# LIBRARIES

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

from matplotlib import pyplot as plt

import pandas as pd

import numpy as np

import seaborn as sns

from wordcloud import WordCloud, STOPWORDS

from collections import Counter

import scipy

%config InlineBackend.figure\_format='retina'

%matplotlib inline

import warnings

from sklearn import tree

from statsmodels.formula.api import ols

from statsmodels.stats.anova import anova\_lm

from statsmodels.stats import multicomp

from sklearn.preprocessing import LabelEncoder

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

from sklearn.externals.six import StringIO

from IPython.display import Image

from sklearn.tree import export\_graphviz

import pydotplus

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsRegressor

from sklearn import tree

from sklearn.metrics import mean\_squared\_error

from keras.models import Sequential

from keras import optimizers

from keras.layers import Dense

from keras.wrappers.scikit\_learn import KerasRegressor

from keras import backend as K

LOADING DATASET

# Snapshot of the dataset

span\_cars = pd.read\_csv('data.csv',sep=';')

span\_cars.head(3)

span\_cars = pd.read\_csv('data.csv',sep=';')

span\_cars.shape

PART-1 – CLEANING DATA

#Explore nulls

span\_cars.columns

print(span\_cars.isnull().sum())

#Fill some nulls with unknown for these two categories (I am using these for my analysis)

span\_cars['gear\_type'].fillna(value='unknown',inplace=True)

span\_cars['fuel\_type'].fillna(value='unknown',inplace=True)

print(span\_cars.isnull().sum())

#Drop num\_owners, sales\_type (useless columns for analysis & too many nulls)

span\_cars.drop(columns=['num\_owners','sale\_type','version'],inplace=True)

#Drop Duplicates

span\_cars.drop\_duplicates(inplace=True)

#Drop nulls for remaining columns

span\_cars.dropna(inplace=True)

print(span\_cars.isnull().sum())

PART-02 - EXPLORATORY ANALYSIS & PREPROCESSING

# Used car analysis, remove brand new cars

span = span\_cars[(span\_cars['kms'] > 0)]

span['months\_old'].astype(int)

twelve = np.ceil((span.months\_old/12)).astype(float) # round up some are less than 1 month old

span\_new = pd.merge(span,twelve.to\_frame(),left\_index=True,right\_index=True)

span\_new.drop(columns=['months\_old\_x'],inplace=True)

span\_new.rename(columns={'months\_old\_y':'age'}, inplace=True) # age is the year of model of the vehicle

span\_new = span\_new[(span\_new['age'] < 32.0)]

year = (2018-span\_new.age).to\_frame()

span\_new = pd.merge(span,year,left\_index=True,right\_index=True)

#span\_new is the new dataset with the appropriate age column

print(span\_new.age.unique())

DISTRIBUTION PLOTS

# Plotting out distributions for the numeric variables to see whether they are normally distributed

warnings.filterwarnings("ignore")

sns.set()

fig, ((ax1,ax2),(ax3,ax4))=plt.subplots(ncols=2,nrows=2,figsize=(6,6))

sns.distplot(span\_new['age'],ax=ax1);

sns.distplot(span\_new['power'],ax=ax2);

sns.distplot(span\_new['kms'],ax=ax3);

sns.distplot(span\_new['price'],ax=ax4);

FREQUENCY PLOTS

categories = ['make','fuel\_type', 'gear\_type'] # categorical features used in my analysis

ranges = [0,1,2,3,4]

counts = []

x = []

y = []

for i,j in zip(ranges,categories):

z = span\_new.groupby([j])['ID'].count().sort\_values(ascending=False).reset\_index()

counts.append(z)

x.append(counts[i][j])

y.append(counts[i]['ID'])

plt.figure()

sns.barplot(x[i],y[i])

plt.ylabel('Frequency')

plt.xticks(rotation=90)

FURTHER CLEANING & PREPROCESSING

# Based off the above graphs, a few things caught my attention in the fuel\_type section (Etanol & CNG)

print(span\_new.groupby(['fuel\_type'])['ID'].count().sort\_values(ascending=True).reset\_index())

# It seems like a safe option to drop Etanol and CNG

et = span\_new[(span\_new['fuel\_type']=='etanol')].index

cng = span\_new[(span\_new['fuel\_type']=='CNG')].index

span\_new.drop(et,axis=0,inplace=True) # etanol

span\_new.drop(cng,axis=0,inplace=True) # CNG

print(span\_new.groupby(['fuel\_type'])['ID'].count().sort\_values(ascending=True).reset\_index())

CLEANING GEAR TYPE

#Not Very Infrequent, So I am going to keep the semi-automatic rows

print(span\_new.groupby(['gear\_type'])['ID'].count().sort\_values(ascending=True).reset\_index())

CLEANING PRICE

span\_new = span\_new.drop(span\_new[span\_new.price>400000].index)

INTEGRATING MORE DATA - COUNTRIES

# I just add countries to my data to increase the amount of data and do some more analysis at the country-level.

#Here, each car maker gets a corresponding country assigned.

Germany = {'Audi','Bmw','Mercedes-Benz','Skoda','Porsche','Smart','Volkswagen','Opel'}

Italy = {'Fiat','Iveco','Alfa'}

USA = {'Ford','Jeep','Chrysler','Chevrolet'}

Japan = {'Honda','Nissan','Suzuki','Ssangyong','Hyundai','Mitsubishi','Mazda','Toyota','Kia'}

Spain ={'Seat'}

Romania= {'Dacia'}

France={'Renault','Peugeot','Citroen','Ds'}

Sweden = {'Volvo','Saab'}

UK = {'Land','Jaguar','Mini','Bentley'}

mydata = {'Germany':Germany, 'Italy':Italy, 'USA':USA, "Japan":Japan, 'Spain':Spain, 'Romania':Romania, 'France':France,\

'Sweden':Sweden, 'UK':UK}

mydicts = [{(z,i) for z in j} for i,j in mydata.items()]

mydict = {}

[mydict.update(i) for i in mydicts]

span\_new = pd.read\_csv('Span\_new.csv')

span\_new['Country'] = span\_new['make'].map(mydict)

PART 3 - VISUALIZATION

# Prepare columns for barplot

span\_new = span\_new.drop(span\_new[span\_new.price>400000].index)

yearly\_price = span\_new.groupby(['age'])['price'].mean().reset\_index()

sem\_price = span\_new.groupby(['age'])['price'].sem().reset\_index()

plt.figure(figsize=(10, 8));

sns.set\_style("ticks", {"xtick.major.size": 16, "ytick.major.size":8});

sns.set(font\_scale=1.1)

fig = sns.barplot(x=yearly\_price['age'].astype(int),y= yearly\_price['price'], yerr=sem\_price['price'],capsize=4,errwidth=3,palette="Blues\_d")

plt.ylabel('Price: Euros',fontsize=16);

plt.xlabel('Model Year',fontsize=16);

plt.xticks(rotation=90)

plt.title('Average Used Car Prices Per Model Year',fontsize=22,fontweight='bold');

#sns.savefig("Average Used Car Price Per Year.png")

plt.savefig('pricect.png', dpi=400)

print(span\_new[(span\_new['age']<1995)].groupby('age')['ID'].count().reset\_index())

WHAT ARE THE CARS IN 1988 THAT ARE MAKING THE BARPLOT LOOK LIKE THIS ?

span\_new[(span\_new['age']== 1988)]

sem\_price = span\_new.groupby(['Country'])['price'].sem().reset\_index()

plt.figure(figsize=(10, 8));

sns.set\_style("ticks", {"xtick.major.size": 16, "ytick.major.size":8});

sns.set(font\_scale=1.1)

fig = sns.barplot(x=span\_new['Country'],y= span\_new['price'],yerr=sem\_price['price'],errwidth=3,palette="Blues\_d")

plt.ylabel('Price: Euros',fontsize=16);

plt.xlabel('Country',fontsize=16);

plt.xticks()

plt.title('Price per Country',fontsize=22,fontweight='bold');

ANOVA (Analysis of Variance)

cprice = span\_new[['Country','price']]

cprice\_lm = ols('price ~ Country', data=cprice).fit()

print(anova\_lm(cprice\_lm))

df = multicomp.pairwise\_tukeyhsd(cprice['price'], cprice['Country']).summary()

print(df)

DISTRIBUTION OF MILEAGE PER MAKE --> ARE SOME CARS DRIVEN MORE THAN OTHERS?

sns.set(font\_scale = 1.5)

plt.figure(figsize=(17, 10));

plot = sns.boxplot(x='kms',y='make',data=span\_new,notch=True,orient='h',palette="coolwarm",showfliers=False)

plt.xlabel('Mileage',fontsize=17,fontweight="bold")

plt.ylabel('Car Maker',fontsize=17,fontweight="bold")

plt.title('Distribution of Mileage per Car Maker',fontsize=22,fontweight="bold")

plt.xlim()

plt.savefig('mileagect.png')

plt.show()

mprice = span\_new[['make','kms']]

mprice\_lm = ols('kms ~ make', data=mprice).fit()

print(anova\_lm(mprice\_lm))

tukey\_mprice = multicomp.pairwise\_tukeyhsd(mprice['kms'], mprice['make']).summary()

print(tukey\_mprice)

tukey\_df = pd.DataFrame(tukey\_mprice.data,columns=['group1','group2','meandiff','lower','upper','reject'])

tukey\_df[((tukey\_df['group1']== 'Iveco') | (tukey\_df['group2']== 'Iveco'))]

plt.figure(figsize=(17, 10));

sns.boxplot(x='Country',y='kms',data=span\_new,notch=True,palette="coolwarm",showfliers=False)

plt.xlabel('Countries',fontsize=14,fontweight="bold")

plt.ylabel('Mileage',fontsize=14,fontweight="bold")

plt.title('Distribution of Mileage per Country of Manufacturer',fontsize=18,fontweight="bold")

plt.xlim()

plt.show()

ANOVA – Analysis of Variance

cprice = span\_new[['Country','kms']]

cprice\_lm = ols('kms ~ Country', data=cprice).fit()

print(anova\_lm(cprice\_lm))

print(multicomp.pairwise\_tukeyhsd(cprice['kms'], cprice['Country']).summary())

WHAT MAKES OF CARS HAVE THE HIGHEST HORSEPOWER? IS THE POWER ASSOCIATED WITH PRICE?

sns.set(font\_scale = 1.5)

plt.figure(figsize=(17, 10));

sns.boxplot(x='power',y='make',data=span\_new,notch=True,orient='h',palette='coolwarm',showfliers=False)

plt.xlabel('HorsePower',fontsize=17,fontweight="bold")

plt.ylabel('Car Maker',fontsize=17,fontweight="bold")

plt.title('Distribution of Horsepower per Car Maker',fontsize=22,fontweight="bold")

plt.xlim(0,500)

plt.show()

COUNTRY LEVEL ANALYSIS OF HORSEPOWER

sns.set(style='dark')

plt.figure(figsize=(15, 10));

sns.boxplot(x='Country',y='power',data=span\_new,showfliers=False,hue='Country')

plt.xlabel('Countries',fontsize=16,fontweight="bold")

plt.ylabel('Horsepower',fontsize=16,fontweight="bold")

plt.title('Distribution of Horsepower per Country of Manufacturer',fontsize=18,fontweight="bold")

plt.xticks(fontsize=15)

plt.yticks(fontsize=15)

plt.xlim()

plt.savefig('hpct.png',dpi=400)

plt.show()

pprice = span\_new[['Country','power']]

pprice\_lm = ols('power ~ Country', data=pprice).fit()

print(anova\_lm(pprice\_lm))

pt = multicomp.pairwise\_tukeyhsd(pprice['power'], pprice['Country']).summary()

ower\_tukey = pd.DataFrame(pt.data,columns=['group1','group2','meandiff','lower','upper','reject'])

power\_tukey[((power\_tukey['group1']== 'UK') | (power\_tukey['group2']== 'UK'))]

WHAT CARS ARE MOST FREQUENT IN THE DATASET?

from PIL import Image

#Some fun Visualizations

#Generate frequencies using counter

freqs = Counter(span\_new['make'])

df = pd.DataFrame.from\_dict(freqs, orient='index').reset\_index()

df.columns = ['brands','freqs']

#Generate image

car\_mask = np.array(Image.open('/Users/nadimyounes/Downloads/octavia.jpg'))

# Generate a word cloud with freqs

wc = WordCloud(background\_color="white", max\_words=1000, mask=car\_mask,contour\_width=2,contour\_color='black')

wc.generate\_from\_frequencies(freqs)

plt.figure(figsize=(8, 8))

plt.axis("off")

make = span\_new.groupby(['make'])['ID'].count().sort\_values(ascending=False).reset\_index()

plt.savefig('wordcloudcar.png')

plt.imshow(wc, interpolation='bilinear');

sns.set(font\_scale = 1.25)

plt.figure(figsize=(10, 9));

sns.set\_style("ticks",{"xtick.major.size": 12, "ytick.major.size":8})

make = span\_new.groupby(['make'])['ID'].count().sort\_values(ascending=False).reset\_index();

sns.barplot(make['ID'],make['make'],orient='h',color="#2874A6");

plt.title("Number of Cars Per Car Maker",fontsize=22)

plt.ylabel('Car Maker',fontsize=16.5)

plt.xlabel('Number of Cars',fontsize=16.5)

plt.xticks(fontsize=15)

plt.yticks(fontsize=13)

plt.savefig('numberofcarzz.png')

plt.show()

ASSESSING THE CORRELATION COEFFICIENTS OF THE NUMERIC VARIABLES

numeric\_corr = span\_new[['price','power','age','kms']].corr(method='spearman'); # non-parametric

sns.heatmap(numeric\_corr,annot=True);

plt.title('Used Car Dataset');

PART 4 - MACHINE LEARNING MODELS

span\_X = span\_new[['make','model','age','gear\_type','fuel\_type','power','kms','Country']]

le\_features = span\_new[['power','age','kms']]

# Dummy Categorical Variables

span\_X = pd.concat([span\_X,pd.get\_dummies(span\_X['make'],drop\_first=True,prefix="Make")],axis=1)

span\_X= pd.concat([span\_X,pd.get\_dummies(span\_X['model'],drop\_first=True,prefix="Model")],axis=1)

span\_X = pd.concat([span\_X,pd.get\_dummies(span\_X['gear\_type'],drop\_first=True,prefix="Gear")],axis=1)

span\_X = pd.concat([span\_X,pd.get\_dummies(span\_X['fuel\_type'],drop\_first=True,prefix="Fuel")],axis=1)

span\_X= pd.concat([span\_X,pd.get\_dummies(span\_X['Country'],drop\_first=True,prefix="Country")],axis=1)

# Remove Original Make, Model .. categorical variable columns

features\_final = span\_X.drop(columns=['make','model','gear\_type','fuel\_type','Country'])

# Subsetted out categorical variables in case I need to use them another time

cat = features\_final.drop(columns=['kms','age','power'])

DECISION TREE REGRESSOR

X\_train, X\_test, y\_train, y\_test = train\_test\_split(features\_final, span\_new['price'], test\_size=0.33, random\_state=42)

my\_DT = tree.DecisionTreeRegressor(max\_depth=3)

my\_DT.fit(X\_train, y\_train)

y\_pred\_dt = my\_DT.predict(X\_test)

print('The Score on the test set with a basic decision tree regressor is:',my\_DT.score(X\_test,y\_test))

print("The Root Mean squared error with a basic decision tree regressor: %.2f"

% np.sqrt(mean\_squared\_error(y\_test, y\_pred\_dt)))

dot\_data = StringIO()

export\_graphviz(my\_DT, out\_file=dot\_data,feature\_names=features\_final.columns,

filled=True, rounded=True,

special\_characters=True)

graph = pydotplus.graph\_from\_dot\_data(dot\_data.getvalue())

Image(graph.create\_png())

trknn\_scores=[]

teknn\_scores= []

rmse\_scores=[]

for i in np.arange(1,20,1):

my\_DT = tree.DecisionTreeRegressor(max\_depth=i,random\_state=42)

my\_DT.fit(X\_train, y\_train)

y\_pred\_dt = my\_DT.predict(X\_test)

train\_scores = my\_DT.score(X\_train,y\_train)

test\_scores = my\_DT.score(X\_test,y\_test)

# The Root mean squared error

trknn\_scores.append(train\_scores)

teknn\_scores.append(test\_scores)

rmse\_scores.append(np.sqrt(mean\_squared\_error(y\_test, y\_pred\_dt)))

sns.set\_style('whitegrid')

fig,(ax1,ax2)=plt.subplots(ncols=2,figsize=(7,4));

ax1.plot(np.arange(1,20,1),rmse\_scores,color='red');

ax2.plot(np.arange(1,20,1),trknn\_scores);

ax2.plot(np.arange(1,20,1),teknn\_scores);

ax1.set\_xlabel('Max Depth of Tree')

ax2.set\_xlabel('Max Depth of Tree')

ax1.set\_ylabel('Root Mean Squared Error')

ax2.set\_ylabel('R-squared');

sns.set\_style('whitegrid')

fig,(ax1,ax2)=plt.subplots(ncols=2,figsize=(7,4));

ax1.plot(np.arange(1,20,1),rmse\_scores,color='red');

ax2.plot(np.arange(1,20,1),trknn\_scores);

ax2.plot(np.arange(1,20,1),teknn\_scores);

ax1.set\_xlabel('Max Depth of Tree')

ax2.set\_xlabel('Max Depth of Tree')

ax1.set\_ylabel('Root Mean Squared Error')

ax2.set\_ylabel('R-squared');

from sklearn import tree

from sklearn.metrics import mean\_squared\_error

X\_train, X\_test, y\_train, y\_test = train\_test\_split(features\_final, span\_new['price'], test\_size=0.33, random\_state=42)

my\_DT = tree.DecisionTreeRegressor(max\_depth=8,random\_state=42)

my\_DT.fit(X\_train, y\_train)

y\_pred\_dt = my\_DT.predict(X\_test)

print('The Score on the training set with a decision tree regressor is:',my\_DT.score(X\_train,y\_train))

print('The Score on the test set with a decision tree regressor is:',my\_DT.score(X\_test,y\_test))

print("Mean squared error: %.2f"

% np.sqrt(mean\_squared\_error(y\_test, y\_pred\_dt)))

KNN REGRESSOR

KNR = KNeighborsRegressor()

trknn\_scores=[]

teknn\_scores= []

rmse\_scores=[]

for i in np.arange(1,100,5):

KNR = KNeighborsRegressor(n\_neighbors=i)

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

KNR.score(X\_test,y\_test)

train\_scores = KNR.score(X\_train,y\_train)

test\_scores = KNR.score(X\_test,y\_test)

# The Root mean squared error

trknn\_scores.append(train\_scores)

teknn\_scores.append(test\_scores)

y\_pred\_knn = KNR.predict(X\_test)

rmse\_scores.append(np.sqrt(mean\_squared\_error(y\_test, y\_pred\_knn)))

sns.set\_style('whitegrid')

fig,(ax1,ax2)=plt.subplots(ncols=2,figsize=(7,4))

ax1.plot(np.arange(1,100,5),rmse\_scores,color='red')

ax2.plot(np.arange(1,100,5),trknn\_scores)

ax2.plot(np.arange(1,100,5),teknn\_scores)

ax1.set\_xlabel('Number of Neighbors')

ax2.set\_xlabel('Number of Neighbors')

ax1.set\_ylabel('Root Mean Squared Error')

ax2.set\_ylabel('R-squared')

KNR = KNeighborsRegressor(n\_neighbors=6)

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

y\_pred\_KNR = KNR.predict(X\_test)

print('The Score on the test set with a KNN regressor is:',KNR.score(X\_test,y\_test))

print("Mean squared error: %.2f"

% np.sqrt(mean\_squared\_error(y\_test, y\_pred\_KNR)))

RANDOM FOREST REGRESSOR

#Random Forest Regressor

from sklearn.ensemble import RandomForestRegressor

rand\_est = RandomForestRegressor()

rand\_est.fit(X\_train,y\_train)

y\_pred = rand\_est.predict(X\_test)

print(' The Score on the test set with a random forest regressor is:', rand\_est.score(X\_test,y\_test))

#pretty good score dude

# The Root mean squared error

print("Mean squared error: %.2f"

% np.sqrt(mean\_squared\_error(y\_test, y\_pred)))

RFR = RandomForestRegressor()

# Number of trees in random forest

n\_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]

# Number of features to consider at every split

max\_features = ['auto', 'sqrt']

# Maximum number of levels in tree

max\_depth = [int(x) for x in np.linspace(10, 110, num = 11)]

max\_depth.append(None)

# Minimum number of samples required to split a node

min\_samples\_split = [2, 5, 10]

# Minimum number of samples required at each leaf node

min\_samples\_leaf = [1, 2, 4]

# Method of selecting samples for training each tree

bootstrap = [True, False]

# Create the random grid

random\_grid = {'n\_estimators': n\_estimators,

'max\_features': max\_features,

'max\_depth': max\_depth,

'min\_samples\_split': min\_samples\_split,

'min\_samples\_leaf': min\_samples\_leaf,

'bootstrap': bootstrap}

rf\_random = RandomizedSearchCV(estimator = RFR, param\_distributions = random\_grid, n\_iter = 100, cv = 3, verbose=2, random\_state=42, n\_jobs = -1)

# Fit the random search model

rf\_random.fit(X\_train, y\_train)

print('These are the best parameters from the randomized search:{max\_depth: 40, n\_estimators: 600, min\_samples\_split: 10, min\_samples\_leaf: 1, bootstrap: False, max\_features: sqrt}')

#Try out hyperparameters that were optimized before

from sklearn.ensemble import RandomForestRegressor

X\_train, X\_test, y\_train, y\_test = train\_test\_split(features\_final, span\_new['price'], test\_size=0.33, random\_state=42)

rand\_est = RandomForestRegressor(max\_depth=40, n\_estimators= 600, min\_samples\_split= 10, min\_samples\_leaf= 1, bootstrap= False, max\_features= 'sqrt')

rand\_est.fit(X\_train,y\_train)

y\_pred\_rfr = rand\_est.predict(X\_test)

print(' The Score on the train set with a hyperparameter optimized random forest regressor is:',rand\_est.score(X\_train,y\_train))

print(' The Score on the test set with a hyperparameter optimized random forest regressor is:',rand\_est.score(X\_test,y\_test))

print("Mean squared error: %.2f"

% np.sqrt(mean\_squared\_error(y\_test, y\_pred\_rfr)))

feature\_importances = pd.DataFrame(rand\_est.feature\_importances\_)

print(feature\_importances.sort\_values(by=0,ascending=False).head(10))

pd.Series(x).reset\_index()[0]

for i in feature\_importances.sort\_values(by=0,ascending=False).tail(5).index:

print(features\_final.columns[i])

XGBOOST REGRESSOR

xgbr = XGBRegressor()

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

y\_pred\_xgbr = xgbr.predict(X\_test)

print(xgbr)

print('The Score on the test set with an XGboost Regressor is:',xgbr.score(X\_test,y\_test))

print("Mean squared error: %.2f", np.sqrt(mean\_squared\_error(y\_test,y\_pred\_xgbr)))

#Let's Optimize the XGBOOST - Run this overnight (I did this and it failed to return scores higher than the score above) )

from sklearn.model\_selection import RandomizedSearchCV

from xgboost import XGBRegressor

xgbr = XGBRegressor()

max\_depth = [3,5,10]

n\_estimators = [500,600,800]

learning\_rate = [0.15,0.3]

params = {

#Hypeparameter tuning

'max\_depth':max\_depth,

'n\_estimators':n\_estimators,

'learning\_rate':learning\_rate}

xgbr\_random = RandomizedSearchCV(estimator = xgbr, param\_distributions = params, n\_iter = 4, cv = 3, verbose=2, random\_state=42, n\_jobs = -1)

# Fit the random search model

xgbr\_random.fit(X\_train, y\_train)

print('These are the best parameters from the randomized search (Xgboost) :{max\_depth: 5, n\_estimators: 500, min\_samples\_split: 10, learning\_rate=0.2}')

# Using the best parameters from randomized search

xgbr = XGBRegressor(learning\_rate= 0.2, max\_depth= 5, n\_estimators= 500)

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

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

print(xgbr)

print('The Score on the test set with an XGboost Regressor is:',xgbr.score(X\_test,y\_test))

print("Mean squared error: %.2f", np.sqrt(mean\_squared\_error(y\_test,y\_pred)))

MODEL COMPARISONS

# Plot Model Results

R\_Squared = [0.69,0.84,0.92,0.93]

Model = ['KNN Regressor','Decision Tree Regressor','XgBoost Regressor','Random Forest Regressor']

RMSE = [7771.91,5590,3980,3702]

f, (ax1,ax2) = plt.subplots(ncols=2, figsize=(18, 6));

plt.subplots\_adjust(left=None, bottom=None, right=None, top=None, wspace=0.48, hspace=None)

sns.set(style="white");

sns.barplot(y=Model, x=R\_Squared,orient='h',label="R-Squared",ax=ax1,color='#4a998a');

ax1.set\_xlabel("R-Squared",fontweight='bold',fontsize=16);

ax1.set\_yticklabels(Model,fontweight='bold',fontsize=12)

ax1.set\_xlim(0,1.1)

sns.barplot(y=Model, x=RMSE ,orient='h',label="Root Mean Squared Error",ax=ax2,color='#da5f28');

ax2.set\_yticklabels(Model,fontweight='bold',fontsize=12)

ax2.set\_xlabel('Root Mean Squared Error (RMSE)',fontweight='bold',fontsize=16);

ax2.set\_xlim(0,10000)

plt.savefig('ctscores2.png',dpi=400)

PLOTTING RESIDUALS VS FITTED VALUES

##KNN regressor

def residuals(prediction,colz,title):

sns.set\_style("white")

Residual = y\_test - prediction

plt.scatter(prediction, Residual,color=str(colz))

plt.tick\_params(axis='both', which='major', labelsize=10)

plt.xlabel('Predicted Prices',fontsize=15)

plt.ylabel('Residual (Errors)',fontsize=15)

plt.axhline(y=0,color=str(colz))

plt.title('Residuals vs Fitted Plot '+ str((title)),fontsize=18)

x = plt.scatter(prediction,Residual,color=str(colz))

return x

residuals(y\_pred\_KNR,'purple','KNN Regressor')

### Decision Tree Residuals

residuals(y\_pred\_dt,'green','Decision Tree Regressor')

## XgBoost Regression

residuals(y\_pred\_xgbr,'blue','XgBoost Regressor')

## Random Forest Regressor

residuals(y\_pred\_rfr,'orange','Random Forest Regressor')