In [1]:
```
#Script Python para Teste de Modelos de Machine Learning
#Autor: Lucas Lai Barbosa

#Importação das bibliotecas necessárias
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import math
from sklearn.preprocessing import OrdinalEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import cross_val_score, RandomizedSearchCV, GridSearchCV
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.inspection import permutation_importance
```

In [2]:
```
#Carregamento do arquivo CSV gerado no "SCRIPT_ANALISE_DOS_DADOS"
df = pd.read_csv('CSV_PRODUTOS_LOJAS.csv', sep=',', decimal = '.', low_memory=False, encoding = 'utf8')
df
```

Out[2]:

	ID_PRODUTO	DEPARTAMENTO	MATERIAL_SEGMENTO	MATERIAL_SUBSEGMENTO	MATERIAL_GRUPO	MATERIAL_SUBGRUPO	Tipo	Subtipo	Ge
0	5054397004	CHINELO	FEMININO	FEMININO	FEMININO	CHL FEMININO	CHINELO	CASUAL	FEMIN
1	5054397004	CHINELO	FEMININO	FEMININO	FEMININO	CHL FEMININO	CHINELO	CASUAL	FEMIN
2	5054397004	CHINELO	FEMININO	FEMININO	FEMININO	CHL FEMININO	CHINELO	CASUAL	FEMIN
3	5054397004	CHINELO	FEMININO	FEMININO	FEMININO	CHL FEMININO	CHINELO	CASUAL	FEMIN
4	5054397004	CHINELO	FEMININO	FEMININO	FEMININO	CHL FEMININO	CHINELO	CASUAL	FEMIN
...	...	...	...	...	...	...	...	...	
114825	5170464041	TENIS ESPORTIVO	MASCULINO	TREINO	TREINO	TNS MASC TREINO	TENIS	TREINO	MASCUL
114826	5170464041	TENIS ESPORTIVO	MASCULINO	TREINO	TREINO	TNS MASC TREINO	TENIS	TREINO	MASCUL
114827	5170464041	TENIS ESPORTIVO	MASCULINO	TREINO	TREINO	TNS MASC TREINO	TENIS	TREINO	MASCUL
114828	5170464041	TENIS ESPORTIVO	MASCULINO	TREINO	TREINO	TNS MASC TREINO	TENIS	TREINO	MASCUL
114829	5170464041	TENIS ESPORTIVO	MASCULINO	TREINO	TREINO	TNS MASC TREINO	TENIS	TREINO	MASCUL

114830 rows × 31 columns

In [3]:
```
#Exclusão das colunas irrelevantes
df2 = df.drop(columns=["ID_LOJA","ID_PRODUTO","DEPARTAMENTO","MATERIAL_SEGMENTO","MATERIAL_SUBSEGMENTO","MATERIAL_GRUPO",\
                       "MATERIAL_SUBGRUPO","MATERIAL_COR","CIDADE","UF","QTD_PRODUTOS_VENDIDOS"])
```

In [4]:
```
#Transformação das colunas categóricas binárias e categóricas ordinais em valores numéricos

cat_gen = ["MASCULINO","FEMININO"]
cat_idade = ["BEBE", "INFANTIL", "JUVENIL", "ADULTO"]
cat_marca = ["TERCEIRO", "PROPRIA"]
cat_local = ["Rua","Shopping"]
cat_pe = ["PEQUENO","GRANDE"]
cat_clima = ["Fria","Quente"]
cat_hl = ["Low","High"]
cat_cluster = ["C","B","A"]
oe1 = OrdinalEncoder(categories=[cat_gen, cat_idade, cat_marca, cat_local, cat_pe, cat_clima, cat_hl, cat_cluster])
df2[["Genero", "Idade", "MATERIAL_MARCA", "Rua/Shopping","TAMANHO_PE","Temp_loja","Cluster_low_high", "CLUSTER"]] = \
    oe1.fit_transform(df2[["Genero", "Idade", "MATERIAL_MARCA", "Rua/Shopping","TAMANHO_PE","Temp_loja","Cluster_low_high", "CLUSTER"]])
df2
```

Out[4]:

	Tipo	Subtipo	Genero	Idade	COR_H	COR_S	COR_B	MATERIAL_MARCA	REGIAO	POPULACAO	ANOS_DESDE_INAUG	AREA	Rua/Shopping	CLUS
0	CHINELO	CASUAL	1.0	3.0	51.0	1.0	1.0	0.0	Centro-Oeste	623614	39.0	884	0.0	
1	CHINELO	CASUAL	1.0	3.0	51.0	1.0	1.0	0.0	Centro-Oeste	623614	24.0	778	0.0	
2	CHINELO	CASUAL	1.0	3.0	51.0	1.0	1.0	0.0	Centro-Oeste	290383	11.0	1713	0.0	
3	CHINELO	CASUAL	1.0	3.0	51.0	1.0	1.0	0.0	Centro-Oeste	916001	21.0	1345	0.0	
4	CHINELO	CASUAL	1.0	3.0	51.0	1.0	1.0	0.0	Centro-Oeste	623614	21.0	1125	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
114825	TENIS	TREINO	0.0	3.0	0.0	0.0	0.0	0.0	Norte	548952	13.0	1309	1.0	
114826	TENIS	TREINO	0.0	3.0	0.0	0.0	0.0	0.0	Sul	366418	11.0	1197	1.0	
114827	TENIS	TREINO	0.0	3.0	0.0	0.0	0.0	0.0	Sul	140597	10.0	1978	0.0	
114828	TENIS	TREINO	0.0	3.0	0.0	0.0	0.0	0.0	Norte	419452	8.0	1144	1.0	
114829	TENIS	TREINO	0.0	3.0	0.0	0.0	0.0	0.0	Sul	22466	3.0	1258	0.0	

114830 rows × 20 columns

In [5]:
```
#Transformação das colunas categóricas nominais em colunas binárias (uma para cata categoria)
pd.set_option('display.max_columns', None)
X = pd.get_dummies(df2.iloc[:,0:-1])

y = df2.iloc[:,-1]
```

In [6]:
```
#Divisão da base em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=23, test_size=0.20)
```

In [7]:
```
#Transformação das escalas das colunas da base de treino em valores entre 0 e 1
scaler = MinMaxScaler()

colunas_scale = ["Idade", "COR_H", "POPULACAO", "ANOS_DESDE_INAUG","AREA","CLUSTER","TEMPO_EM_LOJA","PRECO_UNIT"]
X_train_sc = X_train.copy()
X_train_sc[colunas_scale] = scaler.fit_transform(X_train_sc[colunas_scale])
X_train_sc.describe()
```

Out[7]:

	Genero	Idade	COR_H	COR_S	COR_B	MATERIAL_MARCA	POPULACAO	ANOS_DESDE_INAUG	AREA	Rua/Shoppin
count	91864.000000	91864.000000	91864.000000	91864.000000	91864.000000	91864.000000	91864.000000	91864.000000	91864.000000	91864.00000
mean	0.758828	0.930539	0.201964	0.266829	0.518759	0.585420	0.290449	0.219146	0.411249	0.63955
std	0.427797	0.222711	0.331063	0.386714	0.449645	0.492652	0.282183	0.162362	0.181823	0.48013
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
25%	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.075455	0.102564	0.291091	0.00000
50%	1.000000	1.000000	0.000000	0.020000	0.550000	1.000000	0.179976	0.179487	0.366359	1.00000
75%	1.000000	1.000000	0.171429	0.530000	1.000000	1.000000	0.515646	0.307692	0.528418	1.00000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00000

In [8]:
```
#Transformação das escalas da base de teste na mesma escala da base de treino
X_test_sc = X_test.copy()
X_test_sc[colunas_scale] = scaler.transform(X_test_sc[colunas_scale])
X_test_sc.describe()
```

Out[8]:

	Genero	Idade	COR_H	COR_S	COR_B	MATERIAL_MARCA	POPULACAO	ANOS_DESDE_INAUG	AREA	Rua/Shoppin
count	22966.000000	22966.000000	22966.000000	22966.000000	22966.000000	22966.000000	22966.000000	22966.000000	22966.000000	22966.00000
mean	0.756292	0.930767	0.199234	0.269823	0.517763	0.582731	0.289018	0.218865	0.412330	0.63524
std	0.429328	0.222055	0.326993	0.389674	0.450177	0.493119	0.281666	0.161104	0.182004	0.48137
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
25%	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.075455	0.102564	0.289555	0.00000
50%	1.000000	1.000000	0.000000	0.020000	0.550000	1.000000	0.171676	0.179487	0.366359	1.00000
75%	1.000000	1.000000	0.171429	0.530000	1.000000	1.000000	0.515646	0.307692	0.528418	1.00000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00000

In [9]:
```
#Teste com Regressão Linear Múltipla

linreg = LinearRegression().fit(X_train_sc, y_train)

#Validação cruzada com base de treino
print('--- Reg. Linear - Validação Cruzada com base de treino ---')
cv_r2_linreg = cross_val_score(linreg, X_train_sc, y_train, cv=5, scoring='r2')
print("R2 Médio da Validação Cruzada: {:.4f}".format(cv_r2_linreg.mean()))

cv_mse_linreg = cross_val_score(linreg, X_train_sc, y_train, cv=5, scoring='neg_mean_absolute_error')
print("MAE Médio da Validação Cruzada: {:.4f}".format(-cv_mse_linreg.mean()))

#Validação do modelo
print('\n--- Reg. Linear - Validação do Modelo ---')
print("R2 na base de treinamento: {:.4f}".format(linreg.score(X_train_sc, y_train)))
print("R2 na base de teste: {:.4f}".format(linreg.score(X_test_sc, y_test)))

y_predicted_linreg = linreg.predict(X_train_sc)
print("\nMAE na base de treinamento: {:.4f}".format(mean_absolute_error(y_train, y_predicted_linreg)))
y_predicted_test_linreg = linreg.predict(X_test_sc)
print("MAE na base de teste: {:.4f}".format(mean_absolute_error(y_test, y_predicted_test_linreg)))
```

```
--- Reg. Linear - Validação Cruzada com base de treino ---
R2 Médio da Validação Cruzada: 0.6288
MAE Médio da Validação Cruzada: 297.7803

--- Reg. Linear - Validação do Modelo ---
R2 na base de treinamento: 0.6291
R2 na base de teste: 0.6253

MAE na base de treinamento: 297.5644
MAE na base de teste: 298.3612
```

In [10]:
```
# Teste com Regressão Ridge

#Ajuste de Hiperparâmetros com Grid Search
param_var = {'alpha': [0.01, 0.1, 1, 10, 100]}
```

```
rid = GridSearchCV(estimator=Ridge(), param_grid=param_var, scoring='r2', cv=5, refit=True)
rid.fit(X_train_sc, y_train)
print('--- Ridge - GridSearch ---')
print('Melhor Hiperparâmetro: '+str(rid.best_params_))
print("R2 do melhor estimador: {:.4f}".format(rid.best_score_))

#Validação do modelo
print('\n--- Ridge - Validação do Modelo ---')
print("R2 na base de treinamento: {:.4f}".format(rid.score(X_train_sc, y_train)))
print("R2 na base de teste: {:.4f}".format(rid.score(X_test_sc, y_test)))

y_predicted_rid = rid.predict(X_train_sc)
print("\nMAE na base de treinamento: {:.4f}".format(mean_absolute_error(y_train, y_predicted_rid)))
y_predicted_test_rid = rid.predict(X_test_sc)
print("MAE na base de teste: {:.4f}".format(mean_absolute_error(y_test, y_predicted_test_rid)))
```

```
--- Ridge - GridSearch ---
Melhor Hiperparâmetro: {'alpha': 0.1}
R2 do melhor estimador: 0.6289

--- Ridge - Validação do Modelo ---
R2 na base de treinamento: 0.6291
R2 na base de teste: 0.6253

MAE na base de treinamento: 297.5615
MAE na base de teste: 298.3588
```

In [11]:
```
#Teste com Regressão Lasso

#Ajuste de Hiperparâmetros com Grid Search
param_var = {'alpha': [0.1, 1, 10, 100]}
las = GridSearchCV(estimator=Lasso(), param_grid=param_var, scoring='r2', cv=5, refit=True)
las.fit(X_train_sc, y_train)
print('--- Lasso - GridSearch ---')
print('Melhor Hiperparâmetro: '+str(las.best_params_))
print("R2 do melhor estimador: {:.4f}".format(las.best_score_))

#Validação do modelo
print('\n--- Lasso - Validação do Modelo ---')
print("R2 na base de treinamento: {:.4f}".format(las.score(X_train_sc, y_train)))
print("R2 na base de teste: {:.4f}".format(las.score(X_test_sc, y_test)))

y_predicted_las = las.predict(X_train_sc)
print("\nMAE na base de treinamento: {:.4f}".format(mean_absolute_error(y_train, y_predicted_las)))
y_predicted_test_las = las.predict(X_test_sc)
print("MAE na base de teste: {:.4f}".format(mean_absolute_error(y_test, y_predicted_test_las)))
```

```
--- Lasso - GridSearch ---
Melhor Hiperparâmetro: {'alpha': 0.1}
R2 do melhor estimador: 0.6288

--- Lasso - Validação do Modelo ---
R2 na base de treinamento: 0.6291
R2 na base de teste: 0.6253

MAE na base de treinamento: 297.3416
MAE na base de teste: 298.1085
```

In [12]:
```
#Teste com K-NN Regressor

#Ajuste de Hiperparâmetros com Random Search
param_var = {'n_neighbors': [4, 8, 12, 16], 'p':[1, 2]}
knn = RandomizedSearchCV(estimator=KNeighborsRegressor(), param_distributions=param_var, scoring='r2', cv=3, refit=True,\
                         n_iter=5, random_state=23)
knn.fit(X_train_sc, y_train)
print('--- K-NN - RandomSearch ---')
print('Melhores Hiperparâmetros: '+str(knn.best_params_))
print("R2 do melhor estimador: {:.4f}".format(knn.best_score_))

#Validação do modelo
score_train_knn = knn.score(X_train_sc, y_train)
score_test_knn = knn.score(X_test_sc, y_test)
print('\n--- K-NN - Validação do Modelo ---')
print("R2 na base de treinamento: {:.4f}".format(score_train_knn))
print("R2 na base de teste: {:.4f}".format(score_test_knn))

y_predicted_knn = knn.predict(X_train_sc)
print("\nMAE na base de treinamento: {:.4f}".format(mean_absolute_error(y_train, y_predicted_knn)))
y_predicted_test_knn = knn.predict(X_test_sc)
print("MAE na base de teste: {:.4f}".format(mean_absolute_error(y_test, y_predicted_test_knn)))
```

```
--- K-NN - RandomSearch ---
Melhores Hiperparâmetros: {'p': 1, 'n_neighbors': 8}
R2 do melhor estimador: 0.6259

--- K-NN - Validação do Modelo ---
R2 na base de treinamento: 0.7305
R2 na base de teste: 0.6464

MAE na base de treinamento: 264.3031
MAE na base de teste: 303.6376
```

In [13]:
```
#Teste com Decision Tree Regressor

#Ajuste de Hiperparâmetros com Grid Search
param_var = {'min_samples_split': [2, 20, 40, 80, 120, 160], 'min_samples_leaf': [1, 10, 20, 40, 60]}
dtr = GridSearchCV(estimator=DecisionTreeRegressor(), param_grid=param_var, scoring='r2', cv=5, refit=True)
dtr.fit(X_train_sc, y_train)
print('--- Árvore de Decisão - GridSearch ---')
```

```
print('Melhores Hiperparâmetros: '+str(dtr.best_params_))
print("R2 do melhor estimador: {:.4f}".format(dtr.best_score_))
```

```
--- Árvore de Decisão - GridSearch ---
Melhores Hiperparâmetros: {'min_samples_leaf': 20, 'min_samples_split': 120}
R2 do melhor estimador: 0.7007
```

In [14]:
```
#Decision Tree Regressor - Validação do modelo
print('--- Árvore de Decisão - Validação do Modelo ---')
print("R2 na base de treinamento: {:.4f}".format(dtr.score(X_train_sc, y_train)))
print("R2 na base de teste: {:.4f}".format(dtr.score(X_test_sc, y_test)))

y_predicted_dtr = dtr.predict(X_train_sc)
print("\nMAE na base de treinamento: {:.4f}".format(mean_absolute_error(y_train, y_predicted_dtr)))
y_predicted_test_dtr = dtr.predict(X_test_sc)
print("MAE na base de teste: {:.4f}".format(mean_absolute_error(y_test, y_predicted_test_dtr)))
```

```
--- Árvore de Decisão - Validação do Modelo ---
R2 na base de treinamento: 0.7394
R2 na base de teste: 0.7029

MAE na base de treinamento: 246.7782
MAE na base de teste: 265.1887
```

In [15]:
```
#Teste com Random Forest Regressor

#Ajuste de Hiperparâmetros com Random Search
param_var = {'min_samples_split':[20, 40, 80, 120], 'n_estimators':[10, 50, 100], 'min_samples_leaf': [1, 10, 20]}
rfr = RandomizedSearchCV(estimator=RandomForestRegressor(), param_distributions=param_var, scoring='r2', cv=3, refit=True,\
                         n_iter=5, random_state=23)
rfr.fit(X_train_sc, y_train)
print('--- Random Forest - RandomSearch ---')
print('Melhores Hiperparâmetros: '+str(rfr.best_params_))
print("R2 do melhor estimador: {:.4f}".format(rfr.best_score_))
```

```
--- Random Forest - RandomSearch ---
Melhores Hiperparâmetros: {'n_estimators': 100, 'min_samples_split': 20, 'min_samples_leaf': 10}
R2 do melhor estimador: 0.7374
```

In [16]:
```
#Random Forest Regressor - Validação do modelo
print('--- Random Forest - Validação do Modelo ---')
print("R2 na base de treinamento: {:.4f}".format(rfr.score(X_train_sc, y_train)))
print("R2 na base de teste: {:.4f}".format(rfr.score(X_test_sc, y_test)))

y_predicted_rfr = rfr.predict(X_train_sc)
print("\nMAE na base de treinamento: {:.4f}".format(mean_absolute_error(y_train, y_predicted_rfr)))
y_predicted_test_rfr = rfr.predict(X_test_sc)
print("MAE na base de teste: {:.4f}".format(mean_absolute_error(y_test, y_predicted_test_rfr)))
```

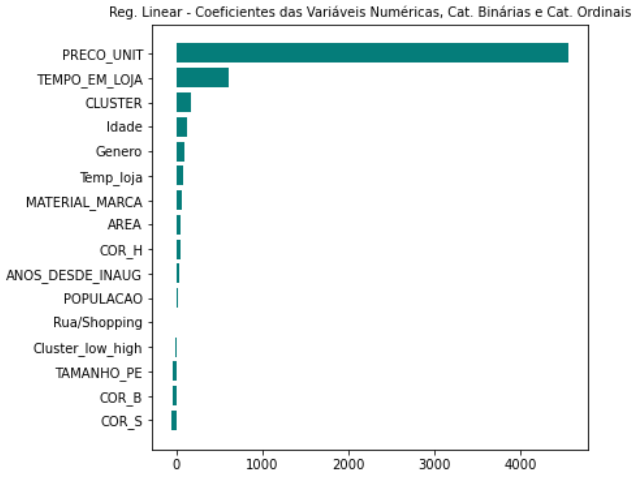
```
--- Random Forest - Validação do Modelo ---
R2 na base de treinamento: 0.8199
R2 na base de teste: 0.7451

MAE na base de treinamento: 204.9909
MAE na base de teste: 248.1406
```

In [18]:
```
#Importância das variáveis na Regressão Linear
importancia_lr = pd.DataFrame(data={'Variável': X_train.columns[:16],'Importância': linreg.coef_[:16]})
importancia_lr.sort_values(by=['Importância'], inplace=True, ignore_index=True)

#Gráfico dos coeficientes
fig = plt.figure(figsize=(6, 6))
fig.add_subplot(111)
plt.barh(y=importancia_lr['Variável'], width=importancia_lr['Importância'], color='#057D7A')
plt.title('Reg. Linear - Coeficientes das Variáveis Numéricas, Cat. Binárias e Cat. Ordinais', size=10)
plt.show()

#Valores dos coeficientes das colunas numéricas, categóricas binárias e categóricas ordinais
print('COEFICIENTES DAS VARIÁVEIS - REGRESSÃO LINEAR')
for i in range(15,-1,-1):
    print(importancia_lr['Variável'][i]+': %.0f' % (importancia_lr['Importância'][i]))
```



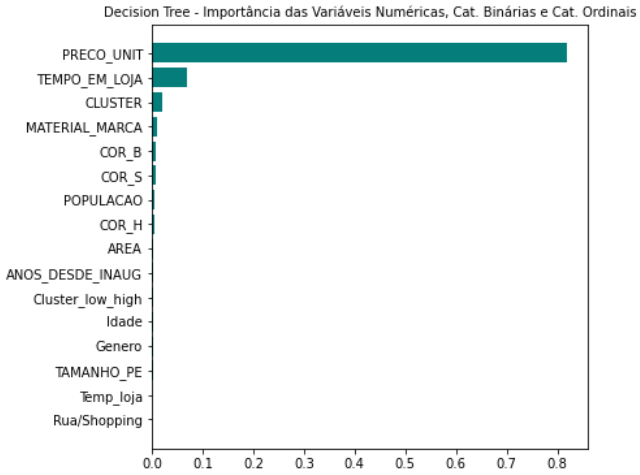
```
COEFICIENTES DAS VARIÁVEIS - REGRESSÃO LINEAR
PRECO_UNIT: 4556
TEMPO_EM_LOJA: 607
CLUSTER: 167
Idade: 132
Genero: 100
Temp_loja: 82
MATERIAL_MARCA: 67
AREA: 60
COR_H: 48
ANOS_DESDE_INAUG: 36
POPULACAO: 23
Rua/Shopping: 6
Cluster_low_high: -6
TAMANHO_PE: -32
COR_B: -34
COR_S: -47
```

In [19]:
```
#Importância das variáveis na Árvore de Decisão - Feature Importance
dtr2 = DecisionTreeRegressor(random_state=23, min_samples_split = 120, min_samples_leaf = 20)
dtr2.fit(X_train_sc, y_train)

importancia_dt = pd.DataFrame(data={'Variável': X_train.columns[:16],'Importância': dtr2.feature_importances_[:16]})
importancia_dt.sort_values(by=['Importância'], inplace=True, ignore_index=True)

#Gráfico dos scores de importância
fig = plt.figure(figsize=(6, 6))
fig.add_subplot(111)
plt.barh(y=importancia_dt['Variável'], width=importancia_dt['Importância'], color='#057D7A')
plt.title('Decision Tree - Importância das Variáveis Numéricas, Cat. Binárias e Cat. Ordinais', size=10)
plt.show()

#Valores dos scores das colunas numéricas, categóricas binárias e categóricas ordinais
print('SCORES DAS VARIÁVEIS - ÁRVORE DE DECISÃO')
for i in range(15,-1,-1):
    print(importancia_dt['Variável'][i]+': %.3f' % (importancia_dt['Importância'][i]))
```



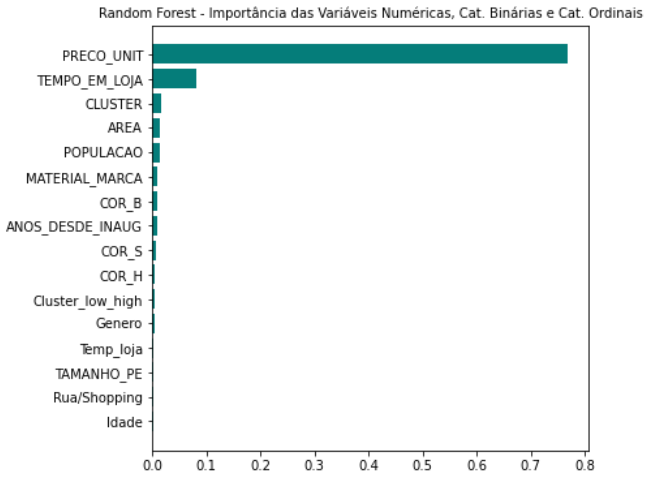

```
SCORES DAS VARIÁVEIS - ÁRVORE DE DECISÃO
PRECO_UNIT: 0.819
TEMPO_EM_LOJA: 0.069
CLUSTER: 0.019
MATERIAL_MARCA: 0.011
COR_B: 0.008
COR_S: 0.006
POPULACAO: 0.004
COR_H: 0.004
AREA: 0.003
ANOS_DESDE_INAUG: 0.002
Cluster_low_high: 0.002
Idade: 0.001
Genero: 0.001
TAMANHO_PE: 0.001
Temp_loja: 0.000
Rua/Shopping: 0.000
```

In [20]:
```
#Importância das variáveis no Random Forest Regressor
rfr2 = RandomForestRegressor(random_state=23, n_estimators=100, min_samples_split=20, min_samples_leaf=10)
rfr2.fit(X_train_sc, y_train)

importancia_rf = pd.DataFrame(data={'Variável': X_train.columns[:16],'Importância': rfr2.feature_importances_[:16]})
importancia_rf.sort_values(by=['Importância'], inplace=True, ignore_index=True)

#Gráfico dos scores de importância
fig = plt.figure(figsize=(6, 6))
fig.add_subplot(111)
plt.barh(y=importancia_rf['Variável'], width=importancia_rf['Importância'], color='#057D7A')
plt.title('Random Forest - Importância das Variáveis Numéricas, Cat. Binárias e Cat. Ordinais', size=10)
plt.show()

#Valores dos scores das colunas numéricas, categóricas binárias e categóricas ordinais
print('SCORES DAS VARIÁVEIS - RANDOM FOREST')
for i in range(15,-1,-1):
    print(importancia_rf['Variável'][i]+': %.3f' % (importancia_rf['Importância'][i]))
```



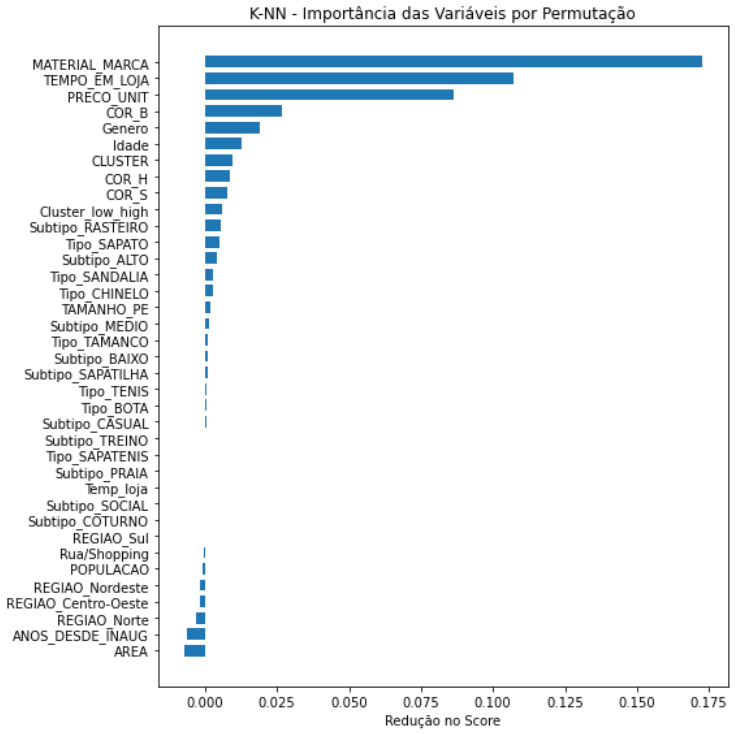

```
SCORES DAS VARIÁVEIS - RANDOM FOREST
PRECO_UNIT: 0.768
TEMPO_EM_LOJA: 0.082
CLUSTER: 0.017
AREA: 0.014
POPULACAO: 0.014
MATERIAL_MARCA: 0.009
COR_B: 0.009
ANOS_DESDE_INAUG: 0.008
COR_S: 0.008
COR_H: 0.005
Cluster_low_high: 0.004
Genero: 0.003
Temp_loja: 0.003
TAMANHO_PE: 0.002
Rua/Shopping: 0.002
Idade: 0.001
```

In [21]:
```python
#Importância das variáveis no K-NN - Permutation Importance
knn2 = KNeighborsRegressor(n_neighbors=8, p=1)
knn2.fit(X_train_sc, y_train)

perm_import = permutation_importance(knn2, X_test_sc, y_test, n_repeats=3, random_state=23)
index_classif = perm_import.importances_mean.argsort()
```

In [22]:
```python
fig, ax1 = plt.subplots(1, figsize=(8, 8))
ax1.barh(X_test_sc.columns[index_classif], perm_import.importances_mean[index_classif], height=0.7)
ax1.set_title("K-NN - Importância das Variáveis por Permutação")
ax1.set_xlabel("Redução no Score")
ax1.figure.tight_layout()

plt.show()
```



In [23]:
```python
#Random Forest - Gráfico comparando y previsto com y real
fig, ax = plt.subplots(figsize=(7, 7))
ax.scatter(y_predicted_test_rfr, y_test, alpha=0.7, color='#45ced0')

#Adição de título nomes dos eixos
ax.set_xlabel('Y Previsto')
```

```
ax.set_ylabel('Y Real')
ax.set_title('Análise de Resultados - Random Forest na Base de Teste')

#Ajuste da escala dos eixos
ax.set_xlim(xmin=0, xmax=12000)
ax.set_ylim(ymin=0, ymax=12000)

#Remoção das barras superior e direita
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

#Adição de gridlines
ax.grid(color='grey', linestyle='-', linewidth=0.25, alpha=0.5)

#Adição de linha de referência
plt.plot([0, 15000], [0, 15000], alpha=0.8, color='#f87833')

plt.show()
```

In [24]:
```
#Carregamento dos dados fictícios para simulação
df_sim = pd.read_excel('BD_SIMULACAO.xlsx')
df_sim.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40 entries, 0 to 39
Data columns (total 39 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Produto_Desc          40 non-null     object
 1   Loja_Desc             40 non-null     object
 2   Genero                40 non-null     int64
 3   Idade                 40 non-null     int64
 4   COR_H                 40 non-null     int64
 5   COR_S                 40 non-null     float64
 6   COR_B                 40 non-null     int64
 7   MATERIAL_MARCA        40 non-null     int64
 8   POPULACAO             40 non-null     int64
 9   ANOS_DESDE_INAUG      40 non-null     int64
 10  AREA                  40 non-null     int64
 11  Rua/Shopping          40 non-null     int64
 12  CLUSTER               40 non-null     int64
 13  TAMANHO_PE            40 non-null     int64
 14  Temp_loja             40 non-null     int64
 15  Cluster_low_high      40 non-null     int64
 16  TEMPO_EM_LOJA         40 non-null     int64
 17  PRECO_UNIT            40 non-null     float64
 18  Tipo_BOTA             40 non-null     int64
 19  Tipo_CHINELO          40 non-null     int64
 20  Tipo_SANDALIA         40 non-null     int64
 21  Tipo_SAPATENIS        40 non-null     int64
 22  Tipo_SAPATO           40 non-null     int64
 23  Tipo_TAMANCO          40 non-null     int64
 24  Tipo_TENIS            40 non-null     int64
 25  Subtipo_ALTO          40 non-null     int64
 26  Subtipo_BAIXO         40 non-null     int64
 27  Subtipo_CASUAL        40 non-null     int64
 28  Subtipo_COTURNO       40 non-null     int64
 29  Subtipo_MEDIO         40 non-null     int64
 30  Subtipo_PRAIA         40 non-null     int64
 31  Subtipo_RASTEIRO      40 non-null     int64
 32  Subtipo_SAPATILHA     40 non-null     int64
 33  Subtipo_SOCIAL        40 non-null     int64
 34  Subtipo_TREINO        40 non-null     int64
 35  REGIAO_Centro-Oeste   40 non-null     int64
 36  REGIAO_Nordeste       40 non-null     int64
 37  REGIAO_Norte          40 non-null     int64
 38  REGIAO_Sul            40 non-null     int64
dtypes: float64(2), int64(35), object(2)
memory usage: 12.3+ KB
```

In [25]:
```
#Transformação das escalas das colunas da base de simulação em valores entre 0 e 1
```

```
X_sim = df_sim.copy()
X_sim[colunas_scale] = scaler.transform(X_sim[colunas_scale])
X_sim.describe()
```

Out[25]:

	Genero	Idade	COR_H	COR_S	COR_B	MATERIAL_MARCA	POPULACAO	ANOS_DESDE_INAUG	AREA	Rua/Shopping	CLUSTER	TAMA
count	40.000000	40.000000	40.000000	40.000000	40.000000	40.000000	40.000000	40.000000	40.000000	40.00000	40.000000	
mean	0.600000	0.733333	0.337143	0.250000	0.600000	0.600000	0.293254	0.163462	0.395449	0.50000	0.500000	
std	0.496139	0.330759	0.430117	0.392232	0.496139	0.496139	0.316593	0.166749	0.164638	0.50637	0.358057	
min	0.000000	0.333333	0.000000	0.000000	0.000000	0.000000	0.028511	0.000000	0.165131	0.00000	0.000000	
25%	0.000000	0.333333	0.000000	0.000000	0.000000	0.000000	0.064574	0.044872	0.262673	0.00000	0.375000	
50%	1.000000	1.000000	0.000000	0.000000	1.000000	1.000000	0.150520	0.102564	0.417819	0.50000	0.500000	
75%	1.000000	1.000000	0.685714	0.250000	1.000000	1.000000	0.389808	0.217949	0.530338	1.00000	0.625000	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	0.931571	0.461538	0.632873	1.00000	1.000000	

In [26]:

```
#Previsão do Valor Total Vendido usando o Modelo de Random Forest
y_sim_pred = rfr.predict(X_sim.iloc[:,2:])
df_sim_result = pd.concat([X_sim, pd.DataFrame(y_sim_pred, columns=['VALOR_TOTAL_VENDIDO_PRED'])],\
                          axis=1)

#Gravação dos resultados em arquivo de Excel
df_sim_result.to_excel('RESULTADOS_SIMULACAO.xlsx', index=False)
```