Air Passenger Time Series Forecasting

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

import pycaret
import kaleido
from pycaret.time_series import *
```

EDA

In [ ]:
```
airpassenger = pd.read_csv('AirPassengers.csv')
airpassenger.head(10)
```

Out[ ]:

	Month	#Passengers
0	1949-01	112
1	1949-02	118
2	1949-03	132
3	1949-04	129
4	1949-05	121
5	1949-06	135
6	1949-07	148
7	1949-08	148
8	1949-09	136
9	1949-10	119

In [ ]:
```
airpassenger.describe()
```

Out[ ]:

	#Passengers
count	144.000000
mean	280.298611
std	119.966317
min	104.000000
25%	180.000000
50%	265.500000
75%	360.500000
max	622.000000

In [ ]:
```
airpassenger.shape
```

Out[ ]:
```
(144, 2)
```

In [ ]:
```
airpassenger.info()
```

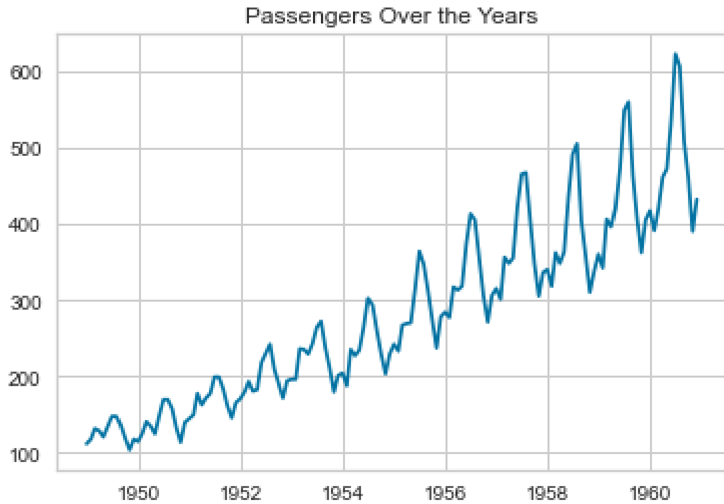
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144 entries, 0 to 143
Data columns (total 2 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   Month        144 non-null    object
 1   #Passengers  144 non-null    int64 
dtypes: int64(1), object(1)
memory usage: 2.4+ KB
```

In [ ]:
```
airpassenger['Month'] = pd.to_datetime(airpassenger['Month'], format='%Y-%m')
airpassenger = airpassenger.set_index('Month')
airpassenger.head()
```

Out[ ]:

	#Passengers
Month	
1949-01-01	112
1949-02-01	118
1949-03-01	132
1949-04-01	129
1949-05-01	121

In [ ]:
```
plt.plot(airpassenger)
plt.title('Passengers Over the Years')
plt.grid(True)
plt.show()
```



In [ ]:

```
# Setup PyCaret
# cross validation will be 3 and forecast horizon will be 12 (last 12 points in
s = setup(airpassenger, fold = 3, fh = 12, session_id = 123)
setup_results = pull()
print(setup_results)
```

	Description	Value
0	session_id	123
1	Target	#Passengers
2	Approach	Univariate
3	Exogenous Variables	Not Present
4	Original data shape	(144, 1)
5	Transformed data shape	(144, 1)
6	Transformed train set shape	(132, 1)
7	Transformed test set shape	(12, 1)
8	Rows with missing values	0.0%
9	Fold Generator	ExpandingWindowSplitter
10	Fold Number	3
11	Enforce Prediction Interval	False
12	Splits used for hyperparameters	all
13	User Defined Seasonal Period(s)	None
14	Ignore Seasonality Test	False
15	Seasonality Detection Algo	auto
16	Max Period to Consider	60
17	Seasonal Period(s) Tested	[12, 24, 36, 11, 48]
18	Significant Seasonal Period(s)	[12, 24, 36, 11, 48]
19	Significant Seasonal Period(s) without Harmonics	[48, 36, 11]
20	Remove Harmonics	False
21	Harmonics Order Method	harmonic_max
22	Num Seasonalities to Use	1
23	All Seasonalities to Use	[12]
24	Primary Seasonality	12
25	Seasonality Present	True
26	Target Strictly Positive	True
27	Target White Noise	No
28	Recommended d	1
29	Recommended Seasonal D	1
30	Preprocess	False
31	CPU Jobs	-1
32	Use GPU	False
33	Log Experiment	False
34	Experiment Name	ts-default-name

	Description	Value
35	USI	0257

```
                                          Description                   Value
0                                          session_id                     123
1                                              Target             #Passengers
2                                            Approach              Univariate
3                                 Exogenous Variables             Not Present
4                                 Original data shape                (144, 1)
5                              Transformed data shape                (144, 1)
6                         Transformed train set shape                (132, 1)
7                          Transformed test set shape                 (12, 1)
8                             Rows with missing values                   0.0%
9                                      Fold Generator  ExpandingWindowSplitter
10                                        Fold Number                       3
11                        Enforce Prediction Interval                   False
12                       Splits used for hyperparameters                  all
13                       User Defined Seasonal Period(s)                 None
14                             Ignore Seasonality Test                  False
15                        Seasonality Detection Algo                     auto
16                               Max Period to Consider                   60
17                           Seasonal Period(s) Tested   [12, 24, 36, 11, 48]
18                      Significant Seasonal Period(s)   [12, 24, 36, 11, 48]
19   Significant Seasonal Period(s) without Harmonics          [48, 36, 11]
20                                  Remove Harmonics                  False
21                            Harmonics Order Method           harmonic_max
22                            Num Seasonalities to Use                    1
23                            All Seasonalities to Use                 [12]
24                                Primary Seasonality                    12
25                               Seasonality Present                   True
26                           Target Strictly Positive                  True
27                                 Target White Noise                    No
28                                    Recommended d                       1
29                           Recommended Seasonal D                       1
30                                        Preprocess                  False
31                                          CPU Jobs                     -1
32                                           Use GPU                  False
33                                    Log Experiment                  False
34                                   Experiment Name        ts-default-name
35                                               USI                   0257
```

In [ ]:

```
s.check_stats()
```

Out[ ]:

	Test	Test Name	Data	Property	Setting	Value
0	Summary	Statistics	Transformed	Length		144.0
1	Summary	Statistics	Transformed	# Missing Values		0.0
2	Summary	Statistics	Transformed	Mean		280.298611
3	Summary	Statistics	Transformed	Median		265.5
4	Summary	Statistics	Transformed	Standard Deviation		119.966317
5	Summary	Statistics	Transformed	Variance		14391.917201
6	Summary	Statistics	Transformed	Kurtosis		-0.364942
7	Summary	Statistics	Transformed	Skewness		0.58316
8	Summary	Statistics	Transformed	# Distinct Values		118.0
9	White Noise	Ljung-Box	Transformed	Test Statictic	{'alpha': 0.05, 'K': 24}	1606.083817
10	White Noise	Ljung-Box	Transformed	Test Statictic	{'alpha': 0.05, 'K': 48}	1933.155822
11	White Noise	Ljung-Box	Transformed	p-value	{'alpha': 0.05, 'K': 24}	0.0
12	White Noise	Ljung-Box	Transformed	p-value	{'alpha': 0.05, 'K': 48}	0.0
13	White Noise	Ljung-Box	Transformed	White Noise	{'alpha': 0.05, 'K': 24}	False
14	White Noise	Ljung-Box	Transformed	White Noise	{'alpha': 0.05, 'K': 48}	False
15	Stationarity	ADF	Transformed	Stationarity	{'alpha': 0.05}	False
16	Stationarity	ADF	Transformed	p-value	{'alpha': 0.05}	0.99188
17	Stationarity	ADF	Transformed	Test Statistic	{'alpha': 0.05}	0.815369
18	Stationarity	ADF	Transformed	Critical Value 1%	{'alpha': 0.05}	-3.481682
19	Stationarity	ADF	Transformed	Critical Value 5%	{'alpha': 0.05}	-2.884042
20	Stationarity	ADF	Transformed	Critical Value 10%	{'alpha': 0.05}	-2.57877
21	Stationarity	KPSS	Transformed	Trend Stationarity	{'alpha': 0.05}	True
22	Stationarity	KPSS	Transformed	p-value	{'alpha': 0.05}	0.1
23	Stationarity	KPSS	Transformed	Test Statistic	{'alpha': 0.05}	0.09615
24	Stationarity	KPSS	Transformed	Critical Value 10%	{'alpha': 0.05}	0.119
25	Stationarity	KPSS	Transformed	Critical Value 5%	{'alpha': 0.05}	0.146
26	Stationarity	KPSS	Transformed	Critical Value 2.5%	{'alpha': 0.05}	0.176
27	Stationarity	KPSS	Transformed	Critical Value 1%	{'alpha': 0.05}	0.216
28	Normality	Shapiro	Transformed	Normality	{'alpha': 0.05}	False
29	Normality	Shapiro	Transformed	p-value	{'alpha': 0.05}	0.000068

Since the p-value of the Ljung-Box test is less than 0.05, we can assume that the values are showing dependence on each other. This time series is not stationary because the ADF p-value is greater than 0.05. Since we also cannot reject the null hypothesis of the KPSS test (p-value greater than 0.05), where the null hypothesis is stationary, we observe that the series is stationary around a deterministic trend (slope of the trend in the series does not change permanently). With the p-value of the Shapiro less than 0.05, we reject the null hypothesis and there is evidence that the data tested are not normally distributed.

In [ ]:
```
# Time Series plot
plot_model(plot = 'ts', fig_kwargs={'hoverinfo':'none'})
```

Upward trend throughout the years. There is a seasonal variation as there are peaks in July or August. There is a mulitplicative seasonal variability with an additive trend.

In [ ]:
```
# Train and Test Plot
plot_model(plot='train_test_split', fig_kwargs={'hoverinfo':'none'})
```

In [ ]:
```
# Cross Validation plot
plot_model(plot = 'cv', fig_kwargs={'hoverinfo':'none'})
```

In [ ]:
```
# Diagnostic plot
plot_model(plot='diagnostics', fig_kwargs={'hoverinfo':'none'})
```

The periodogram shows possible cyclical behavior in a time series. The time series is recomposed using a sum of cosine waves with varying amplitudes and frequencies. This time series are mostly equal amplitude, but further out cosine waves. ACF is the measure of correlation between two datapoints and how that changes as the distance between them increases. As we can see in the ACF plot, the autocorrelations are decreasing slowly with the increasing lags, indicating that the time series is non-stationary. PACF is a conditional correlation between two datapoints assuming we know their dependencies with another set of datapoints. The PACF shows that the first and second lagged values have a clear statistical significance with regards to their partial autocorrelations. We can see on the QQ plot, the ends are tailing off towards the end, showing the distribution being not Normally distributed.

Decomposition Plot

I will plot a multiplicative decomposition plot. Multiplicative decomposition is that the time series is the product of its components.

In [ ]:
```
# Decomposition Plot
plot_model(plot='decomp', data_kwargs={'type': 'multiplicative'}, fig_kwargs={'
```

There is an upward trend and seasonality present based on the decomposition results. The residuals show an interesting result as there is high variability in early and later years.

Differencing

I will now perform differencing on this time series with both first difference, and first difference with seasonal difference. First row: Original time series Second row: First differencing Third row: First difference with seasonal differencing

In [ ]:
```
plot_model(plot='diff', data_kwargs={'lags_list': [[1], [1, 12]], 'acf': True,
```

As we observe these results, the first difference with seasonal differencing shows stationarity since the ACF plot displays the ACF quickly dropping to zero. First difference with seasonal differencing has the best results to show stationarity.

Modeling

Now, I will utilize the compare_models() function in PyCaret, which trains and compares the performance of all estimators available in the library, while using cross-validation.

In [ ]:
```
best = compare_models()
model_results = pull()
print(model_results)
```

```
Processing:   0%|          | 0/125 [00:00<?, ?it/s]
```

	Model	MASE
exp_smooth	Exponential Smoothing	0.5852
ets	ETS	0.5931
et_cds_dt	Extra Trees w/ Cond. Deseasonalize & Detrending	0.6596
huber_cds_dt	Huber w/ Cond. Deseasonalize & Detrending	0.6813
arima	ARIMA	0.683
lr_cds_dt	Linear w/ Cond. Deseasonalize & Detrending	0.7004
ridge_cds_dt	Ridge w/ Cond. Deseasonalize & Detrending	0.7004
lar_cds_dt	Least Angular Regressor w/ Cond. Deseasonalize...	0.7004
en_cds_dt	Elastic Net w/ Cond. Deseasonalize & Detrending	0.7029
lasso_cds_dt	Lasso w/ Cond. Deseasonalize & Detrending	0.7048
catboost_cds_dt	CatBoost Regressor w/ Cond. Deseasonalize & De...	0.7106
br_cds_dt	Bayesian Ridge w/ Cond. Deseasonalize & Detren...	0.7112
knn_cds_dt	K Neighbors w/ Cond. Deseasonalize & Detrending	0.7162
auto_arima	Auto ARIMA	0.7181
gbr_cds_dt	Gradient Boosting w/ Cond. Deseasonalize & Det...	0.783
xgboost_cds_dt	Extreme Gradient Boosting w/ Cond. Deseasonali...	0.8155
lightgbm_cds_dt	Light Gradient Boosting w/ Cond. Deseasonalize...	0.8156
ada_cds_dt	AdaBoost w/ Cond. Deseasonalize & Detrending	0.8193
rf_cds_dt	Random Forest w/ Cond. Deseasonalize & Detrending	0.8352
llar_cds_dt	Lasso Least Angular Regressor w/ Cond. Deseaso...	0.967
theta	Theta Forecaster	0.9729
omp_cds_dt	Orthogonal Matching Pursuit w/ Cond. Deseasona...	1.009
dt_cds_dt	Decision Tree w/ Cond. Deseasonalize & Detrending	1.0429
snaive	Seasonal Naive Forecaster	1.1479
par_cds_dt	Passive Aggressive w/ Cond. Deseasonalize & De...	1.2472
polytrend	Polynomial Trend Forecaster	1.6523
croston	Croston	1.9311
naive	Naive Forecaster	2.3599
grand_means	Grand Means Forecaster	5.5306

\

	RMSSE	MAE	RMSE	MAPE	SMAPE	R2	TT (Sec)
exp_smooth	0.6105	17.1926	20.1633	0.0435	0.0439	0.8918	0.0600
ets	0.6212	17.4165	20.5102	0.044	0.0445	0.8882	0.0900
et_cds_dt	0.7284	19.447	24.0929	0.0484	0.0484	0.846	0.2167
huber_cds_dt	0.7866	20.0334	25.967	0.0491	0.0499	0.8113	0.1300
arima	0.6735	20.0069	22.2199	0.0501	0.0507	0.8677	0.8300
lr_cds_dt	0.7702	20.6084	25.4401	0.0509	0.0514	0.8215	0.2433
ridge_cds_dt	0.7703	20.6086	25.4405	0.0509	0.0514	0.8215	0.1933
lar_cds_dt	0.7702	20.6084	25.4401	0.0509	0.0514	0.8215	0.1433
en_cds_dt	0.7732	20.6816	25.5362	0.0511	0.0516	0.8201	0.1900
lasso_cds_dt	0.7751	20.7373	25.6005	0.0512	0.0517	0.8193	0.1300
catboost_cds_dt	0.8146	20.9112	26.8907	0.0505	0.0509	0.8085	0.7900
br_cds_dt	0.7837	20.9213	25.8795	0.0515	0.0521	0.8144	0.1267
knn_cds_dt	0.8157	21.1613	26.97	0.0521	0.0529	0.7811	0.1467
auto_arima	0.7114	21.0297	23.4661	0.0525	0.0531	0.8509	1.7833
gbr_cds_dt	0.9122	23.0447	30.1134	0.0562	0.0569	0.7514	0.1600
xgboost_cds_dt	0.9591	24.0738	31.695	0.0582	0.0592	0.7118	0.1567
lightgbm_cds_dt	0.9117	24.0002	30.0956	0.0575	0.0587	0.7561	0.2233
ada_cds_dt	0.9655	24.106	31.8637	0.0576	0.0593	0.7058	0.1600
rf_cds_dt	0.9453	24.6117	31.2326	0.0601	0.0607	0.7366	0.2300
llar_cds_dt	1.1915	28.4499	39.3303	0.0665	0.0693	0.5738	0.1933
theta	1.0306	28.3192	33.8639	0.067	0.07	0.671	0.0233
omp_cds_dt	1.237	29.6294	40.8121	0.0685	0.0718	0.5462	0.1267
dt_cds_dt	1.2226	30.48	40.1912	0.0726	0.0753	0.5362	0.1333
snaive	1.0945	33.3611	35.9139	0.0832	0.0879	0.6072	0.0167
par_cds_dt	1.3081	36.7727	43.3215	0.0935	0.0961	0.4968	0.1233
polytrend	1.9202	48.6301	63.4299	0.117	0.1216	-0.0784	0.0167
croston	2.3517	56.618	77.5856	0.1295	0.1439	-0.6281	0.7833

```
naive              2.7612    69.0278    91.0322  0.1569  0.1792 -1.2216    1.1767
grand_means        5.2596   162.4117   173.6492     0.4  0.5075 -7.0462    0.8200
```

Based on the results, the Exponential Smoothing model generates the best model for this time series as the RMSE is the lowest out of all models and its R2 score is the highest out of all models. The RMSE indicates that predictions do not fall far from actual values and R2 exhibits how much of the dependent variable is predictable from the independent variable.

I will now plot the out-of-sample forecasting performance and the in-sample plots using the exponential smoothing model.

In [ ]:
```
plot_model(best, plot = 'forecast', fig_kwargs={'hoverinfo':'none'})
plot_model(best, plot = 'insample', fig_kwargs={'hoverinfo':'none'})
```

The plots show us that the exponential smoothing model generates good results when comparing predictions to actual data.

I will now create a forecast plot for the next 60 months using this exponential smoothing model.

In [ ]:
```
plot_model(best, plot = 'forecast', data_kwargs={'fh': 60}, fig_kwargs={'hoveri
```

I will now output the predictions of passengers for 60 months from January 1960 to December 1964.

In [ ]:
```
predict_model(best, fh=np.arange(1, 61))
```

Out[ ]:

	y_pred
1960-01	417.2810
1960-02	394.0567
1960-03	462.4373
1960-04	448.5887
1960-05	471.8593
1960-06	539.8763
1960-07	623.8054
1960-08	631.1408
1960-09	515.5723
1960-10	449.8958
1960-11	394.2734
1960-12	422.5032
1961-01	446.6778
1961-02	421.6553
1961-03	494.6371
1961-04	479.6441
1961-05	504.3383
1961-06	576.8251
1961-07	666.2561
1961-08	673.8486
1961-09	550.2642
1961-10	479.9997
1961-11	420.5091
1961-12	450.4623
1962-01	476.0745
1962-02	449.2539
1962-03	526.8370
1962-04	510.6994
1962-05	536.8173
1962-06	613.7739
1962-07	708.7069
1962-08	716.5563
1962-09	584.9561
1962-10	510.1035
1962-11	446.7448

	y_pred
1962-12	478.4214
1963-01	505.4713
1963-02	476.8526
1963-03	559.0369
1963-04	541.7548
1963-05	569.2963
1963-06	650.7227
1963-07	751.1576
1963-08	759.2641
1963-09	619.6480
1963-10	540.2073
1963-11	472.9805
1963-12	506.3805
1964-01	534.8680
1964-02	504.4512
1964-03	591.2367
1964-04	572.8102
1964-05	601.7753
1964-06	687.6715
1964-07	793.6084
1964-08	801.9718
1964-09	654.3399
1964-10	570.3112
1964-11	499.2162
1964-12	534.3396

I will now produce the diagnostics plot for the exponential smoothing model.

In [ ]:
```
plot_model(best, plot='diagnostics', fig_kwargs={'hoverinfo':'none'})
```

From the diagnostics plot, the histogram is now normally distributed and that can be confirmed with the QQ Plot has a near straight line. The ACF drops to zero relatively quickly with 2 significant flags, and the PACF cuts off after lag 1.

Model Tuning

In [ ]:

```
expsmooth_tune = tune_model(best)
print(best)
print(expsmooth_tune)
```

	cutoff	MASE	RMSSE	MAE	RMSE	MAPE	SMAPE	R2
0	1956-12	0.3617	0.4124	10.5620	13.4978	0.0272	0.0273	0.9407
1	1957-12	0.8588	0.8856	26.2573	30.0652	0.0738	0.0704	0.7632
2	1958-12	0.3942	0.4126	11.2644	13.4112	0.0261	0.0265	0.9598
Mean	nan	0.5382	0.5702	16.0279	18.9914	0.0424	0.0414	0.8879
SD	nan	0.2271	0.2230	7.2390	7.8304	0.0222	0.0205	0.0885

```
Processing:   0%|          | 0/7 [00:00<?, ?it/s]
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:    1.0s finished
ExponentialSmoothing(seasonal='mul', sp=12, trend='add')
ExponentialSmoothing(seasonal='add', sp=12, trend='additive', use_boxcox=True)
```

The tuned exponential smoothing model has a better MASE, MAE, RMSE, and MAPE have lower values compared to the non-tuned model.

In [ ]:

```
# Tuned model performance
pred_expsmooth = predict_model(expsmooth_tune)
plot_model(expsmooth_tune, fig_kwargs={'hoverinfo':'none'})
plot_model(expsmooth_tune, plot='insample', fig_kwargs={'hoverinfo':'none'})
```

	Model	MASE	RMSSE	MAE	RMSE	MAPE	SMAPE	R2
0	Exponential Smoothing	0.5858	0.6575	17.8364	22.7139	0.0375	0.0364	0.9069

As we can see, performance is better than the previous one. Let's forecast the next 60 points of future passengers with this tuned exponential smoothing model.

In [ ]:

```
plot_model(expsmooth_tune, plot = 'forecast', data_kwargs={'fh': 60}, fig_kwarg
```

In [ ]:

```
predict_model(expsmooth_tune, fh=np.arange(1, 61))
```

Out[ ]:

	y_pred
1960-01	421.1026
1960-02	401.0329
1960-03	470.9998
1960-04	460.4431
1960-05	485.2459
1960-06	556.8663
1960-07	638.3336
1960-08	643.5432
1960-09	529.1200
1960-10	464.7396
1960-11	409.7826
1960-12	445.7128
1961-01	471.3226
1961-02	449.1764
1961-03	526.3262
1961-04	514.6955
1961-05	542.0166
1961-06	620.8156
1961-07	710.2995
1961-08	716.0169
1961-09	590.3038
1961-10	519.4295
1961-11	458.8331
1961-12	498.4609
1962-01	526.6818
1962-02	502.2789
1962-03	587.2305
1962-04	574.4341
1962-05	604.4881
1962-06	691.0698
1962-07	789.2331
1962-08	795.4999
1962-09	657.5612
1962-10	579.6429
1962-11	512.9214

	y_pred
1962-12	556.5663
1963-01	587.6217
1963-02	560.7690
1963-03	654.1850
1963-04	640.1246
1963-05	673.1412
1963-06	768.1513
1963-07	875.7012
1963-08	882.5617
1963-09	731.3986
1963-10	645.8484
1963-11	572.4818
1963-12	620.4857
1964-01	654.6147
1964-02	625.1056
1964-03	727.6943
1964-04	712.2652
1964-05	748.4897
1964-06	852.6174
1964-07	970.3081
1964-08	977.8097
1964-09	812.3567
1964-10	718.5466
1964-11	637.9792
1964-12	690.7077

In [ ]:
```
plot_model(expsmooth_tune, plot='diagnostics', fig_kwargs={'hoverinfo':'none'})
```

There is not a huge difference between the tuned model and non-tuned model, however, since it is slightly better, we will stick to the tuned exponential smoothing model.

In [ ]:
```
save_model(expsmooth_tune, 'exp_smooth_tune_model')
```

```
Transformation Pipeline and Model Successfully Saved
```

Out[ ]:

```
(ForecastingPipeline(steps=[('forecaster',
                             TransformedTargetForecaster(steps=[('model',
                                                                 ExponentialSm
oothing(seasonal='add',

sp=12,

trend='additive',

use_boxcox=True))]))]),
 'exp_smooth_tune_model.pkl')
```

In [ ]:

```
# Load model
load_final_model = load_model('exp_smooth_tune_model')
```

```
Transformation Pipeline and Model Successfully Loaded
```