Amos House Price Predictions

ML workflow:

1. SetUp
2. Import Data
3. Explore(EDA)
4. Splitting
5. Modelling
6. Splitting

1. SetUp

We are going to import all the necesary libraries here.

In [1]:
```
import sys
import logging

import pickle
import pandas as pd
import numpy as np
import math

# creating path object
from pathlib import Path

# visualization
import matplotlib
import plotly
import matplotlib.pyplot as plt
import plotly.express as px
import seaborn as sns

# machine learning
import sklearn

from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor

# model selections eg.splitting
from sklearn.model_selection import train_test_split, learning_curve,GridSearchCV
from sklearn.pipeline import Pipeline

# evaluation metrics
from sklearn.metrics import mean_absolute_percentage_error, mean_absolute_error, r2_score
```

We have all the libraries in place. Let us print our library versions. This step ensures reproducability

In [2]:
```
# Printing version of our libraries
print("Platform: ", sys.platform)
print("Python: ", sys.version)
print("---")
print("Matplotlib: ", matplotlib.__version__)
print("Pandas: ", pd.__version__)
print("Seaborn: ", sns.__version__)
print("Plotly Express: ", plotly.__version__)
print("Numpy: ", np.__version__)
print("Sklearn: ", sklearn.__version__)
```

```
Platform:  win32
Python:  3.13.2 (tags/v3.13.2:4f8bb39, Feb  4 2025, 15:23:48) [MSC v.1942 64 bit (AMD64)]
---
Matplotlib:  3.10.0
Pandas:  2.2.3
Seaborn:  0.13.2
Plotly Express:  6.0.0
Numpy:  2.2.2
Sklearn:  1.6.1
```

Define the logging configurations.

In [3]:
```
# Configure
config_path = Path.cwd()/"Training"/"Configure"
config_path.mkdir(parents=True, exist_ok=True)
logging.basicConfig(
    level=logging.INFO,
    filename = config_path / "logging.log",
)
```

Let us not define matplotlib configurations.

In [4]:
```
#Matplotlib configuration
plt.rc("font", size=12)
plt.rc("axes", labelsize=12, titlesize=14)
plt.rc("legend", fontsize=8)
plt.rc("xtick", labelsize=10)
plt.rc("ytick", labelsize=10)
%matplotlib inline
```

Making two functions that will help us saving images and the other saving a trained model.

In [5]:
```
# save figure function
def save_plot(fname, filetype, fig=None, dpi=300, tight_layout=True, format="png"):
    """Saving the plot as an image
    Save a plot image within ``Train/Images`` folders. The fname will be the name of the image wi
    extension .(format).``Note`` the default format is png.
    The plot by default will be saved under 300 resolution as inches.

    Parameters:
    ----------
    fname: str
        ->String object for the name of the plot.
    filetype: str
        -> Plot type eg plt(matplotlib), or px(plotly.express)
    fig: px.Figure
        -> figure object form plotly.express
    dpi: int
        -> Numerical variable for the pixel resolution.
```

```python
    tight_layout: bool
        -> If true the plot will be save on a tight layout.
    format: str
        -> String object for the image extension. By default is 'png' but we can have: 'jpeg', ':

    Returns:
    --------
        None
    """
    # Root path
    image_path = Path.cwd() / "Training" / "Images"
    # Making the folders
    logging.info("Creting image path")
    image_path.mkdir(parents=True, exist_ok=True)
    # Image name
    image_name = image_path /f"{fname}.{format}"
    # Layout format
    if filetype == "plt":
        if tight_layout:
            plt.tight_layout()
        # Saving the plot
        logging.info(f"Saving the plot as {fname}.{format}")
        plt.savefig(fname=image_name, dpi=dpi, format=format)
        # logging saving
        logging.info(f"Sucess! Saved the plot as {fname}.{format}")
    elif filetype == "px":
        logging.info(f"Saving the plot as {fname}.{format}")
        # writting the image
        fig.write_image(file= image_name, format= format)
        logging.info(f"Sucess! Saved the plot as {fname}.{format}")
```

In [6]:

```python
# Saving the model
def save_model(mname, model):
    """Saving the model.
    Get the model and save it using pickle. The model will have the name from mname.

    Parameters:
    ----------
    mname: str
        Name of the model as a string object
    model: sklearn.model
        The trained model
    Returns:
    --------
    None
    """
    # Root path
    model_path = Path.cwd() / "Training" / "Models"
    # Making the folders
    logging.info("Creating Model path")
    model_path.mkdir(parents=True, exist_ok=True)
    # Model name
    model_name = model_path / f"{mname}.pkl"
    logging.info(f"Saving the model as {mname}.pkl")
    # Creating the pickle file
    with open(model_name, "wb") as f:
        pickle.dump(model, f)
    # final log
    logging.info(f"Sucess! Saved the model as {mname}.pkl")
```

We have a solid setup sections, let us start data importation.

2. Import and EDA

I have cleated a Training module where I have all my classes. We are going to get the `wrangleRepository` class that does the following:

1. Get the data from the csv file
2. Do a basic cleaning
3. Feature selection
4. Feature engineering
5. Outlier removing

In [7]:
```
sub_class = {
        20: "1-STORY 1946 & NEWER ALL STYLES",
        30: "1-STORY 1945 & OLDER",
        40:     "1-STORY W/FINISHED ATTIC ALL AGES",
        45:     "1-1/2 STORY - UNFINISHED ALL AGES",
        50:     "1-1/2 STORY FINISHED ALL AGES",
        60:     "2-STORY 1946 & NEWER",
        70:     "2-STORY 1945 & OLDER",
        75:     "2-1/2 STORY ALL AGES",
        80:     "SPLIT OR MULTI-LEVEL",
        85:     "SPLIT FOYER",
        90:     "DUPLEX - ALL STYLES AND AGES",
       120:     "1-STORY PUD (Planned Unit Development) - 1946 & NEWER",
       150:     "1-1/2 STORY PUD - ALL AGES",
       160:     "2-STORY PUD - 1946 & NEWER",
       180:     "PUD - MULTILEVEL - INCL SPLIT LEV/FOYER",
       190:     "2 FAMILY CONVERSION - ALL STYLES AND AGES"

}

print(type(sub_class))
print(sub_class)
```

```
<class 'dict'>
{20: '1-STORY 1946 & NEWER ALL STYLES', 30: '1-STORY 1945 & OLDER', 40: '1-STORY W/FINISHED ATTIC
ALL AGES', 45: '1-1/2 STORY - UNFINISHED ALL AGES', 50: '1-1/2 STORY FINISHED ALL AGES', 60: '2-S
TORY 1946 & NEWER', 70: '2-STORY 1945 & OLDER', 75: '2-1/2 STORY ALL AGES', 80: 'SPLIT OR MULTI-L
EVEL', 85: 'SPLIT FOYER', 90: 'DUPLEX - ALL STYLES AND AGES', 120: '1-STORY PUD (Planned Unit Dev
elopment) - 1946 & NEWER', 150: '1-1/2 STORY PUD - ALL AGES', 160: '2-STORY PUD - 1946 & NEWER',
180: 'PUD - MULTILEVEL - INCL SPLIT LEV/FOYER', 190: '2 FAMILY CONVERSION - ALL STYLES AND AGES'}
```

In [8]:
```
from Training import WrangleRepository

# instantiating the class
repo = WrangleRepository(sub_class = sub_class)
print(type(repo))
repo
```

```
<class 'Training.WrangleRepository'>
```

Out[8]:
```
WrangleRepository filepath=C:\Users\MY PC\Desktop\Projects\Regression\AmosHousePriceModelling\tr
ain.csv
```

Let us now us the function to lead the data in `Desktop/Projects/Regression/AmosHousePriceModelling/train.csv`

In [9]:
```
repo.wrangle()
df = repo.get_data("wrangled")
print(df.shape)
df.head()
```

```
(1460, 80)
```

Out[9]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotCo
Id										
1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	In
2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	
3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	In
4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	Co
5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	

5 rows × 80 columns

We have our data successfully. We will start by doing basic data cleaning.

In [10]:
```
df.describe().T
```

Out[10]:

	count	mean	std	min	25%	50%	75%	max
MSSubClass	1460.0	56.897260	42.300571	20.0	20.00	50.0	70.00	190.0
LotFrontage	1201.0	70.049958	24.284752	21.0	59.00	69.0	80.00	313.0
LotArea	1460.0	10516.828082	9981.264932	1300.0	7553.50	9478.5	11601.50	215245.0
OverallQual	1460.0	6.099315	1.382997	1.0	5.00	6.0	7.00	10.0
OverallCond	1460.0	5.575342	1.112799	1.0	5.00	5.0	6.00	9.0
YearBuilt	1460.0	1971.267808	30.202904	1872.0	1954.00	1973.0	2000.00	2010.0
YearRemodAdd	1460.0	1984.865753	20.645407	1950.0	1967.00	1994.0	2004.00	2010.0
MasVnrArea	1452.0	103.685262	181.066207	0.0	0.00	0.0	166.00	1600.0
BsmtFinSF1	1460.0	443.639726	456.098091	0.0	0.00	383.5	712.25	5644.0
BsmtFinSF2	1460.0	46.549315	161.319273	0.0	0.00	0.0	0.00	1474.0
BsmtUnfSF	1460.0	567.240411	441.866955	0.0	223.00	477.5	808.00	2336.0
TotalBsmtSF	1460.0	1057.429452	438.705324	0.0	795.75	991.5	1298.25	6110.0
1stFlrSF	1460.0	1162.626712	386.587738	334.0	882.00	1087.0	1391.25	4692.0
2ndFlrSF	1460.0	346.992466	436.528436	0.0	0.00	0.0	728.00	2065.0
LowQualFinSF	1460.0	5.844521	48.623081	0.0	0.00	0.0	0.00	572.0
GrLivArea	1460.0	1515.463699	525.480383	334.0	1129.50	1464.0	1776.75	5642.0
BsmtFullBath	1460.0	0.425342	0.518911	0.0	0.00	0.0	1.00	3.0
BsmtHalfBath	1460.0	0.057534	0.238753	0.0	0.00	0.0	0.00	2.0
FullBath	1460.0	1.565068	0.550916	0.0	1.00	2.0	2.00	3.0
HalfBath	1460.0	0.382877	0.502885	0.0	0.00	0.0	1.00	2.0
BedroomAbvGr	1460.0	2.866438	0.815778	0.0	2.00	3.0	3.00	8.0
KitchenAbvGr	1460.0	1.046575	0.220338	0.0	1.00	1.0	1.00	3.0
TotRmsAbvGrd	1460.0	6.517808	1.625393	2.0	5.00	6.0	7.00	14.0
Fireplaces	1460.0	0.613014	0.644666	0.0	0.00	1.0	1.00	3.0
GarageYrBlt	1379.0	1978.506164	24.689725	1900.0	1961.00	1980.0	2002.00	2010.0
GarageCars	1460.0	1.767123	0.747315	0.0	1.00	2.0	2.00	4.0
GarageArea	1460.0	472.980137	213.804841	0.0	334.50	480.0	576.00	1418.0
WoodDeckSF	1460.0	94.244521	125.338794	0.0	0.00	0.0	168.00	857.0
OpenPorchSF	1460.0	46.660274	66.256028	0.0	0.00	25.0	68.00	547.0
EnclosedPorch	1460.0	21.954110	61.119149	0.0	0.00	0.0	0.00	552.0
3SsnPorch	1460.0	3.409589	29.317331	0.0	0.00	0.0	0.00	508.0
ScreenPorch	1460.0	15.060959	55.757415	0.0	0.00	0.0	0.00	480.0
PoolArea	1460.0	2.758904	40.177307	0.0	0.00	0.0	0.00	738.0
MiscVal	1460.0	43.489041	496.123024	0.0	0.00	0.0	0.00	15500.0

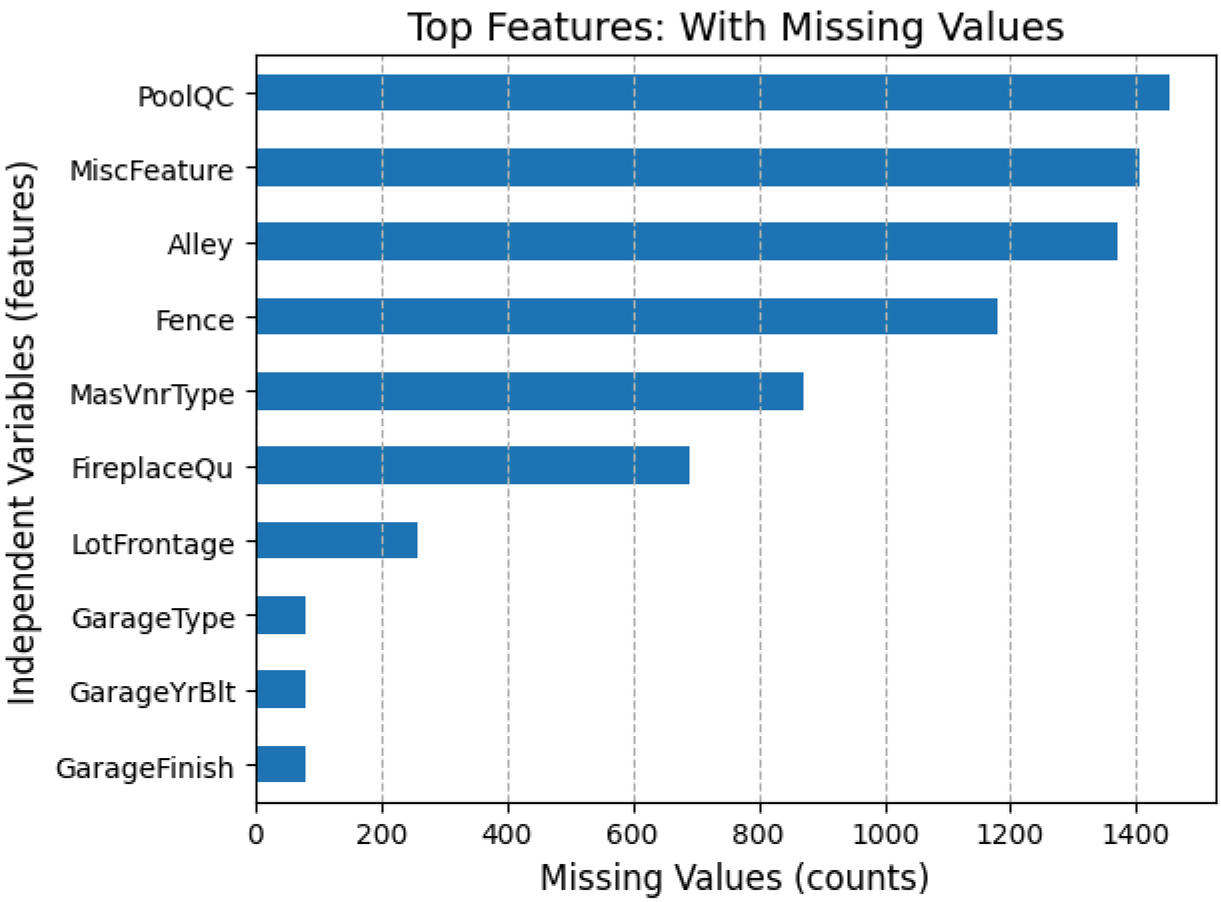
	count	mean	std	min	25%	50%	75%	max
MoSold	1460.0	6.321918	2.703626	1.0	5.00	6.0	8.00	12.0
YrSold	1460.0	2007.815753	1.328095	2006.0	2007.00	2008.0	2009.00	2010.0
SalePrice	1460.0	180921.195890	79442.502883	34900.0	129975.00	163000.0	214000.00	755000.0

As seen, we have 1460 number of houses. Some of the features have missing values and we need to work on that. Also it is important to note that some features have outliers for example `lotArea`. This is determined from the sense that, we have a mean of approximately 10,500 and std of 9900 with the maximum value being at 255,000. Clearly we have outlier and most likely our data will be skewed.

In [11]:
```
# checking for missing values
missing_values = (df.isnull().sum()[df.isnull().sum() > 1]).sort_values()
print(f"Missing values in our Features: \n{missing_values}")
```

```
Missing values in our Features:
MasVnrArea         8
BsmtQual          37
BsmtCond          37
BsmtFinType1      37
BsmtFinType2      38
BsmtExposure      38
GarageCond        81
GarageQual        81
GarageFinish      81
GarageYrBlt       81
GarageType        81
LotFrontage      259
FireplaceQu      690
MasVnrType       872
Fence           1179
Alley           1369
MiscFeature     1406
PoolQC          1453
dtype: int64
```

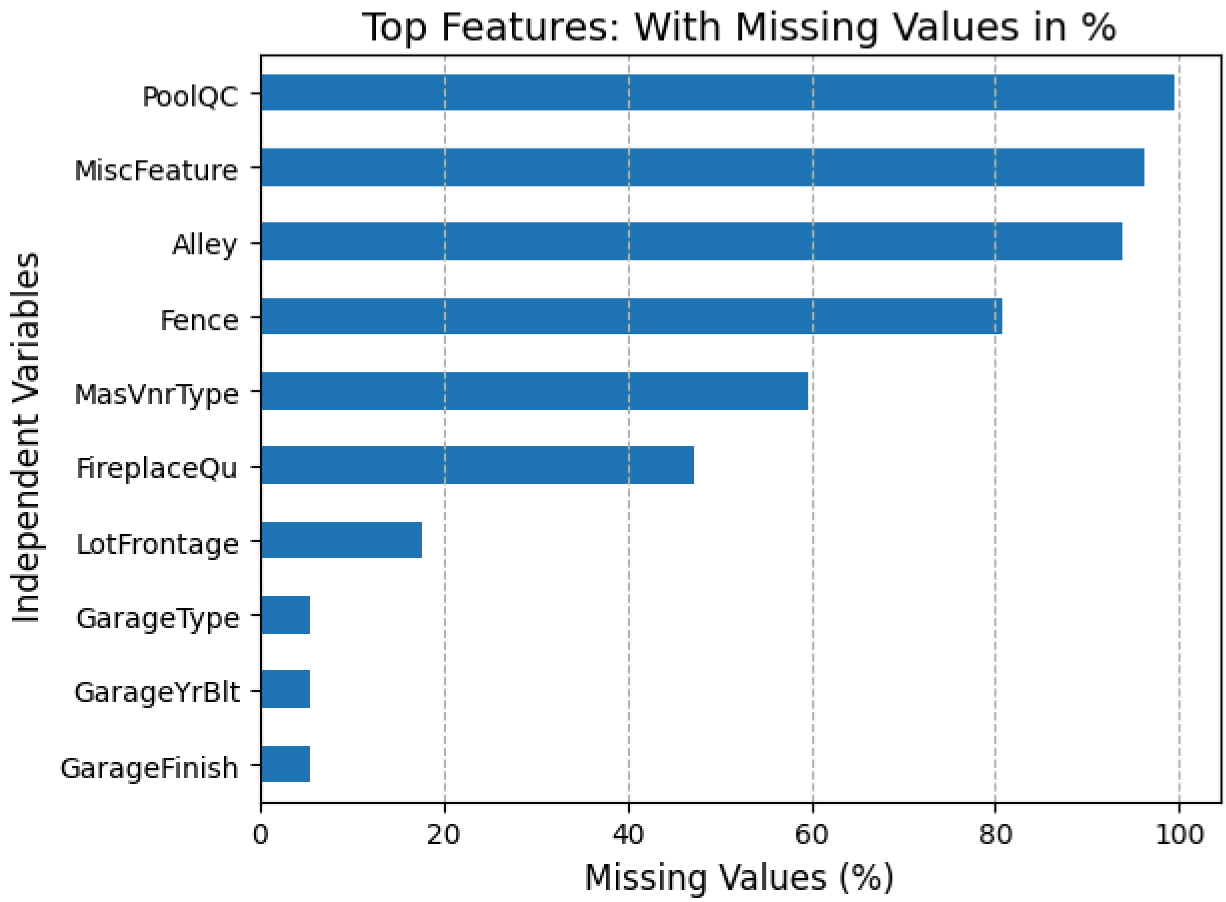
In [12]:
```
# plotting the top 10 most missing values
missing_values.tail(10).plot(kind="barh")
plt.xlabel("Missing Values (counts)")
plt.ylabel("Independent Variables (features)")
plt.title("Top Features: With Missing Values")
plt.grid(linestyle="--", axis="x")
save_plot(fname="top10_missing_values", filetype="plt")
plt.show()
```



In [13]:
```
# percentage counts
missing_values_pct = pd.Series(((100 * missing_values.values / len(df))),
                               index=missing_values.index, name="missing_pct")
missing_values_pct
```

Out[13]:
```
MasVnrArea       0.547945
BsmtQual         2.534247
BsmtCond         2.534247
BsmtFinType1     2.534247
BsmtFinType2     2.602740
BsmtExposure     2.602740
GarageCond       5.547945
GarageQual       5.547945
GarageFinish     5.547945
GarageYrBlt      5.547945
GarageType       5.547945
LotFrontage     17.739726
FireplaceQu     47.260274
MasVnrType      59.726027
Fence           80.753425
Alley           93.767123
MiscFeature     96.301370
PoolQC          99.520548
Name: missing_pct, dtype: float64
```

In [14]:
```
# plotting the top 10 pct features with missing values
missing_values_pct.tail(10).plot(kind="barh")
plt.xlabel("Missing Values (%)")
plt.ylabel("Independent Variables ")
plt.title("Top Features: With Missing Values in %")
plt.grid(linestyle="--", axis="x")
save_plot(fname="top10_missing_values_pct", filetype="plt")
plt.show()
```



The above plot tells us that some features like pool are not present in many properties. We need to remove those features which have many missing values. To do that we will use VarianceThreshold object to compute those features which will be useless to our model.

Some advantes of removing low variance features are:

1. Reduce features dimensionality
2. Improve model performance by reducing overfitting
3. Reduce training time

But before we do Variance reduction we are going to drop those feature with over 50% missing values.

In [15]:
```
# getting columns with over 60% missing values
mask = missing_values_pct > 50
missing_cols = missing_values_pct[mask].index.to_list()
print(type(missing_cols))
print(missing_cols)
```

```
<class 'list'>
['MasVnrType', 'Fence', 'Alley', 'MiscFeature', 'PoolQC']
```

In [16]:
```
repo.basic_cleaning()
df = repo.get_data("basic")
print(df.shape)
df.head()
```

```
(1460, 75)
```

Out[16]:

Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	L
1	60	RL	65.0	8450	Pave	Reg	Lvl	AllPub	Inside	
2	20	RL	80.0	9600	Pave	Reg	Lvl	AllPub	FR2	
3	60	RL	68.0	11250	Pave	IR1	Lvl	AllPub	Inside	
4	70	RL	60.0	9550	Pave	IR1	Lvl	AllPub	Corner	
5	60	RL	84.0	14260	Pave	IR1	Lvl	AllPub	FR2	

5 rows × 75 columns

Done, we have dropped those features with so many missing values. Let us go ahead and check the variance for all the numerical features first.

In [17]:

```
repo.feature_selection()
df = repo.get_data("selected")
print(df.shape)
df.head()
```

```
(1460, 67)
```

Out[17]:

Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea
1	60	65.0	8450	7	5	2003	2003	196.0
2	20	80.0	9600	6	8	1976	1976	0.0
3	60	68.0	11250	7	5	2001	2002	162.0
4	70	60.0	9550	7	5	1915	1970	0.0
5	60	84.0	14260	8	5	2000	2000	350.0

5 rows × 67 columns

Let us check low and high cardinal features.

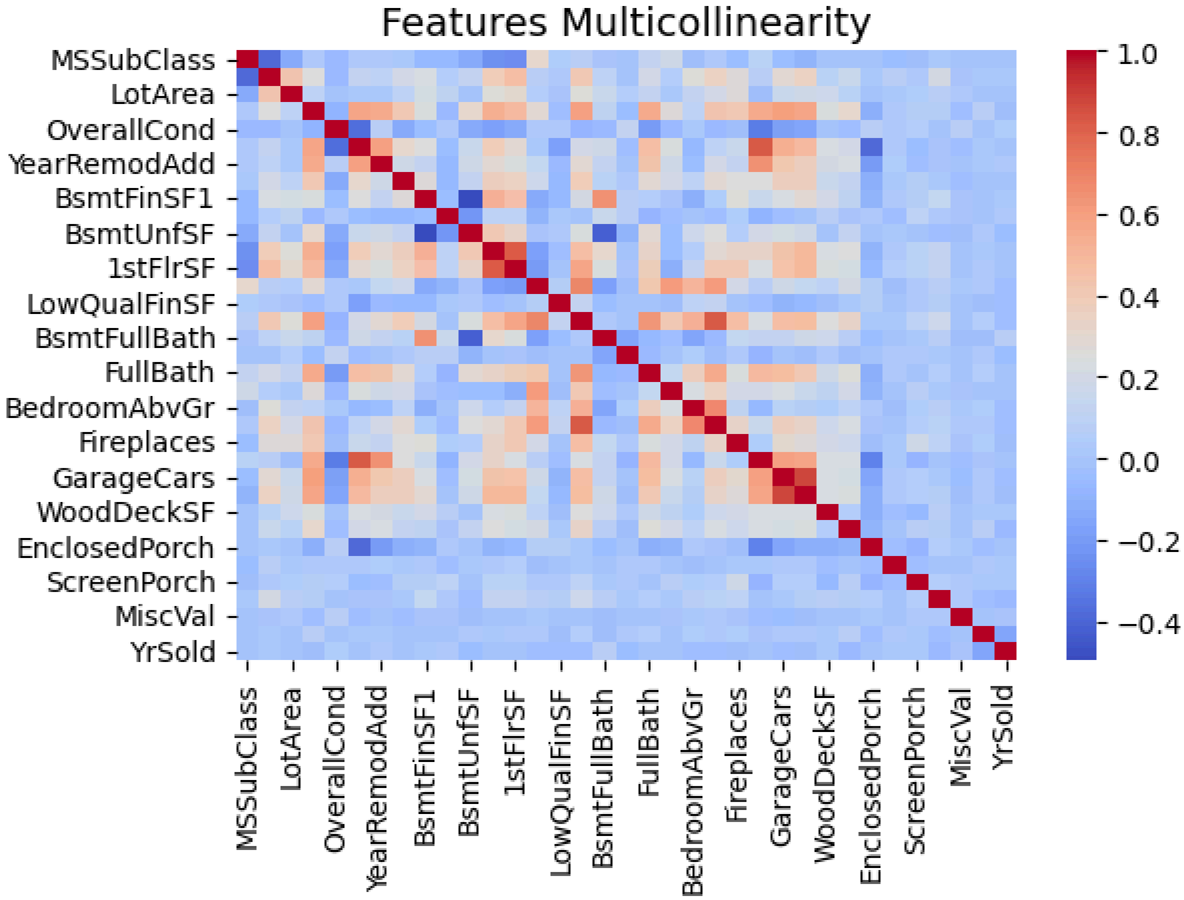
In [18]:

```
df.select_dtypes(include="object").nunique()
```

Out[18]:
```
MSZoning          5
LotShape          4
LandContour       4
LotConfig         5
LandSlope         3
Neighborhood     25
Condition1        9
BldgType          5
HouseStyle        8
RoofStyle         6
Exterior1st      15
Exterior2nd      16
ExterQual         4
ExterCond         5
Foundation        6
BsmtQual          4
BsmtCond          4
BsmtExposure      4
BsmtFinType1      6
BsmtFinType2      6
HeatingQC         5
CentralAir        2
Electrical        5
KitchenQual       4
Functional        7
FireplaceQu       5
GarageType        6
GarageFinish      3
PavedDrive        3
SaleType          9
SaleCondition     6
dtype: int64
```

I think there are no cardinal features.Let us check how our features are correlated with one another.

In [19]:
```
corr = df.select_dtypes(include="number").drop(columns="SalePrice").corr()
sns.heatmap(corr, cmap="coolwarm")
plt.title("Features Multicollinearity")
save_plot(fname="mulitcollinearity", filetype="plt")
```



In [20]:

```
# which features have a correlation above 90%
corr_matrix = abs(df.select_dtypes(include="number").drop(columns="SalePrice").corr())
upper_triangle = np.triu(np.ones(corr_matrix.shape), k=1).astype(bool)
upper_matrix = corr_matrix.where(upper_triangle)
upper_matrix
```

Out[20]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd
MSSubClass	NaN	0.386347	0.139781	0.032628	0.059316	0.027850	0.040581
LotFrontage	NaN	NaN	0.426095	0.251646	0.059213	0.123349	0.088866
LotArea	NaN	NaN	NaN	0.105806	0.005636	0.014228	0.013788
OverallQual	NaN	NaN	NaN	NaN	0.091932	0.572323	0.550684
OverallCond	NaN	NaN	NaN	NaN	NaN	0.375983	0.073741
YearBuilt	NaN	NaN	NaN	NaN	NaN	NaN	0.592855
YearRemodAdd	NaN	NaN	NaN	NaN	NaN	NaN	NaN
MasVnrArea	NaN	NaN	NaN	NaN	NaN	NaN	NaN
BsmtFinSF1	NaN	NaN	NaN	NaN	NaN	NaN	NaN
BsmtFinSF2	NaN	NaN	NaN	NaN	NaN	NaN	NaN
BsmtUnfSF	NaN	NaN	NaN	NaN	NaN	NaN	NaN
TotalBsmtSF	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1stFlrSF	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2ndFlrSF	NaN	NaN	NaN	NaN	NaN	NaN	NaN
LowQualFinSF	NaN	NaN	NaN	NaN	NaN	NaN	NaN
GrLivArea	NaN	NaN	NaN	NaN	NaN	NaN	NaN
BsmtFullBath	NaN	NaN	NaN	NaN	NaN	NaN	NaN
BsmtHalfBath	NaN	NaN	NaN	NaN	NaN	NaN	NaN
FullBath	NaN	NaN	NaN	NaN	NaN	NaN	NaN
HalfBath	NaN	NaN	NaN	NaN	NaN	NaN	NaN
BedroomAbvGr	NaN	NaN	NaN	NaN	NaN	NaN	NaN
TotRmsAbvGrd	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Fireplaces	NaN	NaN	NaN	NaN	NaN	NaN	NaN
GarageYrBlt	NaN	NaN	NaN	NaN	NaN	NaN	NaN
GarageCars	NaN	NaN	NaN	NaN	NaN	NaN	NaN
GarageArea	NaN	NaN	NaN	NaN	NaN	NaN	NaN
WoodDeckSF	NaN	NaN	NaN	NaN	NaN	NaN	NaN
OpenPorchSF	NaN	NaN	NaN	NaN	NaN	NaN	NaN
EnclosedPorch	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3SsnPorch	NaN	NaN	NaN	NaN	NaN	NaN	NaN
ScreenPorch	NaN	NaN	NaN	NaN	NaN	NaN	NaN
PoolArea	NaN	NaN	NaN	NaN	NaN	NaN	NaN
MiscVal	NaN	NaN	NaN	NaN	NaN	NaN	NaN
MoSold	NaN	NaN	NaN	NaN	NaN	NaN	NaN

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd
YrSold	NaN	NaN	NaN	NaN	NaN	NaN	NaN

35 rows × 35 columns

In [21]:
```
high_corr = [col for col in upper_matrix.columns
            if any(upper_matrix[col] > 0.9)
            ]
high_corr
```

Out[21]: []

So we do have highly correlated features. Let us go ahead now and do feature engineering.

In [22]:
```
repo.feature_engineering()
df = repo.get_data("engineered")
print(df.shape)
df.head()
```

```
(1460, 73)
```

Out[22]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea
Id								
1	2-STORY 1946 & NEWER	65.0	8450	7	5	2003	2003	196.0
2	1-STORY 1946 & NEWER ALL STYLES	80.0	9600	6	8	1976	1976	0.0
3	2-STORY 1946 & NEWER	68.0	11250	7	5	2001	2002	162.0
4	2-STORY 1945 & OLDER	60.0	9550	7	5	1915	1970	0.0
5	2-STORY 1946 & NEWER	84.0	14260	8	5	2000	2000	350.0

5 rows × 73 columns

Note we still have missing values and outliers. In the case of missig value, we will impute them inside a pipeline.

Now let us first check the distributions of our features. We will do this step manually step by step.In We will be refrencing the describe objects for our dataframe.

Features with most outliers:

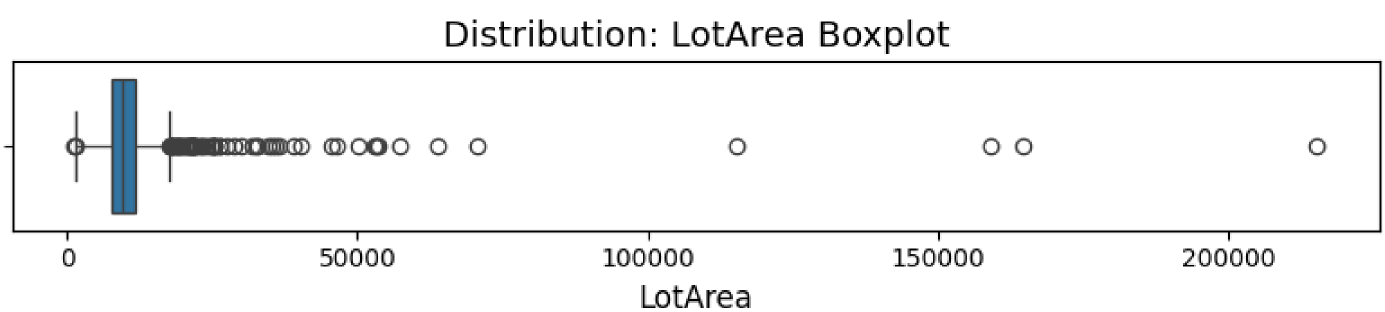
1. LotArea:

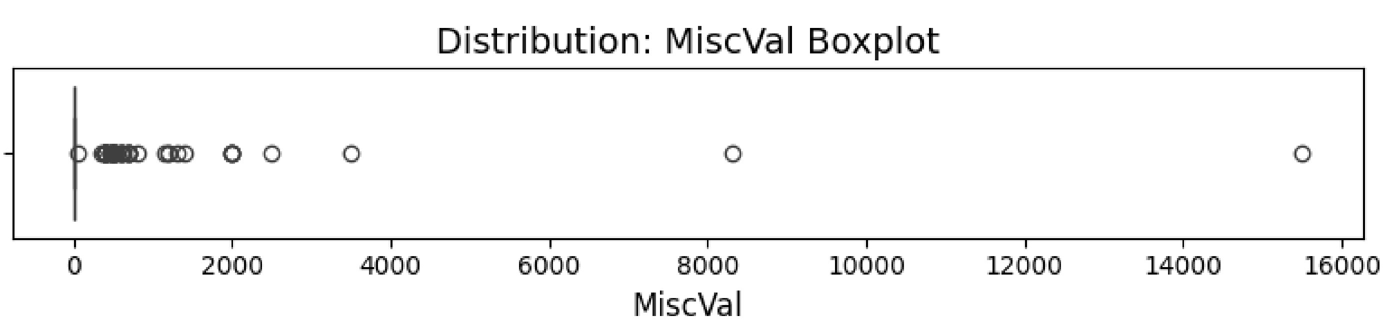
- Mean: 10,500
- std: 9980
- max: 215,000
- Loss: $\frac{215,000}{10,500} = 20.5$

2. LowQualFinSF:
   - Mean: 6
   - std: 48.6
   - max: 572
   - Loss: $\frac{572}{49} = 11.6$
3. MiscVal:
   - Mean: 43.5
   - std: 496
   - Max: 15,500
   - Loss: $\frac{15,500}{496} = 31.6$

We will look into those three features and see those outliears and remove them.

In [23]:
```python
# columns to check/huge outliers
cols = ["LotArea", "LowQualFinSF", "MiscVal"]
for col in cols:
    plt.figure(figsize=(8, 2))
    sns.boxplot(x=df[col])
    plt.title(f"Distribution: {col} Boxplot")
    save_plot(fname=f"Distribution_outlier_{col}", filetype="plt")
    plt.show()
```





Indeed we have outliers in our colums and we need to remove them. We will remove outlier in the upper quantile(90%)

In [24]:
```
# removed outlier
repo.remove_outliers()
df = repo.get_data()
print(df.shape)
df.head()
```

```
(1170, 73)
```

Out[24]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea
Id								
1	2-STORY 1946 & NEWER	65.0	8450	7	5	2003	2003	196.0
2	1-STORY 1946 & NEWER ALL STYLES	80.0	9600	6	8	1976	1976	0.0
3	2-STORY 1946 & NEWER	68.0	11250	7	5	2001	2002	162.0
5	2-STORY 1946 & NEWER	84.0	14260	8	5	2000	2000	350.0
6	1-1/2 STORY FINISHED ALL AGES	85.0	14115	5	5	1993	1995	0.0

5 rows × 73 columns

4. Splitting

Now that we have our class well defined and we also have the most current data, the next thing we want to do is split our data into train and validation set.

In [25]:
```
# Vertical split
target = "SalePrice"
X = df.drop(columns=target)
y = df[target]
print(f"X shape: {X.shape}")
print(f"y shape: {y.shape}")
```

```
X shape: (1170, 72)
y shape: (1170,)
```

In [26]:
```
# Horizontal split
X_train, X_val, y_train, y_val = train_test_split(
    X, y, test_size=0.15, random_state=42
)
print(f"X_train shape: {X_train.shape}")
```

```python
print(f"y_train shape: {y_train.shape}")
print(f"X_val shape: {X_val.shape}")
print(f"y_val shape: {y_val.shape}")
```

```
X_train shape: (994, 72)
y_train shape: (994,)
X_val shape: (176, 72)
y_val shape: (176,)
```

5. Pipeline

In [27]:
```python
from Training import MakePipeline
```

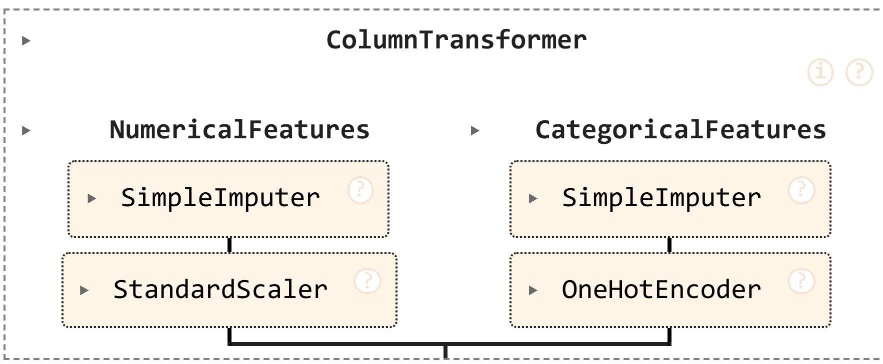
In [28]:
```python
pipe = MakePipeline(X_train)
print(type(pipe))
pipe
```

```
<class 'Training.MakePipeline'>
```

Out[28]:
```
Pipeline Stage presenter:
```

In [29]:
```python
col_pipeline = pipe.make_column_pipeline()
col_pipeline
```

Out[29]:



6. PCA Decomposition

In order to visualize our feature matrix (independent variables) against dependent variable `SalePrice`, we need to reduce the number of dimensions of the feature matrix. In our case we want just one dimension.

We will continue to build the pipeline and created a pca_pipeline that we will then fit and transform into a single vector, of 1-dimension matrix.

In [30]:
```python
# building the pipeline
pca_pipeline = pipe.make_pca_pipeline()
# fitting and transforming
X_t = pca_pipeline.fit_transform(X_train)
print("Type of X_t: ", type(X_t))
print("Total Number of items: ", len(X_t))
print("Number of dimensions: ", X_t.ndim)
print("Transformed X_train: \n", X_t[:4])
```

```
Type of X_t:  <class 'numpy.ndarray'>
Total Number of items:  994
Number of dimensions:  2
Transformed X_train:
 [[ 2.55320129]
 [ 1.64374143]
 [-2.32225458]
 [ 2.90775581]]
```

Now that we have that, the next thing we want to do is making a function that will help us make the image instantly.

In [31]:
```python
def scatter_plot(x, y, y_pred, label):
    """
    Make a scatter plot comparing actual vs. predicted values.

    Parameters:
        x: array-like
            Independent variable (e.g., PCA-transformed features)
        y: array-like
            Actual target values (e.g., true Sale Prices)
        y_pred: array-like
            Predicted target values
        label: str
            Label for the plot title
    """
    df = pd.DataFrame({
        "x": x.ravel(),
        "y": y,
        "y_pred": y_pred
    })

    df_melt = pd.melt(
        frame=df,
        id_vars="x",
        value_vars=["y", "y_pred"],
        var_name="Set",
        value_name="Sale Price"
    )

    fig = px.scatter(
        data_frame=df_melt,
        x="x",
        y="Sale Price",
        color="Set",
        title=f"{label} Scatter Plot: Decomposed Features vs. Sale Price"
    )

    fig.update_layout(
        xaxis_title="Decomposed Feature(s)",
        yaxis_title="Sale Price ($)",
        legend_title="Plot Type",
        template = "plotly_white"
    )

    # return
    return fig
```

In [32]:
```python
fig = scatter_plot(X_t, y_train,y_pred=None, label="")
save_plot(fname="DecomposedScatter", filetype="plt")
```

```
fig.show()
```

```
<Figure size 640x480 with 0 Axes>
```

We have our beautiful scatter plot and it seems like our features follow a polynomial kind of a function. We should keep this in mind. But for now let us have our baseline model.

7. Baseline and Linear Regression Model

In [33]:
```
y_mean = y_train.mean()
baseline_model = len(y_train) * [y_mean]
print("Mean Sale Price: ", int(y_mean))
```

```
Mean Sale Price:  176582
```

We have our baseline model, let us evaluate it's performance using mean absolute error which is computed as

MAE = $\frac{1}{n}\sum_i^n (y_i - \hat y_i)^2$

MAPE = $\frac{100}{n}\sum_i^n \frac{(y_i - \hat y_i)^2}{y_i}$

And finally computing coefficient of determination r2 score as: $R^2 = 1 - \frac{\sum_i^n y_i - \hat y}{\sum_i^n y_i - \bar y}$

Note:

3. $\bar y_i$ - Mean
4. n - Number of samples|

In [34]:
```
mae = mean_absolute_error(y_train, baseline_model)
mape = mean_absolute_percentage_error(y_train, baseline_model)
cod = r2_score(y_train, baseline_model)
print(f"MAE: ${np.round(mae, 2)}")
print(f"MAPE: {100 * np.round(mape, 2)}%")
print(f"R2: {100 * np.round(cod, 2)}%")
```

```
MAE: $51346.51
MAPE: 32.0%
R2: 0.0%
```

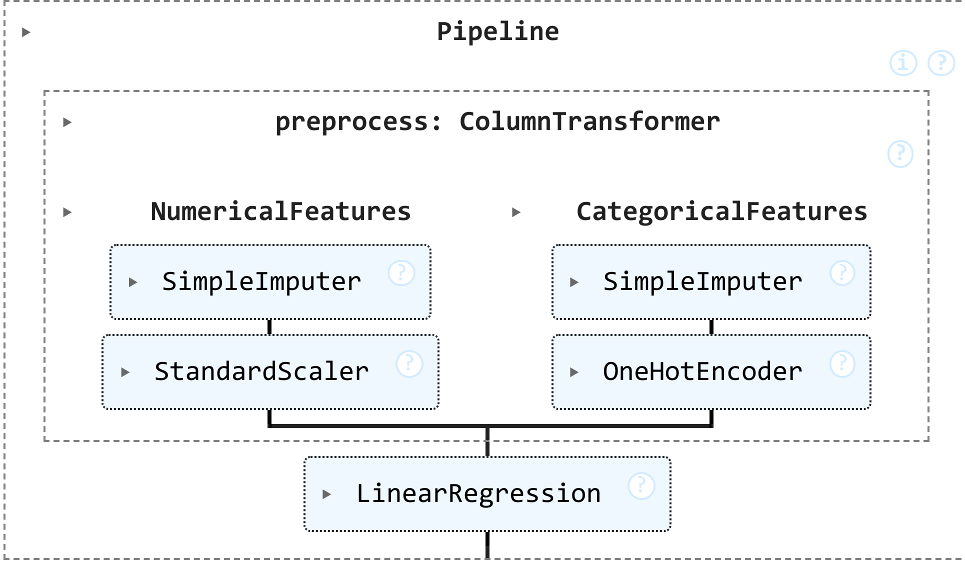
In [35]:
```
fig = scatter_plot(X_t, y_train,y_pred=baseline_model, label="Baseline Model")
save_plot(fname="BaselineModelScatter", filetype="plt")
fig.show()
```

```
<Figure size 640x480 with 0 Axes>
```

We are off by 36% from the actual values. This is a reasonable range but there is room for imporvement. We will train a linear regression model first.

In [36]:
```
# linear regression
linear_model = pipe.make_linear_pipeline()
# Training the model
linear_model.fit(X_train, y_train)
```

Out[36]:



In [37]:
```python
# model evaluation using mae, mape, coefficient of difference(cod, R2)
pred = linear_model.predict(X_train)
# With training data
mae = mean_absolute_error(y_train, pred)
mape = mean_absolute_percentage_error(y_train, pred)
cod = r2_score(y_train, pred)
print(f"Training MAE: ${np.round(mae, 2)}")
print(f"Training MAPE: {100 * mape:.3f}%")
print(f"Training R2: {100 * np.round(cod, 2)}%")
```

```
Training MAE: $11914.33
Training MAPE: 7.123%
Training R2: 94.0%
```

We are of by 7% with our true values. Next let us evaluate our model with the validation dataset. We will check if there is any form of overfitting that need to be addressed.

In [38]:
```python
# model evaluation using mae, mape, coefficient of difference(cod, R2)
pred = linear_model.predict(X_val)
# With training data
mae = mean_absolute_error(y_val, pred)
mape = mean_absolute_percentage_error(y_val, pred)
cod = r2_score(y_val, pred)
print(f"Validation MAE: ${np.round(mae, 2)}")
print(f"Validation MAPE: {100 * np.round(mape, 2)}%")
print(f"Validation R2: {100 * np.round(cod, 2)}%")
```

```
Validation MAE: $13611.39
Validation MAPE: 9.0%
Validation R2: 91.0%
```

In [39]:
```python
from Training import LearningCurve


lc = LearningCurve(estimator=linear_model, X=X, y=y)
print(type(lc))
lc
```

```
<class 'Training.LearningCurve'>
```

Out[39]:
```
LearningCurve: <class 'sklearn.pipeline.Pipeline'>
```

In [40]:
```python
# Getting learning curve list
lc.learning_curve()
```

```python
data = lc.get_data("lc")
print(type(data))
data
```

```
[learning_curve] Training set sizes: [ 93 304 514 725 936]
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:   13.7s finished
<class 'list'>
```

Out[40]:

```
[array([ 93, 304, 514, 725, 936]),
 array([1.        , 0.97686676, 0.95751753, 0.94935131, 0.9460286 ]),
 array([0.43860028, 0.71470291, 0.85816672, 0.87820229, 0.87343716])]
```

In [41]:

```python
lc.plot_lc()
```

```
[learning_curve] Training set sizes: [ 93 304 514 725 936]
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:    2.2s finished
```

In [42]:

```python
# Making the dataframe
lc.make_dataframe()
df_lc = lc.get_data("df_lc")
print(type(df_lc))
df_lc.head()
```

```
[learning_curve] Training set sizes: [ 93 304 514 725 936]
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:    6.2s finished
<class 'pandas.core.frame.DataFrame'>
```

Out[42]:

	Train Size	Train R2	Validation R2
0	93	1.000000	0.438600
1	304	0.976867	0.714703
2	514	0.957518	0.858167
3	725	0.949351	0.878202
4	936	0.946029	0.873437

In [43]:

```
# melt our datframe
lc.melt_dataframe()
df_melt = lc.get_data("melt_lc")
print(type(df_melt))
df_melt.head()
```

```
[learning_curve] Training set sizes: [ 93 304 514 725 936]
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:    7.2s finished
<class 'pandas.core.frame.DataFrame'>
```

Out[43]:

	Train Size	Set	R2
0	93	Train R2	1.000000
1	304	Train R2	0.976867
2	514	Train R2	0.957518
3	725	Train R2	0.949351
4	936	Train R2	0.946029

In [44]:

```
# plotting the figure
lc.plot_lc()
fig = lc.get_data()
fig.show()
```

```
[learning_curve] Training set sizes: [ 93 304 514 725 936]
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:    2.5s finished
```

With the above infomation we might want to build a class called LearningCurve that will have all that information.

That is good information. Next thing we want to do is plot the scatter plot and seen.

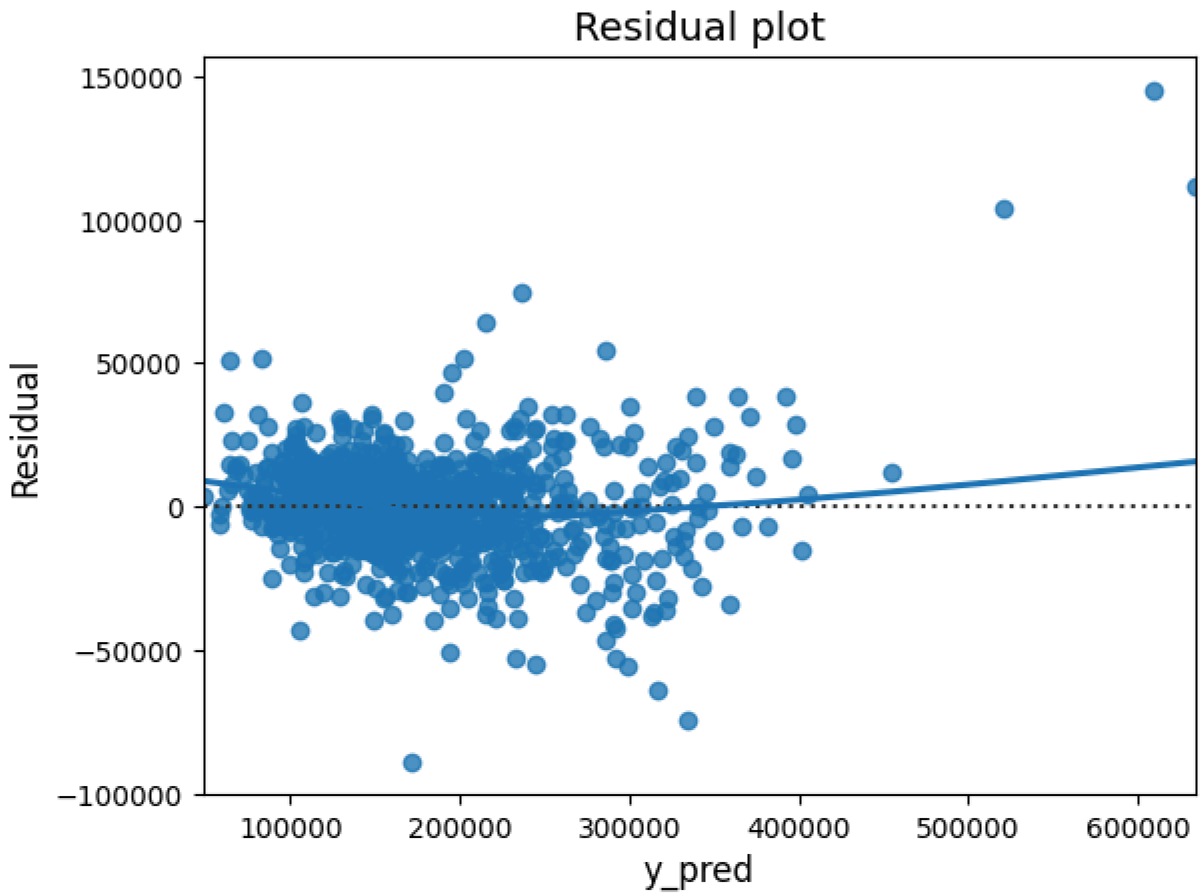
In [45]:
```
val_pred = linear_model.predict(X_val)
val_X_t = pca_pipeline.fit_transform(X_val)
```

In [46]:
```
fig = scatter_plot(x=val_X_t, y=y_val, y_pred=val_pred, label="Linear Model")
save_plot(fname="LinerModelScatter", filetype="plt")
fig.show()
```

<Figure size 640x480 with 0 Axes>

In [47]:
```
# making the residual plots
y_pred = linear_model.predict(X_train)
residuals = y_train - y_pred
```

In [48]:
```
sns.residplot(x=y_pred, y=residuals,lowess=True)
plt.title("Residual plot")
plt.xlabel("y_pred")
plt.ylabel("Residual");
```



In [49]:
```
# residual distributions
fig = px.histogram(
    residuals,
    nbins=50,
    labels = "Residuals",
    title = "Predicted Sale Price: Linear Regression Model Residuals Distribution"
)
fig.update_layout(
    xaxis_title = "Residuals",
    yaxis_title = "Frequency (counts)",
    legend_title = "Resid Parameter"
)
fig.show()
```

In [50]:
```
coeficients = linear_model.named_steps["linear_model"].coef_
features  = linear_model.named_steps["preprocess"].get_feature_names_out().ravel()
features = [f.split("__")[1] for f in features]

# making feature importances
feat_imp = pd.Series(coeficients, index=features, name="feature importance").sort_values(key=abs
feat_imp.tail(10)
```

Out[50]:
```
Functional_Typ          22412.170055
LandSlope_Sev          -23038.429779
Exterior1st_CemntBd    -24097.690938
SaleType_New            24343.858790
Exterior1st_BrkFace     24842.119226
Exterior1st_BrkComm    -25595.153582
Neighborhood_NoRidge    27621.458356
Exterior2nd_CmentBd     31782.161196
Exterior1st_ImStucc    -36284.273998
Functional_Sev         -43386.695237
Name: feature importance, dtype: float64
```

In [51]:
```
fig = px.bar(
    feat_imp.tail(10),
    orientation = "h",
    title = "Linear Model Feature importance plot"
)
fig.update_layout(
    xaxis_title = "Importances",
    yaxis_title = "Features",
    legend_title = "Item"
```

```
)
fig.show()
```

In [52]:
```
# let us save the model
save_model(mname="LinearRegressionModel", model=linear_model)
```

8. ID mapping

In [53]:
```
repo = WrangleRepository(file_name="test (1).csv")
repo
```

Out[53]:
```
WrangleRepository filepath=C:\Users\MY PC\Desktop\Projects\Regression\AmosHousePriceModelling\te
st (1).csv
```

In [54]:
```
# getting the data
repo.wrangle()
df = repo.get_data("wrangled")
print(df.shape)
df.head()
```

```
(1459, 79)
```

Out[54]:

Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	Lot
1461	20	RH	80.0	11622	Pave	NaN	Reg	Lvl	AllPub	
1462	20	RL	81.0	14267	Pave	NaN	IR1	Lvl	AllPub	
1463	60	RL	74.0	13830	Pave	NaN	IR1	Lvl	AllPub	
1464	60	RL	78.0	9978	Pave	NaN	IR1	Lvl	AllPub	
1465	120	RL	43.0	5005	Pave	NaN	IR1	HLS	AllPub	

5 rows × 79 columns

In [55]:

```
# basic cleaning (no)
repo.basic_cleaning(clean=False)
df = repo.get_data("basic")
print(df.shape)
df.head()
```

```
(1459, 79)
```

Out[55]:

Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	Lot
1461	20	RH	80.0	11622	Pave	NaN	Reg	Lvl	AllPub	
1462	20	RL	81.0	14267	Pave	NaN	IR1	Lvl	AllPub	
1463	60	RL	74.0	13830	Pave	NaN	IR1	Lvl	AllPub	
1464	60	RL	78.0	9978	Pave	NaN	IR1	Lvl	AllPub	
1465	120	RL	43.0	5005	Pave	NaN	IR1	HLS	AllPub	

5 rows × 79 columns

In [56]:

```
# feature selction (n0)
repo.feature_selection(variance_selector=False)
df = repo.get_data("selected")
print(df.shape)
df.head()
```

```
(1459, 79)
```

Out[56]:

Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	Lot
1461	20	RH	80.0	11622	Pave	NaN	Reg	Lvl	AllPub	
1462	20	RL	81.0	14267	Pave	NaN	IR1	Lvl	AllPub	
1463	60	RL	74.0	13830	Pave	NaN	IR1	Lvl	AllPub	
1464	60	RL	78.0	9978	Pave	NaN	IR1	Lvl	AllPub	
1465	120	RL	43.0	5005	Pave	NaN	IR1	HLS	AllPub	

5 rows × 79 columns

In [57]:

```
# feature engineering
repo.feature_engineering()
df = repo.get_data("engineered")
print(df.shape)
df.head()
```

```
C:\Users\MY PC\Desktop\Projects\Regression\AmosHousePriceModelling\Training.py:140: FutureWarnin
g:

Series.replace without 'value' and with non-dict-like 'to_replace' is deprecated and will raise i
n a future version. Explicitly specify the new values instead.
```

```
(1459, 85)
```

Out[57]:

Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	Lot
1461	20	RH	80.0	11622	Pave	NaN	Reg	Lvl	AllPub	
1462	20	RL	81.0	14267	Pave	NaN	IR1	Lvl	AllPub	
1463	60	RL	74.0	13830	Pave	NaN	IR1	Lvl	AllPub	
1464	60	RL	78.0	9978	Pave	NaN	IR1	Lvl	AllPub	
1465	120	RL	43.0	5005	Pave	NaN	IR1	HLS	AllPub	

5 rows × 85 columns

In [58]:

```
# final mapping
df_test = df[X_train.columns]
print(df_test.shape)
df_test.head()
```

```
(1459, 72)
```

Out[58]:

Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrAr
1461	20	80.0	11622	5	6	1961	1961	0
1462	20	81.0	14267	6	6	1958	1958	108
1463	60	74.0	13830	5	5	1997	1998	0
1464	60	78.0	9978	6	6	1998	1998	20
1465	120	43.0	5005	8	5	1992	1992	0

5 rows × 72 columns

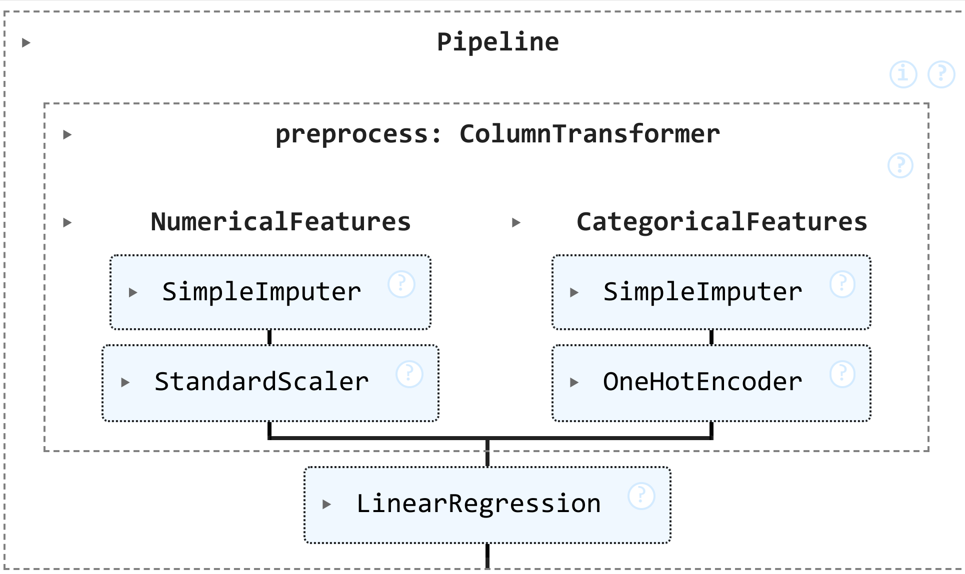
In [59]:
```
from Training import TestPredicter
# Get the predictions
tp = TestPredicter(test_data=df_test, model=linear_model)
print(type(tp))
tp
```

```
<class 'Training.TestPredicter'>
```

Out[59]:
```
TestMapper on C:\Users\MY PC\Desktop\Projects\Regression\AmosHousePriceModelling
```

In [60]:
```
# linear regression
linear_model = Pipeline(
    [
        ("preprocess", col_pipeline),
        ("linear_model", LinearRegression())
    ]
)
# Training the model
linear_model.fit(X_train, y_train)
```

Out[60]:



In [61]:
```
tp.predict()
pred = tp.get_data("prediction")
print(type(pred))
pred[:4]
```

```
<class 'numpy.ndarray'>
```

Out[61]:
```
array([109562.29459681, 149973.17519836, 173746.19441704, 187191.35919247])
```

In [62]:
```python
tp.id_mapper(label="linear_regression")
sub = tp.get_data("mapped")
print(type(sub))
print(sub.shape)
sub.head()
```

```
<class 'pandas.core.frame.DataFrame'>
(1459, 1)
```

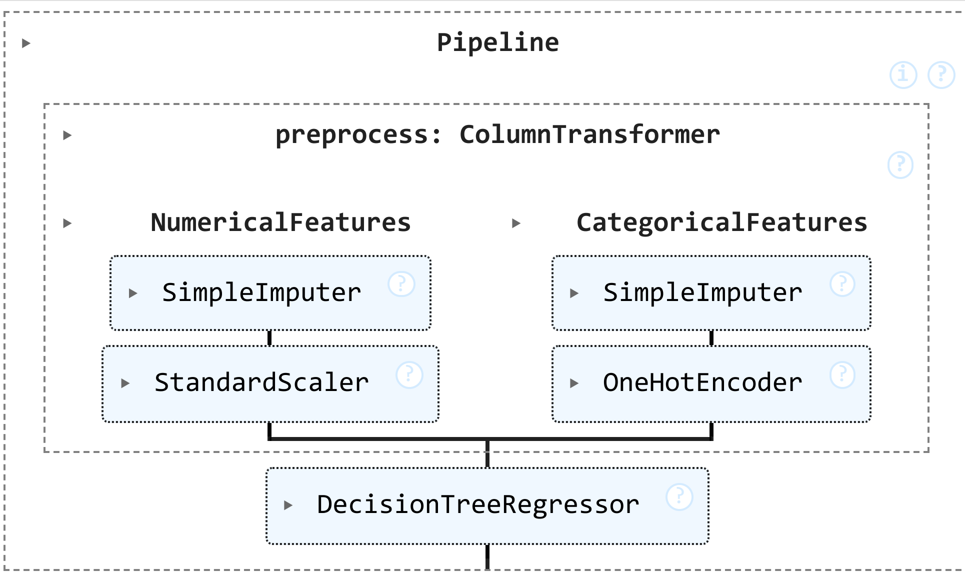
Out[62]:

Id	SalePrice
1461	109562.294597
1462	149973.175198
1463	173746.194417
1464	187191.359192
1465	186420.551938

10. Decision Tree Regressor

In [63]:
```python
# Building the pipeline with decision tree regressor
tree_pipeline = Pipeline(
    [
        ("preprocess", col_pipeline),
        ("tree_model", DecisionTreeRegressor(random_state=42))
    ]
)
# Training the model
tree_model = tree_pipeline.fit(X_train, y_train)
tree_model
```

Out[63]:



We have now trained a decision tree model, next up is to evaluate our model and continue to do hyperameter tuning.

In [64]:
```python
# model evaluation using mae, mape, coefficient of difference(cod, R2)
pred = tree_model.predict(X_train)
# With training data
```

```python
mae = mean_absolute_error(y_train, pred)
mape = mean_absolute_percentage_error(y_train, pred)
cod = r2_score(y_train, pred)
print(f"Training MAE: ${np.round(mae, 2)}")
print(f"Training MAPE: {100 * np.round(mape, 2)}%")
print(f"Training R2: {100 * np.round(cod, 2)}%")
```

```
Training MAE: $0.0
Training MAPE: 0.0%
Training R2: 100.0%
```

In [65]:
```python
# Evaluation using validation set
# model evaluation using mae, mape, coefficient of difference(cod, R2)
pred = tree_model.predict(X_val)
# With training data
mae = mean_absolute_error(y_val, pred)
mape = mean_absolute_percentage_error(y_val, pred)
cod = r2_score(y_val, pred)
print(f"Training MAE: ${np.round(mae, 2)}")
print(f"Training MAPE: {100 * np.round(mape, 2)}%")
print(f"Training R2: {100 * np.round(cod, 2)}%")
```

```
Training MAE: $19773.83
Training MAPE: 13.0%
Training R2: 80.0%
```

Definately we have some element of overfitting, let's visualize this using a plot. We are going to use the LearningCurve class that has all the three modules.

In [66]:
```python
# get the learning curve class
lc = LearningCurve(estimator=tree_model, X=X, y=y)
print(type(lc))
lc
```

```
<class 'Training.LearningCurve'>
```

Out[66]:
```
LearningCurve: <class 'sklearn.pipeline.Pipeline'>
```

In [67]:
```python
# Het learning curve list data
lc.learning_curve()
tree_lc = lc.get_data("lc")
print(type(tree_lc))
tree_lc[:4]
```

```
[learning_curve] Training set sizes: [ 93 304 514 725 936]
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:    2.9s finished
<class 'list'>
```

Out[67]:
```
[array([ 93, 304, 514, 725, 936]),
 array([1., 1., 1., 1., 1.]),
 array([0.11831156, 0.54036325, 0.72716731, 0.69733103, 0.75523564])]
```

In [68]:
```python
# make the learning curve dataframe
lc.make_dataframe()
tree_lc_df = lc.get_data("df_lc")
print(type(tree_lc_df))
print(tree_lc_df.shape)
tree_lc_df
```

```
[learning_curve] Training set sizes: [ 93 304 514 725 936]
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:    2.1s finished
```

```
<class 'pandas.core.frame.DataFrame'>
(5, 3)
```

Out[68]:

	Train Size	Train R2	Validation R2
0	93	1.0	0.118312
1	304	1.0	0.540363
2	514	1.0	0.727167
3	725	1.0	0.697331
4	936	1.0	0.755236

In [69]:

```
# Melting the dataframe
lc.melt_dataframe()
tree_lc_melt = lc.get_data("melt_lc")
print(type(tree_lc_melt))
print(tree_lc_melt.shape)
tree_lc_melt.head()
```

```
[learning_curve] Training set sizes: [ 93 304 514 725 936]
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:    2.1s finished
<class 'pandas.core.frame.DataFrame'>
(10, 3)
```

Out[69]:

	Train Size	Set	R2
0	93	Train R2	1.0
1	304	Train R2	1.0
2	514	Train R2	1.0
3	725	Train R2	1.0
4	936	Train R2	1.0

In [70]:

```
# making the plot
lc.plot_lc()
fig = lc.get_data()
print(type(fig))
save_plot(fname="decion_tree_model_learning_curve", filetype="plt", fig=fig)
fig.show()
```

```
[learning_curve] Training set sizes: [ 93 304 514 725 936]
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:    2.8s finished
<class 'plotly.graph_objs._figure.Figure'>
```

```
<Figure size 640x480 with 0 Axes>
```

As seen we are overfitting and we need to improve our model by doing the following:

1. Reduce model complexity- reducing the depth of the model
2. Do a cross validation (kfold)
3. hyperameter tuning

But before we do that, let us get the maxim depth of our model.

In [71]:

```
# trianing different trees
train_acc = []
val_acc = []
d_params = range(1, 10)
for d in d_params:
    # training the model
    model = Pipeline(
        [
            ("preprocess", col_pipeline),
            ("model", DecisionTreeRegressor(max_depth=d, random_state=42))
        ]
    )
    model.fit(X_train, y_train)
    # R2
    train_acc.append(r2_score(y_train, model.predict(X_train)))
    val_acc.append(r2_score(y_val, model.predict(X_val)))
```

In [72]:

```
# building a dataframe
df_result = pd.DataFrame(
```

```
    {
        "Depth": d_params,
        "Train Accuracy": train_acc,
        "Validation Accuracy": val_acc
    }
)
df_result
```

Out[72]:

	Depth	Train Accuracy	Validation Accuracy
0	1	0.428295	0.470128
1	2	0.614841	0.533776
2	3	0.734256	0.660315
3	4	0.802654	0.707014
4	5	0.870398	0.788757
5	6	0.911107	0.821424
6	7	0.944992	0.838294
7	8	0.964762	0.791172
8	9	0.977373	0.827457

In [73]:

```
# meltiing the dataframe,
result_melt = pd.melt(
    frame=df_result,
    id_vars="Depth",
    value_vars=["Train Accuracy", "Validation Accuracy"],
    value_name="Accuracy",
    var_name="Set"
)
result_melt.head()
```

Out[73]:

	Depth	Set	Accuracy
0	1	Train Accuracy	0.428295
1	2	Train Accuracy	0.614841
2	3	Train Accuracy	0.734256
3	4	Train Accuracy	0.802654
4	5	Train Accuracy	0.870398

In [74]:

```
# plotting
fig = px.line(
    data_frame=result_melt,
    x = "Depth",
    y = "Accuracy",
    color = "Set",
    title="Decision Tree Model: Training and validataion accuracy curves"
)
fig.update_layout(
    xaxis_title="Depth of the Tree",
    yaxis_title="Accuracy",
    legend_title="Accuracy"
```

```
)
fig.show()
```

As seen in the plot above a depth of seven is having the highest validation accuracy. Let us train our final model using the depth of seven first and define other hyperameters that will help reduce overfitting.

In [75]:
```
params = {
    "tree_model__min_samples_split": [3,4,5,6],
    "tree_model__min_samples_leaf": [1,2,3,4]
}
params
```

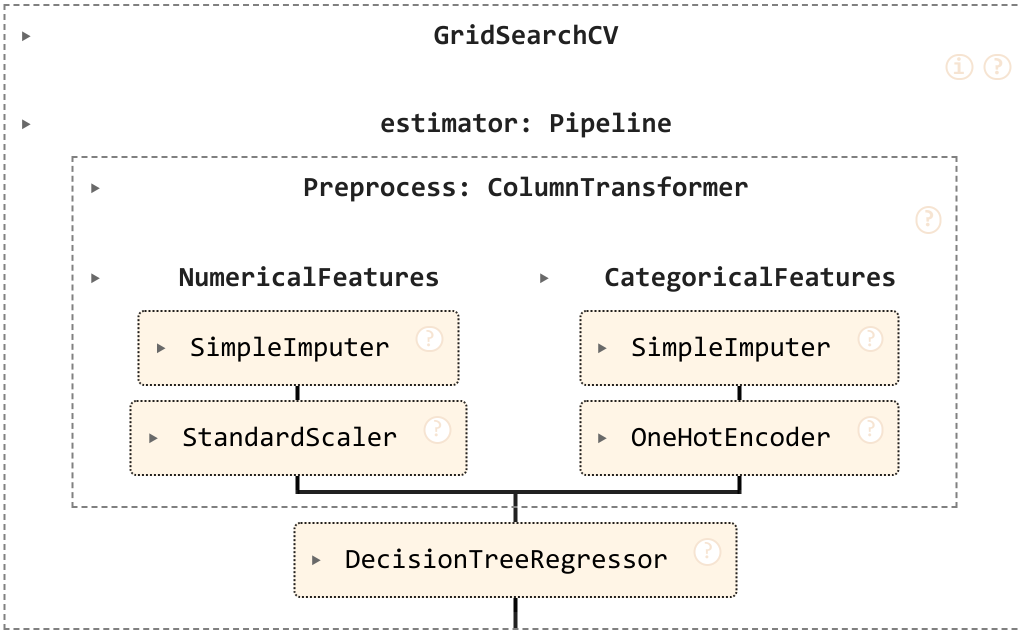
Out[75]:
```
{'tree_model__min_samples_split': [3, 4, 5, 6],
 'tree_model__min_samples_leaf': [1, 2, 3, 4]}
```

In [76]:
```
# model pipeline
tree_model_pipeline = Pipeline(
    [
        ("Preprocess", col_pipeline),
        ("tree_model", DecisionTreeRegressor(max_depth=7, random_state=42))
    ]
)

# Grid search
tree_model_cv = GridSearchCV(
    estimator=tree_model_pipeline,
    param_grid=params,
    n_jobs=-1,
    cv=5,
```

```
    verbose=1
)
tree_model_cv
```

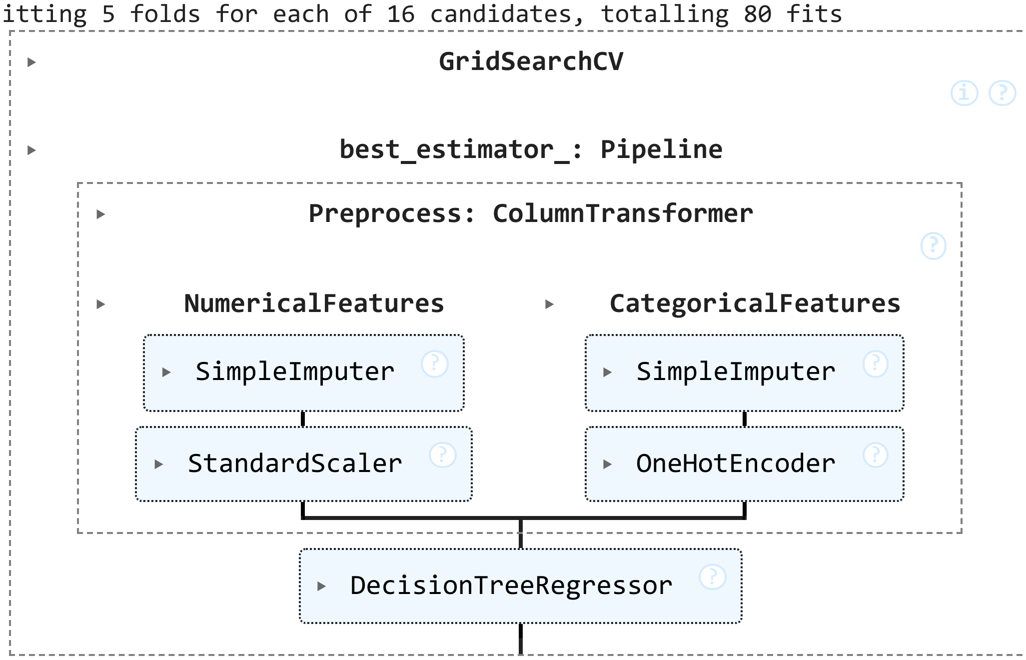
Out[76]:



In [77]:
```
# fitting the model
tree_model_cv.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
```

Out[77]:



In [78]:
```
# getting the best paramers
tree_model_cv.best_params_
```

Out[78]:
```
{'tree_model__min_samples_leaf': 1, 'tree_model__min_samples_split': 6}
```

In [79]:
```
# Betting the model cv results into a dataframe
cv_result = pd.DataFrame.from_dict(tree_model_cv.cv_results_)
print(type(cv_result))
print(cv_result.shape)
cv_result.head()
```

```
<class 'pandas.core.frame.DataFrame'>
(16, 15)
```

Out[79]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_tree_model__min_samples_leaf	para
0	0.366880	0.099283	0.084197	0.023611	1	
1	0.252607	0.045575	0.063367	0.015991	1	
2	0.208718	0.034707	0.052121	0.015735	1	
3	0.236206	0.056709	0.078299	0.029736	1	
4	0.322267	0.053949	0.070477	0.019057	2	

In [80]:

```
# making splits score dataframe
score_col = ["rank_test_score", "split0_test_score", "split1_test_score", "split2_test_score", ":
split_score = cv_result[score_col]
# melting the dataframe
split_score = pd.melt(frame=split_score, var_name="Set", value_name="Score", id_vars="rank_test_
split_score.head()
```

Out[80]:

	rank_test_score	Set	Score
0	16	split0_test_score	0.643135
1	5	split0_test_score	0.728169
2	2	split0_test_score	0.728169
3	1	split0_test_score	0.741359
4	14	split0_test_score	0.667127

In [81]:

```
# plotting
fig = px.line(
    data_frame=split_score,
    x=split_score.index,
    y="Score",
    color="Set",
    title="Hyperameter Sets: Decision Tree Model"
)
fig.update_layout(
    xaxis_title="Training Index",
    yaxis_title="Accuracy (%)",
    legend_title="Split Type"
)
fig.show()
```

As seen some model had really high accuracies but now let us get the best model and parameters.

In [82]:
```
tree_model_cv.best_params_
```

Out[82]:
```
{'tree_model__min_samples_leaf': 1, 'tree_model__min_samples_split': 6}
```

Now that we have our best model, let us evaluate it.

In [83]:
```
# model evaluation using mae, mape, coefficient of difference(cod, R2)
pred = tree_model_cv.predict(X_train)
# With training data
mae = mean_absolute_error(y_train, pred)
mape = mean_absolute_percentage_error(y_train, pred)
cod = r2_score(y_train, pred)
print(f"Training MAE: ${np.round(mae, 2)}")
print(f"Training MAPE: {100 * np.round(mape, 2)}%")
print(f"Training R2: {100 * np.round(cod, 2)}%")
```

```
Training MAE: $12683.2
Training MAPE: 8.0%
Training R2: 94.0%
```

In [84]:
```
# model evaluation using mae, mape, coefficient of difference(cod, R2)
pred = tree_model_cv.predict(X_val)
# With training data
mae = mean_absolute_error(y_val, pred)
mape = mean_absolute_percentage_error(y_val, pred)
cod = r2_score(y_val, pred)
print(f"Training MAE: ${np.round(mae, 2)}")
```

```
print(f"Training MAPE: {100 * np.round(mape, 2)}%")
print(f"Training R2: {100 * np.round(cod, 2)}%")
```

```
Training MAE: $19463.83
Training MAPE: 13.0%
Training R2: 81.0%
```

We have tried to reduce overfitting but one more thing to be done is increaing the data. Since this is a kaggle competition, then we will not do that. we will investige this property using a learning curve.

In [85]:
```
lc = LearningCurve(estimator=tree_model_cv, X=X, y=y)
print(type(lc))
lc
```

```
<class 'Training.LearningCurve'>
```

Out[85]:
```
LearningCurve: <class 'sklearn.model_selection._search.GridSearchCV'>
```

In [86]:
```
# learning curve list
lc.learning_curve()
ls = lc.get_data("lc")
print(type(ls))
ls[:4]
```

```
[learning_curve] Training set sizes: [ 93 304 514 725 936]
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:  1.5min finished
<class 'list'>
```

Out[86]:
```
[array([ 93, 304, 514, 725, 936]),
 array([0.91356261, 0.94798811, 0.95389146, 0.94835213, 0.9373437 ]),
 array([0.27786857, 0.65022828, 0.72233115, 0.75555916, 0.77462749])]
```

In [87]:
```
# getting the data
lc.make_dataframe()
ld = lc.get_data("df_lc")
print(type(ld))
print(ld.shape)
ld.head()
```

```
[learning_curve] Training set sizes: [ 93 304 514 725 936]
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:  1.4min finished
<class 'pandas.core.frame.DataFrame'>
(5, 3)
```

Out[87]:

	Train Size	Train R2	Validation R2
0	93	0.913563	0.277869
1	304	0.947988	0.650228
2	514	0.953891	0.722331
3	725	0.948352	0.755559
4	936	0.937344	0.774627

In [88]:
```
# getting the melted data
lc.melt_dataframe()
ld = lc.get_data("melt_lc")
print(type(ld))
```

```
print(ld.shape)
ld.head()
```

```
[learning_curve] Training set sizes: [ 93 304 514 725 936]
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:  1.6min finished
<class 'pandas.core.frame.DataFrame'>
(10, 3)
```

Out[88]:

	Train Size	Set	R2
0	93	Train R2	0.913563
1	304	Train R2	0.947988
2	514	Train R2	0.953891
3	725	Train R2	0.948352
4	936	Train R2	0.937344

In [89]:

```
# getting the data
lc.plot_lc()
fig = lc.get_data()
print(type(fig))
fig.show()
```

```
[learning_curve] Training set sizes: [ 93 304 514 725 936]
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:  1.4min finished
<class 'plotly.graph_objs._figure.Figure'>
```

In [102…

```python
coeficients = tree_model.named_steps["tree_model"].feature_importances_
features  = tree_model.named_steps["preprocess"].get_feature_names_out().ravel()
features = [f.split("__")[1] for f in features]

# making feature importances
feat_imp = pd.Series(coeficients, index=features, name="feature importance").sort_values(key=abs
feat_imp.tail(10)
```

Out[102…

```
YearRemodAdd    0.006977
1stFlrSF        0.007941
LotArea         0.010304
BsmtFinSF1      0.013944
BsmtFinished    0.015853
TotalBsmtSF     0.047766
GarageArea      0.093964
GrLivArea       0.121806
2ndFlrSF        0.134641
OverallQual     0.453393
Name: feature importance, dtype: float64
```

In [99]:

```python
fig = px.bar(
    feat_imp.tail(10),
    orientation = "h",
    title = "Top 10: Decision Tree Model Feature importance plot"
)
fig.update_layout(
    xaxis_title = "Importances",
    yaxis_title = "Features",
    legend_title = "Item"

)
fig.show()
```

Definately this plot confirms that we need more data which we don't have maybe we can improve our model using another method.One thing we have done successfully is that we have been able to reduce overfitting as seen. We will start doing bagging and boosting learning methods to try and see if our model performance will increase and we reduce overfitting.

In [103…

```
# saving the model
save_model(mname="decision_tree_model", model=tree_model_cv)
```

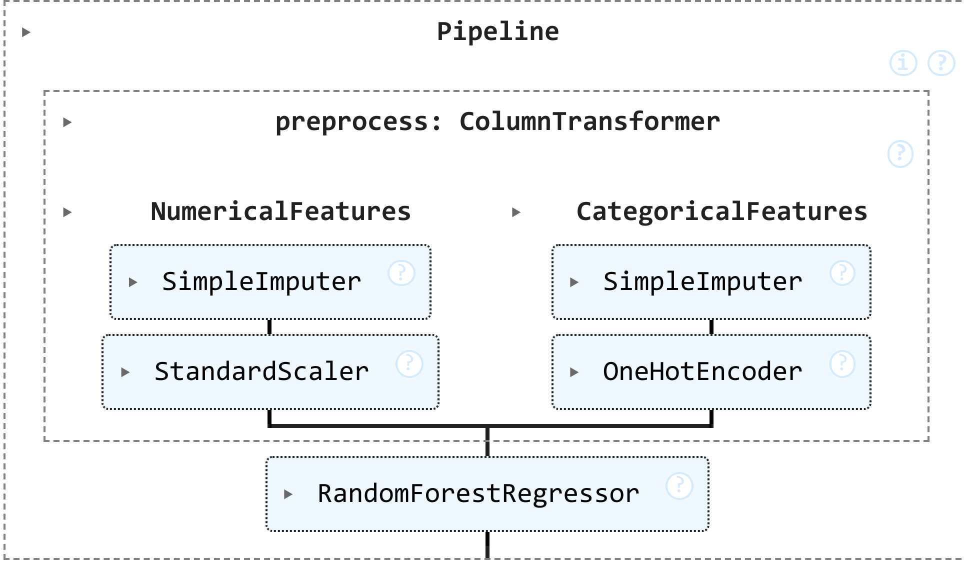
11. Bagging Model

Random forest model is one of the bagging models. We are going to train that and see the progress.

In [104…

```
# Model pipeline
forest_model = Pipeline(
    [
        ("preprocess", col_pipeline),
        ("forest_model", RandomForestRegressor(max_depth=7))
    ]
)
# Training the model
forest_model.fit(X_train, y_train)
```

Out[104...



In [105...

```python
# model evaluation using mae, mape, coefficient of difference(cod, R2)
pred = forest_model.predict(X_train)
# With training data
mae = mean_absolute_error(y_train, pred)
mape = mean_absolute_percentage_error(y_train, pred)
cod = r2_score(y_train, pred)
print(f"Training MAE: ${np.round(mae, 2)}")
print(f"Training MAPE: {100 * np.round(mape, 2)}%")
print(f"Training R2: {100 * np.round(cod, 2)}%")
```

```
Training MAE: $10363.67
Training MAPE: 6.0%
Training R2: 96.0%
```

In [106...

```python
# model evaluation using mae, mape, coefficient of difference(cod, R2)
pred = forest_model.predict(X_val)
# With training data
mae = mean_absolute_error(y_val, pred)
mape = mean_absolute_percentage_error(y_val, pred)
cod = r2_score(y_val, pred)
print(f"Validation MAE: ${np.round(mae, 2)}")
print(f"Validation MAPE: {100 * np.round(mape, 2)}%")
print(f"Validation R2: {100 * np.round(cod, 2)}%")
```

```
Validation MAE: $13951.75
Validation MAPE: 10.0%
Validation R2: 90.0%
```

Actually this model is perfoming better. We will first check for overfitting and them do a hyperameter tuning.

In [107...

```python
lcf = LearningCurve(estimator=forest_model, X=X, y=y)
print(type(lcf))
lcf
```

```
<class 'Training.LearningCurve'>
```

Out[107...

```
LearningCurve: <class 'sklearn.pipeline.Pipeline'>
```

In [108...

```python
# Getting the learning curve list
lcf.learning_curve()
lcf_list = lcf.get_data("lc")
print(type(lcf_list))
lcf
```

```
[learning_curve] Training set sizes: [ 93 304 514 725 936]
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
<class 'list'>
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:   39.3s finished
```

Out[108…

```
LearningCurve: <class 'sklearn.pipeline.Pipeline'>
```

In [109…

```
# getting the dataframe
lcf.make_dataframe()
lcf_df = lcf.get_data("df_lc")
print(type(lcf_df))
print(lcf_df.shape)
lcf_df
```

```
[learning_curve] Training set sizes: [ 93 304 514 725 936]
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:   38.3s finished
<class 'pandas.core.frame.DataFrame'>
(5, 3)
```

Out[109…

	Train Size	Train R2	Validation R2
0	93	0.951846	0.798274
1	304	0.967922	0.832565
2	514	0.968533	0.851900
3	725	0.963995	0.854568
4	936	0.963775	0.873440

In [110…

```
# melting the dataframe
lcf.melt_dataframe()
lcf_melt = lcf.get_data("melt_lc")
print(type(lcf_melt))
print(lcf_melt.shape)
lcf_melt.head()
```

```
[learning_curve] Training set sizes: [ 93 304 514 725 936]
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:   30.3s finished
<class 'pandas.core.frame.DataFrame'>
(10, 3)
```

Out[110…

	Train Size	Set	R2
0	93	Train R2	0.961998
1	304	Train R2	0.970980
2	514	Train R2	0.968924
3	725	Train R2	0.965196
4	936	Train R2	0.963077

In [111…

```
# plotting the learning curve
lcf.plot_lc()
fig = lcf.get_data()
print(type(fig))
fig.show()
```

```
[learning_curve] Training set sizes: [ 93 304 514 725 936]
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:   32.0s finished
<class 'plotly.graph_objs._figure.Figure'>
```

With more data we can actually get better results. Our model is not overfitting but we will try different hyperameter to see if we will get a better optimal parameters

In [114…

```
f_params = {
    "forest_model__n_estimators": range(20, 100, 20),
    "forest_model__max_depth": range(4,12,2),
    "forest_model__min_samples_split": [4,6,8,10],
    "forest_model__min_samples_leaf": [1,2,3,4]
}
f_params
```

Out[114…

```
{'forest_model__n_estimators': range(20, 100, 20),
 'forest_model__max_depth': range(4, 12, 2),
 'forest_model__min_samples_split': [4, 6, 8, 10],
 'forest_model__min_samples_leaf': [1, 2, 3, 4]}
```

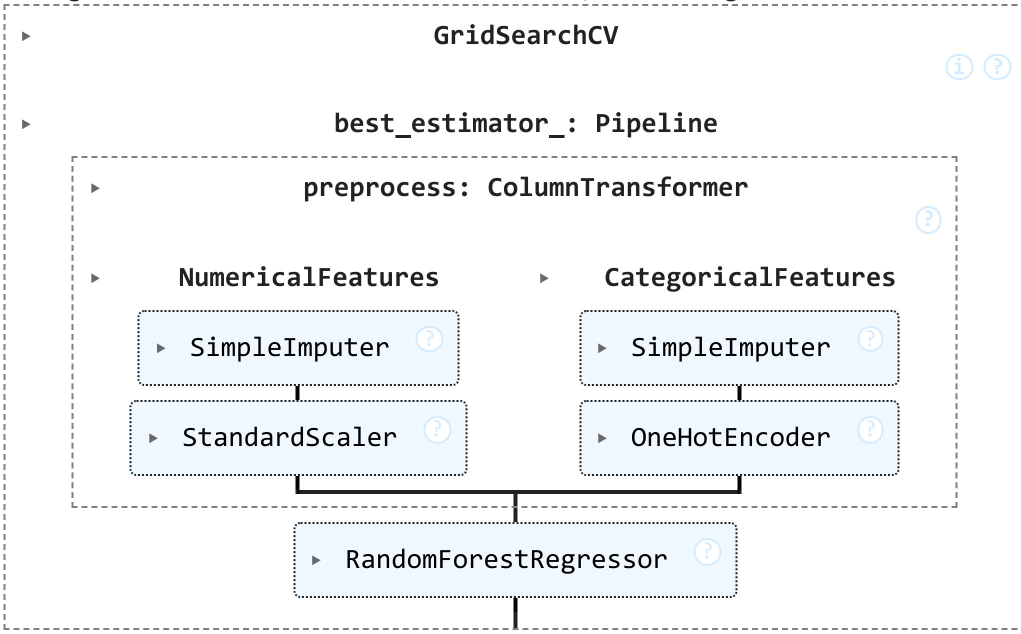
In [115…

```
# model pipeline
forest_model_cv = Pipeline(
    [
        ("preprocess", col_pipeline),
        ("forest_model", RandomForestRegressor(random_state=42))
    ]
)
# Cross validation
forest_model_cv = GridSearchCV(
```

```
    estimator=forest_model_cv,
    param_grid=f_params,
    cv=5,
    n_jobs=-1,
    verbose=1
)
# Training the model
forest_model_cv.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 256 candidates, totalling 1280 fits
```

Out[115…



In [116…

```
# Getting the model best parameters
forest_model_cv.best_params_
```

Out[116…

```
{'forest_model__max_depth': 10,
 'forest_model__min_samples_leaf': 1,
 'forest_model__min_samples_split': 4,
 'forest_model__n_estimators': 60}
```

In [117…

```
# Getting the cv results
# Betting the model cv results into a dataframe
cv_result = pd.DataFrame.from_dict(forest_model_cv.cv_results_)
print(type(cv_result))
print(cv_result.shape)
cv_result.head()
```

```
<class 'pandas.core.frame.DataFrame'>
(256, 17)
```

Out[117…

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_forest_model__max_depth	param_f
0	0.608900	0.036595	0.064277	0.015778	4	
1	1.176057	0.036320	0.067018	0.005829	4	
2	1.696953	0.037536	0.064754	0.003808	4	
3	2.283819	0.036710	0.081712	0.009918	4	
4	0.626586	0.018374	0.048398	0.003170	4	

In [118…

```
# making splits score dataframe
score_col = ["rank_test_score", "split0_test_score", "split1_test_score", "split2_test_score", "s
split_score = cv_result[score_col]
# melting the dataframe
split_score = pd.melt(frame=split_score, var_name="Set", value_name="Score", id_vars="rank_test_s
split_score.head()
```

Out[118…

	rank_test_score	Set	Score
0	201	split0_test_score	0.873174
1	204	split0_test_score	0.871229
2	194	split0_test_score	0.864020
3	193	split0_test_score	0.865940
4	203	split0_test_score	0.874149

In [119…

```
# plotting
fig = px.line(
    data_frame=split_score,
    x=split_score.index,
    y="Score",
    color="Set",
    title="Hyperameter Sets: Decision Tree Model"
)
fig.update_layout(
    xaxis_title="Training Index",
    yaxis_title="Accuracy (%)",
    legend_title="Split Type"
)
fig.show()
```

With that information let us evaluation our best model.

In [120…

```
# model evaluation using mae, mape, coefficient of difference(cod, R2)
pred = forest_model_cv.best_estimator_.predict(X_train)
# With training data
mae = mean_absolute_error(y_train, pred)
mape = mean_absolute_percentage_error(y_train, pred)
cod = r2_score(y_train, pred)
print(f"Training MAE: ${np.round(mae, 2)}")
print(f"Training MAPE: {100 * np.round(mape, 2)}%")
print(f"Training R2: {100 * np.round(cod, 2)}%")
```

```
Training MAE: $7487.2
Training MAPE: 4.0%
Training R2: 98.0%
```

In [121…

```
# model evaluation using mae, mape, coefficient of difference(cod, R2)
pred = forest_model_cv.best_estimator_.predict(X_val)
# With training data
mae = mean_absolute_error(y_val, pred)
mape = mean_absolute_percentage_error(y_val, pred)
cod = r2_score(y_val, pred)
print(f"Validation MAE: ${np.round(mae, 2)}")
print(f"Validation MAPE: {100 * np.round(mape, 2)}%")
print(f"Validation R2: {100 * np.round(cod, 2)}%")
```

```
Validation MAE: $13666.03
Validation MAPE: 9.0%
Validation R2: 90.0%
```

This model is perfoming pletty well.

Let us save map our ids for submission and finally save the model.

In [122…

```
# id mapping
tcf = TestPredicter(test_data=df_test, model=forest_model_cv)
print(type(tcf))
tcf
```

```
<class 'Training.TestPredicter'>
```

Out[122…

```
TestMapper on C:\Users\MY PC\Desktop\Projects\Regression\AmosHousePriceModelling
```

In [123…

```
# Getting the predictions
tcf.predict()
pred_f = tcf.get_data("prediction")
print(type(pred_f))
pred_f[:4]
```

```
<class 'numpy.ndarray'>
```

Out[123…

```
array([124626.98675698, 157442.53347212, 181281.75170203, 186643.43811172])
```

In [124…

```
# mapping the ids
tcf.id_mapper(label="forest")
sub_f = tcf.get_data("mapped")
print(type(df))
print(sub_f.shape)
sub_f.head()
```

```
<class 'pandas.core.frame.DataFrame'>
(1459, 1)
```

Out[124…

	SalePrice
Id	
1461	124626.986757
1462	157442.533472
1463	181281.751702
1464	186643.438112
1465	188474.401278

In [125…

```
val_pred_f = forest_model_cv.predict(X_val)
```

In [128…

```
# Scatter plot
fig = scatter_plot(x=val_X_t, y=y_val, y_pred=val_pred_f, label="Random Forest Model")
save_plot(fname="ForestModelScatter", filetype="plt")
fig.show()
```

<Figure size 640x480 with 0 Axes>

In [129…

```
# saving the model
save_model(mname="RandomFores_model", model=forest_model_cv)
```

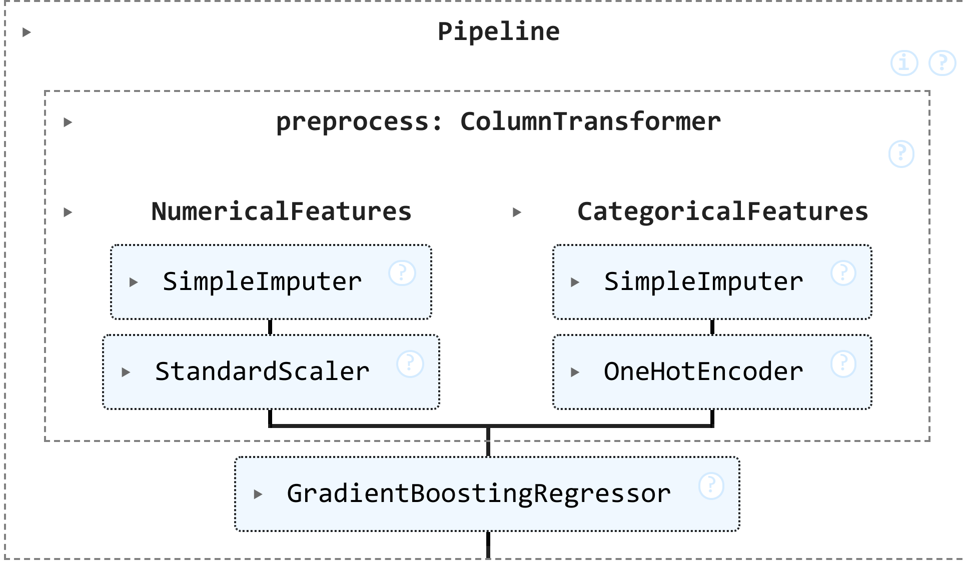
12. Boosting Model

We will try a boosting model, gradient boostng model and see it's perfomance. If it is going to perform better than the rest of the models.

In [130…

```
# gradient boosting model
gradient_model = Pipeline(
    [
        ("preprocess", col_pipeline),
        ("gradient_model", GradientBoostingRegressor(random_state=42))
    ]
)
# fitting the model
gradient_model.fit(X_train, y_train)
```

Out[130…



In [131…

```
# Model evaluation with training set
pred = gradient_model.predict(X_train)
mae = mean_absolute_error(y_train, pred)
mape = mean_absolute_percentage_error(y_train, pred)
R2 = r2_score(y_train, pred)

print(f"Training MAE: ${np.round(mae,2)}")
print(f"Training MAPE: {100 * np.round(mape, 2)}%")
print(f"Training R2: {100 * np.round(R2,2)}%")
```

```
Training MAE: $8732.8
Training MAPE: 5.0%
Training R2: 97.0%
```

In [133…

```
# Model evaluation with training set
pred = gradient_model.predict(X_val)
mae = mean_absolute_error(y_val, pred)
mape = mean_absolute_percentage_error(y_val, pred)
R2 = r2_score(y_val, pred)

print(f"Validation MAE: ${np.round(mae,2)}")
print(f"Validation MAPE: {100 * np.round(mape, 2)}%")
print(f"Validation R2: {100 * np.round(R2,2)}%")
```

```
Validation MAE: $12870.77
Validation MAPE: 9.0%
Validation R2: 91.0%
```

Wow this model has a good performance. Now let us see if we can optimize this and get better ones.

In [134…

```
# PARAMETERS
g_params = {
    "gradient_model__n_estimators":[40, 60, 80, 100],
    "gradient_model__min_samples_split": [3,4,5,6],
    "gradient_model__min_samples_leaf": [1,2,3,4],
    "gradient_model__max_depth": range(2,5)
}
g_params
```

Out[134…

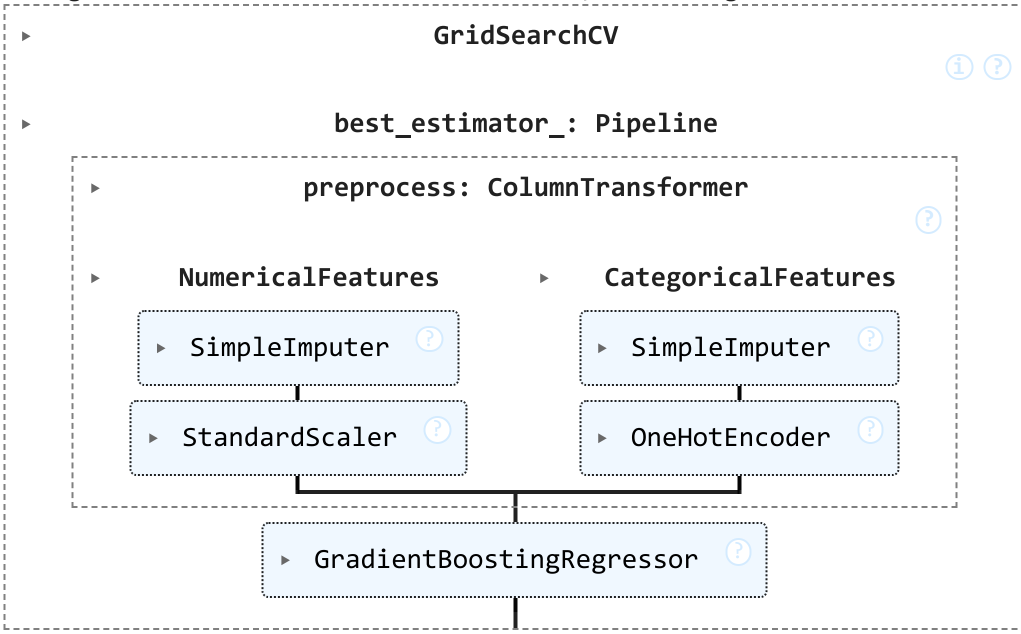
```
{'gradient_model__n_estimators': [40, 60, 80, 100],
 'gradient_model__min_samples_split': [3, 4, 5, 6],
 'gradient_model__min_samples_leaf': [1, 2, 3, 4],
 'gradient_model__max_depth': range(2, 5)}
```

In [135…

```
# making the pipeline
gradient_model_cv = GridSearchCV(estimator=gradient_model,
                                 param_grid=g_params,
                                 cv=5,
                                 n_jobs=-1,
                                 verbose=1
                                )
# fitting the model
gradient_model_cv.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 192 candidates, totalling 960 fits
```

Out[135…



In [137…

```
# evaluating the model
# Model evaluation with training set
pred = gradient_model_cv.predict(X_train)
mae = mean_absolute_error(y_train, pred)
mape = mean_absolute_percentage_error(y_train, pred)
R2 = r2_score(y_train, pred)

print(f"Training MAE: ${np.round(mae,2)}")
print(f"Training MAPE: {100 * mape:.3f}%")
print(f"Training R2: {100 * np.round(R2,2)}%")
```

```
Training MAE: $11550.69
Training MAPE: 7.119%
Training R2: 95.0%
```

In [138…

```
# Model evaluation with training set
pred_val = gradient_model_cv.predict(X_val)
mae = mean_absolute_error(y_val, pred_val)
mape = mean_absolute_percentage_error(y_val, pred_val)
R2 = r2_score(y_val, pred_val)

print(f"Validation MAE: ${np.round(mae,2)}")
print(f"Validation MAPE: {100 * np.round(mape, 2)}%")
print(f"Validation R2: {100 * np.round(R2,2)}%")
```

```
Validation MAE: $13233.12
Validation MAPE: 9.0%
Validation R2: 91.0%
```

In [139…

```
# getting the best parameters
gradient_model_cv.best_params_
```

Out[139…
```
{'gradient_model__max_depth': 2,
 'gradient_model__min_samples_leaf': 1,
 'gradient_model__min_samples_split': 3,
 'gradient_model__n_estimators': 100}
```

In [140…
```
# Id mapping
tp = TestPredicter(test_data=df_test, model=gradient_model_cv)
print(type(tp))
tp
```

```
<class 'Training.TestPredicter'>
```

Out[140…
```
TestMapper on C:\Users\MY PC\Desktop\Projects\Regression\AmosHousePriceModelling
```

In [141…
```
# Predicting
tp.predict()
pred = tp.get_data("prediction")
print(type(pred))
pred[:4]
```

```
<class 'numpy.ndarray'>
```

Out[141…
```
array([120923.12573771, 159079.63441417, 174838.04699479, 181106.65217282])
```

In [142…
```
# getting the csv file mapping
tp.id_mapper(label="Boosting_Model")
df_sub = tp.get_data("mapped")
print(type(df_sub))
print(df_sub.shape)
df_sub.head()
```

```
<class 'pandas.core.frame.DataFrame'>
(1459, 1)
```

Out[142…

	SalePrice
Id	
1461	120923.125738
1462	159079.634414
1463	174838.046995
1464	181106.652173
1465	207832.369161

In [143…
```
# getting the scatter plot
fig = scatter_plot(x=val_X_t, y=y_val, y_pred=pred_val, label="Gradient Boosting Model")
save_plot(fname="gradient_scatter_plot", filetype="plt")
fig.show()
```

```
<Figure size 640x480 with 0 Axes>
```

In [144…

```
# Saving the model
save_model(mname="gradient_boosting_model", model=gradient_model_cv)
```

We have successfully saved the model. Now the next thing we will do is wrap up this project and later on we will create an ineractive dashboard.

13. Conclusions.

Note:

This is what we have done so far in this project:

1. Created a WrangleRepository class in the Training module. This class is doing a couple of thing which are:

    - Getting the raw csv file
    - Do a basic data cleaning by removing features with most missing values i.e more that 50%. The user can decide not to remove those features by including a False in the cleaning argument. why is this important? This will be very helpful in the sense that, in training I want to have a cleaned and well structured dataset but in the case of testing I want to use the data without having it being cleaned.

- Feature selection, where those features with low variance below 95% for numerical features and around 90% for categorical variables. The two variance threshold can be changed to a desirable one. I included a variance_selector just to make sure if I want to pass in test data, I set it to false.
- Feature engineering. This part is adding more data that will help our model perfomance. This is an important data transformation section.
- Remove outliers section which allow the user to indicate if they want to remove outlier or not. I the case of training data we need to remove outlier but when we are using test data, we don't need to remove any outliers.

2. I have also created a LearningCurve class that does the following:

   - Build the learning curve for 5 folds
   - Make a dataframe from the results
   - Melting the dataframe so that, we can easily plot the curves in one figure
   - Plot the figure. A figure will be returned here

3. We also have a TestPredictor class in training module. It does the following:

   - Make a prediction, where the use will only have provided the best model and the training set.This one will help us in building the submission dataframe in the format provided in the kaggle competion
   - Id mapper which will map id with the predicted Sale Price of a house.

4. We also created two special functions that will help us save figures or plots and another one that will help us save the models

5. We have trained different model which are:

   - Baseline model - Predicts the mean over and over again
   - Linear regression model - This model has an R2 score of 90%
   - Decision tree model - With an R2 score of 81%
   - Random forest model - With an R2 score of 89%
   - Gradient boosting model - With an R2 score of 91% I decided to choose the GBM (gradient boosting model) which has a validation accuracy of 91%. We will use this model in further works.

What next:

We are going to create another sub project where we will build an interactive dashboard using dash. We will do this inside our jupyternotebook. This is how we will orgainize our work:

1. Build a presentation layer - This layer will include all the presentation infomation, including texts, headers, figures and more.
2. Build a business layer - This layer will act as a link between the service layer and the presentation layer. It will bridge that gap. Alot will be going on in this layer
3. Build a service layer - This layer will perfom all the operations including buinging the model, data cleaning and more.

```
In [ ]:
```