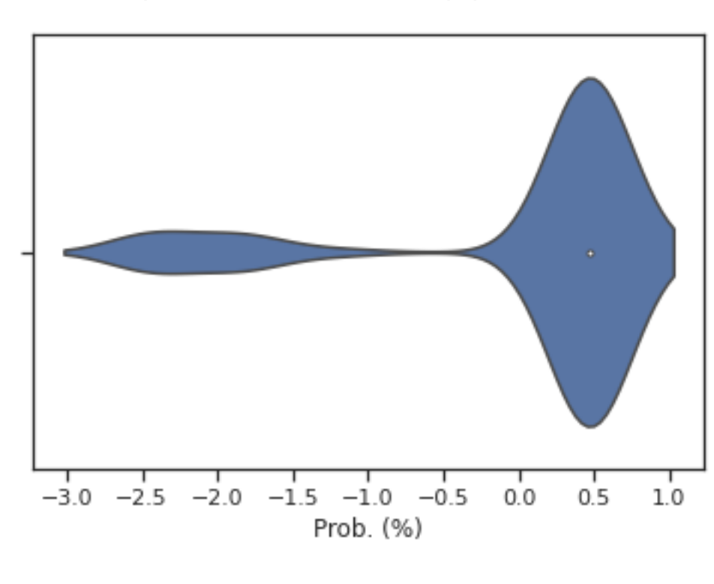


Нормального распределения не наблюдается, используем Z-оценку

In [35]:
```python
ss = StandardScaler()
Probability = ss.fit_transform(data[['Prob. (%)']])
data["Prob. (%)"] = Probability
```

In [36]:
```python
sns.violinplot(data=data, x="Prob. (%)", kde = True)
```

Out[36]:
```
<AxesSubplot:xlabel='Prob. (%)'>
```



Итоговый вид набора данных

In [37]:
```python
data.head()
```

Out[37]:

	Power	Acc.	PP	Prob. (%)	Type_BUG	Type_DARK	Type_DRAGON	Type_ELECTRIC	Type_FAIRY	Type_FIGHTING	...	Type_IC
0	0.08	0.44533	0.615385	0.470806	0	0	0	0	0	0	...	
1	0.16	0.44533	0.743590	-2.464131	0	0	0	0	0	0	...	
2	0.00	0.44533	1.000000	0.470806	0	0	0	0	0	0	...	
3	0.16	0.44533	0.487179	0.470806	0	0	0	0	0	0	...	
4	0.22	0.44533	0.358974	0.470806	0	0	0	0	0	0	...	

5 rows × 25 columns

In [38]:
```python
data_scaled = data.copy()
```

ЛР3

In [39]:
```python
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV, KFold, train_test_split
```

In [40]:
```python
columns = data_scaled.columns.tolist()
column = columns.pop(columns.index("PP"))
columns.append(column)

data_scaled = data_scaled[columns]
```

In [41]:
```python
data_scaled.head()
```

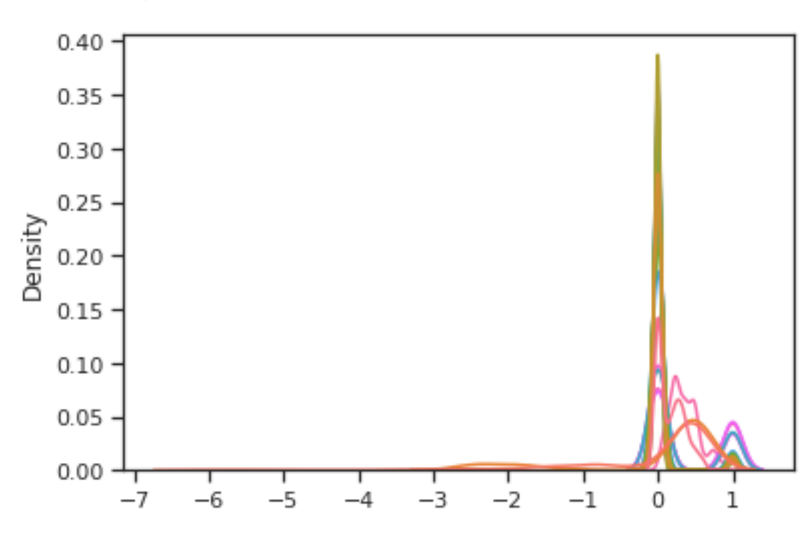
Out[41]:

	Power	Acc.	Prob. (%)	Type_BUG	Type_DARK	Type_DRAGON	Type_ELECTRIC	Type_FAIRY	Type_FIGHTING	Type_FIRE	...	Type_
0	0.08	0.44533	0.470806	0	0	0	0	0	0	0	...	
1	0.16	0.44533	-2.464131	0	0	0	0	0	0	0	...	
2	0.00	0.44533	0.470806	0	0	0	0	0	0	0	...	
3	0.16	0.44533	0.470806	0	0	0	0	0	0	0	...	
4	0.22	0.44533	0.470806	0	0	0	0	0	0	0	...	

5 rows × 25 columns

In [42]:
```python
sns.kdeplot(data = data_scaled, legend = False)
```

Out[42]:
```
<AxesSubplot:ylabel='Density'>
```



Разделим выборку

In [43]:
```python
y_column = "PP"
x_columns = data_scaled.columns.tolist()
x_columns.pop(x_columns.index(y_column))

data_scaled_x_train, data_scaled_x_test, data_scaled_y_train, data_scaled_y_test = train_test_split(data_scaled
```

In [44]:
```python
from sklearn.neighbors import KNeighborsRegressor


knn_scaled = KNeighborsRegressor(n_neighbors = 15)

knn_scaled.fit(data_scaled_x_train, data_scaled_y_train)

knn_scaled_prediction = knn_scaled.predict(data_scaled_x_test)
```

Обучим модель для произвольного гиперпараметра

In [46]:
```python
from sklearn.metrics import mean_absolute_error, mean_squared_error, median_absolute_error, r2_score
from sklearn.model_selection import ShuffleSplit, cross_val_score, cross_validate


print('Средняя абсолютная ошибка:',   mean_absolute_error(data_scaled_y_test, knn_scaled_prediction))
print('Медианная абсолютная ошибка:',   median_absolute_error(data_scaled_y_test, knn_scaled_prediction))
print('Среднеквадратичная ошибка:',   mean_squared_error(data_scaled_y_test, knn_scaled_prediction, squared = F
print('Коэффициент детерминации:',   r2_score(data_scaled_y_test, knn_scaled_prediction))
```

```
Средняя абсолютная ошибка: 0.16896642876024318
Медианная абсолютная ошибка: 0.12820512820512825
Среднеквадратичная ошибка: 0.22033331013001045
Коэффициент детерминации: 0.005385464252665173
```

Подбор гиперпараметров

GridSearch через квадратичную ошибку

In [47]:
```python
from sklearn.model_selection import GridSearchCV
n_range = np.array(range(1, 51, 1))
tuned_parameters = [{'n_neighbors': n_range}]
gs = GridSearchCV(KNeighborsRegressor(), tuned_parameters, cv=10, scoring='neg_mean_squared_error')
gs.fit(data_scaled_x_train, data_scaled_y_train)
```

Out[47]:
```
GridSearchCV(cv=10, estimator=KNeighborsRegressor(),
             param_grid=[{'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
       35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50])}],
             scoring='neg_mean_squared_error')
```

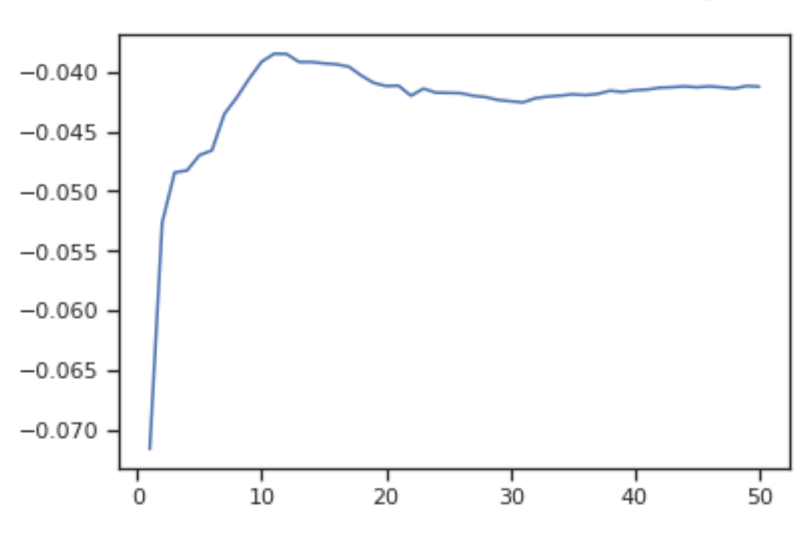
In [48]:
```python
print('Лучшая модель:', gs.best_estimator_)
print('\nЛучшее число ближайших соседей:',gs.best_params_)
print('\nЛучшее значение средней квадратичной ошибки:',gs.best_score_)
```

```
Лучшая модель: KNeighborsRegressor(n_neighbors=11)

Лучшее число ближайших соседей: {'n_neighbors': 11}

Лучшее значение средней квадратичной ошибки: -0.03849163209034105
```

In [49]:
```python
print('Изменение качества тестовой выборки в зависимости от кол-ва соседей:\n')
plt.plot(n_range, gs.cv_results_['mean_test_score'])
```

```
Изменение качества тестовой выборки в зависимости от кол-ва соседей:
```

Out[49]:
```
[<matplotlib.lines.Line2D at 0x7f4b6804fcd0>]
```

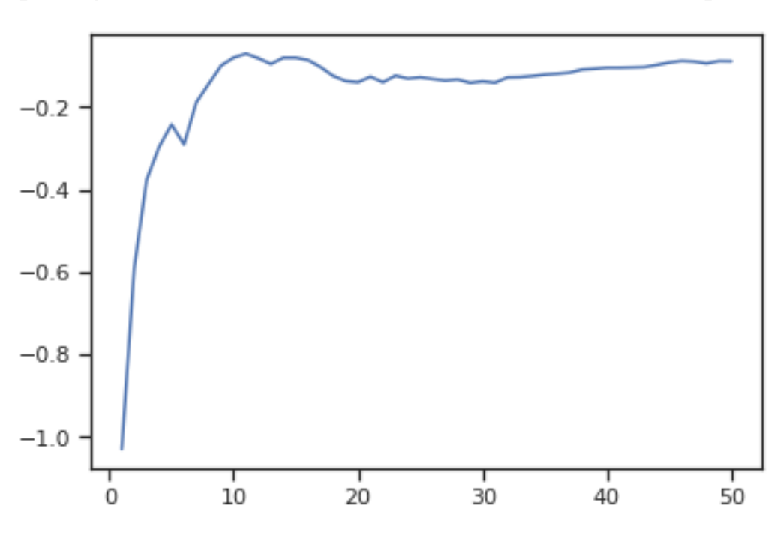


GridSearch через коэффициент детерминации

In [50]:
```python
gs_det = GridSearchCV(KNeighborsRegressor(), tuned_parameters, cv=10, scoring='r2')
gs_det.fit(data_scaled_x_train, data_scaled_y_train)
print('Лучшая модель:', gs_det.best_estimator_)
print('\nЛучшее число ближайших соседей:',gs_det.best_params_)
print('\nЛучшее значение коэффициента детерминации:',gs_det.best_score_)
print('\nИзменение качества тестовой выборки в зависимости от кол-ва соседей:\n')
plt.plot(n_range, gs_det.cv_results_['mean_test_score'])
```

```
Лучшая модель: KNeighborsRegressor(n_neighbors=11)

Лучшее число ближайших соседей: {'n_neighbors': 11}

Лучшее значение коэффициента детерминации: -0.0709306186626206

Изменение качества тестовой выборки в зависимости от кол-ва соседей:
```

Out[50]:
```
[<matplotlib.lines.Line2D at 0x7f4b609c68b0>]
```



Кросс-валидация

In [51]:
```python
from sklearn.model_selection import cross_val_score
scores = cross_val_score(KNeighborsRegressor(n_neighbors = 11),
                         data_scaled[x_columns], data_scaled[y_column],
                         scoring = "r2",
                         cv = ShuffleSplit(n_splits = 8,
                                           test_size = 0.8,
                                           random_state = 11))
```

In [52]:
```python
print('Усредненное значение коэффицента детерминации для оптимальной модели: ', np.mean(scores))
```

```
Усредненное значение коэффицента детерминации для оптимальной модели:  0.035944788846929754
```

Коэффицент детерминации стал меньше, но все еще больше нуля