

PROJE RAPORU

RAPOR TARİHİ

22.06.2023

PROJE ADI

POLİNOMSAL REGRESYON ile
PYTHON UYGULAMASI

HAZIRLAYANLAR:

Ecem
ÇEŞMECİLER
Elif TOP
Erdem ARDUÇ
Hüseyin MURAT
Utku Sait ÇİÇEK
Yunus YILDIRIM

Polinomsal Regresyon ile Python Uygulaması projesinde; hava durumu veri setindeki değişkenlerden yararlanarak Nem oranı değişkeni tahmin edilmiştir. Polinom derecesi ve polinom özellikleri belirlendikten sonra regresyon modeli eğitilmiş ve tahminler yapılmıştır. Elde edilen sonuçlar, tahminlerin doğruluğunu ve modelin performansını değerlendirmek için istatistiksel metriklerle (MSE, RMSE, R2) birlikte sunulmuştur. Polinomsal Regresyon ile eğitilen model Lineer Regresyon, Catboost Regressor ve LightGBM Regressor ile kıyaslanmıştır. Sonuçta en iyi tahmini karar ağaçları tabanlı Catboost Regressor ve LightGBM Regressor modelleri ardından Polinom Regresyon modeli gerçekleştirmiştir. Bu proje Ecem ÇEŞMECİLER, Elif TOP, Erdem ARDUÇ, Hüseyin MURAT, Utku Sait ÇİÇEK, Yunus YILDIRIM tarafından 15 HAZİRAN – 22 HAZİRAN arasında geliştirilmiştir.

GÖREV	YAPILAN İŞLEMLER
Veri Ön Hazırlık ve Temizleme(Elif TOP, Yunus YILDIRIM)	İlk adımda, eksik değerlerin hangi sütunlarda olduğunu ve ne kadar yaygın olduğunu belirlendi. Ardından, uygun bir strateji belirleyerek bu eksik değerleri dolduruldu ve gerektiğinde bu satırları veya sütunları kaldırıldı. Ayrıca, olası aykırı değerleri belirlendi ve uygun bir şekilde kaldırıldı.
Korelasyon Analizi ve Görselleştirme (Elif TOP, Yunus YILDIRIM)	Hava durumu özellikleri arasındaki ilişkileri anlamak için korelasyon analizi yapıldı. Bu, hangi özelliklerin hedef değişken "Nem Oranı" ile güçlü bir ilişkiye sahip olduğunu belirlemek için önemlidir. Ayrıca, önemli özelliklerin ve hedef değişkenin dağılımını göstermek için histogramlar, box-plotlar ve scatter plotlar da dahil olmak üzere çeşitli grafikler çizildi.
Lineer Regresyon Modeli ve Polinom Regresyon Modeli Eğitim ve Test Datası Olarak Tekrar Modellenmesi(Ecem ÇEŞMECİLER, Utku Sait ÇİÇEK, Hüseyin MURAT, Resul Erdem Arduç)	"Nem Oranı" değişkenini tahmin etmek için bir lineer regresyon modeli ve polinom regresyon modeli oluşturuldu. Modeli eğitmek için veri train-test setlerine ayrıldı. Modelin performansını değerlendirmek için uygun metrikler (MSE, RMSE, R2.) seçildi.
Catboost Regressor ve LightGBM Regressor ile Modelleme (Utku Sait ÇİÇEK, Ecem ÇEŞMECİLER)	"Nem Oranı" değişkenini tahmin etmek için bir Catboost Regressor ve LightGBM Regressor modeli oluşturuldu. Modeli eğitmek için veri train-test setlerine ayrıldı. Modelin performansını değerlendirmek için uygun metrikler (MSE, RMSE, R2.) seçildi.
Tüm Sonuç Skorlarının Karşılaştırılması ve Görselleştirilmesi (Utku Sait ÇİÇEK, ECEM ÇEŞMECİLER)	Tüm ekip bir araya gelip 5 modelin performansını karşılaştırdı. Bu, hangi modelin "Nem Oranı" değişkenini daha iyi tahmin ettiğini belirlemek için kritikti. Sonuçları görselleştirmek için çeşitli grafikler çizildi. En iyi model Catboost Regressor model seçildi.
Rapor Oluşturma (Ecem ÇEŞMECİLER)	Yapılan proje başkaları için de anlaşılır olabilmesi için rapor oluşturuldu..

Regresyon analizi, bağımlı değişkenin bağımsız değişkenlerle olan ilişkisini ifade etmek ve bu ilişkiyi kullanarak bağımlı değişkenin değerini tahmin etmek amacıyla kullanılır. Polinom regresyona ihtiyaç duymamızın sebebi bağımlı ve bağımsız değişken arasındaki ilişkiyi doğruya en yakın şekilde açıklamak istememizdir. Bu çalışmada lineer ve polinom regresyon skorlarını da karşılaştırarak verilerimizin polinom regresyona daha uygun olduğunu fark etmiş olduk.

Veri

Hava durumunu tarif eden veri setimizde nem oranının bağımsız değişkenlerle olan ilişkisini görmek istiyoruz. Veri setinde bulunan sütunların açıklaması aşağıdaki gibidir:

	datetime_utc	_conds	_dewptm	_fog	_hail	_heatindexm	_hum	_precipm	_pressurem	_rain	_snow	_tempm	_thunder	_tornado	_vism	_wdird	_wdire	_wgustm	_windchillm	_wspdm
0	19961101-11:00	Smoke	9.0	0	0	NaN	27.0	NaN	1010.0	0	0	30.0	0	0	5.0	280.0	West	NaN	NaN	7.4
1	19961101-12:00	Smoke	10.0	0	0	NaN	32.0	NaN	-9999.0	0	0	28.0	0	0	NaN	0.0	North	NaN	NaN	NaN
2	19961101-13:00	Smoke	11.0	0	0	NaN	44.0	NaN	-9999.0	0	0	24.0	0	0	NaN	0.0	North	NaN	NaN	NaN
3	19961101-14:00	Smoke	10.0	0	0	NaN	41.0	NaN	1010.0	0	0	24.0	0	0	2.0	0.0	North	NaN	NaN	NaN
4	19961101-16:00	Smoke	11.0	0	0	NaN	47.0	NaN	1011.0	0	0	23.0	0	0	1.2	0.0	North	NaN	NaN	0.0
...	
100985	20170424-06:00	Haze	17.0	0	0	NaN	25.0	NaN	1005.0	0	0	34.0	0	0	4.0	320.0	NW	NaN	NaN	11.1
100986	20170424-09:00	Haze	14.0	0	0	NaN	16.0	NaN	1003.0	0	0	38.0	0	0	4.0	320.0	NW	NaN	NaN	22.2
100987	20170424-12:00	Haze	12.0	0	0	NaN	14.0	NaN	1002.0	0	0	36.0	0	0	4.0	270.0	West	NaN	NaN	18.5
100988	20170424-15:00	Haze	15.0	0	0	NaN	27.0	NaN	1004.0	0	0	32.0	0	0	2.0	320.0	NW	NaN	NaN	3.7
100989	20170424-18:00	Haze	15.0	0	0	NaN	30.0	NaN	1005.0	0	0	30.0	0	0	2.0	320.0	NW	NaN	NaN	3.7

100990 rows × 20 columns

```
column_descriptions = {
    '_cond': 'Hava durumu koşulu',
    '_dewptm': 'Çiy noktası sıcaklığı (°C)',
    '_fog': 'Sis durumu (0: Yok, 1: Var)',
    '_hail': 'Dolu durumu (0: Yok, 1: Var)',
    '_heatindexm': 'Hissedilen sıcaklık (°C)',
    '_hum': 'Nem oranı (%)',
    '_precipm': 'Yağış miktarı (mm)',
    '_pressurem': 'Atmosfer basıncı (mbar)',
    '_rain': 'Yağmur durumu (0: Yok, 1: Var)',
    '_snow': 'Kar durumu (0: Yok, 1: Var)',
    '_tempm': 'Sıcaklık (°C)',
    '_thunder': 'Gök gürültüsü durumu (0: Yok, 1: Var)',
    '_tornado': 'Tornado durumu (0: Yok, 1: Var)',
    '_vism': 'Görüş mesafesi (km)',
    '_wdird': 'Rüzgar yönü derecesi',
    '_wdire': 'Rüzgar yönü',
    '_wgustm': 'Rüzgar şiddeti (km/s)',
    '_windchillm': 'Rüzgar hissi sıcaklık (°C)',
    '_wspdm': 'Rüzgar hızı (km/s)'
}
```

Ön Hazırlık ve Temizleme

```
[ ] df = pd.read_csv("testset.csv")
[ ] df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100990 entries, 0 to 100989
Data columns (total 20 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   datetime_utc 100990 non-null   object 
 1   _conds       100918 non-null   object 
 2   _dewptm     100369 non-null   float64
 3   _fog        100990 non-null   int64  
 4   _hail        100990 non-null   int64  
 5   _heatindexm  29155 non-null   float64
 6   _hum         100233 non-null   float64
 7   _precipm    0 non-null      float64
 8   _pressurem  100758 non-null   float64
 9   _rain        100990 non-null   int64  
 10  _snow        100990 non-null   int64  
 11  _tempm       100317 non-null   float64
 12  _thunder     100990 non-null   int64  
 13  _tornado    100990 non-null   int64  
 14  _vism        96562 non-null   float64
 15  _wdird       86235 non-null   float64
 16  _wdire       86235 non-null   object 
 17  _wgustm     1072 non-null    float64
 18  _windchillm 579 non-null    float64
```

Veri setini okuttuk ve veri tiplerini inceledik.

```

def eksik_deger_tablosu(df):
    eksik_deger = df.isnull().sum()
    eksik_deger_yuzde = 100 * df.isnull().sum()/len(df)
    eksik_deger_table = pd.concat([eksik_deger, eksik_deger_yuzde], axis=1)
    eksik_deger_table_son = eksik_deger_table.rename(
        columns = {0 : 'Eksik Değerler', 1 : '% Değeri'})
    return eksik_deger_table_son

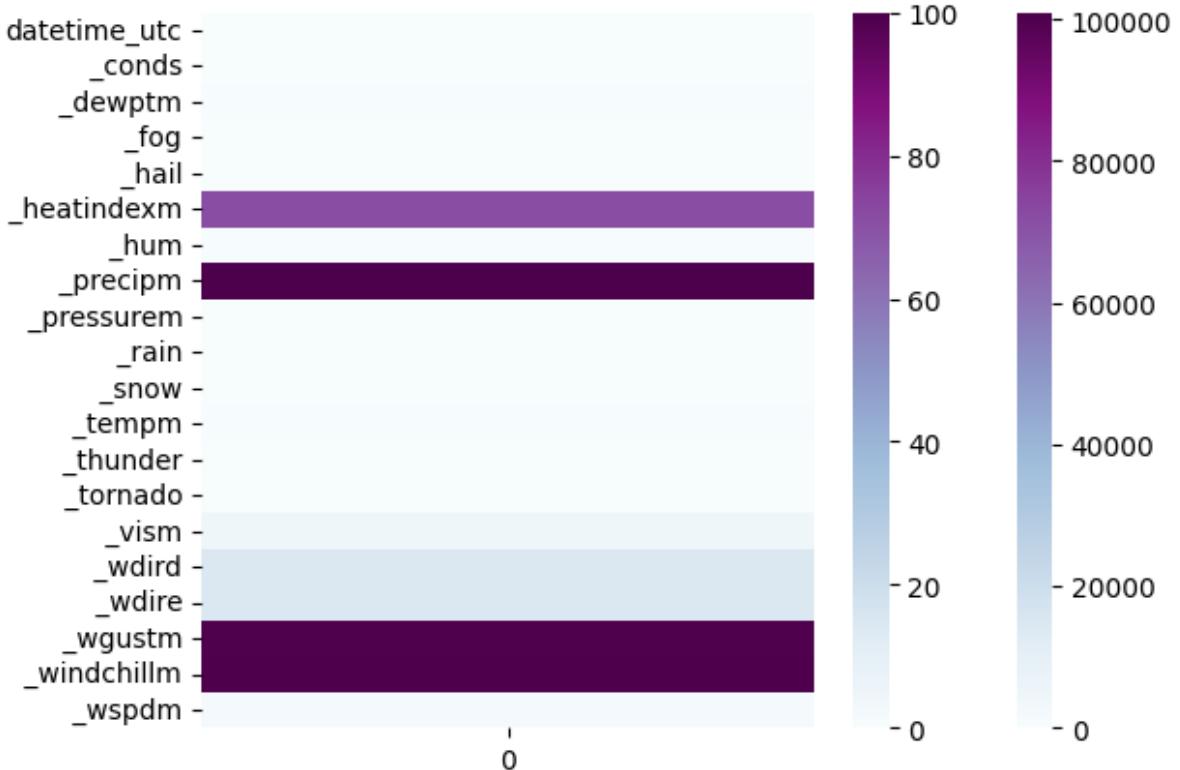
```

eksik_deger_tablosu(df)

	Eksik Değerler	% Değeri
<code>datetime_utc</code>	0	0.000000
<code>_conds</code>	72	0.071294
<code>_dewptm</code>	621	0.614912
<code>_fog</code>	0	0.000000
<code>_hail</code>	0	0.000000
<code>_heatindexm</code>	71835	71.130805
<code>_hum</code>	757	0.749579
<code>_precipm</code>	100990	100.000000
<code>_pressurem</code>	232	0.229726
<code>_rain</code>	0	0.000000

<code>_snow</code>	0	0.000000
<code>_tempm</code>	673	0.666403
<code>_thunder</code>	0	0.000000
<code>_tornado</code>	0	0.000000
<code>_vism</code>	4428	4.384593
<code>_wdird</code>	14755	14.610357
<code>_wdire</code>	14755	14.610357
<code>_wgustm</code>	99918	98.938509
<code>_windchillm</code>	100411	99.426676
<code>_wspdm</code>	2358	2.334885

Eksik değerlerin sayısını ve yüzdelik değerini eksik_deger_tablosu fonksiyonu ile görüntüledik.



Renk scalasına bakarak eksik veri sayısını görebiliriz.

Açık mavi renkte eksik veri yok demektir.

Mor renk 100.000 eksik veriyi temsil etmektedir.

Korelasyon Analizi ve Görselleştirme

Korelasyon analizi için veri setindeki null değerlerin işlenmesi gereklidir.

Veri kaybının %14 ve üzeri olan kolonları droplanmalıdır.

```
[ ] df.drop(["datetime_utc", "windchillm", "wgustm", "wdire", "wdird", "precipm", "heatindexm"], axis = 1, inplace = True)
```

En çok hava şartı olan object bulundu. Haze : puslu hava

```
▶ df["_conds"].value_counts().idxmax()
@ 'Haze'
```

Hava şartlarında ki boşluklar en çok tekrar eden Haze değeri ile dolduruldu.

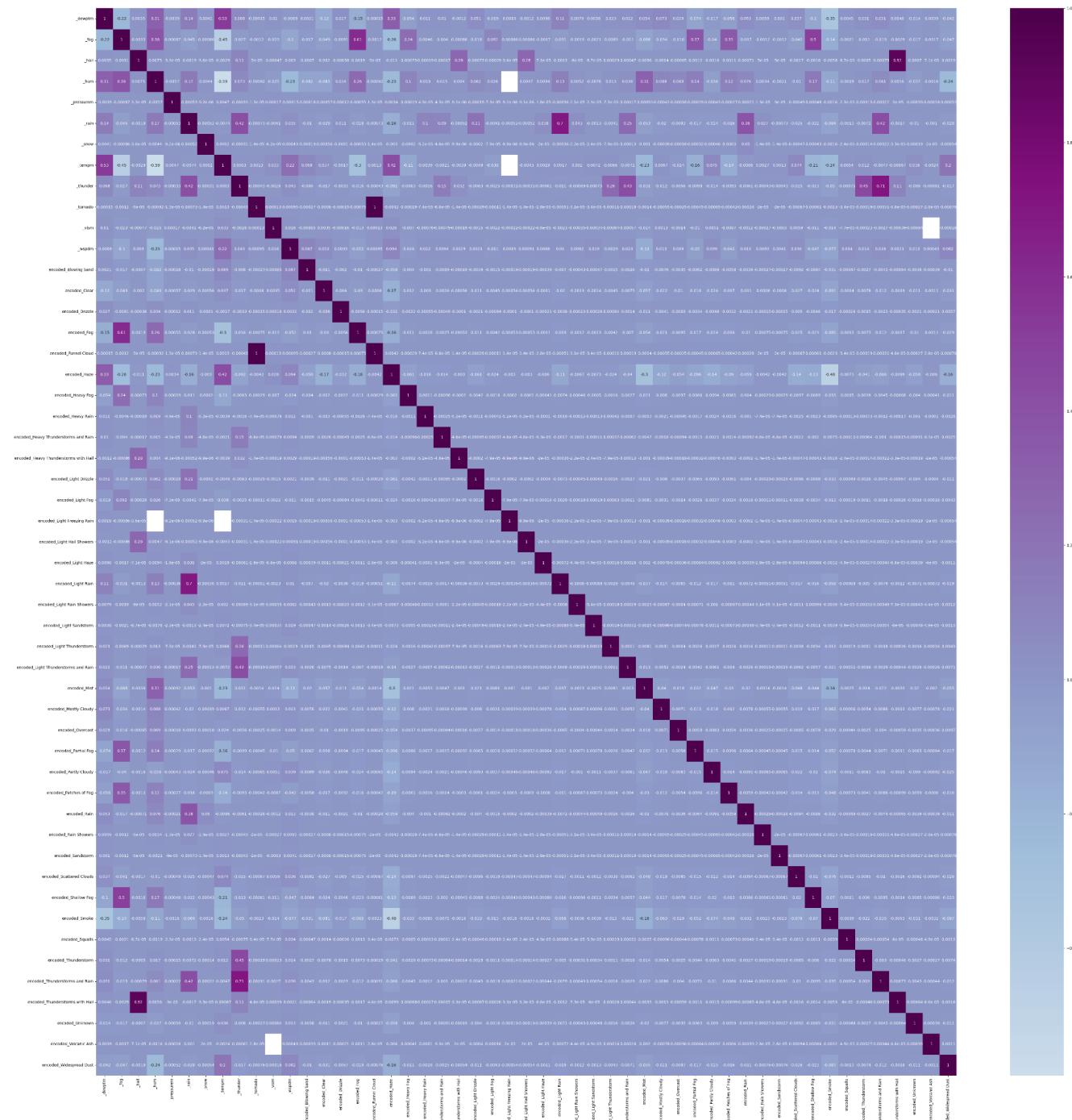
```
[ ] df["_conds"] = df["_conds"].fillna(df["_conds"].value_counts().idxmax())
```

Son olarak kolon object tipinde olduğundan get dummies yapısına çevrildi.

```
[ ] df_encoded = pd.get_dummies(df['_conds'], prefix='encoded')
df = pd.concat([df, df_encoded], axis=1).reindex(df.index)
df.drop('_conds', axis=1, inplace=True)
```

Korelasyon tablosu görselleştirilerek sayısal veriye dahi bakmadan fikir sahibi olunacak hale getirildi.

```
▶ plt.figure(figsize=(50, 50))
p = sns.heatmap(df.corr(), annot=True, cmap='BuPu', center=0)
```



`corr()` değeri NaN olan kolonları atılmalı.

```
[ ] df.drop("encoded_Light_Freezing_Rain", axis=1, inplace=True)
```

Encoded yapıldıktan sonra kolon sayısı 58 e çıktıgı için bir düzenleme yapma gereksinimi duyuldu.

Kolonlardan modelimizi saptıracak düşüncesi ile nem ile arasındaki Korelasyon değeri düşük olanlar minimum 0.05 kabulü ile elemeye edilecek..

```
[ ] column_list = df.columns.tolist()
```

```
[ ] for i in column_list:
    if (abs(df["_hum"]).corr(df[i]) < 0.05):
        df.drop(i, axis=1, inplace=True)
```

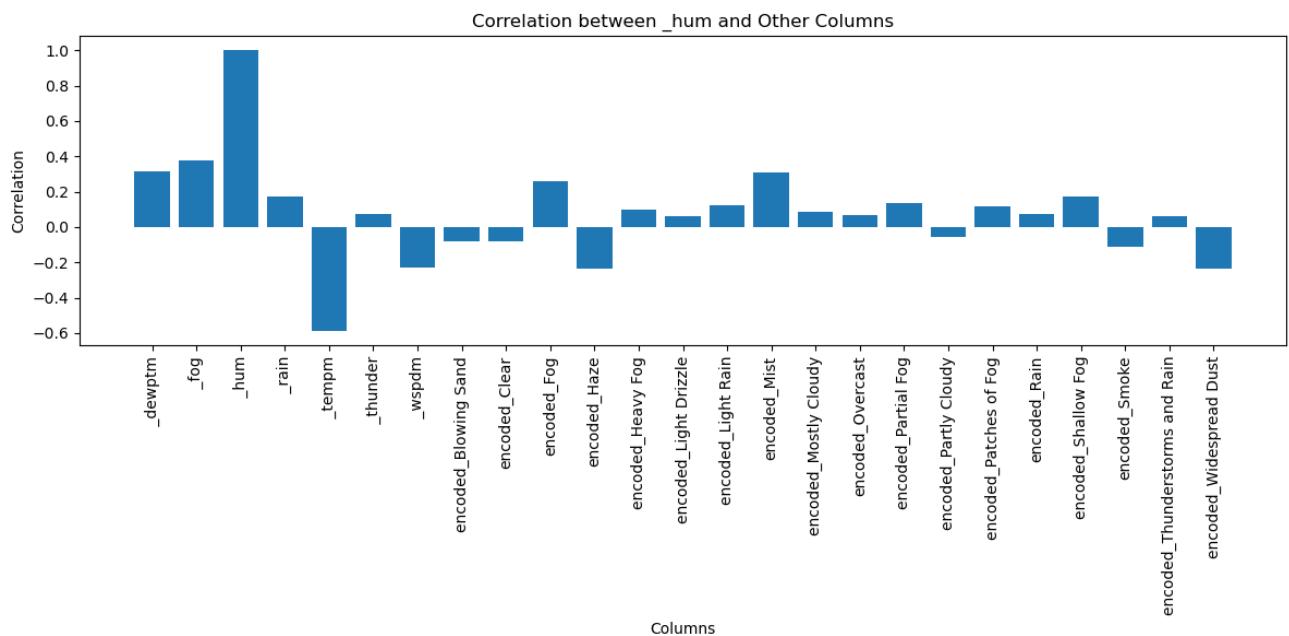
```

● correlation_values = []
column_list = df.columns.tolist()

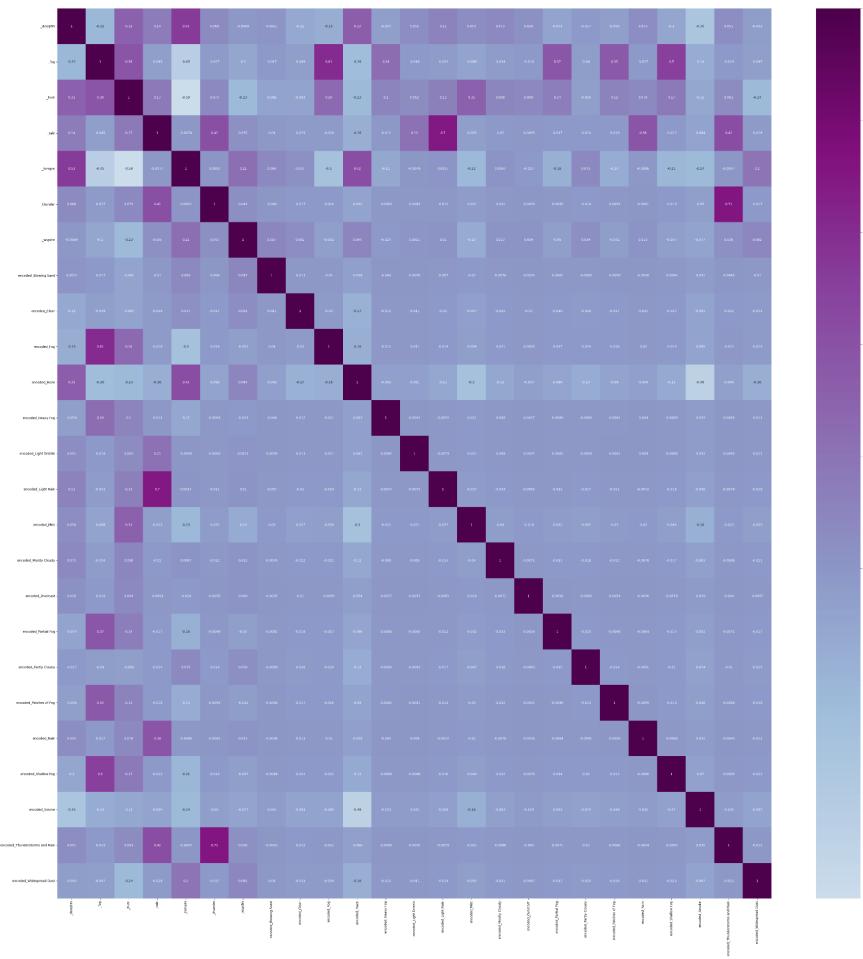
for i in column_list:
    correlation = df["_hum"].corr(df[i])
    correlation_values.append(correlation)
plt.figure(figsize=(12, 6))
plt.bar(column_list, correlation_values)
plt.xticks(rotation=90)
plt.xlabel("Columns")
plt.ylabel("Correlation")
plt.title("Correlation between _hum and Other Columns")
plt.tight_layout()
plt.show()

```

Bağımlı değişken “_hum” ile bağımsız değişkenler arasındaki korelasyonun bar grafiği:



Kalan değişkenler kullanılarak tekrar heatmap çizildi.

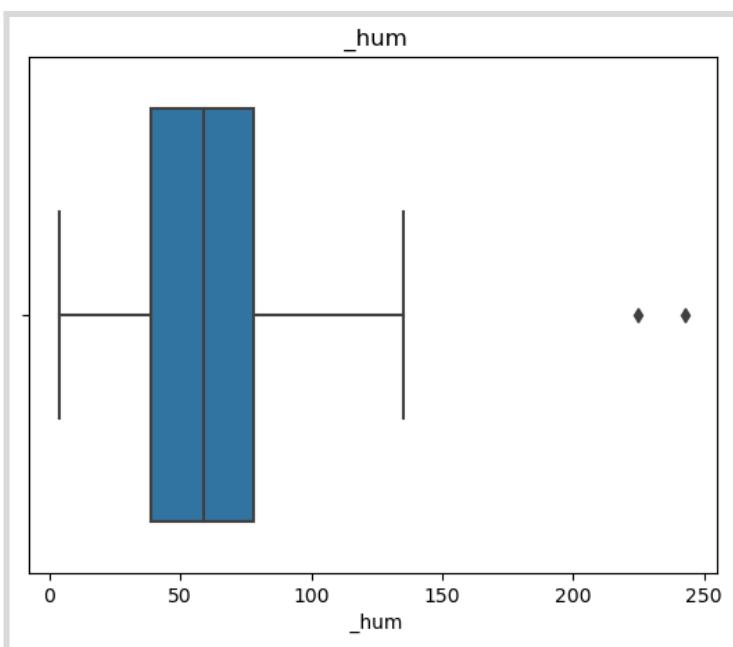
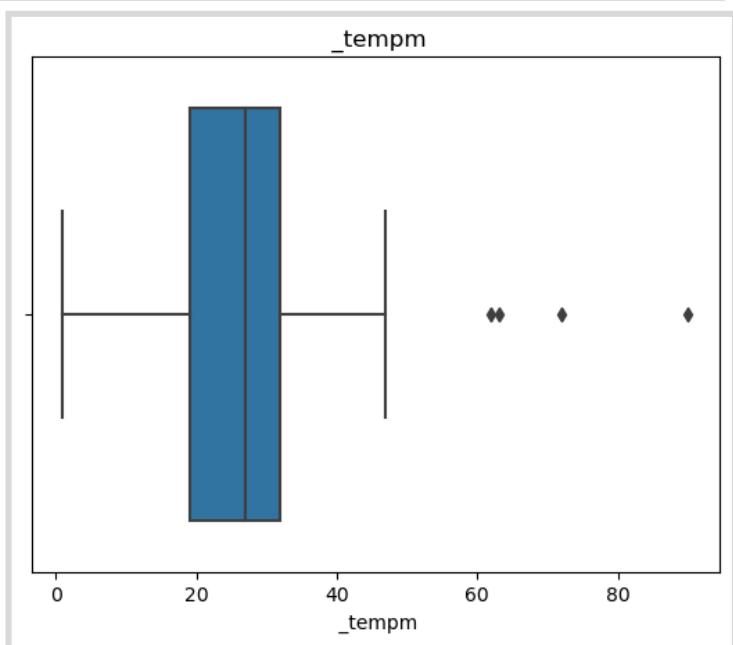
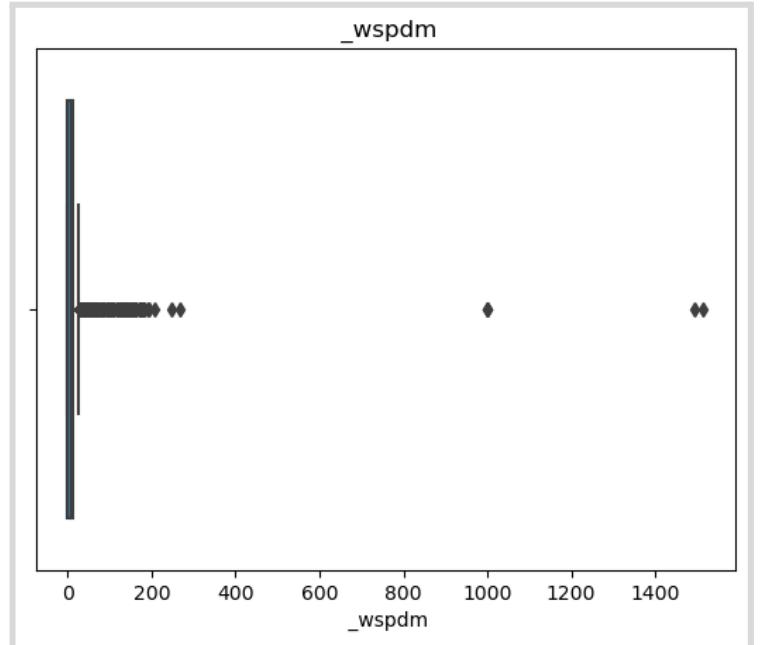


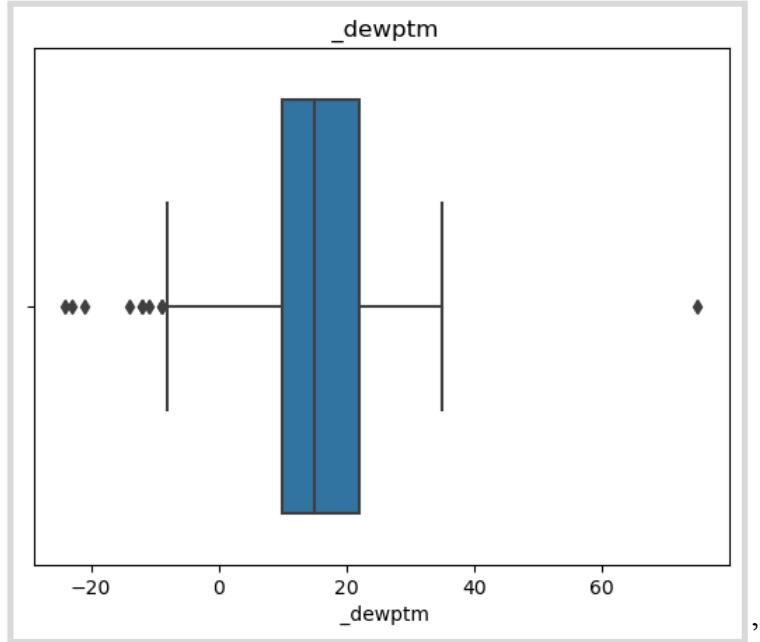
Sayısal kolonlarda aykırı değerler tespit edildi.

```
T = ["_dewptm", "_hum", "_tempm", "_wspd"]

for i in T:
    sns.boxplot(x=df[i])
    plt.title(i)
    plt.show()
    q1 = df[i].describe(percentiles=[0.25])["25%"]
    q3 = df[i].describe(percentiles=[0.75])["75%"]
    a = q3 - q1
    lower_bound = q1 - 1.5 * a
    upper_bound = q3 + 1.5 * a
    outliers = df[i][(df[i] < lower_bound) | (df[i] > upper_bound)]
    print(f"Outliers for {i}:")
    print(outliers)
    print(df[i].describe())
```

Outlier değerlerini grafik halinde görüntülendi. Ayrık veri sayısı ile verideki yüzdelikleri listelendi.





“_wspdm” sütunu veri setine oranla %1'den fazla outlier içermektedir. Bu sebeple sütunu veri setinden çıkardık.

```
[ ] df.drop("_wspdm", axis=1, inplace=True)
```

["_hum"] kolonlarında eksik değerler az olduğu için eksik değer içeren satırları silindi.

```
[ ] df[["_hum"]].fillna(df[["_hum"]].mean(), inplace=True)
```

"_tempm", "_dewptm" kolonlarından boşluk doldurmak için median ve mod kullanamıyoruz.

Cünkü linear ve polynomial yaklaşım tablolarında sapmalarla sebep oluyor.

```
[ ] df = df.dropna(subset=["_tempm"])
```

```
[ ] df = df.dropna(subset=["_dewptm"])
```

Mod ve medyan verinin değişkenliğini yansıtma, özellikle doğrusal olmayan ilişkileri modellemekte yeterli degillerdir.

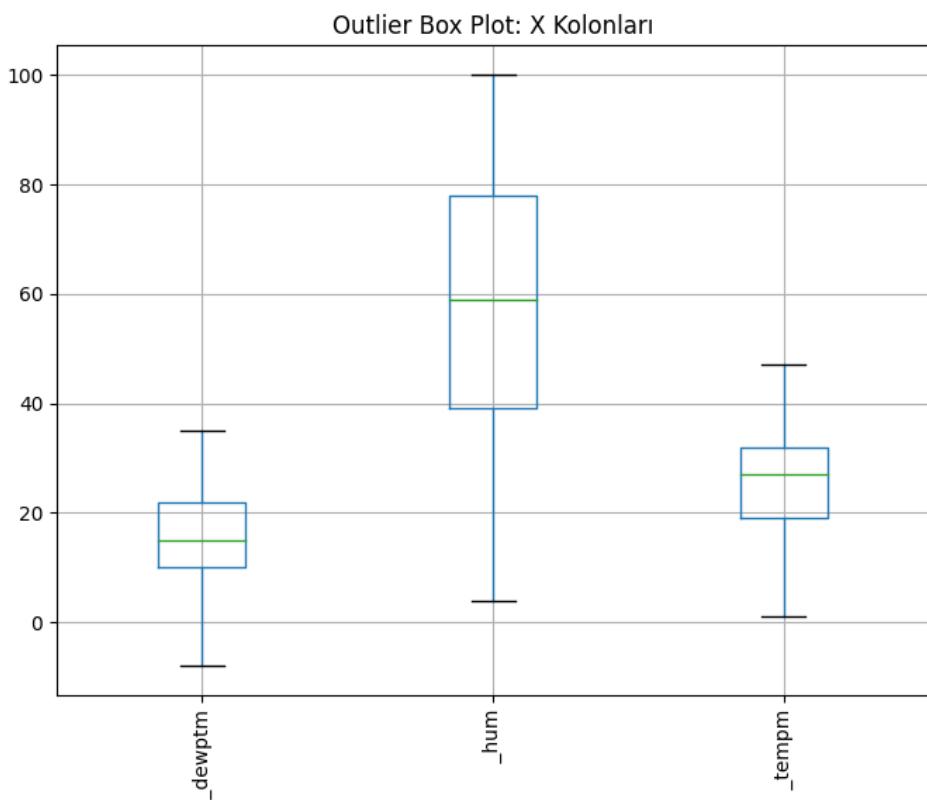
["_dewptm", "_hum", "_tempm"] kolonlarındaki akyarı değerlerin sayısı düşük olduğundan bu kolonlardaki akyarı değerleri içeren satırlar silindi.

```
▶ column_list = ["_dewptm", "_hum", "_tempm"]

for column in column_list:
    q1 = df[column].describe(percentiles=[0.25])["25%"]
    q3 = df[column].describe(percentiles=[0.75])["75%"]
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr

    outliers = df[column][(df[column] < lower_bound) | (df[column] > upper_bound)]
    df = df.drop(outliers.index)

    # Box Plot
    X = df[column_list]
    # Kutu grafiği oluşturma
    plt.figure(figsize=(12, 6)) # Grafiğin boyutu
    X.boxplot()
    plt.xticks(rotation=90) # x eksenini etiketlerinin dönüş açısı
    plt.title("Outlier Box Plot: X Kolonları")
    plt.show()
```



Çoklu Lineer Regresyon

Bir tane bağımlı değişken (Y) ile bununla ilişkisi olan bir dizi bağımsız değişken (X) arasındaki ilişkiyi ifade eden doğrusal fonksiyonu hataların en az olacağı şekilde ortaya koymak için yapılan analizdir.

```

column_list = df.columns.tolist()
print(column_list)
column_list.remove("_hum")

['_dewptm', '_fog', '_hum', '_rain', '_t
◀

X = df[column_list]
y = df["_hum"]

model = LinearRegression()

# Modeli X ve y verilerine uydurma (eğitme)
model.fit(X, y)

# Tahmin yapma
y_pred_linear = model.predict(X)
score1 = model.score(X, y)
print(score1)

```

0.9232226679481867

$$MSE = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$$

MSE (Mean Squared Error) tahmini değerler ile gerçek değer arasındaki ortalama kare farkını ölçer. Gerçek değerini bildiğimiz bağımlı değişkenin değerini, model ile tahmin etmeye çalışıyoruz. Formüle göre, gerçek değerden tahmini değer çıkarılıp karesi alınır. Kaç tane gözlem varsa hepsi için ayrı ayrı bu işlemler yapılp toplanır ve n sayısına bölünerek ortalaması alınır.

```
mse1 = mean_squared_error(y ,y_pred_linear)
mse1
```

```
43.445350921758084
```

MSE değerleri karşılaştırılarak hangi modelin daha uygun olduğu konusunda değerlendirme yapılmış olur.

Root Mean Squared Error (RMSE), Mean Squared Error (**MSE**) değerinin **karekökünün** alınarak elde edilir. MSE, tahminlerin gerçek değerlerden ne kadar farklı olduğunu ölçen bir hata metriğidir. Ancak MSE'nin doğrudan kullanılması durumunda sonuçlar büyük olabilir. Bu nedenle RMSE, MSE'nin bir tür düzeltme işlemi olarak kullanılır.

```
# RMSE (Root Mean Squared Error)
rmse1 = np.sqrt(mean_squared_error(y ,y_pred_linear))
rmse1
```

```
6.591308741195339
```

Polinom Regresyon

```
poly_features = PolynomialFeatures(degree=2)
X_poly = poly_features.fit_transform(X)

# Polinom regresyon modelini oluşturma
model = LinearRegression()

# Modeli X_poly ve y verilerine uydurma (eğitme)
model.fit(X_poly, y)

# Tahmin yapma
y_pred_poly = model.predict(X_poly)
score2 = model.score(X_poly,y)
print(score2)
```

```
0.9745544269338279
```

```
mse2 = mean_squared_error(y ,y_pred_poly)
print("MSE: ",mse2)
# RMSE (Root Mean Squared Error)
rmse2 = np.sqrt(mse2)
print("RMSE: ",rmse2)
```

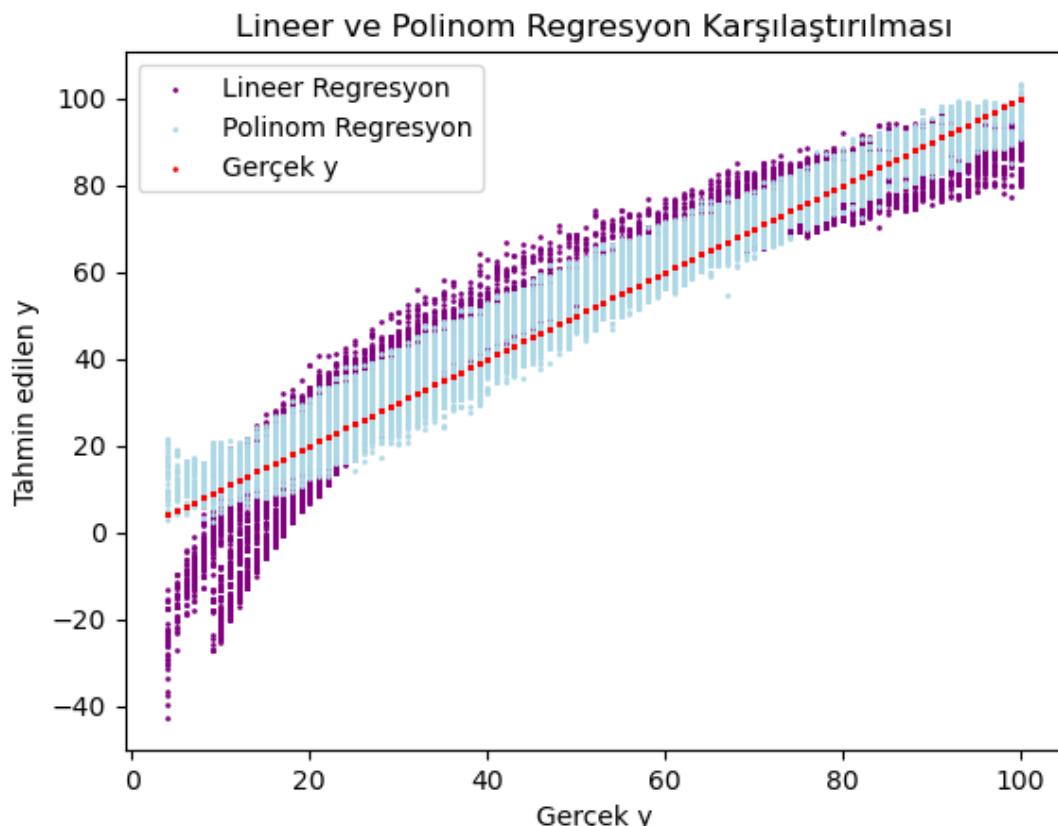
```
MSE:  14.398674995883438
RMSE:  3.7945586035642456
```

Lineer regresyonda model skoru 0.92 iken polinom regresyonda 0.97'dir. MSE VE RMSE değerlerinin de lineer regresyon modelinden düşük olduğu gözlemlenmiştir.

Lineer ve Polinom Regresyonun Karşılaştırılması

İki modeli karşılaştırmak için aşağıdaki kodu kullanarak bir scatter plot çizdik.

```
plt.scatter(y, y_pred_linear, color='purple', label='Lineer Regresyon', s=1)
plt.scatter(y, y_pred_poly, color='lightblue', label='Polinom Regresyon', s=1)
plt.scatter(y, y, color='red', label='Gerçek y', s=1) # Gerçek y değeri
plt.xlabel('Gerçek y')
plt.ylabel('Tahmin edilen y')
plt.title('Lineer ve Polinom Regresyon Karşılaştırılması')
plt.legend()
plt.show()
```



Gerçek değer ile bu iki model tarafından tahmin edilen değerleri karşılaştırdık. Kırmızı ile gösterilen lineer çizgi gerçek y'yi göstermektedir. Lineer regresyon modeliyle tahmin edilen değerler mor renk ile, polinom regresyon modeliyle tahmin edilen veriler ise mavi renk ile gösterilmektedir. Tahmin edilen değerlerin gerçek y çizgisine mümkün olduğunda yakın olması amaçlanmaktadır. Grafikte polinom regresyon tahmin noktalarının çizgiye daha yakın kümelenmesi, polinom regresyon modelinin bağımlı değişkenin bağımsız değişkenlerle olan ilişkisini daha iyi açıkladığı sonucunu verir.

Lineer ve Polinom Regresyonun Hata Karşılaştırılması

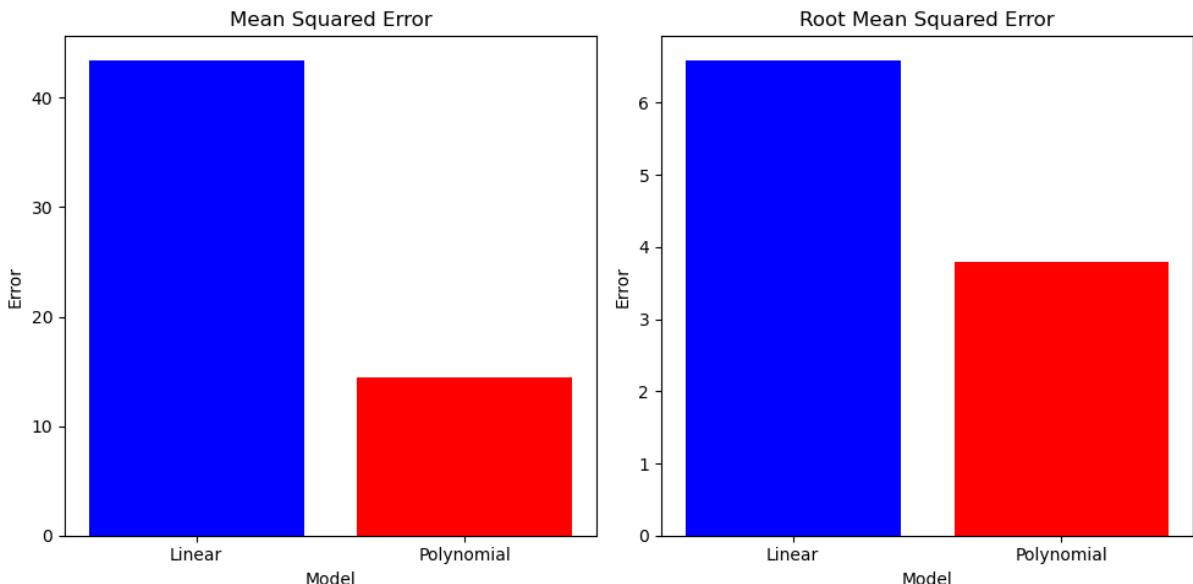
```
# Hesaplanan hata değerlerini bir liste haline getir
mse_values = [mse1, mse2]
rmse_values = [rmse1, rmse2]

# Hata değerlerinin isimlerini bir liste haline getir
labels = ['Linear', 'Polynomial']

# MSE değerlerini gösterleştir
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.bar(labels, mse_values, color=['blue', 'red'])
plt.title('Mean Squared Error')
plt.xlabel('Model')
plt.ylabel('Error')

# RMSE değerlerini gösterleştir
plt.subplot(1, 2, 2)
plt.bar(labels, rmse_values, color=['blue', 'red'])
plt.title('Root Mean Squared Error')
plt.xlabel('Model')
plt.ylabel('Error')

plt.tight_layout()
plt.show()
```



Lineer modelin hata değerlerinin daha yüksek olduğu görülür.

Veri Setinin Eğitim-Test Setlerine Bölünerek Regresyonu ve Değerlendirilmesi

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```

# Lineer regresyon modelini oluşturma
model = LinearRegression()

# Modeli eğitim verilerine uydurma (eğitme)
model.fit(X_train, y_train)

# Test seti üzerinde tahmin yapma
y_pred_linear = model.predict(X_test)

# Modelin test seti üzerindeki başarısını değerlendirme (R-kare değeri)
score3 = model.score(X_test, y_test)
print("R-kare değeri:", score3)

mse3 = mean_squared_error(y_test, y_pred_linear)
print("MSE: ", mse3)
# RMSE (Root Mean Squared Error)
rmse3 = np.sqrt(mse3)
print("RMSE: ", rmse3)

```

R-kare değeri: 0.9237762234732201
 MSE: 43.219933335186354
 RMSE: 6.574186895364806

```

# Polinom özelliklerini oluşturma
poly_features = PolynomialFeatures(degree=2)
X_train_poly = poly_features.fit_transform(X_train)
X_test_poly = poly_features.transform(X_test)

# Polinom regresyon modelini oluşturma
model = LinearRegression()

# Modeli eğitim verilerine uydurma (eğitme)
model.fit(X_train_poly, y_train)

# Test seti üzerinde tahmin yapma
y_pred_poly = model.predict(X_test_poly)

# Modelin test seti üzerindeki başarısını değerlendirme (R-kare değeri)
score4 = model.score(X_test_poly, y_test)
print("R-kare değeri:", score4)

mse4 = mean_squared_error(y_test, y_pred_poly)
print("MSE: ", mse4)
# RMSE (Root Mean Squared Error)
rmse4 = np.sqrt(mse4)
print("RMSE: ", rmse4)

```

R-kare değeri: 0.9746668113115524
 MSE: 14.364267636329057
 RMSE: 3.7900221155461686

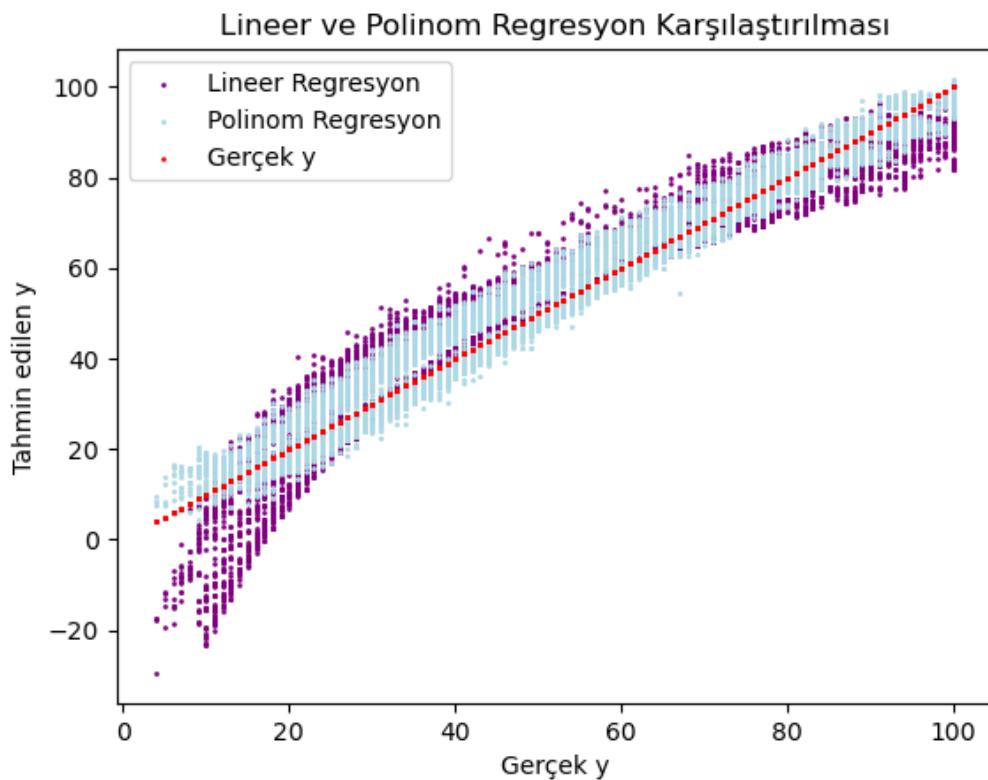
Bu yöntem ile, test veri setini eğitim setinden ayırarak test verilerinin eğitimde kullanılmamasını sağladık. Böylece modelin ezber yapmasını (overfitting) engellemeye çalıştık. Ayrıca gerçek dünya verileri üzerindeki performansını değerlendirmiş olduk. Sonuçlar daha önceki ölçümlerimize yakın çıktı.

```

plt.scatter(y_test, y_pred_linear, color='purple', label='Lineer Regresyon', s=1)
plt.scatter(y_test, y_pred_poly, color='lightblue', label='Polinom Regresyon', s=1)
plt.scatter(y_test, y_test, color='red', label='Gerçek y', s=1) # Gerçek y değeri
plt.xlabel('Gerçek y')
plt.ylabel('Tahmin edilen y')
plt.title('Lineer ve Polinom Regresyon Karşılaştırılması')
plt.legend()
plt.show()

```

Tekrar karşılaştırma grafiği çizdik:



```

# Hesaplanan hata değerlerini bir liste haline getir
mse_values = [mse3, mse4]
rmse_values = [rmse3, rmse4]

# Hata değerlerinin isimlerini bir liste haline getir
labels = ['Linear', 'Polynomial']

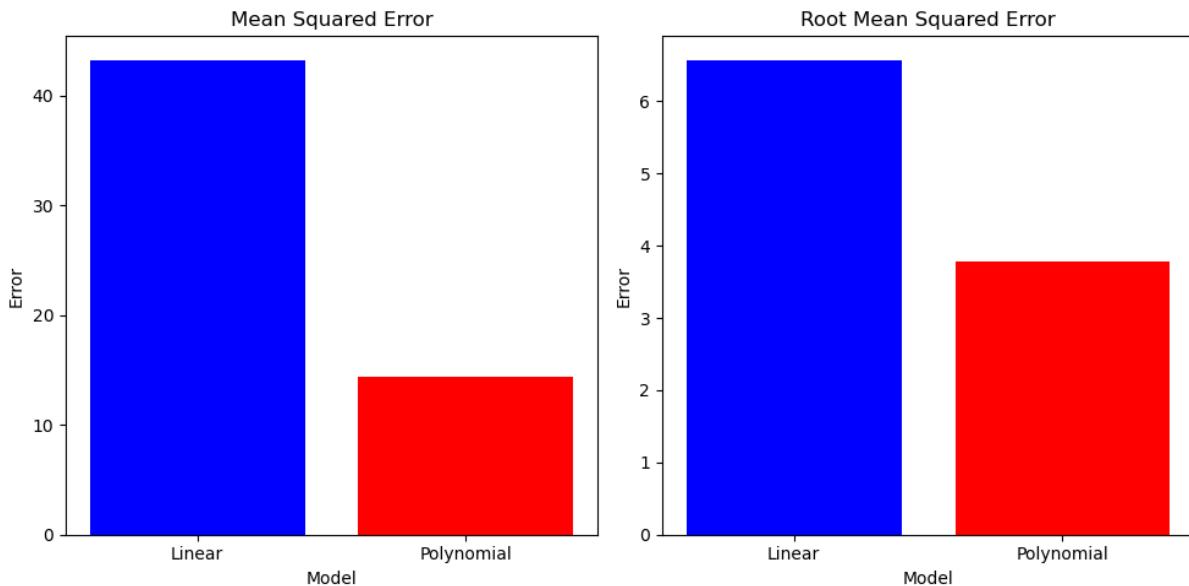
# MSE değerlerini gösterleştir
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.bar(labels, mse_values, color=['blue', 'red'])
plt.title('Mean Squared Error')
plt.xlabel('Model')
plt.ylabel('Error')

# RMSE değerlerini gösterleştir
plt.subplot(1, 2, 2)
plt.bar(labels, rmse_values, color=['blue', 'red'])
plt.title('Root Mean Squared Error')
plt.xlabel('Model')
plt.ylabel('Error')

plt.tight_layout()
plt.show()

```

Hata değerlerinin görselleştirilmesi:



CatboostRegressor ile Tahminleme

CatBoostRegressor, gradient boosting tabanlı bir regresyon modelidir. Gradient boosting, zayıf tahmincileri bir araya getirerek güçlü bir tahminci oluşturmayı hedefleyen bir makine öğrenimi yöntemidir. Kategorik verilerin bulunduğu veri setlerinde tercih edilir.

```
# CatBoost modeli oluşturma
model = CatBoostRegressor()

# Modeli eğitme
model.fit(X_train, y_train, verbose=False) # verbose=False eğitim sürecinde her iterasyonu göstermeyecektir

# Tahminler oluşturma
y_pred_boost = model.predict(X_test)

# Modelin R2 skorunu hesaplama
score5 = r2_score(y_test, y_pred_boost)

print("R2 Score: ", score5)

R2 Score:  0.9778284448644715

# MSE hesaplama
mse5 = mean_squared_error(y_test, y_pred_boost)

# RMSE hesaplama
rmse5 = np.sqrt(mse5)
```

```
# MSE hesaplama
mse5 = mean_squared_error(y_test, y_pred_boost)

# RMSE hesaplama
rmse5 = np.sqrt(mse5)

print("Mean Squared Error: ", mse5)
print("Root Mean Squared Error: ", rmse5)
```

Mean Squared Error: 12.571577774794262
Root Mean Squared Error: 3.545642082161461

LightGBMRegressor ile Tahminleme

LightGBMRegressor de gradient boosting tabanlı bir modeldir.

▼ LightGBMRegressor

```
[46]
0
model = LGBMRegressor()
model.fit(X_train, y_train)
y_pred_lightgbm = model.predict(X_test)
scorelg = r2_score(y_test, y_pred_lightgbm)
print("R2 ", scorelg)
mselg = mean_squared_error(y_test, y_pred_lightgbm)
rmselg = np.sqrt(mselg)
print("Mean Squared Error: ", mselg)
print("Root Mean Squared Error: ", rmselg)

R2  0.9769537443937156
Mean Squared Error:  13.06754501437392
Root Mean Squared Error:  3.614905948205834
```

CatBoost daha doğru tahminlerde bulunur ancak LightGBM daha hızlıdır ve daha az bellek kullanır. Bu sebeple daha büyük veri setlerinde tercih edilir.

Tüm Skorların Görselleştirilmesi

```
# Hesaplanan hata değerlerini ve skorları bir liste haline getir
mse_values = [mse1, mse3, mse2, mse4,mse5,mselg]
rmse_values = [rmse1, rmse3, rmse2, rmse4,rmse5,rmselg]
score_values = [score1, score3, score2, score4,score5,scorelg]

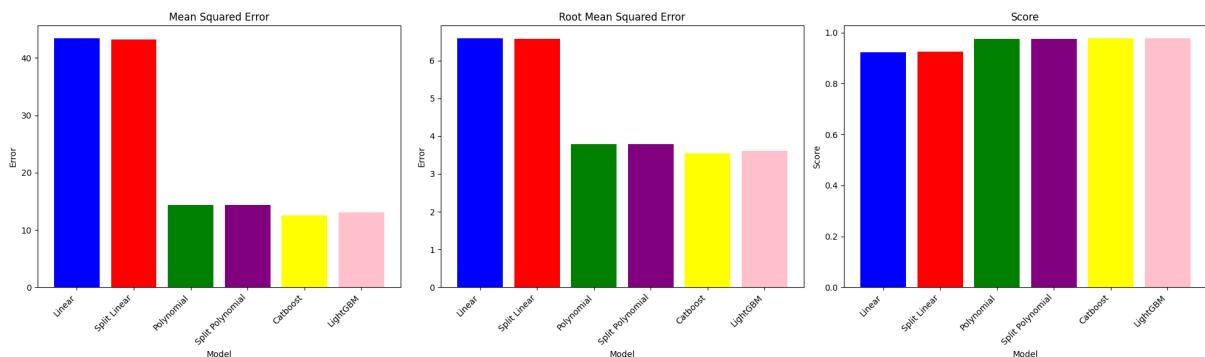
# Modellerin isimlerini bir liste haline getir
labels = ['Linear', 'Split Linear', 'Polynomial', 'Split Polynomial','Catboost','LightGBM']

# MSE değerlerini göster
plt.figure(figsize=(20, 6))
plt.subplot(1, 3, 1)
plt.bar(labels, mse_values, color=['blue', 'red', 'green', 'purple','yellow','pink'])
plt.title('Mean Squared Error')
plt.xlabel('Model')
plt.ylabel('Error')

# RMSE değerlerini göster
plt.subplot(1, 3, 2)
plt.bar(labels, rmse_values, color=['blue', 'red', 'green', 'purple','yellow','pink'])
plt.title('Root Mean Squared Error')
plt.xlabel('Model')
plt.ylabel('Error')

# Score değerlerini göster
plt.subplot(1, 3, 3)
plt.bar(labels, score_values, color=['blue', 'red', 'green', 'purple','yellow','pink'])
plt.title('Score')
plt.xlabel('Model')
plt.ylabel('Score')

plt.tight_layout()
plt.show()
```



Grafiklerde en iyi sonucu Catboost ve LightGBM ardından (split) polinom regresyon vermiştir.