Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут”
Кафедра АСОІУ

ЗВІТ
про виконання лабораторної роботи №5
з дисципліни
“ Аналіз даних в інформаційно-управляючих системах”

Виконав Студент
2 курсу групи ІП-11
Панченко Сергій

Київ 2023

Main exercise

Import libraries

In [36]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sn
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.preprocessing import PolynomialFeatures
```

Load data

In [37]:
```
df = pd.read_csv('data/winequality-red.csv')
df.head(10)
```

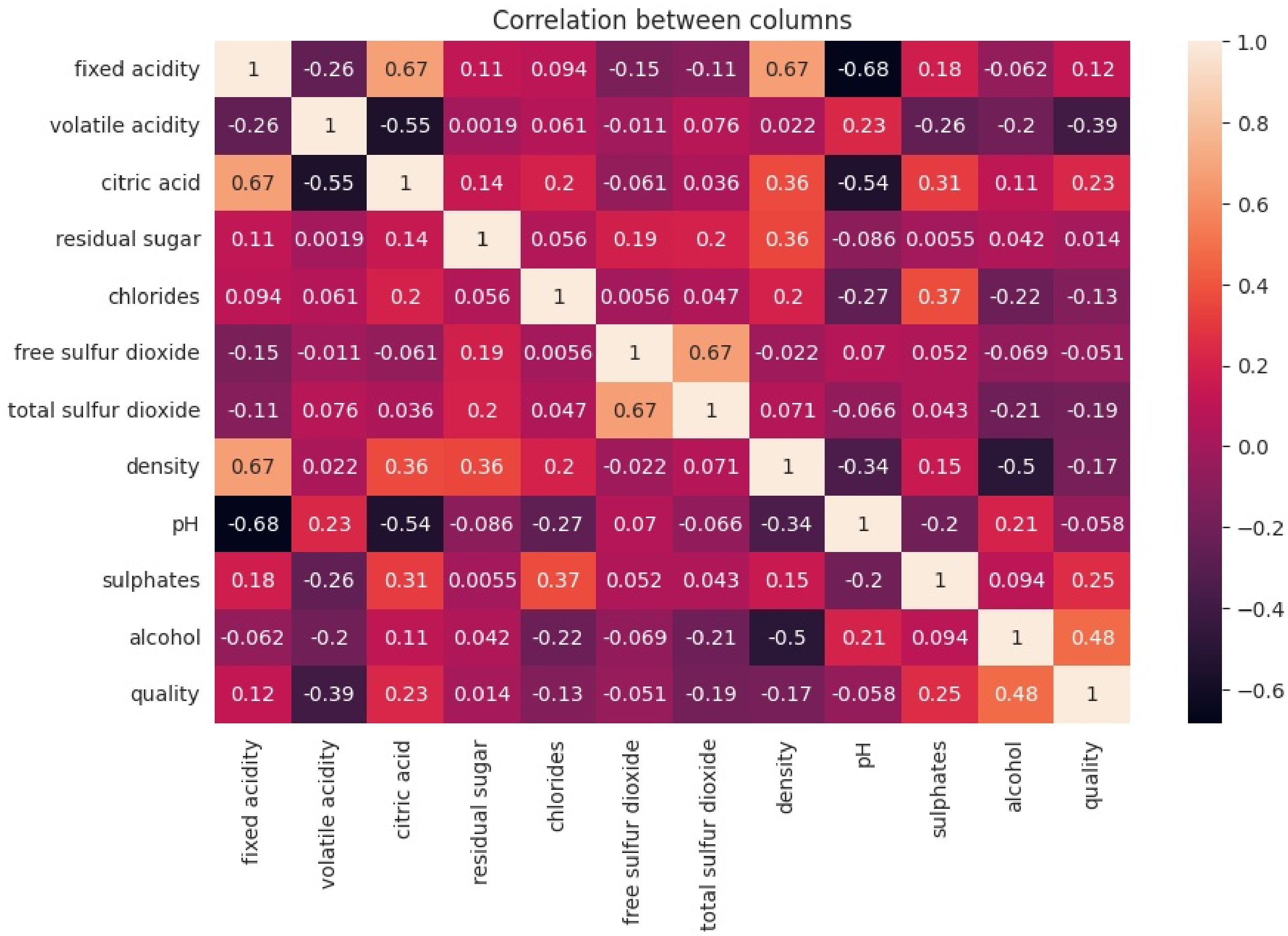
Out[37]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
5	7.4	0.66	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56	9.4	5
6	7.9	0.60	0.06	1.6	0.069	15.0	59.0	0.9964	3.30	0.46	9.4	5
7	7.3	0.65	0.00	1.2	0.065	15.0	21.0	0.9946	3.39	0.47	10.0	7
8	7.8	0.58	0.02	2.0	0.073	9.0	18.0	0.9968	3.36	0.57	9.5	7
9	7.5	0.50	0.36	6.1	0.071	17.0	102.0	0.9978	3.35	0.80	10.5	5

Check for multicollinearity

In [38]:
```
fig, axis = plt.subplots(figsize=(10, 6))
axis.set_title('Correlation between columns')
sn.heatmap(df.corr(), ax=axis, annot=True)
```

Out[38]:
```
<AxesSubplot: title={'center': 'Correlation between columns'}>
```

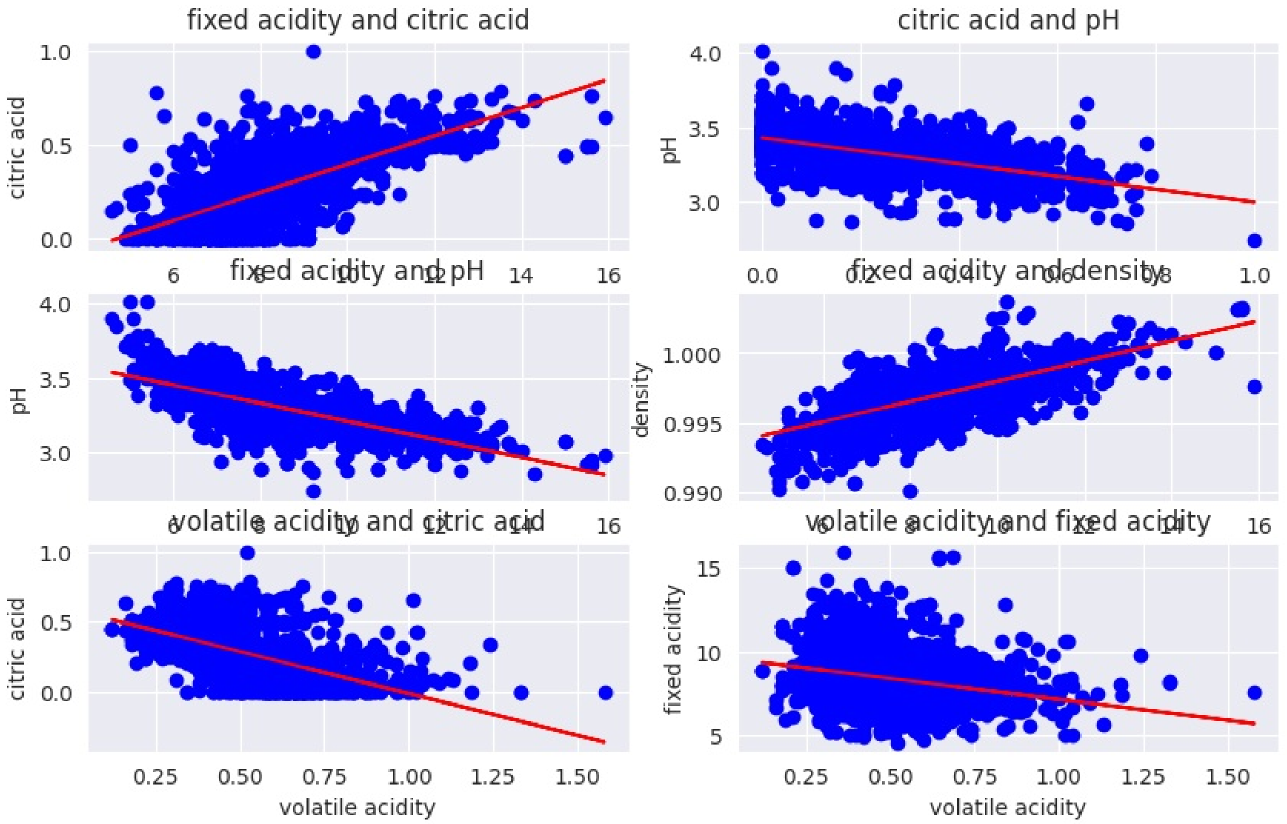


See how fixed acidity and citric acidity, citric acidity and ph, fixed acidity and ph, fixed acidity and density, volatile acidity and citric acidity correlate

In [39]:

```
plot_rows = 3
plot_cols = 2
fig, axis = plt.subplots(plot_rows, plot_cols, figsize=(10, 6))
dependent_pairs = [
    ('fixed acidity', 'citric acid'),
    ('citric acid', 'pH'),
```

```python
    ('fixed acidity', 'pH'),
    ('fixed acidity', 'density'),
    ('volatile acidity', 'citric acid'),
    ('volatile acidity', 'fixed acidity')
]
for index, pair in enumerate(dependent_pairs):
    ax = axis[index // plot_cols][index % plot_cols]
    np_x = df[pair[0]].to_numpy().reshape((-1, 1))
    np_y = df[pair[1]].to_numpy()
    model = LinearRegression().fit(np_x, np_y)
    predictions = model.predict(np_x)
    ax.set_xlabel(pair[0])
    ax.set_ylabel(pair[1])
    ax.set_title(f'{pair[0]} and {pair[1]}')
    ax.scatter(df[pair[0]], df[pair[1]], color='blue')
    ax.plot(np_x, predictions, color='red')
```

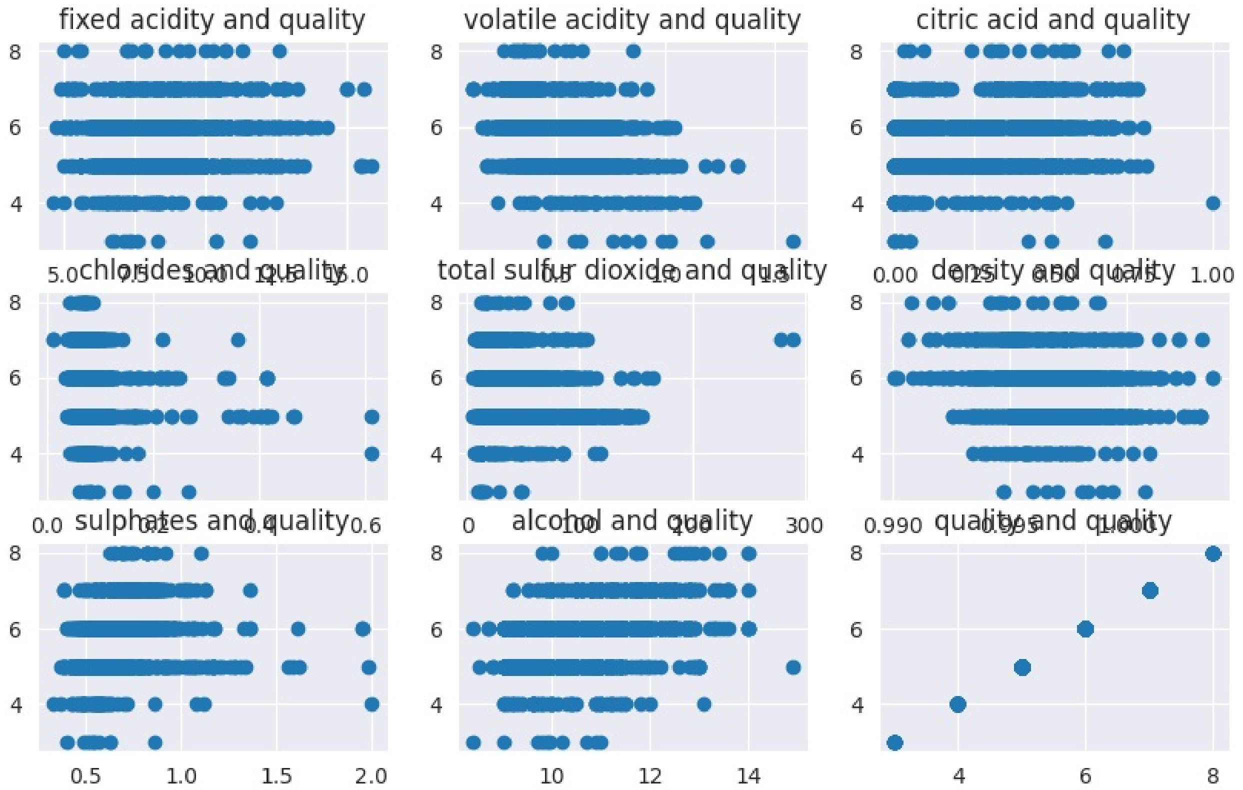


Drop "free sulfur dioxide", "pH", "residual sugar"

In [40]:
```python
df = df.drop(["free sulfur dioxide", "pH", "residual sugar"], axis=1)
```

Graphs projection between factor and quality

In [41]:
```python
plot_rows = 3
plot_cols = 3
fig, axis = plt.subplots(plot_rows, plot_cols, figsize=(10, 6))
for index, col_name in enumerate(df.columns):
    ax = axis[index // plot_cols][index % plot_cols]
    ax.set_title(f'{col_name} and quality')
    ax.scatter(df[col_name], df['quality'])
```



Test and train data

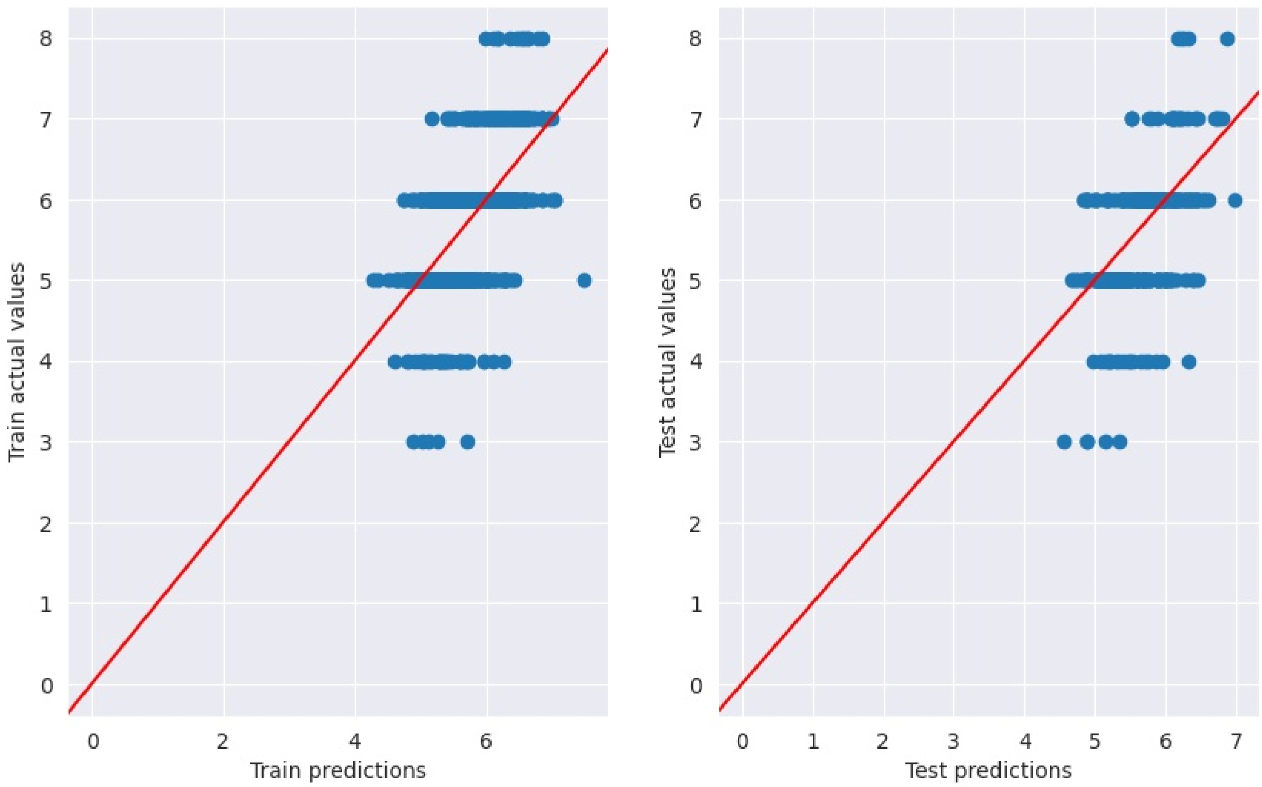
In [42]:
```
df_np = df.to_numpy()
train_data_percentage = 0.75
train_size = int(df_np.shape[0] * train_data_percentage)
x_train, y_train = df_np[:train_size, :-1], df_np[:train_size, -1]
x_test, y_test = df_np[train_size:, :-1], df_np[train_size:, -1]
```

Linear regression

In [43]:
```
linear_model = LinearRegression().fit(x_train, y_train)
linear_predictions_train = linear_model.predict(x_train)
linear_predictions_test = linear_model.predict(x_test)

tests = [
    ('Train', linear_predictions_train, y_train),
    ('Test', linear_predictions_test, y_test),
]
plot_rows = 1
plot_cols = 2
fig, axis = plt.subplots(plot_rows, plot_cols, figsize=(10, 6))
for index, t in enumerate(tests):
    print(f'Linear {t[0]} MAE: {mean_absolute_error(t[1], t[2])}')
    print(f'Linear {t[0]} MSE: {mean_squared_error(t[1], t[2])}')
    axis[index].set_xlabel(f'{t[0]} predictions')
    axis[index].set_ylabel(f'{t[0]} actual values')
    axis[index].scatter(t[1], t[2])
    axis[index].axline([0, 0], [1, 1], color='red')
```

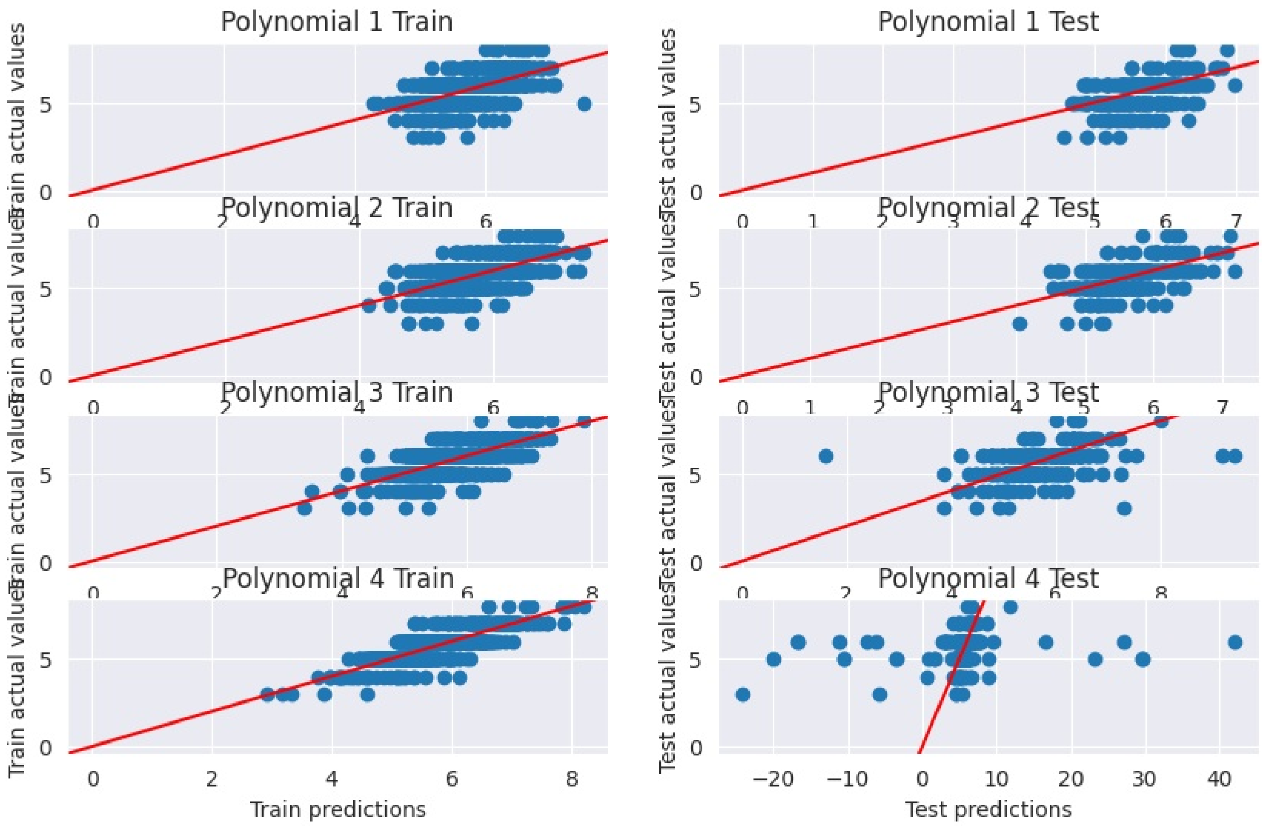
```
Linear Train MAE: 0.5032886845029513
Linear Train MSE: 0.40987623388557665
Linear Test MAE: 0.5145048271954157
Linear Test MSE: 0.46163303389635735
```



Polynomial regression

In [44]:
```
plot_rows = 4
plot_cols = 2
fig, axis = plt.subplots(plot_rows, plot_cols, figsize=(10, 6))
for index, degree in enumerate(range(1, plot_rows + 1)):
    poly_features = PolynomialFeatures(degree=degree, include_bias=False)
    x_poly = poly_features.fit_transform(x_train)
    reg = LinearRegression()
    reg.fit(x_poly, y_train)
    x_train_vals = x_train.copy()
    x_test_vals = x_test.copy()
    x_train_vals_poly = poly_features.transform(x_train_vals)
    x_test_vals_poly = poly_features.transform(x_test_vals)
    polynomial_predictions_train = reg.predict(x_train_vals_poly)
    polynomial_predictions_test = reg.predict(x_test_vals_poly)
    tests = [
        ('Train', polynomial_predictions_train, y_train),
        ('Test', polynomial_predictions_test, y_test),
    ]
    for i, t in enumerate(tests):
        print(f'Polynomial {degree} {t[0]} MAE: {mean_absolute_error(t[1], t[2])}')
        print(f'Polynomial {degree} {t[0]} MSE: {mean_squared_error(t[1], t[2])}')
        axis[index][i].set_title(f'Polynomial {degree} {t[0]}')
        axis[index][i].set_xlabel(f'{t[0]} predictions')
        axis[index][i].set_ylabel(f'{t[0]} actual values')
        axis[index][i].scatter(t[1], t[2])
        axis[index][i].axline([0, 0], [1, 1], color='red')
```

```
Polynomial 1 Train MAE: 0.5032886845029494
Polynomial 1 Train MSE: 0.40987623388557654
Polynomial 1 Test MAE: 0.5145048271954136
Polynomial 1 Test MSE: 0.46163303389635374
Polynomial 2 Train MAE: 0.4764087356043145
Polynomial 2 Train MSE: 0.3701596132951846
Polynomial 2 Test MAE: 0.5206655323786208
Polynomial 2 Test MSE: 0.45591184697065595
Polynomial 3 Train MAE: 0.4131262862314144
Polynomial 3 Train MSE: 0.29338689166854365
Polynomial 3 Test MAE: 0.5833028461650974
Polynomial 3 Test MSE: 0.6430621453420049
Polynomial 4 Train MAE: 0.31437592794325797
Polynomial 4 Train MSE: 0.18137104078176616
Polynomial 4 Test MAE: 1.5299424023370556
Polynomial 4 Test MSE: 18.857602323894717
```



Results

Перед побудовою моделей проаналізували кореляцію та видалили чинники, що не впливають на якість вина. Потім вияснили, як корелюються кислотні чинники. У результаті побудували декілька моделей:

1. Лінійну
2. Поліноміальні зі ступенями від 1 до 4

Бачимо, що зі збільшенням ступенів маємо overfitting, що не дає значну похибко при перевірці моделей на тестових даних. Найкращою виявилася Polynomial 1, яка є по суті тією самою лінійною, але схоже, що метод Polynomial працює більш оптимально за LinearRegression.

Additional assignment

Import libraries

In [45]:
```python
import seaborn as sn
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.preprocessing import PolynomialFeatures

def read_dataset(path: str, sep: str = ';') -> pd.DataFrame:
    data = pd.read_csv(path, sep=sep)
    return data


def replace_comma_with_dots(dataset: pd.DataFrame, column_name: str) -> None:
```

```
    dataset[column_name] = dataset[column_name].astype(str)
    dataset[column_name] = dataset[column_name].str.replace(',', '.')


def convert_column_to_float(dataset: pd.DataFrame, column_name: str) -> None:
    dataset[column_name] = dataset[column_name].astype(float)


def replace_nan_with_mean(dataset: pd.DataFrame, column_name: str):
    mean_value = dataset[column_name].mean()
    dataset[column_name].fillna(value=mean_value, inplace=True)


def convert_float_with_positive(dataset: pd.DataFrame, column_name: str):
    dataset[column_name] = dataset[column_name].abs()


def polynomial_regression(x: pd.DataFrame, y: pd.Series, degree: float):
    np_x = x.values
    np_y = y.to_numpy()
    poly_features = PolynomialFeatures(degree=degree, include_bias=False)
    x_poly = poly_features.fit_transform(np_x)
    reg = LinearRegression()
    reg.fit(x_poly, np_y)
    x_vals_poly = poly_features.transform(np_x)
    predictions = reg.predict(x_vals_poly)
    return np_x, np_y, predictions, poly_features, reg
```

Load data

In [46]:
```
dataset = pd.read_csv('data/Data4.csv', sep=';', encoding='cp1251')
dataset_test = pd.read_csv('data/Data4t.csv', sep=';', encoding='cp1251')
dataset.columns.values[0] = 'COUNTRY'
dataset_test.columns.values[0] = 'COUNTRY'
```

In [47]:
```
dataset.head(10)
```

Out[47]:

	COUNTRY	ISO	UA	Cql	Ie	Iec	Is
0	Albania	ALB	Албанія	0,97392353	0,605347614	0,538672856	0,510112666
1	Algeria	DZA	Алжир	0,782134498	0,58721932	0,348159396	0,497985576
2	Angola	AGO	Ангола	0,372343539	0,27439361	0,332117384	0,346906645
3	Argentina	ARG	Аргентина	0,883830062	0,699685109	0,28199471	0,518820368
4	Armenia	ARM	Вірменія	1,016498793	0,718326882	0,535647909	0,486498047
5	Australia	AUS	Австралія	1,457611163	0,791517131	0,721154637	0,692414115
6	Austria	AUT	Австрія	1,393557395	0,771155098	0,640078065	0,698254396
7	Azerbaijan	AZE	Азербайджан	0,917248997	0,748253484	0,473427808	0,425163807
8	Bangladesh	BGD	Банґладеш	0,401041179	0,194277082	0,38488054	0,386034792
9	Barbados	BRB	Барбадос	1,022513868	0,357017023	0,559189396	0,605988529

In [48]:
```
dataset_test.head(10)
```

Out[48]:

	COUNTRY	ISO	UA	Cql	Ie	Iec	Is
0	Togo	TGO	Того	0,45349807	0,216806252	0,368234721	0,433950896
1	Tunisia	TUN	Туніс	0,899461952	0,659123985	0,418255976	0,514745939
2	Turkey	TUR	Туреччина	0,859283802	0,498840185	0,509228185	0,499453094
3	Uganda	UGA	Уґанда	0,571284014	0,362945628	0,448731923	0,375726313
4	Ukraine	UKR	Україна	0,802203636	0,689164485	0,303554874	0,462744168

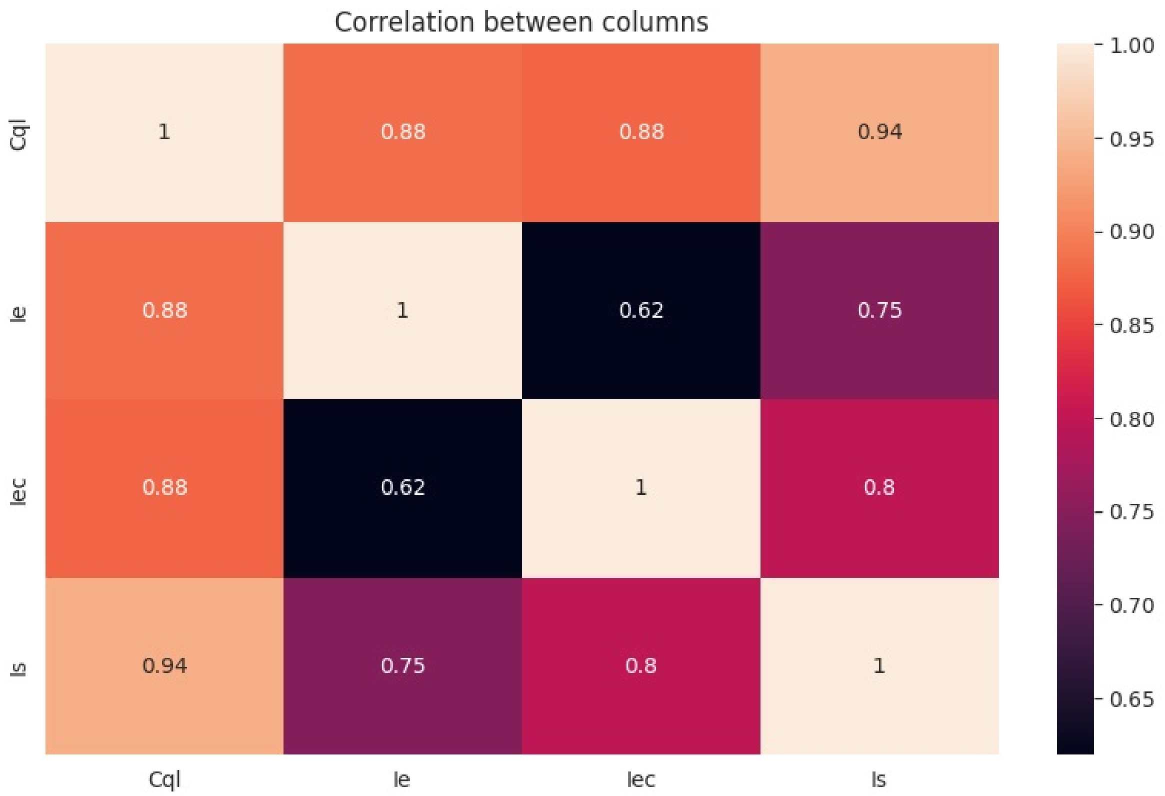
In [49]:
```
for d in [dataset, dataset_test]:
    for column_name in d.columns[3:]:
        replace_comma_with_dots(d, column_name)
        convert_column_to_float(d, column_name)
        replace_nan_with_mean(d, column_name)
        convert_float_with_positive(d, column_name)

df = dataset.loc[:, 'Cql':].copy()
df_test = dataset_test.loc[:, 'Cql':].copy()
```

Check for multicollinearity

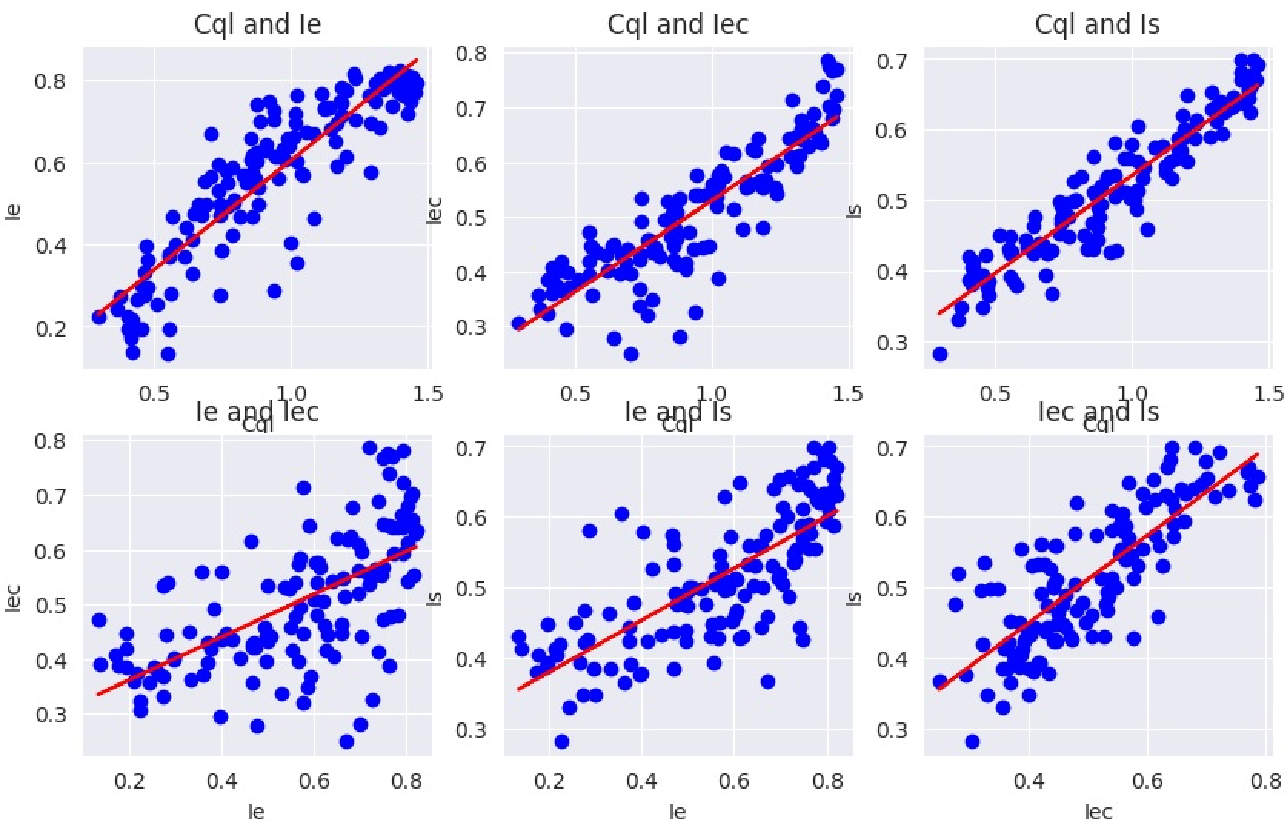
In [50]:
```
fig, axis = plt.subplots(figsize=(10, 6))
axis.set_title('Correlation between columns')
sn.heatmap(df.corr(), ax=axis, annot=True)
```

Out[50]: <AxesSubplot: title={'center': 'Correlation between columns'}>



See correlation between factors

In [51]:
```
plot_rows = 2
plot_cols = 3
fig, axis = plt.subplots(plot_rows, plot_cols, figsize=(10, 6))
dependent_pairs = [
    ('Cql', 'Ie'),
    ('Cql', 'Iec'),
    ('Cql', 'Is'),
    ('Ie', 'Iec'),
    ('Ie', 'Is'),
    ('Iec', 'Is'),
]
for index, pair in enumerate(dependent_pairs):
    ax = axis[index // plot_cols][index % plot_cols]
    ax.scatter(df[pair[0]], df[pair[1]], color='blue')
    np_x, np_y, predictions, _, _ = polynomial_regression(df.loc[:, pair[0]].to_frame(), df[pair[1]], 1)
    ax.set_xlabel(pair[0])
    ax.set_ylabel(pair[1])
    ax.set_title(f'{pair[0]} and {pair[1]}')
    ax.plot(np_x, predictions, color='r')
```

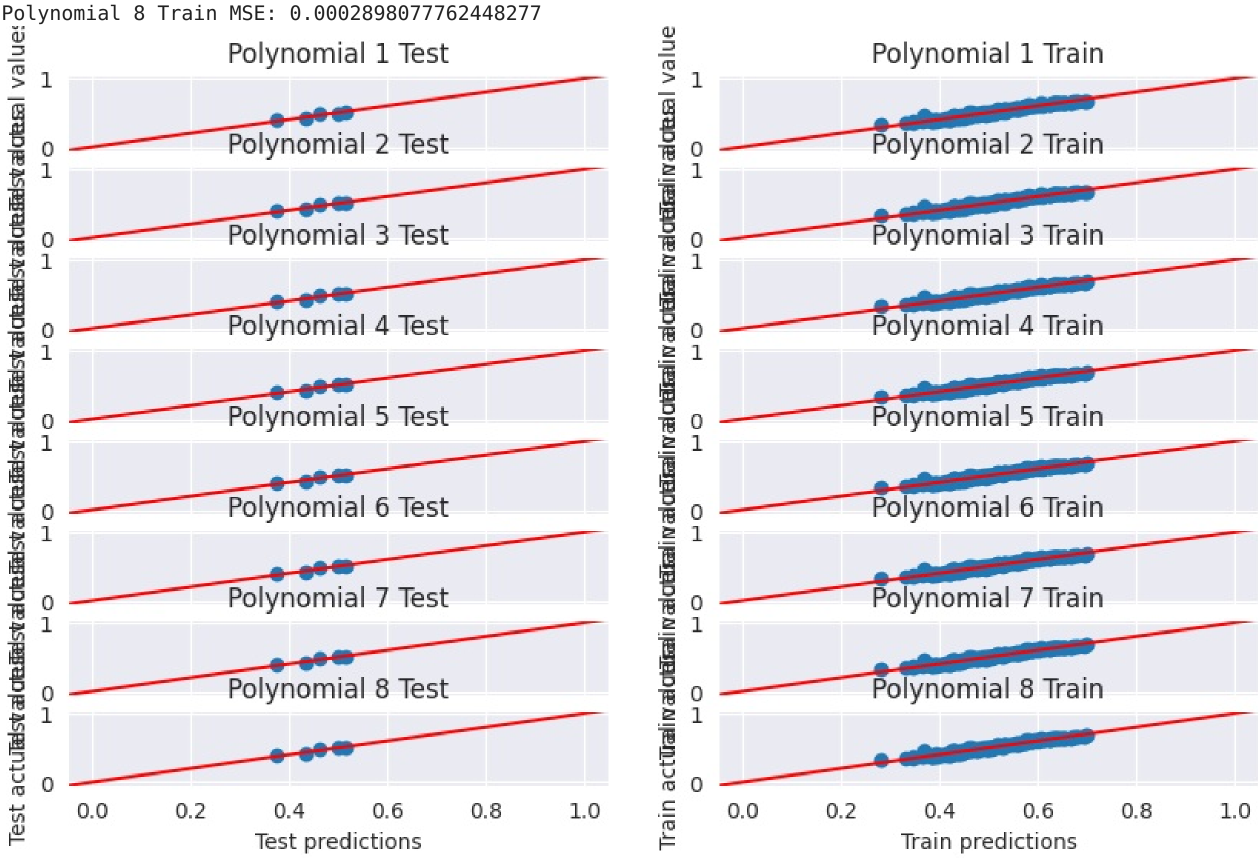


Regressions

In [52]:

```
plot_rows = 8
plot_cols = 2
fig, axis = plt.subplots(plot_rows, plot_cols, figsize=(10, 6))
np_test_x = df_test.iloc[:, :-1].to_numpy()
np_test_y = df_test.iloc[:, -1].to_numpy()
mses = []
for index, degree in enumerate(range(1, plot_rows + 1)):
    res = polynomial_regression(df.iloc[:, :-1], df.iloc[:, -1], 1)
    np_train_x, np_train_y, predictions_train, poly_features, model = res
    x_test_vals_poly = poly_features.transform(np_test_x)
    predictions_test = model.predict(x_test_vals_poly)
    tests = [
        ('Test', np_test_y, predictions_test),
        ('Train', np_train_y, predictions_train)
    ]
    for i, t in enumerate(tests):
        mae = mean_absolute_error(t[1], t[2])
        mse = mean_squared_error(t[1], t[2])
        if t[0] == 'Test':
            mses.append(mse)
        print(f'Polynomial {degree} {t[0]} MAE: {mae}')
        print(f'Polynomial {degree} {t[0]} MSE: {mse}')
        axis[index][i].set_title(f'Polynomial {degree} {t[0]}')
        axis[index][i].set_xlabel(f'{t[0]} predictions')
        axis[index][i].set_ylabel(f'{t[0]} actual values')
        axis[index][i].scatter(t[1], t[2])
        axis[index][i].axline([0, 0], [1, 1], color='red')
```

```
Polynomial 1 Test MAE: 0.01444787215447304
Polynomial 1 Test MSE: 0.0002470262746157831
Polynomial 1 Train MAE: 0.01269984020974859
Polynomial 1 Train MSE: 0.0002898077762448277
Polynomial 2 Test MAE: 0.01444787215447304
Polynomial 2 Test MSE: 0.0002470262746157831
Polynomial 2 Train MAE: 0.01269984020974859
Polynomial 2 Train MSE: 0.0002898077762448277
Polynomial 3 Test MAE: 0.01444787215447304
Polynomial 3 Test MSE: 0.0002470262746157831
Polynomial 3 Train MAE: 0.01269984020974859
Polynomial 3 Train MSE: 0.0002898077762448277
Polynomial 4 Test MAE: 0.01444787215447304
Polynomial 4 Test MSE: 0.0002470262746157831
Polynomial 4 Train MAE: 0.01269984020974859
Polynomial 4 Train MSE: 0.0002898077762448277
Polynomial 5 Test MAE: 0.01444787215447304
Polynomial 5 Test MSE: 0.0002470262746157831
Polynomial 5 Train MAE: 0.01269984020974859
Polynomial 5 Train MSE: 0.0002898077762448277
Polynomial 6 Test MAE: 0.01444787215447304
Polynomial 6 Test MSE: 0.0002470262746157831
Polynomial 6 Train MAE: 0.01269984020974859
Polynomial 6 Train MSE: 0.0002898077762448277
Polynomial 7 Test MAE: 0.01444787215447304
Polynomial 7 Test MSE: 0.0002470262746157831
Polynomial 7 Train MAE: 0.01269984020974859
Polynomial 7 Train MSE: 0.0002898077762448277
Polynomial 8 Test MAE: 0.01444787215447304
Polynomial 8 Test MSE: 0.0002470262746157831
Polynomial 8 Train MAE: 0.01269984020974859
Polynomial 8 Train MSE: 0.0002898077762448277
```



Minimal MSE

In [53]:
```
min_mse = min(mses)
degree = mses.index(min_mse) + 1
print(f'Polynomial {degree} Test MSE: {min_mse}')
```

```
Polynomial 1 Test MSE: 0.0002470262746157831
```

Result

Перед побудовою моделей проаналізували кореляцію. У результаті побудували декілька моделей: поліноміальні зі ступенями від 1 до 4

Бачимо, що зі збільшенням ступенів маємо overfitting, тобто MSE зростає зі збільшенням ступенів. Найкращою виявилася Polynomial 1, яка є по суті тією самою лінійною.