Feature Engineering

2 - Introduction to feature engineering

- What is feature engineering?

It is process of using domain knowledge to extract features from raw data via data mining techniques.

Extracting Information:-

- More complex examples:
  - Text data for deed of house
    - Length of text
    - Number of times certain terms are mentioned -

Combining Information:-

- Could also combine extracted information
- New features:
  - 0 or 1 value indicating:
  - Both weekends and evening?

Transforming Information:-

- Often categorical data is present as string data.
- For example a large data set of social network users could have country of origin as a string feature (e.g. UK, USA, UAE, etc)

Two way / approaches of Transforming data :-

- Integers Encoding
- OneHotEncoding (Dummy variables)

Integers Encoding :-

- Directly convert categories into Integers 1, 2, 3 ... N -

Country	Country	possible Issue - apply order to the column (ordinal variables)
USA	1	
MEX	2	
CAN	3	
	1	

PROS :-
- very easy to do and understand
- Does not increase number of features

CONS
- Implies ordered relationship between categories

OneHotEncoding :-

Convert each category into individual features that are either 0 or 1

USA
MEX
CAN
USA

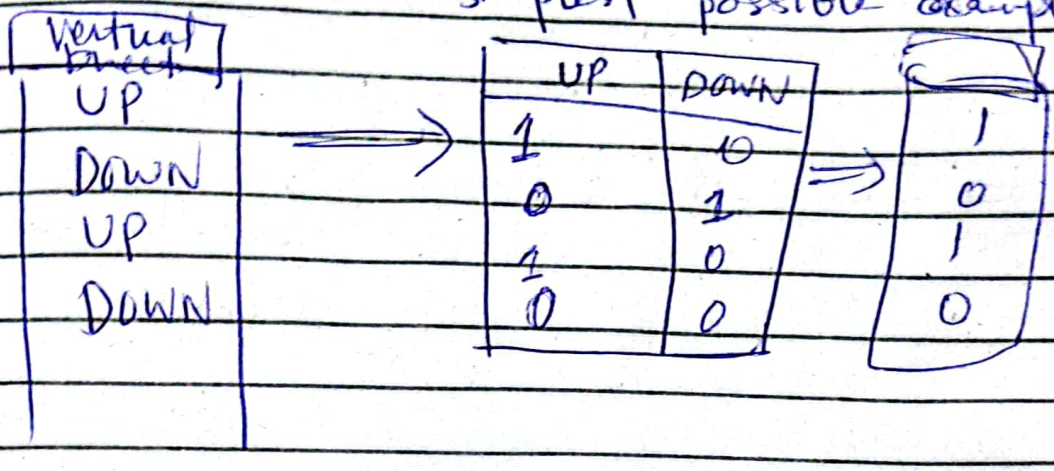
⟹

USA	MEX	CAN
1	0	0
0	1	0
0	0	1
1	0	0

- Issue greatly expand features

aware of

- **Dummy Variable Trap :-**
  - mathimatically known as "~~Multi-collinearity~~"
- Converting dummy variables can cause feature to be duplicated.
- Let consider simplest possible example.



Pros :-

- No ordering implied

CONS :-

- potential to create many more features column and weightages.
- Dummy variable trap consideration.
- Not easy to add new categories.

Throughout this Course :-

- Outliers in data
- Missing Data
- Categorical Data

- Not every issue here is feature engineering but also can be called data cleaning.

3. Dealing with Outliers

Outliers:-

Extreme data points which might disturb our data statistically, which might come in rare.

Its often better to simply remove these few points from the data set in order to have more generalized model.

Outlier consideration:-

- Definition of an Outlier
  - Range and limit
  - percentage of Data
- There are both very domain dependent!

Range and limit:-

we need to decide what will constitute an outliers with some methodology

- Inter Quartile Range
- Standard deviation
- Visualization or Domain limit values.

- percentage of Data
- keep in mind if a large percentage of your data is being labeled as an outlier, then you actually just have a wide distribution, not outliers.
- limit outlier to a few percentage points at most.

Outlier Consideration:-

- Utilize visualization plots to be able to see and identify outliers points -
- keep in mind this will create caveats for your future model (e.g Model not suitable for houses priced over $10 million)

```
Code #
Import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

Sns.displot(samples,)

Sns.boxplot(data = samples)

ser = pd.Series(p$sample)
ser.describe()
IQR = 55.25 - 42.0
1.5 * (IQR)
lowerlimit = 42.0 - 1.5*(IQR)
```

Q4

ser[ser > lower_limit]

q25 = np.percentile(sample, [75, 25])

iqR = q75 - q25

= 1.5 * IQR

= q25 - 1.5 * iqr

df = pd.read_csv("Read file name")

df.corr()["Sales"].sort_values()

sns.scatterplot(y="Saleprice", x="Overalla"

sns.scatterplot(x="Gr liv Area", y="Salespr")

df[(df['Overall Qual']>8) & (df['Saleprice'] <200000)]

df[(df['Gr liv Area']>4000)] & (df['Saleprice'] < 400000)]

dropIndex = ↗ index

df = df.drop(drop_ind, axis=0)

4- Dealing with Missing Data

```
Code #
usuall imports

with open('.../DATA/Ames_Housing.txt','r') as f
    print(f.read())

df = pd.read_csv("Ames_outliers_removed.csv")

df.info()
df.head()

df = df.drop("PID", axis=1)   # drop it

df.isnull().sum()

100 * df.isnull().sum() / len(df)

def percent_missing(df):
    percent_nan = 100 * df.isnull().sum()/len(df)
    percent_nan = percent_nan[percent_nan>0].sort_values()
    return percent_nan

percent_nan = percent_missing()

sns.barplot(x=percent_nan.index, y=percent_nan)
plt.xticks(rotation=90);
```

5- Dealing with Missing Data - part 2

```
percent_nan[percent_nan<1]

100/len(df)

df[df["Electrical'].isnull()]['Guarage Area']

df[df['Bsmt Half Bath'].isnull()]

df = df.dropna(axis=0, subset=['Electrical',
                               'Guarage Cars'])

percent_nan = percentmissy(df)

percent_nan[percent_nan<1]

df[df['Bsmt Half Bath'].isnull()]
```

- Basement Numeric columns ---> fillnull(values)

```
num_cols = ["All basements]

df[bsmt_num] = df[bsmt_numeric cu].
                fillna(0)
```

6. Dealing with Missing Data
part 3: Features columns

- Dropping the feature column:-
  - Very simple to do
  - No longer need to worry about that feature in the future.
  - potential to lose a feature with possible important signal
  - Should consider drop feature approaches where many rows are NaN.

- Filling in the missing features data:
  - potentially changing ground truth in data.
  - Must decide on reasonable estimation to filled value.
  - Must apply transformation to all future data for prediction.

- Simplest case:
  - Replace all NaN values with a reasonable assumption (e.g. zero if assumed NaN implied zero)

- Harder cases:
  - Must use statistical methods based on the columns to fill in NaN values.

pandas.transform (func =

call the groupby and fill values based on groupby

df.groupby('Neighborhood')
['LotFrontage']
.transform(lambda
value: value.fillna(
value.mean()))

7. Dealing with categorical

df['MS Subclass'] = df['MS SubClass'].apply(str)

direction = pd.Series(['Up', 'Down', 'Same'])

pd.get_dummies(direction, drop_first=True)

df = df.select_dtypes(include='object')
df2 = df.select_dtypes(exclude='object')

df_object_dummies = pd.get_dummies(df)

df_objects_dummies = pd.get_dummies(df, drop_first=True)

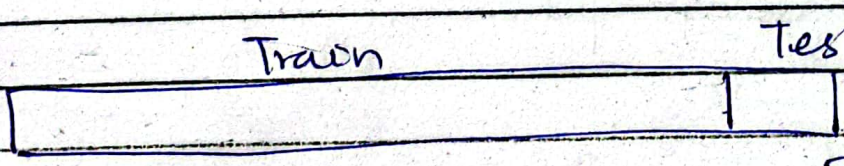
final_df = pd.concat([num_jumma_df, df_objects_dummies], axis=1)

12. Cross Validation, Grid Search and the Linear Regression project:-

Section Overview :-

- Cross Validation in Detail
  - Train | Test split
  - Train | validation | Test split
  - Scikit-learn cross_val_score
  - Scikit-lear cross_validate
- Grid Search
- Linear Regression Project Exercise

2. Cross validation : Train Test Split



allow us to adjust hyperparameter

Code :-

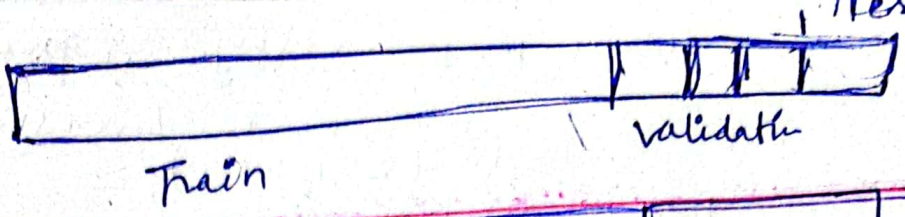
```
X = df.drop('sales', axis=1)
y = df['sales']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
```

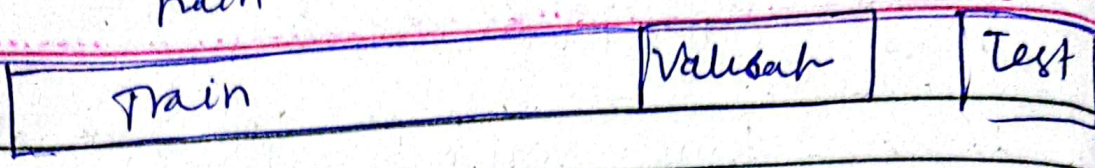
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

from sklearn.linear_model import Ridge
model = Ridge(alpha = 100)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

mean_squared_error(y_test, y_preds)
# model = Ridge(alpha = 1)
model_two.fit(X_train, y_train)
```

3.

3

- We just saw that train/Test split method has a disadvantages of not having a portion of data that can report a performance metrics on truly "unseen" data.
- while adjusting hyperparameters on test data is a fair technique to be referred as "data leakage", it is a potential issue in regards to reporting





- Recall the entire reason to not adjust after the final test data set is to get a ~~good~~ fairest evaluation of the model -
- The model was not fitted to the final test data and the model hyperparameter were adjusted based off the final test ~~data~~.
- This is truly never ~~before~~ seen data -!

```
df = df.head()
x = df.drop('Sales', axis=1)
y = df[Sales]
```

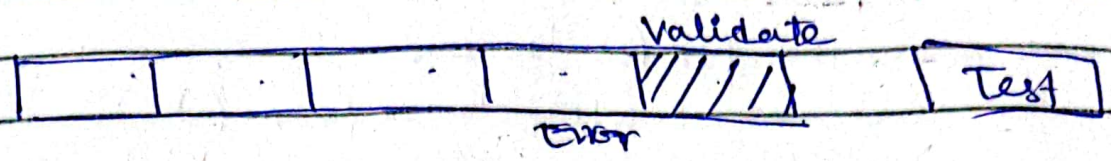
use two time test_train_split

=0.3

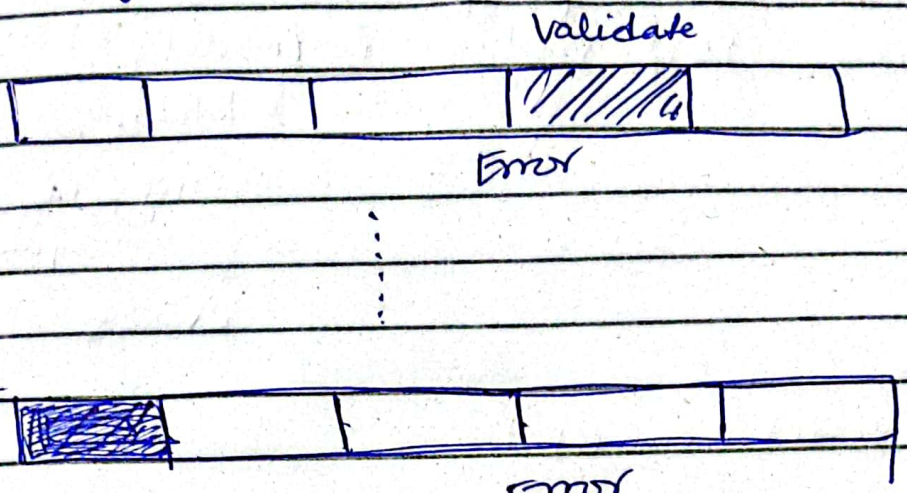
```
X_train, X_other, y_train, y_other = train_test_split(X,y
X_validate, X_test, y_validate, y_test = train_test_sp
from sklearn.preprocessing import st
scaler = StandardScaler()
scaler.fit(X_train)
X_train =
X_test =
```

4. Cross-Validation - cross_val_score

- Start with Training data and



chose K-fold values
- largest possible "leave one out"



and final test.

- The cross_val_score function uses a model and training set (along with cv and chosen metrics) to perform all of this for us. automatically!
- This allows for K-fold cross-validation to be performed on any model.
- Lets explore how to use it -

```
X = df.drop('sales', axis=1)
y = df['sales']
X_train, X_test, y_train, y_test = train_test_split(X, y, 0.3, )

# Scale data
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test  "  "  "    (X_test)

model = Ridge(alpha=100)
score = cross_val_score(model, X_train, scoring='neg_mean_squared_error', cv=5)

scores.
abs(scores.mean())

# Same process with
    alpha = 1
```

5- Cross Validation.
- Cross_Validate

- The cross_validate function allow us to view multiple performance metrics from cross validate on a model and explore how much time fitting and testing took.
- lets quickly review how to use this function call.

Code #

same code as below for Setup

But

```
score = cross_validate(model, X_train,
                       y_train,
                       scoring = ['neg_mean_
                                  squared_error',
                                  'neg_mean_absolute
                                  _error'], cv=10)
```

6- Grid Search

- Often more complex models have multiple adjustable hyperparameters -
- A grid search is a way of training and validating a model on every possible combination of multiple hyperparameters options -

- Scikit learn includes a GridSearchCV class capable of testing a dictionary of multiples hyperparameter option through cross-validation -
- This allows us for both cross-validation and a grid search to be performed in a generalized way for any model.

#1 code

same as before

from sklearn.model_selection ElasticNet

model = ElasticNet()
param_grid = { 'alpha' : [0.1, 1, 5, 10, 50],
'l1_ration' : [0.1, 0.5, 0.7, 0.99] }

from sklearn.model_selection import