ЛАБОРАТОРНАЯ РАБОТА №6

Тема лабораторной работы: автоматизация тестирования

**1. Разработка автотестов**

**a. Цель работы**

Цель работы — разработка автотестов для проверки функциональности приложения прогнозирования цен на акции, учитывая свойства сопровождаемости, надежности и структурированности. Автотесты должны эмулировать поведение пользователя при взаимодействии с элементами пользовательского интерфейса и обеспечивать контроль результатов выполненного сценария.

**b. Описание реализованных автотестов: инструменты, подходы**

**Инструменты:**

1. unittest — стандартная библиотека для модульного тестирования в Python.
2. PyTest — популярная библиотека для тестирования в Python.
3. pytest-qt — плагин для PyTest, который позволяет тестировать PyQt/PySide приложения.

**Подходы:**

* Использование модульных тестов для проверки отдельных функций.
* Эмуляция поведения пользователя для тестирования пользовательского интерфейса.
* Контроль результатов выполненного сценария с помощью проверок и утверждений.

**c. Код автотестов**Код для тестирования предсказаний цен на акции. Приложение 1.

Код для тестирования пользовательского интерфейса. Приложение 2.

**d. Отчет о тестировании**

**Тест-кейс 1: Проверка функции get\_stock\_data**

* **Результат:** Успешно
* **Описание:** Данные акции были успешно загружены и не пусты.

**Тест-кейс 2: Проверка функции create\_dataset**

* **Результат:** Успешно
* **Описание:** Набор данных был создан корректно, размеры массивов совпадают с ожиданиями.

**Тест-кейс 3: Проверка функции build\_lstm\_model**

* **Результат:** Успешно
* **Описание:** Модель LSTM была успешно создана, количество слоев и входная форма соответствуют ожиданиям.

**Тест-кейс 4: Тестирование ввода данных для прогноза через UI**

* **Результат:** Успешно
* **Описание:** Данные были введены, прогнозирование завершено, MSE рассчитан и отображен.

**Тест-кейс 5: Тестирование ввода невалидного тикера через UI**

* **Результат:** Успешно
* **Описание:** Сообщение об ошибке появилось при вводе невалидного тикера.

**e. Выводы по работе**

Проведенное тестирование показало, что основные функции и пользовательский интерфейс приложения прогнозирования цен на акции работают корректно при вводе валидных и невалидных данных. Автотесты успешно проверили функциональность приложения, включая загрузку данных, создание наборов данных, построение моделей и взаимодействие с пользовательским интерфейсом.

**f. Список использованных источников**

1. Документация по работе с библиотекой Pandas.

URL: <https://pandas.pydata.org/>

2. Документация по работе с библиотекой NumPy. URL: <https://numpy.org/>

3. Документация по работе с библиотекой threading. URL:<https://docs-python.ru/standart-library/modul-threading-python/>

4. Документация по работе с библиотекой matplotlib.pyplot.

URL: <https://matplotlib.org/3.5.3/api/_as_gen/matplotlib.pyplot.html>

5. Документация по работе с библиотекой pytest.

URL: <https://pytest-docs-ru.readthedocs.io/ru/latest/>

**Приложение 1**

import unittest

import yfinance as yf

import pandas as pd

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean\_squared\_error

import numpy as np

import datetime

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Dropout

# Функция для получения данных акции

def get\_stock\_data(ticker, start\_date, end\_date):

data = yf.download(ticker, start=start\_date, end=end\_date)

data.dropna(inplace=True)

return data

# Функция для создания набора данных

def create\_dataset(data, time\_step=1):

X, y = [], []

for i in range(len(data)-time\_step-1):

X.append(data[i:(i+time\_step)])

y.append(data[i + time\_step, 0])

return np.array(X), np.array(y)

# Функция для создания модели LSTM

def build\_lstm\_model(input\_shape):

model = Sequential()

model.add(LSTM(100, return\_sequences=True, input\_shape=input\_shape))

model.add(Dropout(0.2))

model.add(LSTM(100, return\_sequences=True))

model.add(Dropout(0.2))

model.add(LSTM(100))

model.add(Dropout(0.2))

model.add(Dense(50))

model.add(Dense(1))

model.compile(optimizer='adam', loss='mean\_squared\_error')

return model

class TestStockPrediction(unittest.TestCase):

def test\_get\_stock\_data(self):

data = get\_stock\_data('AAPL', '2023-01-01', '2023-12-31')

self.assertIsInstance(data, pd.DataFrame)

self.assertFalse(data.empty)

def test\_create\_dataset(self):

data = np.array([[i] for i in range(100)])

X, y = create\_dataset(data, time\_step=10)

self.assertEqual(len(X), 89)

self.assertEqual(len(y), 89)

self.assertEqual(X.shape[1], 10)

self.assertEqual(X.shape[2], 1)

def test\_build\_lstm\_model(self):

model = build\_lstm\_model((60, 5))

self.assertIsInstance(model, Sequential)

self.assertEqual(len(model.layers), 6)

self.assertEqual(model.layers[0].input\_shape, (None, 60, 5))

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

**Приложение 2**

import pytest

from PyQt5 import QtWidgets

from PyQt5.QtCore import Qt

from your\_application\_file import create\_main\_window # Импортируем вашу функцию создания главного окна

@pytest.fixture

def app(qtbot):

main\_window = create\_main\_window()

qtbot.addWidget(main\_window)

return main\_window

def test\_web\_data\_prediction(app, qtbot):

ticker\_input = app.findChild(QtWidgets.QLineEdit, 'ticker\_input')

start\_date\_input = app.findChild(QtWidgets.QLineEdit, 'start\_date\_input')

end\_date\_input = app.findChild(QtWidgets.QLineEdit, 'end\_date\_input')

predict\_button = app.findChild(QtWidgets.QPushButton, 'predict\_button')

qtbot.keyClicks(ticker\_input, 'AAPL')

qtbot.keyClicks(start\_date\_input, '2023-01-01')

qtbot.keyClicks(end\_date\_input, '2023-12-31')

qtbot.mouseClick(predict\_button, Qt.LeftButton)

qtbot.waitUntil(lambda: app.findChild(QtWidgets.QLabel, 'status\_label').text() == 'Прогнозирование завершено', timeout=10000)

mse\_label = app.findChild(QtWidgets.QLabel, 'mse\_label')

assert mse\_label.text() != ''

def test\_invalid\_ticker(app, qtbot):

ticker\_input = app.findChild(QtWidgets.QLineEdit, 'ticker\_input')

start\_date\_input = app.findChild(QtWidgets.QLineEdit, 'start\_date\_input')

end\_date\_input = app.findChild(QtWidgets.QLineEdit, 'end\_date\_input')

predict\_button = app.findChild(QtWidgets.QPushButton, 'predict\_button')

qtbot.keyClicks(ticker\_input, '')

qtbot.keyClicks(start\_date\_input, '2023-01-01')

qtbot.keyClicks(end\_date\_input, '2023-12-31')

qtbot.mouseClick(predict\_button, Qt.LeftButton)

qtbot.waitUntil(lambda: app.findChild(QtWidgets.QLabel, 'status\_label').text() == 'Ошибка: неверный тикер', timeout=10000)