1) Describe your data set, why you chose it, and what you are trying to predict with it

Project1: Data Description

For project 1, I used the New York City Yellow Taxicab Data to predict Trip prices based on some other attributes.

The data was collected between 2008 and 2010 using a mechanical GPS reader in the cab meter. The data is reported by a trip for the 19 000 trips. The data has 26 variables: a starting point, ending point, trip time, distance traveled, and several other features for each trip. The description of the full list of features can be retrieved here.

For training purposes, I switched to using a dataset that would allow practicing much more feature engineering techniques. The taxi data I used for project 1 had only one categorical feature, given that in real-world datasets have more categorical variables I decided to use another dataset to have a fuller experience.

Because all the steps I did to clean the taxi data were replicated in Project2, For the rest of this paper I'm only going to detail what I did to clean up the data I used in the second project.

Project2: Data Description

For project 2, I used the well-known housing prices dataset compiled by Dean De Cock for use in data science education. The dataset has 79 explanatory features describing most aspects of residential homes in Iowa. The goal of the project was to predict the final price of each home based on the remaining 78 variables in the dataset. In other words, to minimize the difference between the real price and the price estimated by our models. The description of the full list of features can be retrieved here.

2) Detail what you did to clean your data and any changes in the data representation that you applied. Discuss any challenges that arose during this process.

To clean the data and apply the necessary changes, I analyzed the data to learn more about the features characteristics. I analyzed the data to identify for the following:

- Missing values
- Duplicate, Constant and Quasi-constant variables
- Numerical variables
- Distribution of the numerical variables
- Categorical variables
- The cardinality of the categorical variables
- The potential relationship between the variables and the target: SalePrice
- Extracted new features: house age, time since renovation etc..

A) Missing values

For missing data imputation it is crucial to understand the representation, the numbers, and percentages of missing data at the Dataframe level as well as at features level. To my knowledge, there isn't a function that does that, and that we would have to use multiple pandas functions to achieve that. I thought we could make the process a lot easier, hence the motivation to write a function that simplifies the process. I named the function, **missing_data**.

The Functions takes 3 arguments:

- **DataFrame**
- **Per_NaN:** This argument allows you to select the features with a percentage of **NaN**. For example, if you set Per_NaN=5 it will return the features where the percentage of NaN is less than 5 percent. It defaults to 100: returns all features with **NaN**.
- **Action**: This argument allows you to have control over what the function does: it is defaulted to print— print the details about the features with null values. if instead, you want just to return the list of features with **NaN** you passing a **'return'** as a string.

I used the function to return the features that contain over 95 percent of missing data, which then I deleted. I imputed the remaining features with missing data with the mean.

B) Duplicate, Constant and Quasi-constant, variables

Duplicate, constant and Quasi-constant features reduced the performance of all machine learning algorithms or at least don't add any predictive value. To identify these features we need to use several pandas functions in multiple lines of code. Given identifying and deleting these features is a common practice I wrote functions that simplifies the processes.

To find The duplicated features I wrote a function named **duplicated_features.** The function takes a DataFrame as an argument and returns a DataFrame with all the duplicated features. I used the function to remove the duplicated features in my dataset. Additionally, because it is common practice to remove the constant and Quasi-constant features, I wrote a function that simplifies the processes of identifying them. I named the function **filter_mothod**. The function takes two arguments: a DataFrame and a variance threshold. To remove the constant I passed the **DataFrame** and set the threshold to **0**. Similarly, to delete Quasi-constant features, I passed the DataFrame and set the threshold to **0.1**. This returns all the features where 99% of the observations are similar.

C) Distribution of the numerical variables

To get quick insights about the distribution of the numerical variables I plotted the histograms of the features. Next, I wrote a function that plots side by side each variable's histogram and Q-Q plots before and after the **Yeo-Johnson Transformation**. The latter transformation improved significantly the probability distribution of the majority of the features. I identified these features and transformed them using the **YeoJohnsonTransformer** object imported from the **feature-engine**.**varaible_transformers**.

D) Categorical variables

D-1) rare labels

Before encoding the categorical variables, I first handled the rare labels. It is known that Labels that are under-represented in the dataset tend to cause over-fitting of machine learning models. I **renamed** all the **labels** of categorical features **(that have cardinality higher than 2)** that are present in less than **1 percent** of observations into the label **rare**. The reason for excluding the features with a cardinality of 2 is that, it is just going to replace whatever the one label name with rare, which doesn't add any value.

To replace the label of features that are present in less than 1 percent of the observations, I wrote a function that takes 5 arguments:

the train_set,
test_set,
rare_list: list of features that have labels we want to replace by **rare**
target: the feature wee want to predict
rare_perc: threshold for rare labels.

The function returns a DataFrame with all the labels in all the features that are present under the specified threshold (in my example 1%) replaced by the label rare. Grouping all the rare labels under one umbrella term helps our model reduce overfitting and hence generalize better.

D-1) Ordinal or one-hot encoding

For encoding categorical variables I performed ordinal encoding on highly cardinal variables and one-hot encoding on the features with low cardinality. The reason for this is that one hot encoding works better with linear regression models but it expands the feature space dramatically. Another reason is that, Given that I used a tree-based algorithm and a linear model that work better with different encoding methods. That is ordinal encoding is suitable for trees based and one hot encoding suitable for linear models. For the above reasons, I performed ordinal encoding on variables with more than 4 categories and one-hot encoded all features with a cardinality of 4 or lower.

Ordinal encoding

To perform ordinal encoding I wrote a function, named **mean_ordinal_encoder,** that replaces the **labels** of each feature with an **integer**. The **integer value** corresponds to the mean of each label's house sales. in other words, the highest the mean of the category the higher the integer it is assigned. This is an attempt to create a monotonic relationship with the target feature.

One hot encoding

I one hot encoded all the remaining features with a cardinality of 4 or lower with to k-1 dummy variables in an attempt to reduce a bit the feature space. To one-hot encode the variables I used the **one-hot-encoder** object imported from **feature-engine**.

E) Extracted new features

Known that the age of the house could potentially be a determinant of the sales price, I extracted the age of the house as well as some useful features from the existing date variables. In the dataset, we have 4 date variables

YearBuilt the year in which the house was built
YearRemodAdd the year in which the house was renovated
GarageYrBlt the year in which the house was built the garage was built
YrSold the year in which the house was sold

From the above variables I extracted

house_age to capture the age of the house when it was sold
garage_age to capture the age of the garage when it was sold
renevated_age to capture the number of years since the house was renovated when it was sold

3) Discuss what you learned by visualizing your data

Visualizing the data is very beneficial as a primary set in any analyses. Visualizing the data to find the correlation between the target feature and the rest of the variables gives us a huge indicator of which variables are the most determinant of the target. A scatter plot between the target and the other variables helps to spot outliers.

4) Describe your experiments with the two supervised learning algorithms you chose. This should include a brief description, in your own words, of what the algorithms do and the parameters that you adjusted. You should also report the relative performance of the algorithms on predicting your target attribute, reflecting on the reasons for any differences in performance between models and parameter settings.

Lasso

Lasso or l1 regularization that gives us the ability to exclude some features when training the model. Regularisation consists of adding a penalty to the different parameters of the machine learning model to reduce the freedom of the model to avoid overfitting. The penalty is applied over the coefficients that multiply each of the predictors. From the different types of regularisation, Lasso or l1 has the property —the *alpha* parameter —that shrink some of the coefficients to zero. Therefore, that feature can be removed from the model.

When training the lasso model to predict the house's sales price, applying different alpha values yielded different results. The higher the value of *alpha* the higher the number of features that were shrunk to zero;

- for the alpha of **0.001,**
  - The model used **84** out of 106 features
  - The model reported **r2** of **0.927** on the **train set**
  - The model reported **r2** of **0.739** on the **test set**

- for the alpha of **0.005,**
  - The model used **32** out of 106 features
  - The model reported **r2** of **0.912** on the **train set**
  - The model reported **r2** of **0.769** on the **test set**

- for the alpha of **0.01,**
  - The model used **20** out of 106 features
  - The model reported **r2** of **0.893** on the **train set**
  - The model reported **r2** of **0.772** on the **test set**

We can conclude from the results that the higher the alpha (the penalty value) the greater the number of features that were shrunk to zero. Consequently, the higher the alpha the better our model performs on the train set, and the less the model overfits.

Random Forest

Random forest is a machine learning method that operates constructing multiple decision trees. The algorithm selects the best score from individual constructed trees using a voting mechanism. It is an ensemble method, it is better than a single decision tree because it reduces over-fitting by averaging the results of the individual trees. Some of the steps the algorithm follows are

1. it divides the dataset into samples
2. it constructs a decision tree from every sample
3. it uses all the constructed individual trees to make a prediction
4. it ranks all the individual trees based on their predictive scores
5. it repeats the operation bases on the set **n_estimators hyperparameter**
6. In the end, it selects the highest-ranked or the most voted prediction result as the final prediction

There are some important hyperparameters to improve the model predictions: the number of estimators (n_estimators), which represents the number of trees the model will be built upon; the maximum number of features allowed to be used in each tree (max_features); the minimum sample leaf size (min_sample_leaf) representing the minimum number of sample in each leaf node; the maximum depth of the tree (max_depth) is the number of levels in each tree.

When training the random forest model with the default parameters the model reported:

- An average Error of 0.0385 and;
- An accuracy of 99.51%

When training the model with the best hyperparameter return by a grid search (

```
{'n_estimators': 1000,
 'min_samples_split': 2,
 'min_samples_leaf': 2,
 'max_features': 'sqrt',
 'max_depth': 100,
 'bootstrap': False})
```

The model reported:

- An average Error of 0.0345 and;
- An accuracy of 99.71

5) Describe your experiments using PCA for feature selection, discussing whether it improved any of your results with your best performing supervised learning algorithm.

Indeed using ONLY the features that capture 95% of the variance has improved the generalization of both the models I used in project two. After applying the PCA method, Random Forest's accuracy has improved from 99.51% to 99.71%. In the case of Lasso Regression, the R_squared score on the test set of the best performing model in project2 is 0.73 while the R_squared score when using only the features that retain 95% of the variance is 0.83.

6) Discuss the results of using PCA as a pre-processing step for clustering. This should include a brief description, in your own words, of what the algorithms do. If you used the Wine dataset for this, briefly explain why your original dataset wasn't appropriate.

Yes, applying PCA as a pre-processing step for my data set led to better clustering performance bt the K-Means. Given that my data set comprised of 106 features, the dimensionality reduction by PCA proved to be beneficial and enhanced K-Means performance.

K-means is a method that (1) defines random centroids in a euclidian space, then (2) fits all the data points to the closest centroids. The algorithm defines randomly the number of centroids. Next, it assigns each point to the nearest corresponding centroid. Then calculates the mean of all the values of all points that belong to that centroid. The new mean is now the is the value of the centroid. These steps repeat until the new centroids are the same as the centroid themselves.

Agglomerative Clustering starts by considering a single data-point as a cluster. Then, it merges the most similar clusters into a bigger one and moves up in the hierarchy. The process continues until there is only one single cluster left containing all the data-points. Such as K-means, it requires the number of clusters (n_clusters). It is also possible to define the affinity, which means the metric to compute the type of distance to be used; the number of leaves (n_leves_) in the hierarchy tree; and the number of components in the graph (n_connected_components). The PCA improved both scores: the Silhouette Score increased from 0.048 to 0.122, and the Adjusted Rand Score came from 0.16 to 0.377.

DBSCAN or Density-based spatial clustering is based on the idea that each cluster is built upon a defined radius that contains a minimum number of data-points. The parameters require two parameters; the maximum distance between two data-points (eps), and the minimum number of samples (min_samples). The DBSCAN performed poorly in clustering the dataset. Despite that, the PCA has improved both scores for its results: the Silhouette Score increased from -0.67 to -0.396, and the Adjusted Rand Score came from 0 to 0.0056.

7) Summarize what you learned across the three projects, including what you think worked and what you would do differently if you had to do it over.

In the course of completing the tree project, I learned all I know now about machine learning. While there is much more learn, I believe the class has given me some solid foundations. I would change a thing if I had to do over, I’m satisfied with what I learned and eager to learn more.