Politechnika Krakowska
im. Tadeusza Kościuszki

Porównanie poprawności wybranych modeli do oceny czasu potrzebnego na transkodowanie wideo

Eryk Dziewoński, Szymon Mazurek

149115, 140324

Sztuczna Inteligencja

Projekt 2025/2026

Wydział Inżynierii Elektrycznej i Komputerowej
Politechnika Krakowska

1. Wstęp i cel projektu

W dobie dynamicznego rozwoju usług cyfrowych i platform streamingowych, takich jak YouTube, Netflix czy serwisy VOD, przetwarzanie materiałów wideo stanowi spore wyzwanie infrastrukturalnych. Każdy materiał wideo przesyłany na serwery musi zostać poddany procesowi transkodowania - konwersji do wielu formatów, rozdzielczości i kodeków, aby zapewnić kompatybilność z szeroką gamą urządzeń końcowych.

Proces ten jest wysoce zasobochłonny i kosztowny pod względem mocy obliczeniowej. W środowisku chmurowym, gdzie koszty naliczane są za czas użycia procesora, umiejętność precyzyjnego oszacowania czasu potrzebnego na przetworzenie pliku ma kluczowe znaczenie biznesowe. Pozwala to na:

- Efektywniejsze planowanie kolejkowania zadań
- Dynamiczne skalowanie zasobów serwerowych
- Znaczącą redukcję kosztów operacyjnych oraz zużycia energii

1.1 Cel projektu

Głównym celem naszego projektu jest zbadanie i porównanie modeli uczenia maszynowego używając je do predykcji czasu transkodowania wideo. Zmienną objaśnianą w projekcie jest utime (czas procesora w przestrzeni użytkownika), natomiast parametrami są dane techniczne materiału wideo m.in.: bitrate, rozdzielczość, liczba klatek na sekundę, rodzaj kodeka wejściowego i wyjściowego oraz rozmiar pliku.

1.2 Narzędzia

Badania oparto na rzeczywistym zbiorze danych **“**Online Video Characteristics and Transcoding Time Dataset”[1], udostępnionym przez UCI Machine Learning Repository. Zbiór ten zawiera szczegółowe pomiary parametrów transkodowania dla kilkudziesięciu tysięcy plików wideo.

Projekt został zrealizowany z wykorzystaniem następujących technologii:

- Język programowania: Python 3.13
- Środowisko pracy: Jupyter Notebook
- Biblioteki analityczne: `scikit-learn` (do budowy i ewaluacji modeli), `pandas` (do manipulacji danymi), `matplotlib` i `seaborn` (do wizualizacji danych)

W ramach projektu porównano skuteczność modelu bazowego o małej mocy opisowej (Regresja Liniowa) z modelem zaawansowanym o dużej mocy opisowej (Las Losowy), poddając je rygorystycznej ocenie statystycznej.

Kod i pliki projektu są dostępne na github’ie pod adresem https://github.com/gdziewon/transcoding.

1 Tewodros Deneke, UCI Machine Learning Repository, 2014. Dostępny online: https://archive.ics.uci.edu/dataset/305/online+video+characteristics+and+transcoding+time+dataset [Dostęp: 29.12.2025].

2. Wprowadzenie teoretyczne

Celem projektu jest rozwiązanie problemu, polegającego na znalezieniu funkcji $f(x)$, która z jak najmniejszym błędem odwzorowuje wektor cech wejściowych x (w tym wypadku parametry wideo) na ciągłą wartość wyjściową y (czas transkodowania). W projekcie przeanalizowano dwa podejścia o fundamentalnie różnej strukturze matematycznej: parametryczny model liniowy i nieparametryczny model zespołowy.

2.1. Regresja Liniowa (Linear Regression)

Regresja liniowa jest jednym z najprostszych i fundamentalnych algorytmów uczenia nadzorowanego. Zakłada ona istnienie liniowej zależności pomiędzy zmiennymi niezależnymi (cechami) a zmienną zależną (celem)[2]. Dla zbioru danych składającego się z n cech $(x1, x2, \ldots, xn)$, model predykcyjny $\hat{y}$ (estymator) wyraża się wzorem:

$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

Zapisując to w notacji wektorowej, gdzie $\mathbf{w}$ to wektor wag, a $\mathbf{x}$ to wektor cech:

$$\hat{y} = \mathbf{w}^T \mathbf{x}$$

Rzeczywista wartość y jest sumą predykcji modelu oraz błędu losowego ϵ (szumu), którego nie da się przewidzieć:

$$y = \mathbf{w}^T \mathbf{x} + \epsilon$$

Funkcja Kosztu i Minimalizacja Błędu

Najczęściej stosowaną funkcją oceny jakości dopasowania jest Błąd Średniokwadratowy (MSE)[3]:

$$MSE(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^{m} (y^{(i)} - \hat{y}^{(i)})^2$$

gdzie:

- m - liczba próbek w zbiorze treningowym,
- $y^{(i)}$ - rzeczywista wartość dla i-tej próbki,
- $\hat{y}^{(i)}$ - wartość przewidziana przez model.

Estymacja parametrów $\mathbf{w}$ odbywa się zazwyczaj przy użyciu Metody Najmniejszych Kwadratów, która dąży do analitycznego lub numerycznego zminimalizowania funkcji kosztu MSE.[4]

2.2. Las Losowy

Las Losowy to zaawansowany algorytm należący do rodziny metod **uczenia zespołowego**. Idea stojąca za tym podejściem zakłada, że grupa słabych estymatorów (w tym wypadku pojedynczych drzew) może wspólnie stworzyć silny estymator o znacznie mniejszej wariancji i wyższej odporności na przeuczenie.[5]

Budowa Drzewa Decyzyjnego w Regresji

2 T. Hastie, R. Tibshirani, J. Friedman, *The Elements of Statistical Learning*, wyd. 2, Springer 2009, s. 30-32.
3 *Ibidem*, s. 43.
4 Ibidem, s. 44-46.
5 A. Géron, *Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow. Droga do sztucznej inteligencji*, Wydawnictwo Helion, Gliwice 2018, s. 199-201.

Podstawowym budulcem lasu jest drzewo regresyjne. Drzewo dzieli przestrzeń cech na prostokątne obszary, podejmując w każdym węźle decyzję na podstawie wartości konkretnej cechy. Dla problemów regresji, kryterium podziału węzła jest redukcja wariancji.

Węzeł jest dzielony w punkcie s cechy j, który minimalizuje sumę błędów kwadratowych w nowo powstałych podzbiorach (lewym R_1 i prawym R_2):

$$\min_{j,s} \left[\sum_{x_i \in R_1(j,s)} (y_i - \bar{y}_{R_1})^2 + \sum_{x_i \in R_2(j,s)} (y_i - \bar{y}_{R_2})^2 \right]^{6}$$

Gdzie $\bar{y}_{R_1}$ i $\bar{y}_{R_2}$ to średnie wartości zmiennej celu (czasu transkodowania) w utworzonych liściach.

Bagging[7]

Kluczowym mechanizmem Lasu Losowego jest Bagging. Proces ten składa się z dwóch etapów:

1. **Bootstrap:** Dla każdego z K drzew w lesie, tworzony jest nowy zbiór treningowy poprzez losowanie ze zwracaniem z oryginalnego zbioru danych.
2. **Agregacja:** Ostateczna predykcja modelu Lasu Losowego jest średnią arytmetyczną predykcji wszystkich K drzew:

$$\hat{y}_{RF} = \frac{1}{K} \sum_{k=1}^{K} f_k(x)$$

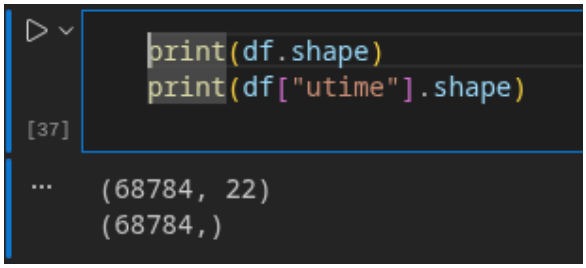
Z matematycznego punktu widzenia, uśrednianie wyników wielu nieskorelowanych modeli znacząco redukuje wariancję błędu predykcji, co sprawia, że Random Forest doskonale radzi sobie z danymi o nieliniowej charakterystyce i jest odporny na wartości odstające.

3. Analiza eksploracyjna danych (EDA)

Przed przystąpieniem do modelowania przeprowadzono szczegółową analizę eksploracyjną której celem było zrozumienie struktury zbioru danych i wykrycie ewentualnych anomalii.

3.1 Charakterystyka zbioru danych

Analizowany zbiór danych (“Online Video Characteristics and Transcoding Time Dataset”) składa się z 68 784 obserwacji oraz 22 zmiennych (Rysunek 3.1). Każdy rekord reprezentuje pojedyncze zadanie transkodowania materiału wideo.



Rysunek 3.1 Wymiary danych

6 T. Hastie, R. Tibshirani, J. Friedman, *The Elements of Statistical Learning*, wyd. 2, Springer 2009, s. 326.
7 *Ibidem*, s. 301

Zmienne w zbiorze można podzielić na dwie grupy:

1. **Cechy wejściowe:** opisujące oryginalny plik wideo przed konwersją (m.in. `duration`, `width`, `height`, `bitrate`, `framerate`, `codec`).
2. **Cechy wyjściowe:** opisujące parametry docelowe po transkodowaniu (m.in. `o_width`, `o_height`, `o_bitrate`, `o_codec`).

Zmienną celu (target), którą model ma przewidywać, jest **`utime`** - czas procesora w przestrzeni użytkownika, wyrażony w sekundach. Rysunek 3.2 pokazuje szczegółowe informacje dotyczące danych.

```
[36]  df.info()

...  <class 'pandas.core.frame.DataFrame'>
     RangeIndex: 68784 entries, 0 to 68783
     Data columns (total 22 columns):
      #   Column       Non-Null Count  Dtype
     ---  ------       --------------  -----
      0   id           68784 non-null  object
      1   duration     68784 non-null  float64
      2   codec        68784 non-null  object
      3   width        68784 non-null  int64
      4   height       68784 non-null  int64
      5   bitrate      68784 non-null  int64
      6   framerate    68784 non-null  float64
      7   i            68784 non-null  int64
      8   p            68784 non-null  int64
      9   b            68784 non-null  int64
      10  frames       68784 non-null  int64
      11  i_size       68784 non-null  int64
      12  p_size       68784 non-null  int64
      13  b_size       68784 non-null  int64
      14  size         68784 non-null  int64
      15  o_codec      68784 non-null  object
      16  o_bitrate    68784 non-null  int64
      17  o_framerate  68784 non-null  float64
      18  o_width      68784 non-null  int64
      19  o_height     68784 non-null  int64
      20  umem         68784 non-null  int64
      21  utime        68784 non-null  float64
     dtypes: float64(4), int64(15), object(3)
     memory usage: 11.5+ MB
```

Rysunek 3.2 Informacje o danych

Wstępna weryfikacja jakości danych wykazała, że zbiór jest kompletny - nie odnotowano żadnych brakujących wartości (Rysunek 3.3), co eliminuje konieczność usuwania pojedynczych danych na etapie preprocessingu. Zidentyfikowano jednak kolumny nieinforamtywne, takie jak `id` (losowy identyfikator) oraz `b_size` (zawierająca wyłącznie zera), które zakwalifikowano do usunięcia.

```
[9]  missing_values = df.isnull().sum().sum()
     print(f"Missing values count in dataset: {missing_values}")
     ✓ 0.0s

...  Missing values count in dataset: 0
```

Rysunek 3.3 Brakujące wartości

3.2 Statystyki opisowe

W celu zbadania rozkładu zmiennych numerycznych wyznaczono podstawowe statystyki opisowe: średnią, odchylenie standardowe, medianę oraz wartości skrajne (Rysunek 3.4).

Analiza statystyczna wykazała istotną cechę zmiennej celu utime:

- Średni czas transkodowania wynosi około 9.9 sekundy, ale maksymalna wartość sięga ponad 224 sekund.
- Występuje znaczna różnica między średnią a medianą, co wraz z wysoką wartością parametru skośności wskazuje na silnie prawoskośny rozkład danych.[8]

Oznacza to, że większość zadań transkodowania trwa krótko, ale istnieje grupa zadań bardzo złożonych obliczeniowo (“długi ogon”). Fakt ten uwzględniono w dalszej analizie wizualnej, stosując skalę logarytmiczną na wykresach, aby poprawić ich czytelność.

```
stats = df.describe().T
stats['skewness'] = df.select_dtypes(include=np.number).skew()
stats
```

[58] ✓ 0.0s

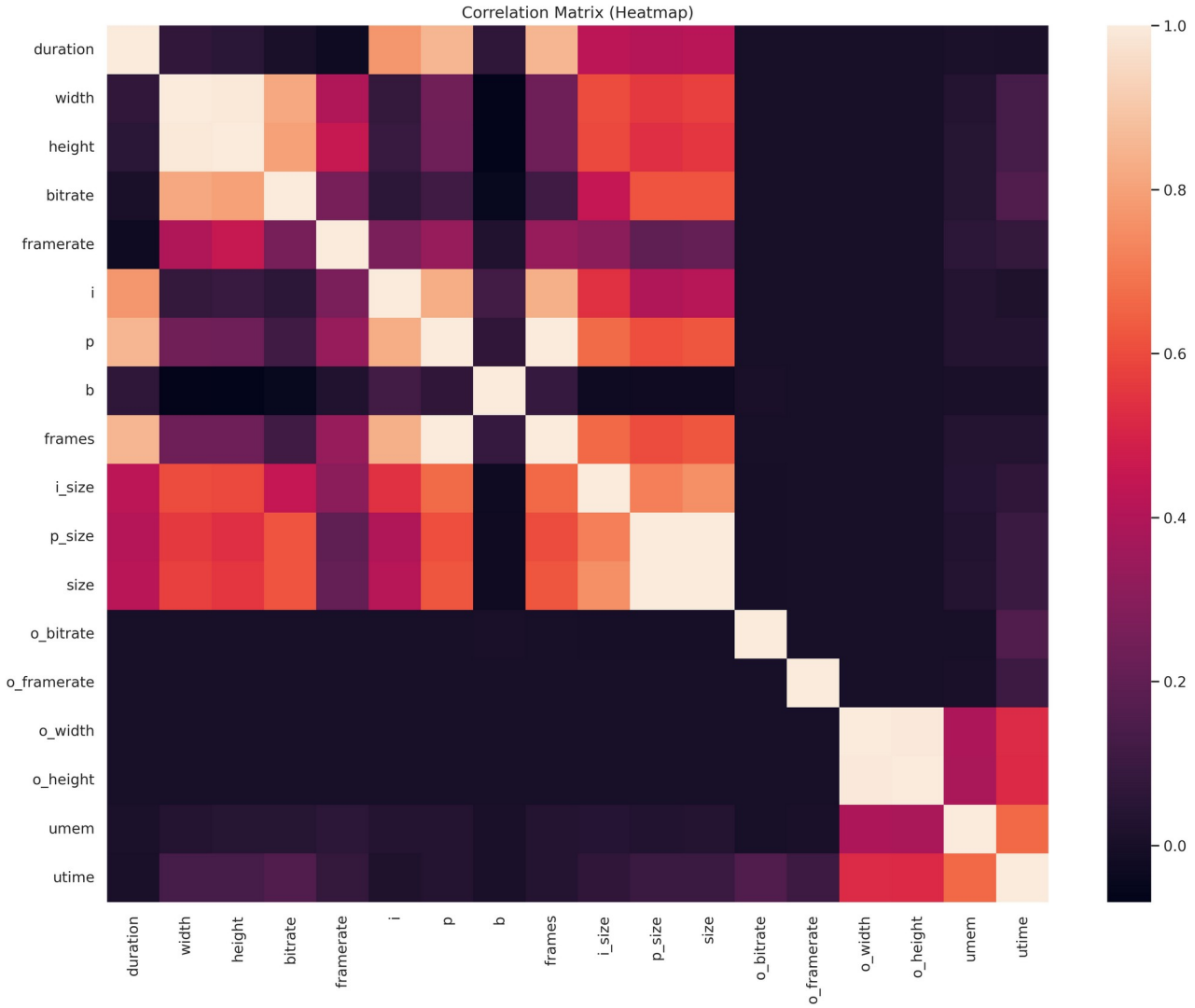
	count	mean	std	min	25%	50%	75%	max	skewness
duration	68784.0	2.864139e+02	2.872576e+02	31.080000	106.765	2.391417e+02	3.793200e+02	2.584409e+04	12.679999
width	68784.0	6.249342e+02	4.631691e+02	176.000000	320.000	4.800000e+02	6.400000e+02	1.920000e+03	1.336629
height	68784.0	4.125722e+02	2.406155e+02	144.000000	240.000	3.600000e+02	4.800000e+02	1.080000e+03	1.094027
bitrate	68784.0	6.937015e+05	1.095628e+06	8384.000000	134334.000	2.911500e+05	6.529670e+05	7.628466e+06	3.203007
framerate	68784.0	2.324132e+01	7.224848e+00	5.705752	15.000	2.502174e+01	2.900000e+01	4.800000e+01	-0.839156
i	68784.0	1.008683e+02	8.476479e+01	7.000000	39.000	8.000000e+01	1.380000e+02	5.170000e+03	5.647818
p	68784.0	6.531692e+03	6.075872e+03	175.000000	2374.000	5.515000e+03	9.155000e+03	3.049590e+05	3.958073
b	68784.0	9.147854e+00	9.251618e+01	0.000000	0.000	0.000000e+00	0.000000e+00	9.407000e+03	28.048834
frames	68784.0	6.641708e+03	6.153342e+03	192.000000	2417.000	5.628000e+03	9.232000e+03	3.101290e+05	3.966428
i_size	68784.0	2.838987e+06	4.325137e+06	11648.000000	393395.000	9.458650e+05	3.392479e+06	9.082855e+07	2.833498
p_size	68784.0	2.218057e+07	5.097306e+07	33845.000000	1851539.000	6.166260e+06	1.515506e+07	7.689970e+08	4.404563
b_size	68784.0	0.000000e+00	0.000000e+00	0.000000	0.000	0.000000e+00	0.000000e+00	0.000000e+00	0.000000
size	68784.0	2.502294e+07	5.414402e+07	191879.000000	2258222.000	7.881069e+06	1.977335e+07	8.067111e+08	4.344798
o_bitrate	68784.0	1.395036e+06	1.749352e+06	56000.000000	109000.000	5.390000e+05	3.000000e+06	5.000000e+06	1.177296
o_framerate	68784.0	2.119086e+01	6.668703e+00	12.000000	15.000	2.400000e+01	2.500000e+01	2.997000e+01	-0.175472
o_width	68784.0	8.023364e+02	6.099598e+02	176.000000	320.000	4.800000e+02	1.280000e+03	1.920000e+03	0.814070
o_height	68784.0	5.038255e+02	3.159704e+02	144.000000	240.000	3.600000e+02	7.200000e+02	1.080000e+03	0.704238
umem	68784.0	2.282247e+05	9.743088e+04	22508.000000	216820.000	2.194800e+05	2.196560e+05	7.118240e+05	3.210104
utime	68784.0	9.996355e+00	1.610743e+01	0.184000	2.096	4.408000e+00	1.043300e+01	2.245740e+02	4.245502

Rysunek 3.4 Statystyki opisowe zmiennych numerycznych

8 P. Bruce, A. Bruce, P. Gedeck, *Statystyka praktyczna w data science. 50 kluczowych zagadnień w językach R i Python*, wyd 2, Helion 2021, s. 32.

3.3 Analiza korelacji

W celu sprawdzenia liniowych zależności między cechami a czasem przetwarzania, wyznaczono macierz korelacji Pearsona[9]. Współczynnik korelacji r przyjmuje wartości z przedziału [−1,1], gdzie 1 oznacza silną korelację dodatnią (Rysunek 3.5).



Rysunek 3.5 Macierz korelacji Pearsona dla zmiennych numerycznych

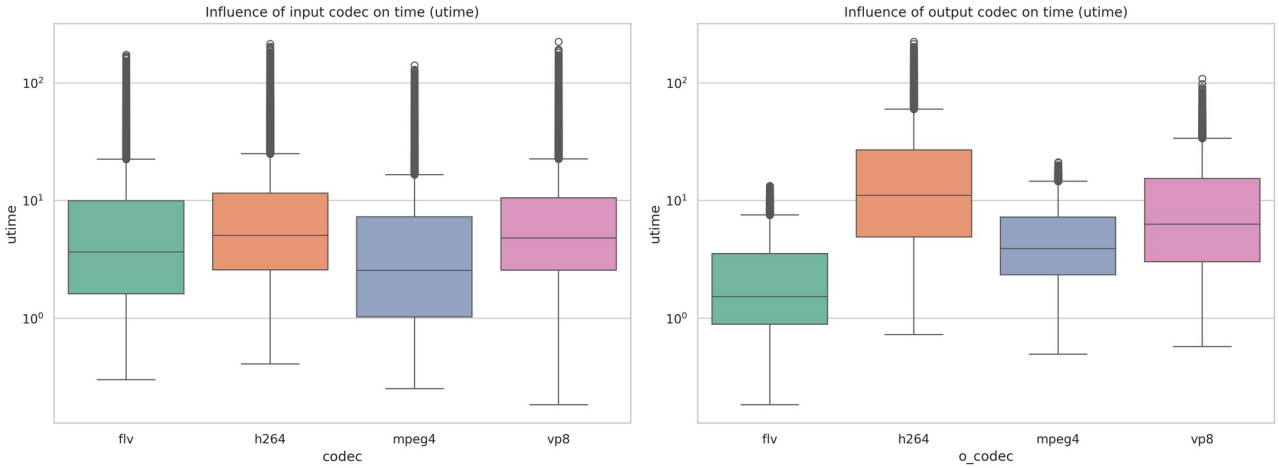
Wnioski z analizy korelacji:

1. **Parametry wyjściowe:** Zaobserwowano silną korelację dodatnią między czasem `utime` a parametrami docelowymi wideo: szerokością wyjściową `o_width` oraz wysokością wyjściową `o_height`. Jest to zgodne z intuicją - renderowanie obrazu o wyższej rozdzielczości wymaga przetworzenia większej liczby pikseli. Bardzo silną korelację wykazuje również zmienna `umem` (pamięć przydzielona do procesu transkodowania).
2. **Parametry wejściowe:** Parametry oryginalnego pliku (np. wejściowy bitrate czy framerate) wykazują znacznie słabszą korelację z czasem przetwarzania niż parametry docelowe. Sugeruje to, że czasochłonność procesu zależy bardziej od tego, jaki format chcemy uzyskać, niż jaki otrzymaliśmy.

9 P. Bruce et al., *op. cit.*, s. 44.

3.4 Analiza wpływ zmiennych kategorycznych (Kodeki)

Ostatnim elementem analizy było zbadanie wpływu zastosowanego algorytmu kompresji (kodeka) na czas obliczeń. Porównano cztery główne standardy: mpeg4, h264, vp8 oraz flv. Ze względu na skośny rozkład zmiennej celu, do wizualizacji wykorzystano wykresy pudełkowe (boxplots) ze skalą logarytmiczną na osi Y (Rysunek 3.6).



Rysunek 3.6 Rozkład czasu transkodowania w zależności od kodeka

Wnioski:

- **H.264:** Zadania wykorzystujące kodek H.264 wykazują najwyższą medianę czasu przetwarzania. Wynika to prawdopodobnie z faktu, że H.264 jest kodekiem o wysokim stopniu kompresji, wykorzystującym zaawansowane algorytmy predykcji ruchu.
- **FLV i MPEG4:** Są to formaty starsze i prostsze obliczeniowo, co przekłada się na znacznie krótsze czasy transkodowania.

Powyższa obserwacja potwierdza konieczność uwzględnienia informacji o kodeku w modelu predykcyjnym poprzez odpowiednie zakodowanie tych zmiennych, używając “One-Hot Encoding” (Rysunek 3.7).

```
# One-Hot Encoding for category variables
df = pd.get_dummies(df, columns=["codec", "o_codec"], drop_first=False)
df = df.astype(float)

print("Columns after encoding:")
print(df.columns)
df.head()
```

[162] ✓ 0.0s

```
Columns after encoding:
Index(['duration', 'width', 'height', 'bitrate', 'framerate', 'i', 'p', 'b',
       'frames', 'i_size', 'p_size', 'size', 'o_bitrate', 'o_framerate',
       'o_width', 'o_height', 'umem', 'utime', 'codec_flv', 'codec_h264',
       'codec_mpeg4', 'codec_vp8', 'o_codec_flv', 'o_codec_h264',
       'o_codec_mpeg4', 'o_codec_vp8'],
      dtype='object')
```

Rysunek 3.7 Zakodowanie informacji o kodeku wejściowym i wyjściowym

4. Badania symulacyjne i ocena modeli

Celem tego rozdziału jest przedstawienie procesu trenowania modeli uczenia maszynowego i porównanie ich skuteczności, a następnie analiza wyników.

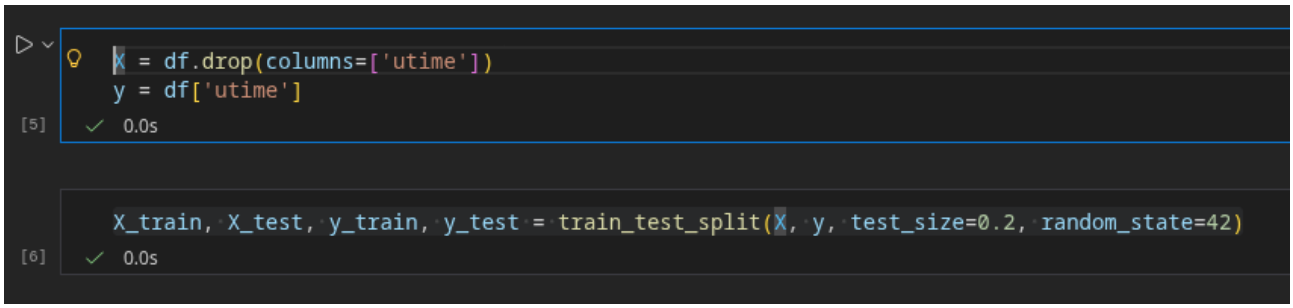
4.1 Metodyka i przygotowanie

Przed uruchomieniem algorytmów uczących, dane przetworzone w poprzednim etapie (plik `transcoding_processed.csv`) poddaliśmy ostatecznej preparacji.

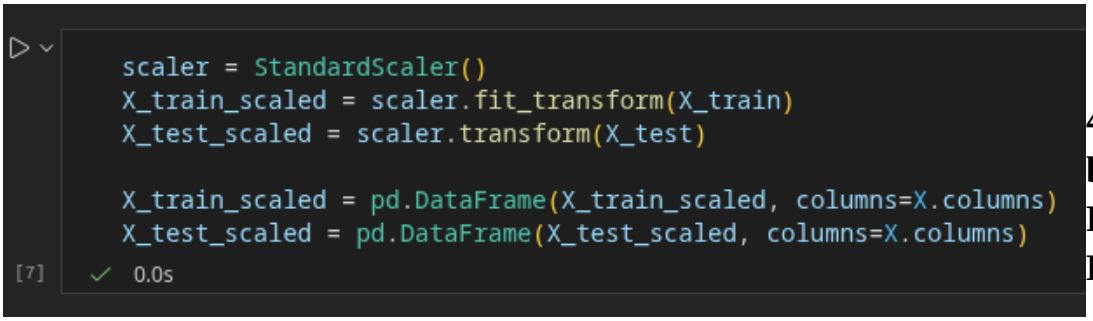
1) **Podział zbioru danych** (Rysunek 4.1)**:**
   - **Zbiór treningowy (80%):** służący do optymalizacji wag modelu - to na nim model będzie się uczył.
   - **Zbiór testowy (20%):** wykorzystywany do weryfikacji poprawności predykcji modelu. Podziału dokonano z wykorzystaniem ziarna stałego losowości (random_state), by zapewnić powtarzalność wyników.
2) **Standaryzacja cech** (Rysunek 4.2)**:** Ze względu na różnice w rzędach wielkości zmiennych, zastosowano standaryzację przy użyciu `StandardScaler'a`. [10]

$$z = \frac{x - \mu}{\sigma}$$

Skaler został dopasowany na danych treningowych, żeby zapobiec wyciekowi informacji do zbioru testowego.



Rysunek 4.1 Podział zbioru danych



Rysunek 4.2 Standaryzacja cech

4.2. Model bazowy: Regresja Liniowa

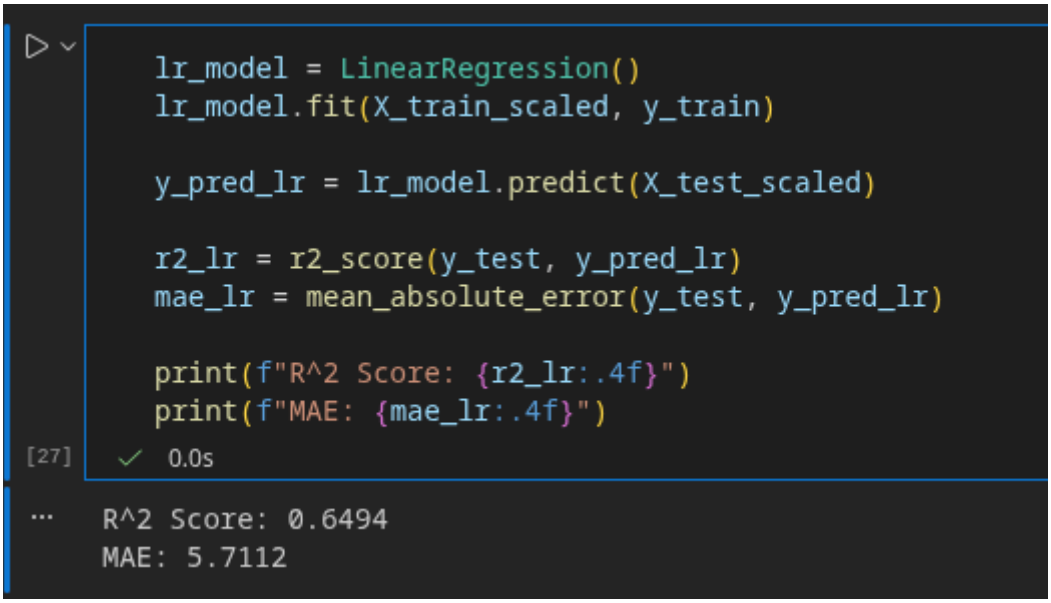
W pierwszej fazie wytrenowano prosty model Regresji Liniowej. Jego zadaniem było ustanowienie punktu odniesienia (ang. baseline) dla bardziej złożonych algorytmów (Rysunek 4.3).

10 T. Hastie et al., *op. cit.*, s. 50

Wyniki:

- **Współczynnik R^2:** 0.6494
- **Błąd średni bezwzględny (MAE):** 5.7112 s

Wynik R^2 na poziomie ok. 65% wskazuje, że model liniowy potrafi wyjaśnić znaczną część wariancji, ale relacja między parametrami wideo a czasem transkodowania nie jest w pełni liniowa.



Rysunek 4.3 Model: Regresja Liniowa

4.3. Model zaawansowany: Las Losowy

Następnie wytrenowano model Lasu Losowego składający się ze 100 drzew decyzyjnych (Rysunek 4.4).

Wyniki:

- **Współczynnik R^2:** 0.9898
- **Błąd średni bezwzględny (MAE):** 0.5162 s

Osiągnięcie niemal idealnego współczynnika dopasowania ($R^2 \approx 1.0$) świadczy o tym, że model wspaniale uchwycił nieliniowe zależności między parametrami a zmienną objaśnianą.

```
# 100 decision trees
rf_model = RandomForestRegressor(n_estimators=100, random_state=42, n_jobs=-1)

start_time = time.time()
rf_model.fit(X_train_scaled, y_train)
end_time = time.time()

y_pred_rf = rf_model.predict(X_test_scaled)

mae_rf = mean_absolute_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

print(f"R^2 Score: {r2_rf:.4f}")
print(f"MAE: {mae_rf:.4f}")

print(f"Training time: {end_time - start_time:.2f} s")
```

[45] ✓ 1.6s

```
R^2 Score: 0.9898
MAE: 0.5162
Training time: 1.57 s
```

Rysunek 4.4 Model: Las Losowy

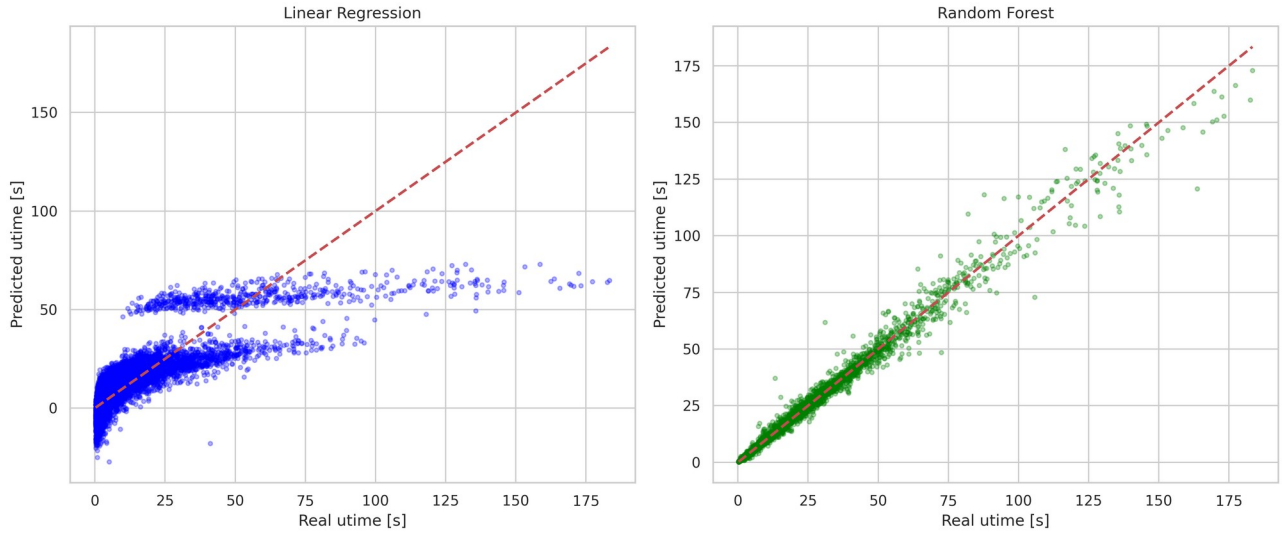
4.4. Porównanie modeli

Zestawiono wyniki modelu bazowego (Regresja Liniowa) z modelem zaawansowanym (Las Losowy).

Zysk wydajności:

- **Zmiana R^2:** +0.3404
- **Zmiana MAE:** -5.195 s

Zastosowanie Lasu Losowego przyniosło ogromną poprawę jakości predykcji. Na porównaniu wizualnym (Rysunek 4.5) widać wyraźnie, że predykcje Lasu Losowego niemalże perfekcyjnie układają się wzdruż lini przekątnej, podczas gdy Regresja Liniowa wykazuje duży rozrzut punktów.



Rysunek 4.5 Porównanie modeli

4.4. Walidacja krzyżowa

Aby upewnić się, że wysoki wynik Lasu Losowego nie jest dziełem przypadku, przeprowadzono 5-krotną walidację krzyżową[11] (Rysunek 4.6).

Wyniki:

- **Średni wynik R2:** 0.9872
- **Odchylenie standardowe:** 0.0012

Niskie odchylenie standardowe potwierdza stabilność modelu.

```
cv_scores = cross_val_score(rf_model, X_train_scaled, y_train, cv=5, scoring='r2', n_jobs=-1)

print(f"R^2 scored after 5 iterations: {cv_scores}")
print(f"Mean R^2: {cv_scores.mean():.4f}")
print(f"Standard deviation: {cv_scores.std():.4f}")
```

```
[48]  ✓ 6.9s

...  R^2 scored after 5 iterations: [0.98616735 0.98866219 0.98546112 0.98881451 0.98671447]
     Mean R^2: 0.9872
     Standard deviation: 0.0013
```

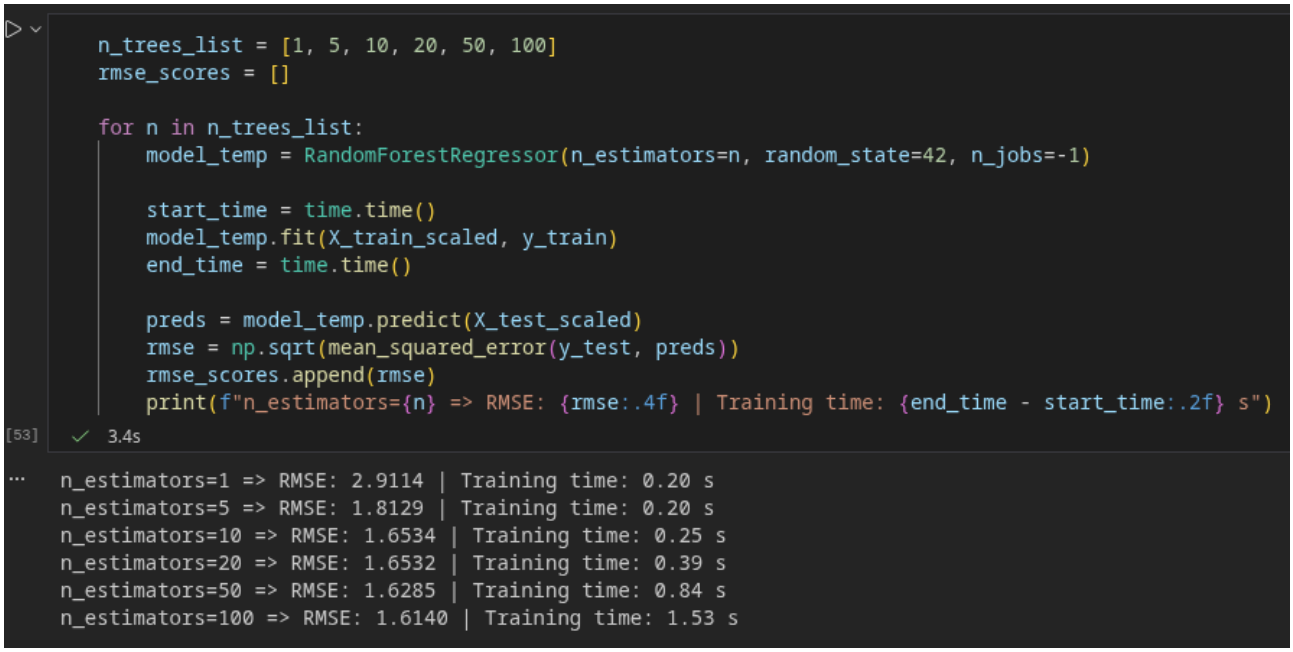
Rysunek 4.6 Walidacja krzyżowa

11 T. Hastie et al., *op. cit.*, s. 247.

4.5. Analiza wpływu liczby drzew na błąd

Zbadano, jak liczba estymatorów (“drzew w lesie”) wpływa na błąd średniokwadratowy (RMSE). Przetestowano różne wartości z zakresu od 1 do 100 drzew (Rysunek 4.7 i 4.8).

Największy spadek błędu następuje przy zwiększeniu liczby drzew z 1 do 10. Powyżej 20 drzew zysk na dokładności staje się minimalny. Wybór 100 drzew jest prawdopodobnie niepotrzebnie duży.

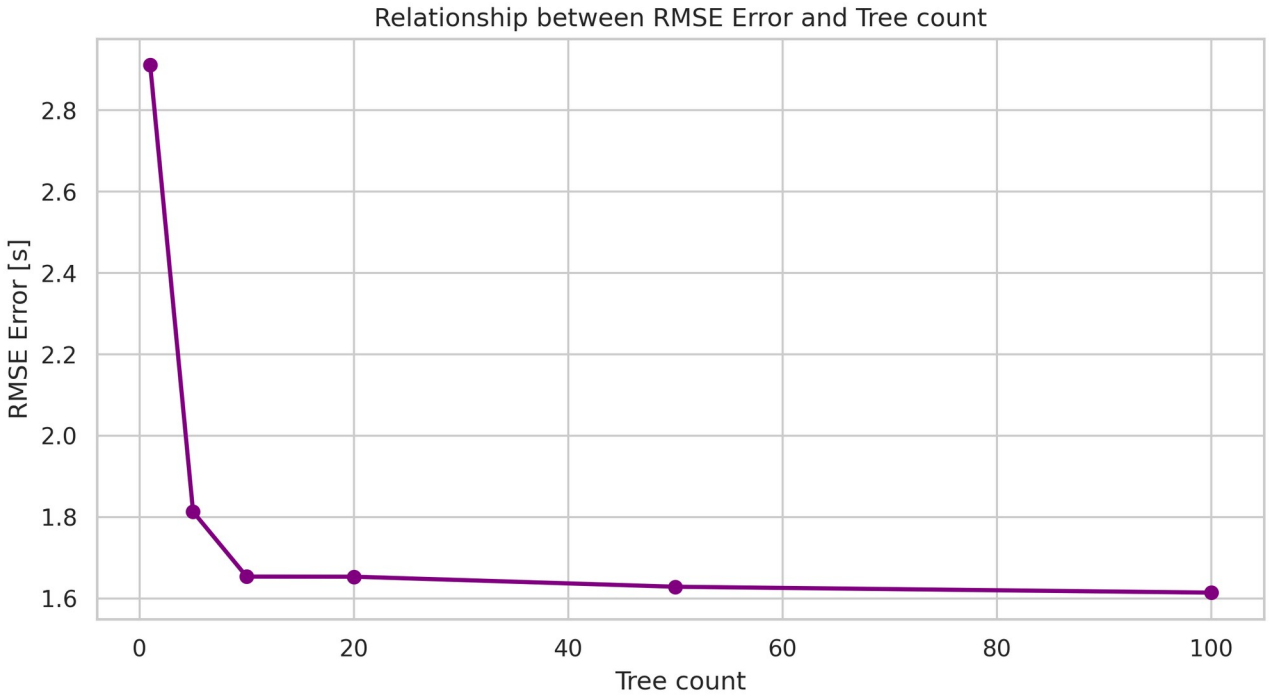


```
n_trees_list = [1, 5, 10, 20, 50, 100]
rmse_scores = []

for n in n_trees_list:
    model_temp = RandomForestRegressor(n_estimators=n, random_state=42, n_jobs=-1)

    start_time = time.time()
    model_temp.fit(X_train_scaled, y_train)
    end_time = time.time()

    preds = model_temp.predict(X_test_scaled)
    rmse = np.sqrt(mean_squared_error(y_test, preds))
    rmse_scores.append(rmse)
    print(f"n_estimators={n} => RMSE: {rmse:.4f} | Training time: {end_time - start_time:.2f} s")
```

[53] ✓ 3.4s

```
n_estimators=1 => RMSE: 2.9114 | Training time: 0.20 s
n_estimators=5 => RMSE: 1.8129 | Training time: 0.20 s
n_estimators=10 => RMSE: 1.6534 | Training time: 0.25 s
n_estimators=20 => RMSE: 1.6532 | Training time: 0.39 s
n_estimators=50 => RMSE: 1.6285 | Training time: 0.84 s
n_estimators=100 => RMSE: 1.6140 | Training time: 1.53 s
```

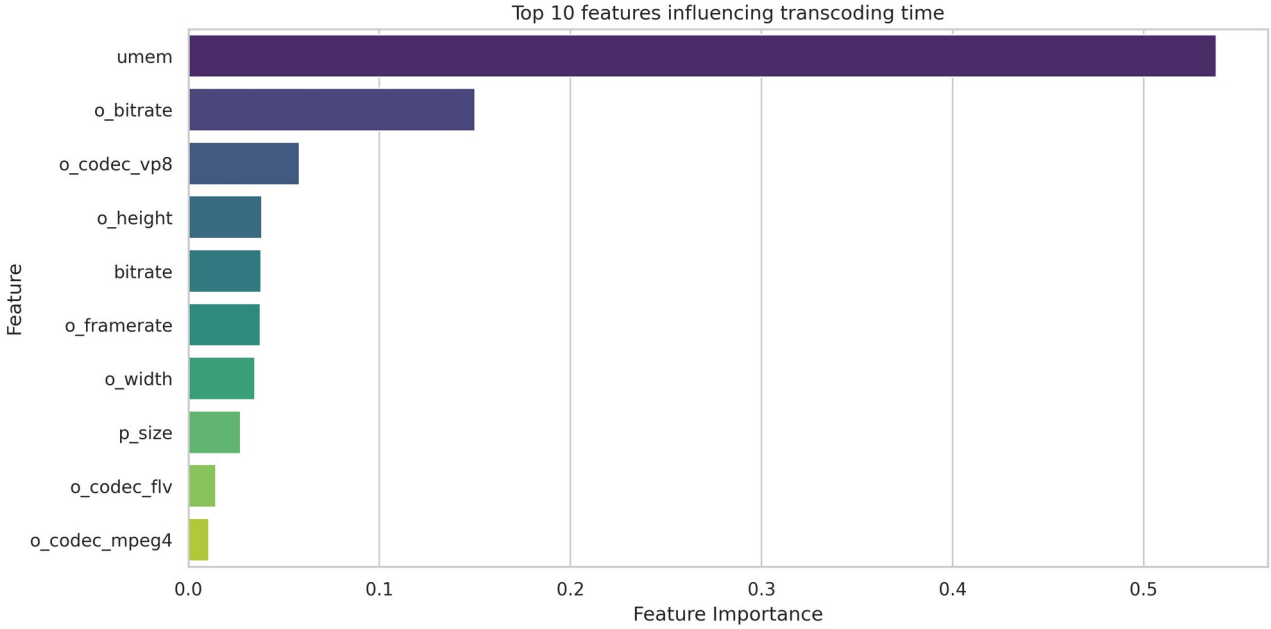
Rysunek 4.7 Wpływ liczby drzew: kod



Rysunek 4.8 Wpływ liczby drzew: wykres

4.6. Analiza ważności cech

W celu zidentyfikowania kluczowych czynników determinujących czas transkodowania, przeprowadzono analizę istotności zmiennych objaśniających na podstawie Lasu Losowego (Rysunek 4.9).



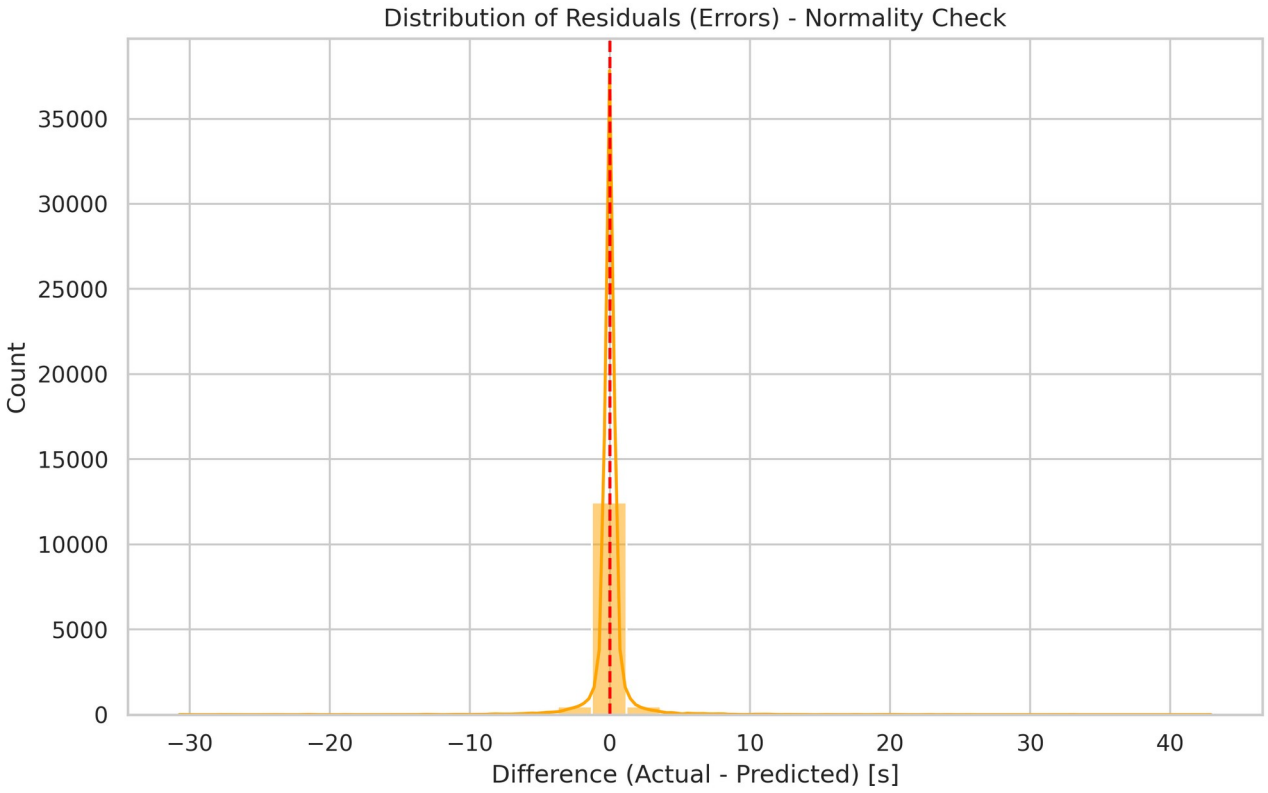
Rysunek 4.8 Ważność cech

Interpretacja:

1. **Pamięć (umem):** Jest to cecha dominująca. Ilość alokowanej pamięci jest bezpośrednio skorelowana ze złożonością procesu transkodowania.
2. **Parametry wyjściowe (o_bitrate, o_framerate):** Parametry docelowe mają kluczowe znaczenie - ma to sens - im wyższa jakość wyjściowa, tym dłuższy czas procesora.
3. **Kodeki:** Obecność kodeków również znajduje się w czołówce, co potwierdza wnioski z analizy EDA.

4.7. Analiza reszt

Ostatnim etapem weryfikacji była analiza rozkładu błędów (reszt) dla Lasu Losowego, zdefiniowanych jako różnica ***ytest - ypred***.[12]



Rysunek 4.10 Analiza reszt

Wykres (Rysunek 4.10) przypomina rozkład normalny skupiony wokół zera - średnia reszt wynosi -0.0058 s. Symetria rozkładu świadczy o braku systematycznego obciążenia modelu (ang. bias), a wąski “dzwon” pokazuje, że zdecydowana większość błędów jest bliska zeru.

5. Wnioski końcowe

Na podstawie uzyskanych wyników sformułowaliśmy następujące wnioski:

- **Rola analizy eksploracyjnej:** Wstępna analiza danych okazała się niezbędna do zrozumienia specyfiki problemu. Pozwoliła ona wykryć silnie prawoskośny rozkład czasu transkodowania, co uzasadniło konieczność stosowania odpowiednich metryk i skali

12 P. Bruce et al., *op. cit.*, s. 446.

logarytmicznej. EDA ujawniła również, że zależności między cechami nie są oczywiste, co ukierunkowało wybór modeli w dalszej części prac.

- **Różnice w skuteczności modeli:** Projekt wykazał drastyczną różnicę w jakości predykcji pomiędzy Regresją Liniową a Lasem Losowym. Wynika to z faktu, że czas transkodowania zależy od parametrów wideo w sposób nieliniowy. Model liniowy, ze względu na swoją sztywną strukturę matematyczną, nie był w stanie odwzorować tych złożonych relacji, a natomiast Las Losowy, jako model oparty na drzewach decyzyjnych, perfekcyjnie poradził sobie z uchwyceniem lokalnych zależności.
- **Ważność parametrów:** Analiza ważności parametrów obaliła intuicyjne przekonanie, że czas przetwarzania zależy głównie od pliku źródłowego. Testy wykazały, że decydujące są parametry wyjściowe oraz alokacja pamięci. Oznacza to, że koszt obliczeniowy jest definiowany bardziej przez to co chcemy uzyskać.

Bibliografia

1. Bruce P., Bruce A., Gedeck P., *Statystyka praktyczna w data science. 50 kluczowych zagadnień w językach R i Python*, Wydawnictwo Helion, Gliwice 2021.
2. Deneke T., *Online Video Characteristics and Transcoding Time Dataset*, UCI Machine Learning Repository, 2014. Dostępny online: https://archive.ics.uci.edu/dataset/305/online+video+characteristics+and+transcoding+time+dataset [Dostęp: 29.12.2025].
3. Géron A., *Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow. Droga do sztucznej inteligencji*, Wydawnictwo Helion, Gliwice 2018.
4. Hastie T., Tibshirani R., Friedman J., *The Elements of Statistical Learning*, wyd. 2, Springer, New York 2009.