
Примеры кода метрик

Просмотр python-кода имеющихся в системе метрик доступен из интерфейса приложения:

1. На странице каталога метрик (*Панель управления > Метрики*) выберите нужную метрику, наведите курсор на символ меню (три точки) в правой части соответствующей строки и нажмите "Просмотр".
2. Откроется экранная форма с информацией о метрике. Прокрутите страницу вниз и нажмите кнопку "Открыть код" в правом нижнем углу экранной формы.

Пример скалярной метрики со светофором без графика

Указаны флаги `is_scalar = True` и `is_signal = True`. Методы `scalar` и `signal` объявлены и имплементированы. Методы `__call__` и `save` объявлены, но имплементация пропущена.

```
from typing import Literal

import pandas as pd
from sklearn.metrics import mean_absolute_percentage_error

class rvc_3_MAPE:
    """
    Средняя абсолютная ошибка в процентах

    Attributes:
        __desc__ (str): Description of the class.
        __tags__ (list[str]): List of tags associated with the class.
        is_scalar (bool): Whether the metric is scalar or not.
        is_signal (bool): Whether the metric has signal or not.

    """

    __desc__ = "Mean Absolute Percentage Error (MAPE). Средняя абсолютная  
ошибка в процентах"
    __tags__ = ["core", "regression", "scalar"]
    is_scalar = True
    is_signal = True

    def __init__(
        self,
        df: pd.DataFrame,
        predict_column: str,
        target_column: str,
```

```

        threshold_yellow: float = 0.3,
        threshold_red: float = 0.4,
    ):
        if df.empty:
            raise Exception("Dataframe is empty")
        if target_column not in df:
            raise ValueError(f"Field {target_column} does not exist in the
dataframe")
        if predict_column not in df:
            raise ValueError(f"Field {predict_column} does not exist in the
dataframe")

        self.predict_column = predict_column
        self.target_column = target_column
        self.df = df.astype({self.predict_column: "float", self.target_column:
"float"})
        self.threshold_yellow = threshold_yellow
        self.threshold_red = threshold_red

    def __call__(self) -> None:
        pass

    def scalar(self) -> int | float:
        df = self.df.loc[:, [self.target_column,
self.predict_column]].dropna()[
            abs(self.df[self.target_column]) > 0
        ]

        self.scalar_value = mean_absolute_percentage_error(
            y_pred=df[self.predict_column],
            y_true=df[self.target_column],
        )

        return self.scalar_value

    def signal(self) -> Literal["red", "yellow", "green"]:
        signal_light = "green"

        if self.scalar_value > self.threshold_red:
            signal_light = "red"
        elif self.scalar_value > self.threshold_yellow:
            signal_light = "yellow"

        return signal_light

    def save(self, output_dir: str) -> dict[str, str] | None:
        pass

```

Результат:

MAPE_metric GREEN

0.16

Пример скалярной метрики со светофором и с графиком

Указаны флаги `is_scalar = True` и `is_signal = True`. Методы `scalar` и `signal` объявлены и имплементированы. Методы `__call__` и `save` объявлены и имплементированы.

```
from typing import Any, Dict, Literal, Optional

import numpy as np
import pandas as pd
import plotly.graph_objects as go

class r_2_5_KS_on_scale:
    """
    Тест Колмогорова-Смирнова

    Показывает насколько хорошо score модели
    отделяет "хороших" клиентов от "плохих" в разрезе рейтинговой шкалы.

    Attributes:
        __desc__ (str): Description of the class.
        __tags__ (list[str]): List of tags associated with the class.
        is_scalar (bool): Whether the metric is scalar or not.
        is_signal (bool): Whether the metric has signal or not.

    """

    __desc__ = "KS-test on scale. Тест Колмогорова-Смирнова"
    __tags__ = ["risk", "scalar"]

    is_scalar = True
    is_signal = True

    def __init__(
        self,
        df: pd.DataFrame,
        scale_column: str,
        target_column: str,
        threshold_yellow: float = 10,
        threshold_red: float = 30,
    ):
        self.scale_column = scale_column
        self.target_column = target_column
        self.df = df.astype({self.target_column: "float"})
        self.threshold_yellow = threshold_yellow
```

```

self.threshold_red = threshold_red

if self.df.empty:
    raise Exception("Dataframe is empty")
if self.target_column not in self.df:
    raise ValueError(f"Field {self.target_column} does not exist in
the dataframe")
if self.scale_column not in self.df:
    raise ValueError(f"Field {self.scale_column} does not exist in the
dataframe")
if self.df[self.scale_column].nunique() > 100:
    raise Exception("Ошибка: переменная scale не является
категориальной")

def __call__(self) -> None:
    dataset = self.df.loc[:, [self.target_column,
self.scale_column]].dropna()

    # номер разряда рейтинговой шкалы
    # (способ получения зависит от формата данных в столбце self.scale)
    dataset["bin_number"] = dataset[self.scale_column].map(
        lambda x: int(x.split("_")[-1]))
    ) # dataset[self.scale].astype('category').cat.codes#

    dataset = dataset.sort_values(by=["bin_number"], ascending=False)

    good_cnt = dataset[dataset[self.target_column] == 0].shape[0]
    bad_cnt = dataset[dataset[self.target_column] == 1].shape[0]

    gr_bad = (
        pd.DataFrame(
            dataset.groupby("bin_number", observed=False)
[self.target_column].sum()
            ).cumsum()
            / bad_cnt
        )
    dataset["target_inverse"] = np.where(dataset[self.target_column] == 1,
0, 1)
    gr_good = (
        pd.DataFrame(
            dataset.groupby("bin_number", observed=False)
["target_inverse"].sum()
            ).cumsum()
            / good_cnt
        )

    ks_calc_temp = pd.merge(gr_good, gr_bad, how="left", left_index=True,
right_index=True)
    ks_calc_temp["diff"] = 0
    ks_calc_temp["diff"] = abs(
        ks_calc_temp.iloc[:, 0:1].values - ks_calc_temp.iloc[:,
1:2].values
    )

    self.scalar_value = 100 * ks_calc_temp["diff"].max()
    ks_result = round(self.scalar_value, 2)
    result_idx = ks_calc_temp["diff"].argmax()

```

```

x_value1 = gr_bad.index.values.tolist()
y_value1 = gr_bad[self.target_column].values.astype("float").tolist()
x_value2 = gr_good.index.values.tolist()
y_value2 = gr_good["target_inverse"].values.astype("float").tolist()
x_value3 = [
    float(gr_bad.index.values[result_idx]),
    float(gr_bad.index.values[result_idx]),
]
y_value3 = [
    float(gr_bad[self.target_column].values[result_idx]),
    float(gr_good["target_inverse"].values[result_idx]),
]

line1 = go.Scatter(
    mode="lines",
    x=x_value1,
    y=y_value1,
    name="bad",
    line={"width": 3},
    marker={"color": "#63666A"},
)
line2 = go.Scatter(
    mode="lines",
    x=x_value2,
    y=y_value2,
    name="good",
    line={"width": 3},
    marker={"color": "#3eb489"},
)
line3 = go.Scatter(
    mode="lines",
    x=x_value3,
    y=y_value3,
    name=f"KS-statistic = {ks_result.astype('float')}",
    marker={"color": "black"},
)
self.fig = go.Figure(data=[line1, line2, line3])
self.fig.layout = self.custom_layout()

def scalar(self) -> int | float:
    return self.scalar_value

def signal(self) -> Literal["red", "yellow", "green"]:
    signal_light = "green"

    if self.scalar_value > self.threshold_red:
        signal_light = "red"
    elif self.scalar_value > self.threshold_yellow:
        signal_light = "yellow"

    return signal_light

def custom_layout(self) -> Optional[Dict[str, Any]]:
    return {
        "title": {"text": "<b>Тест Колмогорова-Смирнова</b>", "x": 0.1,
"y": 0.97},

```

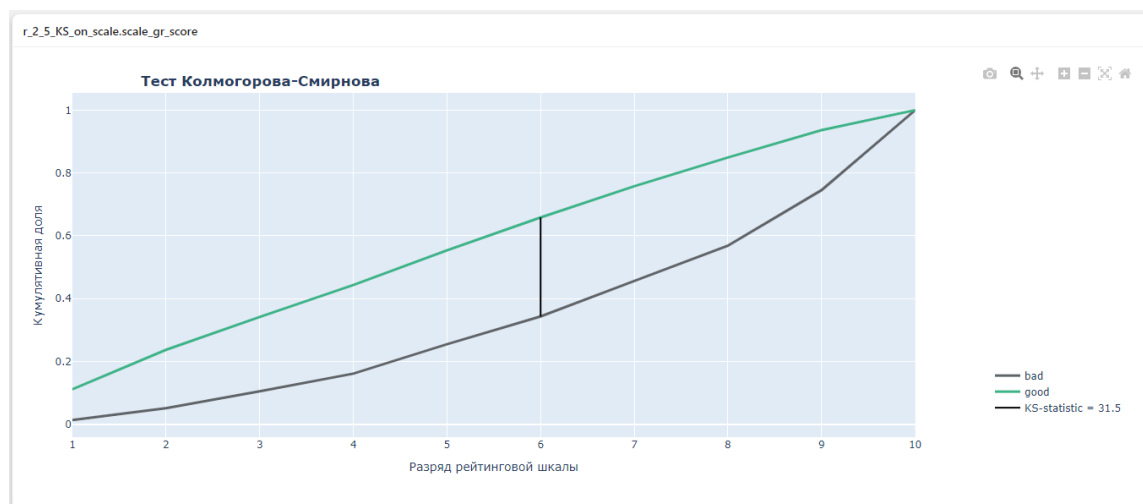
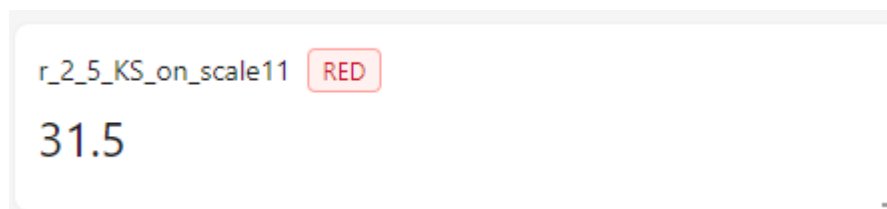
```

"x": 1},
    "legend": {"yanchor": "bottom", "y": 0.05, "xanchor": "right",
    "yaxis": {"title": "Кумулятивная доля", "side": "left"},
    "xaxis": {
        "title": "Разряд рейтинговой шкалы",
        "side": "left",
        "type": "category",
        "domain": [0, 0.8],
    },
    "margin": {"t": 35, "b": 5, "l": 5, "r": 5},
}

def save(self, output_dir: str) -> dict[str, str] | None:
    self.fig.write_html(
        f"{output_dir}/data.html",
        config={"displaylogo": False}, # remove the plotly logo
    )
    return {f"scale_{self.scale_column}": f"{output_dir}/data.html"}

```

Результат:



Пример кода метрики с графиком, но без скаляра и светофора

Указаны флаги `is_scalar = False` и `is_signal = False`. Методы `scalar` и `signal` не объявлены. Методы `__call__` и `save` объявлены и имплементированы.

```

from typing import Any, Dict, Optional

import numpy as np
import pandas as pd

```

```

import plotly.graph_objects as go
from scipy.stats import norm

class r_6_2_Binomial_test:
    """
    Проверяет попадание среднего уровня дефолта по бакеу рейтинговой шкалы в
    доверительный интервал, построенный по скорам модели

    Attributes:
        __desc__ (str): Description of the class.
        __tags__ (list[str]): List of tags associated with the class.
        is_scalar (bool): Whether the metric is scalar or not.
        is_signal (bool): Whether the metric has signal or not.

    """

    __desc__ = "Binomial Test. Биномиальный тест"
    __tags__ = ["risk"]

    is_scalar = False
    is_signal = False

    def __init__(
        self,
        df: pd.DataFrame,
        predict_column: str,
        target_column: str,
        scale_column: str,
        confidence_level: float = 0.99,
    ):
        if predict_column not in df.columns:
            raise ValueError(
                f"Invalid column name for 'predict_column'. "
                f"There is not colomn '{predict_column}' in the dataframe"
            )
        if target_column not in df.columns:
            raise ValueError(
                f"Invalid column name for 'target_column'. "
                f"There is not colomn '{target_column}' in the dataframe"
            )
        if scale_column not in df.columns:
            raise ValueError(
                f"Invalid column name for 'scale_column'. "
                f"There is not colomn '{scale_column}' in the dataframe"
            )

        self.predict_column = predict_column
        self.target_column = target_column
        self.scale_column = scale_column
        self.df = df.astype({self.predict_column: "float", self.target_column:
"float"})
        self.confidence_level = confidence_level

        if self.df.empty:
            raise ValueError("Dataframe is empty")
        if self.df[self.scale_column].nunique() > 100:

```

```

        raise Exception("Ошибка: переменная scale не является
категориальной")

    def __call__(self) -> None:
        data_gr = (
            self.df[[self.scale_column, self.target_column,
self.predict_column]]
                .groupby([self.scale_column], observed=False)
                .agg({self.target_column: ["sum", "count"], self.predict_column:
["mean"]})
                .reset_index()
        )

        data_gr.columns = [self.scale_column, self.target_column, "cnt_all",
self.predict_column]
        data_gr["target_prc"] = data_gr[self.target_column] /
data_gr["cnt_all"]

        data_gr["CI_LEFT"] = data_gr[self.predict_column] - norm.ppf(
            self.confidence_level
        ) * np.sqrt(
            (data_gr[self.predict_column] * (1 -
data_gr[self.predict_column])) / data_gr["cnt_all"]
        )
        data_gr["CI_RIGHT"] = data_gr[self.predict_column] + norm.ppf(
            self.confidence_level
        ) * np.sqrt(
            (data_gr[self.predict_column] * (1 -
data_gr[self.predict_column])) / data_gr["cnt_all"]
        )

        data_gr["color"] = data_gr.apply(
            lambda x: "green"
            if (x["target_prc"] >= x["CI_LEFT"]) & (x["target_prc"] <=
x["CI_RIGHT"])
            else "red",
            axis=1,
        )

        data_gr = data_gr.sort_values(self.scale_column, key=lambda x:
x.str[-3:])
        # упорядочивание выше - под конкретный df,
        # обращать внимание на формат записей в столбце scale
        # при запуске на новых данных

        line1 = go.Scatter(
            mode="lines",
            x=data_gr[self.scale_column].tolist(),
            y=data_gr["CI_RIGHT"].tolist(),
            name="Верхняя граница ДИ",
            marker={"color": "#23654D"},
            xaxis="x1",
            yaxis="y1",
        )
        line2 = go.Scatter(
            mode="lines",
            x=data_gr[self.scale_column].tolist(),

```

```

        y=data_gr["CI_LEFT"].tolist(),
        name="Нижняя граница ДИ",
        marker={"color": "#23654D"},
        xaxis="x1",
        yaxis="y1",
    )
    line3 = go.Scatter(
        mode="markers",
        x=data_gr[self.scale_column].tolist(),
        y=data_gr["target_prc"].tolist(),
        name="Фактическая вероятность дефолта",
        marker={"color": data_gr["color"].tolist(), "size": 24},
        xaxis="x1",
        yaxis="y1",
    )

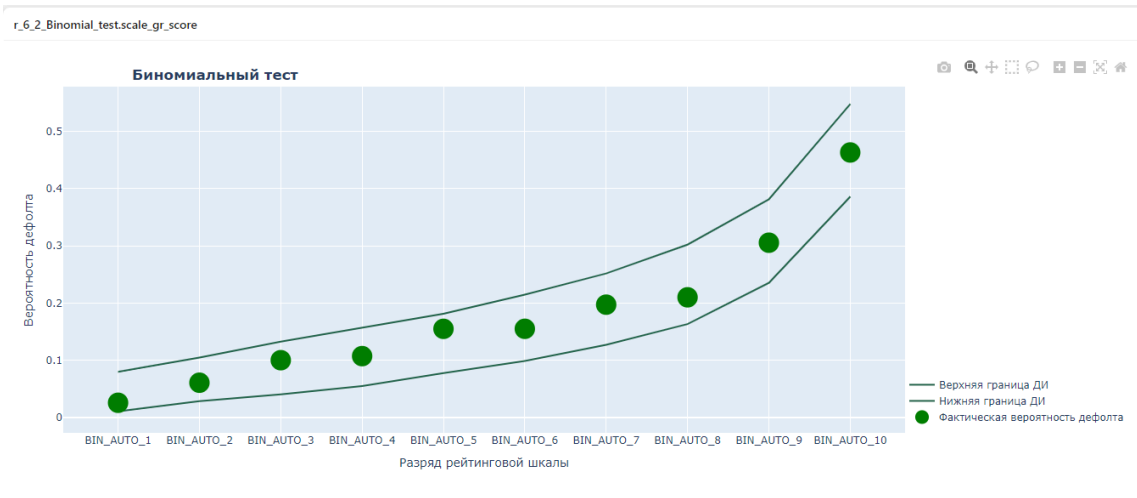
    self.fig = go.Figure(data=[line1, line2, line3])
    self.fig.layout = self.custom_layout()

    def custom_layout(self) -> Optional[Dict[str, Any]]:
        return {
            "title": {"text": "<b>Биномиальный тест</b>", "x": 0.1, "y":
0.97},
            "legend": {"yanchor": "bottom", "y": 0.01, "xanchor": "left", "x":
1},
            "yaxis": {"title": "Вероятность дефолта", "side": "left"},
            "xaxis": {
                "title": "Разряд рейтинговой шкалы",
                "side": "right",
                "type": "category",
                "domain": [0, 1],
            },
            "margin": {"t": 35, "b": 5, "l": 5, "r": 5},
        }

    def save(self, output_dir: str) -> dict[str, str] | None:
        self.fig.write_html(
            f"{output_dir}/data.html",
            config={"displaylogo": False}, # remove the plotly logo
        )
        return {f"scale_{self.scale_column}": f"{output_dir}/data.html"}

```

Результат:



Пример кода метрики с множеством графиков

Методы `__call__` и `save` объявлены и имплементированы.

В методе `save` графики создаются в цикле

```
from typing import Any, Dict, Optional

import pandas as pd
import plotly.graph_objects as go

class cd_2_4_Density_Distr_features:
    """
    Плотность распределения для выбранных полей
    Attributes:
        __desc__ (str): Description of the class.
        __tags__ (list[str]): List of tags associated with the class.
        is_scalar (bool): Whether the metric is scalar or not.
        is_signal (bool): Whether the metric has signal or not.
    """

    __desc__ = (
        "Density Distribution for Selected Columns. Плотность распределения для выбранных полей"
    )
    __tags__ = ["core", "data"]

    is_scalar = False
    is_signal = False

    def __init__(
        self,
        df: pd.DataFrame,
        field_columns: str,
        categorial_threshold: int = 10,
        split_charts: bool = False,
    ):
```

```

):
    self.df = df
    self.categorical_threshold = categorical_threshold
    self.split_charts = split_charts
    self.field_columns = [x.strip() for x in field_columns.split(",")]
    if self.df.empty:
        raise Exception("Dataframe is empty")
    for field in self.field_columns:
        if field not in self.df:
            raise ValueError(f"Field {field} does not exist in the
dataframe")

    def __call__(self) -> None:
        self.df = self.df[self.field_columns]
        charts_dict = self.create_fields_charts()

        if self.split_charts:
            self.figs = {
                column_name: go.Figure(
                    data=[chart],
                    layout=self.custom_layout(column_name=column_name)
                )
                for column_name, chart in charts_dict.items()
            }
        else:
            self.fig = go.Figure(data=list(charts_dict.values()),
layout=self.custom_layout())

    def create_fields_charts(self):
        signal = {}
        self.min_x = 0
        self.max_x = 0
        counted_labels = []

        # цикл по всем столбцам df
        for columnName, columnData in self.df.items():
            # если данные в столбце не числовые, пропускаем его
            if not pd.api.types.is_numeric_dtype(columnData):
                print(f'Column "{columnName}" type is not numeric')
                continue

            counted_labels.append(columnName)
            visible_mode = (
                "legendonly" if columnName != counted_labels[0] and not
self.split_charts else True
            )

            # если данные в столбце категориальные, строим гистограмму
            if columnData.nunique() <= self.categorical_threshold:
                freq_df = (
                    columnData.value_counts(normalize=True, sort=False,
dropna=True)
                    .reset_index()
                    .sort_values(columnName)
                )

                freq_df["percent"] = freq_df["proportion"] * 100

```

```

        if self.split_charts:
            freq_df[columnName] = freq_df[columnName].astype("string")

            elem = go.Bar(
                x=freq_df[columnName].tolist(),
                y=freq_df["percent"].tolist(),
                name=columnName,
                opacity=0.7,
                marker=dict(line=dict(color="black", width=1.0)),
                visible=visible_mode,
            )
            signal[columnName] = elem
            continue

        # иначе - линейный график плотности распределения
        vals = columnData.dropna().values
        nbucket = int(len(vals) / 10) + 1
        den_x = []
        den_y = []
        wgt = (max(vals) - min(vals)) / nbucket # ширина одного
интервала
        minval = min(vals)
        self.max_x = max(max(vals), self.max_x)
        self.min_x = min(min(vals), self.min_x)
        self.max_pos = 0

        for i in range(0, nbucket):
            count = 0
            for j in vals:
                if (minval + i * wgt) <= j < (minval + (i * wgt) +
wgt):
                    count = count + 1
            den_x.append(round((minval + i * wgt + wgt / 2), 6))
            den_y.append(round(count * 100 / (len(vals)), 6))

        elem = go.Scatter(
            x=den_x,
            y=den_y,
            name=columnName,
            mode="lines",
            line_width=4,
            line_dash="solid",
            visible=visible_mode,
        )

        self.max_pos = max(max(den_y), self.max_pos)

        signal[columnName] = elem

    return signal

    def custom_layout(self, column_name: str | None = None) ->
Optional[Dict[str, Any]]:
        column_info = (
            " избранных столбцов" if column_name is None else f" для столбца
<b>{column_name}</b>"
        )

```

```

    return {
        "title": {"text": f"<b>Плотность распределения</b>{column_info}",
        "x": 0.1, "y": 0.98},
        "legend": {"yanchor": "bottom", "y": 0.05, "xanchor": "right",
        "x": 1},
        "xaxis": {
            "title": "Значение величины",
            "side": "left",
            "showgrid": True,
            "zeroline": True,
            "gridcolor": "#bdbdbd",
            "gridwidth": 1.5,
            "zerolinecolor": "#969696",
            "zerolinewidth": 3,
        },
        "yaxis": {
            "title": "Вероятность, %",
            "side": "left",
            "showgrid": True,
            "zeroline": True,
            "gridcolor": "#bdbdbd",
            "gridwidth": 1.5,
            "zerolinecolor": "#969696",
            "zerolinewidth": 3,
        },
        "margin": {"t": 45, "b": 5, "l": 5, "r": 5},
    }

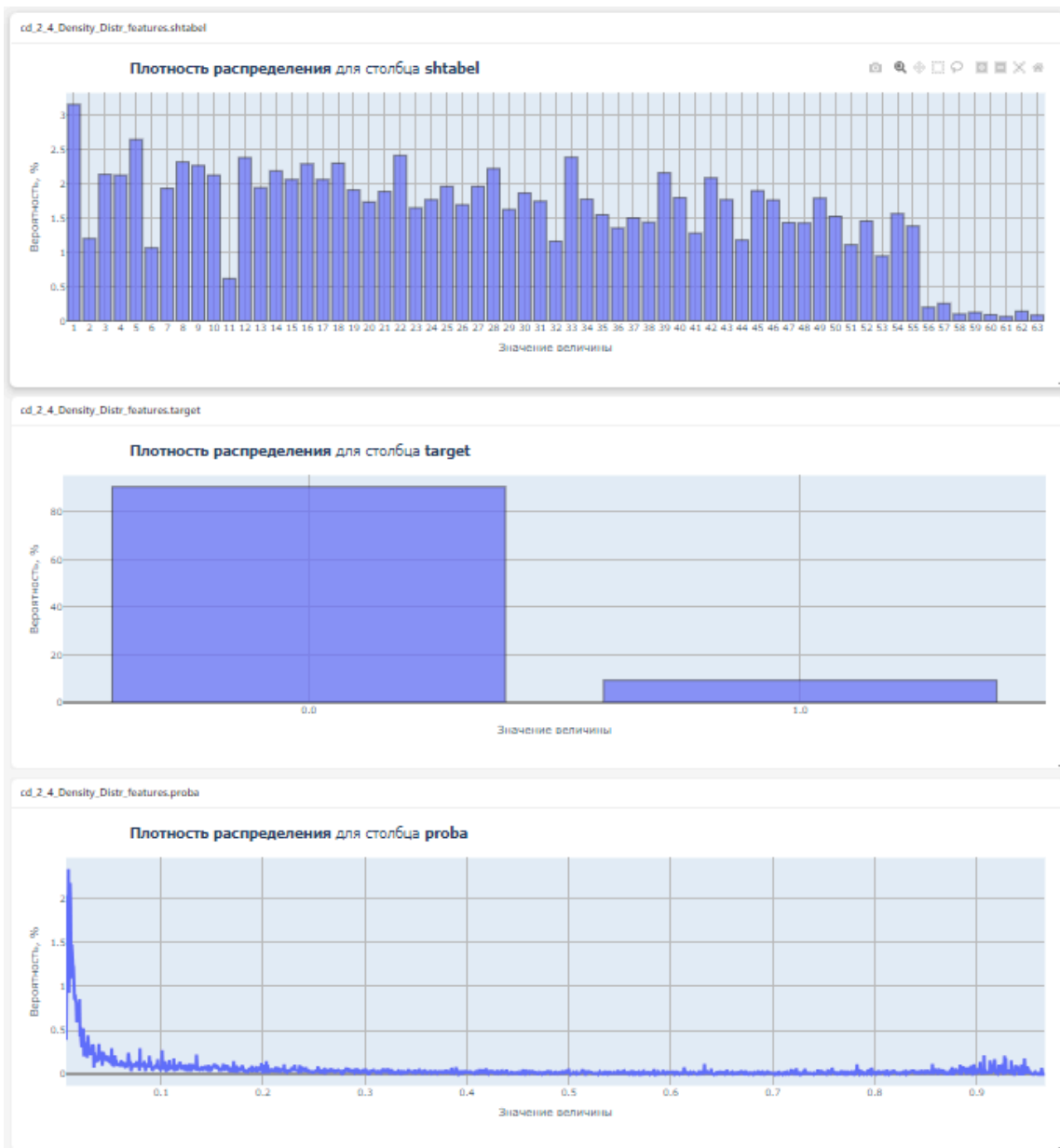
def save(self, output_dir: str) -> dict[str, str] | None:
    if self.split_charts:
        result = {}
        for column_name, fig in self.figs.items():
            file_path = f"{output_dir}/data_{column_name}.html"
            fig.write_html(
                file_path,
                config={"displaylogo": False}, # remove the plotly logo
            )

            result[column_name] = file_path
        return result
    else:
        self.fig.write_html(
            f"{output_dir}/data.html",
            config={"displaylogo": False}, # remove the plotly logo
        )

        return {"fig_name": f"{output_dir}/data.html"}

```

Результат:



Пример кода метрики, сохраняющей результат в виде картинки

Методы `__call__` и `save` объявлены и имплементированы.

В методе `save` графики сохраняются как картинки, а не HTML-файлы

```
from typing import Any, Dict, Literal, Optional

import numpy as np
import pandas as pd
from sklearn.metrics import roc_auc_score, roc_curve
import matplotlib.pyplot as plt

class ROC_AUC_img:
```

```

"""
Значение ROC-AUC и График ROC Curve

Attributes:
    __desc__ (str): Description of the class.
    __tags__ (list[str]): List of tags associated with the class.
    is_scalar (bool): Whether the metric is scalar or not.
    is_signal (bool): Whether the metric has signal or not.

"""

__desc__ = "График ROC Curve, значение ROC-AUC"
__tags__ = ["core", "classification", "scalar"]

is_scalar = True
is_signal = True

def __init__(
    self,
    df: pd.DataFrame,
    predict_column: str,
    target_column: str,
    threshold_yellow: float = 0.75,
    threshold_red: float = 0.65,
):
    self.predict_column = predict_column
    self.target_column = target_column
    self.df = df.astype({self.predict_column: "float", self.target_column:
"float"})
    self.threshold_yellow = threshold_yellow
    self.threshold_red = threshold_red

    if self.df.empty:
        raise Exception("Dataframe is empty")
    if self.target_column not in self.df:
        raise ValueError(f"Field {self.target_column} does not exist in
the dataframe")
    if self.predict_column not in self.df:
        raise ValueError(f"Field {self.predict_column} does not exist in
the dataframe")
    if self.predict_column == self.target_column:
        raise Exception("Ошибка. Проверьте выбор столбцов для расчета")

    def __call__(self) -> None:
        temp = self.df.loc[:, [self.target_column,
self.predict_column]].dropna()
        preds = temp[self.predict_column]
        y_test = temp[self.target_column]

        fpr, tpr, threshold = roc_curve(y_test, preds)
        fpr = np.around(fpr, decimals=4).tolist()
        tpr = np.around(tpr, decimals=4).tolist()
        base_roc = np.around(np.linspace(0, 1, 10), decimals=2).tolist()

        self.scalar_value = float(
            roc_auc_score(temp[self.target_column], temp[self.predict_column])
        )

```

```

self.fig, ax = plt.subplots()
ax.plot(fpr, tpr)
ax.set(xlabel='False Positive Rate', ylabel='True Positive Rate',
title='ROC Curve')
ax.grid()

def scalar(self) -> int | float:
    return self.scalar_value

def signal(self) -> Literal["red", "yellow", "green"]:
    signal_light = "green"

    if self.scalar_value < self.threshold_red:
        signal_light = "red"
    elif self.scalar_value < self.threshold_yellow:
        signal_light = "yellow"

    return signal_light

def save(self, output_dir: str) -> dict[str, str] | None:
    self.fig.savefig(f"{output_dir}/data.svg")
    return {"svg": f"{output_dir}/data.svg"}

```

Результат:

