```
def fit_regression(X_ucz, X_test, y_ucz, y_test):
    r = sklearn.linear_model.LinearRegression()
    r.fit(X_ucz, y_ucz)
    y_ucz_pred  = r.predict(X_ucz)
    y_test_pred = r.predict(X_test)
    mse = sklearn.metrics.mean_squared_error
    mae = sklearn.metrics.mean_absolute_error
    return {
        "r_score": r.score(X_ucz, y_ucz),
        "MSE_u":   mse(y_ucz,  y_ucz_pred),
        "MSE_t":   mse(y_test, y_test_pred),
        "MAE_u":   mae(y_ucz,  y_ucz_pred),
        "MAE_t":   mae(y_test, y_test_pred)
    }
```

Wyznaczmy model regresji na podstawie próby uczącej i oceńmy jego jakość:

```
>>> params = ["zm. liniowe"]      # parametry
>>> res = [fit_regression(X_ucz, X_test, y_ucz, y_test)]
>>> pd.DataFrame(res, index=params)
               MAE_t     MAE_u    MSE_t     MSE_u   r_score
zm. liniowe  0.30647  0.282308  0.54539  0.138808  0.906772
```

W dalszej części rozdziału przeanalizujemy przykładowe modyfikacje powyższego modelu oraz zbadamy ich wpływ na poprawność predykcji.

14.2.3. Model wielomianowy

Klasa `PolynomialFeatures` z modułu sklearn.preprocessing dokonuje transformacji atrybutów, tak by nowa macierz obserwacji zawierała iloczyny poszczególnych zmiennych o stopniu nie większym niż wartość zadana przez użytkownika (argument `degree`). Na przykład, dla `degree` $= 2$ oraz punktów typu $[x_1, x_2, x_3]$ otrzymamy ich przekształcone wersje postaci $[x_1, x_2, x_3, x_1^2, x_1x_2, x_1x_3, x_2^2, x_2x_3, x_3^2]$.

```
>>> import sklearn.preprocessing
>>> p2test = sklearn.preprocessing.PolynomialFeatures(degree=2,
include_bias=False)
>>> p2test.fit_transform(np.array([[2, 3, 5], [1, 2, 3]]))
array([[  2.,   3.,   5.,   4.,   6.,  10.,   9.,  15.,  25.],
       [  1.,   2.,   3.,   1.,   2.,   3.,   4.,   6.,   9.]])
```

Pole `powers_` udostępnia macierz reprezentującą potęgi, do których zostają podniesione poszczególne zmienne.

```
>>> p2test.powers_.T      # uwaga: transpozycja
array([[1, 0, 0, 2, 1, 1, 0, 0, 0],
       [0, 1, 0, 0, 1, 0, 2, 1, 0],
       [0, 0, 1, 0, 0, 1, 0, 1, 2]])
```

Innymi słowy, jeśli i-ta kolumna powyższej macierzy jest postaci $[a_1, a_2, a_3]$, oznacza to, że i-ta zmienna w nowym zbiorze zostaje utworzona jako iloczyn potęg $x_1^{a_1} x_2^{a_2} x_3^{a_3}$.

Dopasujmy model wielomianowy dla zbioru win i `degree` $= 2$:

$$f'_{\boldsymbol{\beta}}(\mathbf{x}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_{10} x_{10} + \beta_{11} x_1^2 + \beta_{12} x_1 x_2 + \beta_{13} x_1 x_3 + \cdots + \beta_{65} x_{10}^2.$$

Możemy tego dokonać przy użyciu poznanej implementacji metody najmniejszych kwadratów i odpowiednio przekształconego zbioru zmiennych objaśniających:

```
>>> p2 = sklearn.preprocessing.PolynomialFeatures(degree=2,
include_bias=False)
>>> X2_ucz = p2.fit_transform(X_ucz)
>>> X2_test = p2.fit_transform(X_test)
>>> params.append("zm. wielom.")
>>> res.append(fit_regression(X2_ucz, X2_test, y_ucz, y_test))
>>> pd.DataFrame(res, index=params)
                MAE_t     MAE_u    MSE_t     MSE_u   r_score
zm. liniowe  0.306470  0.282308  0.54539  0.138808  0.906772
zm. wielom.  0.267944  0.257613  0.15542  0.113192  0.923976
```

Dzięki transformacji wielomianowej uzyskaliśmy mniejsze błędy dopasowania i predykcji. Niestety liczba współczynników modelu znacząco wzrosła:

```
>>> X2_ucz.shape[1] + 1
66
```

14.2.4. Wybór zmiennych do modelu

Okazuje się, że dobranie odpowiednich zmiennych, na podstawie których dopasowujemy model, jest bardzo istotne. Zauważmy, że w przypadku gdy zmiennych objaśniających jest dużo, tracimy przejrzystość jego interpretacji oraz znacząco zwiększamy ryzyko przeuczenia się. Z drugiej strony „mały" model może nie radzić sobie dobrze z opisywaniem odpowiadającej mu rzeczywistości. Powinniśmy więc szukać punktu równowagi między złożonością modelu a jego jakością.

Wyboru zmiennych do modelu możemy dokonać, korzystając np. z kryterium Schwarza (BIC, ang. *Bayesian information criterion*). Polega ono na wyborze takiego modelu regresji liniowej, który minimalizuje:

$$\mathrm{BIC}(\mathrm{MSE}_p, p, n) = n \log(\mathrm{MSE}_p) + p \log(n),$$

gdzie MSE_p oznacza błąd średniokwadratowy dla konkretnego modelu zbudowanego na podstawie $p \leqslant d$ zmiennych. Odnotujmy, że wartość $p \log(n)$ stanowi swego rodzaju karę za złożoność modelu, tzn. zbyt dużą liczbę predyktorów.

```
def BIC(mse, p, n):
    return n*np.log(mse) + p*np.log(n)
```