Assignment – Rodolfo Lerma

The first part of this analysis (as seen in the appendix) was to explore the data to get a better idea of what kind of data that I have available for this analysis and identify where some cleaning and/or formatting might be needed and get familiar with the dataset in general. After this section and the identification of the areas that might need improvement, I started working on cleaning and formatting the data. During this step I decided to normalize the numerical columns, change the format from the Boolean columns to numeric (0 & 1), drop missing values as they were around 15% of the training dataset (another alternative if could have been to replace the missing values for the median in case of numerical columns and the mode for categorical ones, which is what a did for the test data set as the missing values where around 5%), get one hot encodings for some of the categorical columns and to extract as much information as possible from the 2 columns that were formatted as a list of lists (VehFeats & VehHistory).

Once the data was clean and with the needed formatting, I decided to work on 2 different models: A **regression** for the prediction of the Dealer Listing Price and a **classifier** for the Vehicle Trim.

The first part of the **regression** model was to do a feature selection for the features that would be part of the model, for this section I went for an embedded method (Lasso) as it provides a way to minimize to zero those coefficients that contribute the least to the regression. For the Regression model itself I used the ElasticNet model from the sklearn – linear model package. I split the Training dataset into "Training" & "Validation" and selected multiple hyperparameters to select the best combination of values which I check by means of cross validation and looking at the r squared (it is important to notice that other measures of performance could have been used, such as mean squared error. This relates to the end use of the model). After this process the training and validation datasets were combined to train the selected model again and use this last model as the final one for the test data.

For the second model (**classifier**) I followed the same structure as before: Feature selection (used Lasso as well), did a comparison of 3 models looking for the best hyperparameters for each of the classifiers (3 legacy machine learning models) using the "RandomizedSearchCV" for Cross Validation. After looking at the performance of the validation dataset and verifying that no overfitting was present, I re-trained the model using the entire "Training dataset".

I did no used deep learning because the dataset was too small for the noted application, and I believe interpretability was important for this model, but maybe an option to explore for the future. (A basic Deep Learning Section was added at the end of the analysis).

BOEING ASSIGNMENT

Applicant: Rodolfo Lerma

Probem Statement:

Included in the training dataset is information on used cars previously sold. Each row corresponds to one used car listing. The first column of the data contains a unique identifier for the listing. The next twenty-six columns contain information on parameters relevant to the transaction, with those parameters described in more detail in the appendix attached. Finally, the last two columns of the `Training_dataset.csv` contain information on `Vehicle_Trim` and `Dealer_Listing_Price`, which describe the trim of the vehicle involved in the sale, and the price at which the vehicle was listed by the dealer. Your challenge is to build one or more models, through whatever means you find most appropriate, capable of predicting **vehicle trim** and **dealer listing price** given the other twenty-six variables provided.

ABSTRACT:

The analysis is structured in the following way:

Analysis - Prediction (`Price & Trim`)

- **Data Exploration**
  - Training Dataset
  - Testing Dataset
  - General Exploration
  - Distribution of Target Variables
  - Distribution of Numerical Variables
  - Categorical Variables
    - Unique Values Training Dataset
- **Data Formatting (Training Dataset & Testing Dataset)**
  - Missing Values
  - Removing Columns from Dataset
  - Formatting Boolean Variables
  - Normalizing Numerical Variables
  - One Hot Encoding For Categorical Columns
  - Formatting of two *List of Lists* Columns (`VehHistory` & `VehFeats`)
    - `VehFeats`
    - `VehHistory`
  - Splitting the dataset into Target Variables and Features (Training Dataset)
- **Model 1: Prediction of `Dealer Listing Price`**
  - Feature Selection (LASSO) for `Dealer Listing Price` Variable
  - Model for `Dealer Listing Price` prediction
  - Training
  - Results/Predictions (Training & Validation dataset)
  - Re-train the model but now with all the datapoints (`training` + `validation`)
    - Prediction using the `Test_data`
- **Model 2: Prediction of `Vehicle_Trim`**
  - Feature Selection (LASSO) for `Vehicle_Trim` Variable
  - Different Models and Hyperparameters
    - Hyperparameter Settings for the `Decision Tree` Model
    - Hyperparameter Settings for the `Random Forest` Model
    - Hyperparameter Settings for the `Logistic Regression` Model
    - Comparison & Final Model Selection
  - Final Model 2 Selection and Training (using `training` + `validation` data)
- **Export to Dataframe**

Analysis - Prediction (`Price & Trim`)

Data Exploration

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
Test_df = pd.read_csv('Test_Dataset.csv')
Training_df = pd.read_csv('Training_Dataset.csv')
```

Training Dataset

In [3]:
```
Training_df.head(5)
```

Out[3]:

	ListingID	SellerCity	SellerIsPriv	SellerListSrc	SellerName	SellerRating	SellerRevCnt	SellerState	SellerZip	VehBodystyle	...	VehMake	V
0	3287	Warren	False	Inventory Command Center	Prime Motorz	5.0	32	MI	48091.0	SUV	...	Jeep	
1	3920	Fargo	False	Cadillac Certified Program	Gateway Chevrolet Cadillac	4.8	1456	ND	58103.0	SUV	...	Cadillac	
2	4777	Waukesha	False	Jeep Certified Program	Wilde Chrysler Jeep Dodge Ram & Subaru	4.8	1405	WI	53186.0	SUV	...	Jeep	
3	6242	Wentzville	False	Inventory Command Center	Century Dodge Chrysler Jeep RAM	4.4	21	MO	63385.0	SUV	...	Jeep	
4	7108	Fayetteville	False	HomeNet Automotive	Superior Buick GMC of Fayetteville	3.7	74	AR	72703.0	SUV	...	Cadillac	

5 rows × 29 columns

In [4]:
```
Training_df.shape
```

Out[4]: (6298, 29)

It is possible to see that the Training Data Set contains 29 columns and 6298 data points. Out of those 29 columns, 2 are the `target` variables.

Testing Dataset

In [5]:
```
Test_df.head(5)
```

Out[5]:

	ListingID	SellerCity	SellerIsPriv	SellerListSrc	SellerName	SellerRating	SellerRevCnt	SellerState	SellerZip	VehBodystyle	...	VehHistory
0	8622015	Seneca	False	HomeNet Automotive	Lake Keowee Chrysler Dodge Jeep Ram	2.5	59	SC	29678	SUV	...	1 Owner, Non-Personal Use Reported, Buyback Pr...
1	8625693	Bedford	False	Inventory Command Center	North Coast Auto Mall	4.7	2116	OH	44146	SUV	...	1 Owner, Accident(s) Reported, Non-Personal Us...
2	8625750	Webster	False	Jeep Certified Program	Marina Chrysler Dodge Jeep Mitsubishi RAM	3.9	46	NY	14580	SUV	...	1 Owner, Buyback Protection Eligible
3	8626885	Louisville	False	Digital Motorworks (DMi)	Oxmoor Ford Lincoln	4.5	1075	KY	40222	SUV	...	1 Owner, Buyback Protection Eligible
4	8627430	Palmyra	False	Digital Motorworks (DMi)	F.C. Kerbeck & Sons	4.6	162	NJ	8065	SUV	...	1 Owner, Non-Personal Use Reported, Buyback Pr...

5 rows × 27 columns

In [6]:
```
Test_df.shape
```

Out[6]: (1000, 27)

It is possible to see that the Training Dataset contains 27 columns and 1000 data points, this since the 2 `target` variables were removed from the dataset.

General Exploration

In [7]:
```
Training_df.dtypes
```

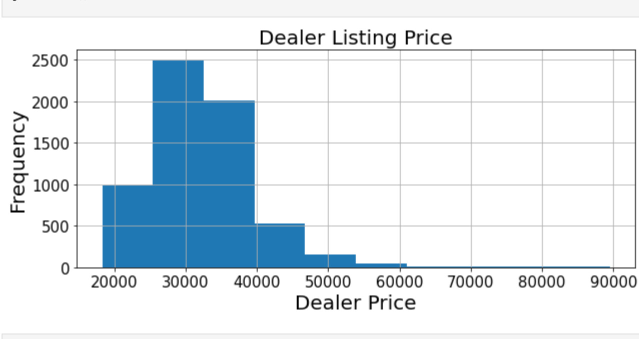
Out[7]:
```
ListingID                 int64
SellerCity               object
SellerIsPriv               bool
SellerListSrc            object
SellerName               object
SellerRating            float64
SellerRevCnt              int64
SellerState              object
SellerZip               float64
VehBodystyle             object
VehCertified               bool
VehColorExt              object
VehColorInt              object
VehDriveTrain            object
VehEngine                object
VehFeats                 object
VehFuel                  object
VehHistory               object
VehListdays             float64
VehMake                  object
VehMileage              float64
VehModel                 object
VehPriceLabel            object
VehSellerNotes           object
VehType                  object
VehTransmission          object
VehYear                   int64
Vehicle_Trim             object
Dealer_Listing_Price    float64
dtype: object
```

Looking at the data, the type distribution is as follows:

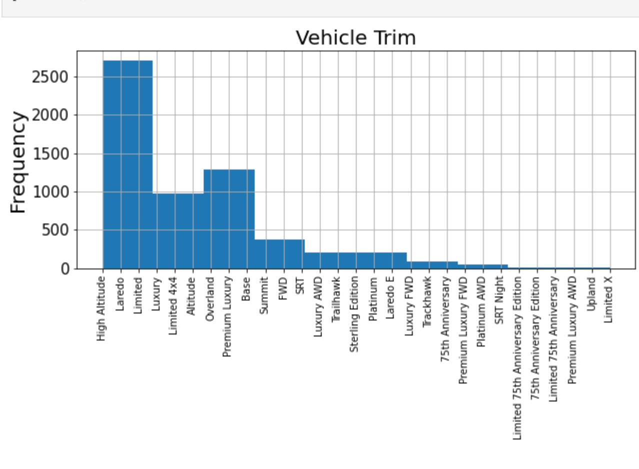
- 8 Numerical Variables (6 `float64` & 2 `int64`)
- 2 Boolean Variables (`bool`)
- 19 Categorical Variables (`object`)

Distribution of the Target Variable (`Price` & `Vehicle Trim`)

In [8]:
```
Training_df['Dealer_Listing_Price'].hist(figsize = (10,4))
plt.xlabel('Dealer Price', fontsize = 20)
plt.ylabel('Frequency',fontsize = 20)
plt.title('Dealer Listing Price', fontsize = 20)
plt.tick_params(axis="x", labelsize=15)
plt.tick_params(axis="y", labelsize=15)
plt.grid(True)
plt.show()
```



In [9]:
```
Training_df['Vehicle_Trim'].hist(figsize = (10,4))
plt.ylabel('Frequency',fontsize = 20)
plt.title('Vehicle Trim', fontsize = 20)
plt.tick_params(axis="x", labelsize=10, rotation=90)
plt.tick_params(axis="y", labelsize=15)
plt.grid(True)
plt.show()
```

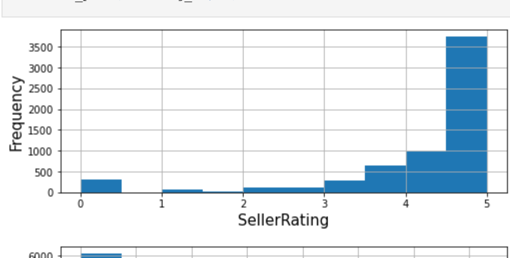


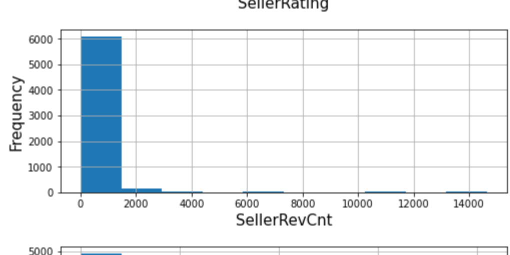
Distribution of Numerical Variables

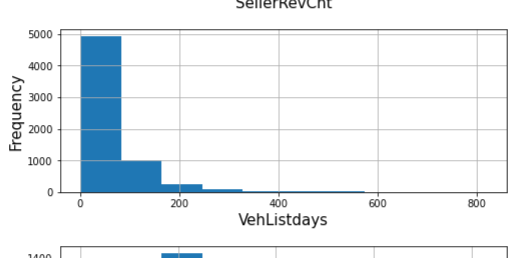
In [10]:
```
numeric_columns = ['SellerRating', 'SellerRevCnt', 'VehListdays', 'VehMileage', 'VehYear']
```

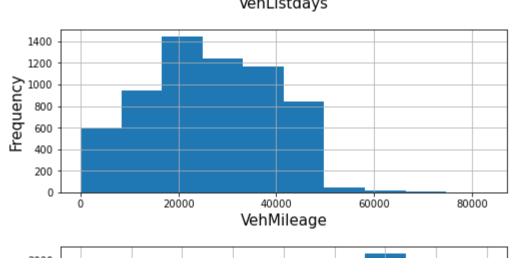
In [11]:
```
def hist_plot(df,a):
    df[a].hist(figsize = (8,3))
    plt.xlabel(a, fontsize = 15)
    plt.ylabel('Frequency',fontsize = 15)
    plt.tick_params(axis="x", labelsize=10)
    plt.tick_params(axis="y", labelsize=10)
    plt.grid(True)
    plt.show()
```

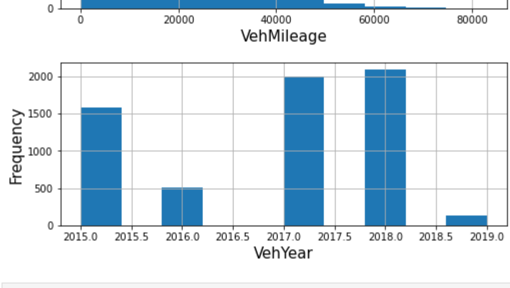
In [12]:
```
for i in numeric_columns:
    hist_plot(Training_df, i)
```



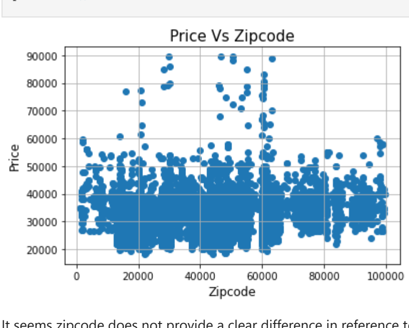








In [13]:
```
plt.scatter(Training_df['SellerZip'], Training_df['Dealer_Listing_Price'])
plt.title('Price Vs Zipcode', fontsize = 15)
plt.xlabel('Zipcode', fontsize = 12)
plt.ylabel('Price', fontsize = 12)
plt.grid()
plt.show()
```



It seems zipcode does not provide a clear difference in reference to the `Price` target variable.

Categorical Variables

In [14]:
```
list_of_types = Training_df.dtypes.tolist()
variables = Training_df.columns.tolist()

categorical = []
others = []

for i in range(len(variables)):
    if list_of_types[i] == 'O':
        categorical.append(variables[i])
    else:
        others.append(variables[i])
```

In [15]:
```
unique_categorical = []
for i in categorical:
    number_of_categories = len(pd.unique(Training_df[i]))
    unique_categorical.append(number_of_categories)
```

Unique Values in Training Dataset

In [16]:
```
unique_categorical_df = pd.DataFrame()
unique_categorical_df['Categorical Variable'] = categorical
unique_categorical_df['Number Of Unique Values'] = unique_categorical
unique_categorical_df
```

Out[16]:

	Categorical Variable	Number Of Unique Values
0	SellerCity	1318
1	SellerListSrc	9
2	SellerName	2452
3	SellerState	50
4	VehBodystyle	1
5	VehColorExt	170
6	VehColorInt	107
7	VehDriveTrain	21
8	VehEngine	97
9	VehFeats	844
10	VehFuel	5
11	VehHistory	34
12	VehMake	2
13	VehModel	2
14	VehPriceLabel	4
15	VehSellerNotes	4921
16	VehType	1
17	VehTransmission	34
18	Vehicle_Trim	30

Here we an see that some categories/columns have values that are very unique as `VehSellerNotes` (which makes sense as these are notes from the seller) and other categories have very unique as `VehType` & `VehBodyStyle` with just one value.

Data Formatting (Training Dataset & Testing Dataset)

Missing Values

In [17]:
```
def missing_values(data):
    #columns = data.columns.to_list()
    missing_values = []
    my_list = data.columns.values.tolist()
    y = data.shape
    for i in my_list:
        x = data[i].isnull().sum()
        missing_values.append(x)

    columns_missing = dict(zip(my_list, missing_values))
    a = sorted(columns_missing.items(), key=lambda x: x[1], reverse = True)

    missing_columns_names = []
    for key, value in columns_missing.items():
        if value > 0:
            x = key
            missing_columns_names.append(x)
    return columns_missing, missing_columns_names
```

In [18]:
```
#Training Data
dic_missing, missing_columns_names = missing_values(Training_df)

#Testing Data
dic_missing_test, missing_columns_names_test = missing_values(Test_df)
```

In [19]:
```
missing_training_df = pd.DataFrame()
missing_training_df['Variable'] = Training_df.columns.values.tolist()
missing_training_df['Number Of Missing Variables'] = dic_missing.values()
missing_training_df
```

Out[19]:

	Variable	Number Of Missing Variables
0	ListingID	0
1	SellerCity	0
2	SellerIsPriv	0
3	SellerListSrc	2
4	SellerName	0
5	SellerRating	0
6	SellerRevCnt	0
7	SellerState	0
8	SellerZip	2
9	VehBodystyle	0
10	VehCertified	0
11	VehColorExt	73
12	VehColorInt	728
13	VehDriveTrain	401
14	VehEngine	361
15	VehFeats	275
16	VehFuel	2
17	VehHistory	201
18	VehListdays	2
19	VehMake	0
20	VehMileage	2
21	VehModel	0
22	VehPriceLabel	285
23	VehSellerNotes	243
24	VehType	0
25	VehTransmission	197
26	VehYear	0
27	Vehicle_Trim	405
28	Dealer_Listing_Price	52

Looking at the Train Dataset we can see that there are some columns with more than 10% (`VehColorInt`) of missing data points.

In [20]:
```
missing_test_df = pd.DataFrame()
missing_test_df['Variable'] = Test_df.columns.values.tolist()
missing_test_df['Number Of Missing Variables'] = dic_missing_test.values()
missing_test_df
```

Out[20]:

	Variable	Number Of Missing Variables
0	ListingID	0
1	SellerCity	0
2	SellerIsPriv	0
3	SellerListSrc	0
4	SellerName	0
5	SellerRating	0
6	SellerRevCnt	0
7	SellerState	0
8	SellerZip	0
9	VehBodystyle	0
10	VehCertified	0
11	VehColorExt	7
12	VehColorInt	108
13	VehDriveTrain	64
14	VehEngine	58
15	VehFeats	37
16	VehFuel	0
17	VehHistory	27
18	VehListdays	0
19	VehMake	0
20	VehMileage	1
21	VehModel	0
22	VehPriceLabel	38
23	VehSellerNotes	41
24	VehType	0
25	VehTransmission	27
26	VehYear	0

In [21]:
```
# For the training data the missing values will be removed
data = Training_df.copy()
data.dropna(inplace = True)
data.shape
data = data.reset_index()
```

If the missing data is dropped we will reduce the dataset by around 20%. Depending on the example and the size of the dataset other options might be replacing the values with the mean/median (if numerical) or mode (if categorica).

In [22]:
```
# For the testing data the missing values will be filled with the mode and median
test_data = Test_df.copy()
categorical_test_missing = ['VehColorExt', 'VehColorInt', 'VehDriveTrain', 'VehEngine', 'VehFeats', 'VehHistory
numerical_test_missing = ['VehMileage']

# Replace missing values by median and mode
import statistics as st

for i in numerical_test_missing:
    test_data[i].fillna(test_data[i].median(), inplace=True)

for i in categorical_test_missing:
    test_data[i].fillna(st.mode(test_data[i]), inplace=True)
```

Removing Columns from dataset

Based on the nature of the data some columns will be removed as they do not add to the prediction:

- `ListingID` (as is a unique value per entry)
- `VehSellerNotes` (need more procesing to be used)
- `SellerZip`
- `VehType` (as it has just one value)
- `VehBodystyle` (as it has just one value)

In [23]:
```
# For Training Dataset
data.drop(columns=['ListingID', 'VehSellerNotes', 'SellerZip', 'VehType', 'VehBodystyle', 'index'], inplace = T
columns_updated = data.columns.to_list()
```

In [24]:
```
# For Testing Dataset
test_data.drop(columns=['VehSellerNotes', 'SellerZip','VehType', 'VehBodystyle'], inplace = True)
test_columns_updated = test_data.columns.to_list()
```

Formatting Boolean Variables

In [25]:
```
boolean_columns = ['SellerIsPriv', 'VehCertified']
def boolean_format(columns, df):
    for i in columns:
        df[i] = df[i].astype(int)
```

In [26]:
```
#Training Dataset
boolean_format(boolean_columns, data)

#Test Dataset
boolean_format(boolean_columns, test_data)
```

Normalizing Numerical Variables

In [27]:
```
def normalization_format(numeric_columns, df):
    from sklearn.preprocessing import StandardScaler
    scaler = StandardScaler()
    scaler.fit(df[numeric_columns])
    data_scaled = scaler.transform(df[numeric_columns])
    df1 = pd.DataFrame(data_scaled, columns = numeric_columns)
    return df1
```

In [28]:
```
# Training Dataset
data_norm = normalization_format(numeric_columns, data)
data.drop(columns = numeric_columns, inplace = True)
data = data.join(data_norm)

# Test Dataset
data_norm_test = normalization_format(numeric_columns, test_data)
test_data.drop(columns = numeric_columns, inplace = True)
test_data = test_data.join(data_norm_test)
```

In [29]:
```
# Special Columns
special_columns = ['VehHistory', 'VehFeats']
targets = ["Vehicle_Trim", "Dealer_Listing_Price"]
```

In [30]:
```
# Categorical Variables:
data['Vehicle_Trim'] = data['Vehicle_Trim'].astype('category')
d = dict(enumerate(data['Vehicle_Trim'].cat.categories))
data["Vehicle_Trim_cat"] = data['Vehicle_Trim'].cat.codes
data.drop(columns = 'Vehicle_Trim', inplace = True)
```

In [31]:
```
categorical_columns = [i for i in columns_updated if i not in numeric_columns]
categorical_columns = [i for i in categorical_columns if i not in boolean_columns]
categorical_columns = [i for i in categorical_columns if i not in special_columns]
categorical_columns = [i for i in categorical_columns if i not in targets]
```

In [32]:
```
for i in categorical_columns:
    data[i] = data[i].str.lower() #Training
    test_data[i] = test_data[i].str.lower() #Test
```

One Hot Encoding For Categorical Columns

In [33]:
```
def hot_encoding(data, categorical_columns):
    one_hot = pd.get_dummies(data[categorical_columns])
    return one_hot
```

In [34]:
```
# Hot encoding for Training
one_hot = hot_encoding(data, categorical_columns)
data = data.join(one_hot)
data.drop(columns = categorical_columns, inplace = True)
```

In [35]:
```
# Hot encoding for Test
one_hot_test = hot_encoding(test_data, categorical_columns)
test_data = test_data.join(one_hot_test)
test_data.drop(columns = categorical_columns, inplace = True)
```

Formatting of two *List of Lists* Columns (`VehHistory` & `VehFeats`)

There are 2 columns with a list of values that might be important for the analysis. For this reason they are going to be split and included as features (One Hot Encoding):

- `VehHistory`
- `VehFeats`

See examples below:

In [36]:
```
data['VehHistory'].head()
```

Out[36]:
```
0                   1 Owner, Buyback Protection Eligible
1    1 Owner, Non-Personal Use Reported, Buyback Pr...
2    1 Owner, Non-Personal Use Reported, Buyback Pr...
3    1 Owner, Non-Personal Use Reported, Buyback Pr...
4    1 Owner, Accident(s) Reported, Non-Personal Us...
Name: VehHistory, dtype: object
```

In [37]:
```
data['VehFeats'].head()
```

Out[37]:
```
0    ['18 WHEEL &amp; 8.4 RADIO GROUP-inc: Nav-Capa...
1    ['Android Auto', 'Antilock Brakes', 'Apple Car...
2    ['4-Wheel Disc Brakes', 'ABS', 'Adjustable Ste...
3    ['1st and 2nd row curtain head airbags', '4-wh...
4    ['1st and 2nd row curtain head airbags', '4-wh...
Name: VehFeats, dtype: object
```

`VehFeats`

In [38]:
```
def veh_features(data):
    x = pd.Series(data['VehFeats'])
    split_df = data['VehFeats'].tolist()
    mySeries = pd.Series(split_df)
    x = mySeries[2][1:-1]
    letter_list = x.split(",")

    list_of_lists = []
    for i in range(len(mySeries)):
        x = mySeries[i][1:-1]
        new_string = x.replace(" '", "")
        new_string = new_string.replace("'", "")
        new_string = new_string.replace(",", "_")
        new_string = new_string.replace(";", "_")
        new_string = new_string.lower()
        letter_list = new_string.split("_")
        list_of_lists.append(letter_list)

    from sklearn.preprocessing import MultiLabelBinarizer

    test = pd.Series(list_of_lists)
    mlb = MultiLabelBinarizer()
    onehot_VehFeats = pd.DataFrame(mlb.fit_transform(test),
                        columns=mlb.classes_,
                        index=test.index)
    return onehot_VehFeats
```

In [39]:
```
# Training Dataset
onehot_VehFeats = veh_features(data)
data = data.join(onehot_VehFeats)
data.drop(columns = 'VehFeats', inplace = True)

# Test Dataset
onehot_VehFeats_test = veh_features(test_data)
test_data = test_data.join(onehot_VehFeats_test)
test_data.drop(columns = 'VehFeats', inplace = True)
```

`VehHistory`

In [40]:
```
def veh_history(data):
    split_df2 = data['VehHistory'].tolist()
    mySeries2 = pd.Series(split_df2)

    list_of_lists2 = []
    for i in range(len(mySeries2)):
        w = mySeries2[i]
        letter_list2 = w.split(",")
        list_of_lists2.append(letter_list2)

    from sklearn.preprocessing import MultiLabelBinarizer

    test2 = pd.Series(list_of_lists2)
    mlb2 = MultiLabelBinarizer()
    onehot_VehHistory = pd.DataFrame(mlb2.fit_transform(test2),
                        columns=mlb2.classes_,
                        index=test2.index)
    return onehot_VehHistory
```

In [41]:
```
# Training Dataset
onehot_VehHistory = veh_history(data)
data = data.join(onehot_VehHistory)
data.drop(columns = 'VehHistory', inplace = True)

# Test Dataset
onehot_VehHistory_test = veh_history(test_data)
test_data = test_data.join(onehot_VehHistory_test)
test_data.drop(columns = 'VehHistory', inplace = True)
```

Splitting the dataset into Target Variables and Features (Training Dataset)

In [42]:

```python
target_price = data['Dealer_Listing_Price']
target_trim = data['Vehicle_Trim_cat']

data.drop(columns = ['Dealer_Listing_Price', 'Vehicle_Trim_cat'], inplace = True)
columns_model = data.columns.to_list()
columns_model_test = test_data.columns.to_list()

list1_as_set = set(columns_model_test)
intersection = list1_as_set.intersection(columns_model)
intersection_as_list = list(intersection)

#Features for both the training and testing dataset
features = data[intersection_as_list]
```

NOTE: Training Features and Test Features have to be the same

Since this is going to help in the prediction of a test dataset that has not been seen by the model, it is important to ensure that variables used by the model are the same for both dataframes (hence the last step).

Model 1: Prediction of `Dealer Listing Price`

Feature Selection (LASSO) for `Dealer Listing Price` Variable

Note: There are multiple ways to do the Feature Selection like Regularization (Lasso & Ridge) or PCA being among the most popular. Each method depend on finding the balance between complexity and interpretability.

In [43]:

```python
# LASSO
def feature_select_lasso(features, target_price, alpha_val):

    from sklearn import linear_model
    alpha = alpha_val # Increasing alpha can shrink more variable coefficients to 0
    clf = linear_model.Lasso(alpha=alpha)

    clf.fit(features, target_price)
    lasso_coef = clf.coef_
    wrapper_columns = features.columns.tolist()

    #Selection of the non zero coefficients from the LASSO regression
    lasso_features = []
    non_lasso_features = []
    for i in range(len(wrapper_columns)):
        w = lasso_coef[i]
        if w != 0:
            u = wrapper_columns[i]
            lasso_features.append(u)
        else:
            t = wrapper_columns[i]
            non_lasso_features.append(t)

    len_lasso = len(lasso_features)
    return lasso_features, len_lasso
```

In [44]:

```python
lasso_features, len_lasso = feature_select_lasso(features, target_price, 2)
print('# of Features using LASSO: ' + str(len_lasso))
```

```
# of Features using LASSO: 353
```

Model for `Dealer Listing Price` prediction

For this the training data will be split into 2 datasets: train and validation. This to be able (if needed) to update the model until an acceptable performance value is achieved and then train the model with the entire dataset and test it in the final test dataset.

In [45]:

```python
from sklearn.model_selection import train_test_split
lasso_variables = data[lasso_features]
X_train, X_test, y_train, y_test = train_test_split(lasso_variables, target_price, test_size=0.20, random_state
```

Training

In [46]:

```python
from sklearn.linear_model import ElasticNet
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import ElasticNetCV

#Fit a linear regression model to this data
lasso = ElasticNetCV(l1_ratio = [0.1, 0.3, 0.5, 0.7, 0.9],
                     alphas = [0.001, 0.005, 0.01, 0.05, 0.1],
                     max_iter=10000)
```

In [47]:

```python
# Fit the data(train the model)
lasso.fit(X_train, y_train)
```

```
C:\Users\rodol\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:526: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations. Duality gap: 240362428.9041099 5, tolerance: 19455929.413019426
  model = cd_fast.enet_coordinate_descent_gram(
C:\Users\rodol\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:526: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations. Duality gap: 282554777.926267 6, tolerance: 19965907.56805779
  model = cd_fast.enet_coordinate_descent_gram(
C:\Users\rodol\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:526: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations. Duality gap: 476683839.138027 2, tolerance: 19133778.023431525
  model = cd_fast.enet_coordinate_descent_gram(
C:\Users\rodol\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:526: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations. Duality gap: 149149527.776838 3, tolerance: 19155100.858702328
  model = cd_fast.enet_coordinate_descent_gram(
C:\Users\rodol\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:526: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations. Duality gap: 84699103.1309051 5, tolerance: 19657557.950388916
  model = cd_fast.enet_coordinate_descent_gram(
C:\Users\rodol\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:530: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations. Duality gap: 5984196687.44132 9, tolerance: 24343437.594786618
  model = cd_fast.enet_coordinate_descent(
```

Out[47]:

```
ElasticNetCV(alphas=[0.001, 0.005, 0.01, 0.05, 0.1],
             l1_ratio=[0.1, 0.3, 0.5, 0.7, 0.9], max_iter=10000)
```

In [48]:

```python
print('The Best alpha value was:')
print(lasso.alpha_)
print('')
print('The Best l1 ratio value was:')
print(lasso.l1_ratio_)
```

```
The Best alpha value was:
0.001

The Best l1 ratio value was:
0.9
```

Results/Predictions

In [49]:

```python
# Predict
y_predicted_train_lasso = lasso.predict(X_train)
y_predicted_test_lasso = lasso.predict(X_test)
```

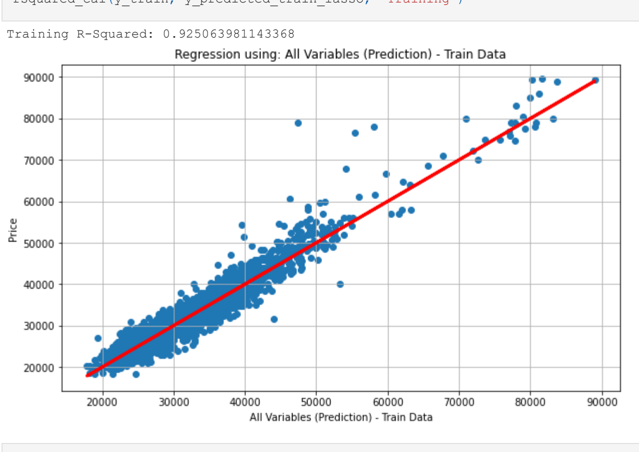
In [50]:

```python
def rsquared_cal(actual, predicted, status):
    corr_matrix = np.corrcoef(actual, predicted)
    corr = corr_matrix[0,1]
    R_sq = corr**2
    print(status + ' R-Squared: {}'.format(R_sq))

def scatters(a,b,c,d):
    plt.figure(figsize=(10, 6)).gca()
    plt.scatter(a, b)
    plt.plot(a, c, linewidth=3, color = 'red')
    plt.grid(True)
    plt.xlabel(d)
    plt.ylabel('Price')
    plt.title('Regression using: ' + d)
```

In [51]:

```python
#Training Results
scatters(y_predicted_train_lasso, y_train, y_predicted_train_lasso, 'All Variables (Prediction) - Train Data')
rsquared_cal(y_train, y_predicted_train_lasso, 'Training')
```

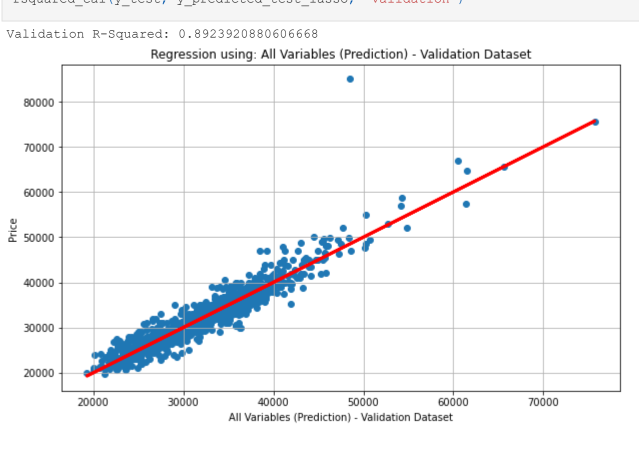
```
Training R-Squared: 0.925063981143368
```



In [52]:

```python
#Validation Results
scatters(y_predicted_test_lasso, y_test, y_predicted_test_lasso, 'All Variables (Prediction) - Validation Datas
rsquared_cal(y_test, y_predicted_test_lasso, 'Validation')
```

```
Validation R-Squared: 0.8923920880606668
```



Re-train the model but now with all the datapoints (training + validation)

In [53]:

```python
#Fit a linear regression model to this data
lassofinal = ElasticNet(alpha = 0.001, max_iter=10000)

# Fit the data(train the model) lasso_variables, target_price
lassofinal.fit(lasso_variables, target_price)
```

Out[53]:

```
ElasticNet(alpha=0.001, max_iter=10000)
```

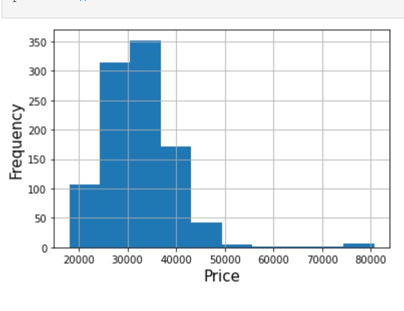
Prediction using the `Test_data`.

In [54]:

```python
# Test Predictions
lasso_variables_test = test_data[lasso_features]
final_prediction_price = lassofinal.predict(lasso_variables_test)
```

In [55]:

```python
plt.hist(final_prediction_price)
plt.xlabel('Price', fontsize = 15)
plt.ylabel('Frequency',fontsize = 15)
plt.tick_params(axis="x", labelsize=10)
plt.tick_params(axis="y", labelsize=10)
plt.grid(True)
plt.show()
```



Model 2: Prediction of `Vehicle_Trim`

Feature Selection (LASSO) for `Vehicle_Trim` Variable

In [56]:

```python
lasso_features_trim, len_lasso_trim = feature_select_lasso(features, target_trim, 0.001)
print('# of Features using LASSO: ' + str(len_lasso_trim))
```

```
# of Features using LASSO: 450
C:\Users\rodol\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:530: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations. Duality gap: 968.727934055394 1, tolerance: 14.667880555004965
  model = cd_fast.enet_coordinate_descent(
```

In [57]:

```python
lasso_variables_tim = data[lasso_features_trim]
X_train_2, X_test_2, y_train_2, y_test_2 = train_test_split(lasso_variables_tim, target_trim, test_size=0.20, r
```

Different Models and Hyperparameters

In [58]:

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn import metrics
from sklearn.metrics import classification_report
```

Hyperparameter Settings for the `Decision Tree` Model

In [59]:

```python
max_depth_options = [2,3,4,5,6,7,8,9,10]
min_sample_options = [2,3,4,5,6,7,8,9,10]
model_criterion = ['entropy', 'gini']
min_samples_leaf_options = [2,3,4,5,6,7,8,9,10]

dt_grid = {'criterion': model_criterion,
           'max_depth': max_depth_options,
           'min_samples_split':min_sample_options,
           'min_samples_leaf':min_samples_leaf_options}

dt_base = DecisionTreeClassifier()

decision_trees_hyper = RandomizedSearchCV(estimator = dt_base, param_distributions = dt_grid,
                                  n_iter = 200, cv = 3, verbose = 2, random_state = 42,
                                  n_jobs = -1)

# Fit the random search model
decision_trees_hyper.fit(X_train_2, y_train_2)

x = decision_trees_hyper.best_params_
x
```

```
C:\Users\rodol\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:666: UserWarning: The least populated class in y has only 1 members, which is less than n_splits=3.
  warnings.warn(("The least populated class in y has only %d"
Fitting 3 folds for each of 200 candidates, totalling 600 fits
```

Out[59]:

```
{'min_samples_split': 10,
 'min_samples_leaf': 2,
 'max_depth': 9,
 'criterion': 'entropy'}
```

Hyperparameter Settings for the `Random Forest` Model

In [60]:

```python
n_estimators_options = [10,20,30,40,50,60,70,80,90,100]

rf_grid = {'criterion': model_criterion,
           'n_estimators':n_estimators_options,
           'max_depth': max_depth_options,
           'min_samples_split':min_sample_options,
           'min_samples_leaf':min_samples_leaf_options}

rf_base = RandomForestClassifier()

random_forest_hyper = RandomizedSearchCV(estimator = rf_base, param_distributions = rf_grid,
                                  n_iter = 200, cv = 3, verbose = 2, random_state = 42,
                                  n_jobs = -1)

# Fit the random search model
random_forest_hyper.fit(X_train_2, y_train_2)

y = random_forest_hyper.best_params_
y
```

```
Fitting 3 folds for each of 200 candidates, totalling 600 fits
C:\Users\rodol\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:666: UserWarning: The least populated class in y has only 1 members, which is less than n_splits=3.
  warnings.warn(("The least populated class in y has only %d"
```

Out[60]:

```
{'n_estimators': 50,
 'min_samples_split': 4,
 'min_samples_leaf': 2,
 'max_depth': 10,
 'criterion': 'entropy'}
```

Hyperparameter Settings for the `Logistic Regression` Model

In [61]:

```python
from sklearn.linear_model import LogisticRegression

max_iter_options = [1000,10000]
penalty_options = ['l1', 'l2', 'elasticnet', 'none']
solver_options = ['saga']
multi = ['multinomial']

lr_grid = {'penalty': penalty_options,
           'solver': solver_options,
           'multi_class': multi,
           'max_iter': max_iter_options}

lr_base = LogisticRegression()

logistic_reg_hyper = RandomizedSearchCV(estimator = lr_base, param_distributions = lr_grid,
                                  n_iter = 200, cv = 3, verbose = 2, random_state = 42,
                                  n_jobs = -1)

# Fit the random search model
logistic_reg_hyper.fit(X_train_2, y_train_2)

z = logistic_reg_hyper.best_params_
z
```

```
C:\Users\rodol\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:285: UserWarning: The total space of parameters 8 is smaller than n_iter=200. Running 8 iterations. For exhaustive searches, use GridSearchCV.
  warnings.warn(
C:\Users\rodol\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:666: UserWarning: The least populated class in y has only 1 members, which is less than n_splits=3.
  warnings.warn(("The least populated class in y has only %d"
Fitting 3 folds for each of 8 candidates, totalling 24 fits
C:\Users\rodol\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:918: UserWarning: One or more of the test scores are non-finite: [0.85852284 0.86100042        nan 0.85877067 0.85877049 0.86100042
        nan 0.85381589]
  warnings.warn(
```

Out[61]:

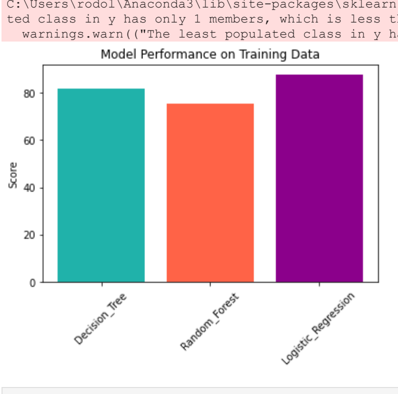
```
{'solver': 'saga',
 'penalty': 'l2',
 'multi_class': 'multinomial',
 'max_iter': 1000}
```

Comparison & Final Model Selection

In [62]:

```python
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc_curve
from sklearn.model_selection import StratifiedKFold, cross_val_score, cross_val_predict, KFold

#Classifiers
dt = DecisionTreeClassifier(criterion=x['criterion'], max_depth = x['max_depth'], min_samples_split = x['min_sa
rf = RandomForestClassifier(n_estimators = y['n_estimators'], criterion = y['criterion'], max_depth = y['max_de
lr = LogisticRegression(solver = z['solver'], penalty = z['penalty'], multi_class = z['multi_class'], max_iter

#List for Classifiers and Names
header = ["Decision_Tree","Random_Forest", "Logistic_Regression"]
V = [dt, rf, lr]

skf = StratifiedKFold(n_splits=10)
summary_accuracy = []
variation_accuracy = []
for i in V:
    results = cross_val_score(i, X_train_2, y_train_2, cv=skf)
    accuracy = results.mean()*100.0
    variation = results.std()*100.0
    summary_accuracy.append(accuracy)
    variation_accuracy.append(variation)

colors = ['lightseagreen', 'tomato', 'darkmagenta']
plt.bar(header, summary_accuracy, color=colors)
plt.xticks(rotation=45)
plt.title('Model Performance on Training Data')
plt.ylabel("Score")
plt.show()
```

```
C:\Users\rodol\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:666: UserWarning: The least populated class in y has only 1 members, which is less than n_splits=10.
  warnings.warn(("The least populated class in y has only %d"
C:\Users\rodol\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:666: UserWarning: The least populated class in y has only 1 members, which is less than n_splits=10.
  warnings.warn(("The least populated class in y has only %d"
C:\Users\rodol\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:666: UserWarning: The least populated class in y has only 1 members, which is less than n_splits=10.
  warnings.warn(("The least populated class in y has only %d"
```



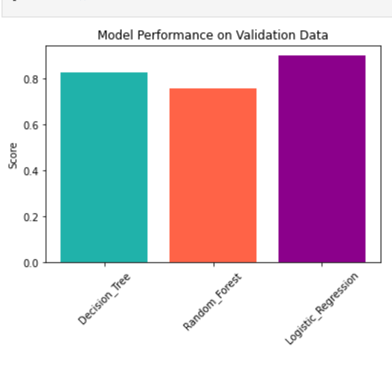
In [63]:

```python
print('Accuracy')
print(summary_accuracy)
print('')
print('Variation in Accuracy')
print(variation_accuracy)
```

```
Accuracy
[81.66541778247304, 75.54449303491143, 87.68641132103285]

Variation in Accuracy
[1.880570227392559, 2.3991881644484847, 1.373546375794306]
```

In [64]:

```python
summary_accuracy_validation = []
for i in V:
    i.fit(X_train_2, y_train_2)
    prediction_val = i.predict(X_test_2)
    AR = accuracy_score(y_test_2, prediction_val)
    summary_accuracy_validation.append(AR)

colors = ['lightseagreen', 'tomato', 'darkmagenta']
plt.bar(header, summary_accuracy_validation, color=colors)
plt.xticks(rotation=45)
plt.title('Model Performance on Validation Data')
plt.ylabel("Score")
plt.show()
```



Final Model 2 Selection and Training (using `training` + `validation` data)

In [67]:

```python
lr = LogisticRegression(solver = z['solver'], penalty = z['penalty'], multi_class = z['multi_class'], max_iter
lr.fit(lasso_variables_tim, target_trim)
```

Out[67]:

```
LogisticRegression(max_iter=1000, multi_class='multinomial', solver='saga')
```

In [68]:

```python
ListingID_column = test_data['ListingID']
test_data.drop(columns=['ListingID'], inplace = True)

# Test Predictions
lasso_variables_test_trim = test_data[lasso_features_trim]
final_prediction_trim = lr.predict(lasso_variables_test_trim)
```

Export to dataframe

In [69]:

```python
final_dataframe = pd.DataFrame()
final_dataframe['ListingID'] = ListingID_column
final_dataframe['trim_prediction'] = final_prediction_trim
final_dataframe['price_prediction'] = final_prediction_price
final_dataframe = final_dataframe.replace({'trim_prediction': d})
final_dataframe.to_csv('predictions.csv', index=False)
```

Instructions:

- Model the problem using whatever means you consider best. Your work is expected to be entirely your own. You may consult any resource or reference of your choosing to aide in solving the problem, but the work must be entirely yours. Please reference any resources you use in the write-up covered in step 5.
- If you use a software package to assist you, please include ALL of your original source code in its entirety, and submit it to us EXACTLY following the instructions in Steps 5 and 6. Please also include information about which package you used and why in your brief problem write-up.
- Do not use or add data from any third-party sources, such as internet car estimating tools, to the data provided. At your discretion, some or all of the provided data in `“Training_dataset.csv”` may be used, omitted or manipulated in any way during modeling, but no additional data may be added from outside sources.
- Once your model is built, use it to make predictions on EACH of the 1,000 vehicle listings included in the `“Test_dataset.csv”` file. Your output should be a comma separated values (.csv) file with one-thousand rows and three columns. The first column should be the `unique identifier` for the listing. The second column should be your predicted value for `vehicle trim`. The third column should be your predicted value for `dealer listing price`.
- Please submit a brief write-up of no more than 500 words describing the approach you selected and why. Please save your response as a PDF if possible. Please copy any source code from Step 2 and paste it as text into an appendix at the end of your write-up.
- Return your submission to us by replying back to the original email before the date and time specified in that email. Please attach the CSV containing your predictions from Step 4 and the PDF containing your write-up and source code from step 5 to the email. It should contain ONLY two attachments: the CSV from step 4 and the PDF from Step 5. Please don't resubmit any of our original data back in your reply.

Extra (Artificial Neural Networks)

The part of the analysis is extra an it is not used for the final predictions. This is just an example of the same problem being worked using Deep Neural Networks.

Basic Artificial Neural Network for Regression Problem

In [70]:

```python
import tensorflow as tf
import tensorflow.keras.models as models
import tensorflow.keras.layers as layers
from tensorflow.keras import regularizers
from tensorflow.keras.layers import Dropout
from tensorflow.keras import optimizers
nn = models.Sequential()

#nn.add(tf.keras.Input(shape=(28*28,)))
nn.add(layers.Dense(512,
                    activation = 'relu',
                    input_dim = X_train.shape[1],
                    kernel_regularizer=regularizers.l2(0.1)))
nn.add(layers.Dropout(0.5))
nn.add(layers.Dense(256,
                    activation = 'relu',
                    kernel_regularizer=regularizers.l2(0.1)))
nn.add(layers.Dropout(0.5))
nn.add(layers.Dense(64,
                    activation = 'relu'))
nn.add(layers.Dense(1, activation = 'linear'))

nn.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 512)               181248

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 256)               131328

 dropout_1 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 64)                16448

 dense_3 (Dense)             (None, 1)                 65

=================================================================
Total params: 329,089
Trainable params: 329,089
Non-trainable params: 0
_________________________________________________________________
```

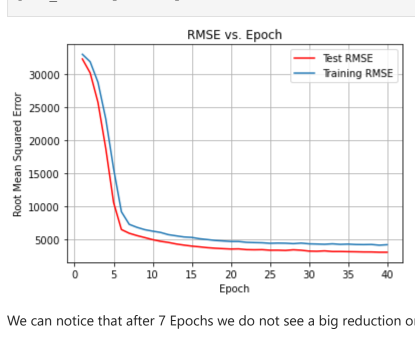
In [71]:

```python
## Set up and call-backs for early stopping
import keras
filepath = 'my_model_file.hdf5' # define where the model is saved
callbacks_list = [
        keras.callbacks.EarlyStopping(
            monitor = 'val_loss', # Use accuracy to monitor the model
            patience = 4 # Stop after one step with lower accuracy
        ),
        keras.callbacks.ModelCheckpoint(
            filepath = filepath, # file where the checkpoint is saved
            monitor = 'val_loss', # Don't overwrite the saved model unless val_loss is worse
            save_best_only = True # Only save model if it is the best
        )
]

nn.compile(optimizer = 'rmsprop', loss = 'mean_squared_error',
               metrics = [tf.keras.metrics.RootMeanSquaredError(name='rmse')])

history = nn.fit(X_train, y_train,
                 epochs = 40, batch_size = 256,
                 validation_data = (X_test, y_test),
                 callbacks = callbacks_list,  # Call backs argument here
                 verbose = 1)
```

```
Epoch 1/40
16/16 [==============================] - 1s 25ms/step - loss: 1087962752.0000 - rmse: 32984.2773 - val_loss: 10
41996544.0000 - val_rmse: 32279.9707
Epoch 2/40
16/16 [==============================] - 0s 11ms/step - loss: 1014362560.0000 - rmse: 31849.0586 - val_loss: 91
0068288.0000 - val_rmse: 30167.3340
Epoch 3/40
16/16 [==============================] - 0s 12ms/step - loss: 826358272.0000 - rmse: 28746.4473 - val_loss: 660
983936.0000 - val_rmse: 25709.6035
Epoch 4/40
16/16 [==============================] - 0s 12ms/step - loss: 538457216.0000 - rmse: 23204.6758 - val_loss: 350
332352.0000 - val_rmse: 18717.1621
Epoch 5/40
16/16 [==============================] - 0s 17ms/step - loss: 245815600.0000 - rmse: 15678.5000 - val_loss: 111
748936.0000 - val_rmse: 10571.1240
Epoch 6/40
16/16 [==============================] - 0s 14ms/step - loss: 84249632.0000 - rmse: 9178.7441 - val_loss: 42143
712.0000 - val_rmse: 6491.7949
Epoch 7/40
16/16 [==============================] - 0s 13ms/step - loss: 52967864.0000 - rmse: 7277.8804 - val_loss: 35159
624.0000 - val_rmse: 5929.5283
Epoch 8/40
16/16 [==============================] - 0s 13ms/step - loss: 46512756.0000 - rmse: 6820.0024 - val_loss: 31084
522.0000 - val_rmse: 5575.3203
Epoch 9/40
16/16 [==============================] - 0s 13ms/step - loss: 41833780.0000 - rmse: 6467.8784 - val_loss: 27804
222.0000 - val_rmse: 5272.9399
Epoch 10/40
16/16 [==============================] - 0s 13ms/step - loss: 39019956.0000 - rmse: 6246.5698 - val_loss: 24625
058.0000 - val_rmse: 4962.3311
Epoch 11/40
16/16 [==============================] - 0s 13ms/step - loss: 36688628.0000 - rmse: 6057.0869 - val_loss: 22236
376.0000 - val_rmse: 4715.5112
Epoch 12/40
16/16 [==============================] - 0s 13ms/step - loss: 32726592.0000 - rmse: 5720.6870 - val_loss: 20657
000.0000 - val_rmse: 4544.9609
Epoch 13/40
16/16 [==============================] - 0s 14ms/step - loss: 30714784.0000 - rmse: 5542.0620 - val_loss: 18599
940.0000 - val_rmse: 4312.7261
Epoch 14/40
16/16 [==============================] - 0s 12ms/step - loss: 28853222.0000 - rmse: 5371.4883 - val_loss: 17136
502.0000 - val_rmse: 4139.5854
Epoch 15/40
16/16 [==============================] - 0s 12ms/step - loss: 28109704.0000 - rmse: 5301.8262 - val_loss: 15908
892.0000 - val_rmse: 3988.5527
Epoch 16/40
16/16 [==============================] - 0s 13ms/step - loss: 26069304.0000 - rmse: 5105.7773 - val_loss: 15043
163.0000 - val_rmse: 3878.5078
Epoch 17/40
16/16 [==============================] - 0s 13ms/step - loss: 24789848.0000 - rmse: 4978.9053 - val_loss: 14079
541.0000 - val_rmse: 3752.2251
Epoch 18/40
16/16 [==============================] - 0s 13ms/step - loss: 23432248.0000 - rmse: 4840.6509 - val_loss: 13444
490.0000 - val_rmse: 3666.6252
Epoch 19/40
16/16 [==============================] - 0s 13ms/step - loss: 22726606.0000 - rmse: 4767.2065 - val_loss: 12979
585.0000 - val_rmse: 3602.6702
Epoch 20/40
16/16 [==============================] - 0s 12ms/step - loss: 22002830.0000 - rmse: 4690.6797 - val_loss: 12579
912.0000 - val_rmse: 3546.7673
Epoch 21/40
16/16 [==============================] - 0s 10ms/step - loss: 21973698.0000 - rmse: 4687.5737 - val_loss: 12749
519.0000 - val_rmse: 3570.5972
Epoch 22/40
16/16 [==============================] - 0s 14ms/step - loss: 20808924.0000 - rmse: 4561.6416 - val_loss: 11944
458.0000 - val_rmse: 3456.0237
Epoch 23/40
16/16 [==============================] - 0s 13ms/step - loss: 20445786.0000 - rmse: 4521.6621 - val_loss: 11801
388.0000 - val_rmse: 3435.2627
Epoch 24/40
16/16 [==============================] - 0s 10ms/step - loss: 20198934.0000 - rmse: 4494.2822 - val_loss: 11925
337.0000 - val_rmse: 3453.2554
Epoch 25/40
16/16 [==============================] - 0s 12ms/step - loss: 19481024.0000 - rmse: 4413.6899 - val_loss: 11351
404.0000 - val_rmse: 3369.1304
Epoch 26/40
16/16 [==============================] - 0s 10ms/step - loss: 19772154.0000 - rmse: 4446.5479 - val_loss: 11387
096.0000 - val_rmse: 3374.4231
Epoch 27/40
16/16 [==============================] - 0s 14ms/step - loss: 19610424.0000 - rmse: 4428.3242 - val_loss: 11136
923.0000 - val_rmse: 3337.1477
Epoch 28/40
16/16 [==============================] - 0s 10ms/step - loss: 19008844.0000 - rmse: 4359.8711 - val_loss: 11786
430.0000 - val_rmse: 3433.0835
Epoch 29/40
16/16 [==============================] - 0s 10ms/step - loss: 19771922.0000 - rmse: 4446.5210 - val_loss: 11336
201.0000 - val_rmse: 3366.8721
Epoch 30/40
16/16 [==============================] - 0s 14ms/step - loss: 18785734.0000 - rmse: 4334.2085 - val_loss: 10463
365.0000 - val_rmse: 3234.6553
Epoch 31/40
16/16 [==============================] - 0s 13ms/step - loss: 18512494.0000 - rmse: 4302.5713 - val_loss: 10293
845.0000 - val_rmse: 3208.3440
Epoch 32/40
16/16 [==============================] - 0s 11ms/step - loss: 18205324.0000 - rmse: 4266.7261 - val_loss: 10691
025.0000 - val_rmse: 3269.6555
Epoch 33/40
16/16 [==============================] - 0s 14ms/step - loss: 18796152.0000 - rmse: 4335.4097 - val_loss: 10055
349.0000 - val_rmse: 3170.9578
Epoch 34/40
16/16 [==============================] - 0s 11ms/step - loss: 18091930.0000 - rmse: 4253.4165 - val_loss: 10073
749.0000 - val_rmse: 3173.8574
Epoch 35/40
16/16 [==============================] - 0s 14ms/step - loss: 18426668.0000 - rmse: 4292.5854 - val_loss: 99126
07.0000 - val_rmse: 3148.3689
Epoch 36/40
16/16 [==============================] - 0s 13ms/step - loss: 17891862.0000 - rmse: 4229.8320 - val_loss: 98029
49.0000 - val_rmse: 3130.9050
Epoch 37/40
16/16 [==============================] - 0s 13ms/step - loss: 17772144.0000 - rmse: 4215.6572 - val_loss: 96372
67.0000 - val_rmse: 3104.3330
Epoch 38/40
16/16 [==============================] - 0s 13ms/step - loss: 17951074.0000 - rmse: 4236.8257 - val_loss: 96189
47.0000 - val_rmse: 3101.3809
Epoch 39/40
16/16 [==============================] - 0s 13ms/step - loss: 17040592.0000 - rmse: 4127.9785 - val_loss: 93822
75.0000 - val_rmse: 3062.9868
Epoch 40/40
16/16 [==============================] - 0s 12ms/step - loss: 17650042.0000 - rmse: 4201.1494 - val_loss: 93682
07.0000 - val_rmse: 3060.6895
```

In [72]:

```python
def plot_accuracy(history):
    train_acc = history.history['rmse']
    test_acc = history.history['val_rmse']
    x = list(range(1, len(test_acc) + 1))
    plt.plot(x, test_acc, color = 'red', label = 'Test RMSE')
    plt.plot(x, train_acc, label = 'Training RMSE')
    plt.grid()
    plt.legend()
    plt.xlabel('Epoch')
    plt.ylabel('Root Mean Squared Error')
    plt.title('RMSE vs. Epoch')

plot_accuracy(history)
```



We can notice that after 7 Epochs we do not see a big reduction on MSE. And after 15 Epochs the RMSE basically is a flat line.