Assignment 2

2.1 EDA

(1) Perform an initial EDA on the given data to gain an understanding of the data. The analyses should explore the data from four different aspects including:

(a) Describe the summary statistics about the data including number of instances, number of features, how many categorical and numerical features, respectively

(b) Find the top 5 numerical features highly correlated with the target variable ("SalePrice") according to the pearson correlation, report the correlation values.

(c) Plot the distributions of these 5 numerical features found in the previous question and the target variable using histograms with 10 bins, one for each feature/variable, describe the shape of their distributions with skewness and kurtosis (use Scipy for obtaining skewness and kurtosis values if you cannot do it in Orange), and tell two patterns from the histograms accordingly

(d) Check for missing values. Is there any missing values in the data? Write a paragraph to briefly summarise the missing information regarding how many features contain missing values and at what percent.

Provide answers to these four questions. Show how you get the answers in your code (if you use python)/workflow (if you use orange). Report your EDA methods and results in the report.

In [107…
```
# importing pandas, and other necessary modules
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import scipy as sp

# load data set
df = pd.read_csv("House_Price.csv", header = 0)
```

Part 1 - Task 1 (a)

Describe the summary statistics about the data including number of instances, number of features, how many categorical and numerical features, respectively.

The data set contains 79 different features, plus an id and the SalesPrice variable, which is what this is meant to find. There are 1460 instances. There are a total of 38 numerical and 43 categorical features.

In [108…
```
print("Information on Variables")
df.info()
```

```
Information on Variables
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             1460 non-null   int64  
 1   MSSubClass     1460 non-null   int64  
 2   MSZoning       1460 non-null   object 
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64  
 5   Street         1460 non-null   object 
 6   Alley          91 non-null     object 
 7   LotShape       1460 non-null   object 
 8   LandContour    1460 non-null   object 
 9   Utilities      1460 non-null   object 
 10  LotConfig      1460 non-null   object 
 11  LandSlope      1460 non-null   object 
 12  Neighborhood   1460 non-null   object 
 13  Condition1     1460 non-null   object 
 14  Condition2     1460 non-null   object 
 15  BldgType       1460 non-null   object 
 16  HouseStyle     1460 non-null   object 
 17  OverallQual    1460 non-null   int64  
 18  OverallCond    1460 non-null   int64  
 19  YearBuilt      1460 non-null   int64  
 20  YearRemodAdd   1460 non-null   int64  
 21  RoofStyle      1460 non-null   object 
 22  RoofMatl       1460 non-null   object 
 23  Exterior1st    1460 non-null   object 
 24  Exterior2nd    1460 non-null   object 
 25  MasVnrType     1452 non-null   object 
 26  MasVnrArea     1452 non-null   float64
 27  ExterQual      1460 non-null   object 
 28  ExterCond      1460 non-null   object 
 29  Foundation     1460 non-null   object 
 30  BsmtQual       1423 non-null   object 
 31  BsmtCond       1423 non-null   object 
 32  BsmtExposure   1422 non-null   object 
 33  BsmtFinType1   1423 non-null   object 
 34  BsmtFinSF1     1460 non-null   int64  
 35  BsmtFinType2   1422 non-null   object 
 36  BsmtFinSF2     1460 non-null   int64  
 37  BsmtUnfSF      1460 non-null   int64  
 38  TotalBsmtSF    1460 non-null   int64  
 39  Heating        1460 non-null   object 
 40  HeatingQC      1460 non-null   object 
 41  CentralAir     1460 non-null   object 
 42  Electrical     1459 non-null   object 
 43  1stFlrSF       1460 non-null   int64  
 44  2ndFlrSF       1460 non-null   int64  
 45  LowQualFinSF   1460 non-null   int64  
 46  GrLivArea      1460 non-null   int64  
 47  BsmtFullBath   1460 non-null   int64  
 48  BsmtHalfBath   1460 non-null   int64  
 49  FullBath       1460 non-null   int64  
 50  HalfBath       1460 non-null   int64  
 51  BedroomAbvGr   1460 non-null   int64  
 52  KitchenAbvGr   1460 non-null   int64  
 53  KitchenQual    1460 non-null   object 
 54  TotRmsAbvGrd   1460 non-null   int64  
 55  Functional     1460 non-null   object 
 56  Fireplaces     1460 non-null   int64  
 57  FireplaceQu    770 non-null    object 
```

```
 58  GarageType     1379 non-null   object
 59  GarageYrBlt    1379 non-null   float64
 60  GarageFinish   1379 non-null   object
 61  GarageCars     1460 non-null   int64
 62  GarageArea     1460 non-null   int64
 63  GarageQual     1379 non-null   object
 64  GarageCond     1379 non-null   object
 65  PavedDrive     1460 non-null   object
 66  WoodDeckSF     1460 non-null   int64
 67  OpenPorchSF    1460 non-null   int64
 68  EnclosedPorch  1460 non-null   int64
 69  3SsnPorch      1460 non-null   int64
 70  ScreenPorch    1460 non-null   int64
 71  PoolArea       1460 non-null   int64
 72  PoolQC         7 non-null      object
 73  Fence          281 non-null    object
 74  MiscFeature    54 non-null     object
 75  MiscVal        1460 non-null   int64
 76  MoSold         1460 non-null   int64
 77  YrSold         1460 non-null   int64
 78  SaleType       1460 non-null   object
 79  SaleCondition  1460 non-null   object
 80  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
```

Part 2 - Task 1 (b)

Find the top 5 numerical features highly correlated with the target variable ("SalePrice") according to the pearson correlation, report the correlation values.

The top five values are the following, all of which are numerical:

1. OverallQual 0.790982
2. GrLivArea 0.708624
3. GarageCars 0.640409
4. GarageArea 0.623431
5. TotalBsmtSF 0.613581

In [109…

```
df.corrwith(df['SalePrice']).abs().sort_values()
```

Out[109]:
```
BsmtFinSF2       0.011378
BsmtHalfBath     0.016844
MiscVal          0.021190
Id               0.021917
LowQualFinSF     0.025606
YrSold           0.028923
3SsnPorch        0.044584
MoSold           0.046432
OverallCond      0.077856
MSSubClass       0.084284
PoolArea         0.092404
ScreenPorch      0.111447
EnclosedPorch    0.128578
KitchenAbvGr     0.135907
BedroomAbvGr     0.168213
BsmtUnfSF        0.214479
BsmtFullBath     0.227122
LotArea          0.263843
HalfBath         0.284108
OpenPorchSF      0.315856
2ndFlrSF         0.319334
WoodDeckSF       0.324413
LotFrontage      0.351799
BsmtFinSF1       0.386420
Fireplaces       0.466929
MasVnrArea       0.477493
GarageYrBlt      0.486362
YearRemodAdd     0.507101
YearBuilt        0.522897
TotRmsAbvGrd     0.533723
FullBath         0.560664
1stFlrSF         0.605852
TotalBsmtSF      0.613581
GarageArea       0.623431
GarageCars       0.640409
GrLivArea        0.708624
OverallQual      0.790982
SalePrice        1.000000
dtype: float64
```

Part 1 - Task 1 (c)

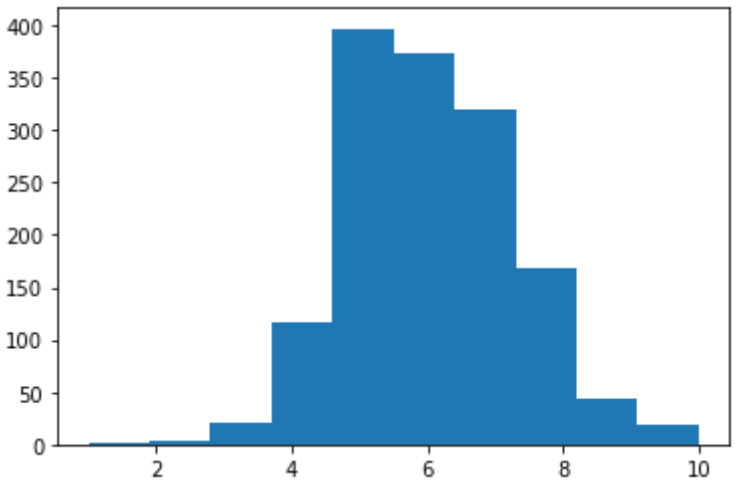
Plot the distributions of these 5 numerical features found in the previous question and the target variable using histograms with 10 bins, one for each feature/variable, describe the shape of their distributions with skewness and kurtosis (use Scipy for obtaining skewness and kurtosis values if you cannot do it in Orange), and tell two patterns from the histograms accordingly.

The SalePrice is skewed to the left (1.880940746034036), like TotalBsmtSF (1.522688086978629) and GrLivArea (1.365155954773434). Most kurtosis values are around 3, but the SalePrice is 9.509812011089439; the closest is GrLivArea at 7.874265760253215, meaning that GrLivArea is closest related to SalePrice.

In [110…
```
names = [ 'OverallQual', 'GrLivArea','GarageCars','GarageArea','TotalBsmtSF', 'Sale

for name in names:
    val = df[name]
    # Plot the distribution
    print("\n", name)
    plt.hist((val.abs()))
```

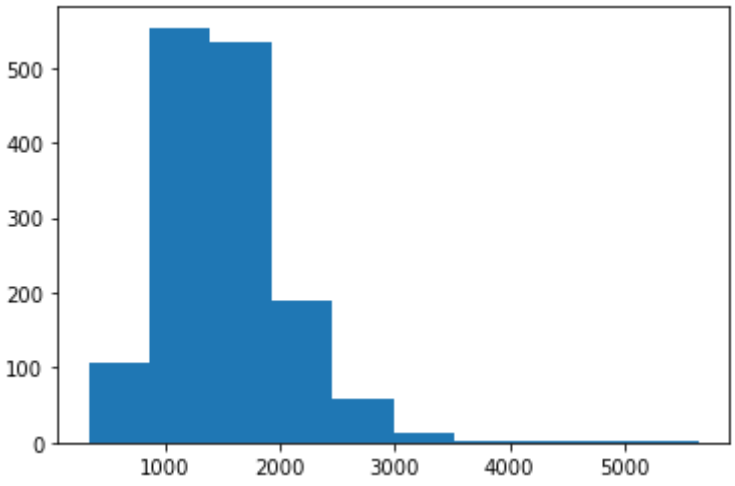
```
    plt.show()

    # Skewness

    print("Skewness: ", sp.stats.skew(val))

    # Kurtosis

    print("Kurtosis: ", sp.stats.kurtosis(val, fisher=False))
```

OverallQual



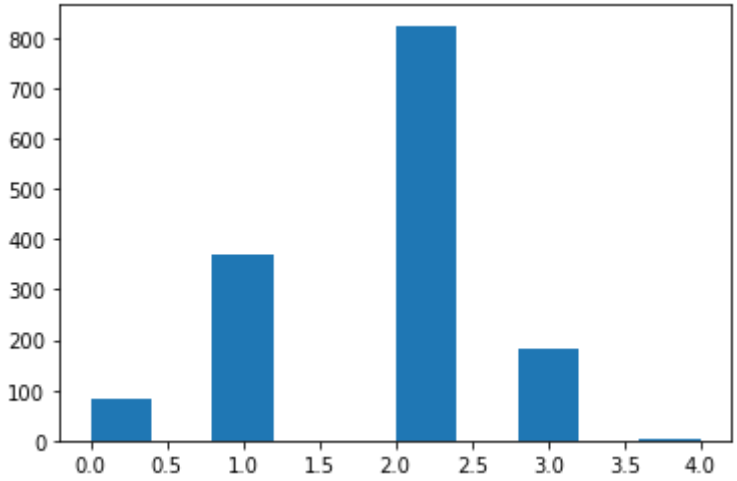

```
Skewness:  0.2167209765258641
Kurtosis:  3.091856548449611
```

GrLivArea



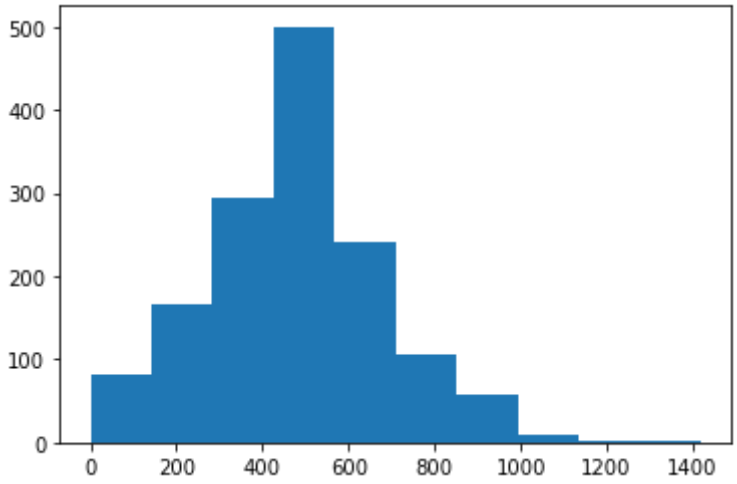

```
Skewness:  1.365155954773434
Kurtosis:  7.874265760253215
```

GarageCars



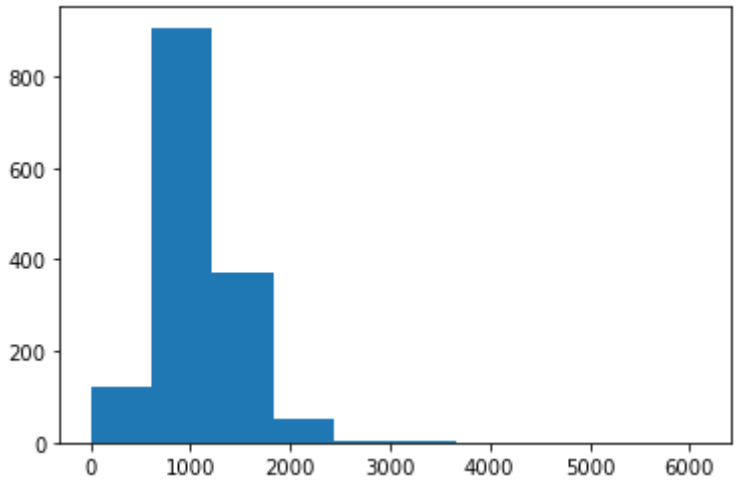

```
Skewness:  -0.34219689543081294
Kurtosis:  3.216134871511073
```

```
GarageArea
```



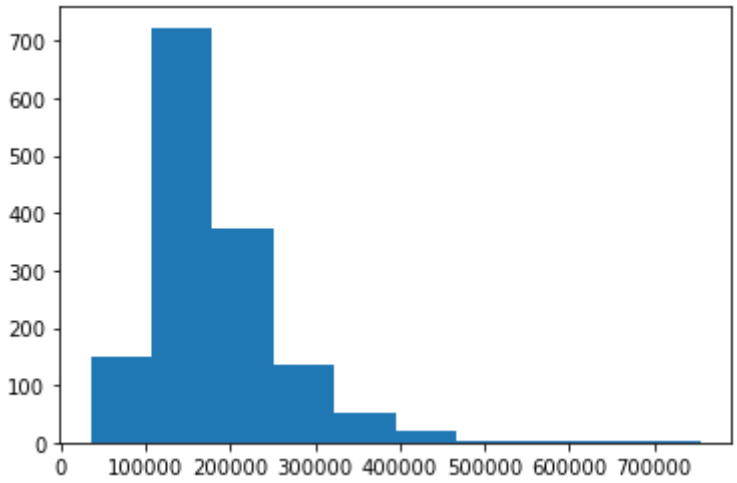

```
Skewness:  0.1797959420587727
Kurtosis:  3.9098227950882034
```

```
TotalBsmtSF
```




```
Skewness:  1.522688086978629
Kurtosis:  16.201041630635025
```

```
SalePrice
```




```
Skewness:  1.880940746034036
Kurtosis:  9.509812011089439
```

Part 1 - Task 1 (d)

Check for missing values. Is there any missing values in the data? Write a paragraph to briefly summarise the missing information regarding how many features contain missing values and at what percent.

19 features are missing values (which is circa 24% of features), and of those, only 3 (circa 4% of all features and 16% of all features missing values) features are missing less than 1% of values.

- LotFrontage: missing 259 values, at circa 18% missing.
- Alley: missing 1369 values, at circa 94% missing.
- MasVnrType: missing 8 values at circa 0.55% missing.
- MasVnrArea: missing 8 values at circa 0.55% missing.
- BsmtQual: missing 37 values at circa 2.53% missing.
- BsmtCond: missing 37 values at circa 2.53% missing.
- BsmtExposure: missing 38 values at circa 2.6% missing.
- BsmtFinType1: missing 37 values at circa 2.53% missing.
- BsmtFinType2: missing 38 values at circa 2.6% missing.
- Electrical: missing 1 values at circa 0.07% missing.
- FireplaceQu: missing 690 values at circa 47.26% missing.
- GarageType: missing 81 values at circa 5.55% missing.
- GarageYrBlt: missing 81 values at circa 5.55% missing.
- GarageFinish: missing 81 values at circa 5.55% missing.
- GarageQual: missing 81 values at circa 5.55% missing.
- GarageCond: missing 81 values at circa 5.55% missing.
- PoolQC: missing 1453 values at circa 99.52% missing.
- Fence: missing 1179 values at circa 80.75% missing.
- MiscFeature: missing 1406 values at circa 96.3% missing.

In [111…

```
print("Information on Variables")
df.info()
```

```
Information on Variables
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             1460 non-null   int64  
 1   MSSubClass     1460 non-null   int64  
 2   MSZoning       1460 non-null   object 
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64  
 5   Street         1460 non-null   object 
 6   Alley          91 non-null     object 
 7   LotShape       1460 non-null   object 
 8   LandContour    1460 non-null   object 
 9   Utilities      1460 non-null   object 
 10  LotConfig      1460 non-null   object 
 11  LandSlope      1460 non-null   object 
 12  Neighborhood   1460 non-null   object 
 13  Condition1     1460 non-null   object 
 14  Condition2     1460 non-null   object 
 15  BldgType       1460 non-null   object 
 16  HouseStyle     1460 non-null   object 
 17  OverallQual    1460 non-null   int64  
 18  OverallCond    1460 non-null   int64  
 19  YearBuilt      1460 non-null   int64  
 20  YearRemodAdd   1460 non-null   int64  
 21  RoofStyle      1460 non-null   object 
 22  RoofMatl       1460 non-null   object 
 23  Exterior1st    1460 non-null   object 
 24  Exterior2nd    1460 non-null   object 
 25  MasVnrType     1452 non-null   object 
 26  MasVnrArea     1452 non-null   float64
 27  ExterQual      1460 non-null   object 
 28  ExterCond      1460 non-null   object 
 29  Foundation     1460 non-null   object 
 30  BsmtQual       1423 non-null   object 
 31  BsmtCond       1423 non-null   object 
 32  BsmtExposure   1422 non-null   object 
 33  BsmtFinType1   1423 non-null   object 
 34  BsmtFinSF1     1460 non-null   int64  
 35  BsmtFinType2   1422 non-null   object 
 36  BsmtFinSF2     1460 non-null   int64  
 37  BsmtUnfSF      1460 non-null   int64  
 38  TotalBsmtSF    1460 non-null   int64  
 39  Heating        1460 non-null   object 
 40  HeatingQC      1460 non-null   object 
 41  CentralAir     1460 non-null   object 
 42  Electrical     1459 non-null   object 
 43  1stFlrSF       1460 non-null   int64  
 44  2ndFlrSF       1460 non-null   int64  
 45  LowQualFinSF   1460 non-null   int64  
 46  GrLivArea      1460 non-null   int64  
 47  BsmtFullBath   1460 non-null   int64  
 48  BsmtHalfBath   1460 non-null   int64  
 49  FullBath       1460 non-null   int64  
 50  HalfBath       1460 non-null   int64  
 51  BedroomAbvGr   1460 non-null   int64  
 52  KitchenAbvGr   1460 non-null   int64  
 53  KitchenQual    1460 non-null   object 
 54  TotRmsAbvGrd   1460 non-null   int64  
 55  Functional     1460 non-null   object 
 56  Fireplaces     1460 non-null   int64  
 57  FireplaceQu    770 non-null    object 
```

```
 58  GarageType     1379 non-null   object
 59  GarageYrBlt    1379 non-null   float64
 60  GarageFinish   1379 non-null   object
 61  GarageCars     1460 non-null   int64
 62  GarageArea     1460 non-null   int64
 63  GarageQual     1379 non-null   object
 64  GarageCond     1379 non-null   object
 65  PavedDrive     1460 non-null   object
 66  WoodDeckSF     1460 non-null   int64
 67  OpenPorchSF    1460 non-null   int64
 68  EnclosedPorch  1460 non-null   int64
 69  3SsnPorch      1460 non-null   int64
 70  ScreenPorch    1460 non-null   int64
 71  PoolArea       1460 non-null   int64
 72  PoolQC         7 non-null      object
 73  Fence          281 non-null    object
 74  MiscFeature    54 non-null     object
 75  MiscVal        1460 non-null   int64
 76  MoSold         1460 non-null   int64
 77  YrSold         1460 non-null   int64
 78  SaleType       1460 non-null   object
 79  SaleCondition  1460 non-null   object
 80  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
```

Part 1 - Task 2

Investigate the business understanding questions based on your exploration of the data. Two key business understanding questions (or business objectives) are “what factors affect the house price?” and “how do these factors affect the house price?”/“in which way do the factors affect the house price?”

Translate the two business questions into two data mining goals.

"What factors affect the house price?" becomes ""By what magnitude does each factor affect the outcome variable (HousePrice)?" "How do these factors affect the house price?"/"In which way do the factors affect the house price?" becomes "How do the features corelate to the outcome? E.g. are they inverse, ..."

Select two machine learning paradigms, e.g. classification, regression, dimensionality reduction and so forth, that can help you achieve these goals. Provide justifications of your decision.

Given that the outcome needs to be a value, regression is the best answer, as it provides means of directly calculating the predicted price based on the features. However, classification may also work if you classify each house price into a category, for example, a class that contains all house prices including the range of 20000 to 30000, 30001 to 40000 etc.

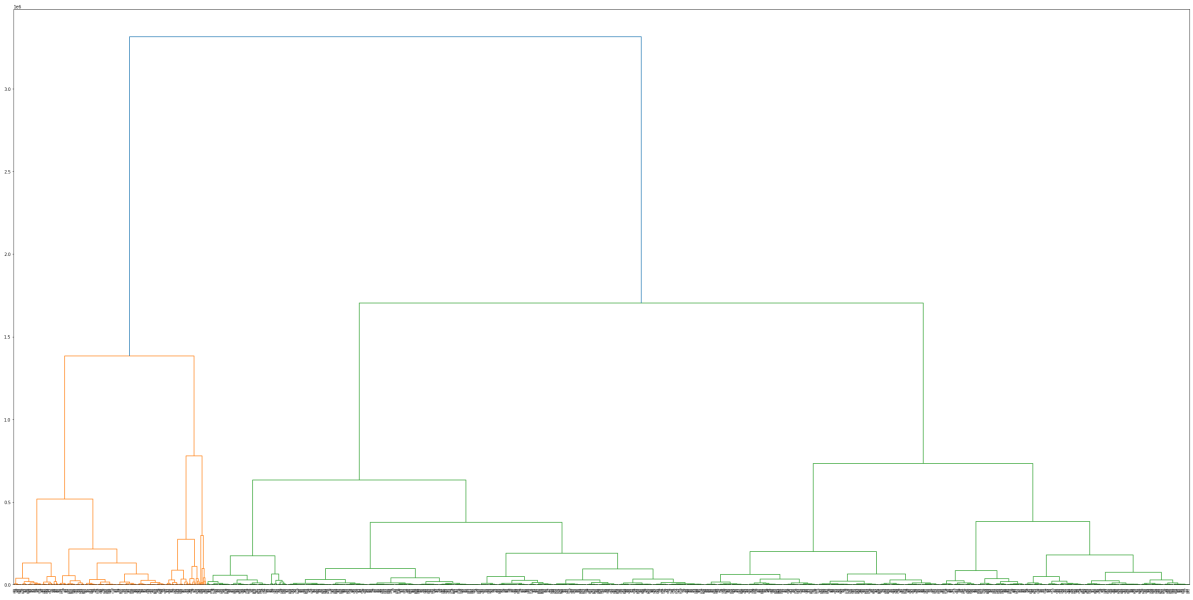
Then, based on the regression algorithm, you can easily uncover the magnitude of influence each feature has, and how they corelate; for example, if you use symbolic regression, you can see that feature x is multiplied by 38.2 while feature y is multiplied by 19.2, meaning x has a greater magnitude. If feature z, for example, is 1/z, it is clearly inversely corelated.

Part 1 - Task 3

EDA using clustering is very useful for understanding the important characteristics of the data. Provide a further EDA on the dataset using Hierarchical clustering on the 5 numerical features found in 1(b) to answer the question — "Does the house prices vary by neighborhood?". Report the output dendrogram and any other plots and show how do they help you to answer the question.

In [112…

```
from scipy.cluster.hierarchy import dendrogram, linkage

# feature selection

selectedData = df[[ 'OverallQual', 'GrLivArea','GarageCars','GarageArea','TotalBsm

Z = linkage(selectedData, 'ward')

plt.figure(figsize=(50, 25))
dendrogram(Z)
plt.show()
```



2.2 Data Preparation and Machine Learning

Address the business question of "what factors affect the house price and in which way?" using the provided dataset. Note that for supervised learning, it is important to partition the data before data preparation to avoid data leakage. Before answering the following questions, you need to split the data into a training set and a test set with a 70-30 splitting (use a random state=309).

In [113…

```
import numpy as np
from sklearn.model_selection import train_test_split

# split data set
X_train, X_test, y_train, y_test = train_test_split(df.loc[:, df.columns != 'SaleP
```

Determine and describe the data preprocessing steps applied to the provided dataset, e.g. handle missing data, encoding categorical data, normalise the data if necessary, and/or remove any unnecessary instances, these could be redundant instances,

outliers or non-effective instances and so forth. Show the process in your code/workflow. Submit a copy of the processed dataset (in CSV format).

In [114…]
```
# handle missing data - delete all features w/ missing data, as per Friday 26/08 he

listL = df.columns
listT = []
labels = []
newDF = df.copy()

for label in listL:
    if(label != "SalePrice" and label !="Id" and X_train[label].isnull().sum() != 0
        X_train = X_train.drop(label, axis=1)
        newDF = newDF.drop(label, axis=1)
        listT.append(label)
    else:
        labels.append(label)

for label in listT:
    X_test = X_test.drop(label, axis=1)

newDF = newDF.drop("Id", axis=1)

newDF = newDF.drop("SalePrice", axis=1)

print("Dropped features: ", listT)
```

```
Dropped features:  ['LotFrontage', 'Alley', 'MasVnrType', 'MasVnrArea', 'BsmtQua
l', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2', 'Electrical', 'Fir
eplaceQu', 'GarageType', 'GarageYrBlt', 'GarageFinish', 'GarageQual', 'GarageCon
d', 'PoolQC', 'Fence', 'MiscFeature']
```

In [115…]
```
# drop id too
# X_train = X_train.drop(['Id'], axis=1)
# X_test = X_test.drop(['Id'], axis=1)

# remove duplicates
X_train = X_train.drop_duplicates()
X_test = X_test.drop_duplicates()
```

In [116…]
```
X_train = pd.DataFrame(X_train)
X_test = pd.DataFrame(X_test)
y_train = pd.DataFrame(y_train)
y_test = pd.DataFrame(y_test)

# encoding categorical data and scaling
listOfCategorica = ['MSZoning', 'Street', 'Alley', 'LotShape',
                    'LandContour', 'Utilities', 'LotConfig',
                    'LandSlope', 'Neighborhood', 'Condition1', 'Condition2',
                    'BldgType', 'HouseStyle',  'RoofStyle', 'RoofMatl',
                    'Exterior1st', 'Exterior2nd', 'MasVnrType',
                    'Foundation',  'Heating', 'CentralAir', 'Electrical', 'SaleType
                    'SaleCondition', 'MiscFeature', 'PavedDrive',
                   'GarageType', 'MSSubClass']
listOfNumericals = ['OverallQual', 'OverallCond',
                    'KitchenQual', 'BsmtQual', 'BsmtCond',
                    'BsmtExposure', 'HeatingQC', 'BsmtFinType1','BsmtFinType2',
                    'ExterQual', 'ExterCond', 'PoolQC', 'Fence', 'GarageCond',
                   'GarageQual', 'GarageFinish', 'FireplaceQu',
                   'Functional', 'MSSubClass']
listOfContinuous = ['LotFrontage', 'LotArea', 'YearBuilt', 'YearRemodAdd',
                    'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF',
                    'TotalBsmtSF','1stFlrSF','2ndFlrSF','LowQualFinSF',
```

```
                    'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
                    'HalfBath', 'Bedroom', 'Kitchen', 'YrSold', 'MoSold', 'MiscVal
                    'PoolArea', 'ScreenPorch', '3SsnPorch', 'EnclosedPorch',
                    'OpenPorchSF', 'WoodDeckSF', 'GarageArea', 'GarageCars',
                   'GarageYrBlt', 'Fireplaces', 'TotRmsAbvGrd', 'KitchenAbvGr',
                   'BedroomAbvGr']


for deleted in listT:
    if(listOfCategorica.count(deleted) != 0):
        listOfCategorica.remove(deleted)
    if(listOfNumericals.count(deleted) != 0):
        listOfNumericals.remove(deleted)
    if(listOfContinuous.count(deleted) != 0):
        listOfContinuous.remove(deleted)
```

In [117…

```
# one hot encoding
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
encoders = [] # also scalers

for label in labels:
    if label == "SalePrice":
        continue
    elif label in listOfNumericals:
        enc = OrdinalEncoder()
        enc.fit(df[[label]])

        transformed1 = enc.transform(X_test[[label]])
        transformed2 = enc.transform(X_train[[label]])

        X_test[label] = transformed1
        X_train[label] = transformed2

    elif label in listOfContinuous: # scaling here
        # based on https://stackabuse.com/dimensionality-reduction-in-python-with-s
        scaler = StandardScaler()
        scaler = scaler.fit(X_train[[label]])

        transformed1 = scaler.transform(X_test[[label]])
        transformed2 = scaler.transform(X_train[[label]])

        X_test[label] = transformed1
        X_train[label] = transformed2


# from https://datagy.io/sklearn-one-hot-encode/
from sklearn.compose import make_column_transformer
transformer = make_column_transformer( (OneHotEncoder(sparse=False, handle_unknown=

transformer.fit(newDF)
transformed = transformer.transform(X_train)
X_train = pd.DataFrame( transformed,  columns=transformer.get_feature_names() )

transformed = transformer.transform(X_test)
X_test = pd.DataFrame( transformed,  columns=transformer.get_feature_names() )

display(X_train)
print(X_train.size)
```

```
D:\Programs\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Func
tion get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and w
ill be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
D:\Programs\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Func
tion get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and w
ill be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

	onehotencoder__x0_C (all)	onehotencoder__x0_FV	onehotencoder__x0_RH	onehotencoder__x0_RL
0	0.0	0.0	0.0	1.0
1	0.0	0.0	0.0	1.0
2	0.0	1.0	0.0	0.0
3	0.0	0.0	0.0	1.0
4	0.0	0.0	0.0	1.0
...	...	...	...	...
1017	0.0	0.0	0.0	1.0
1018	0.0	0.0	0.0	1.0
1019	0.0	0.0	0.0	1.0
1020	1.0	0.0	0.0	0.0
1021	0.0	0.0	0.0	0.0

1022 rows × 209 columns

213598

As you can see, I fitted the encoders with the entire data set, not just the training set. I believe that this is not data leakage, as usually, *ALL* possible categories would be known, so even if the training set doesn't contain a data set, the categories are known to the people developing this data analysis. I ran it on the whole data set, so I wouldn't need to go through each category and manually write down all categories (as I don't have time for that, sorry.)

In [118…

```
# export!

result = pd.concat([X_train, X_test], axis=0)
resul2 = pd.concat([y_train, y_test], axis=0)

result.insert(len(result.columns), "SalePrice", resul2["SalePrice"])

result.to_csv('changed.csv', index=False)
```

Utilise two different dimensionality reduction techniques to identify which features are irrelevant and/or redundant to predicting the house price. Report the dimension reduction process and remove redundant/irrelevant data. Show the process in your code/workflow.

In [119…

```
# dimensionality reduction technique #1
from sklearn.feature_selection import VarianceThreshold
thresholdA = 0.7
```

```
sel = VarianceThreshold(threshold=thresholdA)
data1 = X_train.copy()
data2 = X_test.copy()
outcome1_tr = pd.DataFrame(sel.fit_transform(data1) )
outcome1_te = pd.DataFrame(sel.transform(data2) )

# keep labels
# from https://stackoverflow.com/questions/39812885/retain-feature-names-after-scik
features = sel.get_support(indices = True) #returns an array of integers correspond
features = data1.columns.values[sel.get_support(indices = True)]

outcome1_tr.columns = features
outcome1_te.columns = features

display(outcome1_tr)
print(outcome1_tr.columns)
```

	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	BsmtFinSF1	BsmtFinSF2
0	0.061833	6.0	4.0	0.897528	0.708786	-0.961380	-0.310489
1	0.017372	7.0	4.0	1.202484	1.097845	-0.918642	-0.310489
2	-0.842755	6.0	4.0	0.931412	0.757418	0.128445	-0.310489
3	-0.179351	4.0	6.0	0.185964	1.146477	0.175457	0.546585
4	-0.175954	8.0	4.0	1.100832	1.000580	1.538807	-0.310489
...	...	...	...	...	...	...	...
1017	-0.435121	7.0	4.0	1.168600	1.049212	0.882775	-0.310489
1018	0.012577	5.0	5.0	-0.423948	-1.236509	-0.961380	-0.310489
1019	-0.174755	5.0	4.0	1.134716	1.049212	-0.961380	-0.310489
1020	-0.159369	1.0	2.0	-0.762788	-1.674200	-0.854535	-0.310489
1021	-0.509055	4.0	6.0	-1.440467	0.806050	-0.461343	3.808226

1022 rows × 34 columns

```
Index(['LotArea', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
       'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'HeatingQC',
       '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'BsmtFullBath',
       'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr',
       'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'GarageCars', 'GarageArea',
       'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch',
       'ScreenPorch', 'PoolArea', 'MiscVal', 'MoSold', 'YrSold'],
      dtype='object')
```

In [120…

```
# dimensionality reduction technique #2
from sklearn.svm import LinearSVC
from sklearn.datasets import load_iris
from sklearn.feature_selection import SelectFromModel

lsvc = LinearSVC(C=0.01, penalty="l1", dual=False).fit(X_train, y_train)

data1 = X_train.copy()
data2 = X_test.copy()
outcome2_tr = pd.DataFrame(sel.fit_transform(data1) )
outcome2_te = pd.DataFrame(sel.transform(data2) )

# keep labels
```

```
# from https://stackoverflow.com/questions/39812885/retain-feature-names-after-scik
features = sel.get_support(indices = True) #returns an array of integers correspond
features = data1.columns.values[sel.get_support(indices = True)]

outcome2_tr.columns = features
outcome2_te.columns = features

display(outcome2_tr)
print(outcome2_tr.columns)
```

```
D:\Programs\lib\site-packages\sklearn\utils\validation.py:993: DataConversionWarni
ng: A column-vector y was passed when a 1d array was expected. Please change the s
hape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	BsmtFinSF1	BsmtFinSF2
0	0.061833	6.0	4.0	0.897528	0.708786	-0.961380	-0.310489
1	0.017372	7.0	4.0	1.202484	1.097845	-0.918642	-0.310489
2	-0.842755	6.0	4.0	0.931412	0.757418	0.128445	-0.310489
3	-0.179351	4.0	6.0	0.185964	1.146477	0.175457	0.546585
4	-0.175954	8.0	4.0	1.100832	1.000580	1.538807	-0.310489
...	...	...	...	...	...	...	...
1017	-0.435121	7.0	4.0	1.168600	1.049212	0.882775	-0.310489
1018	0.012577	5.0	5.0	-0.423948	-1.236509	-0.961380	-0.310489
1019	-0.174755	5.0	4.0	1.134716	1.049212	-0.961380	-0.310489
1020	-0.159369	1.0	2.0	-0.762788	-1.674200	-0.854535	-0.310489
1021	-0.509055	4.0	6.0	-1.440467	0.806050	-0.461343	3.808226

1022 rows × 34 columns

```
Index(['LotArea', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
       'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'HeatingQC',
       '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'BsmtFullBath',
       'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr',
       'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'GarageCars', 'GarageArea',
       'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch',
       'ScreenPorch', 'PoolArea', 'MiscVal', 'MoSold', 'YrSold'],
      dtype='object')
```

The final outcome of both dimensionality reduction methods returns the same features; this gives me a lot of confidence into the result. Redundant features have been removed. Going from 200+ features to 34 is a huge difference and feels very appropriate. Looking at the answers, most of them are not categorical; it appears that one-hot encoding doesn't work well with this type of dimensionality reduction, probably due to the relatively low variance (only 0 vs 1) in values. This does bring up something to be considered in future EDAs, with how to deal with that, but at present time, I'm not sure.

Now approach data mining goals on your preprocessed data using machine learning methods.

(a) With the two groups of features selected in previous question, use the ordinary linear regression and ridge regression (with alpha=0.5) for predicting the house prices. Comparing their results regarding the mean squared errors on the training set and the test set. Present and analyse the learnt regression models, and highlight your observations. Submit your code or workflow.

In [121…
```
# ordinary linear regression

from sklearn.linear_model import LinearRegression

reg1 = LinearRegression().fit(outcome1_tr, y_train)
reg2 = LinearRegression().fit(outcome2_tr, y_train)

print("SCORE FOR #1", reg1.score(outcome1_te, y_test))
print("SCORE FOR #2", reg2.score(outcome2_te, y_test))
```

```
SCORE FOR #1 0.8301319438871382
SCORE FOR #2 0.8301319438871382
```

In [122…
```
# ridge regression (with alpha=0.5 )

from sklearn.linear_model import Ridge

clf1 = Ridge(alpha=1.0)
clf1.fit(outcome1_tr, y_train)
print("OUTCOME #1 ", clf1.score(outcome1_te, y_test))


clf2 = Ridge(alpha=1.0)
clf2.fit(outcome2_tr, y_train)
print("OUTCOME #2 ", clf2.score(outcome2_te, y_test))
```

```
OUTCOME #1  0.8299898099627188
OUTCOME #2  0.8299898099627188
```

In [123…
```
def meanSquareError(x, y, model):

    total = 0.0
    res = model.predict(x)
    total = ( (res - y) * (res - y) )
    mse =  (total.sum() / len(x))
    return mse

reg1_m = meanSquareError(outcome1_te, y_test, reg1)
reg2_m = meanSquareError(outcome2_te, y_test, reg2)

clf1_m = meanSquareError(outcome1_te, y_test, clf1)
clf2_m = meanSquareError(outcome2_te, y_test, clf2)

print("Linear - #1: ", reg1_m)
print("Linear - #2: ", reg2_m)
print("Ridge - #1: ", clf1_m)
print("Ridge - #2: ", clf2_m)
```

```
Linear - #1:  SalePrice    1.080232e+09
dtype: float64
Linear - #2:  SalePrice    1.080232e+09
dtype: float64
Ridge - #1:  SalePrice    1.081136e+09
dtype: float64
Ridge - #2:  SalePrice    1.081136e+09
dtype: float64
```

While I'm not 100% sure that the code is correct (I couldn't figure out how to fix the mse), if it is, there isn't a difference between the "two" sets of features, because both models chose the same sets of features. However, there is a slight difference in error, with the ordinary linear regression having a slightly smaller error than the ridge regression.

(b) Using Random Forest, which is a more powerful ensemble regression method to predict the house price, compare with the results of linear regression and ridge regression, highlight your observations.

In [124…

```
# random forest
def meanSquareError(x, y, model):

    total = 0.0
    res = model.predict(x)
    total = ( (res.reshape(438,1) - y) * (res.reshape(438,1) - y) )
    mse =  (total.sum() / len(x))
    return mse

from sklearn.ensemble import RandomForestClassifier

clf1 = RandomForestClassifier(max_depth=2, random_state=0)
clf1 = clf1.fit(outcome1_tr, y_train)

clf2 = RandomForestClassifier(max_depth=2, random_state=0)
clf2 = clf2.fit(outcome2_tr, y_train)

clf1_m = meanSquareError(outcome1_te, y_test, clf1)
clf2_m = meanSquareError(outcome2_te, y_test, clf2)

print("Ridge - #1: ", clf1_m)
print("Ridge - #2: ", clf2_m)
```

```
C:\Users\Ella\AppData\Local\Temp\ipykernel_23340\3192459535.py:13: DataConversionW
arning: A column-vector y was passed when a 1d array was expected. Please change t
he shape of y to (n_samples,), for example using ravel().
  clf1 = clf1.fit(outcome1_tr, y_train)
C:\Users\Ella\AppData\Local\Temp\ipykernel_23340\3192459535.py:16: DataConversionW
arning: A column-vector y was passed when a 1d array was expected. Please change t
he shape of y to (n_samples,), for example using ravel().
  clf2 = clf2.fit(outcome2_tr, y_train)
Ridge - #1:  SalePrice    6.576558e+09
dtype: float64
Ridge - #2:  SalePrice    6.576558e+09
dtype: float64
```

I'm surprised. While I'm not 100% sure that the code is correct (I couldn't figure out how to fix the mse), the method of calculating the mse is the same, and the length of the test array is also the same, meaning that the error of the Random Forest being larger shows that the other two are more correct. Given that the Random Forest is described as more powerful, this is surprising and makes me doubt my code, but I'm already behind the deadline and cannot figure out a bug in the code either.