REPORT

Assignment 2

Dataset: House Price Prediction

Submitted to: **Vahid Hadavi**

Group Members:
Manpreet Kaur c0777008
Rajinder Kaur c0779393
Chetan Ahuja c0770595
Harkeerat c0773564
Akash Dutta c0773441

INTRODUCTION

This Assignment has been done as part of our subject **Data Mining and Analysis** at **Lambton College** under the guidance of our instructor **Vahid Hadavi** which has been a great support throughout.

DATASET

We have selected House price prediction as our dataset which will help us in predicting House prices using all the necessary steps and machine learning algorithm to get the desired results. Our dataset includes a training set, test set and data description. The reason for selecting this dataset is that it is going to be a challenging task to predict the right price as far as current real estate scenario is concerned which will surely give us a clear picture of how to implement machine learning algorithms on real time projects.

The Dataset includes following files:

- train.csv - the training set
- test.csv - the test set
- data_description.txt - full description of each column

DATA SOURCE

Although we had a lot of resources to pick datasets from like UCI Machine Learning Repository, Data.gov by US government and Kaggle Datasets but we shortlisted House Prices – Advanced Regression Techniques from Kaggle datasets https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data

STEP 1: Data Preparation

- Importing the necessary libraries and reading the data.

```
In [1]: import pandas as pd
        import numpy as np
        import seaborn as sb
        import matplotlib.pyplot as plt
        from pandas_profiling import ProfileReport
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVa
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1455	1456	60	RL	62.0	7917	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	
1456	1457	20	RL	85.0	13175	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	MnPrv	NaN	
1457	1458	70	RL	66.0	9042	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	GdPrv	Shed	250
1458	1459	20	RL	68.0	9717	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	
1459	1460	20	RL	75.0	9937	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	

1460 rows × 81 columns

- Importing Pandas Profiling Module which provides analysis like unique values, missing values, null values, skewness, mean, mode, median, histograms and heatmap visualization.

```
profile = ProfileReport(data_select, title='Pandas Profiling Report', explorative=True)
```

```
profile.to_widgets()
```

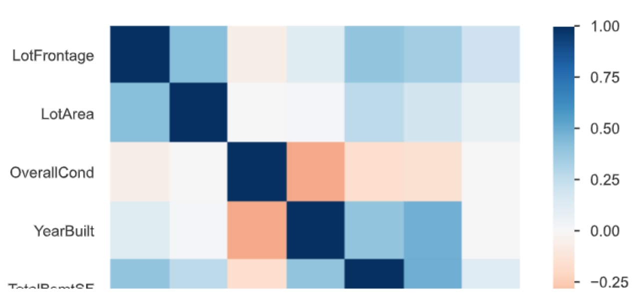
Summarize dataset: 100% 38/38 [00:19<00:00, 1.42s/it, Completed]

Generate report structure: 100% 1/1 [00:11<00:00, 11.05s/it]

Overview | Variables | Interactions | Correlations | Missing values | Sample | Duplicate rov

Toggle correlation descrip

Pearson's r | Spearman's ρ | Kendall's τ | Phik (φk) | Cramér's V (φc)



- We can also save reports generated by pandas profiling as HTML or JSON files:

```
profile.to_file("SelectedHouseFeatures_TrainDataset_report.html")
```

Render HTML: 100% 1/1 [00:03<00:00, 3.07s/it]

Export report to file: 100% 1/1 [00:00<00:00, 16.50it/s]

- Dropping duplicate rows and features with high number (99.5%) of zero values.

```
# Dropping duplicate rows

df = data_select.drop_duplicates()
df
```

```
# Dropping features with high number(99.5%) of zero values

df=df.drop(['PoolArea'], axis=1)
```

- Creating a function to check for missing values and filling up with mean of that column.

```
# Checking for the missing values

def show_missing():
    missing = df.columns[df.isnull().any()].tolist()
    return missing

df[show_missing()].isnull().sum()
```

```
LotFrontage    223
GarageType      72
dtype: int64
```

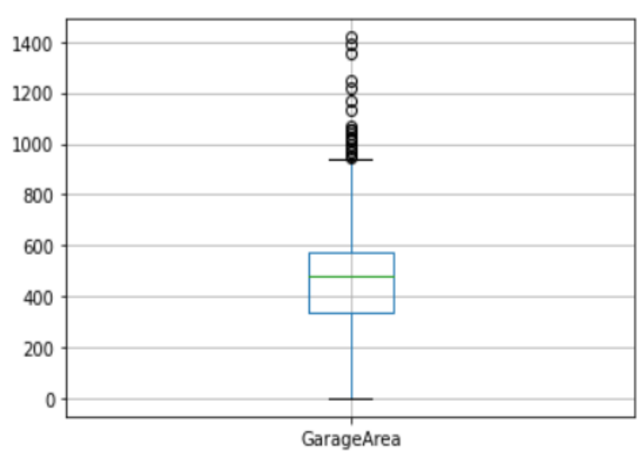
```
df['LotFrontage']=df['LotFrontage'].fillna(df['LotFrontage'].mean())
```

- Detecting and removing outliers. First, we plotted all features together but for better understanding and visualization, later plotted boxplot for some features individually.

```
df_train.boxplot(rot=60, grid='True', fontsize=10, figsize=(10,15))
```

```
df.boxplot(column='GarageArea')
```

```
<AxesSubplot:>
```



```python
# Removing major Outliers
def remove_outlier(data_in, col_name):
    q1 = data_in[col_name].quantile(0.25)
    q3 = data_in[col_name].quantile(0.75)
    iqr = q3-q1 #Interquartile range
    fence_low  = q1-1.5*iqr
    fence_high = q3+1.5*iqr
    data_out = data_in.loc[(data_in[col_name] > fence_low) & (data_in[col_name] < fence_high)]
    return data_out
```

```python
df = remove_outlier(df,'LotArea')
df = remove_outlier(df,'GarageArea')
df = remove_outlier(df,'TotalBsmtSF')
df = remove_outlier(df,'SalePrice')
```

STEP 2: Feature Engineering

- **Dimension reduction**: Overall, we have 81 features in the dataset, but we need to eliminate a couple of features to build the model so that we can only stick to features which are meaningful, informative, and non-redundant.
    - Out of 81 features, we have selected 26 in the **feature selection** step.
    - And **SalePrice** is the **target label**.

```python
# Selecting useful features

data_select = pd.DataFrame(data, columns = ['MSZoning','LotFrontage','LotArea','Street','LotShape','LandContour','Utilities','Lot
```

The selected features are as follows:

1) **MSZoning**: The general zoning classification.
2) **LotFrontage**: Linear feet of street connected to property.
3) **LotArea**: Lot size in square feet.
4) **Street**: Type of road access.
5) **LotShape**: General shape of property.
6) **LandContour**: Flatness of the property.
7) **Utilities**: Type of utilities available.
8) **LotConfig**: Lot Configuration.
9) **Neighborhood**: Physical locations within Ames city limits.
10) **Condition1**: Proximity to main road or railroad.
11) **BldgType**: Type of dwelling.
12) **OverallCond**: Overall condition rating.
13) **YearBuilt**: Original construction date.
14) **RoofStyle**: Type of roof.
15) **RoofMatl**: Material of roof.

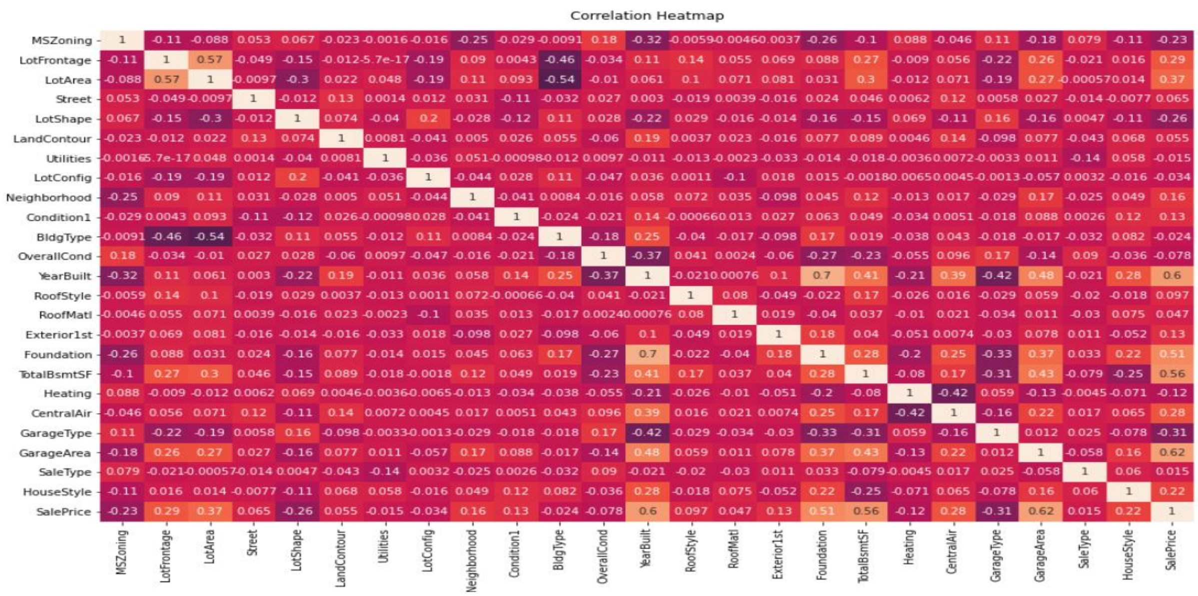
16) **Exterior1st**: Exterior covering on house.
17) **Foundation**: Type of foundation.
18) **TotalBsmtSF**: Total square feet of basement area.
19) **Heating**: Type of heating.
20) **CentralAir**: Central air conditioning.
21) **GarageType**: Garage location.
22) **GarageArea**: Size of garage in square feet.
23) **PoolArea**: Pool area in square feet.
24) **SaleType**: Type of sale.
25) **HouseStyle**: Type of dwelling.
26) **SalePrice**: This is going to be the **TARGET LABEL.**

- Some of the columns in the dataset had datatype as object, and it is not possible to train our ML model with object datatype, so we need to convert them into numeric values with the help of below code:

```
cat_columns = df_train[['MSZoning','Street','LotShape','LandContour','Utilities','LotConfig','Neighborhood','Condition1',
                        'BldgType','RoofStyle','RoofMatl','Exterior1st','Foundation','Heating','CentralAir','SaleType',
                        'HouseStyle']]

for name in cat_columns:
    df_train[name] = df_train[name].astype('category')

non_num_cols = df_train.select_dtypes(['category']).columns
df_train[non_num_cols] = df_train[non_num_cols].apply(lambda x: x.cat.codes)

df_train.dtypes
```

```
MSZoning          int8
LotFrontage    float64
LotArea          int64
Street            int8
LotShape          int8
LandContour       int8
Utilities         int8
LotConfig         int8
Neighborhood      int8
Condition1        int8
BldgType          int8
OverallCond      int64
YearBuilt        int64
RoofStyle         int8
RoofMatl          int8
Exterior1st       int8
Foundation        int8
TotalBsmtSF      int64
Heating           int8
```

- Now we will check for highly correlated features using **Heat Map.**



- Now we will check for features that have **Low Variance**.

```
from sklearn.feature_selection import VarianceThreshold

constant_filter = VarianceThreshold(threshold=0)
constant_filter.fit(df_train)
```

```
VarianceThreshold(threshold=0)
```

```
len(df_train.columns[constant_filter.get_support()])
```

```
25
```

```
# Getting number of columns with no variance

constant_columns = [column for column in df_train.columns
                    if column not in df_train.columns[constant_filter.get_support()]]

print(len(constant_columns))
```

```
0
```

As shown in above code snippet that the features do not have **low variance,** so we need not drop any **non-discriminative** features.

STEP 3: Data Modelling

- At this step, we need to split the dataset into **train** set and **test** set for further assessment.

```
from sklearn.model_selection import train_test_split

X = df_train.drop(['SalePrice'], axis=1)
y = df_train.SalePrice

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

Now, we need to apply different ML algorithms on our data to build an efficient machine learning model. We have applied **Linear Regression**, **Bayesian Ridge** and **Random Forest** algorithms on our data.

- **Linear Regression:**

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train,y_train)
y_pred= regressor.predict(X_test)
round(regressor.score(X_test,y_test), 4)
```

```
0.6783
```

- **Bayesian Ridge:**

```
from sklearn.linear_model import BayesianRidge
model = BayesianRidge()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
round(model.score(X_test,y_test), 4)
```

```
0.6814
```

- **Random Forest:**

```
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(n_estimators=1000, random_state=42)
regressor.fit(X_train,y_train)
y_pred= regressor.predict(X_test)
round(regressor.score(X_test,y_test), 4)
```

```
0.7948
```

As we can see that from the above findings that we did not choose **Linear Regression** and **Bayesian Ridge** algorithm because the accuracy was much lower than the expectations.

On the contrary, **Random forest** gives us **79% accuracy** which can be helpful for our model to predict the prices more accurately.

STEP 4: Performance Measure

- Now we need to assess the performance of our model.

(**NOTE**: We cannot train our model and test its performance on the same dataset, so we need to split the dataset for further assessment of the data.)

- We have different techniques to check the performance of the model like **K fold cross validation**, **mean absolute error**, **mean squared error**, **root mean square** etc.

```
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(n_estimators=100, random_state=42)
regressor.fit(X_train,y_train)
y_pred= regressor.predict(X_test)
print("score is: \n",round(regressor.score(X_test,y_test), 4))
mae_pred = metrics.mean_absolute_error(y_test,y_pred)
print("mae_pred is: \n", mae_pred)
```

```
score is:
 0.7739
mae_pred is:
 19185.99481012658
```

```
from sklearn.ensemble import RandomForestRegressor
from math import sqrt
regressor = RandomForestRegressor(n_estimators=100,max_depth=32, random_state=42)
regressor.fit(X_train,y_train)
y_pred= regressor.predict(X_test)
#print("score is: \n",round(regressor.score(X_test,y_test), 4))
mae_pred = metrics.mean_absolute_error(y_test,y_pred)
root_mean = sqrt(metrics.mean_squared_error(y_test,y_pred))
print("mae_pred is: \n", mae_pred)
print("root mean square error is: \n", root_mean)
```

```
mae_pred is:
 17783.845991561182
root mean square error is:
 23844.696699662076
```

In the above snippet we can see that our **Mean Absolute error (MAE)** value is **19185.99481012658** and **Root mean square error** (**RMSE**) value is **23844.69** which can be calculated by the below formula:

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} \left| y_i - x_i \right|}{n} \qquad \mathrm{RMSD} = \sqrt{\frac{\sum_{i=1}^{N} \left(x_i - \hat{x}_i \right)^2}{N}}$$

MAE = mean absolute error

y_i = prediction

x_i = true value

n = total number of data points

RMSD = root-mean-square deviation

i = variable i

N = number of non-missing data points

x_i = actual observations time series

$\hat{x}_i$ = estimated time series

STEP 5: Performance Improvement

In this step, as we know that our model is Random Forest regressor and to improve our performance we have implemented few lines of code shown below:

We have checked for the two most crucial parameters for the random forest to find the best accuracy for the same.

```
import warnings
warnings.filterwarnings('ignore')
from sklearn.ensemble import RandomForestRegressor
rfc = RandomForestRegressor()
parameters = {
    "n_estimators":[5,10,50,100,250,1000,2000],
    "max_depth":[2,4,8,16,32,None]

}

from sklearn.model_selection import GridSearchCV
cv = GridSearchCV(rfc,parameters,cv=5)
cv.fit(X_train,y_train.values.ravel())

def check(results):
    print(f'Best parameters are: {results.best_params_}')
    print("\n")
    mean_score = results.cv_results_['mean_test_score']
    std_score = results.cv_results_['std_test_score']
    params = results.cv_results_['params']
    for mean,std,params in zip(mean_score,std_score,params):
        print(f'{round(mean,3)} + or -{round(std,3)} for the {params}')
check(cv)
```

```
Best parameters are: {'max_depth': 16, 'n_estimators': 50}
```

Summary

To summarize, we have completed all the checkpoints which were necessary for this assignment like Data Preparation, Feature engineering, Data modelling, Performance measure and Performance improvement. All these checkpoints were equally important and responsible for the outcome of the project. The 26 features out of 81 were helpful in the prediction process. Random forest algorithm helped us in achieving 79% accuracy approximately with Mean Absolute error (**MAE**) of 19185.67 and Root mean square error **(RMSE)** of 23844.69. On the contrary, Linear regression and Bayesian algorithm had unsatisfactory results so they did not contribute to the outcome.

Conclusion

To conclude, our overall goal has been achieved as far as the project requirements are concerned. We used three models on our dataset (Linear Regression, Bayesian Algorithm and Random Forest algorithm) out of which, Random forest gave us the best accuracy which is an exploratory attempt to get the desired results. We have successfully created a model which predicts accurate house prices for the users to give them a wonderful experience in terms of Reliability and Accuracy.