**Project Documentation**

**Introduction:**

With the onset of winter, the demand for the purchase of tires increases, so we decided to take tire research as the basis for the project. We decided to do a tire research on a marketplace such as Fortebank Market ( <https://market.forte.kz/categories/shiny-1113> ) since it provides large amounts of data and provides all the requirements (rows and columns). We will see various visualization graphs of the collected data and also, using machine learning models, we will see how some attributes of the tire affect its size, price, etc.

**Data collection:**

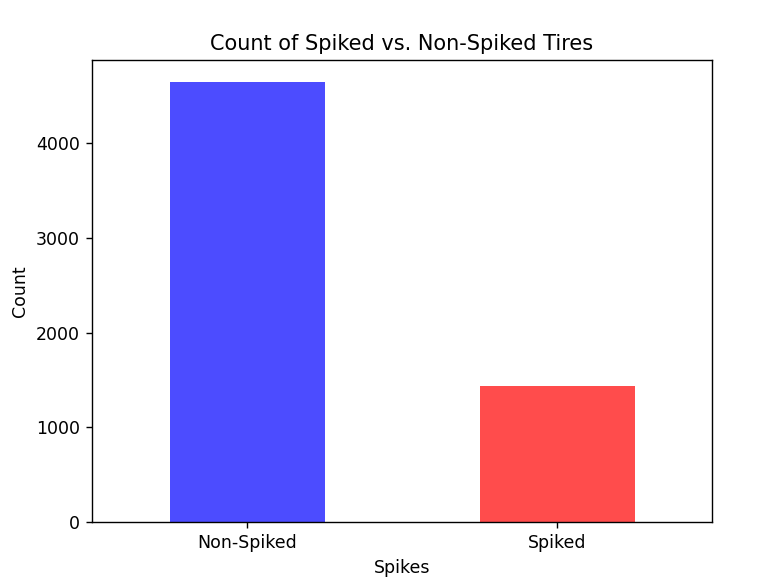
As we said above, we collected data from the fortemarket website because it provides a lot of data (about 6 thousand) and there is also a detailed description for each tire. To collect data, 2 python files were created. In the first file (tires\_link\_generator.py), a base\_URL variable is created that stores a link to the page with tires, then an empty array is created in which all links generated using the for loop will be stored. In a for loop with a range from 1 to 108, an iterable digit is added to the link in base\_URL. This creates 107 unique links that are stored in the fortebank\_tires\_links array.

The second file (forte\_tires\_parser) directly collects data. Here we mainly use 3 libraries such as selenium, beautiful soup, and pandas. Since we need a detailed description for each tire, superficial information is not enough for us. Parsing occurs this way: the program first opens one link from the array (which we created in the first file) and goes to the page with tires. On each page there are advertisements with tires, in total there are 60 advertisements with tires on each page. For each advertisement with a tire, you need to get a link to go to it and get more detailed information about it. Beautiful soup library will help us with this; it collects all links to tires from the page (60 links at a time) and adds them to the array. Next, using the for loop, the program goes through each link and collects more detailed information about the tire. The selenium library will help us collect more detailed information. Since I had previous experience working with java QA, it was possible to do without beautiful soup in parsing information using only selenium and XPath locators. Also advantage of using XPath locators is that, unlike beautiful soup, we do not need to load the entire html code, but only indicate the path to the web element. Using XPath we find such values: Tire Name, Brand, Price in ₸, Season, Spikes, Diameter, High and Width. Using the try catch statement we avoid crashing the program since some buses may not have one or two attributes and it automatically makes them NULL. Once the data is read, it is saved and added to the dataframe. After the data of all buses has been read (about 6000 thousand), the dataframe saves them in .csv and .xlsx files in the DataSets folder

**Data visualisation:**

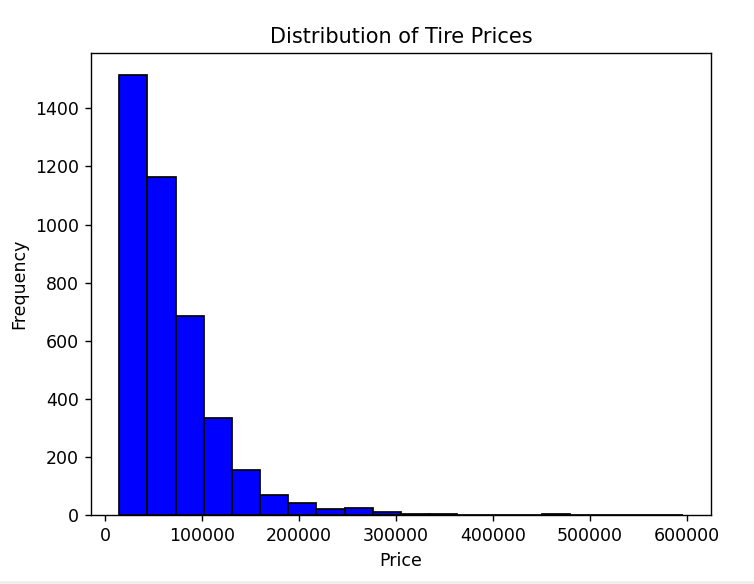
In the visualization part, we used 5 types of graphs, such as: diagram, histogram, pie chart, plot and scatterplot. In this section we will cover their purpose, what they describe and how it was did using Python libraries .

Diagram



So, in this diagram we can see statistics about how many tires are spiked and not. We clearly can see that there’s more non-spiked ones, because spikes can be effective in providing traction on icy roads, that is common occurrence on the winter, but in the any other part of year it is more right to use non-spiked one.   
 Talking about how we get to this, we used pandas library to initialize the dataset, then take values of ‘Spiked’ column as binary map, depending on the value of cell(0 or 1), and then using value\_counts() method we used matplotlib.pyplot for constructing graphic above.

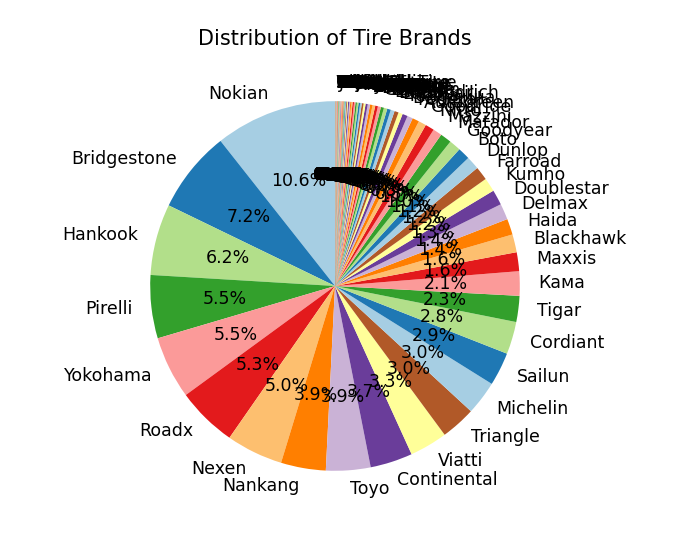
Histogram



This histogram provides to prices distribution. As we can see, most average price for tire is from 10 000 to 100 000 tenge.

Method of doing this histogram is even simpler than diagram, we just take ‘Price’ column of dataset and then create with hist() method corresponding graphic.

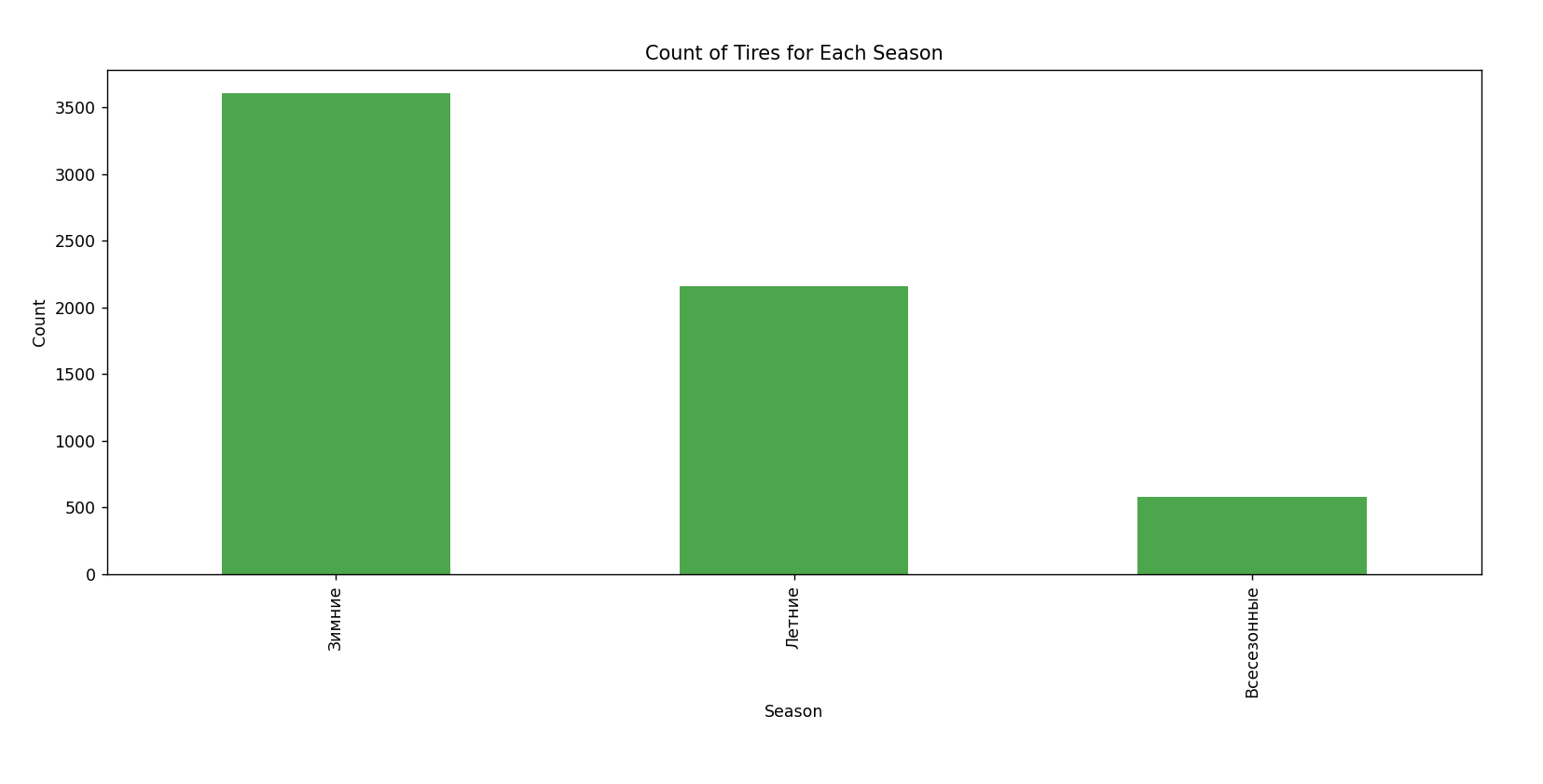
Pie chart



This pie chart shows us the frequency of appearance of certain brands, and their rating according to the percentage of appearance.

We get to this again using value\_counts() method, then pie() method in matplotlib.pyplot, using count of brands and their names, retrieving from the ‘Brands’ column of our dataset.

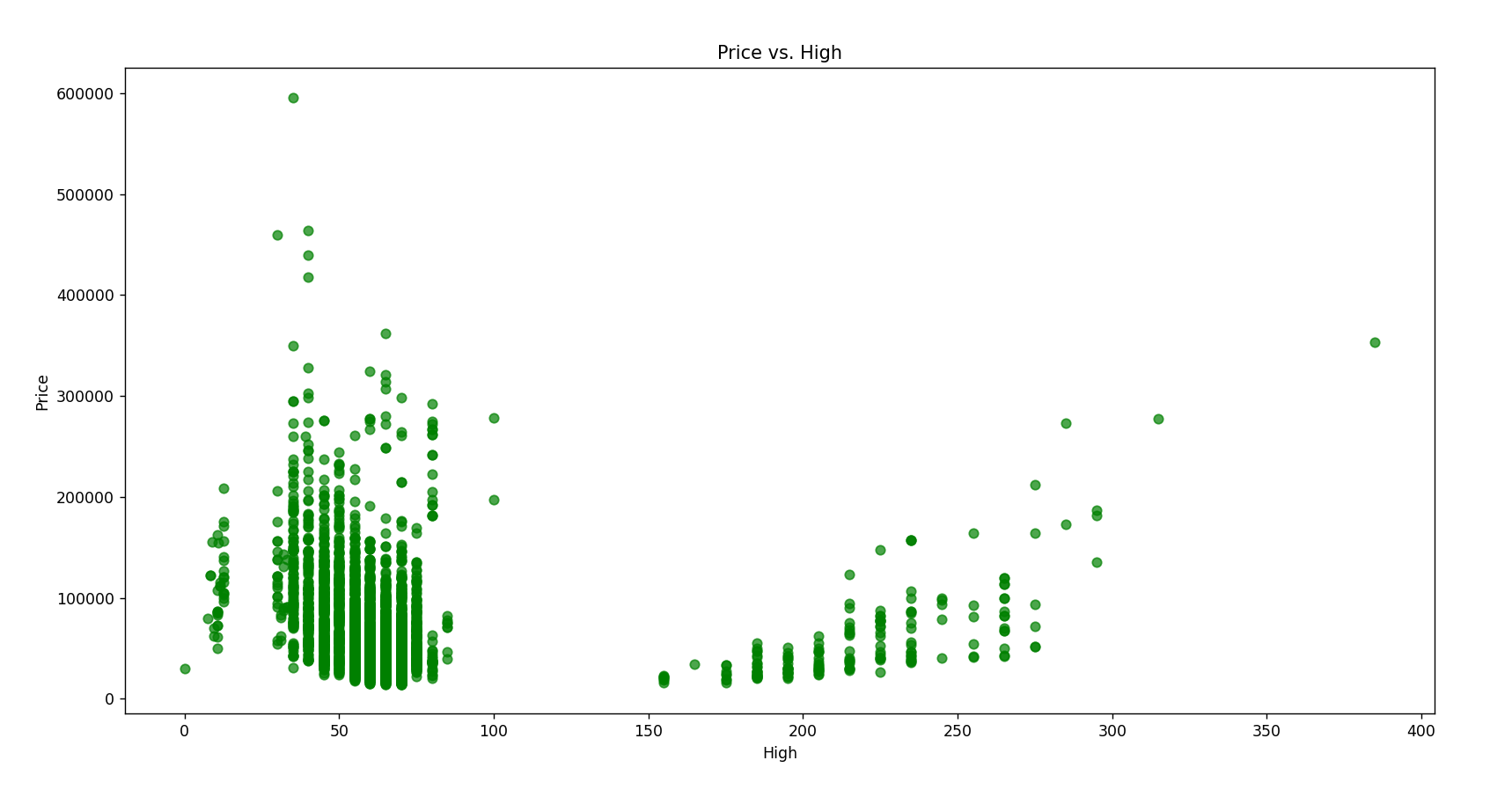
Plot



This plot provides us, which type of tires are more or less popular. As we can see, winter tires more popular now, because it is already winter here. Then going summer, and then all-season tires.

Method of getting to this is not particularly hard, as always, with pandas we initialize our dataset, count number of every season appearance in ‘Season’ column, and with plot() creating graphic above.

Scatterplot



This scatterplot provides us height to price ratio. As we can see, average high of tire is between 40 and 80, and price is not getting upper 200 000 tenge. There are also some exceptions, such as tires higher than 150 and prices upper than 200 000.

To get to this graphic, we used scatter() method, and take ‘High’ and ‘Price’ columns as axes.

**Data Analytics:**

**Data Preprocessing:**

After the successful collection of tire data from the Fortebank Market website, the next crucial step in our project is data preprocessing. This phase ensures that the collected data is clean, standardized, and ready for analysis and machine learning models.

**Handling Missing Values:**

Our dataset, comprising information on approximately 6,000 tires, is subject to missing values. To address this, we implemented a systematic approach. Null values in the 'Brand' column were handled by dropping certain values based on conditions and filling the remaining missing values with the mode. For 'Price', missing values were imputed with the mean. Null values in the 'Season' column were filled using forward-fill (ffill) method after dropping specific values. The 'Spikes' column, crucial for winter tire analysis, had missing values filled based on the corresponding season.

**Creating Dummy Variables:**

To enhance the machine learning readiness of our dataset, we introduced dummy variables for the 'Season' feature. This process involved one-hot encoding, creating binary vectors for each season category, making the data suitable for predictive modeling.

**Standardizing 'Spikes' Column:**

To facilitate machine learning analysis, the 'Spikes' column was converted into binary numeric values. 'Есть' (Yes) was mapped to 1, and 'Нет' (No) was mapped to 0, streamlining the representation of this categorical variable.

**Handling 'Diameter', 'High', and 'Width' Columns:**

The 'Diameter', 'High', and 'Width' columns required special attention. Null values were filled using forward-fill (ffill) method to maintain data continuity. 'Diameter' values were extracted and converted to float, ensuring numerical compatibility. 'High' values were converted to numeric, and missing values were filled with a constant value. 'Width' values were similarly converted to numeric.

**Saving Preprocessed Data:**

Once the data underwent these preprocessing steps, the resulting DataFrame was scrutinized for any remaining null values, ensuring data integrity. The final preprocessed dataset was then saved to two formats - CSV and XLSX - for easy access and compatibility with different analytical tools.

**Feature and Target Variable Definition for Linear Regression:**

The input features ('X') for the linear regression model are defined by excluding the columns 'Tire Name', 'Brand', and 'Price' from the DataFrame. The target variable ('y') is set as the 'Price' column.

**Train-Test Split:**

The dataset is split into training and testing sets using the train\_test\_split function from scikit-learn. The split ratio is 80% training and 20% testing, with a random seed for reproducibility.

**Model Initialization and Training:**

A linear regression model is initialized using the LinearRegression class from scikit-learn, and it is trained on the training set ('X\_train' and 'y\_train').

**Model Evaluation:**

The trained model is then used to predict prices on the test set ('X\_test'). The mean squared error (MSE) is calculated using the mean\_squared\_error function from scikit-learn to evaluate the performance of the linear regression model.

The calculated MSE is printed as the evaluation metric for the linear regression model.

**Classifier models using:**

In this section, various classification models, including Decision Trees, Random Forest, Support Vector Classifier (SVC), and K-Nearest Neighbors (KNN), are applied to the dataset to predict categorical outcomes. The aim is to determine the most accurate classifier for the given data.

**Model Initialization and Training:**

A list of classifiers is defined, including DecisionTreeClassifier, RandomForestClassifier, SVC, and KNeighborsClassifier. Dictionaries are initialized to store accuracy and confusion matrix results for each classifier.

**Iterating Through Classifiers:**

A loop iterates through each classifier in the list. For each classifier, it is trained on the training set ('X\_train' and 'y\_train'), and predictions are made on the test set ('X\_test')

**Evaluation Metrics:**

For each classifier, accuracy and confusion matrix are calculated using the accuracy\_score and confusion\_matrix functions from scikit-learn, respectively.

**Identifying the Best Model:**

The Random Forest classifier's superior accuracy can be attributed to its ensemble approach, combining multiple decision trees to handle non-linear relationships effectively. Its feature importance analysis aids in identifying crucial factors, and its robustness to overfitting and ability to handle missing values contribute to its adaptability. The algorithm's versatility and minimal parameter tuning further enhance its performance, making it well-suited for the tire classification task at hand.

**Visualization of models:**

**Decision Tree Classifier**

**Description:**

A Decision Tree is a supervised machine learning algorithm used for both classification and regression tasks. It works by recursively partitioning the data based on the features, creating a tree-like structure of decision nodes.

**The parameters that we used:**

`criterion`- The function used to measure the quality of a split ,default is 'gini' for Gini impurity.

`max\_depth`: The maximum depth of the tree.

`min\_samples\_split`- The minimum number of samples required to split an internal node.

`min\_samples\_leaf`- The minimum number of samples required to be at a leaf node.

**Visualization of Decision Tree:**

The decision tree is visualized using the `plot\_tree` function, which displays the hierarchical structure of decision nodes, leaf nodes, and decision criteria. Each node in the tree represents a decision based on a feature, and leaf nodes indicate the predicted class.

**Random Forest Classifier**

**Description:**

Random Forest is an ensemble learning method that constructs a multitude of decision trees during training and outputs the class that is the mode of the classification of the individual trees.

**The parameters that we used:**

`n\_estimators`- The number of trees in the forest.

`criterion`- The function used to measure the quality of a split ,default is 'gini' for Gini impurity.

`max\_depth`- The maximum depth of the trees.

`min\_samples\_split`- The minimum number of samples required to split an internal node.

`min\_samples\_leaf`- The minimum number of samples required to be at a leaf node.

**Visualization of Random Forest Classifier:**

Random Forest visualization involves displaying individual decision trees within the forest using the `plot\_tree` function. The visualizations help to understand the diversity of decision trees in the ensemble.

**Support Vector Classifier (SVC)**

**Description:**

A Support Vector Classifier is a supervised machine learning algorithm used for classification tasks. It aims to find a hyperplane that best separates classes in a high-dimensional space.

**The parameters that we used:**

`C`- Regularization parameter.

`kernel`- Specifies the kernel type to be used in the algorithm   
(e.g., 'linear', 'rbf', 'poly').

`gamma`- Kernel coefficient for 'rbf', 'poly', and 'sigmoid'.

`degree`- Degree of the polynomial kernel ('poly').

**Visualization:**

SVC visualization involves representing the decision boundary in the feature space. For instance, in a 2D feature space, the decision boundary might be a line that separates different classes.

**K-Nearest Neighbors (KNN)**

**Description:**

K-Nearest Neighbors is a algorithm that classifies a data point based on the majority class of its k-nearest neighbors in the feature space.

**The parameters that we used:**

`n\_neighbors`: Number of neighbors to consider.

`weights`: Weights assigned to neighbors  
 (e.g., 'uniform' or 'distance').

**Visualization:**

Visualization of KNN involves displaying the decision boundaries and regions of influence for different classes in the feature space. The decision boundaries show where the transition from one class to another occurs.

**These visualizations help to interpret and understand how the models make decisions based on the input features and how well they perform on the given dataset.**

**Documentation:**

Actually, everything that was presented above is documentation itself. Therefore, in this part we will tell you how the file architecture in our project is structured. In the DataSets folder we store all the data that we parse and which we will further preprocess. In the Machine Learning Algorithms folder we store a .ipynb file that can be opened in Google Colab or Jupiter notebook. All ML and statistical part is done in this file. The Visualization folder stores graphics and visualization of the dataset. The parsing and link creation part itself is stored externally in the project root. The venv folder contains libraries used in the project and an iterator for the python language