Price prediction of used cars

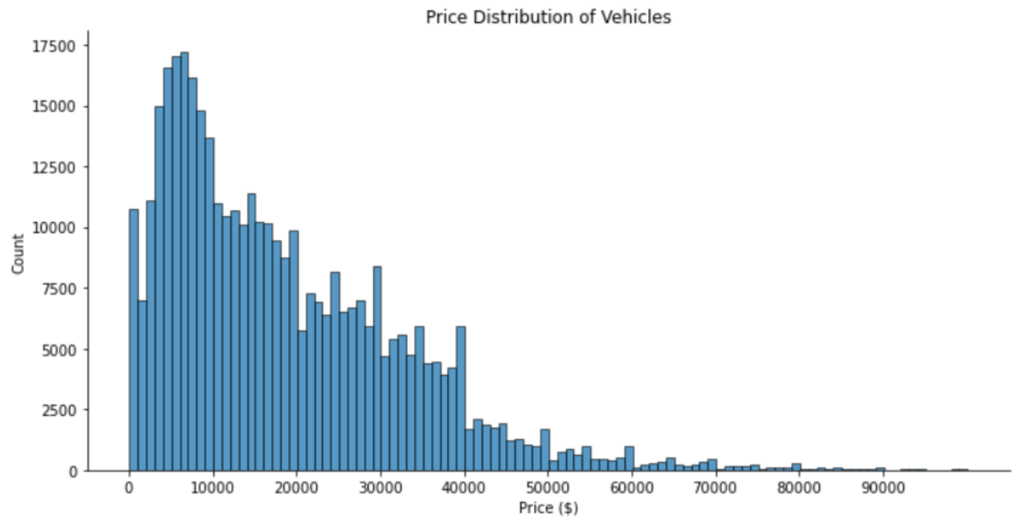
- TEAM MEMBERS:
- MAHALAKSHMI SAI MADHURI PRAKRUTHI (11517836)
- SINDHUJA CHEPURI (11121258)
- SNIGDHA RAO GUJJA (11546755)

Abstract

- The demand for the used cars in the market is increasing day by day due to chipset shortage in the market due to covid. The project aim is to build a machine learning model which predicts the prices of used cars accurately.
- The model building involves the feature selection, transformation, statistical tests and dataset cleaning.
- Accuracy is tested across different regression models like linear regression, Kneighbours regression, Random forest, HGB and XGB

Exploratory Data Analysis

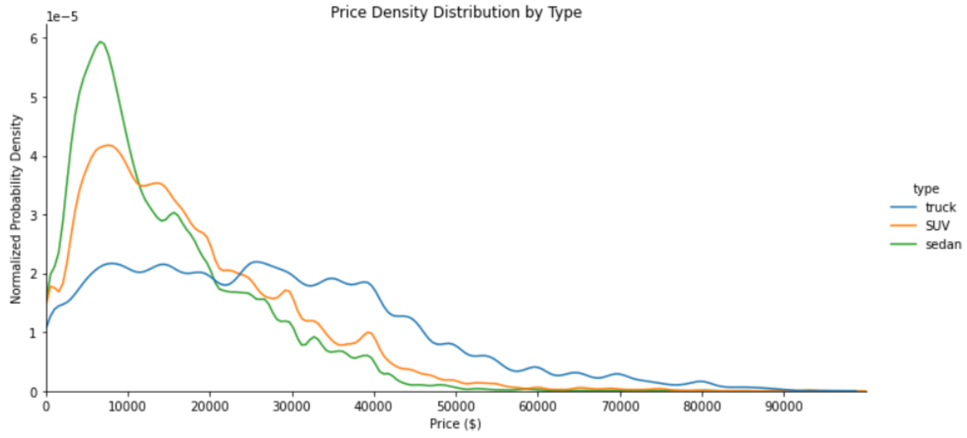
```python
sns.displot(df, x='price', binwidth=1000, height=5, aspect=2)#, bw_adjust=0.4)
plt.xticks(range(0,int(1e5), int(1e4)))
plt.xlabel('Price ($)')
plt.title('Price Distribution of Vehicles')
plt.show()
```



We plotted distplot using seaborn to check the price distribution of all the vehicles. We can see more cars are present in the range of 5000 to 13000 price range.

Exploratory Data Analysis

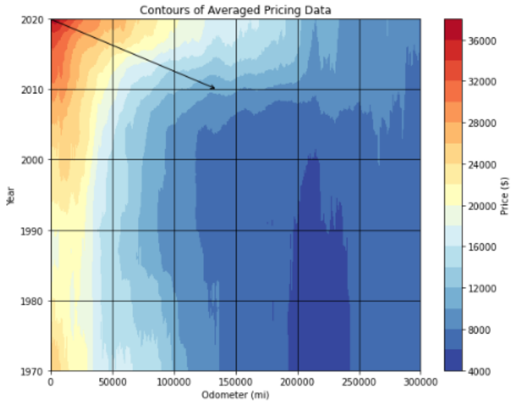
```
cars_plt = df[df.type.isin(['sedan', 'SUV', 'truck'])]
sns.displot(cars_plt, x='price', hue='type', kind='kde', bw_adjust=0.6, cut=0, common_norm=False, height=5, aspect=2)
plt.xticks(range(0,int(1e5), int(1e4)))
plt.xlabel('Price ($)')
plt.xlim(0,int(1e5))
plt.ylabel('Normalized Probability Density')
plt.title('Price Density Distribution by Type')
plt.show()
```



Prices of the sedan are more when compared with SUV and truck.

Exploratory Data Analysis

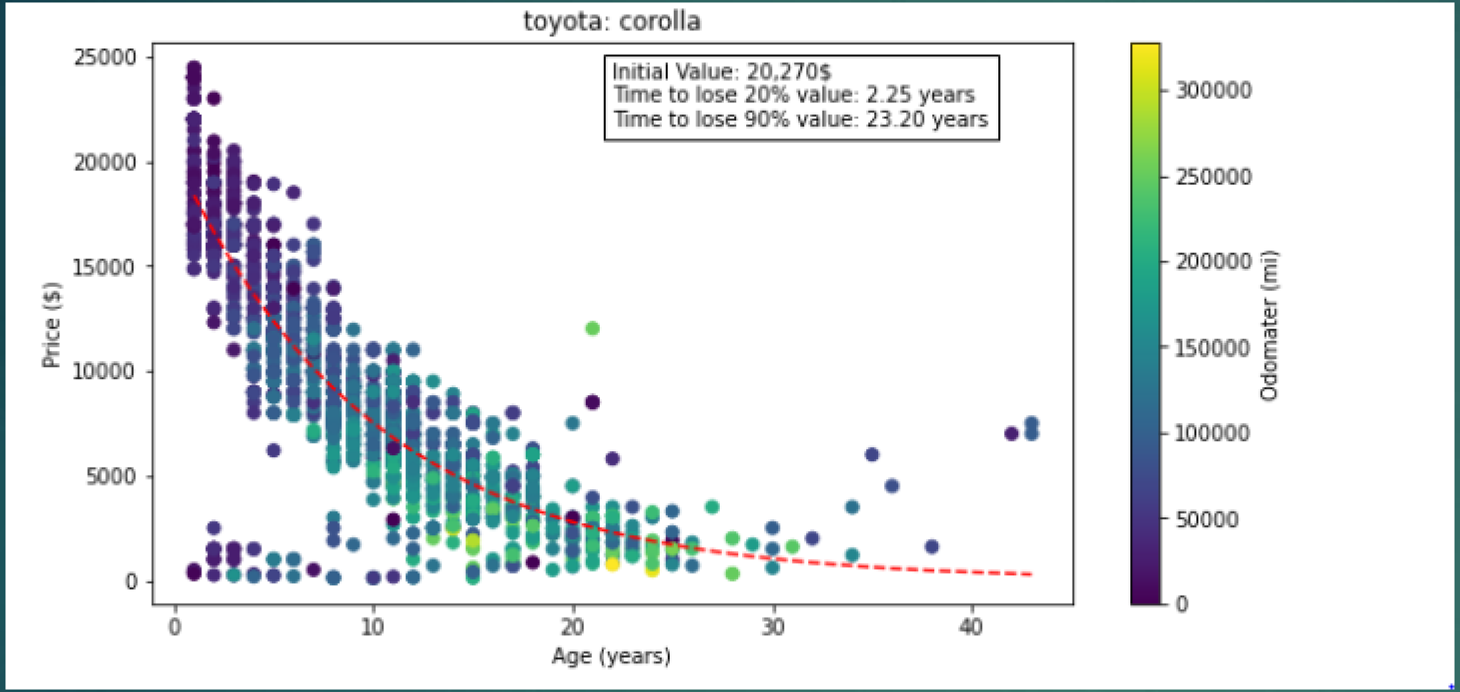
```
from scipy.signal import convolve2d
sz_o = 500
sz_y = 3
kernel = np.ones((sz_y,sz_o))/(sz_y*sz_o)
grid_z0f = convolve2d(grid_z0, kernel, boundary='symm', mode='same')

fig, ax = plt.subplots(1, figsize=(9,7))
im = ax.contourf(grid_x, grid_y, grid_z0f, levels=15, cmap='RdYlBu_r', zorder=0)
cbar = fig.colorbar(im, ax=ax)
cbar.set_label('Price ($)')
ax.set_xlim(0, 3e5)
ax.set_xlabel('Odometer (mi)')
ax.set_ylabel('Year')
ax.set_title('Contours of Averaged Pricing Data')
ax.grid(True, color='k')
ax.annotate("", xy=(1.35e5, 2010), xytext=(0, 2020), arrowprops=dict(arrowstyle="->", color='k'))
plt.show()

xloc_e = np.where((1.349e5<grid_x[0]) & (grid_x[0]<1.36e5))
yloc_e = 40
price_end = grid_z0f[yloc_e,xloc_e[0]]
yloc_s = 50
xloc_s = np.where(grid_x[0]==0)
price_start = grid_z0f[yloc_s,xloc_s[0]]
depr_rate = ((price_start-price_end)/1.35e5)[0]
print('Benchmark Depreciation rate: ${:.2f}/mi'.format(depr_rate))
```




```
Benchmark Depreciation rate: $0.19/mi
```

We used contour plot to check the depreciation rate of price of the car with odometer reading it is $0.19/mile

Exploratory Data Analysis



We took Toyota corolla as an example to check the time it took to lose the value of 20 and 90%.

As Toyota is most efficient car among the used cars list.

Data Specification

- These two screenshots shows each column data type and datset description like number of rows and columns.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 426880 entries, 0 to 426879
Data columns (total 26 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   id            426880 non-null  int64
 1   url           426880 non-null  object
 2   region        426880 non-null  object
 3   region_url    426880 non-null  object
 4   price         426880 non-null  int64
 5   year          425675 non-null  float64
 6   manufacturer  409234 non-null  object
 7   model         421603 non-null  object
 8   condition     252776 non-null  object
 9   cylinders     249202 non-null  object
 10  fuel          423867 non-null  object
 11  odometer      422480 non-null  float64
 12  title_status  418638 non-null  object
 13  transmission  424324 non-null  object
 14  VIN           265838 non-null  object
 15  drive         296313 non-null  object
 16  size          120519 non-null  object
 17  type          334022 non-null  object
 18  paint_color   296677 non-null  object
 19  image_url     426812 non-null  object
 20  description   426810 non-null  object
 21  county        0 non-null       float64
 22  state         426880 non-null  object
 23  lat           420331 non-null  float64
 24  long          420331 non-null  float64
 25  posting_date  426812 non-null  object
dtypes: float64(5), int64(2), object(19)
memory usage: 84.7+ MB
```

```
df.shape
```

```
(426880, 26)
```

```
df.columns
```

```
Index(['id', 'url', 'region', 'region_url', 'price', 'year', 'manufacturer',
       'model', 'condition', 'cylinders', 'fuel', 'odometer', 'title_status',
       'transmission', 'VIN', 'drive', 'size', 'type', 'paint_color',
       'image_url', 'description', 'county', 'state', 'lat', 'long',
       'posting_date'],
      dtype='object')
```

Data specification

- Dataset contains total of 426880 records and 26 columns
- Id -> unique identification number for the car
- url -> url to access the car details
- Region -> Region where the car is available for sale
- Region_url -> url for the region
- Price -> Price of the car
- Year -> Year the car manufactured
- Manufacturer -> manufacturer name
- Model -> model of the car
- Condition -> car condition
- Cylinders -> number of cylinders for the car
- Fuel -> car fuel type
- Odometer -> number of miles driven
- Title_status -> car title
- Transmission -> transmission type

Data specification

- VIN -> vehicle identification number
- Drive -> rear or forward
- Size -> mid or full size
- Type -> type of the car (sedan, truck, suv)
- Paint_color -> color of the car
- Image_url -> url of the car image
- Description -> description of the car
- County -> county
- State -> state the car belongs to
- Lat -> region latitude
- Long -> region longitude
- Posting_date -> Date it was posted for sale

Data Transformation

```
from sklearn.preprocessing import LabelEncoder

cat_columns = ['manufacturer', 'condition', 'cylinders', 'fuel', 'transmission', 'drive', 'size', 'type']

le = {}

for col in cat_columns:
    if col in df.columns:
        le[col] = LabelEncoder()
        le[col].fit(list(df[col].astype(str).values))
        df[col] = le[col].transform(list(df[col].astype(str).values))
```

```
df.head()
```

	price	manufacturer	condition	cylinders	fuel	odometer	transmission	drive	size	type	age	make_model
31	15000	13	0	5	2	128000.0	0	2	1	10	8.0	ford: f-150 xlt
55	19900	13	2	6	0	88000.0	0	0	1	8	17.0	ford: f250 super duty
59	14000	16	0	5	2	95000.0	0	1	1	5	9.0	honda: odyssey
65	22500	13	2	6	0	144700.0	1	2	1	10	20.0	ford: f450
73	15000	10	0	6	2	90000.0	0	2	2	9	4.0	dodge: charger rt 4dr sedan

```
df = df.drop(columns='make_model')
y = df['price']
X = df.drop(columns='price')
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

We took least features and label encoded them to get simpler model.

And we used standard scaler to keep all the values of the columns on same scale.

The reason for scaling the values is while training the model all variables will have same weightage or importance, To avoid the model bias.

Statistical Analysis

- We performed T test to check the null hypothesis which states that the average price of the car is 18000$.
- After conducting the one sample T test we got the p value as 0.00215 which is less than the significant value 0.05.
- We can reject null hypothesis and conclude that the average price of the car is more than that.

```
In [228]: stats.ttest_1samp(a=df['price'],popmean=18000)

Out[228]: Ttest_1sampResult(statistic=3.0677003608216173, pvalue=0.002157262973519216)
```

Statistical Analysis

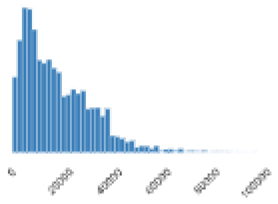
- We used profiler report to check the variables report like minimum, maximum, average and missing value etc..

price

Real number ($\mathbb{R}_{\geq 0}$)

HIGH CORRELATION

Distinct	14995	Minimum	101
Distinct (%)	3.9%	Maximum	99999
Missing	0	Zeros	0
Missing (%)	0.0%	Zeros (%)	0.0%
Infinite	0	Negative	0
Infinite (%)	0.0%	Negative (%)	0.0%
Mean	18761.55647	Memory size	3.0 MiB



Toggle details

Resources

- We checked the approaches did by the other people in the Kaggle.
- But in addition to that we did more exploratory data analysis and we tried to predict the prices of the cars by using different regression methods, we compared all of them and finally went with XGBoost that you can find in our notebook which we posted in Github.

Design and Milestones

- To train the model we used Anaconda Navigator, Jupyter notebook and python.
- We used pandas to store the dataset as dataframe, seaborn and matplotlib to visualize the data for exploratory data analysis.
- We imported many libraries using anaconda prompt.
- We tested the accuracies of different models like linear, Random forest, HGB and XGB.
- Finally, we found XGB is more efficient when compared with others.

Design and Milestones

	models	R2	RMSE
0	LinearRegression()	0.258343	7805.706454
1	Ridge()	0.258342	7805.709323
2	Lasso()	0.258326	7805.794741
3	BayesianRidge()	0.258327	7805.788932
4	SVR()	0.002994	9050.225747
5	KNeighborsRegressor()	0.745327	4574.058555
6	DecisionTreeRegressor()	0.626734	5537.578861
7	(DecisionTreeRegressor(random_state=1956220559...	0.776434	4285.618783
8	(DecisionTreeRegressor(max_features='auto', ra...	0.793266	4121.134721
9	(DecisionTreeRegressor(max_depth=3, random_sta...	0.371734	7184.268381
10	([DecisionTreeRegressor(criterion='friedman_ms...	0.772928	4319.090323
11	HistGradientBoostingRegressor()	0.816486	3882.801576
12	MLPRegressor()	0.696407	4994.092320
13	XGBRegressor(base_score=0.5, booster='gbtree',...	0.819157	3854.442348

We compared R2 and Root mean square error values of different models to select the suitable model for this problem.

We found XGBRegressor, Random forest and HistGradientBossting Regressor are having more R2 values when compared with others.

Design and Milestones

```
parameters = {'n_estimators': np.arange(125, 175, 5), 'max_depth': np.arange(4, 9, 1), 'eta': np.arange(0.1, 0.4, 0.1),
              'subsample': [1], 'colsample_bytree': [1]}

model_randomCV(X_train, y_train, XGB, parameters)
```

```
best_parameters: {'subsample': 1, 'n_estimators': 165, 'max_depth': 6, 'eta': 0.2, 'colsample_bytree': 1}
best_score: 0.8128210525434121
best_estimator: XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
             early_stopping_rounds=None, enable_categorical=False, eta=0.2,
             eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
             importance_type=None, interaction_constraints='',
             learning_rate=0.200000003, max_bin=256, max_cat_to_onehot=4,
             max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
             missing=nan, monotone_constraints='()', n_estimators=165, n_jobs=0,
             num_parallel_tree=1, predictor='auto', random_state=0, reg_alpha=0, ...)
```

```
XGB.fit(X_train, y_train)
```

```
XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
             early_stopping_rounds=None, enable_categorical=False,
             eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
             importance_type=None, interaction_constraints='',
             learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
             max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
             missing=nan, monotone_constraints='()', n_estimators=100, n_jobs=0,
             num_parallel_tree=1, predictor='auto', random_state=0, reg_alpha=0,
             reg_lambda=1, ...)
```

We trained the XGB model using best fine tuned parameters to get more accurate results.

Results

```python
def acc_CV(model, X, y):
    from sklearn.model_selection import cross_val_score

    accuracies = cross_val_score(estimator = model, X= X, y=y,  cv=10)
    accuracies.mean()
    accuracies.std()
    print('Accuracy  {:.2f}% +/- {:.2f}%' .format(accuracies.mean()*100, accuracies.std()*100))
```

```python
rf = RandomForestRegressor()
HGB = HistGradientBoostingRegressor()
XGB = XGBRegressor()
```

```python
acc_CV(rf, X_train, y_train)
```

```
Accuracy  78.77% +/- 0.67%
```

```python
acc_CV(HGB, X_train, y_train)
```

```
Accuracy  80.64% +/- 0.47%
```

```python
acc_CV(XGB, X_train, y_train)
```

```
Accuracy  80.99% +/- 0.47%
```

```python
def predicted_price(manufacturer, condition, cylinders, fuel, odometer, transmission, drive, size, type, age):

    x = np.zeros(10)
    x[0] = le['manufacturer'].transform([manufacturer])
    x[1] = le['condition'].transform([condition])
    x[2] = le['cylinders'].transform([cylinders])
    x[3] = le['fuel'].transform([fuel])
    x[4] = odometer
    x[5] = le['transmission'].transform([transmission])
    x[6] = le['drive'].transform([drive])
    x[7] = le['size'].transform([size])
    x[8] = le['type'].transform([type])
    x[9] = age

    x = scaler.transform([x])


    return XGB.predict(x)
```

```python
predicted_price('toyota', 'excellent', '4 cylinders', 'gas', 1000000.0 , 'automatic', 'rwd', 'mid-size', 'sedan', 20)
```

```
array([5155.77], dtype=float32)
```

```python
predicted_price('toyota', 'excellent', '4 cylinders', 'gas', 1000000.0 , 'automatic', 'rwd', 'mid-size', 'sedan', 1)
```

```
array([22898.25], dtype=float32)
```

```python
predicted_price('ford', 'excellent', '4 cylinders', 'gas', 1000000.0 , 'automatic', 'rwd', 'mid-size', 'sedan', 1)
```

```
array([17779.16], dtype=float32)
```

We predicted the prices of the car in this screenshot using XGB model. These are the prices what we got on the test data.

It was almost accurate with accuracy rate of 80.99%.

Repository link

- We posted the complete code which we accomplished using jupyter notebook.
- We uploaded the file in the github along with instructions how to use it. Here is the link for the github
- https://github.com/MahalakshmiSaiMadhuriPrakruthi/Used-Cars-Price-Prediction


Thank you