In [1]:
```
import pandas as pd
import mysql.connector as connection
```

In [2]:
```
try:
    db = connection.connect(host="127.0.0.1", database='car', user="roo
t", passwd="", use_pure=True)
    query = "select cs.year, cs.listing_mileage, cs.listing_price_in_cen
ts from car_solds cs inner join car_models mo on cs.car_model_id = mo.id
inner join car_makes ma on mo.car_make_id=ma.id where ma.name like '%Ki
a%';"
    df = pd.read_sql(query, db)
    db.close()
except Exception as e:
    print(str(e))
finally:
    db.close()
```

In [3]:
```
df.head()
```

Out[3]:

	year	listing_mileage	listing_price_in_cents
0	2015	53960.0	0
1	2015	86967.0	0
2	2015	NaN	0
3	2021	32261.0	2297800
4	2023	15.0	2389300

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32241 entries, 0 to 32240
Data columns (total 3 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   year                   32241 non-null  object
 1   listing_mileage        28782 non-null  float64
 2   listing_price_in_cents  32241 non-null  int64
dtypes: float64(1), int64(1), object(1)
memory usage: 755.8+ KB
```

In [5]:
```
df.describe()
```

Out[5]:

	listing_mileage	listing_price_in_cents
count	28782.000000	3.224100e+04
mean	48567.133486	2.360502e+06
std	43320.342441	1.317359e+06
min	1.000000	0.000000e+00
25%	13997.000000	1.612200e+06
50%	40253.000000	2.310900e+06
75%	73069.750000	3.053000e+06
max	557784.000000	9.999900e+06

In [6]:
```
df.duplicated().sum()
```

Out[6]:
```
3558
```

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
year                        0
listing_mileage          3459
listing_price_in_cents      0
dtype: int64
```

In [8]:
```
df.dropna(subset=['listing_mileage', 'listing_price_in_cents'], inplace=
True)
df = df[df['listing_price_in_cents'] != 0]
final_data = df[['year', 'listing_mileage', 'listing_price_in_cents']]
final_data.head()
```

Out[8]:

	year	listing_mileage	listing_price_in_cents
3	2021	32261.0	2297800
4	2023	15.0	2389300
5	2020	29583.0	2359000
6	2020	83762.0	1987800
9	2022	10.0	2229000

In [9]:
```
final_data.isnull().sum()
```

Out[9]:
```
year                     0
listing_mileage          0
listing_price_in_cents   0
dtype: int64
```

In [10]:
```
final_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 26244 entries, 3 to 32240
Data columns (total 3 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   year                   26244 non-null  object 
 1   listing_mileage        26244 non-null  float64
 2   listing_price_in_cents  26244 non-null  int64  
dtypes: float64(1), int64(1), object(1)
memory usage: 820.1+ KB
```

In [11]:
```
final_data.shape
```

Out[11]:
```
(26244, 3)
```

In [12]:
```
final_data.head()
```

Out[12]:

	year	listing_mileage	listing_price_in_cents
3	2021	32261.0	2297800
4	2023	15.0	2389300
5	2020	29583.0	2359000
6	2020	83762.0	1987800
9	2022	10.0	2229000

In [13]:
```
final_data['year'] = pd.to_numeric(df['year'], errors='ignore')
final_data.info()
final_data['age'] = 2022 - final_data['year']
final_data.drop(['year'], axis = 1, inplace = True)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 26244 entries, 3 to 32240
Data columns (total 3 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   year                   26244 non-null  int64  
 1   listing_mileage        26244 non-null  float64
 2   listing_price_in_cents  26244 non-null  int64  
dtypes: float64(1), int64(2)
memory usage: 820.1 KB
```

In [14]:
```
final_data.head()
```

Out[14]:

	listing_mileage	listing_price_in_cents	age
3	32261.0	2297800	1
4	15.0	2389300	-1
5	29583.0	2359000	2
6	83762.0	1987800	2
9	10.0	2229000	0

In [15]:
```
df.shape
```

Out[15]:
```
(26244, 3)
```

In [16]:
```
import numpy as np


def detect_outlier(data):
    outlier = []
    threshold = 3
    mean = np.mean(data)
    std = np.std(data)
    for i in data:
        z_score = (i - mean)/std
        if np.abs(z_score)>threshold:
            outlier.append(i)
    return outlier
```

In [17]:
```
outlier_prices = detect_outlier(final_data['listing_price_in_cents'].to_
list())
outlier_prices
```

Out[17]:
```
[6000000,
 7259300,
 6180000,
 7259300,
 5899800,
 5899800,
 5999800,
 6050000,
 6390000,
 6899500,
 6799500,
 5998900,
 6050000,
 6390000,
 6899500,
 6799500,
 5998900,
 6499800,
 6200000,
 6493100,
 5998700,
 6159000,
 6199700,
 5968800,
 6045000,
 6164100,
 6059000,
 6918300,
 6488800,
 6101000,
 6656000,
 5959000,
 6101000,
 6499500,
 6094400,
 5999500,
 6196400,
 6899500,
 6098800,
 6199000,
 5999800,
 6880000,
 5868100,
 9999900,
 6929500,
 7201500,
 6209500,
 5966000,
 6452400,
 6017500,
 6133500,
 5988800,
 6092000,
 5894000,
 6099800,
 6299800,
 6799900,
```

```
6299400,
5959000,
6199000,
5903000,
7499900,
6199800,
5890000,
6580000,
6252000,
5955300,
5992500,
5994700,
6199000,
6599500,
5898700,
5898800,
5885100,
5973400,
5980000,
5899500,
5995000,
5961500,
6499900,
5949900,
5977800,
6399900,
6499500,
6078000,
6078000,
6079000,
6458500,
6090000,
6598800,
6577400,
6186500,
5999900,
6224000,
5864000,
6798000,
6004000,
6308500,
6375100,
6000500,
6100000,
6271000,
6097500,
6102500,
6253200,
6115500,
6199800,
6498400,
5950000,
6225000,
5969000,
5936500,
6007000,
6998700,
```

```
6449500,
6749500,
6750000,
6799100,
6036000,
6199000,
5999700,
6098500,
6594300,
6499000,
7380900,
6209500,
6588800,
6099000,
5904000,
6436500,
5899500,
6078000,
5981500,
9999900,
6035000,
6101500,
5884500,
6008500,
6173500,
5971000,
5969100,
9999900,
6103000,
6031000,
6069000,
6098000,
6098000,
6480000,
6465800,
5896500,
6198700,
5900000,
6410000,
6998800,
5899500,
6102300,
5894500,
6099000,
5995900,
5897300,
6412600,
5865900,
6680000,
5990000,
5999900,
5999700,
6601700,
5987700,
6499900,
6519100,
6499900,
```

```
6456500,
6193400,
5995000,
5999500,
6600000,
5880000,
6298800,
5888700,
6069700,
6255700,
6098900,
6388800,
5999500,
5898800,
6129900,
5959000,
6000000,
5999500,
6090000,
5902500,
5963000,
6590000,
6297200,
5999900,
6288800,
6099500,
6125500,
6036000,
6499100,
5885500,
5897100,
5994000,
5918000,
5889500,
6398800,
5929900,
6788800,
6399900,
5874900,
6299100,
5990000,
5999800,
6298800,
5899900,
5899500,
5997500,
5997500,
5898900,
6043500,
6479900,
6707900,
6249200,
6111300]
```

In [18]:
```
outlier_mileages = detect_outlier(final_data['listing_mileage'].to_list
())
outlier_mileages
```

Out[18]:
```
[209384.0,
 209384.0,
 241436.0,
 190321.0,
 220900.0,
 187000.0,
 232400.0,
 177980.0,
 194483.0,
 192322.0,
 192000.0,
 178500.0,
 185515.0,
 212145.0,
 193855.0,
 181441.0,
 202349.0,
 183000.0,
 221000.0,
 177902.0,
 176561.0,
 175362.0,
 176172.0,
 195000.0,
 199142.0,
 180294.0,
 189085.0,
 182545.0,
 203959.0,
 186049.0,
 215885.0,
 194118.0,
 187112.0,
 220519.0,
 199142.0,
 214000.0,
 199142.0,
 174800.0,
 269000.0,
 190825.0,
 200115.0,
 212177.0,
 199240.0,
 208552.0,
 176257.0,
 224881.0,
 184335.0,
 180221.0,
 239527.0,
 179702.0,
 174675.0,
 178000.0,
 203128.0,
 192927.0,
 179776.0,
 177773.0,
 208000.0,
```

```
223399.0,
196403.0,
202467.0,
184101.0,
241295.0,
190000.0,
190413.0,
184000.0,
182072.0,
226275.0,
186109.0,
232800.0,
244975.0,
191545.0,
177010.0,
177076.0,
199142.0,
175068.0,
177177.0,
178566.0,
180488.0,
205885.0,
218169.0,
207000.0,
177048.0,
189921.0,
190776.0,
177090.0,
212123.0,
175840.0,
181153.0,
179900.0,
204567.0,
178994.0,
185400.0,
193504.0,
177436.0,
219204.0,
230224.0,
184857.0,
178391.0,
177324.0,
196000.0,
196762.0,
228035.0,
185000.0,
251073.0,
197032.0,
199142.0,
185999.0,
182194.0,
175393.0,
210563.0,
292933.0,
200000.0,
190253.0,
186900.0,
```

```
195950.0,
185450.0,
207838.0,
182004.0,
182004.0,
189000.0,
180000.0,
181382.0,
180403.0,
216397.0,
180200.0,
264413.0,
206000.0,
182347.0,
220891.0,
218000.0,
176995.0,
205322.0,
181010.0,
195367.0,
202364.0,
184721.0,
202434.0,
234008.0,
190000.0,
183580.0,
185833.0,
183810.0,
174501.0,
230776.0,
175387.0,
181010.0,
175470.0,
182241.0,
188000.0,
214767.0,
192814.0,
183720.0,
215005.0,
182448.0,
192426.0,
227778.0,
181000.0,
210520.0,
199142.0,
193697.0,
202656.0,
253957.0,
188860.0,
184000.0,
187000.0,
185952.0,
214040.0,
186020.0,
174709.0,
181870.0,
228642.0,
```

```
220200.0,
203100.0,
209546.0,
191178.0,
175460.0,
223811.0,
230919.0,
174814.0,
174814.0,
176000.0,
176000.0,
178365.0,
217000.0,
207257.0,
218703.0,
178805.0,
557784.0,
350000.0,
199142.0,
194046.0,
208000.0,
222957.0,
183480.0,
187146.0,
219115.0,
212933.0,
237653.0,
186200.0,
181858.0,
178145.0,
182407.0,
241077.0,
238711.0,
174563.0,
186515.0,
223944.0,
186217.0,
224214.0,
217000.0,
190852.0,
211988.0,
214707.0,
246911.0,
229123.0,
190000.0,
204000.0,
177360.0,
230000.0,
191439.0,
256000.0,
208000.0,
206859.0,
178479.0,
176580.0,
198212.0,
175543.0,
188177.0,
```

```
219355.0,
184672.0,
277918.0,
175988.0,
197401.0,
236521.0,
185387.0,
179505.0,
205287.0,
188215.0,
216882.0,
185937.0,
199530.0,
194896.0,
199565.0,
183000.0,
198255.0,
182020.0,
199660.0,
250000.0,
183650.0,
185717.0,
214626.0,
183000.0,
228300.0,
181772.0,
245656.0]
```

In [19]:
```
if len(outlier_prices) > 0:
    final_data.drop(final_data[final_data['listing_price_in_cents'] >= o
utlier_prices[0]].index, inplace = True)
if len(outlier_mileages) > 0:
    final_data.drop(final_data[final_data['listing_mileage'] >= outlier_
mileages[0]].index, inplace = True)
```

In [20]:
```
final_data.shape
```

Out[20]:
```
(26027, 3)
```

In [21]:
```
from sklearn.metrics import r2_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split

x = final_data.drop(['listing_price_in_cents'], axis = 1)
y = final_data['listing_price_in_cents']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.
3, random_state = 0)
rf = RandomForestRegressor(n_estimators = 100,
                           criterion = 'mse',
                           random_state = 20,
                           n_jobs = -1)
# rf = RandomForestRegressor(bootstrap=False, max_depth=15, max_features
='sqrt', min_samples_split=2, n_estimators=100)
rf.fit(x_train,y_train)
rf_train_pred = rf.predict(x_train)
rf_test_pred = rf.predict(x_test)


r2_score(y_test, rf_test_pred)
```

Out[21]:
```
0.4444735015361134
```

In [22]:
```
#mileage, age
rf.predict([[80, 0]])[0]/100
```

Out[22]:
```
51422.34
```

In [23]:
```
import pickle
# open a file, where you ant to store the data
file = open('../models/car_value_predict_note.pkl', 'wb')

# dump information to that file
pickle.dump(rf, file)
```