Abaixo estarei mostrando como foi realizado o estudo de previsão de preço de aluguéis do dataset Housing Prices Competition que está presente no Kaggle pelo link abaixo e também explicarei o código que foi feito.

Kaggle Dataset: https://www.kaggle.com/c/home-data-for-ml-course/overview

1- Importando os arquivos necessários

Utilizei as seguintes bibliotecas do sklearn e as padrões para realizar o estudo:

```
import warnings
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pickle

from sklearn.preprocessing import MinMaxScaler, OrdinalEncoder
from sklearn.impute import SimpleImputer
from sklearn.feature_selection import SelectKBest
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score,
max_error

warnings.simplefilter("ignore")
pd.options.display.max_columns = None
```

2- Importação dos arquivos

Há dois arquivos na competição, o de treino e o de teste, no primeiro arquivo será importado apenas o de treino e no secundo arquivo será importado o de testes.
O arquivo está sendo importado, e caso não consiga ele entrará nas exceptions de arquivo não encontrado ou então na exceção geral.

```
try:
    arquivo_treino = pd.read_csv("train.csv", sep=",")

except FileNotFoundError:
    print("ARQUIVO NAO FOI ENCONTRADO")

except Exception as e:
    print(f"ERRO AO SUBIR ARQUIVO: {e}")
```

3- Limpeza e tratamento dos dados

Limpeza geral como substituir pontuação, remoção de acentos e pontos, e padronizar todo o dataset para Uppercase. Padrão que utilizo normalmente em meus datasets.

```
arquivo_treino = arquivo_treino.replace(".", "").replace(",", ".")
arquivo_treino.columns = arquivo_treino.columns.str.upper()

cols = arquivo_treino.select_dtypes(include=["object"]).columns
arquivo_treino[cols] = (arquivo_treino[cols].
                    apply(lambda x1: x1.str.normalize("NFKD").str.encode('ascii', errors='ignore').
                        str.decode('utf-8')))
```

4- Análises para poder tomada de decisão

Verificação das colunas que contém maior quantidade de valores nan e então retirada das que tem maiores valores, se por exemplo a coluna tem mais de 80% de valores nan então elas são desclassificadas e dropadas.
O dataset contém 1458 linhas.

```
for coluna in arquivo_treino:
    "print(arquivo_treino[coluna].isna().sum(), coluna)"

# COLUNAS QUE SE MOSTRARAM INVIÁVEIS PARA O MODELO:
# -> MISCFEATURE = 1406 NAN
# -> POOLQC = 1453 NAN
# -> FENCE = 1179 NAN
# -> ALLEY = 1369 NAN

arquivo_treino = arquivo_treino.drop(["MISCFEATURE", "POOLQC", "FENCE", "ALLEY"],
axis=1)
```

O mesmo acontece com todas as colunas as quais eu encontro valores únicos e retiro as que são acima de 80% do dataset também, excluindo várias colunas após.

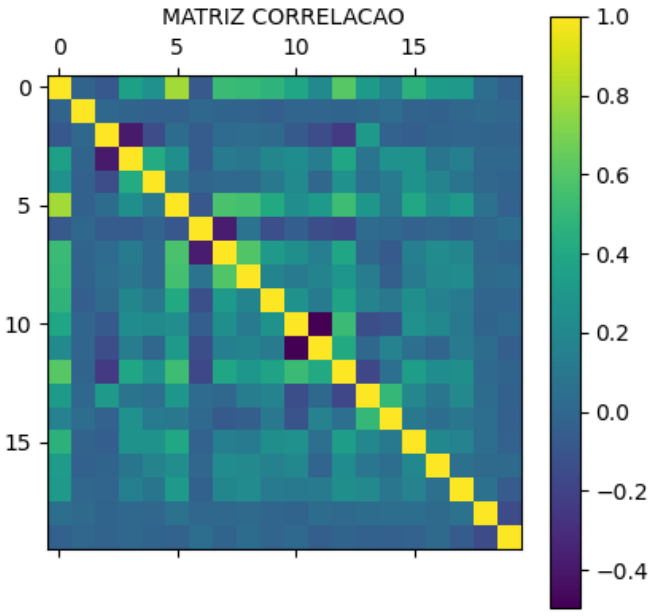
```
lista_dropagem = []

for coluna in arquivo_treino:
    if arquivo_treino[coluna].value_counts().max() >= 1200:
        lista_dropagem.append(coluna)

arquivo_treino = arquivo_treino.drop(lista_dropagem, axis=1)
```

5 - Remoção de colunas correlacionadas

É declarado um valor de corte de 0.60, então correlação acima desse número faz com que a coluna seja descartada.
Criada também uma função para remoção de correlações.

```
def remove_correlacao(arquivo, valor_corte):
    matriz_correlacao = arquivo.corr().abs()

    matriz_superior = matriz_correlacao.where(np.triu(np.ones(matriz_correlacao.shape), k=1).astype(np.bool))

    exclusao_correlacao = [coluna for coluna in matriz_superior.columns if any(matriz_superior[coluna] >= valor_corte)]

    return arquivo.drop(exclusao_correlacao, axis=1)


corte = 0.6

arquivo_sem_target = arquivo_treino.drop(["SALEPRICE"], axis=1)

arquivo_sem_target = remove_correlacao(arquivo_sem_target, corte)

arquivo_treino = pd.concat([arquivo_treino["SALEPRICE"], arquivo_sem_target], axis=1)
```

Foi feita também a tentativa de verificar por uma plotagem a correlação porém a visualização deixou a desejar.



6- Tratando valores Nan

Substituímos valores Nan em variáveis categóricas com sua frequência em que aparecem e para as variáveis numéricas substituímos com a mediana do resultado de cada coluna para evitar uma média muito distorcida do resultado.

```
num_att = arquivo_treino.select_dtypes(exclude=["object", "datetime"]).columns.to_list()
cat_att = arquivo_treino.select_dtypes(include=["object"]).columns.to_list()

imputer_mediana = SimpleImputer(strategy="median")
imputer_frequencia = SimpleImputer(strategy="most_frequent")

encoder = OrdinalEncoder()
scaler = MinMaxScaler()

for num in num_att:
   arquivo_treino[num] =
imputer_mediana.fit_transform(np.array(arquivo_treino[num]).reshape(-1, 1))
   arquivo_treino[num] = scaler.fit_transform(np.array(arquivo_treino[num]).reshape(-1, 1))

for cat in cat_att:
   arquivo_treino[cat] =
imputer_frequencia.fit_transform(np.array(arquivo_treino[cat]).reshape(-1, 1))
   arquivo_treino[cat] = encoder.fit_transform(np.array(arquivo_treino[cat]).reshape(-1, 1))

arquivo_treino.dropna()
```

7- Verificando as melhores colunas para serem utilizadas no modelo

Com a função de KBest podemos verificar as melhores colunas e sua pontuação para assim inserirmos no modelo.
Caso o modelo tenha algum resultado abaixo do esperado podemos então ir alterando as colunas até chegar num resultado satisfatório.

```
y = arquivo_treino["SALEPRICE"]
X = arquivo_treino.drop(["ID", "SALECONDITION", "YRSOLD", "MOSOLD", "YEARBUILT",
"SALEPRICE"], axis=1)

bestfeatures = SelectKBest(k=10)
fit = bestfeatures.fit(X, y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X.columns)

featureScores = pd.concat([dfcolumns, dfscores], axis=1)
featureScores.columns = ['Specs', 'Score']
"print(featureScores.nlargest(15, 'Score'))"
```

```
         Specs     Score
8   OVERALLQUAL  5.729183
```

```
16     EXTERQUAL  3.774880
3       LOTAREA  3.285852
18     BSMTQUAL  3.011938
27   KITCHENQUAL  2.904745
23   TOTALBSMTSF  2.315290
31  GARAGEFINISH  2.251428
15    MASVNRAREA  1.991329
10  YEARREMODADD  1.827603
1       MSZONING  1.742791
```

8- Escolha do melhor modelo

Criei uma função que mostra o resultado de cada algorítmo que eu colocar e assim pode me mostrar qual o melhor modelo que eu possa usar, com os dados de treino então o resultado pode ser conferido logo abaixo.

```
X = arquivo_treino[["BSMTQUAL", "TOTALBSMTSF", "GARAGEFINISH", "MSZONING",
                    "OPENPORCHSF", "WOODDECKSF", "FIREPLACEQU", "LOTFRONTAGE"]]

y = arquivo_treino["SALEPRICE"]

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1, shuffle=True)

def retorna_resultado_modelo(tipo_modelo_parametro):
    # REALIZANDO O FIT E PREDICT DO MODELO

    pred = tipo_modelo_parametro.fit(X_train, y_train).predict(X_test)

    # REALIZANDO TESTES DE DESEMPENHO DE MODELO

    """MEAN SQUARED ERROR"""
    mse = mean_squared_error(y_test, pred)

    """MEAN ABSOLUTE ERROR"""
    mae = mean_absolute_error(y_test, pred)

    """R2"""
    r2 = r2_score(y_test, pred)

    """MAX ERROR"""
    max_erro = max_error(y_test, pred)

    return print(f"MODELO {tipo_modelo_parametro}\nMSE: {mse}\nMAE: {mae}\nR2: {r2}\n"
                 f"MAX ERROR: {max_erro}\n")
```

```
lista_modelos = [RandomForestRegressor(random_state=1), LinearRegression(),
Ridge(solver="auto", alpha=1.0),
            DecisionTreeRegressor(max_depth=3, random_state=1)]

for tipo_modelo in lista_modelos:
   retorna_resultado_modelo(tipo_modelo)
```

O resultado de cada modelo pode ser observado abaixo:

MODELO RandomForestRegressor(random_state=1)
MSE: 0.004341464458756007
MAE: 0.04145714804776789
R2: 0.6646560526674739
MAX ERROR: 0.5304591306762951

MODELO LinearRegression()
MSE: 0.004362623425271584
MAE: 0.04518019136627292
R2: 0.6630216890972505
MAX ERROR: 0.5089827295896261

MODELO Ridge()
MSE: 0.0044519459891189245
MAE: 0.04498481295584546
R2: 0.6561222243127314
MAX ERROR: 0.5182650444236889

MODELO DecisionTreeRegressor(max_depth=3, random_state=1)
MSE: 0.004872093342161713
MAE: 0.048435952487546285
R2: 0.623669149280291
MAX ERROR: 0.4426985287131173

Pelos resultados pude concluir então que o melhor modelo a ser utilizado na base real seria o algoritmo de **Random Forest Regressor**, com altos valores de R2, MAX ERROR e baixo valor de MSE e MAE são também.

9- Treinamento do modelo

Feita a exportação do modelo preparado com a função pickle, assim podendo ser utilizado no segundo arquivo.

```
modelo = RandomForestRegressor(random_state=1).fit(X_train, y_train)

finalizado = "modelo_finalizado.sav"

pickle.dump(modelo, open(finalizado, "wb"))
```

10- Treinamento do arquivo de teste

No arquivo de teste é feito novamente o tratamento/limpeza dos dados do arquivo de teste, a normalização dos dados até chegar o momento do predict do modelo com o arquivo.

O resultado pode ser conferido abaixo:

arquivo_teste["RESULTADO"] = modelo_carregado.predict(X) * 10000

```
0       1511.796973
1       1589.168171
2       2605.189974
3       2142.230246
4       2600.944452
           ...
1454    1219.374624
1455    1198.127575
1456    2040.164422
1457    1507.035273
1458    2269.783225
Name: RESULTADO, Length: 1459, dtype: float64
```

11- Conclusão

Com o estudo pude verificar que o modelo trabalhou bem, e mudando um pouco as colunas e o tipo de modelo possa ter um resultado mais satisfatório. A média dos valores dos aluguéis combinam com os valores reais apresentados.