Prediction of the House Sale Prices.

We started by building intuition for model based learning, explored how the linear regression model worked, understood how the two different approaches to model fitting worked, and some techniques for cleaning, transforming, and selecting features. In this guided project, you can practice what you learned in this course by exploring ways to improve the models we built.

You'll work with housing data for the city of Ames, Iowa, United States from 2006 to 2010. You can read more about why the data was collected here. You can also read about the different columns in the data here.

In [201…

```
import pandas as pd
pd.options.display.max_columns = 999
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold

from sklearn.metrics import mean_squared_error
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
```

In [202…

```
df = pd.read_csv("AmesHousing.txt", delimiter="\t")
```

In [203…

```
def transform_features(df):
    return df

def select_features(df):
    return df[["Gr Liv Area", "SalePrice"]]

def train_and_test(df):
    train = df[:1460]
    test = df[1460:]

    ## You can use `pd.DataFrame.select_dtypes()` to specify column types
    ## and return only those columns as a data frame.
    numeric_train = train.select_dtypes(include=['integer', 'float'])
    numeric_test = test.select_dtypes(include=['integer', 'float'])
```

```
    ## You can use `pd.Series.drop()` to drop a value.
    features = numeric_train.columns.drop("SalePrice")
    lr = linear_model.LinearRegression()
    lr.fit(train[features], train["SalePrice"])
    predictions = lr.predict(test[features])
    mse = mean_squared_error(test["SalePrice"], predictions)
    rmse = np.sqrt(mse)

    return rmse

transform_df = transform_features(df)
filtered_df = select_features(transform_df)
rmse = train_and_test(filtered_df)

rmse
```

Out[203… 57088.25161263909

Feature Engineering

- Handle missing values:
    - All columns:
        - Drop any with 5% or more missing values for row.
    - Text columns:
        - Drop any with 1 or more missing values for row.
    - Numerical columns:
        - For columns with missing values, fill in with the most common value in that column

1: All columns: Drop any with 5% or more missing values for now.

In [204…
```
# accounting of missing values.
missing_val= df.isnull().sum()

# Filter Series to columns containing >5% missing values
drop_missing_val = missing_val[missing_val > (len(df)/20)].sort_values()
drop_missing_val
```

Out[204…
```
Garage Type      157
Garage Yr Blt    159
```

```
Garage Finish     159
Garage Qual       159
Garage Cond       159
Lot Frontage      490
Fireplace Qu     1422
Fence            2358
Alley            2732
Misc Feature     2824
Pool QC          2917
dtype: int64
```

In [205…

```
drop_missing_val =drop_missing_val.index
# Drop those columns from the data frame
df = df.drop(drop_missing_val,axis=1)
```

2: Text columns: Drop any with 1 or more missing values for now.

In [206…

```
# select only object value
text_val = df.select_dtypes(include='object')
text_missing_val  =  text_val.isnull().sum()

drop_text_missing = df[text_missing_val[text_missing_val > 0].index]

## Filter Series to columns containing *any* missing values
df = df.drop(drop_text_missing , axis=1)
```

3: Numerical columns: For columns with missing values, fill in with the most common value in that column

In [207…

```
numeric_val = df.select_dtypes(include=["float","int"])
missing_numeric_val =numeric_val.isnull().sum()
missing_numeric_val = missing_numeric_val[(missing_numeric_val>0) &(missing_numeric_val < (len(df))/20)].sort_values

missing_numeric_val
```

Out[207…

```
BsmtFin SF 1       1
BsmtFin SF 2       1
Bsmt Unf SF        1
Total Bsmt SF      1
Garage Cars        1
Garage Area        1
Bsmt Full Bath     2
Bsmt Half Bath     2
```

```
Mas Vnr Area      23
dtype: int64
```

In [208…

```python
## Compute the most common value for each column in `fixable_nmeric_missing_cols`.
replacement_values_dic = df[missing_numeric_val.index].mode().to_dict(orient = 'records')[0]
replacement_values_dic
```

Out[208…

```
{'BsmtFin SF 1': 0.0,
 'BsmtFin SF 2': 0.0,
 'Bsmt Unf SF': 0.0,
 'Total Bsmt SF': 0.0,
 'Garage Cars': 2.0,
 'Garage Area': 0.0,
 'Bsmt Full Bath': 0.0,
 'Bsmt Half Bath': 0.0,
 'Mas Vnr Area': 0.0}
```

In [212…

```python
## Use `pd.DataFrame.fillna()` to replace missing values.
df = df.fillna(replacement_values_dic)
df
```

Out[212…

	Order	PID	MS SubClass	MS Zoning	Lot Area	Street	Lot Shape	Land Contour	Utilities	Lot Config	Land Slope	Neighborhood	Condition 1	Condition 2	Bldg Type
0	1	526301100	20	RL	31770	Pave	IR1	Lvl	AllPub	Corner	Gtl	NAmes	Norm	Norm	1Fam
1	2	526350040	20	RH	11622	Pave	Reg	Lvl	AllPub	Inside	Gtl	NAmes	Feedr	Norm	1Fam
2	3	526351010	20	RL	14267	Pave	IR1	Lvl	AllPub	Corner	Gtl	NAmes	Norm	Norm	1Fam
3	4	526353030	20	RL	11160	Pave	Reg	Lvl	AllPub	Corner	Gtl	NAmes	Norm	Norm	1Fam
4	5	527105010	60	RL	13830	Pave	IR1	Lvl	AllPub	Inside	Gtl	Gilbert	Norm	Norm	1Fam
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2925	2926	923275080	80	RL	7937	Pave	IR1	Lvl	AllPub	CulDSac	Gtl	Mitchel	Norm	Norm	1Fam
2926	2927	923276100	20	RL	8885	Pave	IR1	Low	AllPub	Inside	Mod	Mitchel	Norm	Norm	1Fam
2927	2928	923400125	85	RL	10441	Pave	Reg	Lvl	AllPub	Inside	Gtl	Mitchel	Norm	Norm	1Fam
2928	2929	924100070	20	RL	10010	Pave	Reg	Lvl	AllPub	Inside	Mod	Mitchel	Norm	Norm	1Fam

	Order	PID	MS SubClass	MS Zoning	Lot Area	Street	Lot Shape	Land Contour	Utilities	Lot Config	Land Slope	Neighborhood	Condition 1	Condition 2	Bldg Type	H
2929	2930	924151050	60	RL	9627	Pave	Reg	Lvl	AllPub	Inside	Mod	Mitchel	Norm	Norm	1Fam	2

2930 rows × 64 columns

In [213…

```
## let's verify that every column has 0 missing values
df.isnull().sum().value_counts()
```

Out[213…

```
0    64
dtype: int64
```

What new features can we create, that better capture the information in some of the features?

In [214…

```
df.columns
```

Out[214…

```
Index(['Order', 'PID', 'MS SubClass', 'MS Zoning', 'Lot Area', 'Street',
       'Lot Shape', 'Land Contour', 'Utilities', 'Lot Config', 'Land Slope',
       'Neighborhood', 'Condition 1', 'Condition 2', 'Bldg Type',
       'House Style', 'Overall Qual', 'Overall Cond', 'Year Built',
       'Year Remod/Add', 'Roof Style', 'Roof Matl', 'Exterior 1st',
       'Exterior 2nd', 'Mas Vnr Area', 'Exter Qual', 'Exter Cond',
       'Foundation', 'BsmtFin SF 1', 'BsmtFin SF 2', 'Bsmt Unf SF',
       'Total Bsmt SF', 'Heating', 'Heating QC', 'Central Air', '1st Flr SF',
       '2nd Flr SF', 'Low Qual Fin SF', 'Gr Liv Area', 'Bsmt Full Bath',
       'Bsmt Half Bath', 'Full Bath', 'Half Bath', 'Bedroom AbvGr',
       'Kitchen AbvGr', 'Kitchen Qual', 'TotRms AbvGrd', 'Functional',
       'Fireplaces', 'Garage Cars', 'Garage Area', 'Paved Drive',
       'Wood Deck SF', 'Open Porch SF', 'Enclosed Porch', '3Ssn Porch',
       'Screen Porch', 'Pool Area', 'Misc Val', 'Mo Sold', 'Yr Sold',
       'Sale Type', 'Sale Condition', 'SalePrice'],
      dtype='object')
```

In [215…

```
## Create new features
years_sold  = df['Yr Sold'] - df['Year Built']
df[years_sold <0]
```

Out[215…

	Order	PID	MS SubClass	MS Zoning	Lot Area	Street	Lot Shape	Land Contour	Utilities	Lot Config	Land Slope	Neighborhood	Condition 1	Condition 2	Bldg Type	Ho S
2180	2181	908154195	20	RL	39290	Pave	IR1	Bnk	AllPub	Inside	Gtl	Edwards	Norm	Norm	1Fam	1S

In [216…
```
years_since_remod = df['Yr Sold'] - df['Year Remod/Add']
years_since_remod[years_since_remod < 0]
```

Out[216…
```
1702   -1
2180   -2
2181   -1
dtype: int64
```

In [217…
```
## Create new columns
df['Years Before Sale'] = years_sold
df['Years Since Remod'] = years_since_remod
```

In [218…
```
# Drop rows with negative values for both of these new features
df =  df.drop([1702,2180,2181],axis=0)

## No longer need original year columns
df = df.drop(["Year Built", "Year Remod/Add"], axis = 1)
```

Drop columns that:

- that aren't useful for ML
- leak data about the final sale, read more about columns here

In [219…
```
## Drop columns that aren't useful for ML
# df = df.drop(["PID", "Order"], axis=1)

## Drop columns that leak info about the final sale
# df = df.drop(["Mo Sold", "Sale Condition", "Sale Type", "Yr Sold"], axis=1)
```

Let's update transform_features()

In [220…
```
def transform_features(df):
```

```
    num_missing = df.isnull().sum()
    drop_missing_cols = num_missing[(num_missing > len(df)/20)].sort_values()
    df = df.drop(drop_missing_cols.index, axis=1)

    text_mv_counts = df.select_dtypes(include=['object']).isnull().sum().sort_values(ascending=False)
    drop_missing_cols_2 = text_mv_counts[text_mv_counts > 0]
    df = df.drop(drop_missing_cols_2.index, axis=1)

    num_missing = df.select_dtypes(include=['int', 'float']).isnull().sum()
    fixable_numeric_cols = num_missing[(num_missing < len(df)/20) & (num_missing > 0)].sort_values()
    replacement_values_dict = df[fixable_numeric_cols.index].mode().to_dict(orient='records')[0]
    df = df.fillna(replacement_values_dict)

    years_sold = df['Yr Sold'] - df['Year Built']
    years_since_remod = df['Yr Sold'] - df['Year Remod/Add']
    df['Years Before Sale'] = years_sold
    df['Years Since Remod'] = years_since_remod
    df = df.drop([1702, 2180, 2181], axis=0)

    df = df.drop(["Year Built", "Year Remod/Add"], axis=1)
    return df

def select_features(df):
    return df[["Gr Liv Area", "SalePrice"]]

def train_and_test(df):
    train = df[:1460]
    test = df[1460:]

    ## You can use `pd.DataFrame.select_dtypes()` to specify column types
    ## and return only those columns as a data frame.
    numeric_train = train.select_dtypes(include=['integer', 'float'])
    numeric_test = test.select_dtypes(include=['integer', 'float'])

    ## You can use `pd.Series.drop()` to drop a value.
    features = numeric_train.columns.drop("SalePrice")
    lr = linear_model.LinearRegression()
    lr.fit(train[features], train["SalePrice"])
    predictions = lr.predict(test[features])
    mse = mean_squared_error(test["SalePrice"], predictions)
    rmse = np.sqrt(mse)
```

```
    return rmse

df = pd.read_csv("AmesHousing.txt", delimiter="\t")
transform_df = transform_features(df)
filtered_df = select_features(transform_df)
rmse = train_and_test(filtered_df)

rmse
```

Out[220… 55275.36731241307

Selection of features.

In [221…
```
df_nummeric = df.select_dtypes(include=["float","int"])
cormat = df_nummeric.corr()
sorted_corrs  = cormat['SalePrice'].abs()
sorted_corrs
```

Out[221…
```
Order              0.031408
PID                0.246521
MS SubClass        0.085092
Lot Frontage       0.357318
Lot Area           0.266549
Overall Qual       0.799262
Overall Cond       0.101697
Year Built         0.558426
Year Remod/Add     0.532974
Mas Vnr Area       0.508285
BsmtFin SF 1       0.432914
BsmtFin SF 2       0.005891
Bsmt Unf SF        0.182855
Total Bsmt SF      0.632280
1st Flr SF         0.621676
2nd Flr SF         0.269373
Low Qual Fin SF    0.037660
Gr Liv Area        0.706780
Bsmt Full Bath     0.276050
Bsmt Half Bath     0.035835
Full Bath          0.545604
Half Bath          0.285056
Bedroom AbvGr      0.143913
```

```
Kitchen AbvGr      0.119814
TotRms AbvGrd      0.495474
Fireplaces         0.474558
Garage Yr Blt      0.526965
Garage Cars        0.647877
Garage Area        0.640401
Wood Deck SF       0.327143
Open Porch SF      0.312951
Enclosed Porch     0.128787
3Ssn Porch         0.032225
Screen Porch       0.112151
Pool Area          0.068403
Misc Val           0.015691
Mo Sold            0.035259
Yr Sold            0.030569
SalePrice          1.000000
Name: SalePrice, dtype: float64
```

In [222…

```
## Let's only keep columns with a correlation coefficient of larger than 0.4 (arbitrary, worth experimenting later!)
sorted_corrs[sorted_corrs > 0.4]
```

Out[222…

```
Overall Qual      0.799262
Year Built        0.558426
Year Remod/Add    0.532974
Mas Vnr Area      0.508285
BsmtFin SF 1      0.432914
Total Bsmt SF     0.632280
1st Flr SF        0.621676
Gr Liv Area       0.706780
Full Bath         0.545604
TotRms AbvGrd     0.495474
Fireplaces        0.474558
Garage Yr Blt     0.526965
Garage Cars       0.647877
Garage Area       0.640401
SalePrice         1.000000
Name: SalePrice, dtype: float64
```

In [223…

```
## Drop columns with less than 0.35 correlation with SalePrice
transform_df = transform_df.drop(sorted_corrs[sorted_corrs < 0.35].index, axis=1)
```

In [224…

```
transform_df
```

Out[224…

	MS Zoning	Street	Lot Shape	Land Contour	Utilities	Lot Config	Land Slope	Neighborhood	Condition 1	Condition 2	Bldg Type	House Style	Overall Qual	Roof Style	Roof Matl	Ext
0	RL	Pave	IR1	Lvl	AllPub	Corner	Gtl	NAmes	Norm	Norm	1Fam	1Story	6	Hip	CompShg	Brk
1	RH	Pave	Reg	Lvl	AllPub	Inside	Gtl	NAmes	Feedr	Norm	1Fam	1Story	5	Gable	CompShg	Vir
2	RL	Pave	IR1	Lvl	AllPub	Corner	Gtl	NAmes	Norm	Norm	1Fam	1Story	6	Hip	CompShg	
3	RL	Pave	Reg	Lvl	AllPub	Corner	Gtl	NAmes	Norm	Norm	1Fam	1Story	7	Hip	CompShg	Brk
4	RL	Pave	IR1	Lvl	AllPub	Inside	Gtl	Gilbert	Norm	Norm	1Fam	2Story	5	Gable	CompShg	Vir
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2925	RL	Pave	IR1	Lvl	AllPub	CulDSac	Gtl	Mitchel	Norm	Norm	1Fam	SLvl	6	Gable	CompShg	HdB
2926	RL	Pave	IR1	Low	AllPub	Inside	Mod	Mitchel	Norm	Norm	1Fam	1Story	5	Gable	CompShg	HdB
2927	RL	Pave	Reg	Lvl	AllPub	Inside	Gtl	Mitchel	Norm	Norm	1Fam	SFoyer	5	Gable	CompShg	HdB
2928	RL	Pave	Reg	Lvl	AllPub	Inside	Mod	Mitchel	Norm	Norm	1Fam	1Story	5	Gable	CompShg	HdB
2929	RL	Pave	Reg	Lvl	AllPub	Inside	Mod	Mitchel	Norm	Norm	1Fam	2Story	7	Gable	CompShg	HdB

2927 rows × 41 columns

Which categorical columns should we keep?

In [225…

```
## Create a list of column names from documentation that are *meant* to be categorical
nominal_features = transform_df.select_dtypes(include=['object'])
nominal_features.columns
```

Out[225…

```
Index(['MS Zoning', 'Street', 'Lot Shape', 'Land Contour', 'Utilities',
       'Lot Config', 'Land Slope', 'Neighborhood', 'Condition 1',
       'Condition 2', 'Bldg Type', 'House Style', 'Roof Style', 'Roof Matl',
       'Exterior 1st', 'Exterior 2nd', 'Exter Qual', 'Exter Cond',
       'Foundation', 'Heating', 'Heating QC', 'Central Air', 'Kitchen Qual',
       'Functional', 'Paved Drive', 'Sale Type', 'Sale Condition'],
      dtype='object')
```

- Which columns are currently numerical but need to be encoded as categorical instead (because the numbers don't have any semantic meaning)?
- If a categorical column has hundreds of unique values (or categories), should we keep it? When we dummy code this column, hundreds of columns will need to be added back to the data frame.

number_unique = {} for col in nominal_features: x = transform_df[col].value_counts().sort_values() number_unique[col] = len(x) number_unique

In [226…

```
## How many unique values in each categorical column?
uniqueness_counts = transform_df[nominal_features.columns].apply(lambda x: len(x.value_counts())).sort_values()
uniqueness_counts
```

Out[226…

```
Street             2
Central Air        2
Utilities          3
Land Slope         3
Paved Drive        3
Exter Qual         4
Lot Shape          4
Land Contour       4
Heating QC         5
Bldg Type          5
Kitchen Qual       5
Lot Config         5
Exter Cond         5
Roof Style         6
Heating            6
Foundation         6
Sale Condition     6
MS Zoning          7
House Style        8
Condition 2        8
Functional         8
Roof Matl          8
Condition 1        9
Sale Type         10
Exterior 1st      16
Exterior 2nd      17
Neighborhood      28
dtype: int64
```

In [227…

```
## Aribtrary cutoff of 10 unique values (worth experimenting)

drop_uniqueness_counts = uniqueness_counts[uniqueness_counts > 10].index
```

```python
# removing category columns with more that 10 uniques.or Aribtrary cutoff of 10 unique values (worth experimenting)
transform_df = transform_df.drop(drop_uniqueness_counts, axis=1)
```

In [228…

```python
## Select just the remaining text columns and convert to categorical
text_cols = transform_df.select_dtypes(include=['object'])

for col in text_cols:
    transform_df[col] = transform_df[col].astype('category')

## Create dummy columns and add back to the dataframe!

transform_df = pd.concat([
    transform_df,
    pd.get_dummies(transform_df.select_dtypes(include=['category']))
], axis=1).drop(text_cols ,axis=1)
```

In [229…

```python
def transform_features(df):
    num_missing = df.isnull().sum()
    drop_missing_cols = num_missing[(num_missing > len(df)/20)].sort_values()
    df = df.drop(drop_missing_cols.index, axis=1)

    text_mv_counts = df.select_dtypes(include=['object']).isnull().sum().sort_values(ascending=False)
    drop_missing_cols_2 = text_mv_counts[text_mv_counts > 0]
    df = df.drop(drop_missing_cols_2.index, axis=1)

    num_missing = df.select_dtypes(include=['int', 'float']).isnull().sum()
    fixable_numeric_cols = num_missing[(num_missing < len(df)/20) & (num_missing > 0)].sort_values()
    replacement_values_dict = df[fixable_numeric_cols.index].mode().to_dict(orient='records')[0]
    df = df.fillna(replacement_values_dict)

    years_sold = df['Yr Sold'] - df['Year Built']
    years_since_remod = df['Yr Sold'] - df['Year Remod/Add']
    df['Years Before Sale'] = years_sold
    df['Years Since Remod'] = years_since_remod
    df = df.drop([1702, 2180, 2181], axis=0)

    df = df.drop(["PID", "Order", "Mo Sold", "Sale Condition", "Sale Type", "Year Built", "Year Remod/Add"], axis=1)
    return df
```

```python
def select_features(df, coeff_threshold=0.4, uniq_threshold=10):
    numerical_df = df.select_dtypes(include=['int', 'float'])
    abs_corr_coeffs = numerical_df.corr()['SalePrice'].abs().sort_values()
    df = df.drop(abs_corr_coeffs[abs_corr_coeffs < coeff_threshold].index, axis=1)

    nominal_features = ["PID", "MS SubClass", "MS Zoning", "Street", "Alley", "Land Contour", "Lot Config", "Neighbor
                    "Condition 1", "Condition 2", "Bldg Type", "House Style", "Roof Style", "Roof Matl", "Exterior 1s
                    "Exterior 2nd", "Mas Vnr Type", "Foundation", "Heating", "Central Air", "Garage Type",
                    "Misc Feature", "Sale Type", "Sale Condition"]

    transform_cat_cols = []
    for col in nominal_features:
        if col in df.columns:
            transform_cat_cols.append(col)

    uniqueness_counts = df[transform_cat_cols].apply(lambda col: len(col.value_counts())).sort_values()
    drop_nonuniq_cols = uniqueness_counts[uniqueness_counts > 10].index
    df = df.drop(drop_nonuniq_cols, axis=1)

    text_cols = df.select_dtypes(include=['object'])
    for col in text_cols:
        df[col] = df[col].astype('category')
    df = pd.concat([df, pd.get_dummies(df.select_dtypes(include=['category']))], axis=1).drop(text_cols,axis=1)

    return df

def train_and_test(df, k=0):
    numeric_df = df.select_dtypes(include=['integer', 'float'])
    features = numeric_df.columns.drop("SalePrice")
    lr = linear_model.LinearRegression()

    if k == 0:
        train = df[:1460]
        test = df[1460:]

        lr.fit(train[features], train["SalePrice"])
        predictions = lr.predict(test[features])
        mse = mean_squared_error(test["SalePrice"], predictions)
        rmse = np.sqrt(mse)

        return rmse
```

```
    if k == 1:
        # Randomize *all* rows (frac=1) from `df` and return
        shuffled_df = df.sample(frac=1, )
        train = df[:1460]
        test = df[1460:]

        lr.fit(train[features], train["SalePrice"])
        predictions_one = lr.predict(test[features])

        mse_one = mean_squared_error(test["SalePrice"], predictions_one)
        rmse_one = np.sqrt(mse_one)

        lr.fit(test[features], test["SalePrice"])
        predictions_two = lr.predict(train[features])

        mse_two = mean_squared_error(train["SalePrice"], predictions_two)
        rmse_two = np.sqrt(mse_two)

        avg_rmse = np.mean([rmse_one, rmse_two])
        print(rmse_one)
        print(rmse_two)
        return avg_rmse
    else:
        kf = KFold(n_splits=k, shuffle=True)
        rmse_values = []
        for train_index, test_index, in kf.split(df):
            train = df.iloc[train_index]
            test = df.iloc[test_index]
            lr.fit(train[features], train["SalePrice"])
            predictions = lr.predict(test[features])
            mse = mean_squared_error(test["SalePrice"], predictions)
            rmse = np.sqrt(mse)
            rmse_values.append(rmse)
        print(rmse_values)
        avg_rmse = np.mean(rmse_values)
        return avg_rmse

df = pd.read_csv("AmesHousing.txt", delimiter="\t")
transform_df = transform_features(df)
filtered_df = select_features(transform_df)
rmse = train_and_test(filtered_df, k=4)
```

```
rmse
```

[26487.56833342798, 26165.360413904215, 27438.093289726785, 36946.5979067728]

Out[229…

29259.404985957946

In [ ]: