**Министерство науки и высшего образования Российской Федерации** **Федеральное государственное бюджетное образовательное учреждение высшего образования** **«Московский государственный технический университет** **имени Н.Э. Баумана** 

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

**Факультет «Информатика и системы управления»**

**Кафедра ИУ5 «Системы обработки информации и управления»**

Курс «Технологии машинного обучения»

Лабораторная работа №4

Выполнил:

студент группы ИУ5-63Б

Комаров Д. С.

Проверил:

Гапанюк Ю. Е.

2024 г.

**Ход работы:**

Загрузка датасета

**import** pandas **as** pd  
**import** seaborn **as** sns  
**import** matplotlib.pyplot **as** plt  
**import** numpy **as** np  
**from** sklearn.preprocessing **import** PolynomialFeatures, MinMaxScaler, StandardScaler  
**from** sklearn.linear\_model **import** LinearRegression, Lasso, Ridge  
**from** sklearn.tree **import** DecisionTreeRegressor, export\_graphviz, export\_text  
**from** sklearn.svm **import** SVR  
**from** sklearn.metrics **import** r2\_score, mean\_squared\_error, mean\_absolute\_error  
**from** sklearn.model\_selection **import** train\_test\_split, GridSearchCV  
**from** IPython.display **import** Image  
**from** IPython.core.display **import** HTML  
**from** sklearn.preprocessing **import** OneHotEncoder  
**from** sklearn.preprocessing **import** OrdinalEncoder,LabelEncoder,StandardScaler  
**from** sklearn.pipeline **import** make\_pipeline  
**from** sklearn.impute **import** SimpleImputer  
**from** sklearn.compose **import** make\_column\_transformer  
**from** sklearn.model\_selection **import** train\_test\_split

data = pd.read\_csv('sample\_data/cleaned\_all\_phones.csv')  
data.head()

{"type":"dataframe","variable\_name":"data"}

data.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1512 entries, 0 to 1511  
Data columns (total 22 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 phone\_name 1512 non-null object   
 1 brand 1512 non-null object   
 2 os 1512 non-null object   
 3 inches 1512 non-null float64  
 4 resolution 1512 non-null object   
 5 battery 1512 non-null int64   
 6 battery\_type 1512 non-null object   
 7 ram(GB) 1512 non-null int64   
 8 announcement\_date 1512 non-null object   
 9 weight(g) 1512 non-null float64  
 10 storage(GB) 1512 non-null int64   
 11 video\_720p 1512 non-null bool   
 12 video\_1080p 1512 non-null bool   
 13 video\_4K 1512 non-null bool   
 14 video\_8K 1512 non-null bool   
 15 video\_30fps 1512 non-null bool   
 16 video\_60fps 1512 non-null bool   
 17 video\_120fps 1512 non-null bool   
 18 video\_240fps 1512 non-null bool   
 19 video\_480fps 1512 non-null bool   
 20 video\_960fps 1512 non-null bool   
 21 price(USD) 1512 non-null float64  
dtypes: bool(10), float64(3), int64(3), object(6)  
memory usage: 156.6+ KB

**Чистка данных**

*# проверим есть ли пропущенные значения*  
data.isnull().sum()

phone\_name 0  
brand 0  
os 0  
inches 0  
resolution 0  
battery 0  
battery\_type 0  
ram(GB) 0  
announcement\_date 0  
weight(g) 0  
storage(GB) 0  
video\_720p 0  
video\_1080p 0  
video\_4K 0  
video\_8K 0  
video\_30fps 0  
video\_60fps 0  
video\_120fps 0  
video\_240fps 0  
video\_480fps 0  
video\_960fps 0  
price(USD) 0  
dtype: int64

data[['width', 'height']] = data.resolution.str.split('x', expand = True).astype('int64')  
  
data.head()

{"type":"dataframe","variable\_name":"data"}

data['announcement\_year'] = data.announcement\_date.apply(**lambda** x: x.split('-')[0]).astype('int64')  
data.head()

{"type":"dataframe","variable\_name":"data"}

data.drop(columns = ['announcement\_date', 'resolution', 'phone\_name'], inplace = True)

data.describe()

{"summary":"{\n \"name\": \"data\",\n \"rows\": 8,\n \"fields\": [\n {\n \"column\": \"inches\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 532.5310851556795,\n \"min\": 0.4770430982109062,\n \"max\": 1512.0,\n \"num\_unique\_values\": 8,\n \"samples\": [\n 6.4224603174603185,\n 6.5,\n 1512.0\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"battery\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 2148.127173043608,\n \"min\": 784.6070221906537,\n \"max\": 7250.0,\n \"num\_unique\_values\": 8,\n \"samples\": [\n 4389.798941798942,\n 4500.0,\n 1512.0\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"ram(GB)\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 531.8729184957466,\n \"min\": 1.0,\n \"max\": 1512.0,\n \"num\_unique\_values\": 7,\n \"samples\": [\n 1512.0,\n 6.6838624338624335,\n 8.0\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"weight(g)\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 482.7337788722055,\n \"min\": 26.20011485546831,\n \"max\": 1512.0,\n \"num\_unique\_values\": 8,\n \"samples\": [\n 187.6362433862434,\n 187.0,\n 1512.0\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"storage(GB)\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 507.5688180604079,\n \"min\": 1.0,\n \"max\": 1512.0,\n \"num\_unique\_values\": 7,\n \"samples\": [\n 1512.0,\n 109.16468253968254,\n 128.0\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"price(USD)\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 803.235184359574,\n \"min\": 40.0,\n \"max\": 2300.0,\n \"num\_unique\_values\": 8,\n \"samples\": [\n 337.8470357142857,\n 260.0,\n 1512.0\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"width\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 1117.8750058935389,\n \"min\": 253.48894039516676,\n \"max\": 3840.0,\n \"num\_unique\_values\": 7,\n \"samples\": [\n 1512.0,\n 1035.212962962963,\n 1080.0\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"height\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 1057.435936394939,\n \"min\": 469.73457812549356,\n \"max\": 3840.0,\n \"num\_unique\_values\": 7,\n \"samples\": [\n 1512.0,\n 2207.190476190476,\n 2400.0\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"announcement\_year\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 710.6079645347033,\n \"min\": 1.7001902014840866,\n \"max\": 2023.0,\n \"num\_unique\_values\": 8,\n \"samples\": [\n 2020.4100529100529,\n 2021.0,\n 1512.0\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n }\n ]\n}","type":"dataframe"}

categorical\_cols = list(data.select\_dtypes(include='object').columns)  
onehot = OneHotEncoder(sparse\_output=False)  
encoded = onehot.fit\_transform(data[categorical\_cols])  
encoded\_cols = onehot.get\_feature\_names\_out()  
encoded\_df = pd.DataFrame(encoded, columns=encoded\_cols)

encoded\_df

{"type":"dataframe","variable\_name":"encoded\_df"}

final\_df = pd.concat([data, encoded\_df], axis=1).drop(columns=categorical\_cols)

final\_df

{"type":"dataframe","variable\_name":"final\_df"}

bool\_cols = list(data.select\_dtypes(include='bool').columns)  
final\_df[bool\_cols] = final\_df[bool\_cols].replace({True: 1, False: 0})

final\_df.info()

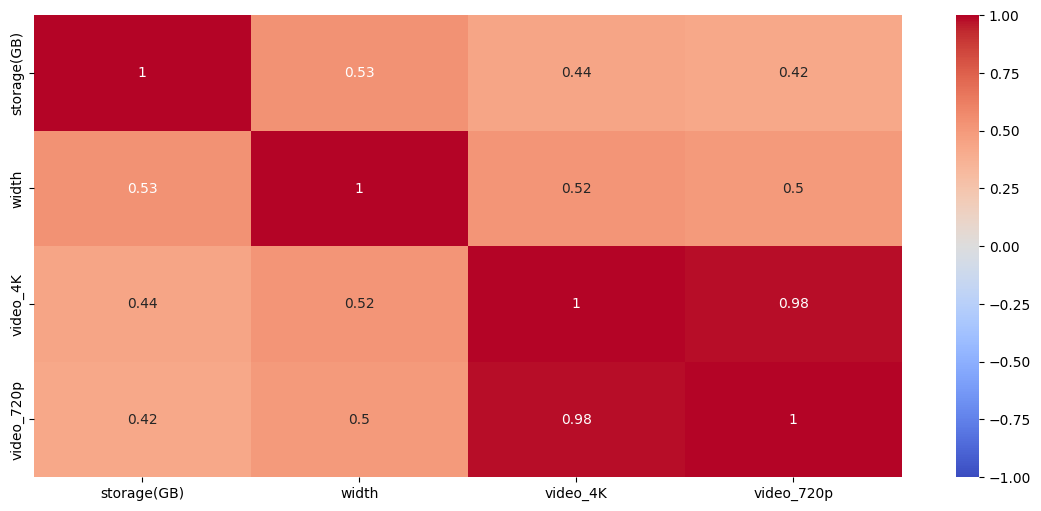
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1512 entries, 0 to 1511  
Data columns (total 68 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 inches 1512 non-null float64  
 1 battery 1512 non-null int64   
 2 ram(GB) 1512 non-null int64   
 3 weight(g) 1512 non-null float64  
 4 storage(GB) 1512 non-null int64   
 5 video\_720p 1512 non-null int64   
 6 video\_1080p 1512 non-null int64   
 7 video\_4K 1512 non-null int64   
 8 video\_8K 1512 non-null int64   
 9 video\_30fps 1512 non-null int64   
 10 video\_60fps 1512 non-null int64   
 11 video\_120fps 1512 non-null int64   
 12 video\_240fps 1512 non-null int64   
 13 video\_480fps 1512 non-null int64   
 14 video\_960fps 1512 non-null int64   
 15 price(USD) 1512 non-null float64  
 16 width 1512 non-null int64   
 17 height 1512 non-null int64   
 18 announcement\_year 1512 non-null int64   
 19 brand\_Apple 1512 non-null float64  
 20 brand\_Google 1512 non-null float64  
 21 brand\_Honor 1512 non-null float64  
 22 brand\_Huawei 1512 non-null float64  
 23 brand\_LG 1512 non-null float64  
 24 brand\_Lenovo 1512 non-null float64  
 25 brand\_OnePlus 1512 non-null float64  
 26 brand\_Oppo 1512 non-null float64  
 27 brand\_Realme 1512 non-null float64  
 28 brand\_Samsung 1512 non-null float64  
 29 brand\_Sony 1512 non-null float64  
 30 brand\_Vivo 1512 non-null float64  
 31 brand\_Xiaomi 1512 non-null float64  
 32 os\_Android 1512 non-null float64  
 33 os\_Android 10 1512 non-null float64  
 34 os\_Android 10/ Android 11 1512 non-null float64  
 35 os\_Android 11 1512 non-null float64  
 36 os\_Android 12 1512 non-null float64  
 37 os\_Android 12 or 13 1512 non-null float64  
 38 os\_Android 12L 1512 non-null float64  
 39 os\_Android 13 1512 non-null float64  
 40 os\_Android 5.1 1512 non-null float64  
 41 os\_Android 6 1512 non-null float64  
 42 os\_Android 6.0 1512 non-null float64  
 43 os\_Android 6.0.1 1512 non-null float64  
 44 os\_Android 7.0 1512 non-null float64  
 45 os\_Android 7.0.1 1512 non-null float64  
 46 os\_Android 7.1 1512 non-null float64  
 47 os\_Android 7.1.1 1512 non-null float64  
 48 os\_Android 7.1.2 1512 non-null float64  
 49 os\_Android 8.0 1512 non-null float64  
 50 os\_Android 8.0 Oreo 1512 non-null float64  
 51 os\_Android 8.1 1512 non-null float64  
 52 os\_Android 8.1 Oreo 1512 non-null float64  
 53 os\_Android 9.0 1512 non-null float64  
 54 os\_Android 9.0 Pie 1512 non-null float64  
 55 os\_EMUI 12 1512 non-null float64  
 56 os\_EMUI 13 1512 non-null float64  
 57 os\_Tizen 3.0 1512 non-null float64  
 58 os\_iOS 11 1512 non-null float64  
 59 os\_iOS 11.1.1 1512 non-null float64  
 60 os\_iOS 12 1512 non-null float64  
 61 os\_iOS 13 1512 non-null float64  
 62 os\_iOS 14.1 1512 non-null float64  
 63 os\_iOS 15 1512 non-null float64  
 64 os\_iOS 15.4 1512 non-null float64  
 65 os\_iOS 16 1512 non-null float64  
 66 battery\_type\_Li-Ion 1512 non-null float64  
 67 battery\_type\_Li-Po 1512 non-null float64  
dtypes: float64(52), int64(16)  
memory usage: 803.4 KB

print('Признаки, имеющие максимальную по модулю корреляцию с ценой телефона')  
best\_params = final\_df.corr()['price(USD)'].map(abs).sort\_values(ascending=False)[1:]  
best\_params = best\_params[best\_params.values > 0.3]  
best\_params

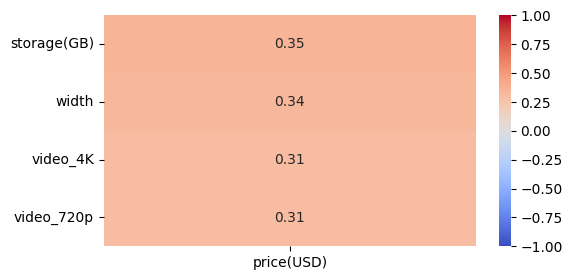
Признаки, имеющие максимальную по модулю корреляцию с ценой телефона

storage(GB) 0.354250  
width 0.338867  
video\_4K 0.312411  
video\_720p 0.310810  
Name: price(USD), dtype: float64

plt.figure(figsize=(14, 6))  
sns.heatmap(final\_df[best\_params.index].corr(), vmin=-1, vmax=1, cmap='coolwarm', annot=True)  
plt.show()



plt.figure(figsize=(6, 3))  
sns.heatmap(pd.DataFrame(data[np.append(best\_params.index.values, 'price(USD)')].corr()['price(USD)'].sort\_values(ascending=False)[1:]), vmin=-1, vmax=1, cmap='coolwarm', annot=True)  
plt.show()



**Разделение выборки на обучающую и тестовую**

y = data['price(USD)']  
X = data[best\_params.index]  
x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=3)

**Линейная регрессия**

**def** print\_metrics(y\_test, y\_pred):  
 *# коэффициент детерминации*  
 print(f"R^2: {r2\_score(y\_test, y\_pred)}")  
 *# среднеквадратичная ошибка*  
 print(f"MSE: {mean\_squared\_error(y\_test, y\_pred)}")  
 *# средняя абсолютная ошибка*  
 print(f"MAE: {mean\_absolute\_error(y\_test, y\_pred)}")

linear\_model = LinearRegression()  
linear\_model.fit(x\_train, y\_train)  
y\_pred\_linear = linear\_model.predict(x\_test)  
print\_metrics(y\_test, y\_pred\_linear)

R^2: 0.13510555261347457  
MSE: 66433.49914733712  
MAE: 163.7848199812498

**Полиноминальная регрессия**

poly\_model = PolynomialFeatures(degree=3)  
x\_train\_poly = poly\_model.fit\_transform(x\_train)  
x\_test\_poly = poly\_model.fit\_transform(x\_test)  
linear\_model = LinearRegression()  
linear\_model.fit(x\_train\_poly, y\_train)  
y\_pred\_poly = linear\_model.predict(x\_test\_poly)  
print\_metrics(y\_test, y\_pred\_poly)

R^2: -0.8520645814281242  
MSE: 142259.12903350135  
MAE: 179.1300123620549

**SVM**

scaler = StandardScaler().fit(x\_train)  
x\_train\_scaled = pd.DataFrame(scaler.transform(x\_train), columns=x\_train.columns)  
x\_test\_scaled = pd.DataFrame(scaler.transform(x\_test), columns=x\_train.columns)  
x\_train\_scaled.describe()

{"summary":"{\n \"name\": \"x\_train\_scaled\",\n \"rows\": 8,\n \"fields\": [\n {\n \"column\": \"storage(GB)\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 373.8174690255228,\n \"min\": -1.4947908459995323,\n \"max\": 1058.0,\n \"num\_unique\_values\": 7,\n \"samples\": [\n 1058.0,\n -6.296160820180472e-17,\n 0.25100927754626334\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"width\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 373.64275208677526,\n \"min\": -2.1055893753998194,\n \"max\": 1058.0,\n \"num\_unique\_values\": 7,\n \"samples\": [\n 1058.0,\n 2.5184643280721886e-16,\n 0.15328536075849372\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"video\_4K\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 373.97091200805585,\n \"min\": -1.046446130935795,\n \"max\": 1058.0,\n \"num\_unique\_values\": 5,\n \"samples\": [\n -4.53323579052994e-17,\n 0.9556153636936283,\n 1.0004729250678182\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"video\_720p\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 373.97376147434903,\n \"min\": -1.058414134902744,\n \"max\": 1058.0,\n \"num\_unique\_values\": 5,\n \"samples\": [\n -1.6453966943404966e-16,\n 0.9448097554856335,\n 1.0004729250678182\n ],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n }\n }\n ]\n}","type":"dataframe"}

params = {'C': np.concatenate([np.arange(0.1, 2, 0.1), np.arange(2, 15, 1)])}  
svm\_model = SVR(kernel='linear')  
grid\_cv = GridSearchCV(estimator=svm\_model, param\_grid=params, cv=10, n\_jobs=-1, scoring='r2')  
grid\_cv.fit(x\_train\_scaled, y\_train)  
print(grid\_cv.best\_params\_)

{'C': 8.0}

best\_svm\_model = grid\_cv.best\_estimator\_  
best\_svm\_model = SVR(kernel='linear', C=8)  
best\_svm\_model.fit(x\_train\_scaled, y\_train)  
y\_pred\_svm = best\_svm\_model.predict(x\_test\_scaled)  
print\_metrics(y\_test, y\_pred\_svm)

R^2: 0.09028888194733287  
MSE: 69875.91719208499  
MAE: 145.91318982938506

**Дерево решений**

params = {'min\_samples\_leaf': range(3, 30)}  
tree = DecisionTreeRegressor(random\_state=3)  
grid\_cv = GridSearchCV(estimator=tree, cv=5, param\_grid=params, n\_jobs=-1, scoring='neg\_mean\_absolute\_error')  
grid\_cv.fit(x\_train, y\_train)  
print(grid\_cv.best\_params\_)

{'min\_samples\_leaf': 13}

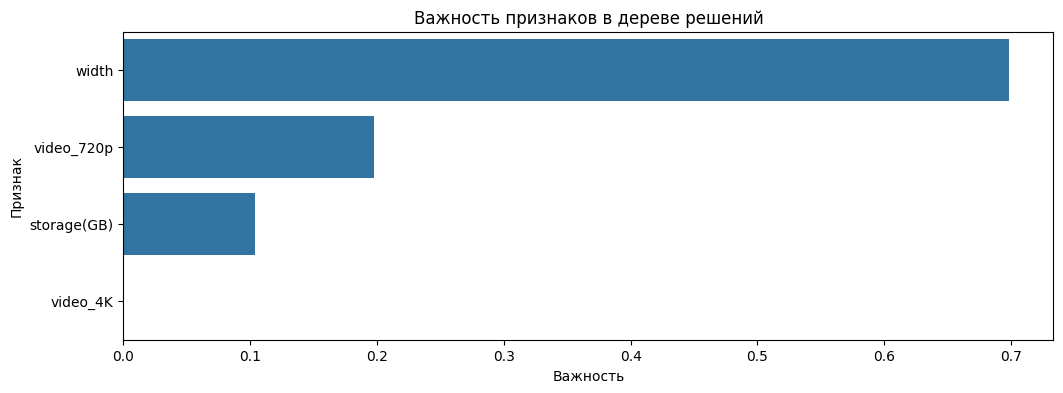
best\_tree = grid\_cv.best\_estimator\_  
best\_tree.fit(x\_train, y\_train)  
y\_pred\_tree = best\_tree.predict(x\_test)  
print\_metrics(y\_test, y\_pred\_tree)

R^2: 0.17201798017858494  
MSE: 63598.21475791498  
MAE: 152.63921401566597

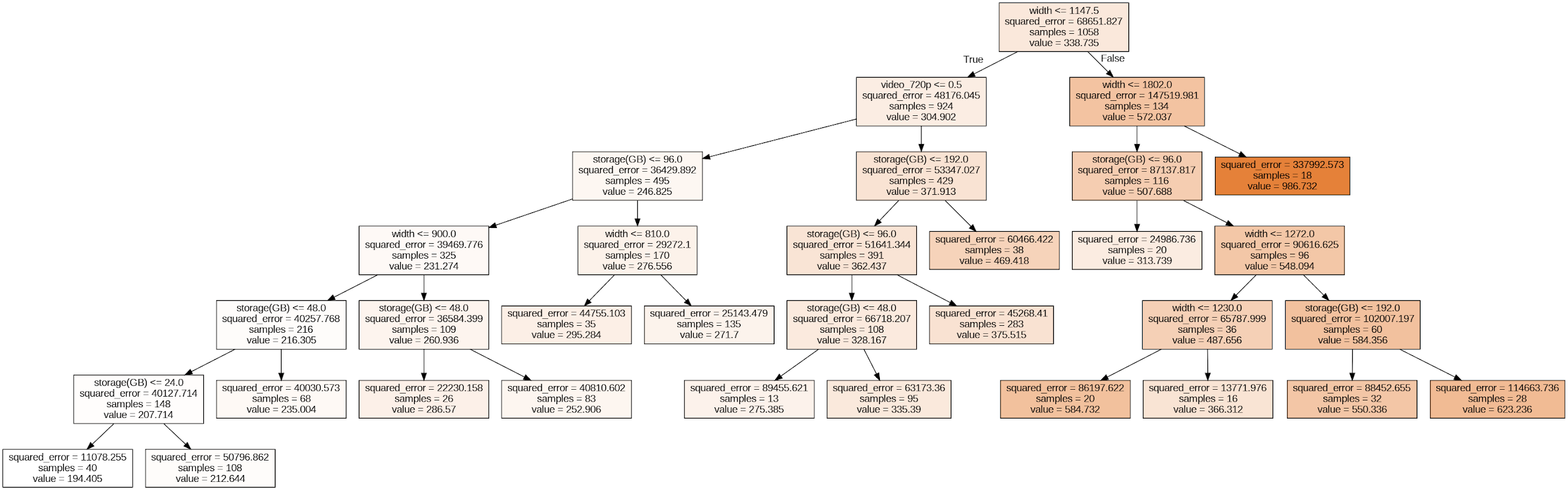
importances = pd.DataFrame(data=zip(x\_train.columns, best\_tree.feature\_importances\_), columns=['Признак', 'Важность'])  
print('Важность признаков в дереве решений\n')  
**for** row **in** importances.sort\_values(by='Важность', ascending=False).values:  
 print(f'{row[0]}: {round(row[1], 3)}')

Важность признаков в дереве решений  
  
width: 0.698  
video\_720p: 0.197  
storage(GB): 0.104  
video\_4K: 0.0

plt.figure(figsize=(12, 4))  
sns.barplot(data=importances.sort\_values(by='Важность', ascending=False), y='Признак', x='Важность', orient='h', )  
plt.title('Важность признаков в дереве решений')  
plt.show()



export\_graphviz(best\_tree, feature\_names=best\_params.index, filled=True, out\_file='tree.dot')  
!dot -Tpng tree.dot -o tree.png  
Image(filename='tree.png')



print('Линейная регрессия')  
print\_metrics(y\_test, y\_pred\_linear)  
  
print('\nПолиномиальная регрессия')  
print\_metrics(y\_test, y\_pred\_poly)  
  
print('\nМетод опорных векторов')  
print\_metrics(y\_test, y\_pred\_svm)  
  
print('\nДерево решений')  
print\_metrics(y\_test, y\_pred\_tree)

Линейная регрессия  
R^2: 0.13510555261347457  
MSE: 66433.49914733712  
MAE: 163.7848199812498  
  
Полиномиальная регрессия  
R^2: -0.8520645814281242  
MSE: 142259.12903350135  
MAE: 179.1300123620549  
  
Метод опорных векторов  
R^2: 0.09028888194733287  
MSE: 69875.91719208499  
MAE: 145.91318982938506  
  
Дерево решений  
R^2: 0.17201798017858494  
MSE: 63598.21475791498  
MAE: 152.63921401566597