Eksploracja Danych 2022 - Projekt

Autorzy: K. Jarek, P. Witek

Zbiór: "Concrete Data"

Link do zbioru danych

Hipotezy:

1. Im wyższa wartość czynnika Cement i Superplasticizer oraz im więcej dni minęło od wylania betonu (im wyższa wartość Age) tym wyższą wartość będzie miał Concrete compressive strength (wytrzymałość na ściskanie).
2. Im wyższa wartość czynnika Fine Aggregate, Fly Ash i Blast Furnace Slag tym wyższą wartość będzie miał Concrete compressive strength.
3. Im mniejsza wartośc czynnika Water i Fine Aggregate oraz im więcej cementu (Cement) tym wyższą wartość będzie miał Concrete compressive strength.

Analiza statystyczna (Statistical analisys)

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
plt.style.use('fivethirtyeight')
```

In [3]:
```
df = pd.read_excel ("Concrete_Data.xls")
```

In [4]:
```
df.head(20)
```

Out[4]:

	Cement (component 1)(kg in a m^3 mixture)	Blast Furnace Slag (component 2)(kg in a m^3 mixture)	Fly Ash (component 3)(kg in a m^3 mixture)	Water (component 4)(kg in a m^3 mixture)	Superplasticizer (component 5) (kg in a m^3 mixture)	Coarse Aggregate (component 6)(kg in a m^3 mixture)	Fi Aggrega (compone 7)(kg ir m^ mixtur
0	540.0	0.0	0.0	162.0	2.5	1040.0	676
1	540.0	0.0	0.0	162.0	2.5	1055.0	676
2	332.5	142.5	0.0	228.0	0.0	932.0	594
3	332.5	142.5	0.0	228.0	0.0	932.0	594
4	198.6	132.4	0.0	192.0	0.0	978.4	825
5	266.0	114.0	0.0	228.0	0.0	932.0	670
6	380.0	95.0	0.0	228.0	0.0	932.0	594
7	380.0	95.0	0.0	228.0	0.0	932.0	594
8	266.0	114.0	0.0	228.0	0.0	932.0	670
9	475.0	0.0	0.0	228.0	0.0	932.0	594
10	198.6	132.4	0.0	192.0	0.0	978.4	825
11	198.6	132.4	0.0	192.0	0.0	978.4	825
12	427.5	47.5	0.0	228.0	0.0	932.0	594
13	190.0	190.0	0.0	228.0	0.0	932.0	670
14	304.0	76.0	0.0	228.0	0.0	932.0	670
15	380.0	0.0	0.0	228.0	0.0	932.0	670
16	139.6	209.4	0.0	192.0	0.0	1047.0	806
17	342.0	38.0	0.0	228.0	0.0	932.0	670
18	380.0	95.0	0.0	228.0	0.0	932.0	594
19	475.0	0.0	0.0	228.0	0.0	932.0	594

In [5]:
```
df.rename(columns={'Cement (component 1)(kg in a m^3 mixture)': 'Cement',
                   'Blast Furnace Slag (component 2)(kg in a m^3 mixture)': 'Blast
                   'Fly Ash (component 3)(kg in a m^3 mixture)': 'Fly Ash',
                   'Water  (component 4)(kg in a m^3 mixture)': 'Water',
                   'Superplasticizer (component 5)(kg in a m^3 mixture)': 'Superpl
                   'Coarse Aggregate  (component 6)(kg in a m^3 mixture)': 'Coarse
                   'Fine Aggregate (component 7)(kg in a m^3 mixture)': 'Fine Aggr
                   'Age (day)': 'Age',
                   'Concrete compressive strength(MPa, megapascals) ': 'Concrete c
                   inplace=True)
df.columns
```

Out[5]:
```
Index(['Cement', 'Blast Furnace Slag', 'Fly Ash', 'Water', 'Superplasticizer',
       'Coarse Aggregate', 'Fine Aggregate', 'Age',
       'Concrete compressive strength'],
      dtype='object')
```

In [6]:
```
df.dtypes
```

Out[6]:
```
Cement                            float64
Blast Furnace Slag                float64
Fly Ash                           float64
Water                             float64
Superplasticizer                  float64
Coarse Aggregate                  float64
Fine Aggregate                    float64
Age                                 int64
Concrete compressive strength     float64
dtype: object
```

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
Cement                            0
Blast Furnace Slag                0
Fly Ash                           0
Water                             0
Superplasticizer                  0
Coarse Aggregate                  0
Fine Aggregate                    0
Age                               0
Concrete compressive strength     0
dtype: int64
```

In [8]:
```
df.shape
```

Out[8]:
```
(1030, 9)
```

In [9]:
```
df.drop_duplicates(inplace=True)
df.head(10)
```

Out[9]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age	Concrete compressive strength
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28	79.986111
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28	61.887366
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270	40.269535
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365	41.052780
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360	44.296075
5	266.0	114.0	0.0	228.0	0.0	932.0	670.0	90	47.029847
6	380.0	95.0	0.0	228.0	0.0	932.0	594.0	365	43.698299
7	380.0	95.0	0.0	228.0	0.0	932.0	594.0	28	36.447770
8	266.0	114.0	0.0	228.0	0.0	932.0	670.0	28	45.854291
9	475.0	0.0	0.0	228.0	0.0	932.0	594.0	28	39.289790

In [10]:
```
df.reset_index(inplace=True,drop=True)
```

In [11]:
```
print('Shape:', df.shape)
df.tail()
```

```
Shape: (1005, 9)
```

Out[11]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age	Concrete compressive strength
1000	276.4	116.0	90.3	179.6	8.9	870.1	768.3	28	44.284354
1001	322.2	0.0	115.6	196.0	10.4	817.9	813.4	28	31.178794
1002	148.5	139.4	108.6	192.7	6.1	892.4	780.0	28	23.696601
1003	159.1	186.7	0.0	175.6	11.3	989.6	788.9	28	32.768036
1004	260.9	100.5	78.3	200.6	8.6	864.5	761.5	28	32.401235

In [12]:
```
df.describe(include='all')
```

Out[12]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Aggre
count	1005.000000	1005.000000	1005.000000	1005.000000	1005.000000	1005.000000	1005.00
mean	278.629055	72.043134	55.535075	182.074378	6.031647	974.376468	772.68
std	104.345003	86.170555	64.207448	21.340740	5.919559	77.579534	80.33
min	102.000000	0.000000	0.000000	121.750000	0.000000	801.000000	594.00
25%	190.680000	0.000000	0.000000	166.610000	0.000000	932.000000	724.30
50%	265.000000	20.000000	0.000000	185.700000	6.100000	968.000000	780.00
75%	349.000000	142.500000	118.270000	192.940000	10.000000	1031.000000	822.20
max	540.000000	359.400000	200.100000	247.000000	32.200000	1145.000000	992.60

In [13]:
```
df.median()
```

Out[13]:
```
Cement                          265.000000
Blast Furnace Slag               20.000000
Fly Ash                           0.000000
Water                           185.700000
Superplasticizer                  6.100000
Coarse Aggregate                968.000000
Fine Aggregate                  780.000000
Age                              28.000000
Concrete compressive strength    33.798114
dtype: float64
```

In [14]:
```
df.mode()
```

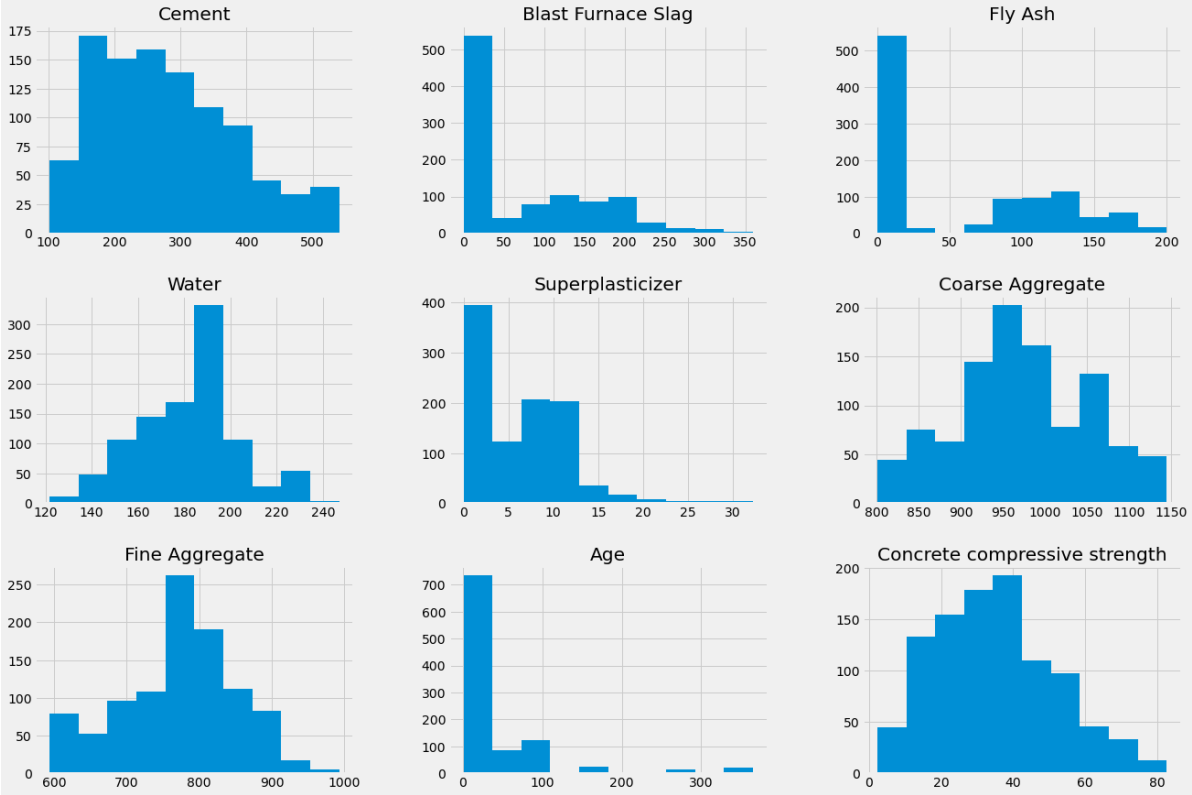
Out[14]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age	Concrete compressive strength
0	251.37	0.0	0.0	192.0	0.0	932.0	594.0	28	31.350474

In [15]:
```
df.var()
```

Out[15]:
```
Cement                               10887.879601
Blast Furnace Slag                    7425.364576
Fly Ash                               4122.596436
Water                                  455.427169
Superplasticizer                        35.041179
Coarse Aggregate                      6018.584052
Fine Aggregate                        6454.491667
Age                                   4062.110923
Concrete compressive strength          265.194960
dtype: float64
```

In [16]:
```
plt.rcParams['figure.figsize'] = [20, 14]
hist = df.hist()
```



In [17]:
```
import statsmodels.api as sm
import statsmodels.formula.api as smf
import scipy.stats as stats
```

In [18]:
```
formula  = "Q('Concrete compressive strength') ~ Cement + Q('Blast Furnace Slag') +
formula += " + Water + Superplasticizer + Q('Coarse Aggregate') + Q('Fine Aggregate

anova_model = smf.ols(formula=formula, data=df).fit()
anova_table = sm.stats.anova_lm(anova_model, typ=1)
anova_table
```

Out[18]:

	df	sum_sq	mean_sq	F	PR(>F)
Cement	1.0	63480.787961	63480.787961	599.397190	5.236427e-104
Q('Blast Furnace Slag')	1.0	18542.677648	18542.677648	175.083348	6.156798e-37
Q('Fly Ash')	1.0	21075.391608	21075.391608	198.997696	2.478336e-41
Water	1.0	10337.111905	10337.111905	97.604898	5.054884e-22
Superplasticizer	1.0	1310.945365	1310.945365	12.378185	4.539953e-04
Q('Coarse Aggregate')	1.0	201.232668	201.232668	1.900076	1.683794e-01
Q('Fine Aggregate')	1.0	2.299374	2.299374	0.021711	8.828880e-01
Age	1.0	45821.207599	45821.207599	432.652208	4.373652e-80
Residual	996.0	105484.086039	105.907717	NaN	NaN

In [19]:
```
anova_model.summary()
```

Out[19]:

OLS Regression Results

Dep. Variable:	Q('Concrete compressive strength')	**R-squared:**	0.604
Model:	OLS	**Adj. R-squared:**	0.601
Method:	Least Squares	**F-statistic:**	189.8
Date:	Wed, 08 Jun 2022	**Prob (F-statistic):**	2.58e-194
Time:	21:42:57	**Log-Likelihood:**	-3764.5
No. Observations:	1005	**AIC:**	7547.
Df Residuals:	996	**BIC:**	7591.
Df Model:	8		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-17.7481	26.419	-0.672	0.502	-69.592	34.096
Cement	0.1172	0.008	13.799	0.000	0.101	0.134
Q('Blast Furnace Slag')	0.0994	0.010	9.790	0.000	0.080	0.119
Q('Fly Ash')	0.0856	0.012	6.862	0.000	0.061	0.110
Water	-0.1526	0.040	-3.835	0.000	-0.231	-0.075
Superplasticizer	0.2834	0.093	3.049	0.002	0.101	0.466
Q('Coarse Aggregate')	0.0156	0.009	1.676	0.094	-0.003	0.034
Q('Fine Aggregate')	0.0183	0.011	1.713	0.087	-0.003	0.039
Age	0.1122	0.005	20.800	0.000	0.102	0.123

Omnibus:	4.034	**Durbin-Watson:**	1.329
Prob(Omnibus):	0.133	**Jarque-Bera (JB):**	3.923
Skew:	-0.150	**Prob(JB):**	0.141
Kurtosis:	3.062	**Cond. No.**	1.05e+05

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.05e+05. This might indicate that there are strong multicollinearity or other numerical problems.

In [20]:
```
corr = df.corr()
corr.style.background_gradient(cmap='coolwarm').set_precision(2)
```

Out[20]:

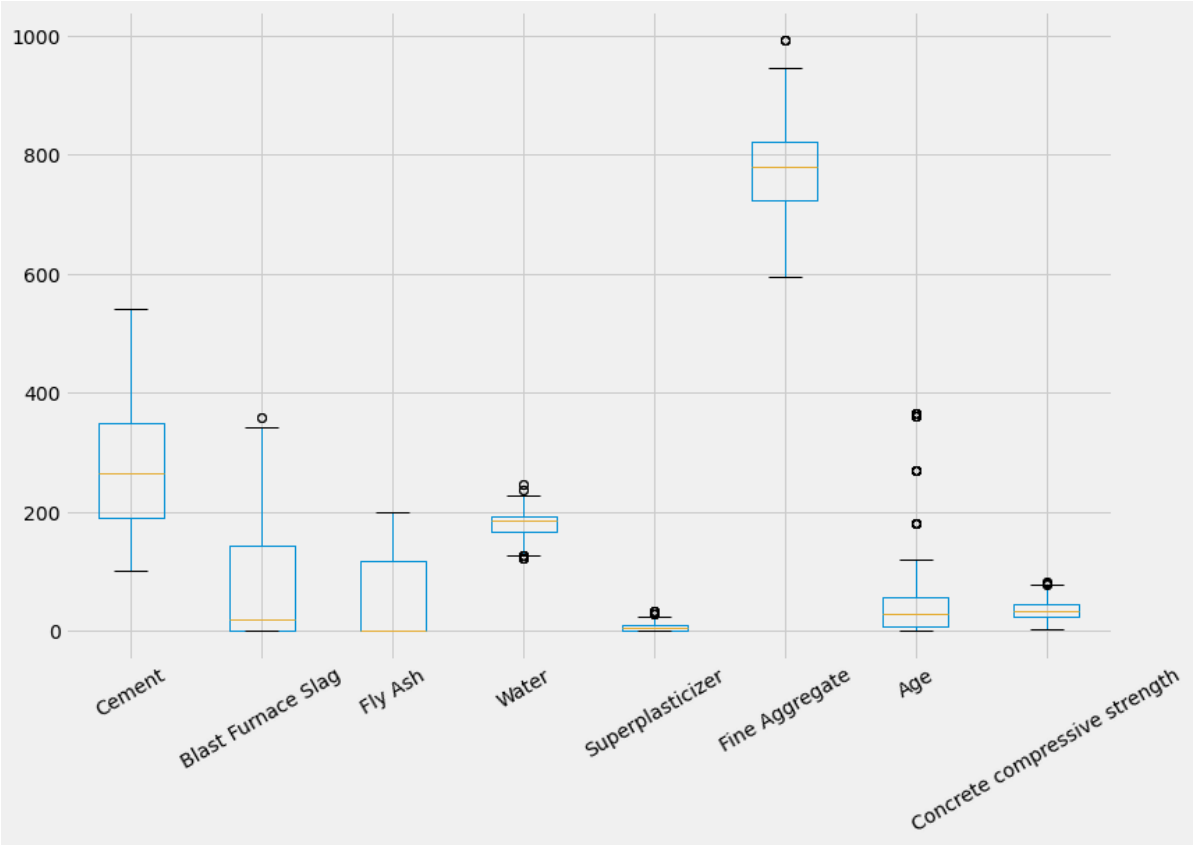
	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age	c
Cement	1.00	-0.30	-0.39	-0.06	0.06	-0.09	-0.25	0.09	
Blast Furnace Slag	-0.30	1.00	-0.31	0.13	0.02	-0.28	-0.29	-0.04	
Fly Ash	-0.39	-0.31	1.00	-0.28	0.41	-0.03	0.09	-0.16	
Water	-0.06	0.13	-0.28	1.00	-0.65	-0.21	-0.44	0.28	
Superplasticizer	0.06	0.02	0.41	-0.65	1.00	-0.24	0.21	-0.19	
Coarse Aggregate	-0.09	-0.28	-0.03	-0.21	-0.24	1.00	-0.16	-0.01	
Fine Aggregate	-0.25	-0.29	0.09	-0.44	0.21	-0.16	1.00	-0.16	
Age	0.09	-0.04	-0.16	0.28	-0.19	-0.01	-0.16	1.00	
Concrete compressive strength	0.49	0.10	-0.08	-0.27	0.34	-0.14	-0.19	0.34	

In [21]:
```
chisquare_stats = stats.chisquare(df, f_exp=None)
print('Statystyka:', chisquare_stats.statistic, '\np-value:', chisquare_stats.pvalu
```

```
Statystyka: [ 39232.91894428 103480.58984564  74531.03916632   2511.33016344
   5832.7924004    6201.56437323   8386.72430923  88937.01262856
   7553.29585965]
p-value: [0.00000000e+000 0.00000000e+000 0.00000000e+000 4.38388802e-130
 0.00000000e+000 0.00000000e+000 0.00000000e+000 0.00000000e+000
 0.00000000e+000]
```

In [22]:
```
df.loc[:, df.columns != 'Coarse Aggregate'].boxplot(rot=30, figsize=(12, 8))
```

Out[22]:
```
<AxesSubplot:>
```

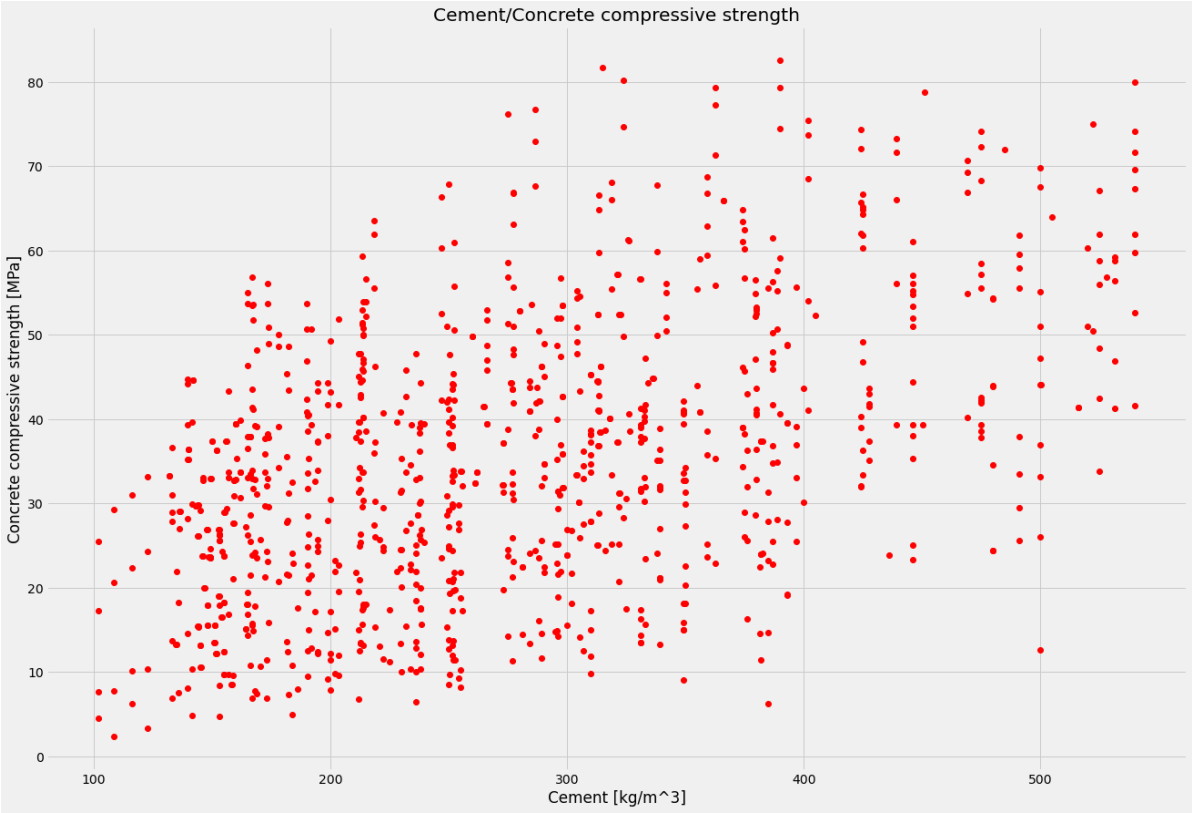


In [23]:
```
# test normalności
shapiro_stat, shapiro_p = stats.shapiro(df[:-1])
print('Statystyka: %f, p-value: %f' % (shapiro_stat, shapiro_p))
```

```
Statystyka: 0.749661, p-value: 0.000000
```

In [24]:
```
plt.plot(df['Cement'], df['Concrete compressive strength'], 'ro')
plt.xlabel('Cement [kg/m^3]')
plt.ylabel('Concrete compressive strength [MPa]')
plt.title('Cement/Concrete compressive strength')

plt.show()
```

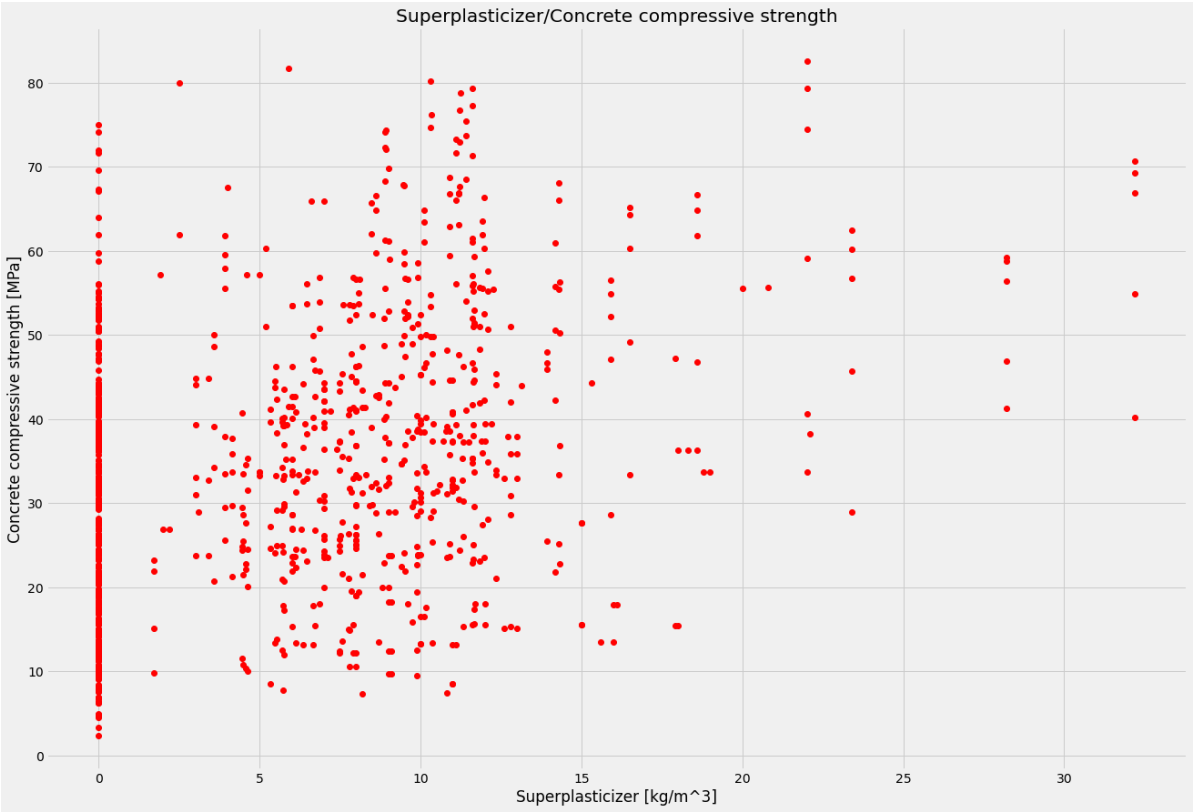


In [25]:
```
plt.plot(df['Age'], df['Concrete compressive strength'], 'go')
plt.xlabel('Age [day]')
plt.ylabel('Concrete compressive strength [MPA]')
plt.title('Age/Concrete compressive strength')

plt.show()
```

In [26]:
```
plt.plot(df['Superplasticizer'], df['Concrete compressive strength'], 'ro')
plt.xlabel('Superplasticizer [kg/m^3]')
plt.ylabel('Concrete compressive strength [MPa]')
plt.title('Superplasticizer/Concrete compressive strength')
```

```
plt.show()
```



Wnioski:

- Predyktory Cement i Superplasticizer są pozytywnie skorelowane z Concrete compressive strength, co potwierdza hipotezę nr 1 i 3.
- Predyktory Water i Fine Aggregate są negatywnie skorelowane z Concrete compressive strength, co potwierdza hipotezę nr 3 i falsyfikuje hipotezę nr 2.
- Predyktory Blast Furnace i Fly Ash nie wykazują korelacji z Concrete compressive strength, co falsyfikuje hipotezę nr 2.
- Cement ma najsilniejszy wpływ na Concrete compressive strength.
- Zmienne objaśniające są ze sobą słabo skorelowane.
- Jednoznacznie trudna jest do określenia zależność danych występująca w ramach zbioru danych. Dla większości zmienych nie da się określeić jej jako liniowej, natomiast dla Cementu pojawia się lionowa rosnąca zależność.

Decision Tree Classifier

In [27]:
```
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn import metrics
from sklearn.model_selection import train_test_split
```

In [28]:
```
X = df.loc[:, df.columns != 'Concrete compressive strength']
y = df['Concrete compressive strength']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

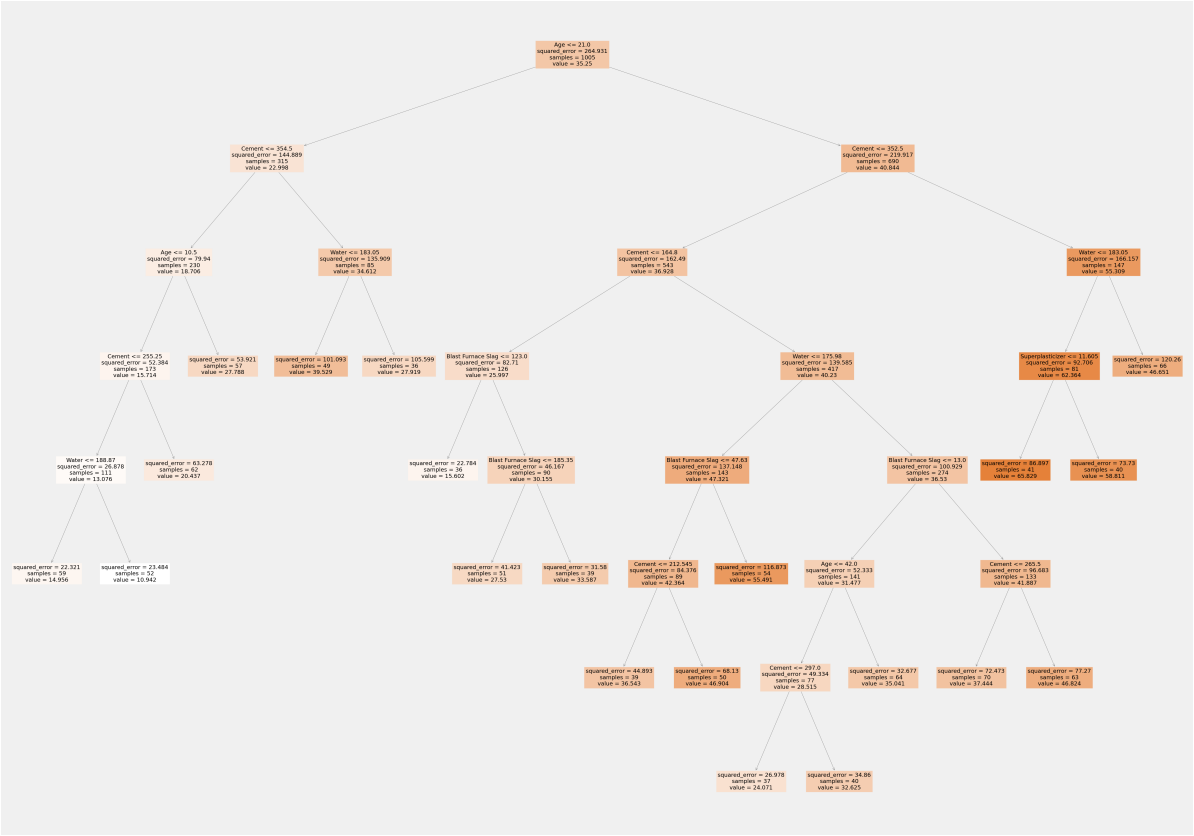
In [29]:
```
dt = DecisionTreeRegressor(min_samples_leaf=0.035, random_state=23)
dt = dt.fit(X, y)
dt.get_params()
```

Out[29]:
```
{'ccp_alpha': 0.0,
 'criterion': 'squared_error',
 'max_depth': None,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 0.035,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'random_state': 23,
 'splitter': 'best'}
```

In [30]:
```
tree_predictions = dt.predict(X_test)
tree_predictions
```

Out[30]:
```
array([27.78806009, 15.60173106, 35.04133878, 46.82405996, 46.65087442,
       27.53028999, 46.82405996, 27.53028999, 10.94206178, 46.82405996,
       15.60173106, 33.58741673, 58.81126859, 55.49065003, 10.94206178,
       55.49065003, 46.9038388 , 14.95611339, 39.52880486, 35.04133878,
       36.54258158, 36.54258158, 35.04133878, 65.82922257, 10.94206178,
       32.62452195, 27.53028999, 58.81126859, 46.9038388 , 37.44360952,
       35.04133878, 27.78806009, 46.82405996, 33.58741673, 46.82405996,
       35.04133878, 35.04133878, 58.81126859, 46.9038388 , 20.43707585,
       20.43707585, 27.78806009, 27.53028999, 36.54258158, 36.54258158,
       10.94206178, 65.82922257, 20.43707585, 27.91922801, 27.53028999,
       46.65087442, 46.65087442, 39.52880486, 20.43707585, 35.04133878,
       10.94206178, 27.53028999, 35.04133878, 32.62452195, 27.53028999,
       32.62452195, 37.44360952, 65.82922257, 20.43707585, 15.60173106,
       46.65087442, 32.62452195, 20.43707585, 46.65087442, 27.78806009,
       27.78806009, 14.95611339, 46.9038388 , 46.82405996, 33.58741673,
       35.04133878, 58.81126859, 35.04133878, 14.95611339, 33.58741673,
       39.52880486, 46.82405996, 27.53028999, 37.44360952, 37.44360952,
       39.52880486, 46.82405996, 37.44360952, 37.44360952, 10.94206178,
       27.78806009, 55.49065003, 32.62452195, 24.07132153, 46.65087442,
       27.53028999, 35.04133878, 46.65087442, 32.62452195, 58.81126859,
       37.44360952, 58.81126859, 39.52880486, 55.49065003, 24.07132153,
       27.91922801, 33.58741673, 46.9038388 , 24.07132153, 10.94206178,
       27.91922801, 27.78806009, 58.81126859, 39.52880486, 37.44360952,
       65.82922257, 58.81126859, 20.43707585, 46.65087442, 14.95611339,
       27.91922801, 55.49065003, 46.65087442, 46.9038388 , 24.07132153,
       14.95611339, 14.95611339, 46.65087442, 65.82922257, 10.94206178,
       39.52880486, 35.04133878, 46.9038388 , 37.44360952, 10.94206178,
       46.82405996, 20.43707585, 65.82922257, 27.91922801, 46.65087442,
       46.65087442, 39.52880486, 24.07132153, 46.82405996, 46.9038388 ,
       27.78806009, 32.62452195, 46.9038388 , 46.65087442, 20.43707585,
       20.43707585, 37.44360952, 65.82922257, 20.43707585, 37.44360952,
       37.44360952, 33.58741673, 46.65087442, 46.9038388 , 39.52880486,
       33.58741673, 20.43707585, 55.49065003, 39.52880486, 24.07132153,
       46.9038388 , 27.91922801, 32.62452195, 14.95611339, 27.78806009,
       36.54258158, 27.78806009, 27.78806009, 39.52880486, 46.65087442,
       35.04133878, 46.9038388 , 46.9038388 , 24.07132153, 58.81126859,
       33.58741673, 33.58741673, 14.95611339, 46.9038388 , 46.65087442,
       10.94206178, 27.53028999, 46.9038388 , 33.58741673, 39.52880486,
       65.82922257, 46.82405996, 46.9038388 , 33.58741673, 55.49065003,
       35.04133878, 46.65087442, 36.54258158, 65.82922257, 20.43707585,
       24.07132153])
```

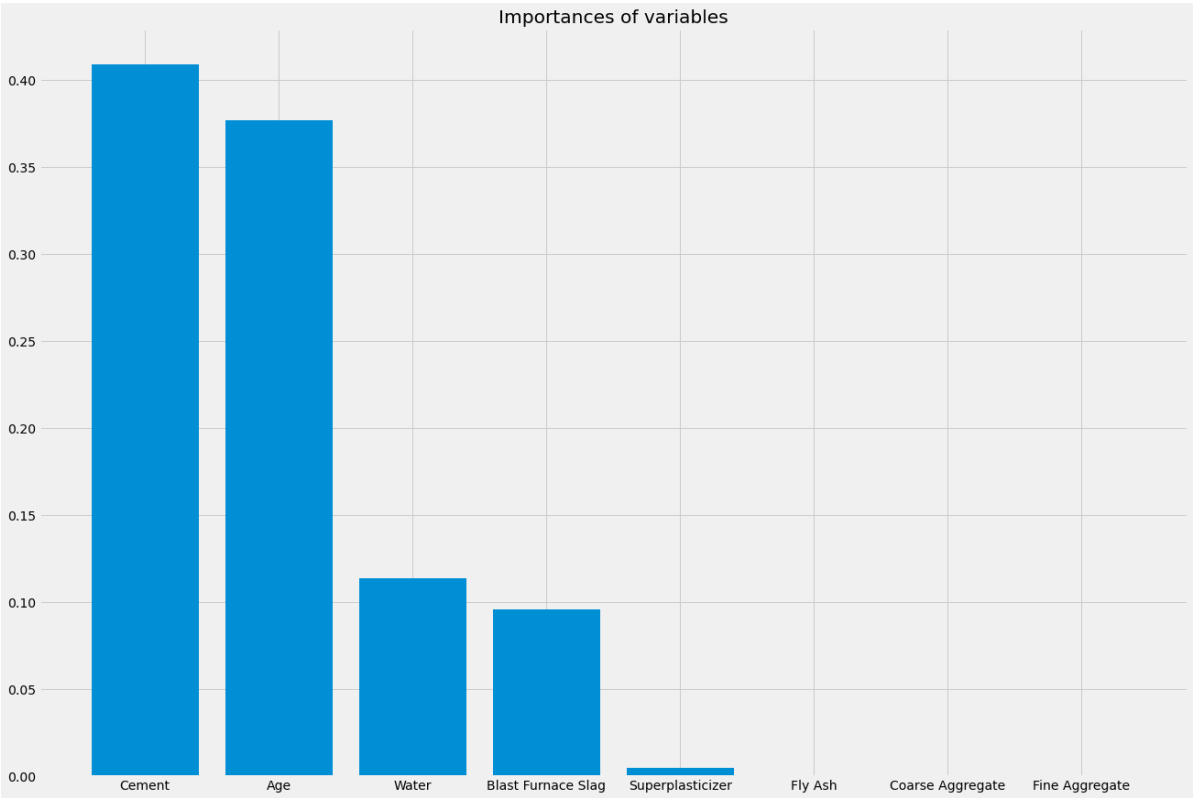
In [31]:
```
plt.figure(figsize=(80,60))
plot_tree(dt,filled=True,feature_names=df.columns)
plt.show()
```



In [32]:
```
r_square = metrics.r2_score(y_test, tree_predictions)
print("Błąd R^2 Dla drzewa regresyjnego:", r_square)
```

```
Błąd R^2 Dla drzewa regresyjnego: 0.7410287339452174
```

In [33]:
```
dt.feature_importances_
importances_df = pd.DataFrame(
                    {'Feature_names': X.columns, 'Importances': dt.feature_importa
                    ).sort_values(by='Importances',ascending=False)
plt.bar(importances_df['Feature_names'],importances_df['Importances'])
plt.title("Importances of variables")
plt.show()
```



Reguły oparte na drzewie klasyfikacynym dla najbardziej wyrazistych klas:

- Cement mający powyżej 21 dni oraz zawierający powyżej 355.5 Concrete i zawierający wody w przedziale <155.5 ; 183> ma wysoką wytrzymałość na średnim poziome 66.124 MPa
- Cement mający powyżej 21 dni oraz zawierający Concrete w przedziale <164.8; 355.5> oraz powyżej 162.7 Blast Furnance Slag ma wytrzymałość o średnim poziomie 19.312 MPa
- Cement mający więcej niż 21 dni ale nie mniej niz 5 dni oraz zawierający mniej niż 354.5 Concrete i zawierający mniej niż 7.85 Superplasticizer ma niską wytrzymałość na średnim poziome 12.032 MPa

Analiza skupień (Clustering)

In [34]:
```
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from mpl_toolkits.mplot3d import Axes3D
from skstab import StadionEstimator
from matplotlib.colors import LogNorm
```

> By skorzystać z pakietu **skstab** konieczne jest wykorzsytanie komendy:
> `python3 setup.py install`

In [35]:
```
feature_1 = 'Cement'
feature_2 = 'Superplasticizer'
feature_3 = 'Concrete compressive strength'
X_cluster = df[[feature_1, feature_2, feature_3]]
```

Zmienne Cement, Superplasticizer, Concrete compressive strength zostały wybrane z racji na wykazywaną pomiędzy sobą nawzajem wyższą korelację niż inne zmienne, czy też z racji na wskazania największego znaczenia dwóch pierwszych zmiennych dla ostatniej rozpoznaną za pomocą użycia drzewa decyzyjnego.

Walidacja 10-krotna:

In [36]:
```
algorithm = KMeans
km_kwargs = {'init': 'k-means++', 'n_init': 10}

k_values = list(range(1, 11))
omega = list(range(2, 6))

stab = StadionEstimator(X_cluster.values, algorithm,
                        param_name='n_clusters',
                        param_values=k_values,
                        omega=omega,
                        extended=True,
                        runs=10,
                        perturbation='uniform',
                        perturbation_kwargs='auto',
                        algo_kwargs=km_kwargs,
                        n_jobs=-1)
```

In [37]:
```
score_val_knn = stab.score(strategy='max', crossing=True)
k_best = stab.select_param()[0]
print('Stadion-max scores:\n', score_val_knn)
print('Uzyskana liczba najbardziej dopasowanych klastrów: k =', k_best)
```

```
Stadion-max scores:
 [[0.00410932]
 [0.01089761]
 [0.01520808]
 [0.02442832]
 [0.03357868]
 [0.01984942]
 [0.02970264]
 [0.03048916]
 [0.03203187]
 [0.01961758]]
Uzyskana liczba najbardziej dopasowanych klastrów: k = 5
```

In [38]:
```
kmeans_model = KMeans(n_clusters=k_best)
kmeans_model.fit(X_cluster)

CC = kmeans_model.cluster_centers_
kmeans_T = X_cluster.values
kmeans_labels = kmeans_model.labels_

print('Centroidy:\n', CC)
print('Przypisanie poszczególnych rekordów do klastrów:', kmeans_labels)
```

```
Centroidy:
 [[383.71719745   7.39127389  42.91697253]
 [160.10722222   6.20944444  26.65424584]
 [306.12078189   5.25798354  37.6450633 ]
 [489.19684211   7.09442105  51.52233723]
 [232.03583333   5.30485417  31.03975315]]
Przypisanie poszczególnych rekordów do klastrów: [3 3 2 ... 1 1 4]
```

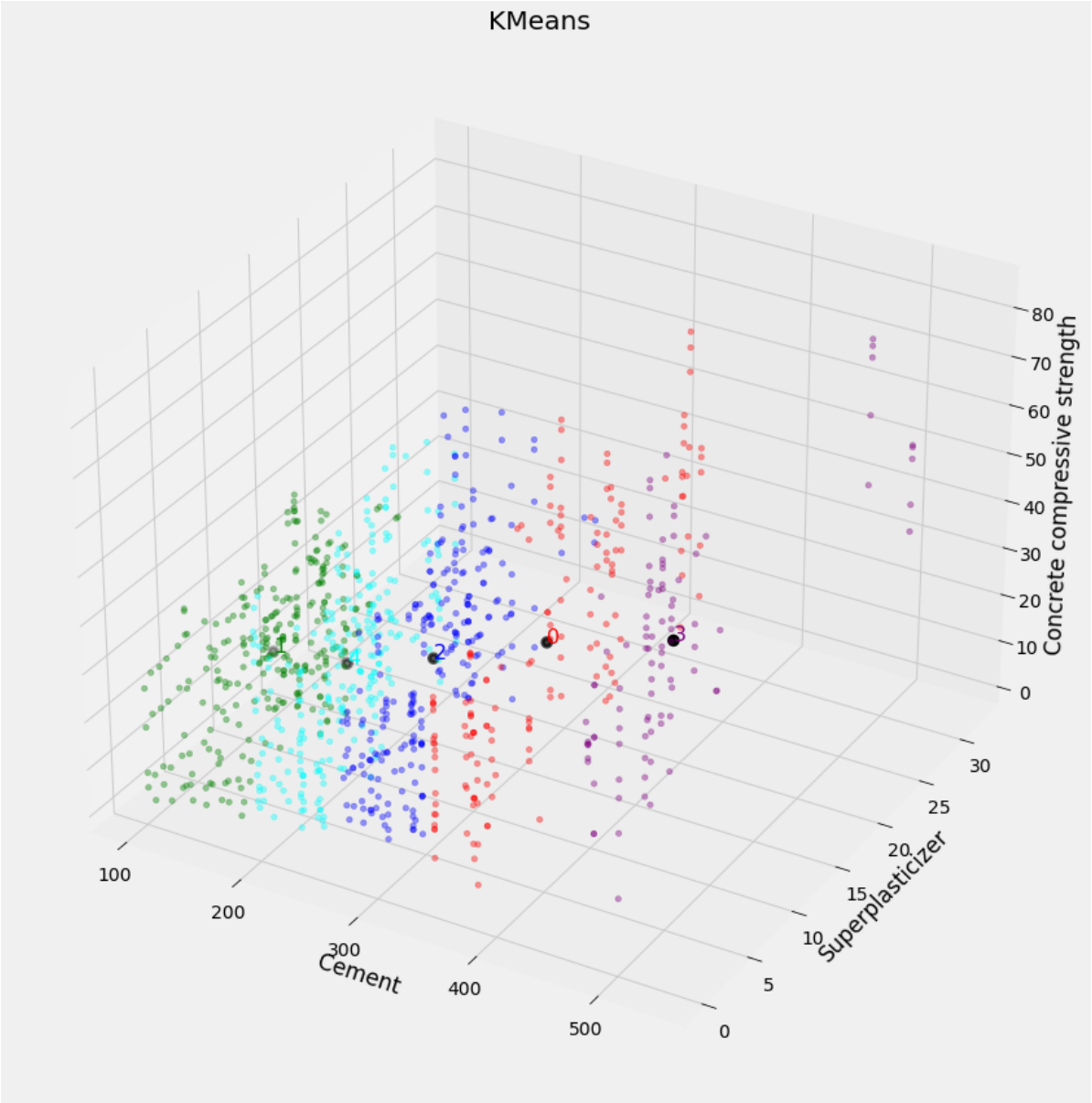
In [39]:
```
color_cluster = ['red', 'green', 'blue', 'purple', 'cyan', 'magenta', 'yellow', 'bl

fig = plt.figure()

ax = fig.add_subplot(projection='3d')
ax.set_title('KMeans')
ax.set_xlabel(feature_1)
ax.set_ylabel(feature_2)
ax.set_zlabel(feature_3)

sample_colors = [ color_cluster[kmeans_labels[i]] for i in range(len(kmeans_T)) ]
ax.scatter(kmeans_T[:, 0], kmeans_T[:, 1], kmeans_T[:, 2], c=sample_colors, marker=
ax.scatter(CC[:, 0], CC[:, 1], CC[:, 2], marker='.', s=169, linewidths=3, zorder=10

for i in range(len(CC)):
    ax.text(CC[i, 0], CC[i, 1], CC[i, 2], str(i), fontsize=16, color=color_cluster
```



Wnioski:

Jak na podstawie wizualizacji można zauważyć to mimo przeprowadzonej walidacji granice między klastrami są mało wyraźne i ciężko podjąć się precyzyjnego opisu poszczególnych zgrupowań z racji na efemeryczny charakter wyników.
Jednak, co można zauważyć analizując współrzędne centroidów:

- dla parametru Cement wynoszącego 383.71719745 i Superplasticizer wynoszącego 7.39127389 udało się uzyskać wartość wytrzymałości na ściskanie betonu wynoszącego 42.91697253, co świadczy, że dany klaster grupował próbki o nieco wyższej wytrzymałości niż średnia populacji,
- dla parametru Cement wynoszącego 160.10722222 i Superplasticizer wynoszącego 6.20944444 udało się uzyskać wartość wytrzymałości na ściskanie betonu wynoszącego 26.65424584, co świadczy, że dany klaster grupował próbki o wytrzymałości jednoznacznie niższej od średniej (słabe),
- dla parametru Cement wynoszącego 306.12078189 i Superplasticizer wynoszącego 5.25798354 udało się uzyskać wartość wytrzymałości na ściskanie betonu wynoszącego

37.6450633, co świadczy, że dany klaster grupował próbki o wytrzymałości zbliżonej do średniej całej populacji,

- dla parametru Cement wynoszącego 489.19684211 i Superplasticizer wynoszącego 7.09442105 udało się uzyskać wartość wytrzymałości na ściskanie betonu wynoszącego 51.52233723, co świadczy, że dany klaster grupował próbki o wysokiej wytrzymałości na ściskanie,
- dla parametru Cement wynoszącego 232.03583333 i Superplasticizer wynoszącego 5.30485417 udało się uzyskać wartość wytrzymałości na ściskanie betonu wynoszącego 31.03975315, co świadczy, że dany klaster grupował próbki o wytrzymałości zbliżonej do średniej, ale jednak od niej niższej (słabsze).

- Na podstawie powyższych można rozpoznać, że wyższe wartości dwóch perwszych zmiennych korespondują z wyższą wartością trzeciej zmiennej, co pokazuje, że powyższa analiza skupień potwierdza hipotezę nr 1.

In [40]:
```
gmm_components = len(X_cluster.columns)

gmm = GaussianMixture(n_components=gmm_components, covariance_type="full", random_
gmm.fit(X_cluster)
```

Out[40]:
```
GaussianMixture(n_components=3, random_state=23)
```

In [41]:
```
print('Średnie dla komponentów:')
print(gmm.means_)
print('\nMacierz kowariancji:')
print(gmm.covariances_)
print('\nMacierz precyzji:')
print(gmm.precisions_)
```

```
Średnie dla komponentów:
[[294.618214     4.63264661  35.56097713]
 [433.79452559   9.38265035  48.65184372]
 [178.15320756   5.91657158  27.83302482]]

Macierz kowariancji:
[[[ 2.56303338e+03 -6.12981530e+01  1.30339349e+02]
  [-6.12981530e+01  2.26543607e+01  3.80408207e+01]
  [ 1.30339349e+02  3.80408207e+01  2.38198631e+02]]

 [[ 4.02172982e+03 -1.61449302e+02  2.56079031e+02]
  [-1.61449302e+02  7.08076369e+01  2.47093257e+01]
  [ 2.56079031e+02  2.47093257e+01  2.32200568e+02]]

 [[ 1.21375458e+03  6.50697118e-01  1.07951634e+02]
  [ 6.50697118e-01  2.24932185e+01  7.85913406e+00]
  [ 1.07951634e+02  7.85913406e+00  1.64016808e+02]]]

Macierz precyzji:
[[[ 0.00047961  0.00237543 -0.0006418 ]
  [ 0.00237543  0.07208177 -0.01281142]
  [-0.0006418  -0.01281142  0.00659537]]

 [[ 0.00031084  0.00086034 -0.00043436]
  [ 0.00086034  0.01704863 -0.00276302]
  [-0.00043436 -0.00276302  0.00507967]]

 [[ 0.00087583  0.00017907 -0.00058503]
  [ 0.00017907  0.04525145 -0.00228616]
  [-0.00058503 -0.00228616  0.00659153]]]
```

In [42]:

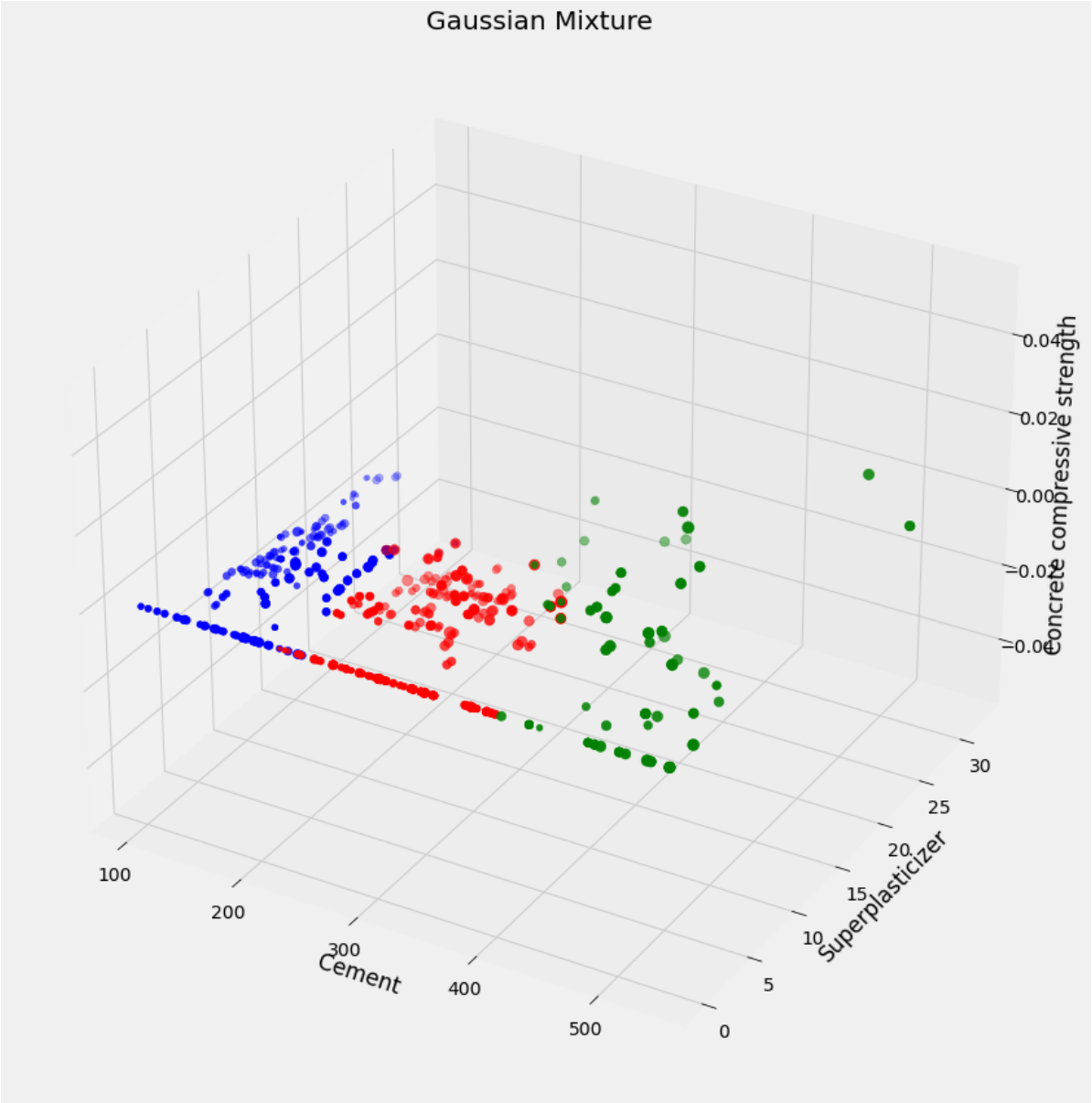
```
fig = plt.figure()
ax = fig.add_subplot(projection='3d')

labels = gmm.predict(X_cluster)
frame = pd.DataFrame(X_cluster)
frame['cluster'] = labels
frame.columns = [feature_1, feature_2, feature_3, 'cluster']

for k in range(gmm_components):
    data = frame[frame['cluster']==k]
    plt.scatter(data[feature_1], data[feature_2], data[feature_3], c=color_cluster

ax.set_xlabel(feature_1)
ax.set_ylabel(feature_2)
ax.set_zlabel(feature_3)
ax.set_title("Gaussian Mixture")

plt.show()
```



Wnioski:

Jak powyżej można zauważyć to na podstawie przeprowadzonego badania z wykorzystaniem Gaussian Mixture zmienne Cement i Superplasticizer pozwalają dokonać selekcji trzech jednoznacznie prezentujących się klastrów, grupujących się elipsoidalnie. Jednak, co można zauważyć analizując współrzędne punktów odpowiadających punktom średnim:

- dla parametru Cement wynoszącego 294.618214 i Superplasticizer wynoszącego 4.63264661 udało się uzyskać wartość wytrzymałości na ściskanie betonu wynoszącej 35.56097713, co świadczy, że dany klaster grupował próbki o wytrzymałości zbliżonej do średniej populacji,
- dla parametru Cement wynoszącego 433.79452559 i Superplasticizer wynoszącego 9.38265035 udało się uzyskać wartość wytrzymałości na ściskanie betonu wynoszącej 48.65184372, co świadczy, że dany klaster grupował próbki o wyższej wytrzymałości niż średnia populacji,
- dla parametru Cement wynoszącego 178.15320756 i Superplasticizer wynoszącego 5.91657158 udało się uzyskać wartość wytrzymałości na ściskanie betonu wynoszącej

27.83302482, co świadczy, że dany klaster grupował próbki o wytrzymałości o wiele niższej niż średnia populacji.

- Na podstawie powyższych można rozpoznać, że wyższe wartości dwóch perwszych parametrów korespondują z wyższą wartością trzeciej zmiennej, co pokazuje, że powyższa analiza skupień z wykorzystaniem Gaussian Mixture potwierdza hipotezę nr 1.

Over-Sampling

In [43]:
```
from sklearn.preprocessing import StandardScaler
try:
    import smogn
except ImportError:
    !pip install smogn
    import smogn
```

In [44]:
```
df.shape
```

Out[44]:
```
(1005, 9)
```

In [45]:
```
df = smogn.smoter(

    data = df,
    y = 'Concrete compressive strength',
    k = 9,
    samp_method = 'extreme',
    rel_thres = 0.80,
    rel_method = 'auto',
    rel_xtrm_type = 'high',
    rel_coef = 1.7
)
```

```
dist_matrix: 100%|##########| 42/42 [00:00<00:00, 43.39it/s]
synth_matrix: 100%|##########| 42/42 [00:02<00:00, 17.79it/s]
r_index: 100%|##########| 37/37 [00:00<00:00, 333.52it/s]
```

In [46]:
```
X = df.loc[:, df.columns != 'Concrete compressive strength']
y = df['Concrete compressive strength']
```

In [47]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(1539, 8) (385, 8) (1539,) (385,)
```

In [48]:
```
scaler = StandardScaler() # ewentulnie MinMaxScaler
```

In [49]:
```
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Artificial Neural Network (ANN)

In [50]:
```
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn import metrics
```

In [51]:
```
from tensorflow.keras.models import Sequential
```

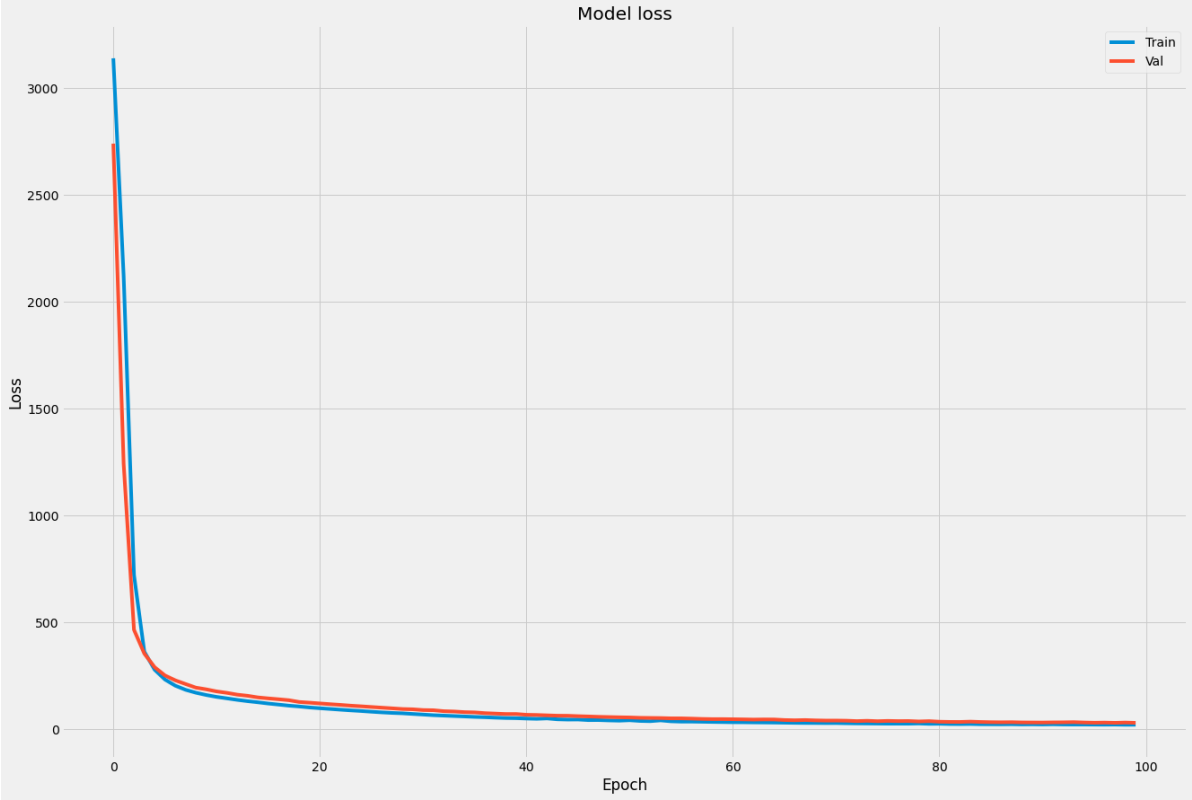
```
from tensorflow.keras.layers import Dense
```

In [52]:
```
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(64, activation='relu'),
    Dense(1, activation='elu'),
])

model.compile(optimizer='adam',
              loss='mean_squared_error',
              metrics=['mean_squared_error'])
```

In [53]:
```
hist = model.fit(X_train, y_train, epochs=100, validation_data=(X_test, y_test), v
```

In [54]:
```
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='best')
plt.show()
```



Wnioski:

- Sieć posiada dwie warstwy 64 neuronów, opytamlizowana jest funkcją: adam oraz zastosowana została funkcja aktywacji: elu.
- Sieć o dwóch warstwach 64 neuronów nauczyła się w oparciu o zbiór i jej błąd wynosi 150.
- Tempo uczenia sieci jest zadowalajace, nauka prowadzona była w 100 epokach.

Random Forest

In [55]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
```

In [56]:
```
param_grid = {
    'bootstrap': [True],
    'max_depth': [3, 5, 7],
    'max_features': [3, 4, 5],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [50, 70, 90, 140, 200]
}

rfr = RandomForestRegressor()

grid_search = GridSearchCV(estimator = rfr, param_grid = param_grid, cv = 3, n_jobs
```

In [57]:
```
grid_search.fit(X_train, y_train)
```

Out[57]:
```
GridSearchCV(cv=3, estimator=RandomForestRegressor(), n_jobs=-1,
             param_grid={'bootstrap': [True], 'max_depth': [3, 5, 7],
                         'max_features': [3, 4, 5],
                         'min_samples_leaf': [3, 4, 5],
                         'min_samples_split': [8, 10, 12],
                         'n_estimators': [50, 70, 90, 140, 200]})
```

In [58]:
```
grid_search.best_params_
```

Out[58]:
```
{'bootstrap': True,
 'max_depth': 7,
 'max_features': 5,
 'min_samples_leaf': 3,
 'min_samples_split': 8,
 'n_estimators': 200}
```

In [59]:
```
rfr = RandomForestRegressor(**grid_search.best_params_)
rfr.fit(X_train, y_train)
```

Out[59]:
```
RandomForestRegressor(max_depth=7, max_features=5, min_samples_leaf=3,
                      min_samples_split=8, n_estimators=200)
```

In [60]:
```
def evaluate(model, test_features, test_labels):
    predictions = model.predict(test_features)
    errors = abs(predictions - test_labels)
    mape = 100 * np.mean(errors / test_labels)
    accuracy = 100 - mape
    print('Wydajność modelu:')
    print('Błąd MAE: {:0.4f} stopni.'.format(np.mean(errors)))
    print('Dokładność = {:0.2f}%.'.format(accuracy))

    return accuracy

grid_accuracy = evaluate(rfr, X_test, y_test)
```

```
Wydajność modelu:
Błąd MAE: 3.7549 stopni.
Dokładność = 89.20%.
```

In [61]:
```
rfr.score(X_train, y_train)
```

Out[61]:
```
0.9617797725947305
```

In [62]:
```
importances = list(rfr.feature_importances_)
feature_importances = [(col_name, round(importance * 100, 2))
                       for col_name, importance in zip(df.columns[:-1], impor
```

```
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse = 

[print('Zmienna: {:18} ,istotność: {}%'.format(*pair)) for pair in feature_importa
```

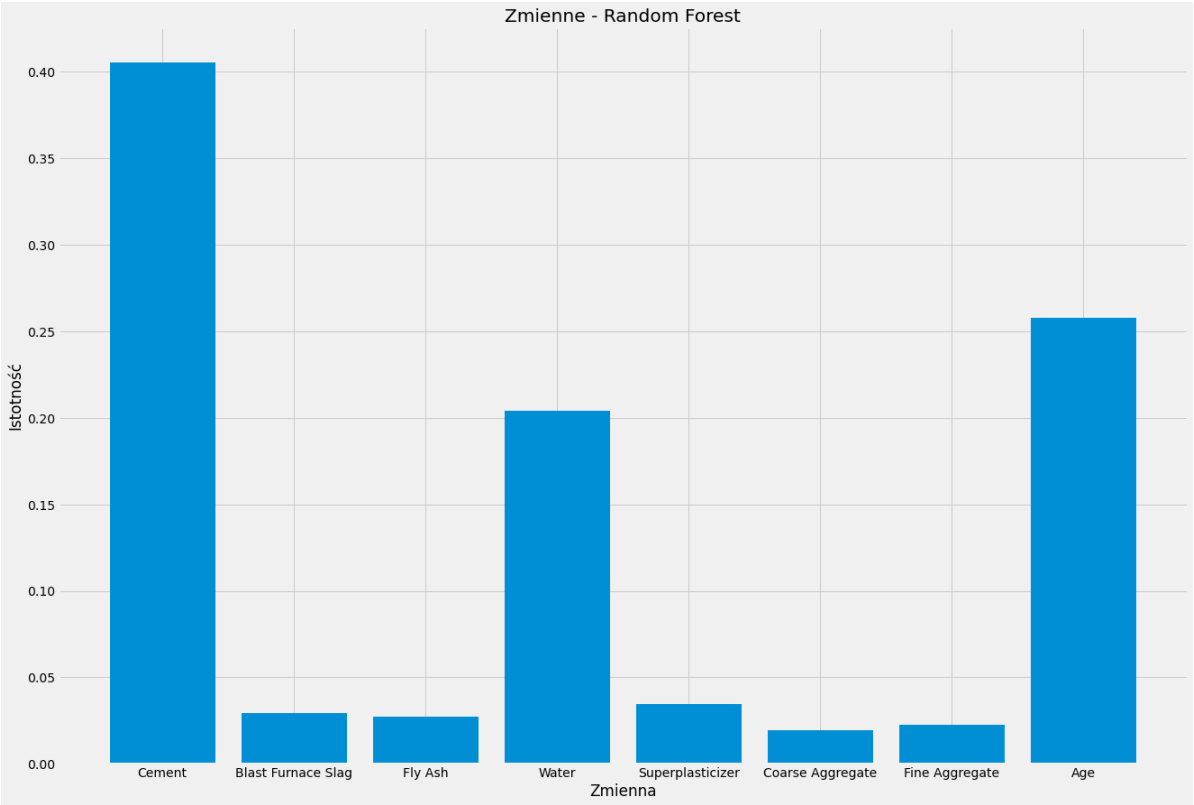
```
Zmienna: Cement             ,istotność: 40.56%
Zmienna: Age                ,istotność: 25.78%
Zmienna: Water              ,istotność: 20.4%
Zmienna: Superplasticizer   ,istotność: 3.44%
Zmienna: Blast Furnace Slag ,istotność: 2.91%
Zmienna: Fly Ash            ,istotność: 2.7%
Zmienna: Fine Aggregate     ,istotność: 2.25%
Zmienna: Coarse Aggregate   ,istotność: 1.96%
```

Wnioski (na podstawie *feature_importances_*):

- Jak można zauważyć to Cement i Age mają nawiększy wpływ na Concrete compressive strength (wytrzymałość na ściskanie), co potwierdza hipotezę nr 1.
- Jak można zauważyć to Fine Aggregate, Fly Ash i Blast Furnace Slag mają pomijalny wpływ na Concrete compressive strength, co falsyfikuje hipotezę nr 2.
- Jak można zauważyć to wysoka wartość Water obala hipotezę nr 3, natomiast znikomo mała Fine Aggregate i najwyższa Cement ją potwierdza.

In [63]:
```
x_axis_values = list(range(len(importances)))

plt.bar(x_axis_values, importances)
plt.xticks(x_axis_values, df.columns[:-1])
plt.ylabel('Istotność')
plt.xlabel('Zmienna')
plt.title('Zmienne - Random Forest')
plt.show()
```

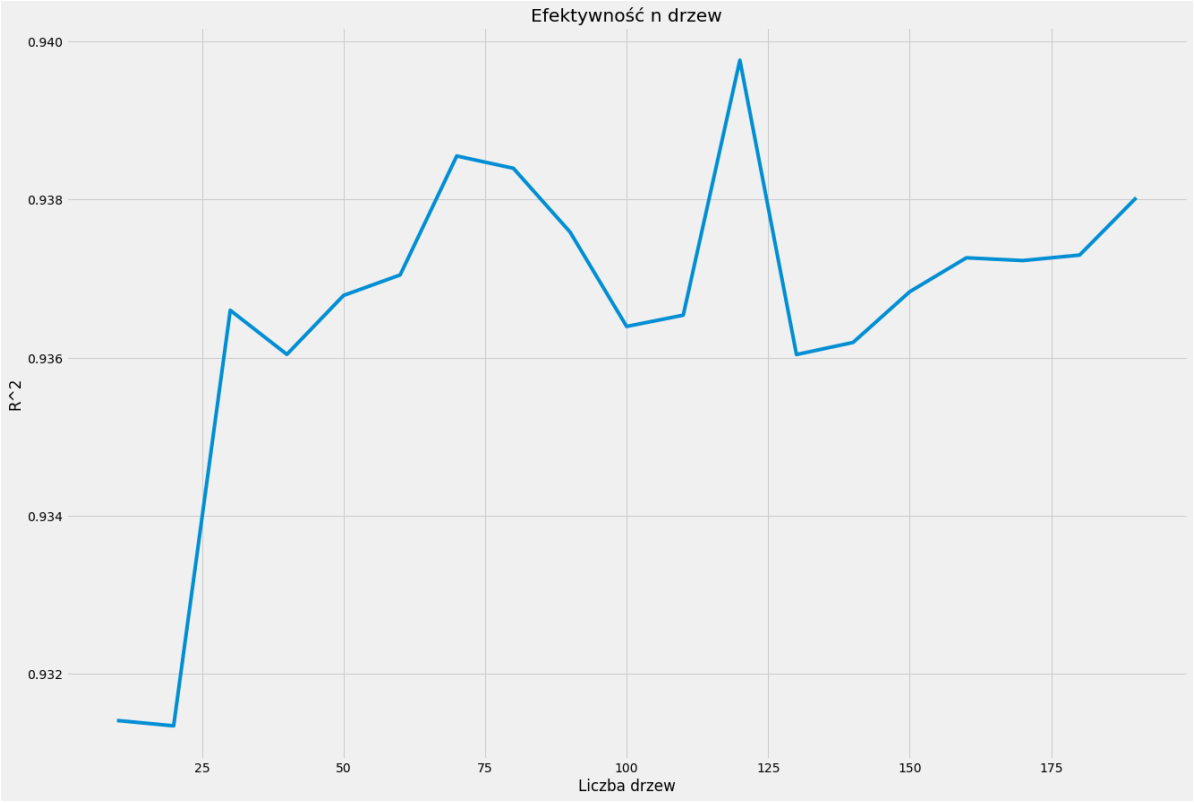


In [64]:
```
estimators = np.arange(10, 200, 10)
scores = []

for n in estimators:
```

```
    rfr.set_params(n_estimators=n)
    rfr.fit(X_train, y_train)
    scores.append(rfr.score(X_test, y_test))

plt.title("Efektywność n drzew")
plt.xlabel("Liczba drzew")
plt.ylabel("R^2")
plt.plot(estimators, scores)
plt.show()
```



In [65]:
```
vector1 = scaler.transform(np.array([[500,72.04,55.54,182.07,30,974.38,772.69,365]]
vector2 = scaler.transform(np.array([[110,80,80,182.07,6.03,974.38,880,45]]))
vector3 = scaler.transform(np.array([[500,72.04,55.54,121.75,6.03,974.38,594,100]]]

print(rfr.predict(vector1))
print(rfr.predict(vector2))
print(rfr.predict(vector3))
```

```
[61.78400557]
[36.35682921]
[63.80947458]
```

Powyższe wyniki potwierdzają hipotezę 1 i 3 oraz falsyfikują hipotezę 2

Wnioski:

- RandomForestRegressor pozwolił na zbudowanie modelu dającego wysoką dokładność predycji wyników dla problemu predykcji wytrzymałości na ściskanie betonu.
- GridSearchCV jest bardzo użytecznym narzędziem służącym przeprowadzaniu automatycznego poszukawania modelu o najlepszych hiperparametrach.

Podsumowanie:

- Hipotezę nr 1 została potwierdzona.
- Hipotezę nr 2 została sfalsyfikowana.
- Hipotezę nr 3 została potwierdzona.