Загружаем данные

```
import pandas as pd
import numpy as np
import copy
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
from sklearn.preprocessing import LabelEncoder
from sklearn.neural_network import MLPRegressor
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn import metrics
from sklearn.metrics import mean_absolute_error
print(tf.__version__)
```

```
xbp_dataFrame = pd.read_excel('/content/drive/MyDrive/X_bp.xlsx',
index_col=0)
xnup_dataFrame = pd.read_excel('/content/drive/MyDrive/X_nup.xlsx',
index_col=0)
```

```
pd.DataFrame(xnup_dataFrame.columns, columns=['характиристики
нашивок'])
```

```
pd.DataFrame(xbp_dataFrame.columns, columns=['характиристики
базальтопластика'])
```

```
xbp_dataFrame.shape
```

```
xnup_dataFrame.shape
```

```
xbp_dataFrame.head()
```

```
xnup_dataFrame.head()
```

Объединяем два файла X_bp.xlsx и X_nup.xslx по индексу, тип объединения INNER.

```
join_dataFrame = xbp_dataFrame.join(xnup_dataFrame, how='inner')
```

```
join_dataFrame.shape
```

```
join_dataFrame.head()
```

```
pd.DataFrame(join_dataFrame.columns, columns=['характиристики
композиционных материалов в join_dataFrame'])
```

Анализ таблицы

```
join_dataFrame.info()
```

Описательная статистика характеристик композиционных материалов

```
join_dataFrame.describe().T
```

Количество уникальных значений

```
join_dataFrame.nunique()
```

```
join_dataFrame.duplicated().sum()
```

```
join_dataFrame.isna()
```

```
join_dataFrame.isna().sum()
```

Гистограммы

```
from matplotlib.colorbar import colorbar_factory
def histodraw(join_dataFrame, variables, n_rows, n_cols):
    fig=plt.figure(figsize=(15, 10))
    for i, var_name in enumerate(variables):
        ax=fig.add_subplot(n_rows,n_cols,i+1)
        sns.histplot(data=join_dataFrame, x=var_name, kde=True,
bins=30, color = 'red')
    fig.tight_layout()
    plt.show()

histodraw(join_dataFrame, join_dataFrame.columns, 4, 4)
```

Попарные графики рассеяния

```
sns.pairplot(join_dataFrame, height=4, diag_kind='kde');
```

Диаграмма "Ящик с усами" до удаления выбросов

```
n = 1  # счетчик
s = 3  # строки
c = 5  # столбцы


fig = plt.figure(figsize=(17, 32))

for i in join_dataFrame.columns:
```

```
    plt.subplot(s, c, n)
    plt.xlabel(i)
    sns.boxplot(y = join_dataFrame[i])
    n = n + 1

plt.show()
```

Визуализация данных с помощью корреляционной матрицы до удаления выбросов

```
cor_map = join_dataFrame.corr()
fig, ax = plt.subplots(figsize=(14, 10))
sns.heatmap(cor_map, vmin=-0.5, vmax=0.5, annot=True,
fmt='.2f',cmap='PRGn', ax=ax, linewidths = 0.1)
plt.show()
```

Корреляции между переменными на тепловой карте визуально не выявлено

Сравнение двух методов удаления выбросов: 3-х сигм или межквартильных расстояний

```
sgm = 0
iqd= 0
for column in join_dataFrame:
    d = join_dataFrame.loc[:, [column]]
    # методом 3-х сигм
    zscore = (join_dataFrame[column] - join_dataFrame[column].mean())
/ join_dataFrame[column].std()
    d['3s'] = zscore.abs() > 3
    sgm += d['3s'].sum()
    # методом межквартильных расстояний
    q1 = np.quantile(join_dataFrame[column], 0.25)
    q3 = np.quantile(join_dataFrame[column], 0.75)
    iqr = q3 - q1
    lower = q1 - 1.5 * iqr
    upper = q3 + 1.5 * iqr
    d['iq'] = (join_dataFrame[column] <= lower) |
(join_dataFrame[column] >= upper)
    iqd+= d['iq'].sum()
print( sgm,'-- выброс методом 3-х сигм',)
print( iqd,'-- выброс методом межквартильных расстояний',)
```

Распределение выбросов по характеристикам

```
temp_dataFrame = join_dataFrame.copy()
for i in join_dataFrame.columns:
    print(f"_____")
    temp_dataFrame[i] = abs((join_dataFrame[i] -
join_dataFrame[i].mean()) / join_dataFrame[i].std())
    print(f"{sum(temp_dataFrame[i] > 3)} -> выбросов в признаке:
```

```
'{i}'")
print(f' Всего - {sum(sum(temp_dataFrame.values > 3))} выброса')
```

"Угол нашивки" закодируем с помощью LabelEncoder

```
le = LabelEncoder()
join_dataFrame['Угол нашивки, град'] =
le.fit_transform(join_dataFrame['Угол нашивки, град'])
```

Удаление выбросов

```
join_dataFrame_drop =
join_dataFrame[(np.abs(stats.zscore(join_dataFrame)) <=
3).all(axis=1)]

join_dataFrame_drop
```

"Ящик с усами" после удаления выбросов

```
n = 1  # счетчик
s = 3  # строки
c = 5  # столбцы


fig = plt.figure(figsize=(17, 32))

for i in join_dataFrame_drop.columns:
    plt.subplot(s, c, n)
    plt.xlabel(i)
    sns.boxplot(y = join_dataFrame_drop[i])
    n = n + 1

plt.show()
```

Корреляционная матрицы после удаления выбросов

```
cor_map = join_dataFrame_drop.corr()
fig, ax = plt.subplots(figsize=(14, 10))
sns.heatmap(cor_map, vmin=-0.5, vmax=0.5, annot=True,
fmt='.2f',cmap='PRGn', ax=ax, linewidths = 0.1)
plt.xticks(rotation=45, ha='right')
plt.show()
```

Корреляции между переменными на тепловой карте визуально не выявлена

Оценка плотности ядра

```
fig, ax = plt.subplots(figsize=(15, 12))
join_dataFrame_drop.plot(kind='kde', ax=ax)
```

Видно, что данные находятся в разных диапазонах. Оценка плотности ядра показывает, что данные нужно нормализовать

```
scaler = preprocessing.MinMaxScaler()
names = join_dataFrame_drop.columns
d = scaler.fit_transform(join_dataFrame_drop)
join_dataFrame_drop_norm = pd.DataFrame(d, columns=names).round(2)
join_dataFrame_drop_norm.head()
```

Описательная статистика характеристик после нормализации

```
join_dataFrame_drop_norm.describe().T.round(2)
```

```
join_dataFrame_drop_norm.info()
```

Оценим ящик с усами после нормализации

```
min_max_scaler = preprocessing.MinMaxScaler()
df_join_clean_norm =
pd.DataFrame(min_max_scaler.fit_transform(join_dataFrame_drop),
    columns = join_dataFrame_drop.columns,
    index = join_dataFrame_drop.index)
sns.set(rc={'figure.figsize':(13,10)})
ax = sns.boxplot(data=join_dataFrame_drop_norm)
ax.set_xticklabels(ax.get_xticklabels(),rotation=90);
```

Гистограмма после нормализации

```
from matplotlib.colorbar import colorbar_factory
def histodraw(join_dataFrame_drop_norm, variables, n_rows, n_cols):
    fig=plt.figure(figsize=(15, 10))
    for i, var_name in enumerate(variables):
        ax=fig.add_subplot(n_rows,n_cols,i+1)
        sns.histplot(data=join_dataFrame_drop_norm, x=var_name,
kde=True, bins=30, color = 'red')
    fig.tight_layout()
    plt.show()

histodraw(join_dataFrame_drop_norm, join_dataFrame_drop.columns, 4, 4)
```

Корреляционная матрица после нормализации данных

```
cor_map = join_dataFrame_drop_norm.corr()
fig, ax = plt.subplots(figsize=(14, 10))
sns.heatmap(cor_map, vmin=-0.5, vmax=0.5,  annot=True, square=True,
fmt='.2f',cmap='PRGn', ax=ax, linewidths = 0.1)
plt.xticks(rotation=45, ha='right')
plt.show()
```

Модели для прогноза модуля упругости при растяжении и прочности при растяжении.

Входы и выходы для моделей

```
output_df_elastic = join_dataFrame_drop_norm['Модуль упругости при
растяжении, ГПа']
output_df_strength = join_dataFrame_drop_norm['Прочность при
растяжении, МПа']

input_df_elastic = join_dataFrame_drop_norm.loc[:,
join_dataFrame_drop_norm.columns != 'Модуль упругости при растяжении,
ГПа']
input_df_strength = join_dataFrame_drop_norm.loc[:,
join_dataFrame_drop_norm.columns != 'Прочность при растяжении, МПа']
```

Разделяем датасет на обучающую и тестовую выборки

```
X_input_elastic, X_output_elastic,  y_input_elastic, y_output_elastic
= train_test_split(input_df_elastic, output_df_elastic ,
test_size=0.3, random_state=42)
X_input_strength, X_output_strength, y_input_strength,
y_output_strength = train_test_split(input_df_strength,
output_df_strength, test_size=0.3, random_state=42)

join_dataFrame_drop_norm.shape[0] - X_input_elastic.shape[0] -
X_output_elastic.shape[0]
join_dataFrame_drop_norm.shape[0] - X_input_strength.shape[0] -
X_output_strength.shape[0]
```

Итоговый датасет ошибок

```
errors_df =
pd.DataFrame(columns=['target_var','model_name','MSE','R2'])
```

Определение функции для визуализации

```
def actual_and_predicted_plot(orig, predict, var, model_name):
    plt.figure(figsize=(17,5))
    plt.title(f'Тестовые и прогнозные значения: {model_name}')
    plt.plot(orig, label='Тест')
    plt.plot(predict, label='Прогноз')
    plt.legend(loc='best')
    plt.ylabel(var)
    plt.xlabel('Количество наблюдений')
    plt.show()
```

Линейная регрессия

```
# модуль упругости при растяжении

linear_model_elastic = LinearRegression()
linear_model_elastic.fit(X_input_elastic,  y_input_elastic)
prediction_y_test_linear_1 =
linear_model_elastic.predict(X_output_elastic)
```

```
MSE_elastic = mean_squared_error(y_output_elastic,
prediction_y_test_linear_1)
R2_elastic = r2_score(y_output_elastic, prediction_y_test_linear_1)

# прочность при растяжении
linear_model_pro = LinearRegression()
linear_model_pro.fit(X_input_strength, y_input_strength)
prediction_y_test_linear_2 =
linear_model_pro.predict(X_output_strength)

MSE_strength = mean_squared_error(y_output_strength,
prediction_y_test_linear_2)
R2_strength = r2_score(y_output_strength, prediction_y_test_linear_2)


linear_errors = pd.DataFrame({'model_name':'Linear Regression',\
                              'target_var':['Модуль упругости при
растяжении, ГПа', 'Прочность при растяжении, МПа'],\
                             'MSE':[MSE_elastic, MSE_strength],\
                             'R2':[R2_elastic, R2_strength]})
errors_df = pd.concat([errors_df, linear_errors], ignore_index=True)
errors_df
```

Визуализация

```
actual_and_predicted_plot(y_output_elastic.values,
prediction_y_test_linear_1, 'Модуль упругости при растяжении, ГПа',
'Linear Regression')
actual_and_predicted_plot(y_output_strength.values,
prediction_y_test_linear_2, 'Прочность при растяжении, МПа', 'Linear
Regression')
```

Регрессия k-ближайших соседей

```
knr_model = KNeighborsRegressor()
neigh_params = {'n_neighbors' : range(1, 101, 1),
          'weights' : ['uniform', 'distance'],
          'algorithm' : ['auto', 'ball_tree', 'kd_tree', 'brute']
          }

GSCV_neigh_1 = GridSearchCV(knr_model, neigh_params, n_jobs=-1, cv=10)
GSCV_neigh_1.fit(X_input_elastic,  y_input_elastic)
# GSCV_neigh_1.best_params_
neigh_1 = GSCV_neigh_1.best_estimator_

MSE_upr = mean_squared_error(y_output_elastic,
neigh_1.predict(X_output_elastic))
R2_upr = r2_score(y_output_elastic, neigh_1.predict(X_output_elastic))
```

```
GSCV_neigh_2 = GridSearchCV(knr_model, neigh_params, n_jobs=-1, cv=10)
GSCV_neigh_2.fit(X_input_strength, y_input_strength)
# GSCV_neigh_2.best_params_
neigh_2 = GSCV_neigh_2.best_estimator_

MSE_pro = mean_squared_error(y_output_strength,
neigh_2.predict(X_output_strength))
R2_pro = r2_score(y_output_strength,
neigh_2.predict(X_output_strength))


neigh_errors = pd.DataFrame({'model_name':'KNeighborsRegressor',\
                                'target_var':['Модуль упругости при
растяжении, ГПа', 'Прочность при растяжении, МПа'],\
                               'MSE':[MSE_upr, MSE_pro],\
                               'R2':[R2_upr, R2_pro]})
errors_df = pd.concat([errors_df, neigh_errors], ignore_index=True)
errors_df
```

Визуализация

```
actual_and_predicted_plot(y_output_elastic.values,
neigh_1.predict(X_output_elastic), 'Модуль упругости при растяжении,
ГПа', 'KNeighbors Regressor')
actual_and_predicted_plot(y_output_strength.values,
neigh_2.predict(X_output_strength), 'Прочность при растяжении, МПа',
'KNeighbors Regressor')
```

Случайный лес

```
rfr_model = RandomForestRegressor(random_state=14)
rfr_model_params = {
    'n_estimators' : range(1, 100, 5),
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : range(1, 5, 1),
    'criterion':['mse']
}
GSCV_rfr_model_upr = GridSearchCV(rfr_model, rfr_model_params, cv=5,
verbose=2)
GSCV_rfr_model_upr.fit(X_input_elastic,  y_input_elastic)
# GSCV_rfr_model_upr.best_params_
rfr_model_upr = GSCV_rfr_model_upr.best_estimator_

MSE_upr = mean_squared_error(y_output_elastic,
rfr_model_upr.predict(X_output_elastic))
R2_upr = r2_score(y_output_elastic,
rfr_model_upr.predict(X_output_elastic))

GSCV_rfr_model_pro = GridSearchCV(rfr_model, rfr_model_params, cv=5,
```

```
verbose=2)
GSCV_rfr_model_pro.fit(X_input_strength, y_input_strength)
# GSCV_rfr_model_pro.best_params_
rfr_model_pro = GSCV_rfr_model_upr.best_estimator_

MSE_pro = mean_squared_error(y_output_strength,
rfr_model_upr.predict(X_output_strength))
R2_pro = r2_score(y_output_strength,
rfr_model_upr.predict(X_output_strength))

rf_errors = pd.DataFrame({'model_name':'RandomForestRegressor',\
                          'target_var':['Модуль упругости при
растяжении, ГПа', 'Прочность при растяжении, МПа'],\
                         'MSE':[MSE_upr, MSE_pro],\
                         'R2':[R2_upr, R2_pro]})
errors_df = pd.concat([errors_df, rf_errors], ignore_index=True)
errors_df
```

Визуализация

```
actual_and_predicted_plot(y_output_elastic.values,
rfr_model_upr.predict(X_output_elastic), 'Модуль упругости при
растяжении, ГПа', 'RandomForestRegressor')
actual_and_predicted_plot(y_output_strength.values,
rfr_model_pro.predict(X_output_strength), 'Прочность при растяжении,
МПа', 'RandomForestRegressor')
```

Многослойный перцептрон

```
mlpr_model = MLPRegressor(random_state=14)
mlpr_model_params = {
    'hidden_layer_sizes' : [(100, 100, 50, 25, 12), (144, 144, 72, 36,
12, 1), (12, 12, 12, 12, 12),
                            (144, 144, 144, 72, 72, 36, 36), ()],
    'activation' : ['identity', 'logistic', 'tanh', 'relu'],
    'solver' : ['sgd', 'adam'],
    'max_iter' : [100],
    'learning_rate' : ['constant', 'adaptive', 'invscaling']
}

GSCV_mlpr_model_upr = GridSearchCV(mlpr_model, mlpr_model_params,
n_jobs=-1, cv=10)
GSCV_mlpr_model_upr.fit(X_input_elastic,  y_input_elastic)
# GSCV_mlpr_model_upr.best_params_
mlpr_model_upr = GSCV_mlpr_model_upr.best_estimator_

MSE_upr = mean_squared_error(y_output_elastic,
mlpr_model_upr.predict(X_output_elastic))
R2_upr = r2_score(y_output_elastic,
mlpr_model_upr.predict(X_output_elastic))
```

```
GSCV_mlpr_model_pro = GridSearchCV(mlpr_model, mlpr_model_params,
n_jobs=-1, cv=10)
GSCV_mlpr_model_pro.fit(X_input_strength, y_input_strength)
# GSCV_mlpr_model_pro.best_params_
mlpr_model_pro = GSCV_mlpr_model_pro.best_estimator_

MSE_pro = mean_squared_error(y_output_strength,
mlpr_model_pro.predict(X_output_strength))
R2_pro = r2_score(y_output_strength,
mlpr_model_pro.predict(X_output_strength))

mlpr_model_errors = pd.DataFrame({'model_name':'MLPRegressor',\
                                  'target_var':['Модуль упругости при
растяжении, ГПа', 'Прочность при растяжении, МПа'],\
                                 'MSE':[MSE_upr, MSE_pro],\
                                 'R2':[R2_upr, R2_pro]})
errors_df = pd.concat([errors_df, mlpr_model_errors],
ignore_index=True)
errors_df
```

Параметры модели по сетке

```
GSCV_mlpr_model_upr.best_params_

GSCV_mlpr_model_pro.best_params_

mlpr_model_upr.predict(X_output_elastic)
```

Визуализация

```
actual_and_predicted_plot(y_output_elastic.values,
mlpr_model_upr.predict(X_output_elastic), 'Модуль упругости при
растяжении, ГПа', 'MLPRegressor')
actual_and_predicted_plot(y_output_strength.values,
mlpr_model_pro.predict(X_output_strength), 'Прочность при растяжении,
МПа', 'MLPRegressor')
```

Лассо регрессия

```
lasso_model = Lasso(random_state=14)
lasso_model_params = {
   'alpha': np.linspace(0, 1, 100)
}
GSCV_lasso_model_upr = GridSearchCV(lasso_model, lasso_model_params,
cv=10, verbose=2)
GSCV_lasso_model_upr.fit(X_input_elastic,  y_input_elastic)
# GSCV_lasso_model_upr.best_params_

lasso_model_upr = GSCV_lasso_model_upr.best_estimator_
```

```
MSE_upr = mean_squared_error(y_output_elastic,
lasso_model_upr.predict(X_output_elastic))
R2_upr = r2_score(y_output_elastic,
lasso_model_upr.predict(X_output_elastic))

GSCV_lasso_model_pro = GridSearchCV(lasso_model, lasso_model_params,
cv=10, verbose=2)
GSCV_lasso_model_pro.fit(X_input_strength, y_input_strength)
# GSCV_lasso_model_pro.best_params_

lasso_model_pro = GSCV_lasso_model_pro.best_estimator_
MSE_pro = mean_squared_error(y_output_strength,
lasso_model_pro.predict(X_output_strength))
R2_pro = r2_score(y_output_strength,
lasso_model_pro.predict(X_output_strength))

lasso_model_errors = pd.DataFrame({'model_name':'lasso_model',\
                                   'target_var':['Модуль упругости при
растяжении, ГПа', 'Прочность при растяжении, МПа'],\
                                  'MSE':[MSE_upr, MSE_pro],\
                                  'R2':[R2_upr, R2_pro]})
errors_df = pd.concat([errors_df, lasso_model_errors],
ignore_index=True)
errors_df
```

Визуализация

```
actual_and_predicted_plot(y_output_elastic.values,
lasso_model_upr.predict(X_output_elastic), 'Модуль упругости при
растяжении, ГПа', 'lasso_model')
actual_and_predicted_plot(y_output_strength.values,
lasso_model_pro.predict(X_output_strength), 'Прочность при растяжении,
МПа', 'lasso_model')
```

Датасет с ошибками

```
errors_df
```

Рекомендательная нейросеть для соотношения матрица-наполнитель

```
df_bp = pd.read_excel('/content/drive/MyDrive/X_bp.xlsx')
df_nup = pd.read_excel('/content/drive/MyDrive/X_nup.xlsx')

df = df_bp.merge(df_nup, on='Unnamed: 0', how='inner')
df.drop(columns=['Unnamed: 0'], inplace=True)

Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3-Q1

df_drop = df[~((df<(Q1-1.5*IQR))|(df>(Q3+1.5*IQR))).any(axis=1)]
```

Сформируем входы и выходы для моделей, разделим обучающую и тестовую

```
target_var = df_drop['Соотношение матрица-наполнитель']
train_vars = df_drop.loc[:, df_drop.columns != 'Соотношение матрица-наполнитель']

x_train, x_test, y_train, y_test = train_test_split(train_vars, target_var, test_size=0.3, random_state=42)
```

Нормализация

```
x_train_norm = tf.keras.layers.Normalization(axis=-1)
x_train_norm.adapt(np.array(x_train))
```

Слои и конфиг нейросети

```
model = tf.keras.Sequential([x_train_norm, layers.Dense(128, activation='relu'),
                                                layers.Dense(128, activation='relu'),
                                                layers.Dense(128, activation='relu'),
                                                layers.Dense(64, activation='relu'),
                                                layers.Dense(32, activation='relu'),
                                                layers.Dense(16, activation='relu'),
                                                layers.Dense(1)
                                               ])

model.compile(optimizer=tf.keras.optimizers.Adam(0.001), loss='mean_squared_error')

model.summary()

model_hist = model.fit(x_train, y_train, epochs=100, verbose=1, validation_split = 0.2)
```

Потери

```
model_hist.history
```

Визуализация потерь на тренировочной и тестовой модели

```
def model_loss_plot(model_hist):
    plt.figure(figsize=(17,5))
    plt.plot(model_hist.history['loss'])
    plt.plot(model_hist.history['val_loss'])
    plt.title('График потерь')
    plt.ylabel('MSE')
```

```
    plt.xlabel('Эпоха')
    plt.legend(['loss', 'val_loss'], loc='best')
    plt.show()
model_loss_plot(model_hist)
```

Визуализация

```
actual_and_predicted_plot(y_test.values, model.predict(x_test.values),
'Соотношение матрица/наполнитель', 'Keras_mlpr_model')
```

Оценка MSE

```
model.evaluate(x_test, y_test, verbose=1)
```

Датасет с ошибками модели

```
MSE = mean_squared_error(y_test, model.predict(x_test.values))
R2 = r2_score(y_test, model.predict(x_test.values))

keras_mlpr_model_errors =
pd.DataFrame({'model_name':'Keras_mlpr_model',\
                                    'target_var':['Соотношение
матрица/наполнитель'],\
                                    'MSE':[MSE],\
                                    'R2':[R2]})
errors_df = pd.concat([errors_df, keras_mlpr_model_errors],
ignore_index=True)
errors_df
```