Assignment Lesson 12, 13, 14

- Time Series Forecasting, Outlier Detection, Feature Selection -

In this assignment, you will try to perform a simple time series forecasting and also perform some advanced data preparation tasks using outlier detection and feature selection.

Part A: Time Series Forecasting on Beer Production

Prior Knowledge

Data Science Objective

Objective

The beer industry contributes significantly to the Australian economy, with the national total contribution in 2018 hitting 4.6 billion Australian dollars. Craft beer is increasing in popularity and availability in Australia, accounting for around 23% of the beer market.

This data shows quarterly Australian beer production from 1956-2010. It has four columns: Time, Year, Quarter, and Beer Production. Using this dataset, we are wondering if we can make a forecast for future beer production

Source: https://www.kaggle.com/mpwolke/australian-monthly-beer-production

Link

Rapidminer

Beer Dataset

Python

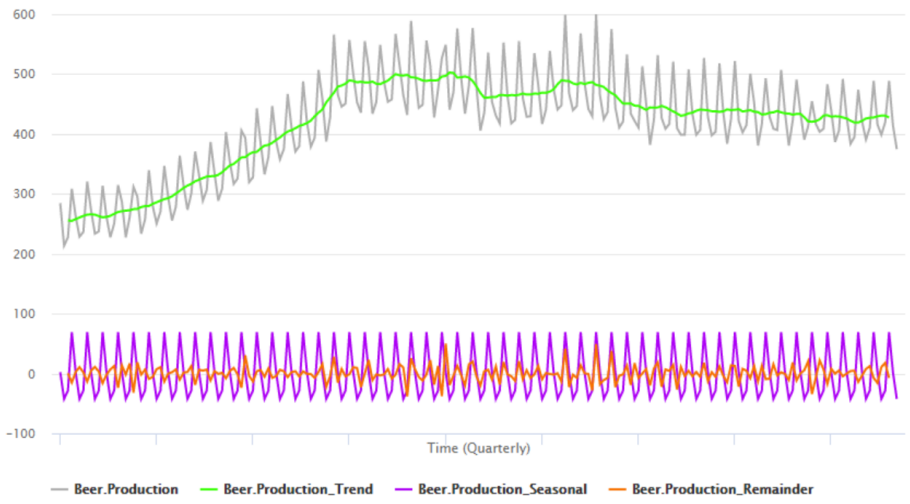
You can also use the following code to retrieve the dataset:

```
!curl https://dl.dropboxusercontent.com/s/fx94e9w8xgyxho5/beer_dataset.csv?dl=0 -so beer_dataset.csv
```

Data Preparation & Exploration

Time Series Decomposition

This is the line plot showing the **original time series**, the **trend**, **seasonal component**, and **noise**.




```
col_quarter = _beers["Quarter"]
conditions = [
(col_quarter == "Q1"),
(col_quarter == "Q2"),
(col_quarter == "Q3"),
(col_quarter == "Q4")
]
values = ["01", "04", "07", "10"]
_beers["Month"] = np.select(conditions, values)

_beers["Date"] = pd.to_datetime(_beers[["Year","Month"]].assign(day=1)).dt.to_period("M")
_beers.set_index("Date", inplace=True)

_beers.drop("Quarter", axis=1, inplace=True)
_beers.drop("Time", axis=1, inplace=True)
_beers.drop("Month", axis=1, inplace=True)
_beers.drop("Year", axis=1, inplace=True)

print(_beers.head())
```

✓ 0.3s

```
         Beer.Production
Date
1956-01              284
1956-04              213
1956-07              227
1956-10              308
1957-01              262
```

```
    _beers.index = _beers.index.to_timestamp('s').strftime('%Y-%m-%d %H:%M:%S.000')
    _beers.index = pd.DatetimeIndex(_beers.index, freq='infer')
    print(_beers.head())
    _beers.plot()
```

✓ 0.3s

```
            Beer.Production
Date
1956-01-01              284
1956-04-01              213
1956-07-01              227
1956-10-01              308
1957-01-01              262
```

Time Series Forecasting

Your task is either **to forecast the next 12 quarters OR to test the last 12 quarters** (2008, 2009, 2010). Choose one.

Exponential Smoothing 15/15 5/5

Make one exponential-smoothing model for your task. Adjust the parameters accordingly. Attach the code/ architecture with parameters here.

Exponential Smoothing

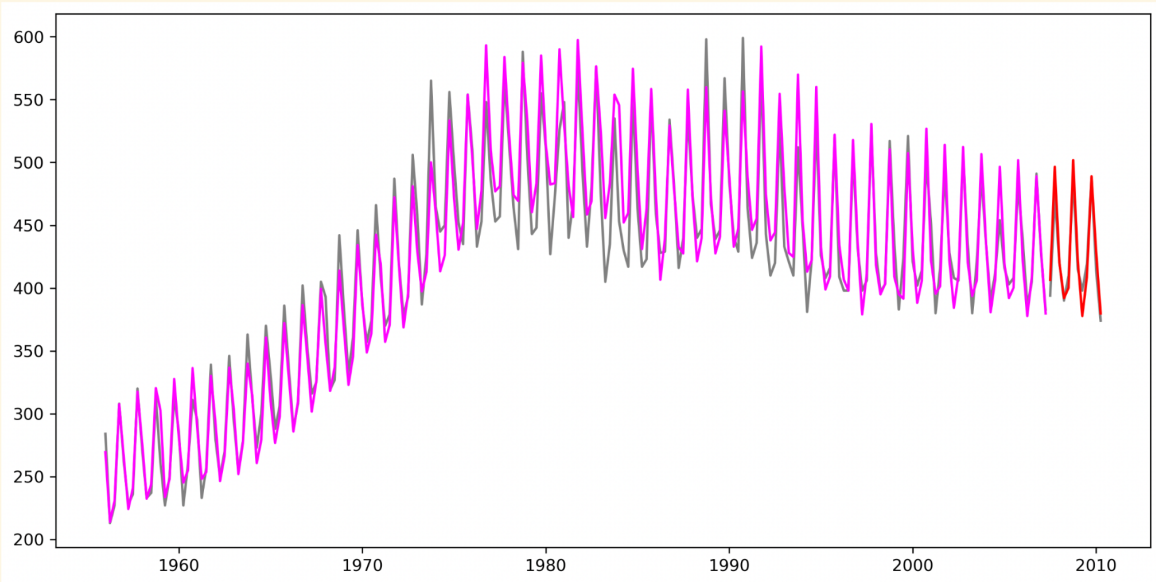
```
_HORIZON = 12  # The most recent 12 months as the test set
train = _beers.iloc[:-_HORIZON, :]
test = _beers.iloc[-_HORIZON:, :]
train.index = pd.to_datetime(train.index)
test.index = pd.to_datetime(test.index)
```

✓ 0.2s

```
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from sklearn.metrics import mean_squared_error

model = ExponentialSmoothing(train['Beer.Production'],
                             trend = 'additive',
                             seasonal = 'additive',
                             seasonal_periods = _HORIZON
                             )

model_fit = model.fit(smoothing_level=0.07,  # The alpha value of the simple exponential smoothing
                      smoothing_trend=0.05,    # The beta value of the Holt's trend method
                      smoothing_seasonal=0.6 # The gamma value of the holt winters seasonal method
                      )
model_forecast = model_fit.predict(start=-_HORIZON)

# Plotting the Data
fig, ax = plt.subplots(figsize=(12, 6))
ax.plot(train.index, train.values, color='gray')
ax.plot(test.index, test.values, color="gray")
ax.plot(train.index, model_fit.fittedvalues, color='magenta')
ax.plot(test.index, model_forecast, color='red')

# Model Evaluation
rmse = mean_squared_error(test.values, model_forecast, squared=False)
print(f"RMSE = {rmse}")
```

Line plot of the prediction (the forecast/prediction part should be colored differently):

```
RMSE = 12.296374034401078
```



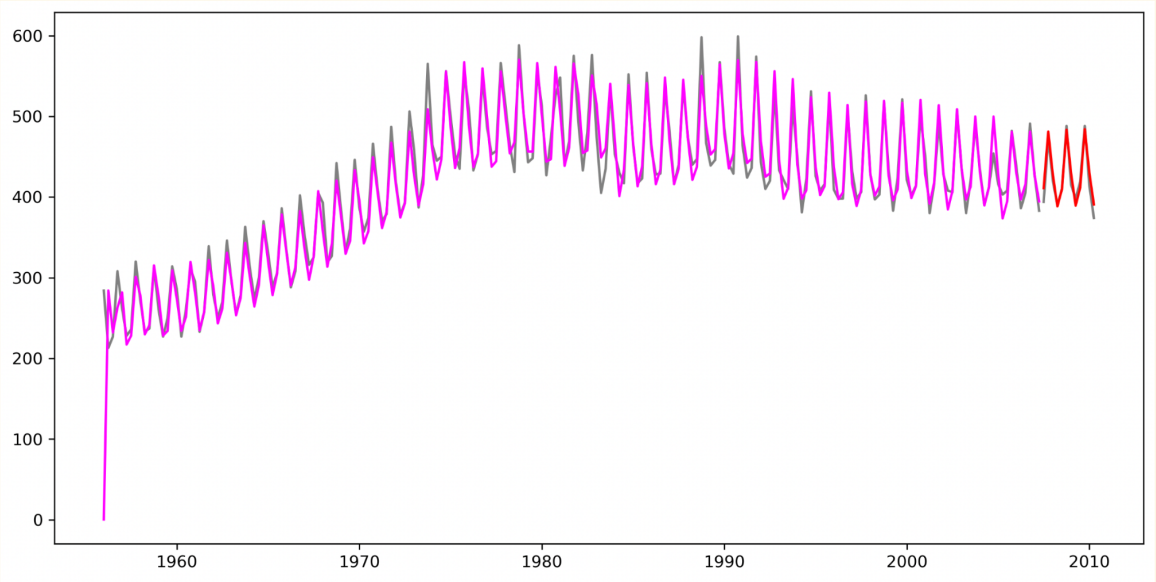
Autoregressive Model 15/15 5/5 10/10

Make one autoregressive model for your task. Adjust the parameters accordingly. Attach the code/architecture with parameters here.

```
from pmdarima.arima import auto_arima
model = auto_arima(train['Beer.Production'],
                   seasonal=True,
                   trace=True,
                   stepwise=True,
                   random_state=1,
                   n_fits=50)

model_fit = model.fit(train['Beer.Production'])
model_forecast = model_fit.predict(n_periods = _HORIZON)

# Plotting the Data
fig, ax = plt.subplots(figsize=(12, 6))
ax.plot(train.index, train.values, color='gray')
ax.plot(test.index, test.values, color="gray")
ax.plot(train.index, model_fit.fittedvalues(), color='magenta')
ax.plot(test.index, model_forecast, color='red')

# Model Evaluation
from sklearn.metrics import mean_squared_error
rmse = mean_squared_error(test.values, model_forecast, squared=False)
print(f"RMSE = {rmse}")
```

Line plot of the prediction (the forecast/prediction part should be colored differently):

```
Best model:  ARIMA(5,1,3)(0,0,0)[0] intercept
Total fit time: 5.307 seconds
RMSE = 10.408130912165838
```



Analysis（50 - 150 words）

(Hint: Which of the two methods do you think work well on this dataset? Give your opinion on this.)

Around 1975 to 1995, there are a lot of variations (maybe due to some noises?). Exponential smoothing will also reproduce these variations but they will be a little bit shifted. Hence, when the big variations abruptly stop, the exponential smoothing will still reproduce them for some times, resulting in bigger errors. Indeed, exponential smoothing gives more weight to the most recent observations, whereas autoregressive models rely on the past values of the time series to predict future values. Therefore we think the ARIMA model worked better with this dataset.

Part B: Outlier Detection & Feature Selection

The goal of this part is to perform data cleaning using outlier detection and feature selection on **one of the data** you have analyzed in **Assignment Lesson 4-5**, **Assignment Lesson 6-7**, or **Assignment 8-9**. You will then continue to do predictive modeling (classification or regression) using the cleaned data.

Prior Knowledge

Data Science Objective

Write down the data science objective for this task. (You can just paste here the objective from your past assignments.)

Dataset from Assignment Lesson 6 - 7
Data Science Objective:
The price of a house depends on different factors (location, renovation, number of rooms, size…). We are actually interested to see which factors impact the final price of a house. Can we predict the price of a house, given its condition?

With 79 explanatory variables describing a different aspect of residential homes in Ames, Iowa, we will try to predict the final price of each home. This would be helpful to sellers to estimate the price of their house, but it is also price references to buyers preventing them from being scammed. From the point of view of architects, it also provides what people prefer (what they want to pay for).

Data Preparation & Exploration

Outlier Detection 11/15

Perform **two** different outlier detection methods to your dataset. Attach the code/architecture with parameters here.

Outlier Detection 1 (on the original dataset)

Method Name: Nearest Neighbor-based / KNN Global Anomally Score

```
from scipy.stats import scoreatpercentile
from pyod.models.knn import KNN

outliers_fraction = 0.5

model = KNN(
    contamination=outliers_fraction,
    n_neighbors=3,
    method='mean'
)

model.fit(df_train)

df['anomaly_score'] = model.decision_scores_
df['is_outlier'] = model.predict(df_train)

df_clean = df[df['is_outlier']==0].iloc[:,:-2]

print("Shape Original Dataset: ", df.shape)
print("Shape Clean    Dataset: ", df_clean.shape)
```

✓ 0.9s

```
Shape Original Dataset:  (2919, 334)
Shape Clean    Dataset:  (2109, 332)
```

Outlier Detection 2 (on the original dataset)

Method Name: Model-based / Minimum Covariance Determinant (MCD)

```python
from pyod.models.mcd import MCD

model = MCD (
    contamination= 0.4,
)

model.fit(df_train)

df['anomaly_score'] = model.decision_scores_
df['is_outlier'] = model.predict(df_train)

df_clean = df[df['is_outlier']==0].iloc[:,:-2]

print("Shape Original Dataset: ", df.shape)
print("Shape Clean    Dataset: ", df_clean.shape)
```

✓ 40.7s

```
/Users/cameliaguerraoui/.pyenv/versions/anaconda3-2021
covariance matrix associated to your dataset is not fu
  warnings.warn("The covariance matrix associated to y

Shape Original Dataset:  (2919, 335)
Shape Clean    Dataset:  (1751, 333)
```

Feature Selection 15/15

Perform **one filter-type** and **one wrapper-type** method on your dataset. Attach the code/architecture with parameters here.

Filter-type (on the original dataset) Dimensionality Reduction-based

Method Name: Weight by Principal Component Analysis

```
df_att = df.copy()
df_att.drop("Price", axis=1, inplace=True)
```

```
from numpy.linalg import eigh

pca_comp = 5

cov_matrix = np.cov(df_att, rowvar=False)
egnvalues, egnvectors = eigh(cov_matrix)

total_egnvalues = sum(egnvalues)
var_exp = [(i/total_egnvalues) for i in sorted(egnvalues, reverse=True)]

egnpairs = [(np.abs(egnvalues[i]), egnvectors[:, i]) for i in range(len(egnvalues))]
egnpairs.sort(key=lambda k: k[0], reverse=True)
projection_matrix = np.hstack([egnpairs[i][1][:, np.newaxis] for i in range(pca_comp)])

df_proj_matrix = pd.DataFrame(projection_matrix.T.round(4), columns=df_att.columns)

features_names = np.absolute(df_proj_matrix).idxmax(axis=1)
print(features_names)
```

✓ 0.1s

```
0     OverallQual
1        2ndFlrSF
2     GarageYrBlt
3       BsmtUnfSF
4    KitchenAbvGr
dtype: object
```

Wrapper-type (on the original dataset)

Method Name: Forward/Backward Elimination (Sequential Feature Selection)

```python
from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.linear_model import LinearRegression

model = LinearRegression()
eval_metric = "neg_mean_squared_error"
selector = SequentialFeatureSelector(
    estimator=model,
    direction='forward',
    scoring=eval_metric,
    n_features_to_select=5,
)


df_att = df.copy()
df_att.drop("Price", axis=1, inplace=True)
selector.fit(df_att, df['Price'])
selected_features = selector.transform(df_att)
selected_feature_names = selector.get_feature_names_out(input_features=df_att.columns)
pd.DataFrame(selected_features, columns=selected_feature_names).head()
```

	YearBuilt	TotalBsmtSF	GrLivArea	BldgType_1Fam	RoofMatl_WdShngl
0	1.046078	-0.444176	0.413476	1.0	0.0
1	0.154737	0.476948	-0.471810	1.0	0.0
2	0.980053	-0.298974	0.563659	1.0	0.0
3	-1.859033	-0.671053	0.427309	1.0	0.0
4	0.947040	0.211501	1.377806	1.0	0.0

Predictive Modeling

<!> The goal here is to try out different outlier detection & feature selection methods. If applying both methods is giving a worse performance, that is OK.

● Modeling with and without outlier detection & feature selection 10/10 9/10
Choose one outlier detection and feature selection method that you think best suit your model.
OUTLIER DETECTION
Method Name: KNN
FEATURE SELECTION
Method Name: PCA
Perform predictive modeling (classification/regression) on the data that has and has not been processed with outlier detection & feature selection. Attach the code/architecture here and write their performance (accuracy, RMSE, etc.). You can always reuse the model you have made in past assignments.

With outlier detection and feature selection:

```
from sklearn.ensemble import RandomForestRegressor

X = df_att[features_names]
y = df_clean["Price"]
X_train, X_test, y_train, y_test = sklearn.model_selection.train_test_split(X,
                                                                            y,
                                                                            test_size=0.1,
                                                                            random_state=1
                                                                           )


k_folds = sklearn.model_selection.KFold(n_splits = 5)

def _evaluate(model, X, y):
    values = sklearn.model_selection.cross_val_score(model,
                                                     X,
                                                     y,
                                                     cv = k_folds,
                                                     scoring="neg_mean_squared_log_error"
                                                    )
    print("Eval : ", values.mean())


regressor = RandomForestRegressor(
    n_estimators=90,
)
regressor.fit(X_train, y_train)
prediction = regressor.predict(X_test)
_evaluate(regressor, X_train, y_train)
_evaluate(regressor, X_test, y_test)
```

✓ 2.9s

```
Eval :  -0.03635921835406593
Eval :  -0.04345552054670116
```

No outlier detection and feature selection:

```
X = df.drop("Price", axis=1)
y = df["Price"]
X_train, X_test, y_train, y_test = sklearn.model_selection.train_test_split(X,
                                                                            y,
                                                                            test_size=0.1,
                                                                            random_state=1
                                                                           )

regressor.fit(X_train, y_train)
prediction = regressor.predict(X_test)
_evaluate(regressor, X_train, y_train)
_evaluate(regressor, X_test, y_test)
```

✓ 38.4s

```
Eval :  -0.04577794089711061
Eval :  -0.0449132784282586
```

Performance (no outlier detection & feature selection):
Root Mean Square Log Error = 0.045
Performance (with outlier detection & feature selection):
Root Mean Square Log Error = 0.043

Analysis（100 - 200 words）

(Hint: What made you choose those outlier detection & feature selection methods? Does either outlier detection & feature selection helps in making the better performance? Describe why/why not.)

We chose the KNN method over the MCD method as it gave better performances. One reason might be that MCD considers some points as outliers even though they are not. We chose PCA because it is a good and easy choice when the goal is to reduce the dimensionality of the data, which can help remove noise and irrelevant information from the data, making the modeling process more efficient. PCA identifies the most important variables and creates a smaller set of new variables that capture the maximum amount of variance in the data.

While outlier detection and feature selection can help improve the performance of a model, here, the improvement is rather small in terms of accuracy but important in terms of speed. Indeed, the RMSLE is only 0.002 smaller with outlier detection and feature selection, but the computing time jumps from 38.4 sec to 2.9 sec, so 13 times faster.
Outlier detection and feature selection reduce the complexity of the model by removing data points that are unlikely to be representative of the overall population or by focusing on the most important variables. This can make the model more efficient and faster to run, as it works with a smaller and cleaner set of data.
Additionally, the faster performance of the model enables faster iterations and allows us to test more models in less time, ultimately leading to improved results.