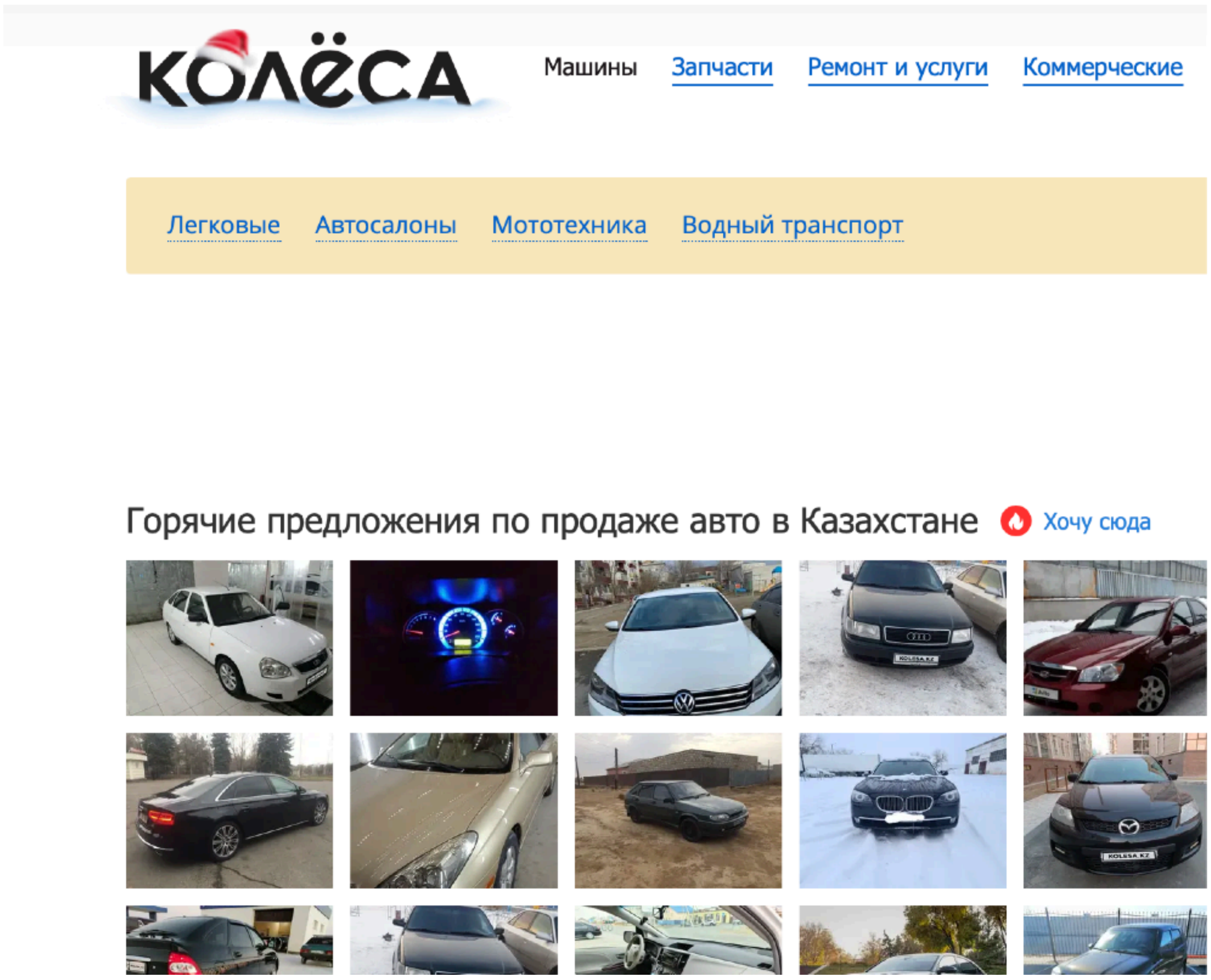
MACHINE LEARNING FINAL PROJECT



Project Proposal

Prepared by: Serik Birzhan, Bexapin Hakim

23 December 2020

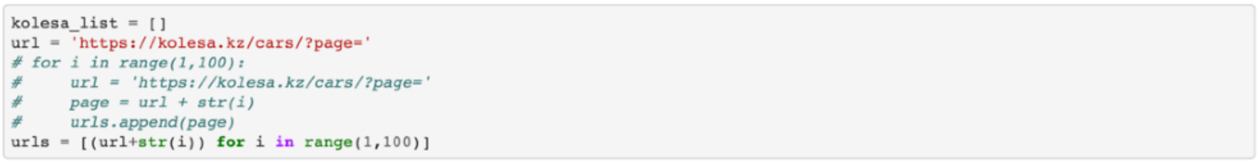
Website for car price prediction and finding good offers in kolesa.kz

INTRODUCTION

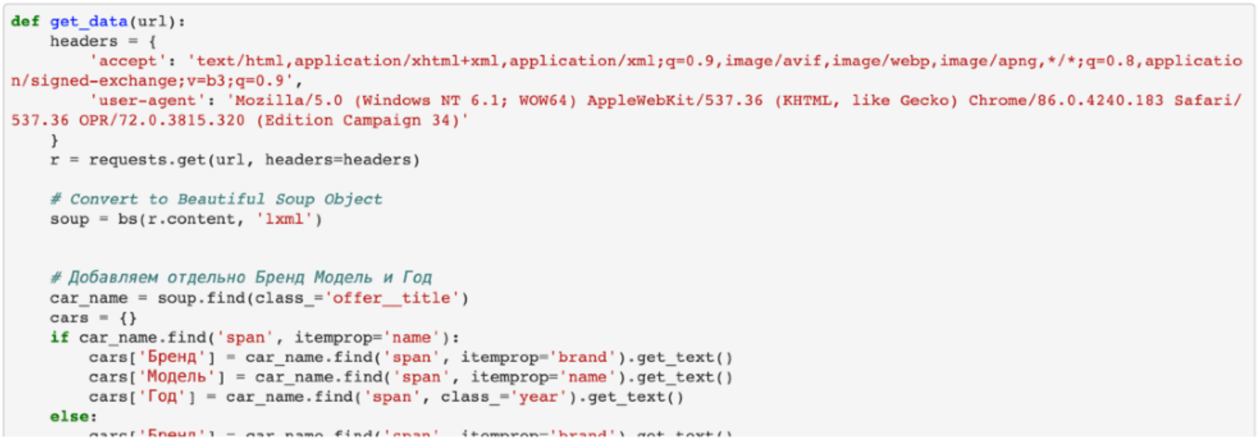
Car selling is one of the largest markets in the world. Nowadays, a majority of cars are selling on websites. The most popular website in Kazakhstan is kolesa.kz. There you can search and publish your car.
Our team was wondered that there are no platforms which analyze a car market in our country and we decided to deal with it in our project.
Firstly, we thought about how we can help people in car selling industry. There are a huge amount of data which is not processed. One of the possible solutions would be to represent basic analysis on the data. Secondly, it might be useful to show the most profitable variants depending on user's filters such as price range, year, engine etc. Lastly, we could use a data in order to predict car price in case if owner can't decide.

DATASET

The first step at project development was to gather data about cars. In order to do this we started to parse kolesa.kz. At this step, there are several possible solutions. One of them was to use native javascript query selectors. However, it is much easier to get data from page source using such library is BeautifulSoup which is used for parsing HTML and XML documents. Using library's interface we can reduce an effort while data mining.



```
kolesa_list = []
url = 'https://kolesa.kz/cars/?page='
# for i in range(1,100):
#     url = 'https://kolesa.kz/cars/?page='
#     page = url + str(i)
#     urls.append(page)
urls = [(url+str(i)) for i in range(1,100)]
```



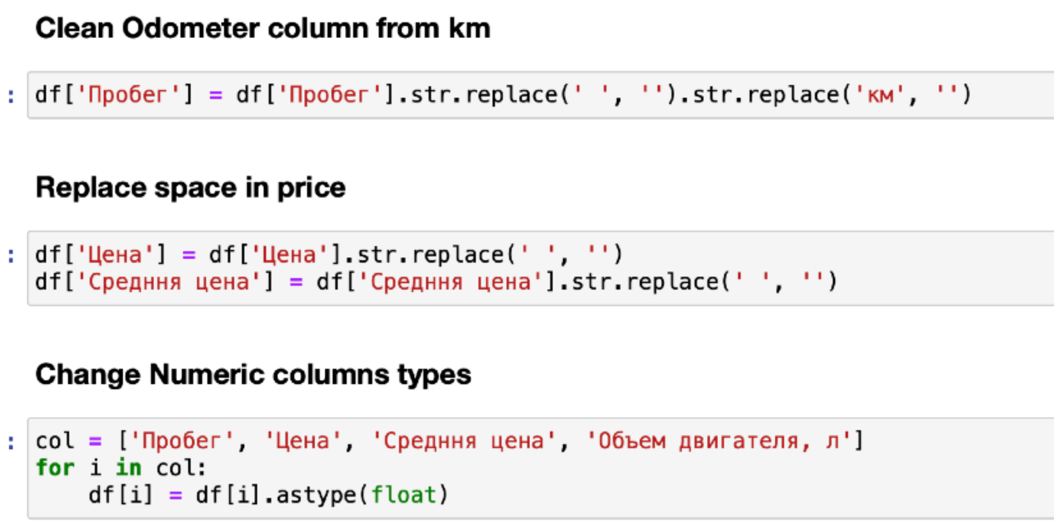
```
def get_data(url):
    headers = {
        'accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applicatio
n/signed-exchange;v=b3;q=0.9',
        'user-agent': 'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.183 Safari/
537.36 OPR/72.0.3815.320 (Edition Campaign 34)'
    }
    r = requests.get(url, headers=headers)

    # Convert to Beautiful Soup Object
    soup = bs(r.content, 'lxml')


    # Добавляем отдельно Бренд Модель и Год
    car_name = soup.find(class_='offer__title')
    cars = {}
    if car_name.find('span', itemprop='name'):
        cars['Бренд'] = car_name.find('span', itemprop='brand').get_text()
        cars['Модель'] = car_name.find('span', itemprop='name').get_text()
        cars['Год'] = car_name.find('span', class_='year').get_text()
    else:
```

At the image above you can see how we send get request at the whole pages using page argument. We opened page inspector to detect class for each field. At the end we had 19 000 rows where each row points to a published car.

DATA CLEANING

After data mining process it is crucial to set each particle in a known format. There are several steps that were made in data cleaning process:

1. Created a new column called “fuel type” by splitting last word from engine volume column.
2. Processed mileage from string to float.
3. Removed spaces from price and converted to integer.
4. Changed numeric columns using astype method for mean price column.
5. Filled mileage null values with median.



Clean Odometer column from km

```
df['Пробег'] = df['Пробег'].str.replace(' ', '').str.replace('км', '')
```

Replace space in price

```
df['Цена'] = df['Цена'].str.replace(' ', '')
df['Среднян цена'] = df['Среднян цена'].str.replace(' ', '')
```

Change Numeric columns types

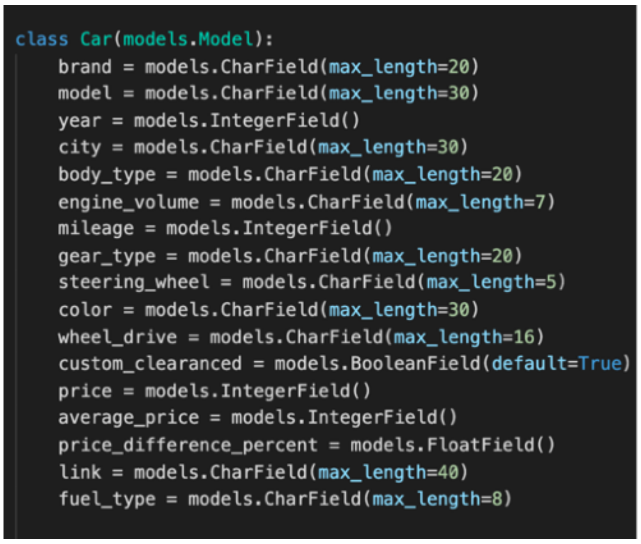
```
col = ['Пробег', 'Цена', 'Среднян цена', 'Объем двигателя, л']
for i in col:
    df[i] = df[i].astype(float)
```

PROJECT IMPLEMENTATION

The project could be implemented in 3 ways: mobile application, desktop software and website. We decided to use the third option as it seems to be most relevant in our case. Since data analysis and prediction models will be written on python 3, it is much simpler to pick web framework which uses the same program language. In the set of web frameworks that use python we have Django and Flask. We chose Django because it has more built in features such as admin panel, simple routing and localization.

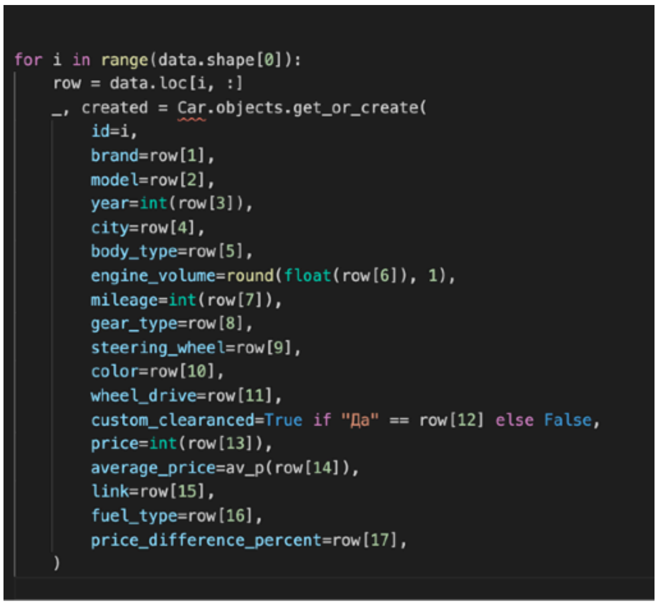
Consequently, we faced with new task of how to store data. There are huge amount of data base management systems, however we picked sqlite3 as it is default dbms for Django and very simple in configuration and dml statements. Definitely, sqlite has disadvantages such as it can't support several servers in way as Postgres can do.

A car model was created to hold data for each row in object. The model has such fields:



```
class Car(models.Model):
    brand = models.CharField(max_length=20)
    model = models.CharField(max_length=30)
    year = models.IntegerField()
    city = models.CharField(max_length=30)
    body_type = models.CharField(max_length=20)
    engine_volume = models.CharField(max_length=7)
    mileage = models.IntegerField()
    gear_type = models.CharField(max_length=20)
    steering_wheel = models.CharField(max_length=5)
    color = models.CharField(max_length=30)
    wheel_drive = models.CharField(max_length=16)
    custom_clearanced = models.BooleanField(default=True)
    price = models.IntegerField()
    average_price = models.IntegerField()
    price_difference_percent = models.FloatField()
    link = models.CharField(max_length=40)
    fuel_type = models.CharField(max_length=8)
```

The data insertion into database was made using simple script which parses through csv file and creates new Car object. The script is given below.



```
for i in range(data.shape[0]):
    row = data.loc[i, :]
    _, created = Car.objects.get_or_create(
        id=i,
        brand=row[1],
        model=row[2],
        year=int(row[3]),
        city=row[4],
        body_type=row[5],
        engine_volume=round(float(row[6]), 1),
        mileage=int(row[7]),
        gear_type=row[8],
        steering_wheel=row[9],
        color=row[10],
        wheel_drive=row[11],
        custom_clearanced=True if "Да" == row[12] else False,
        price=int(row[13]),
        average_price=av_p(row[14]),
        link=row[15],
        fuel_type=row[16],
        price_difference_percent=row[17],
    )
```

After installing Django project, configuring static files, setting up routes, we created app called Car. This app is dedicated for returning overall and specific data regarding cars. This app contains such urls:

1. / - root path, processed by index function which illustrates all cars in the database.
2. /<id> - unique path with car id which shows information about specific car.
3. /find-car - page which filters data on users queries and shows the most relevant variants.
4. /predict-car - url, responsible for gathering data about users car and predicting it’s price.
5. /analysis - some insights regarding to dataset.

Despite these routes we have such static pages from root path:

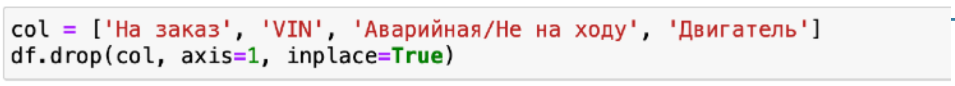
1. / - index page with welcome message.
2. /about - a brief information about project, authors and GitHub link.

TEMPLATES AND STYLING

In this project we used bootstrap framework with predefined css stylings and font awesome icons.

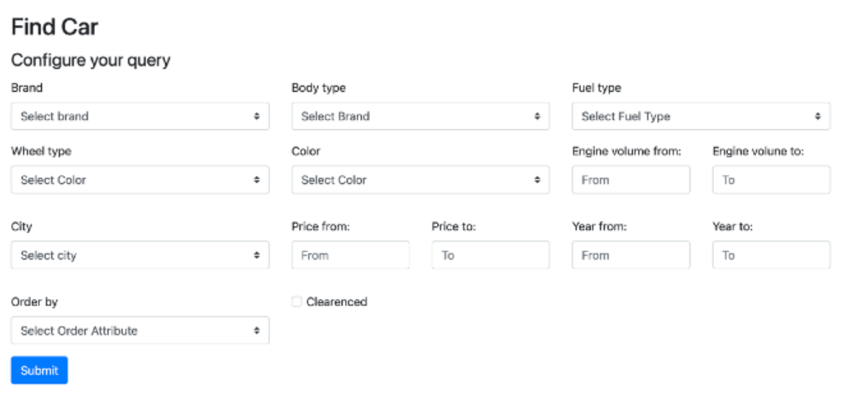
SHOWING PROFITABLE OPTIONS

As it was discussed in introduction section, one of the main features of our website was to show possible beneficial cars. We argued that dataset will be filtered from inappropriate cars with defects, crashes and other invalid sets. Because of this we dropped records with such values.



```
col = ['На заказ', 'VIN', 'Аварийная/Не на ходу', 'Двигатель']
df.drop(col, axis=1, inplace=True)
```

Next, added input fields and selectors in case if user wants to scope data by price, year or other fields. We added pointers to that fields and filtered them in case if they were presented.



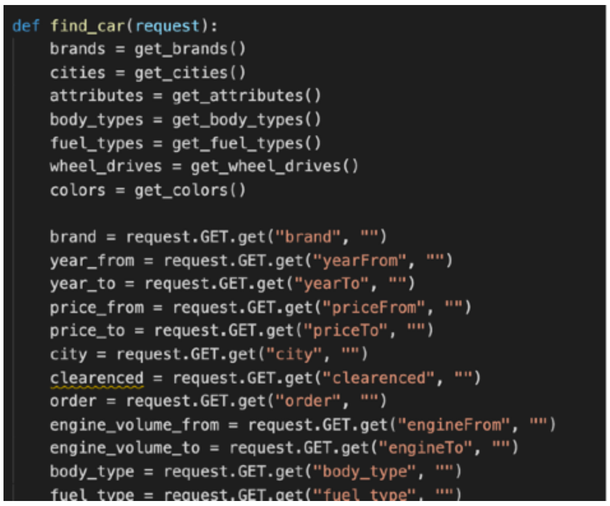
Find Car

Configure your query

Brand: Select brand	Body type: Select Brand	Fuel type: Select Fuel Type	
Wheel type: Select Color	Color: Select Color	Engine volume from: From	Engine volune to: To
City: Select city	Price from: From / Price to: To	Year from: From	Year to: To
Order by: Select Order Attribute	Clearenced		

Submit

Controller for car search:



```
def find_car(request):
    brands = get_brands()
    cities = get_cities()
    attributes = get_attributes()
    body_types = get_body_types()
    fuel_types = get_fuel_types()
    wheel_drives = get_wheel_drives()
    colors = get_colors()

    brand = request.GET.get("brand", "")
    year_from = request.GET.get("yearFrom", "")
    year_to = request.GET.get("yearTo", "")
    price_from = request.GET.get("priceFrom", "")
    price_to = request.GET.get("priceTo", "")
    city = request.GET.get("city", "")
    clearenced = request.GET.get("clearenced", "")
    order = request.GET.get("order", "")
    engine_volume_from = request.GET.get("engineFrom", "")
    engine_volume_to = request.GET.get("engineTo", "")
    body_type = request.GET.get("body_type", "")
    fuel_type = request.GET.get("fuel_type", "")
```

The functions like get_brands, get_cities, get_attributes and others return unique values for that field in order to fill options.

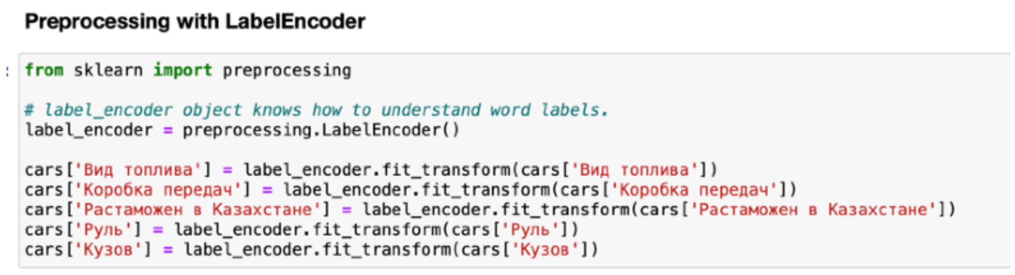
Lastly, we calculated a percentage difference between actual price and mean price and sorted them in descending order such that users see the most profitable ones at first rows.

PRICE PREDICTION

The second most important feature of this project was to predict price for user typed car. Like in the last feature we added input fields to get data about car fields (year, brand, model etc). After getting data from user (if and only if all fields were properly filled) we run our prediction model.

Preprocessing

However, we don’t have any price prediction model yet. Firstly, we need to preprocess data. As we finished data cleaning part, we don’t have any missing values. In addition, all numerical data was converted into appropriate data type. The last thing to do was to convert string values into categorical ones. We used label encoding technique where each unique value was assigned to some integer.


Preprocessing with LabelEncoder

```
from sklearn import preprocessing

# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

cars['Вид топлива'] = label_encoder.fit_transform(cars['Вид топлива'])
cars['Коробка передач'] = label_encoder.fit_transform(cars['Коробка передач'])
cars['Растаможен в Казахстане'] = label_encoder.fit_transform(cars['Растаможен в Казахстане'])
cars['Руль'] = label_encoder.fit_transform(cars['Руль'])
cars['Кузов'] = label_encoder.fit_transform(cars['Кузов'])
```

At this step, our data is ready to be used in machine learning models. Before this action, we splitted data into training and testing sets in 1/5 ratio as we have enough amount of data.

The model selection process in such manner: we picked algorithms and tested them on our dataset to find out the with the highest performance. We used variance thershold in feature selection to use fields with the highest relationship with price. The result for each prediction model is illustrated below.

Linear Regression

```
from sklearn import linear_model
model = linear_model.LinearRegression()
model.fit(X_train, y_train)
# Feature selected fit
model_s = linear_model.LinearRegression()
model_s.fit(X_train_select, y_train_select)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
print('Accuracy of test--> ', model.score(X_test, y_test)*100)
print('Accuracy of test with Feature Selection--> ', model_s.score(X_test_select, y_test_select)*100)
```

```
Accuracy of test--> 52.601833238523966
Accuracy of test with Feature Selection--> 52.601833238523966
```

Гребневая Регрессия or Ridge Regression

```
from sklearn.linear_model import Ridge
ridge = Ridge().fit(X_train, y_train)
# Feature selected fit
ridge_s = Ridge().fit(X_train_select, y_train_select)
print('Accuracy of test--> ', ridge.score(X_test, y_test)*100)
print('Accuracy of test with Feature Selection--> ', ridge_s.score(X_test_select, y_test_select)*100)
```

```
Accuracy of test--> 52.60194047078264
Accuracy of test with Feature Selection--> 52.60194047078264
```

Lasso Regression

```
from sklearn.linear_model import Lasso
lasso = Lasso().fit(X_train, y_train)
# Feature selected fit
lasso_s = Lasso().fit(X_train_select, y_train_select)
print('Accuracy of test--> ', lasso.score(X_test, y_test)*100)
print('Accuracy of test with Feature Selection--> ', lasso_s.score(X_test_select, y_test_select)*100)
```

```
Accuracy of test--> 52.60183313914928
Accuracy of test with Feature Selection--> 52.60183313914928
```

Random Forest

```
from sklearn.ensemble import RandomForestRegressor
rfr = RandomForestRegressor(n_estimators=200, max_depth=10)
rfr.fit(X_train, y_train)
# Feature selected fit
rfr_s = RandomForestRegressor(n_estimators=200, max_depth=10)
rfr_s.fit(X_train_select, y_train_select)
print('Accuracy of test--> ', rfr.score(X_test, y_test)*100)
print('Accuracy of test with Feature Selection--> ', rfr_s.score(X_test_select, y_test_select)*100)
```

```
Accuracy of test--> 87.99652261620251
Accuracy of test with Feature Selection--> 88.32700052816551
```

Gradient Boosting Regressor

```
from sklearn.ensemble import GradientBoostingRegressor
GBR = GradientBoostingRegressor(n_estimators=200, max_depth=10)
GBR.fit(X_train, y_train)
# Feature selected fit
GBR_s = GradientBoostingRegressor(n_estimators=200, max_depth=10)
GBR_s.fit(X_train_select, y_train_select)
print('Accuracy of test--> ', GBR.score(X_test, y_test)*100)
print('Accuracy of test with Feature Selection--> ', GBR_s.score(X_test_select, y_test_select)*100)
```

```
Accuracy of test--> 86.58932859409079
Accuracy of test with Feature Selection--> 86.32415140107993
```

KNeighborsRegressor

```
from sklearn.neighbors import KNeighborsRegressor
neigh = KNeighborsRegressor(n_neighbors=5)
neigh.fit(X_train, y_train)
# Feature selected fit
neigh_s = KNeighborsRegressor(n_neighbors=5)
neigh_s.fit(X_train, y_train)
print('Accuracy of test--> ', neigh.score(X_test, y_test)*100)
print('Accuracy of test with Feature Selection--> ', neigh_s.score(X_test_select, y_test_select)*100)
```

```
Accuracy of test--> 59.013380206261466
Accuracy of test with Feature Selection--> 59.013380206261466
```

Decision Tree Regression

```
from sklearn.tree import DecisionTreeRegressor
d_tree = DecisionTreeRegressor(max_depth=10)
d_tree.fit(X_train, y_train)
# Feature selected fit
d_tree_s = DecisionTreeRegressor(max_depth=10)
d_tree_s.fit(X_train, y_train)
print('Accuracy of test--> ', d_tree.score(X_test, y_test)*100)
print('Accuracy of test with Feature Selection--> ', d_tree_s.score(X_test_select, y_test_select)*100)
```

```
Accuracy of test-->  84.56205772860307
Accuracy of test with Feature Selection-->  84.50300059708314
```

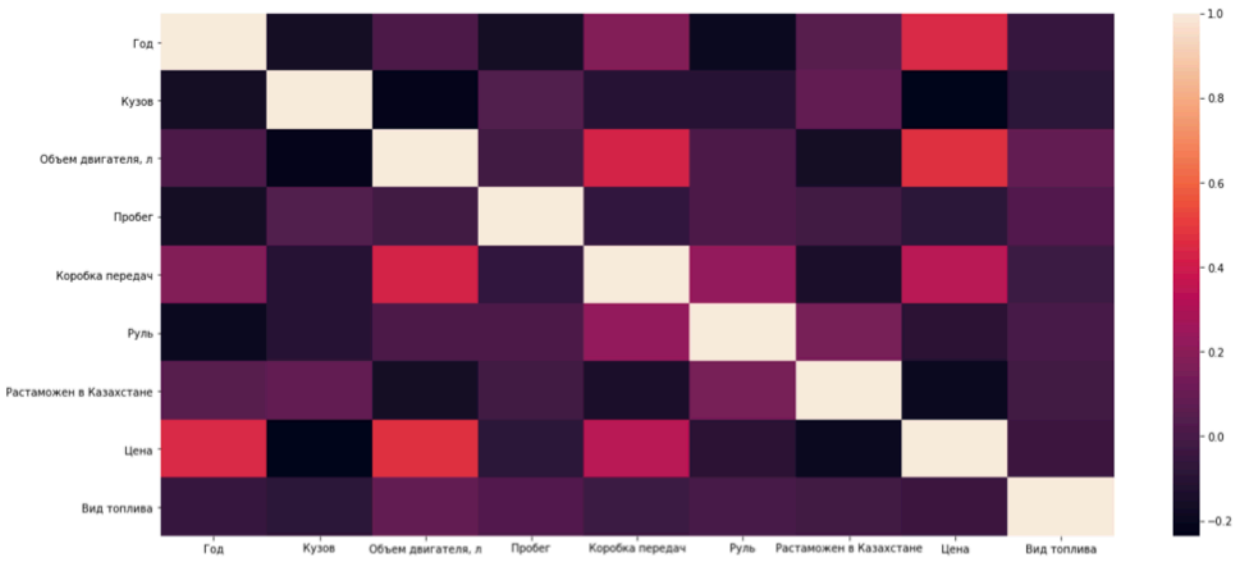
As can be seen from the accuracy test, Random Forrest regressor shows the best results.

ANALYSIS

The data analysis part was aimed to show some basic insights about dataset. We used pandas methods to show relations, distributions and dependencies between fields. For instance, we plotted heat map matrix between columns vs price.

```
plt.subplots(figsize=(20,9))
sns.heatmap(corr)
```

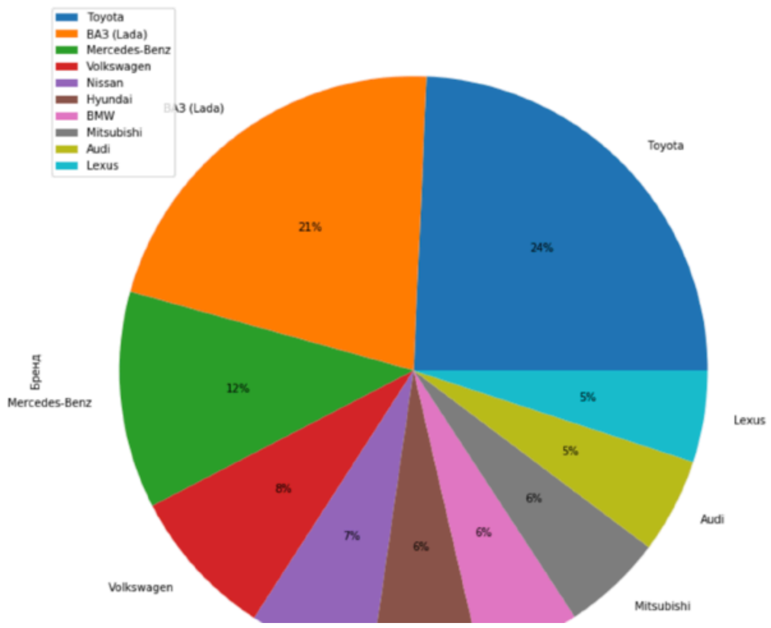
```
<matplotlib.axes._subplots.AxesSubplot at 0xbc3a0c8>
```



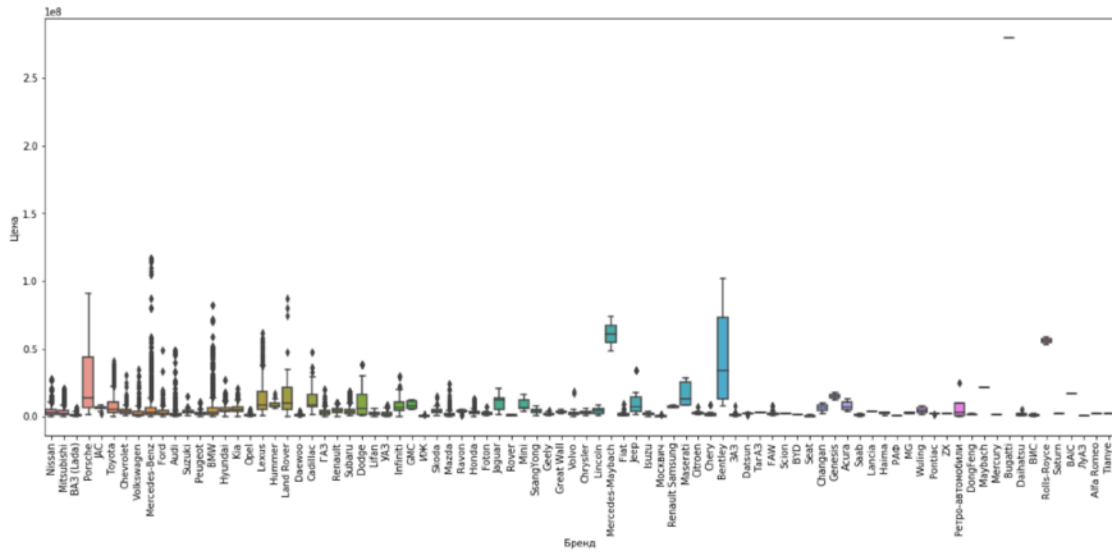
Got the top 10 brands on value counts and plot their price box plots.

```
top_10 = cars['Бренд'].value_counts().head(10)
top_10.plot.pie(figsize=(12,15), autopct='%1.0f%%')
plt.legend(loc='upper left')
```

```
<matplotlib.legend.Legend at 0xccd1808>
```



```
plt.figure(figsize=(20, 8))
sns.boxplot(x = 'Бренд', y = 'Цена', data = cars)
plt.xticks(rotation = 90)
plt.show()
```



THINGS TO COVER

1. Data validation on both view and model levels.
2. Auto refreshing data mining.
3. Auto refreshing analysis.