Підготовка до роботи: завантаження необхідних бібліотек

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from pandas_profiling import ProfileReport
from sklearn.preprocessing import OneHotEncoder,StandardScaler,PowerTransformer, Nor
from sklearn.pipeline import make_pipeline
from sklearn.compose import make_column_transformer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score,classification_report
from sklearn.model_selection import KFold, cross_val_predict, train_test_split,GridS
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score, f1_score, accuracy_score, precision_score
from sklearn.impute import KNNImputer as ki
from statsmodels.stats.outliers_influence import variance_inflation_factor
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

Передбачення хвороб серця на базі клінічних досліджень

1. Формулювання мети дослідження

Метою дослідження, що проводиться, є побудова математичної моделі передбачення виникнення у пацієнтів гострої серцевої недостатності. В якості базового набору даних використовується набір даних клінічних досліджень пацієнтів різного віку, який нараховує дані 918 пацієнтів.

В ході дослідження необхідно побудувати двоїчний класифікатор з використанням різних математичних моделей класифікаторів, оцінити результати роботи кожного з класифікаторів за відповідними метриками та порівняти отримані результати для разних математичних моделей класифікаторів та їх програмних реалізацій.

При використанні програмних імплементацій математичних моделей будемо проводити також налаштування гіперпараметрів моделей, що дозволить підвищити якість результатів, що отримуються.

В якості основи програмної реалізації використовуємо Python, як найбільш поширену мову проектів в галузі DataScience. Реалізації математичнихї моделей та розрахунок метрик

проводимо з використанням Scikit learn, Pandas та NumPy, а такод додаткових бібліотек для візуалізації попередньої розвідки даних та отриманих результатів дослідження.

2. Аналіз інформації предметної області (даних)

Початковий датасет містить наступні результати клінічних досліджень пацієнтів:

Age - вік пацієнта

Sex - стать пацієнта

ChestPainType - тип болю в грудях (TA: Typical Angina, ATA: Atypical Angina, NAP: Non-Anginal Pain, ASY: Asymptomatic)

RestingBP - артеріальний тиск у спокої

Cholesterol - рівень холестерину

FastingBS - рівень цукру в крові

RestingECG - результати електрокардіограми спокою

MaxHR - максимальне значення пульсу при навантажені

ExerciseAngina - стенокардія, викликана фізичними навантаженнями

Oldpeak - пригнічення (скорочення) S-T сегменту (на електрокардіограмі) при фізичному навантаженні у порівнянні із спокоєм.

_ST*Slope* - міра підвищення пульсу пацієнта при фізичному навантаженні (https://pubmed.ncbi.nlm.nih.gov/3739881/)

HeartDisease - ознака наявності хвороби серця

Загалом маємо 11 ознак та бульову цільову (таргетну) змінну.

Наводимо приклад з декількох рядків датасету, що вивчається.

In [2]:
```
df = pd.read_csv('heart.csv')
df.sample(7)
```

Out[2]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAng
864	60	M	ASY	145	282	0	LVH	142	
589	74	M	NAP	140	237	1	Normal	94	
19	36	M	ATA	120	267	0	Normal	160	
706	61	F	ASY	130	330	0	LVH	169	
609	46	M	ASY	134	310	0	Normal	126	
539	57	M	ASY	110	197	0	LVH	100	
42	35	M	ATA	150	264	0	Normal	168	

3. Препроцесінг даних

До попередньої обробки даних відносимо наступні дії:

- Поділ даних на нумеричні та категорійні змінні.
- Пошук та відновлення відсутніх даних.
- Пошук та видалення викидів

3.1. Огляд даних.

Для швидкого попереднього огляду даних використаємо Pandas Profiling, встановивши опцію формування розширеного звіту

In [3]:
```
profile = ProfileReport(df, title="Pandas Profiling Report", explorative=True)
profile.to_notebook_iframe()
```

Overview

Dataset statistics

Number of variables	12
Number of observations	918
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	317.2 KiB
Average record size in memory	353.8 B

Variable types

Numeric	5
Categorical	6
Boolean	1

Alerts

`Age` is highly correlated with `FastingBS`	**High correlation**
`RestingBP` is highly correlated with `FastingBS`	**High correlation**
`Cholesterol` is highly correlated with `FastingBS` and 1 other	**High correlation**

Висновки з Profile Report:

1. Кількість нумеричних змінних - 5, категоріальних - 6.
2. Пропущених значень немає.
3. Треба приділити особливу увагу кореляції змінних між собою з метою усунення мультиколінеарності.
4. Нумеричні змінні, включаючи таргетну, та без Oldpeak мають розподіл, наближений до нормального. Розподіл Oldpeak треба вивчити більш детально.

5. По змінній Oldpeak також більш детально треба вивчити викиди.

6. Зниження розмірності датасету проводити не потрібно.

3.2. Поділ даних на нумеричні та категорійні змінні.

In [4]:
```
for col in df.drop(['HeartDisease'], axis=1).columns:
    print(f'{col}: {df[col].nunique()}')
```

```
Age: 50
Sex: 2
ChestPainType: 4
RestingBP: 67
Cholesterol: 222
FastingBS: 2
RestingECG: 3
MaxHR: 119
ExerciseAngina: 2
Oldpeak: 53
ST_Slope: 3
```

Як було отримано й в Profile Report, кількість нумеричних змінних складає 5, категоріальних - 6 (без урахування таргетної змінної).
Слід зауважити, що змінна FastingBS є фактично категоріальною, маючи значення: 0 - Ні, 1 - Так. *За нумеричні змінні приймаємо ті, для яких кількість унікальних значень складатиме більше за 10*

In [5]:
```
numerical = [col for col in df.columns if df[col].nunique() > 10]
categorical = [col for col in df.columns if col not in (numerical + ['HeartDisease']
print(f'Нумеричні ознаки:  {list(df[numerical].columns)}\n')
print(f'Категоріальні ознаки: {list(df[categorical].columns)}')
```

```
Нумеричні ознаки:  ['Age', 'RestingBP', 'Cholesterol', 'MaxHR', 'Oldpeak']

Категоріальні ознаки: ['Sex', 'ChestPainType', 'FastingBS', 'RestingECG', 'ExerciseA
ngina', 'ST_Slope']
```

З використанням згаданого вище фільтру, отримали очикувані кількості нумеричних та категоріальних змінних.

3.3. Пошук мультиколінеарності

VIF *(variance inflation factor)* є оберненою величиною допуску. Чим більше значення, тим очевидніша проблема колінеарності.
Зазвичай як межа оцінки використовується 10.

- коли VIF <10, мультиколлінеарність відсутня;
- коли 10 <= VIF <100, спостерігається мультиколлінеарність;
- коли VIF> = 100, спостерігається сильна мультиколлінеарність.

In [6]:
```
for col_name, vif_value in zip(df[numerical].columns, [variance_inflation_factor(df[
                                                                    df[numerical
    print(f'{col_name} VIF = {vif_value:5.3f}')
```

```
Age VIF = 28.515
RestingBP VIF = 42.020
Cholesterol VIF = 4.678
```

```
MaxHR VIF = 19.677
Oldpeak VIF = 1.849
```

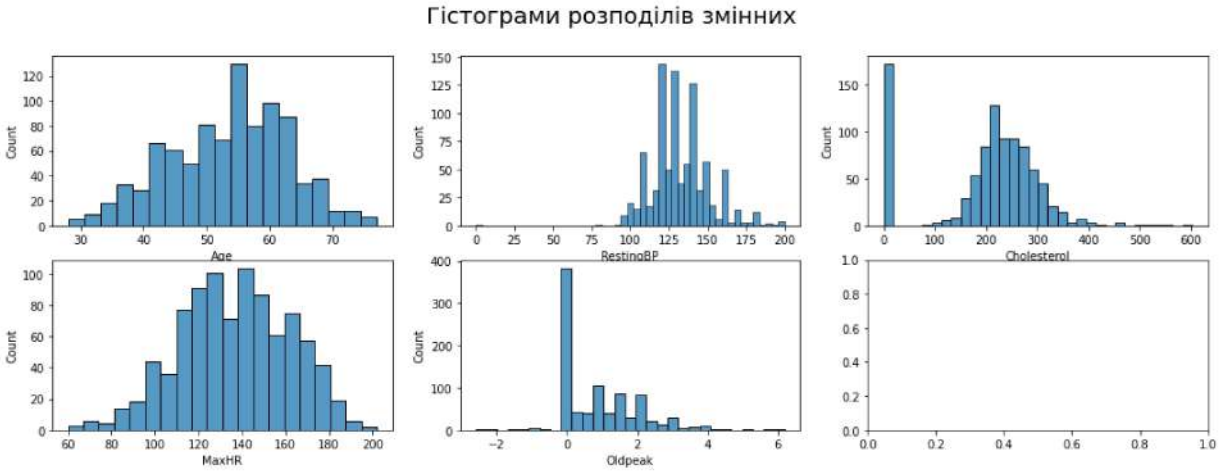
Деякі з нумеричних ознак можуть бути колінеарними з іншими ознаками (змінними) датасету. Необхідно розрахувати коефіцієнти кореляції з іншими змінними, з таргетною змінною та зробити висновки відносно необхідності виключення деяких з ознак з датасету.

Також отримані результати можно пояснити з медичної точки зору: типові (середні) параметри змінюються в залежності від віку пацієнта. RestingBP, що можна назвати одним з важливих параметрів здоров'я/міцності серця, впливає на інші характеристики сердця конткретного пацієнта, що отримані у спокої або під навантаженням.

3.4. Вивичення розподілів даних

Продовжуємо роботи з нумеричними ознаками датасету. Вивчаємо характер розподілу (близкість до нормального (гаусового)) та наявність викидів.
Для вивчення розподілів нумеричних змінних будуємо гістограми.

In [7]:
```
def numericals_histplots():
    fig, axs = plt.subplots(2, 3, figsize=(18, 6))
    for col, ax in zip(numerical, axs.flatten()):
        sns.histplot(x=df[col], ax=ax)
    plt.suptitle('Гістограми розподілів змінних', fontsize=20)
    plt.show()

numericals_histplots()
```



З вивчення розподілу змінних можна зробити наступні висновки:

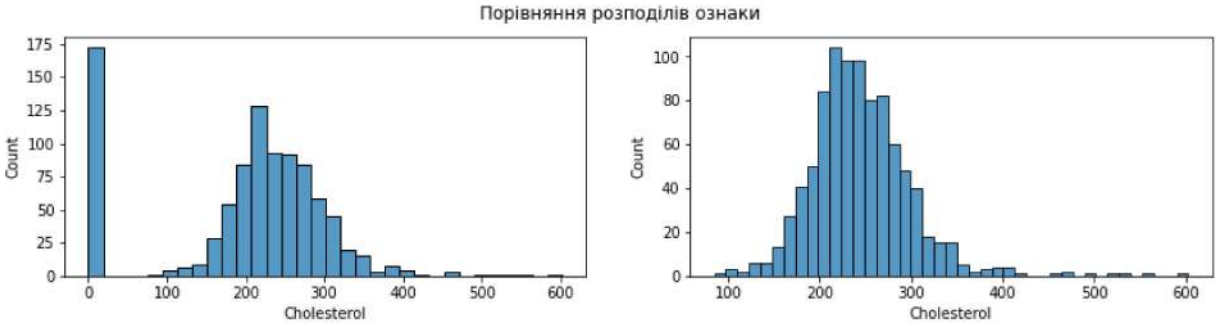
1. Розподіл змінних наближений до нормального розподілу
2. По змінній *Cholesterol* для ряда досліджень маємо нульові значення замість реальних. Треба виконувати підстановку значень з використанням *Imputers*.
3. Частина значень *Oldpeak* скоріш за все є округленою. Частина значень зустрічається частіше інших, значення дискретні. Разом з тим подальше налаштування моделі показує, що використання заміщення нульових значень погіршує якість моделі. Отже велика кількість нульових значень є природньою для цієї ознаки.

In [8]:
```
# заміняємо нульові значення на NaN
old_cholesterol = df['Cholesterol']
df['Cholesterol'] = df['Cholesterol'].apply(lambda x: x if x > 0 else np.nan)
values = df[numerical].values
```

```
# використовуємо KNN imputer
values = ki(n_neighbors=4).fit_transform(values)
for n, col in enumerate(df[numerical].columns):
    df[col] = values[:,n]
```

Побудуємо розподіли ознаки *Cholesterol* до та після підстановки

In [9]:
```
fig, ax = plt.subplots(1, 2, figsize=(14, 3))
sns.histplot(x=old_cholesterol, ax=ax[0])
sns.histplot(x=df['Cholesterol'], ax=ax[1])
plt.suptitle('Порівняння розподілів ознаки')
plt.show()
```



3.5. Викиди

Базаючись на невеликій кількості нумеричних ознак та розподілах, що близькі до гаусових, видалимо викиди простою фільтрацією нетипових значень.

In [10]:
```
df = df.loc[(df['RestingBP'] > 80) & (df['Cholesterol'].between(100, 400)) & (df['Ol
print(f'Розмірність датасету після видалення викидів {df.shape}')
```

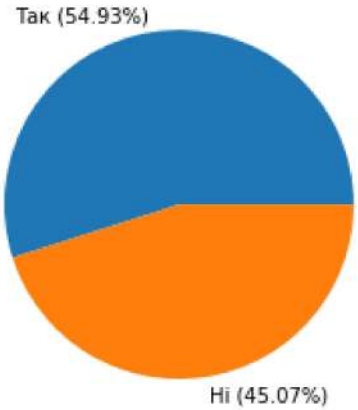
```
Розмірність датасету після видалення викидів (883, 12)
```

Кількість елементів вибірки зменшилася лише на *4%* при тому, що значення до *10%* вважається прийнятним

4. Попередня розвідка даних (EDA) (статистичний аналіз даних)

Цільова (таргетна) змінна

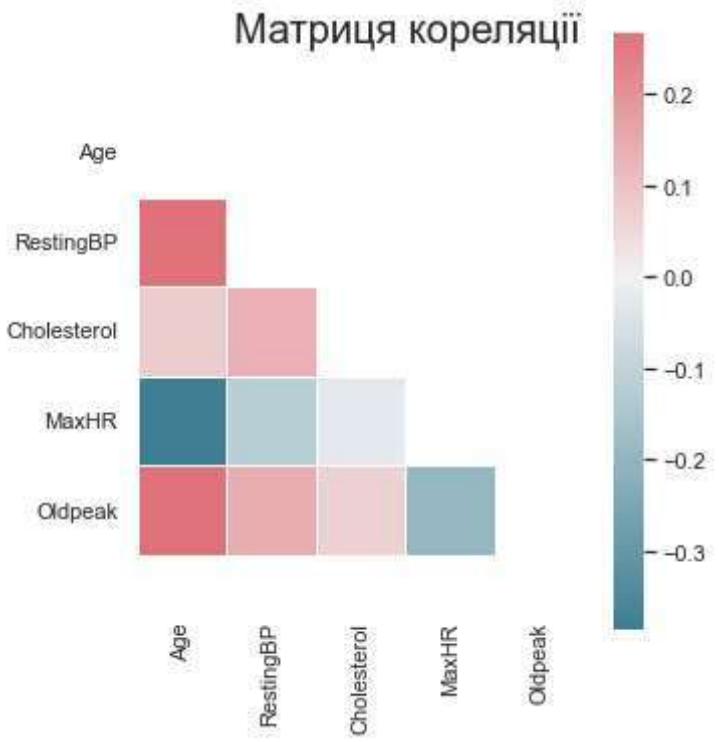
In [11]:
```
plt.pie(x=(srs:=df['HeartDisease'].value_counts()).values, labels=[f'{value} ({perce
plt.show()
```



Матриця кореляції нумеричних змінних

In [12]:

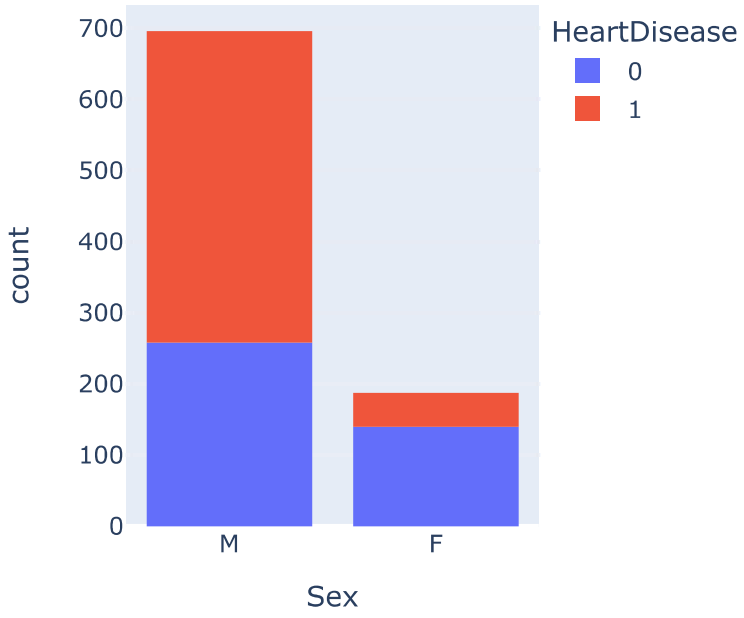
```
def plot_corr_matrix(metric_df, abs_apply=False, values=False, fig_size = (7, 7)):
    sns.set(style="white")

    # расчет корреляционной матрицы с использованием pandas
    corr = metric_df.corr()
    if abs_apply == True:
        corr = corr.apply(np.abs)

    # генерация маски для верхнего треугольника
    mask = np.zeros_like(corr, dtype=bool)
    mask[np.triu_indices_from(mask)] = True

    # создание экземпляра класса matplotlib figure
    fig, ax = plt.subplots(figsize=fig_size)

    # Генерация карты цветов
    cmap = sns.diverging_palette(220, 10, as_cmap=True) #Make a diverging palette be

    # Отображение HeatMap и коррекция масштаба
    if values:
        mask_annot = np.absolute(corr.values)>=0.50
        annot_arr = np.where(mask_annot, corr.values.round(2), np.full(fig_size,""))
        sns.heatmap(corr, mask=mask, cmap=cmap, center=0, square=True, linewidths=.5
    else:
        sns.heatmap(corr, mask=mask, cmap=cmap, center=0, square=True, linewidths=.5

    # отображение фигуры matplotlib
    plt.subplots_adjust(top=0.95)
    plt.suptitle("Матриця кореляції", fontsize=20)
    plt.yticks(rotation=0)
    b, t = plt.ylim()
    b += 0.5
    t -= 0.5
    plt.ylim(b, t)

    plt.show()

plot_corr_matrix(df[numerical], values=True, fig_size=(5, 5))
```



Усі нумеричні змінні мають низькі коефіцієнти попарної кореляції, виключення змінних не потрібне. Єдина залежність, яку можна відстежити: залежність MaxHR від віку (Age).

Вивчення категоріальних ознак

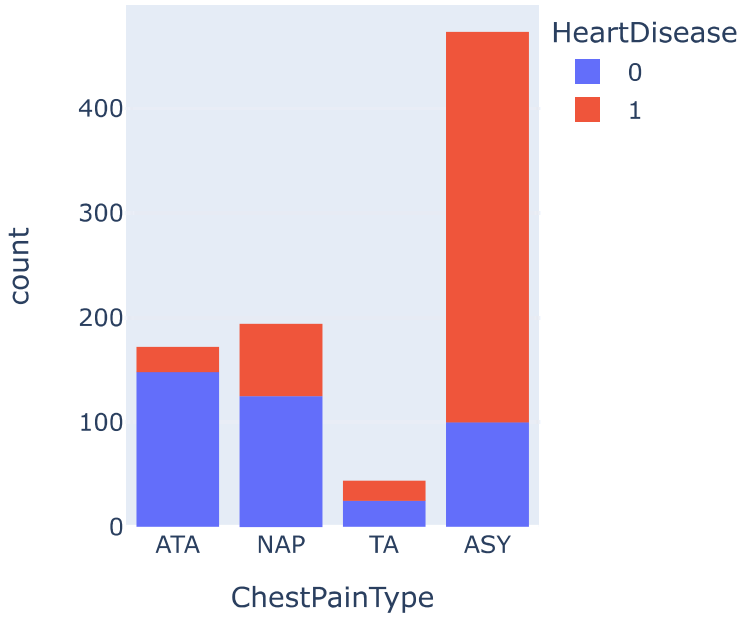
In [13]:
```
fig = px.histogram(df, x="Sex", color="HeartDisease",width=400, height=400)
fig.show()
```



Висновок: чоловіки майже в 2,5 рази частіше мають захворювання серця, ніж жінки.

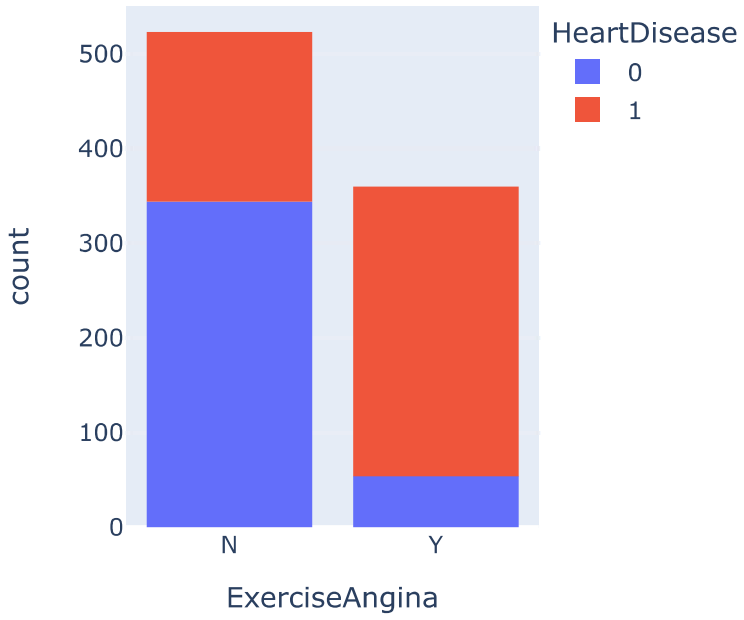
In [14]:
```
fig = px.histogram(df, x="ChestPainType", color="HeartDisease",width=400, height=400
```

```
fig.show()
```



Висновок: безсимптомний біль у грудях має майже в 6 разів більше шансів перетворитись на захворювання серця, ніж атитпова стенокардія.

In [15]:
```
fig = px.histogram(df, x="ExerciseAngina", color="HeartDisease",width=400, height=40
fig.show()
```



Висновок: людина, в якої фізичні навантаження викликають стенокардію, має в 2.5 більш високий ризик серцевих захворювань.

5. Підготовка датасету до застосування математичних моделей

Кодування категоріальних ознак з використанням OneHotEncoder

In [16]:
```
oh_encoder = OneHotEncoder()
ct_1 = make_column_transformer((oh_encoder, categorical), remainder='passthrough')
```

Створенне перетворення буде застосовано при побудові *pipelines* для застосуванні моделей. При використанні моделі SVM обов'язковою є нормалізація нумеричних ознак, тому додамо її в загальний pipeline.

In [17]:
```
ss = StandardScaler()
norm = Normalizer()
pt = PowerTransformer()
ct_2 = make_column_transformer((oh_encoder, categorical), (ss, numerical), remainder
ct_3 = make_column_transformer((oh_encoder, categorical), (norm, numerical), remaind
ct_4 = make_column_transformer((oh_encoder, categorical), (pt, numerical), remainder
```

При застосуванні pipeline'ів також порівняємо результати "з" та "без" використання нормалізації нумеричних ознак.

Поділ датасету на навчальний та валідаційний

In [18]:
```
X= df.drop('HeartDisease', axis=1)
y= df['HeartDisease']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_sta
```

6. Застосування моделей для створення класифікації

Порівняння класифікаторів та пайплайнів підготовки даних з використанням метрики *Accuracy*

In [19]:
```
# створення екземлярів відповідних класифікаторів
log_reg = LogisticRegression(solver='liblinear')
svm = SVC(gamma='scale')
dt = DecisionTreeClassifier()
models_to_apply = [log_reg, svm, dt]
```

In [20]:
```
# створення пайплайнів
accuracies, predictions = {}, []
for model_name, model in zip(['Логістична регресія', 'Метод опорних векторів', 'Дере
    pipes = [make_pipeline(ct, model) for ct in [ct_1, ct_2, ct_3, ct_4]]
    for pipe in pipes:
        pipe.fit(X_train, y_train)
    predictions.append(current_prediction := [pipe.predict(X_test) for pipe in pipes
    accuracies.update({model_name: [round(accuracy_score(y_test, y_pred),3) for y_pr
```

In [21]:
```
# табличне подання результатів
result = pd.DataFrame(accuracies, index=['Без нормалізації', 'Standard scaler', 'Nor
result
```

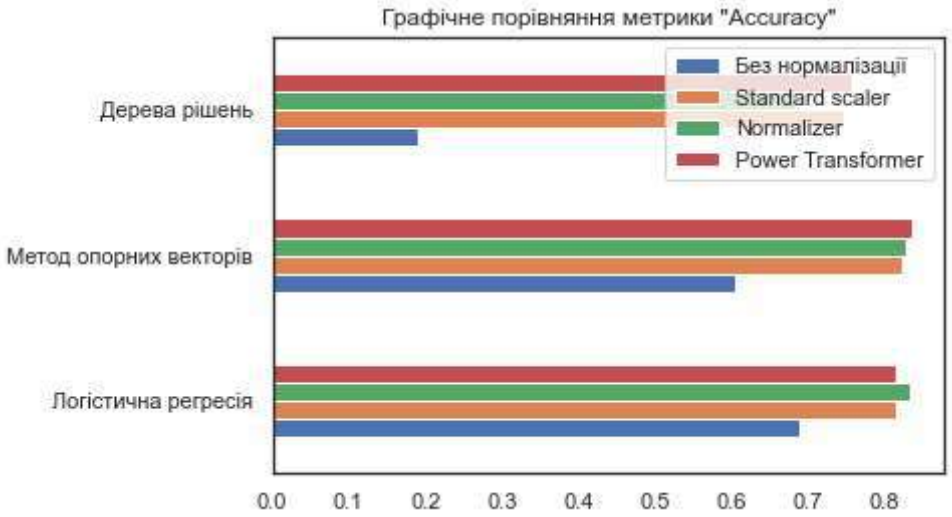
Out[21]:

	Логістична регресія	Метод опорних векторів	Дерева рішень

	Логістична регресія	Метод опорних векторів	Дерева рішень
Без нормалізації	0.688	0.606	0.190
Standard scaler	0.814	0.824	0.747
Normalizer	0.833	0.828	0.683
Power Transformer	0.814	0.837	0.756

Графічне порівняння отриманих результатів

In [22]:
```
result.T.plot.barh()
plt.title('Графічне порівняння метрики "Accuracy"')
plt.show()
```



Порівнянням можна зробити висновок, що нормалізовані дані впливають на підвищення точності класифікації при застосуванні логістичної регресії та методу опорних векторів. При використанні дерев рішень вплив нормалізації суттєво нижчий.

На цьому етапі найвищий результат досягнуто з використанням *методу опорних векторів* та *дерев рішень* за умови гаусової нормалізації нумеричних ознак.

7. Налаштування кращого з класифікаторів

In [23]:
```
svc = SVC()
pipeline = make_pipeline(ct_2, svc)
pipeline
```

Out[23]:
```
Pipeline(steps=[('columntransformer',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('onehotencoder',
                                                  OneHotEncoder(),
                                                  ['Sex', 'ChestPainType',
                                                   'FastingBS', 'RestingECG',
                                                   'ExerciseAngina',
                                                   'ST_Slope']),
                                                 ('standardscaler',
                                                  StandardScaler(),
                                                  ['Age', 'RestingBP',
                                                   'Cholesterol', 'MaxHR',
                                                   'Oldpeak'])])),
                ('svc', SVC())])
```

In [24]:
```
# використання GridSearchCV для налаштування гіперпараметрів класифікатору
parameters = {'svc__kernel': ['linear', 'poly', 'rbf', 'sigmoid', 'precomputed'],
              'svc__degree': [2, 3, 4, 5],
              'svc__gamma': ['scale', 'auto'],
              'svc__decision_function_shape': ['ovo', 'ovr']}
clf = GridSearchCV(pipeline, parameters, n_jobs=-1, scoring='accuracy')
best_estimator = clf.fit(X_train, y_train)
```

In [25]:
```
print("Найкращий результат, що досягнуто (CV score=%0.3f):" % clf.best_score_)
```

```
Найкращий результат, що досягнуто (CV score=0.872):
```

Параметри, з використанням яких досягнуто найкращого результату

In [26]:
```
clf.best_params_
```

Out[26]:
```
{'svc__decision_function_shape': 'ovo',
 'svc__degree': 2,
 'svc__gamma': 'auto',
 'svc__kernel': 'sigmoid'}
```

Оцінка кращого з класифікаторів за розглянутими в теоретичній частині метриками

In [27]:
```
print(f'Accuracy={accuracy_score(y_test, y_pred:=clf.predict(X_test)):5.3}')
print(f'Оцінка AUC={roc_auc_score(y_test, y_pred):5.3}')
print(f'Точність={precision_score(y_test, y_pred):5.3}')
print(f'Повнота={recall_score(y_test, y_pred):5.3}')
print(f'F1 score={f1_score(y_test, y_pred):5.3}')
```

```
Accuracy=0.824
Оцінка AUC=0.822
Точність=0.874
Повнота=0.828
F1 score=0.851
```

В цілому отриманий класифікатор може бути оцінений *"добре, ближче до відмінного*