DIT824 – Software Engineering for Data-Intensive AI Applications

LETUS for used car prediction

Team 01

Team members:

George Sarkisian 930103

Majed Dalain 901207

Martin Stanchev 980605

Amjad Alshihabi 930918

Index of Contents :

Project Description

LETUS for used car prediction is a system that uses a machine learning model to predict the price of cars in Sweden based on specific features.

The machine learning model is trained from the data that is scraped from a second-hand sales website to get the best accuracy and to have up to date data. The system can also show to the admin if the machine learning model is still relevant to the new data that admin can add.

Individual Contribution Form

Members	Contribution	Signature
Amjad Alshihabi	Section 1. Stakeholders. Appendix I & II. , Data validation, database connections	
Martin Stanchev	Section 2., data cleaning,model training , frontend	
Majed Dalain	Section 1. User Stories, frontend, admin validation , model versioning	
George Sarkisian	Section 1. Functional Requirement., Section 3. Section 5, backend, restful API, setting up the pipeline, model training, feature engineering , data cleaning	
Mohammad Dergham	-	

Section 1. The concept of the AI system

LETUS for car prediction is a system that uses a machine learning model(Random forest) to predict the price of cars in Sweden based on specific features(description of the features can be found in section 2).The Random forest model is trained after cleaning the noises and outliers and performing feature engineering from the data that is scraped from a second-hand sales website to get the best accuracy and to have up to date data.

Our system will have client/server architecture where our server will run python(flask) and our client will run using javascript(react).

We are going to build a system to predict prices of cars, based on a large dataset of car sales in Sweden (subject to change). A user of the system can check the price of a car based on some features that they entered using the web page. Admins can upload more data in the admin portal to validate the model with the new data using the frontend.

- Stakeholders and Characteristics(Users):
    - A user - can check the price of a car based on basic car features(Description of the features found in Section 2).
    - The admin - can check if the machine learning model is still valid for new data and retrain the model, if required.
- User Stories:
    - Users:
        - As a person who wants to buy a used car, I want to check if the car I'm buying has a good market price.
        - As a seller of a used car, I want to check what price will be suitable for my car.
        - As a seller of a used car, I want to be able to see if the price I am offered for my car is above or below market price.
        - As a buyer of used cars, I want to see if the price that I am offered from a seller is above or below market price.
    - Admin:
        - As an administrator of the website for used car sales price prediction, I want to be able to add new data to the model and check if it is still valid for the new data.
- Functional Requirements
    - The system shall have one UI page for any users that want to check the price of a used car.
    - The system should allow users to check the price of the car based on features they input in the form.
    - The system should make a price prediction based on user-entered features.
    - The system shall have a UI page for an admin.
    - The system should be able to test the model with a new dataset uploaded by the admin.
    - The system should allow the admin to retrain the model based on all data in the database.

Section 2. Data sources

We used a website scraper created by Martin D. Larsson which can be found at https://github.com/martindlarsson/blocket_scraper .

The website scraper scraped second hand car sales from a popular Swedish website. The data we gathered has approximately 110,000 examples in raw form(Fig. 2.1), with 14 features. We will clean the data we will only used 10 main features to train the model.

Initially, we decided to remove the columns id, regnr, make_year and add_date, since they have low correlation to the model and were unnecessary based on the tests that we did. We will only keep the columns for brand name(brand), model(model), Transmission type(gear), fuel type(fuel) ,milage(milage), horsepower of the car(hp), type of the chassis(type), Place where the car was sold(geo). The target in our dataset is the price in SEK(price). We will create and use a vocabulary of car makers and models so that we can make similarly named models the same e.g. - “Golf 2.0 GTI” and “GTI Golf” are the same car models, so they both can be named “Golf GTI”.

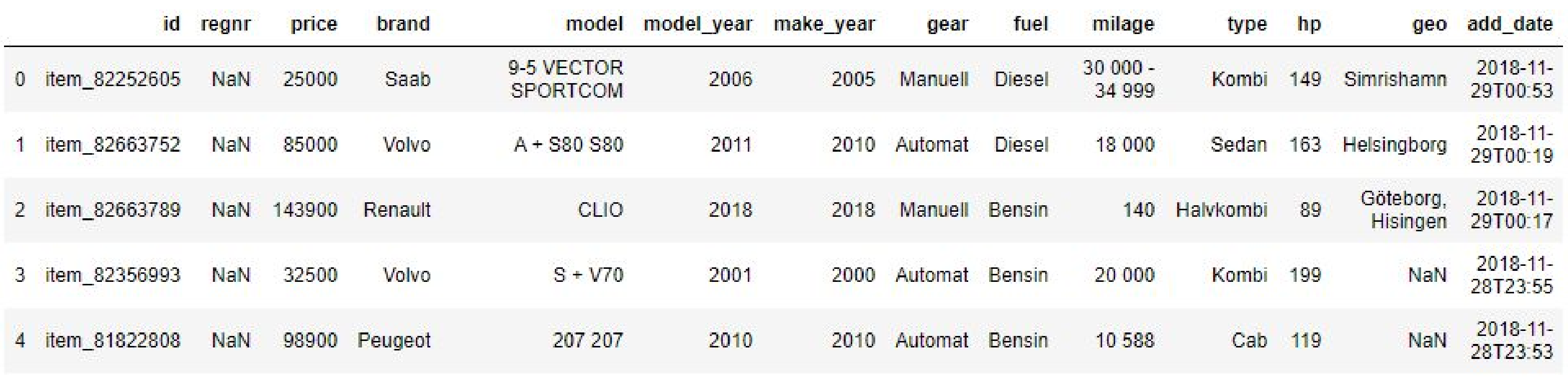
There are some numerical features which are saved as string, e.g. mileage,hp. Some values in mileage are saved as “30000 - 35000” and we will use only one of the numbers and convert it from string type to float.

We will do the same for the horsepower(hp), where we will convert the string to float then we will perform feature engineering in which we will replace null values with the median based on the brand and the model of the car.

We will also remove all examples that had empty values or null in the columns price, brand, model since does features are the critics and can’t be filled or feature engineered. We will use median value when doing feature engineering for horsepower based on the car model and brand, and for features like transmission type, fuel type,geo and car type, we used default values; default values are:

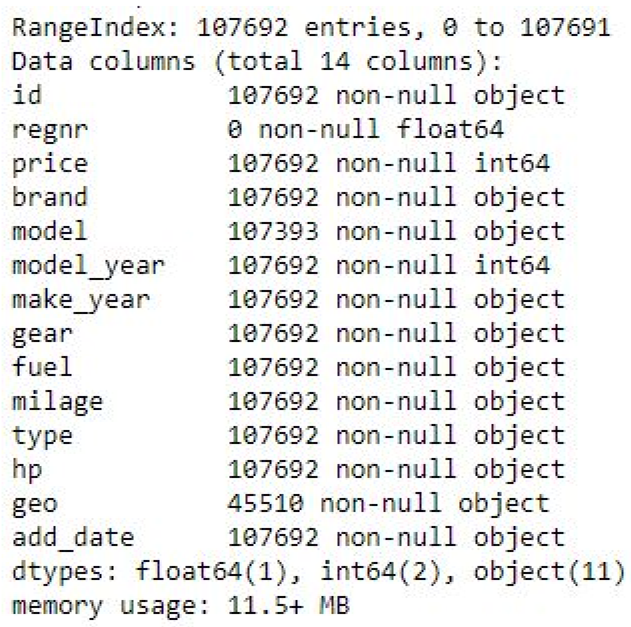
- Transmission: manuell
- fuel type: bensin
- Car type: sedan
- Geo: sweden

Since the most common type of the transmissions in sweden are manuel and most common type of fuel are bensin and most common type of the cars are sedan and since they don’t have that big correlation to the target(price) based on the testing that we did using jupyter notebook.



	id	regnr	price	brand	model	model_year	make_year	gear	fuel	milage	type	hp	geo	add_date
0	item_82252605	NaN	25000	Saab	9-5 VECTOR SPORTCOM	2006	2005	Manuell	Diesel	30 000 - 34 999	Kombi	149	Simrishamn	2018-11-29T00:53
1	item_82663752	NaN	85000	Volvo	A + S80 S80	2011	2010	Automat	Diesel	18 000	Sedan	163	Helsingborg	2018-11-29T00:19
2	item_82663789	NaN	143900	Renault	CLIO	2018	2018	Manuell	Bensin	140	Halvkombi	89	Göteborg, Hisingen	2018-11-29T00:17
3	item_82356993	NaN	32500	Volvo	S + V70	2001	2000	Automat	Bensin	20 000	Kombi	199	NaN	2018-11-28T23:55
4	item_81822808	NaN	98900	Peugeot	207 207	2010	2010	Automat	Bensin	10 588	Cab	119	NaN	2018-11-28T23:53

figure(2.1)



```
RangeIndex: 107692 entries, 0 to 107691
Data columns (total 14 columns):
id              107692 non-null object
regnr           0 non-null float64
price           107692 non-null int64
brand           107692 non-null object
model           107393 non-null object
model_year      107692 non-null int64
make_year       107692 non-null object
gear            107692 non-null object
fuel            107692 non-null object
milage          107692 non-null object
type            107692 non-null object
hp              107692 non-null object
geo             45510 non-null object
add_date        107692 non-null object
dtypes: float64(1), int64(2), object(11)
memory usage: 11.5+ MB
```

figure(2.2)

As you can see in figure (2.2) our data is full of null points. For some features the value of the feature is “-” instead of null. These values will need to be cleaned as explained above.

Section 3. Features and inference model

Our pipeline for our machine learning model consists of:

1. Extensive data cleanup: in which we read the initial csv file and we only take the columns that are relevant to our model, We also remove some features that don't have good correlation to the price, or that we deemed unfit for our model. The complete list of the features we keep and use in our model are described in section 2. and we divide the dataset into two parts:
    a. First part will be used to train the model.
    b. Second part will be used to validate the model by the admin
2. The first set of data that is generated by the pipeline in previous step is then goes through few steps:
    a. First we drop unneeded columns and we drop all the rows that are missing critical information like price, brand name or model name. Since those features have the highest dependencies to the price and cannot be filled using feature engineering
    b. The data that is generated from (2.a) is run through feature engineering pipeline in which
        i. we fill the empty rows of column geolocation “geo” to sweden
        ii. we fill the empty rows of column type of the car “type” to Sedan.
        iii. we fill the empty rows of column gear of the car “gear” to manuell.
        iv. we fill the empty rows of column fuel type “fuel” to bensin.
        v. we fill the empty rows of column horsepower “hp” with mean of the same car brand.
        vi. we correct the format of the millage, since some of them are written in format (20000 - 25000) we convert them into 20000 and then cast the string to float.
3. After the data that is run through step 1 & 2, we drop outliers :
    a. We drop everything that horsepower is less than 40 or more than 800.
    b. Drop all car prices that is less than 4000 kr. Or more then 5000000 kr.
    c. Drop all car mileage that is more than 300000.
4. After the data is cleaned we save it into our SQLite database.
5. After data has successfully saved into our database we retrive the data from the database and we perform OneHot encoder and Label encoder to convert categorical values into numerical:
    a. We perform OneHot encoder on gear and fuel, since gear only can have 2 values (manuel & automat) and fuel can have 4 values (bansin , diesel, el & haybrade).

b. We perform Label encoder on geolocation, type of the car, car model & car brand, because using one hot encoder will generate 5000 different columns and that will complicate our understanding of the data.
c. We also save the label encoder to later use them in transferring the categorical data while performing prediction.

6. After processing all the data we train our model using Random Forest regression because we tested our dataset with dummy regression , KNeighbors regression , lasso regression, Ridge regression, ElasticNet regression, GradienBoosteing regression , Random Forest regression, ExtraTrees regression and the Random Forest regression gave the best prediction (R-squared) . See figure 3.1

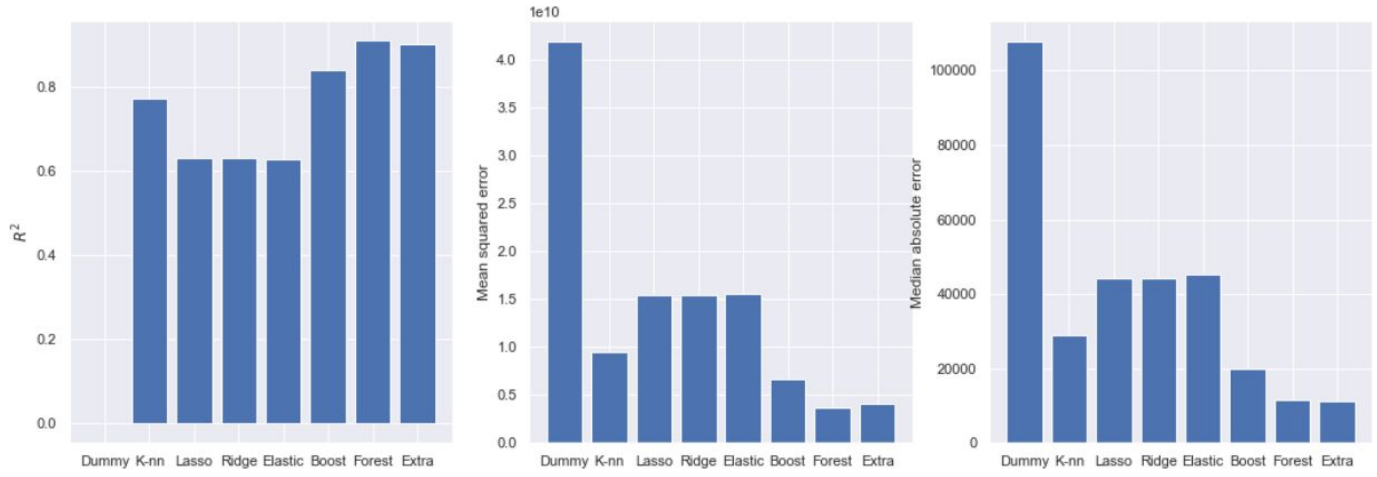


Figure 3.1

You can see and read more about our experiments and in our jupyter notebook that is appended to this document.

When doing a prediction on data entered by a user, we only use the features car maker, car model, mileage, model year, transmission and fuel type. Since the feature horsepower has a high correlation, we take the mean value for the model and use that, but we don’t have it as an option for the user to enter, as we decided it is not good for the user experience and the accuracy was good enough. We use the same label encoders that we used in the training step for car make and car model to convert the string type categorical feature to a numerical one as that is required by the machine learning model.

In our pipeline, it is possible to retrain the machine learning model using data that we have in our database from the admin page. In which admin will choose two different date, and our system will retain the model based on all the data that has been added to the database between those dates. Once the admin chooses to train the model, our pipeline will be triggered starting from step 5 and 6

Section 4. Production system

The system is divided into three components; the UI, the scripts component and the service component.

The UI component contains the web application frontend for both users, the customer and the admin. The frontend is built with react and javascript. We have two pages one for users and one for the admin.

The service component contains the backend. The backend application starts from “app.py” file that perform as a server. We use a RESTfull API to handle the requests coming from the frontend. Here is a list of different API method we use for the backend:

1. Get (api/predict): this method is used to make predictions on a car price. It takes in car information like brand, model, model year, mileage, transmission and fuel type. It returns a prediction for the given car information.
2. Post (api/validate): The method is used to check whether the trained and deployed model we have still valid for new entries. The method takes in a .csv file, it validates the data in the file and returns the MAE and R2 for the passed data as well as for the data that is already registered in the database. The method produces two chart images and saves them to the “dataset” folder.
3. Post (api/retrain): The method is used to perform dynamic training, it takes in two datetime instances to train the machine learning model on the dataset that are registered to the database between the two given datetime instances. The new machine learning model will be saved to the “datasets” folder to be then deployed to the web application, mobel and metadata if the model is also saved in Neptune tool.
4. Get(api/experiments): The method is used to get all experiments that are saved by training the model, in order to make comparison between them so the admin can have a better understanding of different models.

The last component in our system is the script(model) component, it contains the functions needed to handle and process the dataset entries to prepare it before using it in training the machine learning model. Following the main modules in the script component:

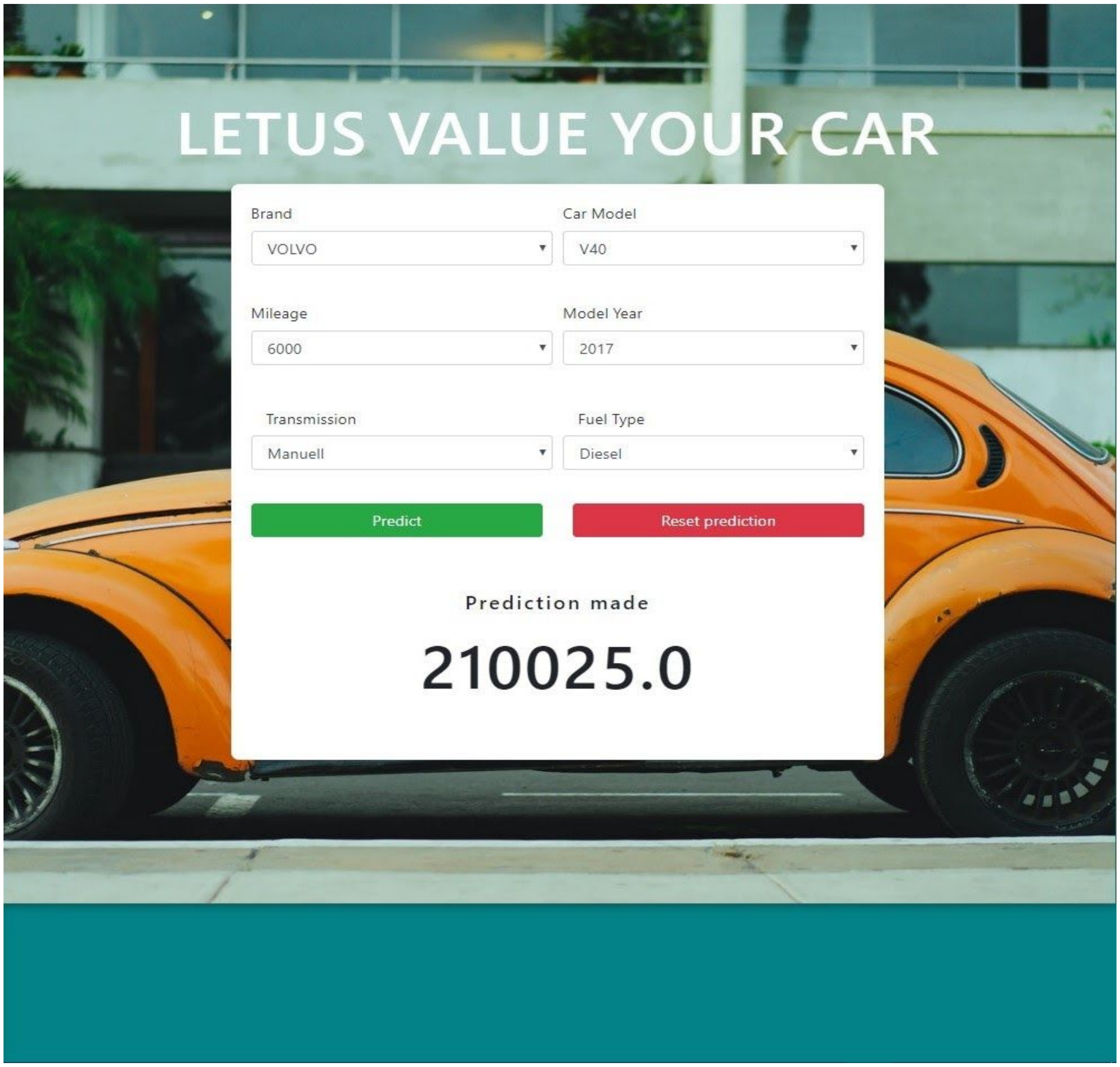
1. Data_cleaner (clean_up_dataset.py): This subcomponent is responsible for handling the dataset set. It basically handles the outliers, removes entries with many missing values, removes the columns that are not appropriate for the data validation schema and at last it does one hot encoding for categorical features.
2. csv2sqlite: This subcomponent is mainly responsible for handling the database, it contains many functions for writing and reading from the database using an sqlite3 library for the creation of the database and handling the database connection and retrieving data with different queries.
3. data_validator.py: Here we have the data validation, we used pandas_schema library that provides methods for data validation where can create a validation schema and specify different constraints for each column. to validate the dataset we pass it as a pandas data frame and it return true for valid dataset and false for invalid ones along with the detected error for each row in the data frame.
4. Controller.py: This component is the brain of our model, It is triggered when there isn't any model in our server and it runs the pipeline that is described in section 3.
5. MLModel.py: This component responsible for training the model and saving the model in both database and in the neptune.(neptune is a tool we use in versioning the model).

Extra libraries or Tools:

Neptune.ml: it is experiment management tool that allows tracking the entire experimentation process. We used neptune.ml in order to log properties, metrics, and the model. We used the library to fetch all experiments also so we can give the admin the ability to compare different experiments. Further improvements might be versioning the Data and the source code.
https://neptune.ml/

Screenshots for the use of the System:

1. **User enters the input to get the predicted car price (access point Localhost:5000):** the user is asked to enter car brand, model, mileage, model year, transmission and fuel type. If there is not any model exist, the system will create a new one, this might take some time due to creation and saving of the model.

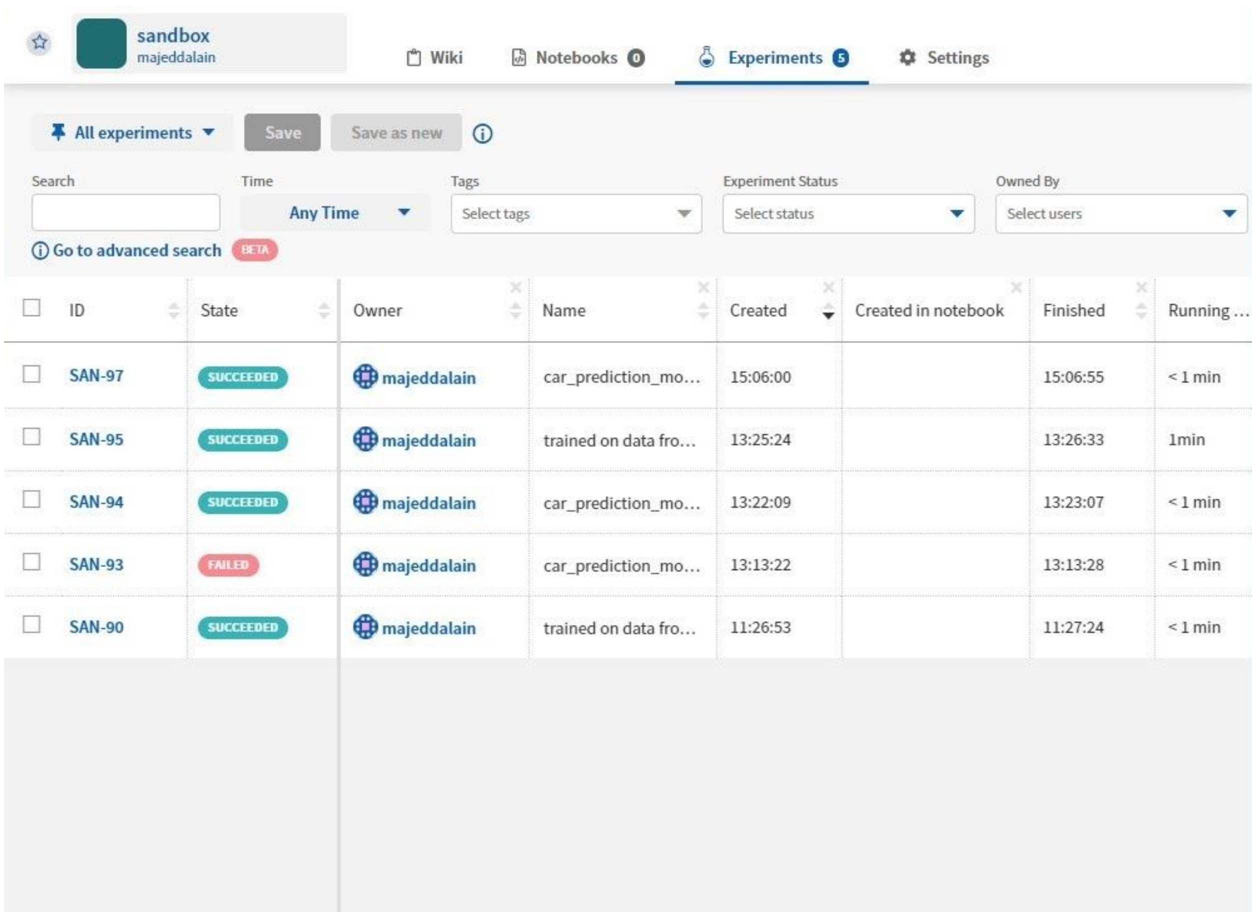


Brand: VOLVO
Car Model: V40
Mileage: 6000
Model Year: 2017
Transmission: Manuell
Fuel Type: Diesel

Prediction made

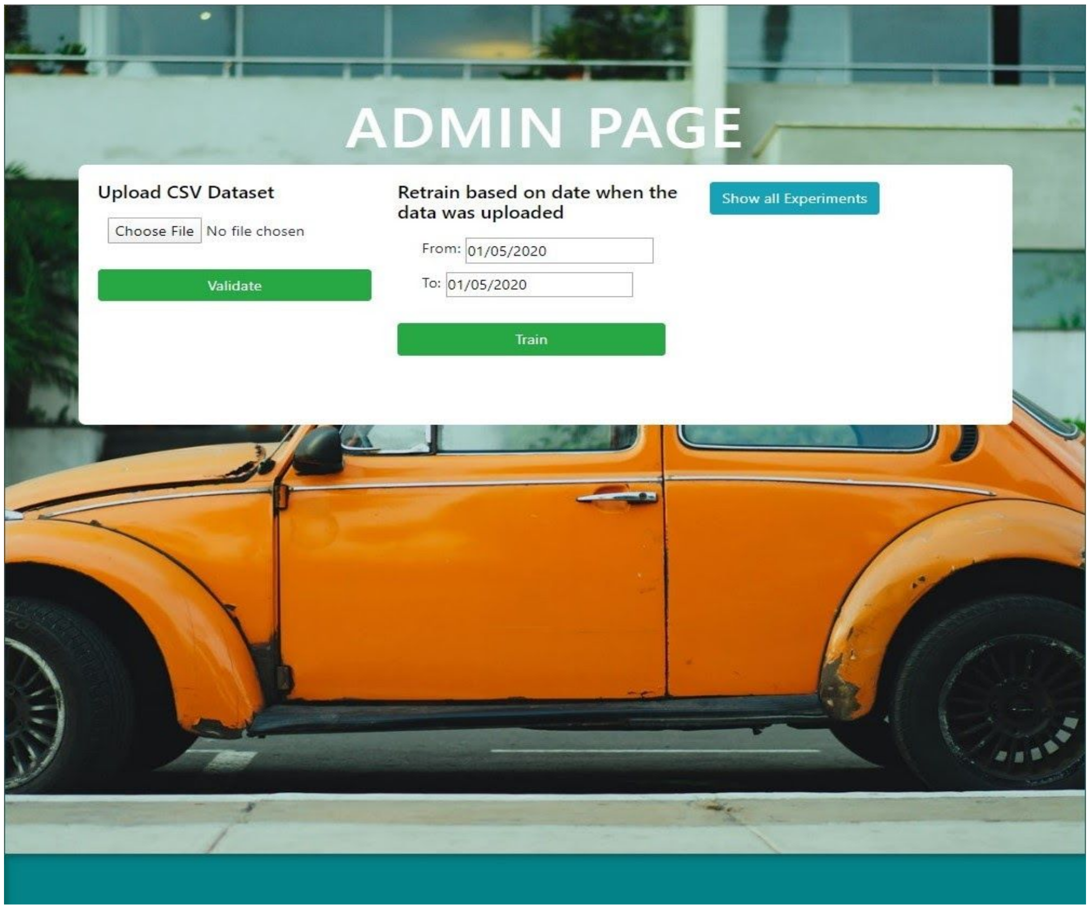
210025.0

2. **Versioning all experiments to the Neptune tool:** whenever a new model(experiment) is created, it is directly saved into the Neptune tool. This feature is used only by admin, the new models are created either initially for the first prediction made by the user (if there is no mode exist yet) or whenever the admin retrain the model based on two time points.

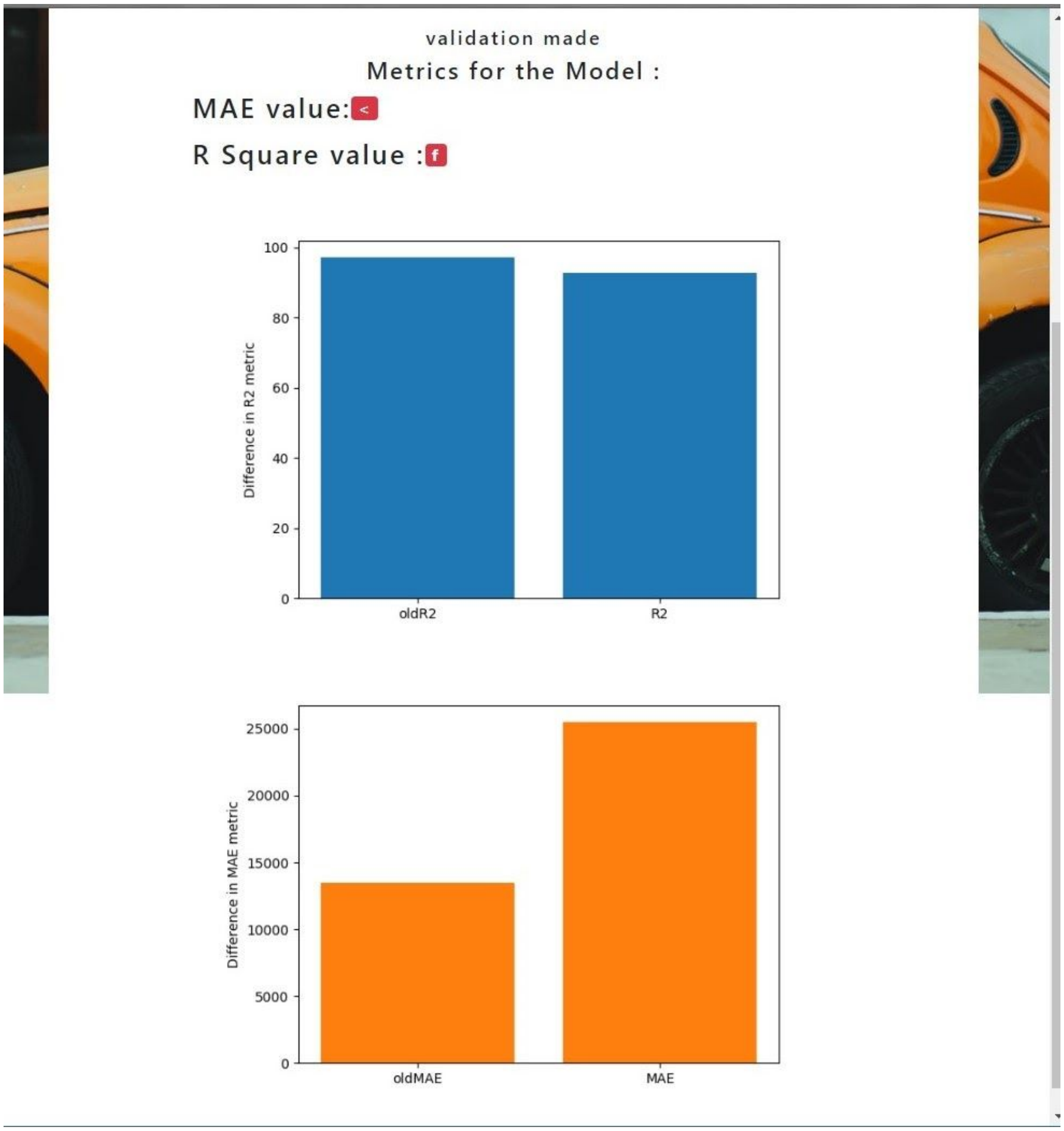


ID	State	Owner	Name	Created	Created in notebook	Finished	Running ...
SAN-97	SUCCEEDED	majeddalain	car_prediction_mo...	15:06:00		15:06:55	< 1 min
SAN-95	SUCCEEDED	majeddalain	trained on data fro...	13:25:24		13:26:33	1min
SAN-94	SUCCEEDED	majeddalain	car_prediction_mo...	13:22:09		13:23:07	< 1 min
SAN-93	FAILED	majeddalain	car_prediction_mo...	13:13:22		13:13:28	< 1 min
SAN-90	SUCCEEDED	majeddalain	trained on data fro...	11:26:53		11:27:24	< 1 min

3. **The Admin page (access point Localhost:5000/admin):** admin can validate the model, this means admin can upload a csv file to check if the model is still valid for the data that is added through the csv file. Also, the admin can retrain the model on a batch of the same dataset but between two specific time stamps.



4. **Validate new dataset:** the admin can upload a new dataset and the system will check if the model is still valid/suitable for the uploaded dataset. The system will return charts comparing the metrics (R-square & MAE) for the old and new(uploaded) dataset. If the uploaded dataset is invalid, admin will get an error.


validation made

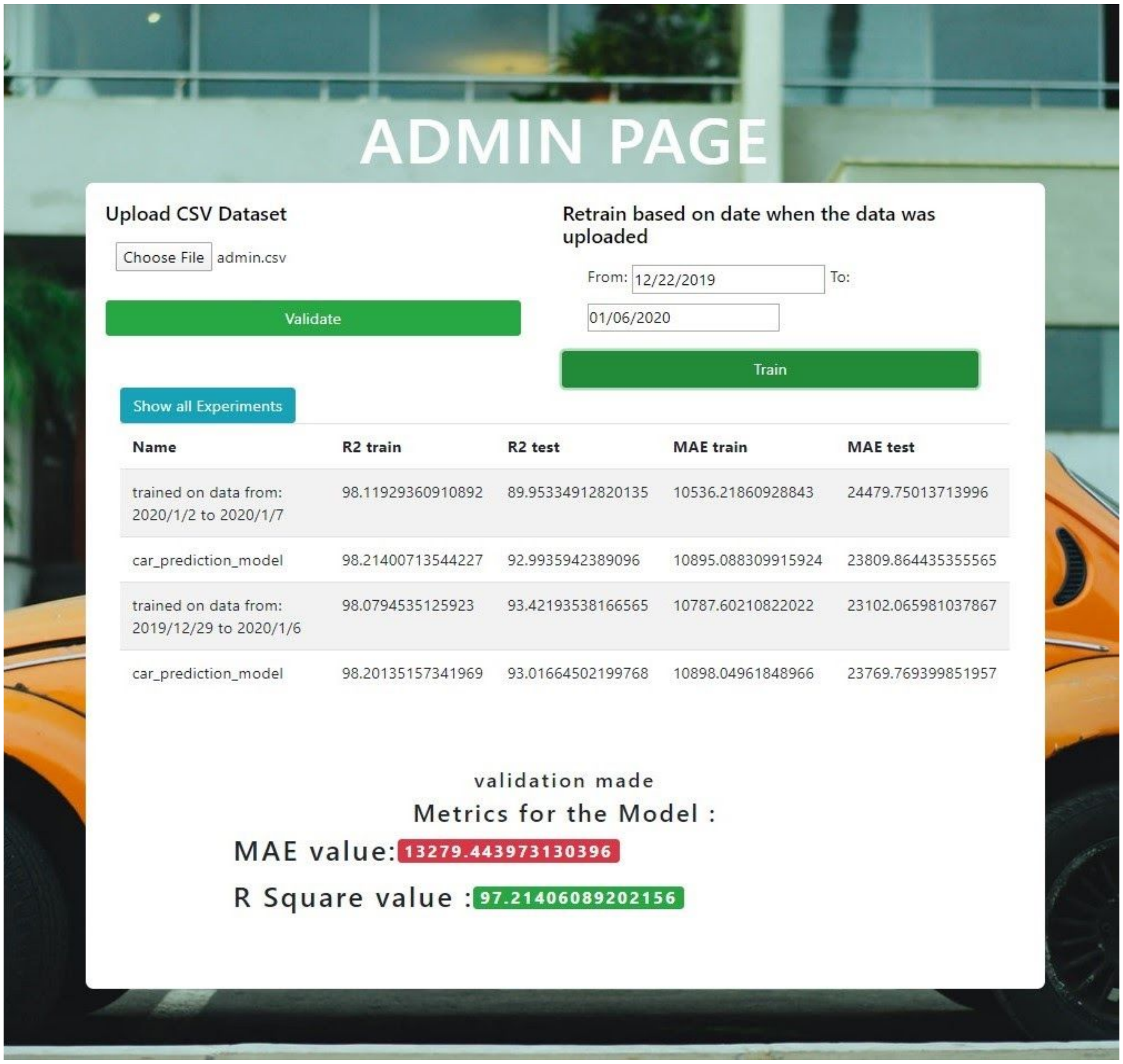
Metrics for the Model :

MAE value:<

R Square value :f

5. **Retrain the model:** admin car retrain the model between two timestamps, the trained model will be added to the Neptune and the admin can get all the info such as metric values, and the model name. Important to notice that red values represent a warning, while green values represent acceptable values.

   The unexceptable values are 12000 < MAE & 95 > R-square



From: 12/22/2019 To: 01/06/2020

Name	R2 train	R2 test	MAE train	MAE test
trained on data from: 2020/1/2 to 2020/1/7	98.11929360910892	89.95334912820135	10536.21860928843	24479.75013713996
car_prediction_model	98.21400713544227	92.9935942389096	10895.088309915924	23809.864435355565
trained on data from: 2019/12/29 to 2020/1/6	98.0794535125923	93.42193538166565	10787.60210822022	23102.065981037867
car_prediction_model	98.20135157341969	93.01664502199768	10898.04961848966	23769.769399851957

validation made

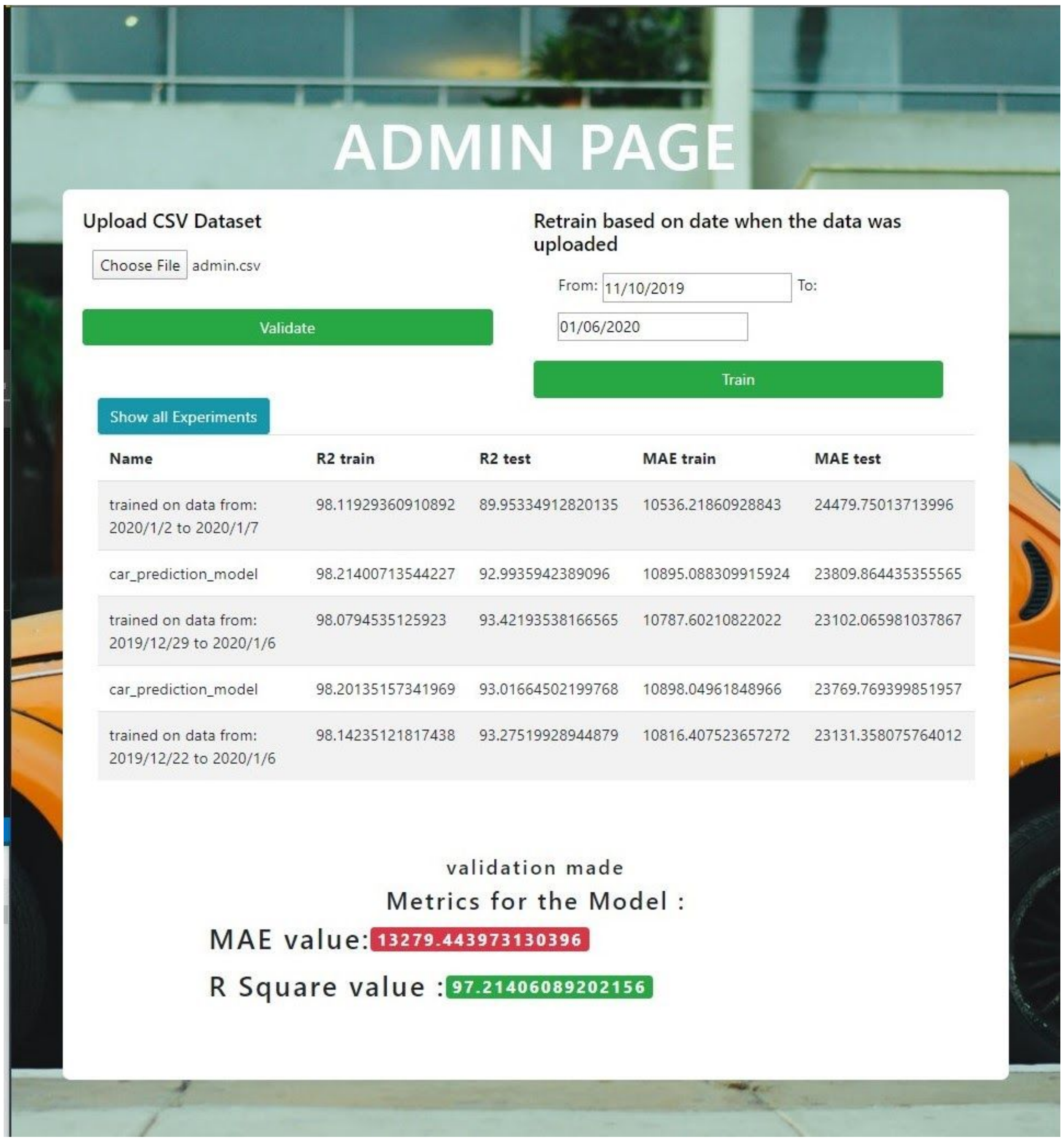
Metrics for the Model :

MAE value: 13279.443973130396

R Square value : 97.21406089202156

6. **Show all experiments:** the admin can get all the info about saved models by clicking the button show all experiments.

   The information are: what data model has been trained, R-square & MAE for both training and test data



admin.csv

From: 11/10/2019 To: 01/06/2020

Name	R2 train	R2 test	MAE train	MAE test
trained on data from: 2020/1/2 to 2020/1/7	98.11929360910892	89.95334912820135	10536.21860928843	24479.75013713996
car_prediction_model	98.21400713544227	92.9935942389096	10895.088309915924	23809.864435355565
trained on data from: 2019/12/29 to 2020/1/6	98.0794535125923	93.42193538166565	10787.60210822022	23102.065981037867
car_prediction_model	98.20135157341969	93.01664502199768	10898.04961848966	23769.769399851957
trained on data from: 2019/12/22 to 2020/1/6	98.14235121817438	93.27519928944879	10816.407523657272	23131.358075764012

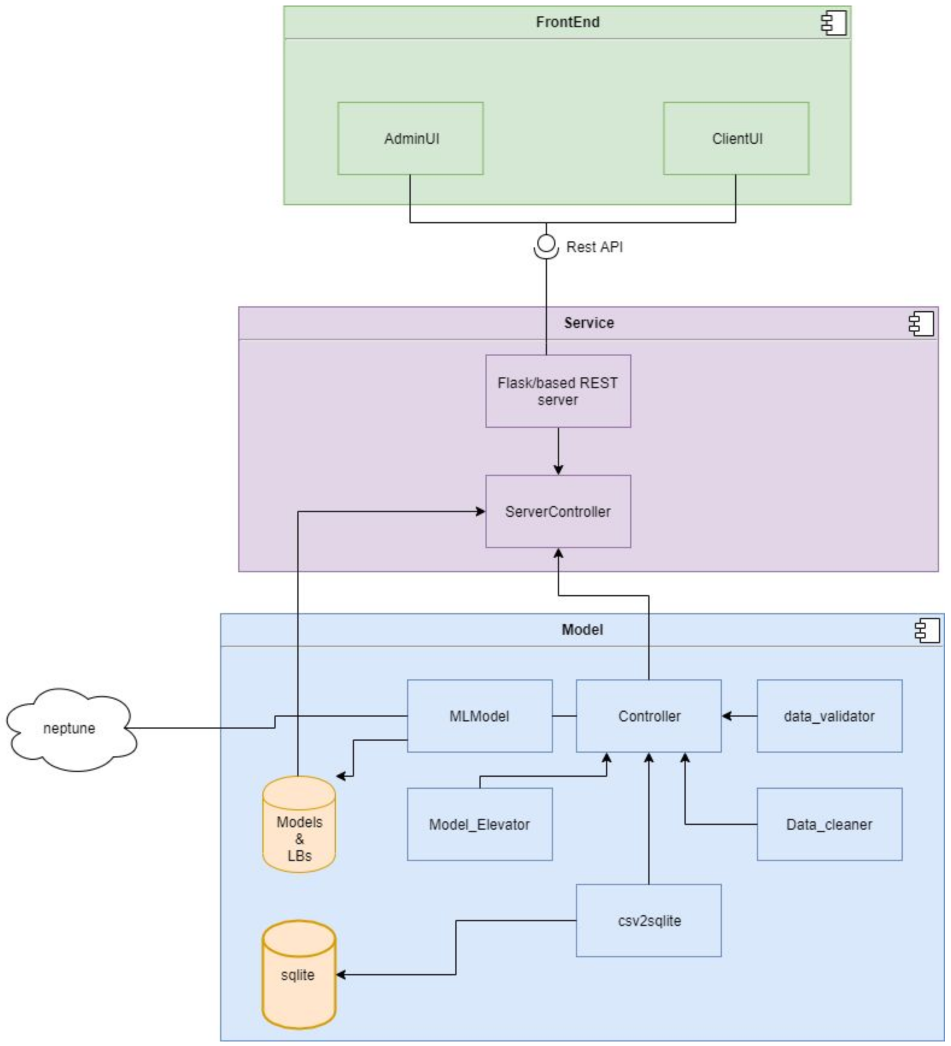
validation made

Metrics for the Model :

MAE value: 13279.443973130396

R Square value : 97.21406089202156

Brief representation of our system:



Section 5. Reflections

1. Like any project and any team we had our challenges, some challenges that we faced:
    a. New domain, none of us ever worked with machine learning model and we didn't know what kind of challenges/bugs that can we face and how we should address it. For example we had a bug in our model in which our model always returned the same number when performing the predication! We spent one week just

    debugging it and then we understood that the order of the columns while performing a prediction should be the same order of the columns while training the model.
    b. We also had some problems in scheduling our meeting with some of our team members, for example one of our team members came late to our weekly meeting (that issue later solved by bringing the problem up) and one other team member just disappeared after the week one and never showed up again.
2. Things that we could have done some differently:
    a. We could have better software development process: from the week one we choose scrum, in which we decided to have a sprints of one week long and have a meeting twice a week, but we skipt a main activity in the scrum which is daily stand up meetings, that activity will insure that we had a method to follow up on a team members progress on everyday instead of just having it twice a week
    b. Better organization of the tasks: some tasks had dependencies on each other which created a problem! For example, some team members had to wait for others to finish their tasks to be able to start their own.
3. Possible extensions to our system:
    a. Authentication, our system should have an authentication for admin, to only let the admin to train and validate the model. For now admin page can be accessed with a specific URL and doesn't ask for any authentication.
    b. Sporting new features, Like how many times a car has been through a car accident or what is the condition of the car. Since the condition of the car can have a high correlation with the price.
    c. Better UI, improve the UI to improve the user experience.
    d. Add extensions; since our data is scraped directly from blocket.se, we can connect to blocket and maybe they will allow us to automatically import the data that is added to their website to improve our model accuracy and keep our model always relevant and updated.
    e. Implement a new feature to our system in which user will input all the car information and the car price and our system will predict if the price is above, below or just right price base on the market.

Appendix I – Team activity

Member	Assignment 1 Out of 8	Assignment 2 Out of 7	Notes
George	8	7	None
Martin	8	7	None
Amjad	7	7	He skipped one meeting because of the doctor's appointment
Majed	7	7	He skipped the first meeting because he had a rescheduled meeting with another teacher
Mohammad	2	-	He stopped contributing after week 1

Appendix II - Responsibilities

Feature description	Week 1			
	Who was responsible	Delivered	Integrated	Notes
Assignment 1	All	Yes	Yes	
Code of conduct	George	Yes	Yes	
Markdown	All	Yes	Yes	
Searching for datasets	All	yes	yes	
	Week 2			
Feature Engineering	George	yes	yes	
Clean up the data	Martin	yes	yes	
CSV to SQLite	Amjad	yes	yes	
Research in different regression models. ex. Cart	Majed	yes	yes	

Feature description	Week 3			
	Who was responsible	Delivered	Integrated	Notes
Data validation schema and	Amjad	Yes	Yes	

store in SQLite				
Feature engineering on model and maker, on Blocket dataset	Martin	Yes	Yes	
Feature engineering HP, Mileage based on manufacture year	George	Yes	Yes	
Initial FrontEnd	Majed	Yes	No	Will be integrated next sprint
Model Evaluation Metrics with code	Majed	Yes	Yes	
Refactor the code	All	Yes	Yes	

Feature description	Week 4			
	Who was responsible	Delivered	Integrated	Notes
Read data from SQLite. Add new columns for the date of the data has been added to the database. Make a function to only retrieve the data for a special date.	Amjad	Yes	Yes	
Check & implement a way to understand	Martin	Yes	Yes	

new input from the user to get prediction from the model(labelEncoder). Clean MakeModel.json.				
Implement & integrate server with flask with simple APIs for GET & POST. Implement feature eng. on the data that user will enter to predict price. Create the https request to POST the csv file to the backend.	George	Yes	Yes	
Implement & integrate server with flask with simple APIs for GET & POST. Implement Model and Maker filed in frontend with autocomplete.	Majed	Yes	Yes	

Feature description	Week 5			
	Who was responsible	Delivered	Integrated	Notes
Save date as timestamp in the database. Create function	Amjad	Yes	Yes	

to get all entries between two dates from the database.				
Add an admin page in the frontend.	Martin	Yes	Yes	
Fix the bug in label encoder. Fix the bug in predicting new values(Always the same price). Fix the bug in the lambda median function, when the user doesn't input hp. Write code to retrain the model based on data from specific date from the database.	George	Yes	Yes	
Create API endpoint to retrain the model with a specific date.	Majed	Yes	Yes	

Feature description	Week 6			
	Who was responsible	Delivered	Integrated	Notes
add visualization to Admin page.	Amjad	Yes	No	Will be integrated in next sprint

integrating the code.	Martin	Yes	Yes	
Unit testing. Prepare clean jupyter notebook file for the model.	George	Yes	Yes	
add versioning to the model. autocomplete for car model. from previous sprint.	Majed	Yes	No	Needs to be updated(the information of the model is now in Nuptone this information should be in our website)

Feature description	Week 7			
	Who was responsible	Delivered	Integrated	Notes
add visualization to Admin page.	Amjad	Yes	No	
Feature engineering on model and maker, on Blocket dataset	Martin	Yes	Yes	
Feature engineering HP, Mileage based on manufacture year	George	Yes	Yes	
add versioning to the model. Improve the validation part on Admin page by	Majed	Yes	No	

adding padges to values.				
Update the report.	All	Yes	Yes	

UNIVERSITY OF GOTHENBURG

DIT824: Software Engineering for Data-Intensive AI Applications

Deliverable: ________________ (First Written Report, Second Written Report, Prototype)

Team number: _____

Individual Contribution in Team Work

The contribution of each team member to the overall work in the delivered artifact must be stated in the table below (one row per team member). The form must be signed by all team members.

Individual Name	**Description of contribution**	**Signature**