



ЛІНК НА ГІТХАБ

ПАВЛО ЧІСТЯКОВ

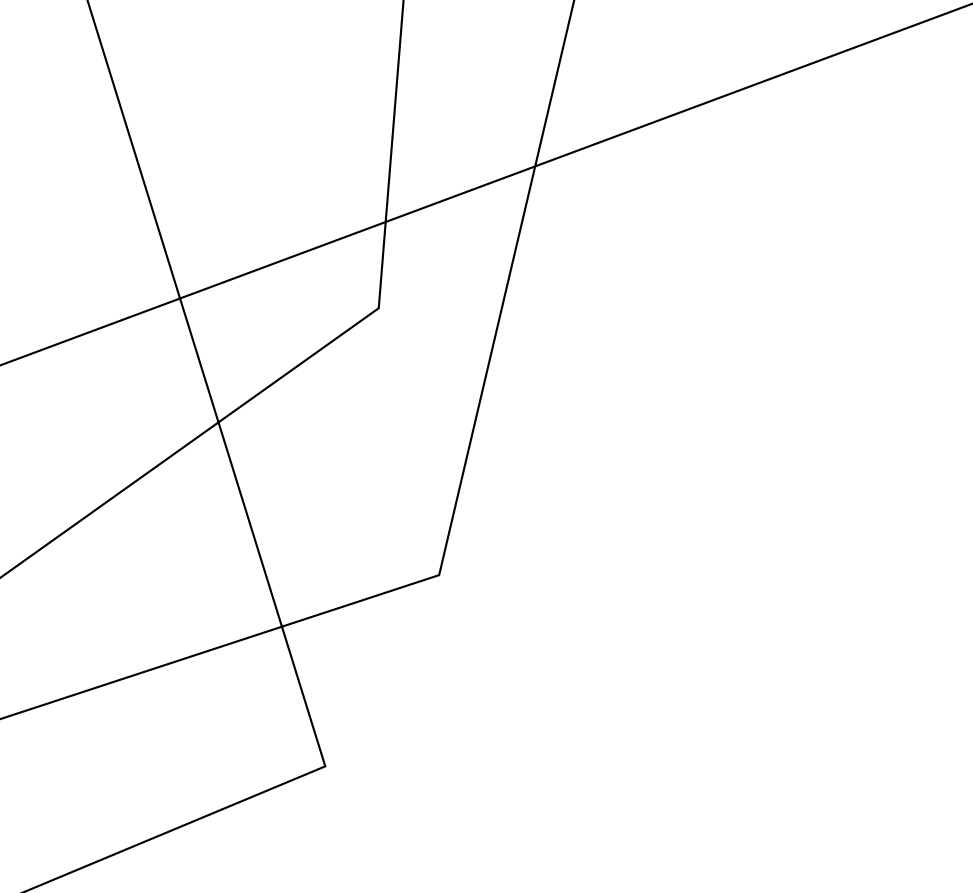
ЄВА ВЕЛИЧКО

ЄГОР ВІЛЯНСЬКИЙ

ПРОГНОЗУВАННЯ РИЗИКУ РОЗВИТКУ ДІАБЕТУ У ЛЮДИНИ «NEUROPREDICTD»

АКТУАЛЬНІСТЬ ПРОБЛЕМИ

Діабет є одним з найпоширеніших хронічних захворювань сучасності, що суттєво впливає на якість та тривалість життя людей. За останні роки кількість випадків діабету стрімко зростає, особливо серед молоді та людей середнього віку. Таким чином, розробка ефективних методів прогнозування ризику розвитку діабету стає надзвичайно важливою для здоров'я нації.



МЕТА ІНФОРМАЦІЙНОЇ СИСТЕМИ

Метою нашої інформаційної системи є розробка імовірнісно-статистичної моделі, яка на основі комплексного аналізу клінічних та демографічних даних буде прогнозувати наявність у людини діабету.

Крім того, ІС підходить для ранньої діагностики ризику діабету. За її допомогою можна буде ідентифікувати людей, які ще не показують симптомів, але вже мають підвищений ризик. Це дозволить приймати превентивні заходи та змінювати стиль життя для попередження захворювання.

МЕДИЧНІ КОНСУЛЬТАЦІЇ

Лікарі зможуть використовувати нашу модель для точнішого визначення ризику розвитку діабету в пацієнтів на ранніх стадіях.

ПЕРСОНАЛЬНІ ТРЕКЕРИ- ЗДОРОВ'Я

Люди можуть використовувати модель у застосунках для моніторингу здоров'я, щоб відстежувати свій ризик розвитку діабету і вживати проактивних заходів для його зниження.

ШВИДКА ПЕРЕВІРКА ЗДОРОВ'Я НА РОБОЧОМУ МІСЦІ

Роботодавці можуть використовувати нашу модель для швидкого оцінювання ризику розвитку діабету в співробітників і надання відповідних програм із підтримки здоров'я.

ОСВІТНІ ПРОГРАМИ

Використання моделі в освітніх програмах дасть змогу навчати людей про фактори ризику та способи запобігання діабету.

ПРИКЛАДИ ВИКОРИСТАННЯ У МАЙБУТНЬОМУ

ТЕХНІЧНІ ЗАДАЧІ

ПІДГОТОВКА ДАНИХ

АНАЛІЗ ДАНИХ

ОБРОБКА ДАНИХ

ВИБІР МОДЕЛІ

СТВОРЕННЯ НЕЙРОМЕРЕЖІ

ПЕРЕВІРКА РЕЗУЛЬТАТІВ

[ЛІНК НА ГІТХАБ](#)

ПІДГОТОВКА ДАНИХ

	gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_glucose_level	diabetes
0	Female	80.0	0	1	never	25.19	6.6	140	0
1	Female	54.0	0	0	No Info	27.32	6.6	80	0
2	Male	28.0	0	0	never	27.32	5.7	158	0
3	Female	36.0	0	0	current	23.45	5.0	155	0
4	Male	76.0	1	1	current	20.14	4.8	155	0
5	Female	20.0	0	0	never	27.32	6.6	85	0
6	Female	44.0	0	0	never	19.31	6.5	200	1
7	Female	79.0	0	0	No Info	23.86	5.7	85	0
8	Male	42.0	0	0	never	33.64	4.8	145	0
9	Female	32.0	0	0	never	27.32	5.0	100	0

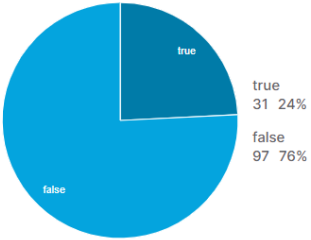
100000 rows × 9 columns

ДАНІ СТА ТИСЯЧ
ДІАБЕТИКІВ

МИ ПОРІВНЯЛИ ДАНІ, ТА ЗРОЗУМІЛИ

✓ Diagnosis

Distributed into: Yes(diagnosed) and No(Not Diagnosed).



Valid	128	100%
Mismatched	0	0%
Missing	0	0%
True	31	24%
False	97	76%

11 columns

String	4
Integer	3
Boolean	3
Other	1

128 ДАНИХ
11 КОЛОНОК

ІНШІ

diabetes

Diabetes is the target variable being predicted, with values of 1 indicating the presence of diabetes and 0 indicating the absence of diabetes.



Valid	100.0k	100%
Mismatched	0	0%
Missing	0	0%
Mean	0.09	
Std. Deviation	0.28	
Quantiles		
	0	Min
	0	25%
	0	50%
	0	75%
	1	Max

9 columns

Integer	4
Decimal	3
String	2

100К ДАНИХ
9 КОЛОНОК

НАШІ

100,000

Number of records

8,500

diabetes



91,500

no diabetes

Hypertension

No

Yes



Heart Disease

No

Yes

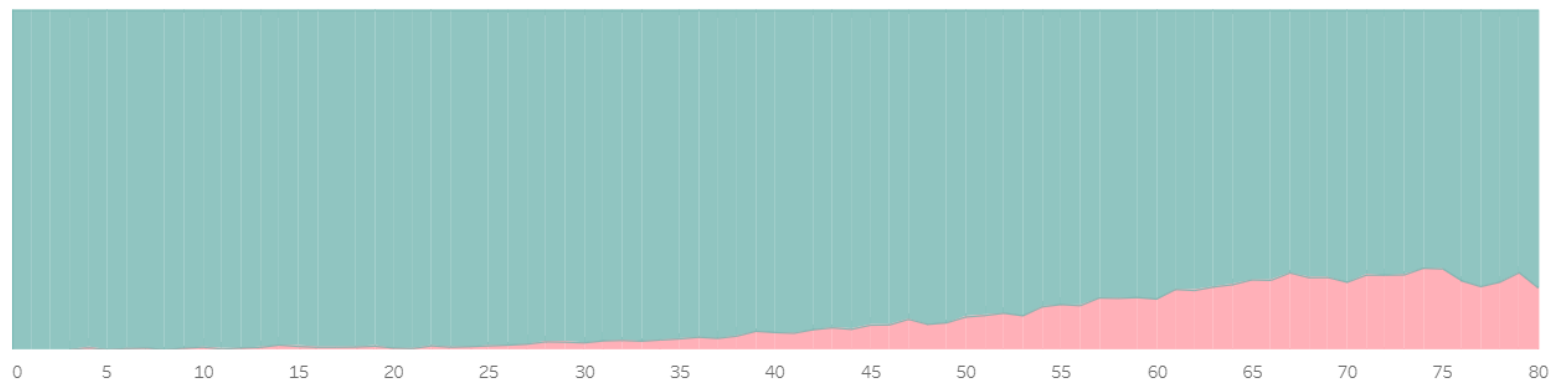


Blood Glucose Level

Normal (<7)

Risk (7 - 8)

Diabetes (>8)



Gender

Male

Female



HbA1c

Normal (<6)

Prediabetes (6 - 6.5)

Diabetes (>6.5)



BMI

Underweight

Normal

Overweight

Obese (Class I)

Obese (Class II)

Obese (Class III)



Smoking History

No Info

never

ever

not current

former

current



[LINK](#)

[LINK](#)

АНАЛІЗ ДАНИХ

Використовуємо потрібні бібліотеки

Виводимо дані

Ілюструємо

```
import matplotlib.pyplot as plt  
import numpy as np  
import pandas as pd
```

```
... # Получаем общую информацию о наборе данных
data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 9 columns):
#   Column                Non-Null Count  Dtype
---  -
0   gender                100000 non-null object
1   age                   100000 non-null float64
2   hypertension          100000 non-null int64
3   heart_disease         100000 non-null int64
4   smoking_history       100000 non-null object
5   bmi                   100000 non-null float64
6   HbA1c_level           100000 non-null float64
7   blood_glucose_level   100000 non-null int64
8   diabetes              100000 non-null int64
```

Получаем статистику по набору данных. Для понимания с какими данными мы будем работать

data.describe()

	age	hypertension	heart_disease	bmi	HbA1c_level	blood_glucose_level	diabetes
count	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000
mean	41.885856	0.07485	0.039420	27.320767	5.527507	138.058060	0.085000
std	22.516840	0.26315	0.194593	6.636783	1.070672	40.708136	0.278883
min	0.080000	0.00000	0.000000	10.010000	3.500000	80.000000	0.000000
25%	24.000000	0.00000	0.000000	23.630000	4.800000	100.000000	0.000000
50%	43.000000	0.00000	0.000000	27.320000	5.800000	140.000000	0.000000
75%	60.000000	0.00000	0.000000	29.580000	6.200000	159.000000	0.000000
max	80.000000	1.00000	1.000000	95.690000	9.000000	300.000000	1.000000

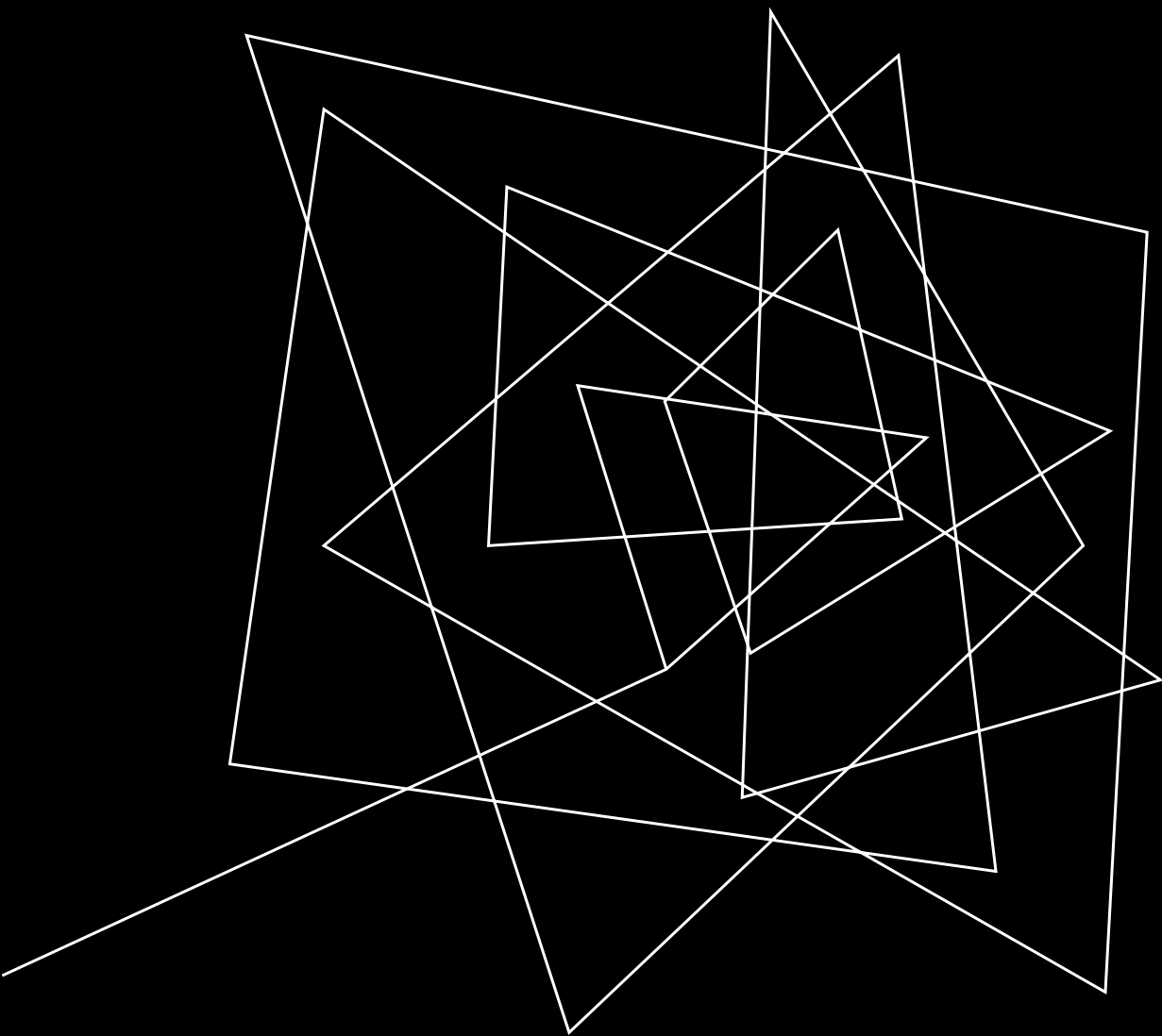
Выводим первые 10 значений набора данных

data.head(10)

	gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_glucose_level	diabetes
0	Female	80.0	0	1	never	25.19	6.6	140	0
1	Female	54.0	0	0	No Info	27.32	6.6	80	0
2	Male	28.0	0	0	never	27.32	5.7	158	0
3	Female	36.0	0	0	current	23.45	5.0	155	0
4	Male	76.0	1	1	current	20.14	4.8	155	0
5	Female	20.0	0	0	never	27.32	6.6	85	0
6	Female	44.0	0	0	never	19.31	6.5	200	1
7	Female	79.0	0	0	No Info	23.86	5.7	85	0
8	Male	42.0	0	0	never	33.64	4.8	145	0
9	Female	32.0	0	0	never	27.32	5.0	100	0

```
# Вывод уникальных значений для каждого столбца
for column in data.columns:
    value = data[column].unique()
    print(f"Уникальные значения столбца {column}: {value}")

Уникальные значения столбца gender: ['Female' 'Male' 'Other']
Уникальные значения столбца age: [80. 54. 28. 36. 76. 20. 44. 79. 42. 32. 53. 78.
67. 15. 37. 40. 5. 69. 72. 4. 30. 45. 43. 50.
41. 26. 34. 73. 77. 66. 29. 60. 38. 3. 57. 74.
19. 46. 21. 59. 27. 13. 56. 2. 7. 11. 6. 55.
9. 62. 47. 12. 68. 75. 22. 58. 18. 24. 17. 25.
0.08 33. 16. 61. 31. 8. 49. 39. 65. 14. 70. 0.56
48. 51. 71. 0.88 64. 63. 52. 0.16 10. 35. 23. 0.64
1.16 1.64 0.72 1.88 1.32 0.8 1.24 1. 1.8 0.48 1.56 1.08
0.24 1.4 0.4 0.32 1.72 1.48]
Уникальные значения столбца hypertension: [0 1]
Уникальные значения столбца heart_disease: [1 0]
Уникальные значения столбца smoking_history: ['never' 'No Info' 'current' 'former' 'ever' 'not current']
Уникальные значения столбца bmi: [25.19 27.32 23.45 ... 59.42 44.39 60.52]
Уникальные значения столбца HbA1c_level: [6.6 5.7 5. 4.8 6.5 6.1 6. 5.8 3.5 6.2 4. 4.5 9. 7. 8.8 8.2 7.5 6.8]
Уникальные значения столбца blood_glucose_level: [140 80 158 155 85 200 145 100 130 160 126 159 90 260 220 300 280 240]
Уникальные значения столбца diabetes: [0 1]
```



ОБРОБКА
ДАНИХ

```
# Найдем все столбцы с null значениями
cols_with_missing_values = data.columns[data.isnull().any()]

# Визуально определяем столбцы с пропущенными значениями
print(data.isnull().any())

# Заменяем эти значения средним
data[cols_with_missing_values] = data[cols_with_missing_values].apply(lambda col: col.fillna(data[col.name].mean()))
```

```
gender          False
age             False
hypertension     False
heart_disease    False
smoking_history  False
bmi             False
HbA1c_level      False
blood_glucose_level  False
diabetes         False
dtype: bool
```

```
# Находим неверные значения и заменяем их средними
values = ['Male', 'Female']
data.loc[data["gender"] == "Other", "gender"] = random.choice(values)

# Преобразуем категориальные значения в числа с помощью map
# Для этих задач отлично подходит код ниже, но мы делаем это вручную.
# data["smoking_history"] = LabelEncoder().fit_transform(data["smoking_history"])
# print(data['smoking_history'])

# Дадим значения пола
mapping = {'Female': 0, 'Male': 1}
data['gender'] = data['gender'].map(mapping)

# Определим историю курения
mapping = {'never': 0, 'current': 1, 'No Info': 0, 'former': 0.5, "ever": 0.1, "not current": 0.2}
data['smoking_history'] = data['smoking_history'].map(mapping)

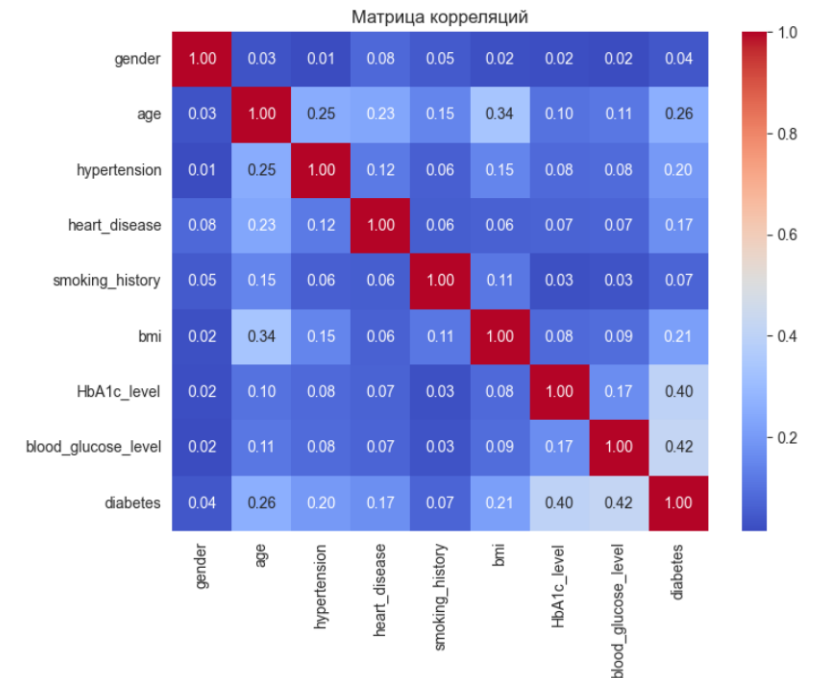
# Изменим ml\ds на mmol\l
data['blood_glucose_level'] = data['blood_glucose_level'] * 0.056
```

```
correlation_matrix = data.corr().abs()
```

```
# Визуализация матрицы корреляций с помощью heatmap из библиотеки Seaborn  
plt.figure(figsize=(8, 6))  
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")  
plt.title('Матрица корреляций')  
plt.show()
```

```
# Зависимости значений к наличию диабета.  
correlation_matrix["diabetes"].sort_values(ascending=False)
```

```
diabetes          1.000000  
blood_glucose_level 0.419558  
HbA1c_level       0.400660  
age               0.258008  
bmi               0.214357  
hypertension      0.197823  
heart_disease     0.171727  
smoking_history   0.069648  
gender            0.037553  
Name: diabetes, dtype: float64
```



МАТРИЦА КОРЕЛЯЦІЙ

```
# Определяем каждой колонке пронумерованную по группам, с суффиксом _labeled
columns = data.columns
for column in columns:
    data[column+"_labeled"] = pd.qcut(data[column], q=5, labels=False, duplicates='drop')

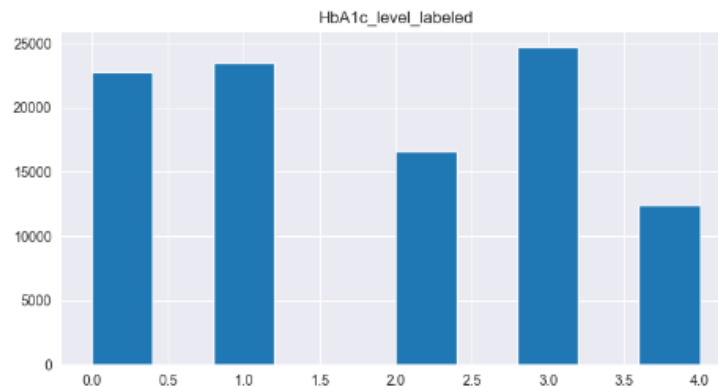
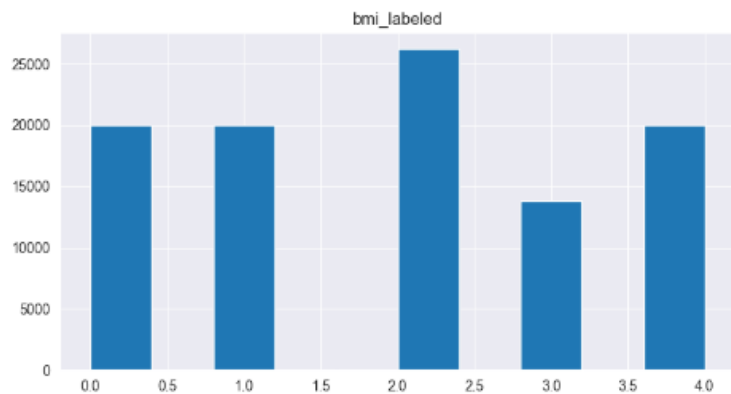
# Подсчет количества уникальных значений в каждом новом столбце Labeled
unique_counts = data.filter(like="_labeled").nunique()

# Получение названий столбцов с одним уникальным значением
single_label_columns = unique_counts[unique_counts <= 2].index

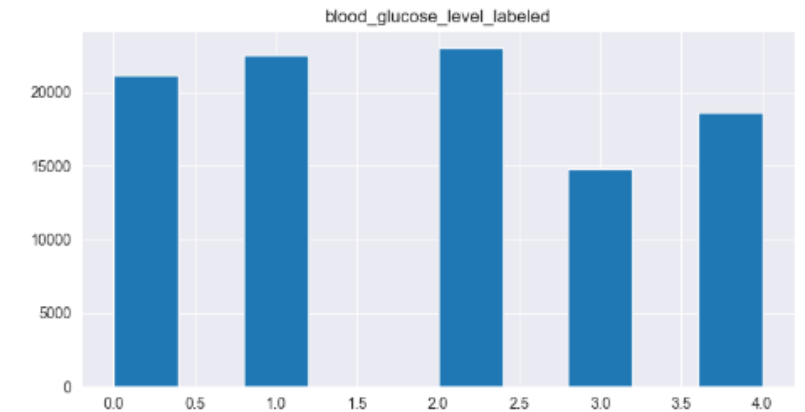
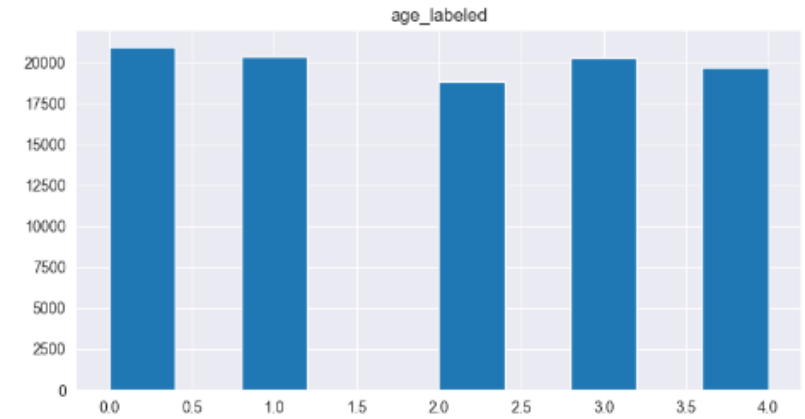
# Удаление столбцов с одним уникальным значением
data = data.drop(columns=single_label_columns)
```

```
# Находим столбцы с суффиксом _labeled и выводим их для сравнения корреляции данных
labeled_columns = [col for col in data.columns if col.endswith("_labeled")]

data[labeled_columns].hist(figsize=(20, 10))
plt.show()
```



СТРАТИФІКАЦІЯ



РЕЗУЛЬТАТИ СТРАТИФІКАЦІЇ

```
Original Data:  
age_labeled  
0    1.04520  
1    1.01705  
3    1.01260  
4    0.98330  
2    0.94185  
Name: count, dtype: float64
```

```
Train:  
age_labeled  
0    0.209037  
1    0.203412  
3    0.202513  
4    0.196662  
2    0.188375  
Name: count, dtype: float64
```

```
Test:  
age_labeled  
0    0.20905  
1    0.20340  
3    0.20255  
4    0.19665  
2    0.18835  
Name: count, dtype: float64
```

```
# Разбиваем данные на обучающие и тестовые в соотношении 80\20 и с использованием стратификации по столбцу current_column  
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=22)
```

```
current_column = "age_labeled"
```

```
for train_index, test_index in split.split(data, data[current_column]):  
    strat_train_set = data.loc[train_index]  
    strat_test_set = data.loc[test_index]
```

```
# Сравниваем результаты применения StratifiedShuffleSplit функции  
test_strat = strat_test_set[current_column].value_counts() / len(strat_test_set)  
data_set = data[current_column].value_counts() / len(strat_test_set)  
train_strat = strat_train_set[current_column].value_counts() / len(strat_train_set)  
  
print(f"\nOriginal Data: \n{data_set}\n\nTrain: \n{train_strat}\n\nTest: \n{test_strat}")
```

ГОТУЄМО ДАНІ ДЛЯ МОДЕЛІ

Та завершуємо
роботу з даними

```
# Удаляем столбец с labels, так как он может мешать обучению и в целом не нужен.  
for labeled in labeled_columns:  
    if labeled in strat_train_set:  
        strat_train_set = strat_train_set.drop(labeled,axis=1)  
        strat_test_set = strat_test_set.drop(labeled,axis=1)  
        data = data.drop(labeled, axis=1)  
    else:  
        print(f"Столбец {labeled} уже удален")
```

```
# Определяем рабочую переменную с обучающими данными и тестовыми  
train_data = strat_train_set.copy()  
test_data = strat_test_set.copy()
```

```
# split into input and output columns  
X_train = train_data.values[:, :-1]  
y_train = train_data.values[:, -1]  
  
# split into input and output columns  
X_test = test_data.values[:, :-1]  
y_test = test_data.values[:, -1]
```

```
n_features = X_train.shape[1]  
print(n_features)
```

8



ВИБІР МОДЕЛІ ТА СТВОРЕННЯ НЕЙРОМЕРЕЖІ

**LinearRegression \ DecisionTreeRegressor \
KNeighborsRegressor \ RandomForestRegressor
(GridSearchCV)**

СТВОРЕННЯ ФУНКЦІЇ ОЦІНКИ МОДЕЛІ

R2 приймає значення від 0 до 1 і показує частку поясненої дисперсії пояснюваного ряду.

Що ближче R2 до 1, то краща модель, то менша частка нез'ясованого.

```
models_r2_values = {}

def model_evaluation(model, model_name: str):
    y_pred = model.predict(X_train)
    mse_train = mean_squared_error(y_train, y_pred)
    rmse_train = np.sqrt(mse_train)

    # Оценка модели на тестовых данных
    y_pred_test = model.predict(X_test)
    mse_test = mean_squared_error(y_test, y_pred_test)
    rmse_test = np.sqrt(mse_test)

    print(f"\n\n{model_name}\n\n")

    print(np.concatenate((y_pred_test.reshape(len(y_pred_test), 1), y_test.reshape(len(y_test), 1)), 1)[:10])

    print("Root mean squared error on test set: ", mse_test)
    print("Root mean squared error on test set (RMSE): ", rmse_test)

    print("\nRoot mean error: ", mse_train)
    print("Root mean r error: ", rmse_train)

    difference_mse = (mse_test - mse_train) / mse_train * 100
    difference_rmse = (rmse_test - rmse_train) / rmse_train * 100

    print(f"\nDifferences in MSE (train to test): {difference_mse:.2f}%")
    print(f"Differences in RMSE (train to test): {difference_rmse:.2f}%")

    # Оценка модели на тестовых данных
    """
    Коэффициент R2 измеряет долю дисперсии зависимой переменной (целевой переменной), которая может быть объяснена моделью.
    """
    r2_test = r2_score(y_test, y_pred_test)
    print(f"R-squared test: {r2_test:.4f}")

    r2 = r2_score(y_train, y_pred)
    print(f"R-squared train: {r2:.4f}")

    models_r2_values[model_name] = r2_test
```

R2

MSE

RMSE

LinearRegression

```
lin_reg = LinearRegression()  
lin_reg.fit(X_train, y_train)  
  
model_evaluation(lin_reg, "Линейная Регрессия")
```

Линейная Регрессия

```
[[ 0.11  0. ]  
 [-0.11  0. ]  
 [ 0.17  0. ]  
 [ 0.48  1. ]  
 [ 0.33  1. ]  
 [ 0.1  0. ]  
 [ 0.43  1. ]  
 [ 0.14  0. ]  
 [ 0.08  0. ]  
 [-0.05  0. ]]
```

Root mean squared error on test set: 0.050955019673839454

Root mean squared error on test set (RMSE): 0.22573218572866266

Root mean error: 0.05060196979154161

Root mean r error: 0.22494881593718516

Differences in MSE (train to test): 0.70%

Differences in RMSE (train to test): 0.35%

R-squared test: 0.3582

R-squared train: 0.3460

DecisionTreeRegressor

```
tree_reg = DecisionTreeRegressor()  
tree_reg.fit(X_train, y_train)  
  
model_evaluation(tree_reg, "Дерево Решений")
```

Дерево Решений

```
[[0. 0.]  
 [0. 0.]  
 [0. 0.]  
 [1. 1.]  
 [1. 1.]  
 [0. 0.]  
 [1. 1.]  
 [0. 0.]  
 [0. 0.]  
 [0. 0.]]
```

Root mean squared error on test set: 0.046729958333333335

Root mean squared error on test set (RMSE): 0.21617113205359623

Root mean error: 0.0007097767857142857

Root mean r error: 0.02664163631825729

Differences in MSE (train to test): 6483.75%

Differences in RMSE (train to test): 711.40%

R-squared test: 0.4114

R-squared train: 0.9908

KNeighborsRegressor

```
knn = KNeighborsRegressor(n_neighbors=2, metric='minkowski', p=2) # minkowski
knn.fit(X_train, y_train)

model_evaluation(knn, "KHH")
```

KHH

```
[[0. 0.]
 [0. 0.]
 [0. 0.]
 [1. 1.]
 [0. 1.]
 [0. 0.]
 [1. 1.]
 [0. 0.]
 [0. 0.]
 [0. 0.]]
```

Root mean squared error on test set: 0.043375

Root mean squared error on test set (RMSE): 0.20826665599658528

Root mean error: 0.014284375

Root mean r error: 0.11951725816801521

Differences in MSE (train to test): 203.65%

Differences in RMSE (train to test): 74.26%

R-squared test: 0.4536

R-squared train: 0.8154

```
### RandomForestRegressor \ GridSearchCV
```

```
param_grid = [  
    {'n_estimators': [10, 30, 60, 100], 'max_features': [2, 4, 6, 8]},  
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},  
]  
forest_reg = RandomForestRegressor()  
grid_search = GridSearchCV(forest_reg, param_grid, cv=5, # it will train each model five times  
                           scoring='neg_mean_squared_error',  
                           return_train_score=True)  
grid_search.fit(X_train, y_train)
```

```
best_params = grid_search.best_params_  
print(best_params)
```

Executed at 2023.12.10 23:18:15 in 6m 6s 104ms

```
{'max_features': 2, 'n_estimators': 100}
```

```
for_reg = RandomForestRegressor(**best_params, criterion='squared_error',  
                               random_state=0) # n_estimators - кількість дерев у лісі. criterion - функція вимірювання якості розбиття.  
for_reg.fit(X_train, y_train)
```

```
model_evaluation(for_reg, "Grid Search")
```

Executed at 2023.12.10 23:18:21 in 6s 523ms

```
Mean squared error on test set: 0.024940668620135716  
Root mean squared error on test set (RMSE): 0.15792614926013904
```

```
Mean error: 0.00401134765141068  
Root mean r error: 0.06333520072922072
```

```
Differences in MSE (train to test): 521.75%  
Differences in RMSE (train to test): 149.35%
```

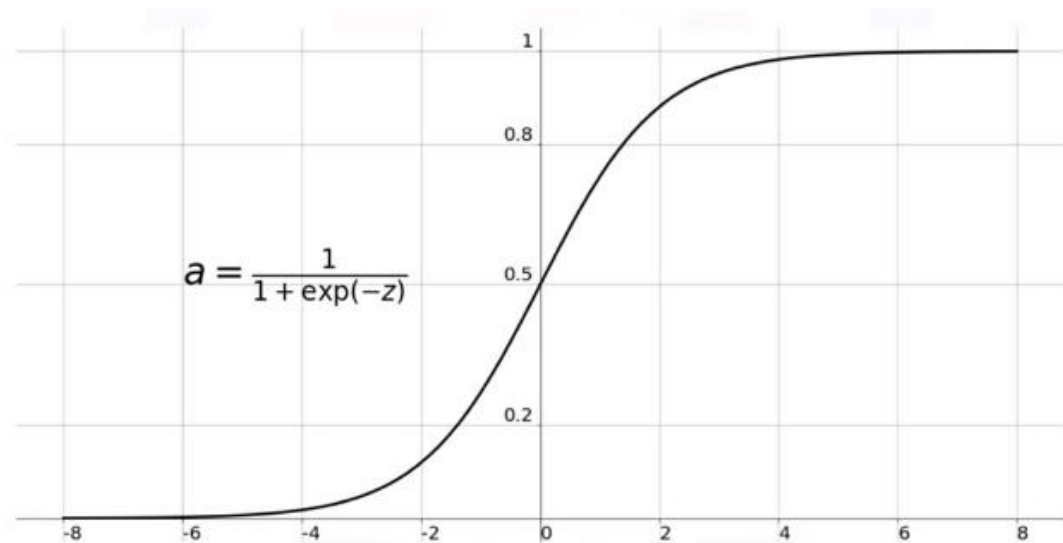
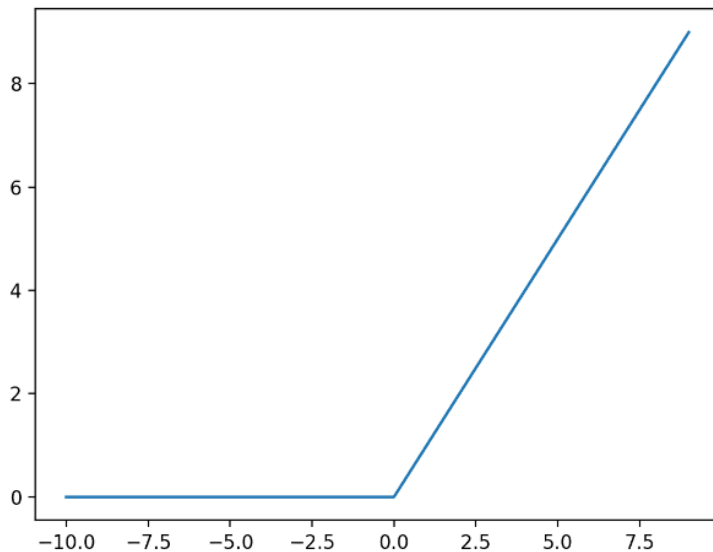
```
R-squared test: 0.6858  
R-squared train: 0.9482
```

Neural Network Creation

```
# Определяем что наша модель последовательная. Т.е. мы добавляем слои один за другим.  
model = Sequential()
```

```
model.add(Dense(10, activation='relu', kernel_initializer='he_normal', input_shape=(n_features,)))  
model.add(Dense(8, activation='relu', kernel_initializer='he_normal'))  
model.add(Dense(1, activation='sigmoid'))
```

8 features → 3 layers → value





MODEL.COMPILE

Оптимізатор у контексті навчання нейронних мереж у машинному навчанні являє собою алгоритм, який визначає спосіб оновлення ваг (параметрів) моделі з метою мінімізації функції втрат.

Функція втрат являє собою математичну функцію, яка вимірює різницю між прогнозованими значеннями моделі та фактичними цільовими значеннями.

Ставимо прапорець на те що хочемо бачити **точність моделі**.

```
model.compile(optimizer='adam', loss="binary_crossentropy", metrics=['accuracy'])
```


and Tuning

```
def build_model(hp):
    model = Sequential()

    model.add(Dense(units=hp.Int('units1', min_value=10, max_value=100, step=10),
                      activation='relu',
                      kernel_initializer='he_normal',
                      input_shape=(n_features,)))
    model.add(Dense(units=hp.Int('units2', min_value=1, max_value=10, step=1),
                      activation='relu',
                      kernel_initializer='he_normal'))
    model.add(Dense(1, activation='sigmoid'))

    model.compile(optimizer='adam', loss="binary_crossentropy", metrics=['accuracy'])

    # Include epochs and batch_size directly in the fit function
    model.fit(X_train, y_train, epochs=hp.Int('epochs', min_value=3, max_value=10),
              batch_size=hp.Int('batch_size', min_value=8, max_value=128, step=16),
              validation_data=(X_train, y_train),
              verbose=0) # Set verbose to 1 for more detailed output during the search

    return model
```

Get epochs

Get batches

Get units

Tuning Results

```
# Instantiate the tuner and perform the search
tuner = RandomSearch(build_model,
                      objective='val_accuracy',
                      max_trials=12, # Adjust as needed
                      directory='classification_diabetes_model', # Change to a directory of your choice
                      project_name='diabetes_recognition',
                      overwrite=True)
```

```
# Perform the search
tuner.search(X_train, y_train, validation_data=(X_train, y_train), verbose=1)
```

```
# Get the best hyperparameters
best_hps = tuner.oracle.get_best_trials(num_trials=1)[0].hyperparameters
```

Executed at 2023.12.10 23:26:41 in 8m 19s 198ms

```
Trial 12 Complete [00h 00m 38s]
val_accuracy: 0.9561874866485596
```

```
Best val_accuracy So Far: 0.9603000283241272
Total elapsed time: 00h 07m 04s
```

```
# Print the best hyperparameters
print("Best Hyperparameters:")
print("units1:", best_hps.get('units1'))
print("units2:", best_hps.get('units2'))
print("epochs:", best_hps.get('epochs'))
print("batch_size:", best_hps.get('batch_size'))
```

Executed at 2023.12.10 23:26:41 in 11ms

Best Hyperparameters:

units1: 70

units2: 3

epochs: 6

batch_size: 56

Model Creation with Tuned parameters

```
# Определяем что наша модель последовательная. Т.е. мы добавляем слои один за другим.
model = Sequential()
# Создаем 3 слоя на 10, 8 и 1 нейрон.
model.add(Dense(best_hps.get('units1'), activation='relu', kernel_initializer='he_normal', input_shape=(n_features,)))
model.add(Dense(best_hps.get('units2'), activation='relu', kernel_initializer='he_normal'))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss="binary_crossentropy", metrics=['accuracy'])
# Берем тренировочные данные. Устанавливаем количество эпох, вводим размер батча и устанавливаем правило вывода. verbose=1
model.fit(X_train, y_train, epochs=best_hps.get('epochs'), batch_size=best_hps.get('batch_size'), verbose=1)
```

```
model_evaluation(model, "Нейросеть")
```

```
Epoch 1/8
10000/10000 [=====] - 10s 976us/step - loss: 0.2427 - accuracy: 0.9257
Epoch 2/8
10000/10000 [=====] - 10s 1ms/step - loss: 0.1634 - accuracy: 0.9433
Epoch 3/8
10000/10000 [=====] - 12s 1ms/step - loss: 0.1390 - accuracy: 0.9520
Epoch 4/8
10000/10000 [=====] - 12s 1ms/step - loss: 0.1296 - accuracy: 0.9550
Epoch 5/8
10000/10000 [=====] - 13s 1ms/step - loss: 0.1238 - accuracy: 0.9567
Epoch 6/8
10000/10000 [=====] - 12s 1ms/step - loss: 0.1216 - accuracy: 0.9570
Epoch 7/8
10000/10000 [=====] - 11s 1ms/step - loss: 0.1172 - accuracy: 0.9593
Epoch 8/8
10000/10000 [=====] - 13s 1ms/step - loss: 0.1171 - accuracy: 0.9592
2500/2500 [=====] - 2s 943us/step
625/625 [=====] - 1s 786us/step
```

Нейросеть

```
[[0.  0. ]
 [0.  0. ]
 [0.  0. ]
 [0.86 1. ]
 [0.64 1. ]
 [0.02 0. ]
 [0.96 1. ]
 [0.  0. ]
 [0.  0. ]
 [0.  0. ]]
```

Root mean squared error on test set: 0.029561349758336092

Root mean squared error on test set (RMSE): 0.17193414366650997

Root mean error: 0.029202876762090227

Root mean r error: 0.17088849218742094

Differences in MSE (train to test): 1.23%

Differences in RMSE (train to test): 0.61%

R-squared test: 0.6276

R-squared train: 0.6226

SVC model creation without tuning

```
from sklearn.preprocessing import StandardScaler
import seaborn as sns

scaler = StandardScaler().fit(X_train)
x_train = scaler.transform(X_train)
x_test = scaler.transform(X_test)

from sklearn.svm import SVC
svc_model = SVC()

svc_model.fit(x_train, y_train)

SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='rbf', max_iter=-1, probability=False, random_state=None,
    shrinking=True, tol=0.001, verbose=False)

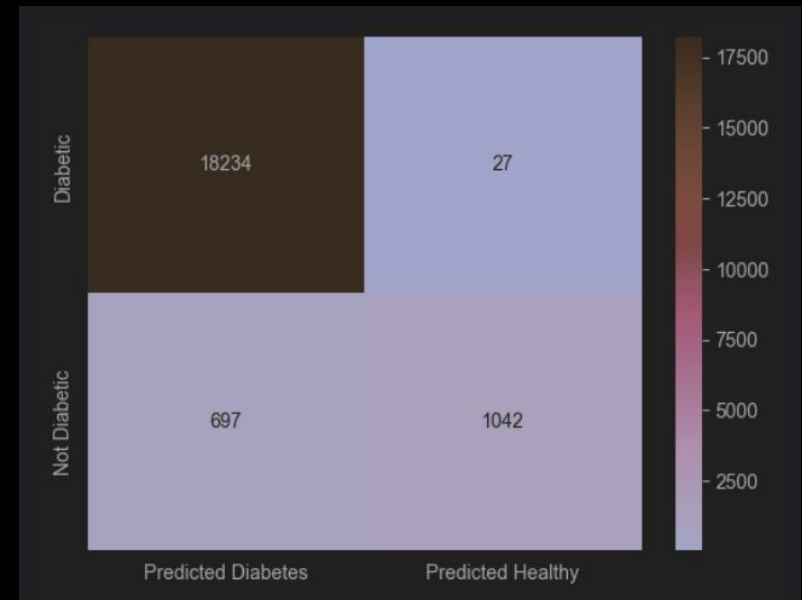
y_predict = svc_model.predict(x_test)

# aaa = pd.DataFrame(["1", "19", "0", "0", "0", "20", "5", "120"])
# x_qwer = svc_model.predict([1, aaa])

from sklearn.metrics import classification_report, confusion_matrix
cm = np.array(confusion_matrix(y_test, y_predict, labels=[0,1]))
confusion = pd.DataFrame(cm, index=['Diabetic', 'Not Diabetic'], columns=['Predicted Diabetes', 'Predicted Healthy'])
sns.heatmap(confusion, annot=True, fmt='g')

print(classification_report(y_test, y_predict))
```

	precision	recall	f1-score	support
0.0	0.96	1.00	0.98	18261
1.0	0.97	0.60	0.74	1739
accuracy			0.96	20000
macro avg	0.97	0.80	0.86	20000
weighted avg	0.96	0.96	0.96	20000



Watch the results for each model

```
# Создание DataFrame из словаря
df = pd.DataFrame(list(models_r2_values.items()), columns=['Model Name', 'R2'])

# Вывод DataFrame
print(f"Результаты: \n{df}")
```

Результаты:

	Model Name	R2
0	Линейная Регрессия	0.358166
1	Дерево Решений	0.411385
2	КНН	0.453644
3	Нейросеть	0.627642

Random Forest: 0.68

Sequential test eq: 96% \ SVC test: 96%

```

new_data = ["1", #Введите пол Female or Male \ 0 or 1
            "29", #Введите возраст 10.0
            "1", #'hypertension' 0 or 1,
            "1", #'heart_disease' 0 or 1,
            "1", #'smoking_history' never, ever, current, not current
            "22.92", #'bmi' 10.0,
            "6.5", #'HbA1c_level' 10.0,
            "10.5" ] #'blood_glucose_level' mmol 6.2 or md\dl
# data_process(new_data)

# строки в числа с плавающей запятой
new_data_numeric = np.array(new_data, dtype=float)

#
new_data_numeric = new_data_numeric.reshape(1, -1) # Изменяем форму на (1, num_features)

#
prediction = model.predict(new_data_numeric)

#
print("Предсказанная вероятность:", prediction * 100)

# Определите классификацию на основе порога
if 0.5 <= prediction <= 0.6:
    print("Человек вероятно заболеет диабетом, в течении 5-10 лет")
elif 0.61 <= prediction <= 0.8:
    print("Человек вероятно заболеет диабетом, в течении 1-3 лет")
elif 0.81 <= prediction <= 1:
    print("Человек вероятно заболеет диабетом, в течении года.")
    if 0.9 <= prediction <= 1:
        print("Срочно обратитесь к эндокринологу!")
elif 0.3 <= prediction <= 0.49:
    print("Человек не находится в зоне риска, но имеет предрасположенность к развитию заболевания. Или проблем с эндокри")
elif 0.2 <= prediction <= 0.29:
    print("Человек не находится в зоне риска, но при этом имеет невыраженные проблемы со здоровьем которые могут развива")
else:
    print("Человек не имеет предпосылок к диабету")

```

Neural Network Predictions

1/1 [=====] - 0s 54ms/step
 Предсказанная вероятность: [[81.01]]
 Человек вероятно заболеет диабетом, в течении года.

ПОБУДОВА СИСТЕМИ БЕЗПЕКИ ІС

Інтеграція нейронної мережі в Інформаційну Систему



ПОБУДОВА СИСТЕМИ БЕЗПЕКИ ІС

ЗАДАЧІ

БІЗНЕС-ПРОЦЕСИ, РЕСУРСИ, ДАНІ

SWOT АНАЛІЗ

ОПИС МОЖЛИВИХ ПОРУШНИКІВ

МЕТОДИ ЗАХИСТУ

ЗАДАЧІ

Пошук та реєстрація пацієнтів

Збір та оновлення показників пацієнтів

Формування та друк результатів прогнозування та наступних рекомендацій

Ведення обліку внутрішніх ресурсів

Формування загальної статистики

Моніторинг та систематичне оновлення системи відповідно до нових медичних вимог

Інтеграція з електронними системами здоров'я

Захист конфіденційності та безпеки даних

Підтримка та навчання користувачів

МАРКЕТИНГ

Розробка маркетингової стратегії для просування системи.

Визначення цільової аудиторії та проведення кампаній для привертання уваги пацієнтів, лікарів та інших зацікавлених сторін.

ЗБІР ПОКАЗНИКІВ КЛІЄНТІВ ТА ПРОГНОЗУВАННЯ РИЗИКІВ ДІАБЕТУ

Розробка інтерфейсу для збору медичних даних від клієнтів.

Використання нейронної мережі для обробки та аналізу зібраних даних з метою прогнозування ризику розвитку діабету.

ОФОРМЛЕННЯ ПЕРСОНАЛІЗОВАНОГО ЗВІТУ ТА ДОСТАВКА ВІДПОВІДІ

Автоматизоване створення персоналізованих звітів для клієнтів на основі результатів прогнозування.

Забезпечення безпечної доставки звітів через електронну пошту або мобільні додатки.

ОТРИМАННЯ ВІДГУКІВ ТА ПОДАЛЬШЕ ВДОСКОНАЛЕННЯ СИСТЕМИ

Впровадження механізмів збору відгуків від клієнтів щодо точності та корисності прогнозування.

Аналіз отриманих відгуків та використання їх для подальшого вдосконалення нейронної мережі та процесів прогнозування.

БІЗНЕС-ПРОЦЕСИ

РЕСУРСИ

Обчислювальні ресурси

- Обладнання для обробки великої кількості даних та проведення аналізу.
- Високопродуктивні обчислювальні ресурси для тренування та використання нейронної мережі.

Дані

- Дані для тренування та вдосконалення нейронної мережі
- Медичні дані клієнтів

Програмне забезпечення

- Використання спеціалізованих бібліотек, таких як TensorFlow та Scikit-learn, для розробки та навчання нейронних мереж.

Інфраструктура для збору даних

- Створення інфраструктури для ефективного збору, зберігання, захисту та обробки медичних даних.

Системи взаємодії та інтеграції

- Розробка зручного інтерфейсу для користувачів, які можуть бути як медичними фахівцями, так і пацієнтами.

Супровід та обслуговування

- Забезпечення технічної підтримки для користувачів та регулярне оновлення системи.
- Постійний моніторинг роботи системи для виявлення помилок та можливостей вдосконалення.

ДАНІ ДЛЯ ЗБОРУ



SWOT АНАЛІЗ

Сильні сторони

- Точність Прогнозування
- Інтеграція з Електронними Системами
 - Швидкість роботи
- Хмарний сервер для зберігання всіх даних
 - Власні сховища даних
 - Простота взаємодії з системою
- Захист Конфіденційності та Безпеки Клієнтів Можливість поєднати експертні знання та статистичні дані
 - Інтелектуальний аналіз даних

Слабкі сторони

- Потреба у Великій Кількості Даних для Навчання
- Залежність від Якості Вхідних Даних
 - Низька Адаптивність
- Підходить лише для попередньої діагностики
- Не здатна повністю замінити спеціаліста у сфері
- Збої при роботі з хмарними сховищами
 - Людський фактор

SWOT АНАЛІЗ

Можливості

- Розширення Функціональності
- Співпраця з Медичними Установами
 - Привабливість для Клієнтів
- Підвищення Якості та Швидкості Лікування
- Можливість модифікації для врахування більшої кількості показників для прогнозування
- Можливість виявити невідомі причини виникнення діабету

Загрози

- Технічні Загрози та Кібербезпека
- Потреба в Швидкій Адаптації до Медичних Змін
 - Недовіра Користувачів до Технологій
 - Нестабільність Медичних Протоколів
 - Зміни в Законодавстві щодо обробки медичної інформації
- Наявність неврахованих важливих показників здоров'я

ОПИС МОЖЛИВИХ ПОРУШНИКІВ



ВНУТРІШНІ ПОРУШНИКИ

АДМІНІСТРАТОРИ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Адміністратори мають великий рівень доступу до системи та можуть мати можливість внесення змін, включаючи конфігурацію системи та доступ до даних.

Міри захисту: Регулярна аудиторія дій адміністраторів, обмеження доступу до критичних даних.

МЕДИЧНИЙ ПЕРСОНАЛ

Лікарі, медсестри та інший медичний персонал, які мають доступ до медичних даних та можуть впливати на процеси системи.

Міри захисту: Застосування строгих правил доступу, обмеження доступу до конкретних пацієнтських даних.

УПОВНОВАЖЕНИЙ ПЕРСОНАЛ РОЗРОБНИКІВ СИСТЕМИ ПО

Розробники, які мають доступ до вихідних кодів та можуть внесення змін у програмний продукт.

Міри захисту: Строга система контролю версій, моніторинг дій розробників, застосування принципу "необхідного доступу".

СПІВРОБІТНИКИ З САНКЦІОНОВАНИМ ДОСТУПОМ ДО ПРИМІЩЕНЬ

Особи, які мають фізичний доступ до приміщень, де зберігаються обчислювальні ресурси.

Міри захисту: Системи контролю доступу та відеоспостереження, обмеження фізичного доступу до обладнання.



ЗОВНІШНІ ПОРУШНИКИ

ХАКЕРИ ТА КІБЕРЗЛОЧИНЦІ

Отримання несанкціонованого доступу до медичних даних.

Заходи захисту: Використання шифрування для збереження конфіденційності даних, регулярне оновлення системи безпеки, використання заходів аутентифікації.

КОНКУРЕНТИ АБО НЕДОБРОЗИЧЛИВІ КОМПАНІЇ

Викривлення або навмисне впливання на результати прогнозування для наведення сумніву на ефективність системи.

Заходи захисту: Забезпечення високої точності та прозорості алгоритмів, використання заходів для виявлення та запобігання впливу сторін ззовні.

НЕДОБРОСОВІСНІ КОРИСТУВАЧІ

Надання неправдивої інформації для отримання спотворених результатів.

Заходи захисту: Перевірка та валідація даних, застосування методів виявлення шахрайства.

ВНЕСОК У РОЗРОБКУ

Павло Чістяков ІТШІ-21-4

Розробка моделі
Розробка нейромережі
Робота з даними
Оформлення

Єва Величко ІТШІ-21-4

Розробка моделі
Розробка нейромережі
Робота з аналізом даних
Оформлення

Єгор Вілянський ІТШІ-21-5

Розробка моделі
Робота з даними
Оформлення