```python
from google.colab import files
uploaded = files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving clean_kolesa_dataset.csv to clean_kolesa_dataset (1).csv
```

```python
import io
import pandas as pd
df = pd.read_csv(io.BytesIO(uploaded['clean_kolesa_dataset (1).csv']))
```

```python
df.head()
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 19985,\n  \"fields\":
[\n    {\n      \"column\": \"\\u0411\\u0440\\u0435\\u043d\\u0434\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 87,\n        \"samples\": [\n
\"Maybach\",\n          \"Nissan\",\n          \"Lifan\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"\\u041c\\u043e\\u0434\\u0435\\
u043b\\u044c\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 1008,\n
\"samples\": [\n          \"Prairie Joy\",\n          \"Charger\",\n
\"CLK 200\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"\\
u0413\\u043e\\u0434\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 8,\n        \"min\": 1954,\n
\"max\": 2020,\n        \"num_unique_values\": 62,\n
\"samples\": [\n          1978,\n          1955,\n          1995\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"\\u0413\\u043e\\u0440\\u043e\\
u0434\",\n      \"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 230,\n        \"samples\": [\n          \"\\
u041a\\u0430\\u043f\\u0430\\u043b\",\n          \"\\u0418\\u043d\\
u0434\\u0435\\u0440\\u0431\\u043e\\u0440\\u0441\\u043a\\u0438\\
u0439\",\n          \"\\u0423\\u0440\\u0430\\u043b\\u044c\\u0441\\
u043a\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"\\
u041a\\u0443\\u0437\\u043e\\u0432\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 18,\n
\"samples\": [\n          \"\\u0441\\u0435\\u0434\\u0430\\u043d\",\n
\"\\u043f\\u0438\\u043a\\u0430\\u043f\",\n          \"\\u0445\\u044d\\
u0442\\u0447\\u0431\\u0435\\u043a\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"\\u041e\\u0431\\u044a\\u0435\\
u043c \\u0434\\u0432\\u0438\\u0433\\u0430\\u0442\\u0435\\u043b\\u044f,
\\u043b\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.9774208274564512,\n        \"min\": 0.2,\n        \"max\":
9.0,\n        \"num_unique_values\": 81,\n        \"samples\": [\n
4.5,\n          1.6,\n          0.8\n        ],\n
```

```
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"\\u041f\\u0440\\u043e\\u0431\\
u0435\\u0433\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 441077.6561358999,\n        \"min\":
1.0,\n        \"max\": 9999999.0,\n        \"num_unique_values\":
2734,\n        \"samples\": [\n          445687.0,\n          198.0,\n
157244.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"\\
u041a\\u043e\\u0440\\u043e\\u0431\\u043a\\u0430 \\u043f\\u0435\\
u0440\\u0435\\u0434\\u0430\\u0447\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 5,\n
\"samples\": [\n          \"\\u0430\\u0432\\u0442\\u043e\\u043c\\
u0430\\u0442\",\n          \"\\u0440\\u043e\\u0431\\u043e\\u0442\",\n
\"\\u0432\\u0430\\u0440\\u0438\\u0430\\u0442\\u043e\\u0440\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"\\
u0420\\u0443\\u043b\\u044c\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 2,\n
\"samples\": [\n          \"\\u0441\\u043f\\u0440\\u0430\\u0432\\
u0430\",\n          \"\\u0441\\u043b\\u0435\\u0432\\u0430\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"\\
u0426\\u0432\\u0435\\u0442\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 42,\n
\"samples\": [\n          \"\\u0437\\u043e\\u043b\\u043e\\u0442\\
u0438\\u0441\\u0442\\u044b\\u0439 \\u043c\\u0435\\u0442\\u0430\\
u043b\\u043b\\u0438\\u043a\",\n          \"\\u0431\\u0435\\u043b\\
u044b\\u0439 \\u043c\\u0435\\u0442\\u0430\\u043b\\u043b\\u0438\\
u043a\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"\\
u041f\\u0440\\u0438\\u0432\\u043e\\u0434\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 3,\n
\"samples\": [\n          \"\\u043f\\u0435\\u0440\\u0435\\u0434\\
u043d\\u0438\\u0439 \\u043f\\u0440\\u0438\\u0432\\u043e\\u0434\",\n
\"\\u043f\\u043e\\u043b\\u043d\\u044b\\u0439 \\u043f\\u0440\\u0438\\
u0432\\u043e\\u0434\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"\\
u0420\\u0430\\u0441\\u0442\\u0430\\u043c\\u043e\\u0436\\u0435\\u043d
\\u0432 \\u041a\\u0430\\u0437\\u0430\\u0445\\u0441\\u0442\\u0430\\
u043d\\u0435\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          \"\\u041d\\u0435\\u0442\",\n          \"\\u0414\\u0430\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"\\
u0426\\u0435\\u043d\\u0430\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 7473011.643419682,\n
\"min\": 10000.0,\n        \"max\": 280000000.0,\n
\"num_unique_values\": 1333,\n        \"samples\": [\n
4125000.0,\n          9950000.0\n        ],\n
```

```
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"\\u0421\\u0440\\u0435\\u0434\\
u043d\\u043d\\u044f \\u0446\\u0435\\u043d\\u0430\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
7031030.41258355,\n        \"min\": 280000.0,\n        \"max\":
450000000.0,\n        \"num_unique_values\": 2518,\n
\"samples\": [\n          13350000.0,\n          5682000.0\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"\\
u0421\\u0441\\u044b\\u043b\\u043a\\u0430\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 18627,\n
\"samples\": [\n          \"https://kolesa.kz/a/show/112832811\",\n
\"https://kolesa.kz/a/show/112842010\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"\\u0412\\u0438\\u0434 \\u0442\\
u043e\\u043f\\u043b\\u0438\\u0432\\u0430\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 4,\n
\"samples\": [\n          \"\\u0434\\u0438\\u0437\\u0435\\u043b\\
u044c\",\n          \"\\u0433\\u0438\\u0431\\u0440\\u0438\\u0434\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe","variable_name":"df"}
```


```python
df.drop('Ссылка', axis=1, inplace=True)
```

```python
df['Модель'].fillna(df['Модель'].mode()[0], inplace=True)
df['Объем двигателя, л'].fillna(df['Объем двигателя, л'].median(),
inplace=True)
df['Привод'].fillna(df['Привод'].mode()[0], inplace=True)
df['Среднння цена'].fillna(df['Среднння цена'].median(), inplace=True)
df['Вид топлива'].fillna(df['Вид топлива'].mode()[0], inplace=True)
```

```
<ipython-input-314-662584bcd21f>:1: FutureWarning: A value is trying
to be set on a copy of a DataFrame or Series through chained
assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  df['Модель'].fillna(df['Модель'].mode()[0], inplace=True)
<ipython-input-314-662584bcd21f>:2: FutureWarning: A value is trying
to be set on a copy of a DataFrame or Series through chained
assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
```

```
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  df['Объем двигателя, л'].fillna(df['Объем двигателя, л'].median(),
inplace=True)
<ipython-input-314-662584bcd21f>:3: FutureWarning: A value is trying
to be set on a copy of a DataFrame or Series through chained
assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  df['Привод'].fillna(df['Привод'].mode()[0], inplace=True)
<ipython-input-314-662584bcd21f>:4: FutureWarning: A value is trying
to be set on a copy of a DataFrame or Series through chained
assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  df['Средння цена'].fillna(df['Средння цена'].median(), inplace=True)
<ipython-input-314-662584bcd21f>:5: FutureWarning: A value is trying
to be set on a copy of a DataFrame or Series through chained
assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.
```

```
  df['Вид топлива'].fillna(df['Вид топлива'].mode()[0], inplace=True)

missing_values = df.isnull().sum()
print(missing_values)

Бренд                          0
Модель                         0
Год                            0
Город                          0
Кузов                          0
Объем двигателя, л             0
Пробег                         0
Коробка передач                0
Руль                           0
Цвет                           0
Привод                         0
Растаможен в Казахстане        0
Цена                           0
Средння цена                   0
Вид топлива                    0
dtype: int64

df['is_expensive'] = (df['Цена'] > df['Средння цена']).astype(int)

categorical_columns = ['Бренд', 'Модель', 'Город', 'Кузов', 'Коробка
передач', 'Руль', 'Цвет', 'Привод', 'Растаможен в Казахстане', 'Вид
топлива']
df = pd.get_dummies(df, columns=categorical_columns, drop_first=True)

#from sklearn.preprocessing import StandardScaler

#numerical_columns = ['Объем двигателя, л', 'Пробег', 'Цена', 'Средння
цена']

#scaler = StandardScaler()
#df[numerical_columns] = scaler.fit_transform(df[numerical_columns])

df.head()
```

{"type":"dataframe","variable_name":"df"}

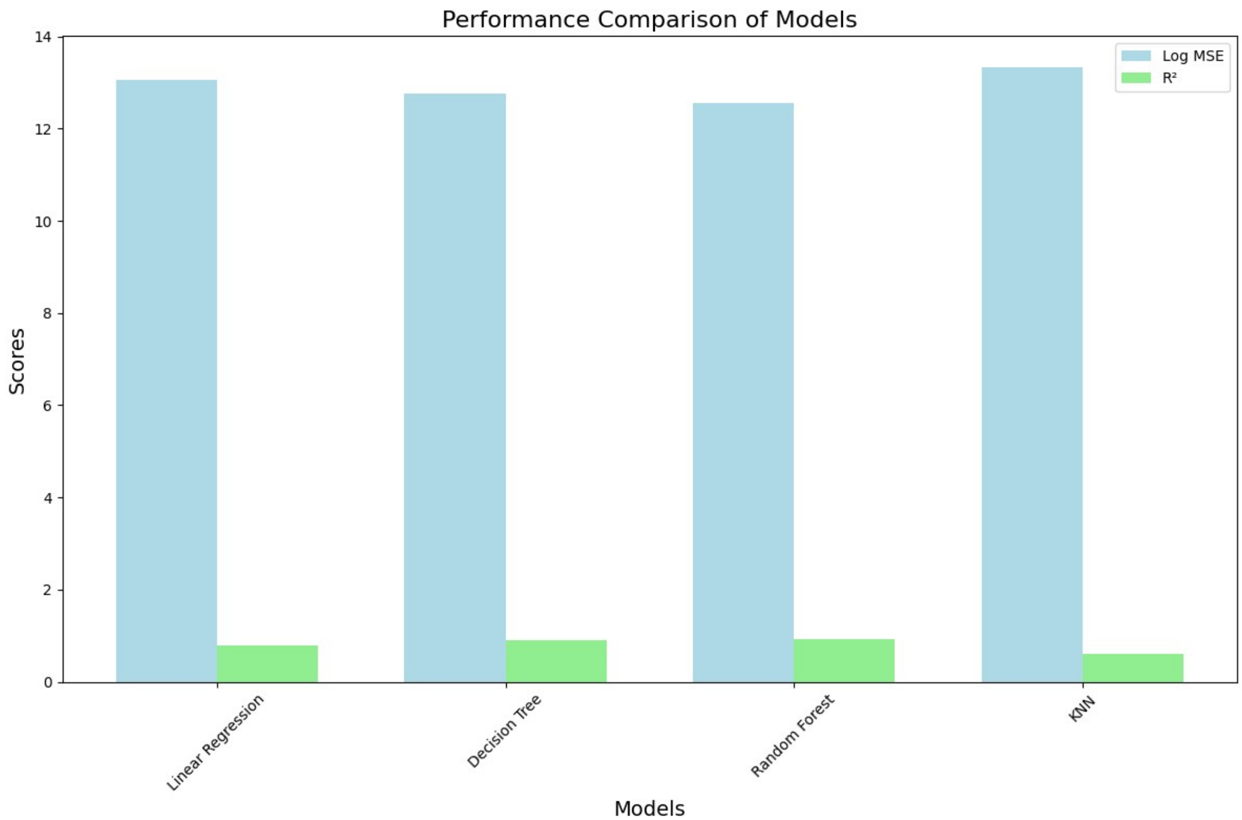
```
X = df.drop(columns=['Цена', 'Средння цена', 'is_expensive'])
y_linear = df['Цена']

y_logistic = df['is_expensive']

from sklearn.model_selection import train_test_split

X_train_linear, X_test_linear, y_train_linear, y_test_linear =
train_test_split(X, y_linear, test_size=0.2, random_state=42)
```

```python
X_train_logistic, X_test_logistic, y_train_logistic, y_test_logistic =
train_test_split(X, y_logistic, test_size=0.2, random_state=42)

from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_squared_error, r2_score

models_linear = {
    "Linear Regression": LinearRegression(),
    "Decision Tree": DecisionTreeRegressor(),
    "Random Forest": RandomForestRegressor(),
    "SVR": SVR(),
    "KNN": KNeighborsRegressor()
}


best_mse = float('inf')
best_r2 = -float('inf')
best_linear_model_mse = None
best_linear_model_r2 = None
linear_metrics = {}


for model_name, model in models_linear.items():
    model.fit(X_train_linear, y_train_linear)

    y_pred_linear = model.predict(X_test_linear)

    mse = mean_squared_error(y_test_linear, y_pred_linear)
    r2 = r2_score(y_test_linear, y_pred_linear)

    linear_metrics[model_name] = {'MSE': mse, 'R²': r2}

    if mse < best_mse:
        best_mse = mse
        best_linear_model_mse = model_name
    if r2 > best_r2:
        best_r2 = r2
        best_linear_model_r2 = model_name

print("\nAll Linear Regression Model Metrics:")
for model_name, metrics in linear_metrics.items():
    print(f"{model_name} - MSE: {metrics['MSE']} | R²:
{metrics['R²']}")

print(f"\nBest Linear Regression Model by MSE: {best_linear_model_mse}
with MSE: {best_mse}")
```

```
print(f"Best Linear Regression Model by R²: {best_linear_model_r2}
with R²: {best_r2}")
```

```
All Linear Regression Model Metrics:
Linear Regression - MSE: 11504448601686.875 | R²: 0.7905541400747852
Decision Tree - MSE: 5124336370728.435 | R²: 0.9067081722147118
Random Forest - MSE: 3460637822164.3726 | R²: 0.9369968705456547
SVR - MSE: 59751497009700.125 | R²: -0.08781429743451907
KNN - MSE: 21871357628155.297 | R²: 0.6018179171586537

Best Linear Regression Model by MSE: Random Forest with MSE:
3460637822164.3726
Best Linear Regression Model by R²: Random Forest with R²:
0.9369968705456547
```

```
import matplotlib.pyplot as plt
import numpy as np

models_linear = list(linear_metrics.keys())
mse_values = [linear_metrics[model]['MSE'] for model in models_linear]
r2_values = [linear_metrics[model]['R²'] for model in models_linear]

mse_values_log = np.log10(np.array(mse_values) + 1)

valid_models = [model for model, r2 in zip(models_linear, r2_values)
if r2 >= 0]
valid_mse_values = [mse_values_log[i] for i, r2 in
enumerate(r2_values) if r2 >= 0]
valid_r2_values = [r2 for r2 in r2_values if r2 >= 0]

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 8))

x = np.arange(len(valid_models))
ax1.bar(x, valid_mse_values, color="lightblue")
ax1.set_xlabel('Models', fontsize=14)
ax1.set_ylabel('Log MSE', fontsize=14)
ax1.set_title('Log MSE Comparison of Models', fontsize=16)
ax1.set_xticks(x)
ax1.set_xticklabels(valid_models, rotation=45)

ax2.bar(x, valid_r2_values, color="lightgreen")
ax2.set_xlabel('Models', fontsize=14)
ax2.set_ylabel('R²', fontsize=14)
ax2.set_title('R² Comparison of Models', fontsize=16)
ax2.set_xticks(x)
ax2.set_xticklabels(valid_models, rotation=45)

plt.tight_layout()
plt.show()
```



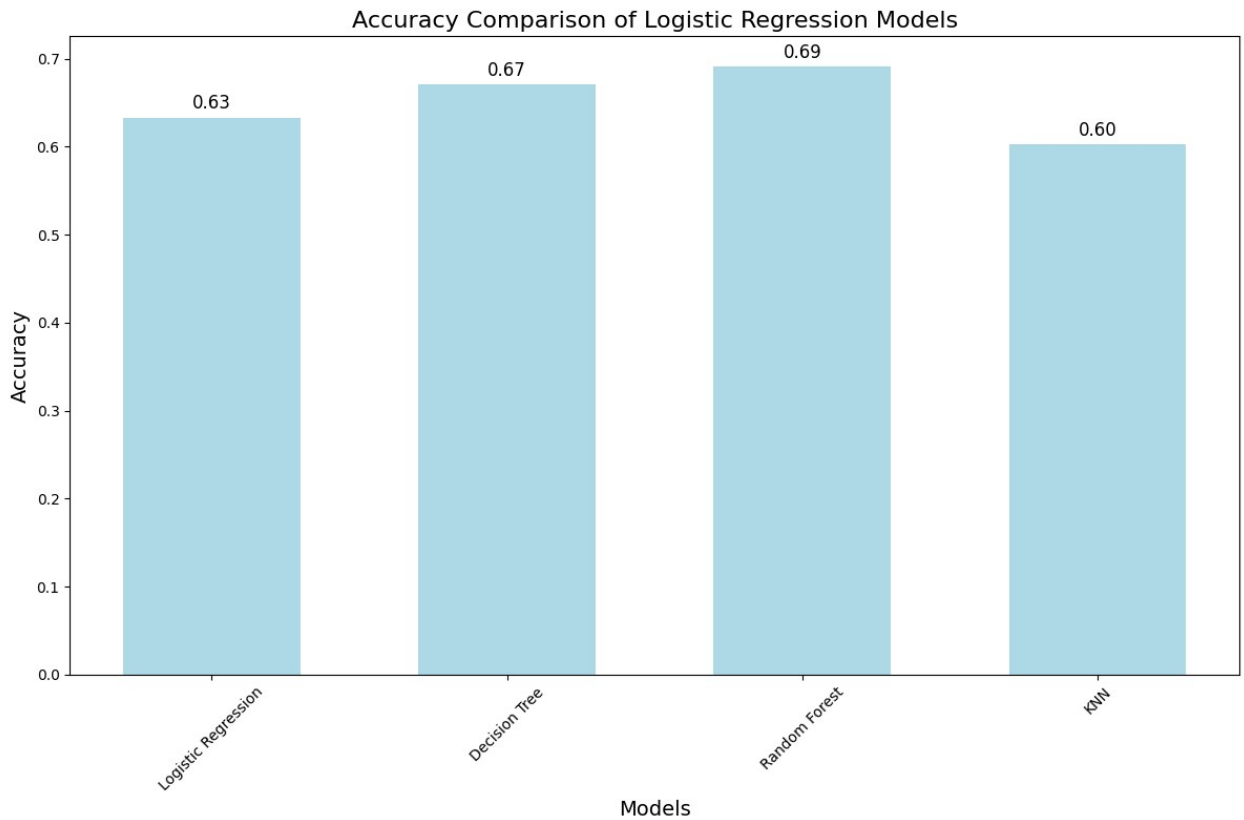

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report

models_logistic = {
    "Logistic Regression": LogisticRegression(),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(),
    "KNN": KNeighborsClassifier()
}

best_accuracy = 0
best_logistic_model = None
logistic_accuracy_list = {}

for model_name, model in models_logistic.items():
    model.fit(X_train_logistic, y_train_logistic)

    y_pred_logistic = model.predict(X_test_logistic)

    accuracy = accuracy_score(y_test_logistic, y_pred_logistic)
    logistic_accuracy_list[model_name] = accuracy
```

```
    print(f"{model_name} Accuracy: {accuracy}")
    print(classification_report(y_test_logistic, y_pred_logistic))

    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_logistic_model = model_name

print("\nAll Logistic Regression Model Accuracies:")
for model_name, accuracy in logistic_accuracy_list.items():
    print(f"{model_name}: {accuracy}")

print(f"\nBest Logistic Regression Model: {best_logistic_model} with
Accuracy: {best_accuracy}")
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/
_logistic.py:469: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(

Logistic Regression Accuracy: 0.6329747310482862
              precision    recall  f1-score   support

           0       0.67      0.78      0.72      2407
           1       0.55      0.40      0.47      1590

    accuracy                           0.63      3997
   macro avg       0.61      0.59      0.59      3997
weighted avg       0.62      0.63      0.62      3997

Decision Tree Accuracy: 0.6710032524393295
              precision    recall  f1-score   support

           0       0.73      0.73      0.73      2407
           1       0.59      0.58      0.58      1590

    accuracy                           0.67      3997
   macro avg       0.66      0.66      0.66      3997
weighted avg       0.67      0.67      0.67      3997

Random Forest Accuracy: 0.6910182636977733
              precision    recall  f1-score   support
```

```
              0       0.72      0.80      0.76      2407
              1       0.64      0.52      0.57      1590

    accuracy                              0.69      3997
   macro avg       0.68      0.66      0.67      3997
weighted avg       0.69      0.69      0.68      3997

KNN Accuracy: 0.6024518388791593
              precision    recall  f1-score   support

              0       0.66      0.71      0.68      2407
              1       0.50      0.44      0.47      1590

    accuracy                              0.60      3997
   macro avg       0.58      0.58      0.58      3997
weighted avg       0.60      0.60      0.60      3997


All Logistic Regression Model Accuracies:
Logistic Regression: 0.6329747310482862
Decision Tree: 0.6710032524393295
Random Forest: 0.6910182636977733
KNN: 0.6024518388791593

Best Logistic Regression Model: Random Forest with Accuracy:
0.6910182636977733
```

```
import matplotlib.pyplot as plt
import numpy as np

models_logistic = list(logistic_accuracy_list.keys())
accuracy_values = list(logistic_accuracy_list.values())

fig, ax = plt.subplots(figsize=(12, 8))

x = np.arange(len(models_logistic))
plt.bar(x, accuracy_values, color="lightblue", width=0.6)

plt.xlabel('Models', fontsize=14)
plt.ylabel('Accuracy', fontsize=14)
plt.title('Accuracy Comparison of Logistic Regression Models',
fontsize=16)
plt.xticks(x, models_logistic, rotation=45)

for i, acc in enumerate(accuracy_values):
    plt.text(x[i], acc + 0.01, f"{acc:.2f}", ha='center', fontsize=12)

plt.tight_layout()
plt.show()
```



+bonus

```
df.head()

{"type":"dataframe","variable_name":"df"}

data = pd.read_csv('clean_kolesa_dataset.csv')

data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19985 entries, 0 to 19984
Data columns (total 16 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Бренд                         19985 non-null  object
 1   Модель                        19950 non-null  object
 2   Год                           19985 non-null  int64
 3   Город                         19985 non-null  object
 4   Кузов                         19985 non-null  object
 5   Объем двигателя, л            19971 non-null  float64
 6   Пробег                        19985 non-null  float64
 7   Коробка передач               19985 non-null  object
 8   Руль                          19985 non-null  object
 9   Цвет                          19985 non-null  object
```

```
 10  Привод                     17740 non-null  object
 11  Растаможен в Казахстане  19985 non-null  object
 12  Цена                       19985 non-null  float64
 13  Среднння цена              18806 non-null  float64
 14  Ссылка                     19985 non-null  object
 15  Вид топлива                19971 non-null  object
dtypes: float64(4), int64(1), object(11)
memory usage: 2.4+ MB
```

```
import pandas as pd

brands = data['Бренд'].unique()
models = data['Модель'].unique()
years = data['Год'].unique()
cities = data['Город'].unique()
body_types = data['Кузов'].unique()
fuel_values = data['Объем двигателя, л'].unique()
run_values = data['Пробег'].unique()
transmission_types = data['Коробка передач'].unique()
wheel_drives = data['Руль'].unique()
colors = data['Цвет'].unique()
drive_types = data['Привод'].unique()
fuel_types = data['Вид топлива'].unique()
registered = data['Растаможен в Казахстане'].unique()

import ipywidgets as widgets
from IPython.display import display


brand_dropdown = widgets.Dropdown(
    options=brands.tolist(),
    description='Бренд:'
)
model_dropdown = widgets.Dropdown(
    options=models.tolist(),
    description='Модель:'
)
city_dropdown = widgets.Dropdown(
    options=cities.tolist(),
    description='Город:'
)
body_type_dropdown = widgets.Dropdown(
    options=body_types.tolist(),
    description='Кузов:'
)
transmission_dropdown = widgets.Dropdown(
    options=transmission_types.tolist(),
    description='Коробка:'
)
wheel_drive_dropdown = widgets.Dropdown(
```

```
    options=wheel_drives.tolist(),
    description='Руль:'
)
color_dropdown = widgets.Dropdown(
    options=colors.tolist(),
    description='Цвет:'
)
drive_type_dropdown = widgets.Dropdown(
    options=drive_types.tolist(),
    description='Привод:'
)
fuel_type_dropdown = widgets.Dropdown(
    options=fuel_types.tolist(),
    description='Топливо:'
)
year_slider = widgets.IntSlider(
    value=years.min(),
    min=years.min(),
    max=years.max(),
    step=1,
    description='Год:'
)
fuel_value_dropdown = widgets.Dropdown(
    options=fuel_values.tolist(),
    description='Объем двигателя:'
)
run_slider = widgets.IntSlider(
    value=run_values.min(),
    min=run_values.min(),
    max=run_values.max(),
    step=1000,
    description='Пробег:'
)
registered_dropdown = widgets.Dropdown(
    options=registered.tolist(),
    description='Растаможен:'
)


display(brand_dropdown, model_dropdown, city_dropdown,
body_type_dropdown,
        transmission_dropdown, wheel_drive_dropdown, color_dropdown,
        drive_type_dropdown, fuel_type_dropdown, year_slider,
fuel_value_dropdown, run_slider, registered_dropdown)
```

{"model_id":"fabaa4523e104ec5be51cfcf30743625","version_major":2,"version_minor":0}

{"model_id":"d5aef94f84c3410589a0d598412425a1","version_major":2,"version_minor":0}

```
{"model_id":"3aee04e82e744e1e9c91fabdd52318ca","version_major":2,"version_minor":0}

{"model_id":"340cfd8cc055483b9cd24cd8be5dce03","version_major":2,"version_minor":0}

{"model_id":"58309fe3e21447398b89f386babe4546","version_major":2,"version_minor":0}

{"model_id":"3e1510a13e2c4854b0f619236425ed1e","version_major":2,"version_minor":0}

{"model_id":"d900d33fd19b46839707a4d36cfb869a","version_major":2,"version_minor":0}

{"model_id":"af61059c932748b9a346f022d5e31ec5","version_major":2,"version_minor":0}

{"model_id":"9a13f32192db484d9ea2f5059cfb54aa","version_major":2,"version_minor":0}

{"model_id":"c9626a69b7e2448d9d496374d6150c25","version_major":2,"version_minor":0}

{"model_id":"b1f4dacd1d9f49f584d7404fcd994050","version_major":2,"version_minor":0}

{"model_id":"0ecd53608c184b958fada2f6528deb34","version_major":2,"version_minor":0}

{"model_id":"3d5aa86a509d4035ae228ccbae77ed4a","version_major":2,"version_minor":0}

user_input = {
    'Бренд': brand_dropdown.value,
    'Модель': model_dropdown.value,
    'Год': year_slider.value,
    'Город': city_dropdown.value,
    'Кузов': body_type_dropdown.value,
    'Объем двигателя, л': fuel_value_dropdown.value,
    'Пробег': run_slider.value,
    'Коробка передач': transmission_dropdown.value,
    'Руль': wheel_drive_dropdown.value,
    'Цвет': color_dropdown.value,
    'Привод': drive_type_dropdown.value,
    'Растаможен в Казахстане': registered_dropdown.value,
    'Вид топлива': fuel_type_dropdown.value,

}
print("User input:", user_input)
```

```
User input: {'Бренд': 'Mitsubishi', 'Модель': 'L200', 'Год': 2004,
'Город': 'Шымкент', 'Кузов': 'седан', 'Объем двигателя, л': 1.6,
'Пробег': 3201001, 'Коробка передач': 'механика', 'Руль': 'слева',
'Цвет': 'серебристый', 'Привод': 'передний привод', 'Растаможен в
Казахстане': 'Да', 'Вид топлива': 'газ'}
```

```
input_data = pd.DataFrame([user_input])

input_data = pd.concat([input_data, data], ignore_index=True)

input_data.head()
```


{"summary":"{\n \"name\": \"input_data\",\n \"rows\": 19986,\n
\"fields\": [\n   {\n   \"column\": \"\\u0411\\u0440\\u0435\\
u043d\\u0434\",\n   \"properties\": {\n   \"dtype\":
\"category\",\n   \"num_unique_values\": 87,\n
\"samples\": [\n   \"Maybach\",\n   \"Mitsubishi\",\n
\"Lifan\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"\\
u041c\\u043e\\u0434\\u0435\\u043b\\u044c\",\n   \"properties\": {\n
\"dtype\": \"category\",\n   \"num_unique_values\": 1008,\n
\"samples\": [\n   \"Prairie Joy\",\n   \"Charger\",\n
\"CLK 200\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"\\
u0413\\u043e\\u0434\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 8,\n   \"min\": 1954,\n
\"max\": 2020,\n   \"num_unique_values\": 62,\n
\"samples\": [\n   1978,\n   1955,\n   2004\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"\\u0413\\u043e\\u0440\\u043e\\
u0434\",\n   \"properties\": {\n   \"dtype\": \"category\",\n
\"num_unique_values\": 230,\n   \"samples\": [\n   \"\\
u041a\\u0430\\u043f\\u0430\\u043b\",\n   \"\\u0418\\u043d\\
u0434\\u0435\\u0440\\u0431\\u043e\\u0440\\u0441\\u043a\\u0438\\
u0439\",\n   \"\\u0423\\u0440\\u0430\\u043b\\u044c\\u0441\\
u043a\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"\\
u041a\\u0443\\u0437\\u043e\\u0432\",\n   \"properties\": {\n
\"dtype\": \"category\",\n   \"num_unique_values\": 18,\n
\"samples\": [\n   \"\\u0441\\u0435\\u0434\\u0430\\u043d\",\n
\"\\u043f\\u0438\\u043a\\u0430\\u043f\",\n   \"\\u0445\\u044d\\
u0442\\u0447\\u0431\\u0435\\u043a\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"\\u041e\\u0431\\u044a\\u0435\\
u043c \\u0434\\u0432\\u0438\\u0433\\u0430\\u0442\\u0435\\u043b\\u044f,
\\u043b\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 0.9774105958603385,\n   \"min\": 0.2,\n   \"max\":
9.0,\n   \"num_unique_values\": 81,\n   \"samples\": [\n
4.5,\n   1.6,\n   0.8\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\

```
n    },\n    {\n      \"column\": \"\\u041f\\u0440\\u043e\\u0431\\
u0435\\u0433\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 441580.85722190613,\n        \"min\":
1.0,\n        \"max\": 9999999.0,\n        \"num_unique_values\":
2735,\n        \"samples\": [\n          229245.0,\n
228300.0,\n          25500.0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"\\u041a\\u043e\\u0440\\u043e\\u0431\\u043a\\u0430 \\
u043f\\u0435\\u0440\\u0435\\u0434\\u0430\\u0447\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 5,\n        \"samples\": [\n          \"\\
u0430\\u0432\\u0442\\u043e\\u043c\\u0430\\u0442\",\n          \"\\
u0440\\u043e\\u0431\\u043e\\u0442\",\n          \"\\u0432\\u0430\\
u0440\\u0438\\u0430\\u0442\\u043e\\u0440\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"\\u0420\\u0443\\u043b\\u044c\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n          \"\\
u0441\\u043f\\u0440\\u0430\\u0432\\u0430\",\n          \"\\u0441\\
u043b\\u0435\\u0432\\u0430\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"\\u0426\\u0432\\u0435\\u0442\",\n      \"properties\":
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
42,\n        \"samples\": [\n          \"\\u0437\\u043e\\u043b\\
u043e\\u0442\\u0438\\u0441\\u0442\\u044b\\u0439 \\u043c\\u0435\\
u0442\\u0430\\u043b\\u043b\\u0438\\u043a\",\n          \"\\u0431\\
u0435\\u043b\\u044b\\u0439 \\u043c\\u0435\\u0442\\u0430\\u043b\\
u043b\\u0438\\u043a\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"\\
u041f\\u0440\\u0438\\u0432\\u043e\\u0434\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 3,\n
\"samples\": [\n          \"\\u043f\\u0435\\u0440\\u0435\\u0434\\
u043d\\u0438\\u0439 \\u043f\\u0440\\u0438\\u0432\\u043e\\u0434\",\n
\"\\u043f\\u043e\\u043b\\u043d\\u044b\\u0439 \\u043f\\u0440\\u0438\\
u0432\\u043e\\u0434\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"\\
u0420\\u0430\\u0441\\u0442\\u0430\\u043c\\u043e\\u0436\\u0435\\u043d
\\u0432 \\u041a\\u0430\\u0437\\u0430\\u0445\\u0441\\u0442\\u0430\\
u043d\\u0435\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          \"\\u041d\\u0435\\u0442\",\n          \"\\u0414\\u0430\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"\\
u0412\\u0438\\u0434 \\u0442\\u043e\\u043f\\u043b\\u0438\\u0432\\
u0430\",\n      \"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 4,\n        \"samples\": [\n          \"\\
u0431\\u0435\\u043d\\u0437\\u0438\\u043d\",\n          \"\\u0433\\
u0438\\u0431\\u0440\\u0438\\u0434\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
```

```
n    },\n    {\n      \"column\": \"\\u0426\\u0435\\u043d\\u0430\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
7473011.643419682,\n        \"min\": 10000.0,\n        \"max\":
280000000.0,\n        \"num_unique_values\": 1333,\n
\"samples\": [\n          4125000.0,\n          9950000.0\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"\\u0421\\u0440\\u0435\\u0434\\
u043d\\u043d\\u044f \\u0446\\u0435\\u043d\\u0430\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
7031030.41258355,\n        \"min\": 280000.0,\n        \"max\":
450000000.0,\n        \"num_unique_values\": 2518,\n
\"samples\": [\n          13350000.0,\n          5682000.0\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"\\
u0421\\u0441\\u044b\\u043b\\u043a\\u0430\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 18627,\n
\"samples\": [\n          \"https://kolesa.kz/a/show/112832811\",\n
\"https://kolesa.kz/a/show/112842010\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"input_data"}
```


```
categorical_columns = ['Бренд', 'Модель', 'Город', 'Кузов', 'Коробка
передач', 'Руль', 'Цвет', 'Привод', 'Растаможен в Казахстане', 'Вид
топлива']
input_data = pd.get_dummies(input_data, columns=categorical_columns,
drop_first=True)

numerical_columns = ['Объем двигателя, л', 'Пробег', 'Год']

scaler = StandardScaler()
df[numerical_columns] = scaler.fit_transform(df[numerical_columns])

input_data.head()
```


```
{"type":"dataframe","variable_name":"input_data"}
```


```
input_data = input_data.iloc[:1]

input_data = input_data.drop(columns=['Ссылка','Цена', 'Среднння
цена'])

input_data.head()
```


```
{"type":"dataframe","variable_name":"input_data"}
```


```
model = RandomForestRegressor()
model.fit(X_train_linear, y_train_linear)

RandomForestRegressor()

prediction = model.predict(input_data)
```

```python
prediction
```

```
array([2363400.])
```

```python
print("Predicted price:", prediction[0])
```

```
Predicted price: 2363400.0
```