Capstone Project (Model Selection)

Aim: To select the best model

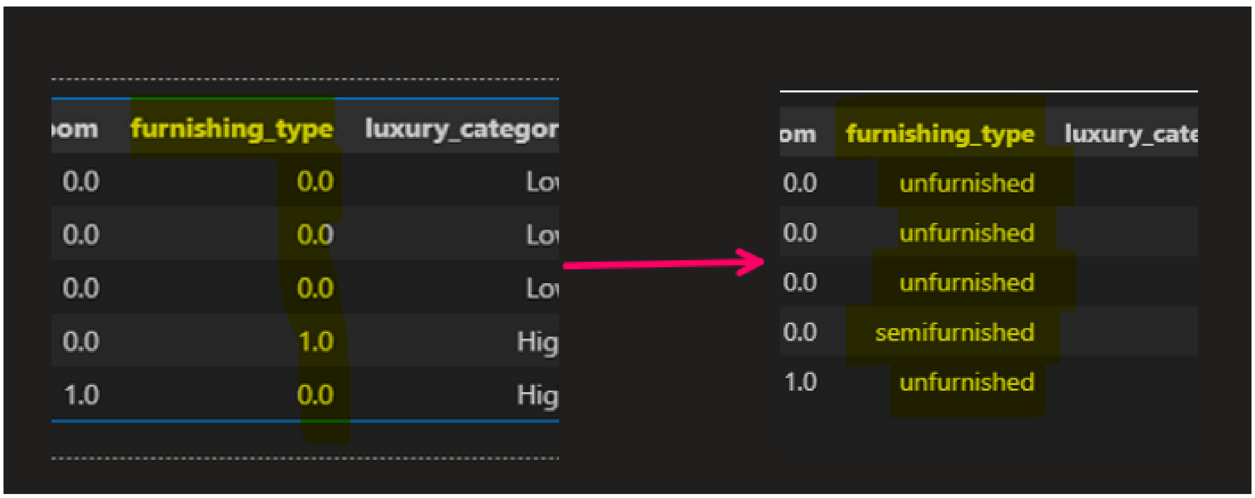
```
import numpy as np
import pandas as pd

from sklearn.model_selection import KFold, cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler, OrdinalE
ncoder
from sklearn.compose import ColumnTransformer
from sklearn.svm import SVR

from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error

from sklearn.decomposition import PCA
```

```
df = pd.read_csv('gurgaon_properties_post_feature_selection_v2.csv')
```

```
df['furnishing_type'].value_counts()
```

```
furnishing_type
0.0    2349
1.0    1018
2.0     187
Name: count, dtype: int64
```

```
# 0 -> unfurnished
# 1 -> semifurnished
# 2 -> furnished
df['furnishing_type'] = df['furnishing_type'].replace({0.0:'unfurnished',1.0:'semifurnished',2.0:'furnished'})
```



Define X & y:

```
X = df.drop(columns=['price'])
y = df['price']
```

Apply log transformation on y:

```
# Applying the log1p transformation to the target variable
y_transformed = np.log1p(y)
```

Ordinal Encoding

ColumnTransformer:

`from sklearn.` `compose` `import` `ColumnTransformer`

```
columns_to_encode = ['property_type','sector', 'balcony', 'agePossession', 'fur
nishing_type', 'luxury_category', 'floor_category']

# Creating a column transformer for preprocessing
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), ['bedRoom', 'bathroom', 'built_up_area', 'servan
t room', 'store room']),
        ('cat', OrdinalEncoder(), columns_to_encode)
    ],
    remainder='passthrough'
)
```

- Apply this transformation inside a sklearn pipeline.

```
# Creating a pipeline
pipeline = Pipeline([
    ('preprocessor', preprocessor),
```

```
    ('regressor', LinearRegression())
])
```

The reason the regressor (e.g., `LinearRegression`) is not applied inside the `ColumnTransformer` is because the `ColumnTransformer` is specifically designed for preprocessing (e.g., scaling, encoding, etc.), not for model training.

- K-fold CV

```
# K-fold cross-validation
kfold = KFold(n_splits=10, shuffle=True, random_state=42)
scores = cross_val_score(pipeline, X, y_transformed, cv=kfold, scoring='r2')
```

```
scores.mean(),scores.std()

Output: (0.7363096633436828, 0.03238005754429936)
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y_transformed,test_size=0.2,random_state=42)

pipeline.fit(X_train,y_train)

y_pred = pipeline.predict(X_test)

y_pred = np.expm1(y_pred)

mean_absolute_error(np.expm1(y_test),y_pred)

Output: 0.9463822160089356
```

- The average mistake is of **94 Lacs**

Convert this into a function

- It will take model name
- It will give you **R2** score & **MAE**

```
def scorer(model_name, model):

    output = []

    output.append(model_name)

    pipeline = Pipeline([
        ('preprocessor', preprocessor),
        ('regressor', model)
    ])

    # K-fold cross-validation
    kfold = KFold(n_splits=10, shuffle=True, random_state=42)
    scores = cross_val_score(pipeline, X, y_transformed, cv=kfold, scoring='r2')

    output.append(scores.mean())

    X_train, X_test, y_train, y_test = train_test_split(X,y_transformed,test_size=0.
2,random_state=42)

    pipeline.fit(X_train,y_train)

    y_pred = pipeline.predict(X_test)

    y_pred = np.expm1(y_pred)
```

```
    output.append(mean_absolute_error(np.expm1(y_test),y_pred))

    return output
```

```
# Creating a column transformer for preprocessing
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), ['bedRoom', 'bathroom', 'built_up_area', 'ser
vant room', 'store room']),
        ('cat', OrdinalEncoder(), columns_to_encode)
    ],
    remainder='passthrough'
    )
```

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, ExtraTreesRegressor,
GradientBoostingRegressor, AdaBoostRegressor
from sklearn.neural_network import MLPRegressor
from xgboost import XGBRegressor

model_dict = {
    'linear_reg':LinearRegression(),
    'svr':SVR(),
    'ridge':Ridge(),
    'LASSO':Lasso(),
    'decision tree': DecisionTreeRegressor(),
    'random forest':RandomForestRegressor(),
    'extra trees': ExtraTreesRegressor(),
    'gradient boosting': GradientBoostingRegressor(),
    'adaboost': AdaBoostRegressor(),
    'mlp': MLPRegressor(),
    'xgboost':XGBRegressor()
}
```

```
model_output = []
for model_name,model in model_dict.items():
    model_output.append(scorer(model_name, model))
```

- 👆Took 45 sec to run

```
model_df = pd.DataFrame(model_output, columns=['name','r2','mae'])

model_df.sort_values(['mae'])
```

	name	r2	mae
10	xgboost	0.889488	0.504048
5	random forest	0.880850	0.525509
6	extra trees	0.867689	0.554182
7	gradient boosting	0.872627	0.576293
9	mlp	0.802582	0.721199
4	decision tree	0.775987	0.724134
8	adaboost	0.758385	0.846483
1	svr	0.764201	0.847264
2	ridge	0.736313	0.946339
0	linear_reg	0.736310	0.946382
3	LASSO	0.059434	1.528906

Apply OneHotEncoder

- Performance of the linear models should increase after applying OHE

```
columns_to_encode = ['property_type','sector', 'balcony', 'agePossession', 'furnis

# Creating a column transformer for preprocessing
preprocessor = ColumnTransformer(
    transformers=[
```

```
        ('num', StandardScaler(), ['bedRoom', 'bathroom', 'built_up_area', 'servant rc
        ('cat', OrdinalEncoder(), columns_to_encode),
        ('cat1',OneHotEncoder(drop='first',sparse_output=False),['sector','agePosse
    ],
    remainder='passthrough'
)
```

- When `sparse_output=True`, the output is a **sparse matrix** (efficient for large datasets with many categories).
    - **stores only non-zero values, saving memory**
- When `sparse_output=False`, the output is a **dense array** (easier to work with for small datasets or when you need a regular NumPy array or DataFrame).

```
# Creating a pipeline
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('regressor', LinearRegression())
])
```

```
# K-fold cross-validation
kfold = KFold(n_splits=10, shuffle=True, random_state=42)
scores = cross_val_score(pipeline, X, y_transformed, cv=kfold, scoring='r2')
```

```
scores.mean(),scores.std()
```

```
(0.8546150908270336, 0.01599252910813069)
```

- The r2 score has improved

```
def scorer(model_name, model):

    output = []

    output.append(model_name)

    pipeline = Pipeline([
        ('preprocessor', preprocessor),
        ('regressor', model)
    ])

    # K-fold cross-validation
    kfold = KFold(n_splits=10, shuffle=True, random_state=42)
    scores = cross_val_score(pipeline, X, y_transformed, cv=kfold, scoring='r2')

    output.append(scores.mean())

    X_train, X_test, y_train, y_test = train_test_split(X,y_transformed,test_size=0.2,r

    pipeline.fit(X_train,y_train)

    y_pred = pipeline.predict(X_test)

    y_pred = np.expm1(y_pred)

    output.append(mean_absolute_error(np.expm1(y_test),y_pred))

    return output
```

```
model_dict = {
    'linear_reg':LinearRegression(),
    'svr':SVR(),
```

```
    'ridge':Ridge(),
    'LASSO':Lasso(),
    'decision tree': DecisionTreeRegressor(),
    'random forest':RandomForestRegressor(),
    'extra trees': ExtraTreesRegressor(),
    'gradient boosting': GradientBoostingRegressor(),
    'adaboost': AdaBoostRegressor(),
    'mlp': MLPRegressor(),
    'xgboost':XGBRegressor()
}
```

```
model_output = []
for model_name,model in model_dict.items():
    model_output.append(scorer(model_name, model))
```

- 👆**Took 1 min 37 sec min to run**

```
model_df = pd.DataFrame(model_output, columns=['name','r2','mae'])

model_df.sort_values(['mae'])
```

	name	r2	mae
5	random forest	0.900529	0.453631
6	extra trees	0.902299	0.457174
10	xgboost	0.900643	0.483409
7	gradient boosting	0.889074	0.509390
4	decision tree	0.830704	0.543814
9	mlp	0.852892	0.629715
8	adaboost	0.820024	0.681451
0	linear_reg	0.829522	0.713011
2	ridge	0.829536	0.713523
1	svr	0.782917	0.818851
3	LASSO	0.059434	1.528906

- Performance of linear model has improved

Target Encoding (Mean Encoding)

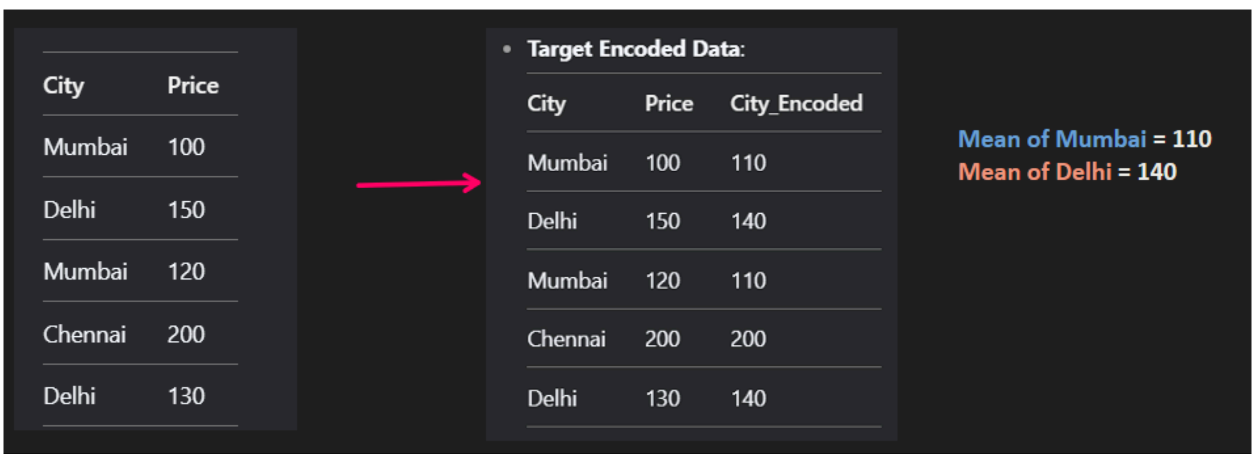
`import category_encoders as ce`

- The **Target Encoder** is a technique used in machine learning to encode categorical variables by replacing each category with the **mean of the target variable** for that category.
- It's particularly useful for handling **high-cardinality categorical features** (features with many unique categories) and capturing the relationship between the categorical feature and the target variable.

> 💡 **TargetEncoder gives good results for Tree based algos**

```
from category_encoders import TargetEncoder
import pandas as pd
```

```
# Sample data
data = pd.DataFrame({
    'City': ['Mumbai', 'Delhi', 'Mumbai', 'Chennai', 'Delhi'],
    'Price': [100, 150, 120, 200, 130]
})

# Initialize TargetEncoder
encoder = TargetEncoder()

# Fit and transform the data
data['City_Encoded'] = encoder.fit_transform(data['City'], data['Price'])

print(data)
```



City	Price
Mumbai	100
Delhi	150
Mumbai	120
Chennai	200
Delhi	130

- **Target Encoded Data:**

City	Price	City_Encoded
Mumbai	100	110
Delhi	150	140
Mumbai	120	110
Chennai	200	200
Delhi	130	140

First, you need to install the library `category_encoders`

```
!pip install category_encoders
```

```
import category_encoders as ce
```

```
columns_to_encode = ['property_type','sector', 'balcony', 'agePossession', 'furnis

# Creating a column transformer for preprocessing
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), ['bedRoom', 'bathroom', 'built_up_area', 'servant rc
        ('cat', OrdinalEncoder(), columns_to_encode),
        ('cat1',OneHotEncoder(drop='first',sparse_output=False),['agePossession'])
        ('target_enc', ce.TargetEncoder(), ['sector'])
    ],
    remainder='passthrough'
)
```

- We're doing OHE on `agePossession` column only.

```
# Creating a pipeline
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('regressor', LinearRegression())
])
```

```
# K-fold cross-validation
kfold = KFold(n_splits=10, shuffle=True, random_state=42)
scores = cross_val_score(pipeline, X, y_transformed, cv=kfold, scoring='r2')
```

```
scores.mean(),scores.std()
```

```
(0.8295219182255362, 0.018384463379122782)
```

```
def scorer(model_name, model):

    output = []

    output.append(model_name)

    pipeline = Pipeline([
        ('preprocessor', preprocessor),
        ('regressor', model)
    ])

    # K-fold cross-validation
    kfold = KFold(n_splits=10, shuffle=True, random_state=42)
    scores = cross_val_score(pipeline, X, y_transformed, cv=kfold, scoring='r2')

    output.append(scores.mean())

    X_train, X_test, y_train, y_test = train_test_split(X,y_transformed,test_size=0.2,r

    pipeline.fit(X_train,y_train)

    y_pred = pipeline.predict(X_test)

    y_pred = np.expm1(y_pred)

    output.append(mean_absolute_error(np.expm1(y_test),y_pred))

    return output
```

```
model_dict = {
    'linear_reg':LinearRegression(),
    'svr':SVR(),
    'ridge':Ridge(),
```

```
    'LASSO':Lasso(),
    'decision tree': DecisionTreeRegressor(),
    'random forest':RandomForestRegressor(),
    'extra trees': ExtraTreesRegressor(),
    'gradient boosting': GradientBoostingRegressor(),
    'adaboost': AdaBoostRegressor(),
    'mlp': MLPRegressor(),
    'xgboost':XGBRegressor()
}
```

```
model_output = []
for model_name,model in model_dict.items():
    model_output.append(scorer(model_name, model))
```

```
model_df = pd.DataFrame(model_output, columns=['name','r2','mae'])

model_df.sort_values(['mae'])
```

	name	r2	mae
10	xgboost	0.904798	0.447518
5	random forest	0.901350	0.453575
6	extra trees	0.902094	0.464719
7	gradient boosting	0.888918	0.508625
4	decision tree	0.828258	0.554059
9	mlp	0.851327	0.655331
8	adaboost	0.818569	0.689384
0	linear_reg	0.829522	0.713011
2	ridge	0.829536	0.713523
1	svr	0.782917	0.818851
3	LASSO	0.059434	1.528906

Hyperparameter Tuning

```
from sklearn.model_selection import GridSearchCV
```

```
param_grid = {
    'regressor__n_estimators': [50, 100, 200, 300],
    'regressor__max_depth': [None, 10, 20, 30],
    'regressor__max_samples':[0.1, 0.25, 0.5, 1.0],
    'regressor__max_features': ['auto', 'sqrt']
}
```

```
columns_to_encode = ['property_type','sector', 'balcony', 'agePossession', 'furnis

# Creating a column transformer for preprocessing
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), ['bedRoom', 'bathroom', 'built_up_area', 'servant ro
        ('cat', OrdinalEncoder(), columns_to_encode),
        ('cat1',OneHotEncoder(drop='first',sparse_output=False),['agePossession']),
        ('target_enc', ce.TargetEncoder(), ['sector'])
    ],
    remainder='passthrough'
)
```

```
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('regressor', RandomForestRegressor())
])

kfold = KFold(n_splits=10, shuffle=True, random_state=42)

search = GridSearchCV(pipeline, param_grid, cv=kfold, scoring='r2', n_jobs=-1, v
```

```
search.fit(X, y_transformed)
```

Final Pipeline:

```
final_pipe = search.best_estimator_
```

```
search.best_params_
```

```
{'regressor__max_depth': 20,
 'regressor__max_features': 'sqrt',
 'regressor__max_samples': 1.0,
 'regressor__n_estimators': 300}
```

```
search.best_score_
```

```
0.9025359727355461
```

- Tried with `XGBRegressor`
- Took 3x time compared to RF

Score is similar

```
0.9049944228673052
```

Train the data on final pipeline:

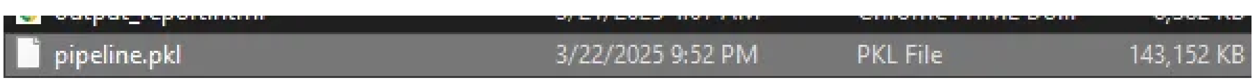
```
final_pipe.fit(X,y_transformed)
```

Exporting the model

```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), ['bedRoom', 'bathroom', 'built_up_area', 'servant rc
        ('cat', OrdinalEncoder(), columns_to_encode),
        ('cat1',OneHotEncoder(drop='first',sparse_output=False),['sector','agePosse
    ],
    remainder='passthrough'
)
```

```
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('regressor', RandomForestRegressor(n_estimators=500))
])

pipeline.fit(X,y_transformed)
```

```
import pickle

with open('pipeline.pkl', 'wb') as file:
    pickle.dump(pipeline, file)
```




```
with open('df.pkl', 'wb') as file:
    pickle.dump(X, file)
```