**Análise Exploratória de Dados (EDA)**

**Passos:**

1. **Carregar os dados**: Carregaremos os dados de train.csv.
2. **Visualizar os dados**: Obter uma visão geral do dataframe.
3. **Verificar valores ausentes**: Identificar se há valores ausentes nas colunas.
4. **Estatísticas Descritivas**: Obter estatísticas descritivas das variáveis.
5. **Análise de Correlação**: Calcular e visualizar a correlação entre as variáveis.

Vamos começar com essas etapas.

**Carregando os Dados e Explorando**

import pandas as pd

# Carregar os dados

train\_data = pd.read\_csv('/mnt/data/train.csv')

# Visualizar as primeiras linhas do dataframe

train\_data.head()

Vamos executar isso primeiro.

Os dados foram carregados com sucesso. A seguir, vamos explorar um pouco mais as características do dataframe:

1. Verificar a presença de valores ausentes.
2. Obter estatísticas descritivas das variáveis numéricas.
3. Calcular a correlação entre as variáveis numéricas e a variável SalePrice.

**Verificando Valores Ausentes**

# Verificar valores ausentes

missing\_values = train\_data.isnull().sum().sort\_values(ascending=False)

missing\_values = missing\_values[missing\_values > 0]

missing\_values

**Estatísticas Descritivas**

# Estatísticas descritivas das variáveis numéricas

desc\_stats = train\_data.describe()

desc\_stats

**Análise de Correlação**

# Calcular a correlação entre as variáveis numéricas e SalePrice

correlation\_matrix = train\_data.corr()

saleprice\_correlation = correlation\_matrix["SalePrice"].sort\_values(ascending=False)

saleprice\_correlation

Vamos executar essas etapas. ​​

**Resultados da Análise Exploratória de Dados (EDA)**

**1. Valores Ausentes**

As colunas com mais valores ausentes são:

* PoolQC: 1453 valores ausentes
* MiscFeature: 1406 valores ausentes
* Alley: 1369 valores ausentes
* Fence: 1179 valores ausentes
* FireplaceQu: 690 valores ausentes

Estas colunas têm uma quantidade significativa de valores ausentes, e devemos decidir como lidar com eles (remoção ou imputação).

**2. Estatísticas Descritivas**

As estatísticas descritivas mostram que as variáveis SalePrice, OverallQual, GrLivArea, GarageCars, e GarageArea têm valores que variam consideravelmente, indicando que podem ser características importantes para a previsão do preço.

**3. Análise de Correlação**

A correlação entre as variáveis numéricas e SalePrice revela que as seguintes variáveis têm a maior correlação positiva:

* OverallQual (0.79)
* GrLivArea (0.71)
* GarageCars (0.64)
* GarageArea (0.62)
* TotalBsmtSF (0.61)
* 1stFlrSF (0.61)
* FullBath (0.56)
* TotRmsAbvGrd (0.53)
* YearBuilt (0.52)
* YearRemodAdd (0.51)

Essas variáveis são candidatos potenciais para serem incluídos no modelo de regressão.

**Próximos Passos**

1. **Engenharia de Características**:
   * Lidar com valores ausentes.
   * Selecionar e criar novas características relevantes.
2. **Aprendizagem Supervisionada**:
   * Construir o modelo de regressão linear.
   * Criar um modelo de classificação.

Vamos começar com a engenharia de características para preparar os dados para o modelo de regressão.

**Engenharia de Características**

1. **Lidar com Valores Ausentes**: Podemos imputar ou remover colunas com muitos valores ausentes.
2. **Selecionar Variáveis Relevantes**: Usar variáveis com alta correlação com SalePrice.

**Lidando com Valores Ausentes**

Para simplicidade, vamos:

* Remover colunas com mais de 50% de valores ausentes.
* Imputar valores ausentes restantes com a mediana (para colunas numéricas) ou a moda (para colunas categóricas).

Vamos implementar essas etapas no código a seguir.

**Seleção de Variáveis**

Vamos selecionar variáveis com correlação alta com SalePrice para nosso modelo de regressão.

**Implementação**

# Remover colunas com mais de 50% de valores ausentes

threshold = len(train\_data) \* 0.5

train\_data = train\_data.drop(columns=missing\_values[missing\_values > threshold].index)

# Imputar valores ausentes nas colunas restantes

for column in train\_data.columns:

if train\_data[column].isnull().sum() > 0:

if train\_data[column].dtype == "object":

train\_data[column] = train\_data[column].fillna(train\_data[column].mode()[0])

else:

train\_data[column] = train\_data[column].fillna(train\_data[column].median())

# Selecionar variáveis relevantes

selected\_features = ["OverallQual", "GrLivArea", "GarageCars", "GarageArea", "TotalBsmtSF",

"1stFlrSF", "FullBath", "TotRmsAbvGrd", "YearBuilt", "YearRemodAdd"]

# Features e target

X = train\_data[selected\_features]

y = train\_data["SalePrice"]

# Dividir os dados em conjunto de treinamento e teste

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

Vamos executar essas etapas. ​​

Os dados foram preparados com sucesso e divididos em conjuntos de treinamento e teste. Agora vamos construir o modelo de regressão linear para prever o preço de venda das casas.

**Construindo o Modelo de Regressão Linear**

Vamos seguir os seguintes passos:

1. Treinar o modelo de regressão linear com o conjunto de treinamento.
2. Avaliar o desempenho do modelo no conjunto de teste.

**Implementação do Modelo de Regressão Linear**

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

# Instanciar o modelo de regressão linear

model = LinearRegression()

# Treinar o modelo

model.fit(X\_train, y\_train)

# Fazer previsões no conjunto de teste

y\_pred = model.predict(X\_test)

# Avaliar o desempenho do modelo

mse = mean\_squared\_error(y\_test, y\_pred)

rmse = mean\_squared\_error(y\_test, y\_pred, squared=False)

r2 = r2\_score(y\_test, y\_pred)

mse, rmse, r2

Vamos executar este código para treinar o modelo e avaliar seu desempenho. ​​

Parece que não consigo realizar uma análise de dados mais avançada no momento. Por favor, tente novamente mais tarde.

**Classificação**

Para avaliar o modelo de classificação, usaremos métricas como precisão, recall, F1-score e a matriz de confusão.

from sklearn.preprocessing import Binarizer

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix

# Converta a variável de saída em uma variável binária (Preço alto ou baixo)

threshold = data['SalePrice'].median()

y\_bin = (y > threshold).astype(int)

# Dividir os dados em treino e teste

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_bin, test\_size=0.2, random\_state=42)

# Treinar o modelo de regressão logística

clf = LogisticRegression()

clf.fit(X\_train, y\_train)

# Fazer previsões

y\_pred = clf.predict(X\_test)

# Calcular as métricas de avaliação

accuracy = accuracy\_score(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred)

recall = recall\_score(y\_test, y\_pred)

f1 = f1\_score(y\_test, y\_pred)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print(f"Accuracy: {accuracy}")

print(f"Precision: {precision}")

print(f"Recall: {recall}")

print(f"F1 Score: {f1}")

print("Confusion Matrix:")

print(conf\_matrix)

**4. Aprendizagem Não Supervisionada**

**a. Clusterização**

Vamos usar o algoritmo K-Means para identificar grupos de casas com características semelhantes. Para isso, precisamos selecionar algumas características relevantes para a clusterização.

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt

import seaborn as sns

# Selecionar algumas características relevantes para a clusterização

features = ['OverallQual', 'GrLivArea', 'GarageCars', 'TotalBsmtSF', 'FullBath']

X = data[features].dropna()

# Padronizar os dados

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Determinar o número ideal de clusters usando o método do cotovelo

wcss = []

for i in range(1, 11):

kmeans = KMeans(n\_clusters=i, random\_state=42)

kmeans.fit(X\_scaled)

wcss.append(kmeans.inertia\_)

# Plotar o método do cotovelo

plt.figure(figsize=(10, 6))

plt.plot(range(1, 11), wcss, marker='o')

plt.title('Método do Cotovelo')

plt.xlabel('Número de Clusters')

plt.ylabel('WCSS')

plt.show()

# Treinar o modelo K-Means com o número ideal de clusters (vamos supor 3 clusters)

kmeans = KMeans(n\_clusters=3, random\_state=42)

clusters = kmeans.fit\_predict(X\_scaled)

# Adicionar a coluna de clusters ao DataFrame original

data['Cluster'] = pd.Series(clusters, index=X.index)

# Visualizar os clusters

plt.figure(figsize=(10, 6))

sns.scatterplot(x='GrLivArea', y='SalePrice', hue='Cluster', data=data, palette='viridis')

plt.title('Clusters de Casas')

plt.xlabel('Área do Terreno')

plt.ylabel('Preço de Venda')

plt.show()

**b. Redução de Dimensionalidade**

Vamos usar a técnica PCA (Análise de Componentes Principais) para reduzir a dimensionalidade dos dados e visualizá-los em um espaço de menor dimensão.

from sklearn.decomposition import PCA

# Aplicar PCA

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(X\_scaled)

# Adicionar os componentes principais ao DataFrame original

data['PCA1'] = X\_pca[:, 0]

data['PCA2'] = X\_pca[:, 1]

# Visualizar os dados em 2D

plt.figure(figsize=(10, 6))

sns.scatterplot(x='PCA1', y='PCA2', hue='Cluster', data=data, palette='viridis')

plt.title('Redução de Dimensionalidade com PCA')

plt.xlabel('PCA1')

plt.ylabel('PCA2')

plt.show()

**c. Análise de Associação**

Vamos usar o algoritmo Apriori para identificar associações entre as características das casas.

from mlxtend.frequent\_patterns import apriori, association\_rules

# Selecionar algumas características binarizadas para a análise de associação

features\_bin = ['OverallQual', 'GrLivArea', 'GarageCars', 'TotalBsmtSF', 'FullBath']

data\_bin = data[features\_bin].applymap(lambda x: 1 if x > x.median() else 0)

# Aplicar o algoritmo Apriori

frequent\_itemsets = apriori(data\_bin, min\_support=0.1, use\_colnames=True)

# Gerar as regras de associação

rules = association\_rules(frequent\_itemsets, metric="lift", min\_threshold=1.0)

# Visualizar as regras de associação

rules.head()

**d. Análise de Outlier**

Vamos usar o algoritmo Local Outlier Factor (LOF) para identificar casas que podem ser consideradas outliers.

from sklearn.neighbors import LocalOutlierFactor

# Treinar o modelo LOF

lof = LocalOutlierFactor(n\_neighbors=20, contamination=0.05)

data['Outlier'] = lof.fit\_predict(X\_scaled)

# Visualizar os outliers

plt.figure(figsize=(10, 6))

sns.scatterplot(x='GrLivArea', y='SalePrice', hue='Outlier', data=data, palette='viridis')

plt.title('Análise de Outliers com LOF')

plt.xlabel('Área do Terreno')

plt.ylabel('Preço de Venda')

plt.show()