HYUNDAI CAR PRICE PREDICTION

PREPARED BY

SALONI JAIN

MAYANK CHOUHAN

TARUN KUMAR SHARMA

TEJASWINI ADIGOPUL

RITURAJ PAWAR

Problem Statement

Determine the prices of Hyundai cars based on different factors such as model, year, mileage, fuel type, transmission etc and calculating the regression models to finding out which will be the best model to get the best results

Models

model	Accent	Amica	Getz	I10	I20	I30	I40	I800	IX20	IX35	Ioniq	Kona	Santa Fe	Terracan	Tucson	Veloster
fuelType																
Diesel	0	0	0	0	10	220	118	117	20	106	0	2	242	2	758	0
Hybrid	0	0	0	0	0	0	0	0	0	0	272	19	0	0	49	0
Other	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
Petrol	1	1	6	1061	477	315	2	0	182	12	2	301	2	0	473	3

Pipeline

- Data Cleaning
- Feature Engineering
- One hot encoding
- Outlier Analysis
- Model Creation

DATASET DETAILS

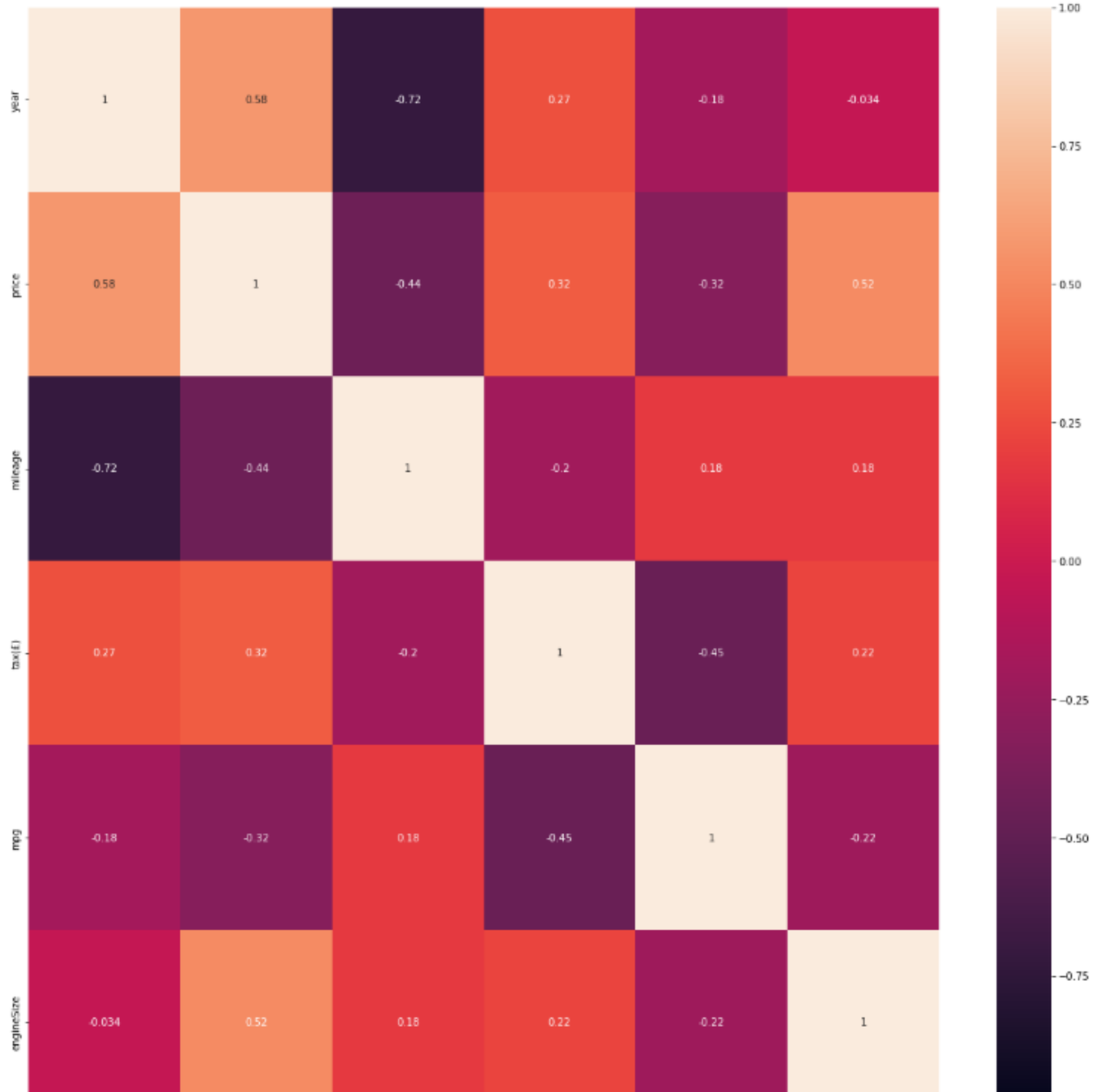
```
n [164]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4860 entries, 0 to 4859
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   model         4860 non-null   object 
 1   year          4860 non-null   int64  
 2   price         4860 non-null   int64  
 3   transmission  4860 non-null   object 
 4   mileage       4860 non-null   int64  
 5   fuelType      4860 non-null   object 
 6   tax(£)        4860 non-null   int64  
 7   mpg           4860 non-null   float64
 8   engineSize    4860 non-null   float64
dtypes: float64(2), int64(4), object(3)
memory usage: 341.8+ KB
```

- The data is taken from hyundai cars data with 4860 instances and 9 attributes.
- Converted the categorical columns into numerical using one hot encoding scheme.
- Feature Standardization was performed on all numeric data varaibles.
- The dataset was split into Train-Test using scikit-package.

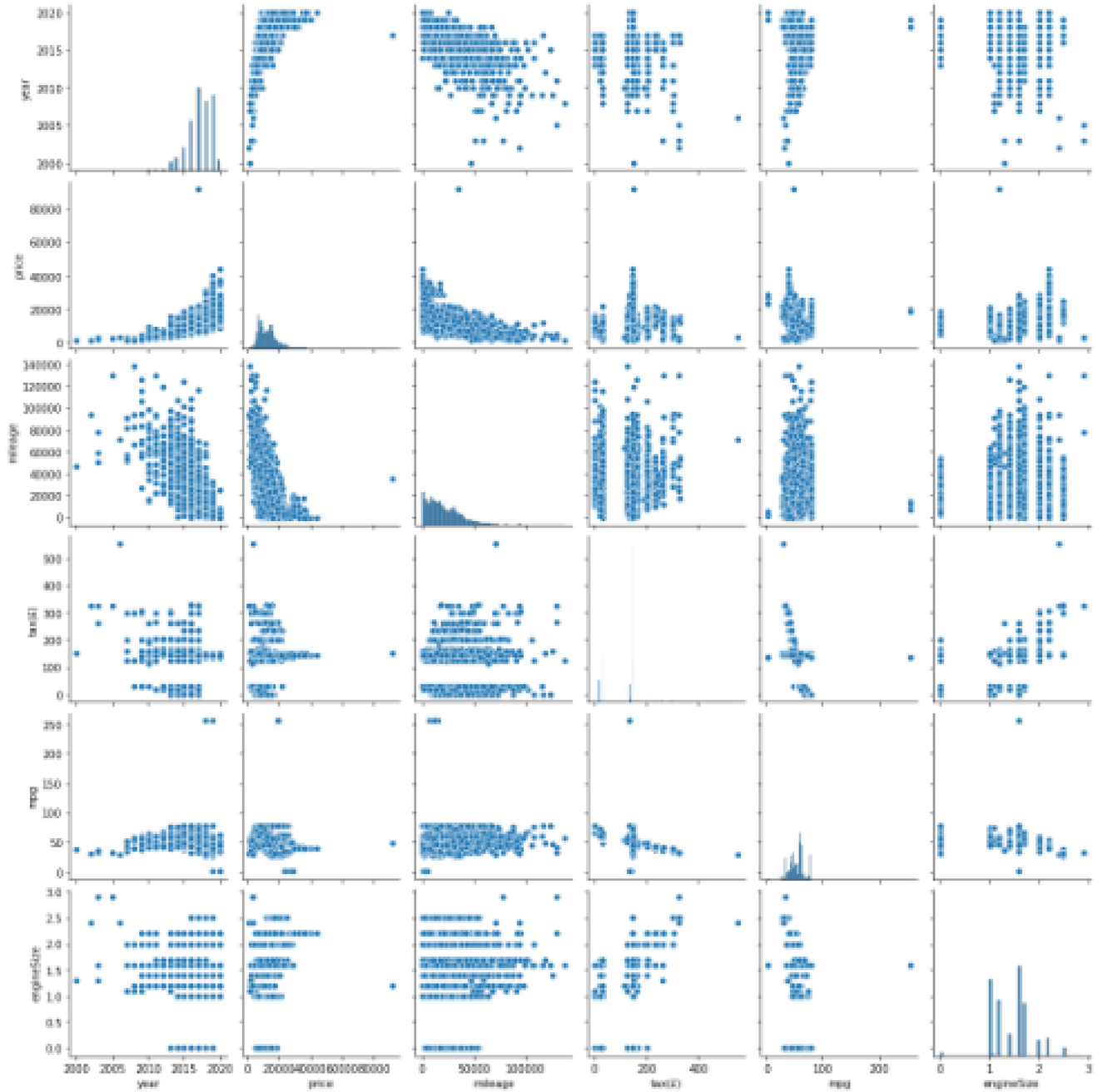
CORRELATION HEATMAP

```
In [32]: plt.figure(figsize=(20,20))
         sns.heatmap(df.corr(), annot=True,vmax=+1, vmin=-1)
         plt.show()
```



PAIRWISE RELATIONSHIP

In [15]:
```
sns.pairplot(data=df)
plt.show()
```



TRAINING-TESTING MODELS

1. Linear Regression
2. Decision Tree Regressor
3. Random Forest Regressor
4. Support Vector Regressor
5. Fine Tuning the Hyperparameters for Random Forest Regressor using Grid search cv.

Linear Regression

Linear Regression helped understand which variable are significant & which not also since many of our attributes are continues, linear regression is a good approach to use as a starting step.

```
In [51]: from sklearn.linear_model import LinearRegression

In [52]: reg=LinearRegression()

In [53]: reg.fit(x_train,y_train)

Out[53]: LinearRegression()

In [54]: reg.coef_

Out[54]: array([ 2.22220163e+03, -1.23196285e+03, -1.69637935e+03,  2.52668893e+03,
                -9.68461047e+01,  1.46204299e+00,  6.72926167e+02,  1.19039580e+03,
                 6.55185101e+01, -9.70624809e+02])

In [55]: reg.intercept_

Out[55]: 12729.694956615862

In [56]: y_pred = reg.predict(x_test)

In [57]: from sklearn import metrics

In [58]: metrics.mean_squared_error(y_test,y_pred)

Out[58]: 5915132.971440323

In [59]: np.sqrt(metrics.mean_squared_error(y_test,y_pred))

Out[59]: 2432.104638258873

In [60]: metrics.r2_score(y_test,y_pred)

Out[60]: 0.8166499558834083
```

Decision Tree Regressor

```
In [62]: y=df['price'].values

In [63]: from sklearn.tree import DecisionTreeRegressor

In [64]: from sklearn.model_selection import train_test_split

In [65]: x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=43)

In [66]: dtc = DecisionTreeRegressor(max_depth = 4, min_samples_leaf = 0.1, random_state=43)

In [67]: dtc.fit(x_train,y_train)
Out[67]: DecisionTreeRegressor(max_depth=4, min_samples_leaf=0.1, random_state=43)

In [68]: y_pred = dtc.predict(x_test)

In [69]: from sklearn.metrics import mean_squared_error as MSE

In [70]: mse_dt = MSE(y_test,y_pred)

In [71]: rmse_dt = mse_dt**(1/2)

In [72]: rmse_dt
Out[72]: 3805.414636244989

In [73]: score = dtc.score(x_train,y_train)

In [74]: score*100
Out[74]: 56.9085068009598
```

Random Forest Regressor

In [75]:
```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.ensemble import RandomForestRegressor
```

In [76]:
```
df.columns
```

Out[76]:
```
Index(['year', 'price', 'mileage', 'mpg', 'engineSize', 'transmission_Manual',
       'transmission_Other', 'transmission_Semi-Auto', 'fuelType_Hybrid',
       'fuelType_Other', 'fuelType_Petrol'],
      dtype='object')
```

In [77]:
```
x = df[['year', 'mileage', 'mpg', 'engineSize', 'transmission_Manual',
       'transmission_Other', 'transmission_Semi-Auto', 'fuelType_Hybrid',
       'fuelType_Other', 'fuelType_Petrol']]
y = df['price']
```

In [78]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,random_state=42)
```

In [79]:
```
regr = RandomForestRegressor()
regr.fit(x_train, y_train)
```

Out[79]:
```
RandomForestRegressor()
```

In [80]:
```
y_pred = regr.predict(x_test)
```

In [81]:
```
mse = metrics.mean_squared_error(y_test, y_pred)
rmse = mse**0.5
```

In [82]:
```
mse
```

Out[82]:
```
1706899.4962833382
```

Hyperparameter tuning using Grid search cv

In [93]:
```
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

mae = mean_absolute_error(y_test.values.ravel(), y_pred)

mse = mean_squared_error(y_test.values.ravel(), y_pred)

r2 = r2_score(y_test.values.ravel(), y_pred)

print('Mean Absolute Error:', round(mae, 2))
print('Mean Squared Error:', round(mse, 2))
print('R-squared scores:', round(r2, 2))
```

```
Mean Absolute Error: 880.93
Mean Squared Error: 1772786.28
R-squared scores: 0.95
```

In [94]:
```
from sklearn.model_selection import GridSearchCV
```

In [95]:
```
parameters = {
    'max_depth': [70, 80, 90, 100],
    'n_estimators': [900, 1000, 1100]
}
```

In [96]:
```
gridforest = GridSearchCV(regr, parameters, cv = 3, n_jobs = -1, verbose = 1)
gridforest.fit(x_train, y_train)
gridforest.best_params_
```

```
Fitting 3 folds for each of 12 candidates, totalling 36 fits
```

Out[96]:
```
{'max_depth': 70, 'n_estimators': 900}
```

Support Vector Regressor

SVR

In [ ]:
```
x = df[['year','mileage', 'mpg', 'engineSize', 'transmission_Manual',
       'transmission_Other', 'transmission_Semi-Auto', 'fuelType_Hybrid',
       'fuelType_Other', 'fuelType_Petrol']].values
```

In [99]:
```
y = df[['price']].values
```

In [100]:
```
from sklearn.preprocessing import StandardScaler
sc_x = StandardScaler()
x = sc_x.fit_transform(x)
sc_y = StandardScaler()
y = sc_y.fit_transform(y)
```

In [103]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 28)
```

In [104]:
```
from sklearn.svm import SVR
```

In [105]:
```
model = SVR(kernel = 'rbf')
model.fit(x_train, y_train.ravel())
```

Out[105]:
```
SVR()
```

In [106]:
```
y_pred = model.predict(x_test)
```

In [107]:
```
rmse = float(format(np.sqrt(mean_squared_error(y_test, y_pred)), '.3f'))
print("\nRMSE: ", rmse)
```

```
RMSE:  0.248
```

Comparative Analysis

- In Multiple Linear Regression , the best R-squared 0.8166 intercept is 12729.69 and RMSE is 2432.10
- In Decision Tree, the best regression model comes from random forest with rmse 1772786.28 and R2 as 0.95.
- In SVM model with linear kernel performs best with RMSE - 0.248.
- Of the four models random forest performs better than the others.

Challenges

- Researched about support vector regression and implemented it using scikit learn.
- Used polynomial and multiple linear regression in order to compare the performance of both the models .
- chi square analysis to find out the relation between two variables.

Learnings

- Team Work & collaboration
- Time Management.
- Task delegation.


THANK
YOU