COGS 118A - Final Project

Used Car Market Price Prediction

Group members

- Jisu Kim
- Taekyu Lee
- Brian Chu
- Richard Li

```
%%capture download__info
# about ~7s download
%pip install xgboost
```

```
# imports for pre-processing
import numpy as np
import pandas as pd

from pandas.plotting import scatter_matrix

import seaborn as sns

from matplotlib import rcParams
import matplotlib.pyplot as plt
```

```
# imports for training model
import xgboost as xgb
import sklearn
from sklearn import preprocessing
from sklearn.model_selection import train_test_split, GridSearchCV, train_test_split,
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsRegressor
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import explained_variance_score, max_error, r2_score, mean_absolu
from sklearn.exceptions import ConvergenceWarning
```

```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter("ignore", category=ConvergenceWarning)
warnings.filterwarnings('ignore')
warnings.simplefilter('ignore')
```

```
# uncomment if running file and these packages aren't installed
# !pip install geopandas
# !pip install Shapely
```

Abstract

In evaluating the used car market, it often becomes difficult to determine what the ‘correct’ value of a used car is, especially with user-generated listings. We found a dataset with scraped listings off Craigslist for all used vehicles across the country, with a plethora of features that can help us classify listings and predict how much a car ought to be listed for.

We first approach the dataset by analyzing its features, and begin preprocessing with an understanding of which features we found relevant to prediction, as well as visualizing the dataset to better understand what we're working with. During this, we eliminate certain features we found to be commonly omitted, and undergo feature engineering in a way that our data becomes more usable to us.

Finally, we discover that [model] is the most viable approach to this problem, with [results]. Through this, we realize [blank .]

Background

Car is one of the vital things in our daily life. Especially in the United States, we may see a significant number of automakers compete for their market and industry every day. While the automakers lunch a different type of brand-new vehicle annually or so, customers get some option to purchase their car, brand new or used. From the past, people might consider that buying a used car is the best option with two aspects, their budget, and sustainability. Among previous options, budget is apparently the biggest factor that people lean on buying a used car.

There are numerous factors that determine the selling price of a vehicle. We can also confirm that there are a variety of additional costs depending on the model, mileage, fuel consumption, etc. In most cases, the older the car is, the less it is worth. For example, a brand new car may lose 20-30% of its value in only 1-2 years, but the price drop may be slower thereafter.

People have made a myriad of the attempt to predict the used car price. Especially in the 2010s, with the huge leap in the machine learning technique, people did get even more opportunities to predict the used car price using solid models. In this project, our team's ultimate goal and the aim is to get the highest accuracy using proper ML models and algorithms.

Problem Statement

We are planning to use the supported vector machine and logistic regression to predict the used car price in the current market. During the checkpoint, we were about to use the neural network(model representation) since we had only few features and even needed to drop some columns which we do not need for our prediction. Our dataset was way too small to use a neural network and we decided to implement the svm and logistic regression instead of a neural network since the changed algorithm is enough for our dataset. We use numpy, pandas, sklearn, matplotlib, seaborn and LogisticRegression from sklearn.linear_model. We will import more libraries upon our needs.

Data

Dataset

Used Cars Dataset Vehicles listings from Craigslist.org

This data is scraped every few months, it contains almost all relevant information that Craigslist provides on car sales including columns like price, condition, manufacturer, latitude/longitude, and 18 other categories.

The dataset originally included

- 26 columns
- 426880 unique value

The column that we kept while the data cleaning process is following :

region = craigslist region price = entry price year = entry year manufacturer = manufacturer of vehicle model = model of vehicle fuel = fuel type odometer = miles traveled by vehicle title_status = title status of vehicle transmission = transmission of vehicle state = state of listing

The column that we dropped while the data cleaning process is following :

id = entry ID

url = listing URL

region_url = region URL

condition = condition of vehicle

cylinders = number of cylinders

VIN = vehicle identification number

drive = type of drive

size = size of vehicle

type = generic type of vehicle

paint_color = color of vehicle

image_url = image URL

description = listed description of vehicle

county = useless column left in by mistake

posting_date = posted date

lat = latitude of listing

long = longitude of listing

```
cars = pd.read_csv('vehicles.csv')
cars.sample(5) # random sample to peek at data
```

	id	url	region	regi
217758	7310383363	https://stcloud.craigslist.org/ctd/d/sauk-cent...	st cloud	https://stcloud.craig

Through exploratory data analysis, our preprocessing, and visualizations, we made many discoveries regarding the feasibility of building a model around this dataset.

- We discovered that many features were simply not generalizable to be used for prediction, and that the dataset omitted many features that we needed to remove,
- we discovered that many listings were not necessarily accurate or listed in bad faith through falsely representing price or mileage, and we had to remove a substantial amount of outliers in a blanket fashion, since we had no way of accurately determining which listings were outliers,
- we discovered the necessity for one hot encoding or label encoding many of the features, and that feature engineering provided us with quantifiable data that could then be passed into our modelling.

Preprocessing / EDA

5 rows × 26 columns

choosing important features

We want to select for only the features that would have an impact on our prediction. As such,

- due to lack of relevance, we will remove the `id`, `url`, `region_url`, and `image_url` features;
- due to uniqueness, we will remove the `VIN` and `description` features, since VIN provides no easy information about the vehicle, and the description feature is not easily generalizable enough without undergoing natural language processing on the value;
- due to frequent omission in the dataset, we remove the `posting_date`, `drive`, `type`, `paint_color`, `cylinders`, `size`, `lat`, `long`, and `county` features, since far more than half of our samples do not have this feature (our criteria was more than 100k missing values in that column);
- due to the subjective nature of the self-rating, we remove the `condition` feature as well.

```
cars = cars.drop(['id', 'url', 'region_url', 'image_url', 'VIN', 'description', 'count
                  'size', 'condition', 'cylinders', 'drive', 'type', 'paint_color'], axi
```

removing null data

Since we want generalizable data, we believe that listings omitting any data ought to not be included.

```
cars = cars.dropna()
cars = cars[cars['price']>0]
```

outliers

Let's take a look at outliers, for quantifiable metrics such as price and mileage, through getting their deciles.

A common issue with user-generated car listings is that frequently, the listing price is either arbitrarily high, arbitrarily low, or a projected monthly payment price (in the case the listing is made by a dealer) rather than the price of the car. To avoid this issue, we've determined that we need to drop all observations with a negative price, observations with an arbitrarily low price (to avoid the monthly payment issue), and observations with an arbitrarily high price.

Similarly, we discover the issue with mileage. Many listings are set at arbitrarily high mileage values, such as a million. We remove these, as well as listings with arbitrarily low values.

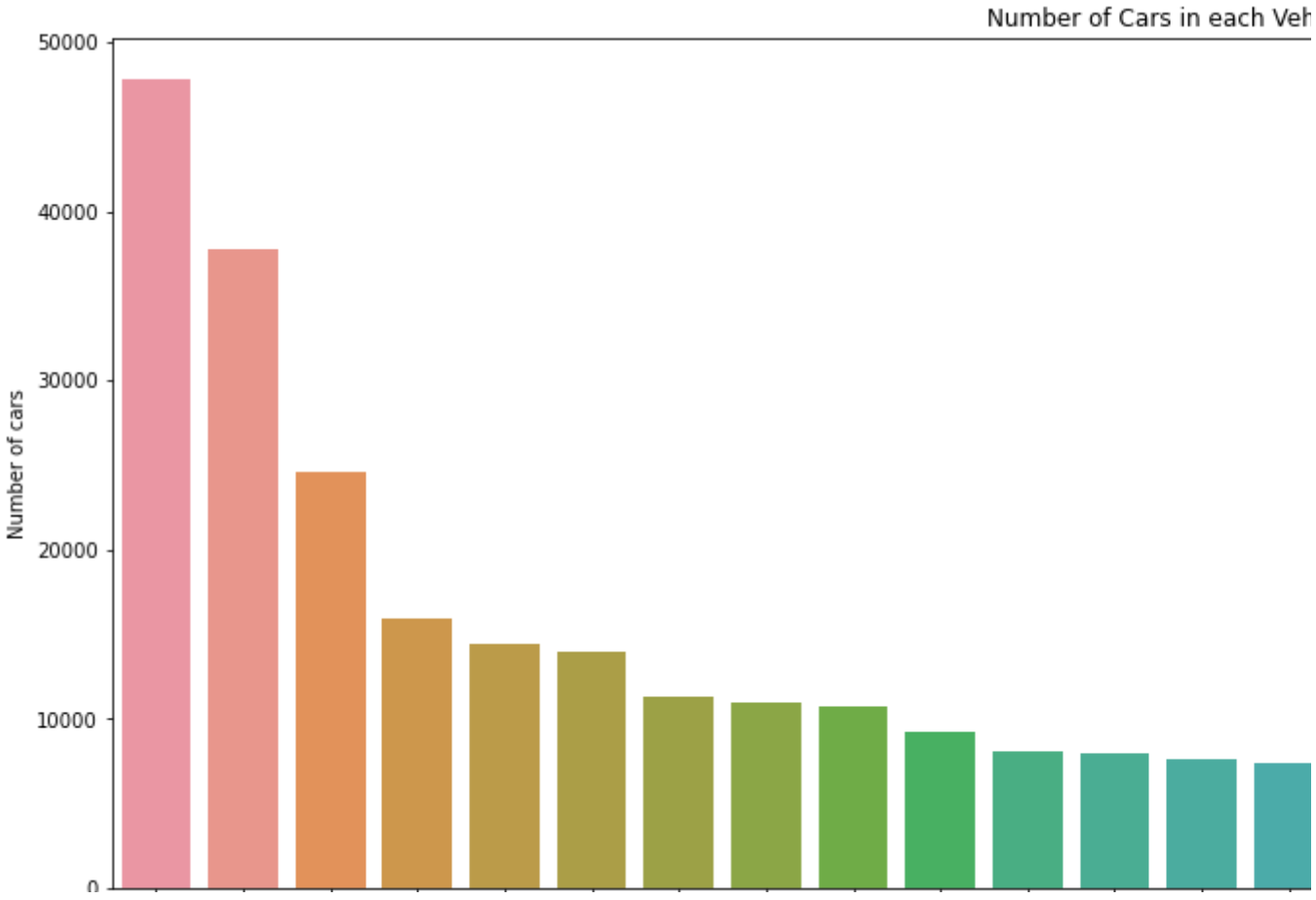
```
# get deciles
price_deciles = np.append(np.percentile(cars['price'], np.arange(0, 100, 10)), np.max(
mileage_deciles = np.append(np.percentile(cars['odometer'], np.arange(0, 100, 10)), np
print("price deciles:", price_deciles, "\n mileage deciles:", mileage_deciles)

# drop outlier ranges
price, mileage = cars['price'], cars['odometer']
cars = cars[price.between(price.quantile(0.05), price.quantile(0.95))]
cars = cars[mileage.between(mileage.quantile(0.05), mileage.quantile(0.95))]

# check again
new_price_deciles = np.append(np.percentile(cars['price'], np.arange(0, 100, 10)), np.
new_mileage_deciles = np.append(np.percentile(cars['odometer'], np.arange(0, 100, 10))
print("\nnew price deciles:", new_price_deciles, "\n mileage deciles:", new_mileage_de
```

```
price deciles: [1.00000000e+00 3.85000000e+03 6.00000000e+03 8.50000000e+03
 1.19000000e+04 1.55000000e+04 1.96200000e+04 2.49900000e+04
 3.05366000e+04 3.79900000e+04 3.73692871e+09]
 mileage deciles: [       0.     15747.7    30223.     46583.     68000.     870
   104940.6   124890.    147064.    178000.  10000000. ]

new price deciles: [ 2100.  4999.  7000.  9500. 12497. 15800. 18995. 23927. 2859
 44944.]
 mileage deciles: [  7619.   22618.   36359.8  53500.   72500.   89730.  105000.
 140921.2 164000.  203575. ]
```

visualizations

manufacturers

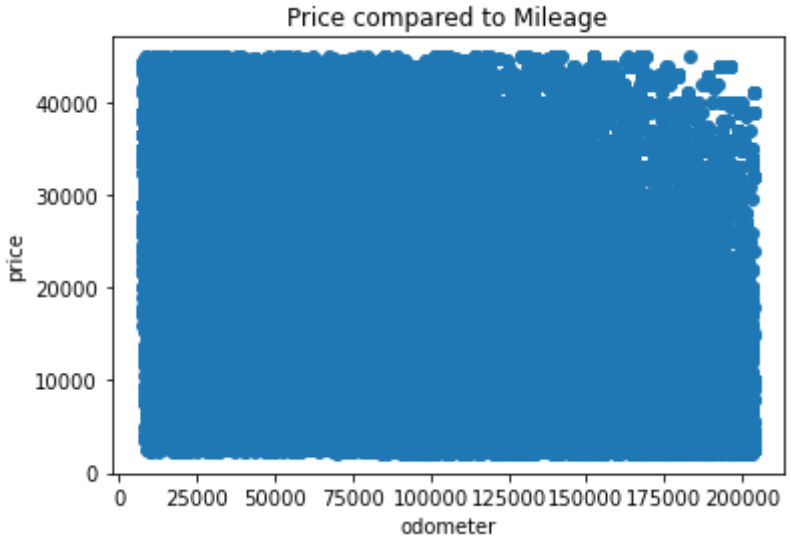
```
top_manufacturers = cars.manufacturer.value_counts(dropna=False).iloc[:25]

plt.figure(figsize=(20,8))
sns.barplot(x=top_manufacturers.index,y=top_manufacturers.values)
plt.xlabel('Vehicle type')
plt.ylabel('Number of cars')
plt.title('Number of Cars in each Vehicle Type');
```

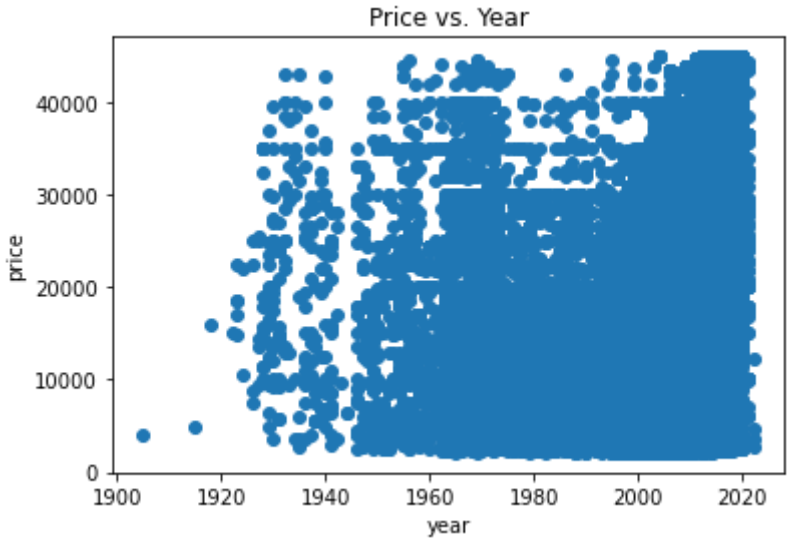


mileage and year

```
y = cars['price']
x = cars['odometer']
plt.scatter(x, y)
plt.xlabel('odometer')
plt.ylabel('price')
plt.title('Price compared to Mileage')
plt.show()
```

```
y = cars['price']
x = cars['year']
plt.scatter(x, y)
plt.xlabel('year')
plt.ylabel('price')
plt.title('Price vs. Year')
plt.show()
```



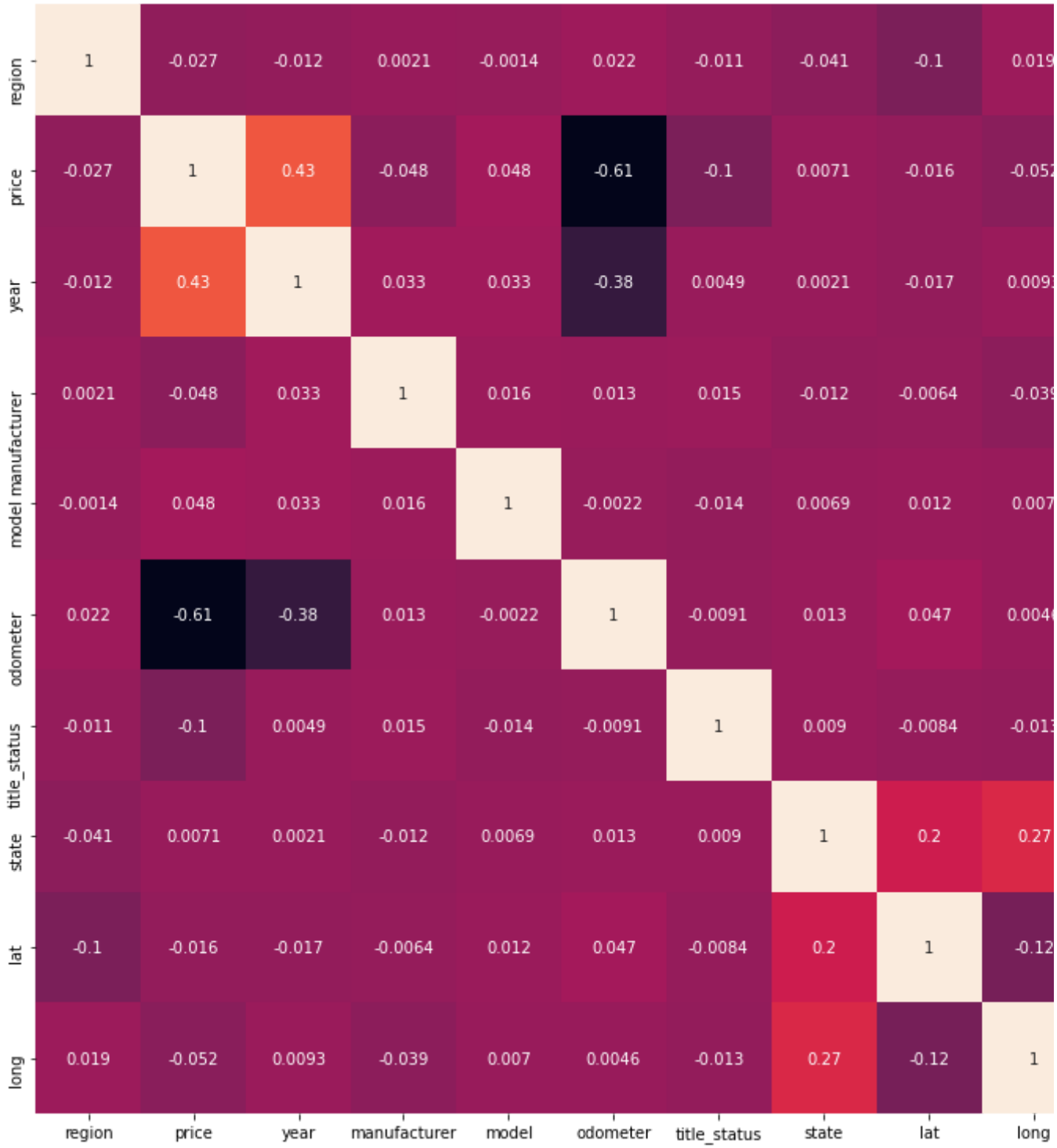


Feature Engineering

Here, we utilize label encoding and one hot encoding to transform certain data into something we can pass into the model, especially data that can't be quantified.

```
# label

encodeL = preprocessing.LabelEncoder()
llist = ['region', 'manufacturer', 'model', 'title_status', 'state']
```

```
cars[llist] = cars[llist].apply(encodeL.fit_transform)


cars["odometer"] = np.sqrt(preprocessing.minmax_scale(cars["odometer"]))

plt.figure(figsize=(15,13))
cor = cars.corr()
sns.heatmap(cor, annot=True)
plt.show()
```




```
# one hot
```

```
encodeO = preprocessing.OneHotEncoder()
olist = ['fuel', 'transmission']
for col in olist:
    if col in cars.columns:
        carsadd = pd.DataFrame(encodeO.fit_transform(cars[col].array.reshape(-1, 1)).t
        cars = pd.concat([cars, carsadd], axis=1).drop([col], axis=1)


carsModel = cars.sample(n=1000, random_state = 42)


# Train-test split with 25% test, 18.75% validation, 57.25% train

# consistent dataset splits
seed = 42

df_train, df_test= train_test_split(carsModel, test_size=0.25, random_state=seed)
df_train, df_val= train_test_split(df_train, test_size=0.25, random_state=seed)
df_train.head()
```

	region	price	year	manufacturer	model	odometer	title_status	state
409874	312	6995	2000.0	38	3696	0.552270	0	47
335982	271	25950	2015.0	7	13988	0.654751	0	38
231124	12	17990	2011.0	16	11359	0.523397	0	27
292681	61	18442	2013.0	33	86	0.791152	0	35
294300	61	5000	2010.0	13	8415	0.937193	0	35

```
# drop price column
X_train1 = df_train.drop(['price'], axis = 1)
y_train1 = df_train['price']

X_val1 = df_val.drop(['price'], axis = 1)
y_val1 = df_val['price']

X_test1 = df_test.drop(['price'], axis = 1)
y_test1 = df_test['price']

# setup KNN for lat n long
X_trainpk = X_train1.drop(['lat','long'], axis = 1)
X_traink = df_train[['lat','long']]

X_valpk = X_val1.drop(['lat','long'], axis = 1)
X_valk = df_val[['lat','long']]
```

```
X_testpk = df_test.drop(['lat','long'], axis = 1)
X_testk = df_test[['lat','long']]
```

Proposed Solution

Our solution will implement linear and logistic regression, support vector machines, random forest classifiers, and cross validation. We will also test the K Nearest Neighbors Regression with the locational values as a layered neural network in addition to the algorithm without the locational values to compare with using the locational values as part of the algorithm. We use numpy, pandas, sklearn, matplotlib, seaborn libraries.

The link below is our benchmark model dealing with linear regression among a few others that have used the same dataset to train their model.

Comparing these different combinations of approaches and algorithms allows us to evaluate performance of the models that have different levels of complexity while being able to modify parameters to find the optimal model.

https://www.kaggle.com/code/vbmokin/used-cars-price-prediction-by-15-models

Evaluation Metrics

We are using a regression model to predict used car prices, so we will use Mean Squared Error and Mean Squared Log Error, since it nullifies the effect of outliers, as some used supercars may be a very large outlier. Mean square error takes the average of the square of predicted error, the predicted value difference to the actual true value. This is a good evaluation metric, since it gives us a numerical variance of the pricing predictions. We also use mean absolute error and mean absolute percentage error, which doesn't penalize large values as much as mean square error, but the percentage gives us a different viewpoint. Overall, we can use the r-squared scoring to see the correlation among the attributes of each used car and true or predicted pricing.

```
scaler = StandardScaler()

def scoring(y_train, y_predicted):
    # ['MSE', 'MSLE', 'MAE', 'MAPE', 'R2']
    mse = mean_squared_error(y_train, y_predicted)
    try:
        msle = mean_squared_log_error(y_train, y_predicted)
    except:
        msle = 0
    mae = mean_absolute_error(y_train, y_predicted)
    mape = mean_absolute_percentage_error(y_train, y_predicted)
```

```
    r2 = r2_score(y_train, y_predicted)
```

Results

Linear Regression

```
modelLI = LinearRegression().fit(X_train1, y_train1)
y_pred1LI = modelLI.predict(X_train1)
modelLI.score(X_test1, y_test1)
```

```
    0.5037488304491067
```

```
scoring(y_train1, y_pred1LI)
```

```
    [57682514.402621776,
     0,
     5922.512286145776,
     0.5060348133875044,
     0.49662266119859755]
```

Logisitic Regression

```
imp = SimpleImputer(missing_values=np.nan, strategy='mean')
logistic = LogisticRegression(C=0.01, penalty='l1',solver='liblinear')

pipeLR = Pipeline(steps=[("scaler", scaler), ("imputer", imp), ("logistic", logistic)])
pipeLR.fit(X_train1, y_train1)
pipeLR.score(X_test1, y_test1)
```

```
    0.004
```

```
pipeLR.fit(X_train1, y_train1)
y_pred1LR = pipeLR.predict(X_train1)
scoring(y_train1, y_pred1LR)
```

```
    [154659227.72241992,
     0.5439770151110105,
     9263.338078291816,
     0.5971667881633733,
     -0.3496629139393639]
```

```
param_grid = {
    'penalty' : ['l1','l2'],
    'C'       : np.logspace(-3,3,7),
    'solver'  : ['newton-cg', 'lbfgs', 'liblinear']}
```

```
gcv = GridSearchCV(LogisticRegression(), param_grid = param_grid, cv=2, verbose=True)
best = gcv.fit(X_val1, y_val1)
```

```
Fitting 2 folds for each of 42 candidates, totalling 84 fits
```

```
print(best.best_score_)
print(best.best_params_)
print(best.cv_results_)
print(best.score(X_test1, y_test1))
```

```
0.02127659574468085
{'C': 0.01, 'penalty': 'l1', 'solver': 'liblinear'}
{'mean_fit_time': array([9.76443291e-04, 5.55396080e-04, 1.21322870e-02, 1.99861
       7.10572004e-02, 2.04422474e-02, 3.84569168e-04, 3.68356705e-04,
       2.33453512e-02, 2.06570935e+00, 6.42782450e-02, 2.81949043e-02,
       3.77774239e-04, 4.24385071e-04, 8.77099037e-02, 2.17409456e+00,
       5.98366261e-02, 3.38833332e-02, 3.60846519e-04, 3.44038010e-04,
       3.96802306e-01, 2.10166287e+00, 6.98866844e-02, 4.43997383e-02,
       4.03881073e-04, 3.67879868e-04, 9.03492570e-01, 1.94510221e+00,
       6.23201132e-02, 4.51225042e-02, 1.73223019e-03, 3.56435776e-04,
       6.51607513e-01, 1.90966690e+00, 6.46368265e-02, 5.09492159e-02,
       4.64081764e-04, 3.47137451e-04, 4.19149399e-01, 2.14243245e+00,
       6.14906549e-02, 4.58818674e-02]), 'std_fit_time': array([3.87549400e-04,
       5.99563122e-03, 3.67641449e-04, 7.39097595e-06, 2.38418579e-07,
       1.74164772e-04, 1.07975006e-02, 5.53965569e-04, 4.38928604e-04,
       3.93390656e-06, 5.24520874e-06, 4.44507599e-03, 1.77543998e-01,
       3.03435326e-03, 3.43132019e-03, 1.37090683e-05, 8.34465027e-06,
       1.20543242e-02, 9.84795094e-02, 7.57932663e-03, 3.65257263e-04,
       1.74045563e-05, 7.62939453e-06, 7.13268518e-02, 5.53860664e-02,
       2.29752064e-03, 7.33137131e-05, 1.37507915e-03, 4.76837158e-06,
       1.14293098e-02, 8.73357058e-02, 3.20279598e-03, 6.84010983e-03,
       3.77893448e-05, 0.00000000e+00, 6.68313503e-02, 6.39319420e-03,
       2.30443478e-03, 7.86662102e-04]), 'mean_score_time': array([0.        , 0
       0.00126505, 0.        , 0.        , 0.00117683, 0.00169384,
       0.00119066, 0.00123656, 0.        , 0.        , 0.00135398,
       0.00213826, 0.00117016, 0.00114393, 0.        , 0.        ,
       0.00115252, 0.00307453, 0.00132155, 0.0012958 , 0.        ,
       0.        , 0.00275946, 0.00194168, 0.00114775, 0.0012058 ,
       0.        , 0.        , 0.00199652, 0.00229895, 0.00157607,
       0.00145769, 0.        , 0.        , 0.00113189, 0.00203705,
       0.00111639, 0.00117099]), 'std_score_time': array([0.00000000e+00, 0.0000
       2.09808350e-05, 6.43730164e-06, 0.00000000e+00, 0.00000000e+00,
       6.43730164e-06, 2.02655792e-06, 3.17096710e-05, 5.84125519e-06,
       0.00000000e+00, 0.00000000e+00, 2.79664993e-04, 4.77910042e-04,
       5.36441803e-05, 3.57627869e-06, 0.00000000e+00, 0.00000000e+00,
       5.67436218e-05, 6.19292259e-04, 1.88350677e-04, 1.47819519e-05,
       0.00000000e+00, 0.00000000e+00, 1.13081932e-03, 5.02347946e-04,
       1.28746033e-05, 7.51018524e-06, 0.00000000e+00, 0.00000000e+00,
       4.12940979e-04, 6.09040260e-04, 4.65989113e-04, 2.82764435e-04,
       0.00000000e+00, 0.00000000e+00, 2.20537186e-05, 4.62532043e-04,
       3.57627869e-07, 1.60932541e-05]), 'param_C': masked_array(data=[0.001, 0.
                  0.01, 0.01, 0.01, 0.01, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1,
                  1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 10.0, 10.0, 10.0, 10.0,
```

```
                       10.0, 10.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0,
                       1000.0, 1000.0, 1000.0, 1000.0, 1000.0, 1000.0],
                 mask=[False, False, False, False, False, False, False, False,
                       False, False, False, False, False, False, False, False,
                       False, False, False, False, False, False, False, False,
                       False, False, False, False, False, False, False, False,
                       False, False, False, False, False, False, False, False,
                       False, False],
           fill_value='?',
                dtype=object), 'param_penalty': masked_array(data=['l1', 'l1', 'l1',
                       'l2', 'l2', 'l2', 'l1', 'l1', 'l1', 'l2', 'l2', 'l2',
                       'l1', 'l1', 'l1', 'l2', 'l2', 'l2', 'l1', 'l1', 'l1',
                       'l2', 'l2', 'l2', 'l1', 'l1', 'l1', 'l2', 'l2', 'l2',
                       'l1', 'l1', 'l1', 'l2', 'l2', 'l2'],
                 mask=[False, False, False, False, False, False, False, False,
```

Logisitic Regression with KNN Regression

```
imp = SimpleImputer(missing_values=np.nan, strategy='mean')
logistic = LogisticRegression(C=0.5, penalty='elasticnet',solver='saga', l1_ratio = 0.
pipeLRK = Pipeline(steps=[("scaler", scaler), ("imputer", imp), ("logistic", logistic)
pipeLRK.fit(X_trainpk, y_train1)
y_pred1LRK = pipeLRK.predict(X_trainpk)
scoring(y_train1, y_pred1LRK)
```

```
[93387282.08540925,
 0.3696520771668765,
 6742.601423487545,
 0.4227243394431131,
 0.18503827336776868]
```

```
kNR = KNeighborsRegressor(n_neighbors=3)
kNR.fit(X_traink, y_pred1LRK)
y_predk = kNR.predict(X_traink)
scoring(y_train1, y_predk)
```

```
[107433400.77382363,
 0.4412506271040967,
 7818.623962040333,
 0.6151871049252203,
 0.06246217003581489]
```

Random Forest Classifiers

Initial Random Hyperparameters

```
rf = RandomForestClassifier(n_jobs = -1)
pipeRFC = Pipeline(steps=[("scaler", scaler), ("rf", rf)])
```

```
pipeRFC.fit(X_train1, y_train1)
pipeRFC.score(X_test1, y_test1)
```

```
0.012
```

Grid Search Best Hyperparameters

```
param_grid = {'rf__max_depth': (100, 500), 'rf__min_samples_split': (2, 5)}
gscv = GridSearchCV(estimator = pipeRFC, param_grid = param_grid, cv = 5, scoring = 'a
gscv.fit(X_val1, y_val1)
```

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('scaler', StandardScaler()),
                                       ('rf',
                                        RandomForestClassifier(n_jobs=-1))]),
             n_jobs=-1,
             param_grid={'rf__max_depth': (100, 500),
                         'rf__min_samples_split': (2, 5)},
             scoring='accuracy')
```

```
print(gscv.best_score_)
print(gscv.best_params_)
print(gscv.cv_results_)
```

```
0.03203223767383059
{'rf__max_depth': 100, 'rf__min_samples_split': 2}
{'mean_fit_time': array([1.62026978, 1.56395845, 1.83056784, 1.15958261]), 'std_
             mask=[False, False, False, False],
       fill_value='?',
            dtype=object), 'param_rf__min_samples_split': masked_array(data=[2, 
             mask=[False, False, False, False],
       fill_value='?',
            dtype=object), 'params': [{'rf__max_depth': 100, 'rf__min_samples_sp
```

```
gscv.score(X_test1, y_test1)
```

```
0.032
```

```
pipe = Pipeline([('classifier' , RandomForestClassifier())])
param_grid = [
    {'classifier' : [LogisticRegression()],
     'classifier__penalty' : ['l1', 'l2'],
    'classifier__C' : np.logspace(-4, 4, 10),
    'classifier__solver' : ['liblinear']},
    {'classifier' : [RandomForestClassifier()],
    'classifier__n_estimators' : list(range(10,101,10)),
    'classifier__max_features' : list(range(6,32,5))}
]
```

```
gcv = GridSearchCV(pipe, param_grid = param_grid, cv = 2, verbose=True, scoring = 'neg
best = gcv.fit(X_val1, y_val1)
```

Show hidden output

```
print(best.best_score_)
print(best.best_params_)
print(best.cv_results_)
```

```
-133736362.36702128
{'classifier': RandomForestClassifier(max_features=6, n_estimators=40), 'classifi
{'mean_fit_time': array([0.02279997, 0.03060079, 0.02273583, 0.03537869, 0.02632
       0.03145051, 0.05029011, 0.03682637, 0.26286721, 0.04309094,
       0.73177958, 0.04918647, 1.06537211, 0.05404544, 0.75711989,
       0.0600971 , 0.51088524, 0.05904531, 0.32941616, 0.05873346,
       0.01737607, 0.03389335, 0.05048227, 0.06371772, 0.08147407,
       0.1029985 , 0.11222196, 0.13042879, 0.14326429, 0.16321027,
       0.02272654, 0.04444671, 0.06405294, 0.08619308, 0.10840392,
       0.12532747, 0.14535081, 0.17138624, 0.18689191, 0.22239816,
       0.02697098, 0.0517993 , 0.07440865, 0.10433757, 0.12744892,
       0.16662133, 0.19158328, 0.20271444, 0.24686015, 0.25138152,
       0.00544286, 0.00853932, 0.01097119, 0.01249647, 0.01557136,
       0.02036035, 0.02401459, 0.02272713, 0.02552962, 0.03048658,
       0.00521314, 0.00758052, 0.01114154, 0.01275527, 0.01701534,
       0.01936615, 0.02174878, 0.02466595, 0.02652407, 0.02941597,
       0.00516999, 0.00761926, 0.01026356, 0.0132941 , 0.01580417,
       0.01841319, 0.02043462, 0.02914643, 0.02970719, 0.02929962]), 'std_fit_ti
       1.47294998e-03, 1.35421753e-04, 2.38418579e-06, 5.79357147e-05,
       7.80820847e-03, 2.05218792e-03, 4.15279865e-02, 1.84273720e-03,
       1.47358179e-02, 4.88042831e-04, 5.39324284e-02, 1.06012821e-03,
       1.22680902e-01, 8.81433487e-04, 8.32196474e-02, 4.81367111e-04,
       4.13298607e-04, 9.20295715e-04, 2.82526016e-04, 2.53558159e-04,
       4.01878357e-03, 3.15570831e-03, 1.35445595e-03, 2.95662880e-03,
       4.79936600e-04, 6.27398491e-04, 3.68118286e-04, 1.19185448e-03,
       2.74574757e-03, 2.09498405e-03, 6.70671463e-04, 2.04885006e-03,
       1.08706951e-03, 3.28445435e-03, 1.87754631e-04, 2.58934498e-03,
       1.48177147e-04, 2.82907486e-03, 3.85987759e-03, 3.41999531e-03,
       1.58512592e-03, 5.13732433e-03, 5.32996655e-03, 9.89437103e-03,
       5.54263592e-03, 6.59716129e-03, 3.55243683e-05, 3.54409218e-04,
       2.35676765e-04, 7.12871552e-05, 4.04834747e-04, 1.44660473e-03,
       3.28338146e-03, 1.64389610e-04, 2.06708908e-04, 8.94069672e-04,
       2.23517418e-04, 7.65323639e-05, 8.92162323e-04, 1.54376030e-04,
       1.64735317e-03, 7.85589218e-05, 2.77519226e-04, 1.65379047e-03,
       9.37223434e-04, 1.08063221e-03, 2.09450722e-04, 1.43647194e-04,
       6.94990158e-05, 2.17556953e-04, 2.39610672e-05, 5.33699989e-04,
       5.72204590e-06, 3.76701355e-04, 2.58517265e-03, 5.15818596e-04]), 'mean_s
       0.0016489 , 0.0016712 , 0.00151765, 0.00161934, 0.00140452,
       0.00273728, 0.00145352, 0.00605345, 0.00138283, 0.0028801 ,
       0.00135684, 0.00169456, 0.00358343, 0.00175989, 0.00136268,
       0.0026052 , 0.00368989, 0.00459445, 0.00517249, 0.00696349,
       0.00826573, 0.0099355 , 0.0097574 , 0.01186204, 0.01192284,
       0.00276387, 0.00365949, 0.00486338, 0.00603902, 0.00729454,
       0.00813937, 0.00870502, 0.01105809, 0.01057577, 0.01209486,
       0.00259626, 0.0035677 , 0.00455296, 0.0061028 , 0.00790226,
       0.00832725, 0.01200366, 0.01165605, 0.01101565, 0.01079202,
       0.        , 0.        , 0.        , 0.        , 0.        ,
```

```
            0.        , 0.        , 0.        , 0.        , 0.        ,
            0.        , 0.        , 0.        , 0.        , 0.        ,
            0.        , 0.        , 0.        , 0.        , 0.        ,
            0.        , 0.        , 0.        , 0.        , 0.        ,
            0.        , 0.        , 0.        , 0.        , 0.        ]), 'std_score_
            8.90493393e-05, 3.83853912e-05, 1.71780586e-04, 3.45706940e-06,
            4.29153442e-06, 1.79290771e-04, 7.39574432e-04, 1.70469284e-05,
            4.59575653e-03, 5.69820404e-05, 1.20759010e-03, 9.41753387e-05,
            2.99215317e-05, 2.01559067e-03, 6.80685043e-05, 2.50339508e-06,
            1.31130219e-05, 1.67489052e-04, 2.80976295e-04, 3.13758850e-04,
```

```
best.score(X_test1, y_test1)
```

```
    -114759897.728
```

Subsection 3

We are implementing the SVC from sklearn.svm and logistic regression from sklearn.linear_model. We train our model with logistic regression(LogisticRegression). First of all, we initialize the scaler object, the simple imputer and the logistic regression from the linear model. We built the pipeline with the steps of scaler, imputer and logistic regression. We have been through the Grid Search cross validation with the pipeline, our parameter grid and cv = 7.

Discussion

Limitations

Undoubtedly, we encountered many limitations during this process, both pertaining to the data and with our ability to model the data.

1. the outliers we removed from the dataset were done so in a blanket fashion, by simply removing the extremities of two features - we perhaps removed many valid samples in doing so, yet could not devise a better alternative
2. the time it took to run the training heavily limited our ability to make iterative changes - the runtime often went into the hours, so we often had the approach of designing a model such that we hoped it would work as we intended, since we knew any small change thereafter would take hours to remedy, furthering dampening collaboration due to scheduling concerns and dependencies.
3. the data we used was publicly sources as listing prices, which can be arbitarily set as desired - there can be no access inherently to the actual *sale* price of these cars, which is a much more valuable metric for prediction

4. since these are market listings, we do not know if sales ever took place as a result of these

Ethics and Privacy

Given the nature of the project, we believe that we will not be in possession of any personally identifiable information that holds a risk to the persons in question. However, we understand that in processing vehicular sales data, the VIN or identifiable information pertaining to the car is revealed in sales ads, and so on. This provides us with records of car maintenance and thereby the locations of previous owners, which may hold a risk. However, this is not unique to the dataset, but rather openly provided information for the sake of transparency - products such as CARFAX have been using this information for buyer transparency without any apparent ethical violations. Additionally, the inclusion of coordinates as far as where a vehicle is located presents itself as a privacy risk as well, and was omitted not only due to this, but also due to being inaccurate since many listings were found to originate from the middle of the ocean (thereby also mitigating the privacy issue).

In that sense, we are aware but not concerned about privacy risks We spent significant time trying to imagine the ethical implications of such a project. It is entirely true that every machine learning project is liable to have intrinsic interactions with ethics, but in this case we think that these pertain to how such a project can be used, rather than in the making of the project. For example, the outputs of our model could be used to monitor car prices, and manipulate sales as opposed to an organic market. However, such metrics of what cars are “worth” and “market value” have already been established, and we believe the likely usage of this model would be to track the perpetuation of car values in a market heavily impacted by the pandemic and its ensuing shortages in components.

Conclusion

Through our exploratory data analysis, general trends, such as price vs. mileage, are evident, but our models ultimately did not attain high accuracies in correctly predicting car list prices - i.e., 0.51 score for linear regression. We believe that future approaches when greater computational power is availed to us could deliver better results, such that we could use GridSearch with proper parameters or neural networks.

Footnotes

1.^: Shen, S. (13 Feb 2022) Used cars cost 40.5% more than last year as gas prices rise. New car prices also climbing. https://www.usatoday.com/story/money/cars/2022/02/13/used-cars-cost-more/6778705001/

2.^: (4 Oct 2021) State by state: Here's how much used car prices went up in August https://www.newsnationnow.com/business/your-money/state-by-state-heres-how-much-used-car-prices-went-up-in-august/