VYTAUTO DIDŽIOJO UNIVERSITETAS

INFORMATIKOS FAKULTETAS

TAIKOMOSIOS INFORMATIKOS KATEDRA

Giliojo mokymo sistemų taikymai

INF4039

Vytenis Ropė – IF2000035

Individualus namų darbas

Tikrino: Prof. Aušra Saudargienė, Lek. Vytautas Kučinskas

Kaunas, 2023

1. **Duomenų rinkinio aprašymas**

   Duomenų rinkiniui pasirinkau šlifuotų deimantų kainas iki 2023 metų pagal jų ypatybes. Šį duomenų rinkinį sudaro net 53944 eilutės individualių deimantų specifikacijų.

   Deimantų kaina Jungtinių Valstijų doleriu valiuta nustatoma pagal šias savybes:

   - Karatai (laukelio pav. carat) (0.2--5.01). Karatas (angl. carat, sutrumpinimas ct) matavimo vienetas, skirtas išmatuoti šlifuotų juvelyrinių dirbinių ir brangakmenių svorį.
   - Pjūvio kokybė (laukelio pav. cut) (Pakankamai gera, gera, labai gera, aukštos kokybės, idealus)
   - Spalva (laukelio pav. color) (Nuo J (blogiausios) iki D (geriausios))
   - Skaidrumas (laukelio pav. clarity) (I1 (blogiausia), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (geriausia))
   - Bendras gylis (laukelio pav. depth) (%) – 2 * z / (x + y) Deimanto gylis - tai jo aukštis (milimetrais), matuojamas nuo kiuletės (apatinio galo) iki stalo (plokščio viršutinio paviršiaus).
   - Plačiausios plokštumos plotis (laukelio pav. table) Deimanto plokštuma - tai plokščia deimanto briauna, kuri matoma, kai akmuo nukreiptas į viršų. Pagrindinė deimanto plokštumos paskirtis - laužti įeinančius šviesos spindulius ir leisti atsispindėjusiems šviesos spinduliams iš deimanto vidaus patekti stebėtojui į akis.
   - X – ilgis (mm)
   - Y – plotis (mm)
   - Z – aukštis (mm)

   Duomenų rinkinio nuoroda:
   https://datasetsearch.research.google.com/search?src=0&query=Diamonds%20Prices&docid=L2cvMTFyOW52a2o2cw%3D%3D

2. **Hipotezė**

   Ar įmanoma nuspėti sėkmingai deimanto kainą pasitelkus mašininio mokymo algoritmą.

3. Darbo tikslas ir uždaviniai

Darbo tikslas: Sukurti mašininio mokymo algoritmą, kuris suteiks galimybę, vėliau apmokius, prognozuoti specifinių deimantų kainas.

Uždaviniai:

1. Sutvarkyti turimą duomenų rinkinį.
2. Atlikti duomenų žvalgomają analizė.
3. Sukurti mašininio mokymo algoritmą.
4. Apmokyti modelį pasitelkiant apmokymo duomenimis.
5. Ištestuoti modelį pasitelkint testavimo duomenimis.

4. Duomenų importavimas, tvarkymas (trūkstamų reikšmių sutvarkymas, išskirčių pašalinimas ir t.t), žvalgomoji analizė (histogramos, sklaidos diagramos, koreliacijos koef.).

1) Duomenų importavimas.

```
#Importavimo metodas lokaliam failui
df = pd.read_csv("Deimantai_duomenys_2023.csv")
df
```

✓ 0.1s

	id	carat	cut	color	clarity	depth	table	price	x	y	z
0	1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	3	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	4	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	5	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75
...	...	...	...	...	...	...	...	...	...	...	...
53938	53939	0.86	Premium	H	SI2	61.0	58.0	2757	6.15	6.12	3.74
53939	53940	0.75	Ideal	D	SI2	62.2	55.0	2757	5.83	5.87	3.64
53940	53941	0.71	Premium	E	SI1	60.5	55.0	2756	5.79	5.74	3.49
53941	53942	0.71	Premium	F	SI1	59.8	62.0	2756	5.74	5.73	3.43
53942	53943	0.70	Very Good	E	VS2	60.5	59.0	2757	5.71	5.76	3.47

2) Stulpelių duomenų tipo nustatymas.

```
df.info()
✓ 0.0s
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53943 entries, 0 to 53942
Data columns (total 11 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   id       53943 non-null  int64
 1   carat    53943 non-null  float64
 2   cut      53943 non-null  object
 3   color    53943 non-null  object
 4   clarity  53943 non-null  object
 5   depth    53943 non-null  float64
 6   table    53943 non-null  float64
 7   price    53943 non-null  int64
 8   x        53943 non-null  float64
 9   y        53943 non-null  float64
 10  z        53943 non-null  float64
dtypes: float64(6), int64(2), object(3)
memory usage: 4.5+ MB
```

3) Eilučių ir stulpelių skaičiaus nustatymas.

```
df.shape
✓ 0.0s
(53943, 11)
```

4) Peržiūrėjimas ar nėra trūkstamų duomenų.

```
df.isnull().sum()
✓ 0.0s
id         0
carat      0
cut        0
color      0
clarity    0
depth      0
table      0
price      0
x          0
y          0
z          0
dtype: int64
```

5) Nereikalingų stulpelių pašalinimas. Pašalinamas „id“ stulpelis, kadangi jis nesuteiks vertės neuroniniam tinklui.

```
df = df.copy()
print(f"Prieš pašalinimą: {list(df)}")

df.drop('id', axis=1, inplace=True)
print(f"Po pašalinimo: {list(df)}")

df
```

✓ 0.0s

```
Prieš pašalinimą: ['id', 'carat', 'cut', 'color', 'clarity', 'depth', 'table', 'price', 'x', 'y', 'z']
Po pašalinimo: ['carat', 'cut', 'color', 'clarity', 'depth', 'table', 'price', 'x', 'y', 'z']
```

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75
...	...	...	...	...	...	...	...	...	...	...
53938	0.86	Premium	H	SI2	61.0	58.0	2757	6.15	6.12	3.74
53939	0.75	Ideal	D	SI2	62.2	55.0	2757	5.83	5.87	3.64
53940	0.71	Premium	E	SI1	60.5	55.0	2756	5.79	5.74	3.49
53941	0.71	Premium	F	SI1	59.8	62.0	2756	5.74	5.73	3.43
53942	0.70	Very Good	E	VS2	60.5	59.0	2757	5.71	5.76	3.47

53943 rows × 10 columns

6) Pasikartojančių eilučių radimas ir pašalinimas. Rastos besiduplikuojančios 295 eilutės pašalintos.

```
duplicates = df[df.duplicated(keep=False)]
duplicates
```

✓ 0.0s

	carat	cut	color	clarity	depth	table	price	x	y	z
1004	0.79	Ideal	G	SI1	62.3	57.0	2898	5.90	5.85	3.66
1005	0.79	Ideal	G	SI1	62.3	57.0	2898	5.90	5.85	3.66
1006	0.79	Ideal	G	SI1	62.3	57.0	2898	5.90	5.85	3.66
1007	0.79	Ideal	G	SI1	62.3	57.0	2898	5.90	5.85	3.66
1008	0.79	Ideal	G	SI1	62.3	57.0	2898	5.90	5.85	3.66
...	...	...	...	...	...	...	...	...	...	...
53931	0.71	Premium	F	SI1	59.8	62.0	2756	5.74	5.73	3.43
53932	0.70	Very Good	E	VS2	60.5	59.0	2757	5.71	5.76	3.47
53940	0.71	Premium	E	SI1	60.5	55.0	2756	5.79	5.74	3.49
53941	0.71	Premium	F	SI1	59.8	62.0	2756	5.74	5.73	3.43
53942	0.70	Very Good	E	VS2	60.5	59.0	2757	5.71	5.76	3.47

295 rows × 10 columns

```
df = df.drop_duplicates()
df
```
✓ 0.0s

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75
...	...	...	...	...	...	...	...	...	...	...
53935	0.72	Ideal	D	SI1	60.8	57.0	2757	5.75	5.76	3.50
53936	0.72	Good	D	SI1	63.1	55.0	2757	5.69	5.75	3.61
53937	0.70	Very Good	D	SI1	62.8	60.0	2757	5.66	5.68	3.56
53938	0.86	Premium	H	SI2	61.0	58.0	2757	6.15	6.12	3.74
53939	0.75	Ideal	D	SI2	62.2	55.0	2757	5.83	5.87	3.64

53794 rows × 10 columns

7) Atstatomas indeksas.

```
df = df.reset_index()
df.head()
```
✓ 0.0s

	index	carat	cut	color	clarity	depth	table	price	x	y	z
0	0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

```
df.drop('index', axis=1, inplace=True)
```
✓ 0.0s

8) Kainos stulpelis perkeliamas į priekį.

```
col_to_move = df.pop('price')
df.insert(0, 'price', col_to_move)
df
```
✓ 0.0s

	price	carat	cut	color	clarity	depth	table	x	y	z
0	326	0.23	Ideal	E	SI2	61.5	55.0	3.95	3.98	2.43
1	326	0.21	Premium	E	SI1	59.8	61.0	3.89	3.84	2.31
2	327	0.23	Good	E	VS1	56.9	65.0	4.05	4.07	2.31
3	334	0.29	Premium	I	VS2	62.4	58.0	4.20	4.23	2.63
4	335	0.31	Good	J	SI2	63.3	58.0	4.34	4.35	2.75
...	...	...	...	...	...	...	...	...	...	...
53789	2757	0.72	Ideal	D	SI1	60.8	57.0	5.75	5.76	3.50
53790	2757	0.72	Good	D	SI1	63.1	55.0	5.69	5.75	3.61
53791	2757	0.70	Very Good	D	SI1	62.8	60.0	5.66	5.68	3.56
53792	2757	0.86	Premium	H	SI2	61.0	58.0	6.15	6.12	3.74
53793	2757	0.75	Ideal	D	SI2	62.2	55.0	5.83	5.87	3.64

53794 rows × 10 columns

9) Aprašomoji statistika. "Karatų" vidurkis yra maždaug 0,78, o std - maždaug 0,47, t. y. daugiau nei pusė vidurkio, o tai rodo, kad duomenys labai skiriasi. Kainos vidurkis yra apie 3933 , o std apie 3988. Nuokrypis yra didesnis nei vidurkis.

```
df.describe()
```
✓ 0.0s

	price	carat	depth	table	x	y	z
count	53794.000000	53794.00000	53794.000000	53794.000000	53794.000000	53794.000000	53794.000000
mean	3933.065082	0.79778	61.748080	57.458109	5.731214	5.734653	3.538714
std	3988.114460	0.47339	1.429909	2.233679	1.120695	1.141209	0.705037
min	326.000000	0.20000	43.000000	43.000000	0.000000	0.000000	0.000000
25%	951.000000	0.40000	61.000000	56.000000	4.710000	4.720000	2.910000
50%	2401.000000	0.70000	61.800000	57.000000	5.700000	5.710000	3.530000
75%	5326.750000	1.04000	62.500000	59.000000	6.540000	6.540000	4.030000
max	18823.000000	5.01000	79.000000	95.000000	10.740000	58.900000	31.800000

10) Pastebima, kad „x“, „y“, „z“ turi mažiausias reikšmes nulį. Tai reiškias, kad yra klaidingų eilučių ir reikia jas pašalinti. Pašalinta 19 eilučių.

```
df = df.drop(df[df["x"]==0].index)
df = df.drop(df[df["y"]==0].index)
df = df.drop(df[df["z"]==0].index)
df.shape
```
✓ 0.0s

```
(53775, 10)
```

11) Koreliacijos koeficientai. Pagal šią koreliacijos lentelę galima teigti, kad labiausiai kaina priklauso nuo karatų, ilgio(x), pločio(y), aukščio(z). Labai maža

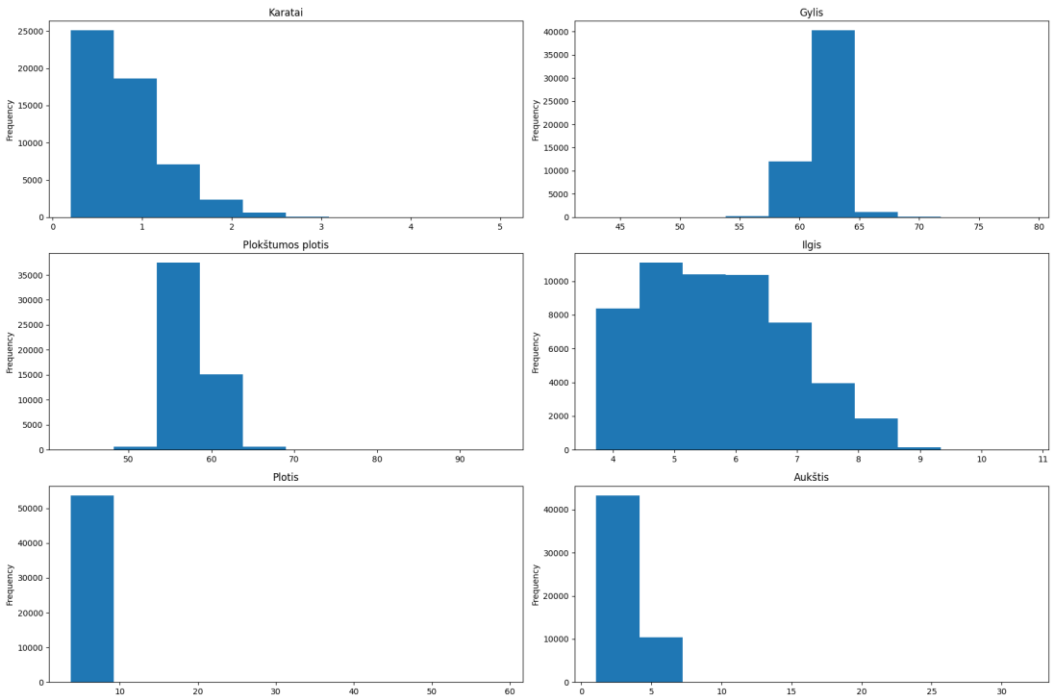
koreliacija pastebima lyginant plokštumos pločio parametrą. Ypatingai maža kaino priklausomybė nuo gylio.

```
df_statistical = df.copy()
df_statistical = df_statistical.drop(['cut', 'color', 'clarity'], axis=1)
corr_matrix = df_statistical.corr(method='kendall')
display(corr_matrix)
```

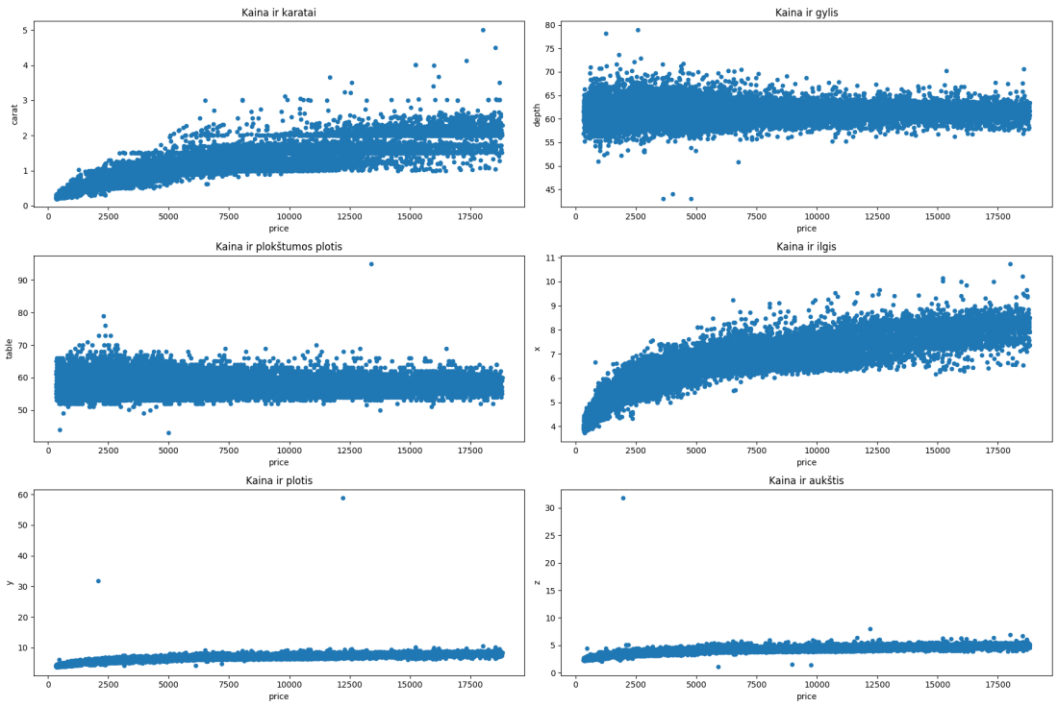
✓ 0.1s

	price	carat	depth	table	x	y	z
price	1.000000	0.834193	0.005866	0.121171	0.831074	0.829802	0.820423
carat	0.834193	1.000000	0.019889	0.138540	0.961081	0.957789	0.952488
depth	0.005866	0.019889	1.000000	-0.178778	-0.017887	-0.019270	0.071050
table	0.121171	0.138540	-0.178778	1.000000	0.143004	0.138194	0.112186
x	0.831074	0.961081	-0.017887	0.143004	1.000000	0.968273	0.913323
y	0.829802	0.957789	-0.019270	0.138194	0.968273	1.000000	0.912111
z	0.820423	0.952488	0.071050	0.112186	0.913323	0.912111	1.000000

12) Histogramos. Karatų histograma (viršuje kairėje): Šioje histogramoje matomas mažėjantis dažnis didėjant x ašies reikšmei, o tai rodo, kad pasiskirstymas yra dešininis. Ji atspindi diskretųjį kintamąjį, kurio vertės svyruoja nuo 0 iki 5. Gylio histograma (viršuje dešinėje): Šioje histogramoje pavaizduotas dažnių pasiskirstymas. Pagrindiniai atvejai susitelkę ties 65, o mažiau - 55. Tai rodo, kad pasiskirstymas koncentruotas aplink centrinę reikšmę su galimu nuokrypiu. Plokštumos pločio histograma (centre kairėje): Ši histograma rodo, kad pasiskirstymas yra kairysis. Dauguma duomenų taškų patenka į 60-70 taškų intervalą pagal x ašį, o dažnis mažėja, kai reikšmės yra didesnės. Ilgio histograma (viduryje dešinėje): Šioje histogramoje matomas šiek tiek tolygus pasiskirstymas tarp 4 ir 9 reikšmių x ašyje. Didėjant vertei dažnis palaipsniui mažėja. Pločio histograma (apačioje kairėje): Šioje histogramoje pateikiamas duomenų rinkinys, kurio didžiausias dažnis yra mažiausiame intervale (0-10), o didėjant vertei jis greitai mažėja, o tai rodo, kad pasiskirstymas yra dešininis. Duomenys yra diskretūs, svyruojantys maždaug nuo 0 iki 50. Aukščio histograma (apačioje dešinėje): Šioje histogramoje matomas kairės pusės pasiskirstymas su dideliu dažniu esant mažai x ašies vertei (apie 5) ir staigiu kritimu didėjant vertei. Reikės atlikti duomenų normalizacija.



13) Sklaidos diagramos. Kaina ir karatai (viršuje kairėje): Atrodo, kad egzistuoja teigiama koreliacija, kai didesnės karatų vertės brangakmeniai yra brangesni. Didėjant karatų vertei, duomenų taškų sklaida tampa įvairesnė. Kaina ir gylis (viršuje dešinėje): Taškai yra plačiai išsibarstę, be aiškios tendencijos, o tai rodo, kad koreliacija tarp kainos ir gylio yra maža arba jos visai nėra. Yra keletas taškų, nutolusių nuo pagrindinio klasterio ir galinčių būti nuokrypiais. Kaina ir plokštumos plotis (centre kairėje): Panašiai kaip ir gylio atveju, taškų sklaida yra didelė, o tai rodo mažą koreliaciją tarp kainos ir plokštumos pločio. Dauguma duomenų taškų sukoncentruoti horizontalioje juostoje, o tai gali reikšti, kad plokštumos plotis turi standartinį diapazoną nepriklausomai nuo kainos. Kaina ir ilgis (centre dešinėje): Pastebima akivaizdi teigiama koreliacijos tendencija, rodanti, kad didėjant ilgiui didėja ir kaina. Kaina ir plotis (apačioje kairėje): Pastebima nedidelė teigiama tendencija, rodanti, kad platesnių deimantų kainos gali būti didesnės. Kai kurie taškai gerokai nutolę nuo pagrindinės tendencijos, todėl gali prireikti juos pašalinti. Kaina ir aukštis (apačioje dešinėje): Yra nedidelė teigiama koreliacija, tačiau ji nėra tokia stipri kaip ilgio ar karatų diagramose. Kai kurie taškai gerokai nutolę nuo pagrindinės tendencijos, todėl gali prireikti juos pašalinti. Pastebima maža koreliacija tarp kainos ir gylio, bei ploštumos pločio. Taip pat pastebimas nuokrypis gylio(depth), plokštumos pločio(table), pločio(y) ir aukščio(z) elementuose.



14) Nuokrypių identifikavimas ir pašalinimas. Nuokrypių identifikavimui ir pašalinimui panaudotas Inter-Quartile Range (IQR). Kadangi šis metodas patikimesnis nei standartinis nuokrypis ir nereikalauja tiek skaičiavimo resursų, kaip Local Outlier Factor (LOF). Šiuo metodu pašalinta 6363 eilučių.

```
# Selecting only numerical columns
data_numerical = df2.select_dtypes(include=['int', 'float'])

# Calculating the Interquartile Range (IQR)
Q1 = data_numerical.quantile(0.25)
Q3 = data_numerical.quantile(0.75)
IQR = Q3 - Q1

# Creating a mask for outliers in the numerical columns
mask = (data_numerical < (Q1 - 1.5 * IQR)) | (data_numerical > (Q3 + 1.5 * IQR))

# Applying the mask to the original dataframe to keep all columns
data_outliers_IQR = df2[~(mask).any(axis=1)]

# Resetting the index
data_outliers_IQR.reset_index(drop=True, inplace=True)

# Check the first few rows of the resulting dataframe
data_outliers_IQR
```

✓ 0.0s

	price	carat	cut	color	clarity	depth	table	x	y	z
0	326	0.23	Ideal	E	SI2	61.5	55.0	3.95	3.98	2.43
1	326	0.21	Premium	E	SI1	59.8	61.0	3.89	3.84	2.31
2	334	0.29	Premium	I	VS2	62.4	58.0	4.20	4.23	2.63
3	335	0.31	Good	J	SI2	63.3	58.0	4.34	4.35	2.75
4	336	0.24	Very Good	J	VVS2	62.8	57.0	3.94	3.96	2.48
...	...	...	...	...	...	...	...	...	...	...
47407	2757	0.72	Ideal	D	SI1	60.8	57.0	5.75	5.76	3.50
47408	2757	0.72	Good	D	SI1	63.1	55.0	5.69	5.75	3.61
47409	2757	0.70	Very Good	D	SI1	62.8	60.0	5.66	5.68	3.56
47410	2757	0.86	Premium	H	SI2	61.0	58.0	6.15	6.12	3.74
47411	2757	0.75	Ideal	D	SI2	62.2	55.0	5.83	5.87	3.64

47412 rows × 10 columns

15) Kategorinių stulpelių keitimas skaitinėmis reikšmėmis.

Nustatytos unikalio reikšmės:

```
df3['cut'].unique()
✓ 0.0s
array(['Ideal', 'Premium', 'Good', 'Very Good', 'Fair'], dtype=object)

df3['color'].unique()
✓ 0.0s
array(['E', 'I', 'J', 'H', 'F', 'G', 'D'], dtype=object)

df3['clarity'].unique()
✓ 0.0s
array(['SI2', 'SI1', 'VS2', 'VVS2', 'VVS1', 'VS1', 'I1', 'IF'],
      dtype=object)
```

Sukuriami papildomi stulpeliai remiantis kategorinėmis reikšmėmis:

```
# Convert 'cut', 'color', and 'clarity' columns to categorical types
df3['cut'] = df3['cut'].astype('category')
df3['color'] = df3['color'].astype('category')
df3['clarity'] = df3['clarity'].astype('category')

# Create an instance of OneHotEncoder
enc = OneHotEncoder()

# Passing multiple columns to the encoder
enc_data = pd.DataFrame(enc.fit_transform(df3[['cut', 'color', 'clarity']]).toarray(),
                        columns=enc.get_feature_names_out(['cut', 'color', 'clarity']))

# Ensure the indices are aligned
enc_data.index = df3.index

# Merge with the main DataFrame
df4 = df3.join(enc_data)

# Display the DataFrame
display(df4)
```

0.1s

	price	carat	cut	color	clarity	depth	table	x	y	z	...	color_I	color_J	clarity_I1	clarity_IF	clarity_SI1	clarity_SI2	clarity_VS1	clarity_VS2	clarity_VVS1	clarity_VVS2
0	326	0.23	Ideal	E	SI2	61.5	55.0	3.95	3.98	2.43	...	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
1	326	0.21	Premium	E	SI1	59.8	61.0	3.89	3.84	2.31	...	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
2	334	0.29	Premium	I	VS2	62.4	58.0	4.20	4.23	2.63	...	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
3	335	0.31	Good	J	SI2	63.3	58.0	4.34	4.35	2.75	...	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
4	336	0.24	Very Good	J	VVS2	62.8	57.0	3.94	3.96	2.48	...	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
7407	2757	0.72	Ideal	D	SI1	60.8	57.0	5.75	5.76	3.50	...	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
7408	2757	0.72	Good	D	SI1	63.1	55.0	5.69	5.75	3.61	...	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
7409	2757	0.70	Very Good	D	SI1	62.8	60.0	5.66	5.68	3.56	...	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
7410	2757	0.86	Premium	H	SI2	61.0	58.0	6.15	6.12	3.74	...	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
7411	2757	0.75	Ideal	D	SI2	62.2	55.0	5.83	5.87	3.64	...	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0

7412 rows × 30 columns

Išmetami buvę kategoriniai stulpeliai:

```
df4.drop('cut', axis=1, inplace=True)
df4.drop('color', axis=1, inplace=True)
df4.drop('clarity', axis=1, inplace=True)
✓ 0.0s
```

16) Duomenų normalizacija. Naudotas MinMaxScaler.

```
# Assuming df3 is your original dataframe
columns_for_rescale = ['price', 'carat', 'depth', 'table', 'x', 'y', 'z']
columns_other = ['cut_Fair', 'cut_Good', 'cut_Ideal', 'cut_Premium', 'cut_Very Good', 'color_D', 'color_E', 'color_F', 'colo

# Initialize the MinMaxScaler
scaler = MinMaxScaler()

# Apply MinMax scaling (normalization) to the numerical columns
df5_normalized = pd.DataFrame(scaler.fit_transform(df5[columns_for_rescale]), columns=columns_for_rescale, index=df5.index)

# Merge the normalized numerical data with the categorical data
df5 = pd.concat([df5_normalized, df5[columns_other]], axis=1)

✓ 0.0s
```

17) Duomenų padalinimas į apmokymo ir testavimo dalis.

Suskirstyta į įvestis ir išvestis:

```
X = df5.iloc[:, 1:]  #inde
y = df5.iloc[:, 0]   #targ
print(X.shape, y.shape)
✓ 0.0s
(47412, 26) (47412,)
```

Suskirstyta į apmokymo ir testavimo duomenis (Testavimui skirta 20 proc.):

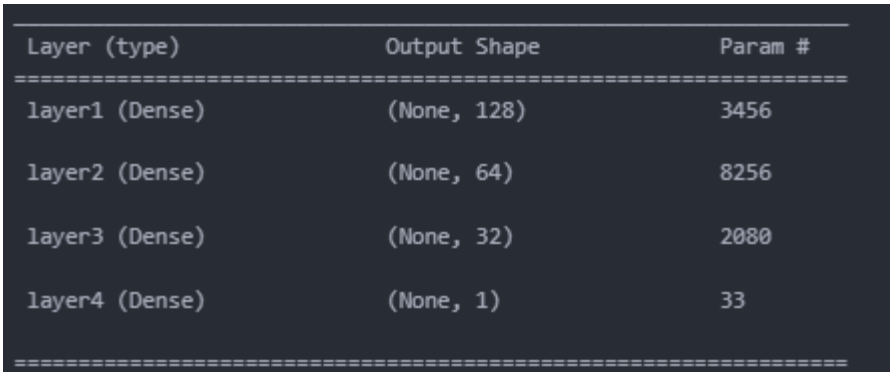
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
✓ 0.0s
(37929, 26) (9483, 26) (37929,) (9483,)
```

5. Sukurti mašininio mokymo algoritmą su 2 pasirinktais neuroniniais tinklais, bei 1 kitu mašininio mokymo metodu ir apmokyti savo pasirinktais duomenimis.

Neuroniniam tinklui įgyvendinti pasirinktas MLP. Buvo bandomos struktūros:

Layer (type)	Output Shape	Param #
layer1 (Dense)	(None, 32)	864
layer2 (Dense)	(None, 1)	33

Layer (type)	Output Shape	Param #
layer1 (Dense)	(None, 64)	1728
layer2 (Dense)	(None, 32)	2080
layer3 (Dense)	(None, 1)	33



Layer (type)	Output Shape	Param #
layer1 (Dense)	(None, 128)	3456
layer2 (Dense)	(None, 64)	8256
layer3 (Dense)	(None, 32)	2080
layer4 (Dense)	(None, 1)	33

Mašininiam mokymui įgyvendinti pasirinktas atsitiktinio miško regresorius.

6. Parametrų analizė. Randami geriausi parametrai kiekvieno algoritmo geriausiam tikslumui gauti (pvz. neuroninio tinklo struktūra)

Naudojant MinMaxScaller:

Random Forest regressor: MSE: 0.0021704490717859115

Neural network: MSE: 0.00120.0011817804770544171

Naudojant RobustScaller:

Random Forest regressor: MSE: 0.02065251983851111

Neural network: MSE: 0.01320.013241766020655632

Naudojant RobustScaller gaunama mažiau klaidų.

Mašininio mokymo metodas:

N_estimators = 200

MSE: 0.0021704490717859115

Laikas: 8.4s

N_estimators = 500

MSE: 0.0021857605335437983

Laikas: 19.5s

N_estimators = 1000

MSE: 0.00217549764988900 5

Laikas: 41.5s

Išvada naudojant 200 ar 1000 n_estimators skirtumo didelio nėra.

Max_depth = 10

MSE: 0.002175497649889005

Laikas: 41.5s

Max_depth = 15

MSE: 0.0012597468916757215

Laikas: 1m 16s

Max_depth = 20

MSE: 0.0011082761322727863

Laikas:1m 51s

Išvada: Naudojant 15 gylį klaidos žymiai sumažėja, toliau naudojant 20 gylį skirtumo didelio nebėra.

```
rf_regressor = RandomForestRegressor(
    n_estimators=1000,      # Medžių skaiči
    max_depth=20,           # Didžiausias me
    min_samples_split=4,    # Mažiausias mėg
    min_samples_leaf=2,     # Mažiausias mėg
    max_features='sqrt',    # Added paramete
    random_state=42
)
rf_regressor.fit(X_train, y_train)
```

Be K-Fold:

MSE: 0.0011082761322727863

Laikas: 1m 51.1s

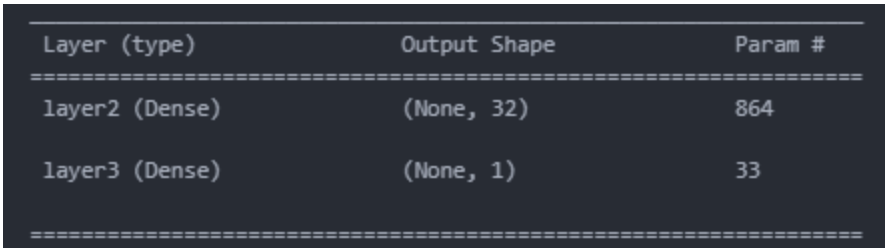
Su K-Fold:

MSE: 0.005833425561323271

Laikas: 8m 56.4s

Išvada: Su K-Fold užtrunka 8 kartus daugiau ir mse pakyla. Bet tokiu atveju galima tikėtis tikslesnio modelio.

Neuroninio tinklo struktūros pasirinkimas:



Layer (type)	Output Shape	Param #
layer2 (Dense)	(None, 32)	864
layer3 (Dense)	(None, 1)	33

Epoch: 200

Optimizer: Adam

Su K-Fold cross:

K-Fold Cross-Validation Scores: [0.0015098085405153261, 0.00501767549338297, 0.006176091729978808, 0.00021477114447029263, 0.0005097444288109526]

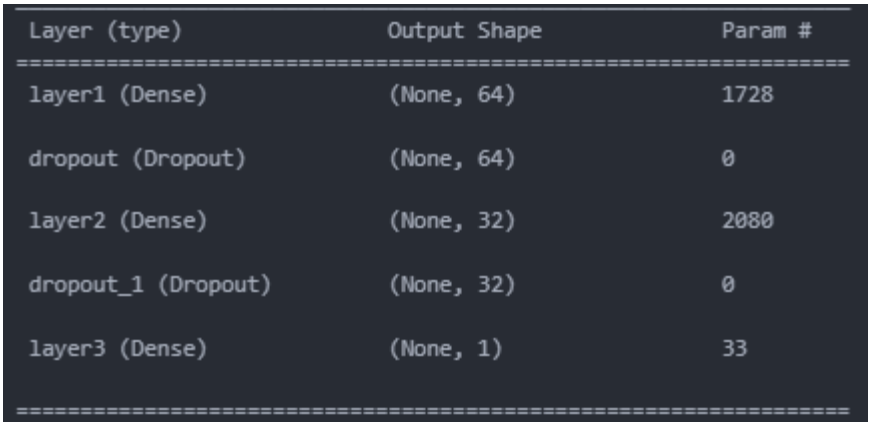
Mean Squared Error: 0.00268561826743167

Laikas: 34m 40.1s

Be K-Fold cross:

Mean Squared Error: 0.001078673744505916

Laikas: 10m 5.3s



Layer (type)	Output Shape	Param #
layer1 (Dense)	(None, 64)	1728
dropout (Dropout)	(None, 64)	0
layer2 (Dense)	(None, 32)	2080
dropout_1 (Dropout)	(None, 32)	0
layer3 (Dense)	(None, 1)	33

Epoch: 200

Optimizer: Adam

Su K-Fold cross:

K-Fold Cross-Validation Scores: [0.0028300625728349188, 0.009339429098533053, 0.0073863846347377235, 0.00039028683684758455, 0.0018051365696004327]

Mean Squared Error: 0.004350259942510743

Laikas: 44m 37.3s

Be K-Fold cross:

Mean Squared Error: 0.0010307865467485884

Laikas: 12m 6.4s

Layer (type)	Output Shape	Param #
layer1 (Dense)	(None, 128)	3456
dropout_2 (Dropout)	(None, 128)	0
layer2 (Dense)	(None, 64)	8256
dropout_3 (Dropout)	(None, 64)	0
layer3 (Dense)	(None, 32)	2080
dropout_4 (Dropout)	(None, 32)	0
layer4 (Dense)	(None, 1)	33

Epoch: 200

Optimizer: Adam

Su K-Fold cross:

K-Fold Cross-Validation Scores: [0.00383945093015455, 0.011651399530261723, 0.012894252702073249, 0.005876493592919912, 0.0018176542836694921]

Mean Squared Error: 0.007215850207815786

Laikas: 51m 50.3s

Be K-Fold cross:

Mean Squared Error: 0.0019157848474146994

Laikas: 14m 25.3s

Išvada:

Naudojant K-Fold galima pastebėti kaip kinta MSE. Jos variacija yra svari, tad naudoti paprasta „train and split“ metodą nėra tikslu. Taip pat paprastesni MLP modelis duoda gersni rezultatą per mažesnį laikotarpį.

Optimizers palyginimas

Epoch: 200

Batch size: 32

Neural network: 32 - > 1

Adam:

K-Fold Cross-Validation Scores: [0.0015098085405153261, 0.00501767549338297, 0.006176091729978808, 0.00021477114447029263, 0.0005097444288109526]

Mean Squared Error: 0.00268561826743167

Laikas: 34m 40.1s

Sgd:

K-Fold Cross-Validation Scores: [0.0016844956640854147, 0.0034314744440802052, 0.006546239492851202, 0.0005872120940270915, 0.0005948035947992228]

Mean Squared Error: 0.0025688450579686275

Laikas: 32m 52.2s

Rmsprop:

K-Fold Cross-Validation Scores: [0.0015553725417060146, 0.003487680036479138, 0.003082420358376085, 0.0005206742584050798, 0.0004365722290103986]

Mean Squared Error: 0.0018165438847953432

Laikas: 34m 1.3s

Išvada:

Naudojant rmsprop optimizatoriu padaroma mažiausiai klaidų.

Skirtingas Epochu skaičius:

200:

Mean Squared Error: 0.0012046757619827986

Laikas: 11m 5s

500:

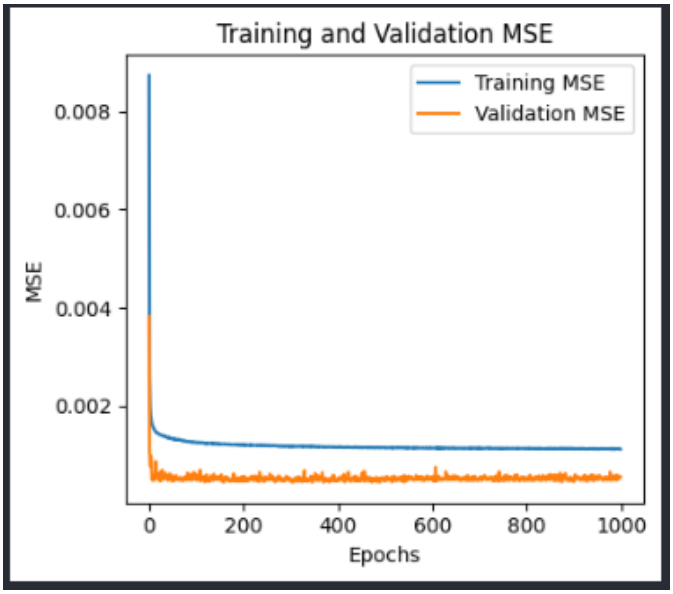
Mean Squared Error: 0.001156431157141924

Laikas: 23m 6.75s

1000:

Mean Squared Error: 0.0011258381103723513

Laikas: 46m 13.5s



7. Tarpusavio metodų tikslumo palyginimas (palyginami skirtingi metodai tiems patiems duomenims klasifikuoti ar prognozuoti) tarp geriausią tikslumą duodančių algoritmų t.y radus tinkamiausius parametrus.

Geriausias neuroninis tinklas:

Layer (type)	Output Shape	Param #
layer2 (Dense)	(None, 32)	864
layer3 (Dense)	(None, 1)	33

Epoch: 200

Optimizer: Adam

Be K-Fold cross:

MSE: 0.001078673744505916

Laikas: 10m 5.3s

Su K-Fold cross:

K-Fold Cross-Validation Scores: [0.0015098085405153261, 0.00501767549338297, 0.006176091729978808, 0.00021477114447029263, 0.0005097444288109526]

MSE: 0.00268561826743167

Laikas: 34m 40.1s

Geriausias ML metodas:

```
rf_regressor = RandomForestRegressor(
    n_estimators=1000,      # Medžių skaiči
    max_depth=20,           # Didžiausias me
    min_samples_split=4,    # Mažiausias mėg
    min_samples_leaf=2,     # Mažiausias mėg
    max_features='sqrt',    # Added paramete
    random_state=42
)
rf_regressor.fit(X_train, y_train)
```

Be K-Fold:

MSE: 0.0011082761322727863

Laikas: 1m 51.1s

Su K-Fold:

MSE: 0.005833425561323271

Laikas: 8m 56.4s

Išvada: MLP neuroninio tinklo metodas šiuo atveju tikslenis nei ML.

8. Išvados, pastebėjimai

   Papraščiausia ML modelis apmokomas per trumpesni laika parode geresni rezultatą nei gilusis neuroninis tinklas. MLP modelis buvo tikslesnis nei ML metodas.